Deep Sequence Modeling

Ava Soleimany

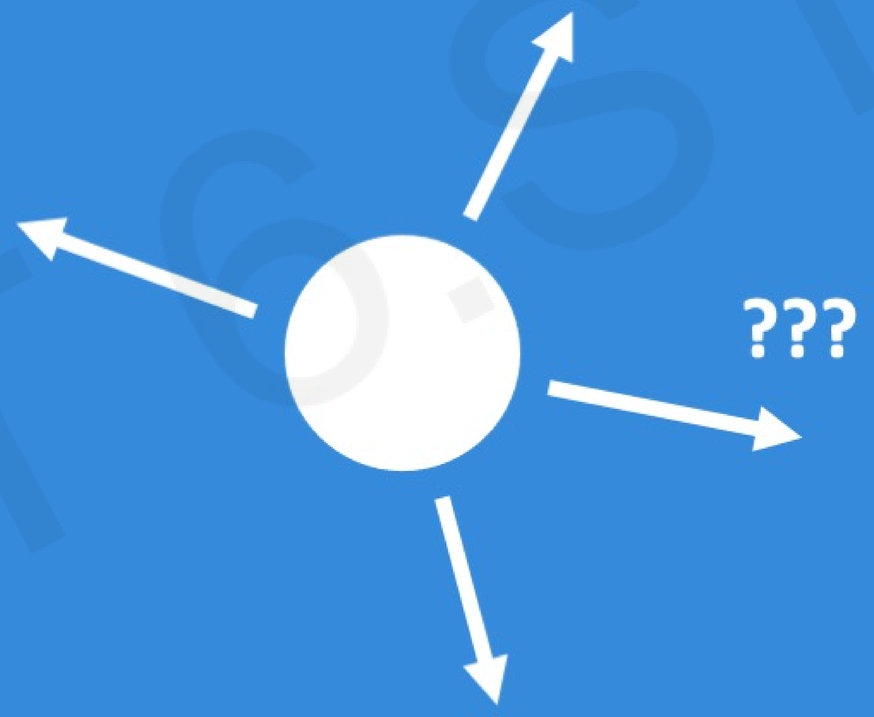
MIT 6.S191

January 19, 2021

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

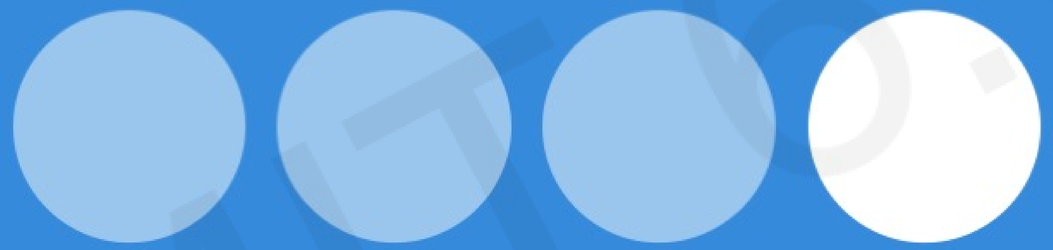
MIT 6.S191

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



MIT 6.S191

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

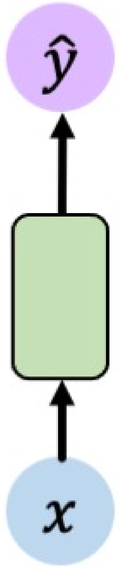
Sequences in the Wild

Audio

MIT 6.S191

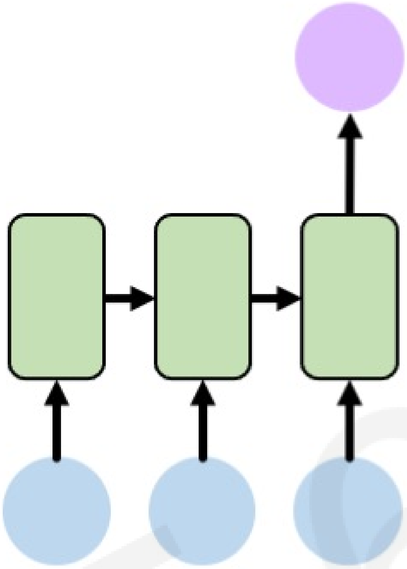
Sequences in the Wild

Sequence Modeling Applications

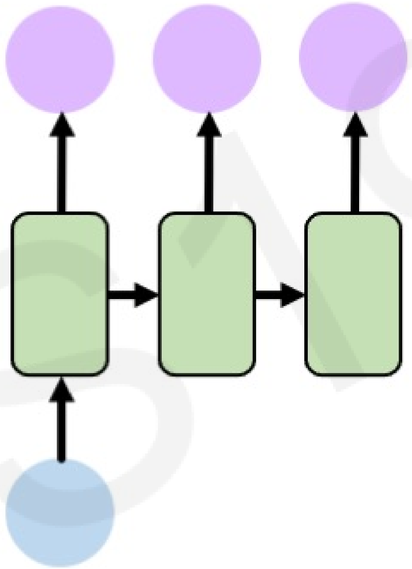


One to One
Binary Classification

"Will I pass this class?"
Student → Pass?

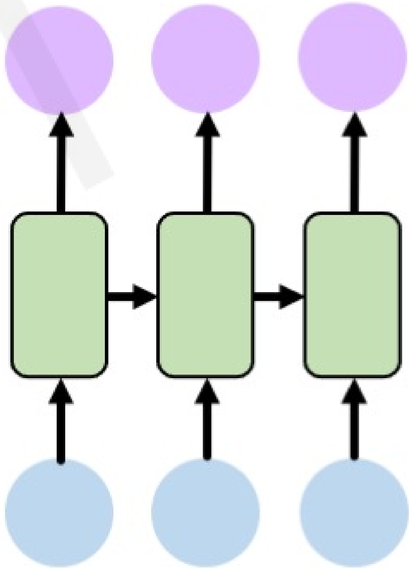


Many to One
Sentiment Classification



One to Many
Image Captioning

"A baseball player throws a ball."



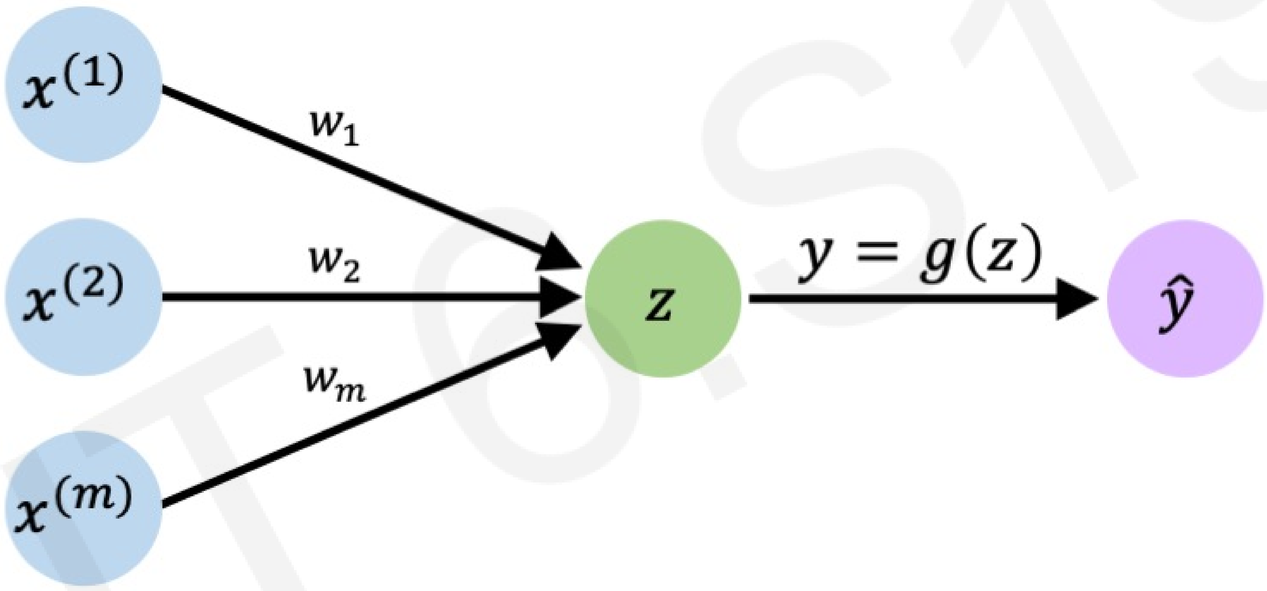
Many to Many
Machine Translation

MIT 6.S191

Neurons with Recurrence

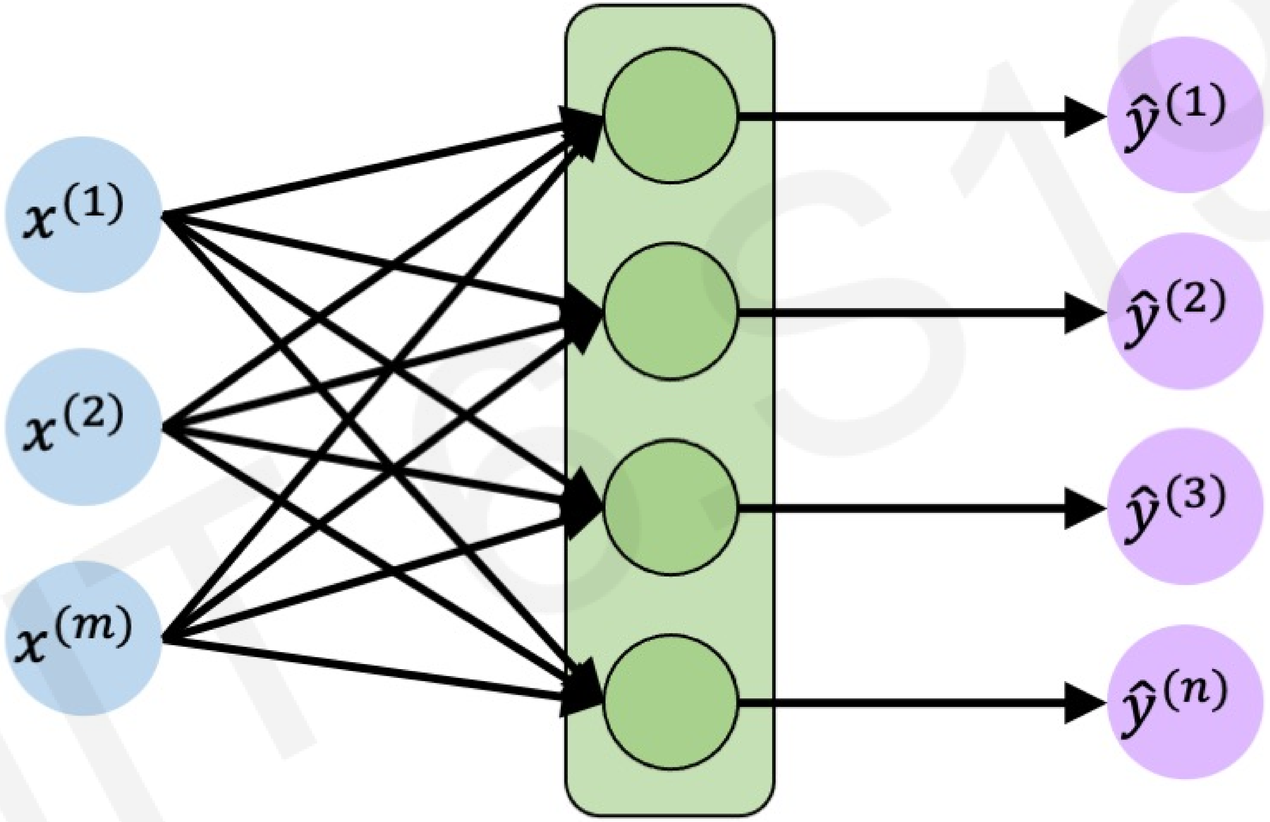
MIT 6.S191

The Perceptron Revisited



MIT 6.S191

Feed-Forward Networks Revisited

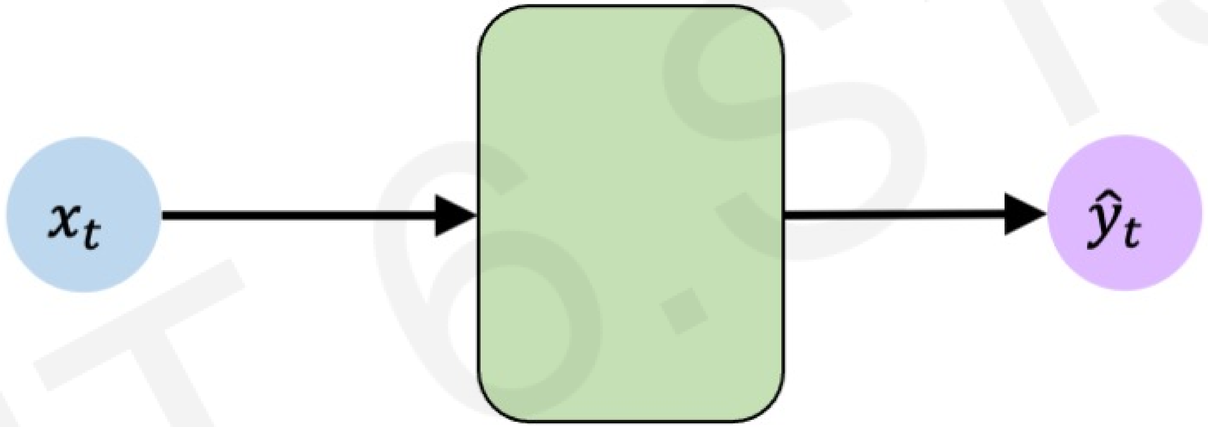


$\boldsymbol{x} \in \mathbb{R}^m$

$\hat{\boldsymbol{y}} \in \mathbb{R}^n$

MIT 6S191

Feed-Forward Networks Revisited

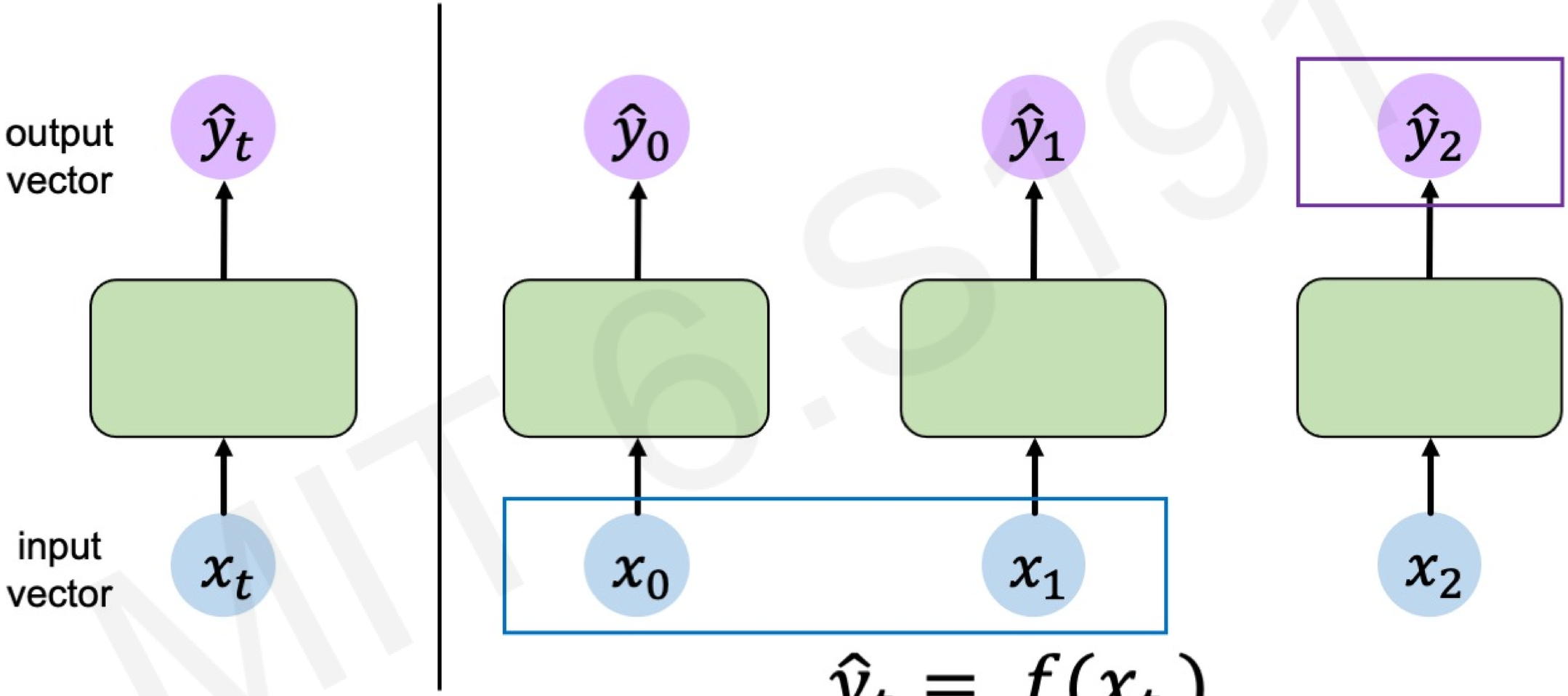


$\boldsymbol{x}_t \in \mathbb{R}^m$

$\hat{\boldsymbol{y}}_t \in \mathbb{R}^n$

MIT 6.S191

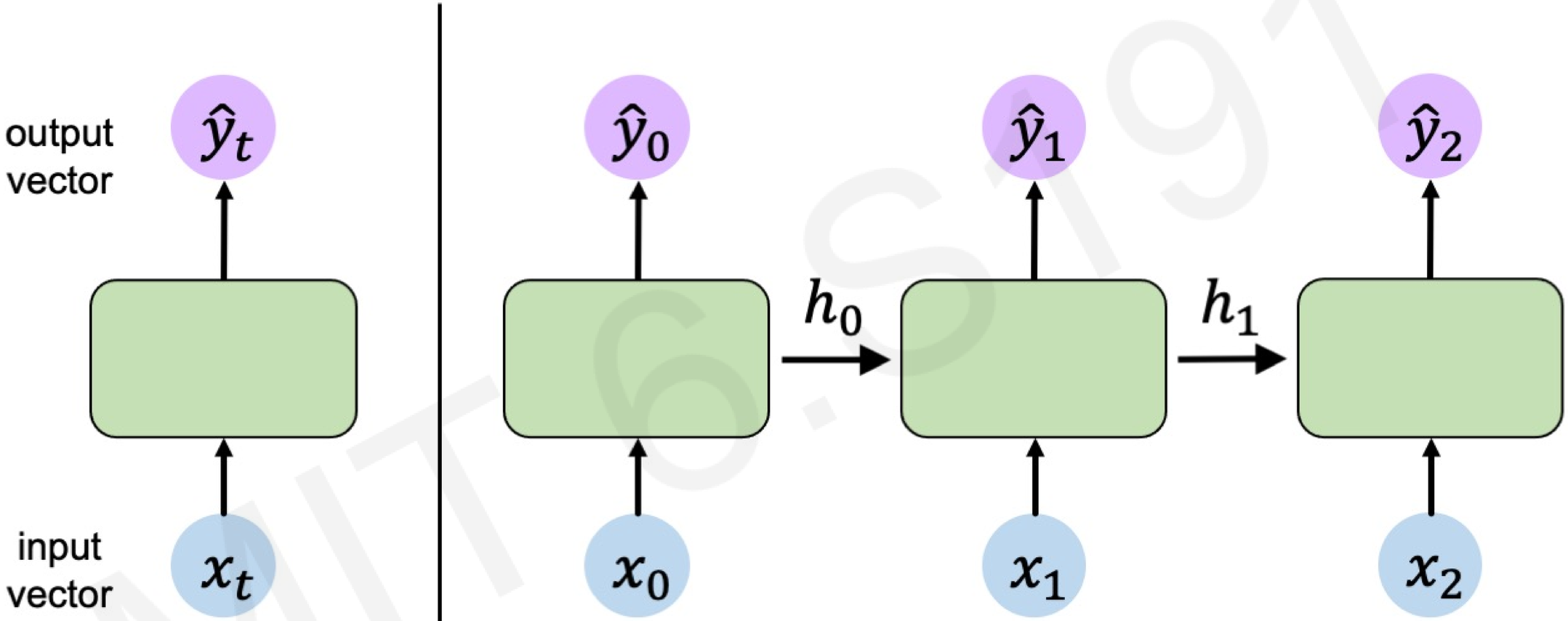
Handling Individual Time Steps



$$\hat{y}_t = f(x_t)$$

MIT 6.S191

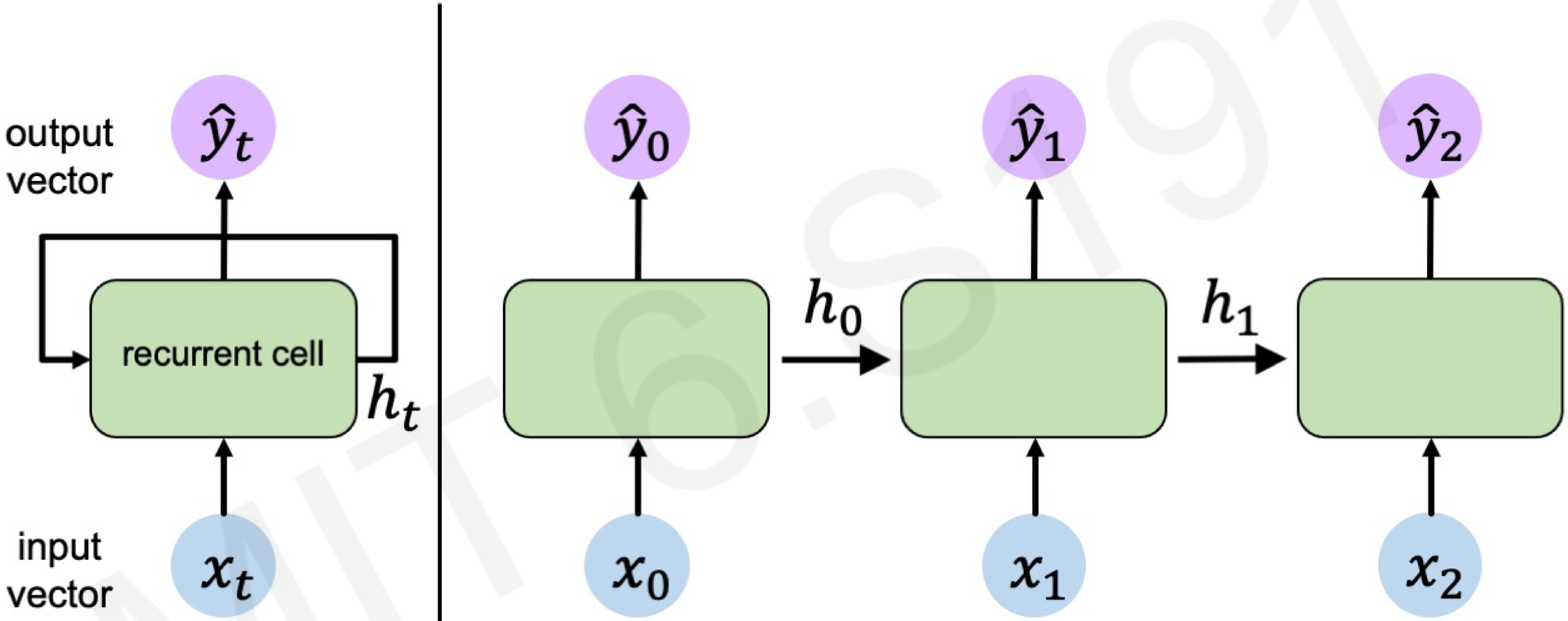
Neurons with Recurrence



$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

MIT 6.S191

Neurons with Recurrence



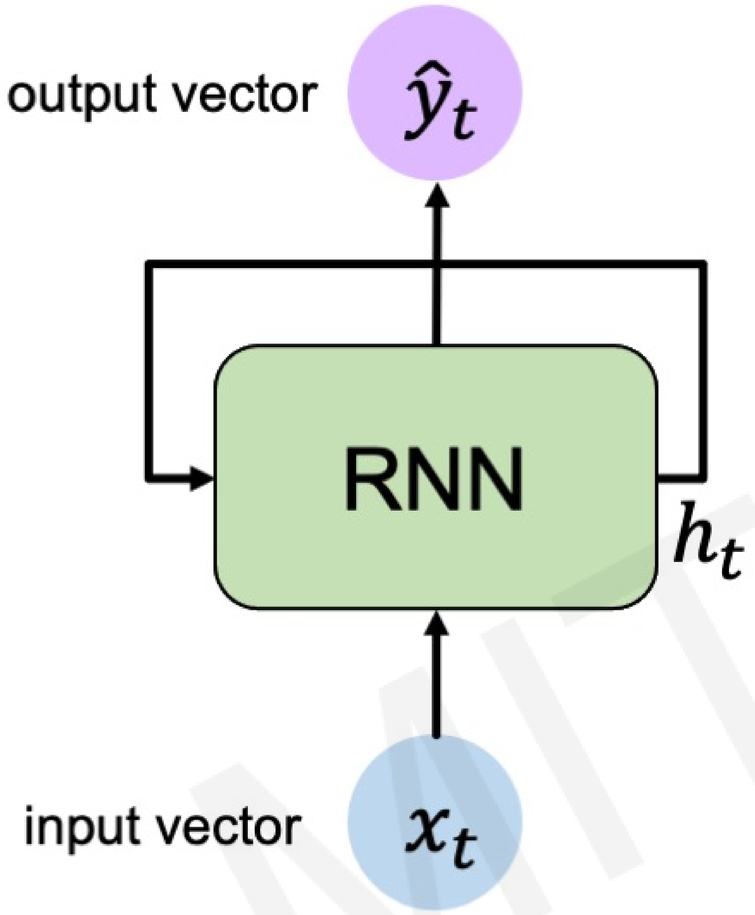
$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

MIT 6.S191

Recurrent Neural Networks (RNNs)

MIT 6.S191

Recurrent Neural Networks (RNNs)



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

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

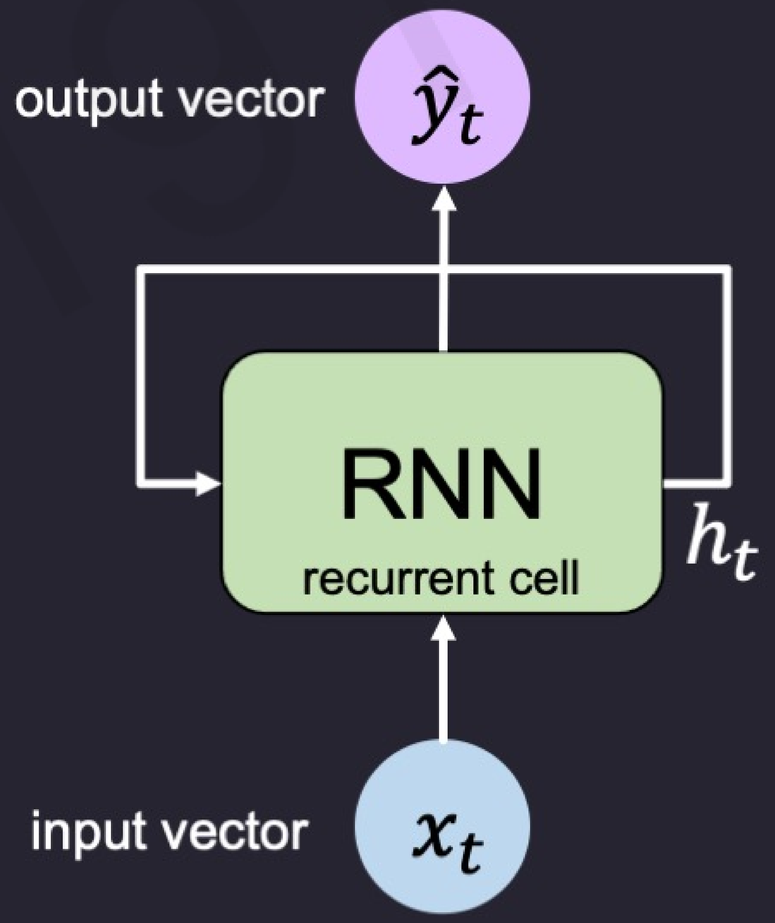
cell state — function with weights W — input — old state

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

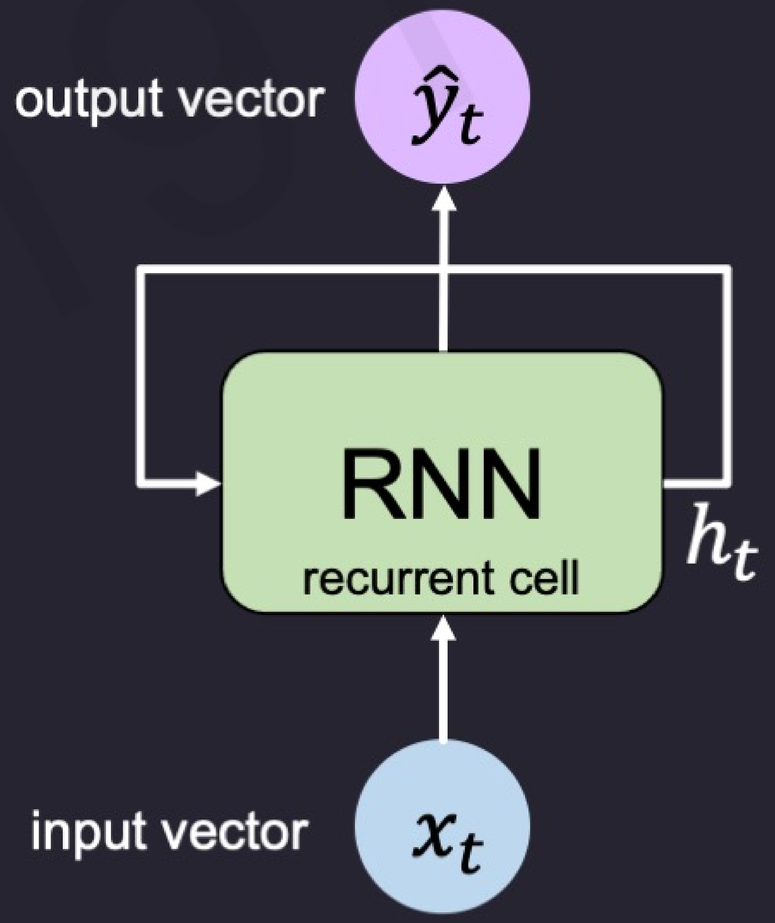
RNN Intuition

```
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!"
```



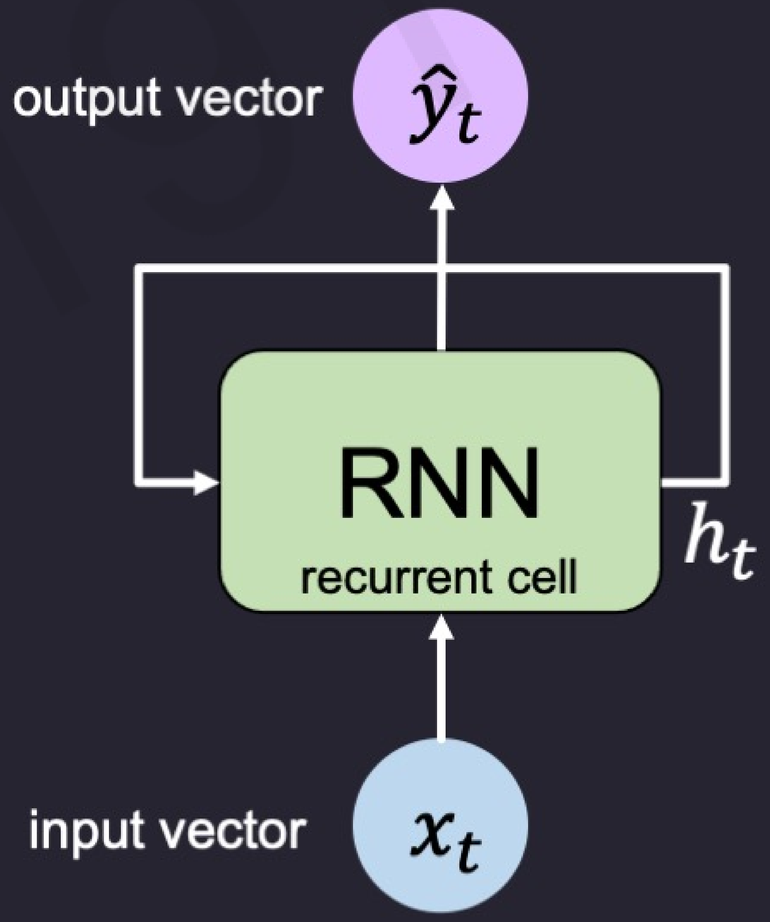
RNN Intuition

```
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!"
```

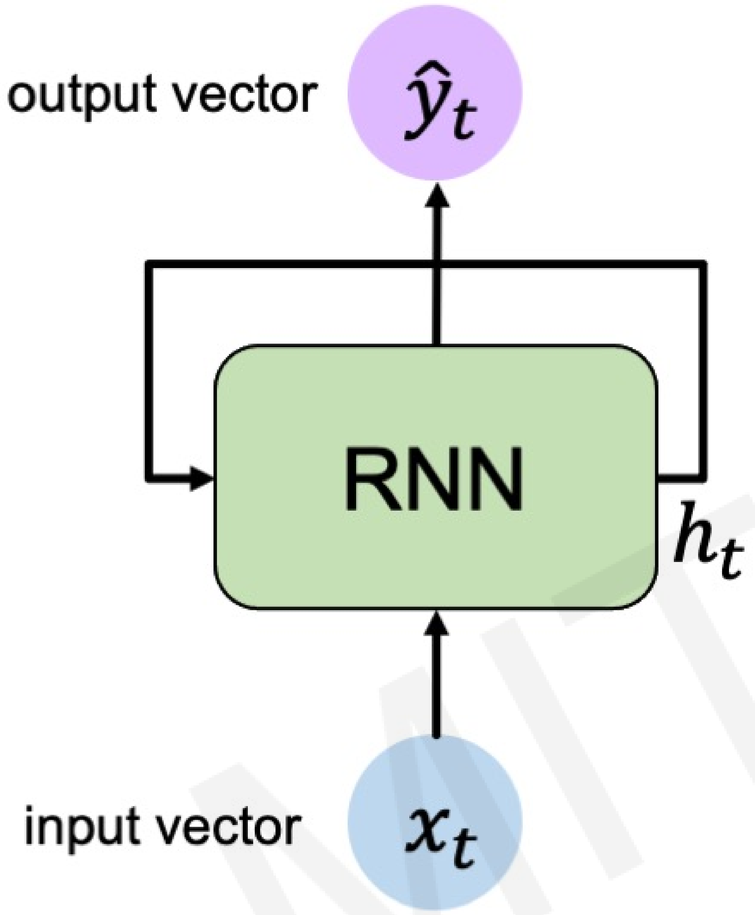


RNN Intuition

```
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!"
```

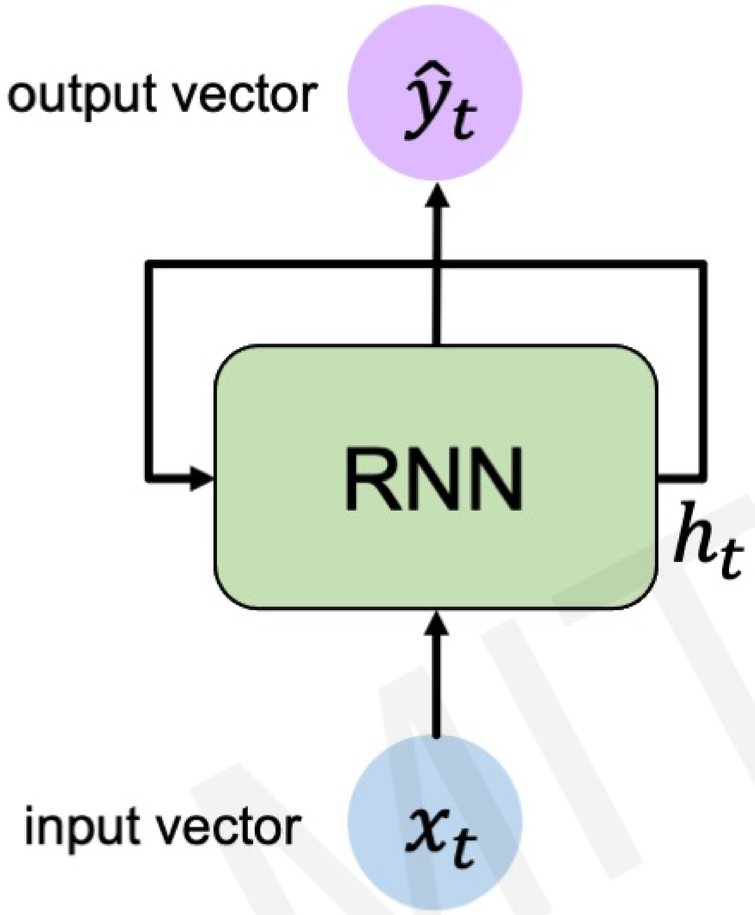


RNN State Update and Output



MIT 6.S191

RNN State Update and Output

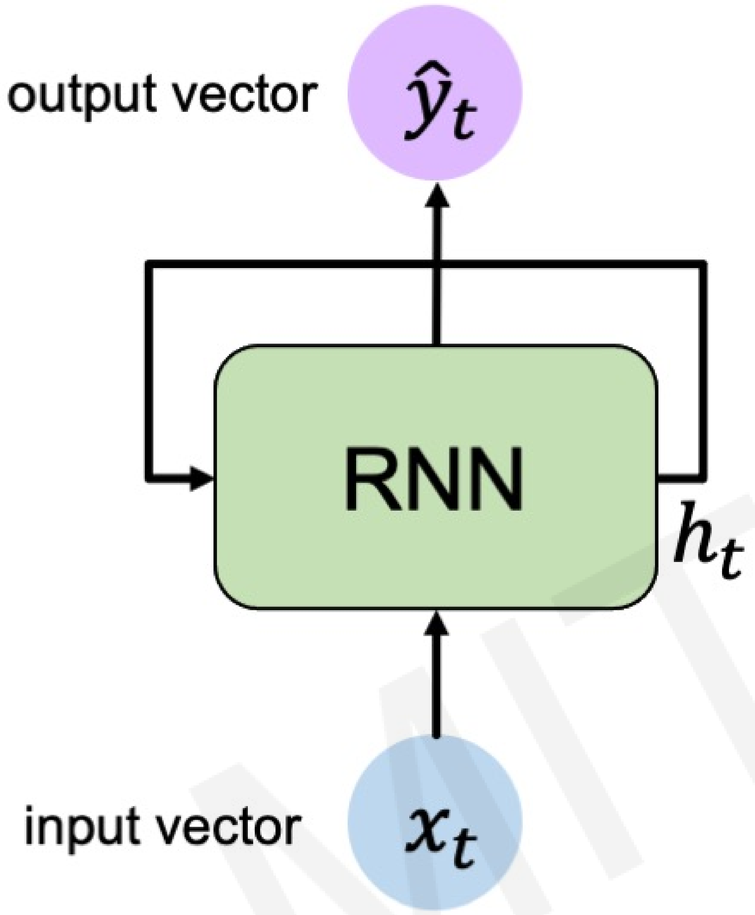


Input Vector

x_t

MIT 6.S191

RNN State Update and Output



Update Hidden State

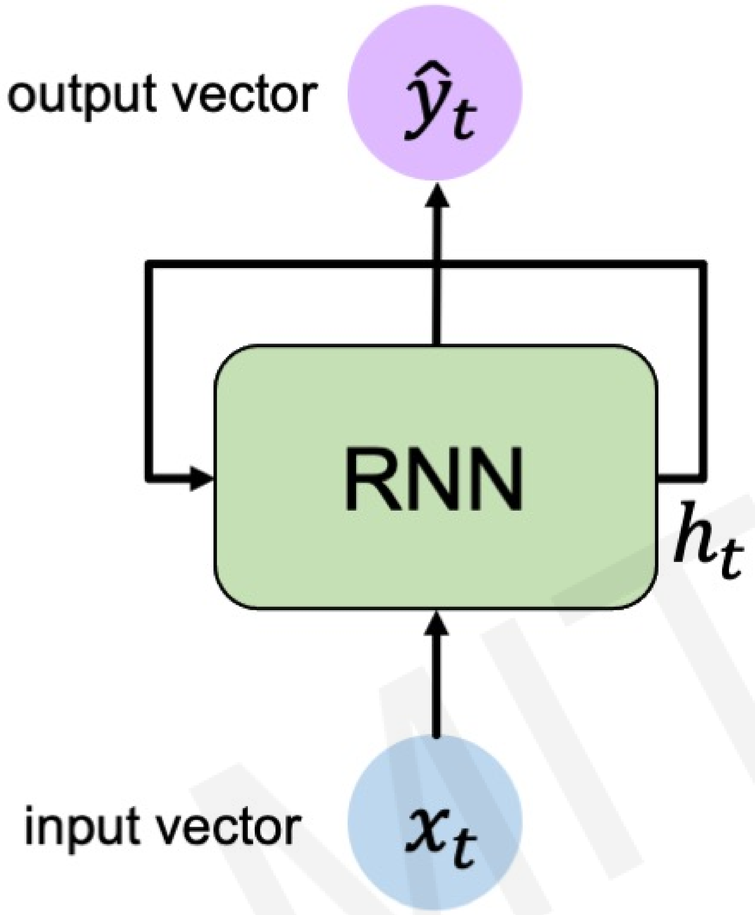
$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

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

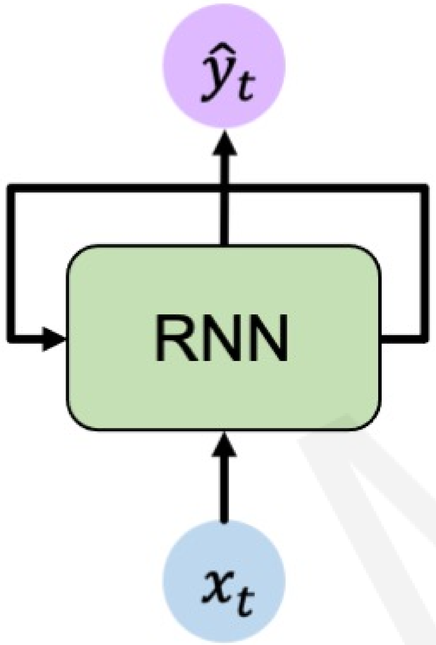
$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

Input Vector

$$x_t$$

MIT 6.S191

RNNs: Computational Graph Across Time

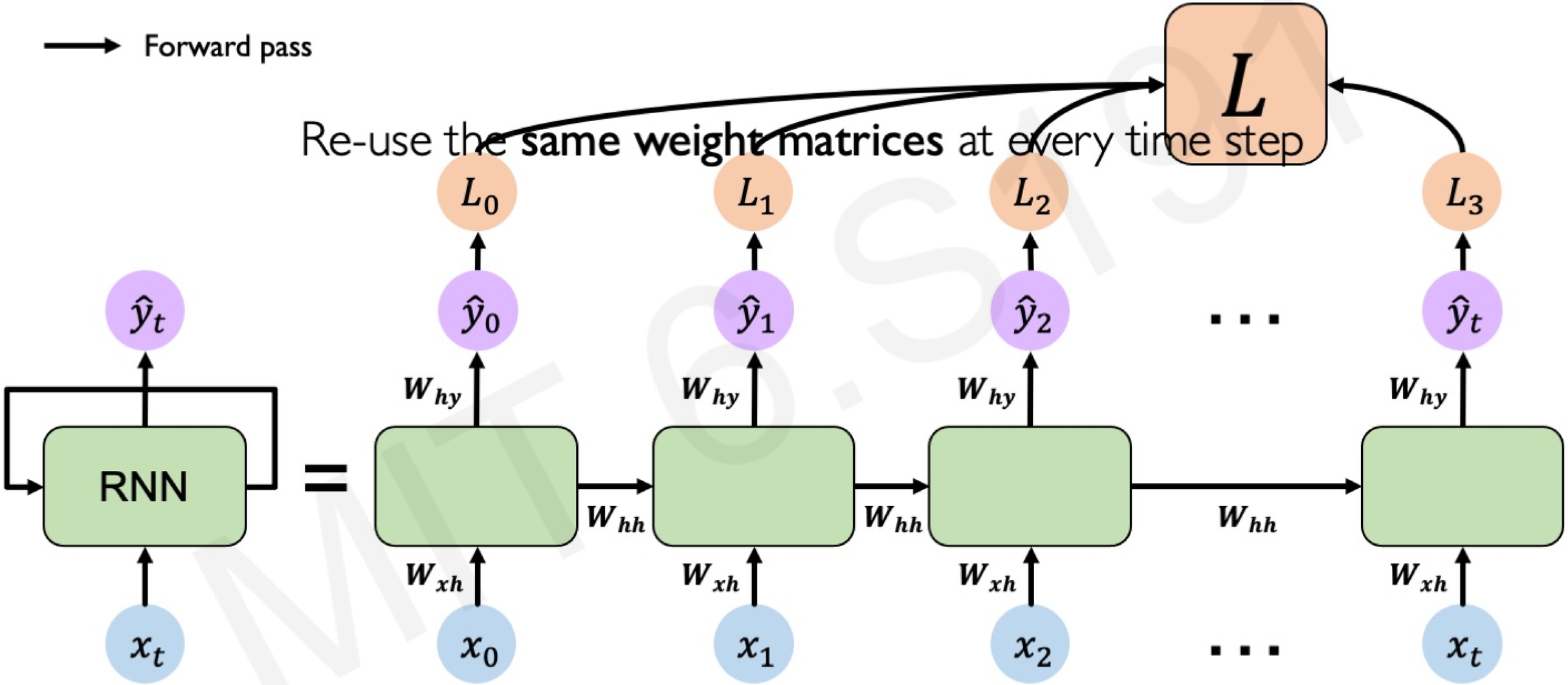


=

Represent as computational graph unrolled across time

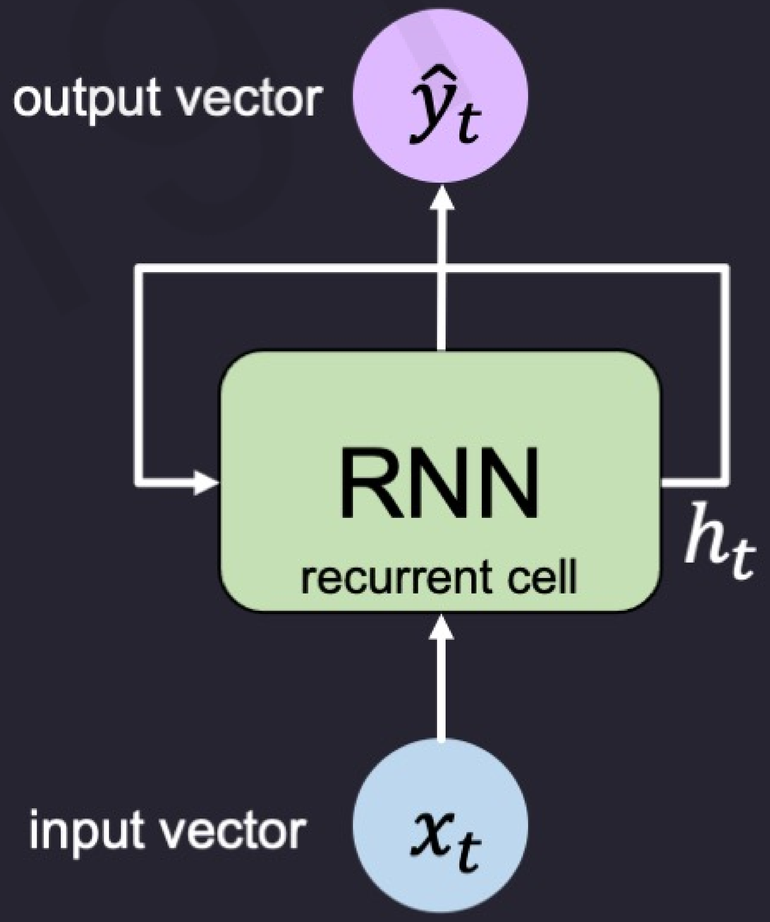
MIT 6.S191

RNNs: Computational Graph Across Time



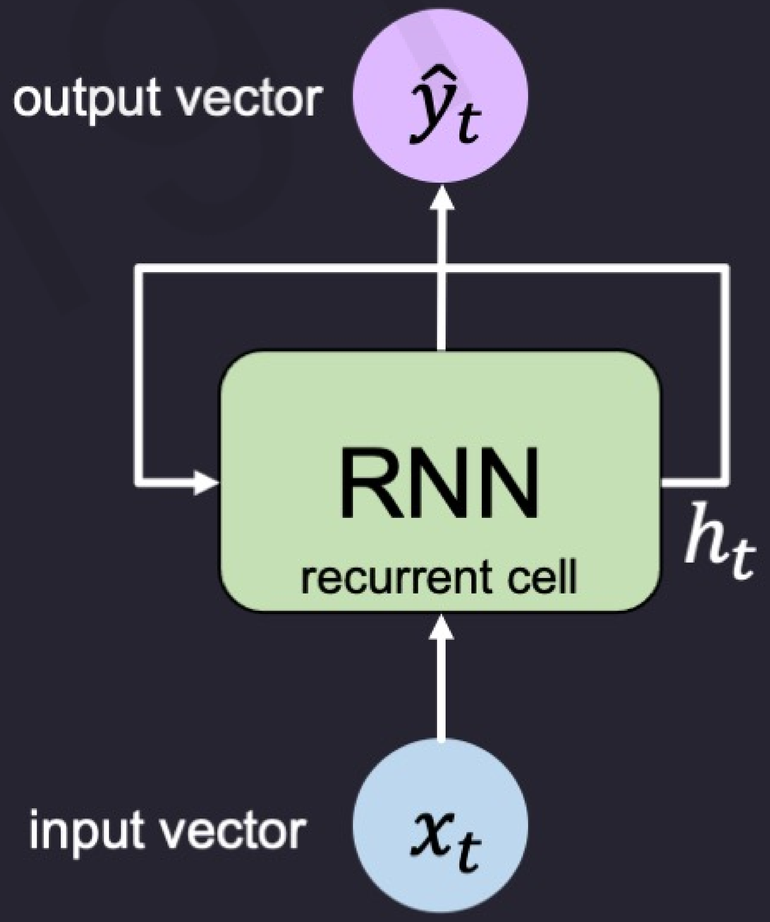
RNNs from Scratch

```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
```

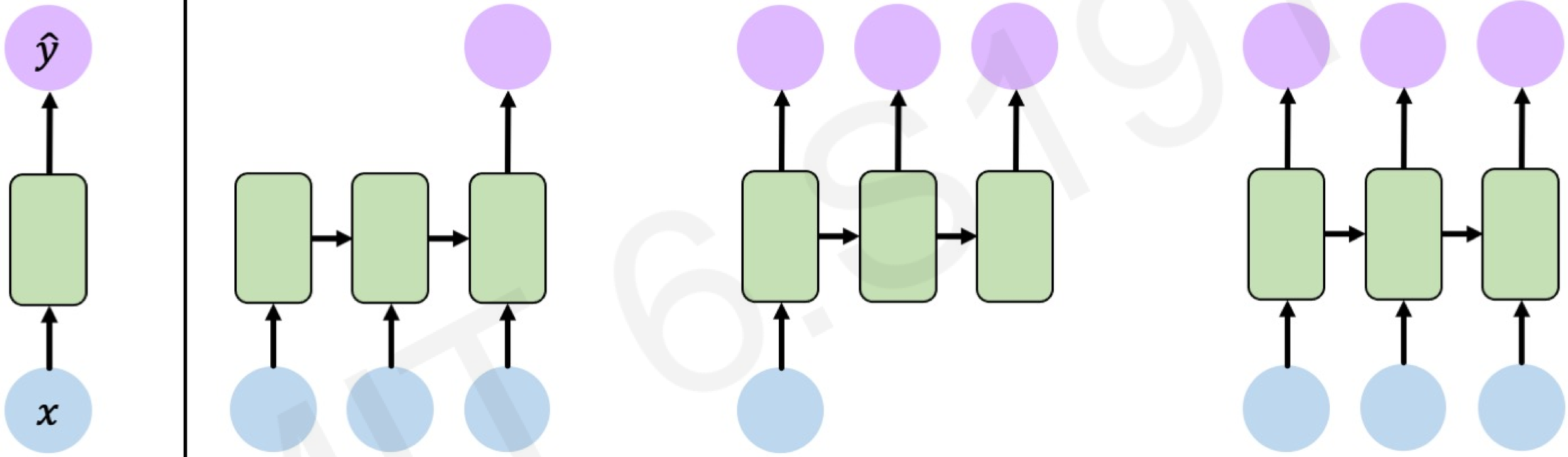


RNN Implementation in TensorFlow

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



RNNs for Sequence Modeling



One to One
"Vanilla" NN
Binary classification

Many to One
Sentiment Classification

One to Many
Text Generation
Image Captioning

Many to Many
Translation & Forecasting
Music Generation

… and many other architectures and applications

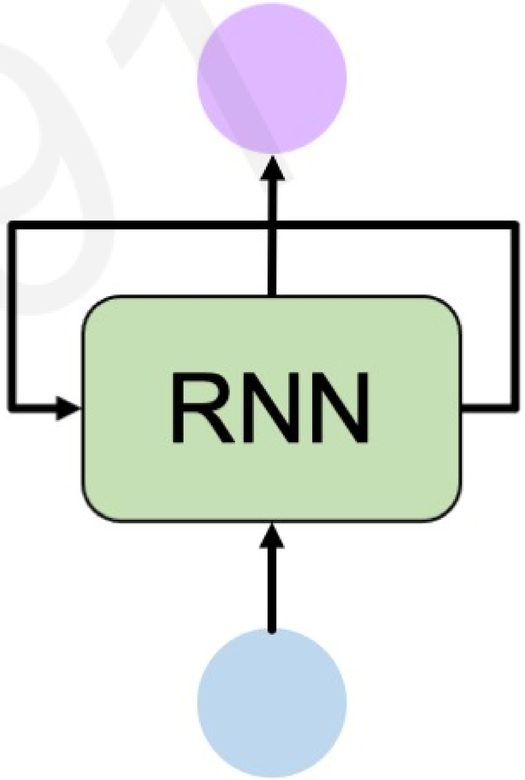
6.S191 Lab!

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



Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria

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

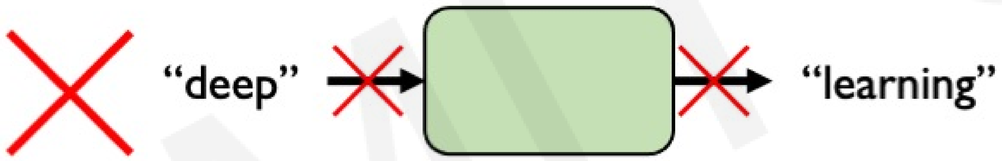
A Sequence Modeling Problem: Predict the Next Word

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

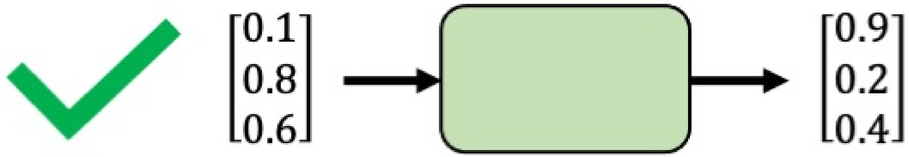
given these words

predict the next word

Representing Language to a Neural Network



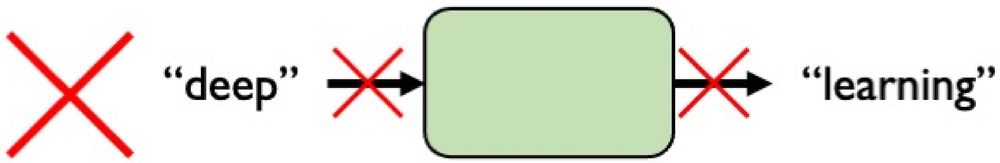
Neural networks cannot interpret words



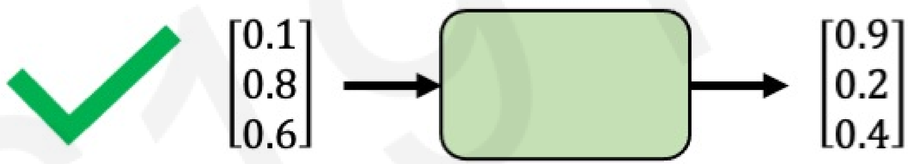
Neural networks require numerical inputs

MIT 6.S191

Encoding Language for a Neural Network

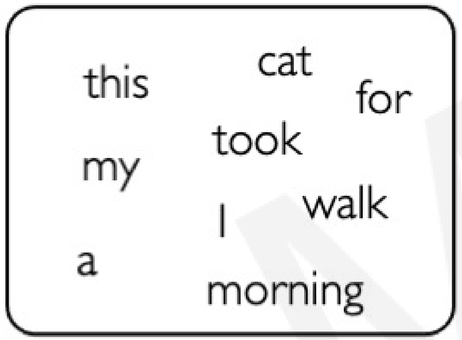


Neural networks cannot interpret words

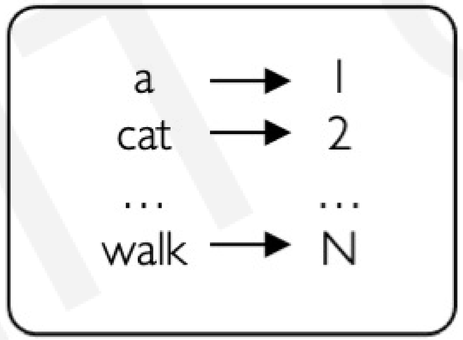


Neural networks require numerical inputs

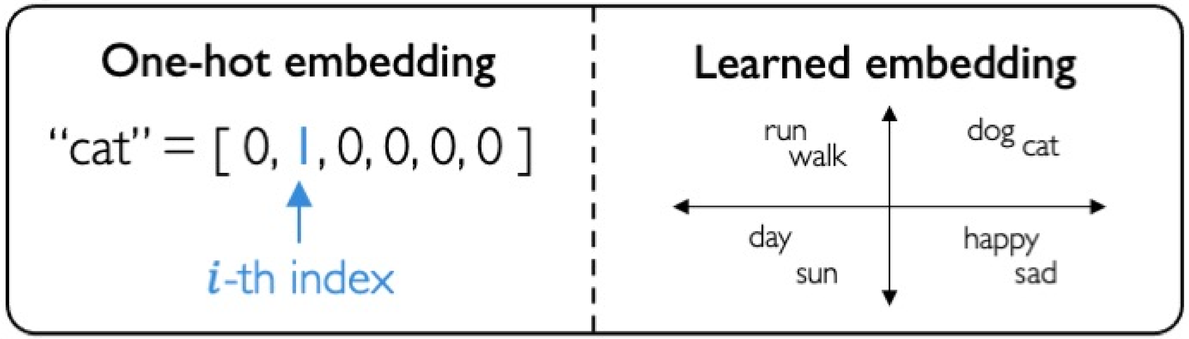
Embedding: transform indexes into a vector of fixed size.



1. Vocabulary:
Corpus of words



2. Indexing:
Word to index



3. Embedding:
Index to fixed-sized vector

MIT 6.S191

Handle Variable Sequence Lengths

The food was great

vs.

We visited a restaurant for lunch

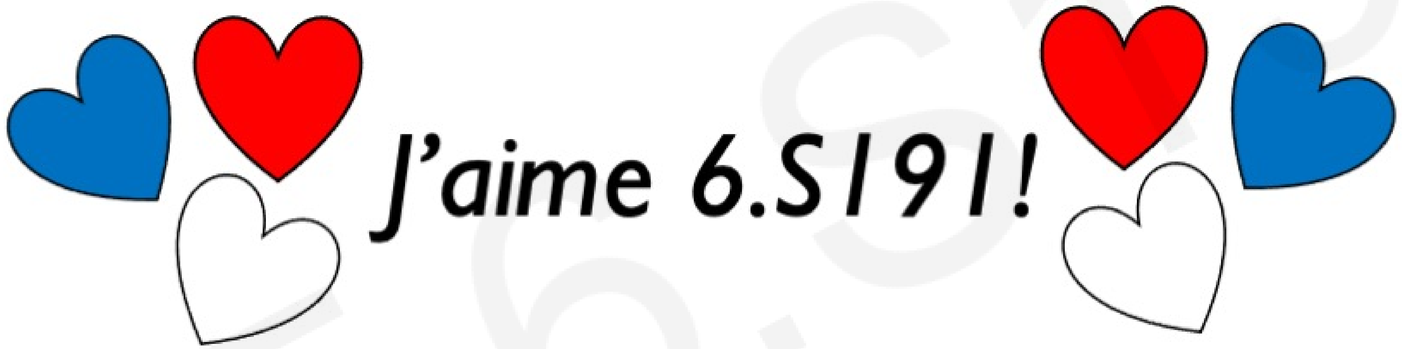
vs.

We were hungry but cleaned the house before eating

MIT 6.S191

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

Capture Differences in Sequence Order

The food was good, not bad at all.

vs.

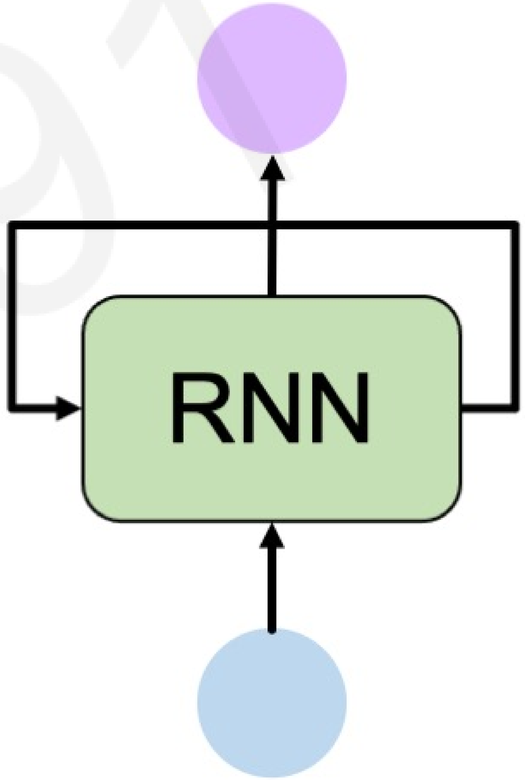
The food was bad, not good at all.

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



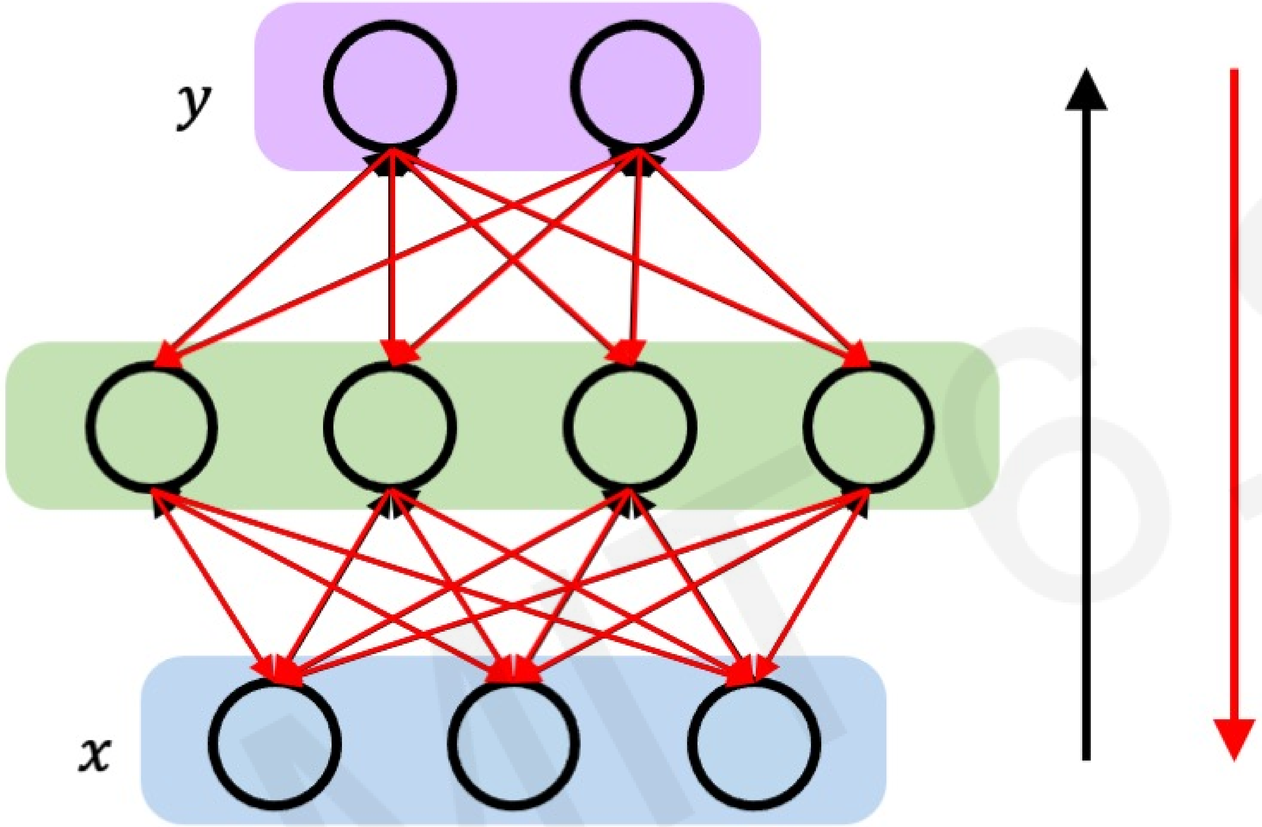
Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria

MIT 6.S191

Backpropagation Through Time (BPTT)

MIT 6.S191

Recall: Backpropagation in Feed Forward Models

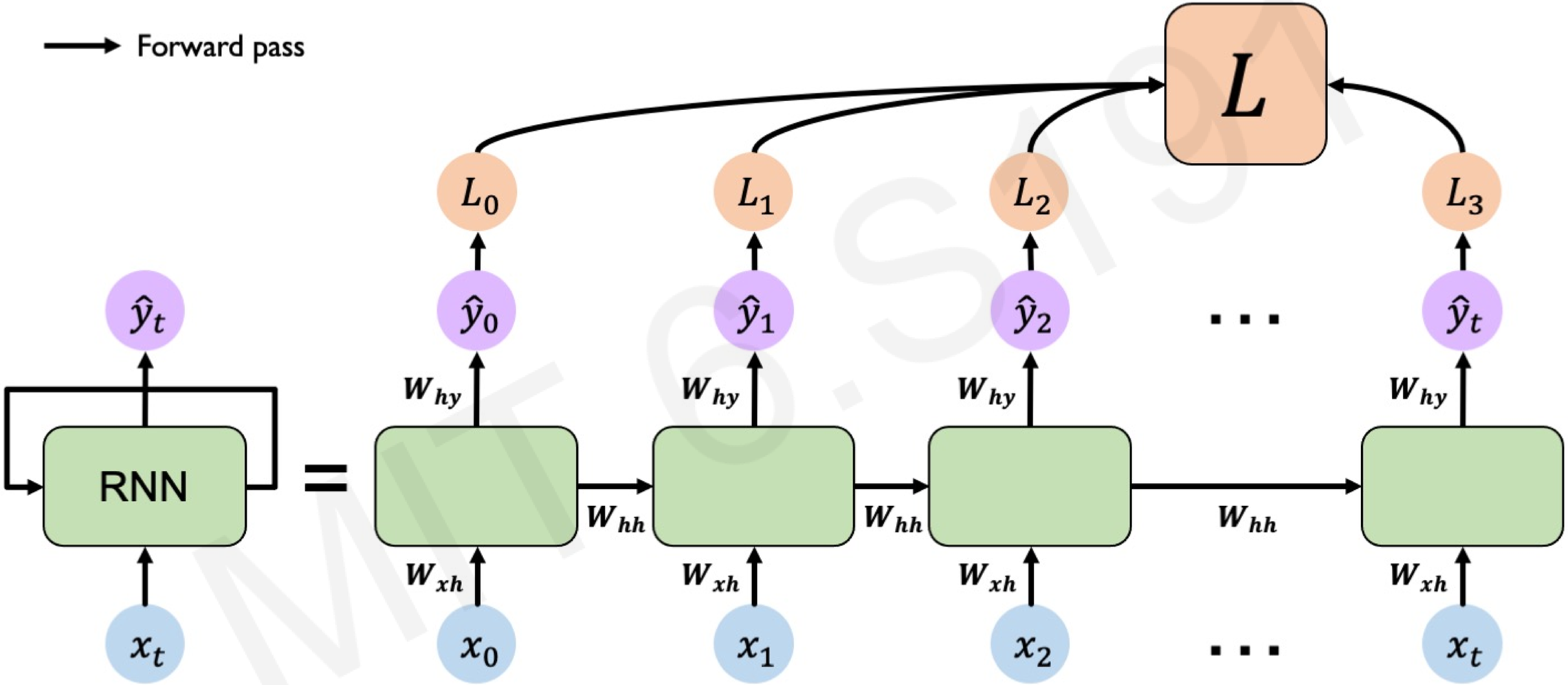


Backpropagation algorithm:

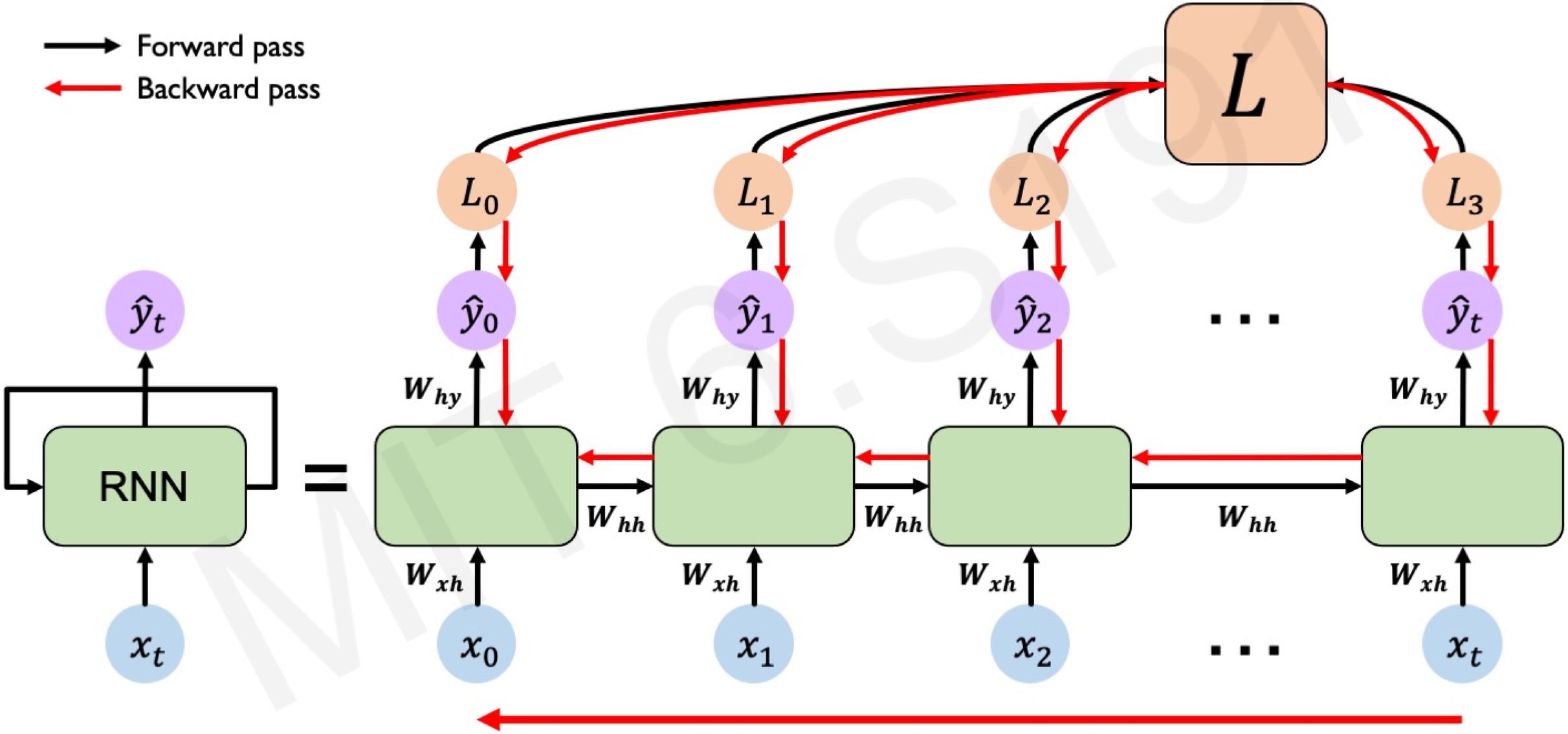
1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

MIT 6.S191

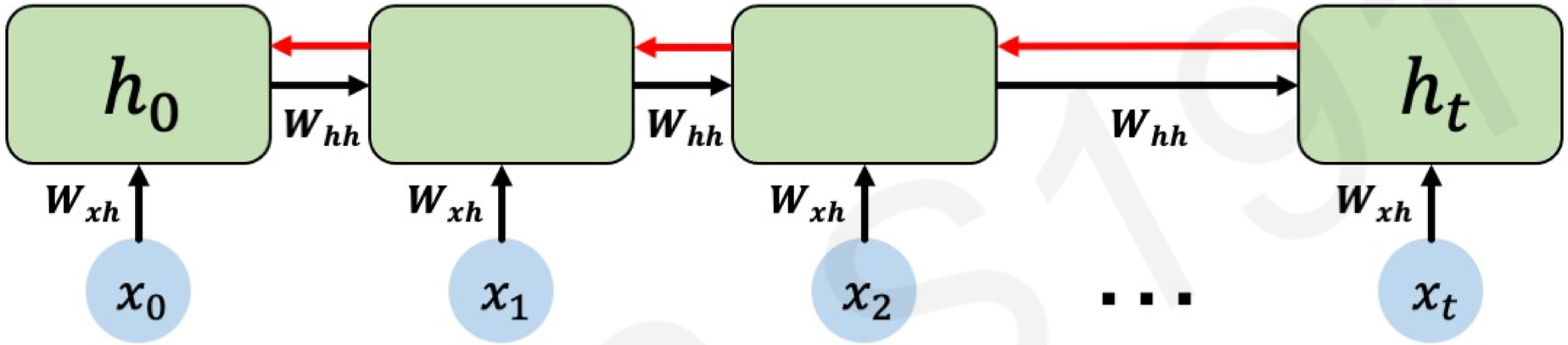
RNNs: Backpropagation Through Time



RNNs: Backpropagation Through Time

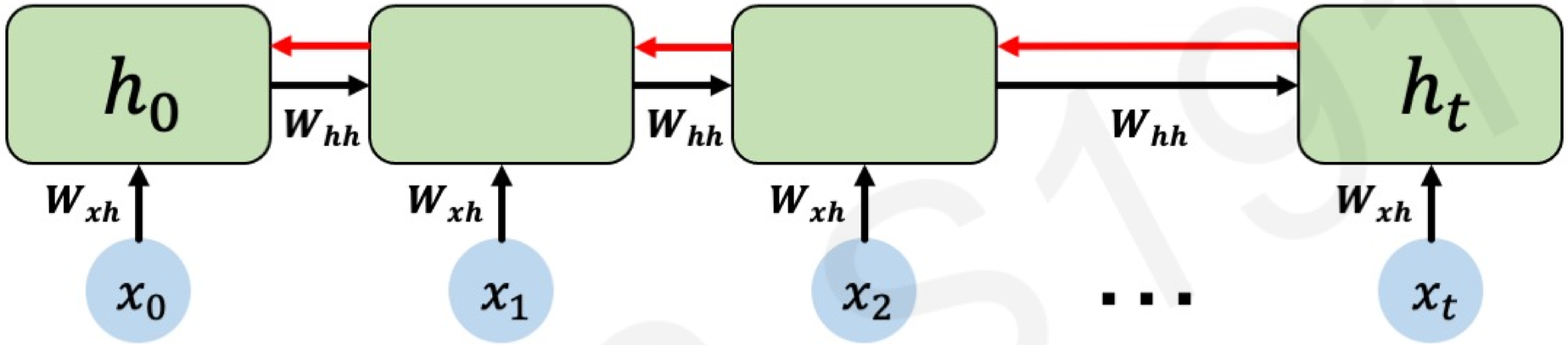


Standard RNN Gradient Flow



MIT 6.S191

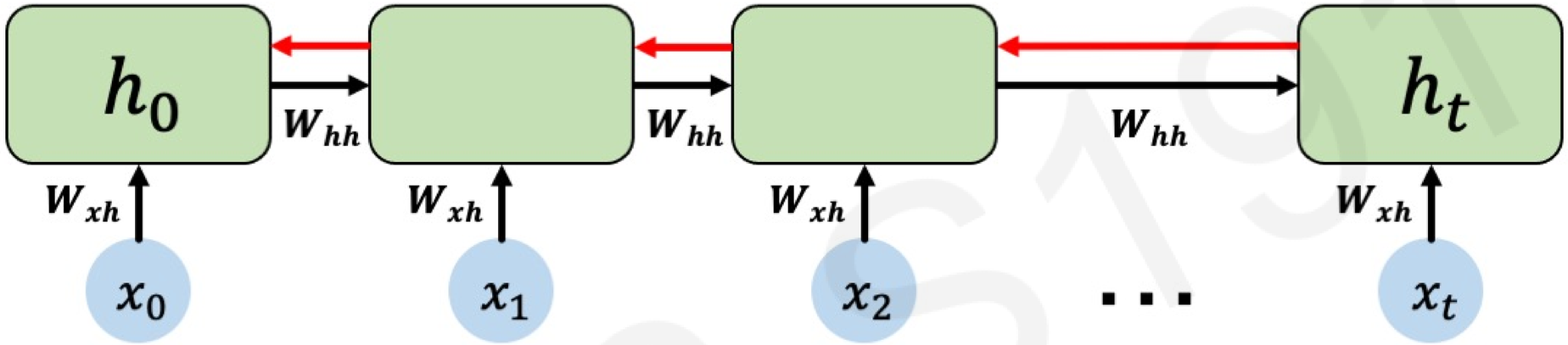
Standard RNN Gradient Flow



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

MIT 6.S191

Standard RNN Gradient Flow: Exploding Gradients



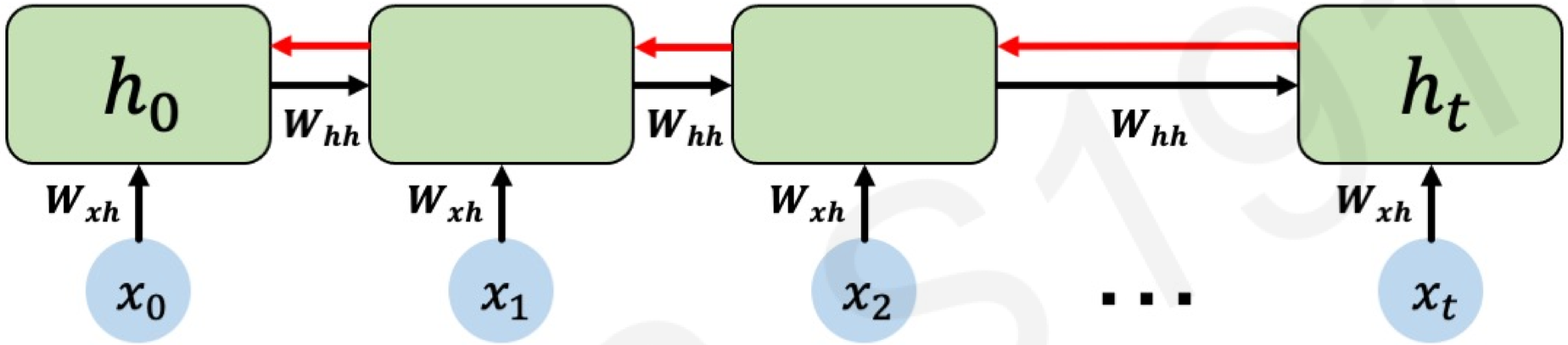
Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation**!

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

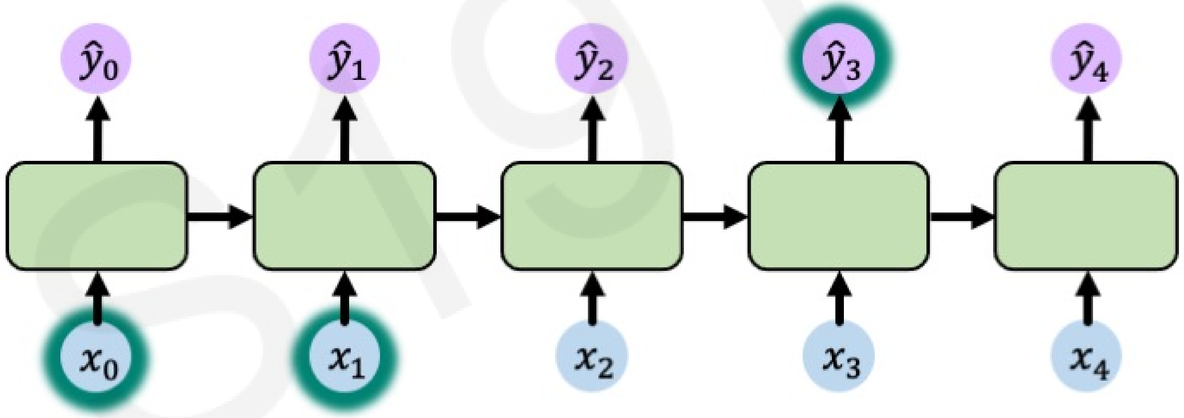
↓

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

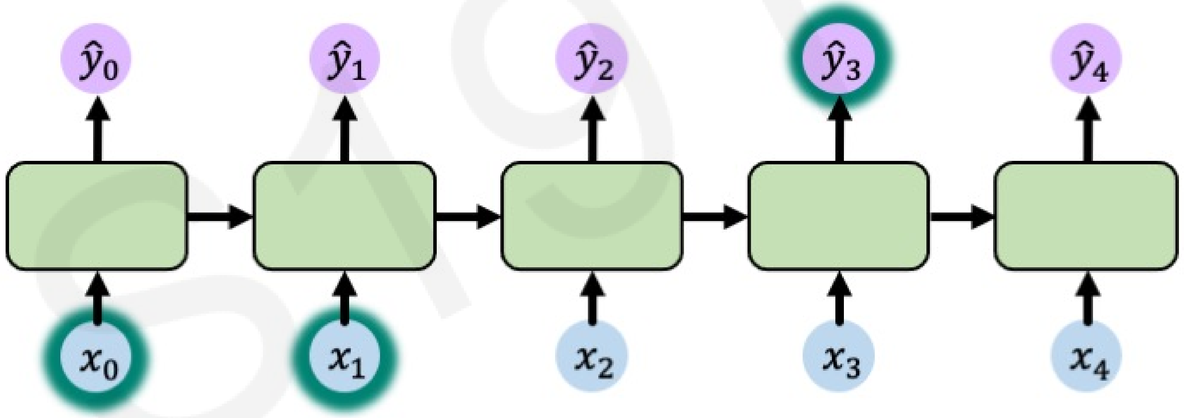
↓

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

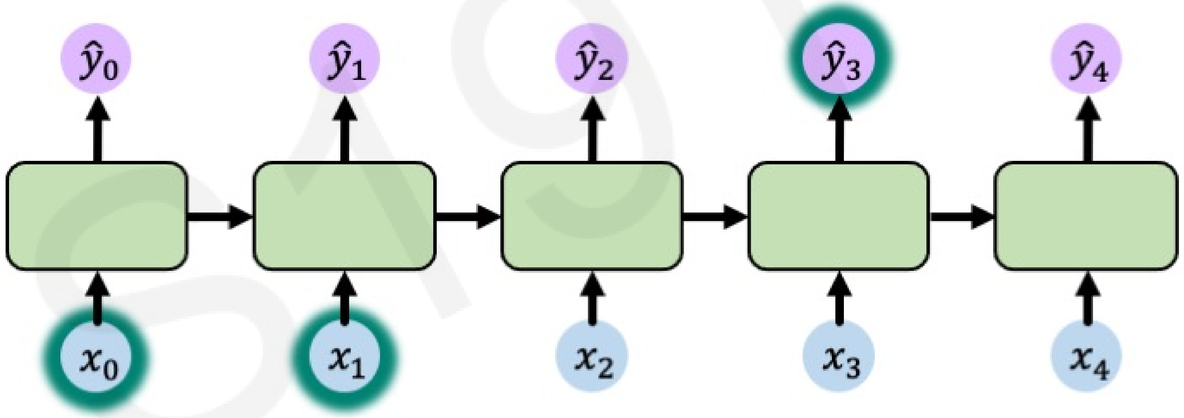
↓

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

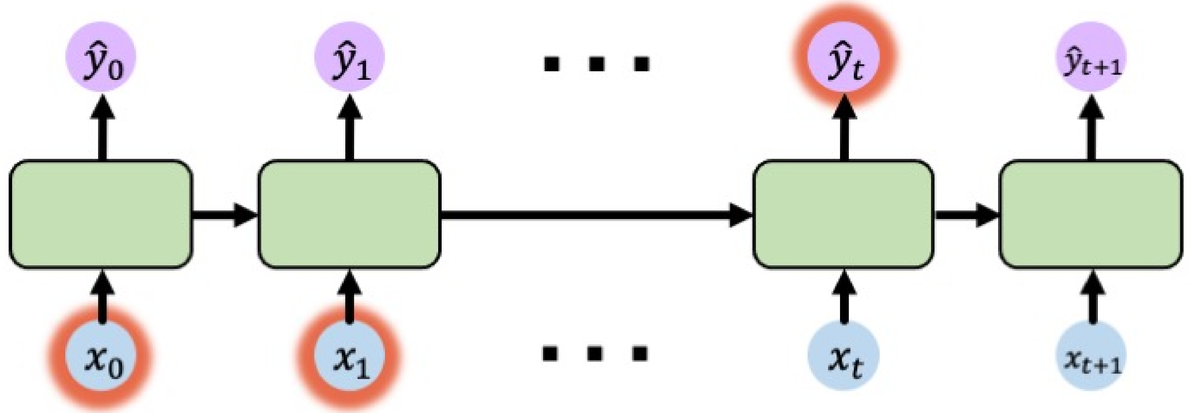
↓

Bias parameters to capture short-term dependencies

"The clouds are in the ___"

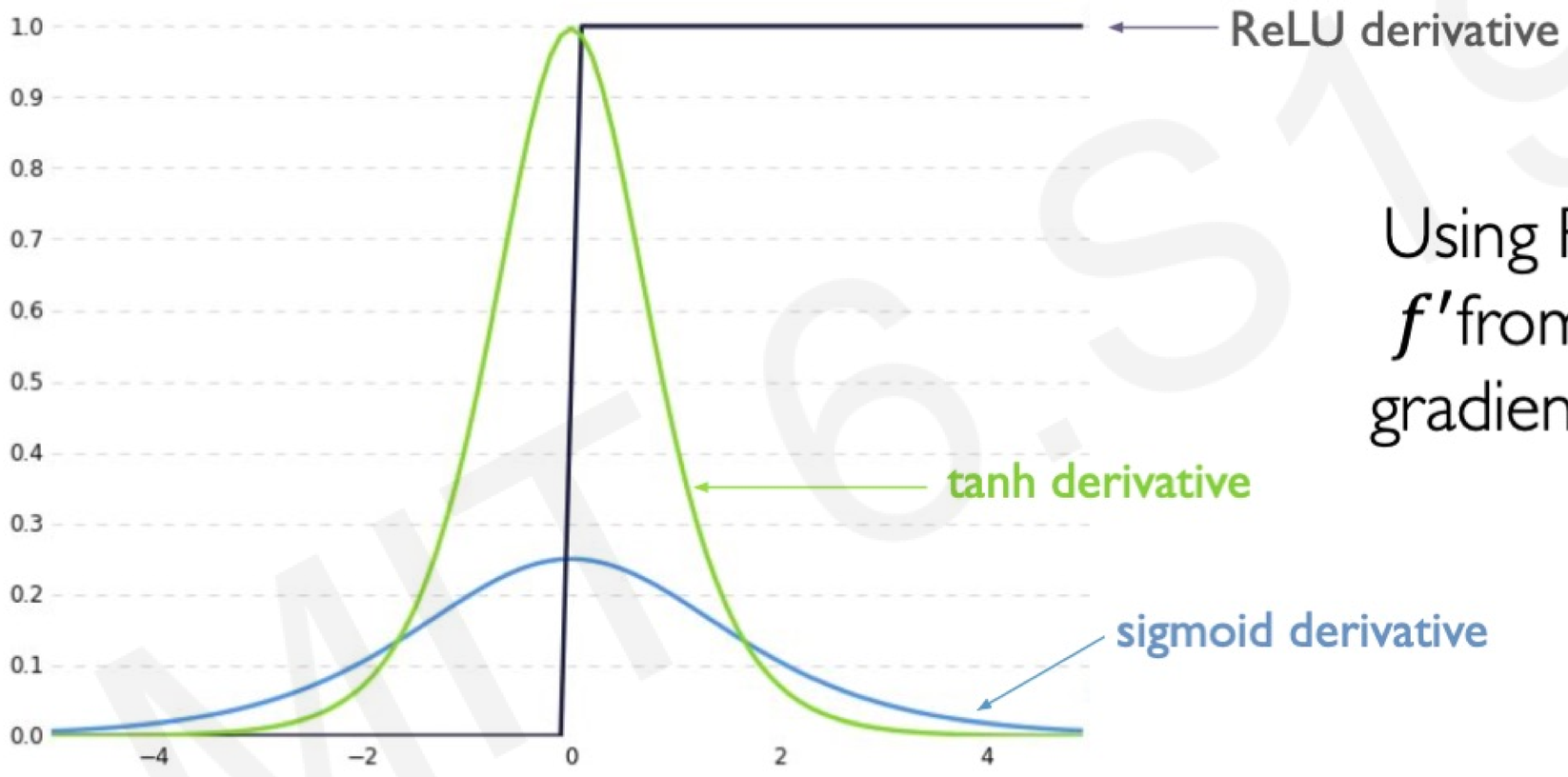


"I grew up in France, … and I speak fluent___ "



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.

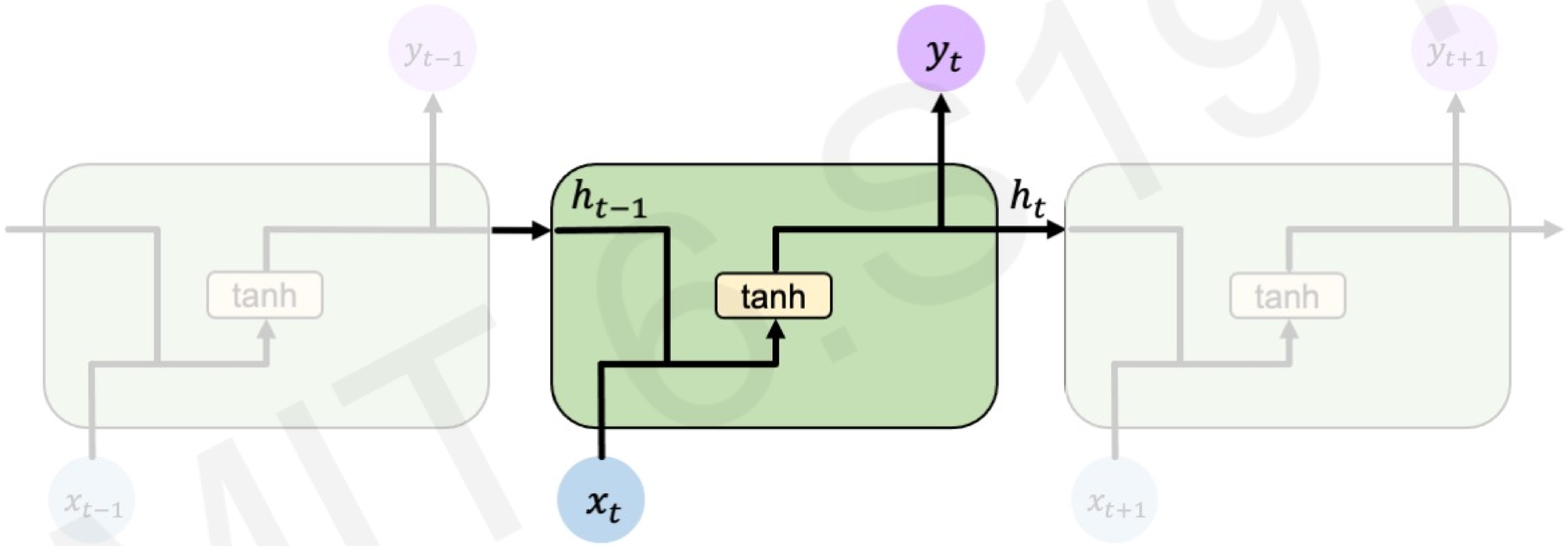
MIT 6.S191

Long Short Term Memory (LSTM) Networks

MIT 6.S191

Standard RNN

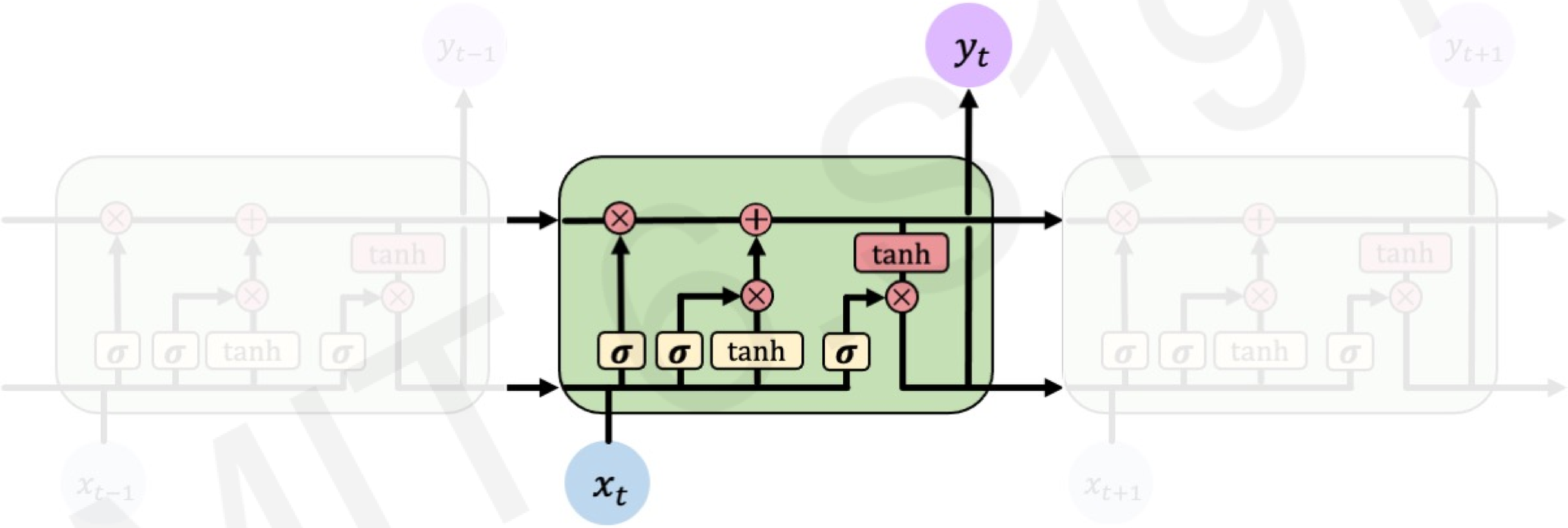
In a standard RNN, repeating modules contain a **simple computation node**



MIT 6.S191

Long Short Term Memory (LSTMs)

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



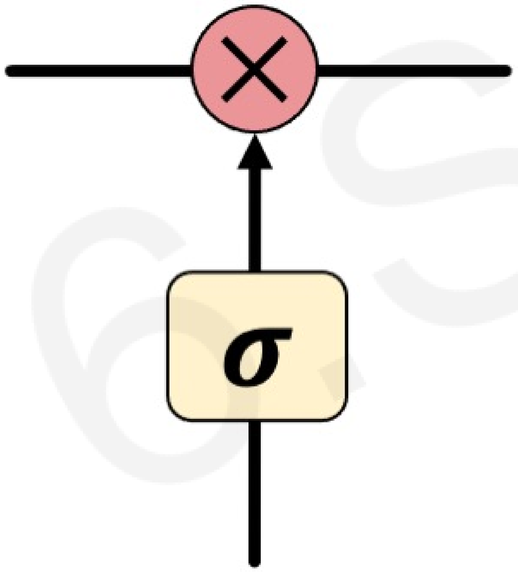
LSTM cells are able to track information throughout many timesteps

```
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**



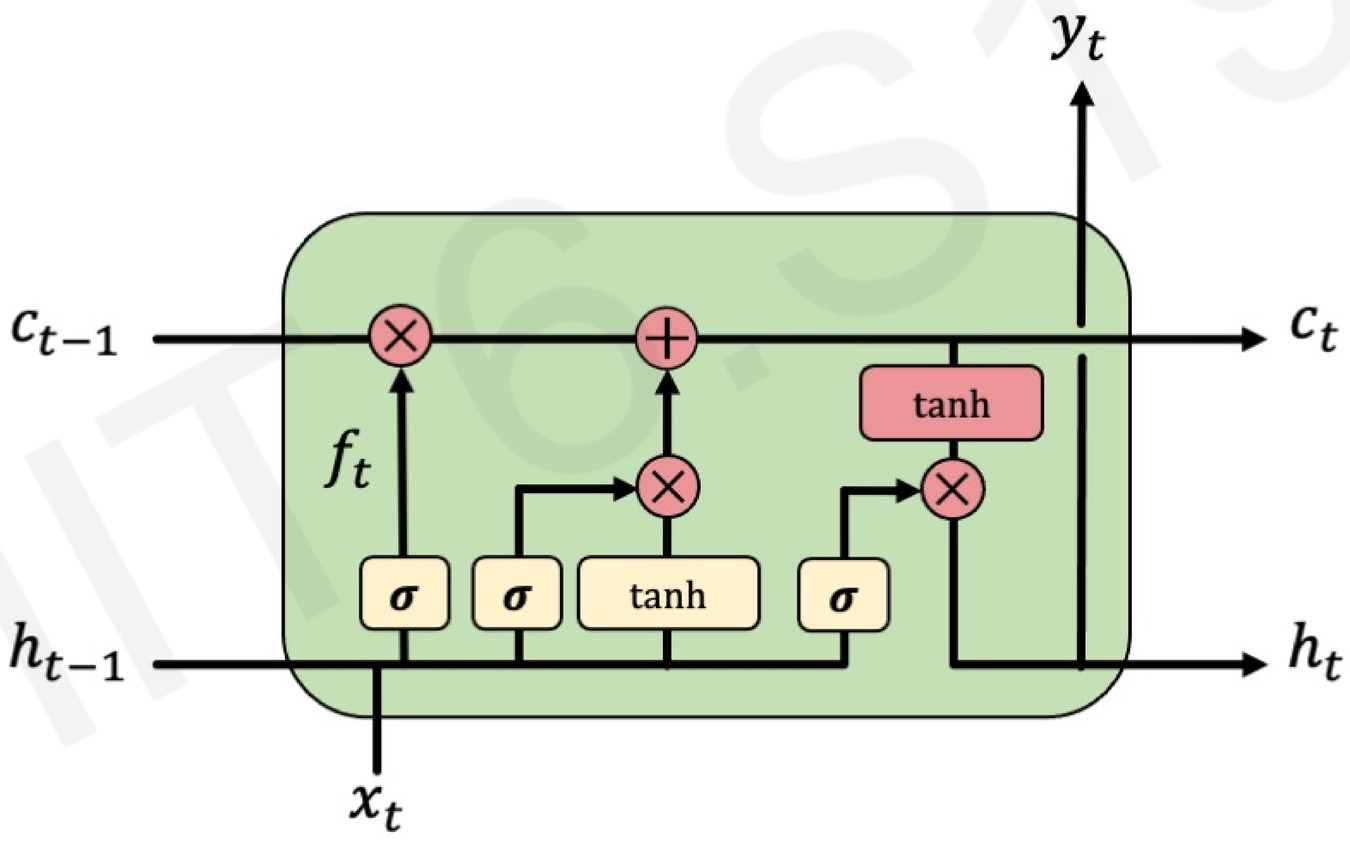
Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

MIT 6.S191

Long Short Term Memory (LSTMs)

How do LSTMs work?

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

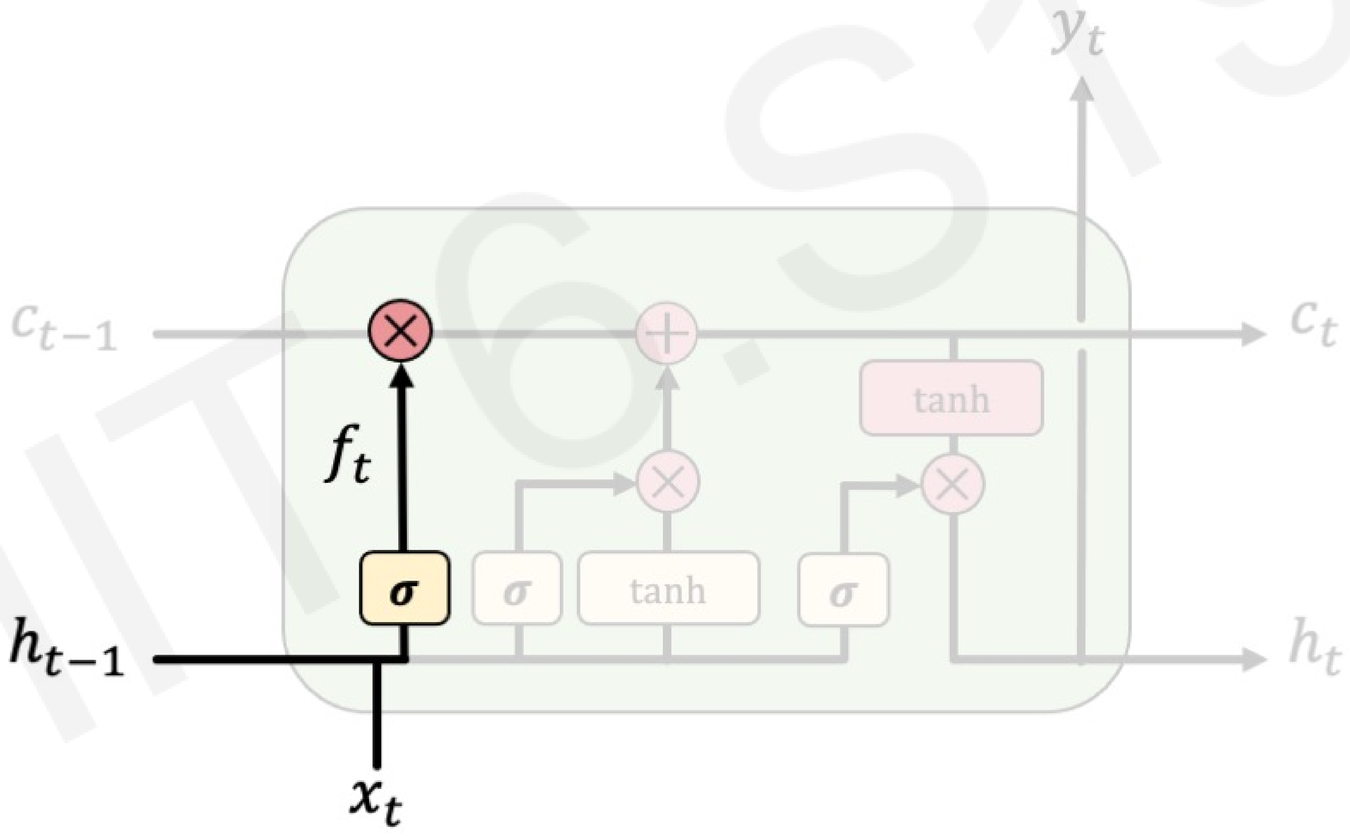


MIT 6.S191

Long Short Term Memory (LSTMs)

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

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

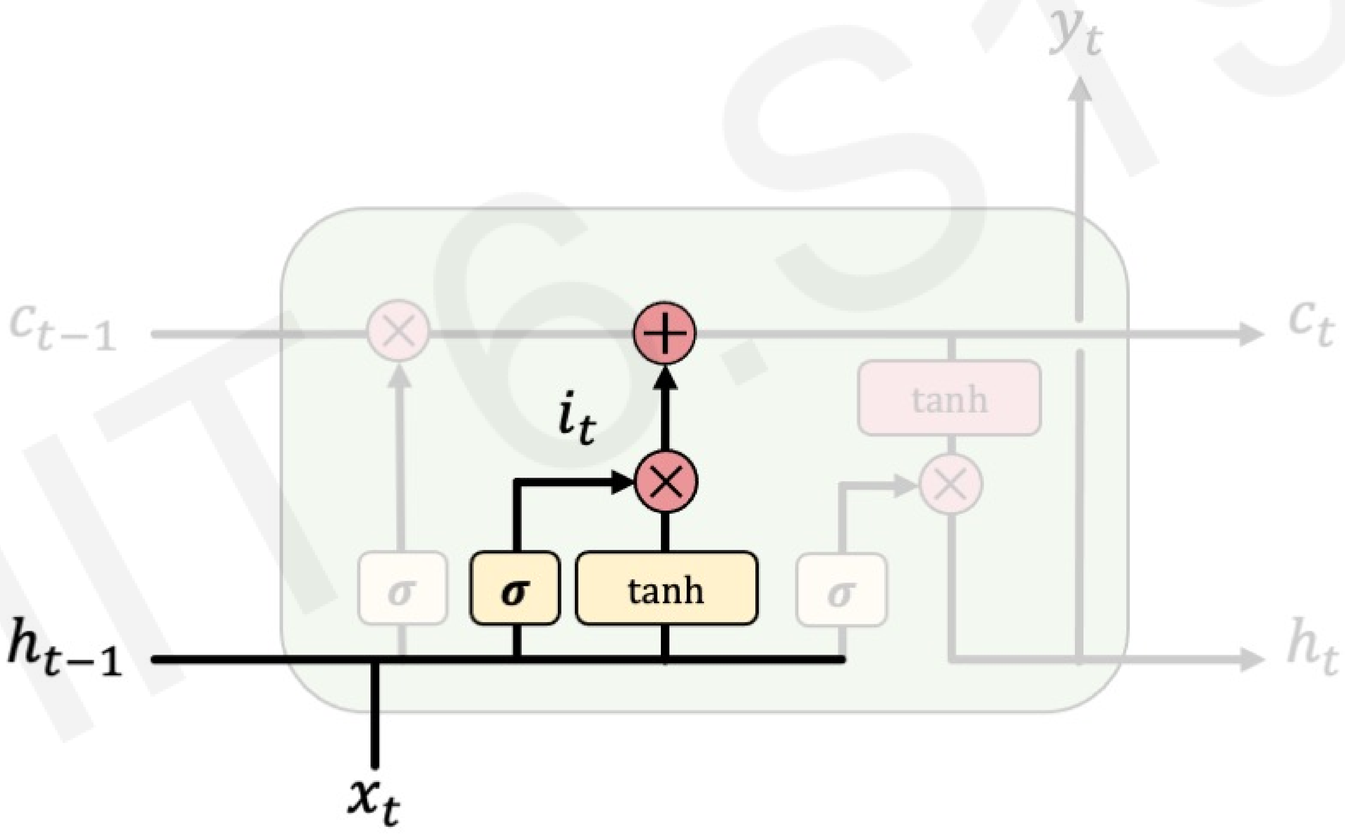


MIT 6.S191

Long Short Term Memory (LSTMs)

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

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

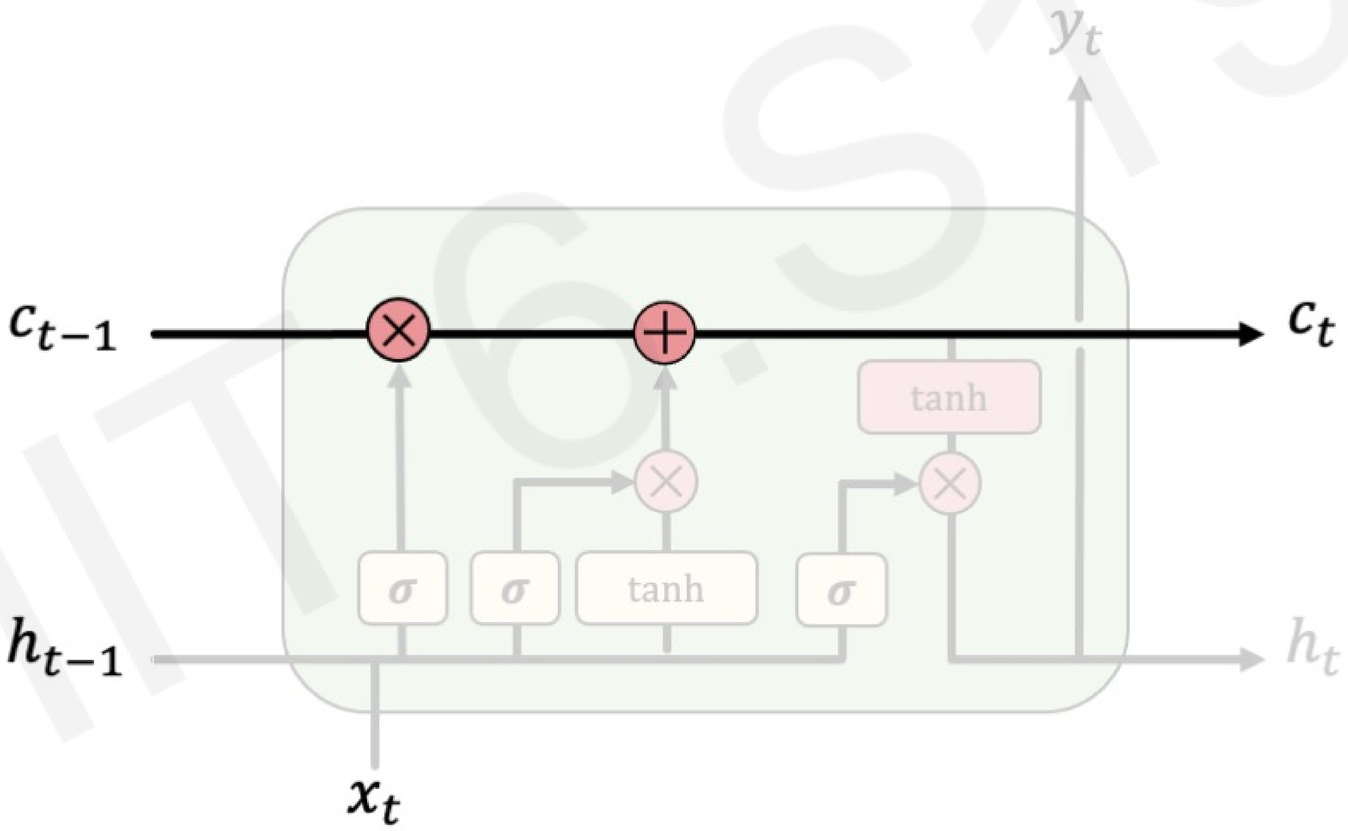


MIT 6.S191

Long Short Term Memory (LSTMs)

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

LSTMs **selectively update** cell state values

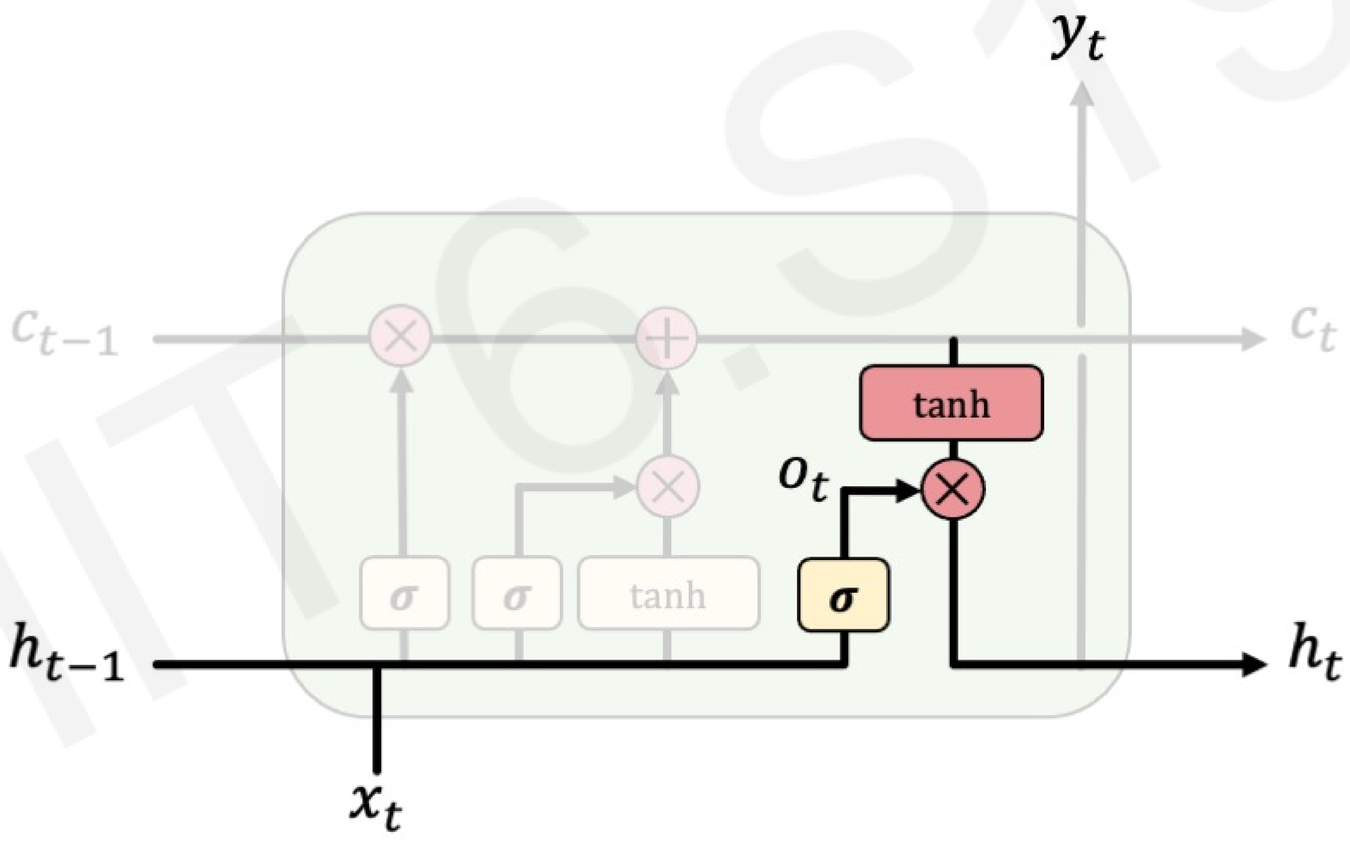


MIT 6.S191

Long Short Term Memory (LSTMs)

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

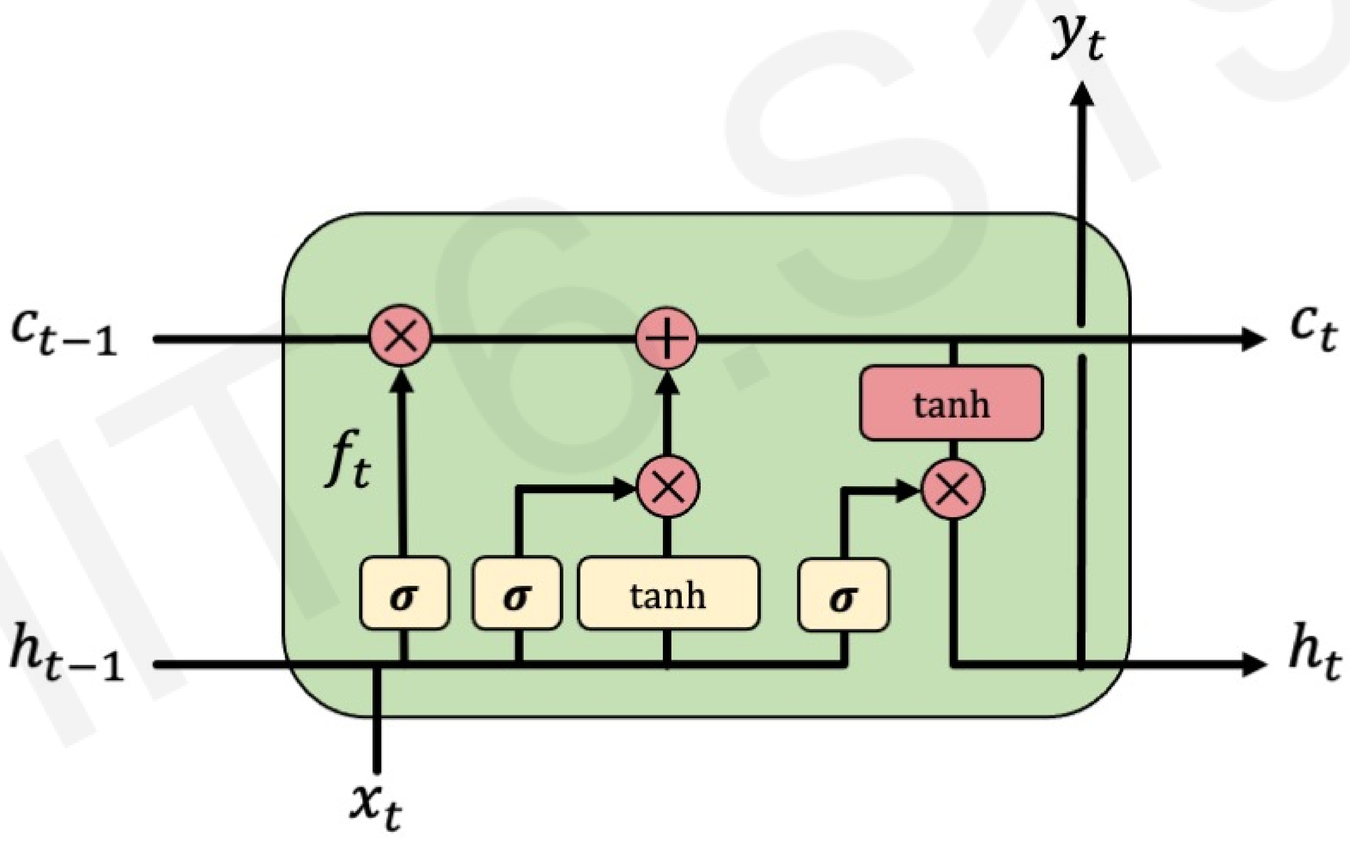
The **output gate** controls what information is sent to the next time step



MIT 6.S191

Long Short Term Memory (LSTMs)

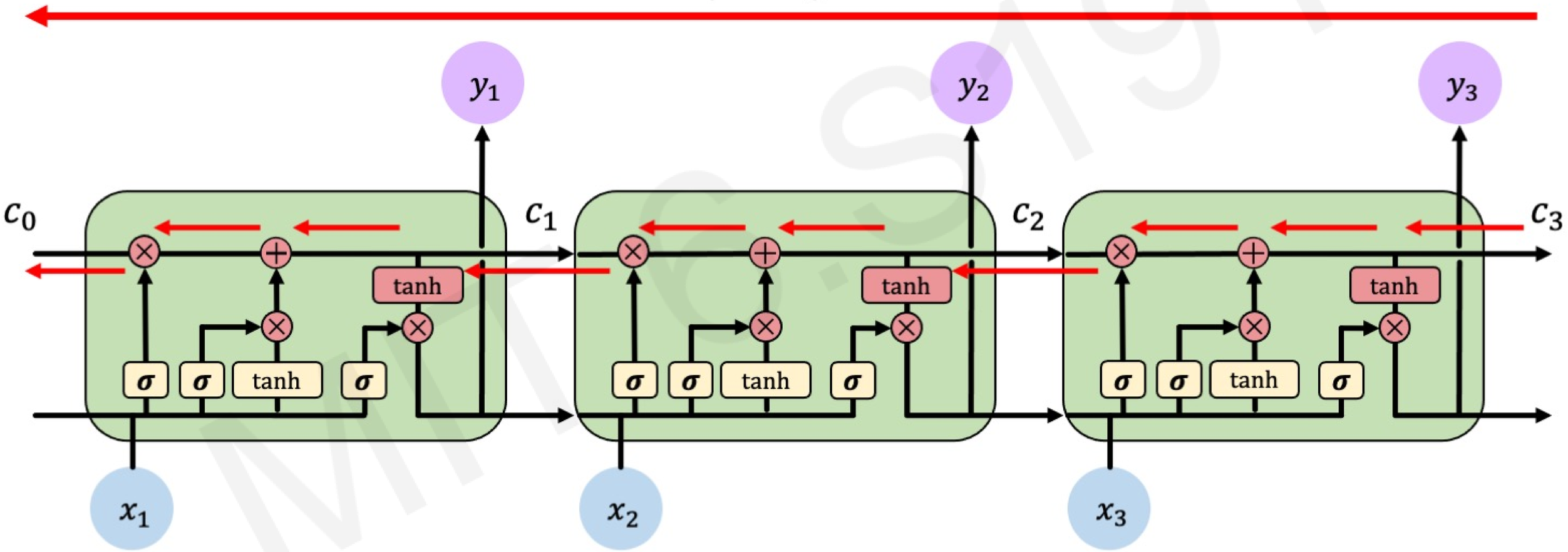
1) Forget 2) Store 3) Update 4) Output



MIT 6.S191

LSTM Gradient Flow

Uninterrupted gradient flow!



LSTMs: Key Concepts

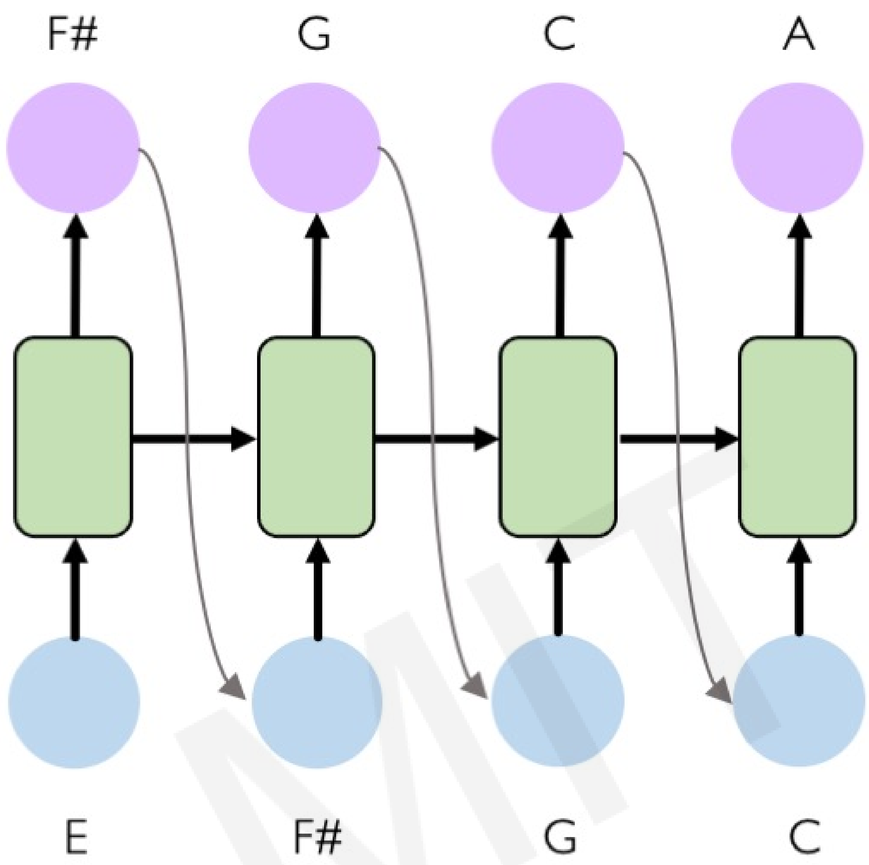
1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
   - **Forget** gate gets rid of irrelevant information
   - **Store** relevant information from current input
   - Selectively **update** cell state
   - **Output** gate returns a filtered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

MIT 6.S191

RNN Applications

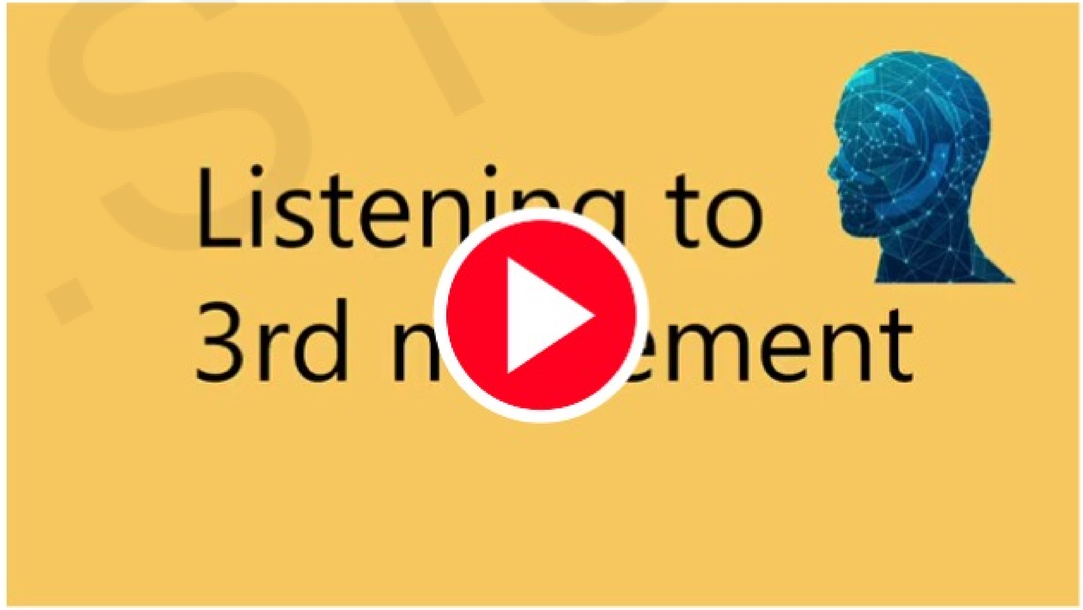
MIT 6.S191

Example Task: Music Generation



Input: sheet music

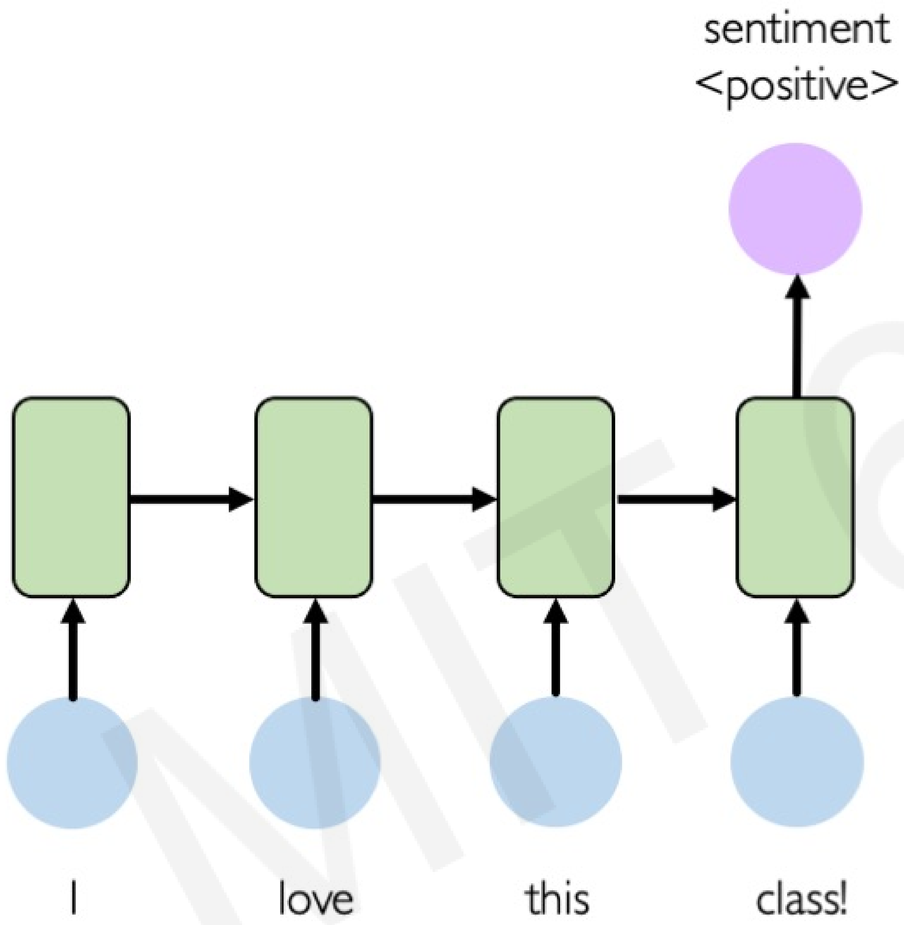
Output: next character in sheet music



6.S191 Lab!

MIT 6.S191

Example Task: Sentiment Classification



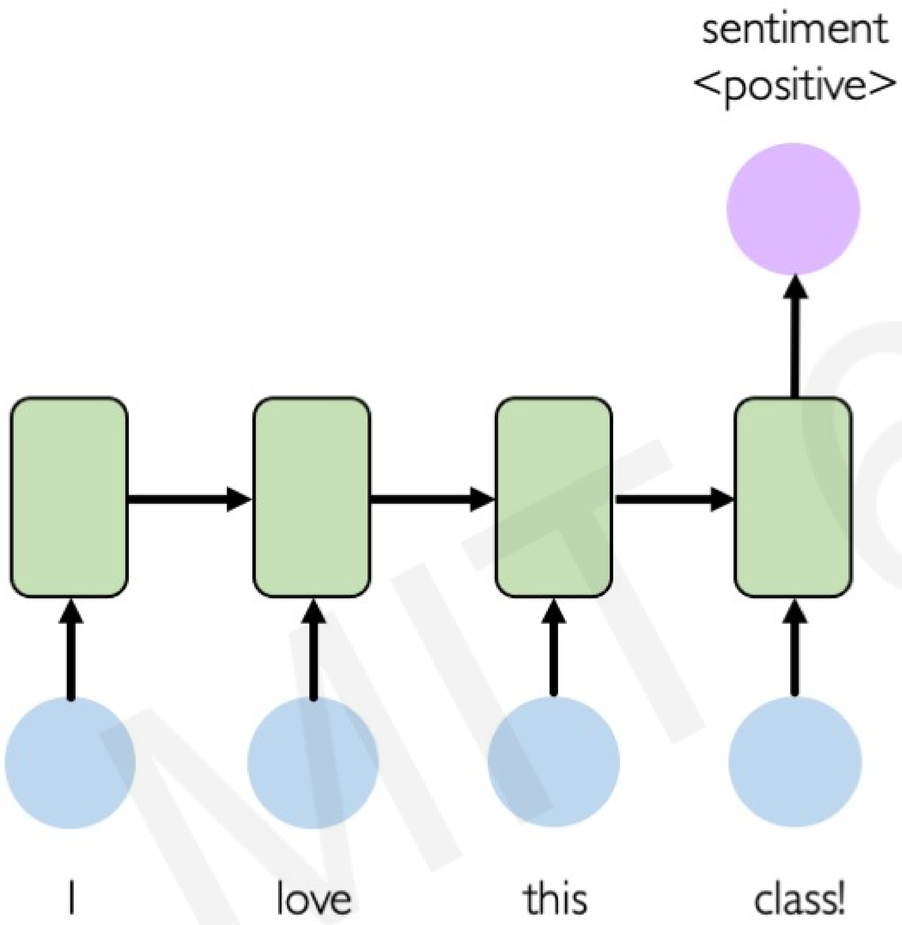
Input: sequence of words

Output: probability of having positive sentiment

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

MIT 6.S191

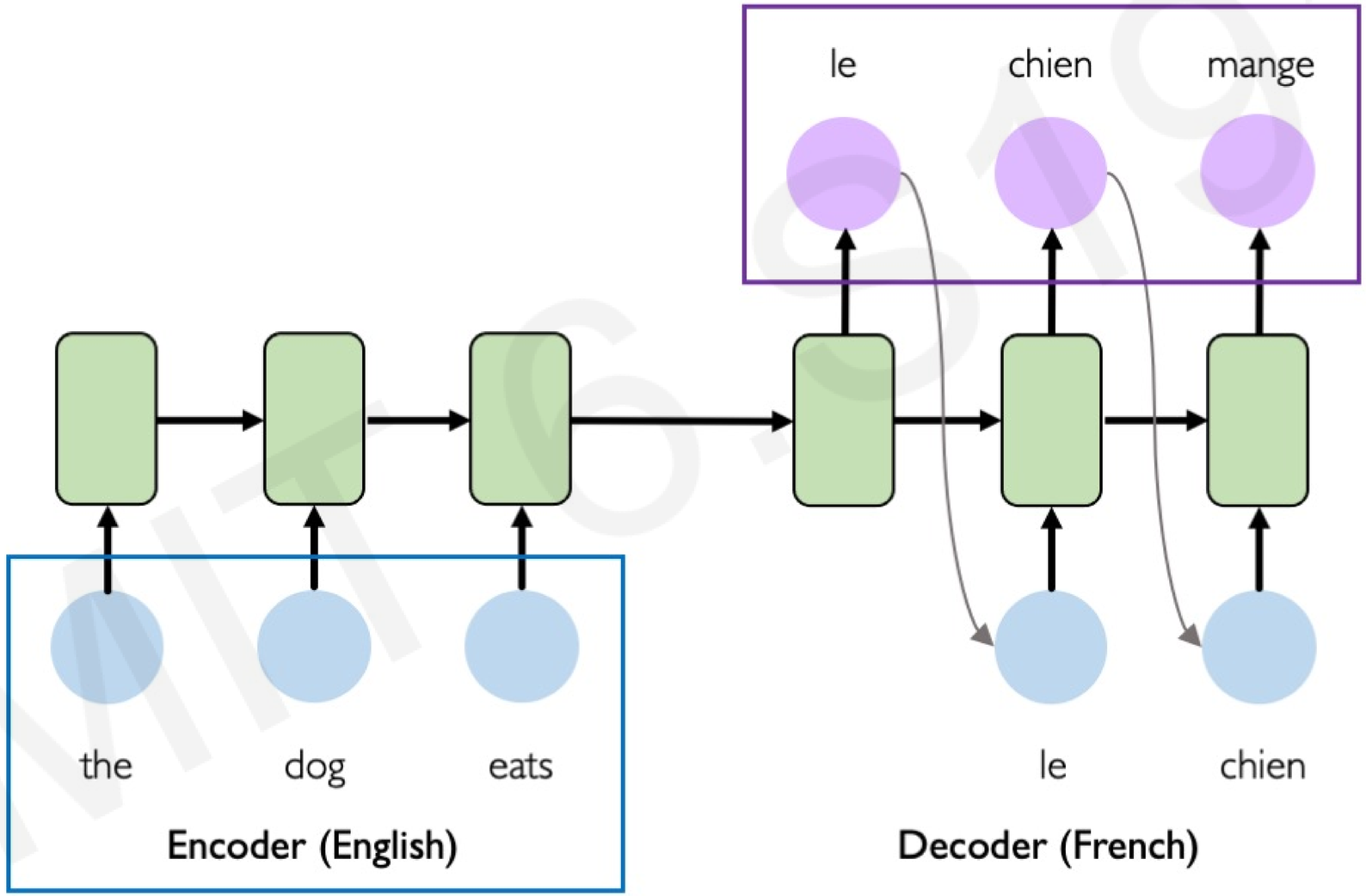
Example Task: Sentiment Classification



Tweet sentiment classification

MIT 6.S191

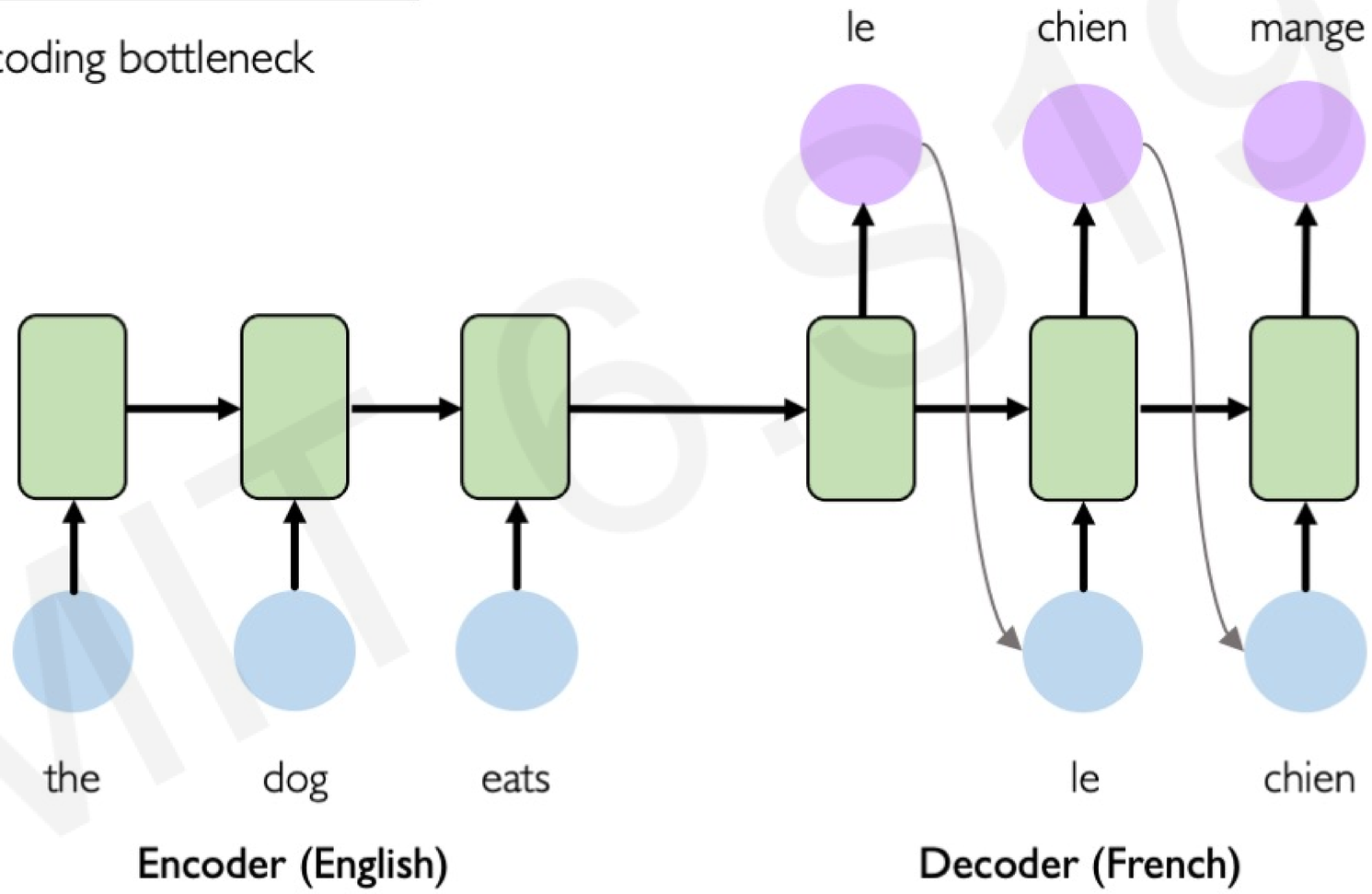
Example Task: Machine Translation



Example Task: Machine Translation

Potential Issues

- Encoding bottleneck

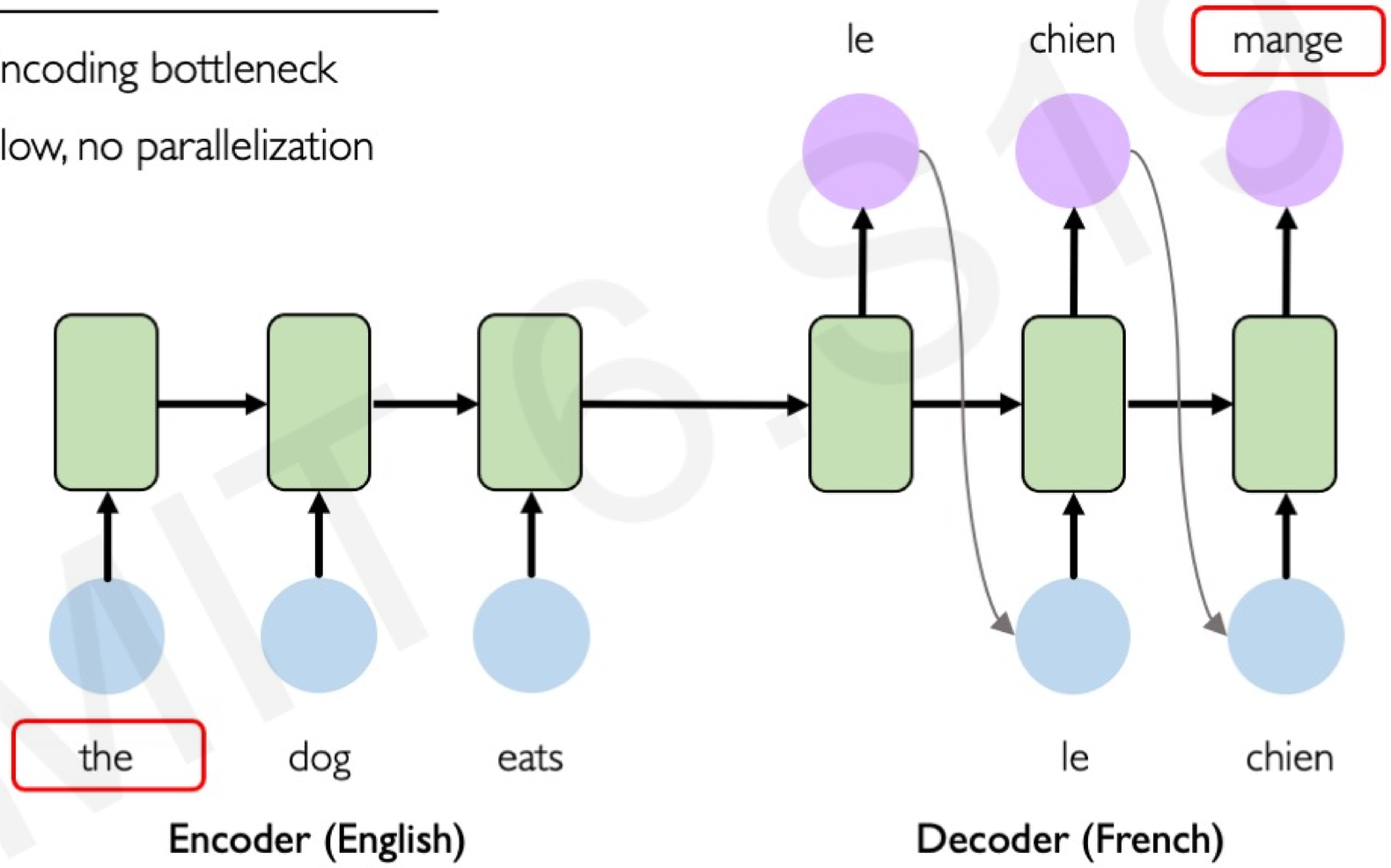


MIT 6.S191

Example Task: Machine Translation

Potential Issues

- Encoding bottleneck
- Slow, no parallelization

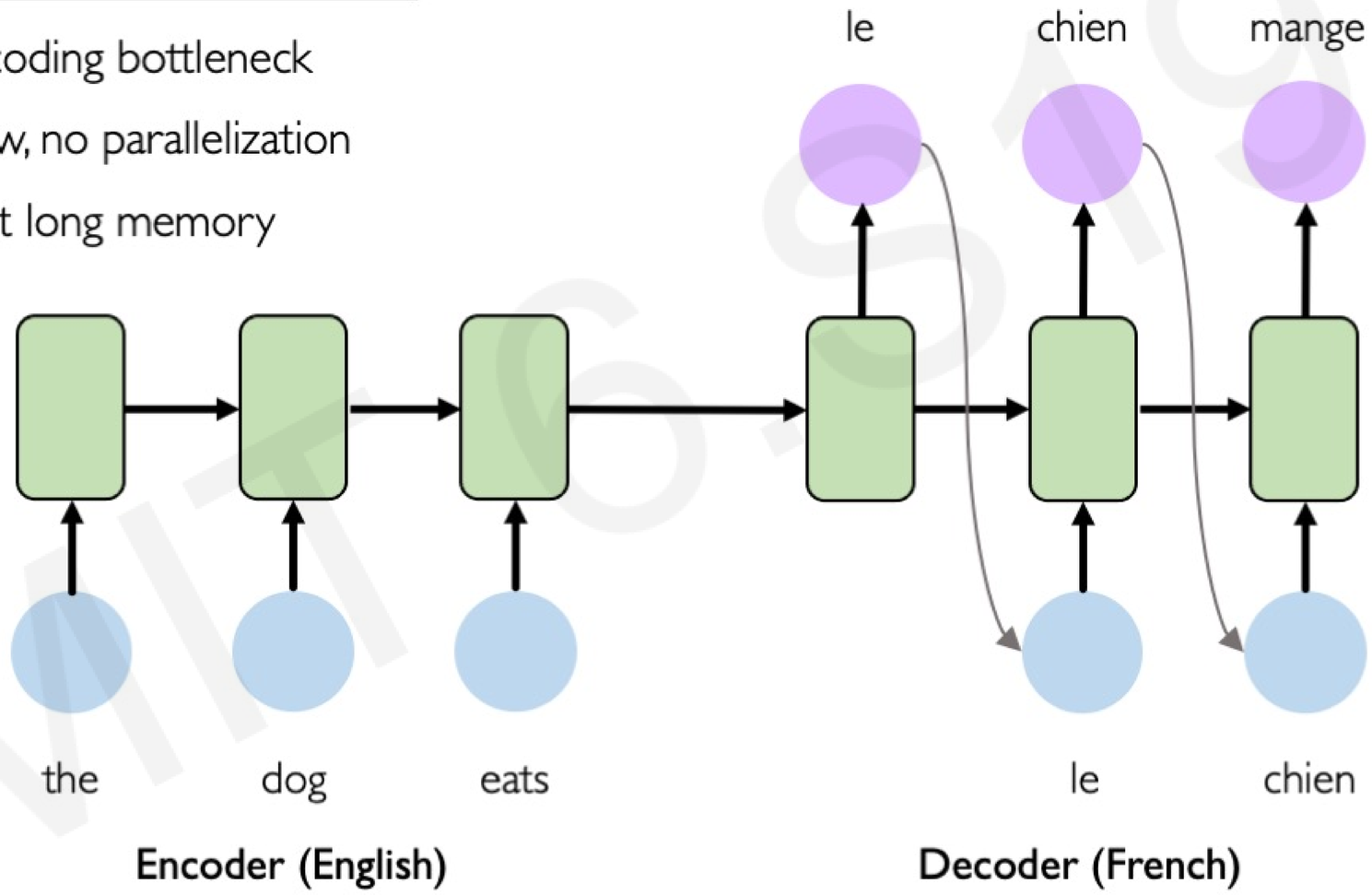


MIT 6.S191

Example Task: Machine Translation

Potential Issues

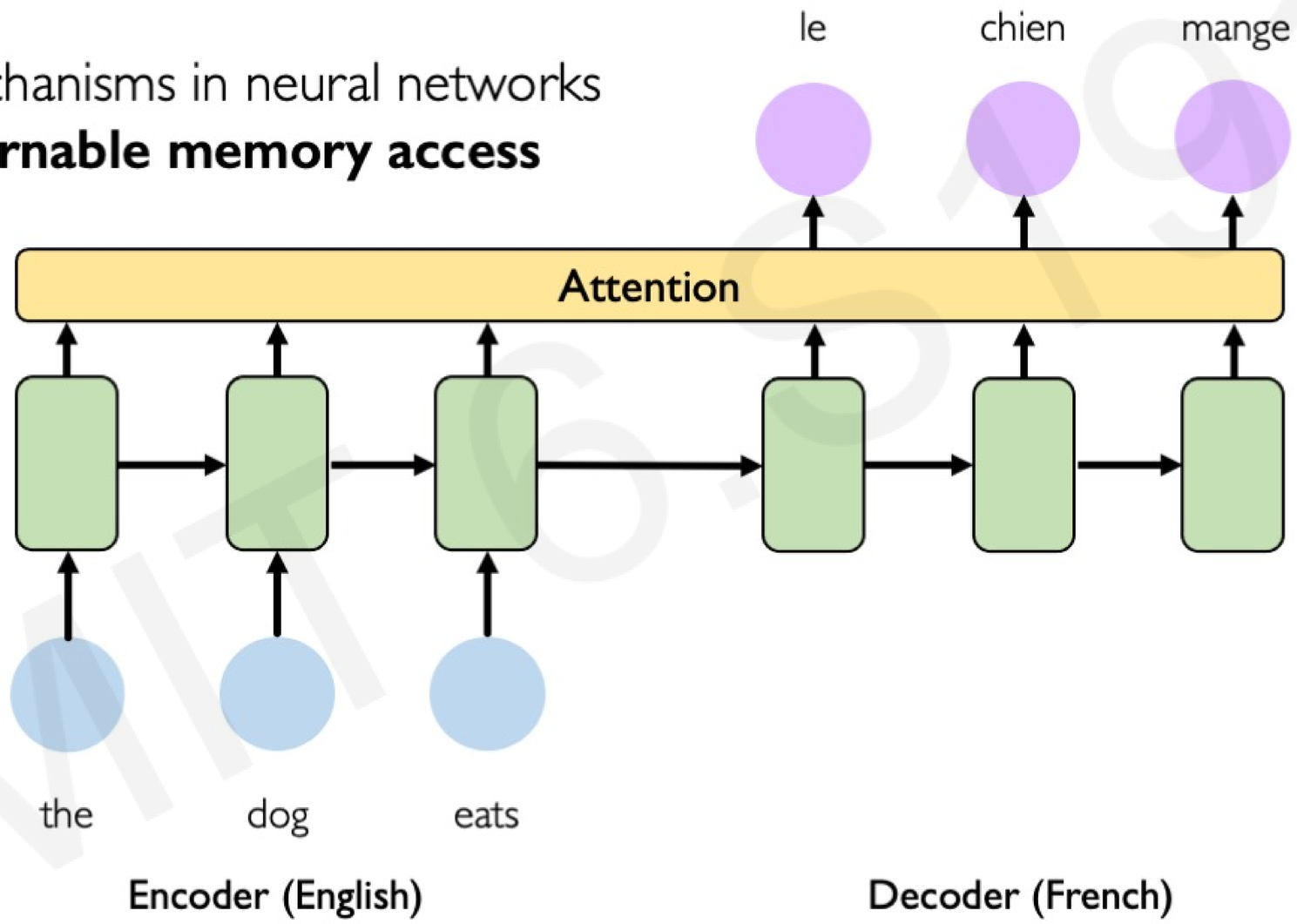
- Encoding bottleneck
- Slow, no parallelization
- Not long memory



MIT 6.S191

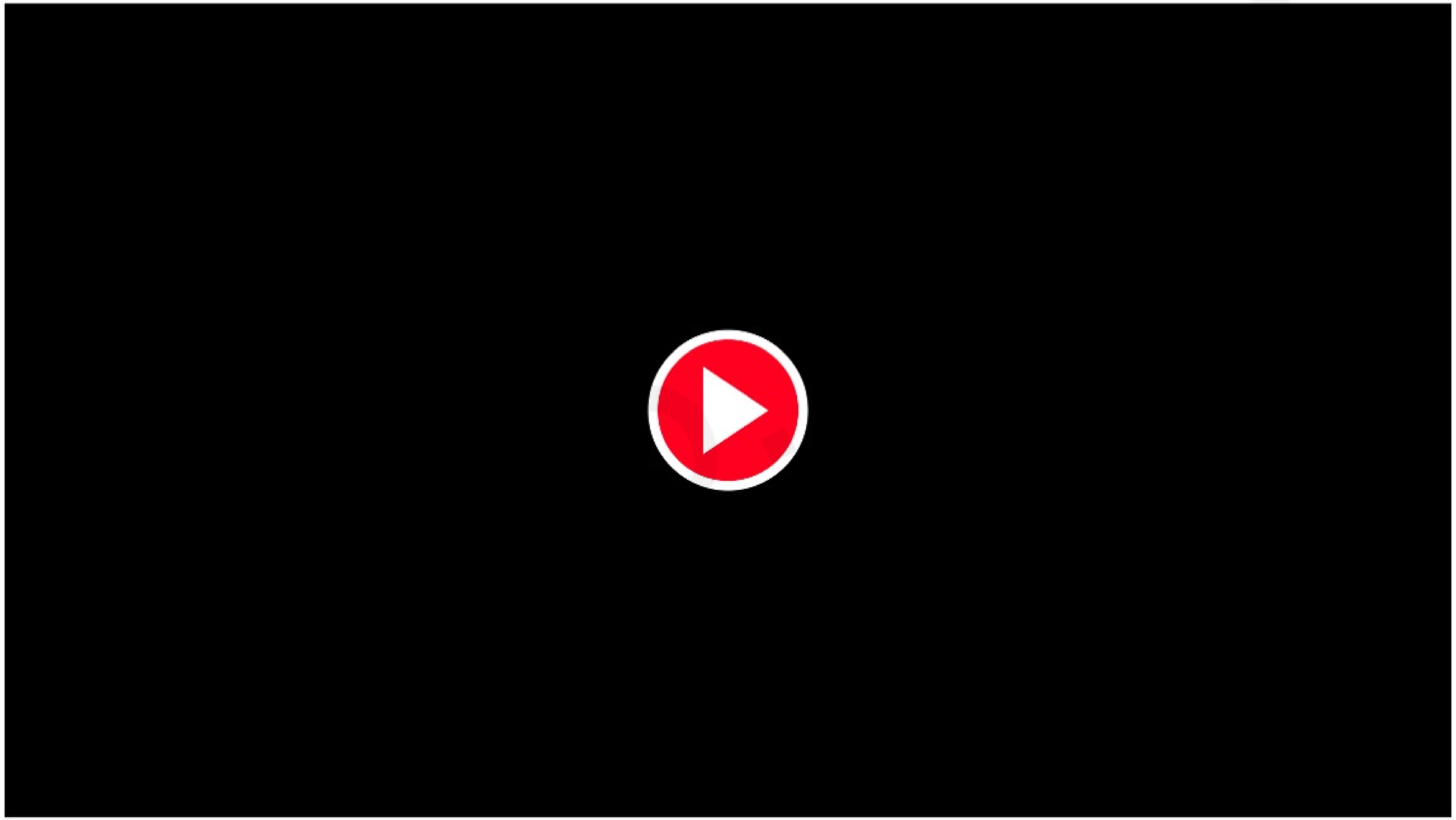
Example Task: Machine Translation

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



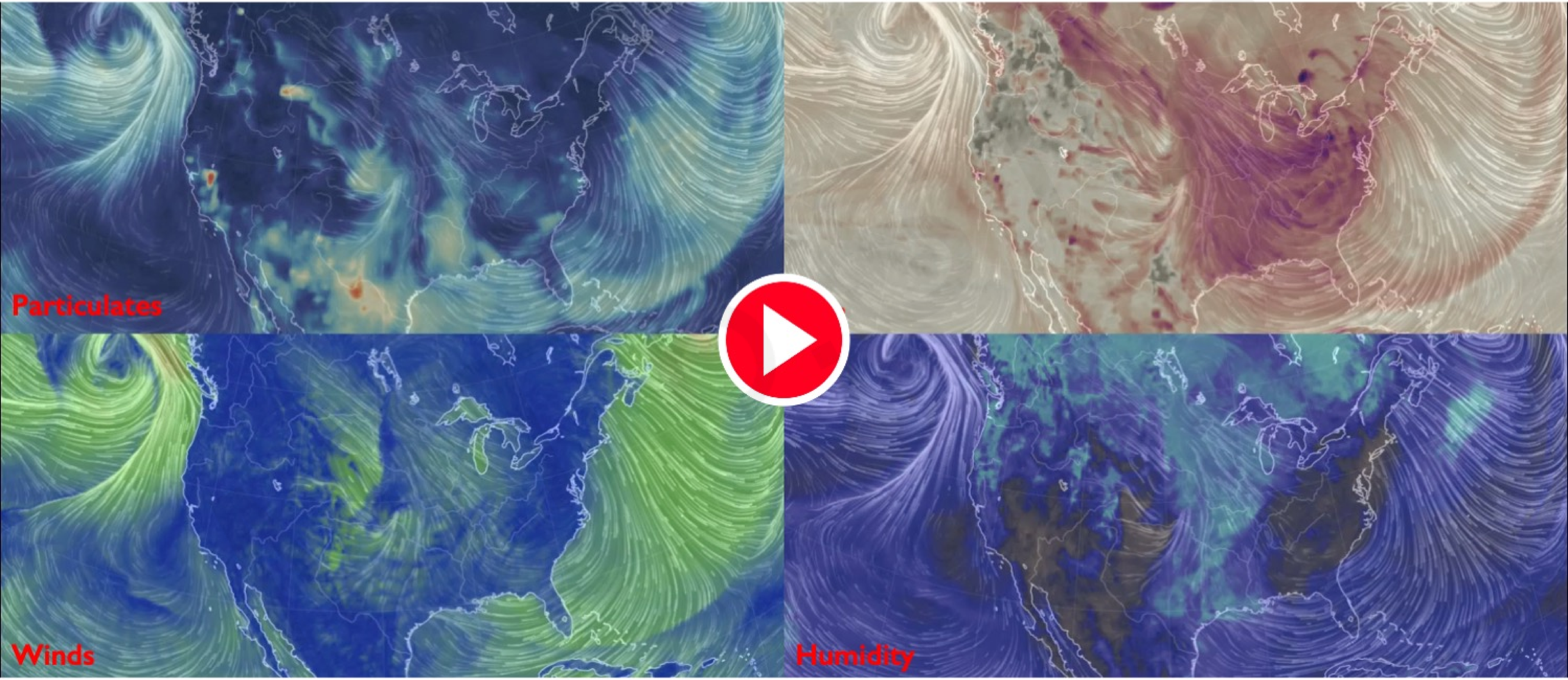
MIT 6.S191

Application: Trajectory Prediction for Self-Driving Cars



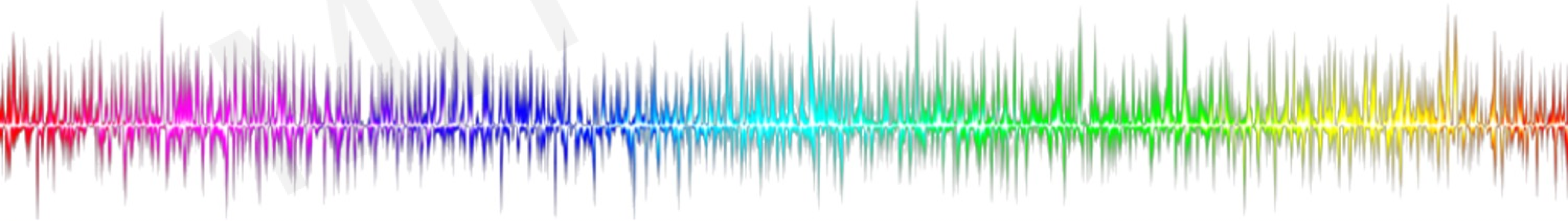
Waymo. 1/19/21

Application: 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? Come to the class Gather.Town!