



# Deep Sequence Modeling

Ava Soleimany

MIT 6.S191

January 27, 2020



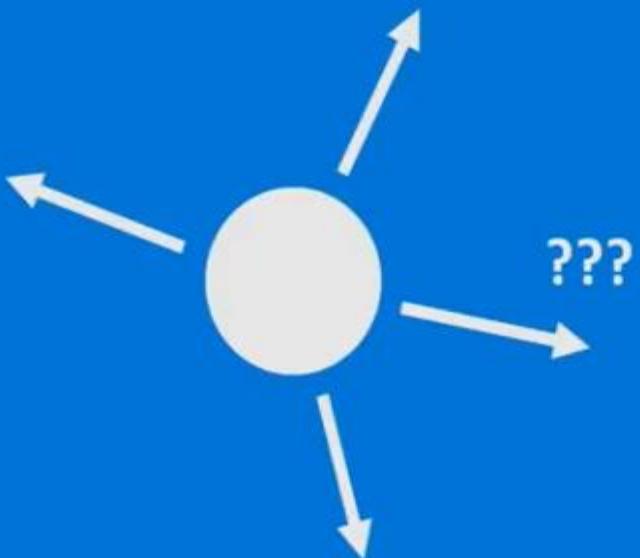
6.S191 Introduction to Deep Learning  
[introtodeeplearning.com](http://introtodeeplearning.com)   [@MITDeepLearning](https://twitter.com/MITDeepLearning)



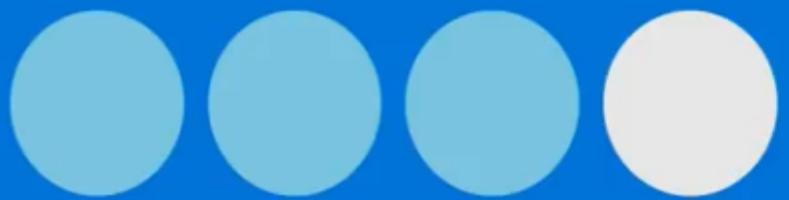
Given an image of a ball,  
can you predict where it will go next?



Given an image of a ball,  
can you predict where it will go next?



Given an image of a ball,  
can you predict where it will go next?



Given an image of a ball,  
can you predict where it will go next?



# Sequences in the Wild



Audio



# Sequences in the Wild



Audio



# Sequences in the Wild

6.S191 Introduction to Deep Learning



Text

# Sequences in the Wild

character:

6 . S | 9 |

word: Introduction to Deep Learning

Text



# A Sequence Modeling Problem: Predict the Next Word



# A Sequence Modeling Problem: Predict the Next Word

“This morning I took my cat for a walk.”



# A Sequence Modeling Problem: Predict the Next Word

“This morning I took my cat for a walk.”

given these words



# A Sequence Modeling Problem: Predict the Next Word

“This morning I took my cat for a walk.”

given these words

predict the  
next word



# Idea #1: Use a Fixed Window

“This morning I took my cat for a walk.”

given these      predict the  
two words      next word



# Idea #1: Use a Fixed Window

“This morning I took my cat for a walk.”

given these      predict the  
two words      next word

One-hot feature encoding: tells us what each word is

[ 1 0 0 0 0 0 1 0 0 0 ]

for                  a



prediction



# Problem #1: Can't Model Long-Term Dependencies

"France is where I grew up, but I now live in Boston. I speak fluent \_\_\_\_."



We need information from **the distant past** to accurately predict the correct word.



## Idea #2: Use Entire Sequence as Set of Counts

“This morning I took my cat for a”



“bag of words”

[ 0 1 0 0 1 0 0 ... 0 0 1 1 0 0 0 1 ]



prediction



## Problem #2: Counts Don't Preserve Order



The food was good, not bad at all.

vs.

The food was bad, not good at all.



# Idea #3: Use a Really Big Fixed Window

“This morning I took my cat for a walk.”

given these  
words      predict the  
next word

[ 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 ... ]  
morning      I      took      this      cat



## Problem #3: No Parameter Sharing

[1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 ...]  
this morning took the cat

Each of these inputs has a **separate parameter**:

[0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 ...]  
this morning

Things we learn about the sequence **won't transfer** if  
they appear **elsewhere** in the sequence.



# Sequence Modeling: Design Criteria

To model sequences, we need to:

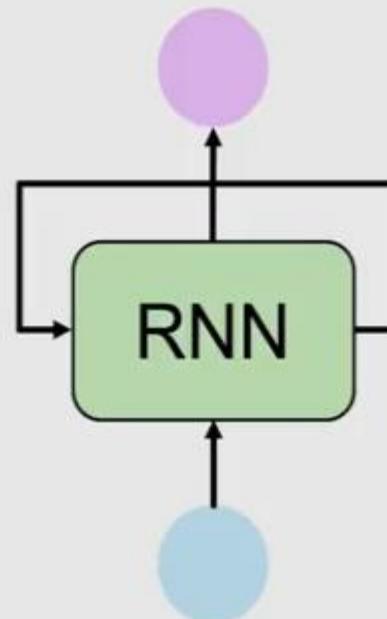
1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



# 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



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

# Recurrent Neural Networks (RNNs)



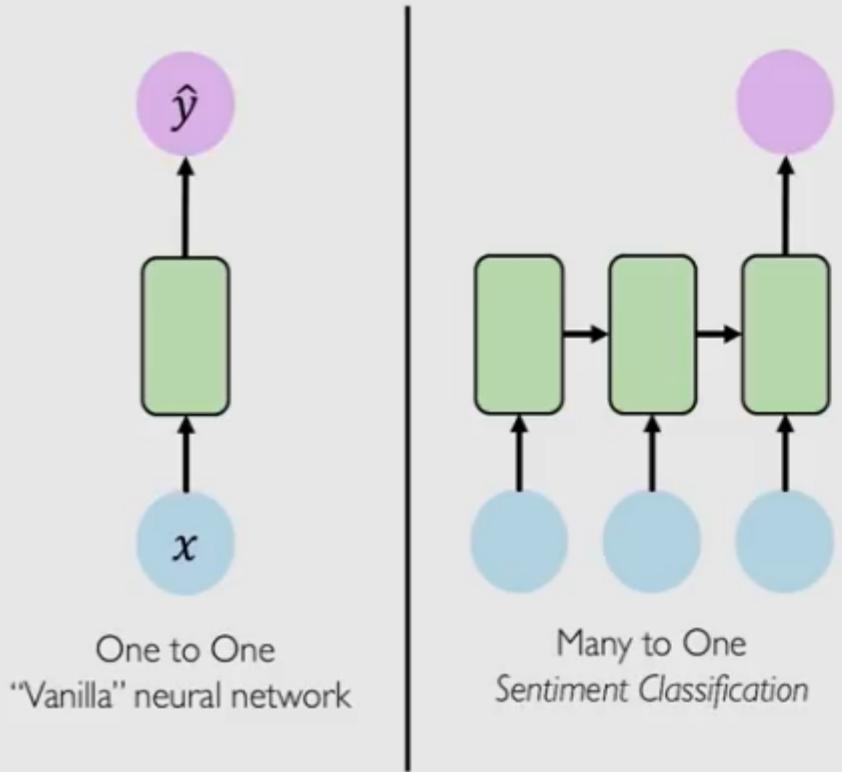
# Standard Feed-Forward Neural Network



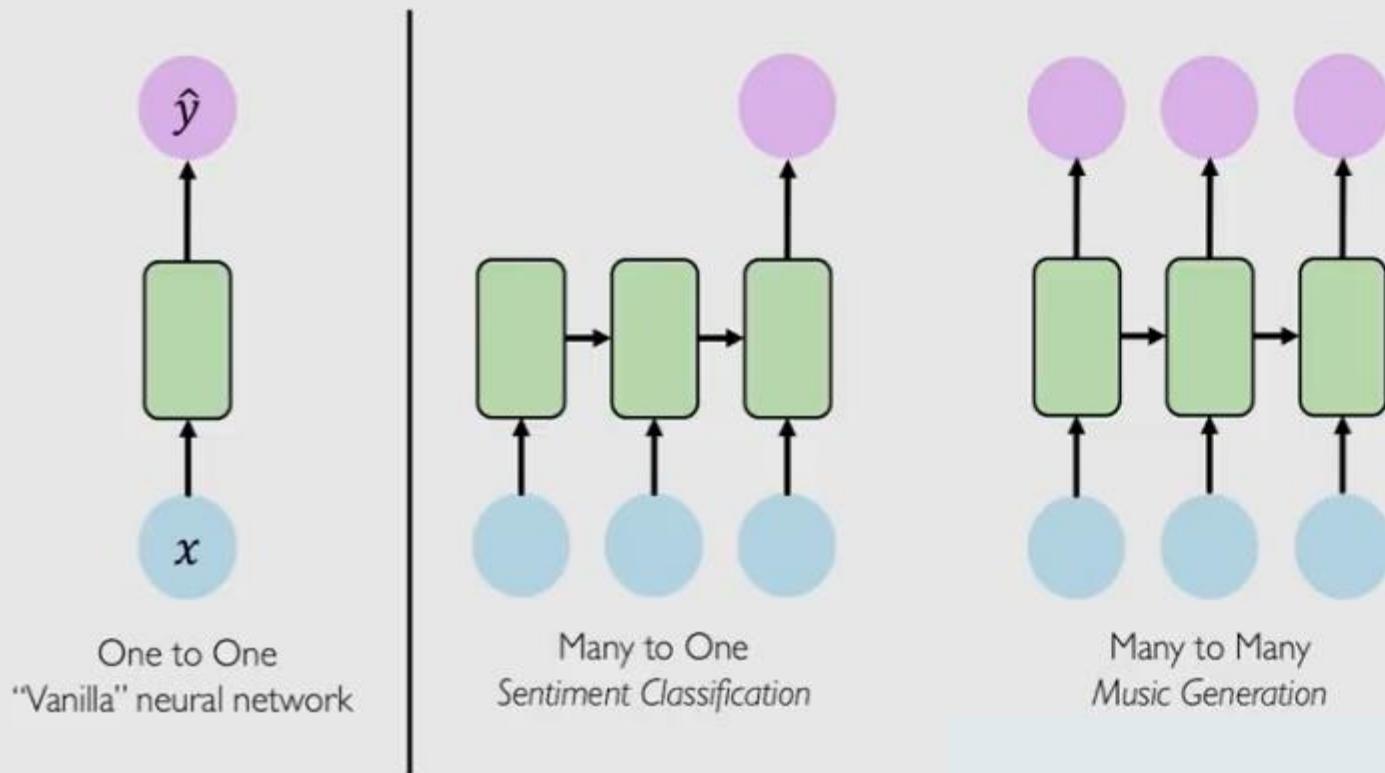
One to One  
“Vanilla” neural network



# Recurrent Neural Networks for Sequence Modeling

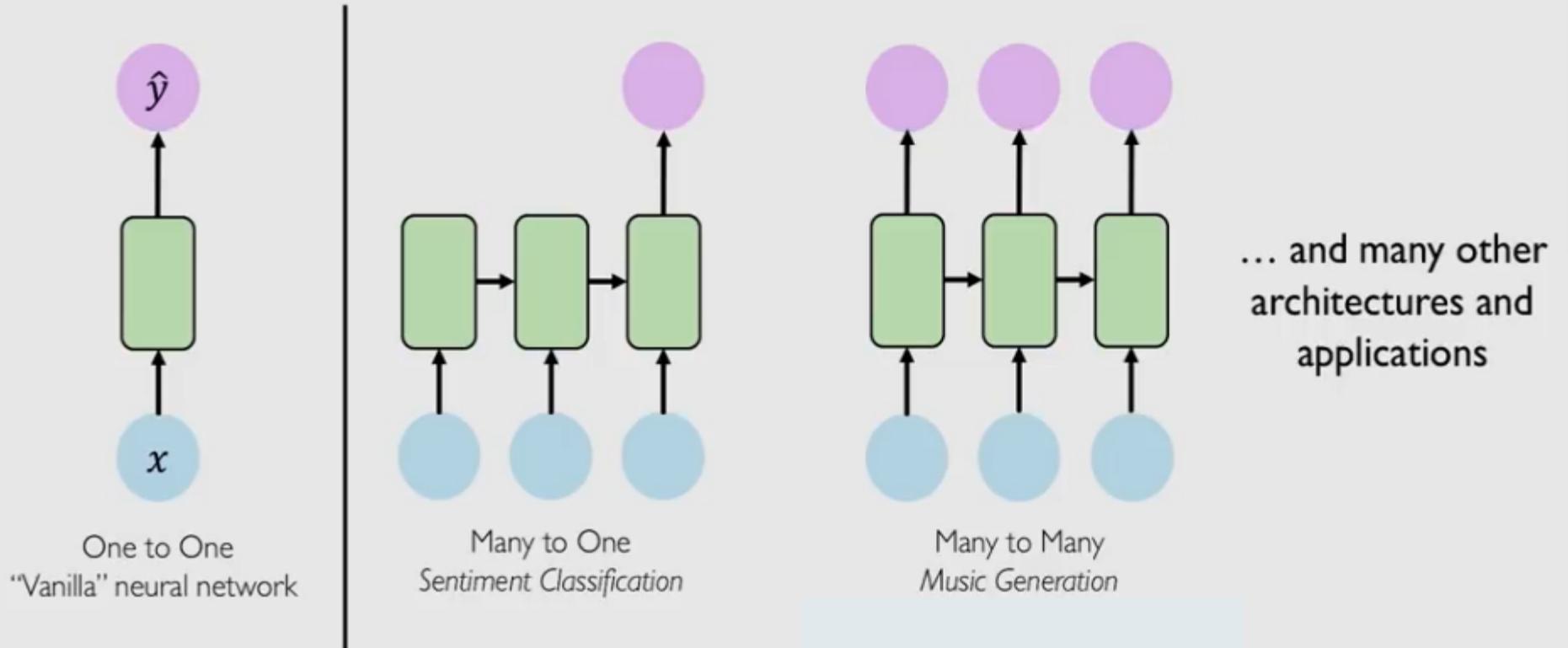


# Recurrent Neural Networks for Sequence Modeling



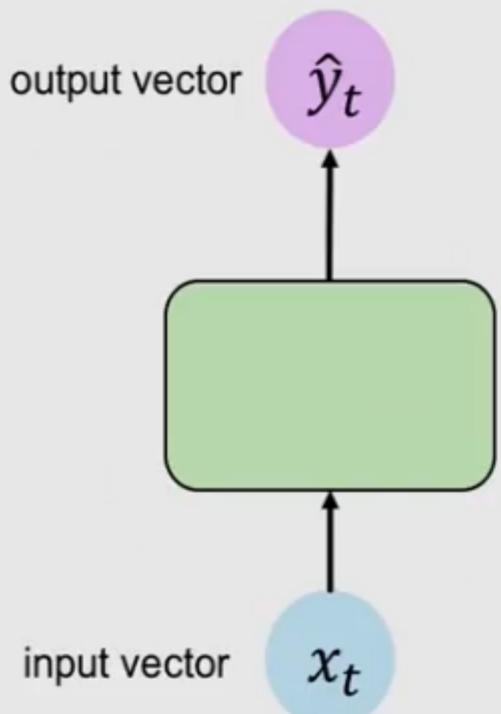
MIT Deep Learning

# Recurrent Neural Networks for Sequence Modeling



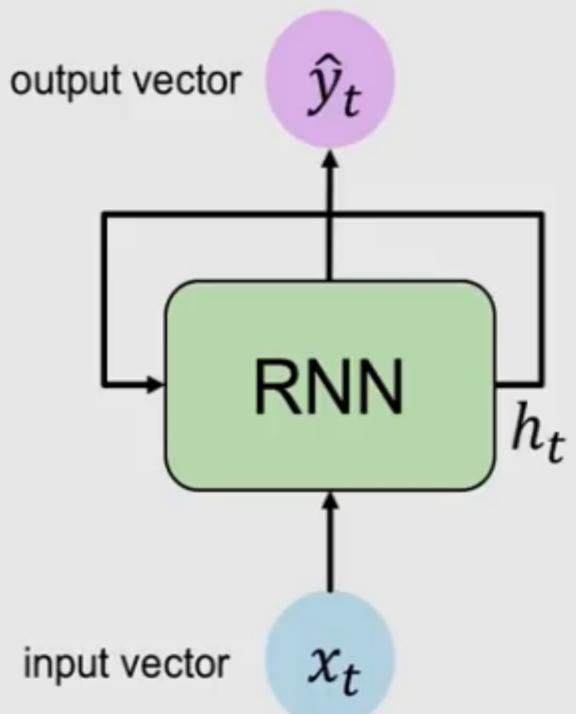
MIT Deep Learning

# Standard “Vanilla” Neural Network

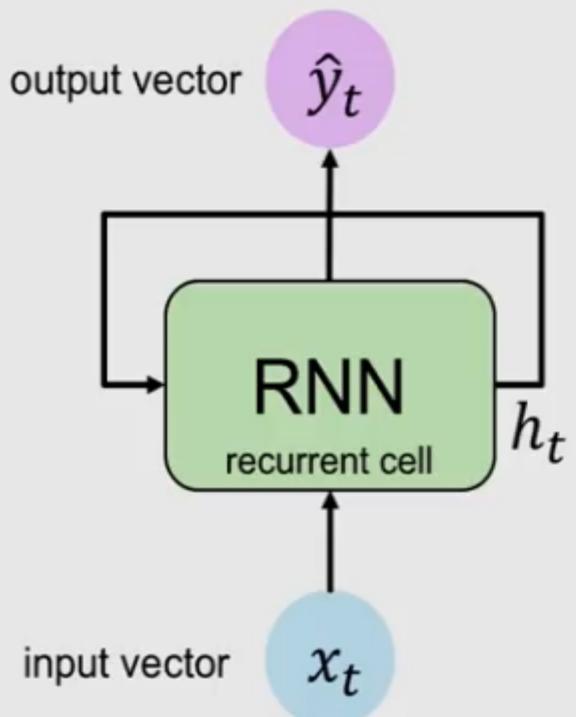


MIT Deep  
Learning

# Recurrent Neural Network (RNN)



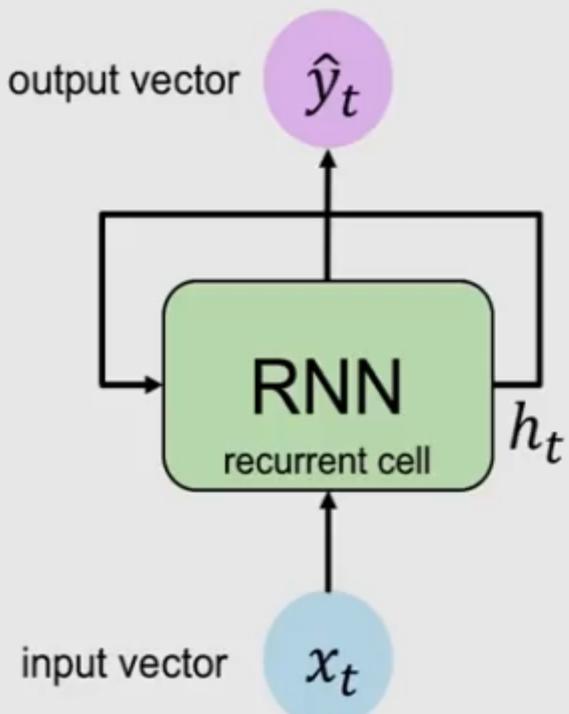
# Recurrent Neural Network (RNN)



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



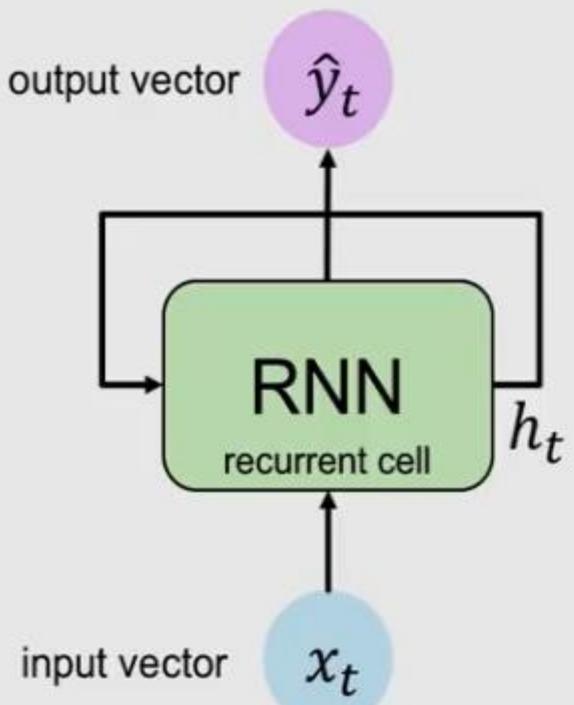
# Recurrent Neural Network (RNN)



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

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

# Recurrent Neural Network (RNN)



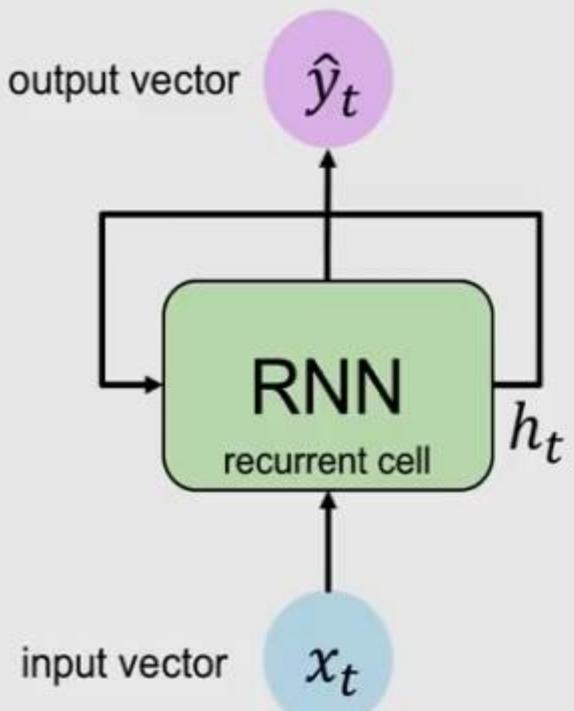
Apply a **recurrence relation** at every time step to process a sequence:

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

cell state      function  
                  parameterized  
                  by W



# Recurrent Neural Network (RNN)



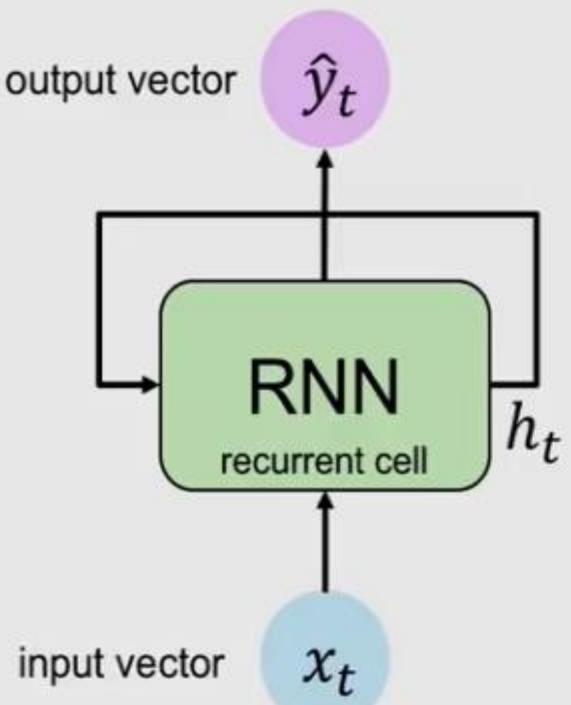
Apply a **recurrence relation** at every time step to process a sequence:

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

cell state      function  
                  parameterized  
                  by W



# Recurrent Neural Network (RNN)



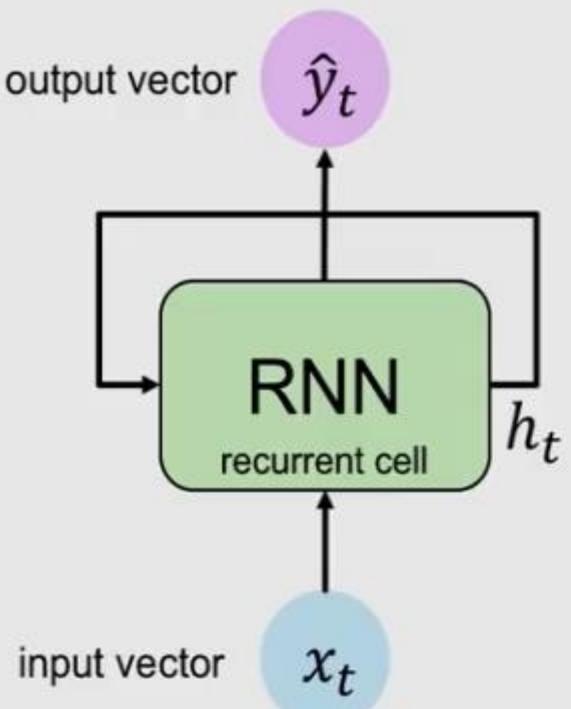
Apply a **recurrence relation** at every time step to process a sequence:

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

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



# Recurrent Neural Network (RNN)



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

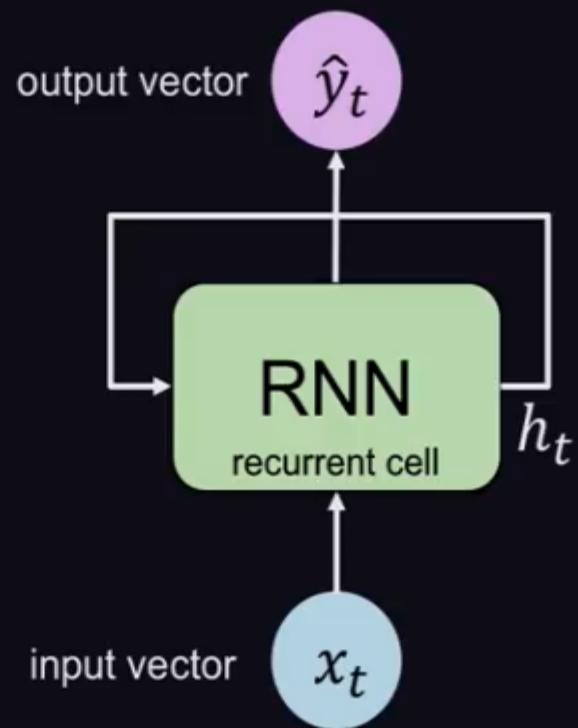
$$h_t = f_W(h_{t-1}, x_t)$$

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

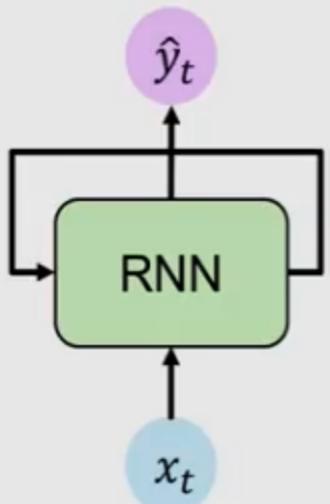
Note: the same function and set of parameters are used at every time step

# RNN Intuition

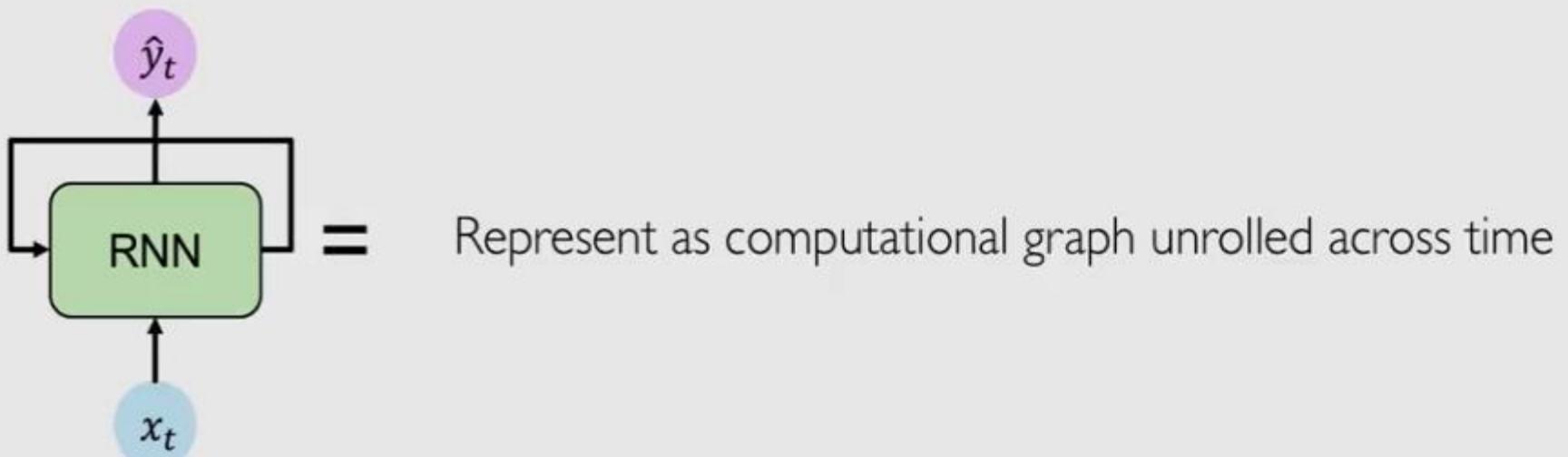
```
my_rnn = RNN()  
hidden_state = [0, 0, 0, 0]  
  
sentence = ["I", "love", "recurrent", "neural"]  
  
for word in sentence:  
    prediction, hidden_state = my_rnn(word, hidden_state)  
  
next_word_prediction = prediction  
# >>> "networks!"
```



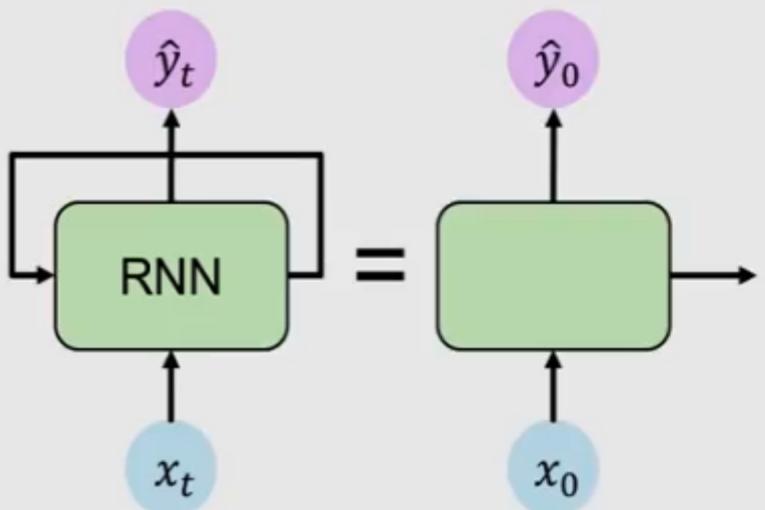
# RNNs: Computational Graph Across Time



# RNNs: Computational Graph Across Time

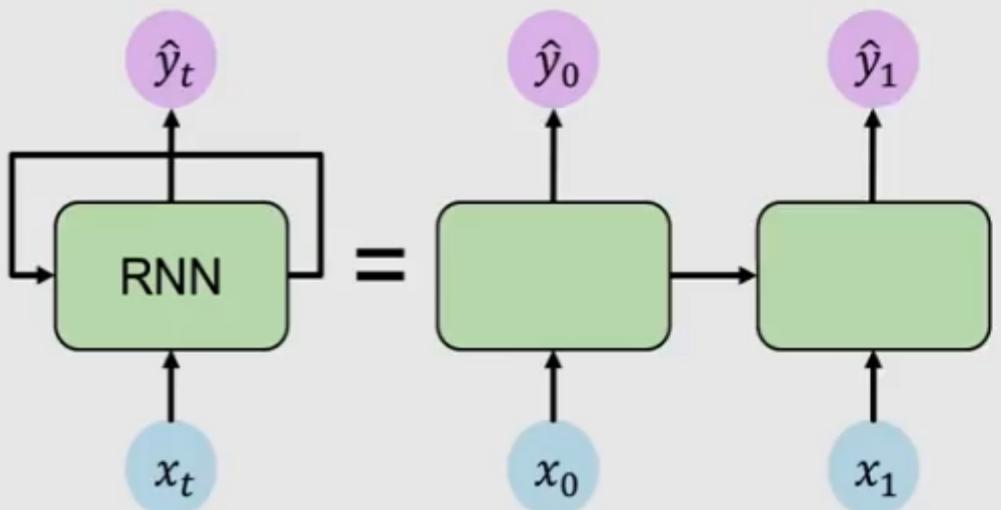


# RNNs: Computational Graph Across Time



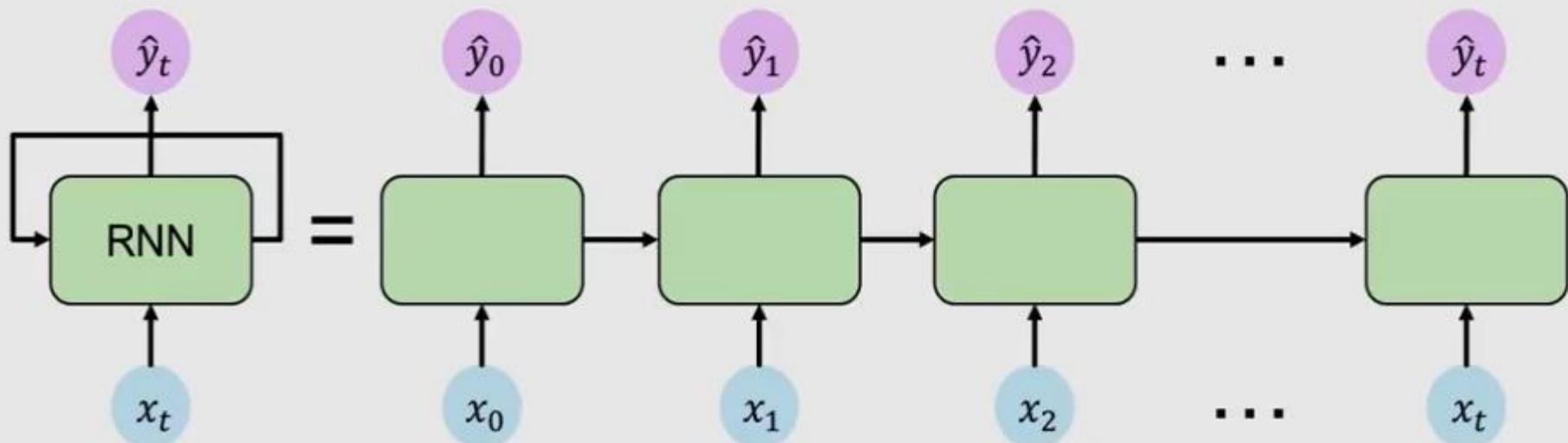
MIT Deep Learning

# RNNs: Computational Graph Across Time



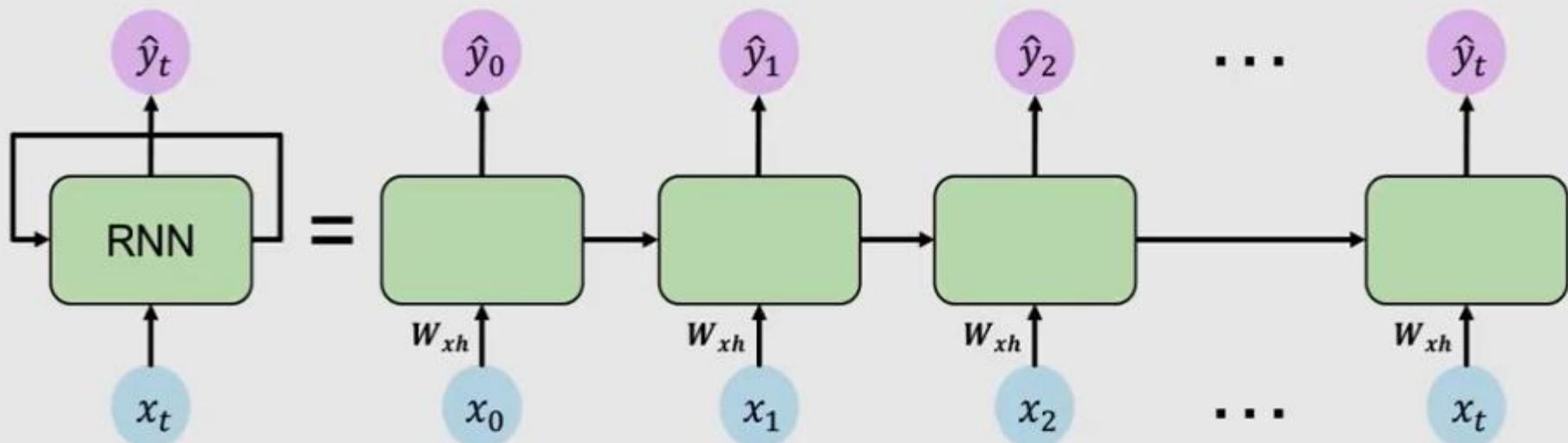
MIT Deep Learning

# RNNs: Computational Graph Across Time



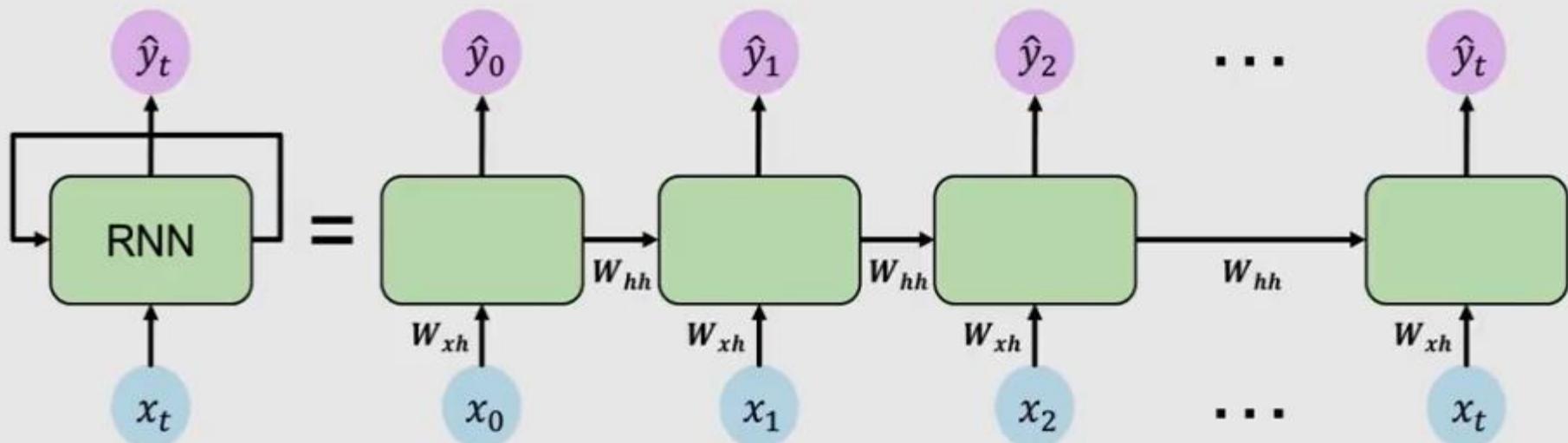
MIT Deep Learning

# RNNs: Computational Graph Across Time



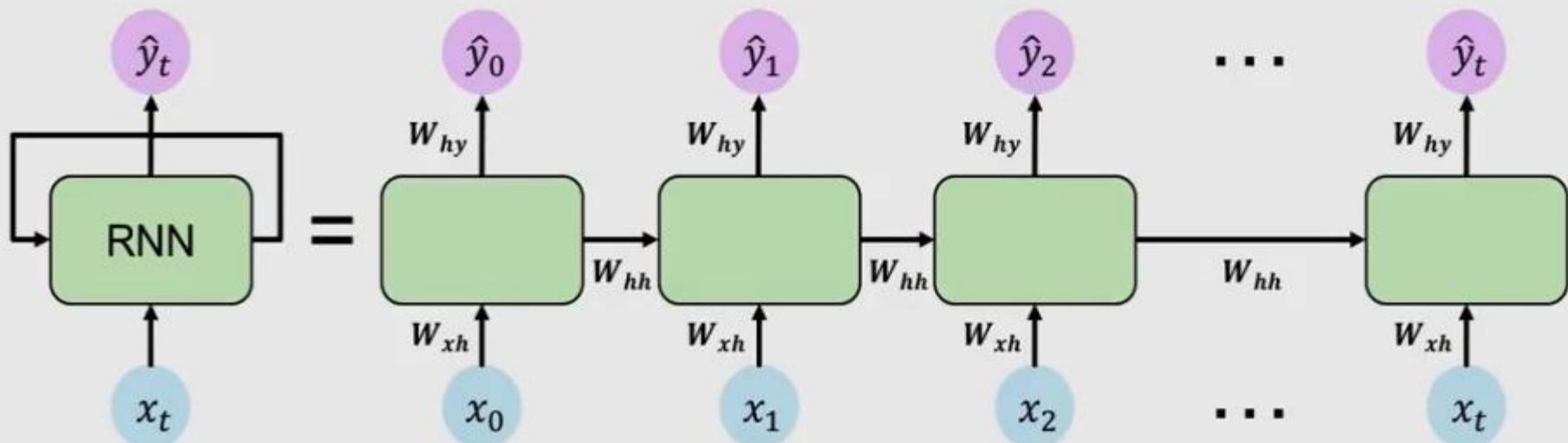
MIT Deep Learning

# RNNs: Computational Graph Across Time



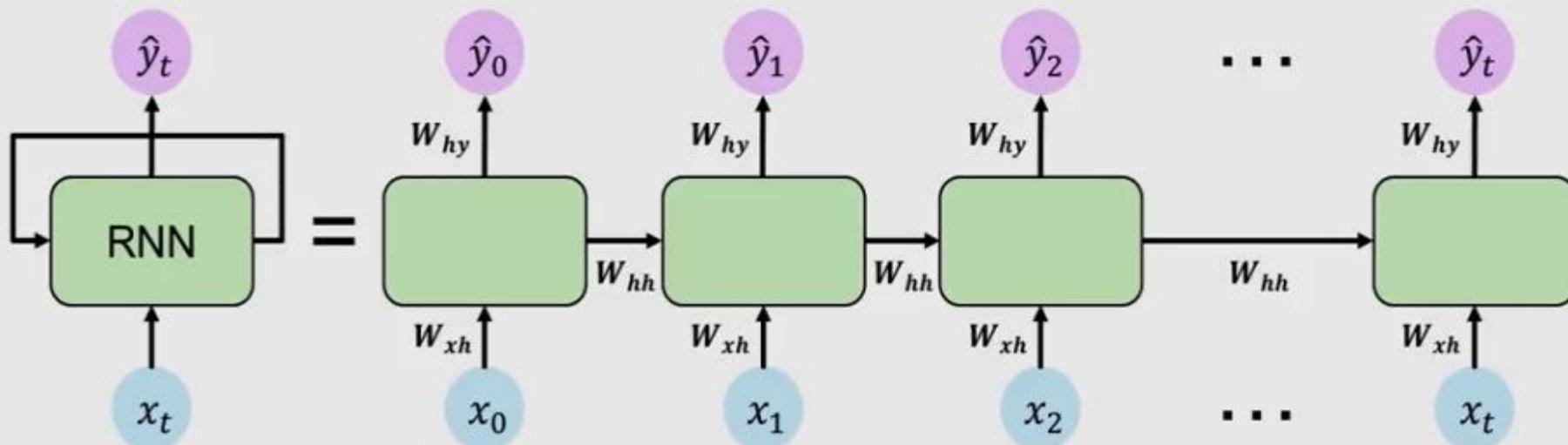
MIT Deep Learning

# RNNs: Computational Graph Across Time



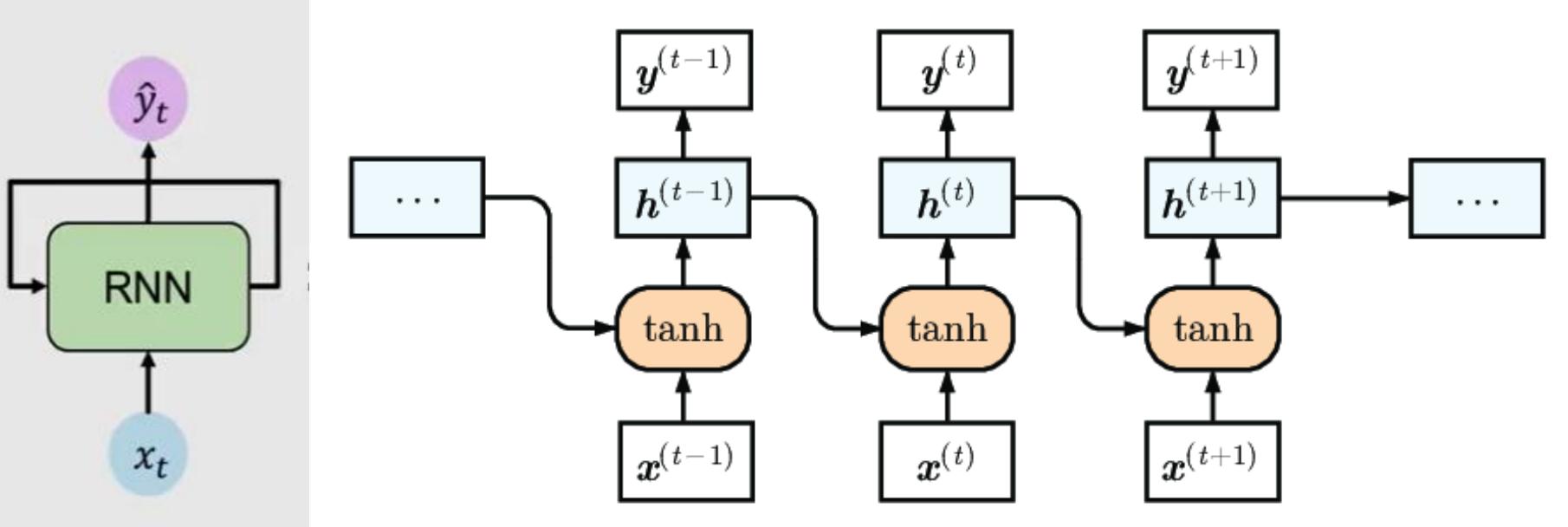
# RNNs: Computational Graph Across Time

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

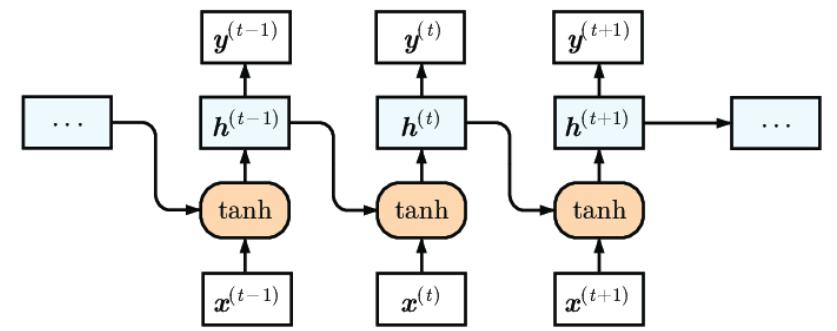
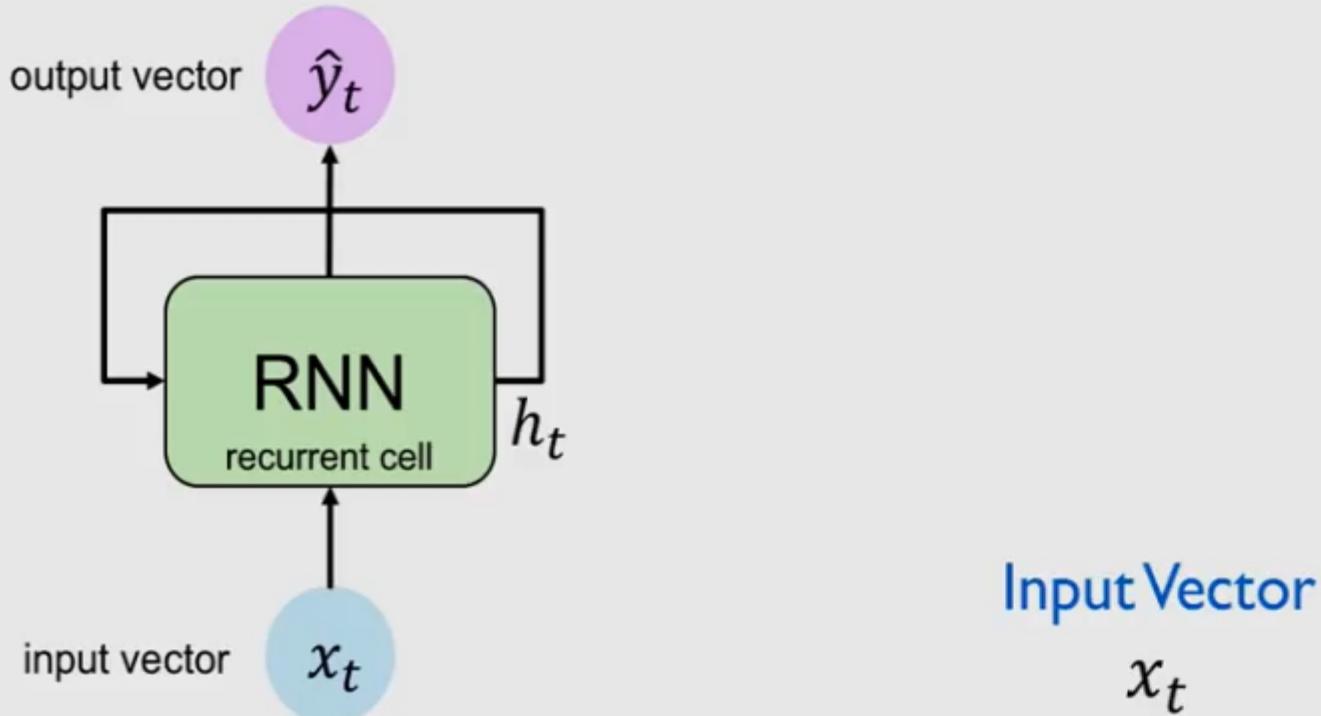


# RNNs: Computational Graph Across Time

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

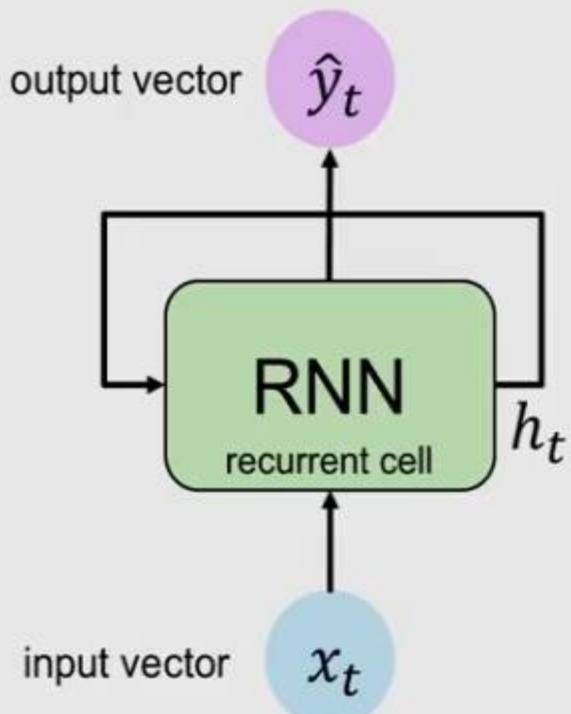
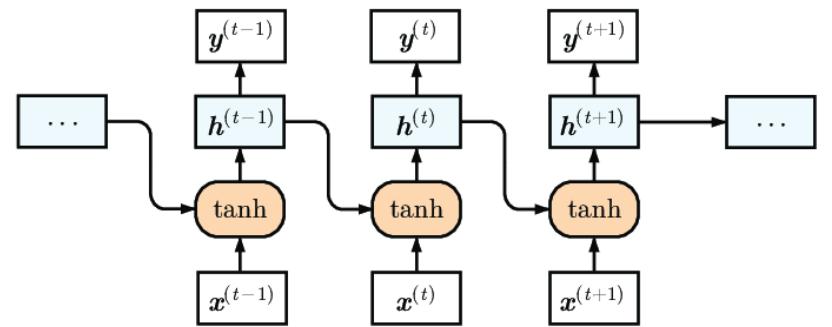


# RNN State Update and Output



MIT Deep Learning

# RNN State Update and Output



Update Hidden State

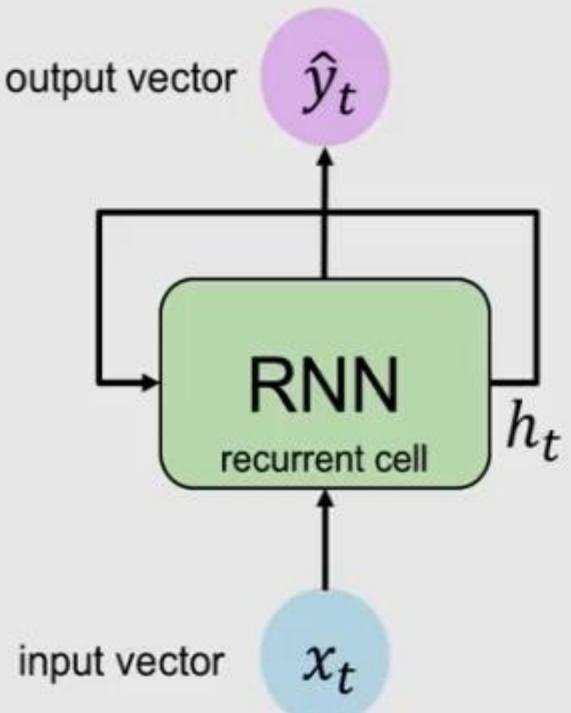
$$h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t)$$

Input Vector

$x_t$

MIT Deep Learning

# RNN State Update and Output



Output Vector

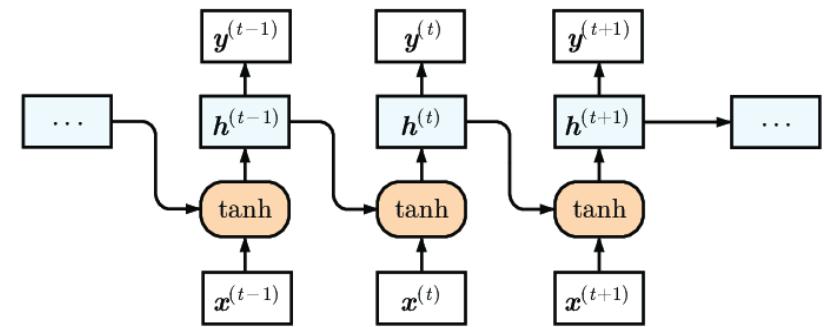
$$\hat{y}_t = W_{hy}^T h_t$$

Update Hidden State

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

Input Vector

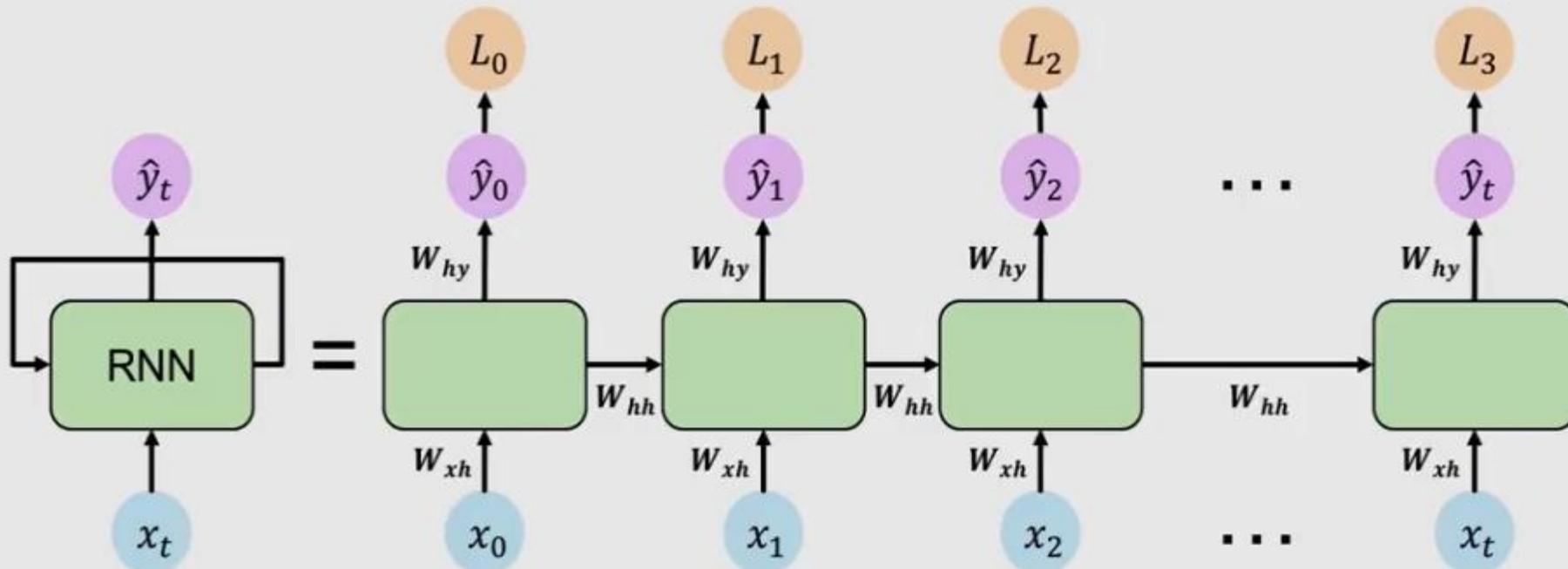
$$x_t$$



MIT Deep Learning

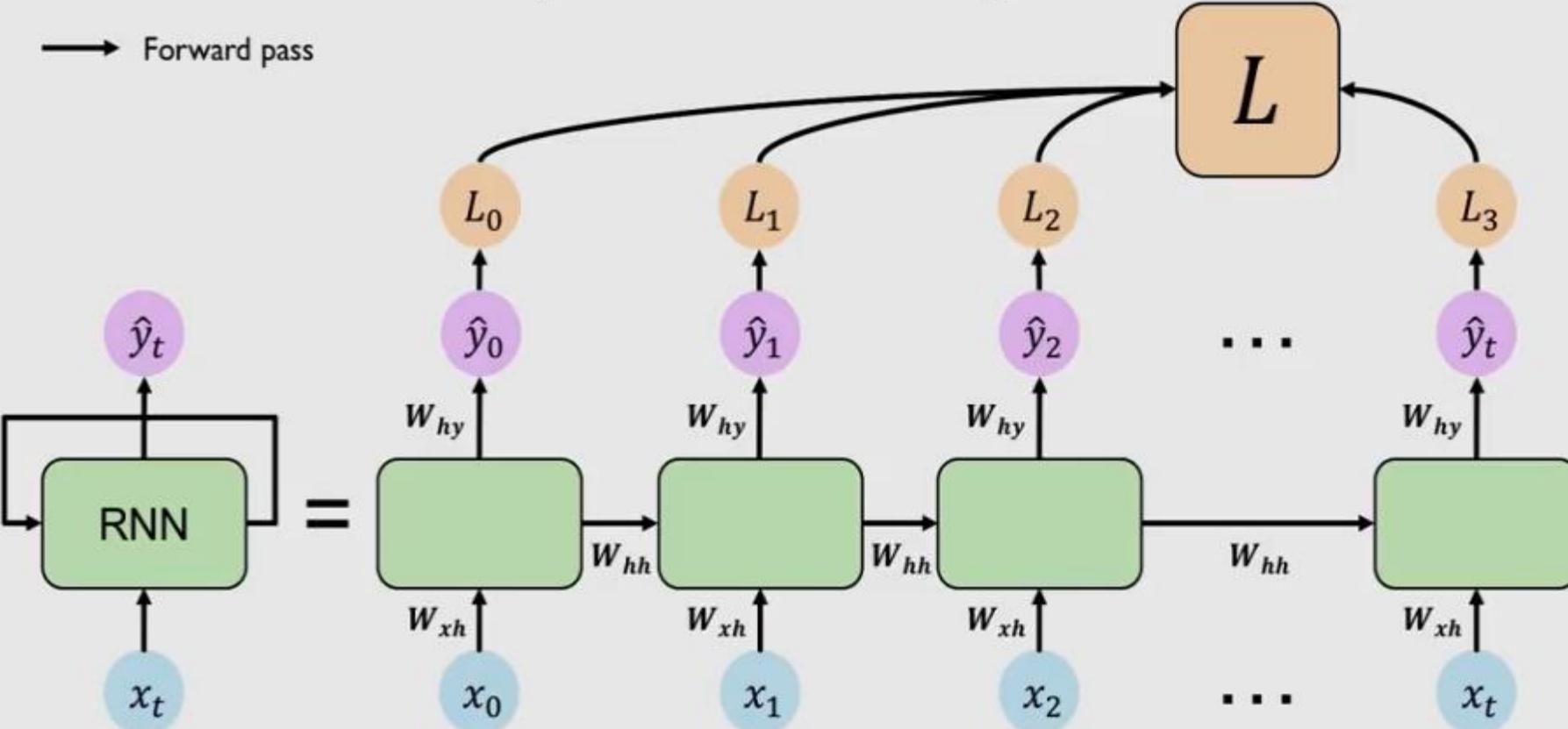
# RNNs: Computational Graph Across Time

→ Forward pass



MIT Deep Learning

# RNNs: Computational Graph Across Time



MIT Deep Learning

# RNNs from Scratch



```
class MyRNNCell(tf.keras.layers.Layer):
    def __init__(self, rnn_units, input_dim, output_dim):
        super(MyRNNCell, self).__init__()

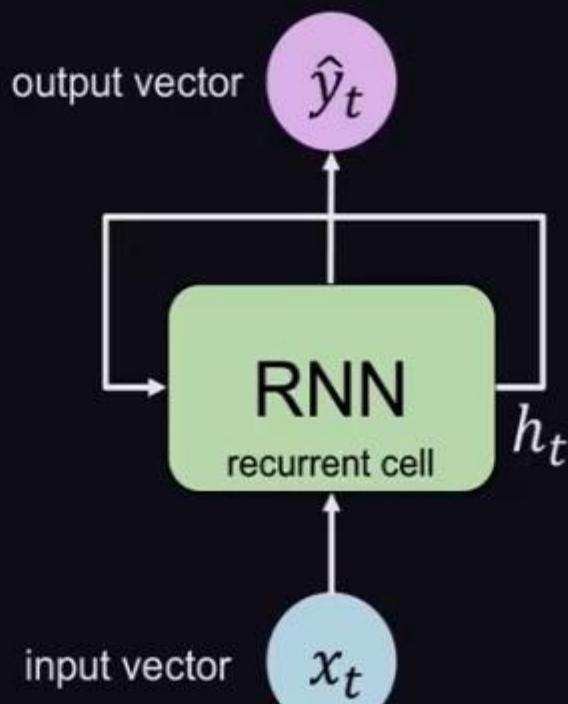
        # Initialize weight matrices
        self.W_xh = self.add_weight([rnn_units, input_dim])
        self.W_hh = self.add_weight([rnn_units, rnn_units])
        self.W_hy = self.add_weight([output_dim, rnn_units])

        # Initialize hidden state to zeros
        self.h = tf.zeros([rnn_units, 1])

    def call(self, x):
        # Update the hidden state
        self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

        # Compute the output
        output = self.W_hy * self.h

        # Return the current output and hidden state
        return output, self.h
```



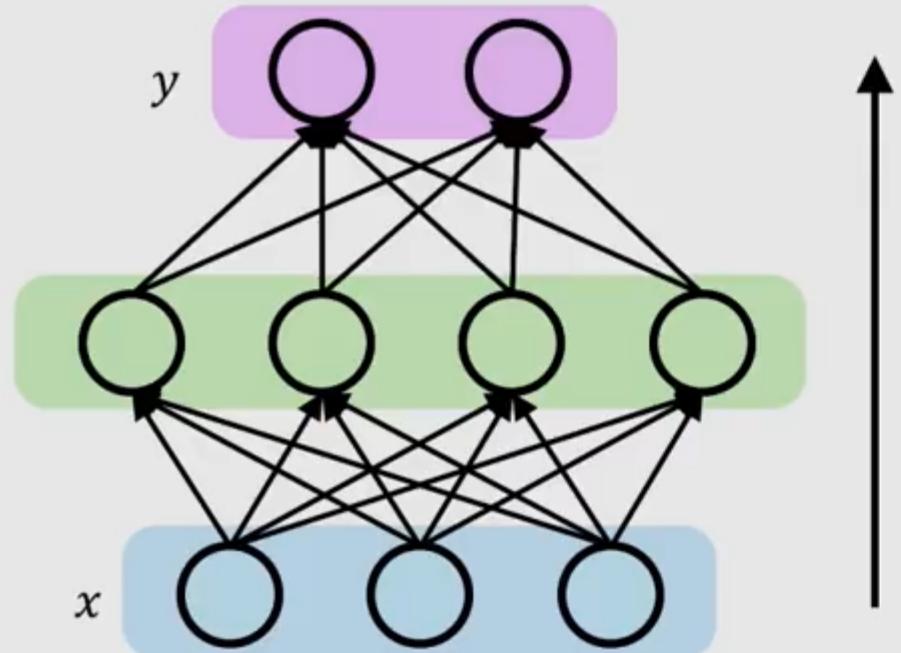
# Backpropagation Through Time (BPTT)



# Recall: Backpropagation in Feed Forward Models

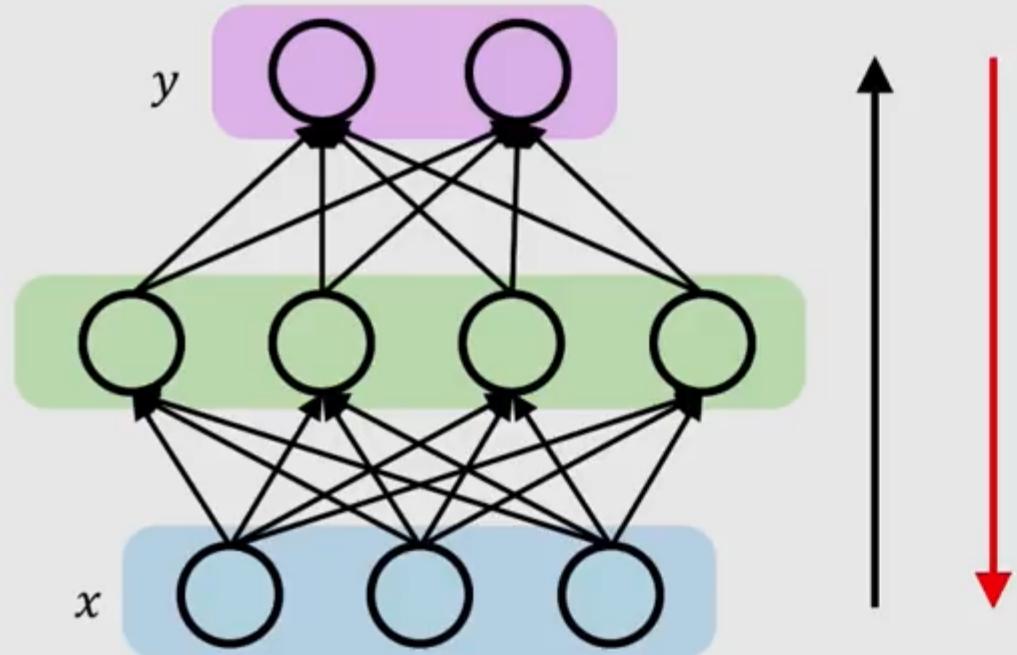


# Recall: Backpropagation in Feed Forward Models

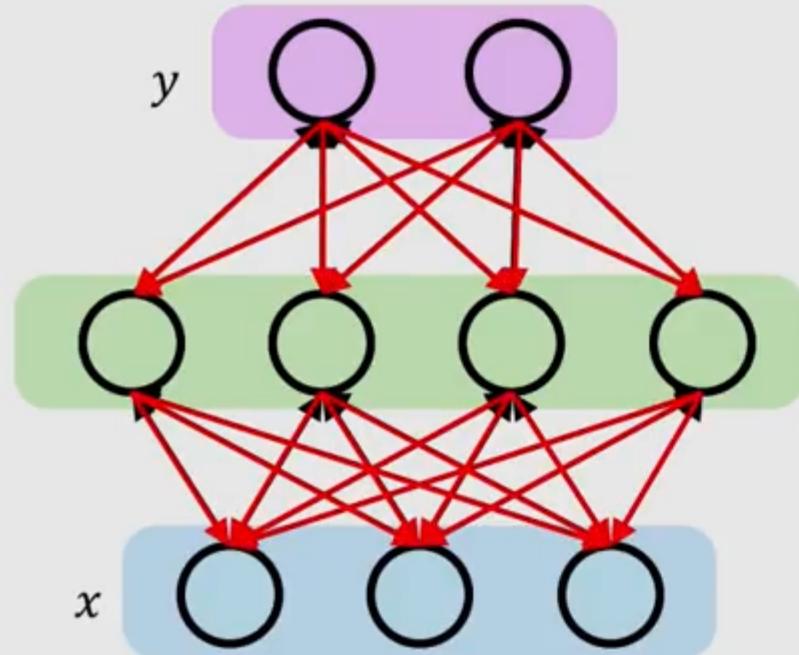


MIT Deep Learning

# Recall: Backpropagation in Feed Forward Models



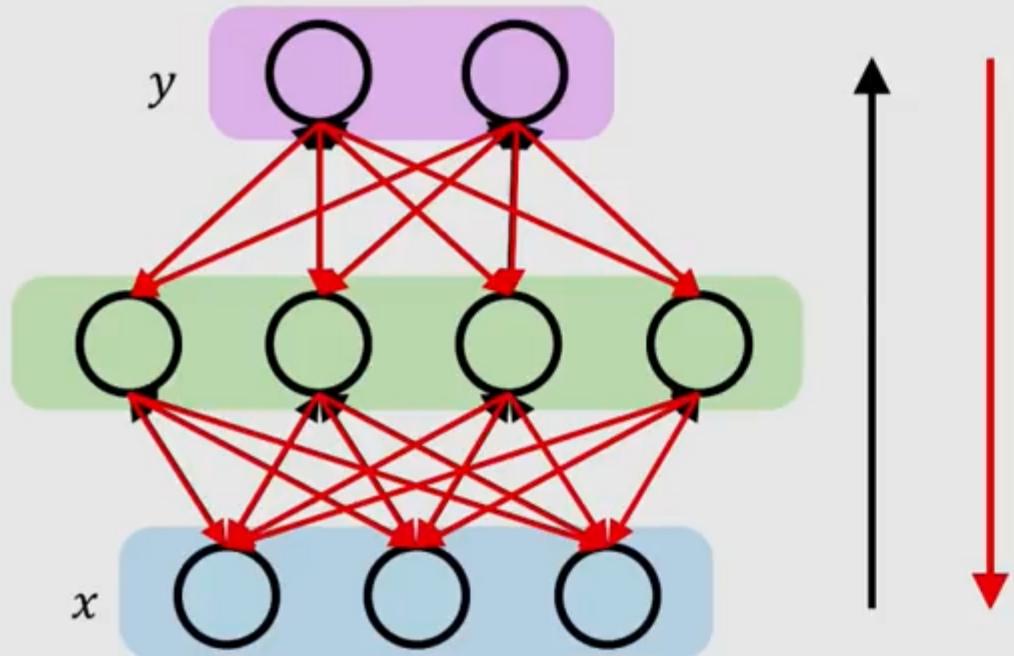
# Recall: Backpropagation in Feed Forward Models



Backpropagation algorithm:



# Recall: Backpropagation in Feed Forward Models

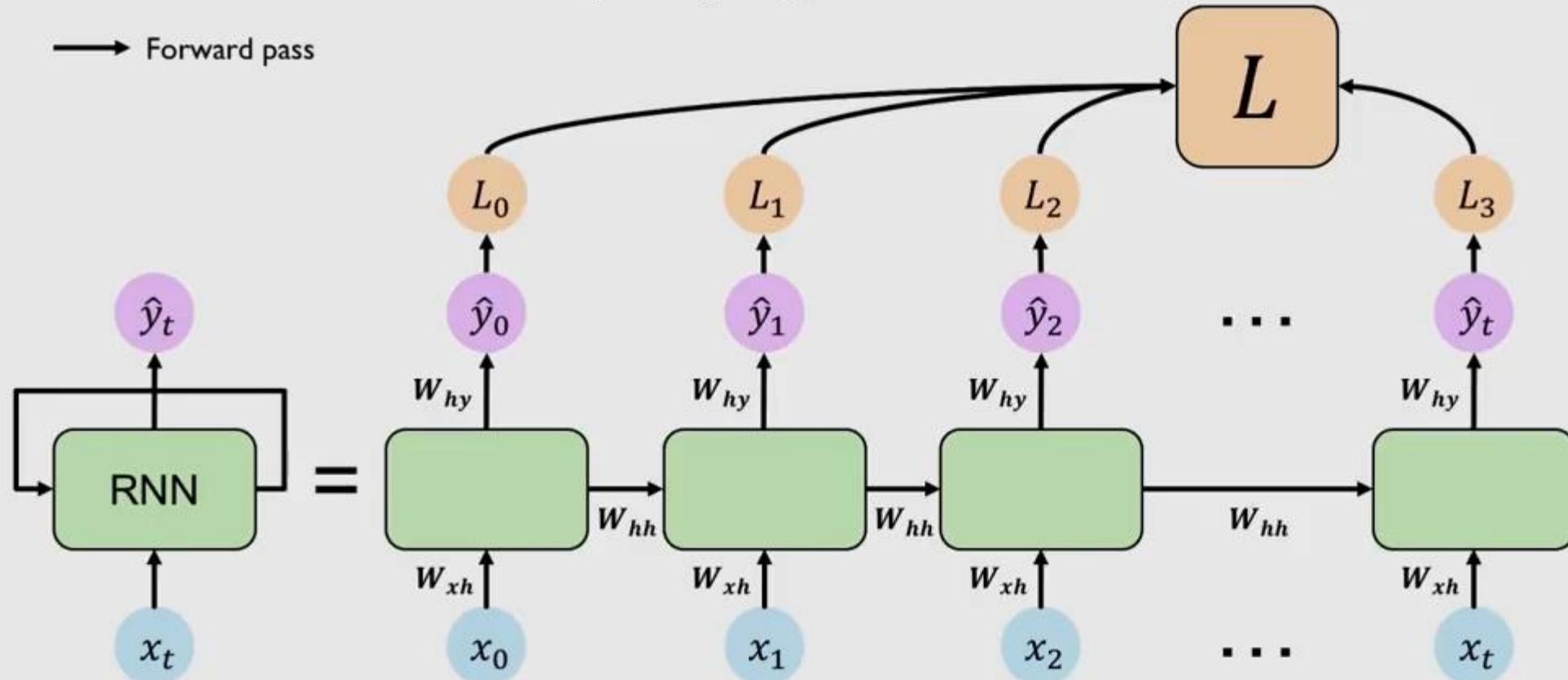


**Backpropagation algorithm:**

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

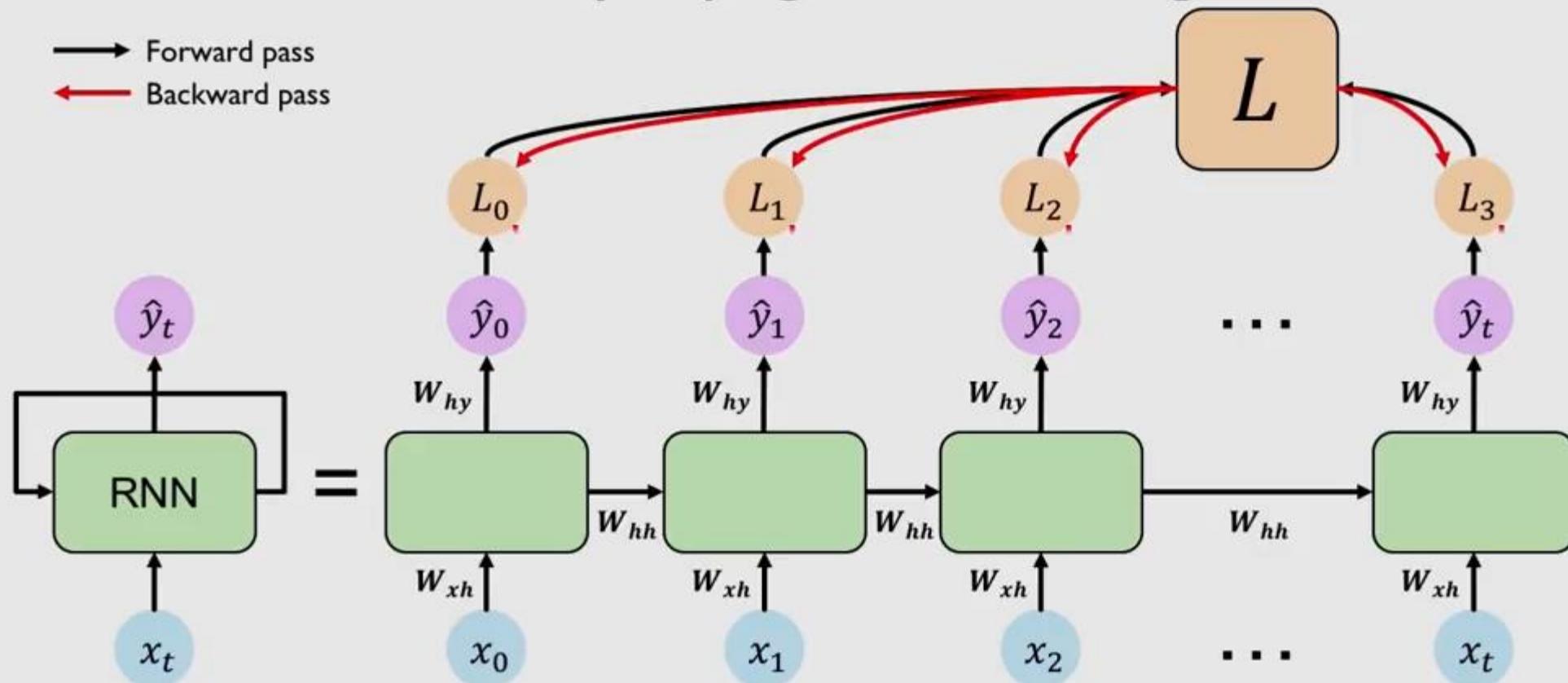
# RNNs: Backpropagation Through Time

→ Forward pass



# RNNs: Backpropagation Through Time

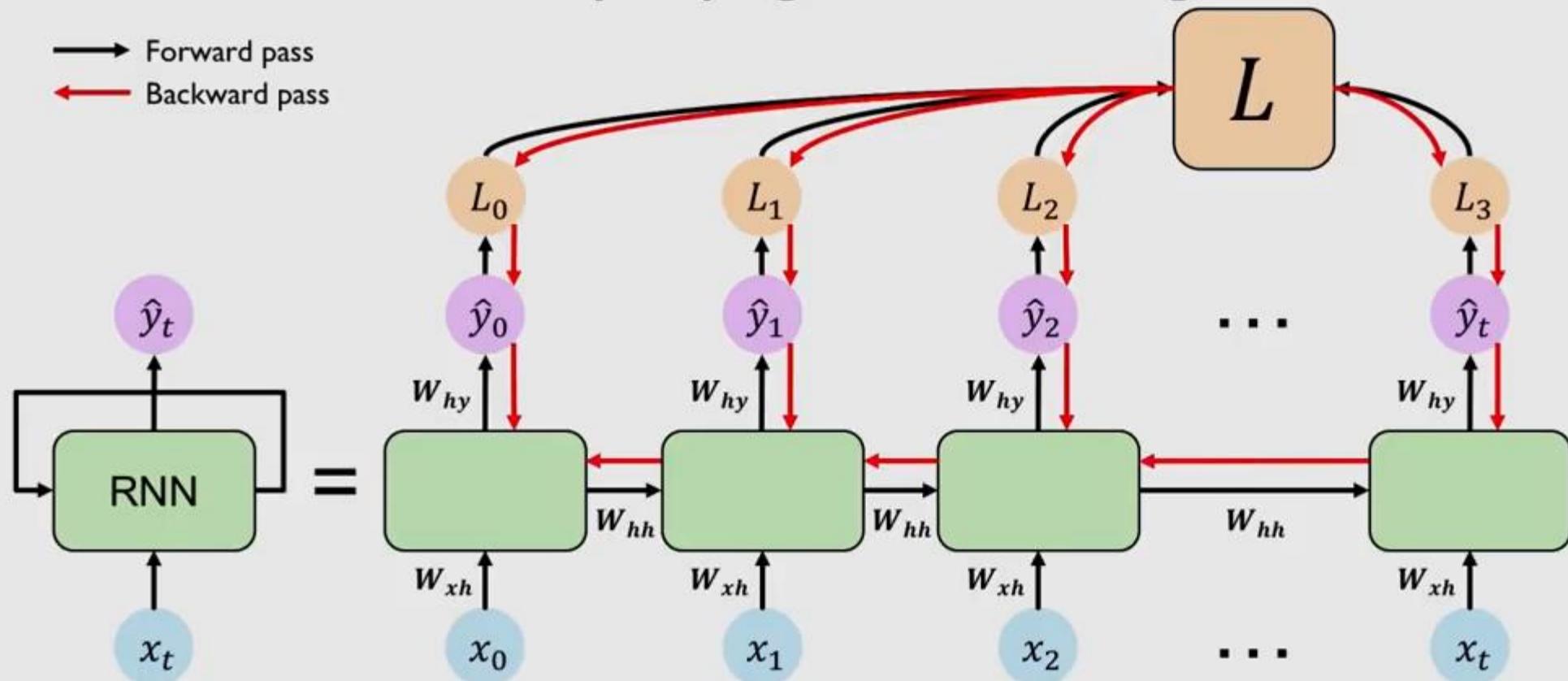
→ Forward pass  
← Backward pass



MIT Deep Learning

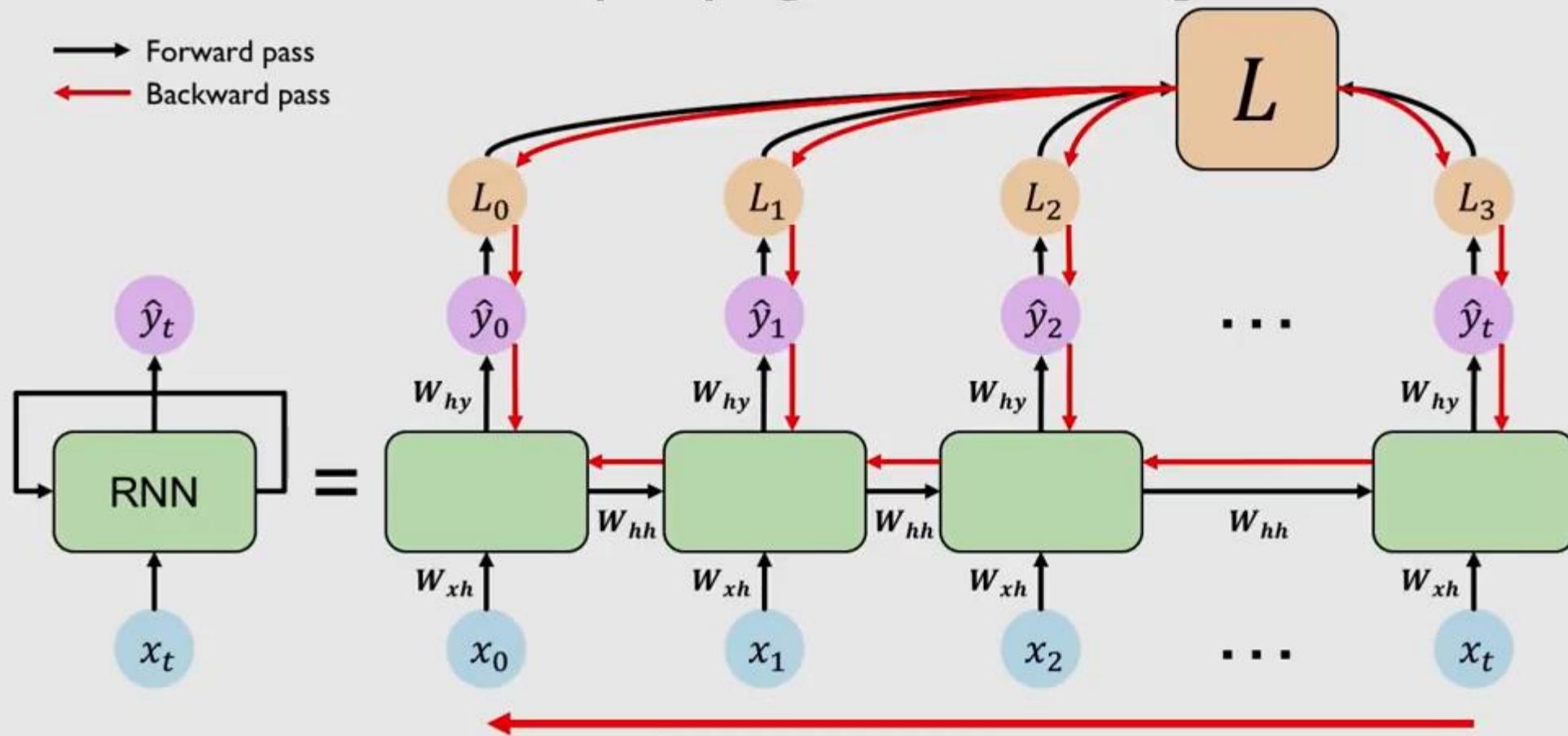
# RNNs: Backpropagation Through Time

→ Forward pass  
← Backward pass

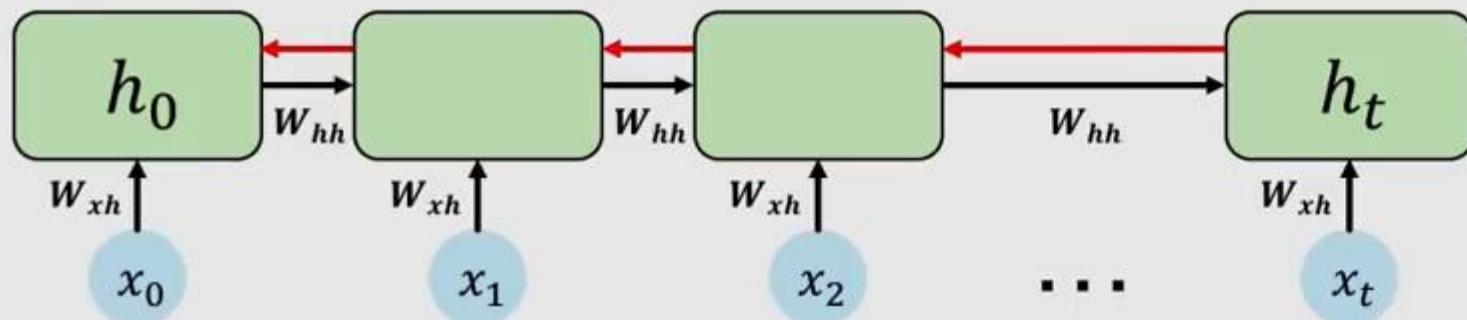


# RNNs: Backpropagation Through Time

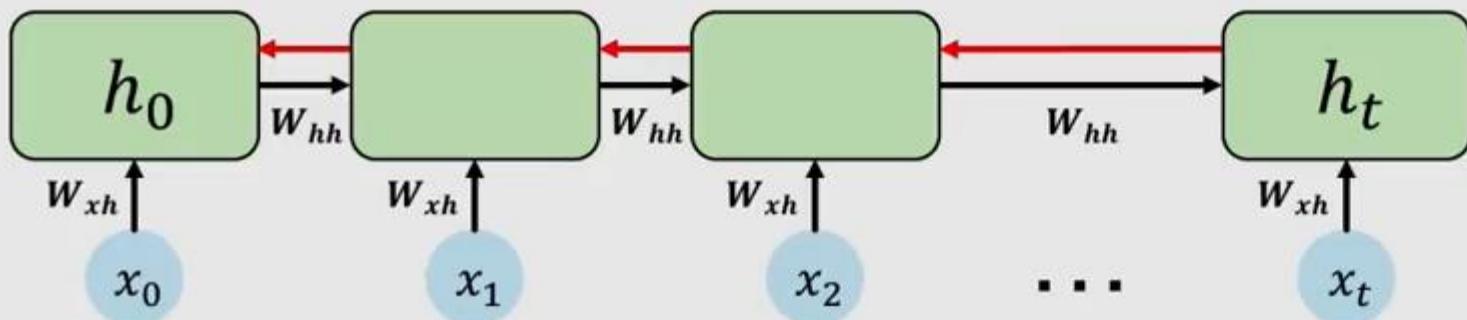
→ Forward pass  
← Backward pass



# Standard RNN Gradient Flow

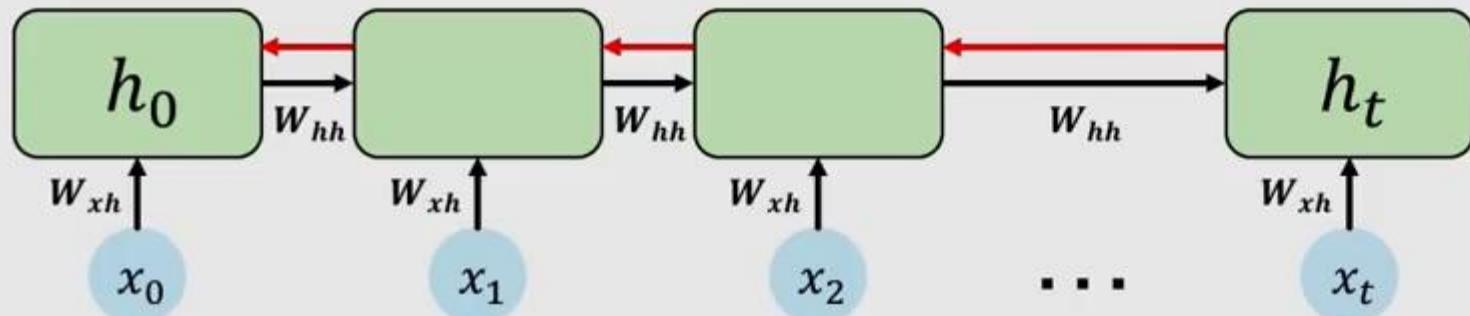


# Standard RNN Gradient Flow



Computing the gradient wrt  $h_0$  involves **many factors of  $W_{hh}$**  + repeated gradient computation!

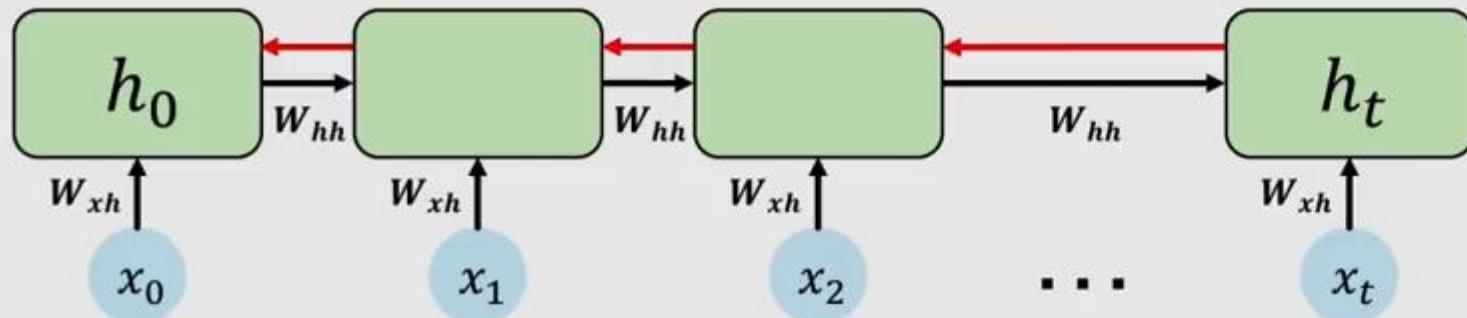
# Standard RNN Gradient Flow: Exploding Gradients



Computing the gradient wrt  $h_0$  involves **many factors of  $W_{hh}$**  + repeated gradient computation!

Many values  $> 1$ :  
exploding gradients

# Standard RNN Gradient Flow: Vanishing Gradients



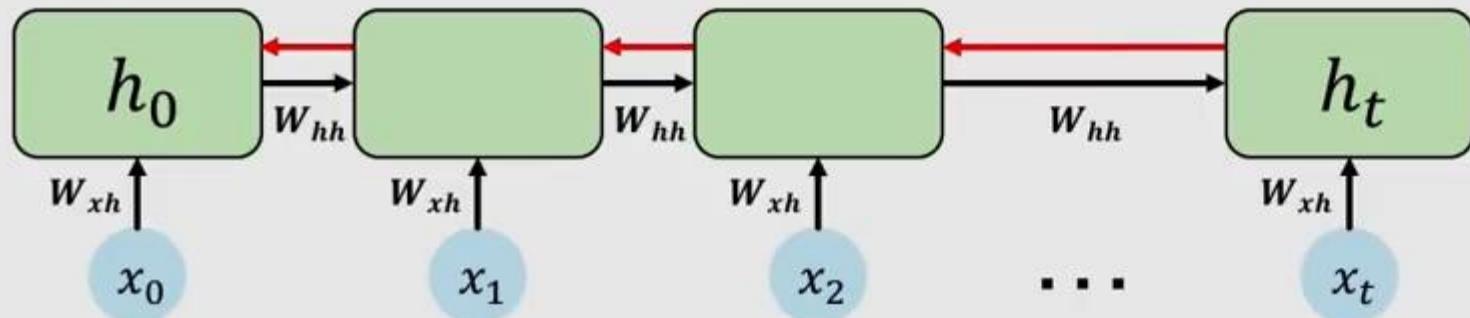
Computing the gradient wrt  $h_0$  involves **many factors of  $W_{hh}$**  + repeated gradient computation!

Many values  $> 1$ :  
exploding gradients

Gradient clipping to  
scale big gradients

Many values  $< 1$ :  
vanishing gradients

# Standard RNN Gradient Flow: Vanishing Gradients



Computing the gradient wrt  $h_0$  involves **many factors of  $W_{hh}$**  + repeated gradient computation!



Many values > 1:  
exploding gradients

Gradient clipping to  
scale the gradients

Many values < 1:  
vanishing gradients

1. Activation function
2. Weight initialization
3. Network architecture

# The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?



# The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together



# 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



# 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



# The Problem of Long-Term Dependencies

"The clouds are in the \_\_\_"

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



# The Problem of Long-Term Dependencies

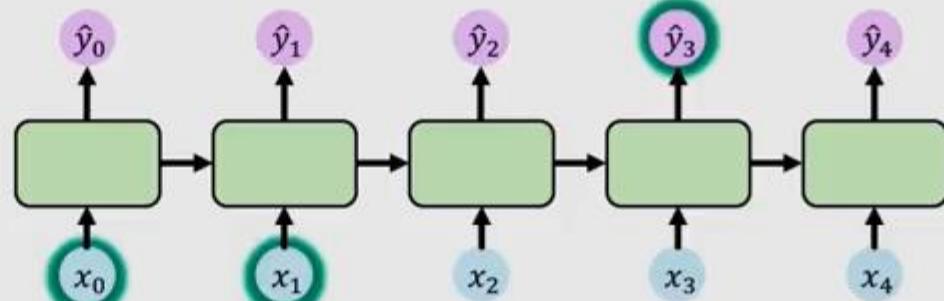
Why are vanishing gradients a problem?

Multiply many **small numbers** together

↓  
Errors due to further back time steps  
have smaller and smaller gradients

↓  
Bias parameters to capture short-term  
dependencies

"The clouds are in the \_\_\_"



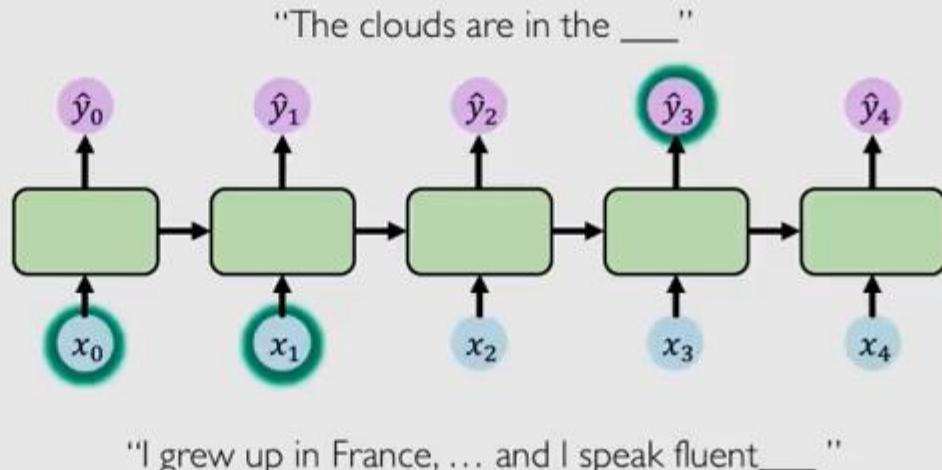
# The Problem of Long-Term Dependencies

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



# The Problem of Long-Term Dependencies

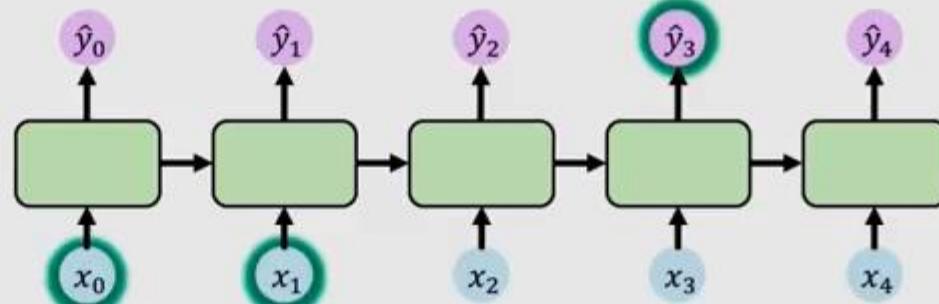
Why are vanishing gradients a problem?

Multiply many **small numbers** together

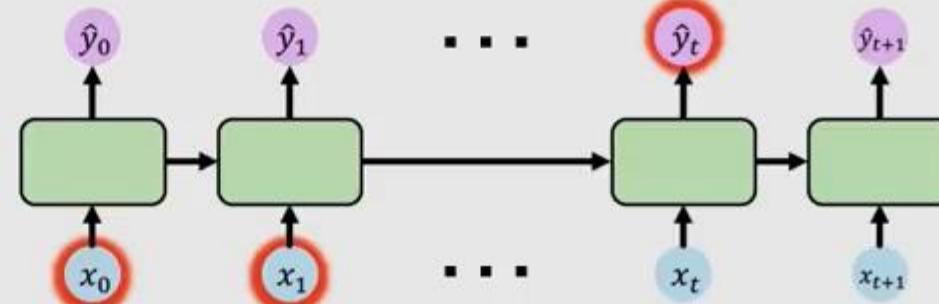
↓  
Errors due to further back time steps  
have smaller and smaller gradients

↓  
Bias parameters to capture short-term  
dependencies

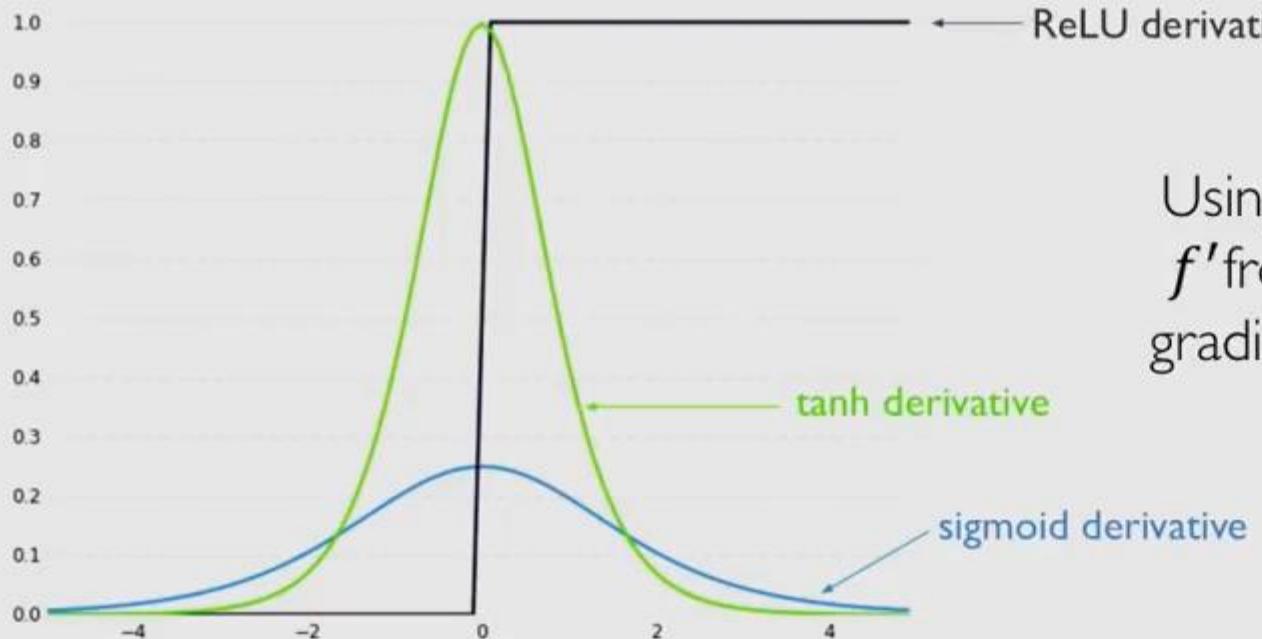
"The clouds are in the \_\_\_"



"I grew up in France, ... and I speak fluent \_\_\_"



# Trick #1: Activation Functions



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

## Trick #2: Parameter Initialization

Initialize **weights** to identity matrix

Initialize **biases** to zero

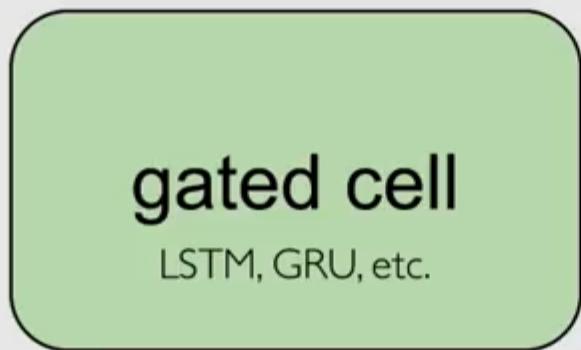
$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

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



# Solution #3: Gated Cells

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



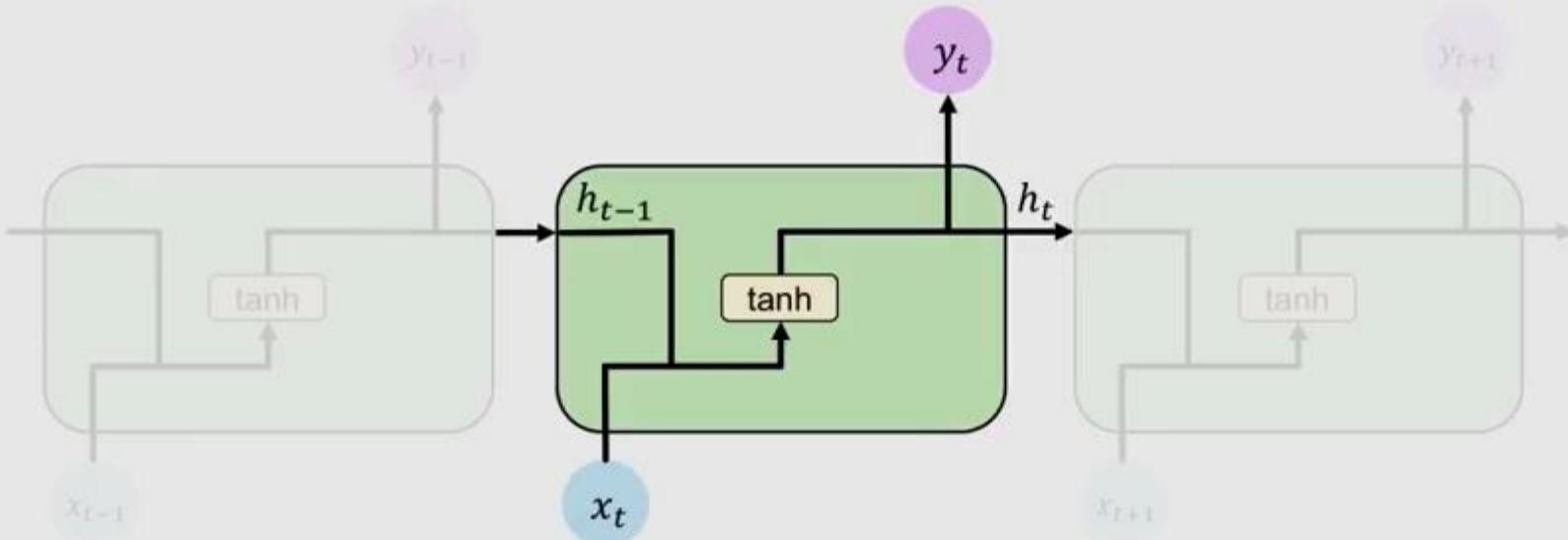
**Long Short Term Memory (LSTMs)** networks rely on a gated cell to track information throughout many time steps.

# Long Short Term Memory (LSTM) Networks



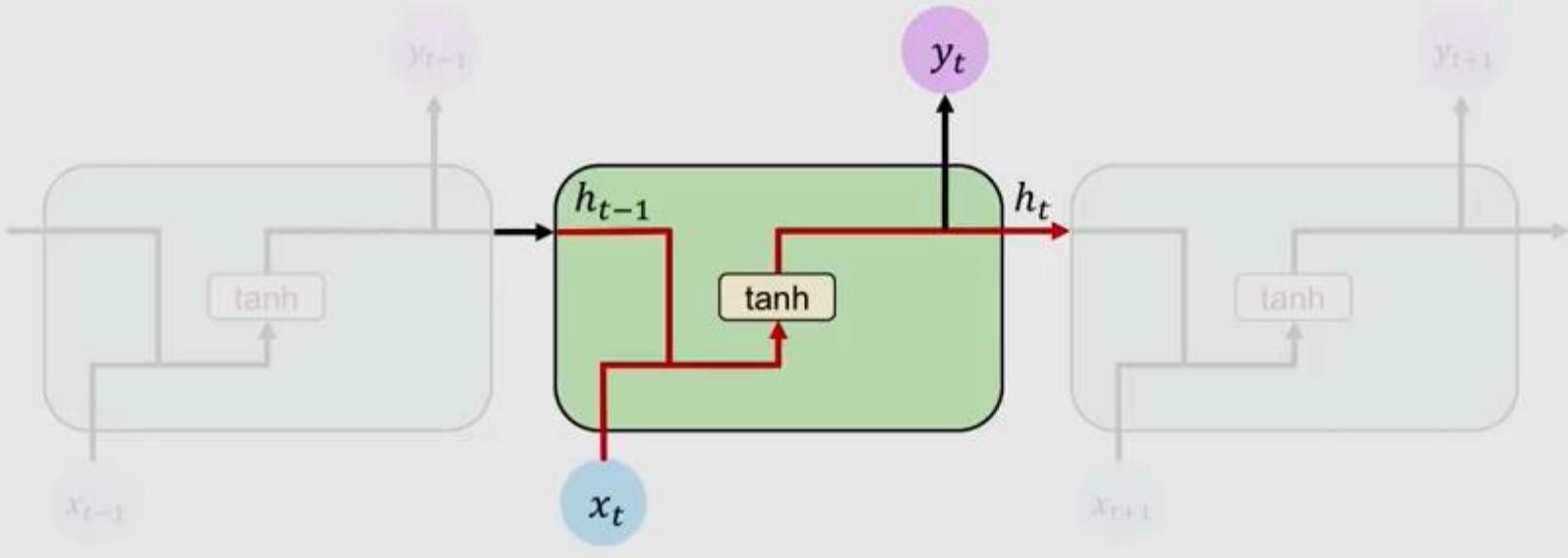
# Standard RNN

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



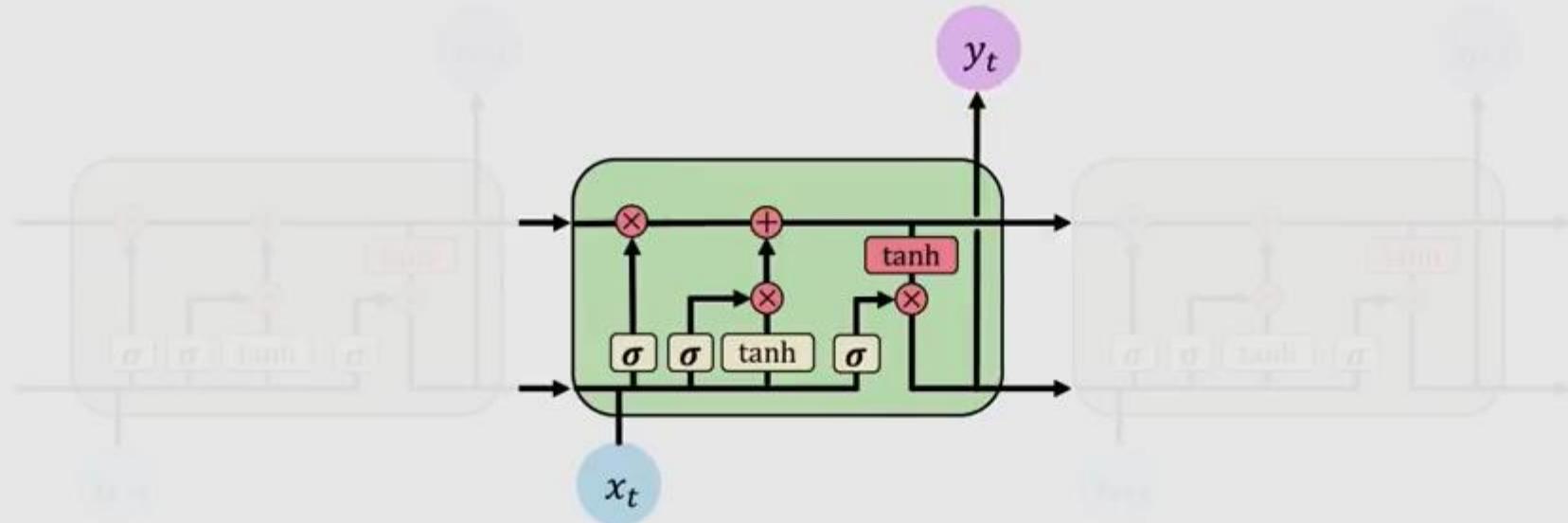
# Standard RNN

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**

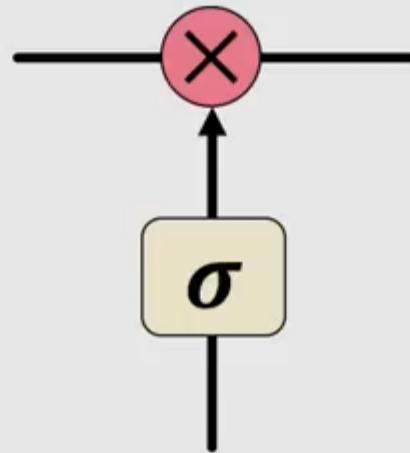


LSTM cells are able to track information throughout many timesteps

```
tf.keras.layers.LSTM(num_units)
```

# Long Short Term Memory (LSTMs)

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

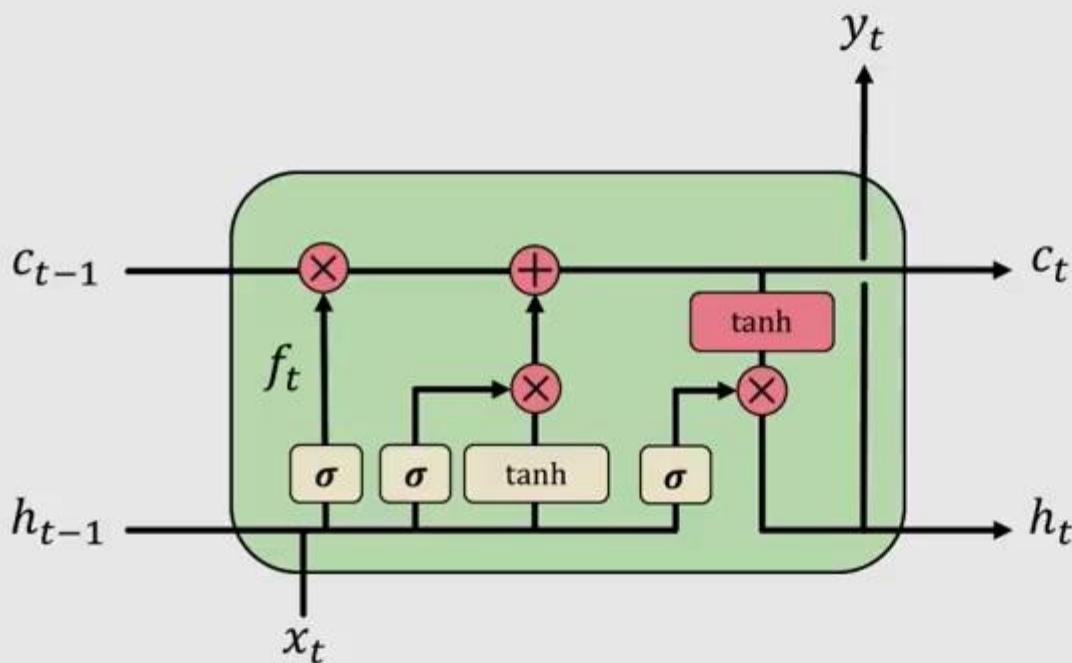


Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

# Long Short Term Memory (LSTMs)

How do LSTMs work?

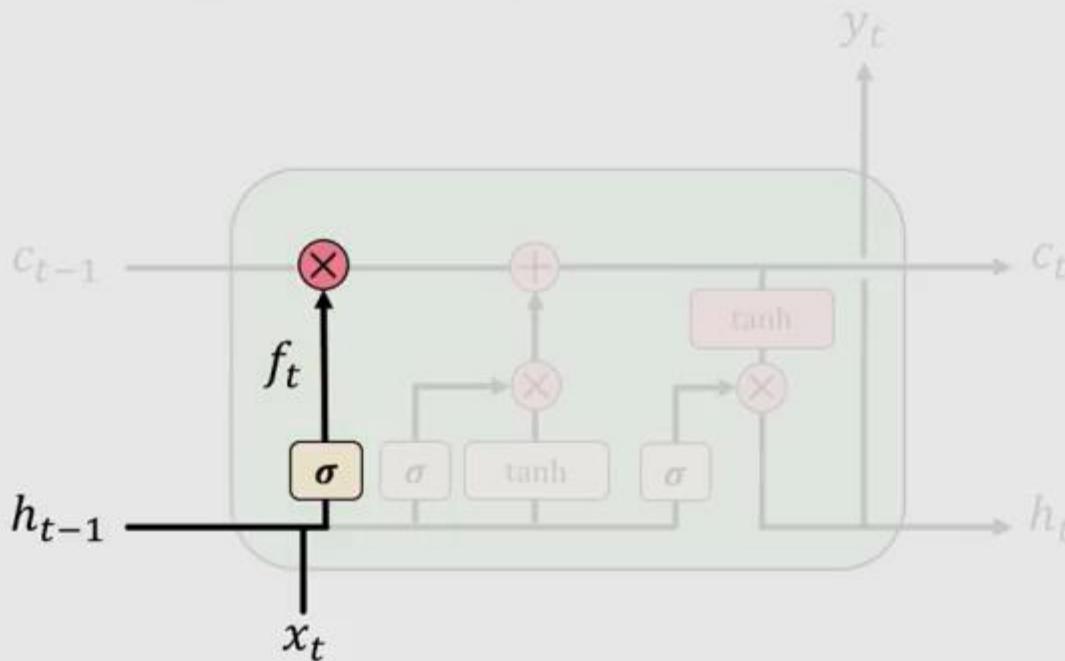
- 1) Forget
- 2) Store
- 3) Update
- 4) Output



# Long Short Term Memory (LSTMs)

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

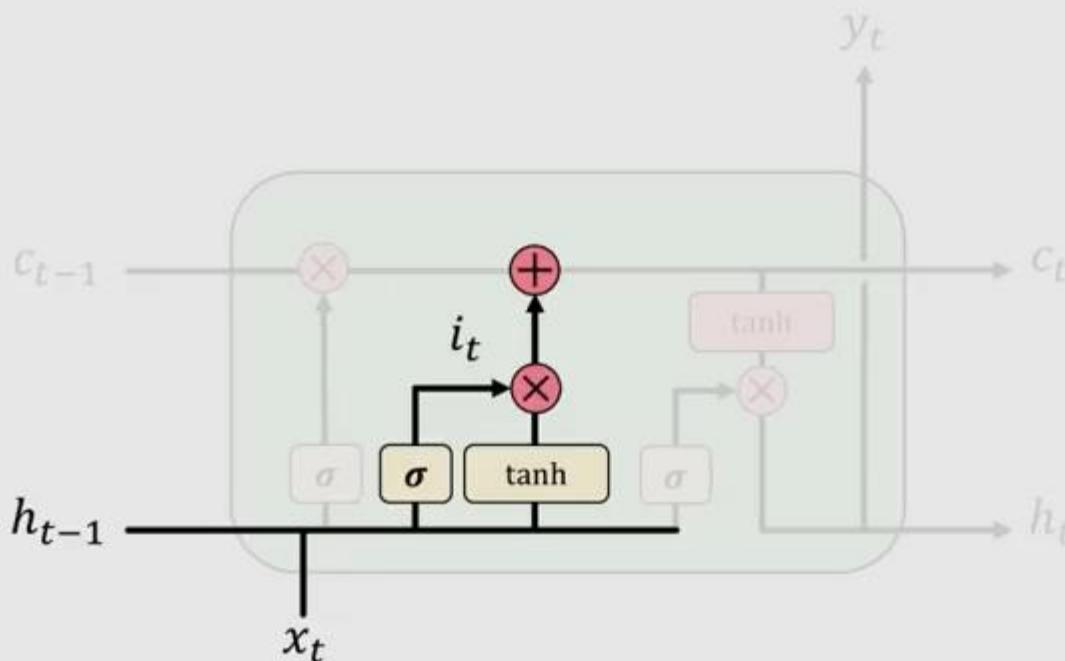
LSTMs **forget irrelevant** parts of the previous state



# Long Short Term Memory (LSTMs)

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

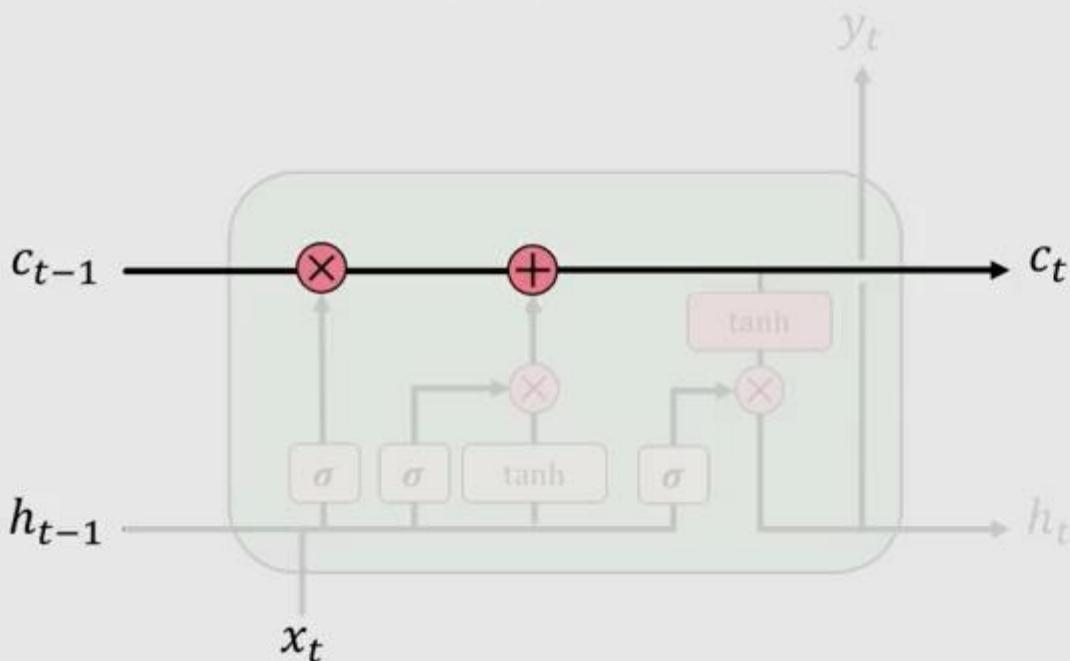
LSTMs **store relevant** new information into the cell state



# Long Short Term Memory (LSTMs)

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

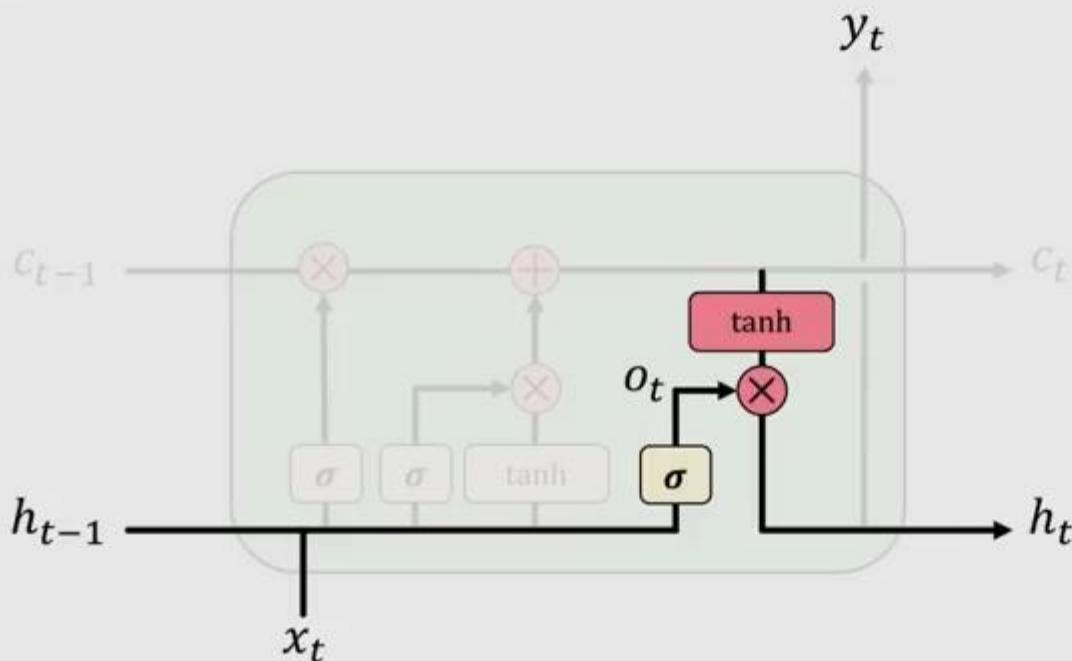
LSTMs **selectively update** cell state values



# Long Short Term Memory (LSTMs)

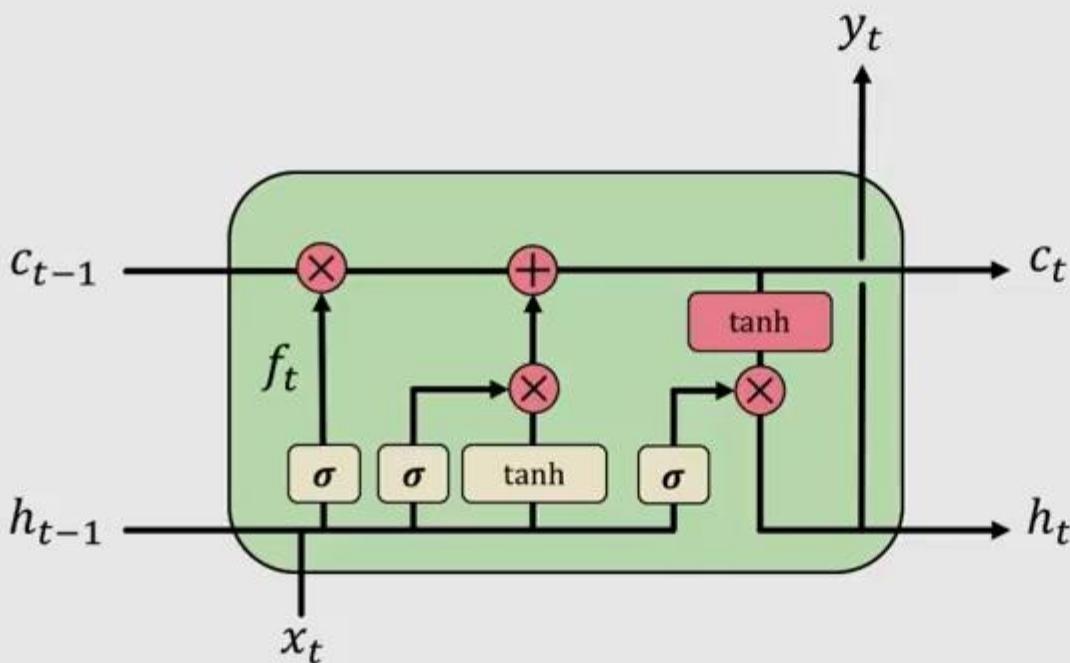
- 1) Forget
- 2) Store
- 3) Update
- 4) Output**

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



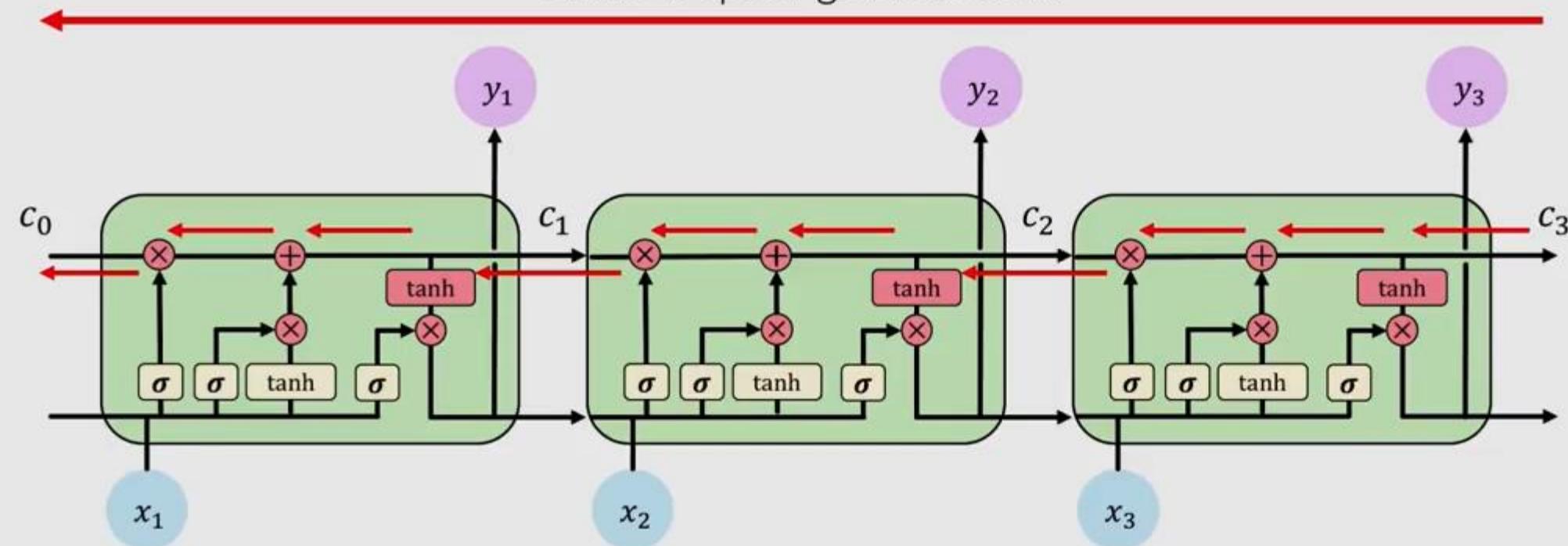
# Long Short Term Memory (LSTMs)

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



# LSTM Gradient Flow

Uninterrupted gradient flow!



# LSTMs: Key Concepts



# LSTMs: Key Concepts

- I. Maintain a **separate cell state** from what is outputted



# LSTMs: Key Concepts

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**



# LSTMs: Key Concepts

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
  - **Forget** gate gets rid of irrelevant information



# LSTMs: Key Concepts

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
  - **Forget** gate gets rid of irrelevant information
  - **Store** relevant information from current input



# LSTMs: Key Concepts

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
  - **Forget** gate gets rid of irrelevant information
  - **Store** relevant information from current input
  - Selectively **update** cell state



# LSTMs: Key Concepts

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
  - **Forget** gate gets rid of irrelevant information
  - **Store** relevant information from current input
  - Selectively **update** cell state
  - **Output** gate returns a filtered version of the cell state



# LSTMs: Key Concepts

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
  - **Forget** gate gets 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**



**THANK YOU!**