Deep Sequence Modeling

Ava Soleimany

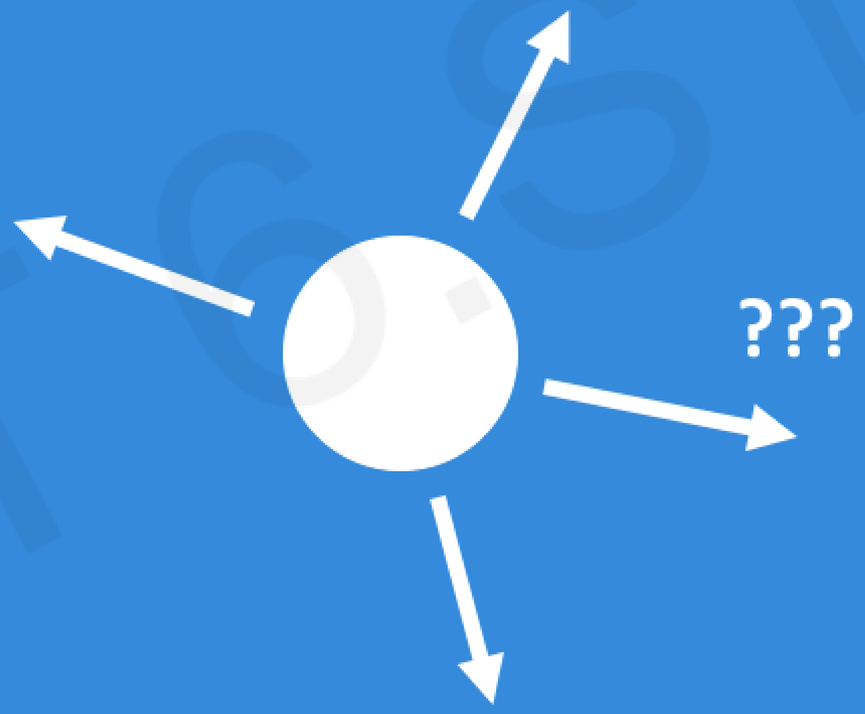
MIT 6.S191

January 27, 2020

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

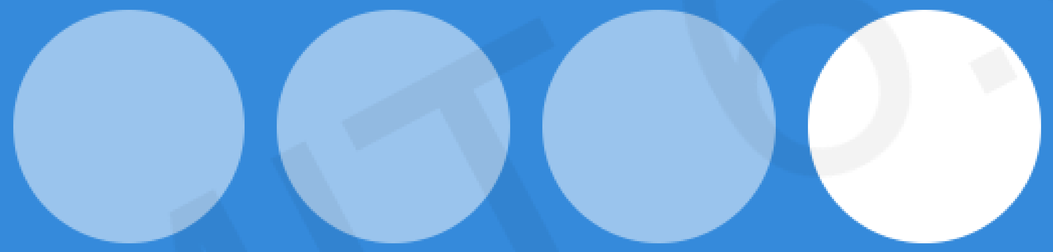
MIT 6.S191

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



MIT 6.819

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



MIT 6.S191

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

MIT 6.S191

Sequences in the Wild

Audio

MIT 6.S191

Sequences in the Wild

character:

6.S191 Introduction to Deep Learning

word:

Text

MIT 6.S191

A Sequence Modeling Problem: Predict the Next Word

MIT 6.S191

A Sequence Modeling Problem: Predict the Next Word

"This morning I took my cat for a walk."

MIT 6.S191

A Sequence Modeling Problem: Predict the Next Word

"This morning I took my cat for a walk."

given these words

MIT 6.S191

A Sequence Modeling Problem: Predict the Next Word

"This morning I took my cat for a walk."

given these words

predict the next word

MIT 6.S191

Idea #1: Use a Fixed Window

"This morning I took my cat for a walk."

given these two words

predict the next word

MIT 6.S191

Idea #1: Use a Fixed Window

"This morning I took my cat for a walk."

given these two words

predict the next word

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

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

for

a

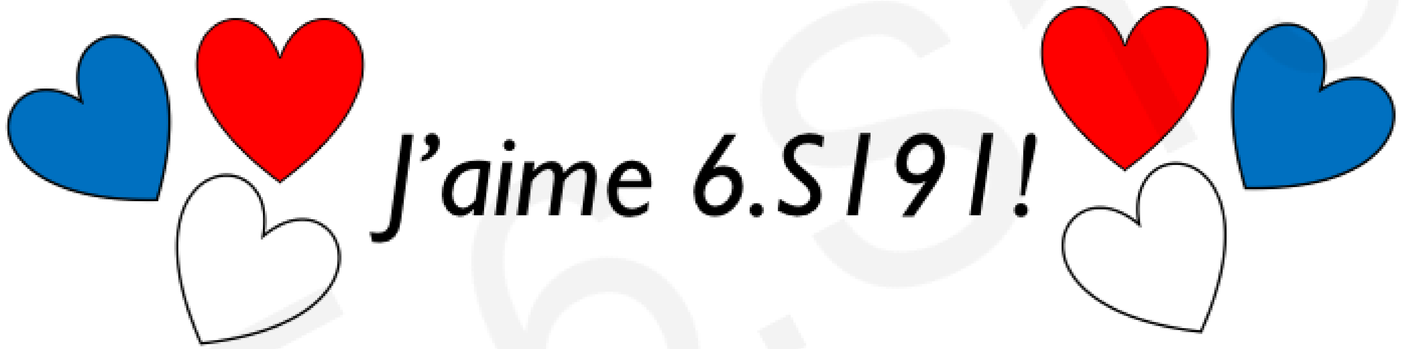
↓

prediction

MIT 6.S191

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.

MIT 6.S191

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

MIT 6.S191

Problem #2: Counts Don't Preserve Order

The food was good, not bad at all.

vs.

The food was bad, not good at all.

MIT 6.S191

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

morning I took this cat

prediction

MIT 6.S191

Problem #3: No Parameter Sharing

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

this morning took the cat

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

MIT 6.S191

Problem #3: No Parameter Sharing

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

this morning

MIT 6.S191

Problem #3: No Parameter Sharing

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

this morning

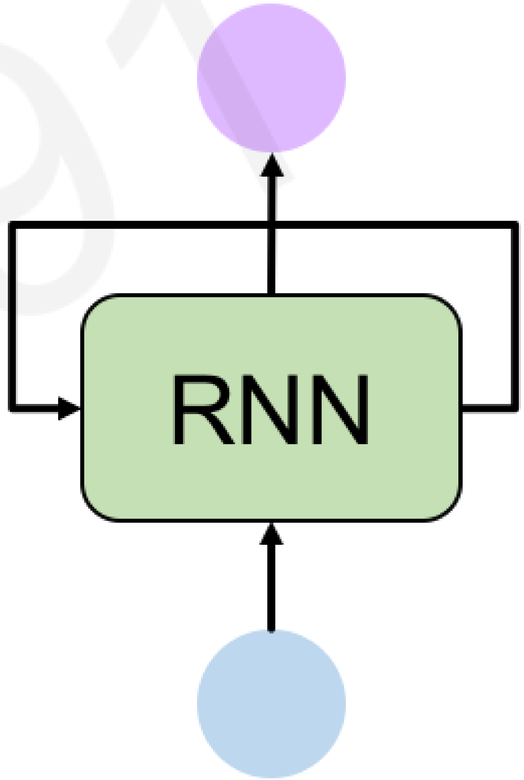
Things we learn about the sequence **won't transfer** if they appear **elsewhere** in the sequence.

MIT 6.S191

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



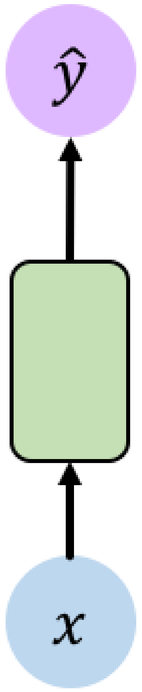
Today: **Recurrent Neural Networks (RNNs)** as an approach to sequence modeling problems

MIT 6.S191

Recurrent Neural Networks (RNNs)

MIT 6.S191

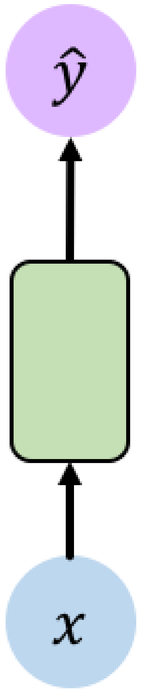
Standard Feed-Forward Neural Network



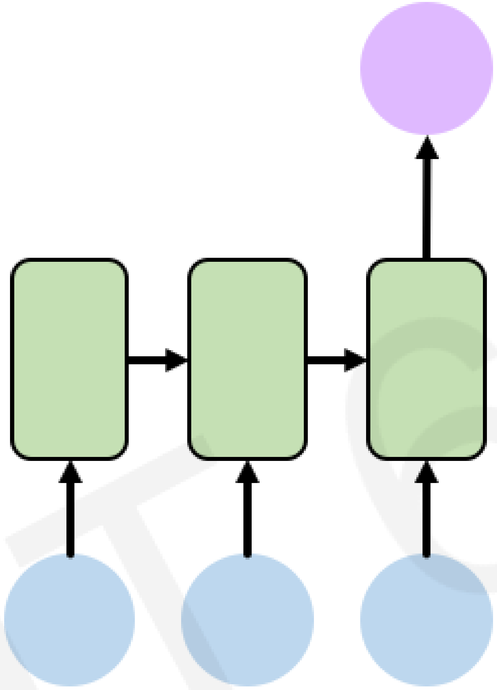
One to One
"Vanilla" neural network

MIT 6.S191

Recurrent Neural Networks for Sequence Modeling



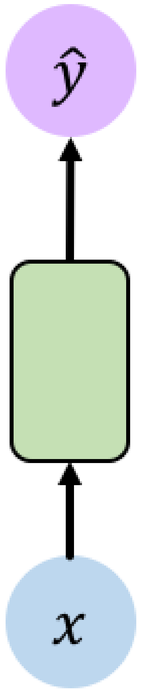
One to One
"Vanilla" neural network



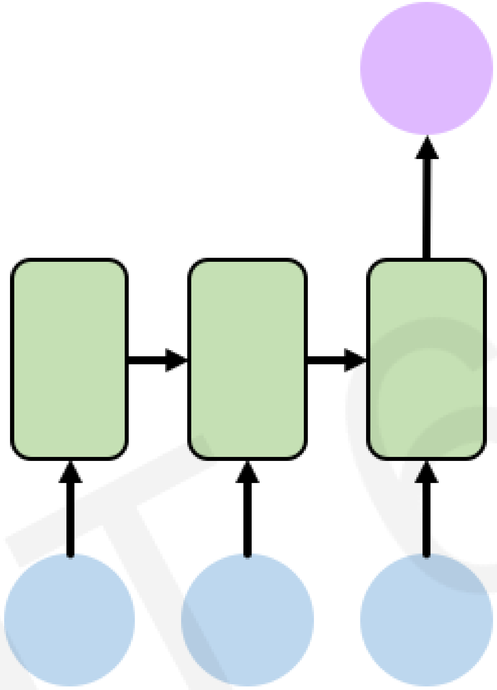
Many to One
Sentiment Classification

MIT 6.S191

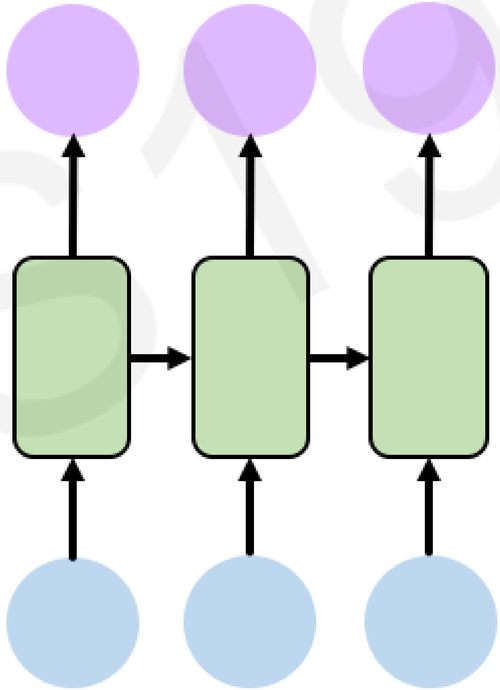
Recurrent Neural Networks for Sequence Modeling



One to One
"Vanilla" neural network



Many to One
Sentiment Classification

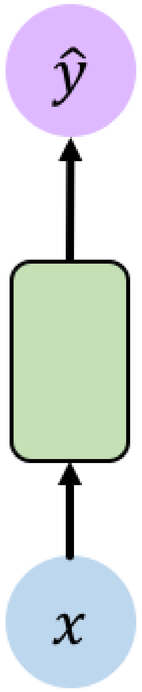


Many to Many
Music Generation

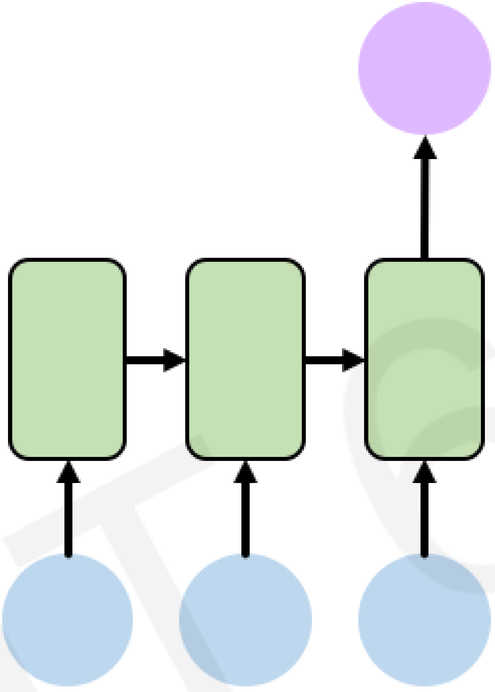
6.S191 Lab!

MIT 6.S191

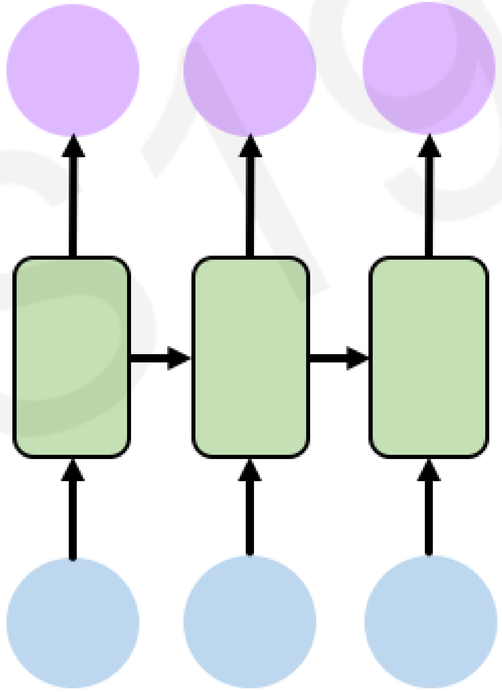
Recurrent Neural Networks for Sequence Modeling



One to One
"Vanilla" neural network



Many to One
Sentiment Classification



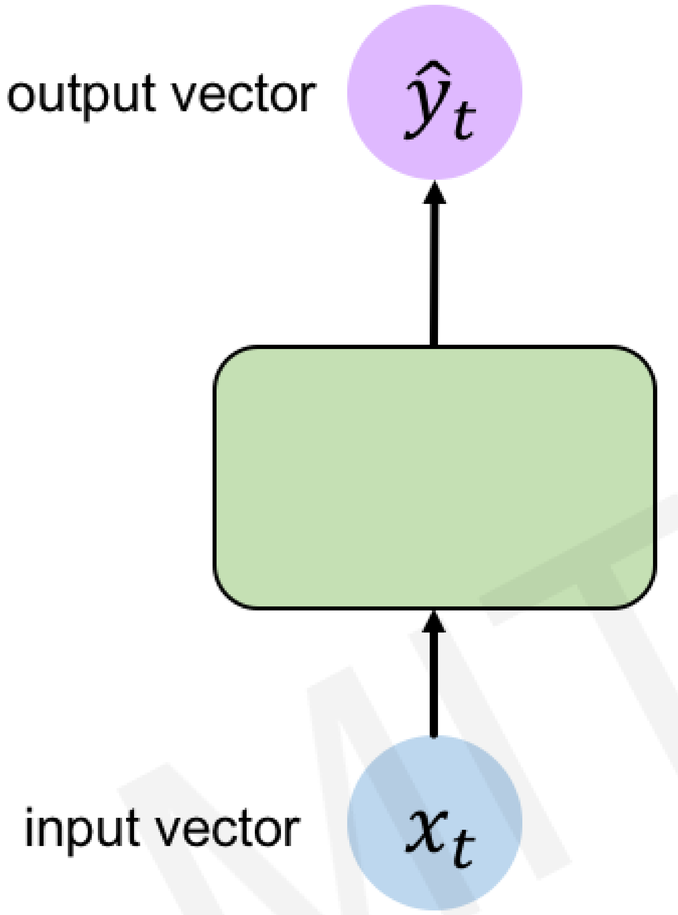
Many to Many
Music Generation

6.S191 Lab!

… and many other architectures and applications

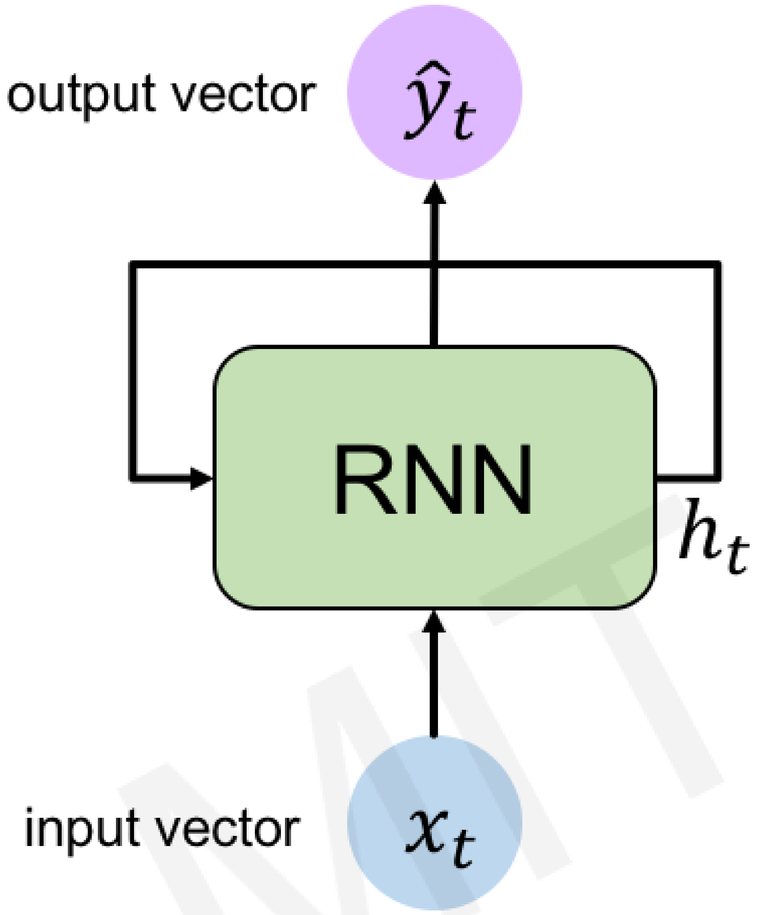
MIT 6.S191

Standard "Vanilla" Neural Network



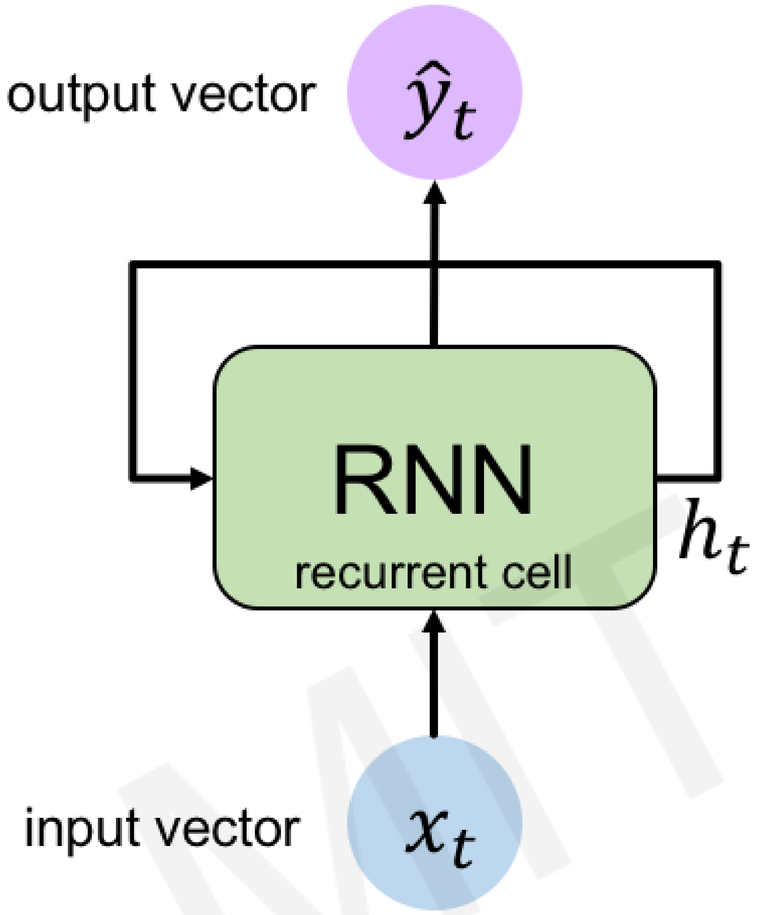
MIT 6.S191

Recurrent Neural Network (RNN)



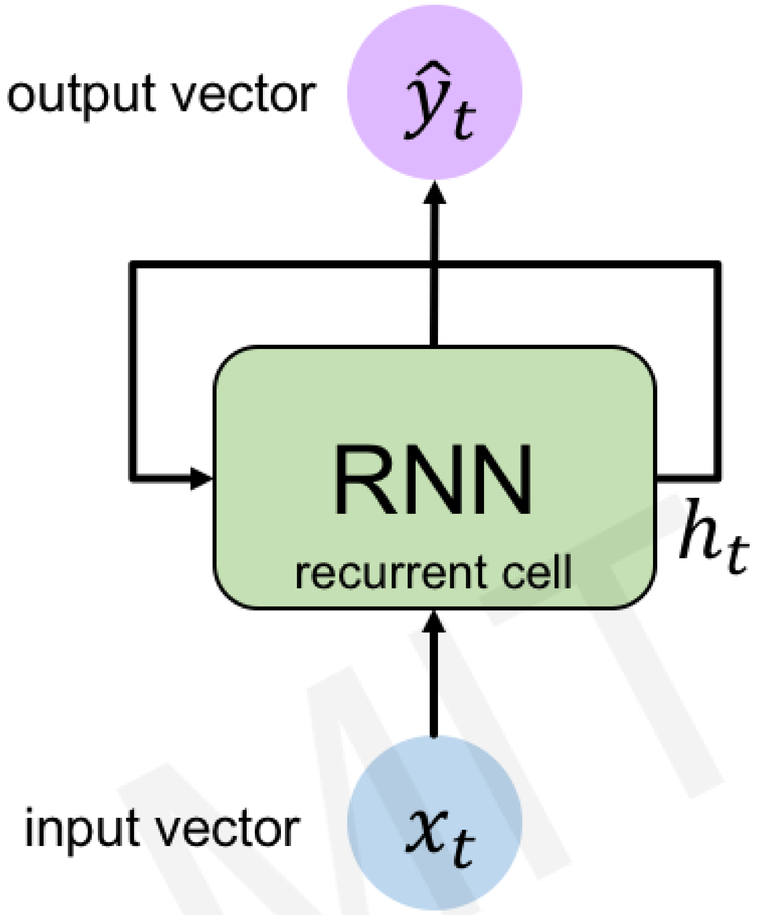
MIT 6.S191

Recurrent Neural Network (RNN)



MIT 6.S191

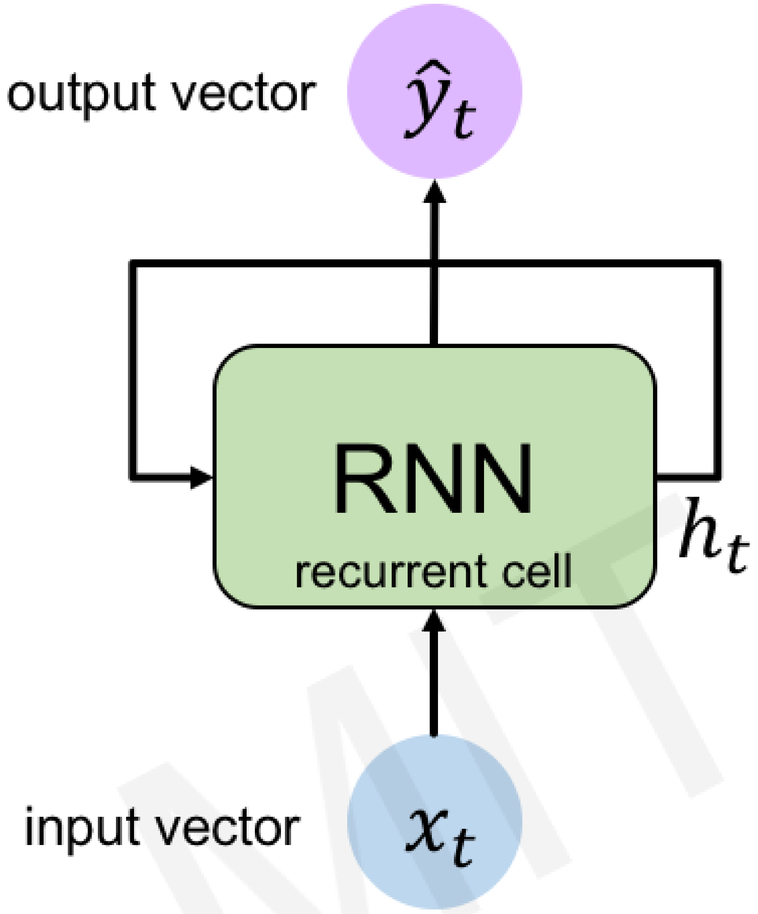
Recurrent Neural Network (RNN)



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

MIT 6.S191

Recurrent Neural Network (RNN)



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

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

new state

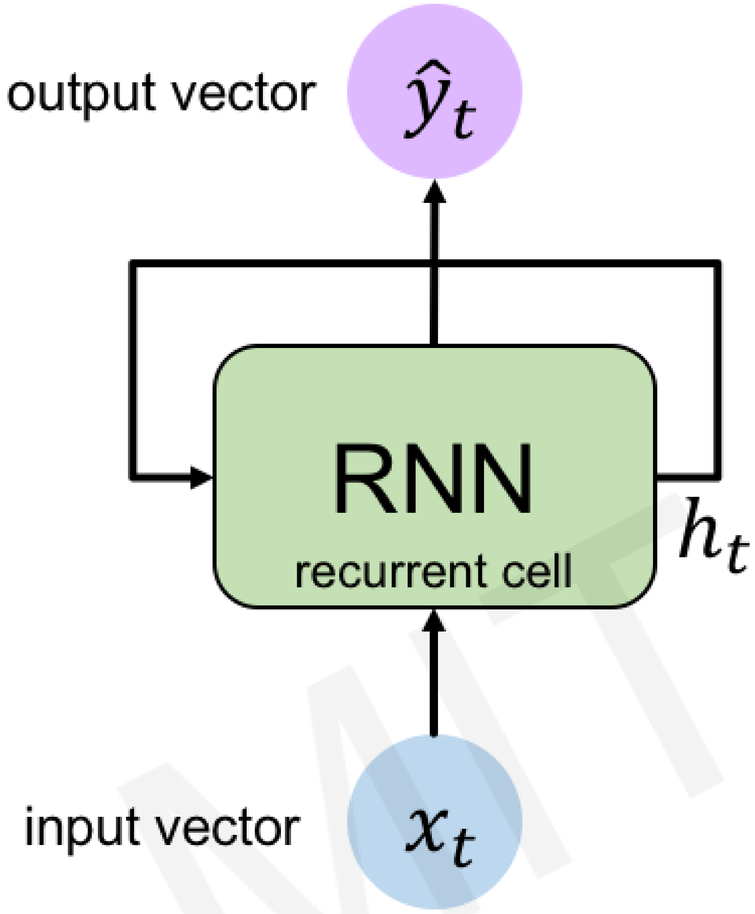
function parameterized by W

old state

input vector at time step *t*

MIT 6.S191

Recurrent Neural Network (RNN)



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

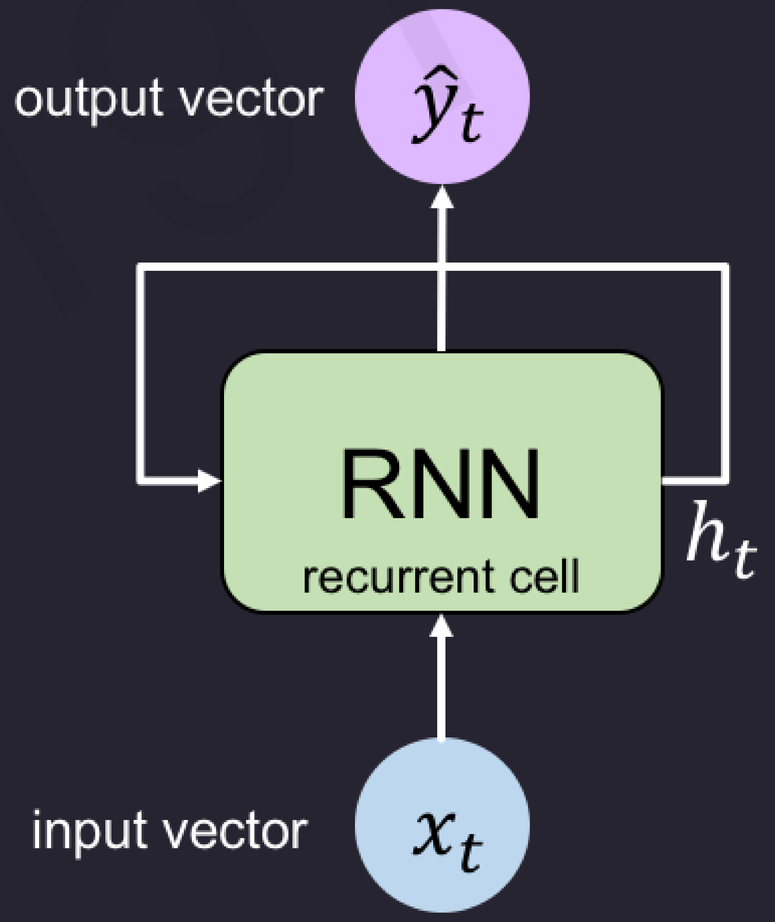
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new 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

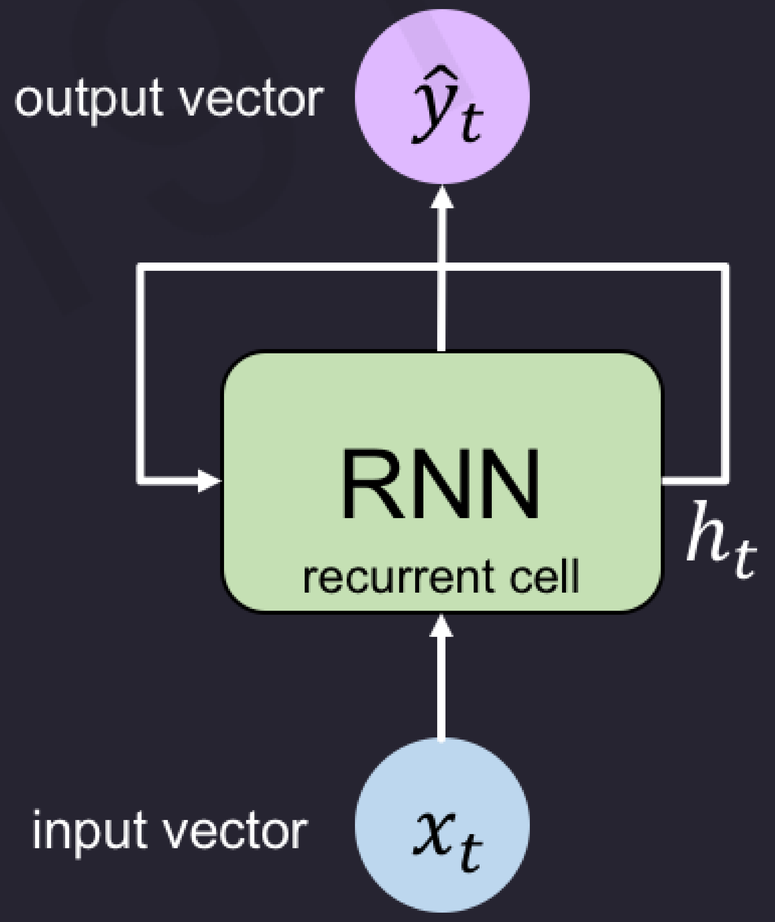
RNN Intuition

```
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!"
```



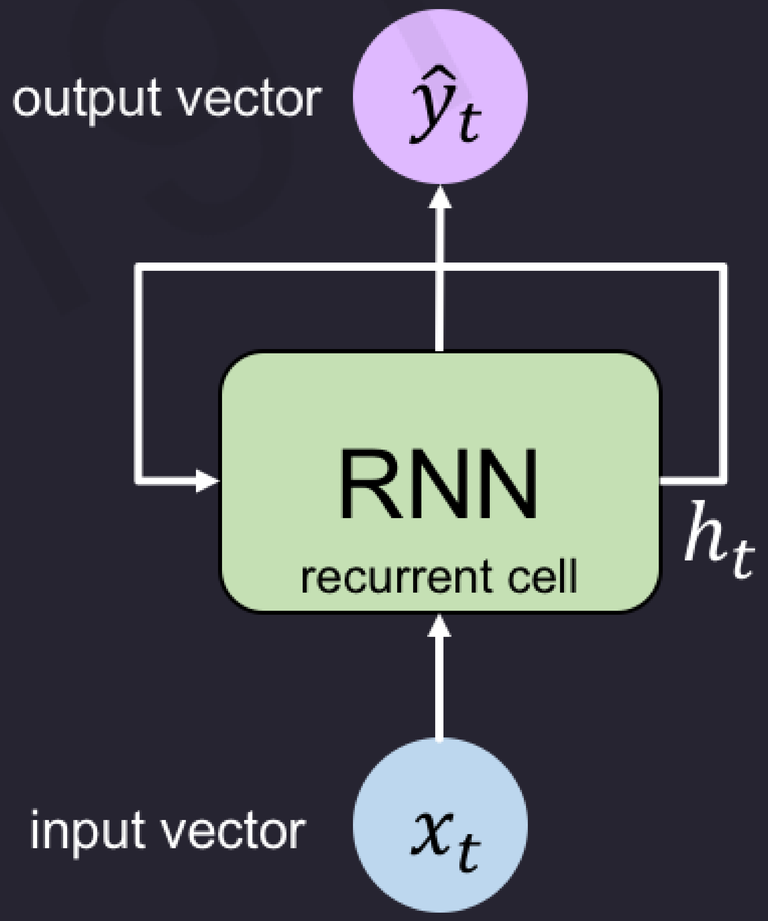
RNN Intuition

```
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!"
```

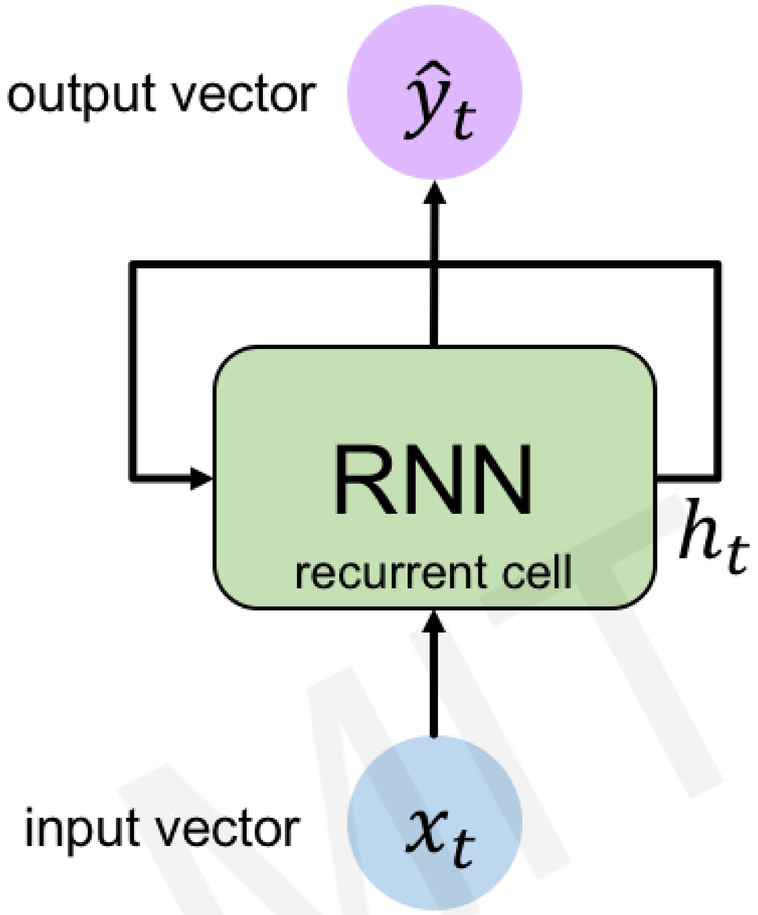


RNN Intuition

```
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!"
```

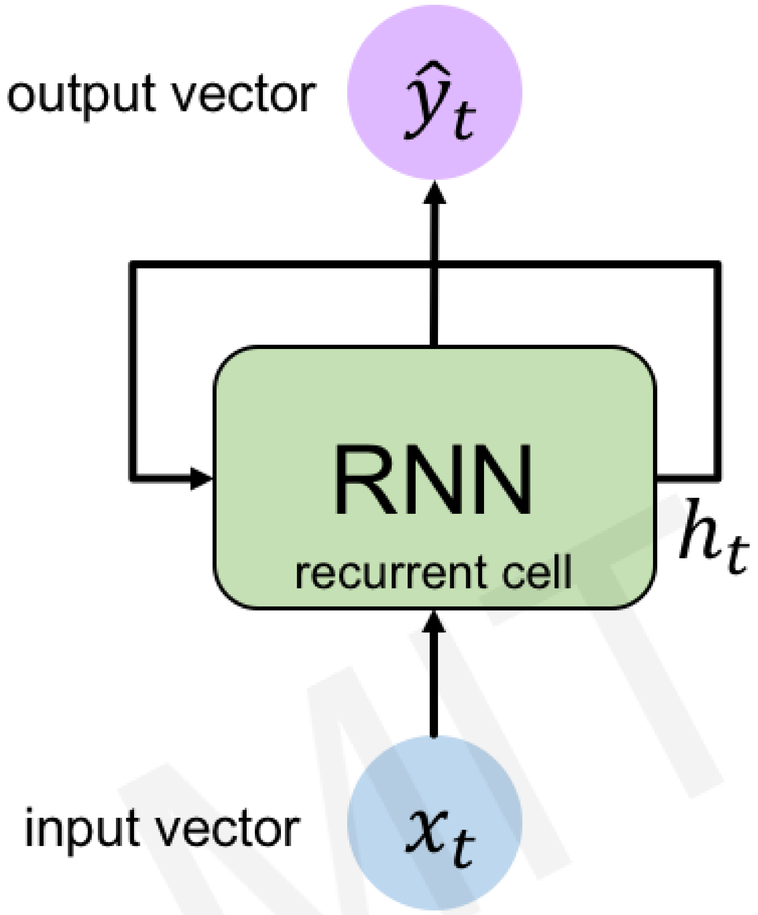


RNN State Update and Output



MIT 6.S191

RNN State Update and Output

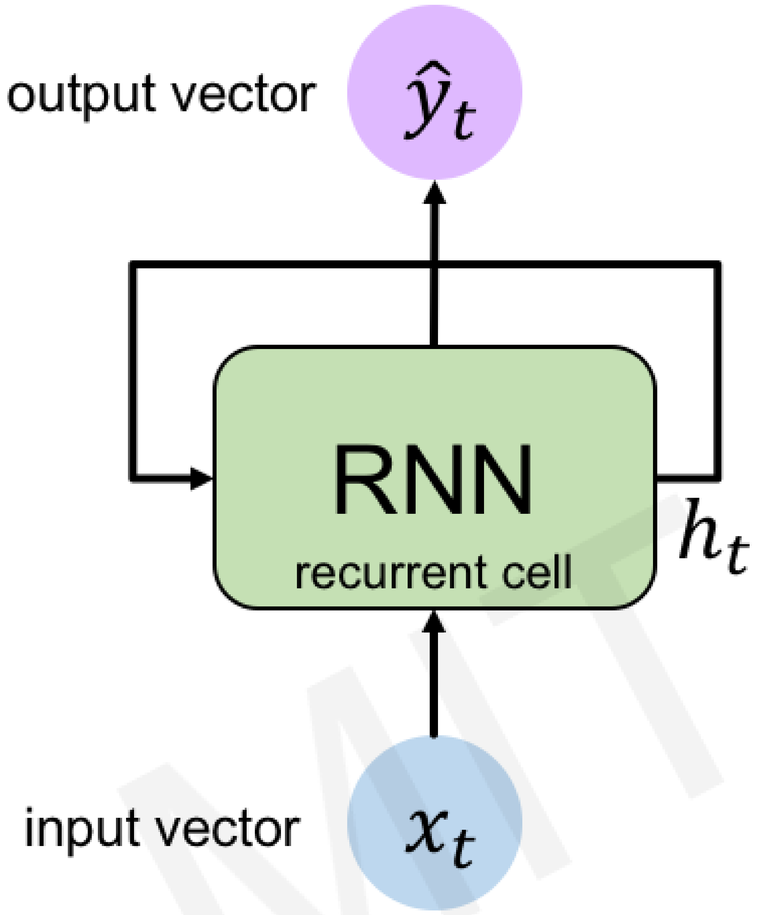


Input Vector

x_t

MIT 6.S191

RNN State Update and Output



Update Hidden State

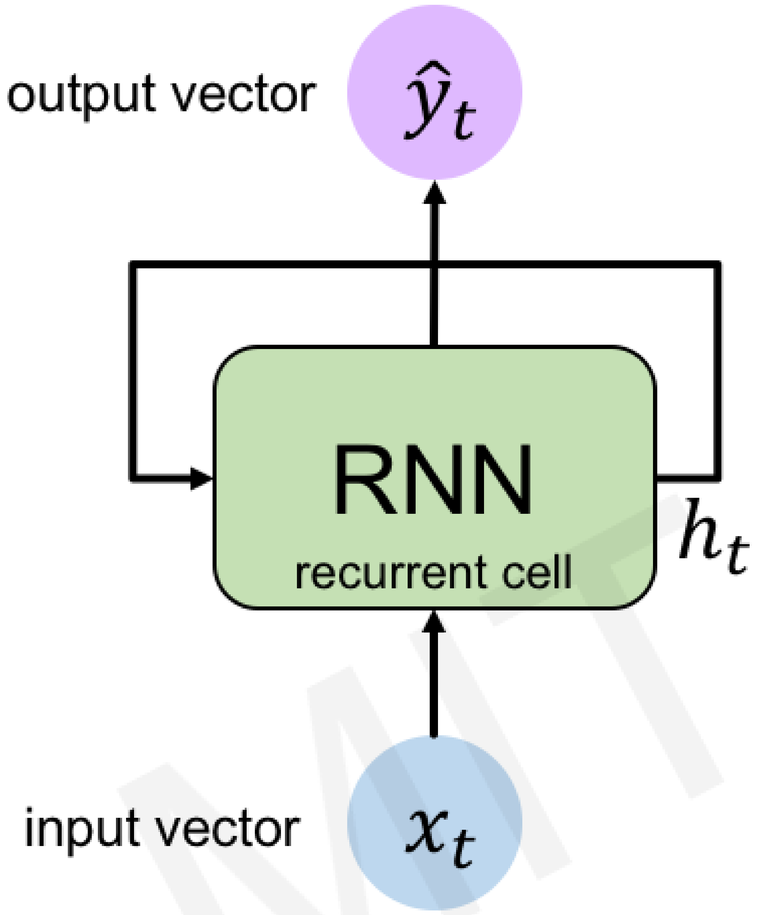
$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

Input Vector

$$x_t$$

MIT 6.S191

RNN State Update and Output



Output Vector

$$\hat{y}_t = \boldsymbol{W}_{\boldsymbol{hy}}^{\boldsymbol{T}} h_t$$

Update Hidden State

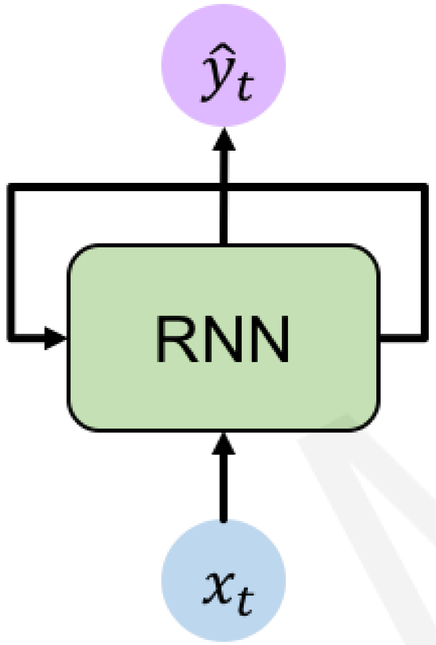
$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

Input Vector

$$x_t$$

MIT 6.S191

RNNs: Computational Graph Across Time

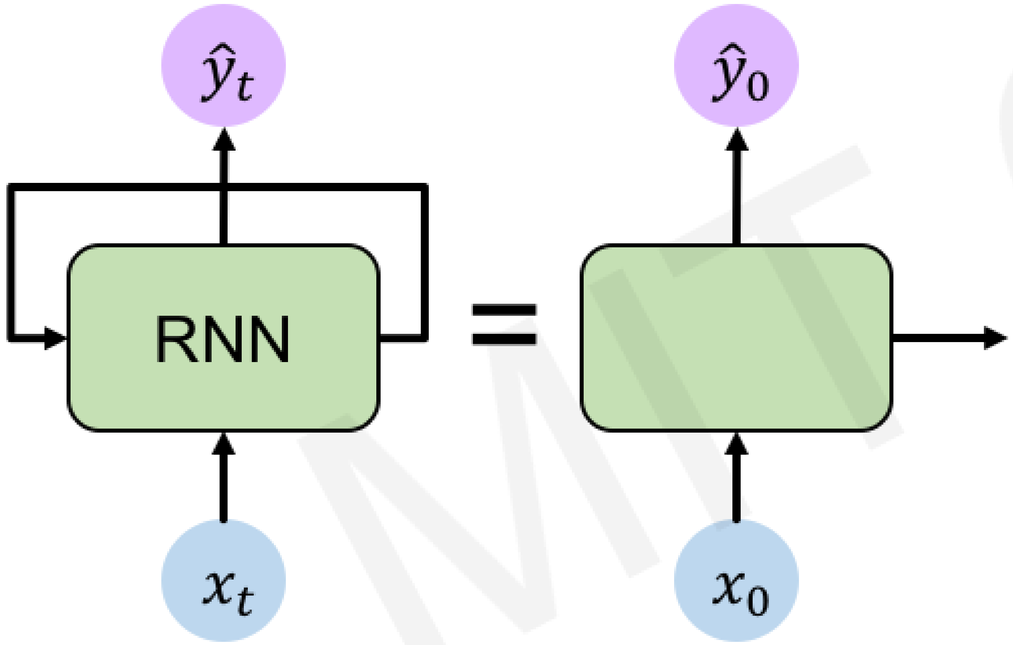


=

Represent as computational graph unrolled across time

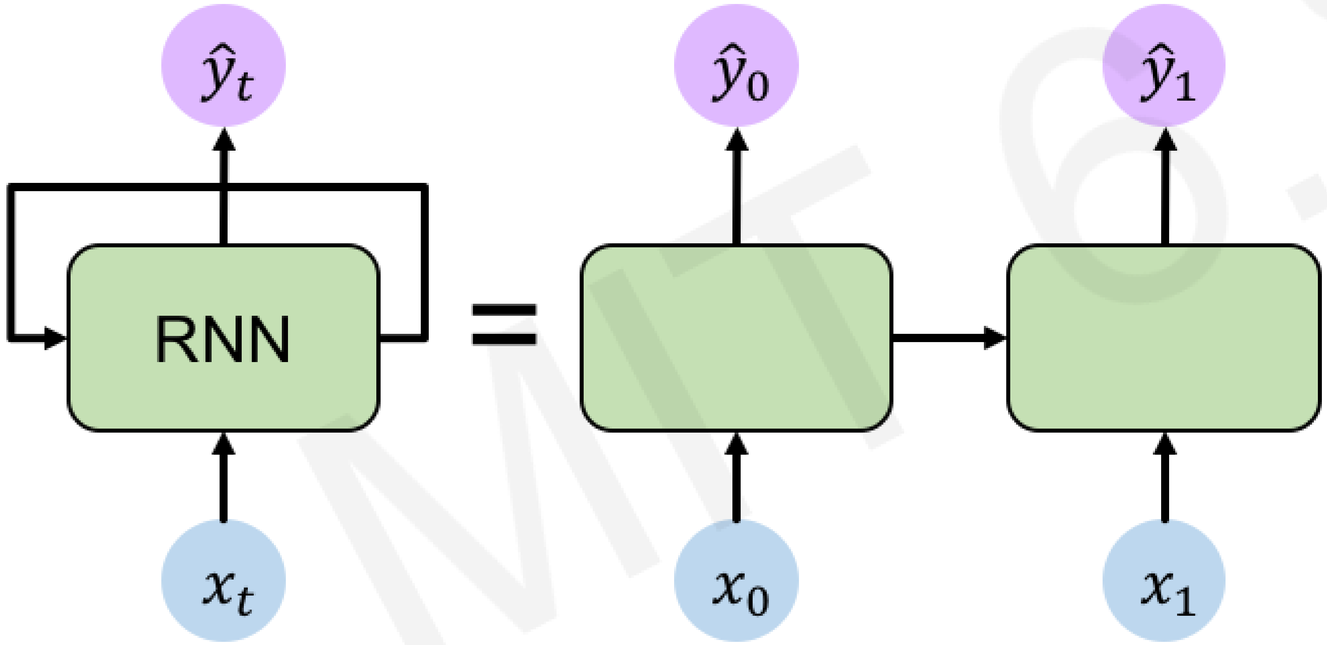
MIT 6.S191

RNNs: Computational Graph Across Time



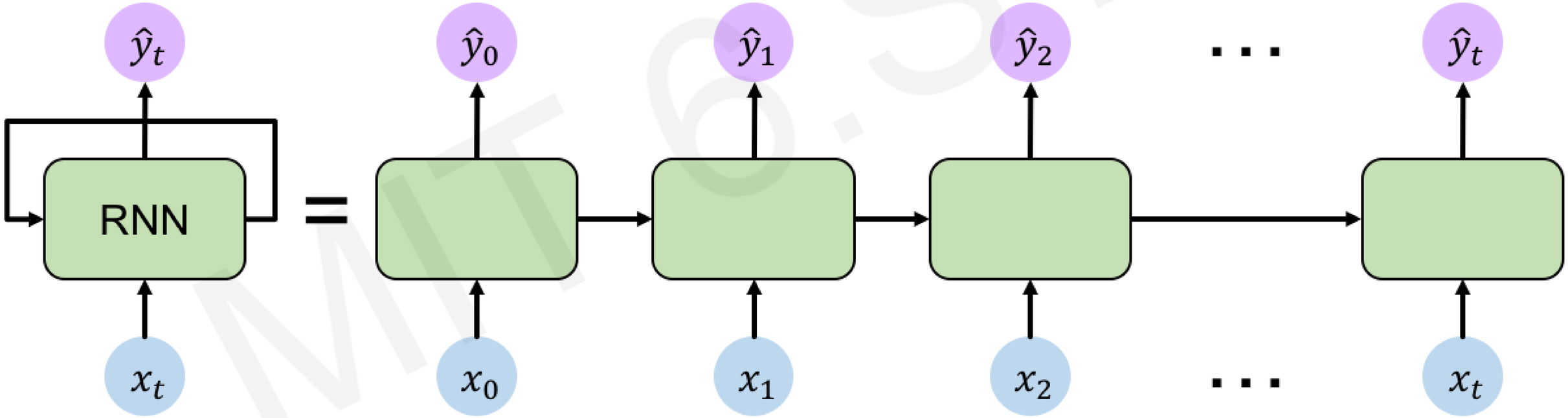
MIT 6.S191

RNNs: Computational Graph Across Time



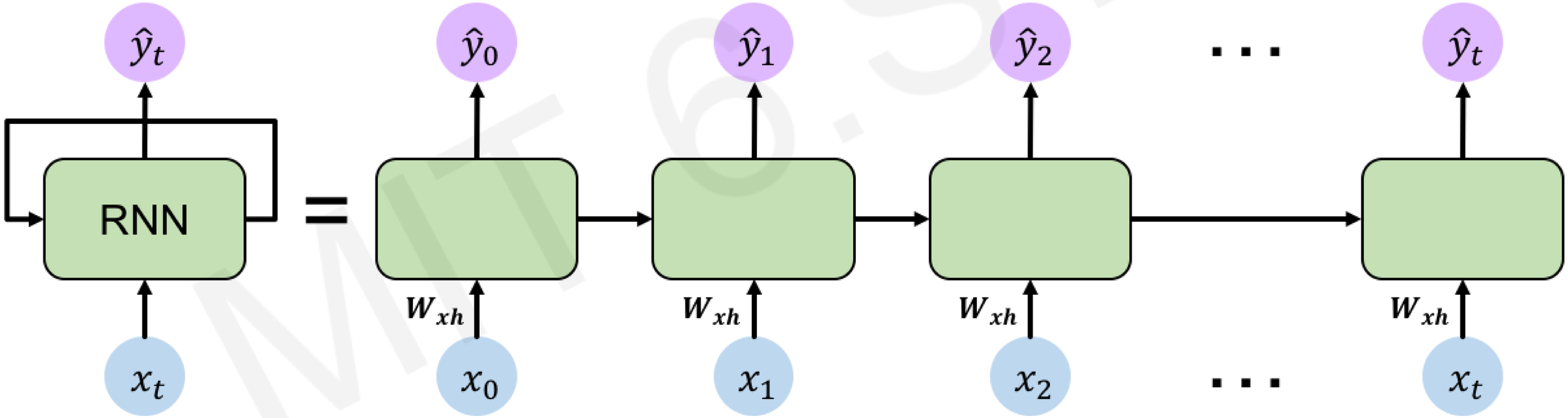
MIT 6.S191

RNNs: Computational Graph Across Time



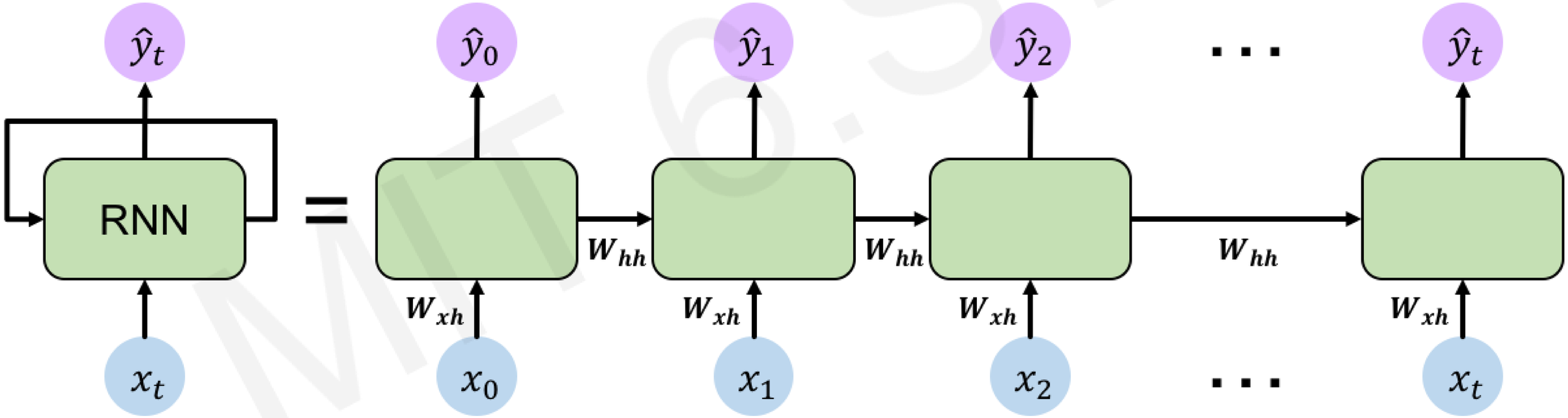
MIT 6.S191

RNNs: Computational Graph Across Time



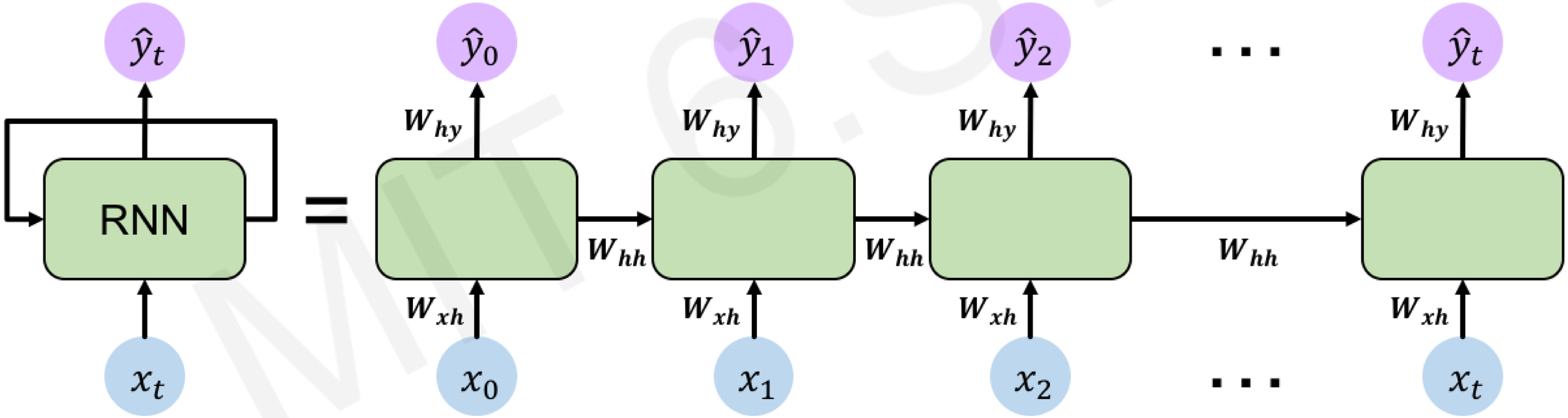
MIT 6.S191

RNNs: Computational Graph Across Time



MIT 6.S191

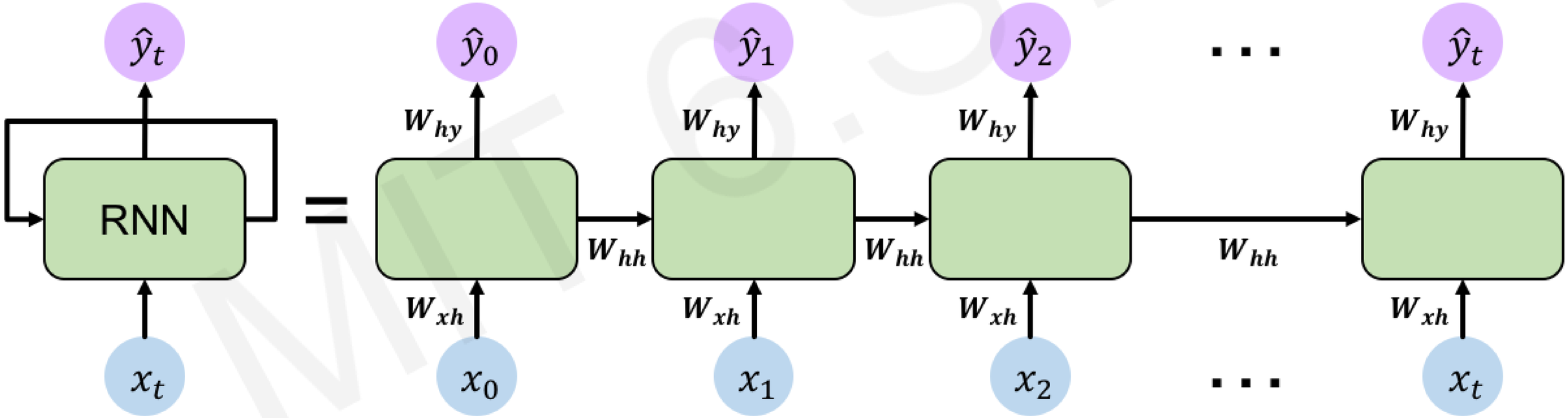
RNNs: Computational Graph Across Time



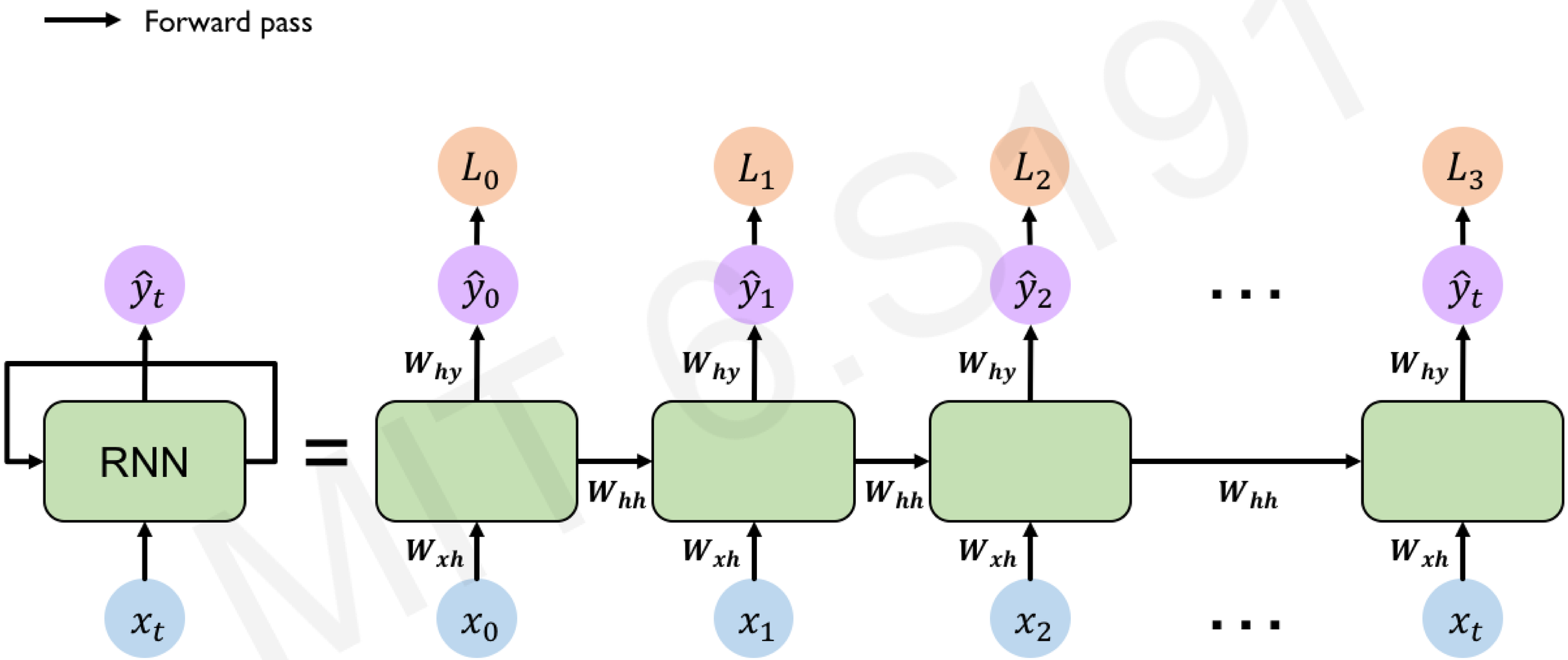
MIT 6.S191

RNNs: Computational Graph Across Time

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

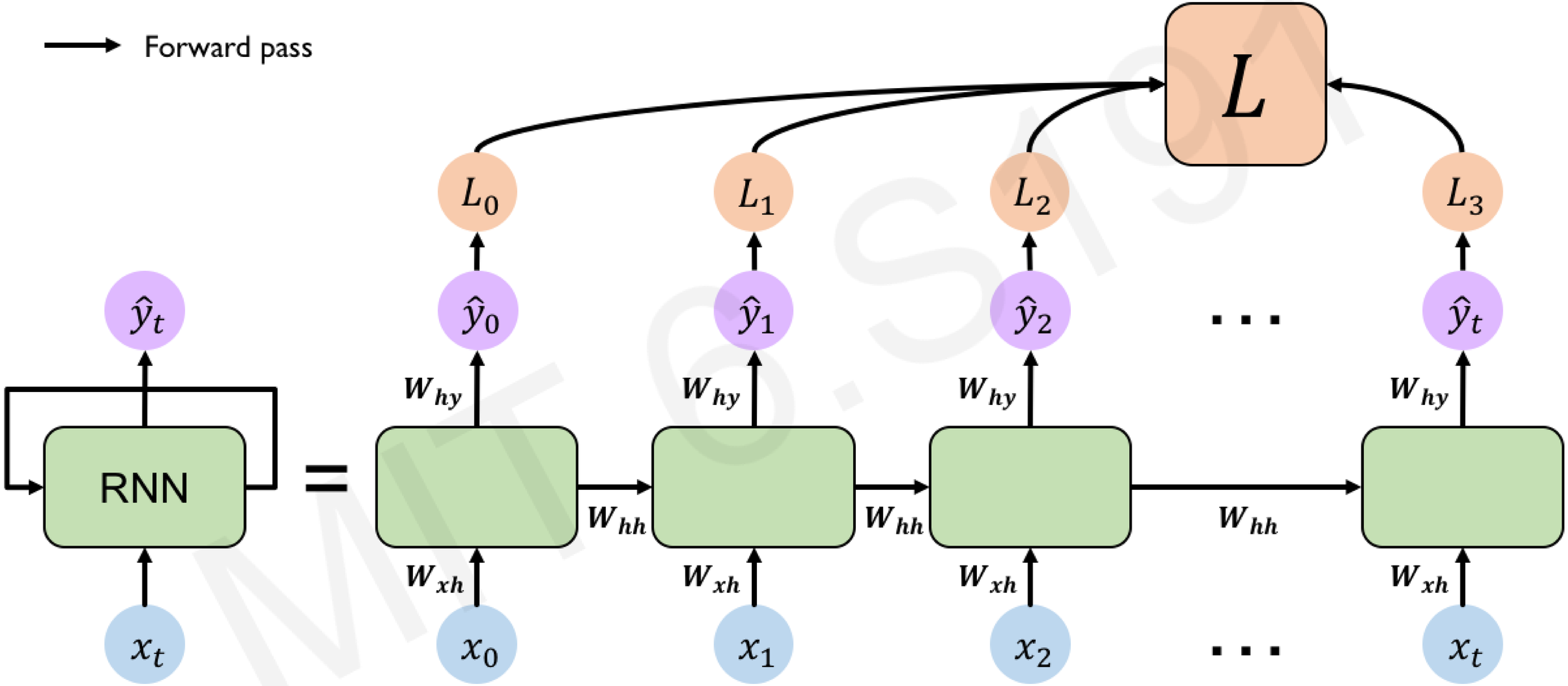


RNNs: Computational Graph Across Time



MIT 6.S191

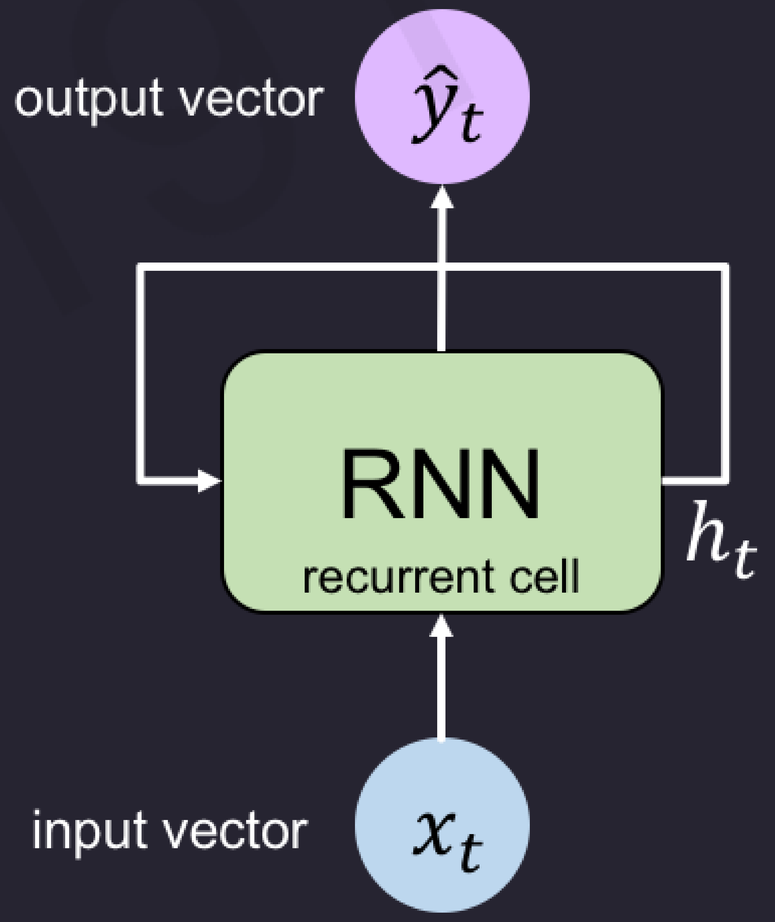
RNNs: Computational Graph Across Time



MIT 6.S191

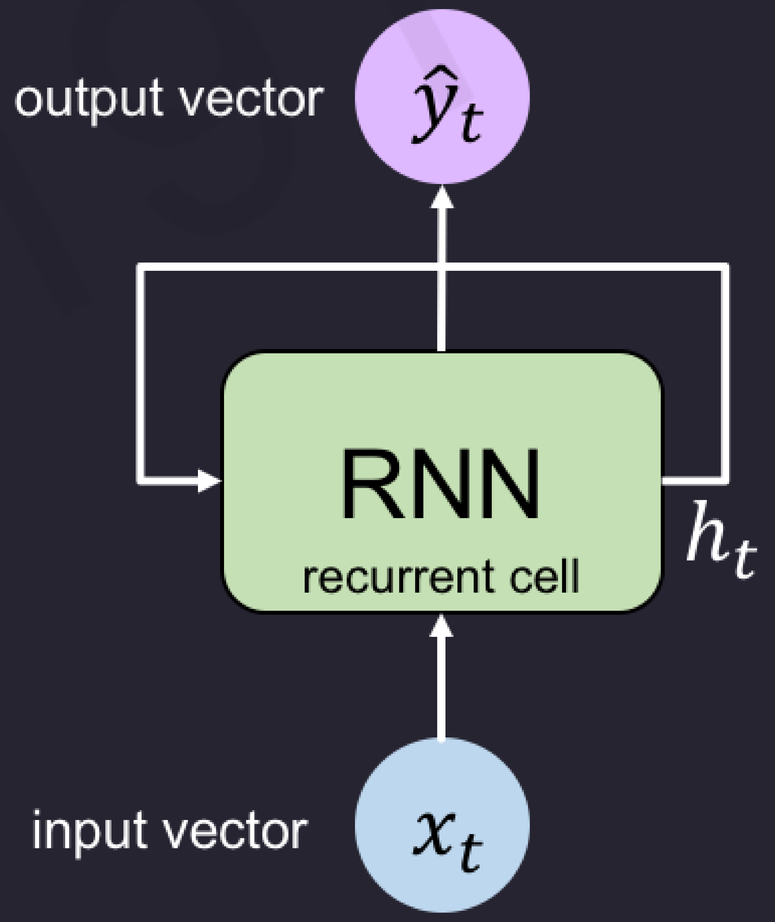
RNNs from Scratch

```python
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
```



RNN Implementation in TensorFlow

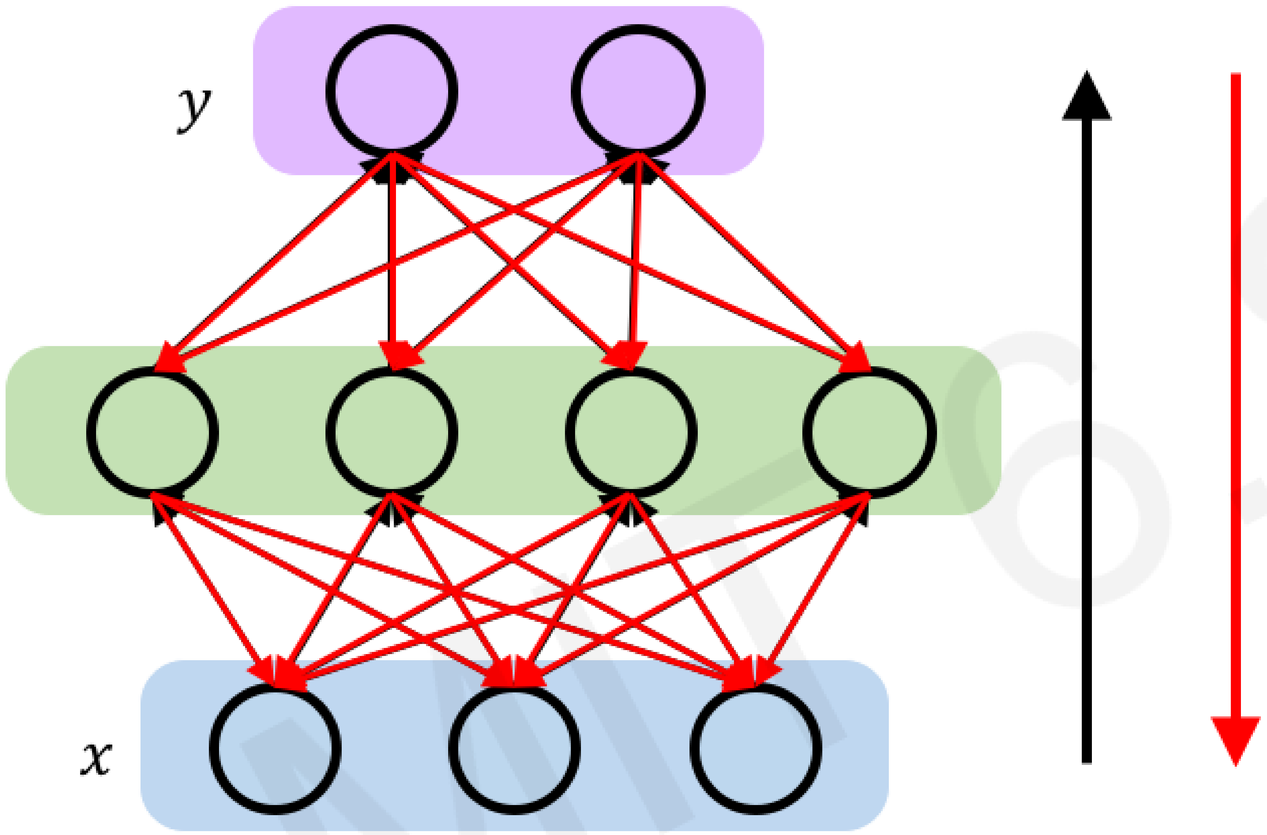
```
tf.keras.layers.SimpleRNN(rnn_units)
```



Backpropagation Through Time (BPTT)

MIT 6.S191

Recall: Backpropagation in Feed Forward Models

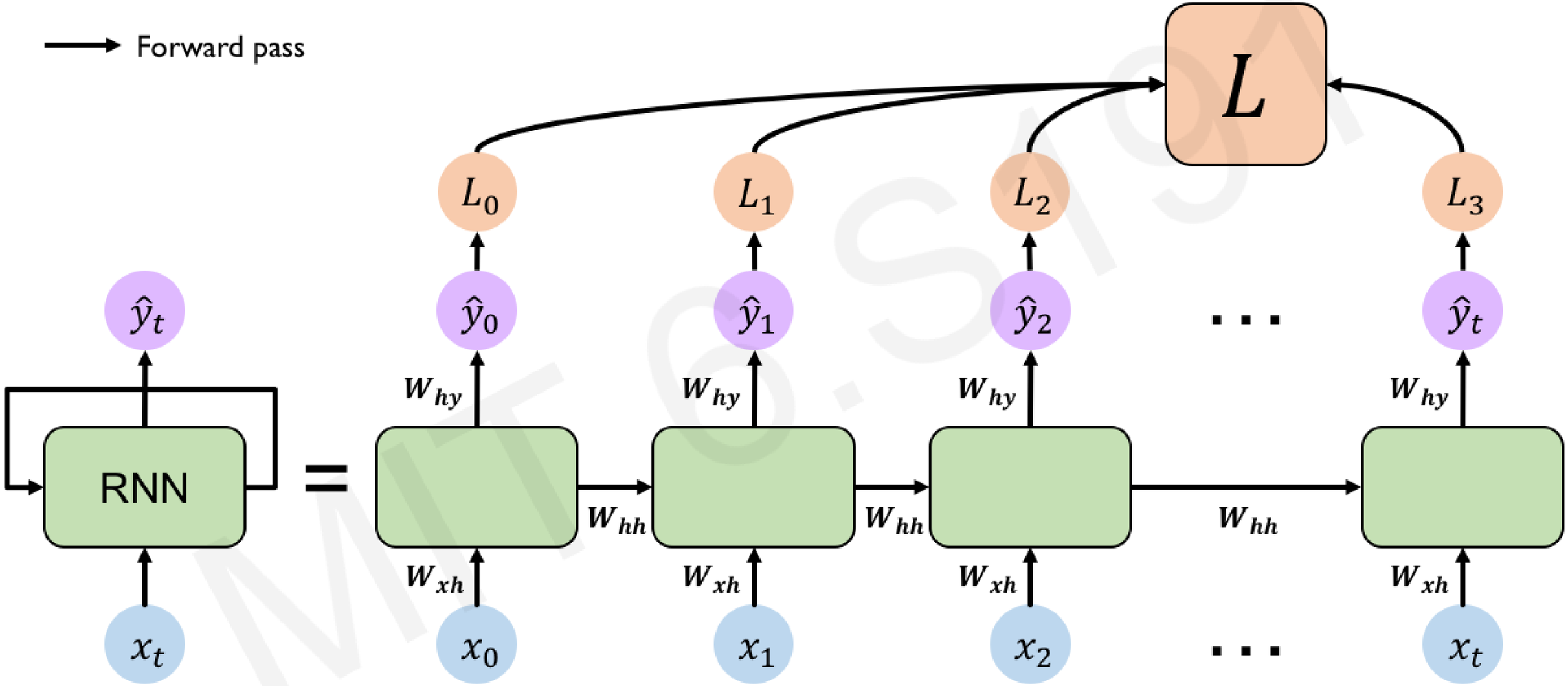


Backpropagation algorithm:

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

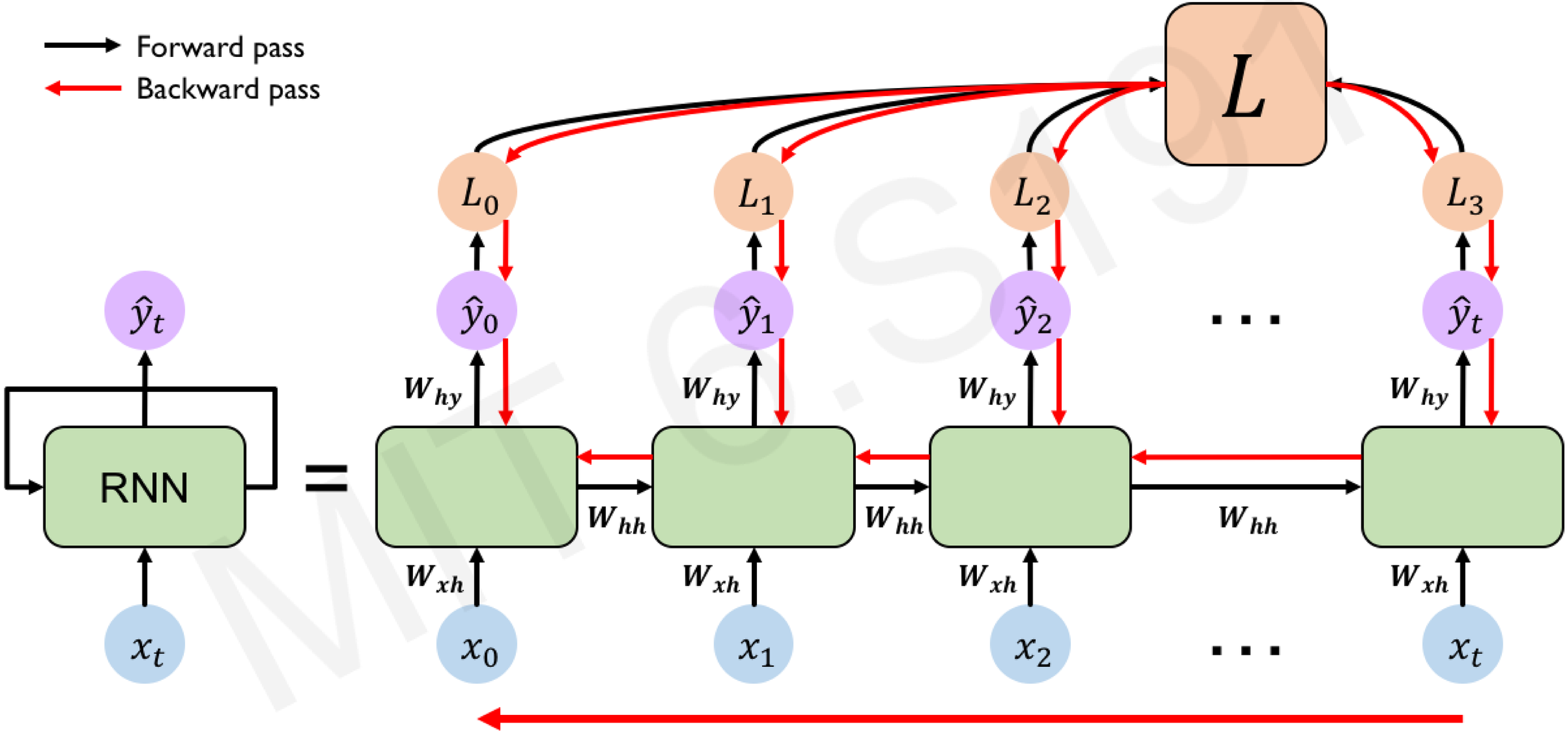
MIT 6.S191

RNNs: Backpropagation Through Time

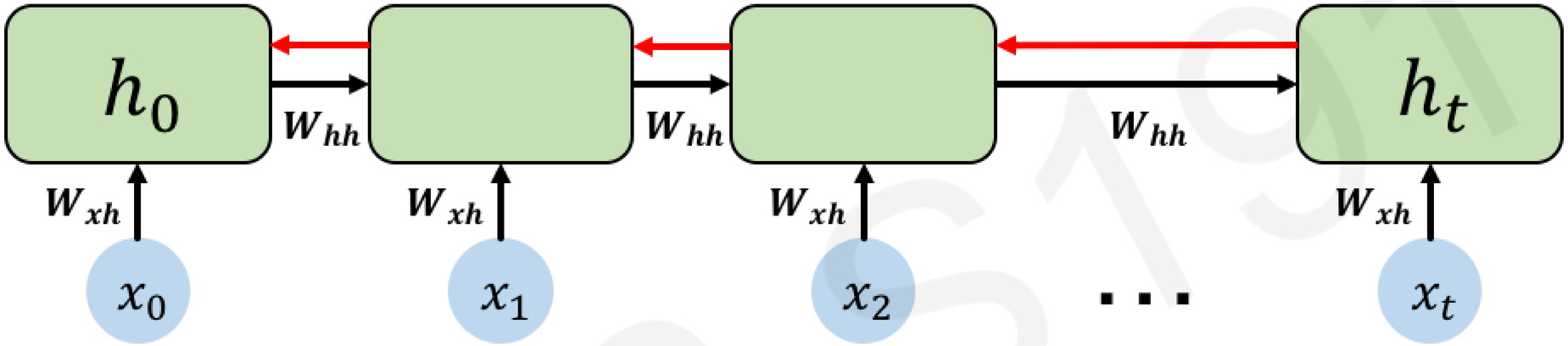


MIT 6.S191

RNNs: Backpropagation Through Time

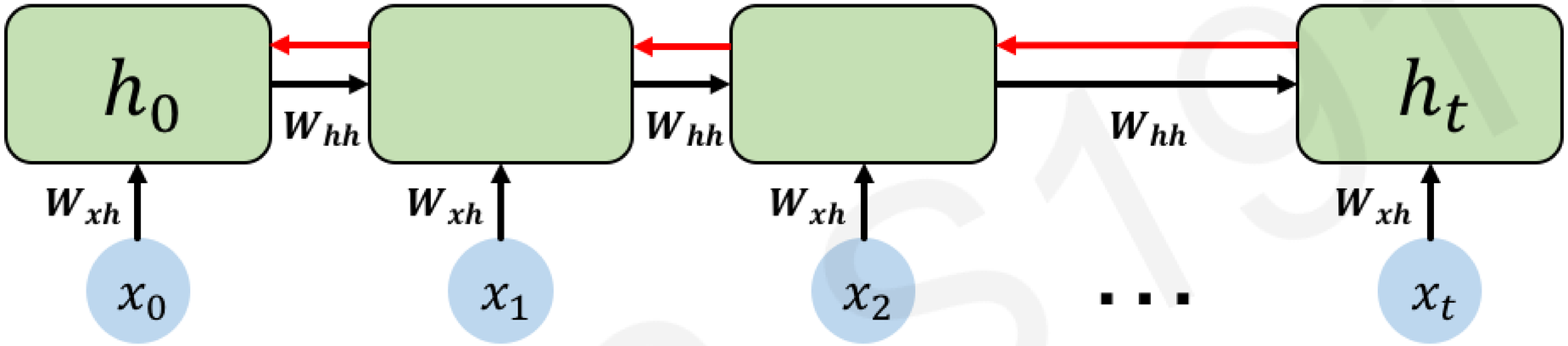


Standard RNN Gradient Flow



MIT 6.S191

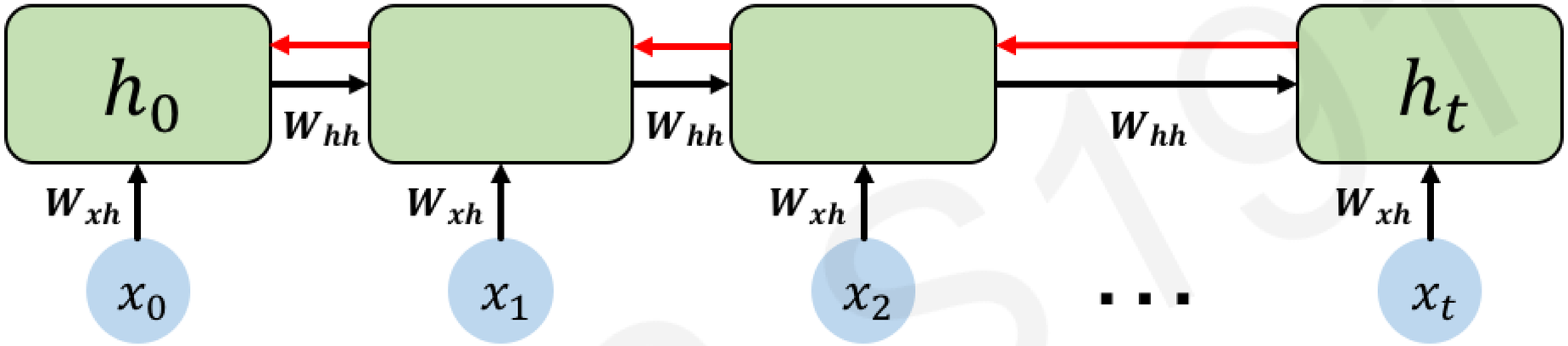
Standard RNN Gradient Flow



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

MIT 6.S191

Standard RNN Gradient Flow: Exploding Gradients



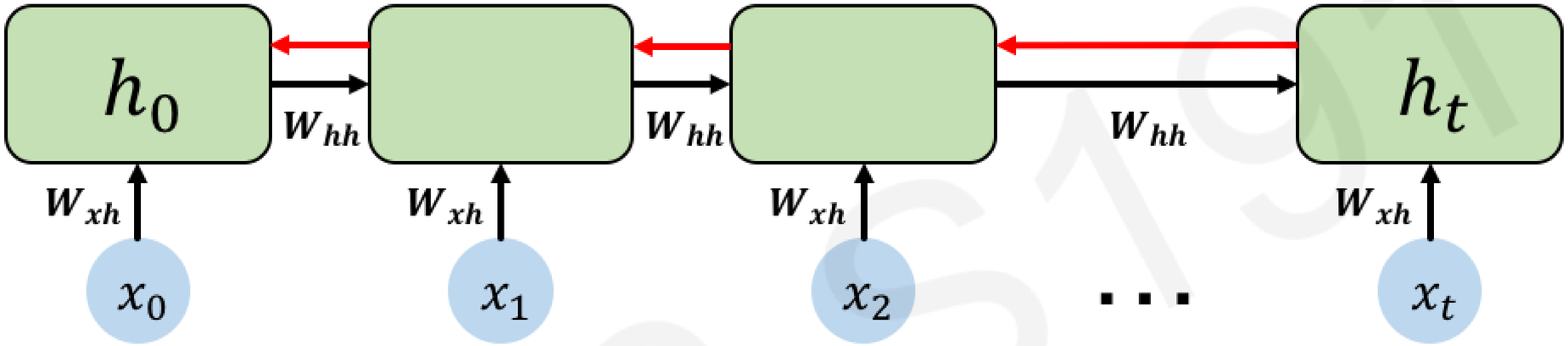
Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation**!

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

MIT 6.S191

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

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

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

MIT 6.S191

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

MIT 6.S191

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

MIT 6.S191

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

MIT 6.S191

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

MIT 6.S191

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

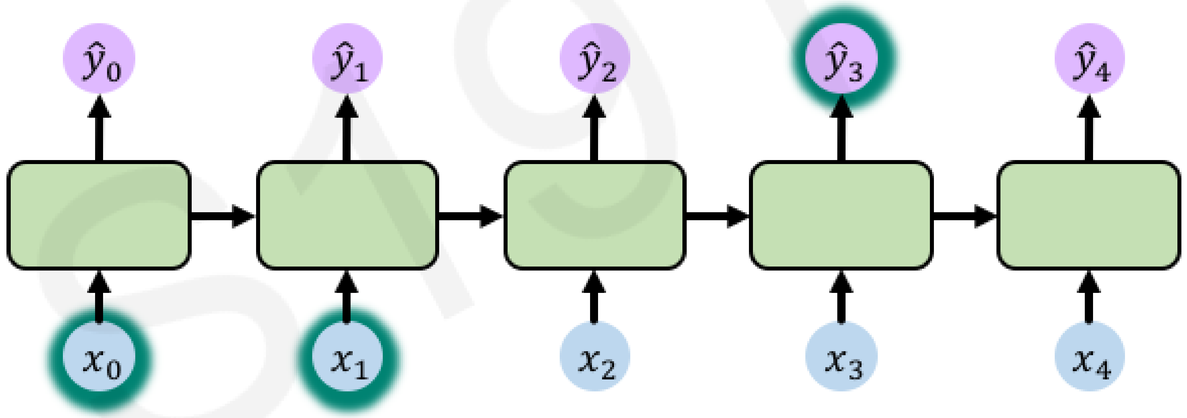
↓

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

↓

Bias parameters to capture short-term dependencies

"The clouds are in the ___"



MIT 6.S191

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

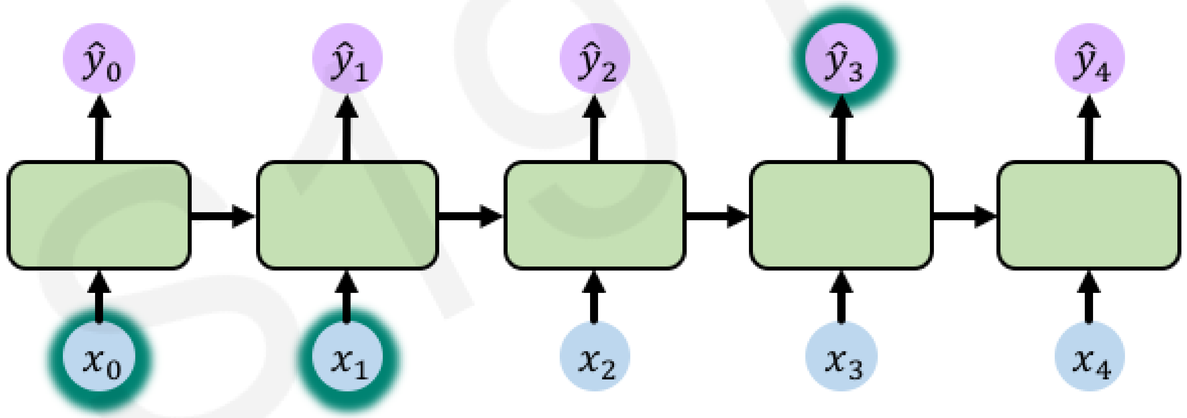
↓

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___ "

MIT 6.S191

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

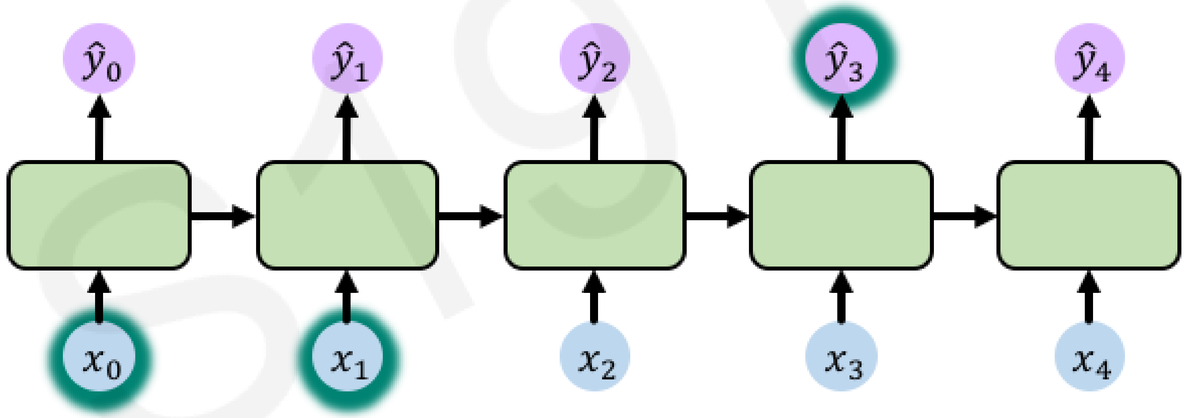
↓

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

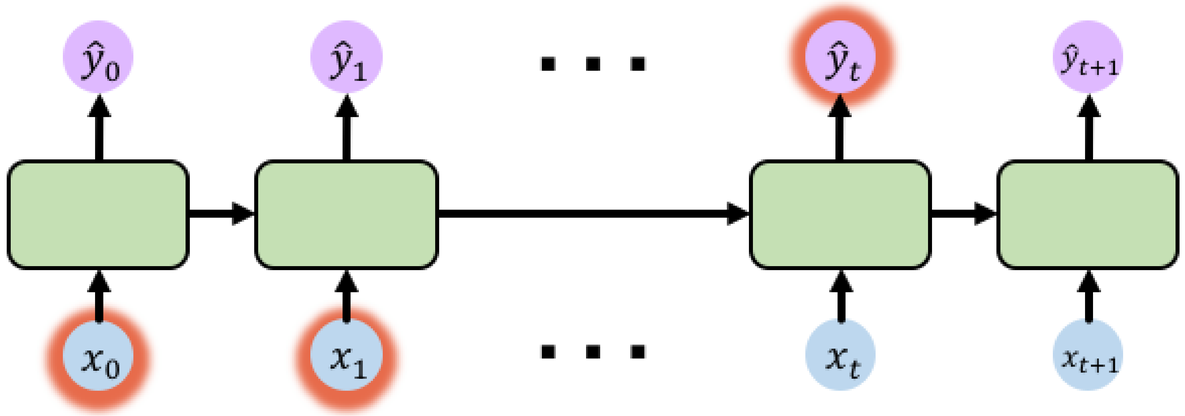
↓

Bias parameters to capture short-term dependencies

"The clouds are in the ___"

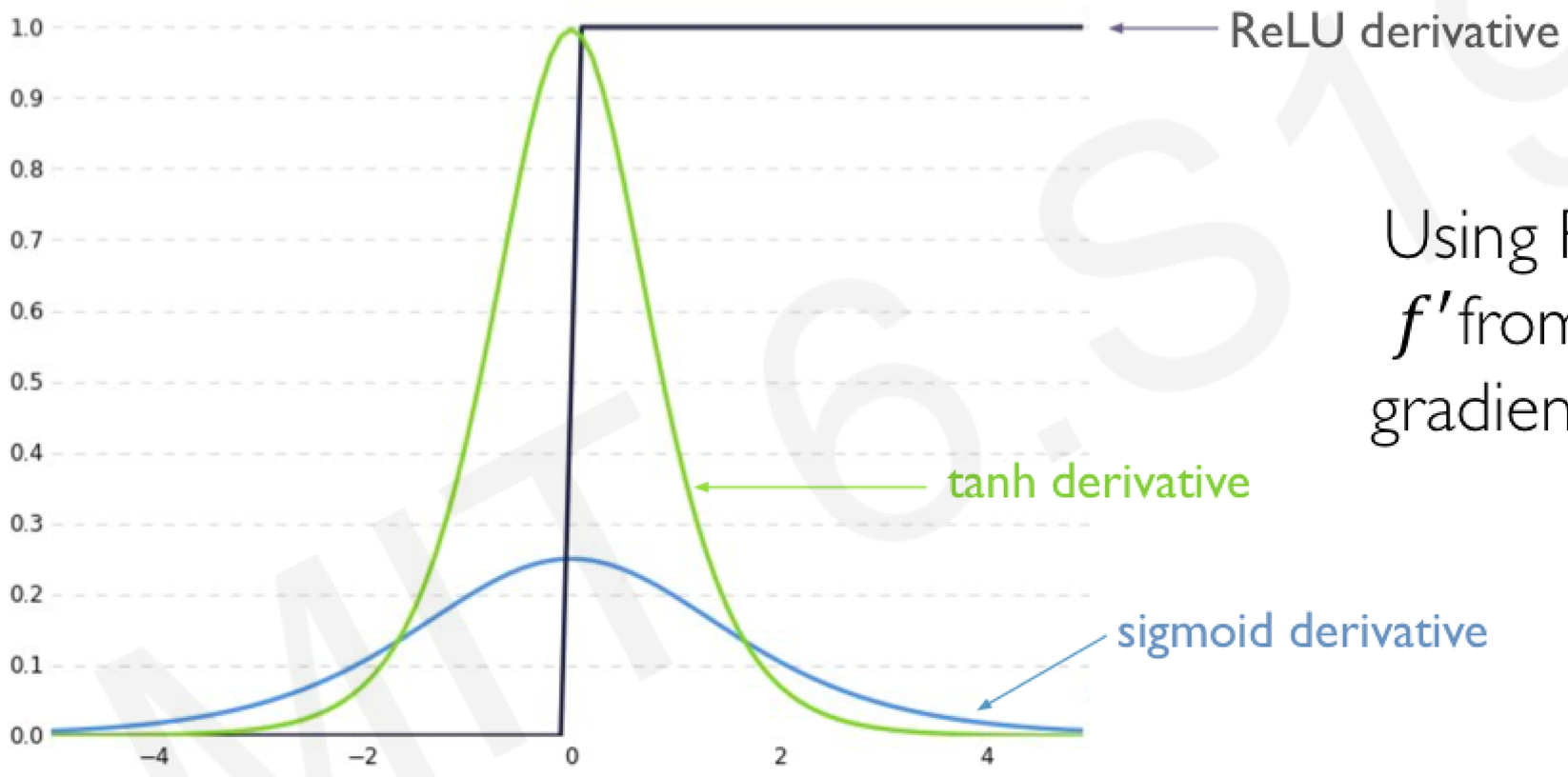


"I grew up in France, … and I speak fluent___ "



MIT 6.S191

Trick #1: Activation Functions



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

MIT 6.S191

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.

MIT 6.S191

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 (LSTMs) networks rely on a gated cell to track information throughout many time steps.

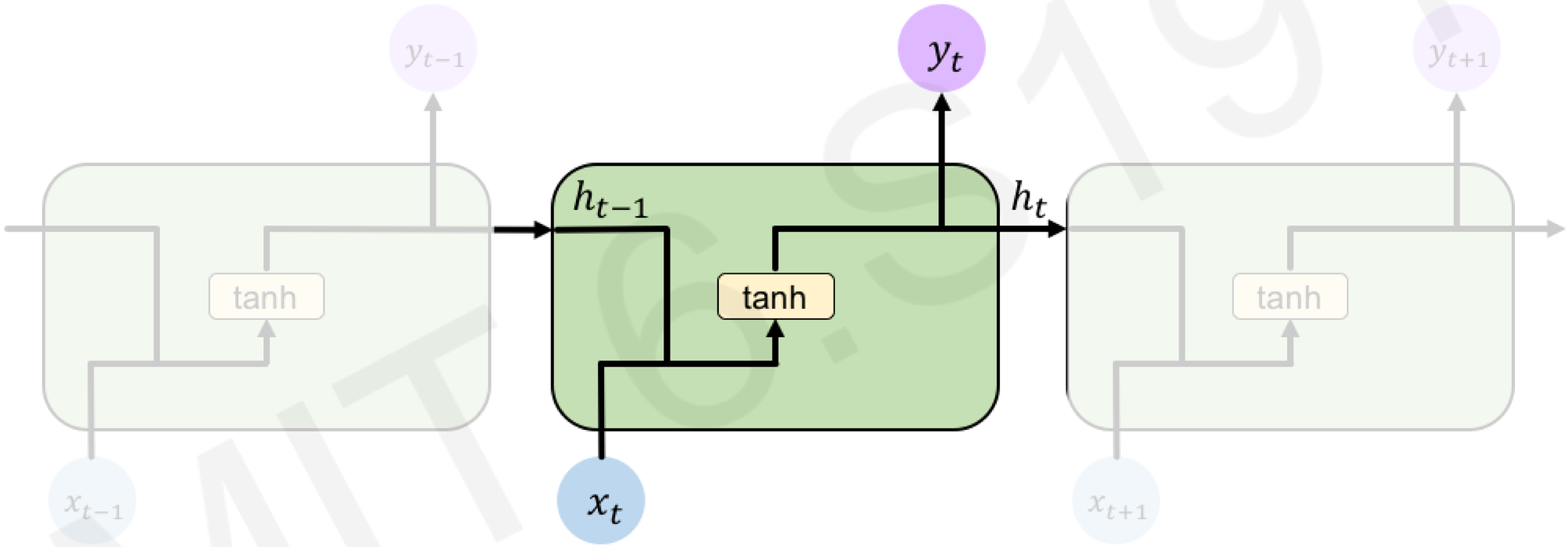
MIT 6.S191

Long Short Term Memory (LSTM) Networks

MIT 6.S191

Standard RNN

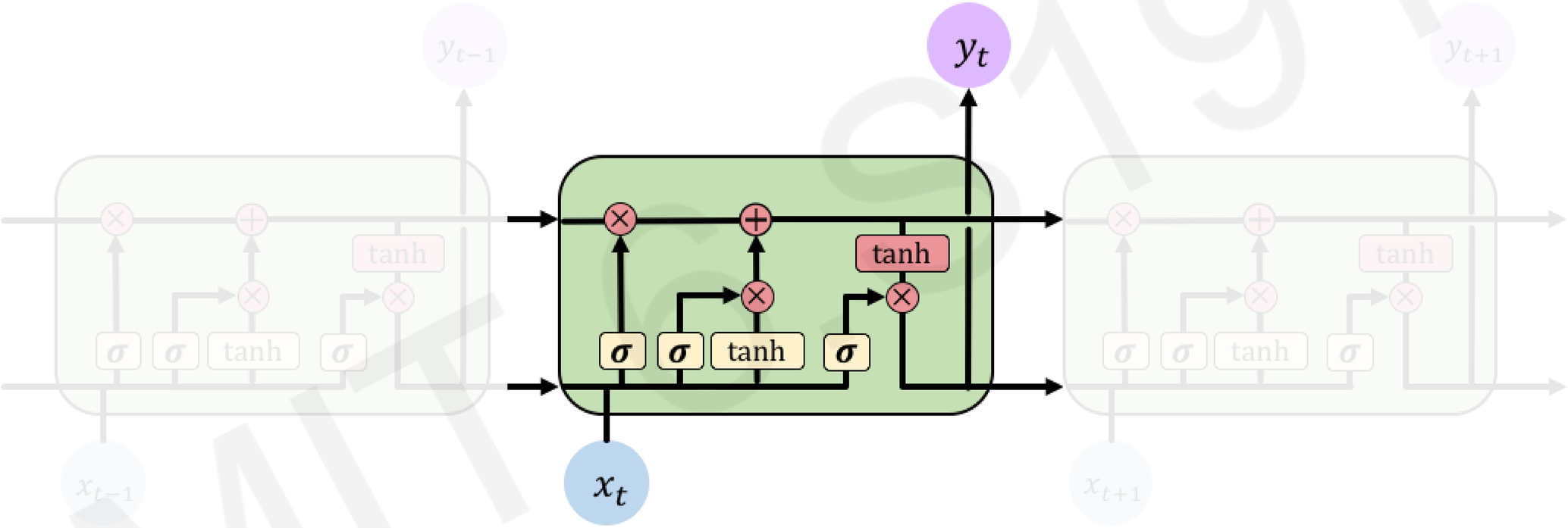
In a standard RNN, repeating modules contain a **simple computation node**



MIT 6.S191

Long Short Term Memory (LSTMs)

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



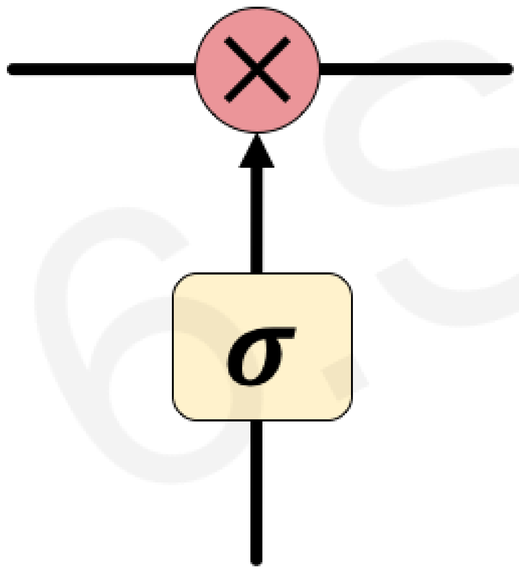
LSTM cells are able to track information throughout many timesteps

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


Hochreiter & Schmidhuber, *Neural Computation* 1997.

Long Short Term Memory (LSTMs)

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



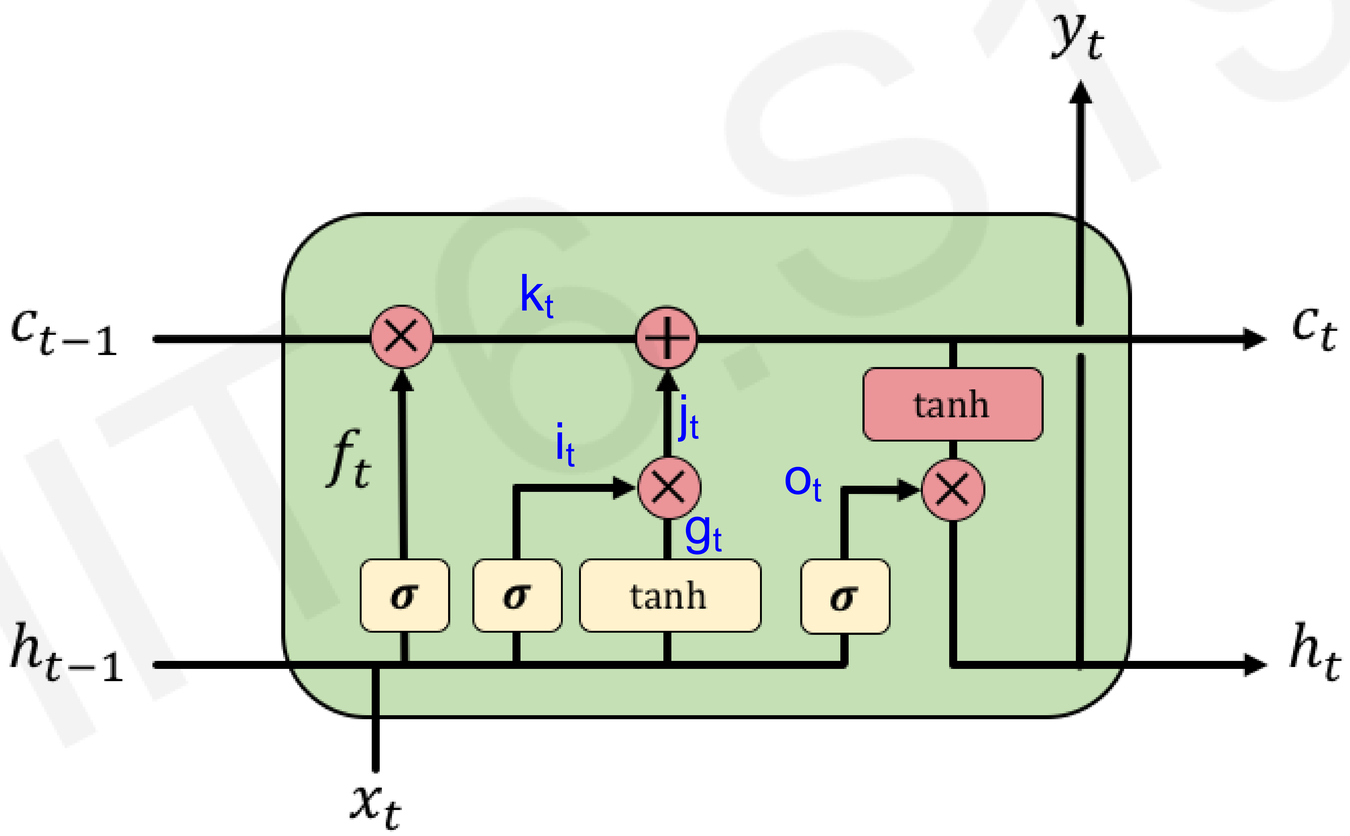
Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

MIT 6.S191

Long Short Term Memory (LSTMs)

How do LSTMs work?

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



$C_t = f_t * C_{t-1} + i_t * g_t$

$h_t = o_t * \tanh(C_t)$

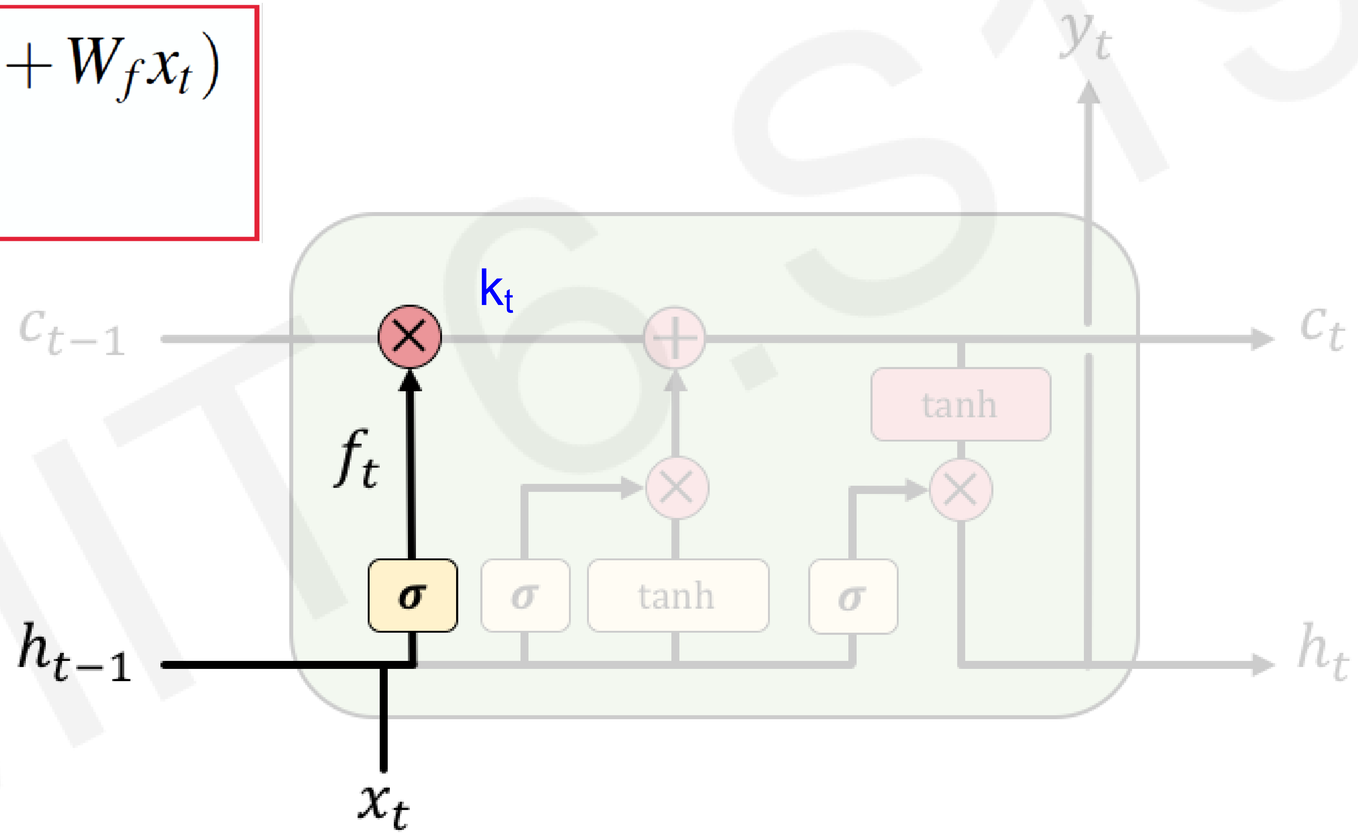
MIT 6.S191

Long Short Term Memory (LSTMs)

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

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

$$
\begin{aligned}
f_t &= \sigma(U_f h_{t-1} + W_f x_t) \\
k_t &= c_{t-1} \odot f_t
\end{aligned}
$$



MIT 6.S191

Long Short Term Memory (LSTMs)

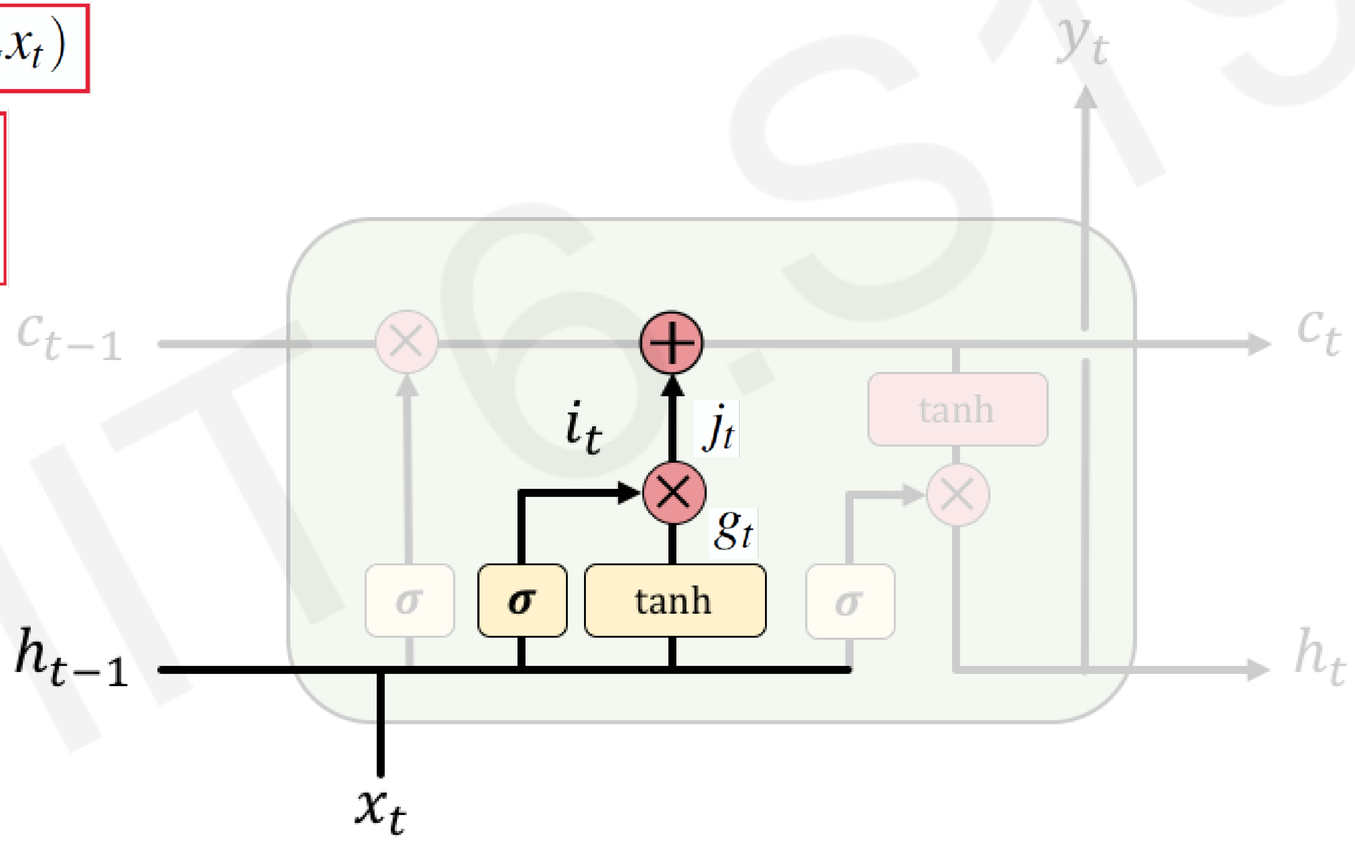
1) Forget **2) Store** 3) Update 4) Output

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

$$g_t = tanh(U_g h_{t-1} + W_g x_t)$$

$$i_t = \sigma(U_i h_{t-1} + W_i x_t)$$

$$j_t = g_t \odot i_t$$

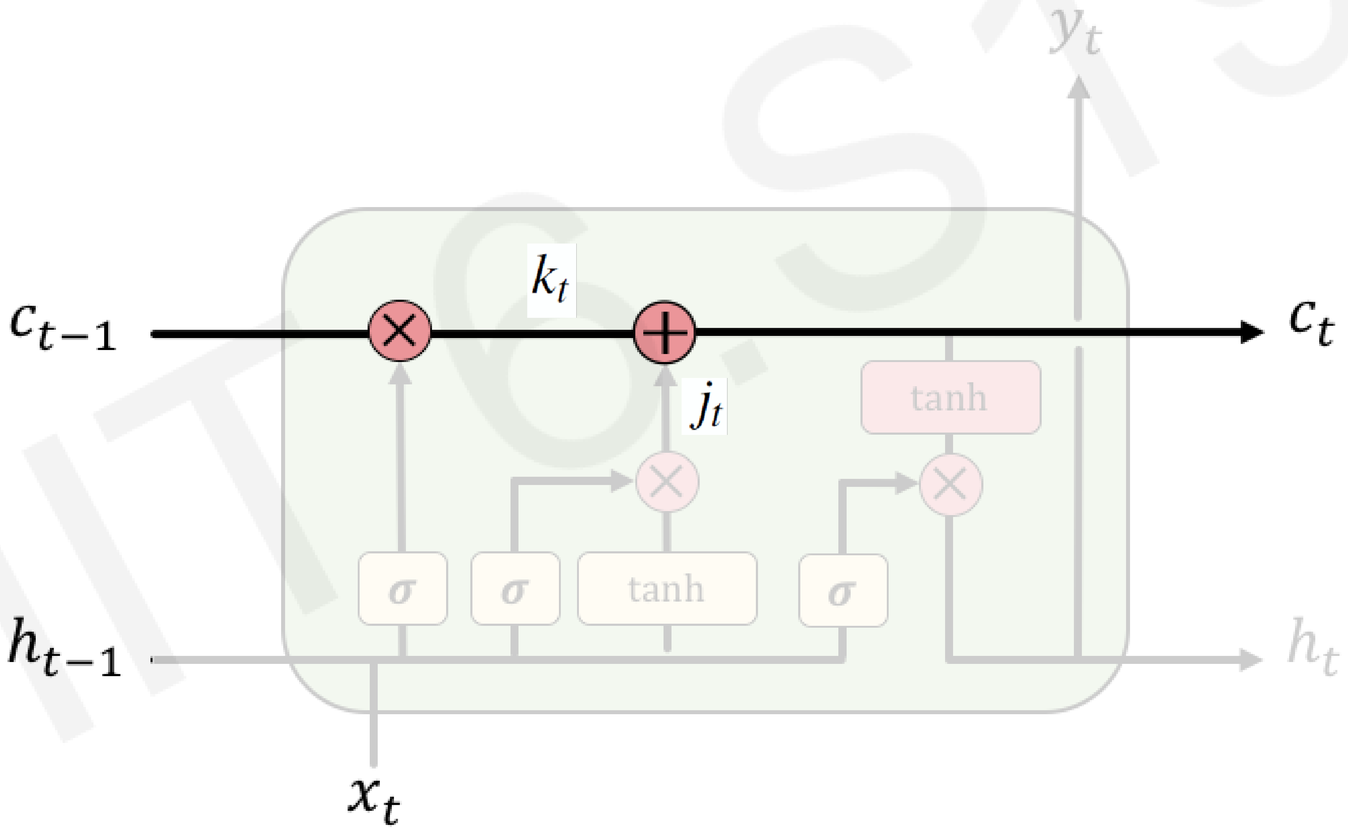


MIT 6.S191

Long Short Term Memory (LSTMs)

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

LSTMs **selectively update** cell state values



$C_t = k_t + j_t$

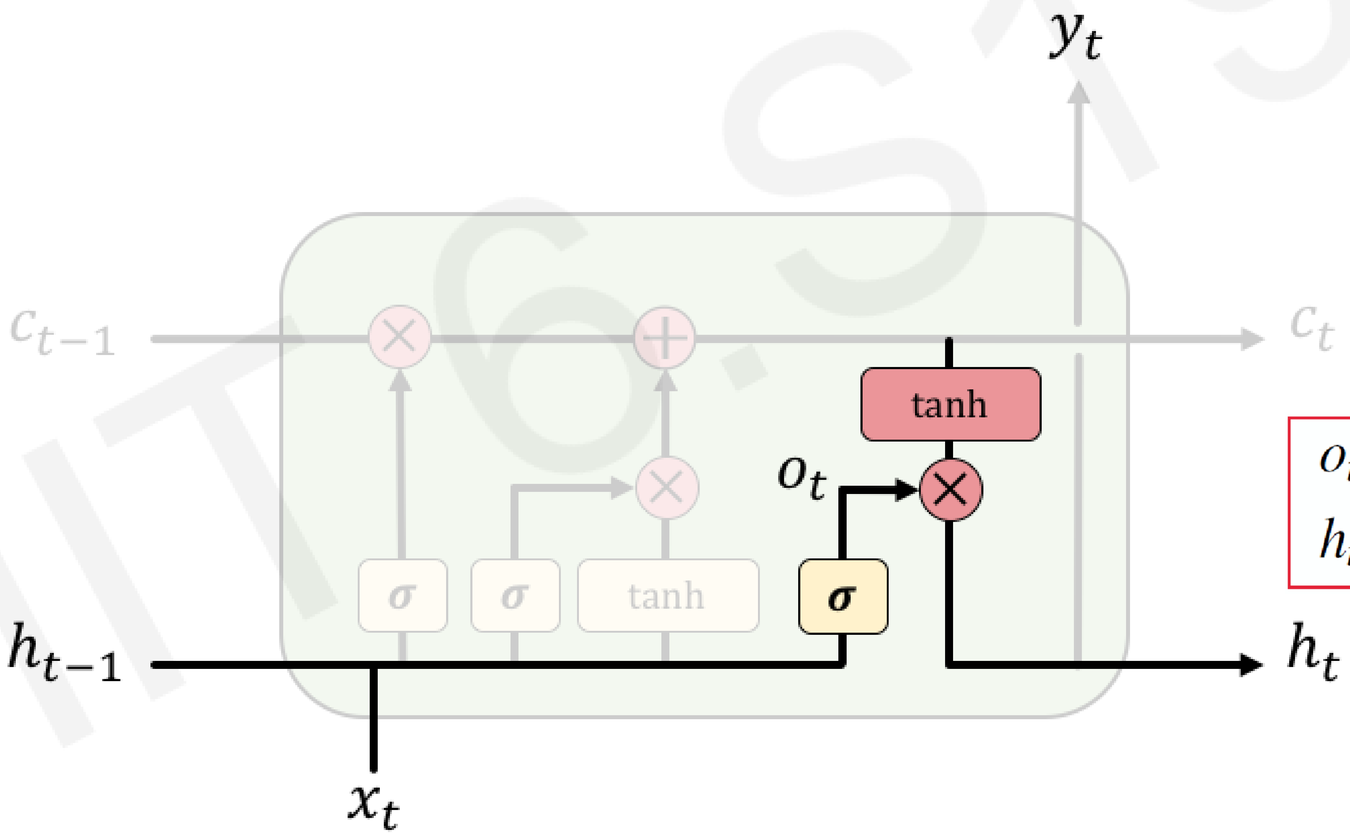
$C_t = f_t * C_{t-1} + i_t * g_t$

MIT 6.S191

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

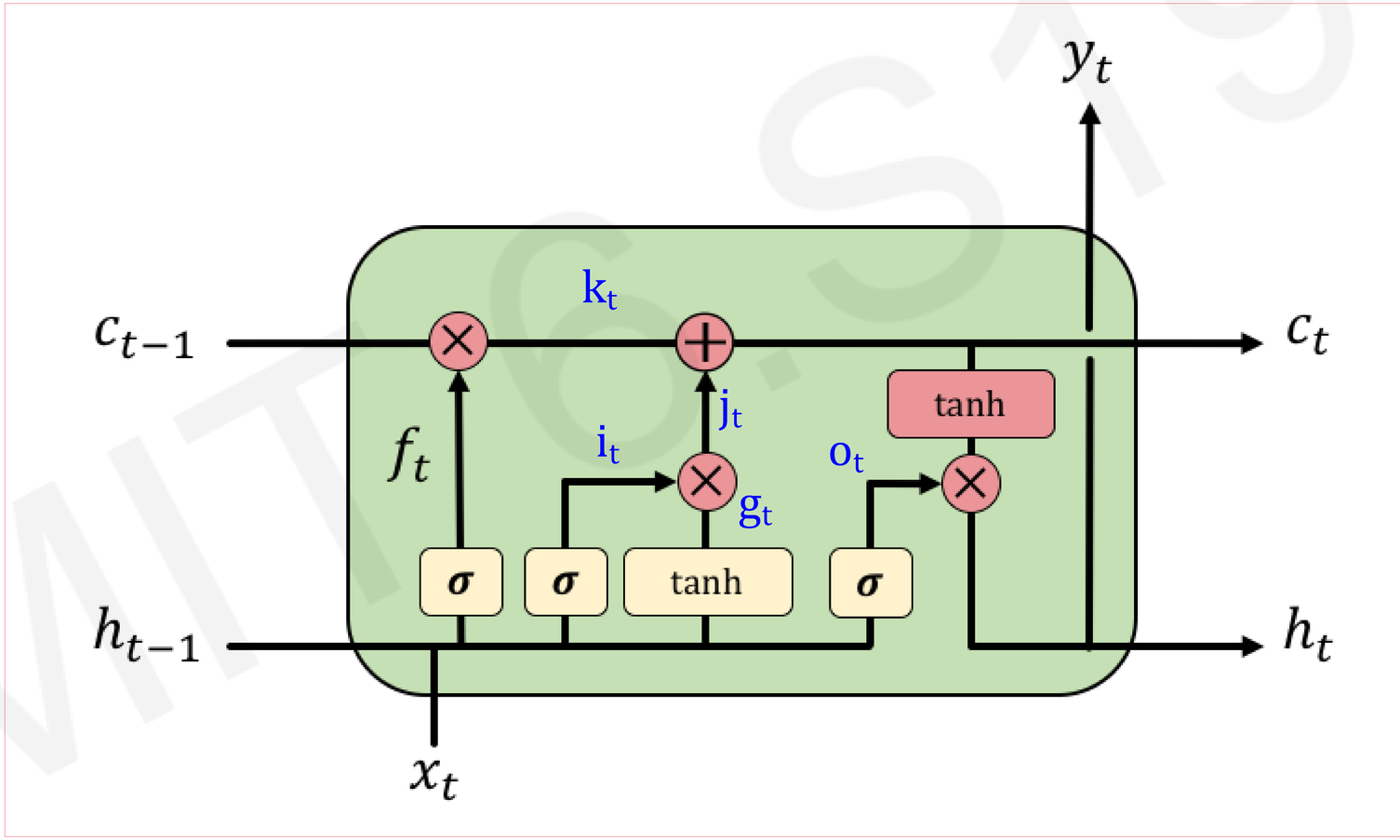


$$o_t = \sigma(U_o h_{t-1} + W_o x_t)$$
$$h_t = o_t \odot tanh(c_t)$$

MIT 6.S191

Long Short Term Memory (LSTMs)

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



$C_t = k_t + j_t$

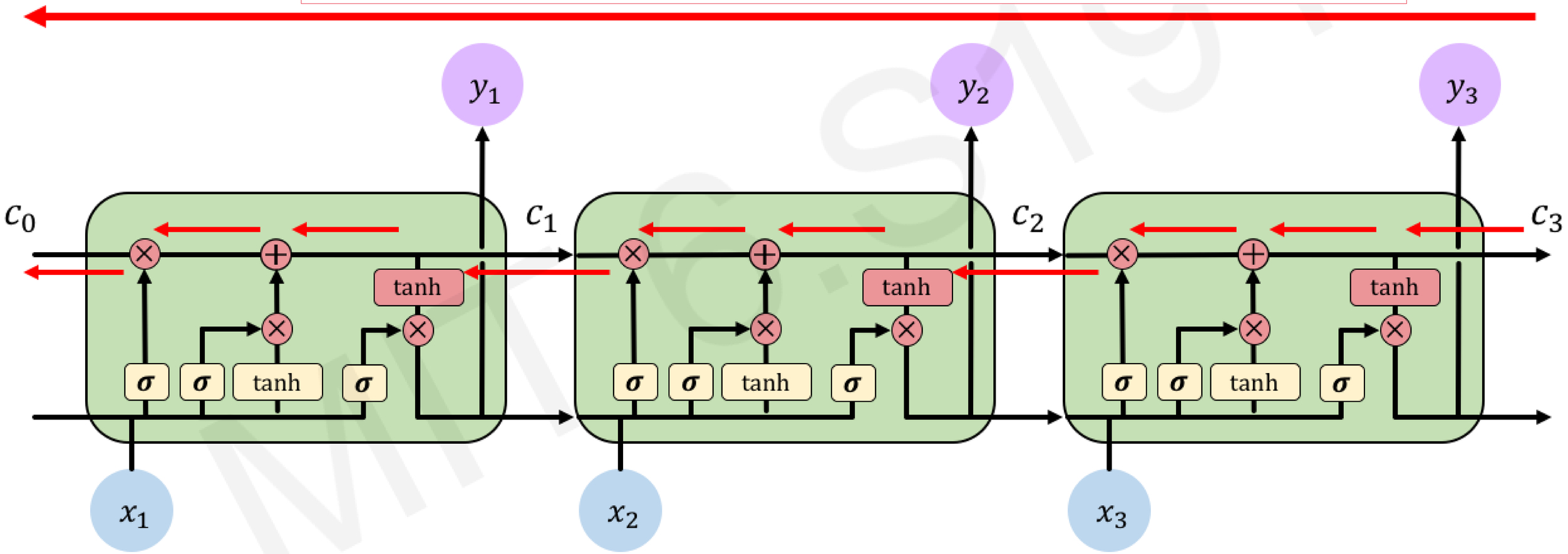
$C_t = f_t * C_{t-1} + i_t * g_t$

$$o_t = \sigma(U_o h_{t-1} + W_o x_t)$$
$$h_t = o_t \odot tanh(c_t)$$

MIT 6.S191

LSTM Gradient Flow

Back propagation from C_t to C_{t-1} requires only elementwise multiplication!
No matrix multiplication; avoid vanishing gradient problem.
Uninterrupted gradient flow!



LSTMs: Key Concepts

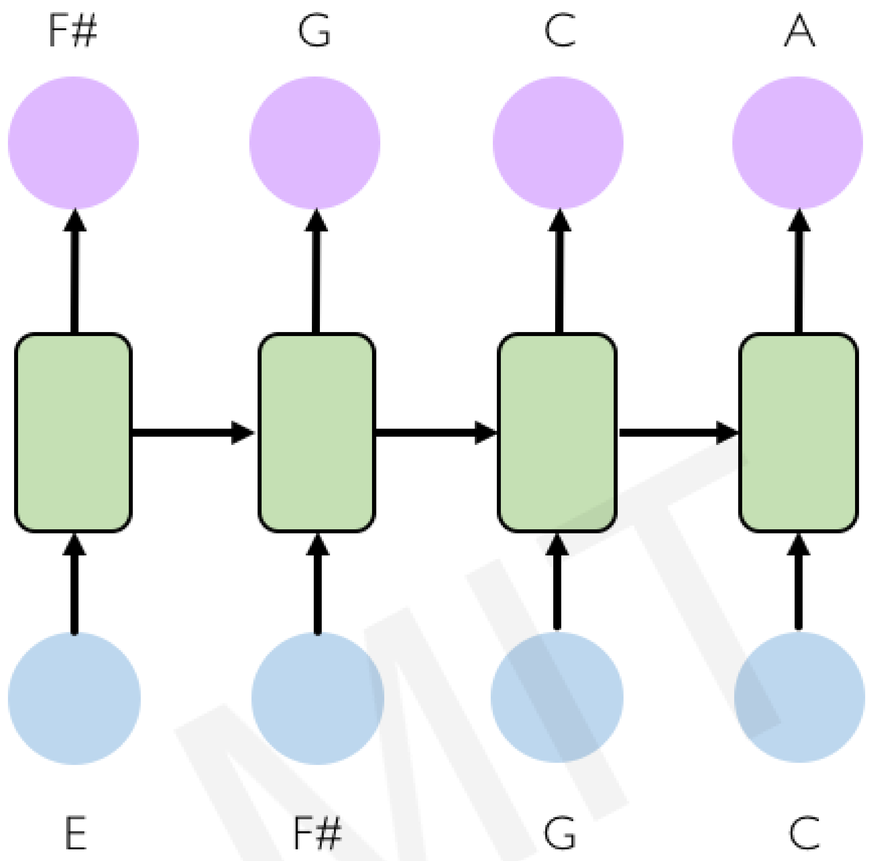
1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered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

MIT 6.S191

RNN Applications

MIT 6.S191

Example Task: Music Generation



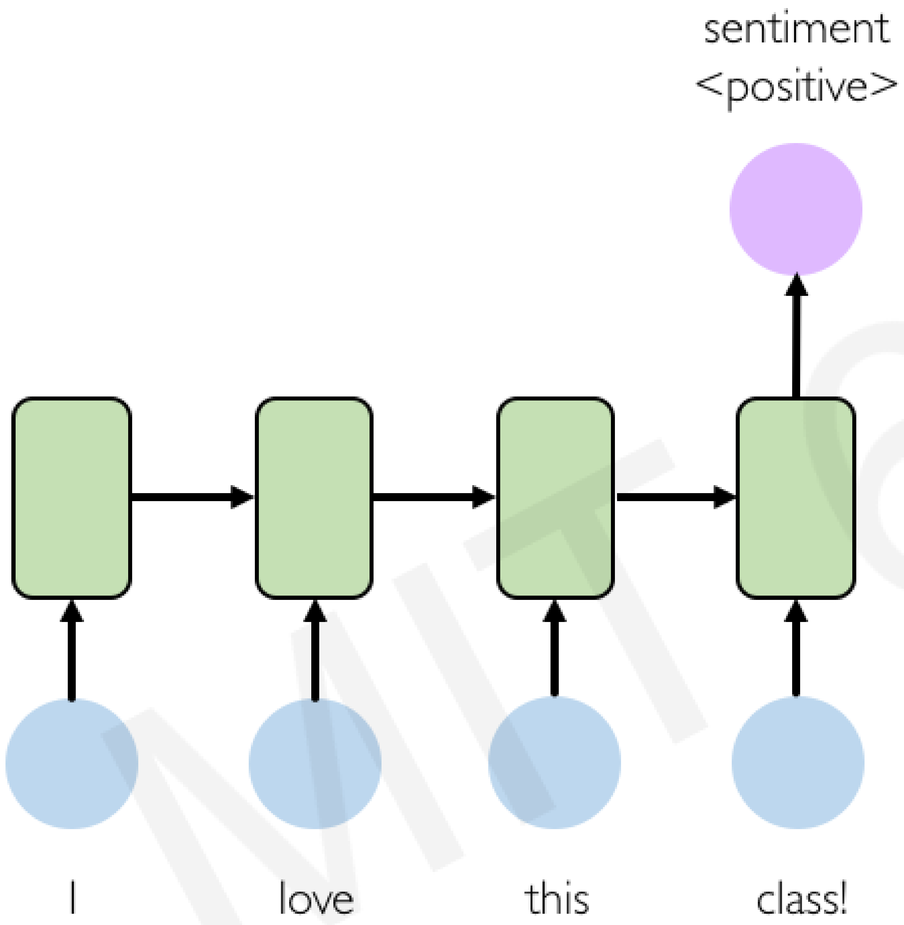
Input: sheet music

Output: next character in sheet music

6.S191 Lab!

MIT 6.S191

Example Task: Sentiment Classification



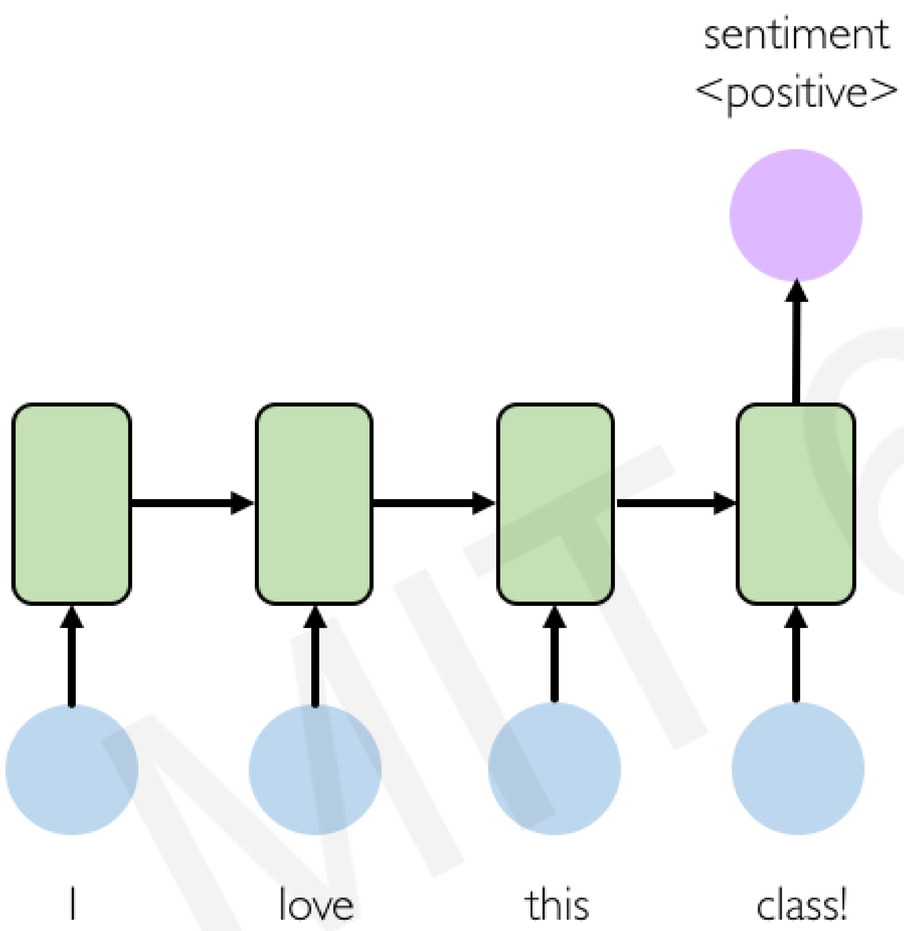
Input: sequence of words

Output: probability of having positive sentiment

```
loss = tf.nn.softmax_cross_entropy_with_logits(y, predicted)
```

MIT 6.S191

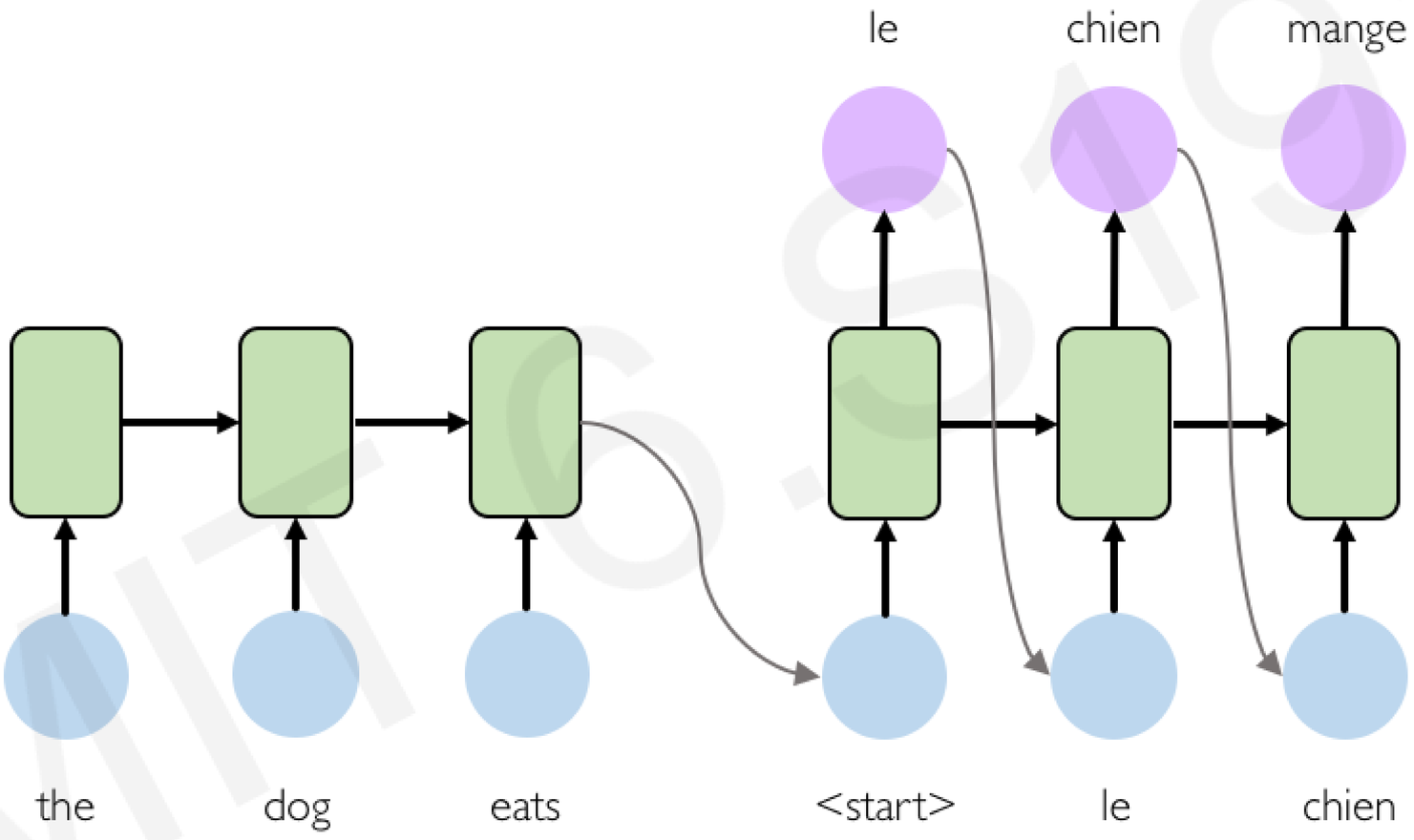
Example Task: Sentiment Classification



Tweet sentiment classification

MIT 6.S191

Example Task: Machine Translation

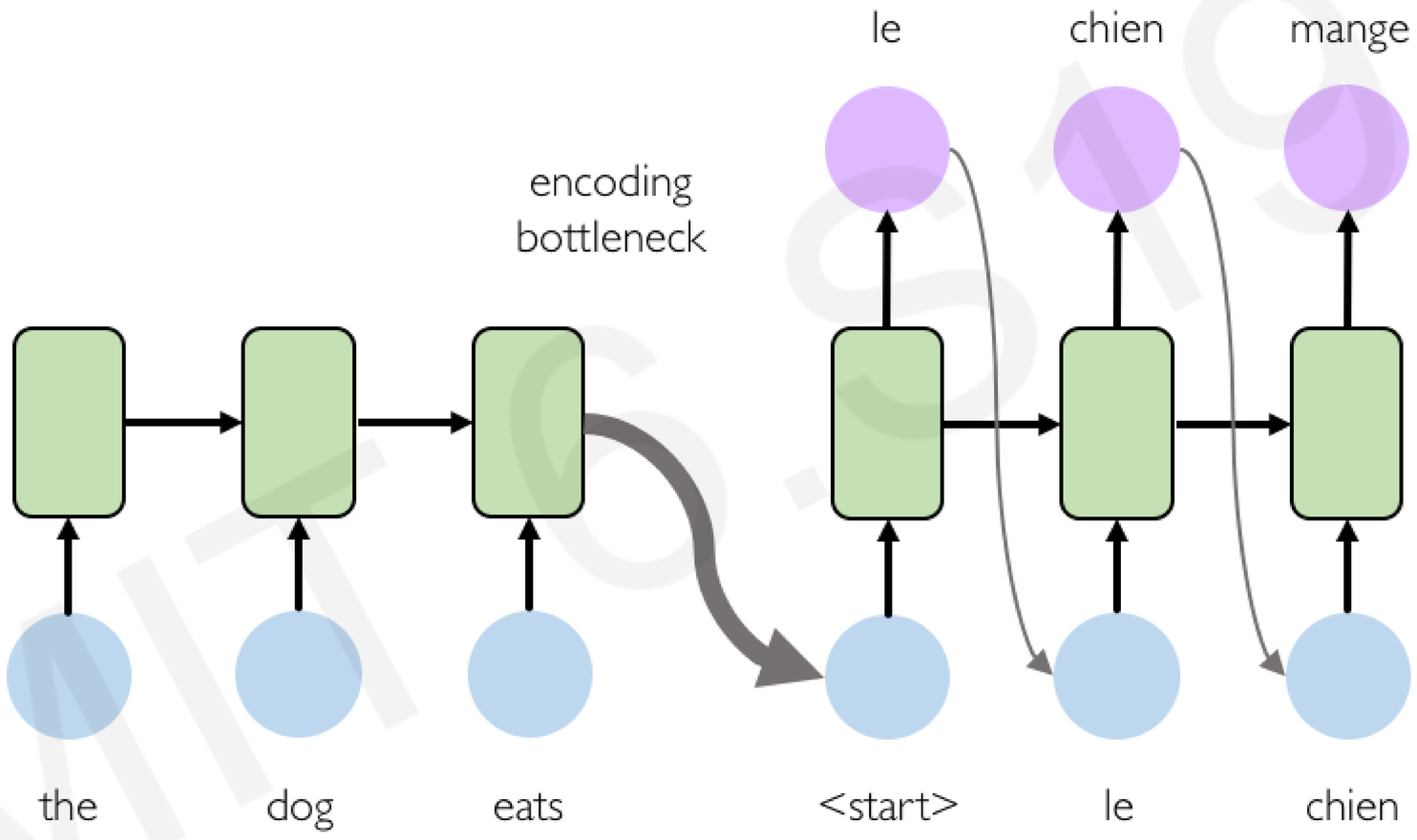


Encoder (English)

Decoder (French)

MIT 6.S191

Example Task: Machine Translation



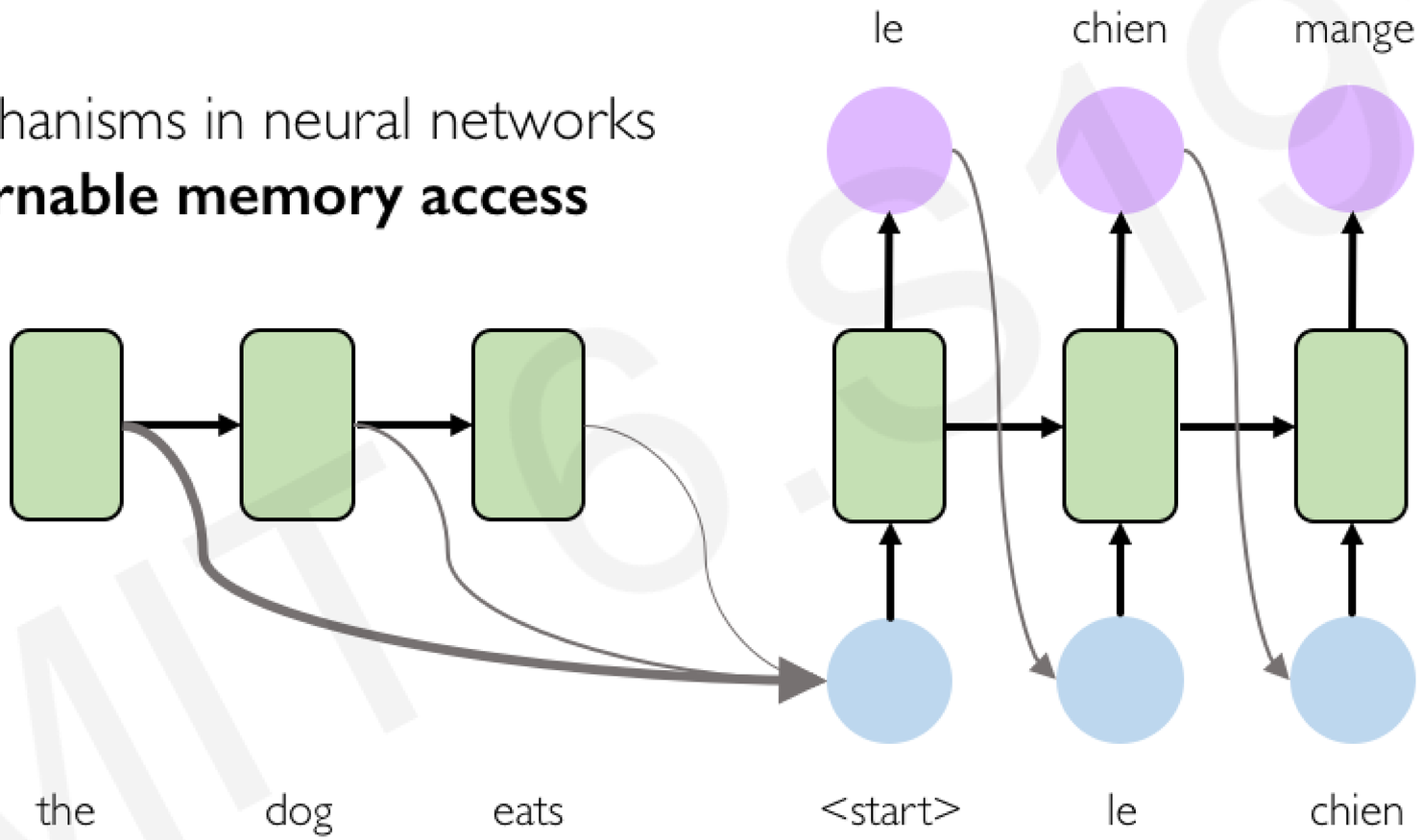
Encoder (English)

Decoder (French)

MIT 6.S191

Attention Mechanisms

Attention mechanisms in neural networks provide **learnable memory access**

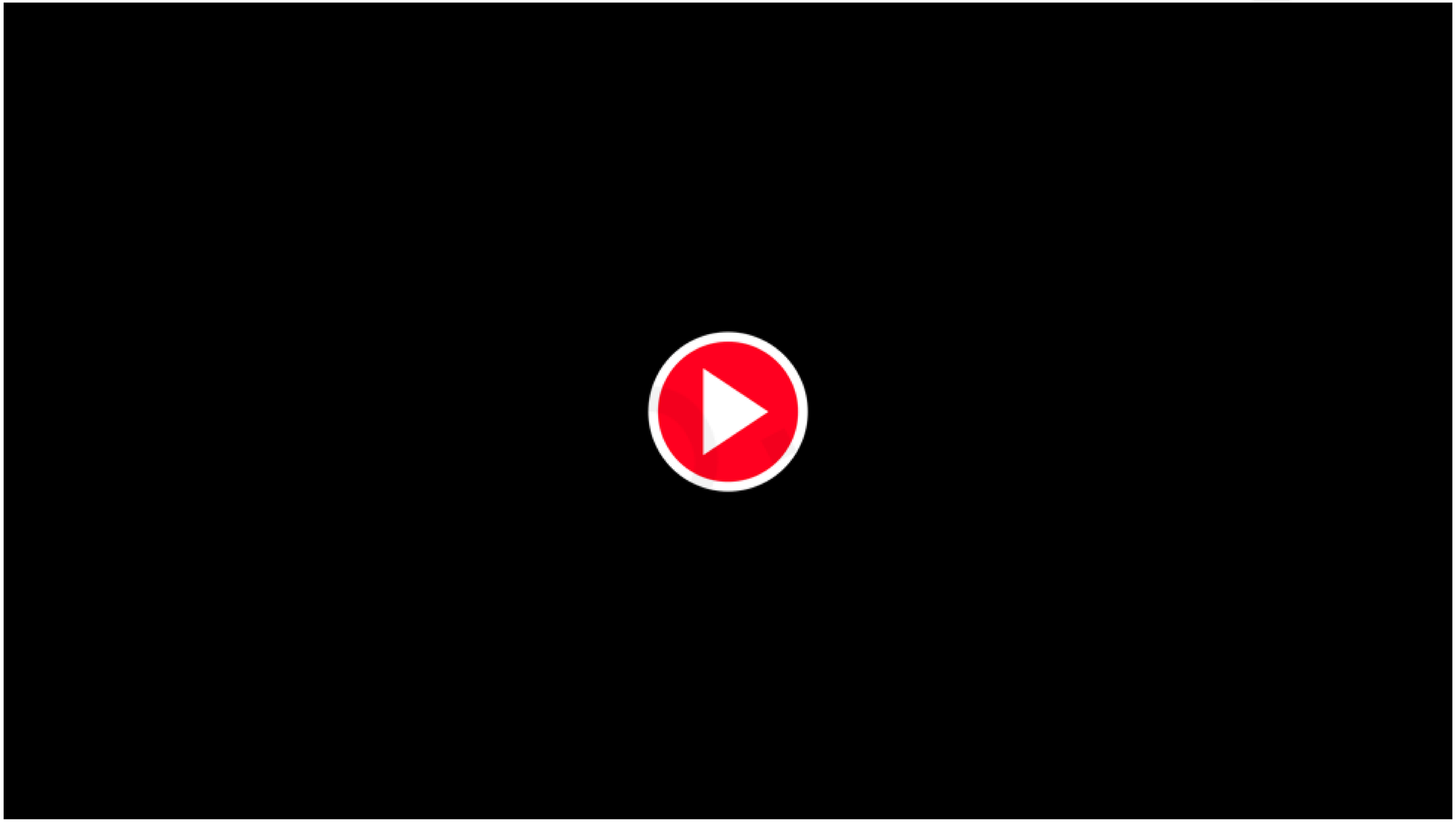


Encoder (English)

Decoder (French)

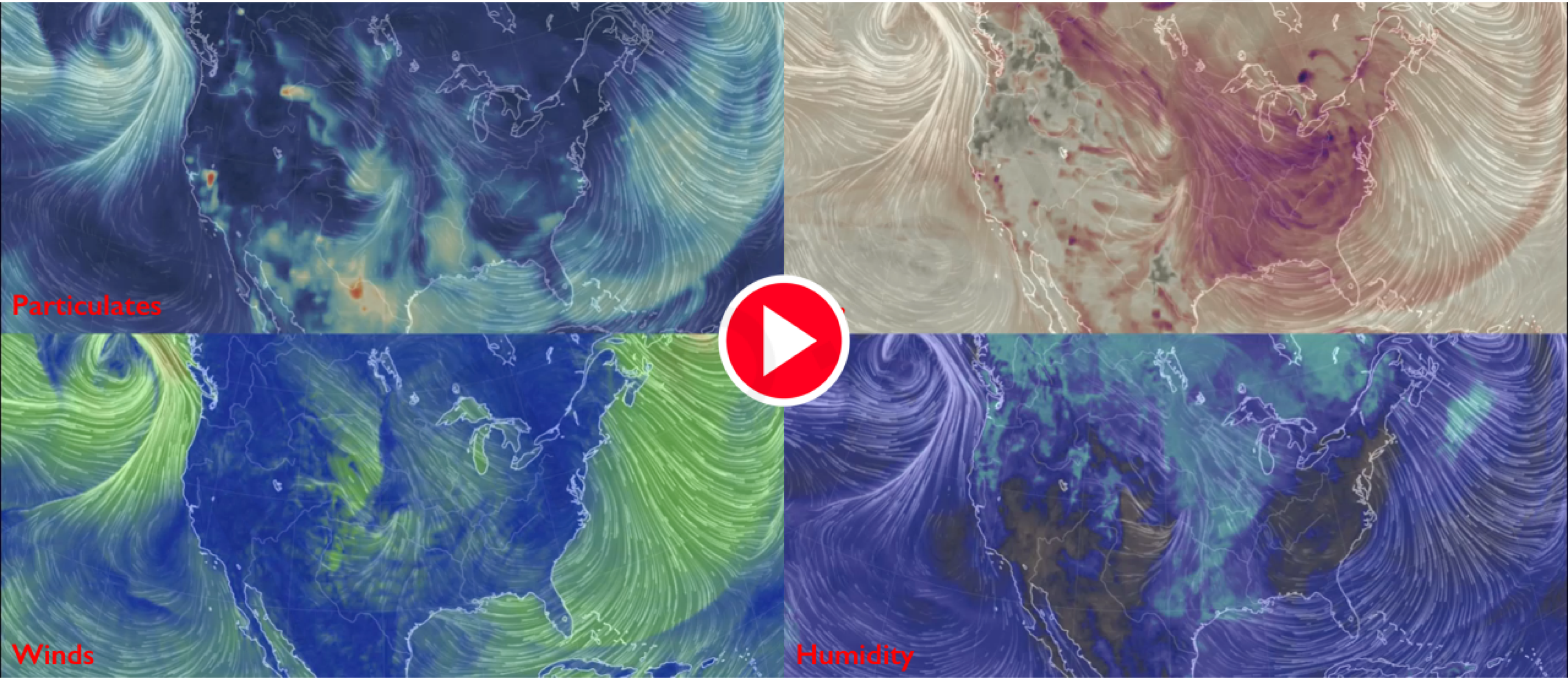
MIT 6.S191

Trajectory Prediction: Self-Driving Cars



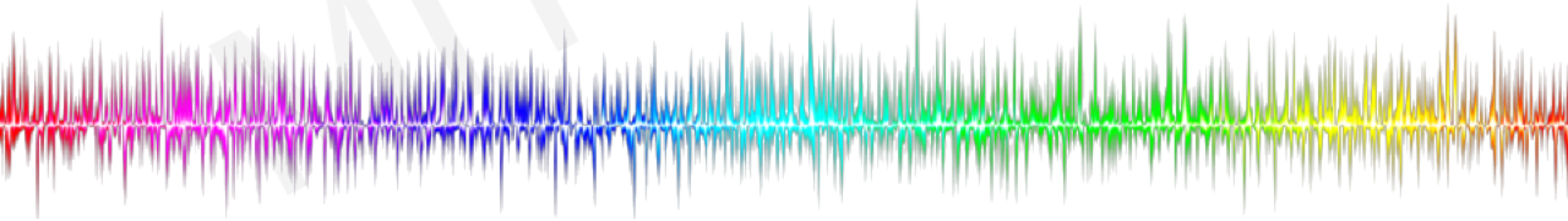
Waymo. 1/27/20

Environmental Modeling



Deep Learning for Sequence Modeling: Summary

1. RNNs are well suited for **sequence modeling** tasks
2. Model sequences via a **recurrence relation**
3. Training RNNs with **backpropagation through time**
4. Gated cells like **LSTMs** let us model **long-term dependencies**
5. Models for **music generation**, classification, machine translation, and more



MIT 6.S191

6.S191: Introduction to Deep Learning

Lab 1: Introduction to TensorFlow and Music Generation with RNNs

Link to download labs:
http://introtodeeplearning.com#schedule

1. Open the lab in Google Colab
2. Start executing code blocks and filling in the #TODOs
3. Need help? Find a TA or come to the front!!