# ECE 884 Deep Learning

Lecture 17-18: Recurrent Neural Networks (RNN)

03/23/25/2021

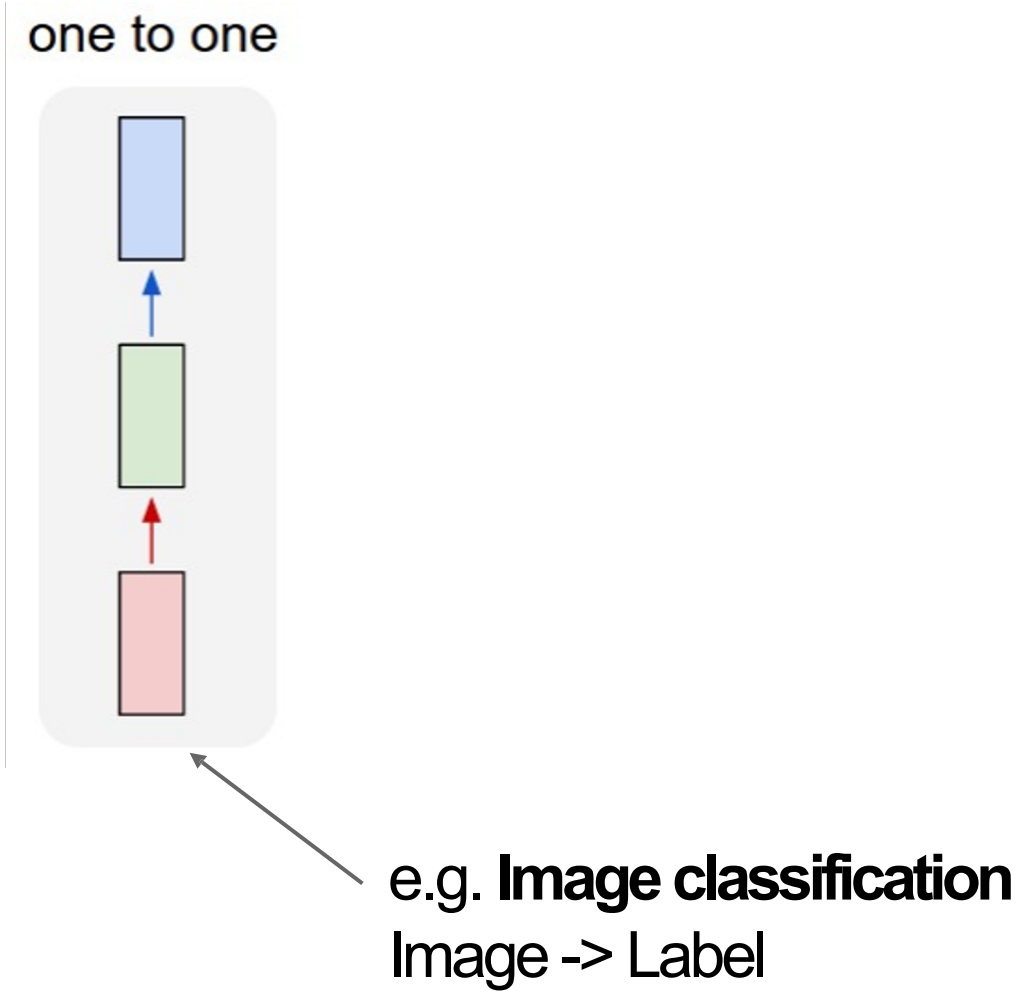
# Review of last lecture

- Convolutional Neural Network (ConvNet)

# Today’s lecture

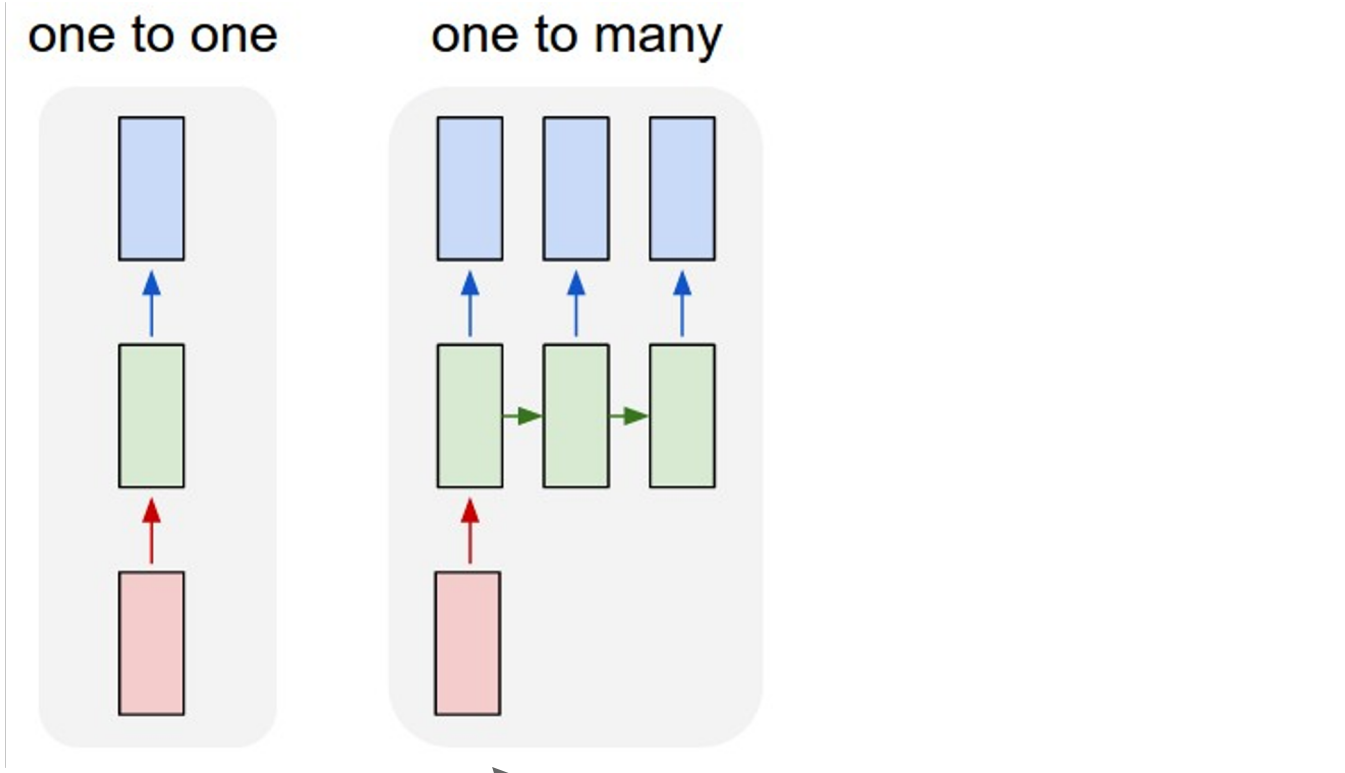
- Recurrent Neural Network (RNN)
  - Word Embeddings
  - Simple / Vanilla RNN
  - Long Short Term Memory (LSTM)
  - Making RNNs More Effective

# So far: “Feedforward” NeuralNetworks



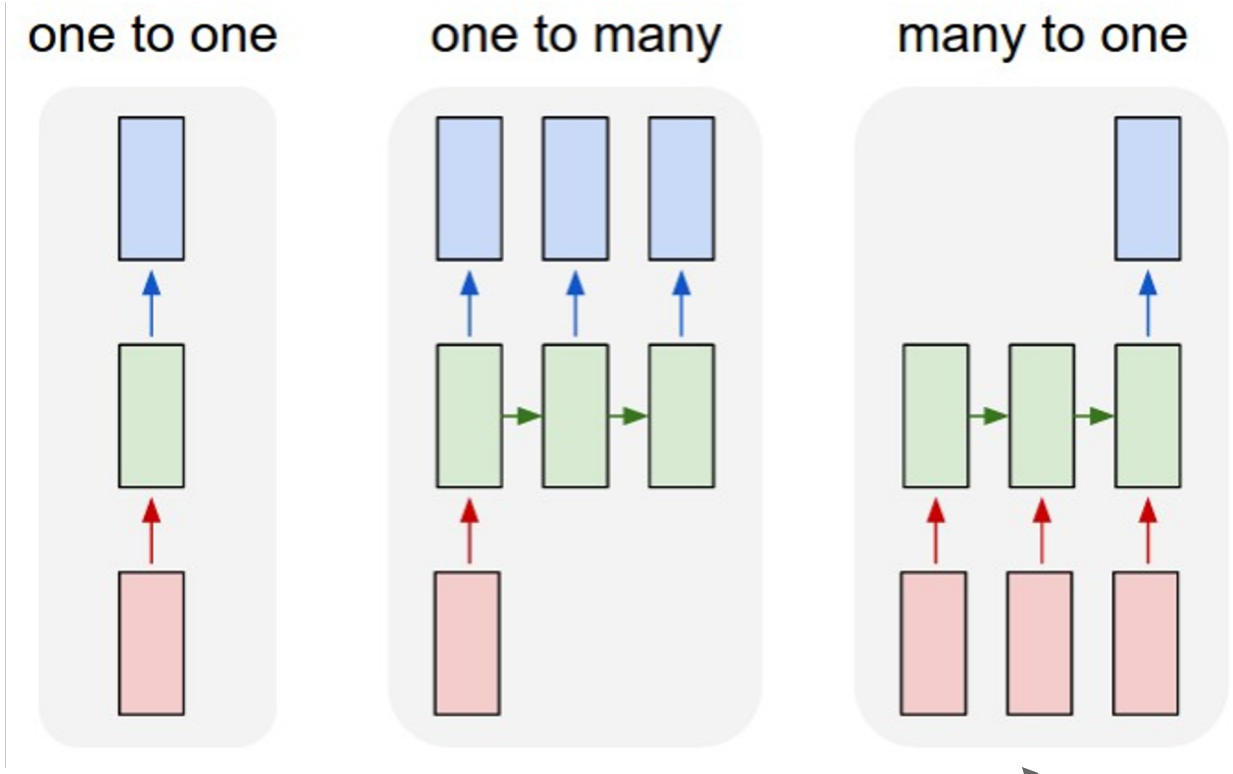

- Limitations of FCNets and ConvNets:
  - Process a paragraph as a whole.
  - Fixed-size input (e.g., image).
  - Fixed-size output (e.g., predicted probabilities).

# How to model sequential data?



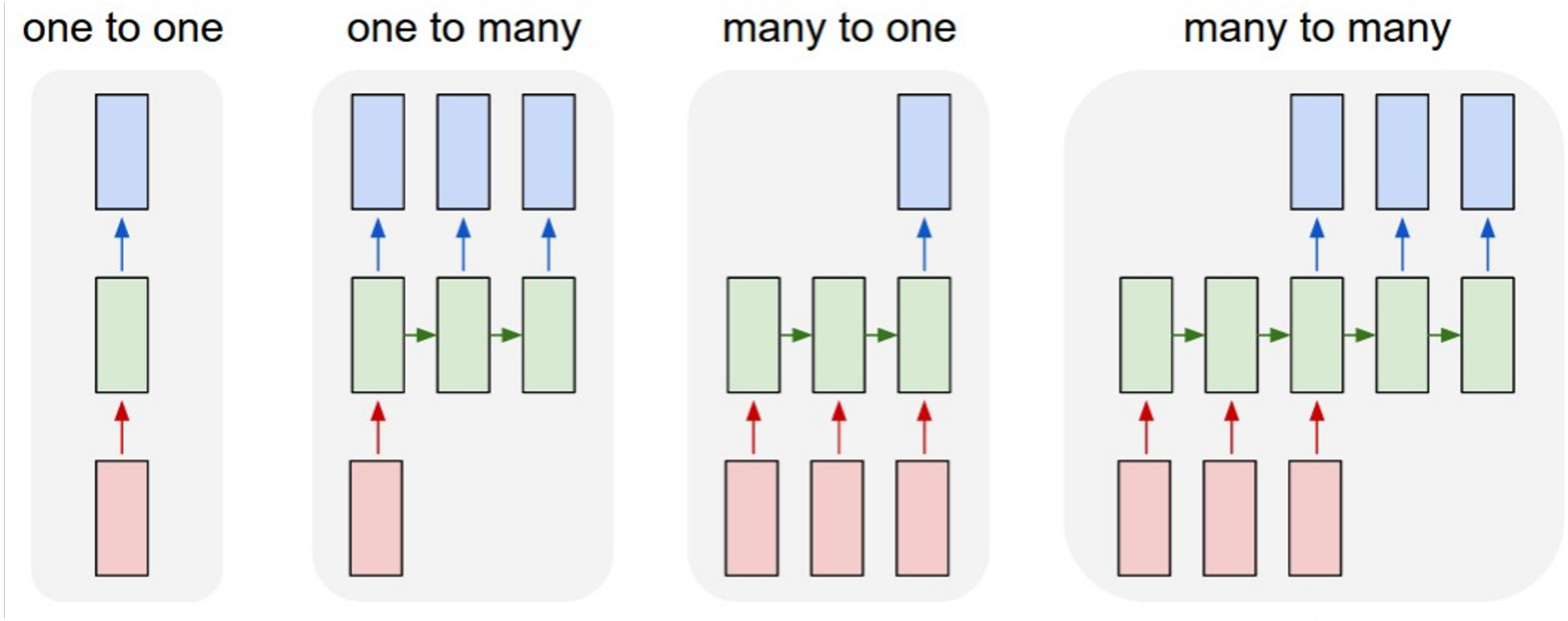

e.g. **Image Captioning**:
Image -> sequence of words

# How to model sequential data?



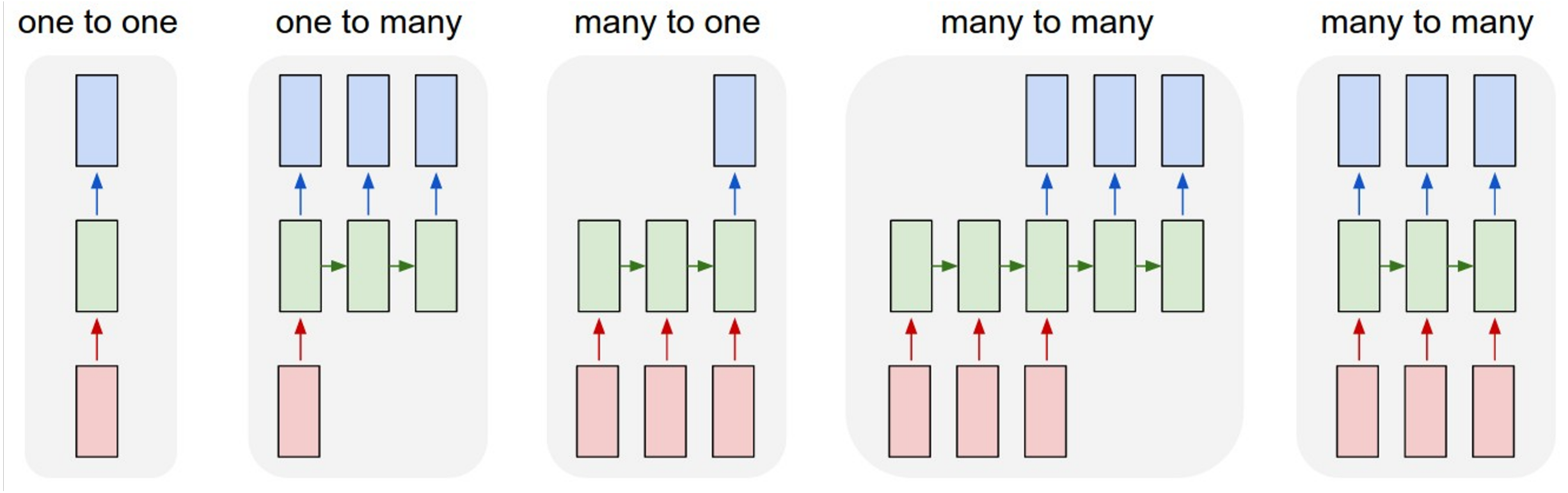

e.g. **Video classification:**
Sequence of images -> label

# How to model sequential data?




e.g. **Machine Translation:**
Sequence of words -> Sequence of words

# How to model sequential data?




e.g. **Per-frame video classification:**
Sequence of images -> Sequence of labels

# Representing Sequential Data

# Step 1: Tokenization (Text to Words)

- We are given a piece of text (string), e.g.,

```
S = "… to be or not to be…".
```

- Break the string (string) into a list of words:

```
L = […, to, be, or, not, to, be, …],
```

# Step 2: Count Word Frequencies

- Build a dictionary (e.g., hash table) to count words' frequencies.
- Initially, the dictionary is empty.

| Key (word) | Value (frequency) |
| --- | --- |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

# Step 2: Count Word Frequencies

- Update the dictionary in this way:
  - If word $w$ is **not** in the dictionary, add $(w, 1)$ to the dictionary.
  - If word $w$ is in the dictionary, increase its frequency counter.

| Key (word) | Value (frequency) |
|---|---|
| a | 219 |
| to | 398 |
| | |
| hamlet | 5 |
| | |
| | |
| be | 131 |
| not | 499 |
| prince | 12 |
| | |
| kill | 31 |

# Step 2: Count Word Frequencies

- Update the dictionary in this way:
  - If word $w$ is **not** in the dictionary, add $(w,1)$ to the dictionary.
  - If word $w$ is in the dictionary, increase its frequency counter.

| ... | to | be | or | not | to | be | ... |
|---|---|---|---|---|---|---|---|

| Key (word) | Value (frequency) |
|---|---|
| a | 219 |
| to | 398 |
| | |
| hamlet | 5 |
| | |
| | |
| be | 131 |
| not | 499 |
| prince | 12 |
| | |
| kill | 31 |

# Step 2: Count Word Frequencies

- Update the dictionary in this way:
  - If word $w$ is **not** in the dictionary, add $(w, 1)$ to the dictionary.
  - If word $w$ is in the dictionary, increase its frequency counter.

| ... | to | be | or | not | to | be | ... |
|---|---|---|---|---|---|---|---|

- Word "to" is in the dictionary.

| Key (word) | Value (frequency) |
|---|---|
| a | 219 |
| to | 398 |
| | |
| hamlet | 5 |
| | |
| | |
| be | 131 |
| not | 499 |
| prince | 12 |
| | |
| kill | 31 |

# Step 2: Count Word Frequencies

- Update the dictionary in this way:
  - If word $w$ is **not** in the dictionary, add $(w, 1)$ to the dictionary.
  - If word $w$ is in the dictionary, increase its frequency counter.

| ... | to | be | or | not | to | be | ... |
|---|---|---|---|---|---|---|---|

- Word "to" is in the dictionary.
- Increase its counter.

| Key (word) | Value (frequency) |
|---|---|
| a | 219 |
| to | 399 |
| | |
| hamlet | 5 |
| | |
| | |
| be | 131 |
| not | 499 |
| prince | 12 |
| | |
| kill | 31 |

# Step 2: Count Word Frequencies

- Update the dictionary in this way:
  - If word $w$ is **not** in the dictionary, add $(w,1)$ to the dictionary.
  - If word $w$ is in the dictionary, increase its frequency counter.

| ... | to | be | or | not | to | be | ... |
|---|---|---|---|---|---|---|---|

- Word "be" is in the dictionary.

| Key (word) | Value (frequency) |
|---|---|
| a | 219 |
| to | 399 |
| | |
| hamlet | 5 |
| | |
| | |
| **be** | **131** |
| not | 499 |
| prince | 12 |
| | |
| kill | 31 |

# Step 2: Count Word Frequencies

- Update the dictionary in this way:
  - If word $w$ is **not** in the dictionary, add $(w, 1)$ to the dictionary.
  - If word $w$ is in the dictionary, increase its frequency counter.

| ... | to | be | or | not | to | be | ... |
|---|---|---|---|---|---|---|---|

- Word "be" is in the dictionary.
- Increase its counter.

| Key (word) | Value (frequency) |
|---|---|
| a | 219 |
| to | 399 |
| | |
| hamlet | 5 |
| | |
| | |
| be | 132 |
| not | 499 |
| prince | 12 |
| | |
| kill | 31 |

# Step 2: Count Word Frequencies

- Update the dictionary in this way:
  - If word $w$ is **not** in the dictionary, add $(w,1)$ to the dictionary.
  - If word $w$ is in the dictionary, increase its frequency counter.

| ... | to | be | or | not | to | be | ... |
|---|---|---|---|---|---|---|---|

- Word "or" is not in the dictionary.

| Key (word) | Value (frequency) |
|---|---|
| a | 219 |
| to | 399 |
| | |
| hamlet | 5 |
| | |
| | |
| be | 132 |
| not | 499 |
| prince | 12 |
| | |
| kill | 31 |

# Step 2: Count Word Frequencies

- Update the dictionary in this way:
  - If word $w$ is **not** in the dictionary, add $(w, 1)$ to the dictionary.
  - If word $w$ is in the dictionary, increase its frequency counter.

| ... | to | be | or | not | to | be | ... |
|---|---|---|---|---|---|---|---|

- Word "or" is not in the dictionary.
- Add ("or", 1) to the dictionary.

| Key (word) | Value (frequency) |
|---|---|
| a | 219 |
| to | 399 |
| | |
| hamlet | 5 |
| | |
| or | 1 |
| be | 132 |
| not | 499 |
| prince | 12 |
| | |
| kill | 31 |

# Step 2: Count Word Frequencies

- Sort the table so that the frequency is in the descending order.

| Key (word) | Value (frequency) |
|---|---|
| a | 219 |
| to | 399 |
| | |
| hamlet | 5 |
| | |
| or | 1 |
| be | 132 |
| not | 499 |
| prince | 12 |
| | |
| kill | 31 |

# Step 2: Count Word Frequencies

- Sort the table so that the frequency is in the descending order.

| Key (word) | Value (frequency) |
|---|---|
| not | 499 |
| to | 399 |
| a | 219 |
| be | 132 |
| kill | 31 |
| prince | 12 |
| hamlet | 5 |
| or | 1 |
| | |
| | |
| | |

# Step 2: Count Word Frequencies

- Sort the table so that the frequency is in the descending order.
- Replace “frequency” by “index” (starting from 1.)

| Key (word) | Value (frequency) |
|---|---|
| not | 499 |
| to | 399 |
| a | 219 |
| be | 131 |
| kill | 31 |
| prince | 12 |
| hamlet | 5 |
| or | 1 |
| | |
| | |
| | |

# Step 2: Count Word Frequencies

- Sort the table so that the frequency is in the descending order.
- Replace “frequency” by “index” (starting from 1.)
- The number of unique words is called “vocabulary”.

| Key (word) | Value (index) |
|---|---|
| not | 1 |
| to | 2 |
| a | 3 |
| be | 4 |
| kill | 5 |
| prince | 6 |
| hamlet | 7 |
| or | 8 |
| | |
| | |
| | |

# Step 3: One-Hot Encoding

- Map every word to its index.
- For example,

```
Words: [to, be, or, not, to, be]
```

```
Indices: [2,  4,  8,  1,  2,  4]
```

| Key (word) | Value (index) |
|---|---|
| not | 1 |
| to | 2 |
| a | 3 |
| be | 4 |
| kill | 5 |
| prince | 6 |
| hamlet | 7 |
| or | 8 |
| | |
| | |
| | |

# Step 3: One-Hot Encoding

- Map every word to its index.
- For example,

```
Words: [to, be, or, not, to, be]
```

```
Indices: [2,  4,  8,  1,  2,  4]
```

- Convert every index to a one-hot vector.
  - The one-hot vector' dimension is the vocabulary.
  - Vocabulary means # of unique words in the dictionary.

| Key (word) | Value (index) |
|---|---|
| not | 1 |
| to | 2 |
| a | 3 |
| be | 4 |
| kill | 5 |
| prince | 6 |
| hamlet | 7 |
| or | 8 |
| | |
| | |
| | |

# Word Embedding: Word to Vector

# How to map word to vector?

| Word | Index | |
|---|---|---|
| “movie” | 1 | |
| “good” | 2 | |
| “fun” | 3 | |
| “boring” | 4 | |
| … | … | |

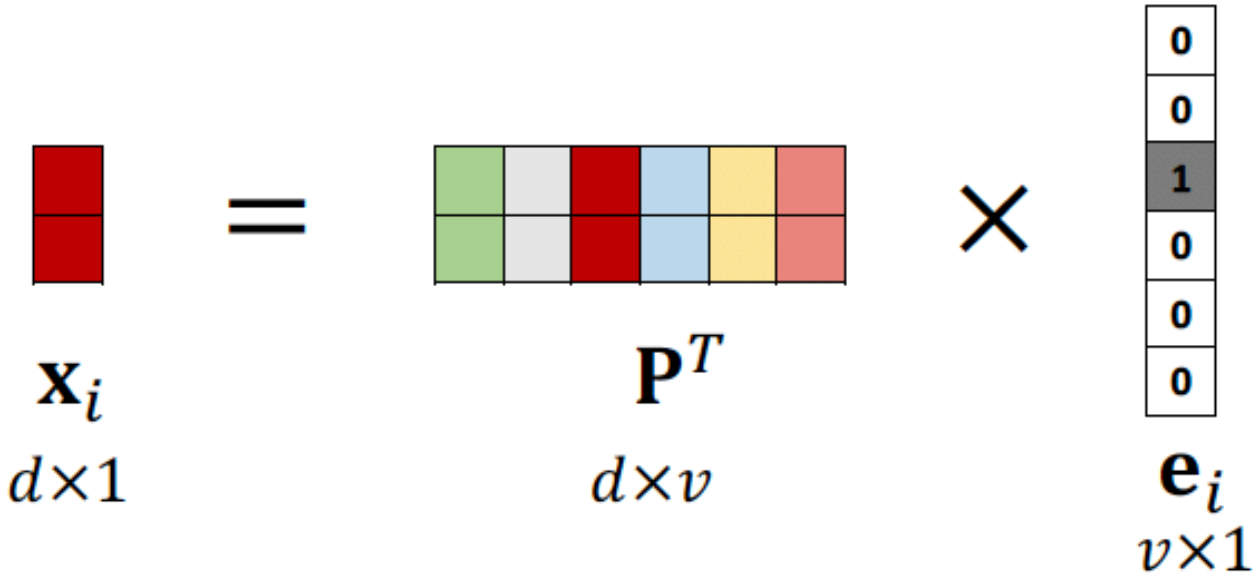
# One-Hot Encoding

- First, represent words using one-hot vectors.
  - Suppose the dictionary contains $v$ unique words (vocabulary= $v$).
  - Then the one-hot vectors are $v$-dimensional.

| Word | Index | One-hot encoding |
|---|---|---|
| "movie" | 1 | $[1, 0, 0, 0, 0, \cdots, 0]$ |
| "good" | 2 | $[0, 1, 0, 0, 0, \cdots, 0]$ |
| "fun" | 3 | $[0, 0, 1, 0, 0, \cdots, 0]$ |
| "boring" | 4 | $[0, 0, 0, 1, 0, \cdots, 0]$ |
| … | … | … |

# Word Embedding

- Second, map the one-hot vectors to low-dimensional vectors by



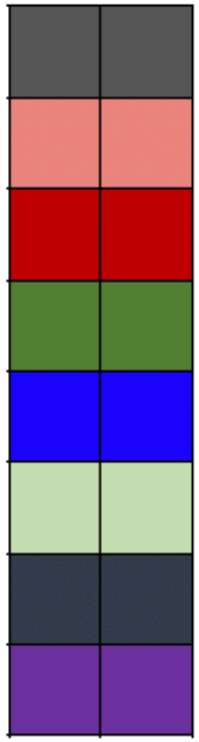

$\mathbf{x}_i$ $d \times 1$ $=$ $\mathbf{P}^T$ $d \times v$ $\times$ $\mathbf{e}_i$ $v \times 1$

- $\mathbf{P}$ is parameter matrix which can be learned from training data.
- $\mathbf{e}_i$ is the one-hot vector of the $i$-th word in dictionary.

# How to interpret the parameter matrix?

**Parameter matrix**

$$\mathbf{P} \in \mathbb{R}^{v \times d}$$



# How to interpret the parameter matrix?

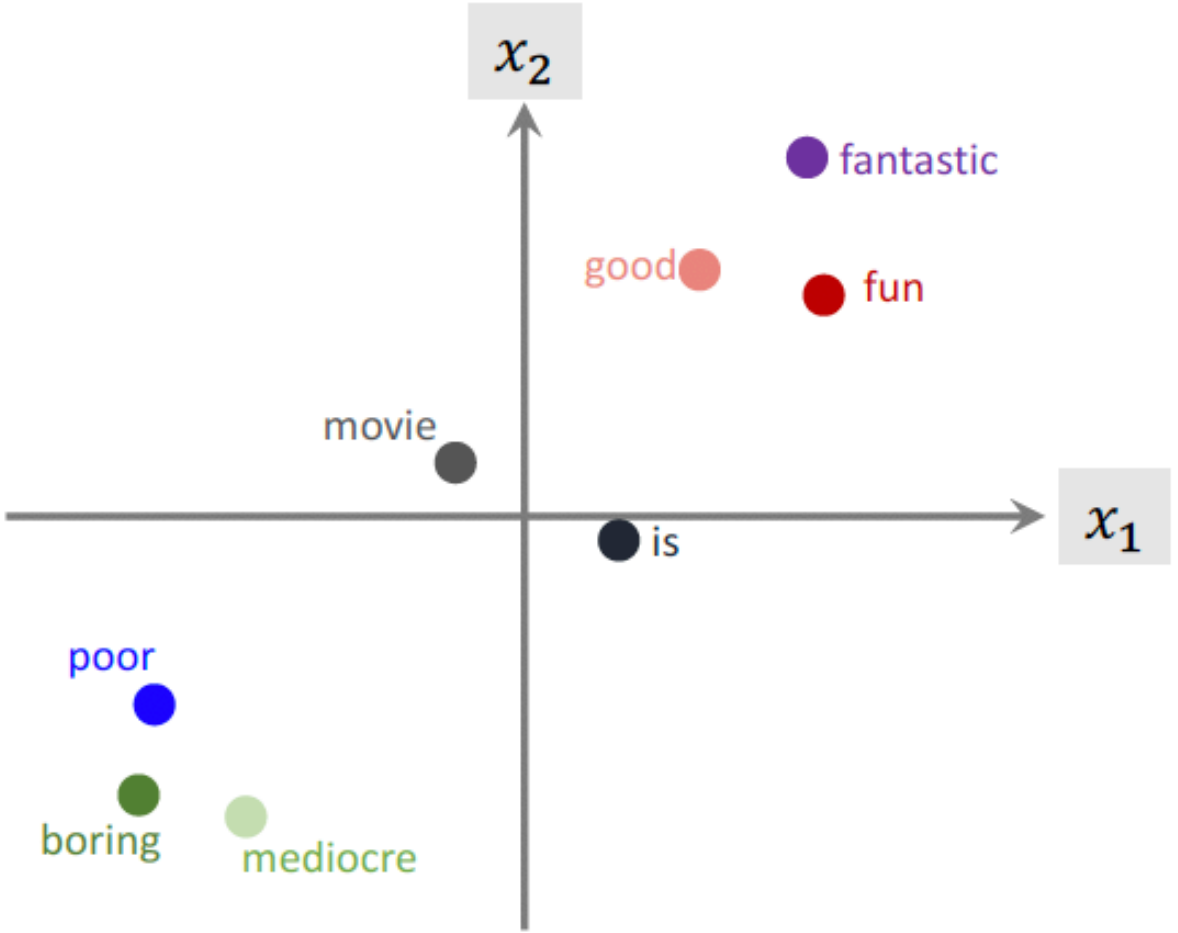
**Parameter matrix**

$$\mathbf{P} \in \mathbb{R}^{v \times d}$$

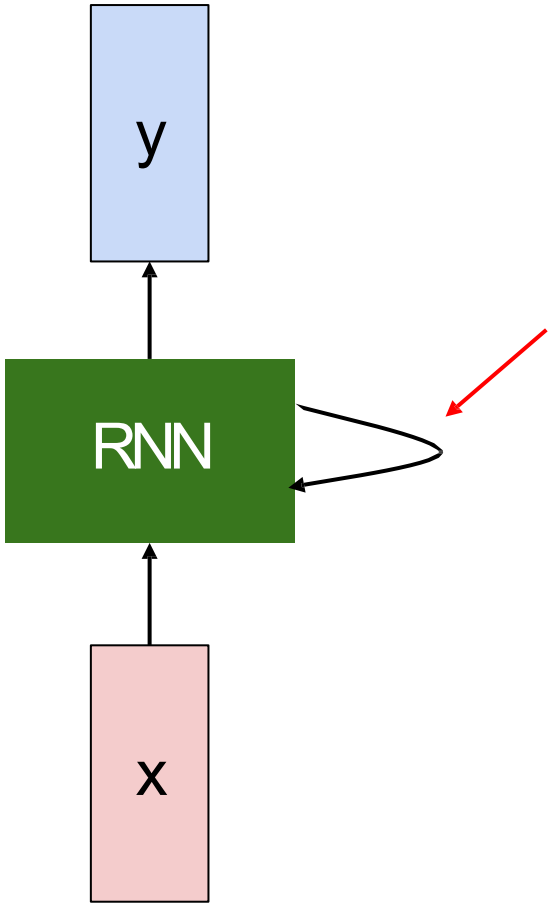
# How to interpret the parameter matrix?

**Parameter matrix**

$\mathbf{P} \in \mathbb{R}^{v \times d}$

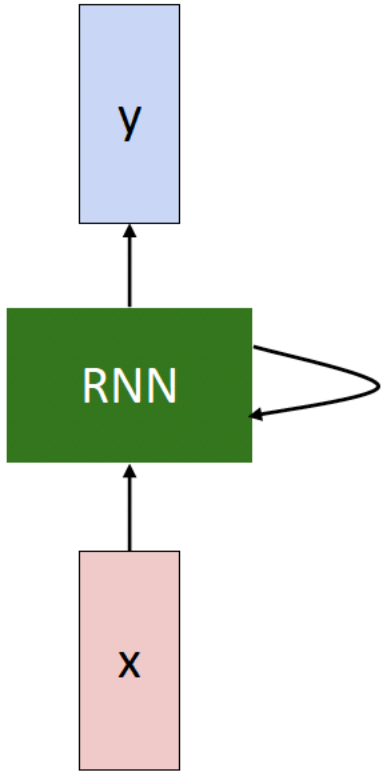



# Recurrent Neural Networks




Key idea: RNNs have an "internal state" that is updated as a sequence is processed

# Recurrent Neural Networks




We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state

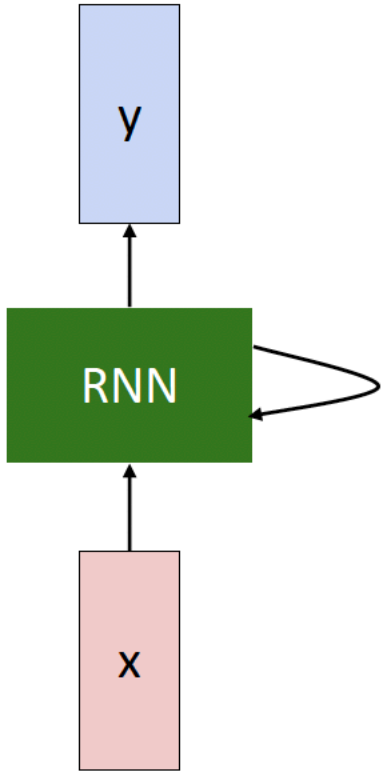
old state

input vector at some time step

some function with parameters W

# Recurrent Neural Networks

Notice: the same function and the same set of parameters are used at every time step.




We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

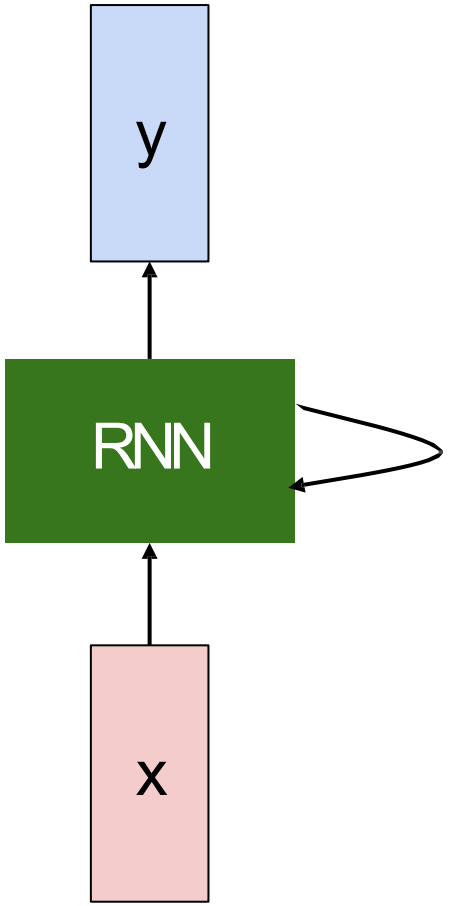
new state

some function with parameters W

old state

input vector at some time step

# (Vanilla) Recurrent Neural Networks



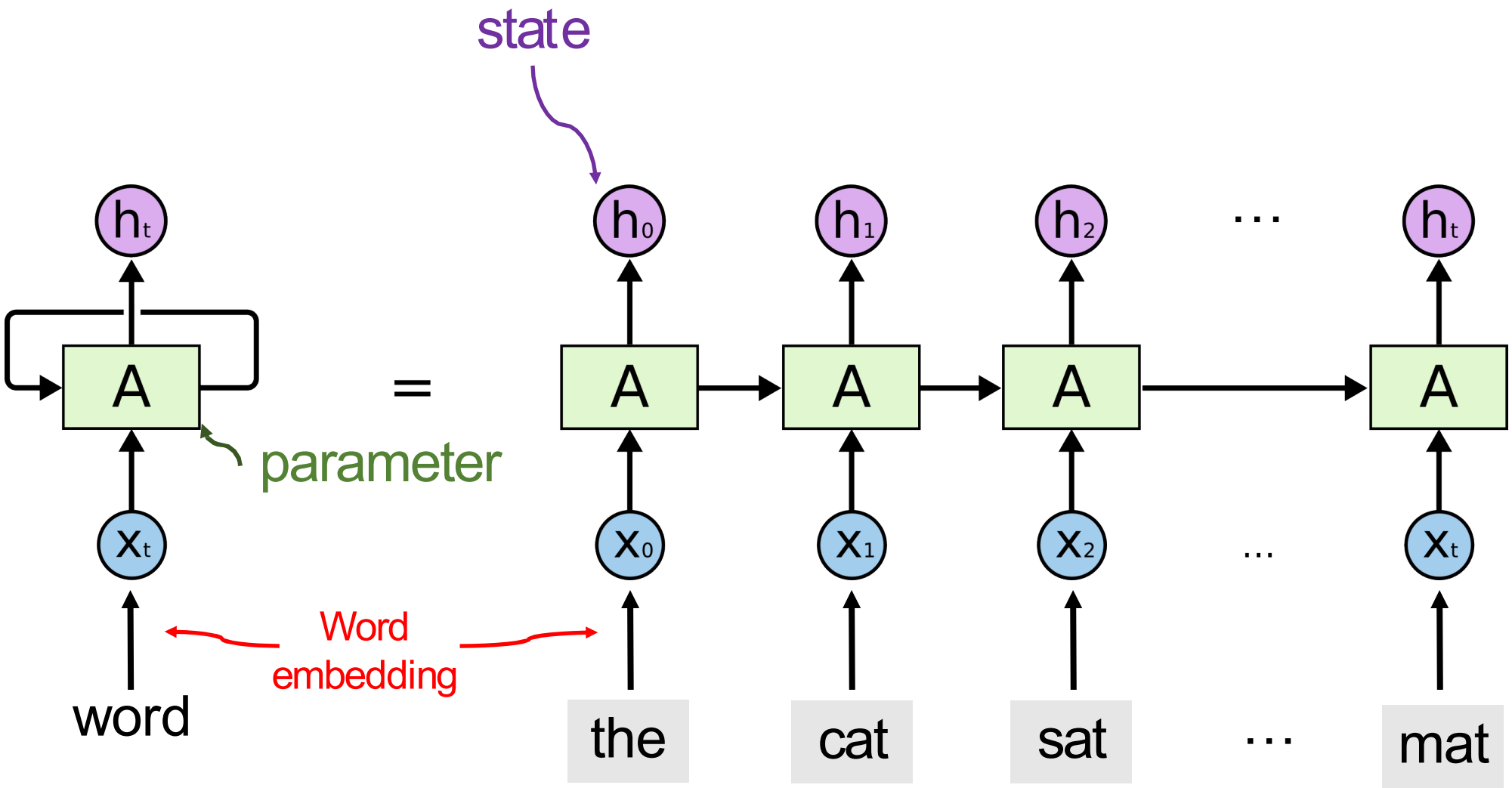

The state consists of a single *"hidden"* vector **h**:

$$h_t = f_W(h_{t-1}, x_t)$$

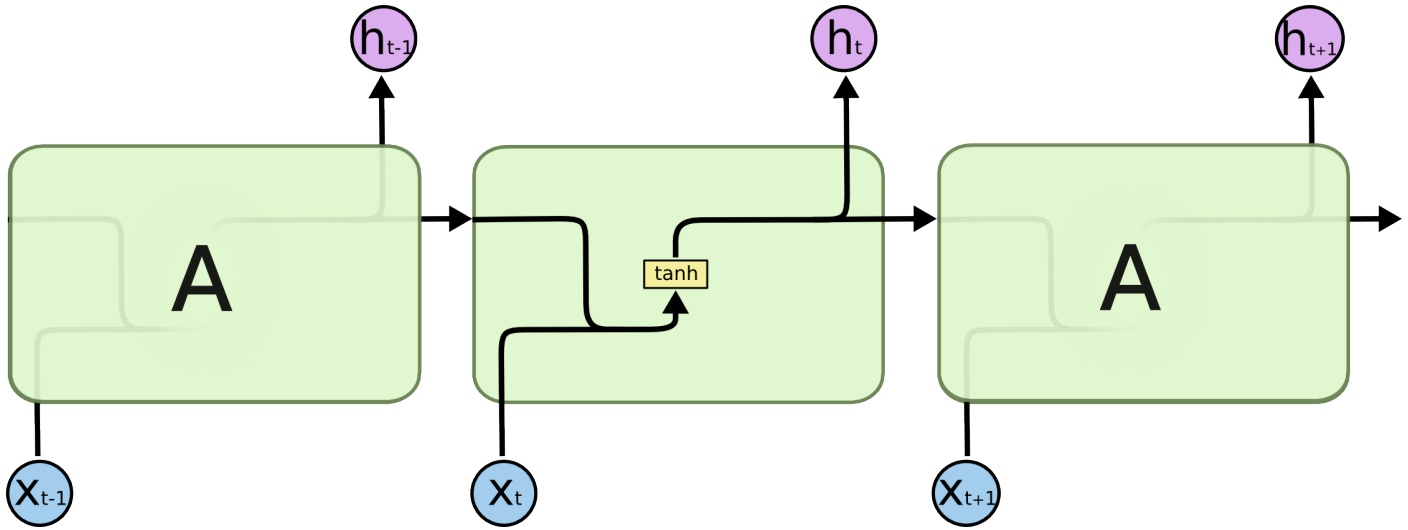
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

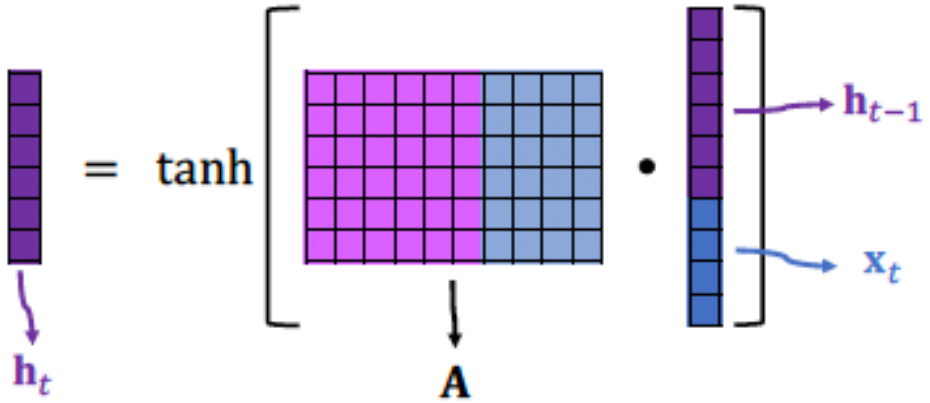
$$y_t = W_{hy}h_t + b_y$$

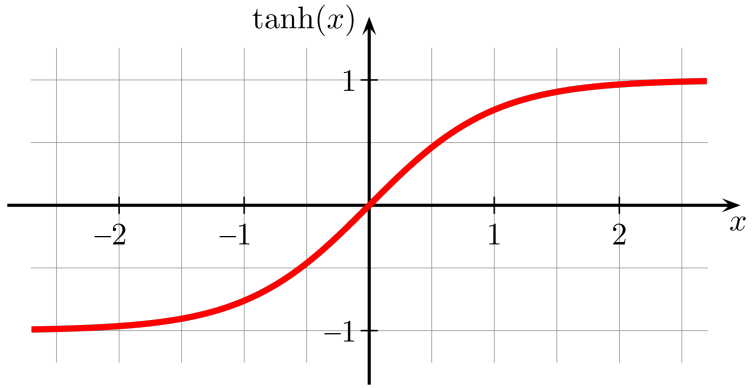
# Unfold


state
$h_t$
A
parameter
$x_t$
word
=
$h_0$
$h_1$
$h_2$
...
$h_t$
A
A
A
A
$x_0$
$x_1$
$x_2$
...
$x_t$
Word
embedding
the
cat
sat
...
mat

# A Closer Look



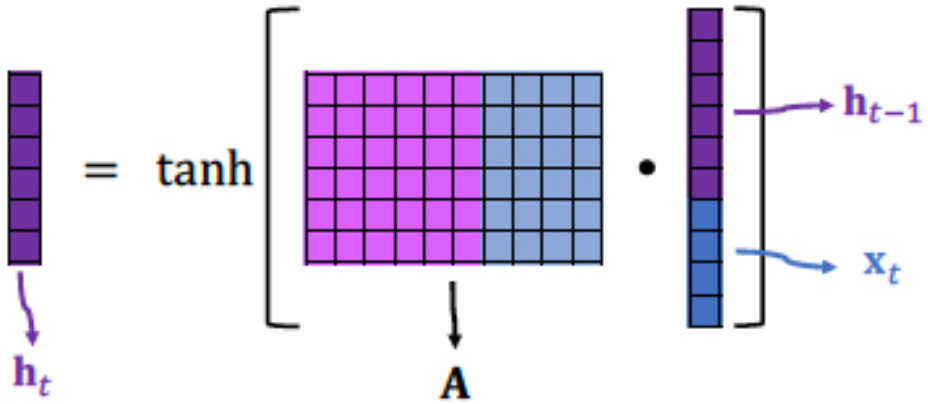



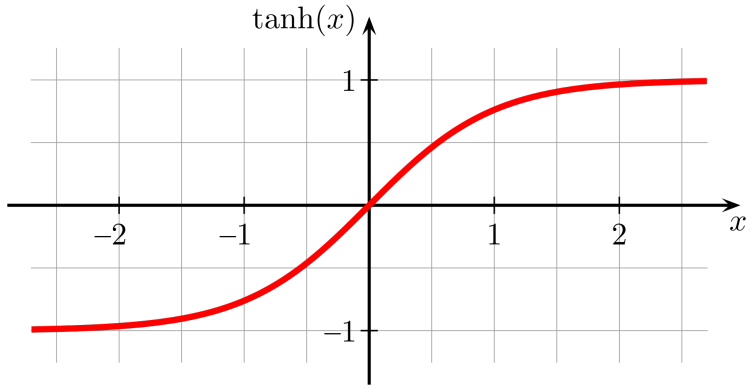




hyperbolic tangent function

# A Closer Look

**Question:** Why do we need the tanh function?



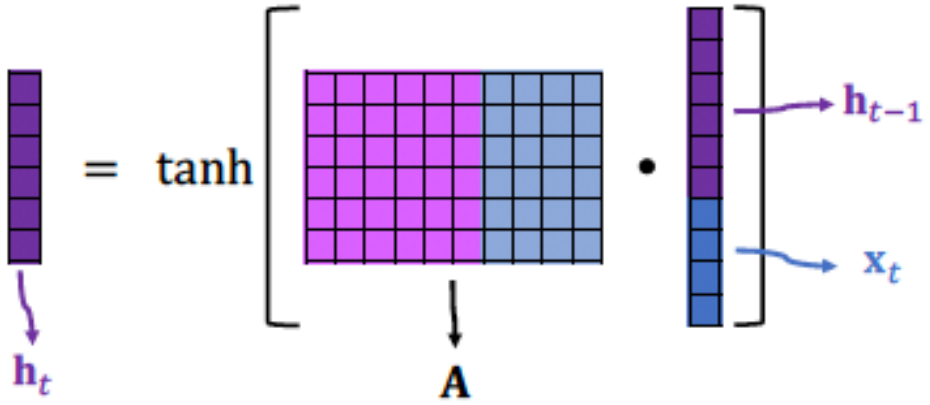



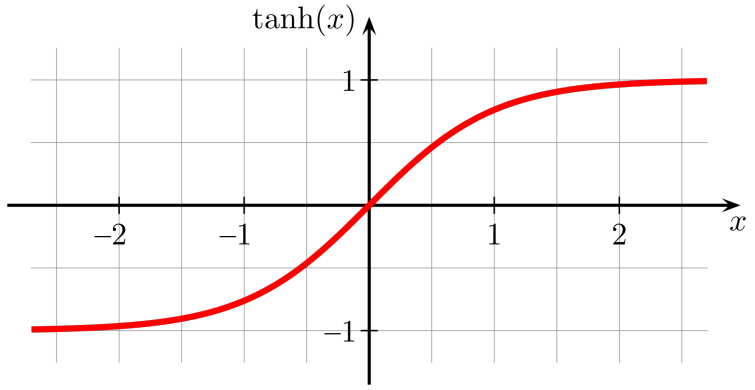

hyperbolic tangent function

# A Closer Look

**Question:** Why do we need the tanh function?

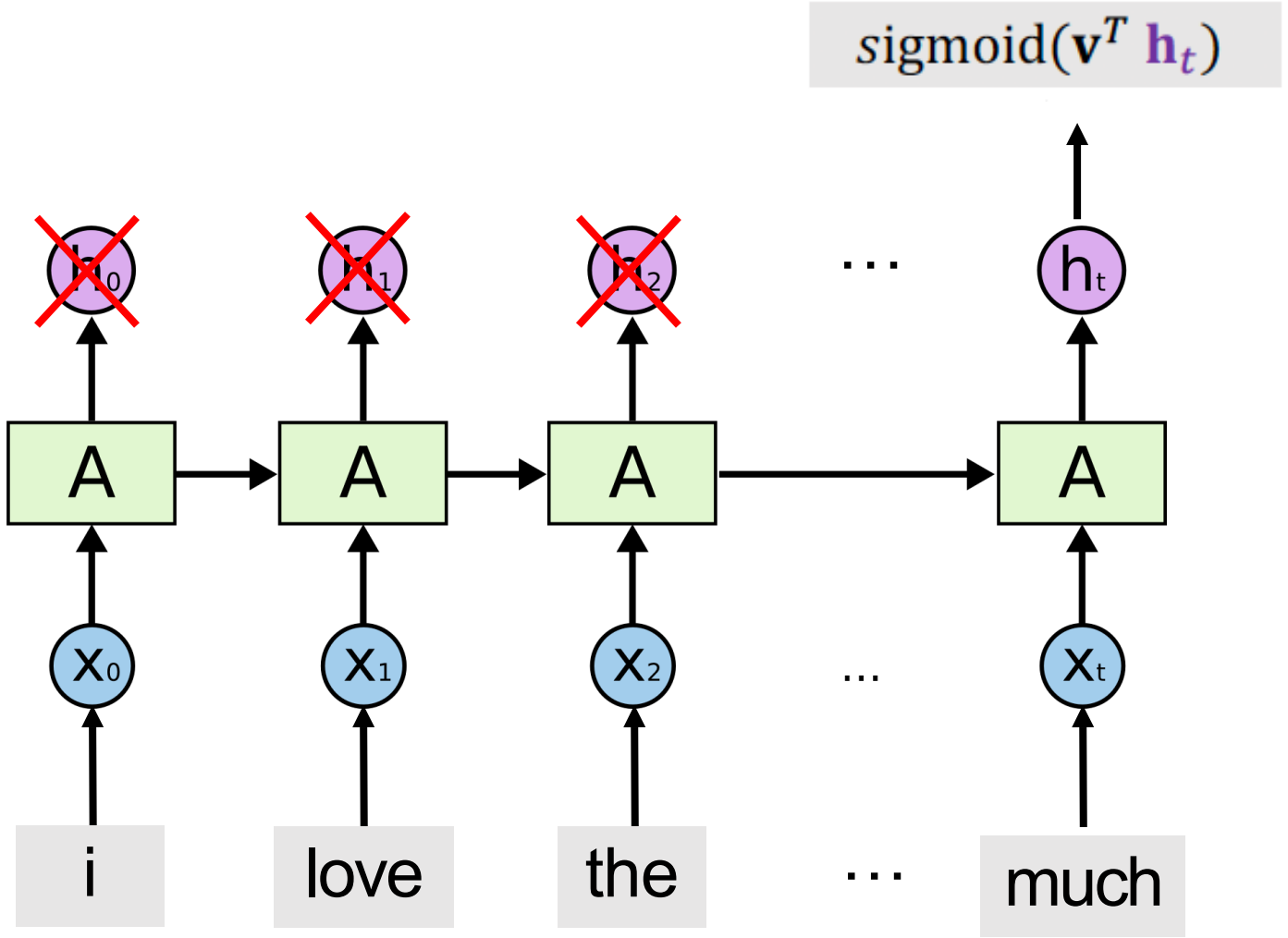
- Suppose $\mathbf{x}_0 = \cdots = \mathbf{x}_{100} = 0$.
- $\mathbf{h}_{100} = \mathbf{A}\mathbf{h}_{99} = \mathbf{A}^2\mathbf{h}_{98} = \cdots = \mathbf{A}^{100}\mathbf{h}_0$.
- What will happen if $\lambda_{\max}(\mathbf{A}) = 0.9$?
- What will happen if $\lambda_{\max}(\mathbf{A}) = 1.2$?



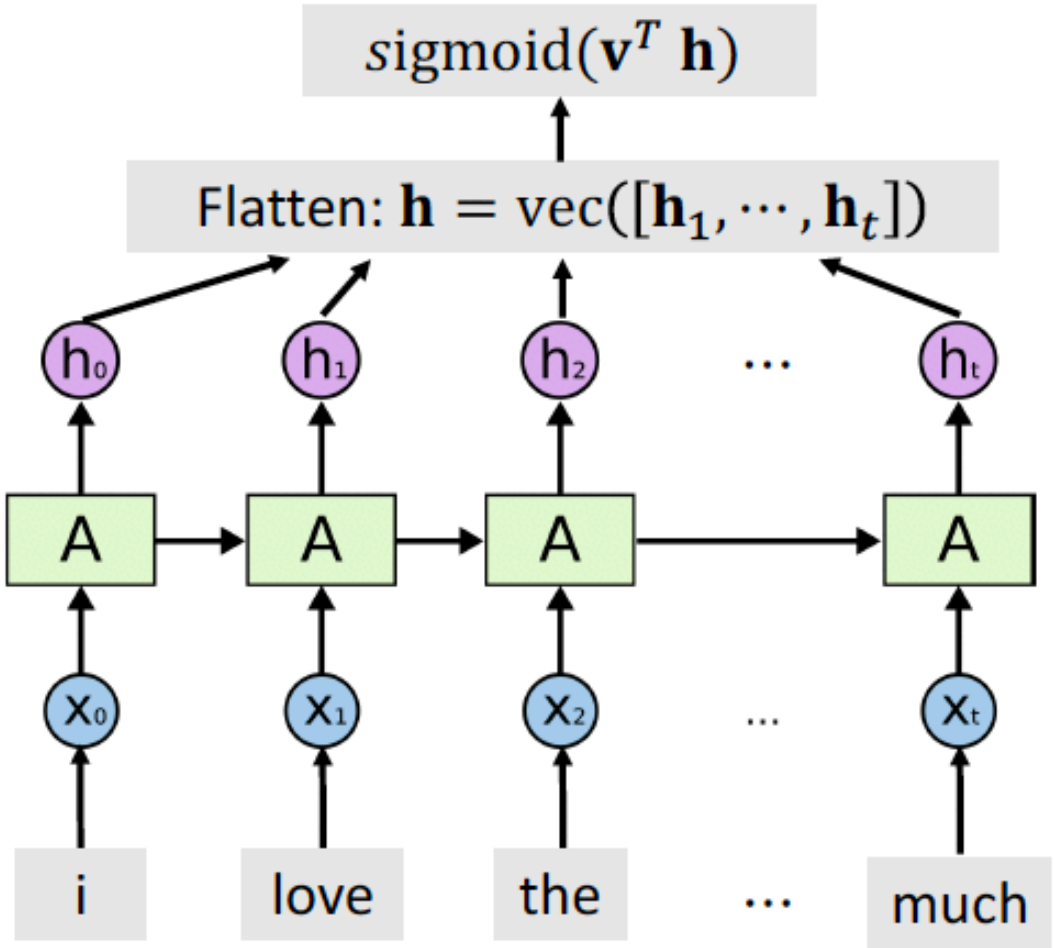




hyperbolic tangent function

# Simple RNN for Classification



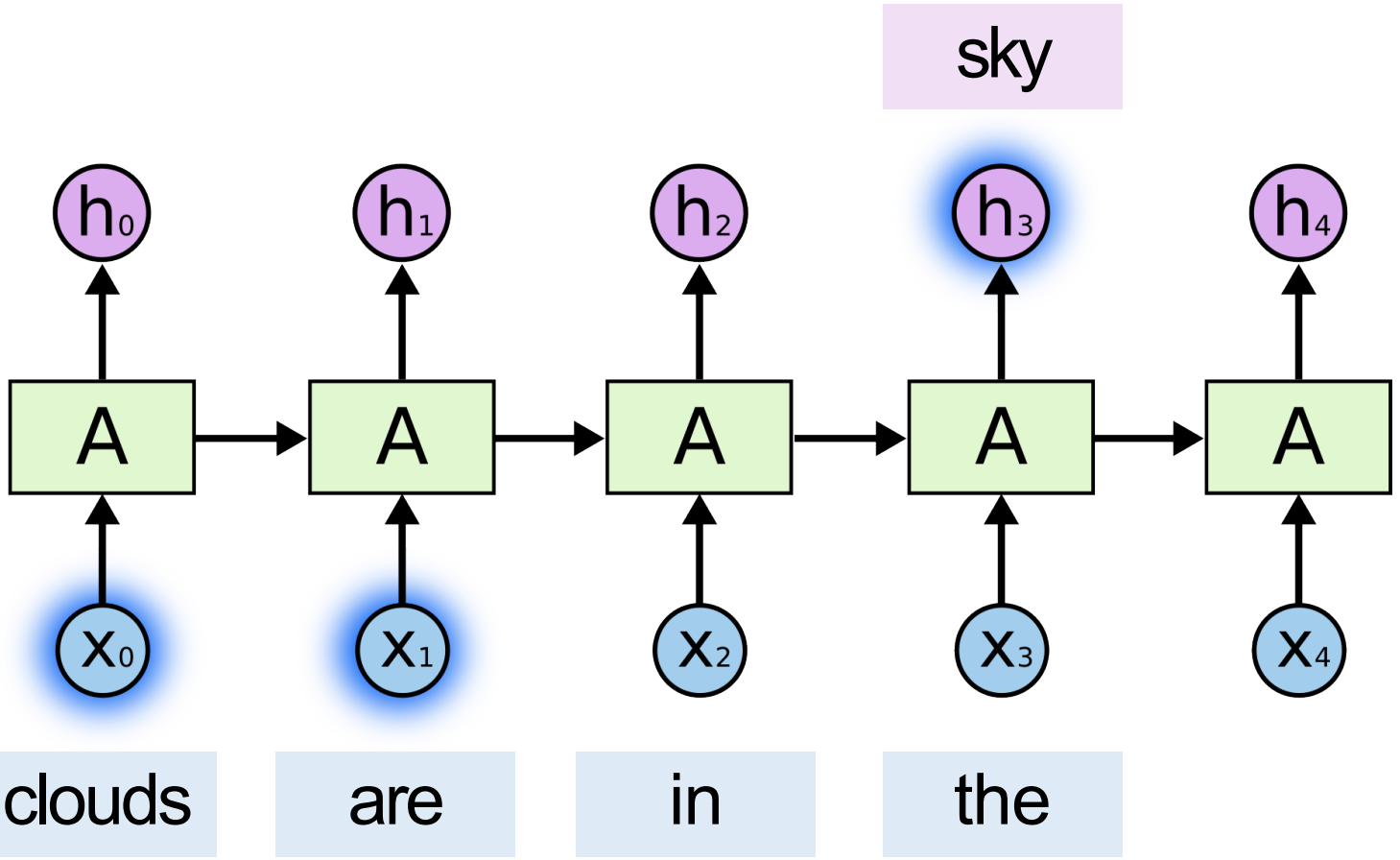
# Simple RNN for Classification



# Shortcomings of Simple RNN

# SimpleRNN is good at short-term dependence

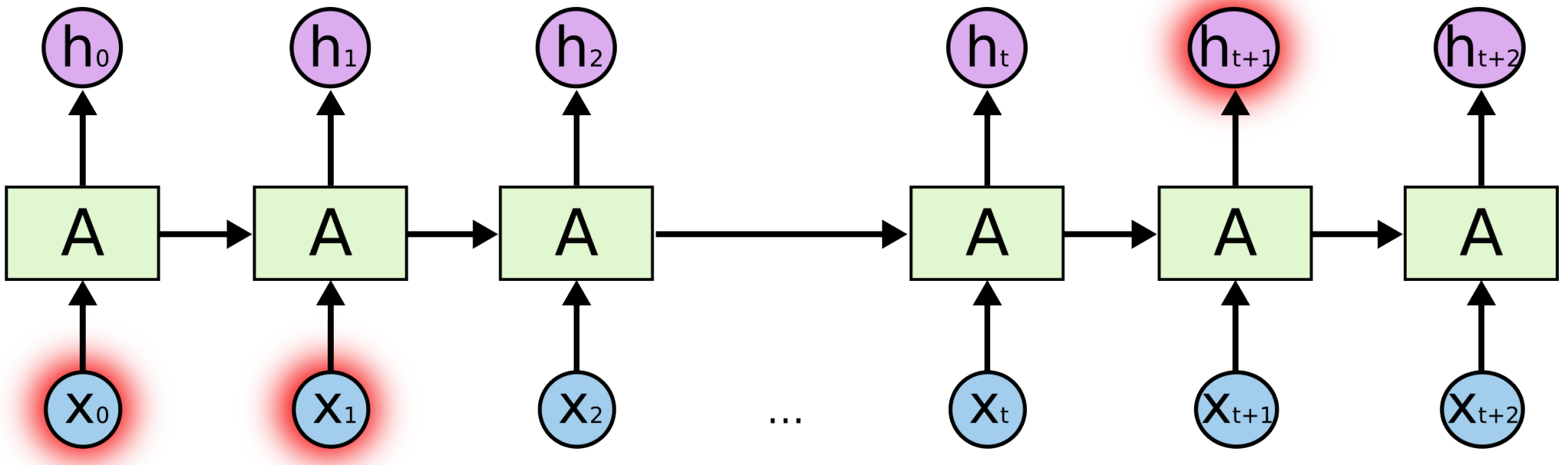
Predicted next words:




Input text:

Figures are from Christopher Olah's blog: Understanding LSTM Networks.

# SimpleRNN is good at short-term dependence



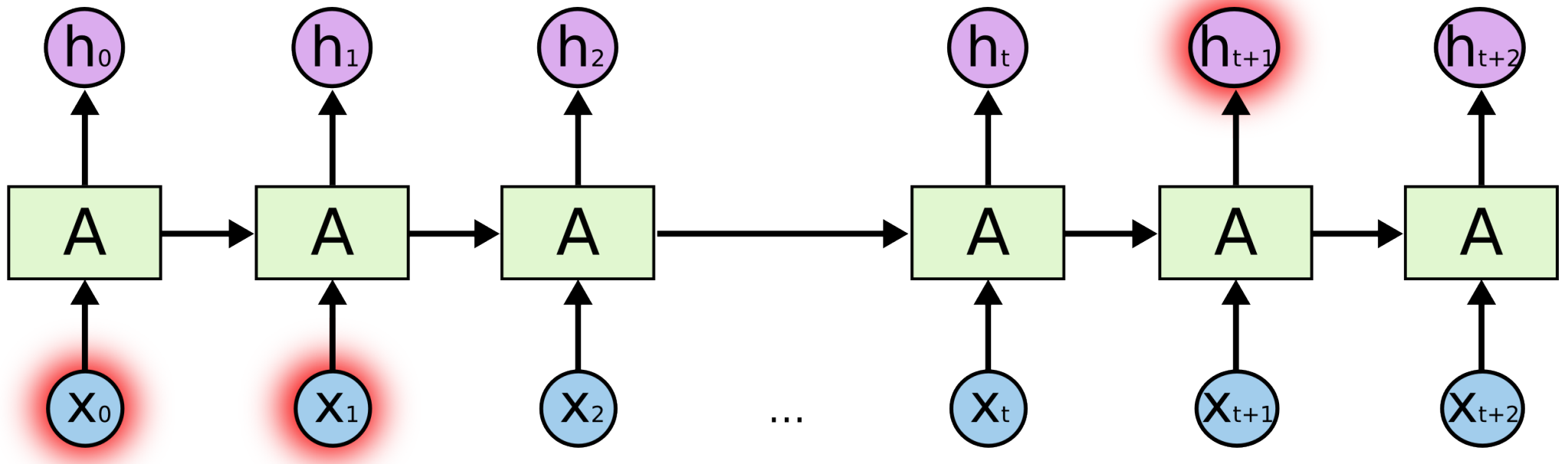

$\mathbf{h}_{100}$ is almost irrelevant to $\mathbf{x}_1$: $\frac{\partial \mathbf{h}_{100}}{\partial \mathbf{x}_1}$ is near zero.

Figures are from Christopher Olah's blog: Understanding LSTM Networks.

# SimpleRNN is good at short-term dependence

Predicted next words:

Chinese



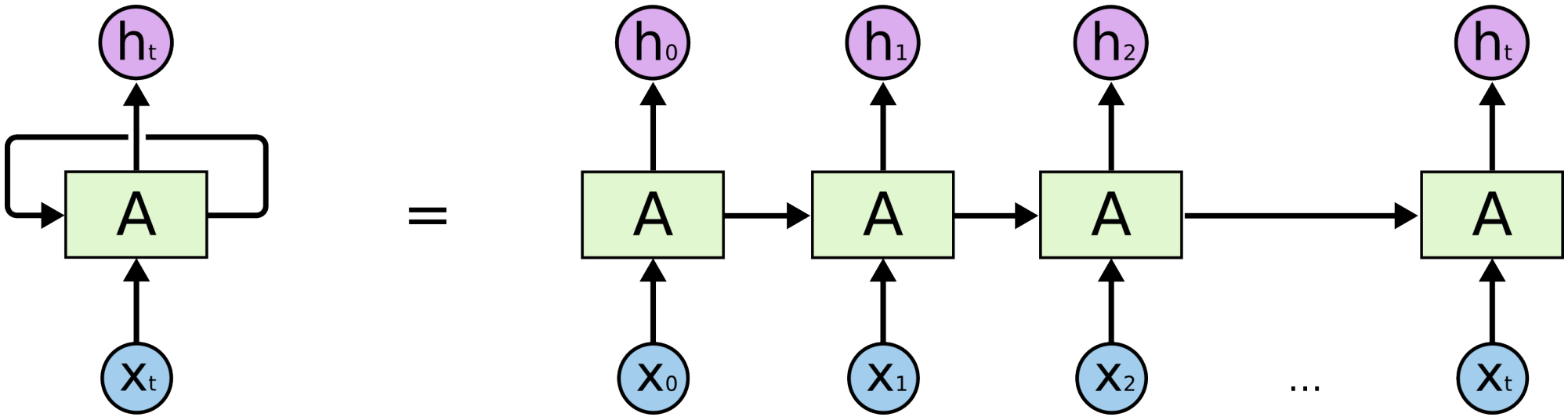

Input text:

in | China | speak | fluent

Figures are from Christopher Olah's blog: Understanding LSTM Networks.

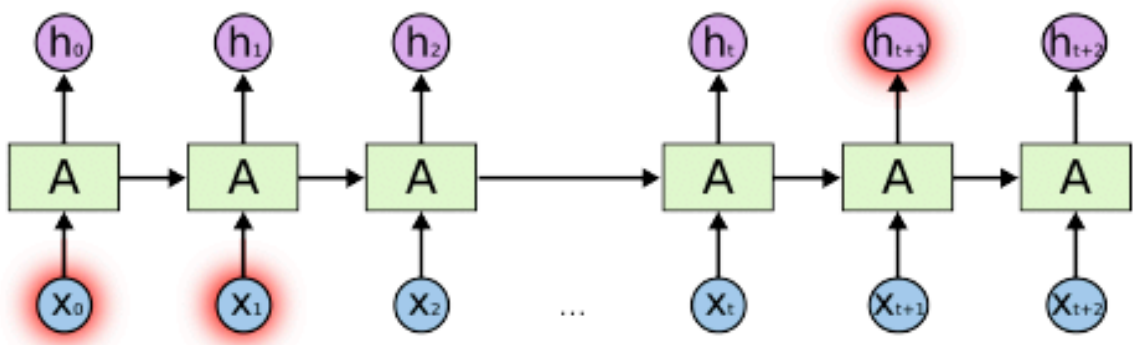
# Summary

- RNN for text, speech, and time series data.
- Hidden state $\mathbf{h}_t$ aggregates information in the inputs $\mathbf{x}_0, \cdots, \mathbf{x}_t$.



# Summary

- RNN for text, speech, and time series data.
- Hidden state $\mathbf{h}_t$ aggregates information in the inputs $\mathbf{x}_0, \cdots, \mathbf{x}_t$.
- RNNs can forget early inputs.
  - It forgets what it has seen early on.
  - If $t$ is large, $\mathbf{h}_t$ is almost irrelevant to $\mathbf{x}_0$.

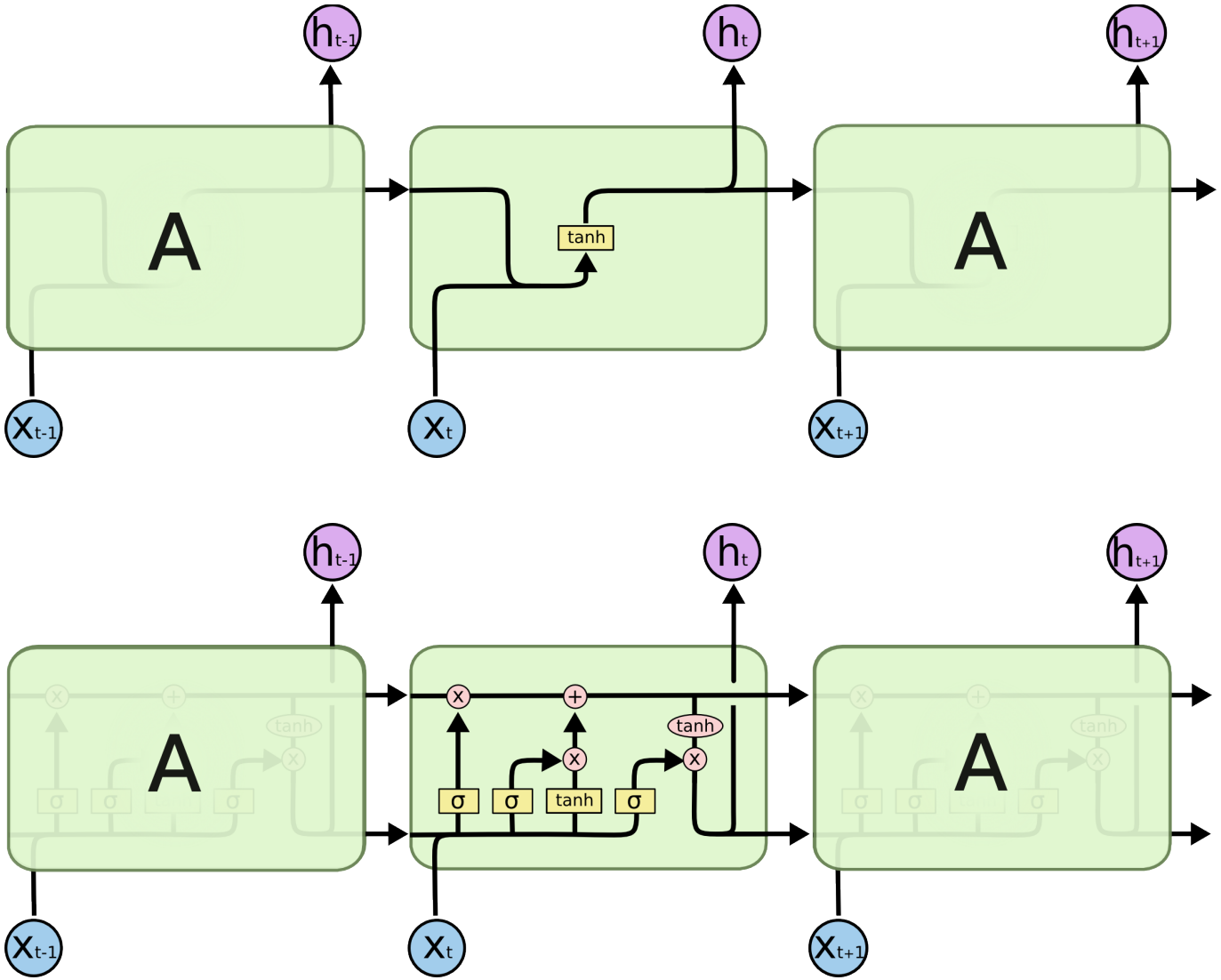


# LSTM Model

## Reference


- Hochreiter and Schmidhuber. Long short-term memory. *Neural computation,* 1997.

# LSTM Networks



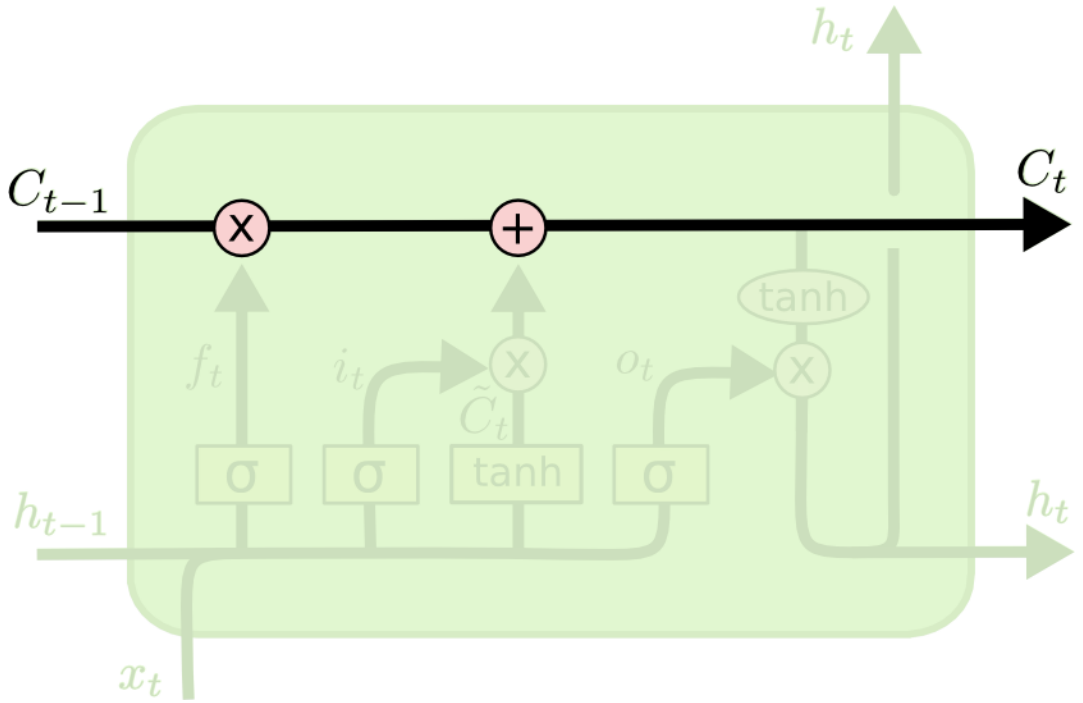

Simple RNN

LSTM

Figures are from Christopher Olah's blog: Understanding LSTM Networks.

# LSTM: Conveyor Belt

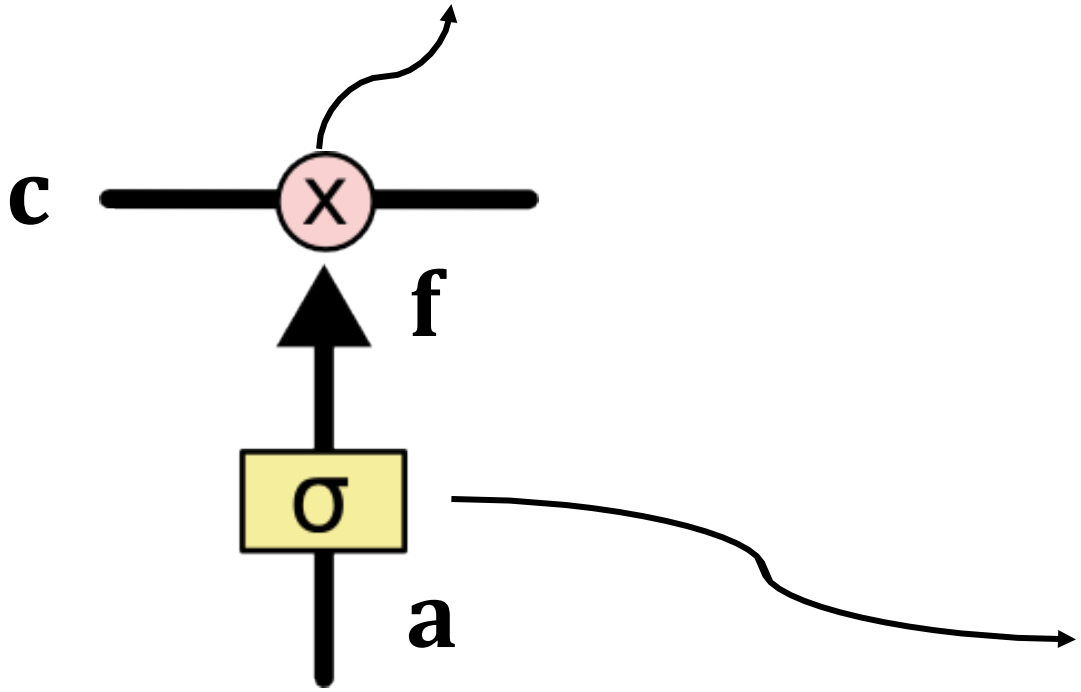
- Conveyor belt: the past information directly flows to the future.

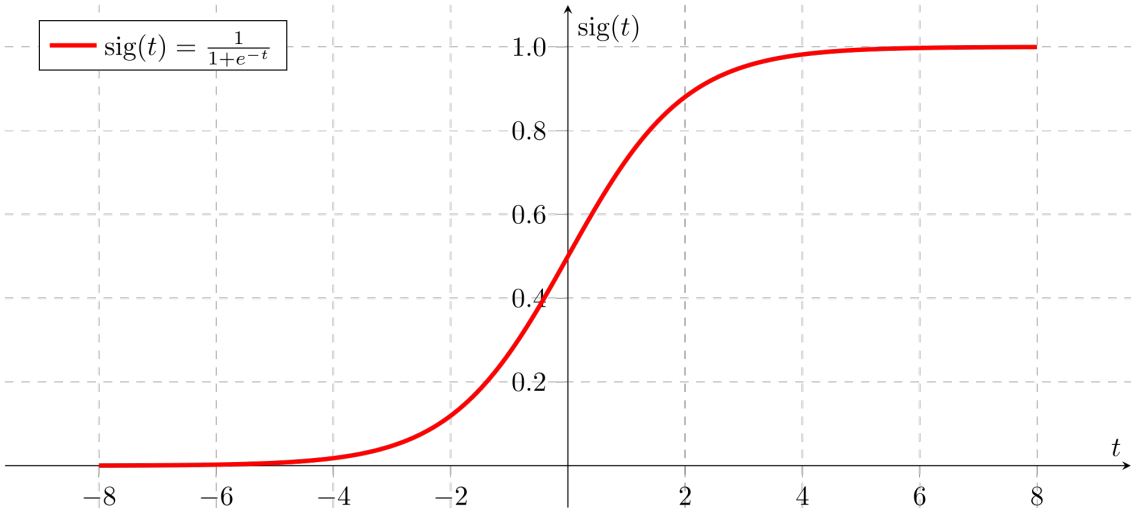



The Figure is from Christopher Olah's blog: Understanding LSTM Networks.

# LSTM: Forget Gate

Elementwise multiplication of 2 vectors.

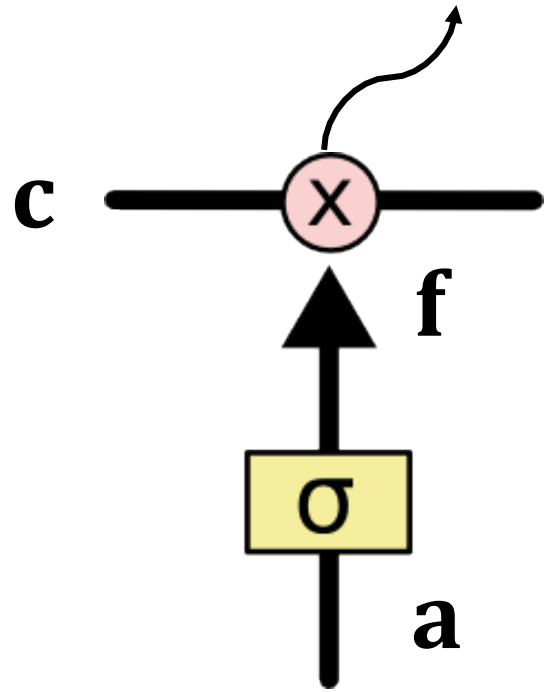



Sigmoid function: between 0 and 1.

# LSTM: Forget Gate

Elementwise multiplication of 2 vectors.

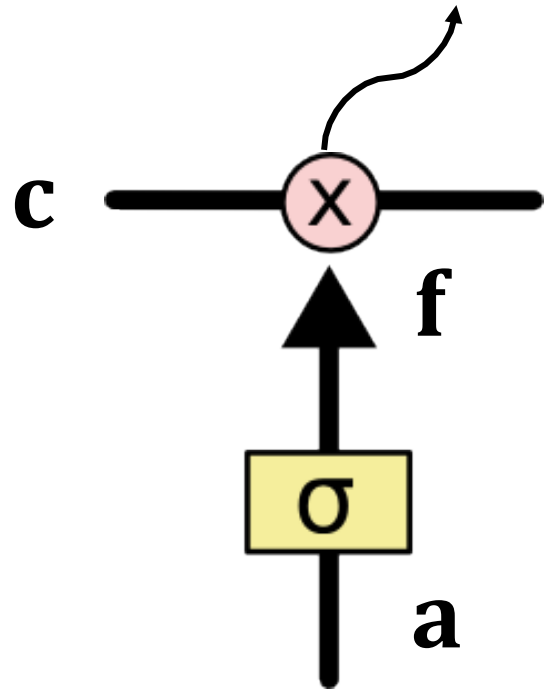



$$\sigma\left(\begin{bmatrix}1\\3\\0\\-2\end{bmatrix}\right) = \begin{bmatrix}0.73\\0.95\\0.5\\0.12\end{bmatrix}$$

$\mathbf{a}$ $\mathbf{f}$

# LSTM: Forget Gate

Elementwise multiplication of 2 vectors.



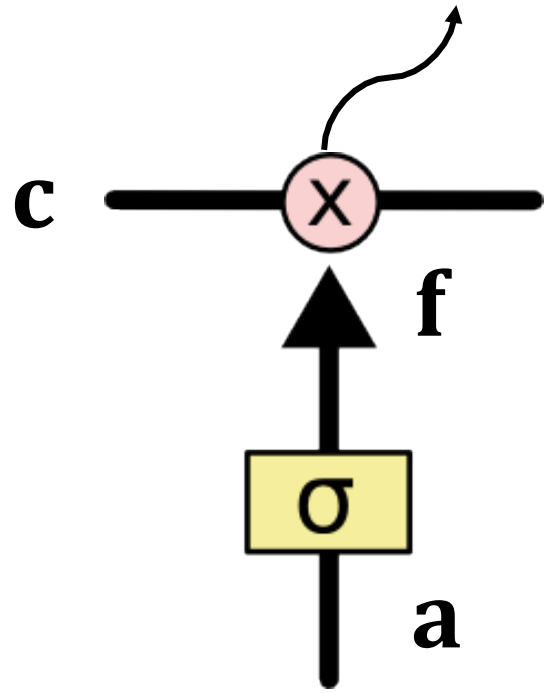

$$\begin{bmatrix} 0.9 \\ 0.2 \\ -0.5 \\ -0.1 \end{bmatrix} \circ \begin{bmatrix} 0.5 \\ 0 \\ 1 \\ 0.8 \end{bmatrix} = \begin{bmatrix} 0.45 \\ 0 \\ -0.5 \\ -0.08 \end{bmatrix}$$

**c** **f** **output**

# LSTM: Forget Gate

- Forget gate:
  - A value of zero means "let nothing through".
  - A value of one means "let everything through!"

Elementwise multiplication of 2 vectors.

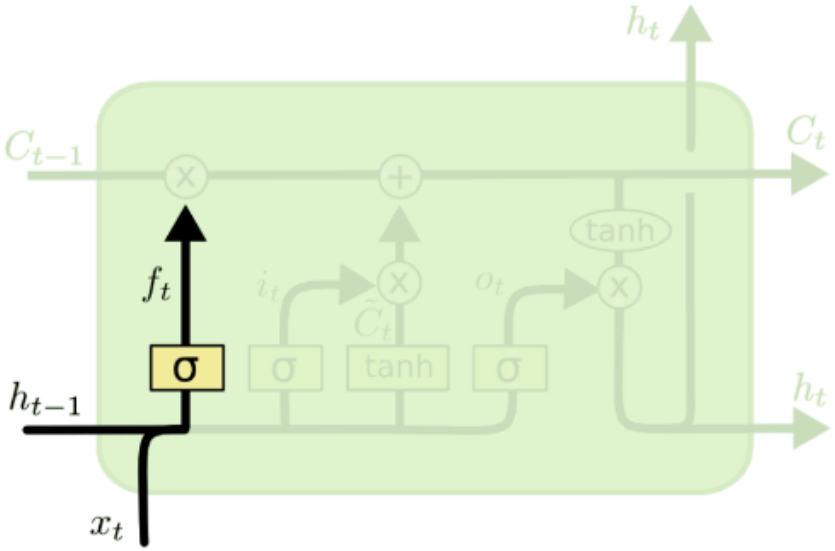


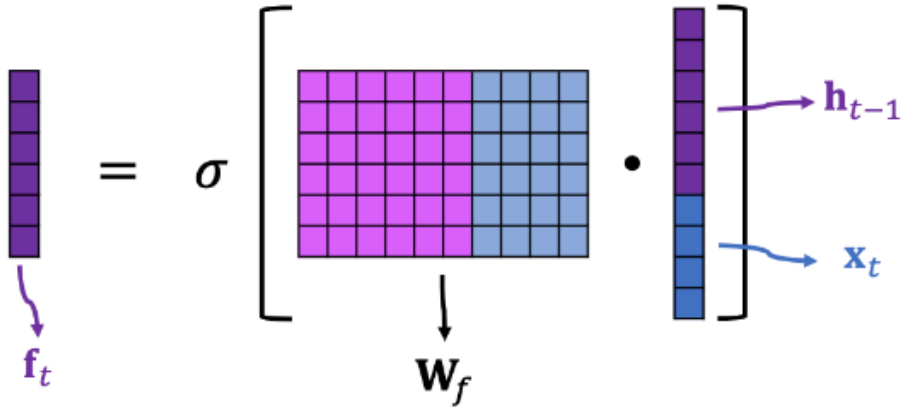

$$\begin{bmatrix} 0.9 \\ 0.2 \\ -0.5 \\ -0.1 \end{bmatrix} \circ \begin{bmatrix} 0.5 \\ 0 \\ 1 \\ 0.8 \end{bmatrix} = \begin{bmatrix} 0.45 \\ 0 \\ -0.5 \\ -0.08 \end{bmatrix}$$

c f output

# LSTM: Forget Gate

- Forget gate f
  - A value of zero means "let nothing through".
  - A value of one means "let everything through!"

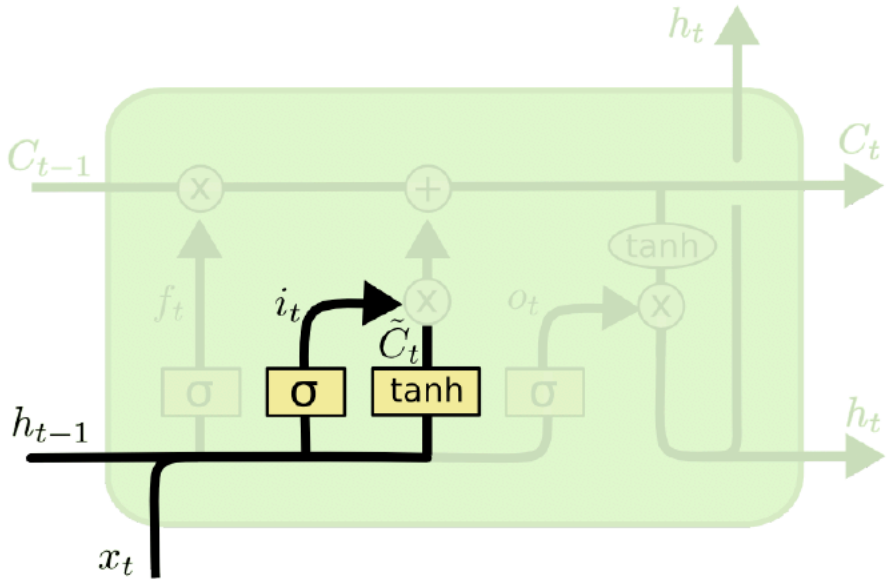


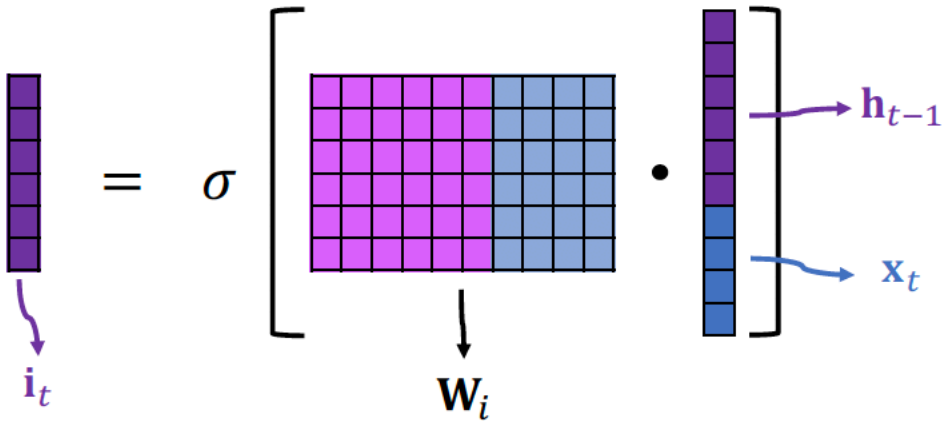




The left figure is from Christopher Olah's blog: Understanding LSTMNetworks.

# LSTM: Input Gate

- Input gate i: decides which values of the conveyor belt we'll update.

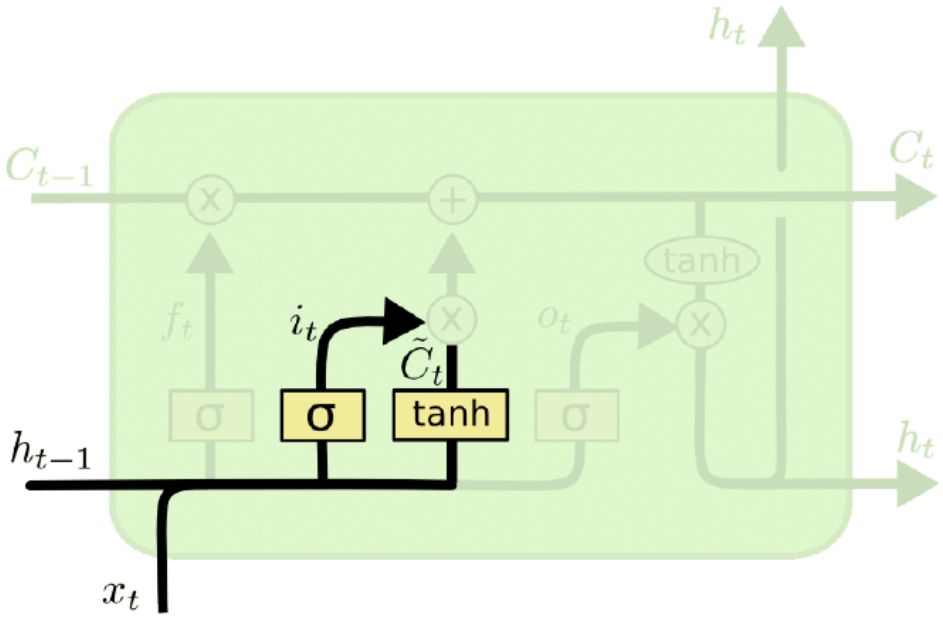


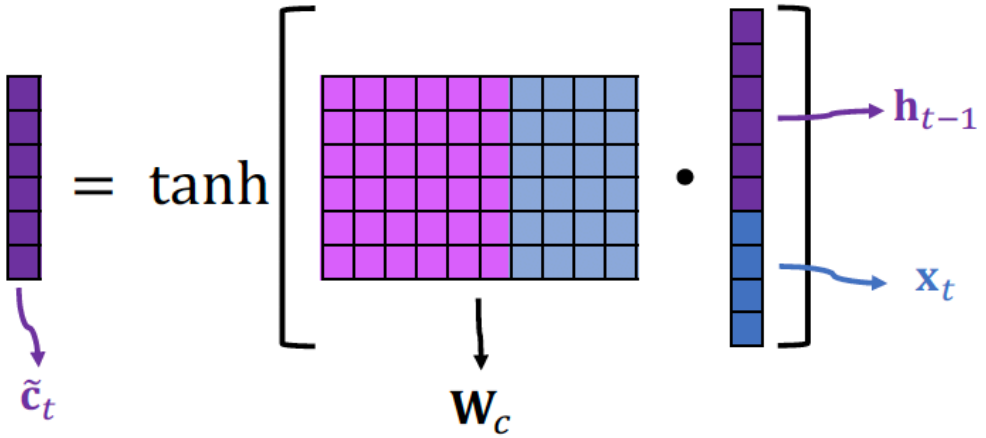




The left figure is from Christopher Olah's blog: Understanding LSTMNetworks.

# LSTM: New Value

- New value ($\tilde{\mathbf{c}}_t$): to be added to the conveyor belt.

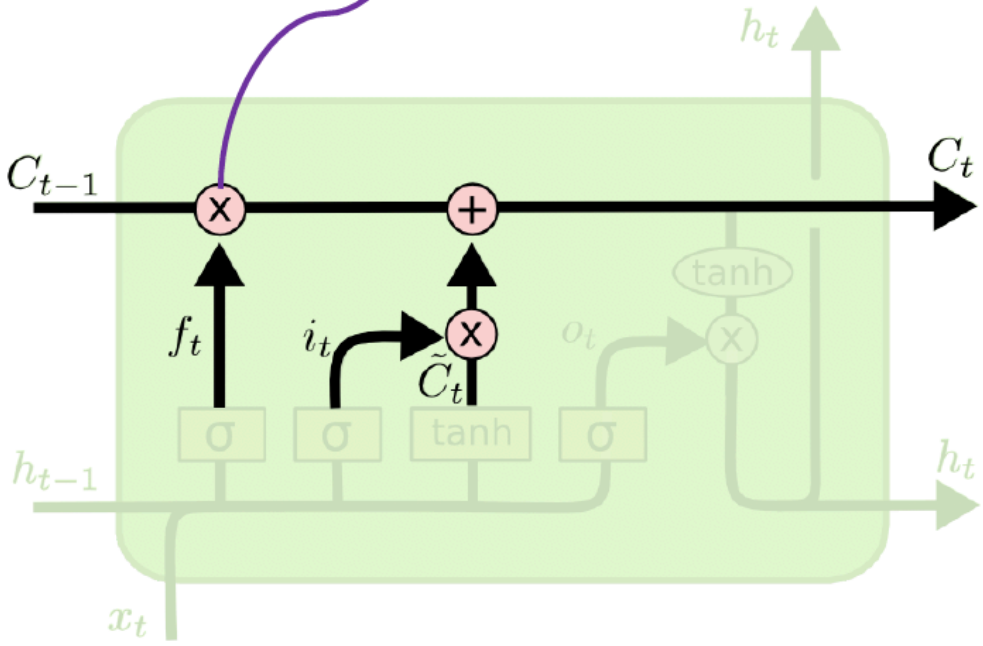


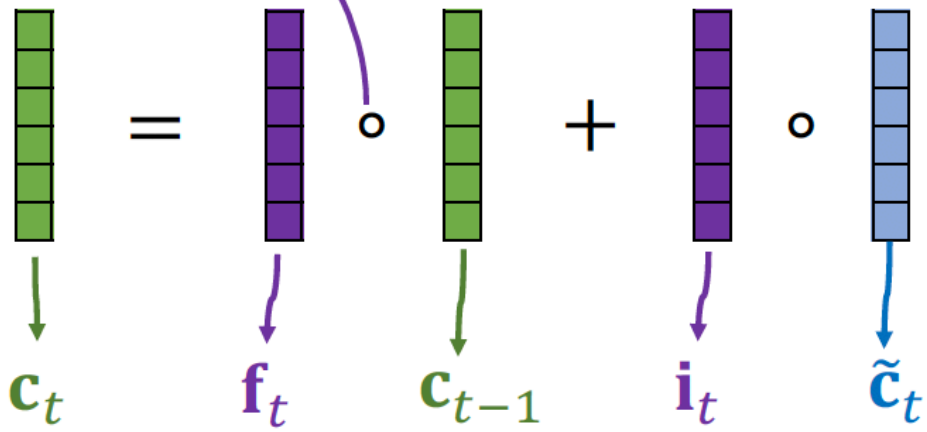




The left figure is from Christopher Olah's blog: Understanding LSTMNetworks.

# LSTM: Update the Conveyor Belt

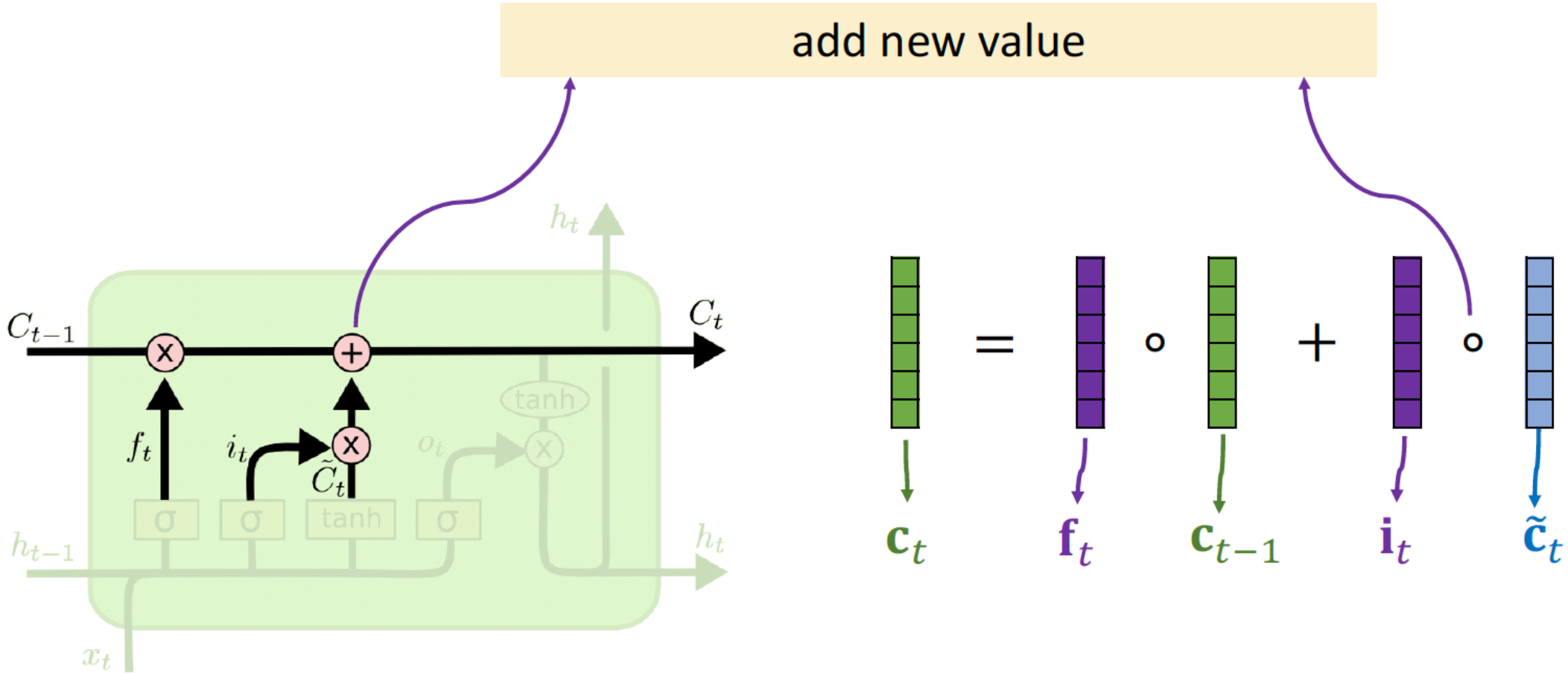
optionally let information through




The left figure is from Christopher Olah's blog: Understanding LSTMNetworks.

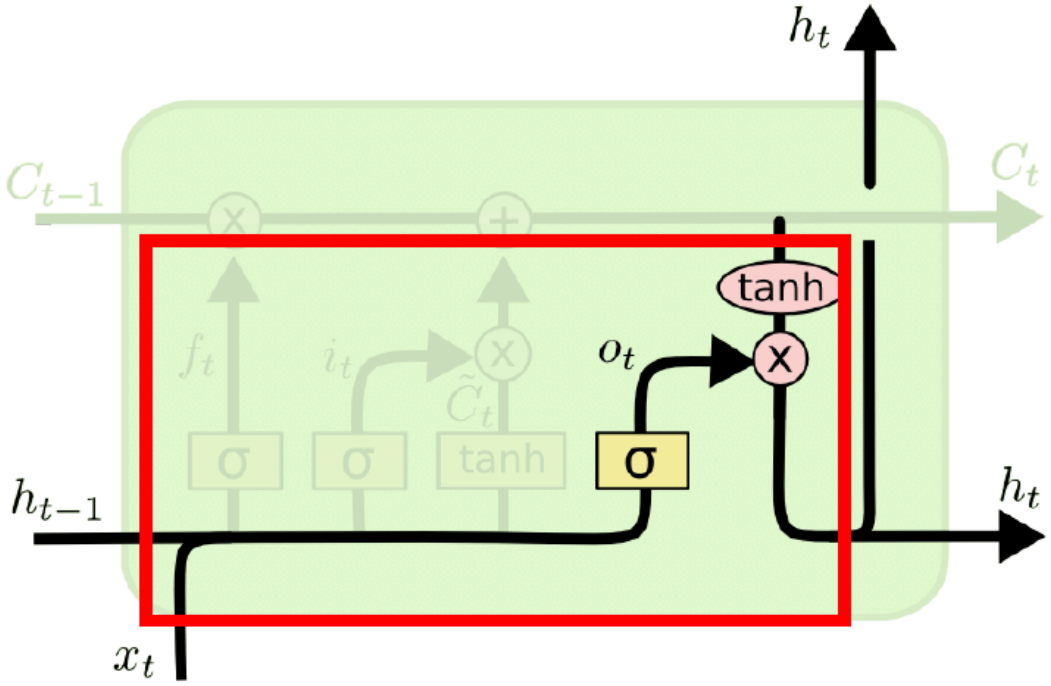
# LSTM: Update the Conveyor Belt




The left figure is from Christopher Olah's blog: Understanding LSTMNetworks.

# LSTM: Output Gate

- Output gate ($\mathbf{o}_t$): decide what flows from the conveyor belt $C_{t-1}$ to the state $h_t$.

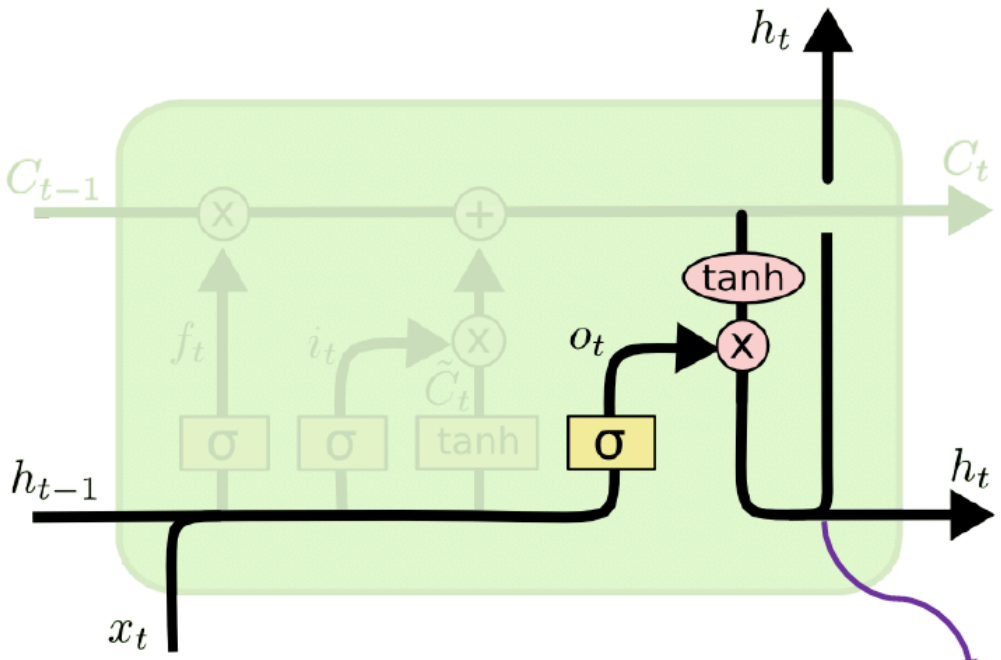



$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o \cdot \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix}\right)$$

The left figure is from Christopher Olah's blog: Understanding LSTMNetworks.

# LSTM: Update State

- State ($\mathbf{h}_t$): the output of LSTM.

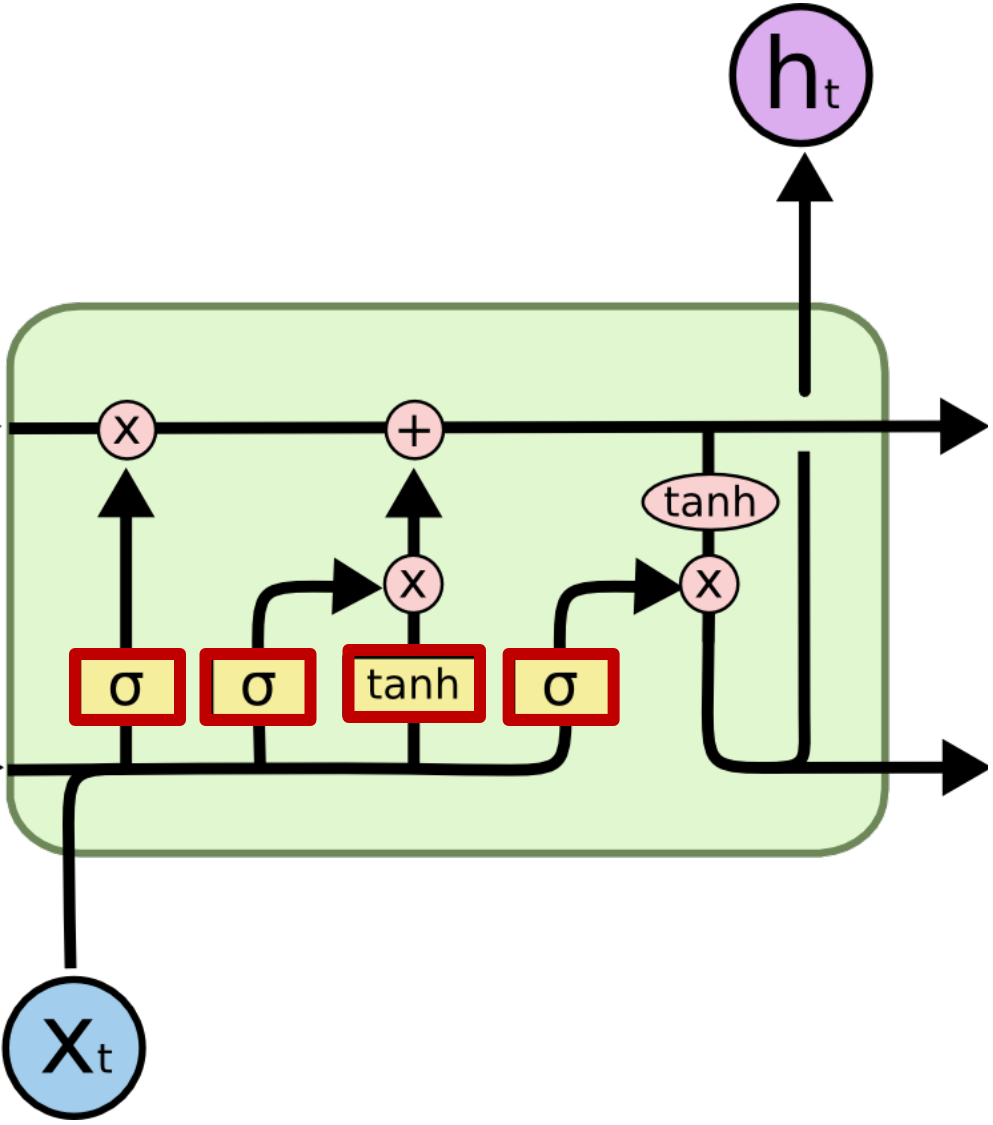



Two copies of $h_t$

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh \begin{bmatrix} \\ \\ \\ \\ \\ \\ \end{bmatrix}_{\mathbf{c}_t}$$

The left figure is from Christopher Olah's blog: Understanding LSTMNetworks.

# LSTM: Number of Parameters



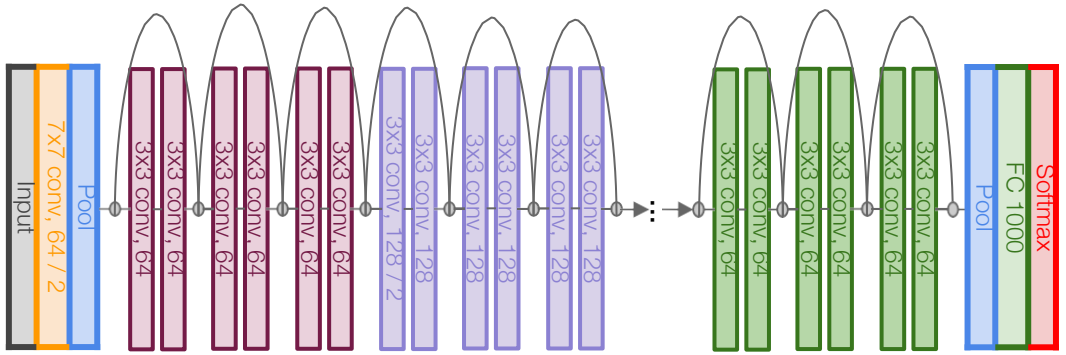

**#parameters: 4 times as many as SimpleRNN**

- 4 parameter matrices:
  - `f, i, c, o`

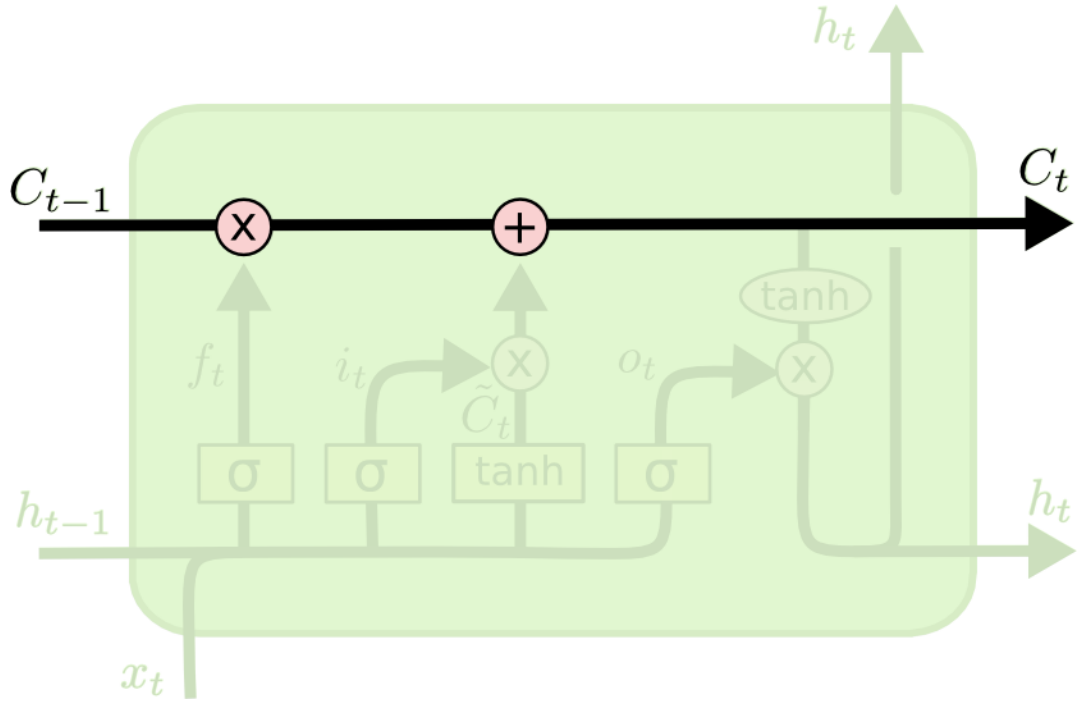
The left figure is from Christopher Olah's blog: Understanding LSTMNetworks.

# Summary

- LSTM uses a "conveyor belt" to get longer memory than SimpleRNN.




Similar to ResNet!




Uninterrupted gradient flow!

# Summary

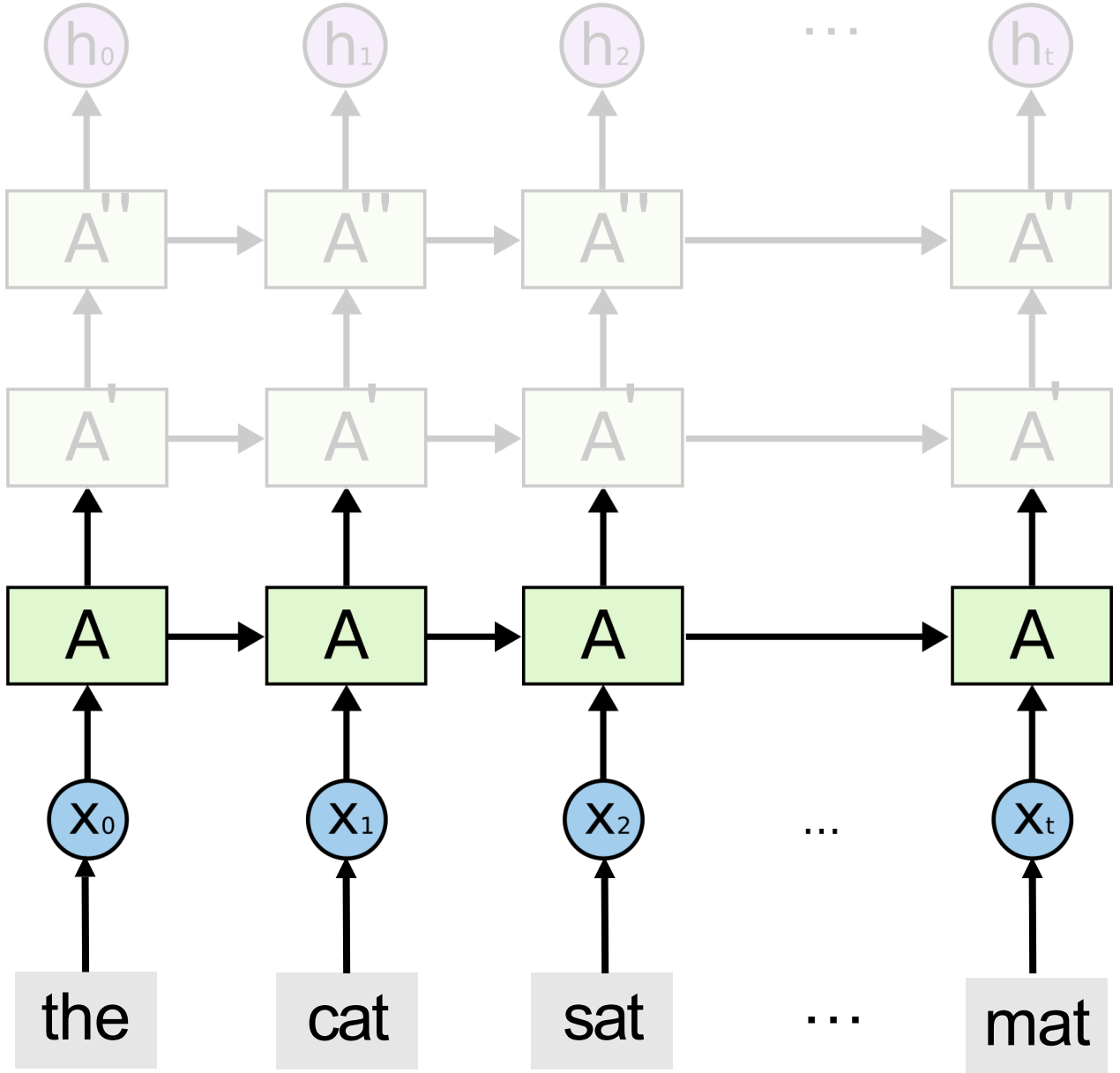
- LSTM uses a "conveyor belt" to get longer memory than SimpleRNN.
- Each of the following blocks has a parameter matrix:
  - Forget gate.
  - Input gate.
  - New values.
  - Output gate.

# Making RNNs More Effective

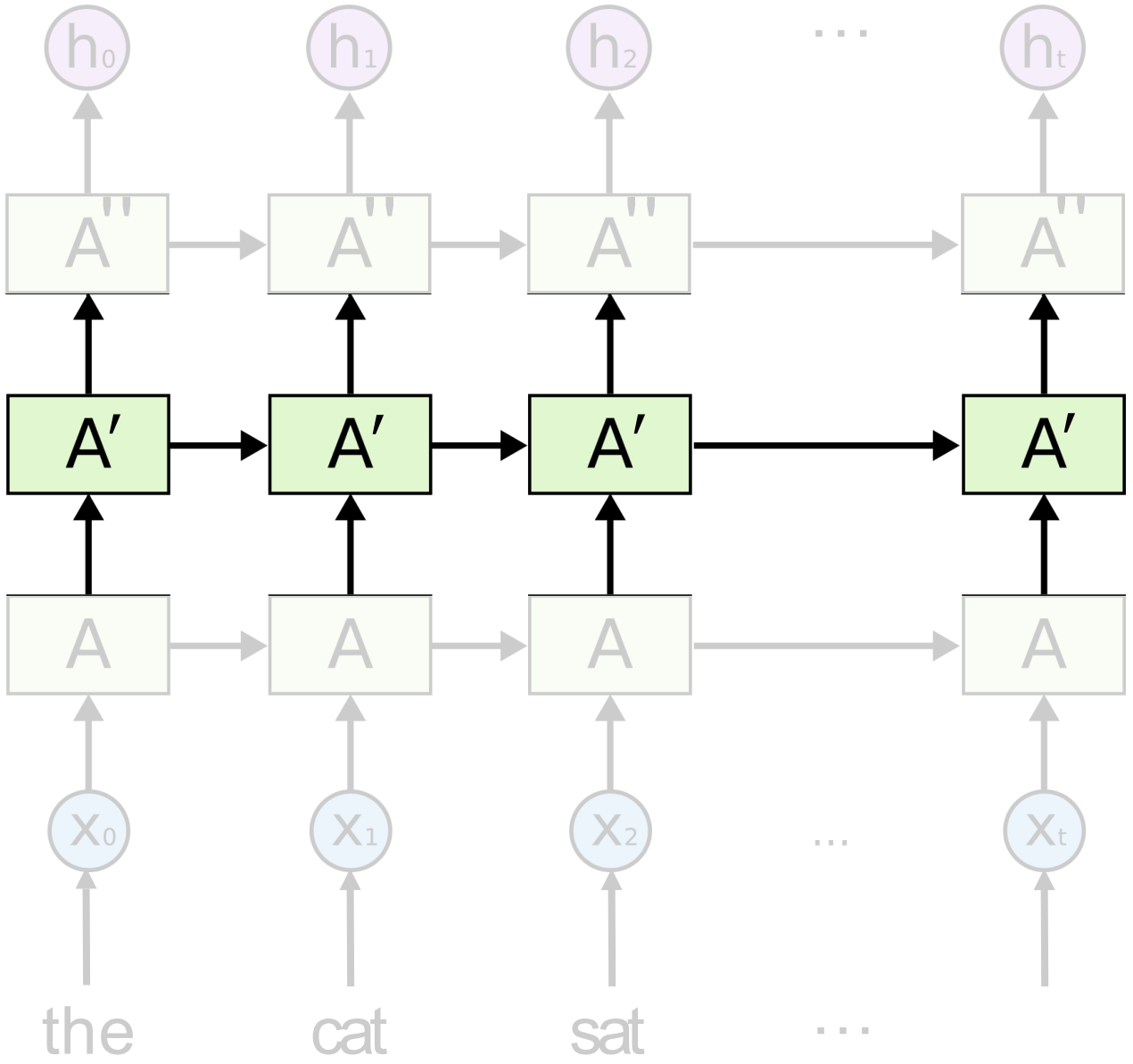
1. Stacked RNN
2. Bi-directional RNN
3. Pretraining Embedding Layers

# Stacked RNN

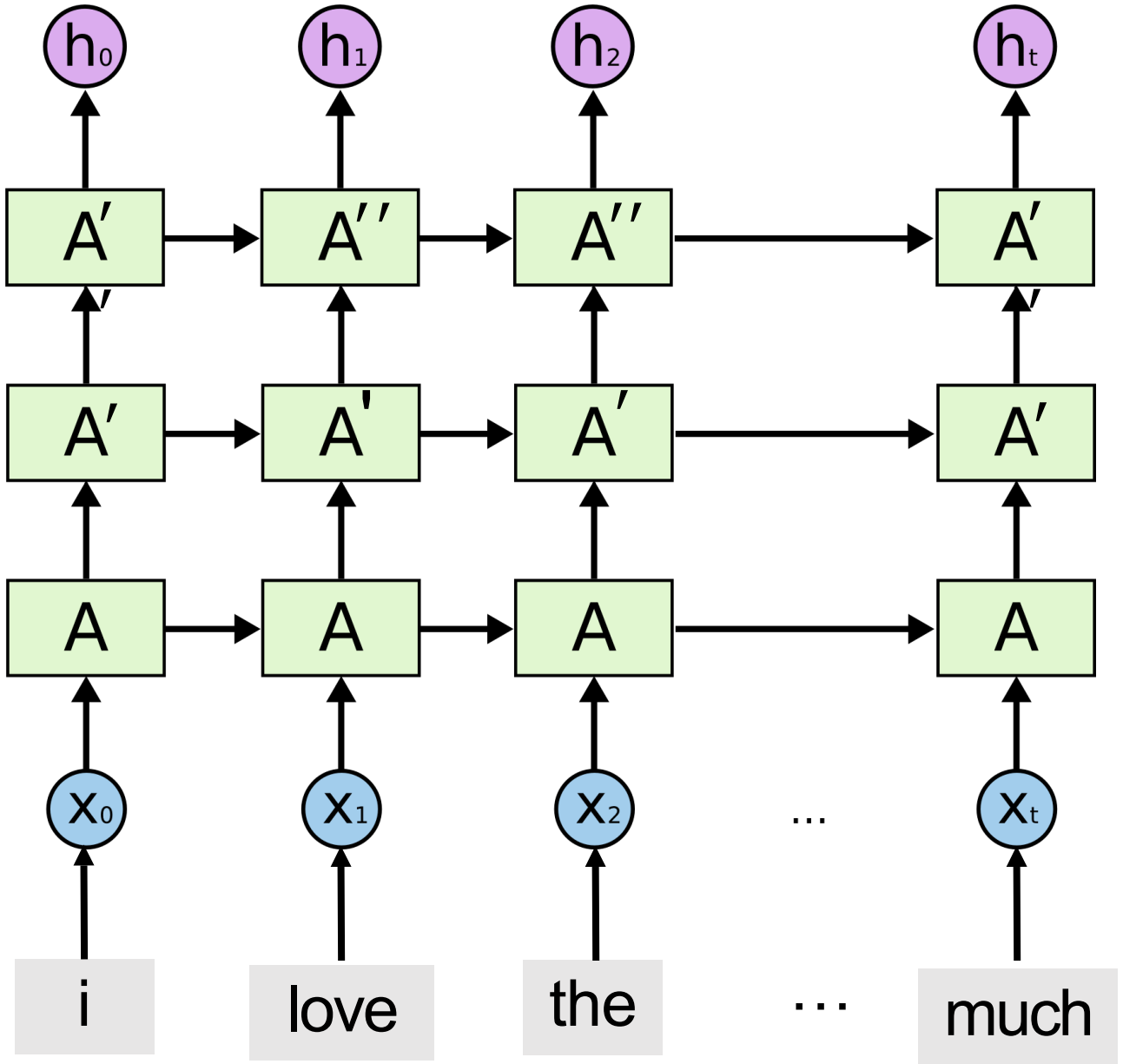
# Stacked RNN



# Stacked RNN

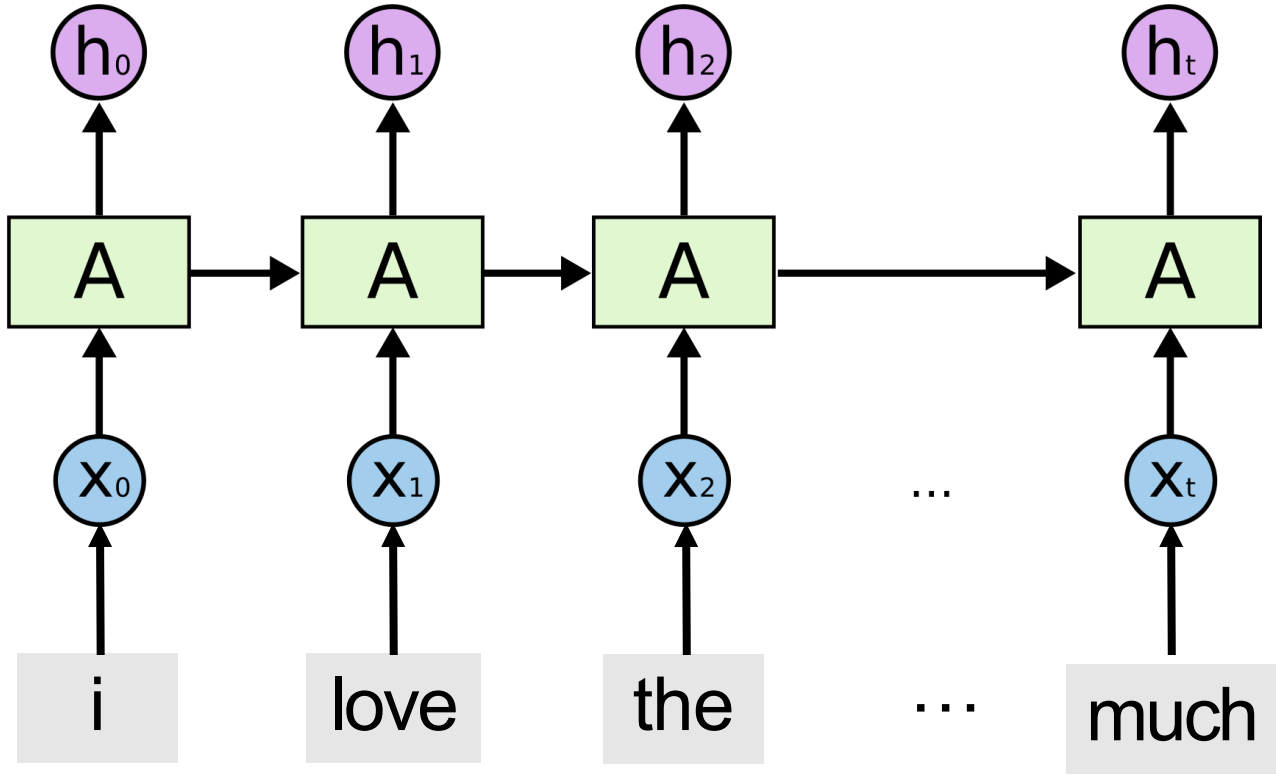


# Stacked RNN



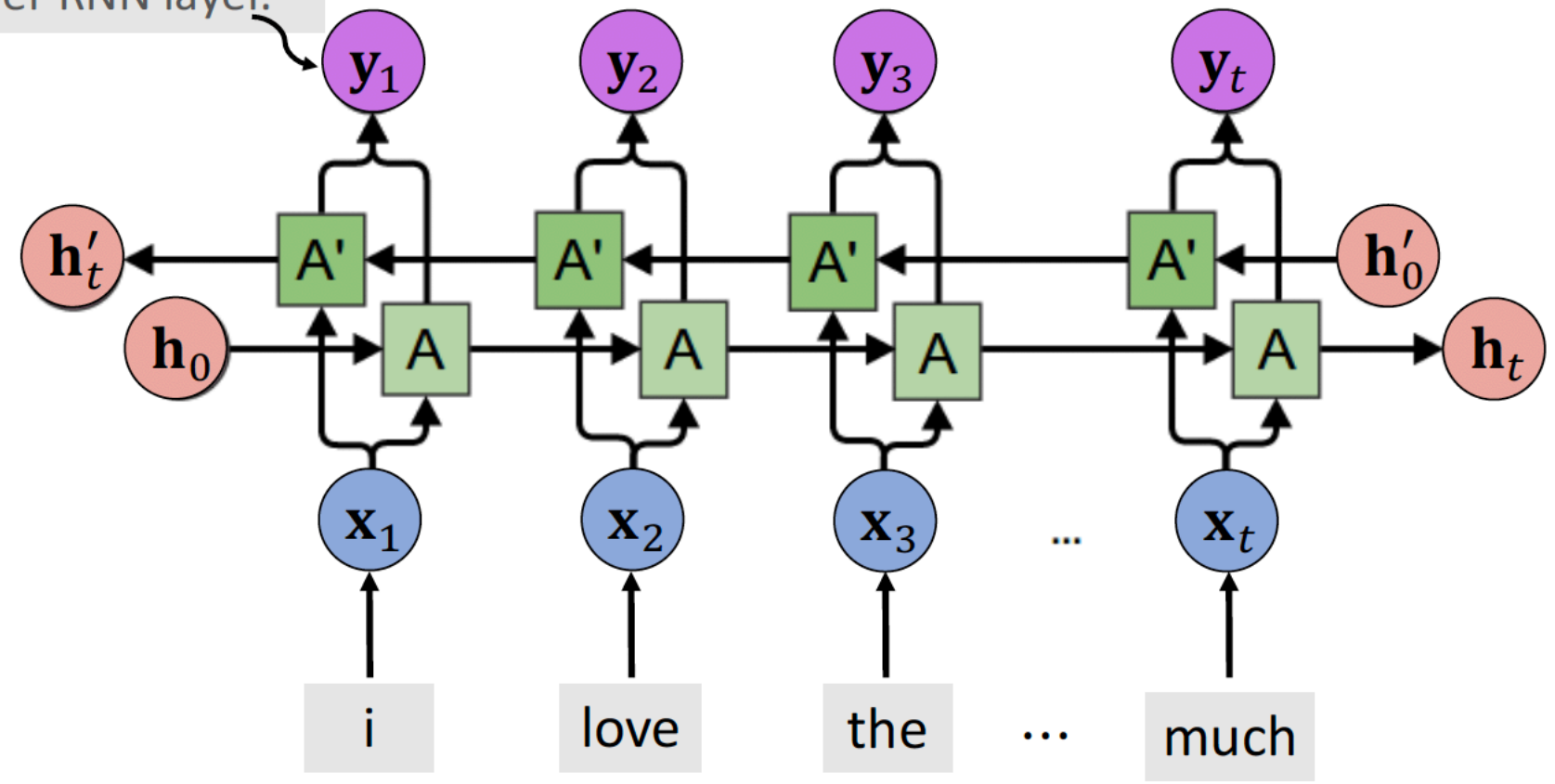
# Bidirectional RNN

# Standard RNN



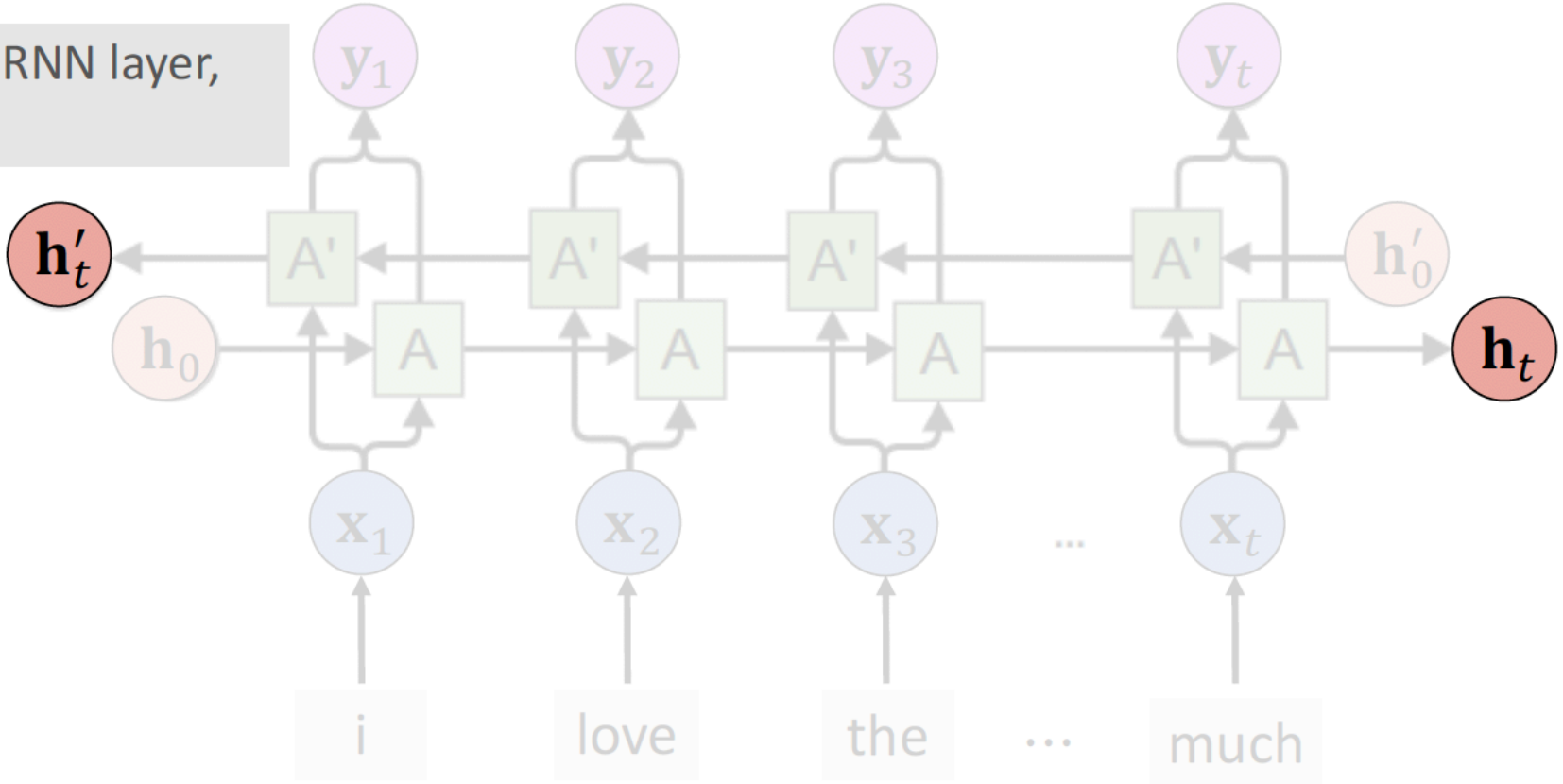
# Bidirectional RNN

- Stack of 2 states.
- Passed to the upper RNN layer.



# Bidirectional RNN

If there is no upper RNN layer, then return $[\mathbf{h}_t, \mathbf{h}'_t]$



# Pretrain

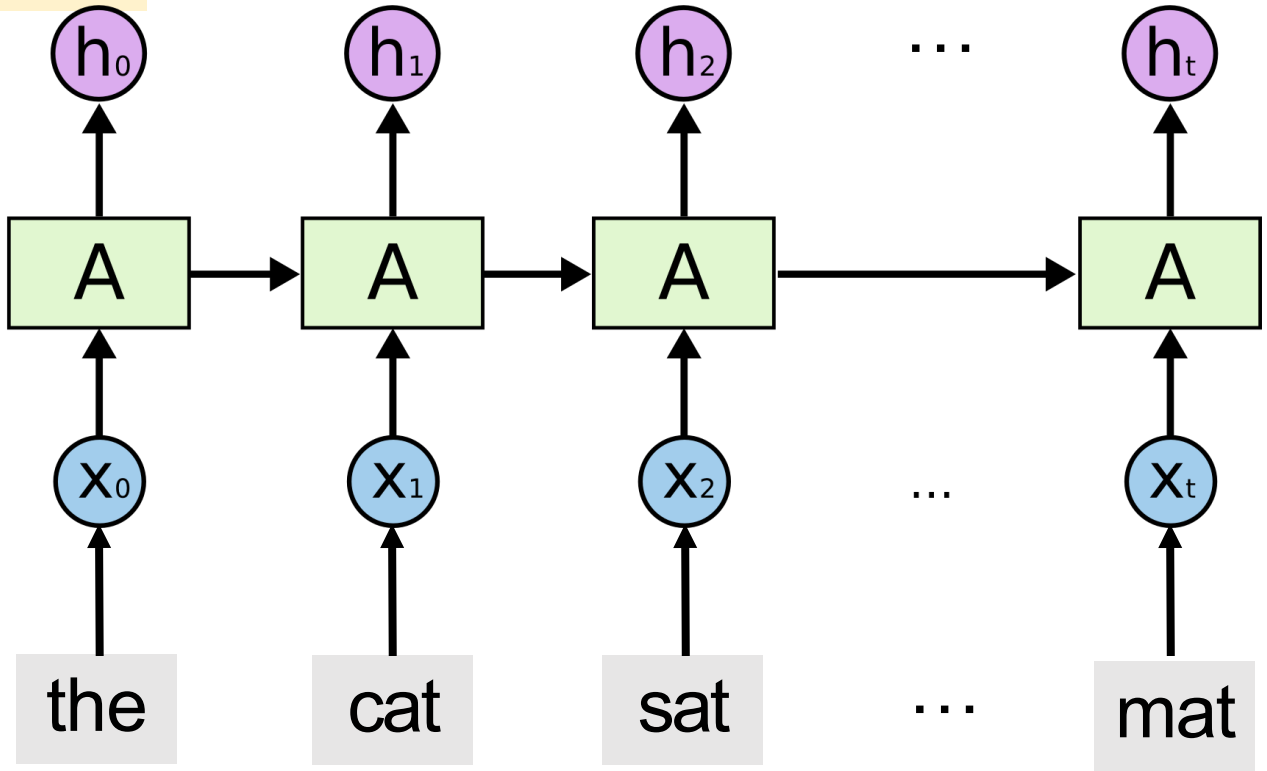
# Why pretraining?

**Observation:** The embedding layer contributes most of the parameters!

# Pretrain the Embedding Layer

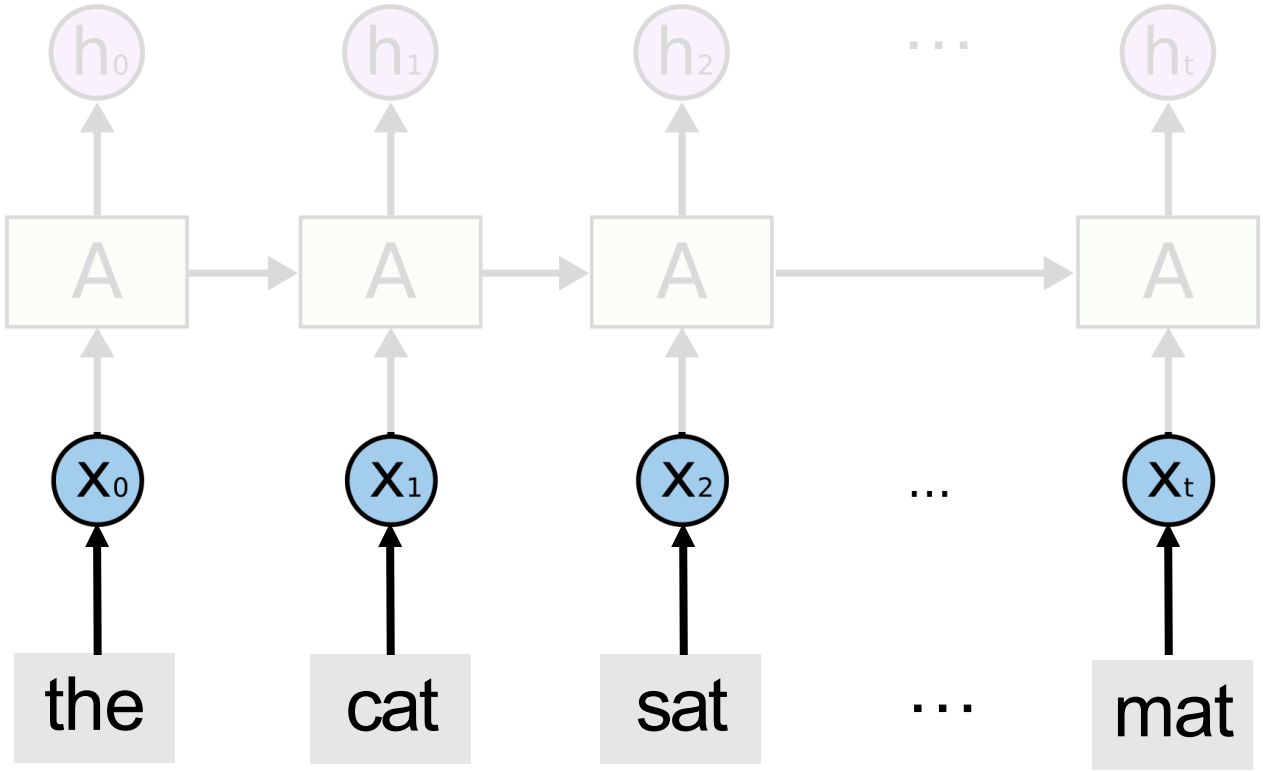
**Step 1:** Train a model on large dataset.

- Perhaps different problem.
- Perhaps different model.



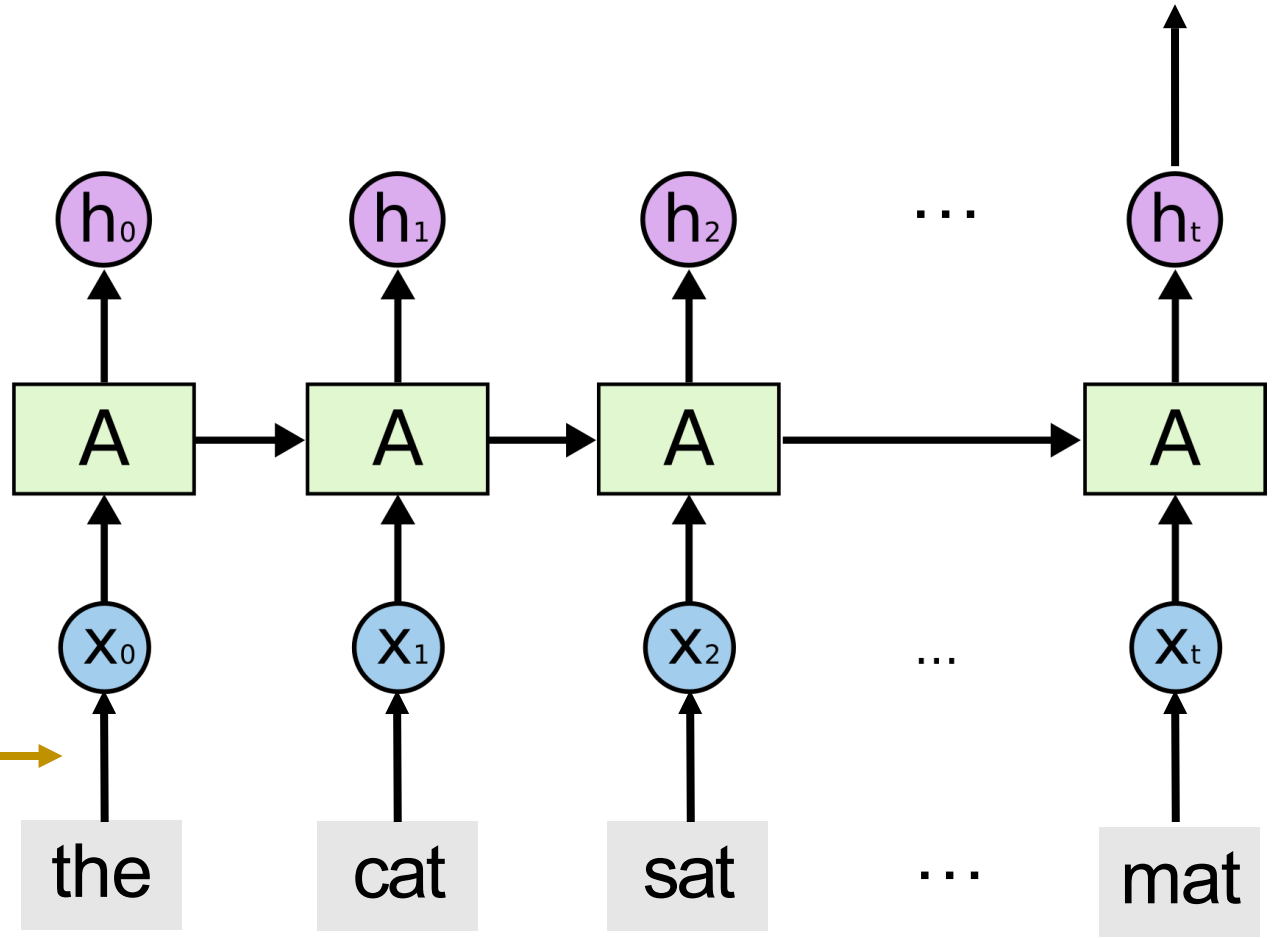
# Pretrain the Embedding Layer

**Step 2:** Keep only the embedding layer.



# Pretrain the Embedding Layer

**Step 3:** Train new LSTM and output layers.




Set the embedding layer non-trainable.

# Summary

- SimpleRNN and LSTM are two kinds of RNNs; always use LSTM instead of SimpleRNN.
- Use Bi-RNN instead of RNN whenever possible.
- Stacked RNN may be better than a single RNN layer.
- Pretrain the embedding layer.

Any Question ?