CSCE 638 Natural Language Processing Foundation and Techniques

Lecture 5: Convolutional Neural Network and Recurrent Neural Network

Kuan-Hao Huang

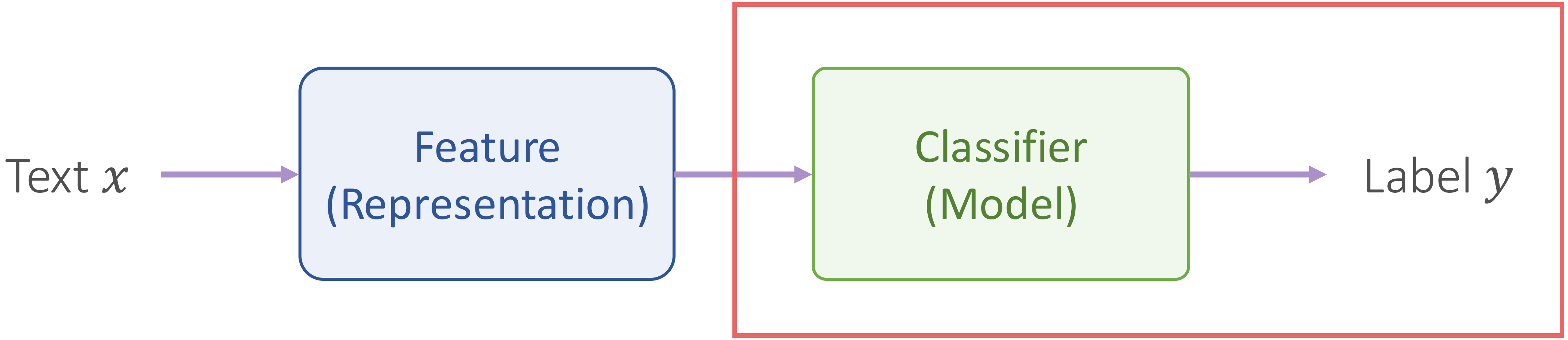
Spring 2026

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

Lecture Plan

- Convolutional Neural Network
- Recurrent Neural Network
  - Long Short-Term Memory
  - Gated Recurrent Units

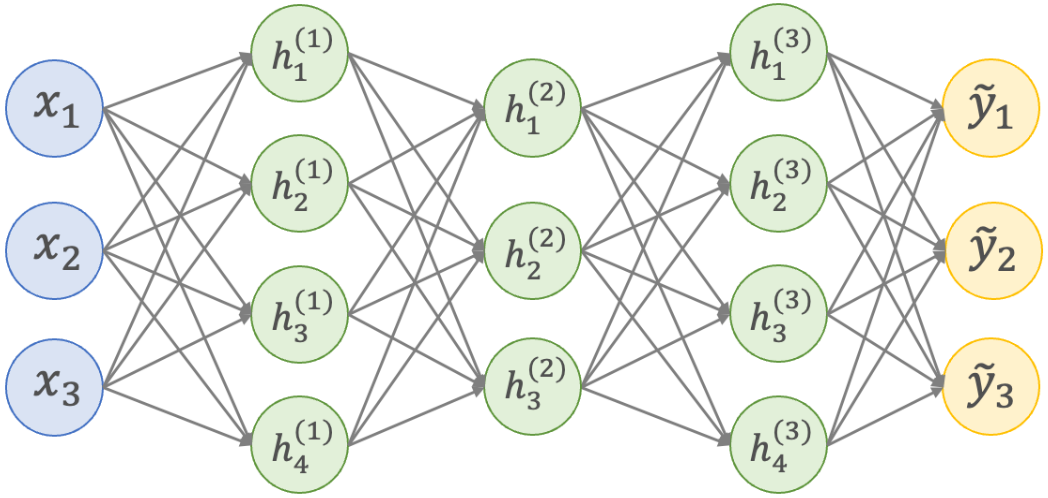
Recap: A General Framework for Text Classification



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

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



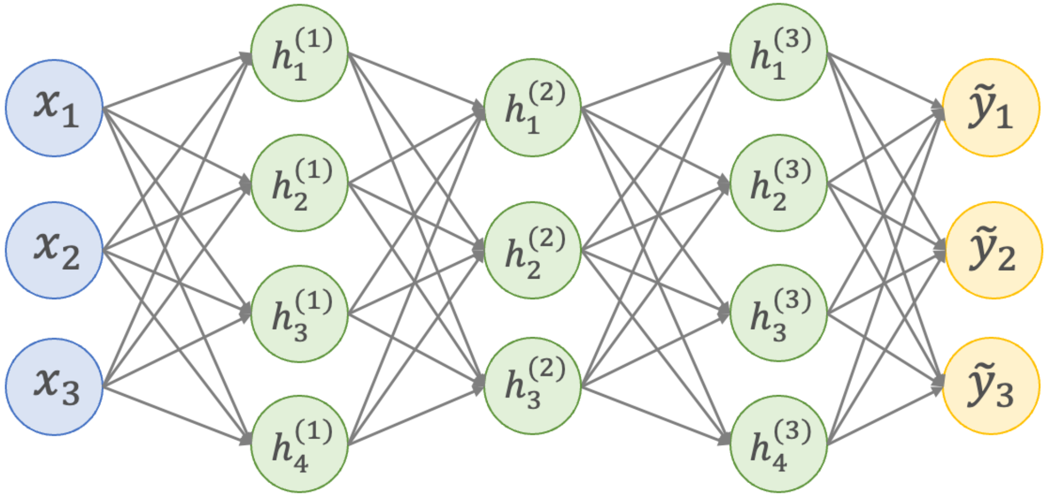
$$\mathcal{L}_{CE}(y, \tilde{y}) = -\sum_{c=0}^{C} y_c \log P(y = c | \, \mathbf{x})$$

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 1: 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

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

	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

$w_{1,1}$*2.7 + $w_{1,2}$*-0.1 + $w_{1,3}$*-5.7

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

$w_{3,1}$*-5.6 + $w_{3,2}$*1.5 + $w_{3,3}$*-1.6

$w_{4,1}$*1.1 + $w_{4,2}$*2.0 + $w_{4,3}$*-1.0

$v_{1,1}$	$v_{2,1}$
+	+
$v_{1,2}$	$v_{2,2}$
+	+
$v_{1,3}$	$v_{2,3}$
+	+
$v_{1,4}$	$v_{2,4}$
v_1	v_2

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	so	Bob	thinks
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			

$v_{1,1}$	$v_{2,1}$	$v_{3,1}$	$v_{4,1}$	$v_{5,1}$
+	+	+	+	+
$v_{1,2}$	$v_{2,2}$	$v_{3,2}$	$v_{4,2}$	$v_{5,2}$
+	+	+	+	+
$v_{1,3}$	$v_{2,3}$	$v_{3,3}$	$v_{4,3}$	$v_{5,3}$
+	+	+	+	+
$v_{1,4}$	$v_{2,4}$	$v_{3,4}$	$v_{4,4}$	$v_{5,4}$
v_1	v_2	v_3	v_4	v_5

Max Pooling

v

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

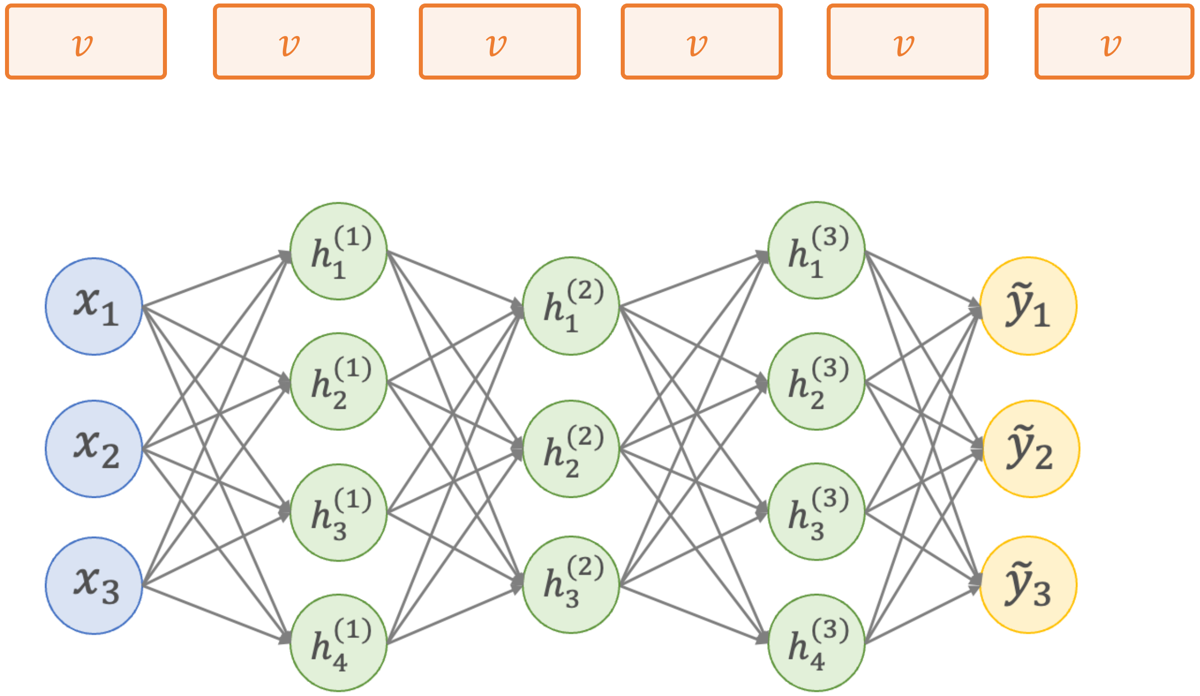
$w_{1,1}$*0.7 + $w_{1,2}$*2.7

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

v v v

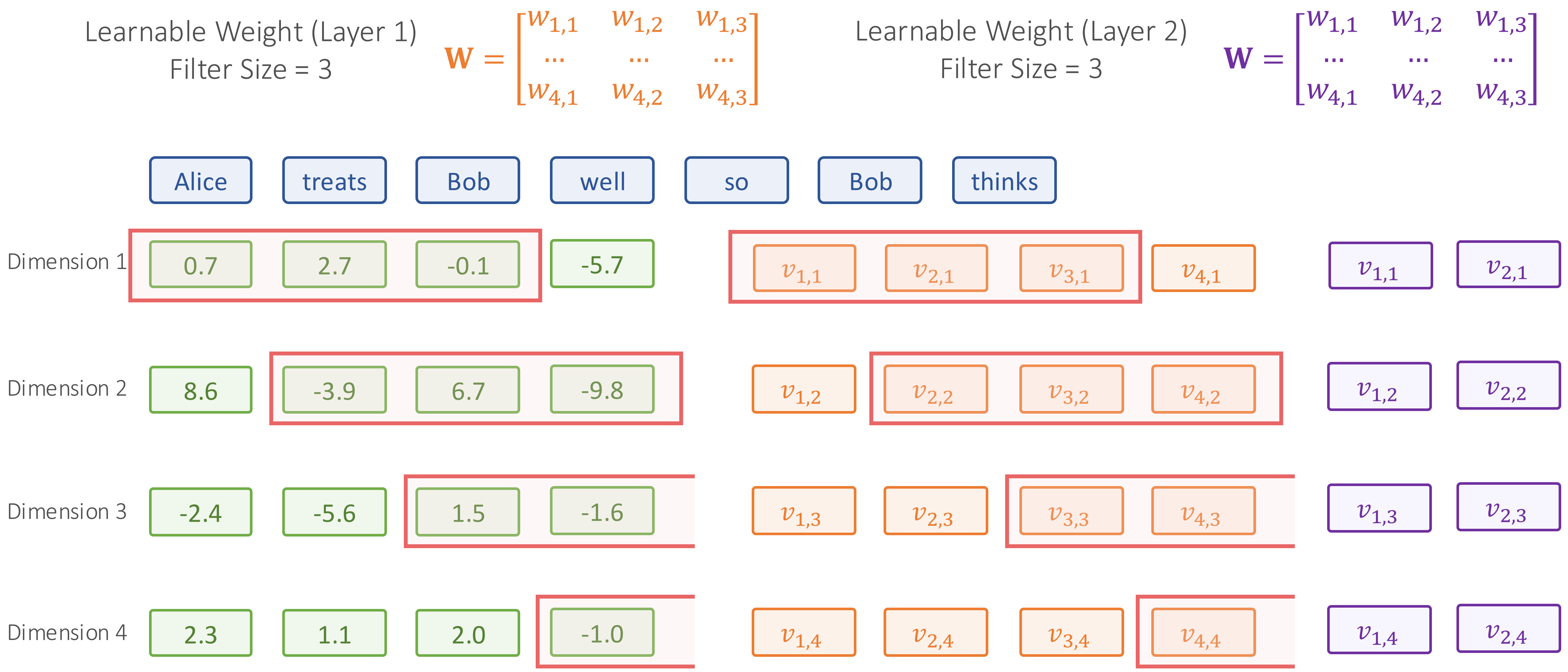
Convolutional Neural Network (CNN)

	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



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

From Single Layer to Multiple Layers



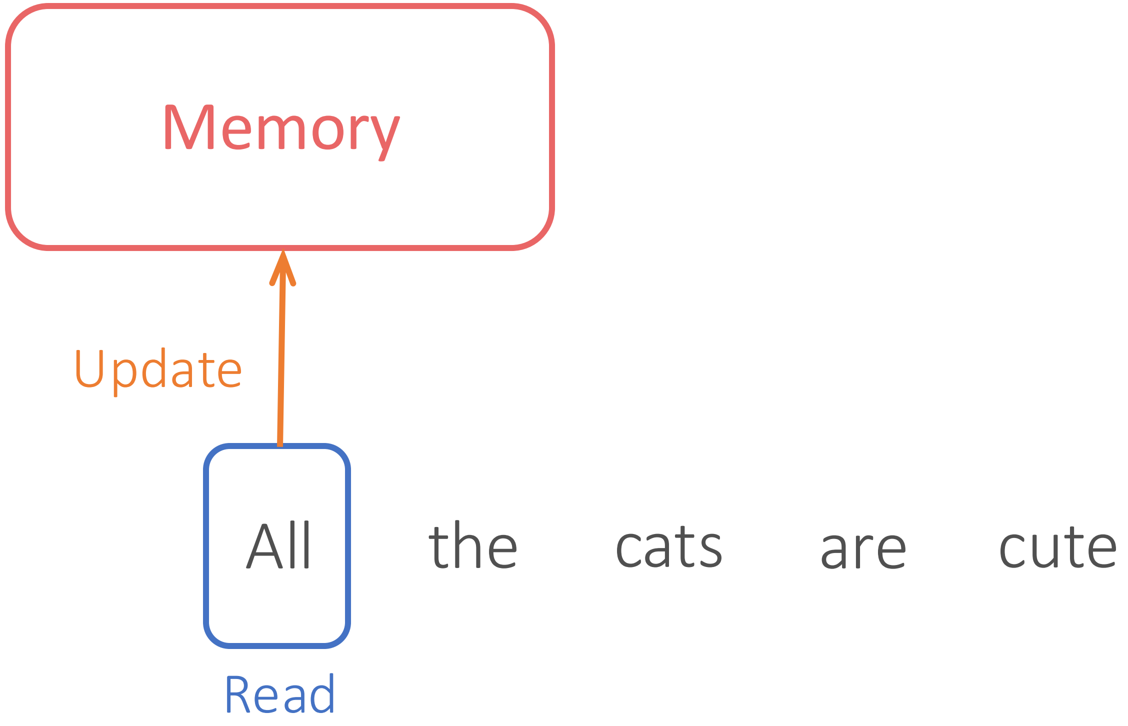
Capture high-order or hierarchical information

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?

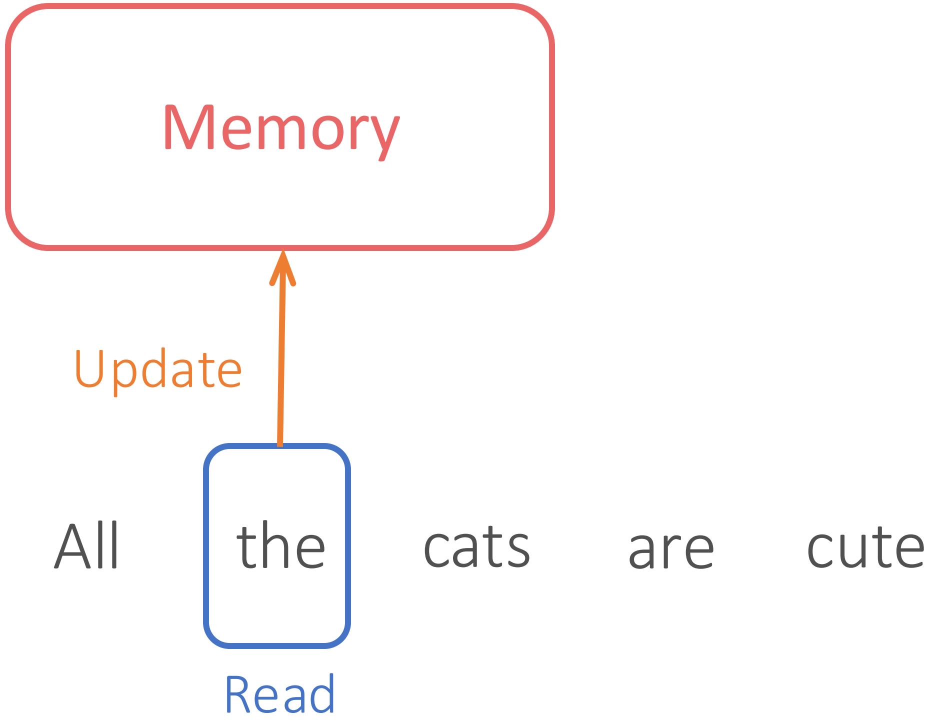
Recurrent Neural Network (RNN)

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



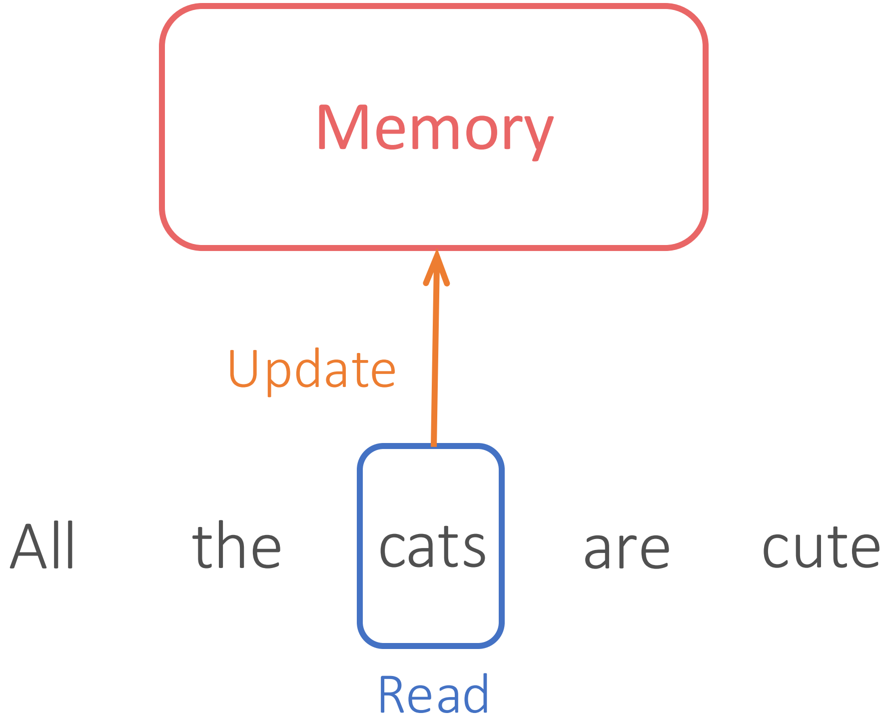
Recurrent Neural Network (RNN)

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



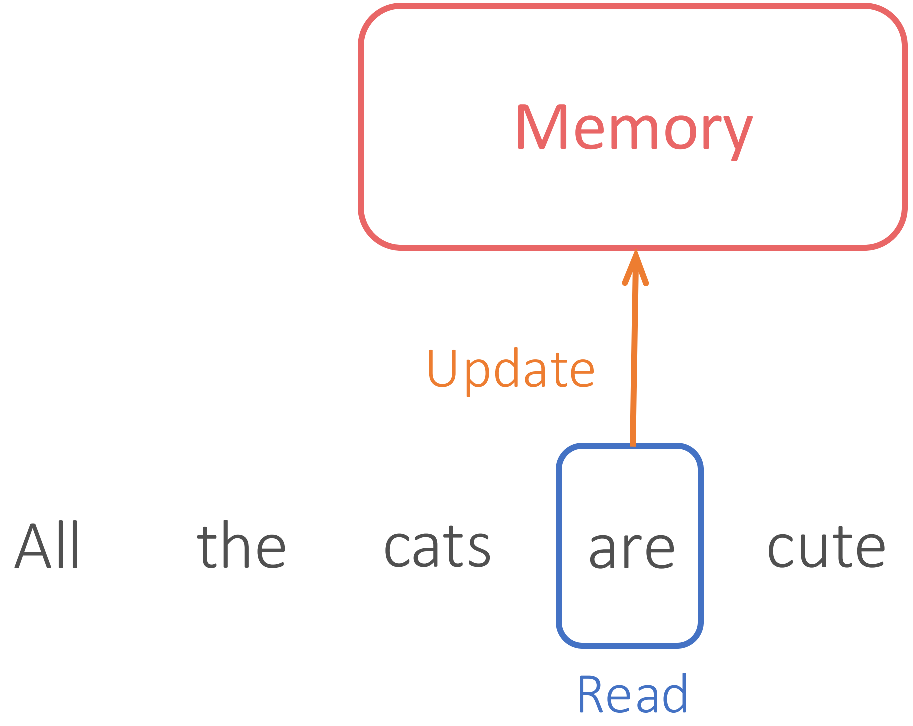
Recurrent Neural Network (RNN)

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



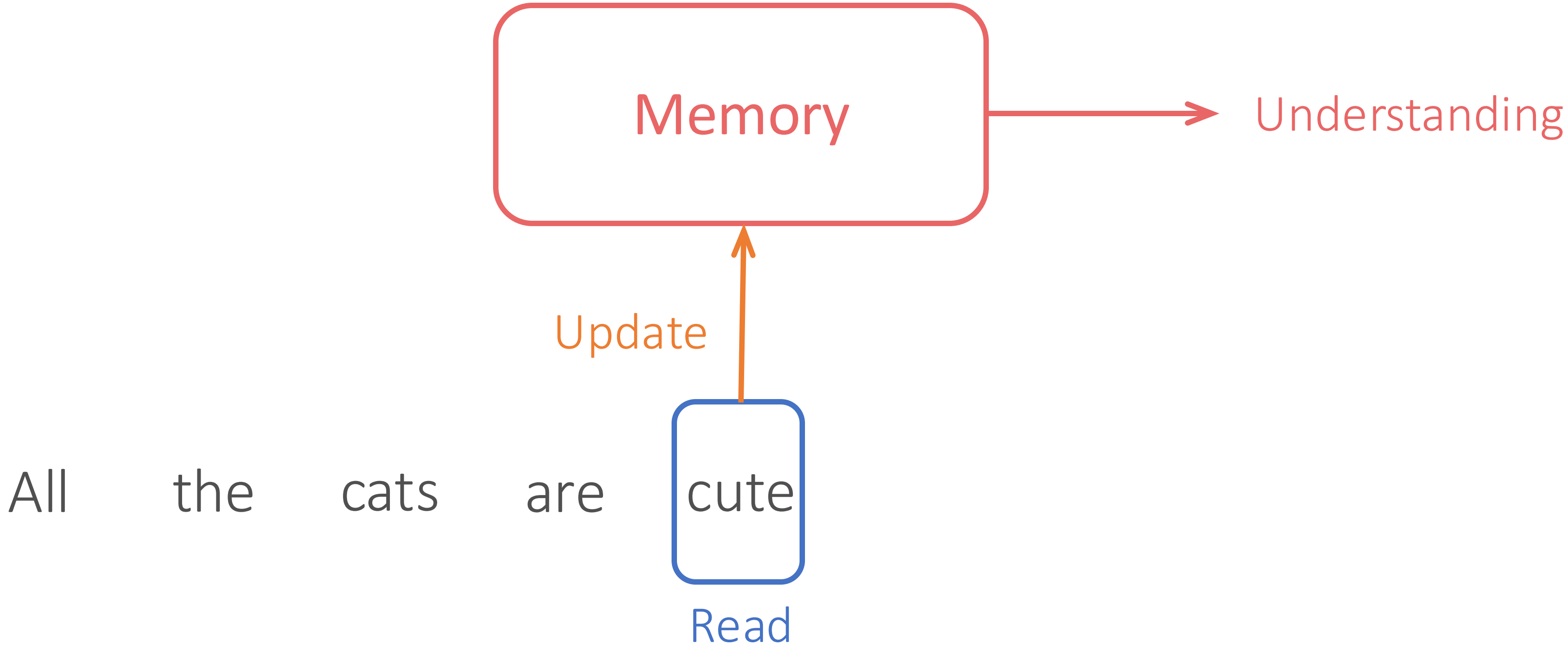
Recurrent Neural Network (RNN)

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



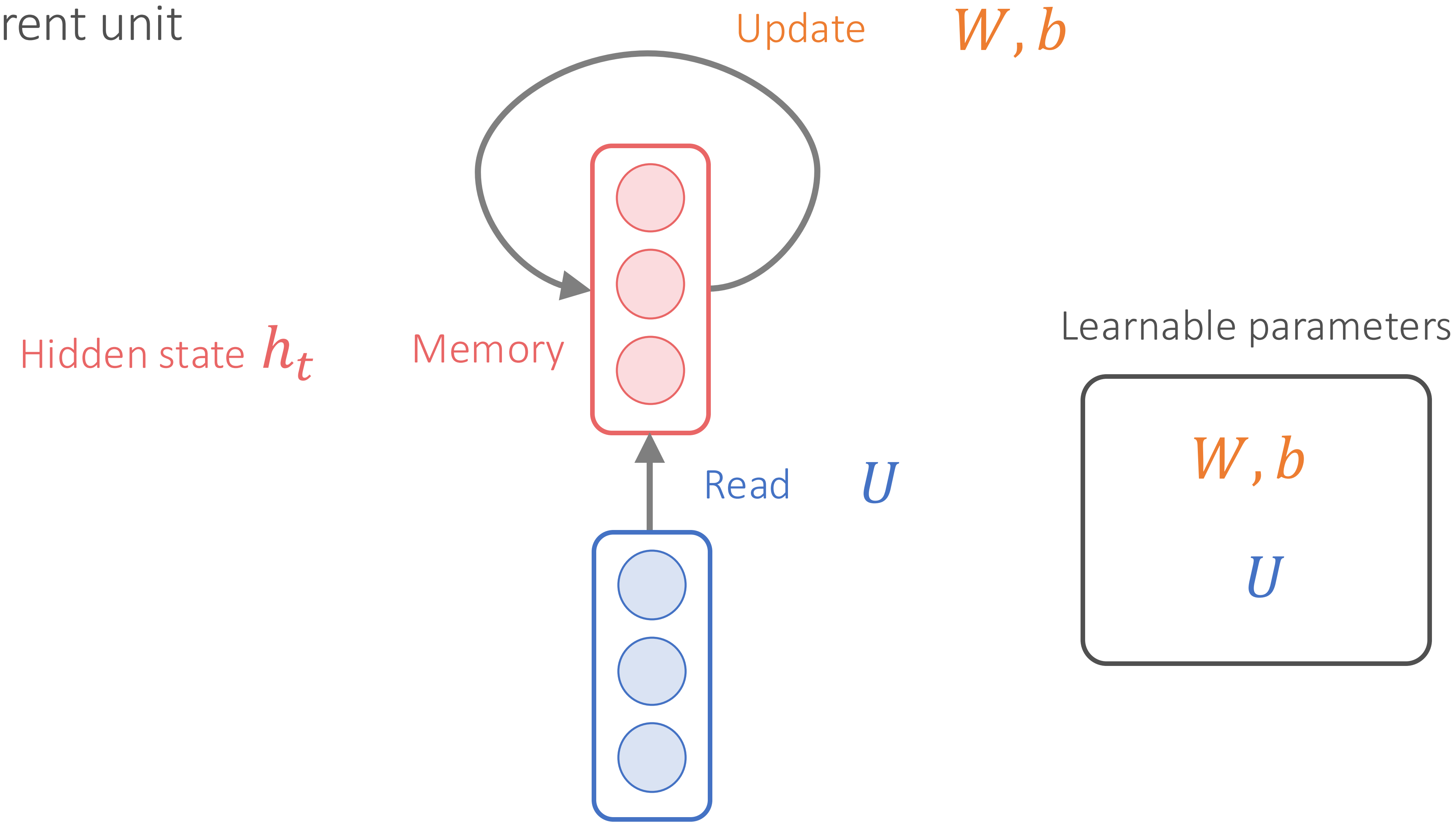
Recurrent Neural Network (RNN)

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



Recurrent Neural Network (RNN)

- Recurrent unit



Recurrent Neural Network (RNN)

Hidden States

Read

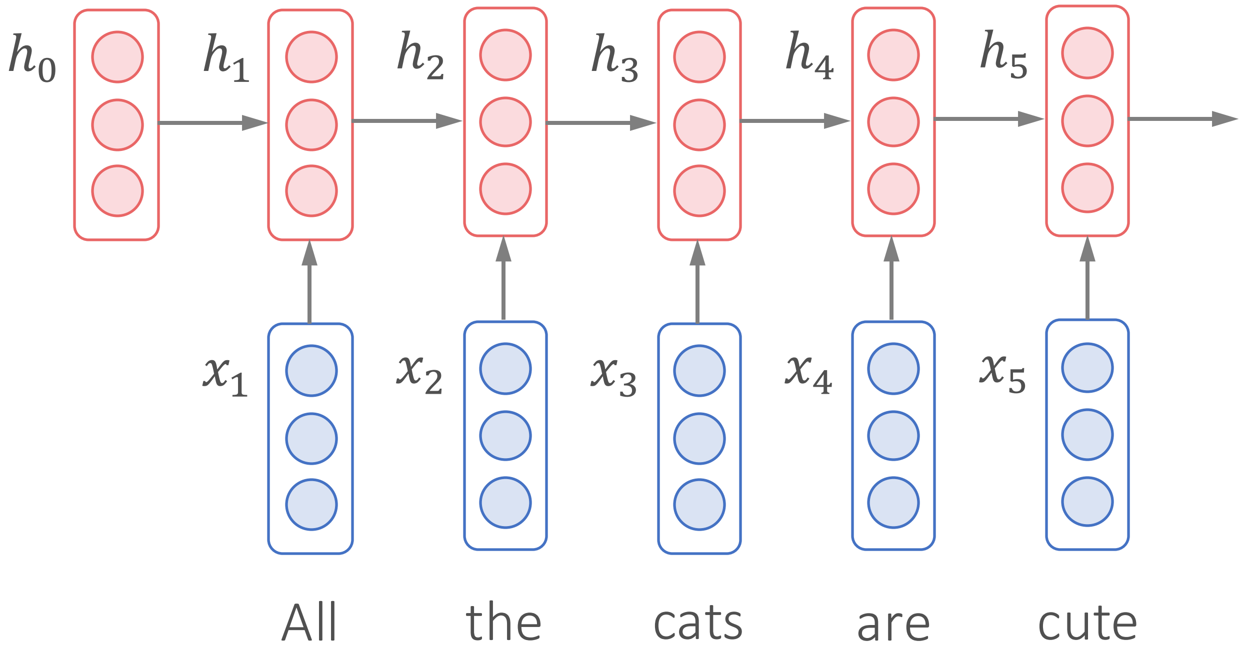
$$h_t = \sigma(W h_{t-1} + U x_t + b)$$

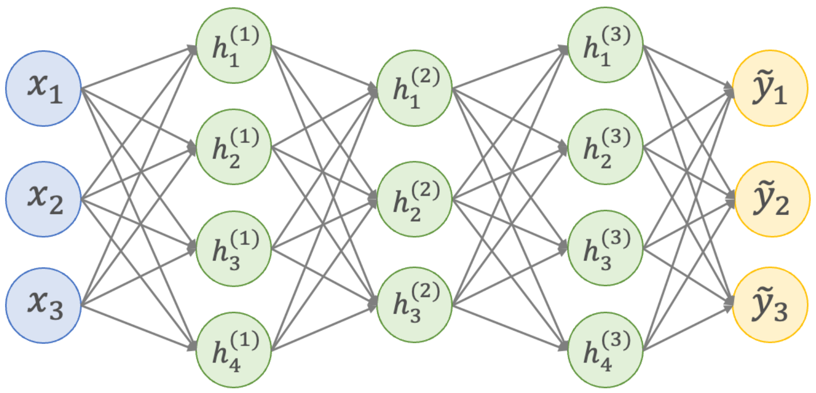
Activation Function
(tanh, sigmoid)

Update

Recurrent Neural Network (RNN)

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



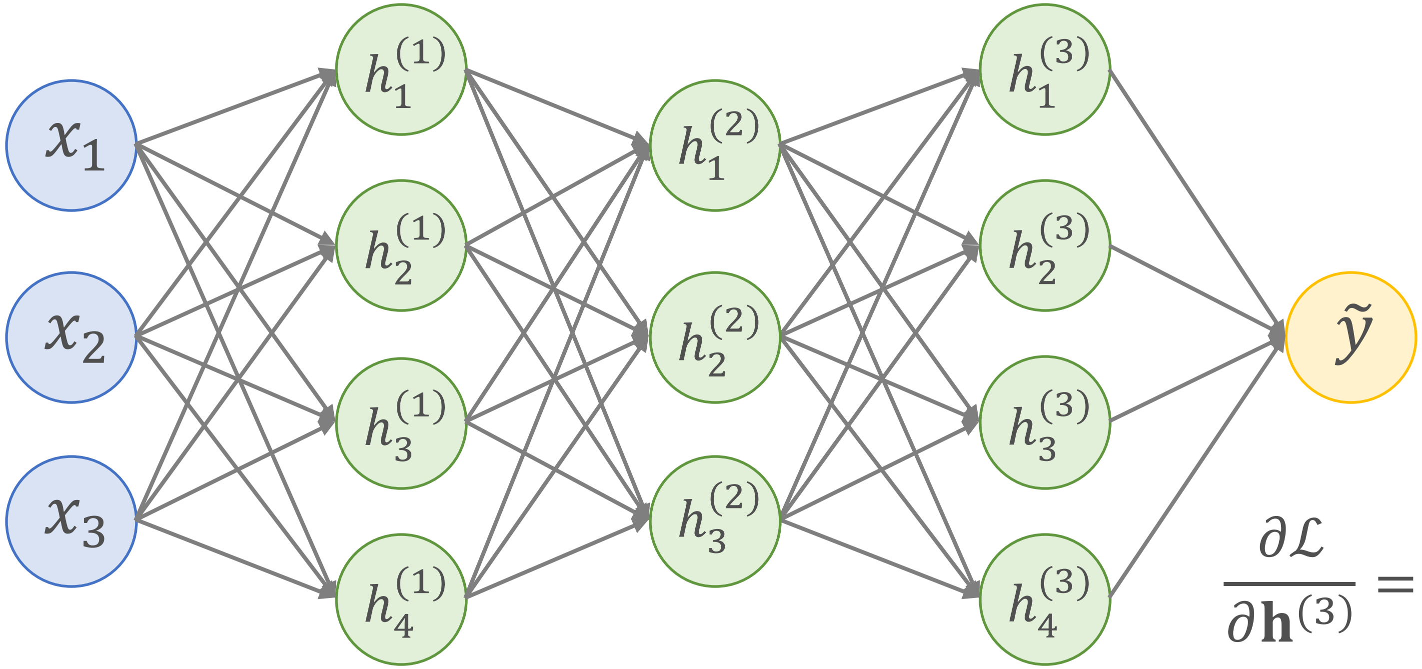


$$\mathcal{L}_{CE}(y, \tilde{y}) = -\sum_{c=0}^{C} y_c \log P(y = c|\, \mathbf{x})$$

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



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

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

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

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

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

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

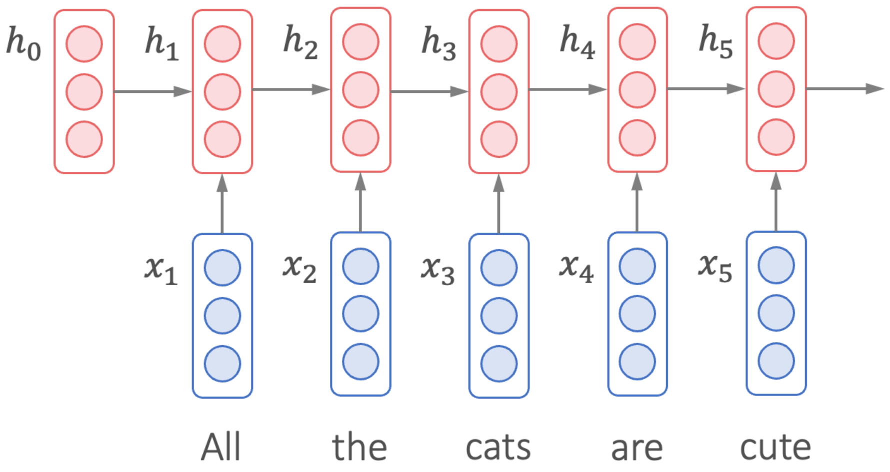
Vanishing Gradient Problem

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

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

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

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



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

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

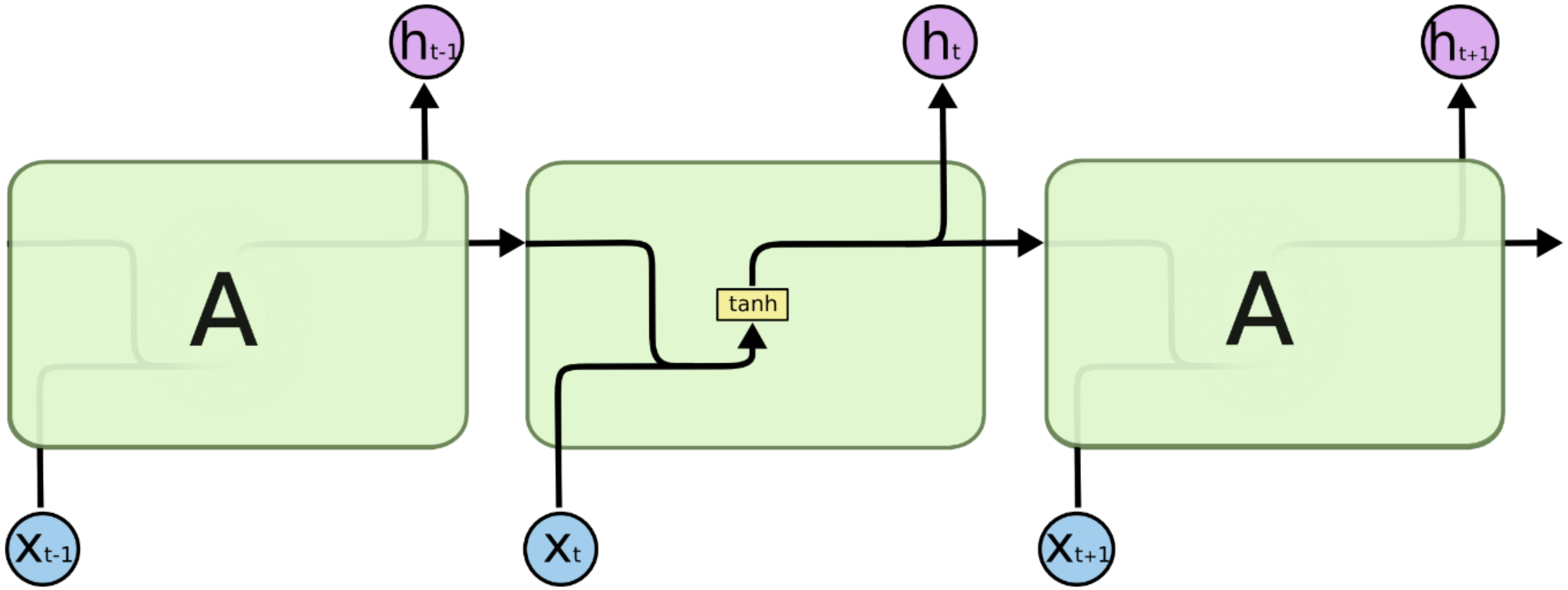
When these are small, the gradient signal gets smaller and smaller as it back-propagates further

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

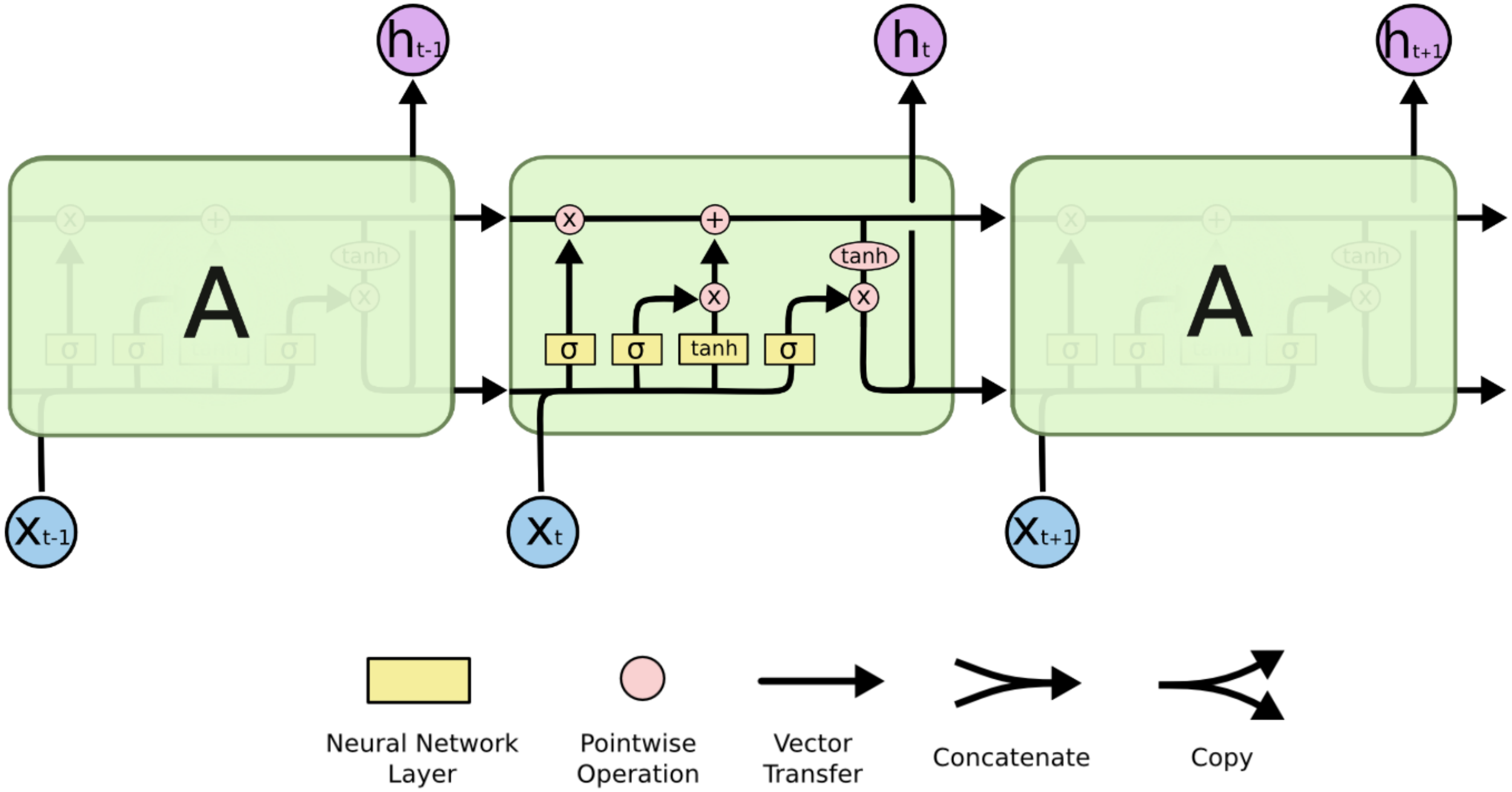
Recurrent Neural Network (RNN)



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

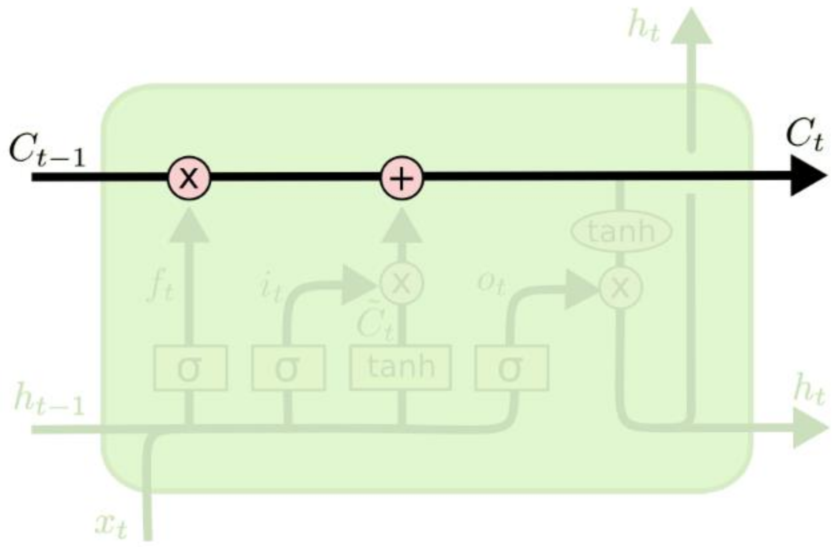
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)



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

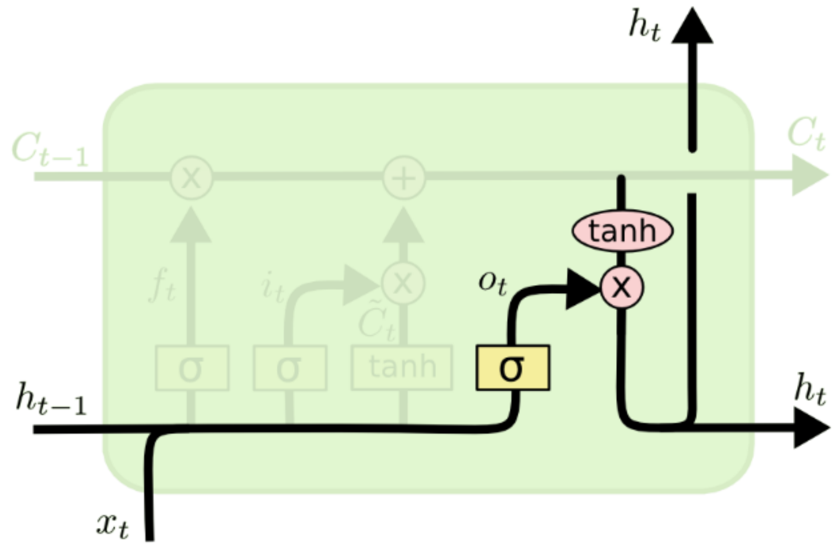
Long Short-Term Memory (LSTM)



The cell state stores long-term information

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

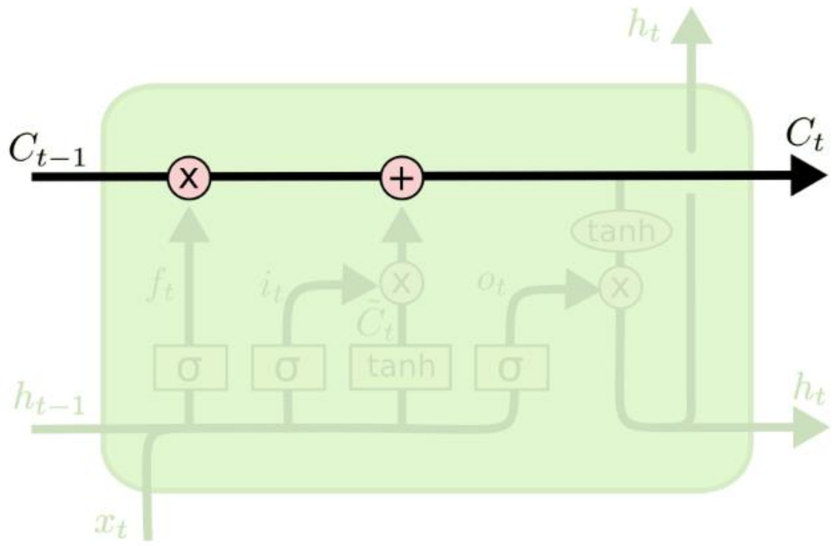
Long Short-Term Memory (LSTM)



The hidden state stores short-term information

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

Long Short-Term Memory (LSTM)

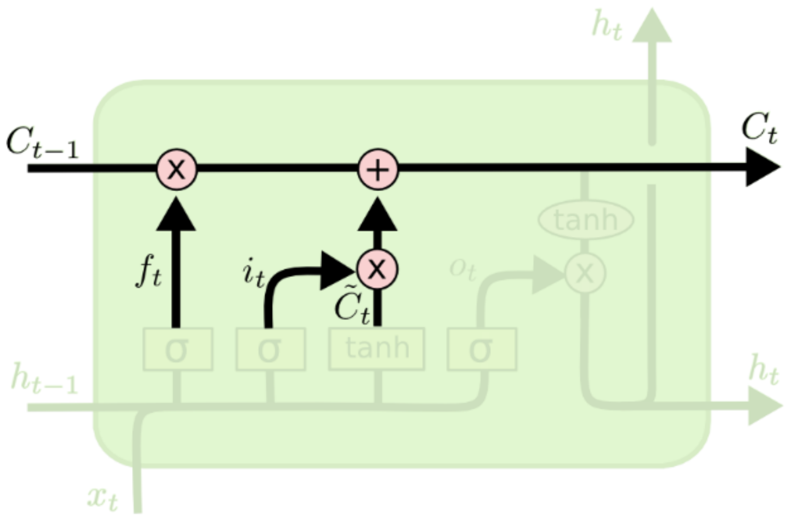


The cell state stores long-term information

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)



What we should write

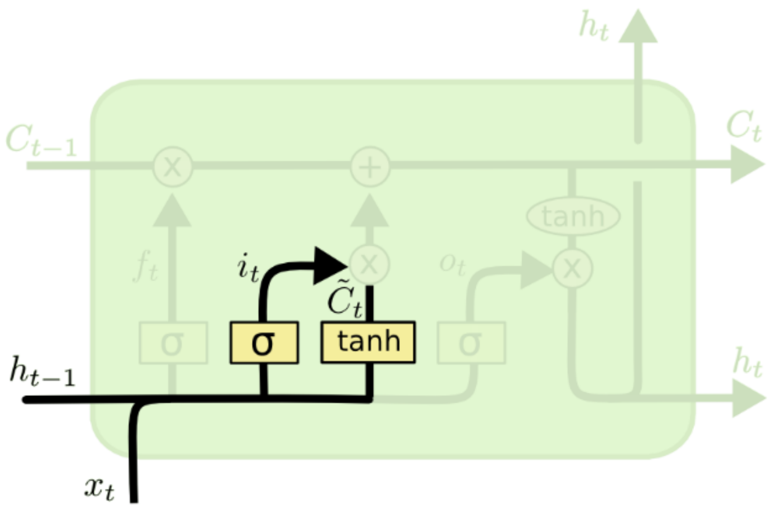
Update cell state $$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

How much we should erase

How much we should write

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

Long Short-Term Memory (LSTM)



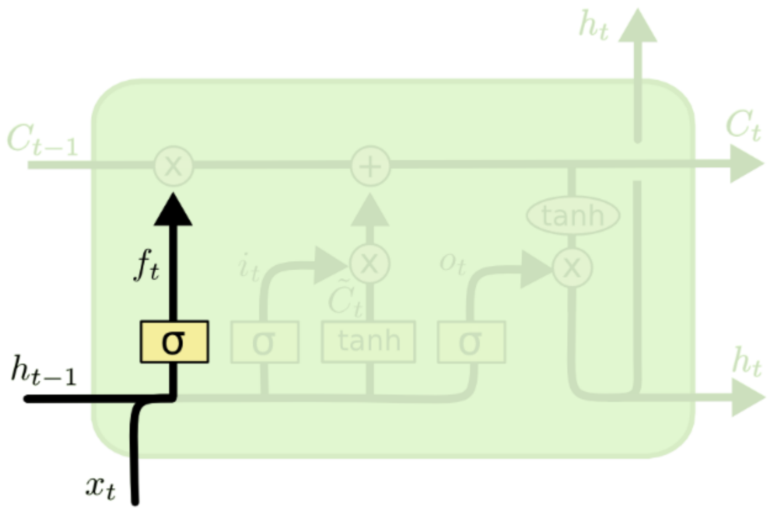
New information $\tilde{C}_t = \tanh\left(W^{(C)}h_{t-1} + U^{(C)}x_t + b^{(C)}\right)$ What we should write

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

Sigmoid function: gate values are between 0 and 1

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

Long Short-Term Memory (LSTM)

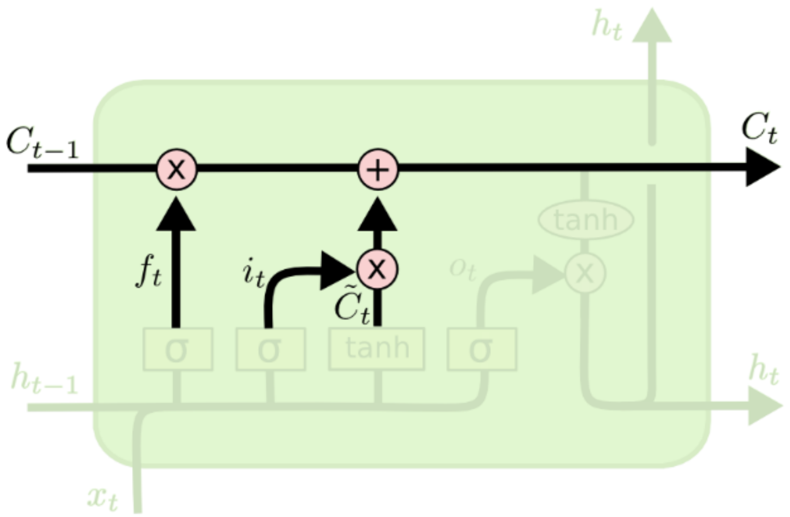


Forget gate $$f_t = \sigma\left(W^{(f)}h_{t-1} + U^{(f)}x_t + b^{(f)}\right)$$ How much we should erase

Sigmoid function: gate values are between 0 and 1

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

Long Short-Term Memory (LSTM)



What we should write

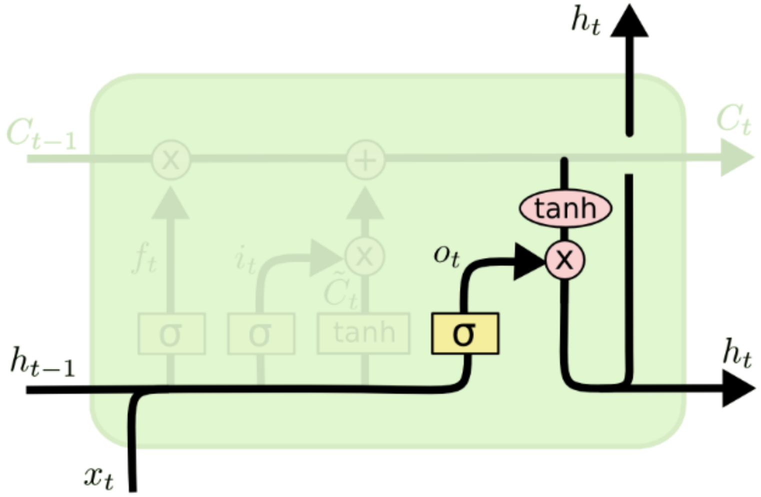
Update cell state $$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

How much we should erase

How much we should write

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

Long Short-Term Memory (LSTM)



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

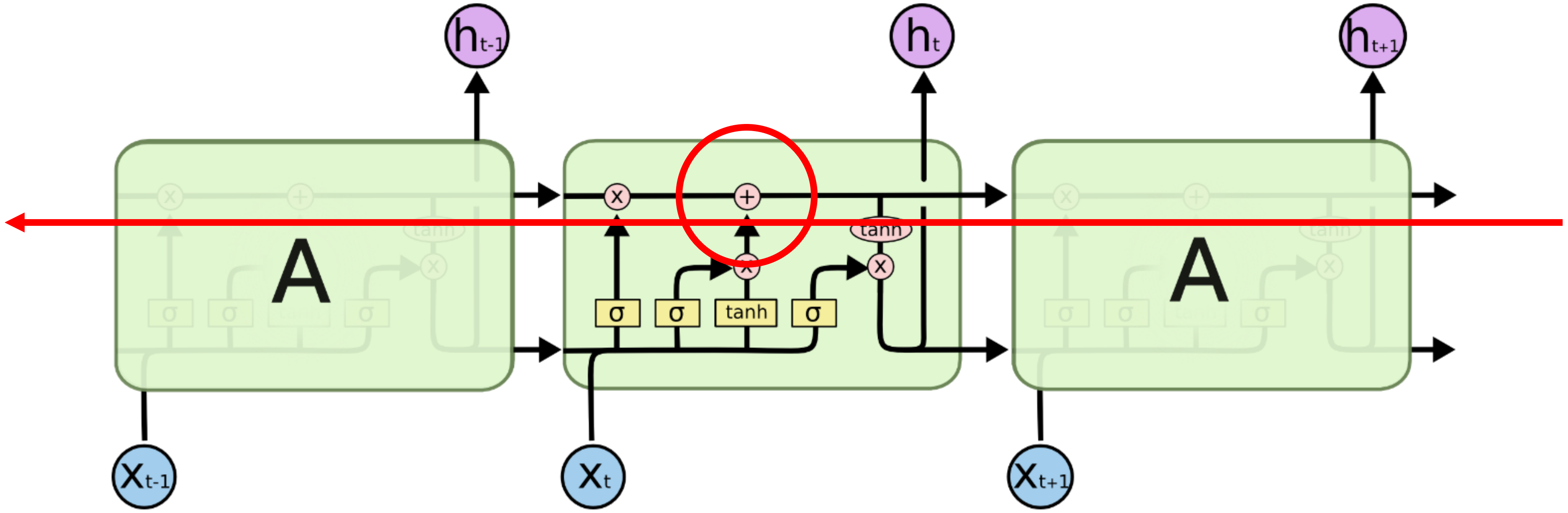
Update hidden state

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

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

Long Short-Term Memory (LSTM)

Uninterrupted gradient flow



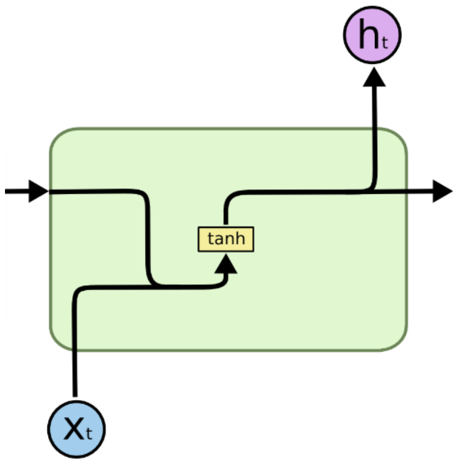
The addition is the key

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/

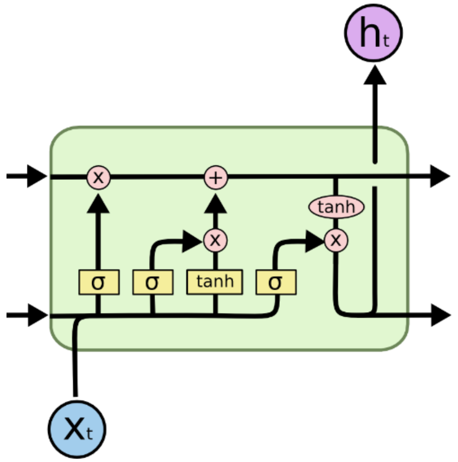
RNN vs. LSTM

RNN



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

LSTM



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

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

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

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

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

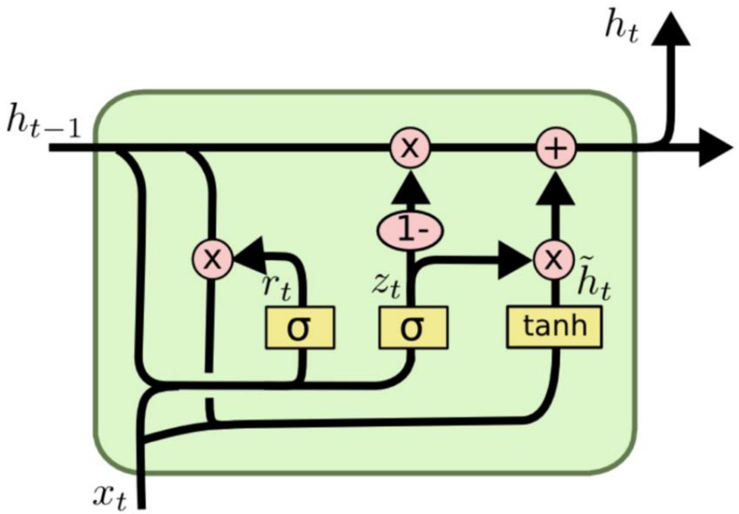
$$h_t = o_t * \tanh(C_t)$$

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

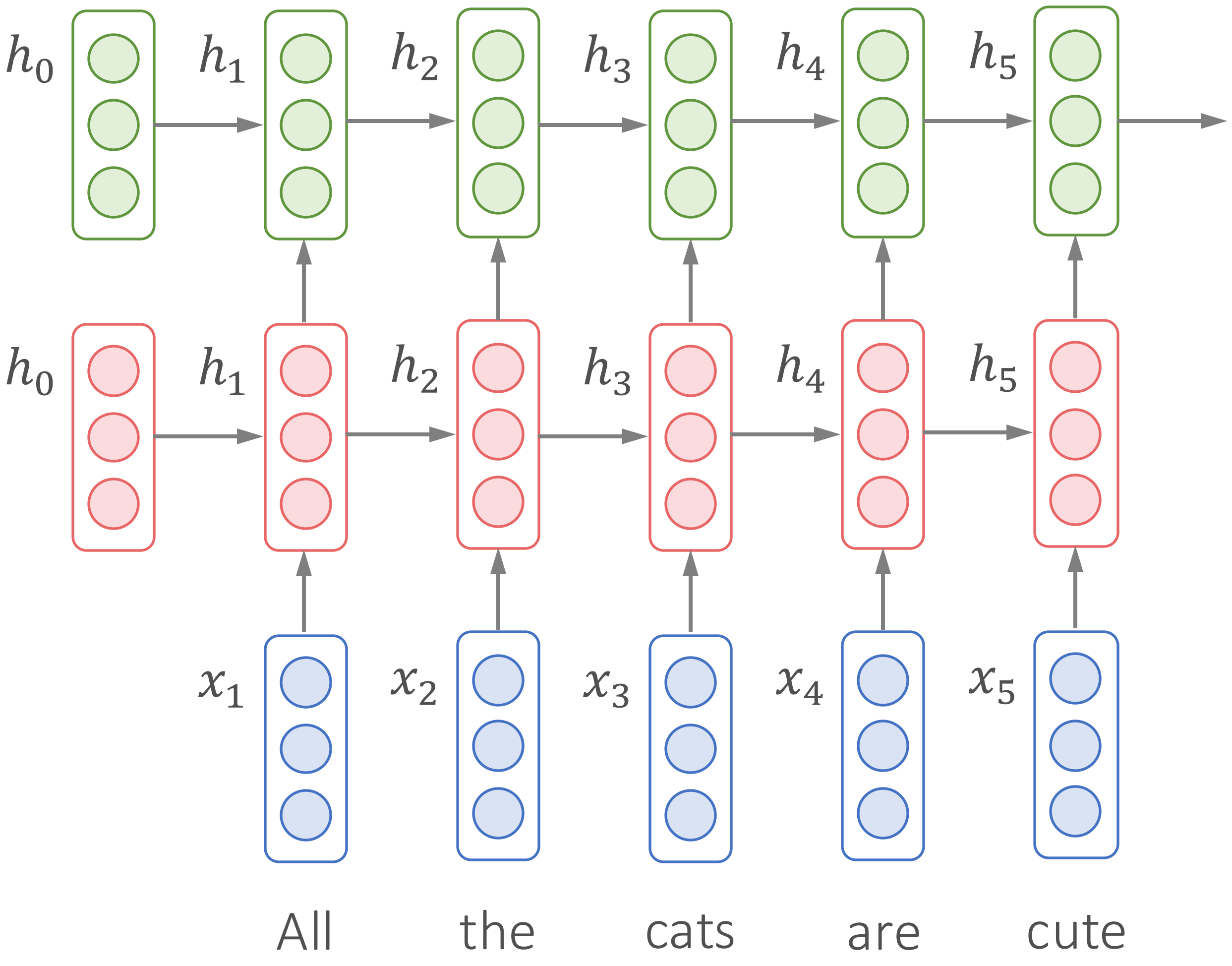
New hidden state

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

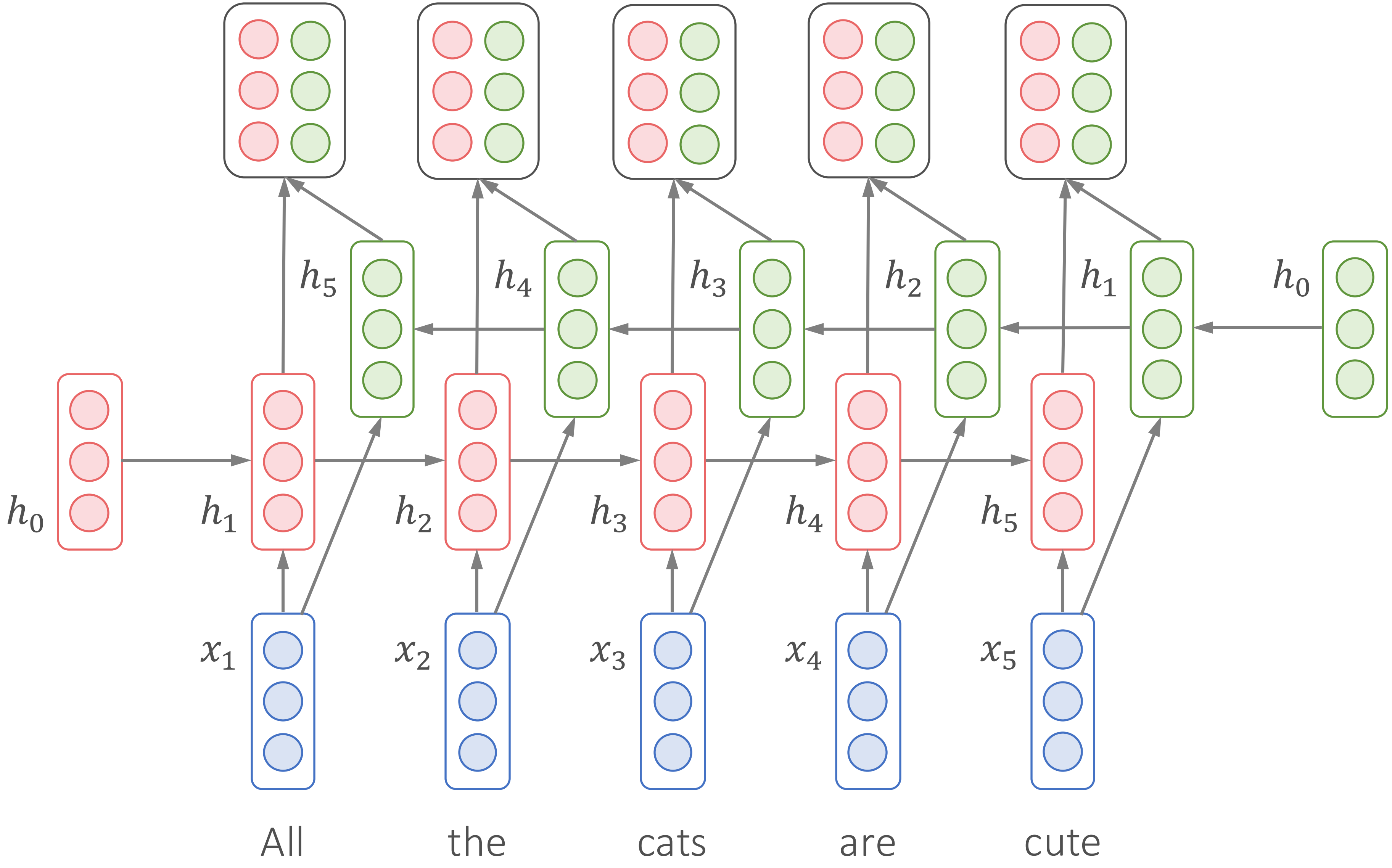
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Merge input and forget gate

Multi-Layer RNN (Stacked RNN)



Bidirectional RNN



Lecture Plan

- Convolutional Neural Network
- Recurrent Neural Network
  - Long Short-Term Memory
  - Gated Recurrent Units