Machine Learning

Chapter 11: Recurrent Neural Networks

Fall 2021

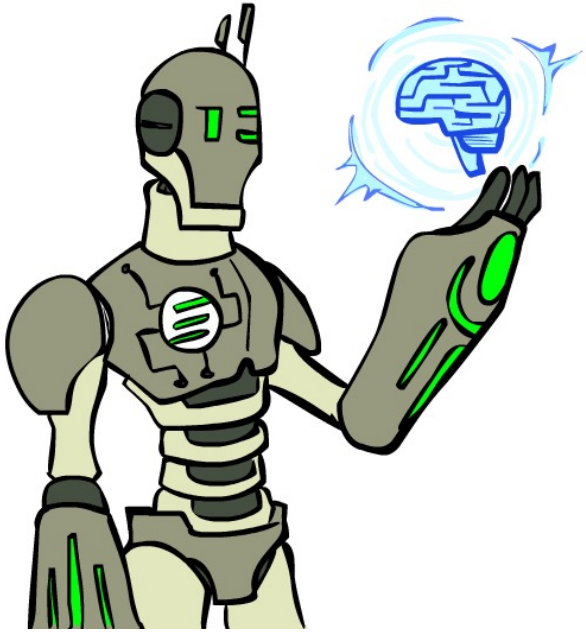
Instructor: Xiaodong Gu

Deep Neural Networks

- Feedforward neural networks
- Convolutional neural networks
- Recurrent neural networks
- …

Today

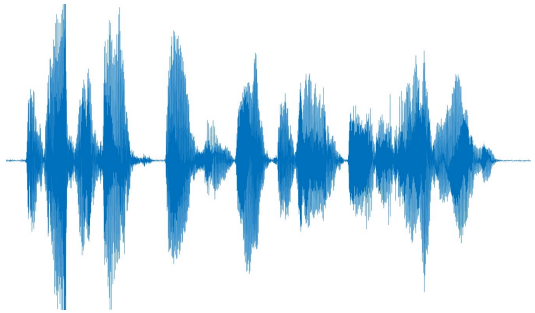
- Recurrent Neural Networks



Sequence in the Wild

Sequences in the Wild

- Audio:



- Word:

SE-125 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 use **language model:**

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

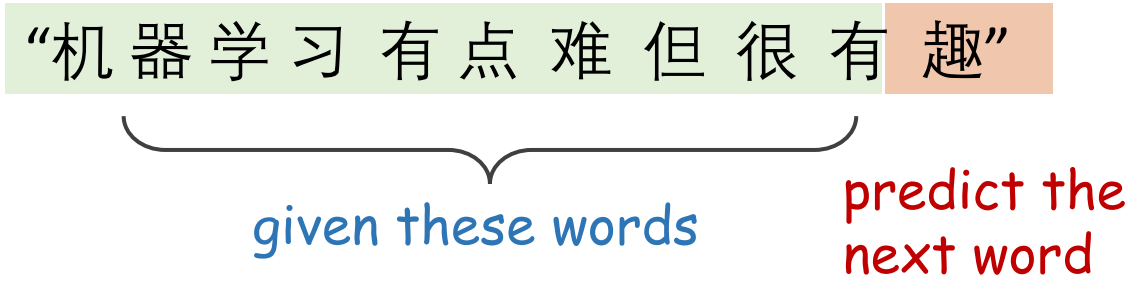
$$p(w_1, \ldots, w_N) = p(w_1)\, p(w_2|w_1) \ldots p(w_N|w_1, \ldots, w_{N-1})$$
$$\approx p(w_1)p(w_2|w_1)p(w_3|w_2)\ldots p(w_N|w_{N-1})$$

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

- The key problem:
  - given a sequence of n-1 tokens
  - predict the next token

A Sequence Modeling Problem

- **Predict the next token** $p(w_t|w_1,\ldots,w_{t-1})$



A Sequence Modeling Problem

- **Idea #1: Counting?**

“机 器 学 习 有 点 难 但 很 有 趣”

given these words

predict the next word

$$p(\text{趣} \mid \text{机器学习有点难但很有}) = \frac{\text{count}(\text{机器学习有点难但很有趣})}{\text{count}(\text{机器学习有点难但很有})}$$

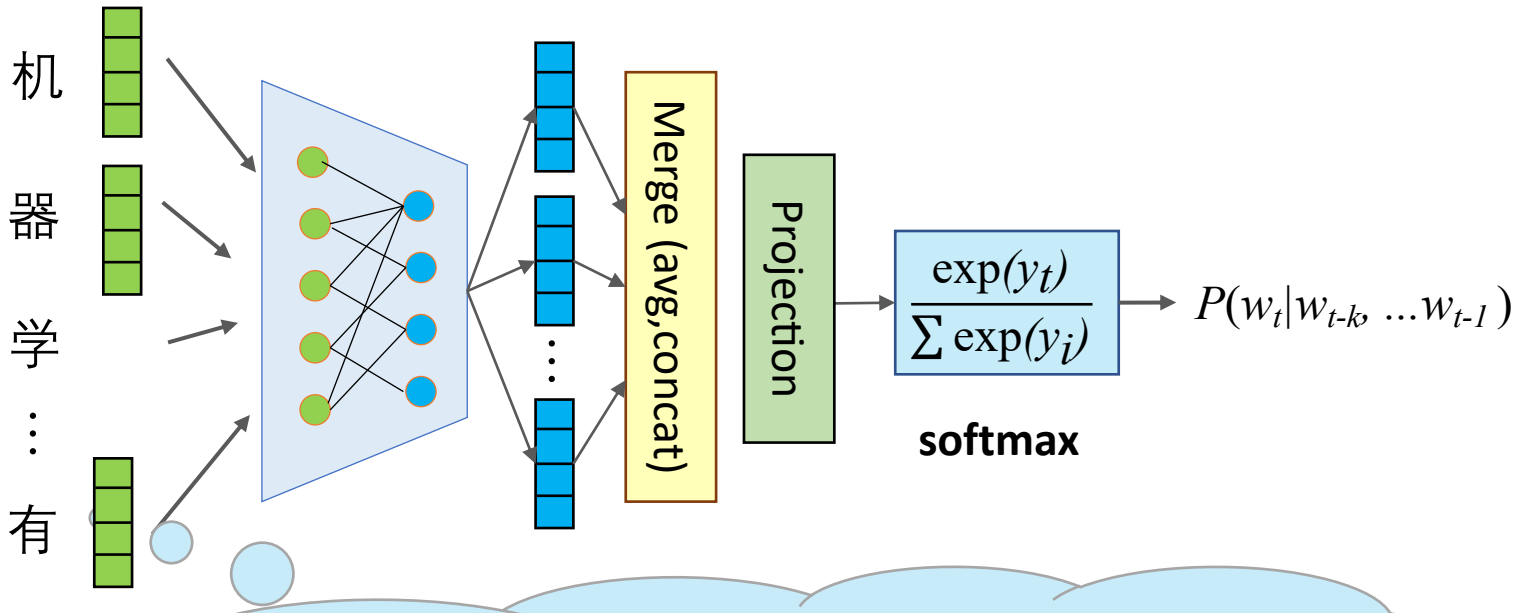
Problems:

- Discrete symbols,
- No word semantics,
- Ignore similarity between words and sentences.

How about “机器学习不容易,不过挺有__”?

Language Model using Neural Networks

- **Idea #2: Using Neural Networks?**

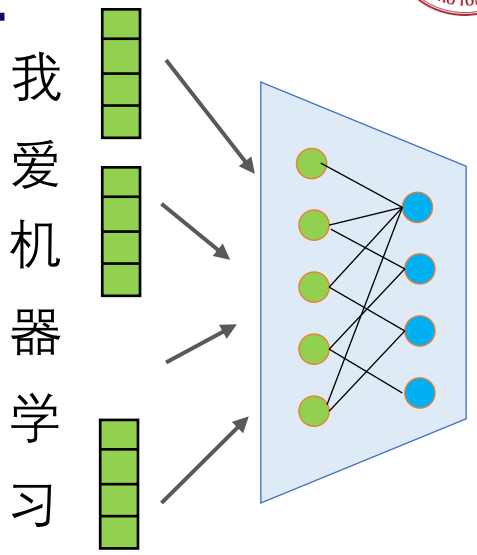


Limitations:

- Bag-of-words? (average)
- Variant Input Length? (concatenate)
- (Long) dependences between words?

Sequence Modeling: Design Criteria

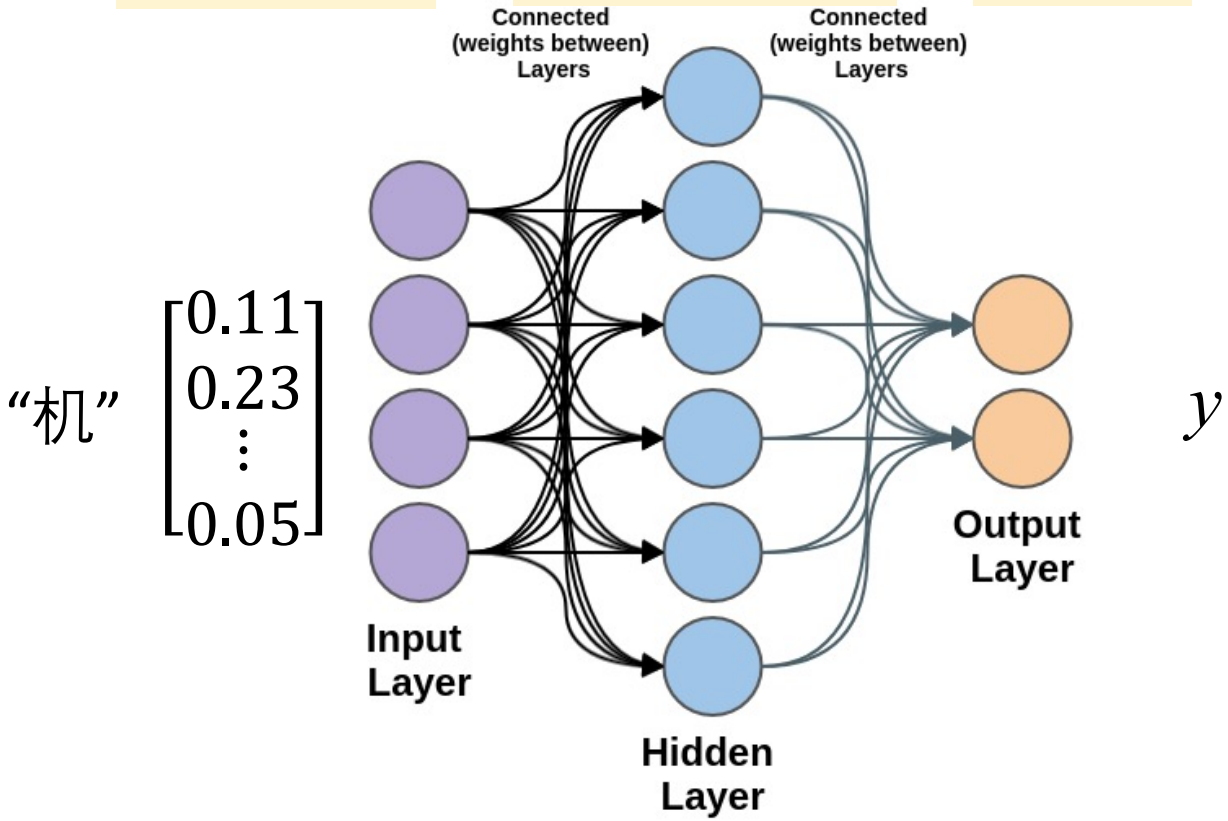
- To model sequences, we need to:
  1. Handle variable-length sequences
  2. Track long-term dependencies
  3. Maintain information about order
  4. Share parameters across the sequence



- **Recurrent Neural Networks (RNNs)** as an approach to sequence modeling problems

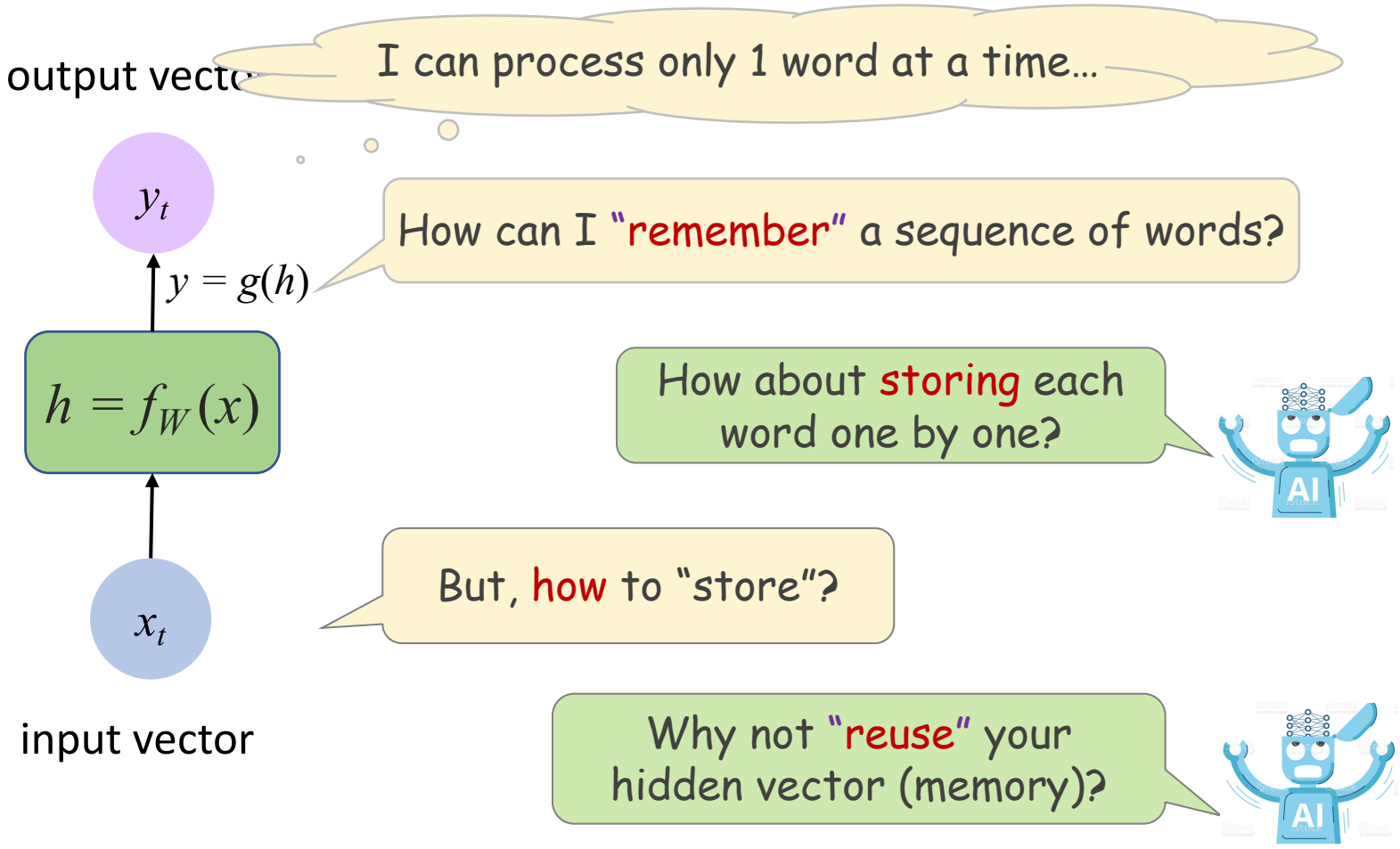
Standard Neural Networks Revisited

Information | Hidden State | Output



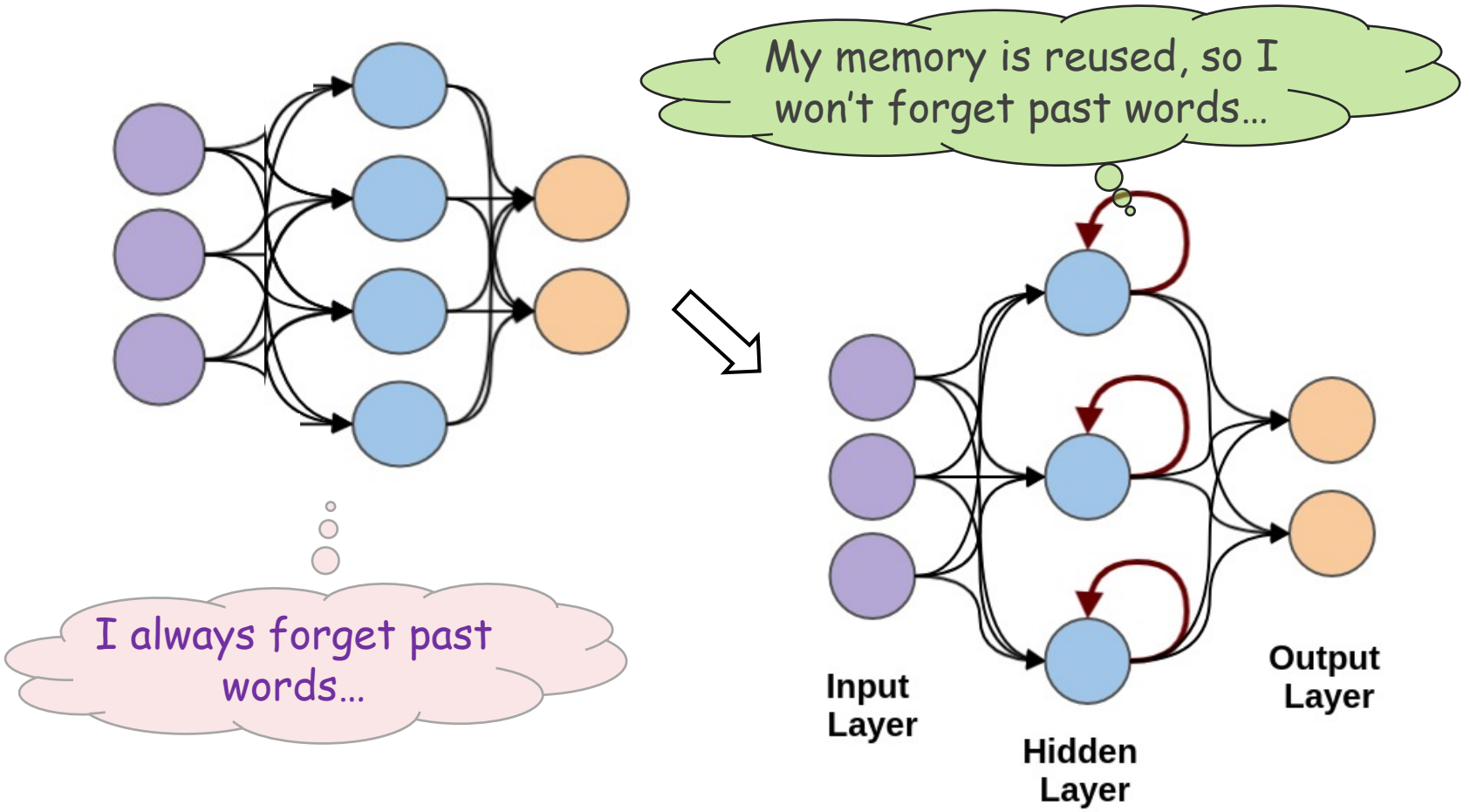
$x \in R^d$ $\quad h = \tanh(W^T x)$ $\quad y = Vh$

Standard "Vanilla" Neural Network

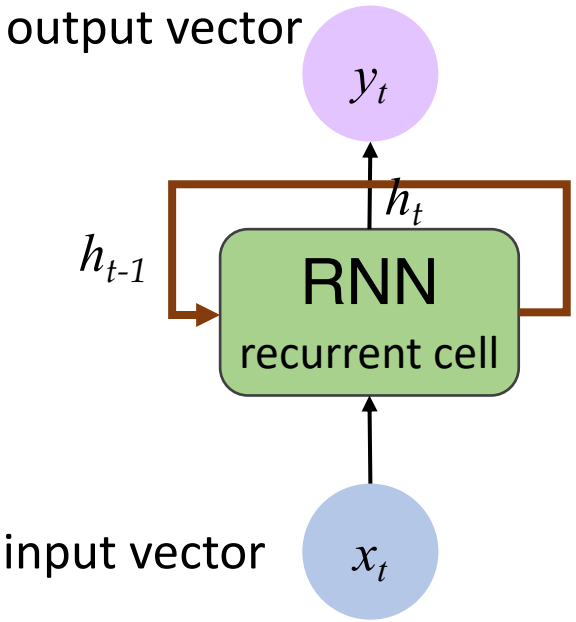


Recurrent Neural Networks

- **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:

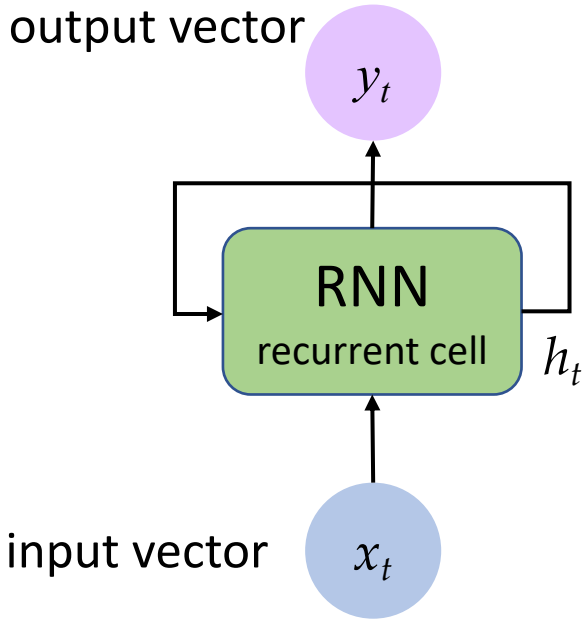
$$h_t = f_W(h_{t-1}, x_t)$$

cell state — function parameterized by W — old state — Input vector at 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

$$y_t = \boldsymbol{W}_{hy}^T h_t$$

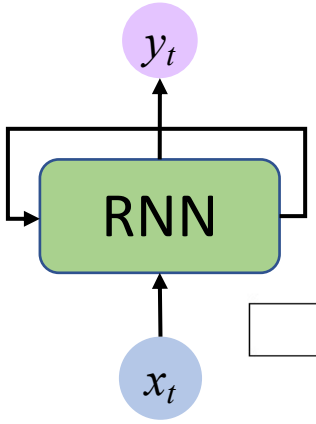
Update Hidden State

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

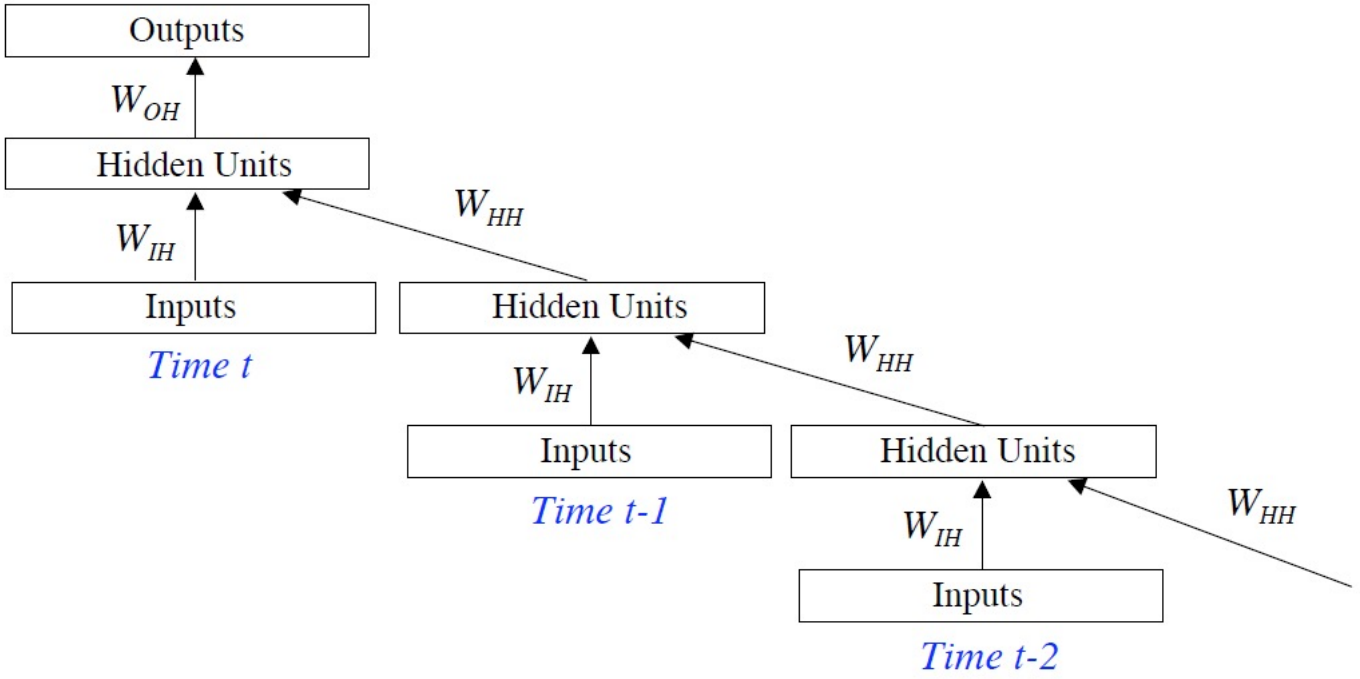
Input Vector

$$x_t$$

RNNs: Computational Graph Across Time

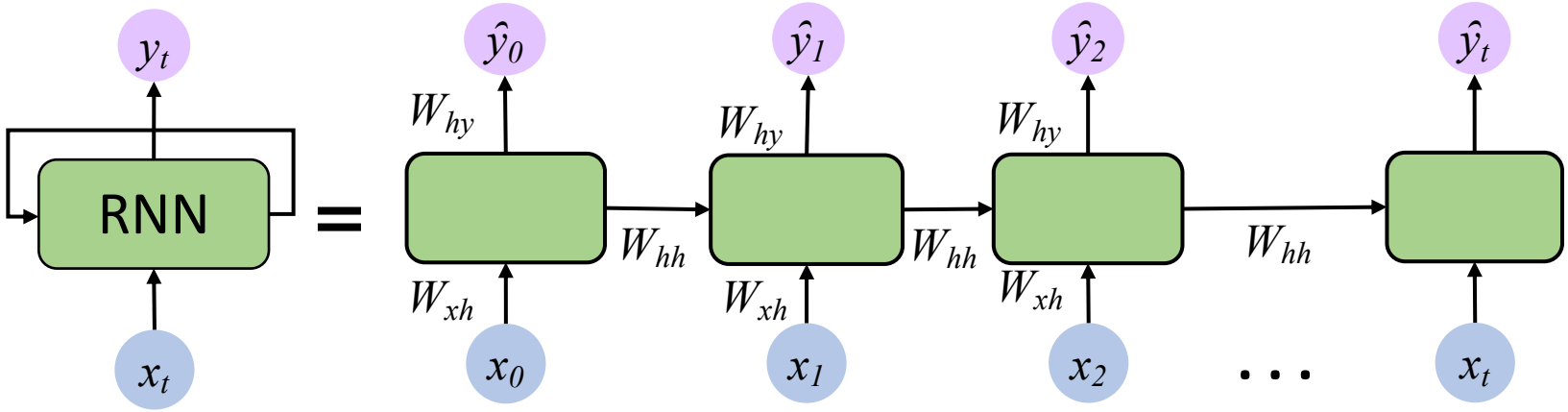


= Represent as computational graph unrolled across time



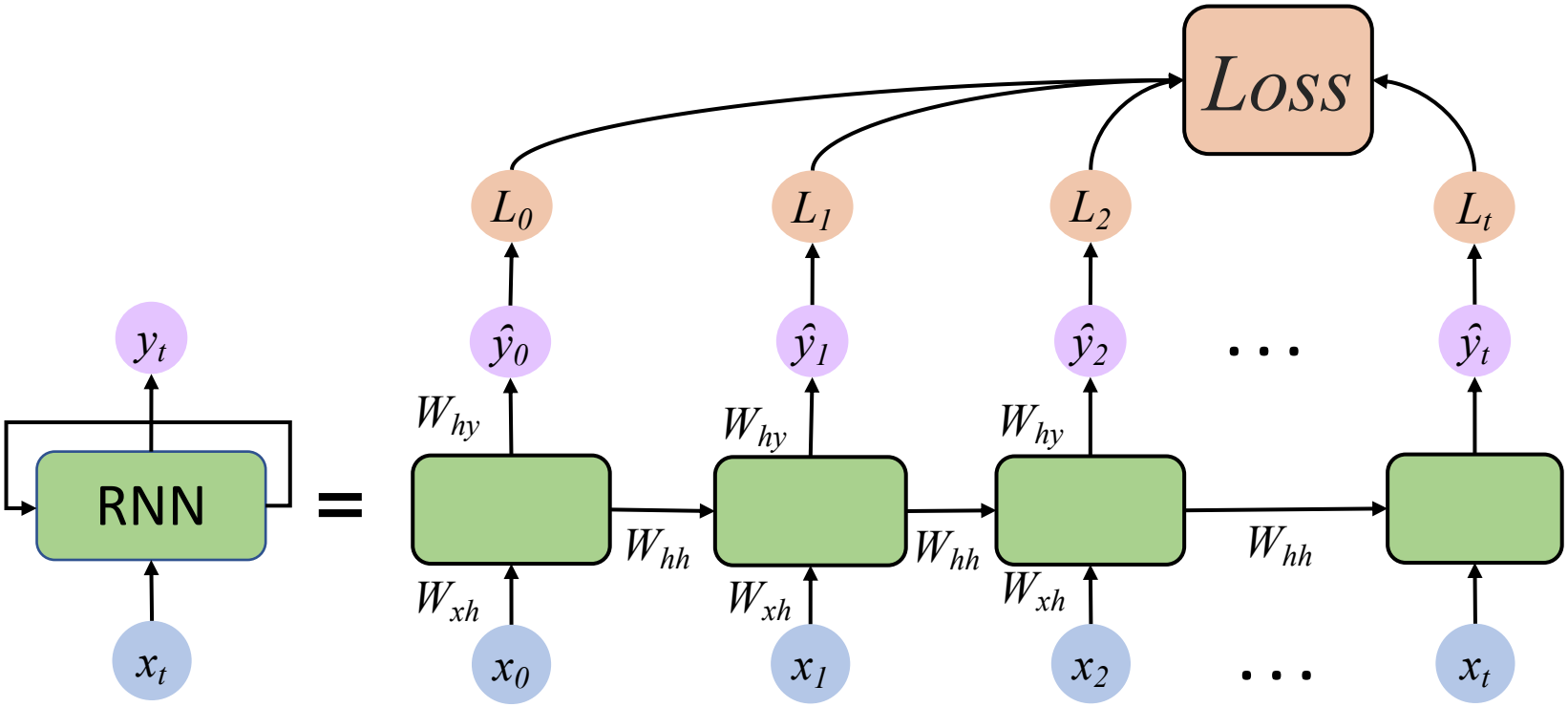
RNNs: Computational Graph Across Time

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



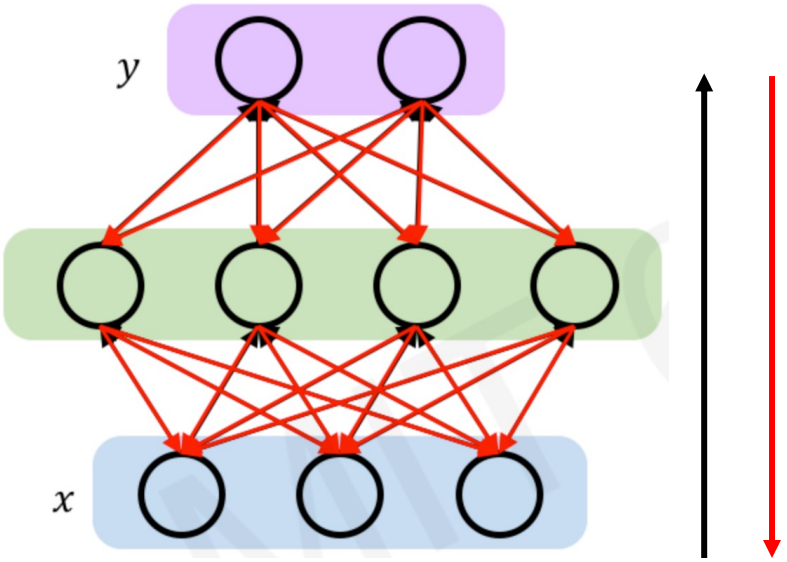
RNNs: Computational Graph Across Time

- → **forward pass**



Backpropagation Through Time (BPTT)

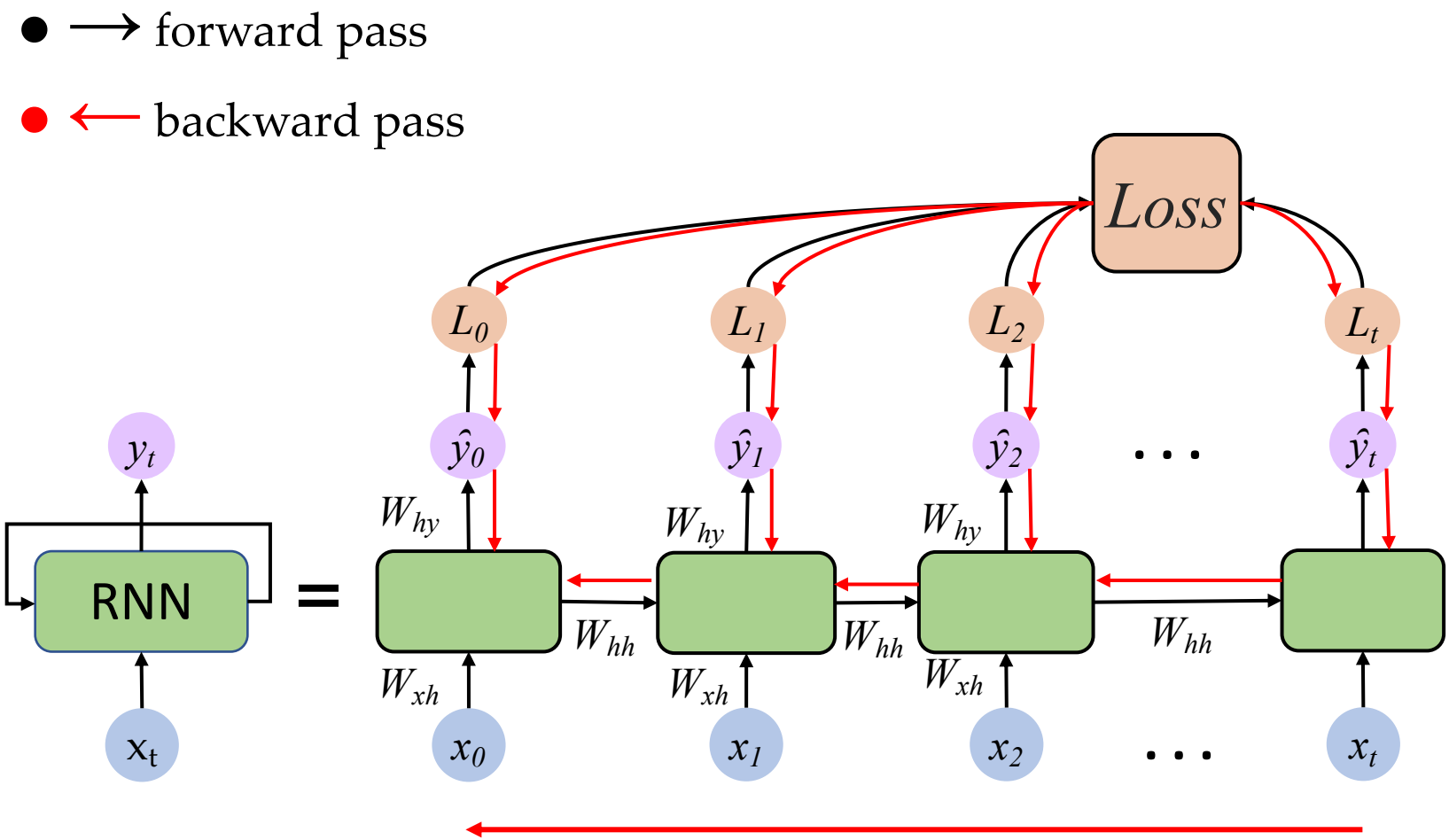
Recall: Backpropagation in Feed Forward NNs



Backpropagation algorithm:

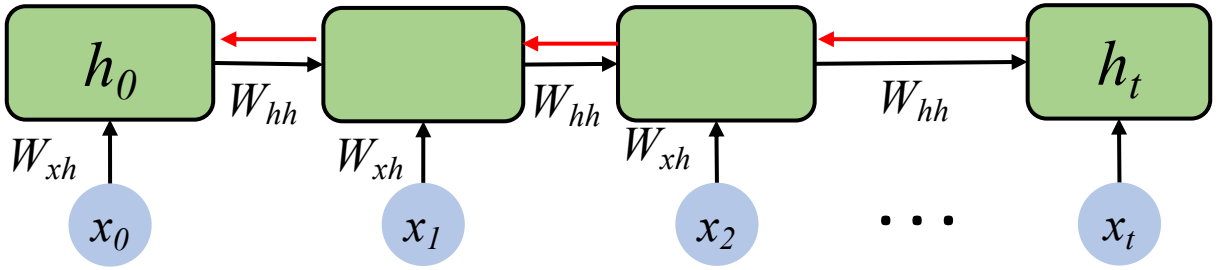
1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

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

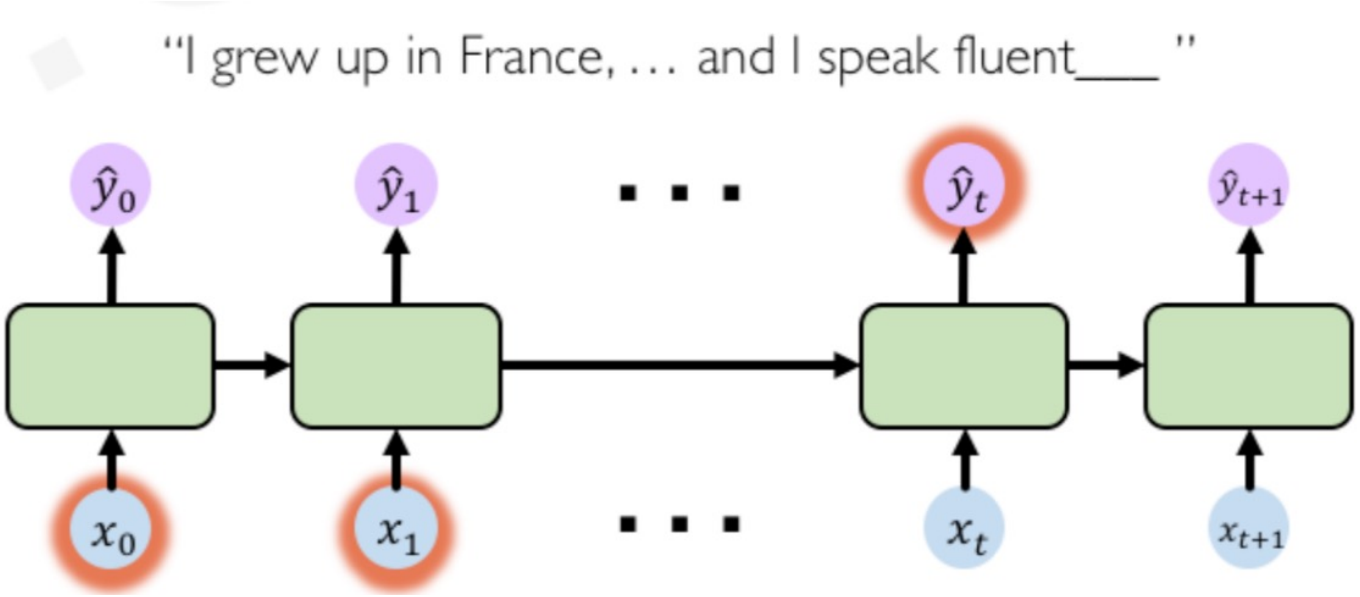
Many values <1:
Vanishing gradients

Gradient clipping to 0 gradients

The Problem of Long-Term Dependencies

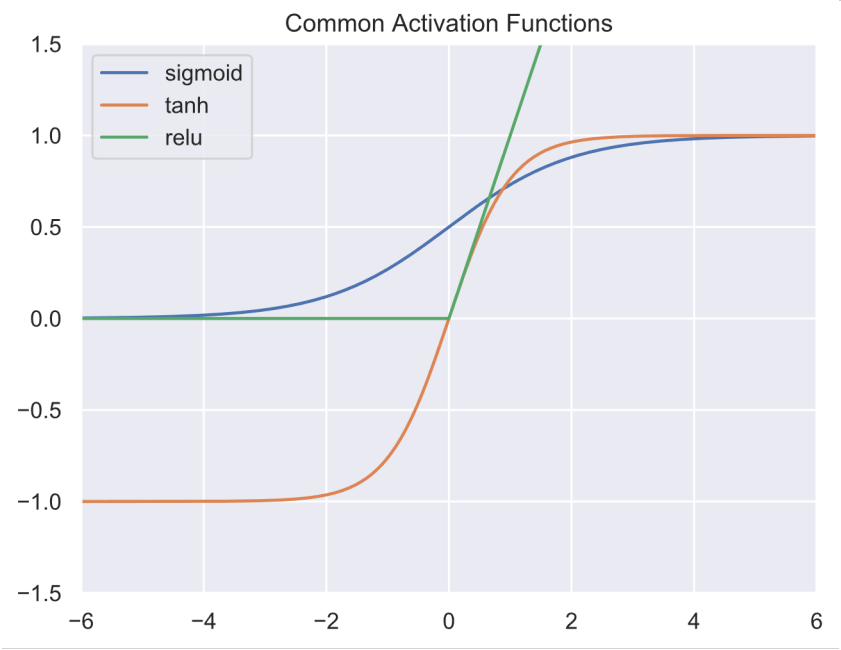
- **Why are vanishing gradients a problem?**

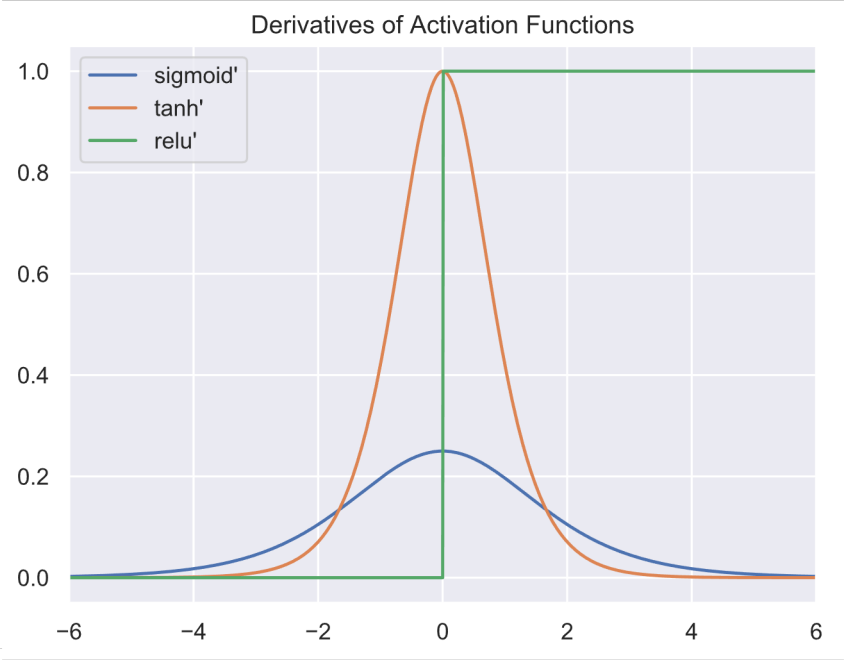
multiply many small numbers together ⇨ errors due to further back time steps have smaller and smaller gradients ⇨ bias parameters to capture short-term dependencies



Trick #1: Better Activation Functions

- **Using activation functions that have larger derivatives**





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

Trick #2: Parameter Initialization

- **Initialize weights that have orthogonal eigen vectors**

Example

- Initialize weights to identify matrix
- Initial biases to zero

$$\boldsymbol{W}_{init} = \begin{pmatrix} 1 & 0 & 0 & & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

This helps prevent the weights from shrinking to zero.

Solution #3: Gated Cells

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

gated cell

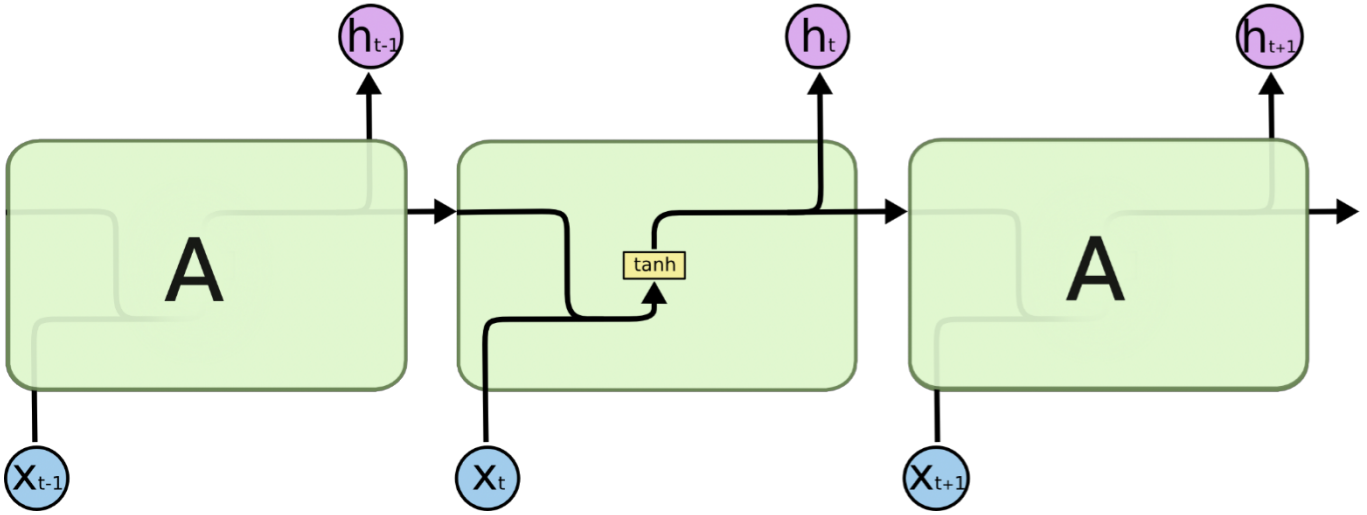
LSTM, GRU, etc.

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

Long Short Term Memory (LSTM) Networks

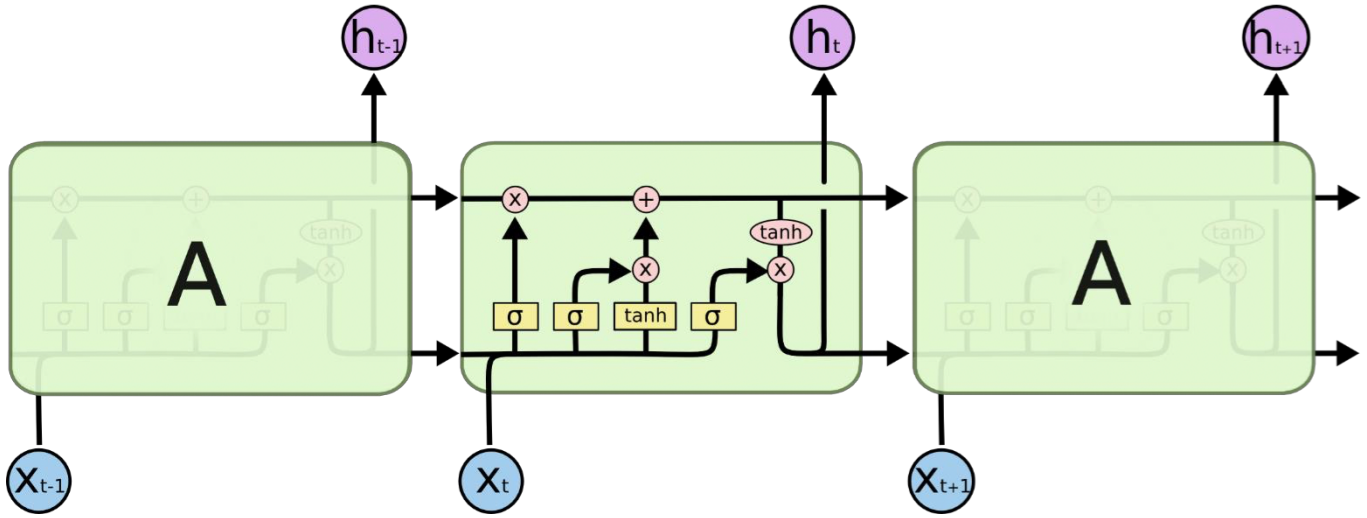
Standard RNN

- In a standard RNN, repeating modules contain a **simple computation node**.



Long Short Term Memory (LSTMs)

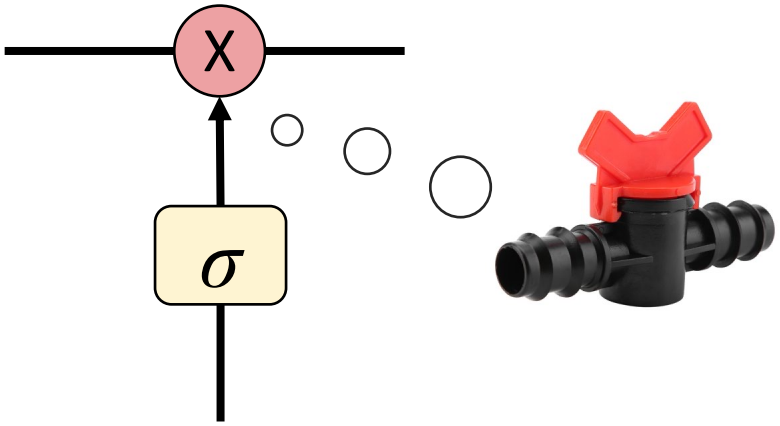
- LSTM modules contain **computational blocks** that **control information flow.**



LSTM cells are able to track information throughout many timesteps.

Long Short Term Memory (LSTMs)

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

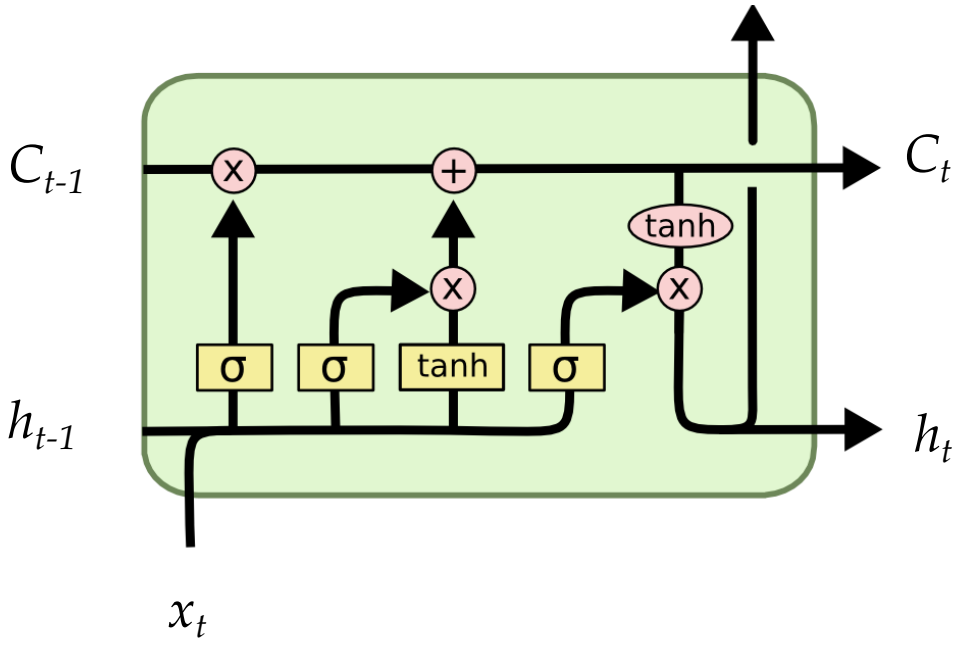


Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

Long Short Term Memory (LSTMs)

- How do LSTMs work?

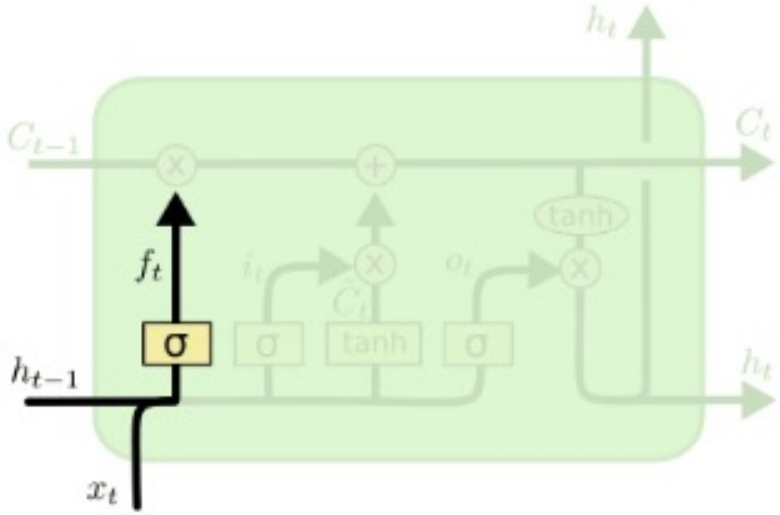
1) Forget 2) Store 3) Update 4) Output



Long Short Term Memory (LSTMs)

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

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

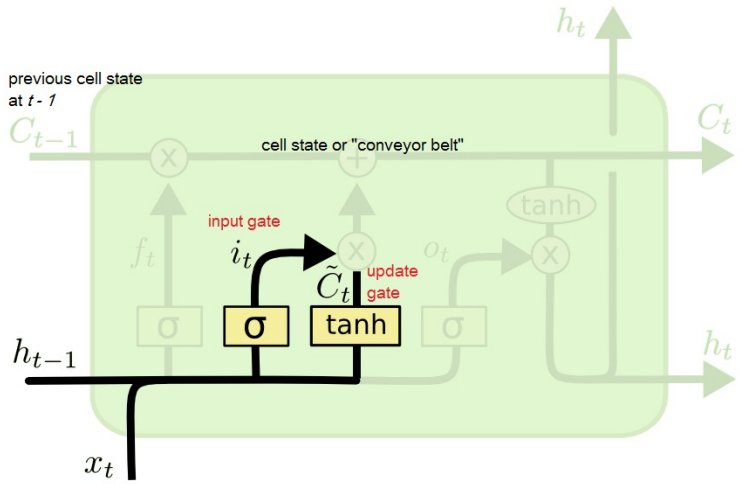


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Long Short Term Memory (LSTMs)

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

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



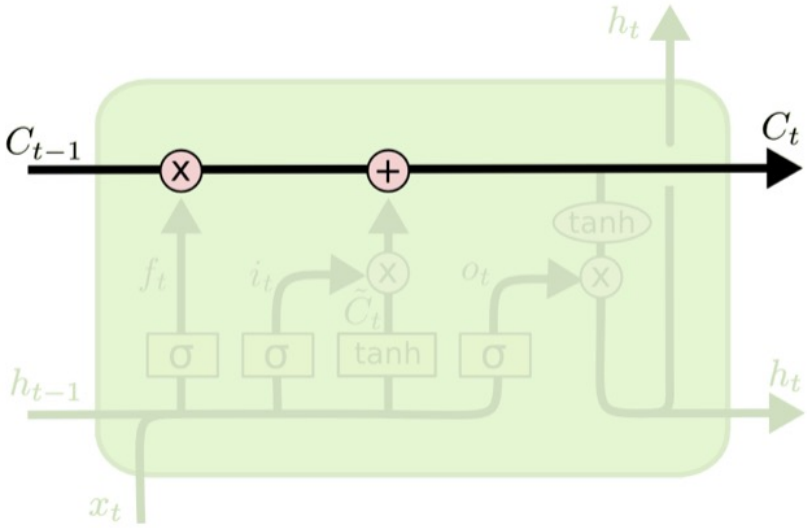
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Long Short Term Memory (LSTMs)

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

- LSTMs **selectively update** cell state values

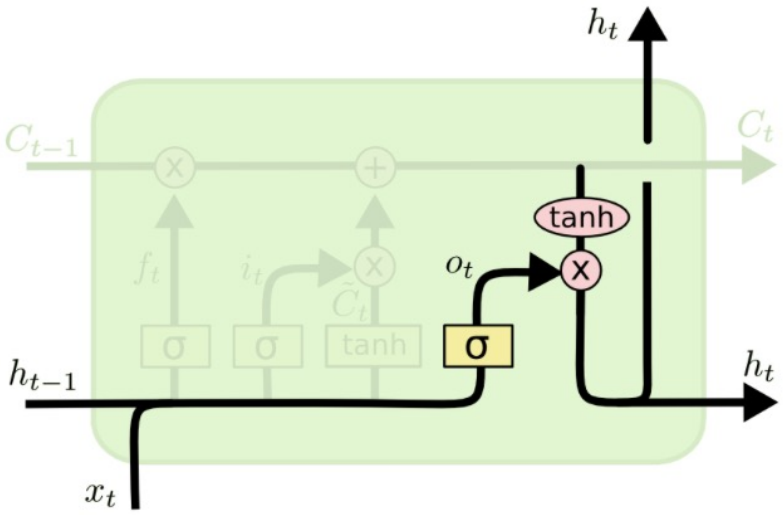


$$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$

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.

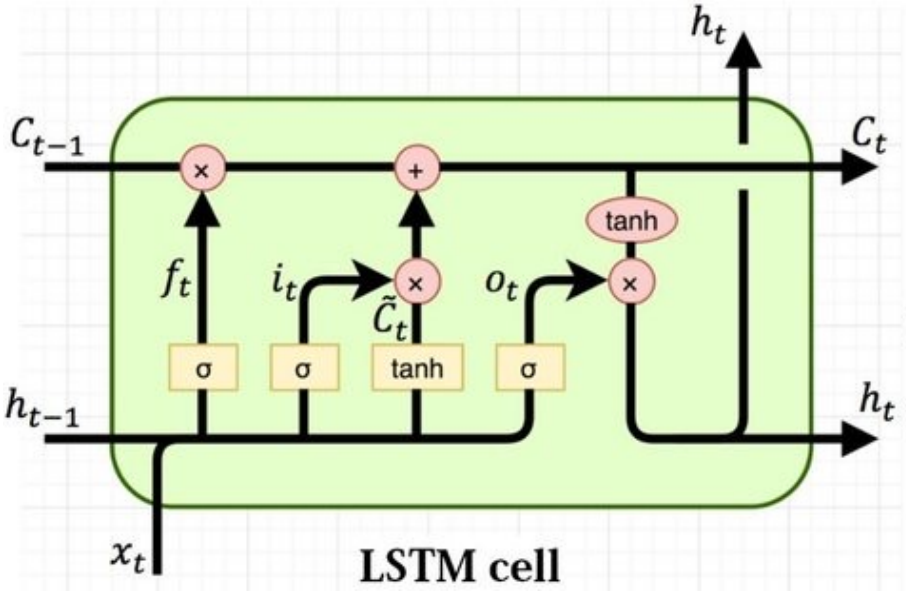


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

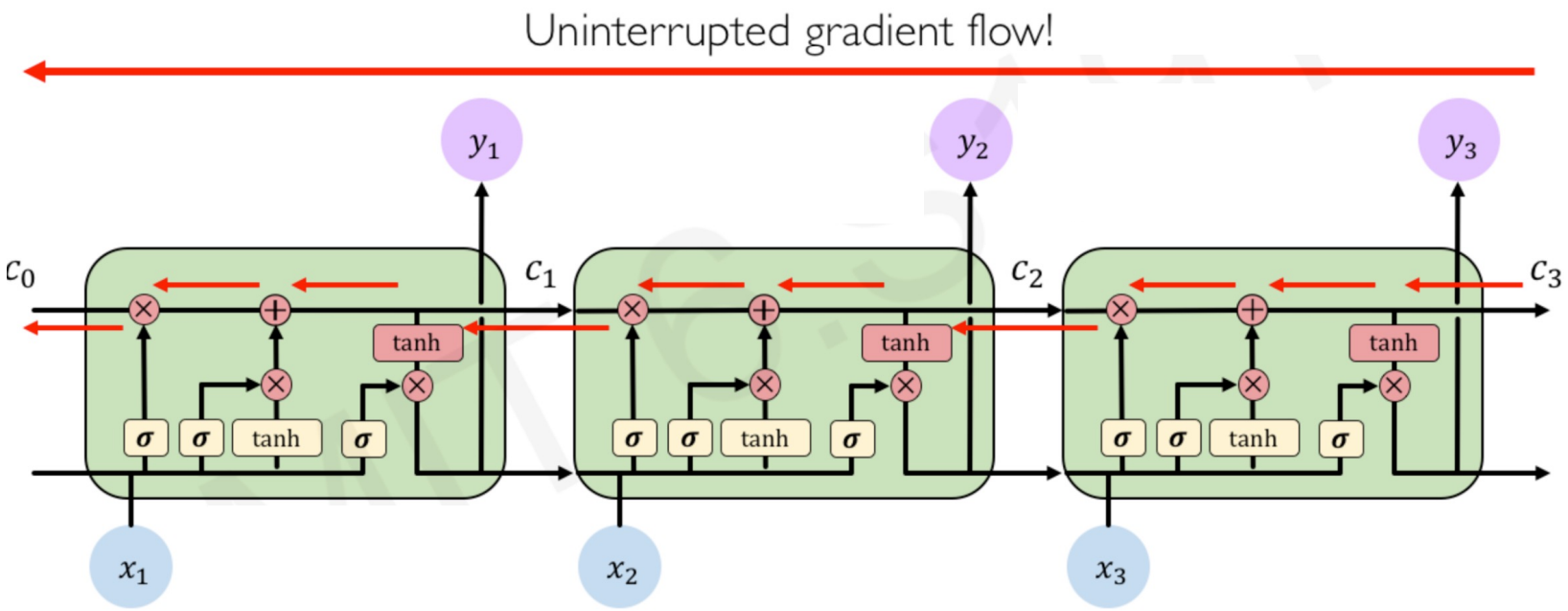
Long Short Term Memory (LSTMs)

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



$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

LSTM Gradient Flow



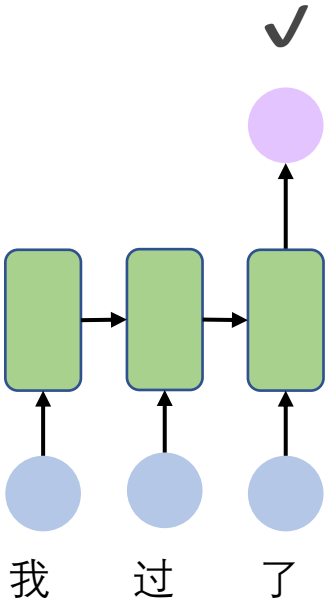
LSTMs: Key Concepts

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**

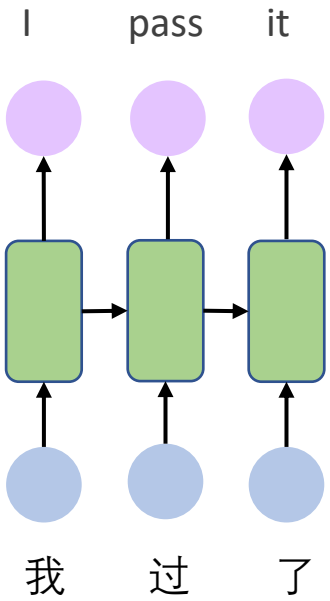
RNN Applications

RNNs for Sequence Modeling

One to One
"Vanilla" neural network



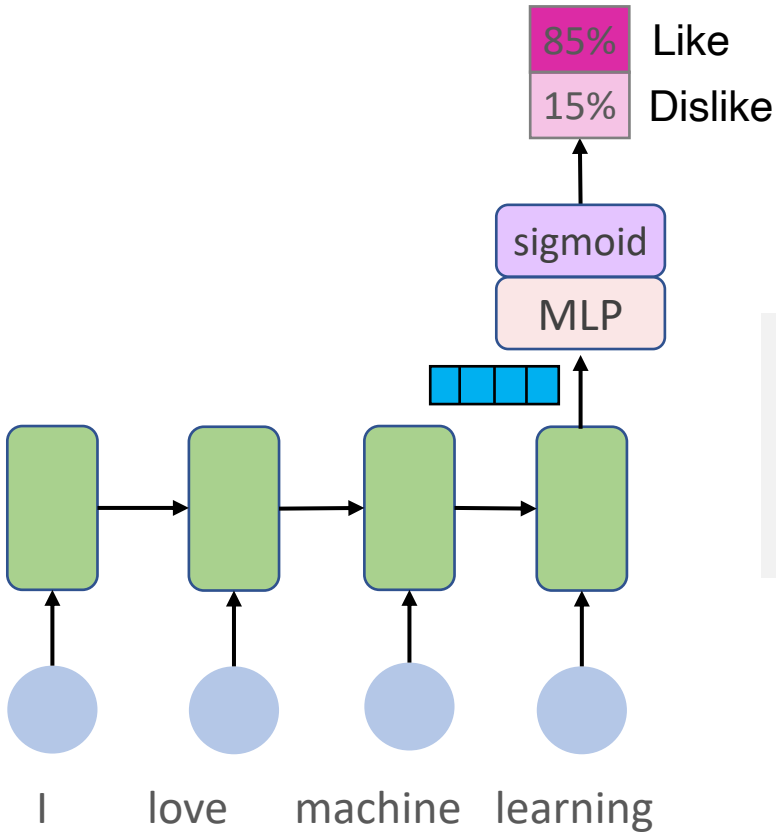
Many to One
e.g., sentiment classification



Many to Many
e.g., machine translation

…and many other architectures and applications

RNN for Sequence Classification

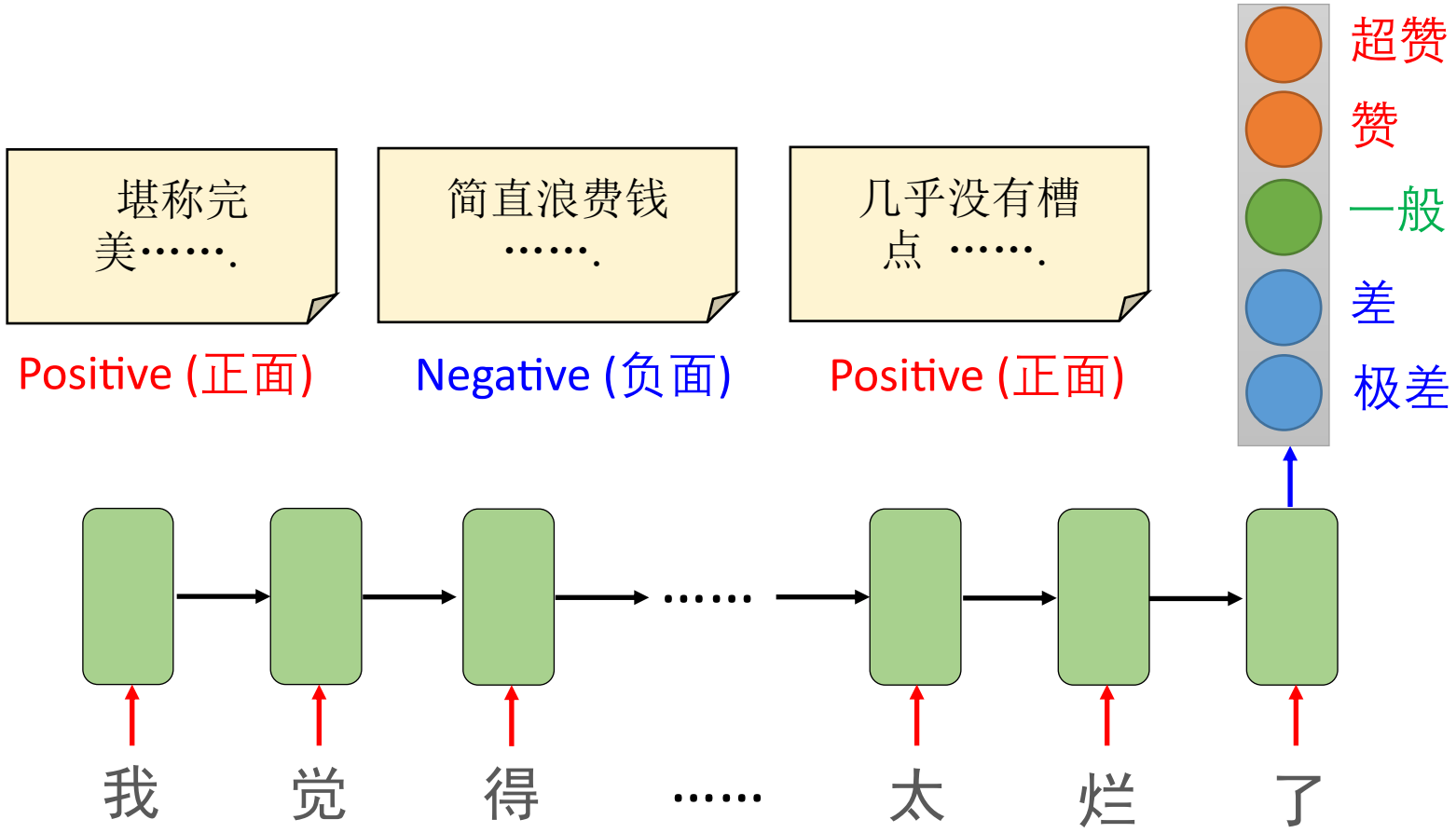


Input: a sequence of words.

Output: probability of each class that the sequence belongs to

Example Task: Sentiment Classification

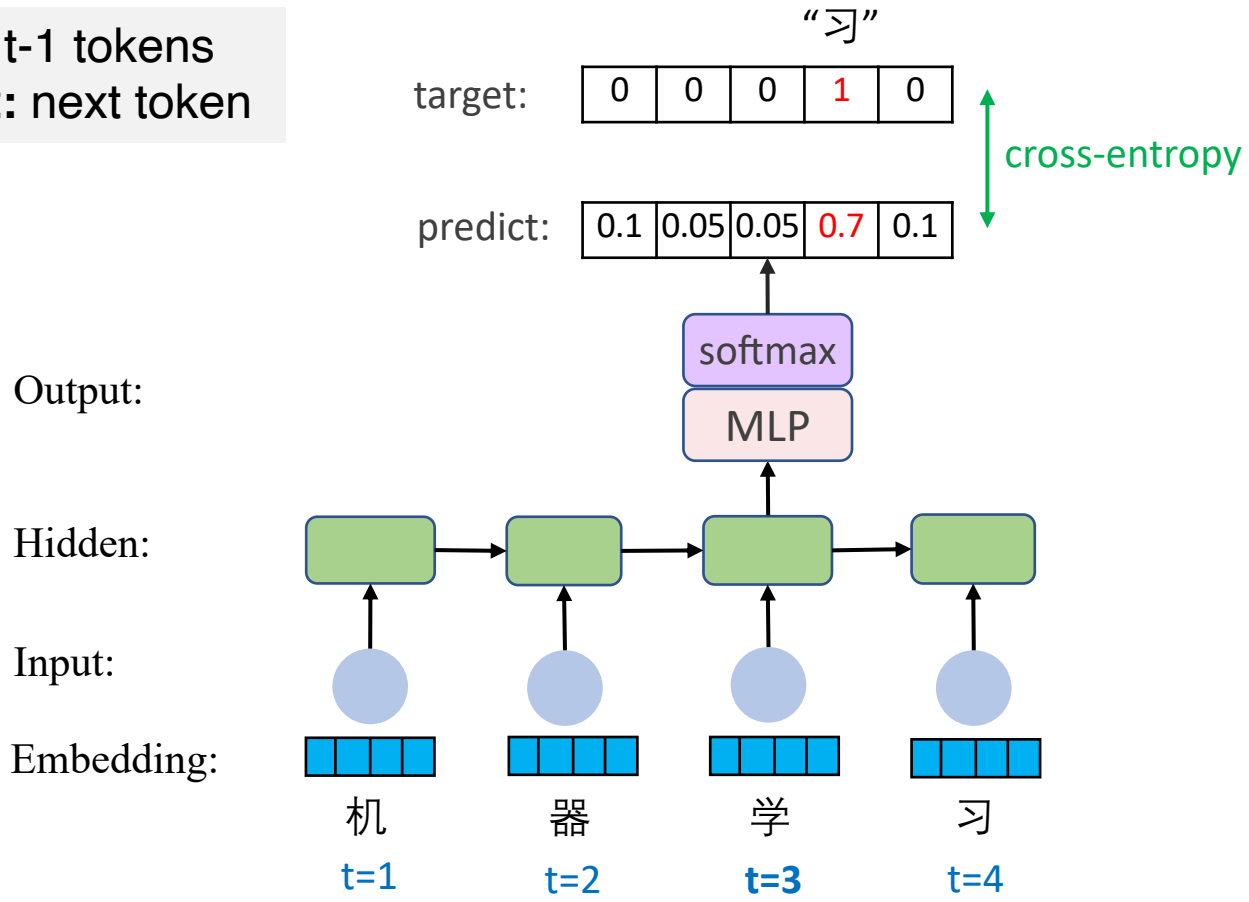
- Input is a vector sequence, but output is only one vector



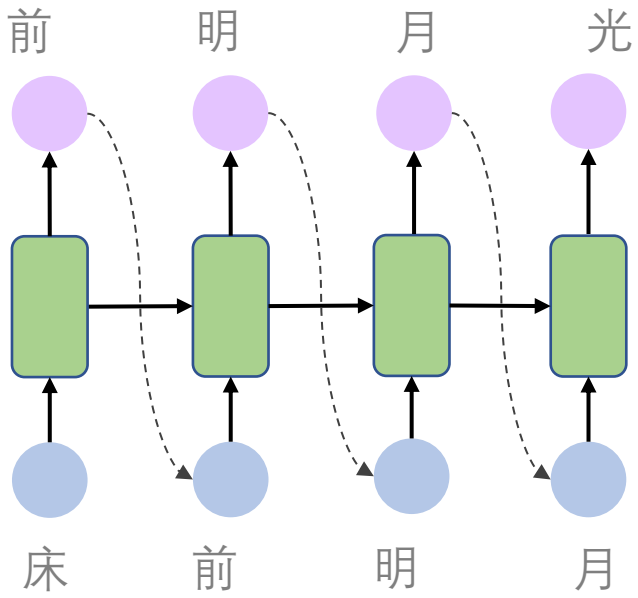
RNN for Sequence Generation

- **RNN can also be used to generate a sequence.**

Input: t-1 tokens
Output: next token



Example Task: Poetry Generation



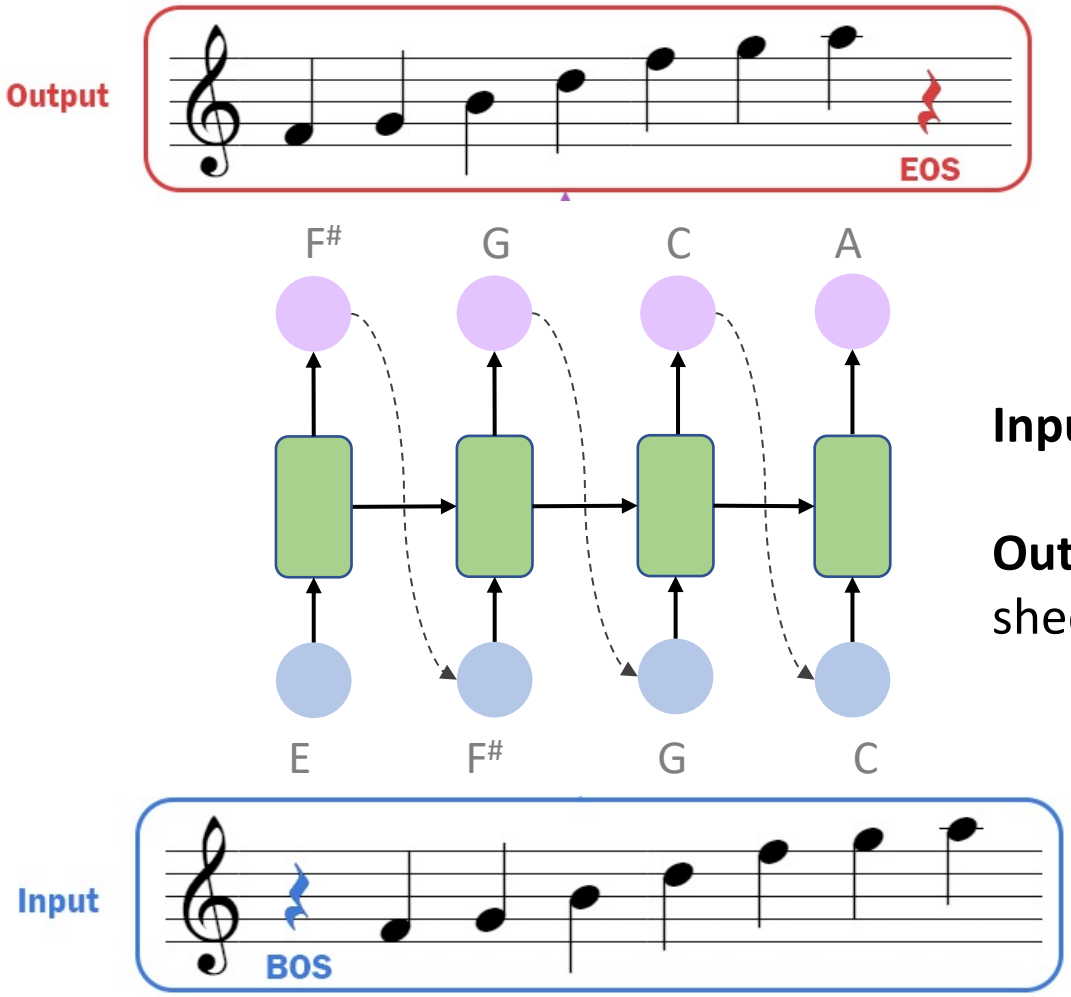
Input: n-1 poetry characters

Output: next character in a poetry

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

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

Example Task: Music Generation

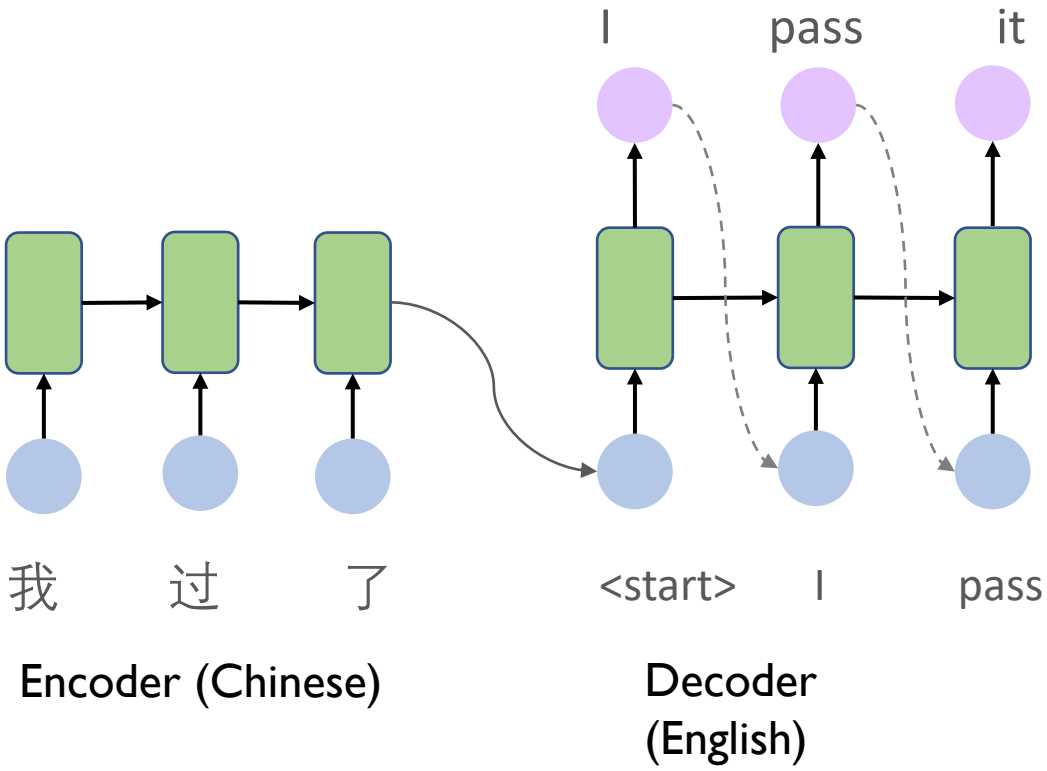


Input: sheet music

Output: the next character in sheet music

Example Task: Machine Translation

- Generate a sequence conditioned on the other sequence.



To be elaborated in the next lecture.