



Machine Learning

Lecture 11: Recurrent Neural Networks

Fall 2023

Instructor: Xiaodong Gu



Deep Neural Networks

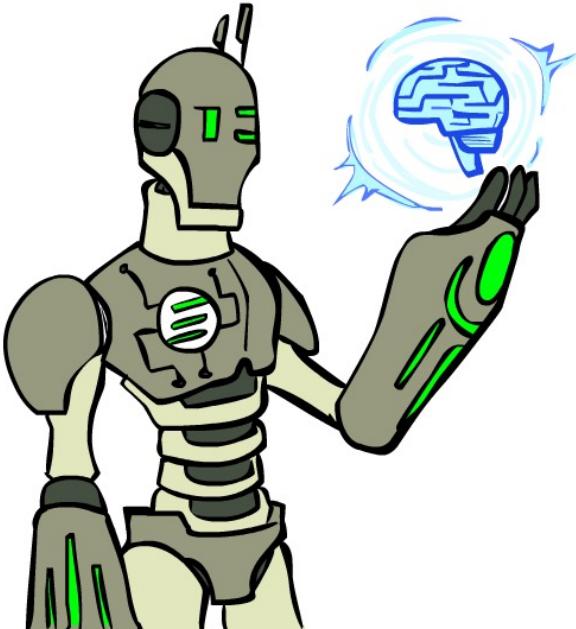


- Feedforward neural networks
- Convolutional neural networks
- Recurrent neural networks
- ...

Today



- Recurrent Neural Networks



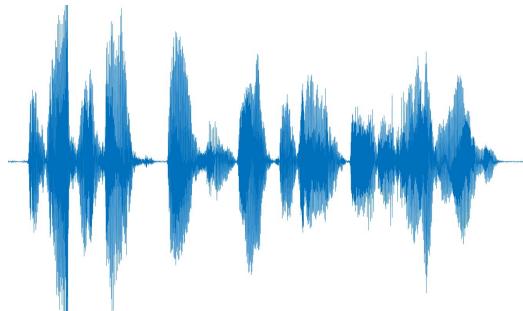
Sequence in the Wild





Sequences in the Wild

- Audio:



- Word:

SE-3332 Machine Learning

- Sentence:

machine learning is so fun !

- Program:

```
for (int i = 0; i < 10 ; i ++ ) {
```

- ...



How to model a sequence?

- Consider any sequence as a sentence in a language and apply the **language model**:

$$p(x) = p(w_1, w_2, \dots, w_N)$$

$$\begin{aligned} p(w_1, \dots, w_N) &= p(w_1) p(w_2|w_1) \dots p(w_N|w_1, \dots, w_{N-1}) \\ &\approx p(w_1)p(w_2|w_1)p(w_3|w_2)\dots p(w_N|w_{N-1}) \end{aligned}$$

How to estimate $p(w_t|w_1, \dots, w_{t-1})$?

- **The key problem:**

- given a sequence of n-1 tokens
- predict the next token



A Sequence Modeling Problem

- Given a sequence of $t-1$ words w_1, \dots, w_{t-1}
- Predict the next token $p(w_t | w_1, \dots, w_{t-1})$

“天 将 降 大 任 于 是”

given these words predict the next word





A Sequence Modeling Problem

- Idea #1: Counting?

“天 将 降 大 任 于 是”

given these words predict the next word

$$p(\text{是} | \text{天将降大任于}) = \frac{\text{count}(\text{天将降大任于是})}{\text{count}(\text{天将降大任于})}$$

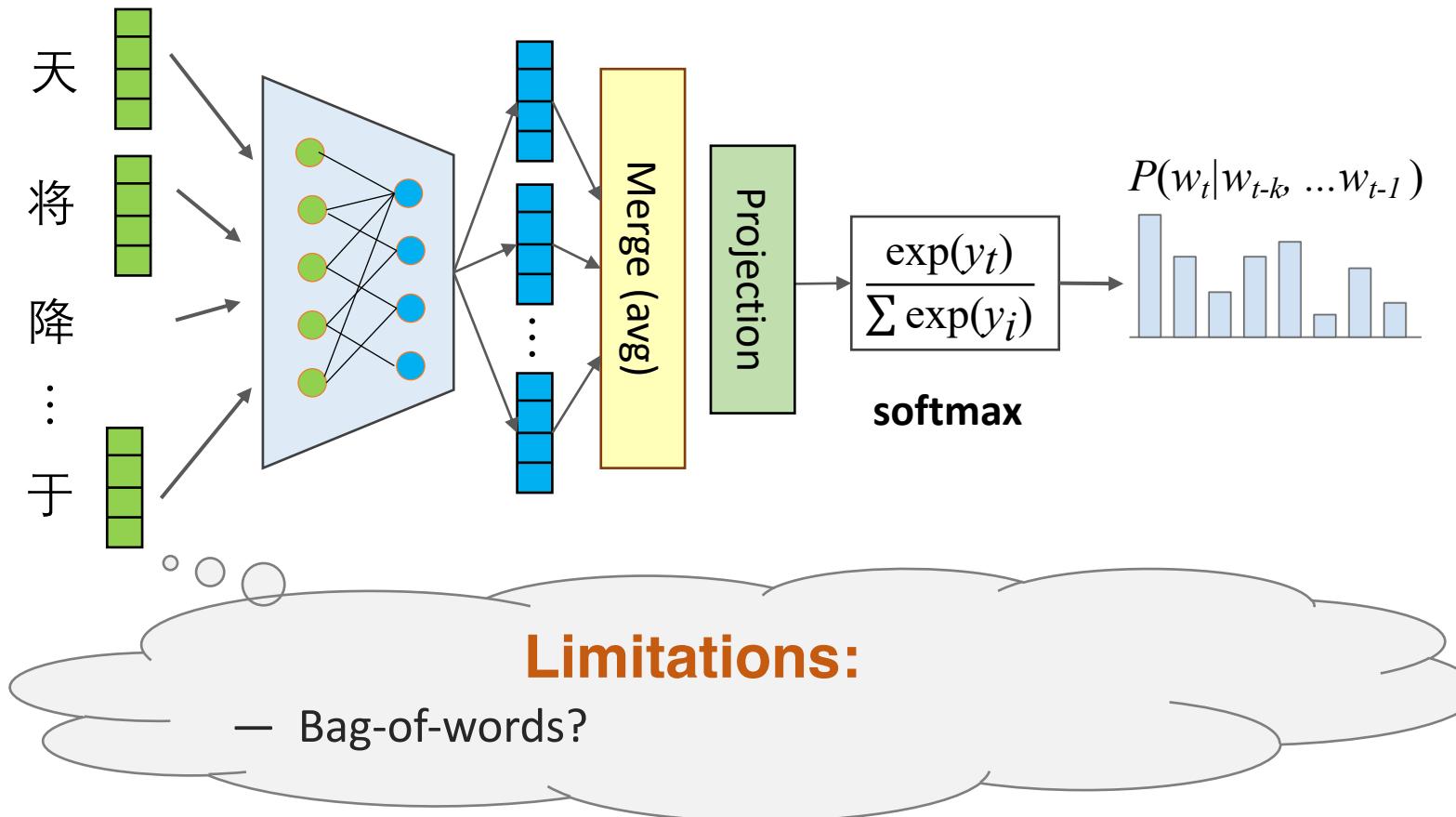
Problems:

- Discrete symbols,
- No word semantics,
- Insensitive to similar words and sentences.

How about “天欲把
重担给__”?

Neural Networks as Language Models

- Idea #2: Using Neural Networks?

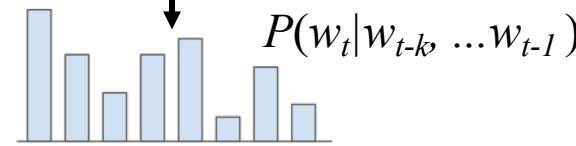
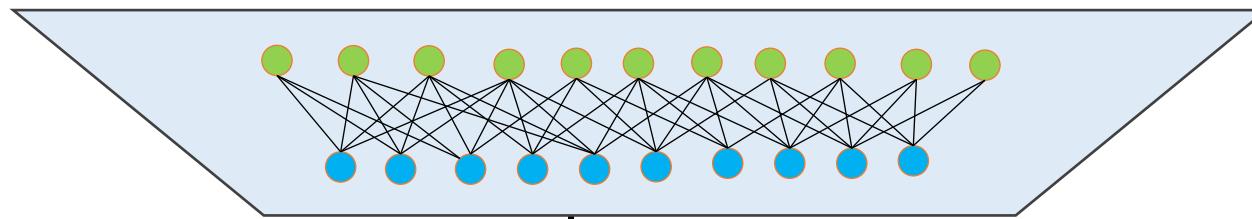


Neural Networks as Language Models



- Idea #3: Use a Really Big Fixed Window (concat word vectors)

天 将 降 大 任
[1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 ...]



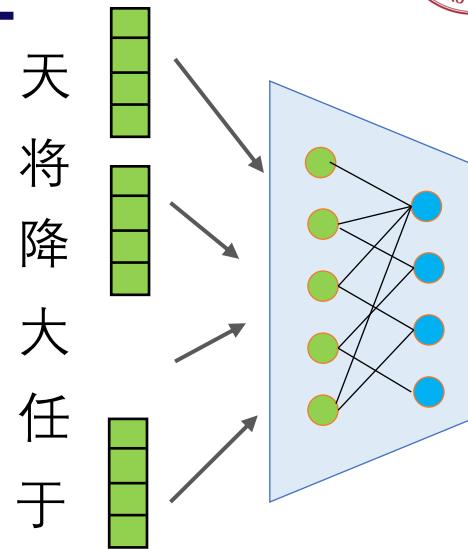
Limitations:

- Variant Input Length?
- Cannot share parameters

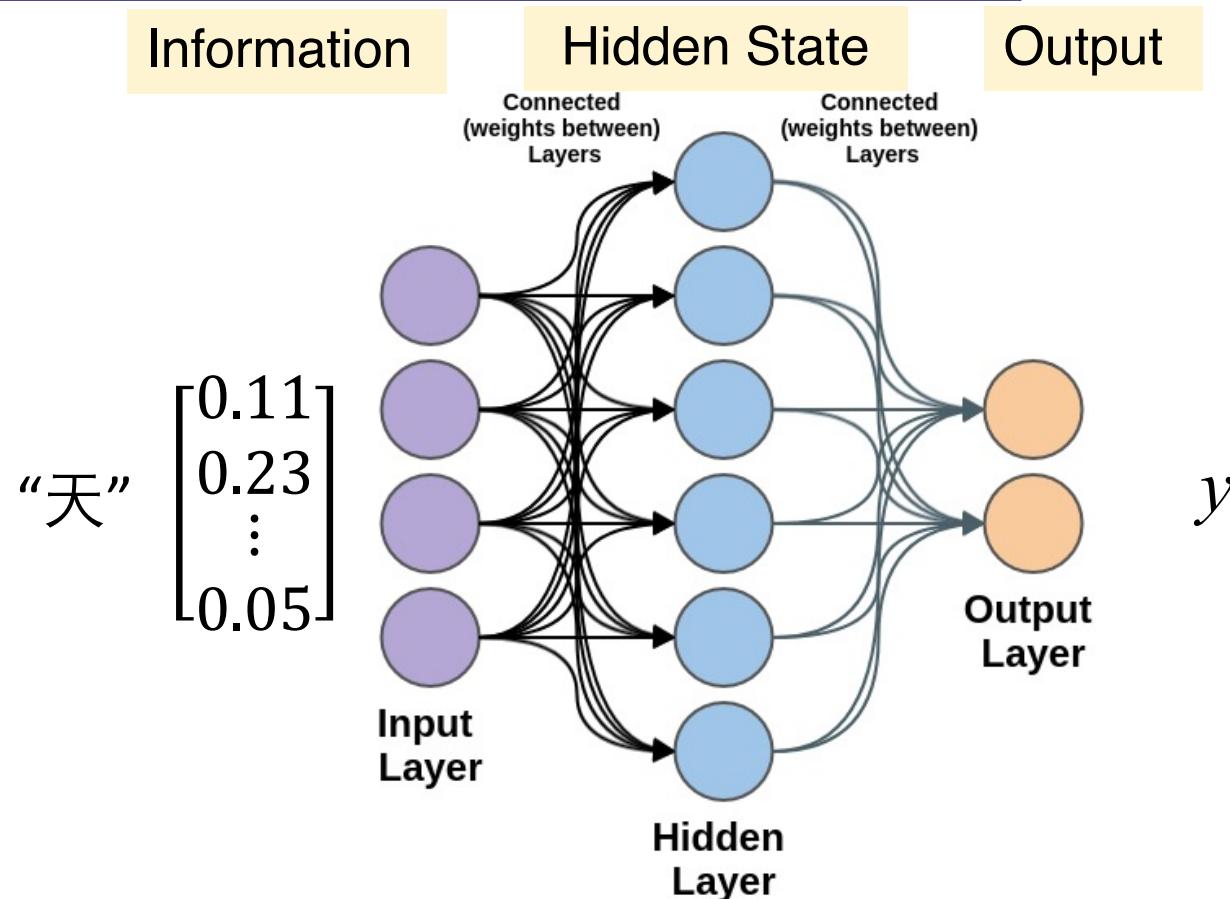
Sequence Modeling: Design Criteria



- To model sequences, we need to:
 1. Handle **variable-length** sequences
 2. Track **long-term** dependencies
 3. Maintain information about **order**
 4. Share parameters across the sequence
- **Recurrent Neural Networks (RNNs)** as an approach for sequence modeling.



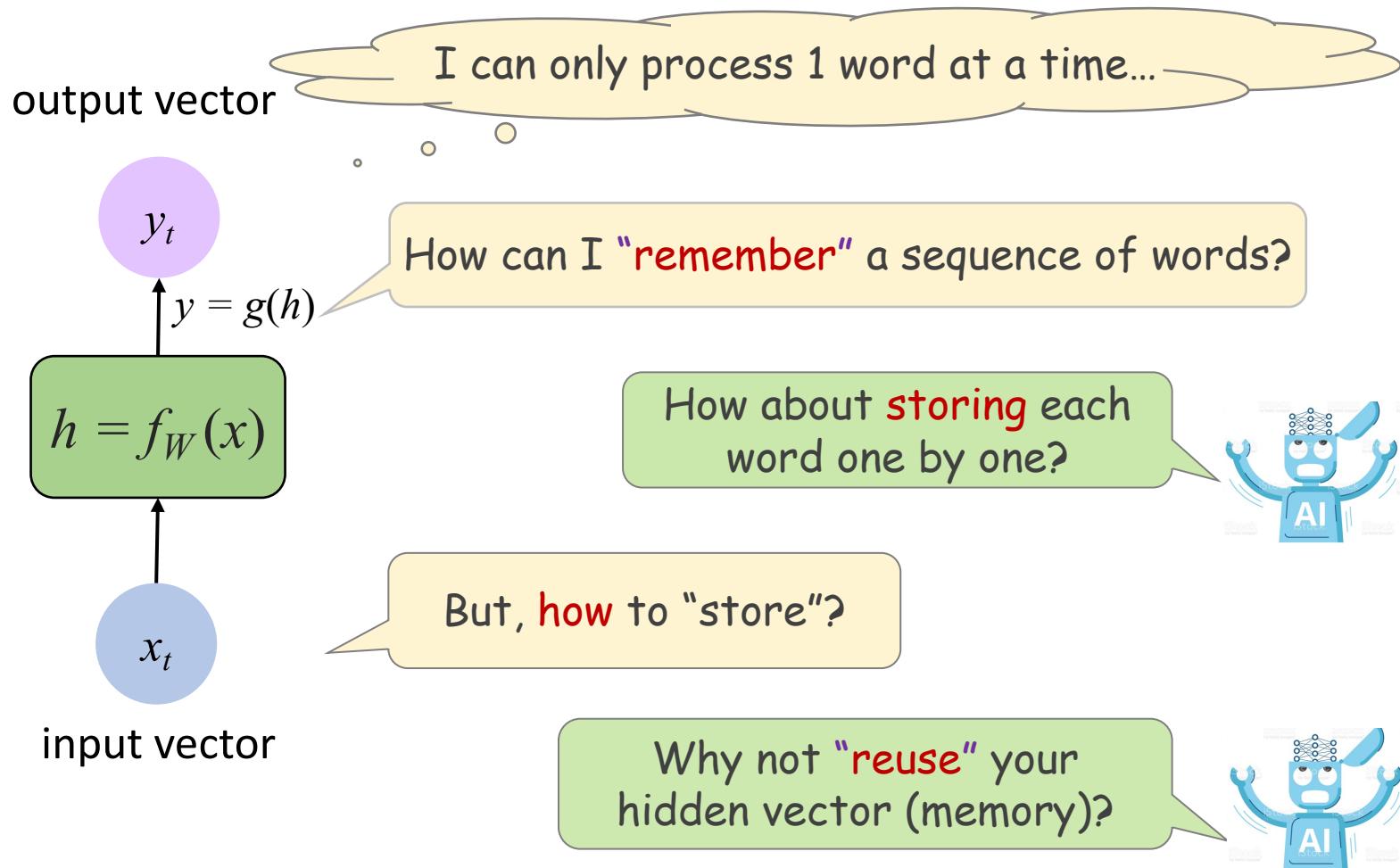
Standard Neural Networks Revisited



$$x \in R^d$$

$$h = \tanh(W^T x) \quad y = Vh$$

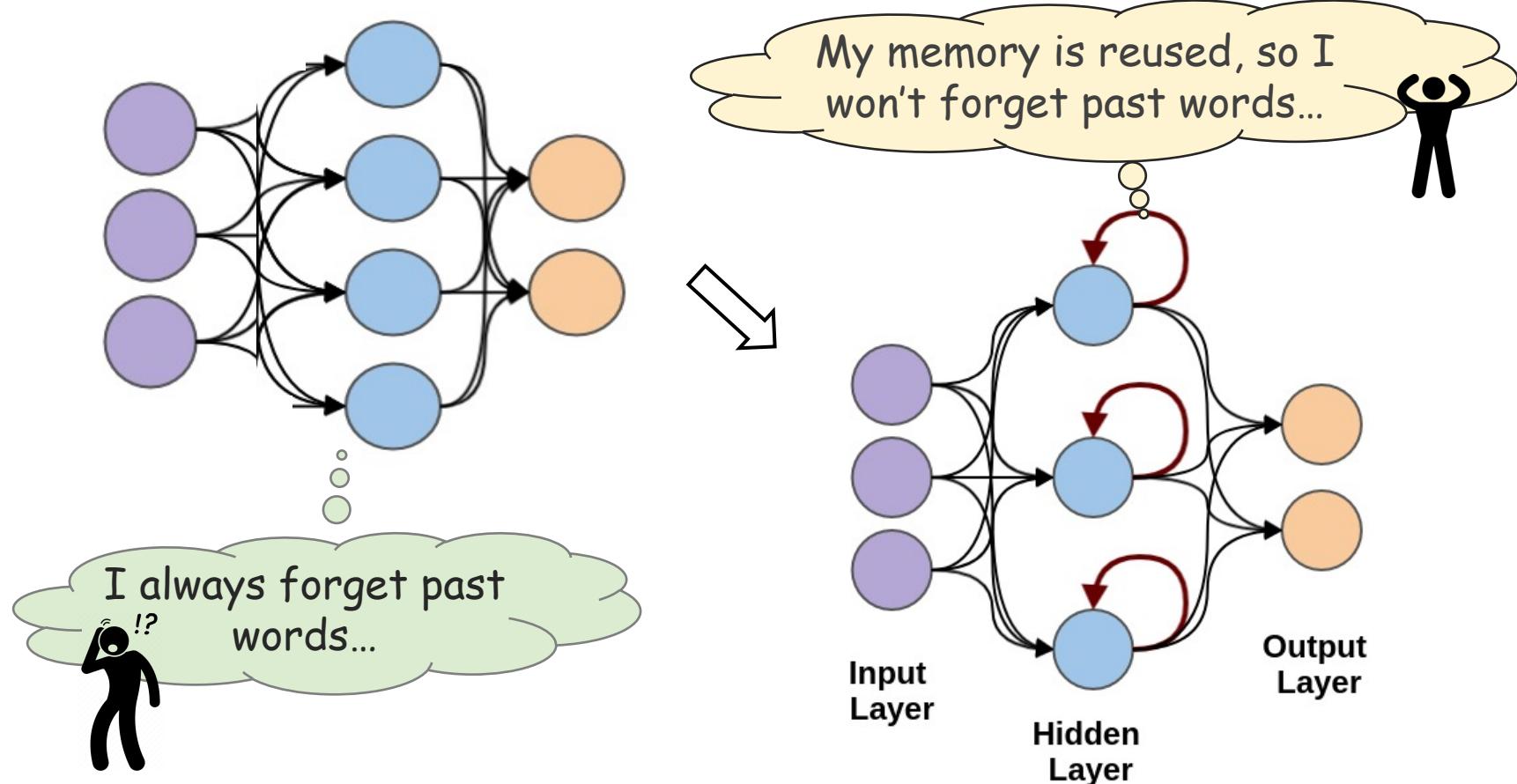
Standard “Vanilla” Neural Network



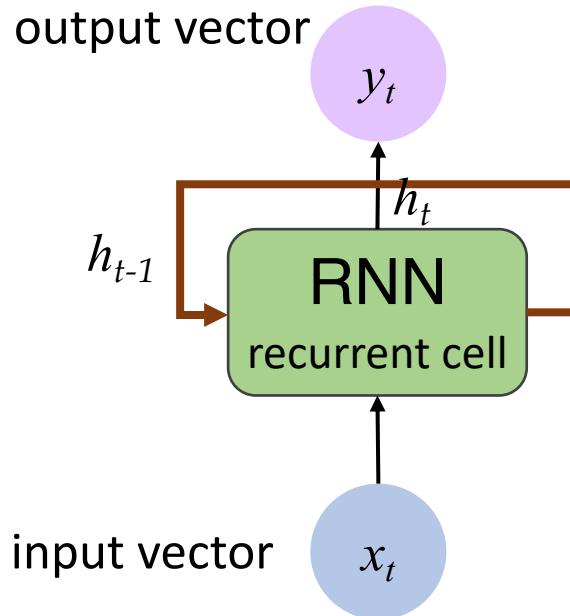
Recurrent Neural Networks



- Reuse the hidden layer to store the input information.



Recurrent Neural Network (RNN)



Apply a **recurrent relation** at every time step to process a sequence:

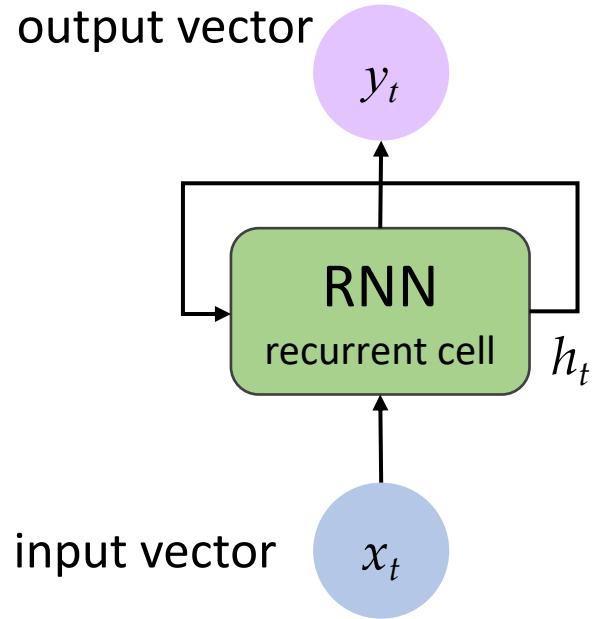
$$h_t = f_W(h_{t-1}, x_t)$$

cell state function old state Input vector at
parameterized by W time step t

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

RNNs have a **state**, h_t , that is updated **at each time step** as a sequence is processed.

RNN State Update and Output



Output Vector

$$y_t = W_{hy}^T h_t$$

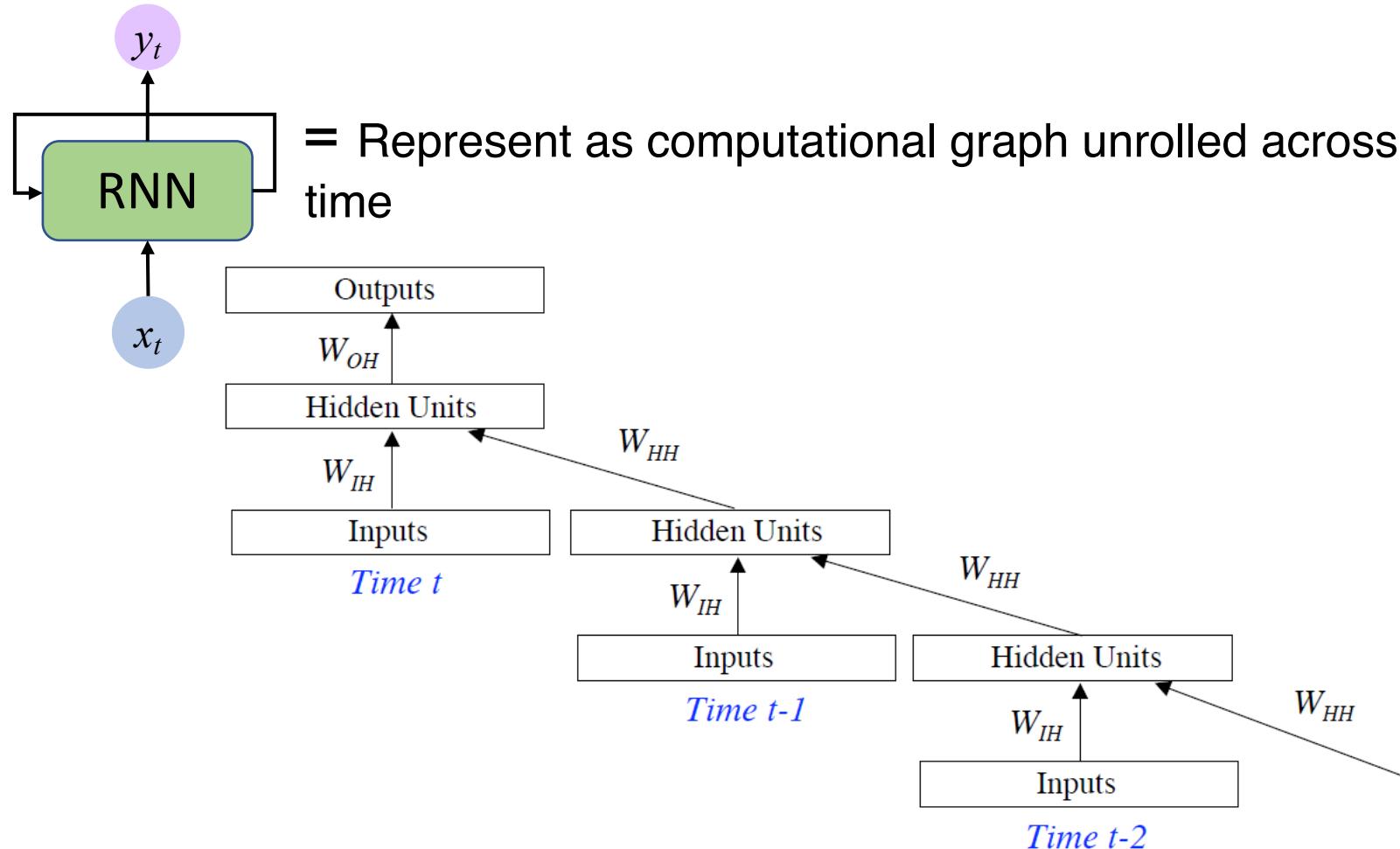
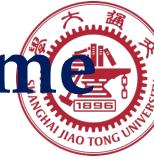
Update Hidden State

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

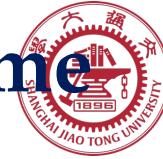
Input Vector

$$x_t$$

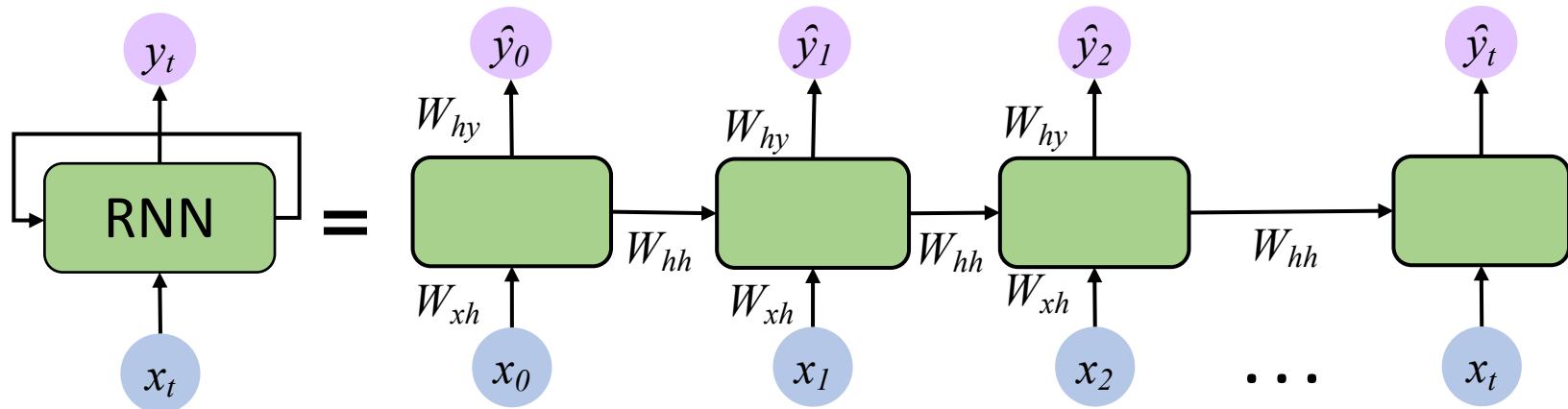
RNNs: Computational Graph Across Time



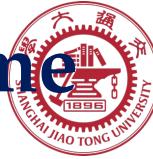
RNNs: Computational Graph Across Time



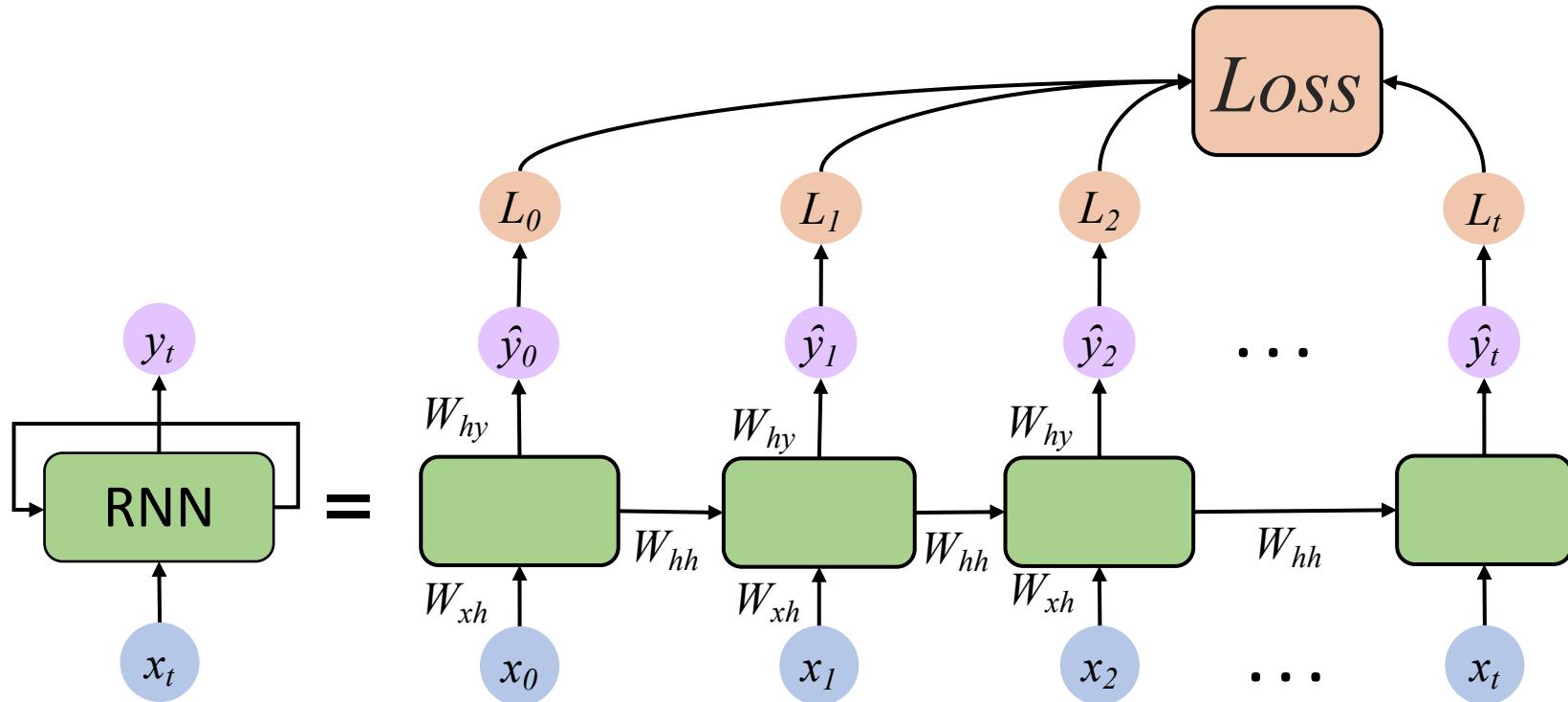
- Re-use the **same weight matrices** at every time step



RNNs: Computational Graph Across Time



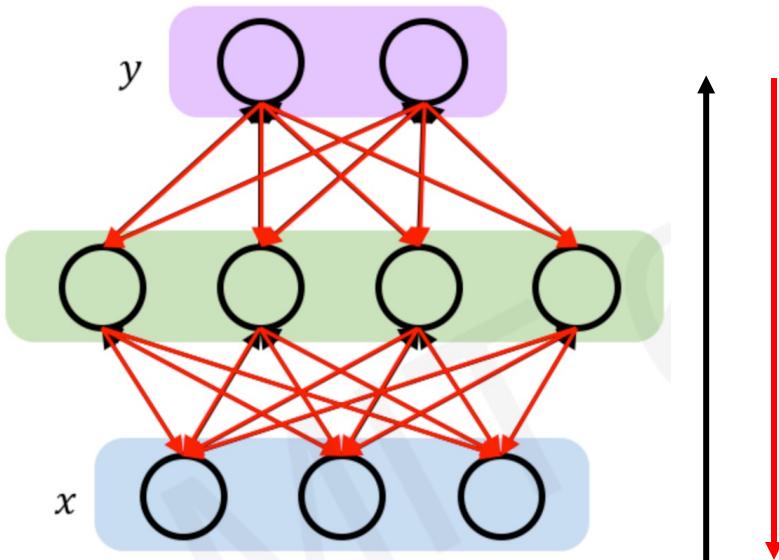
→ forward pass





Backpropagation Through Time (BPTT)

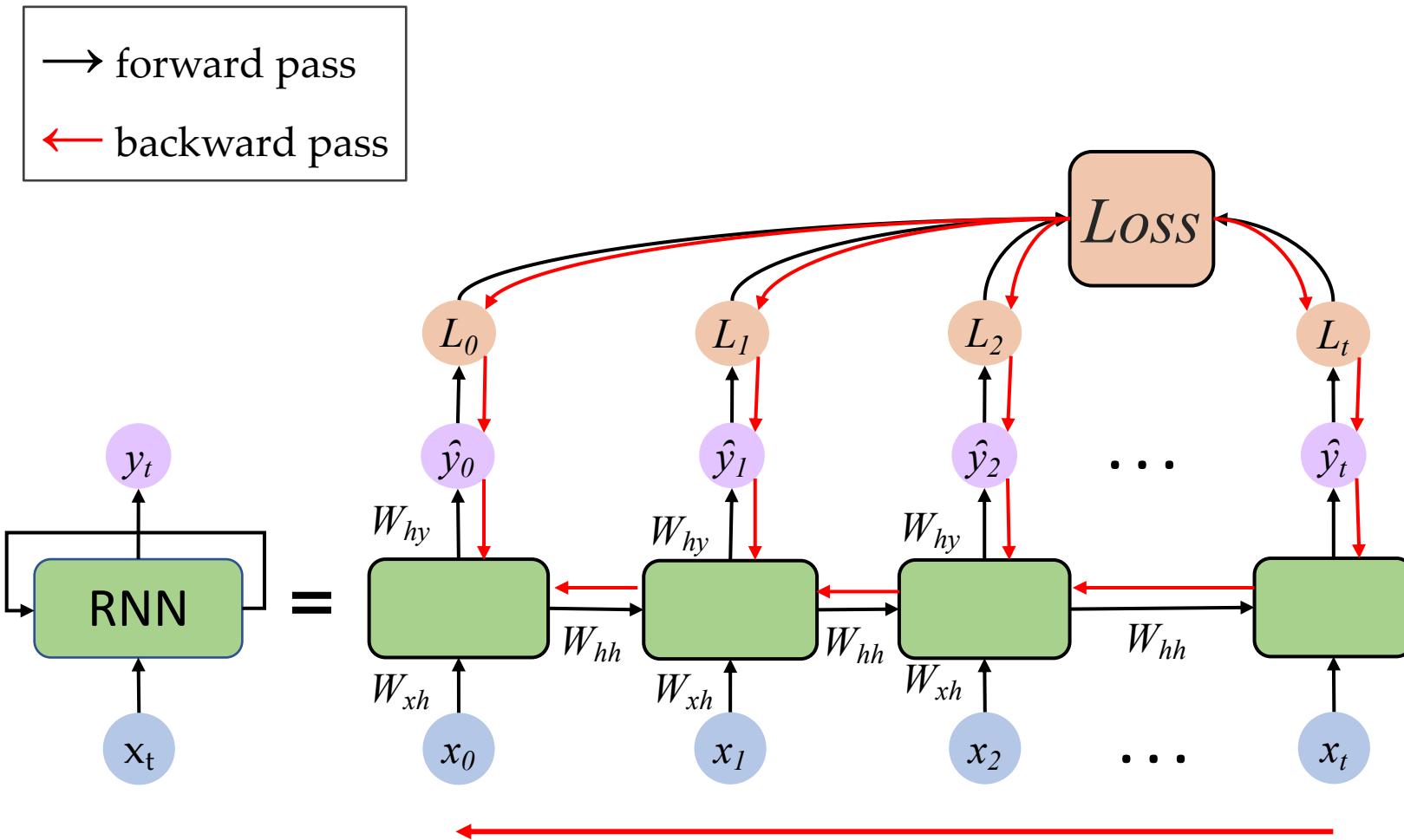
Recall: Backpropagation in Feed Forward NNs



Backpropagation algorithm:

1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

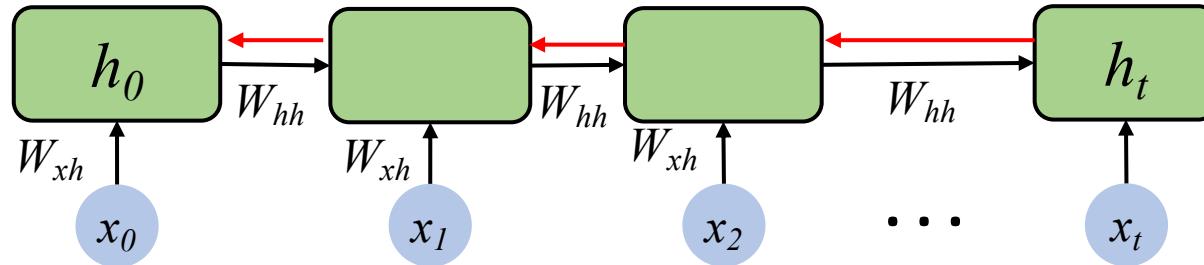
RNNs: Backpropagation Through Time





The Problems of Standard RNNs

- Gradient Flow of Standard RNNs:



Computing the gradient w.r.t. h_0 involves many factors of W_{hh} + repeated gradient computation!

Many values > 1 :
Exploding gradients

Gradient clipping to scale big gradients

Many values < 1 :
Vanishing gradients

Gradient clipping to 0 gradients

The Problem of Long-Term Dependencies



- Why are vanishing gradients a problem?

multiply many small numbers together

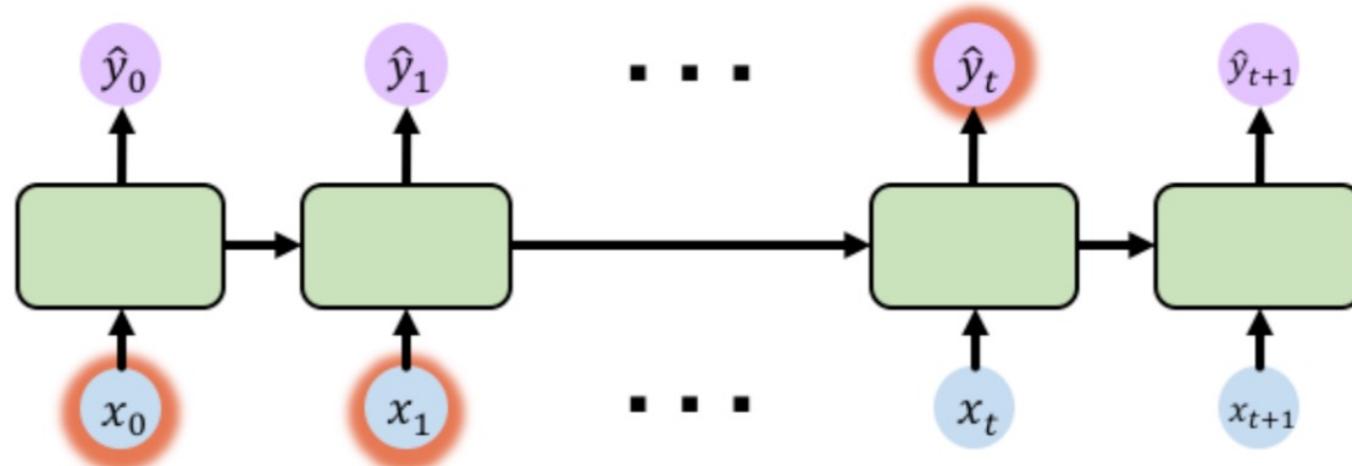


errors due to further back time steps have smaller and smaller gradients



bias parameters to capture short-term dependencies

“I grew up in France, ... and I speak fluent ___ ”



Trick #1: Parameter Initialization



- Initialize weights

Example

- Initialize weights to identity matrix
- Initial biases to zero

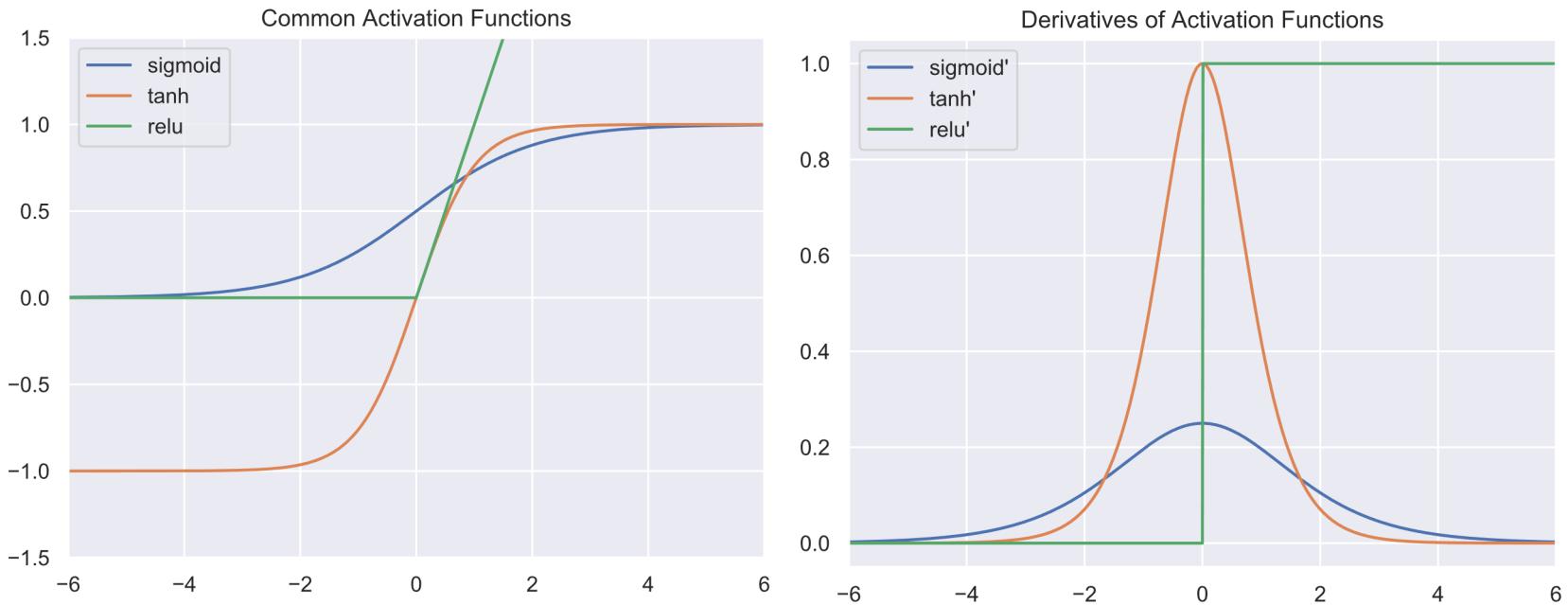
$$W_{init} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

This helps prevent the weights from shrinking to zero.

Trick #2: Better Activation Functions



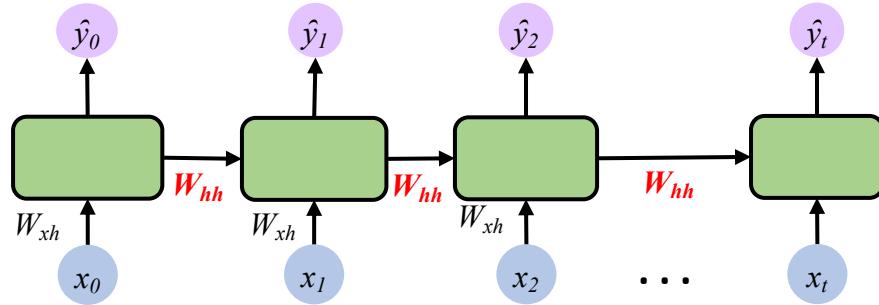
- Using activation functions that have larger derivatives



Using ReLU prevents f' from shrinking the gradients when $x > 0$



Long vs. Short-term Memory



RNN: information is added through multiplication.

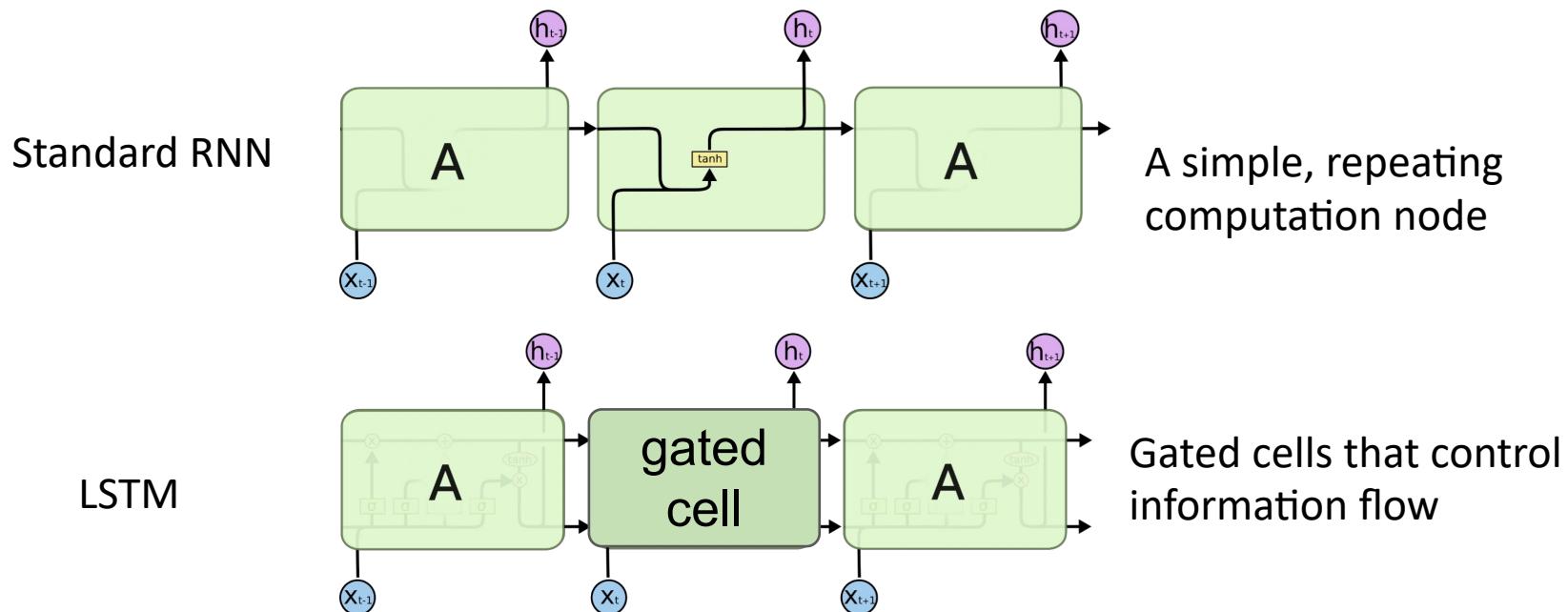
How does our memory evolve?
Multiplication?



Solution #3: Gated Cells



- Idea: use a more complex recurrent unit with gates to control what information is passed through.

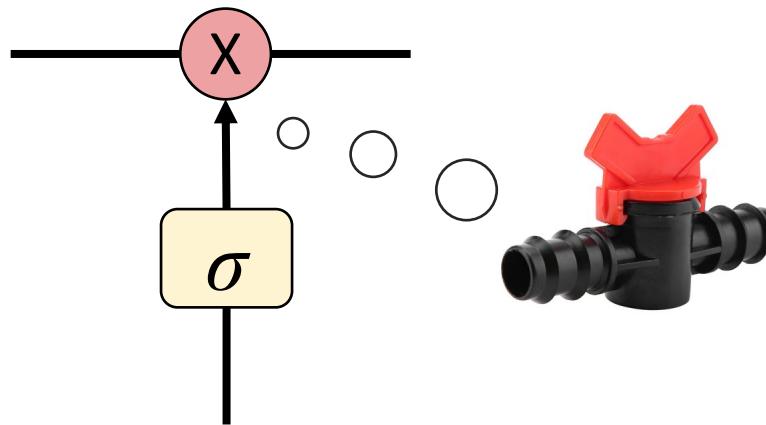


Long Short Term Memory (LSTM) networks rely on a **gated cell** to track information through many time steps.

Memory Gates



- Information is **added** or **removed** through structures called **gates**.



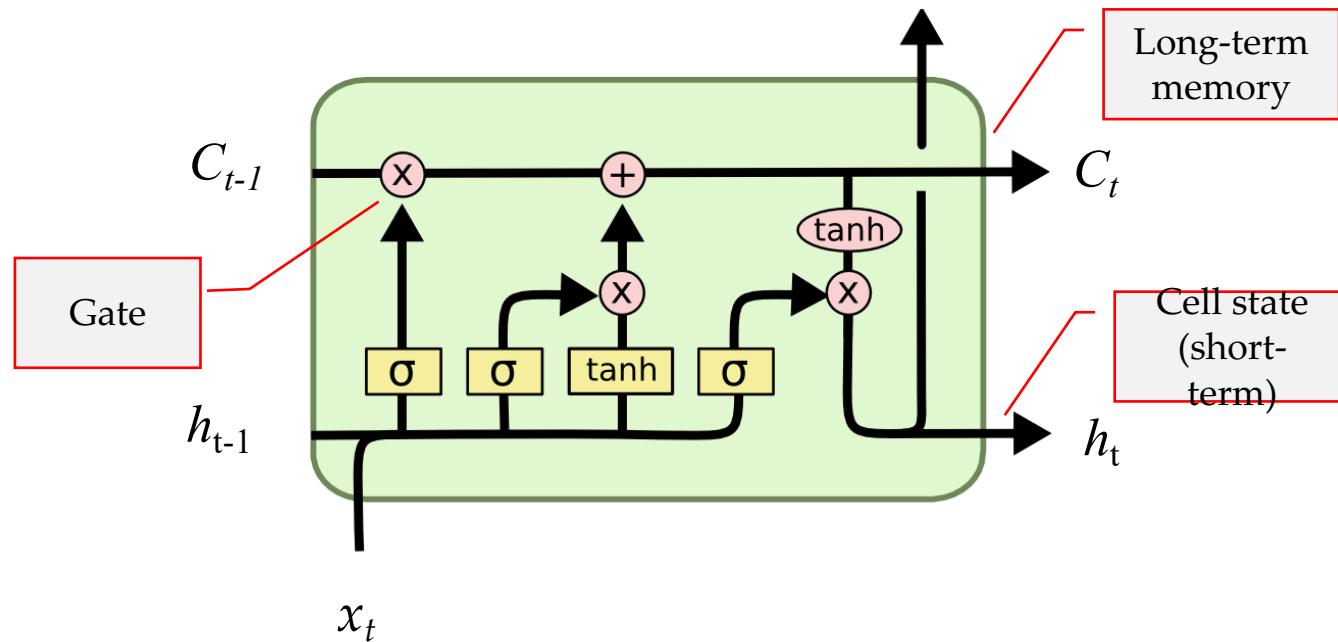
Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

Long Short Term Memory (LSTMs)



- How do LSTMs work?

1) Forget 2) Store 3) Update 4) Output

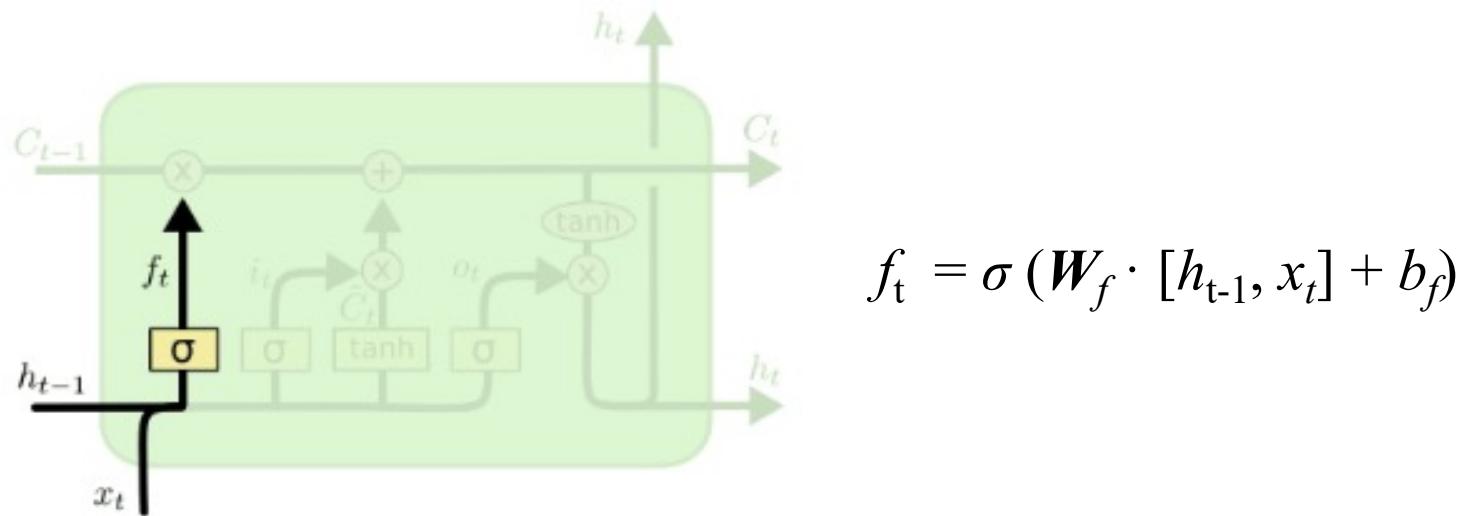


Long Short Term Memory (LSTMs)



1) Forget 2) Store 3) Update 4) Output

- LSTMs **forget irrelevant** parts of the previous state

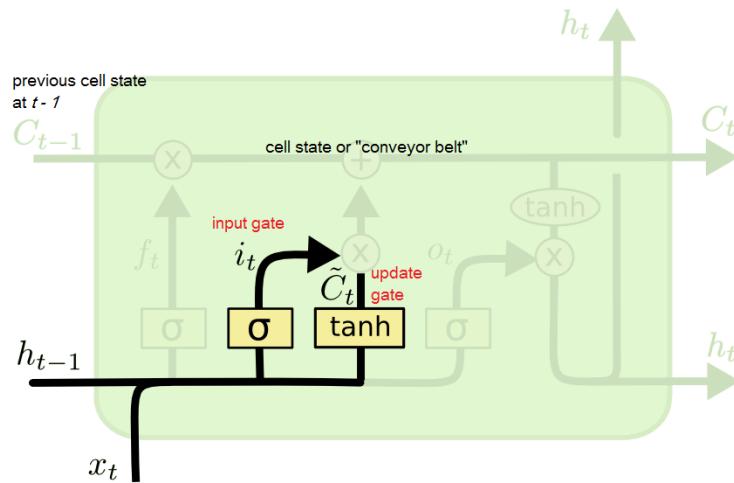


Long Short Term Memory (LSTMs)



1) Forget 2) Store 3) Update 4) Output

- LSTMs **store relevant** new information into the cell state.



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

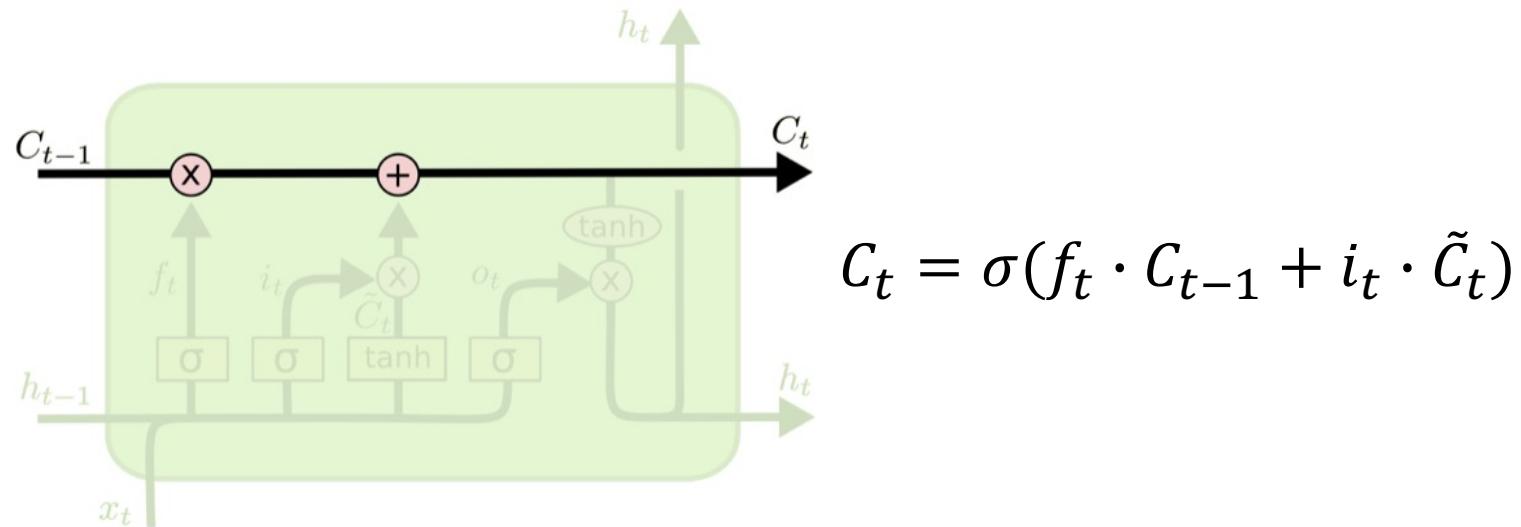
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



Long Short Term Memory (LSTMs)

- 1) Forget
- 2) Store
- 3) Update
- 4) Output

- LSTMs **selectively update** cell state values

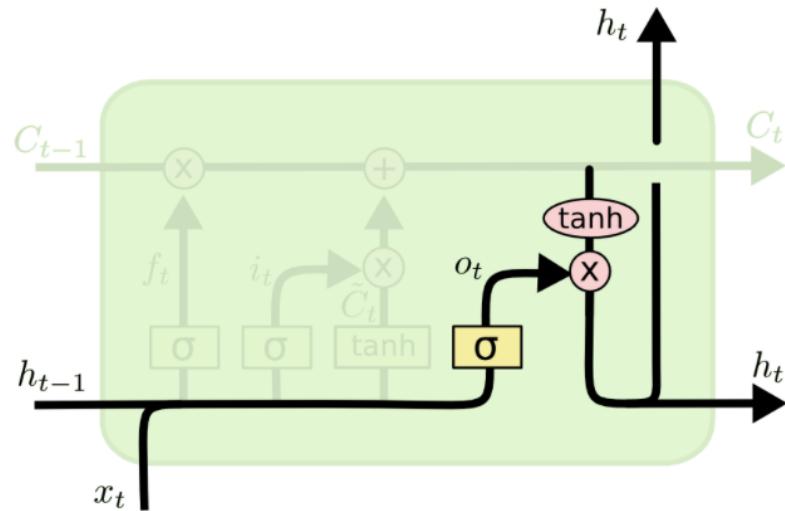


Long Short Term Memory (LSTMs)



1) Forget 2) Store 3) Update 4) Output

- The **output gate** controls what information is sent to the next time step.



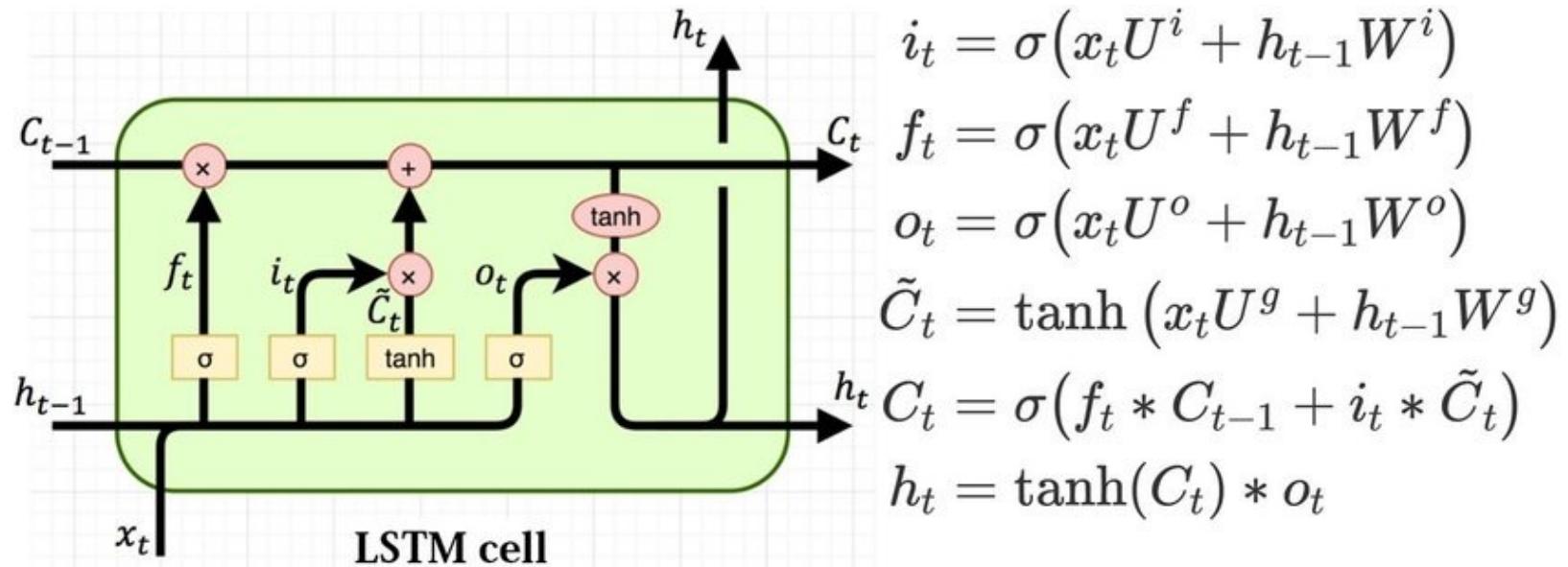
$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

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

Long Short Term Memory (LSTMs)



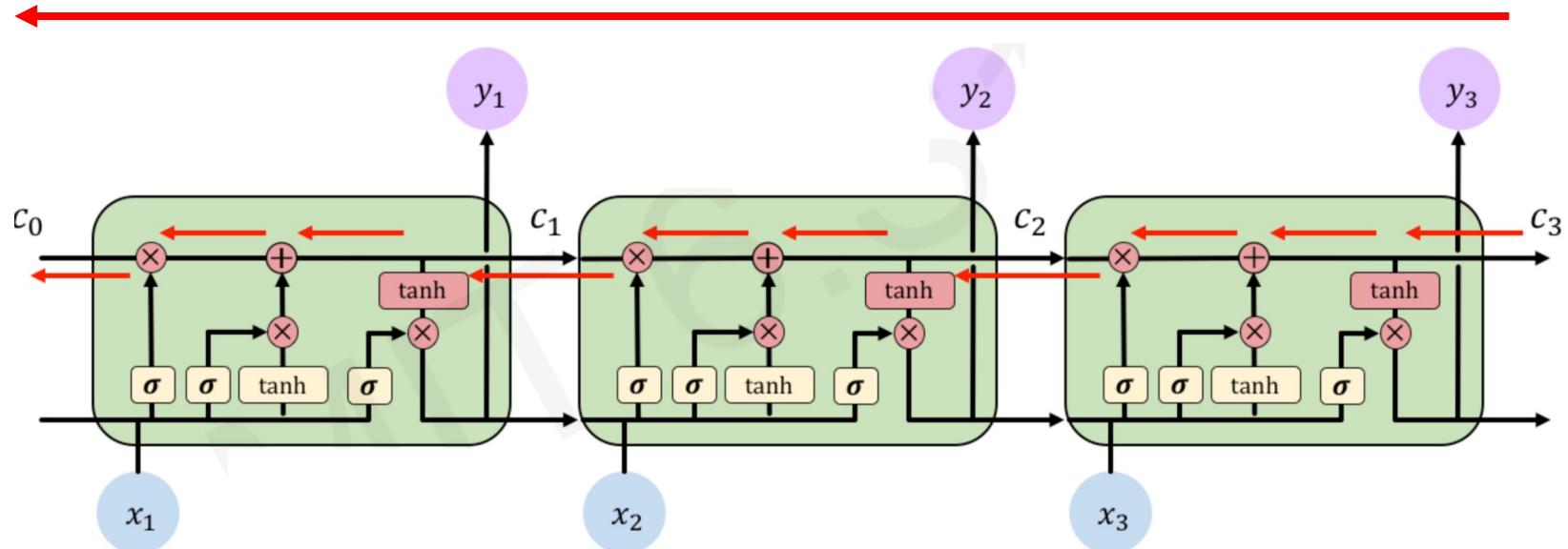
- 1) Forget
- 2) Store
- 3) Update
- 4) Output



LSTM Gradient Flow



Uninterrupted gradient flow!



LSTMs: Key Concepts

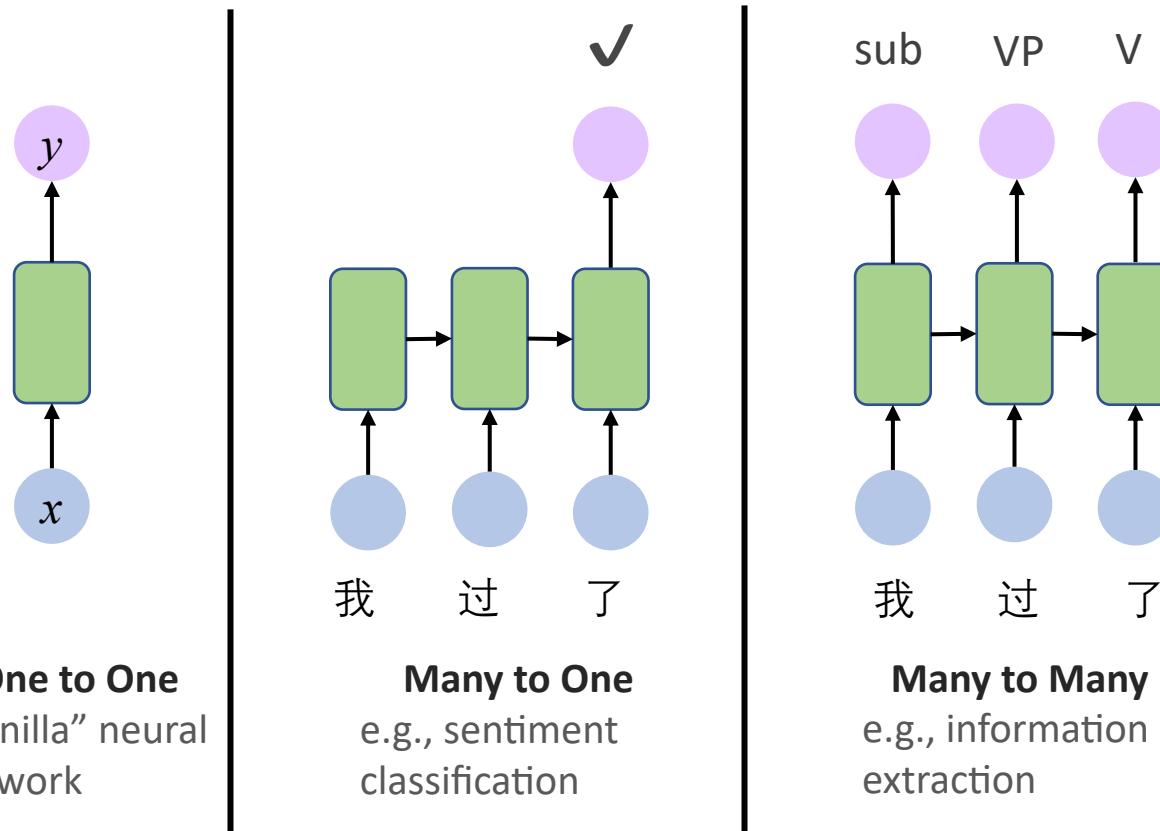


1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the flow of information
 - ▷ **Forget** gate get rid of irrelevant information
 - ▷ **Store** relevant information from current input
 - ▷ Selectively **update** cell state
 - ▷ **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with **uninterrupted gradient flow**



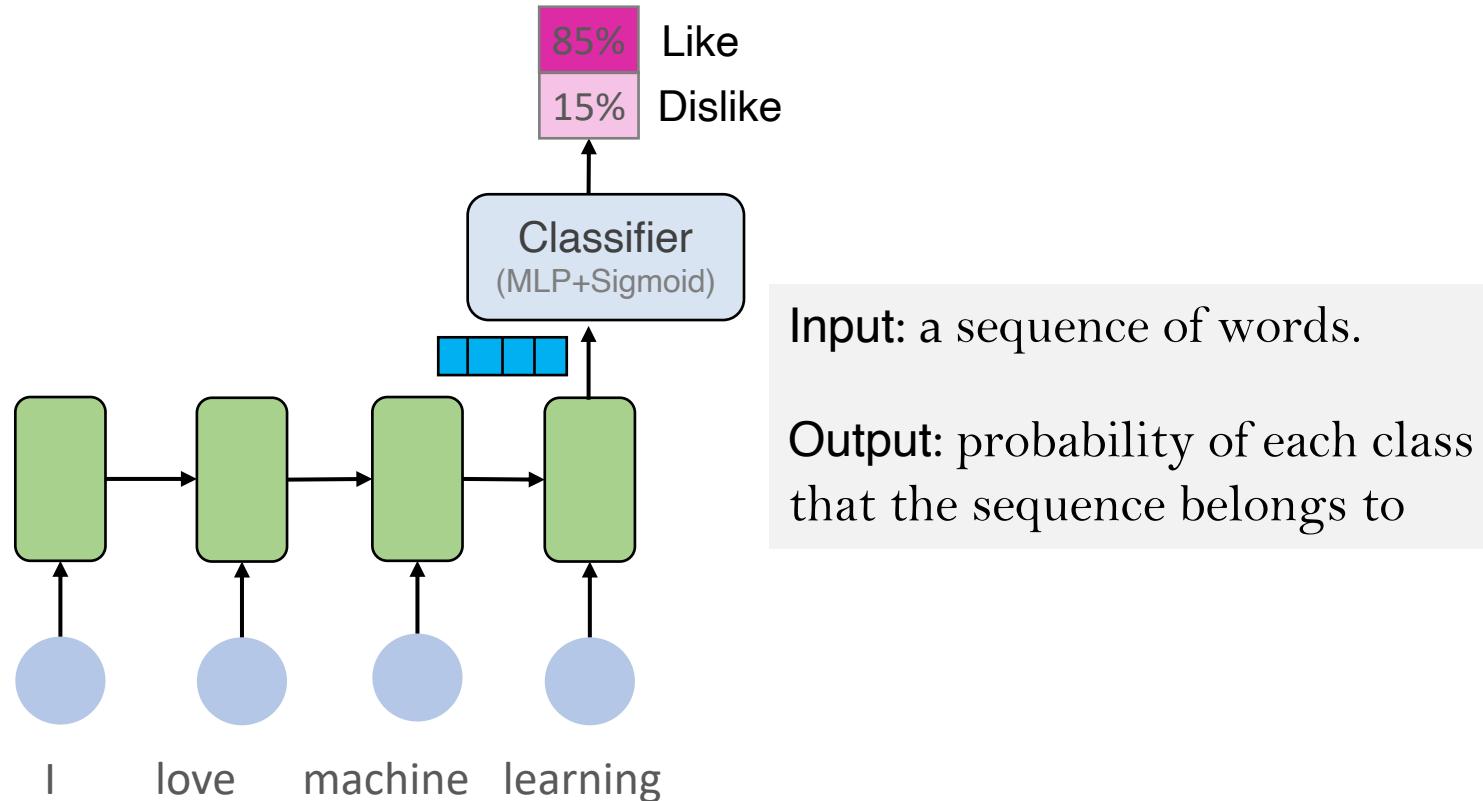
RNN Applications

RNNs for Sequence Modeling



...and many other
architectures and
applications

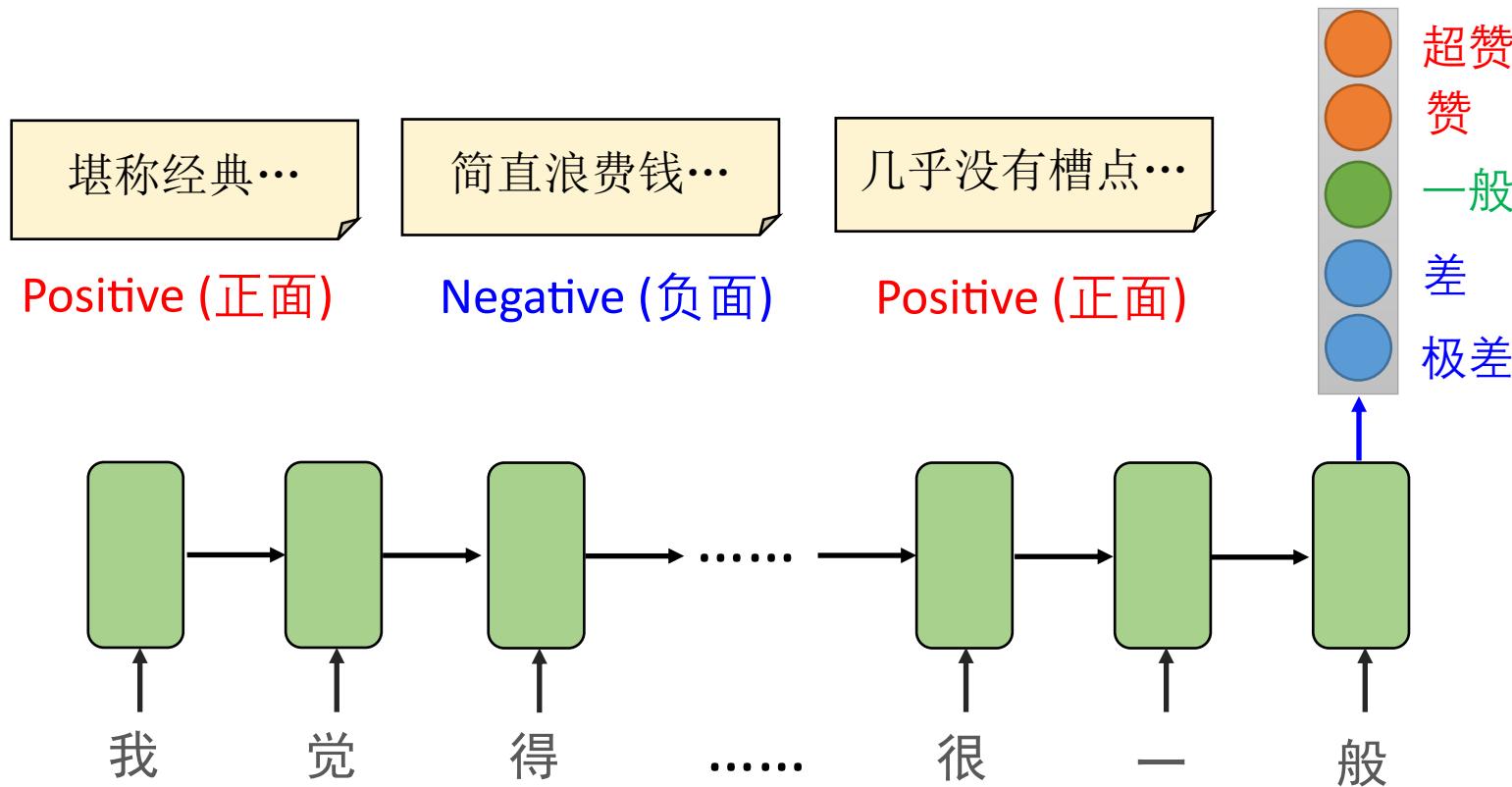
RNN for Sequence Classification



Example Task: Sentiment Classification



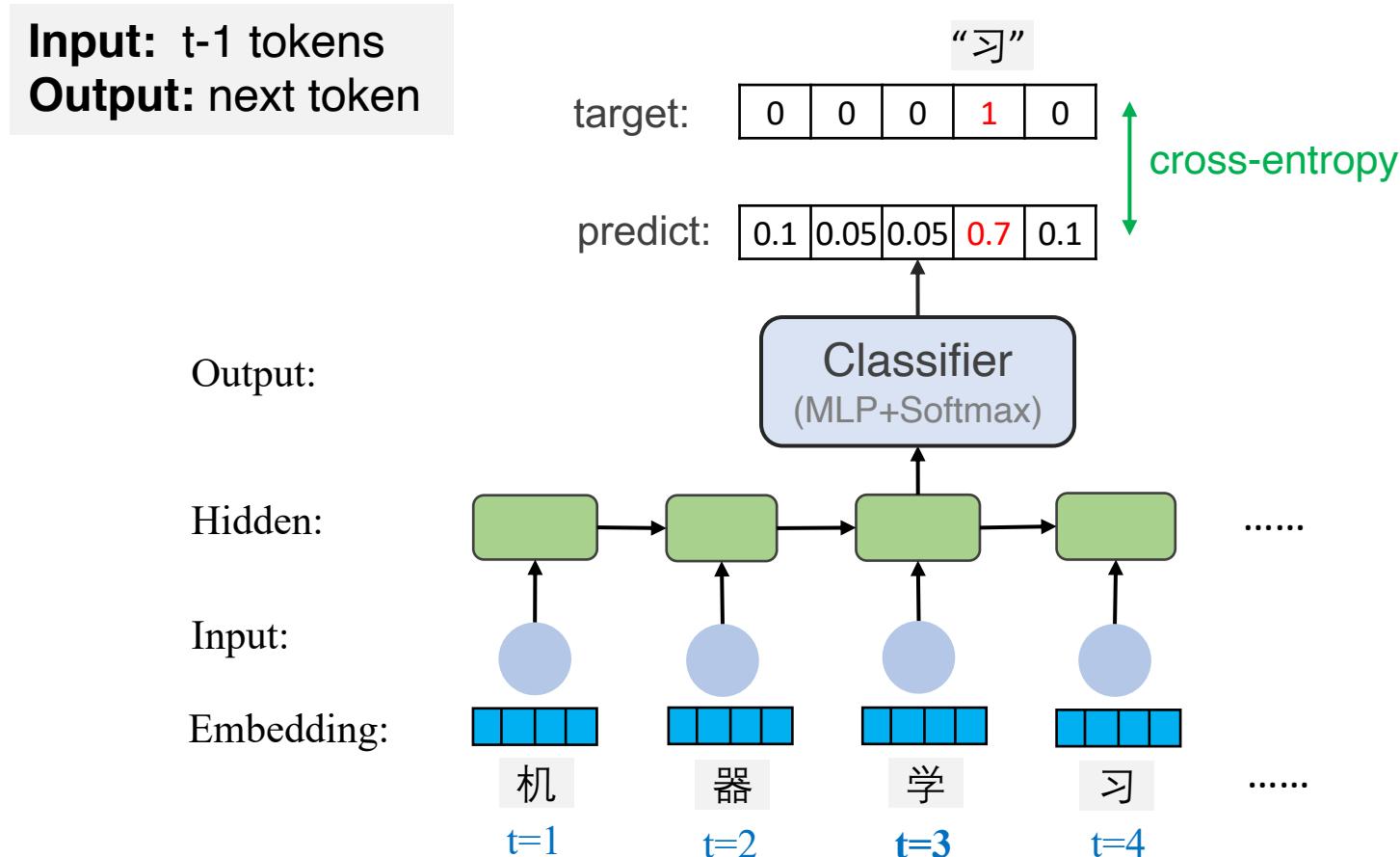
- Input is a vector sequence, but output is only one vector



RNN for Sequence Generation

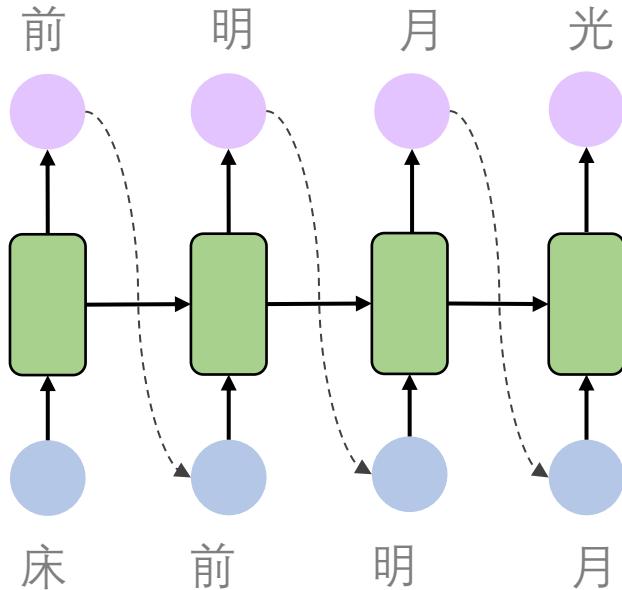


- RNN can also be used to generate a sequence.





Example Task: Poetry Generation



Input: n-1 poetry characters

Output: next character in a poetry

TIME for Coding



Tutorial: poetry generation using Pytorch

https://gitee.com/wannabe-9/LSTM_poem1

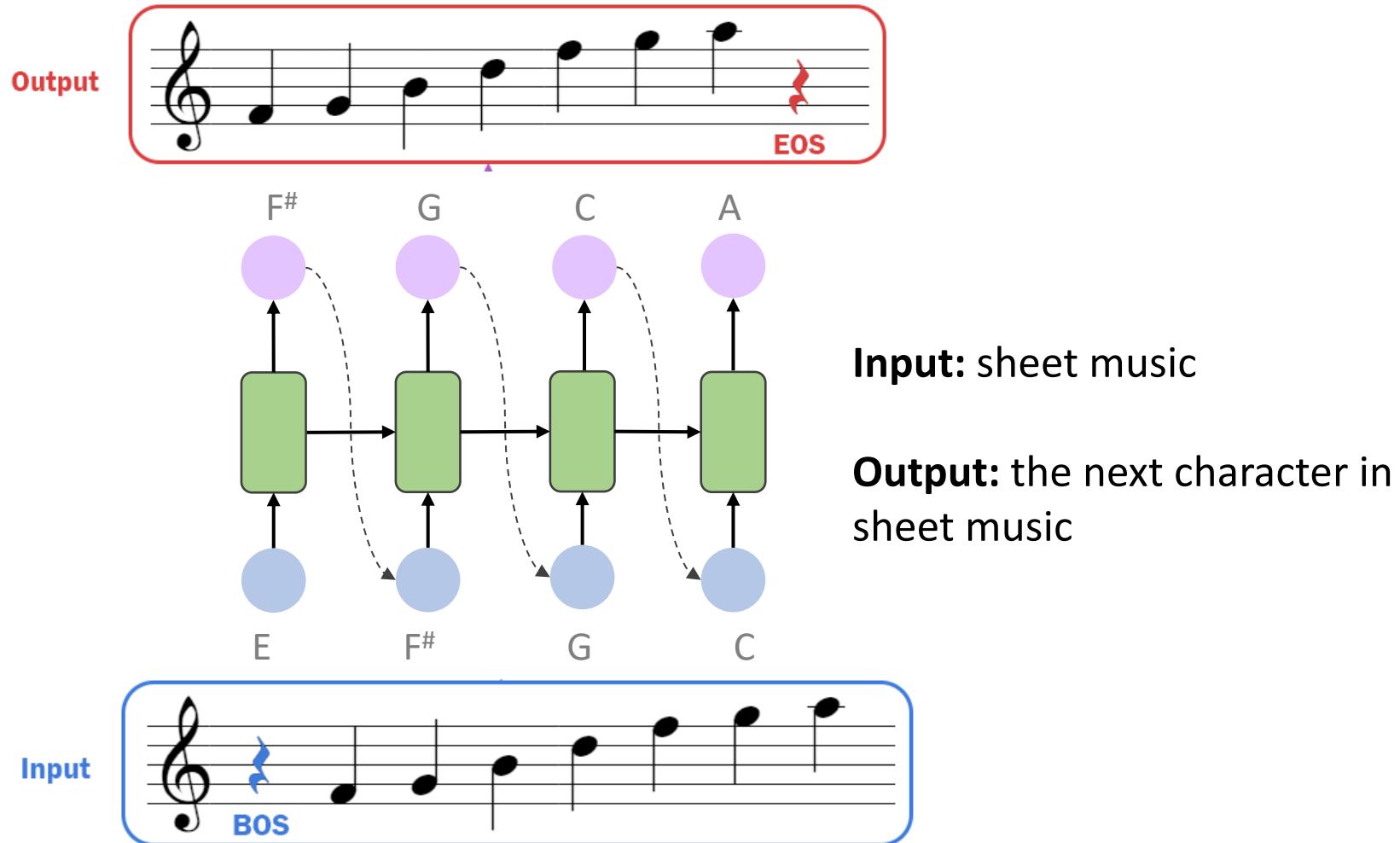
<https://zhuanlan.zhihu.com/p/138270447>

<https://www.kaggle.com/code/jiaminggogogo/ancient-chinese-poem-generation-rnn/notebook>





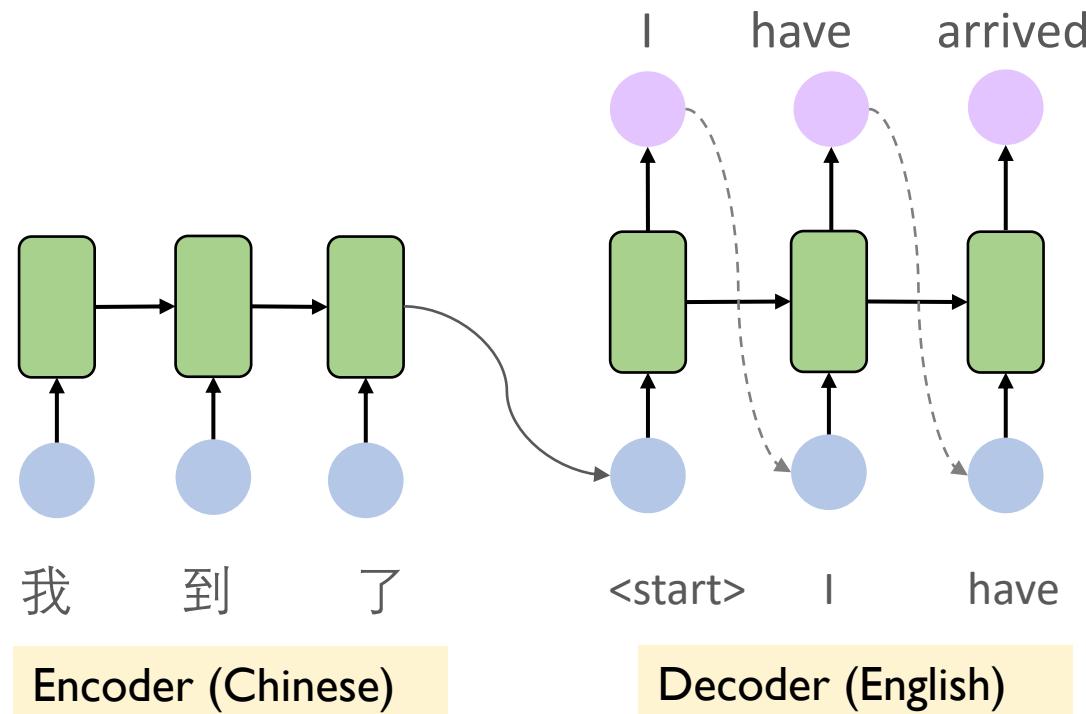
Example Task: Music Generation



Example Task: Machine Translation



- Generate a sequence conditioned on the other sequence.



To elaborate in the next lecture.

