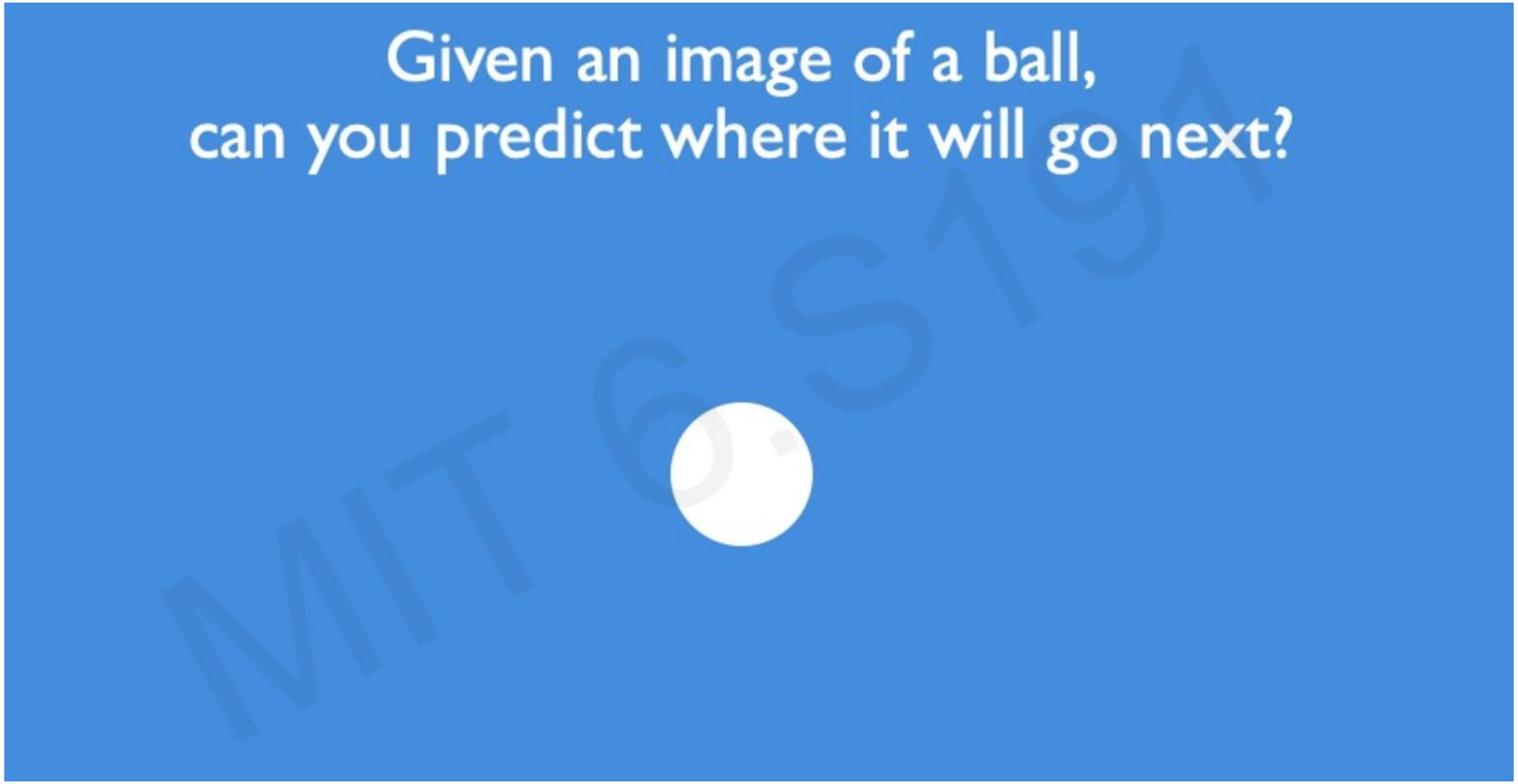
Intro. to Sequence Modelling

Rina BUOY

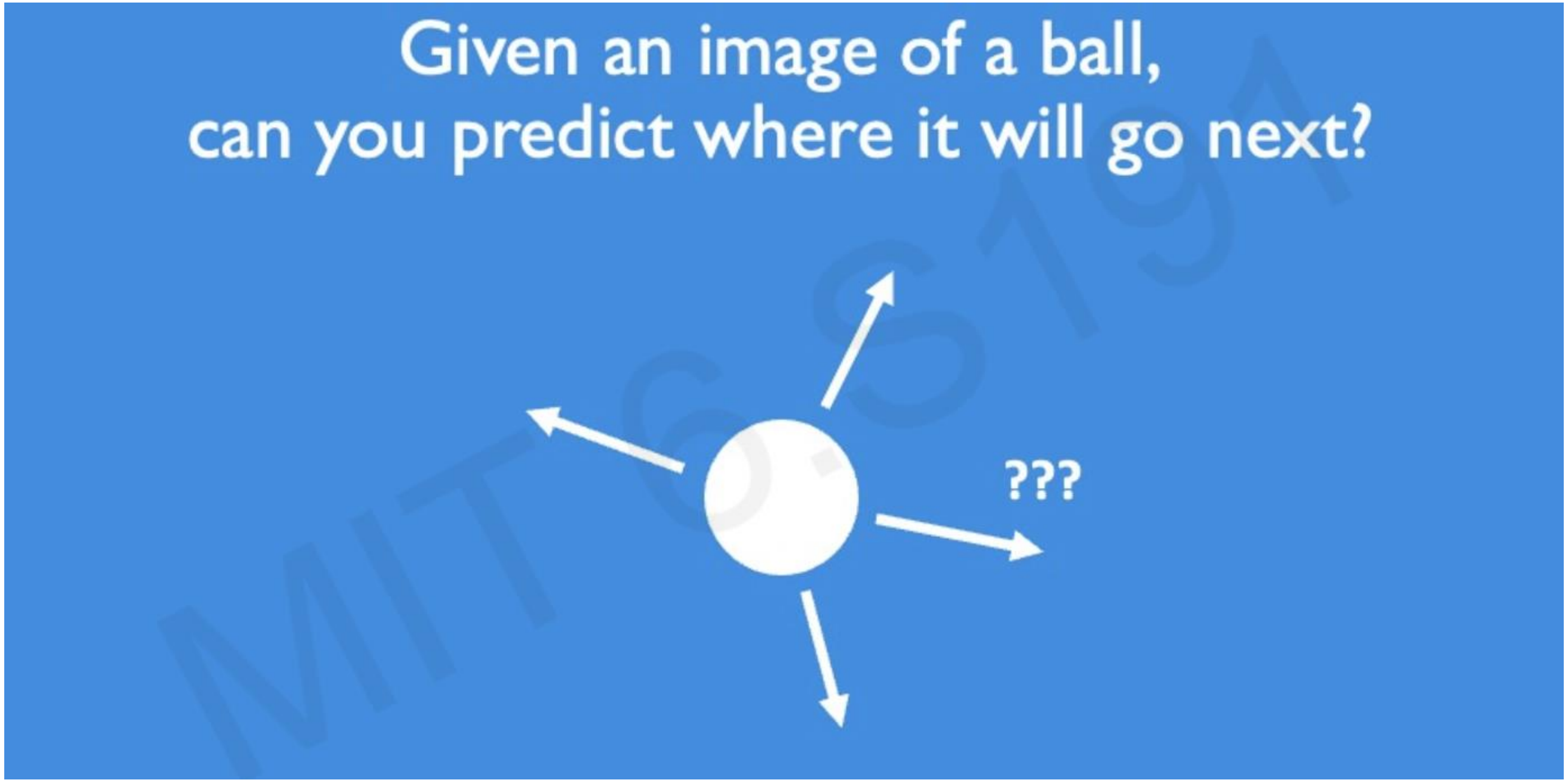
AMERICAN UNIVERSITY
OF PHNOM PENH

STUDY LOCALLY. LIVE GLOBALLY.

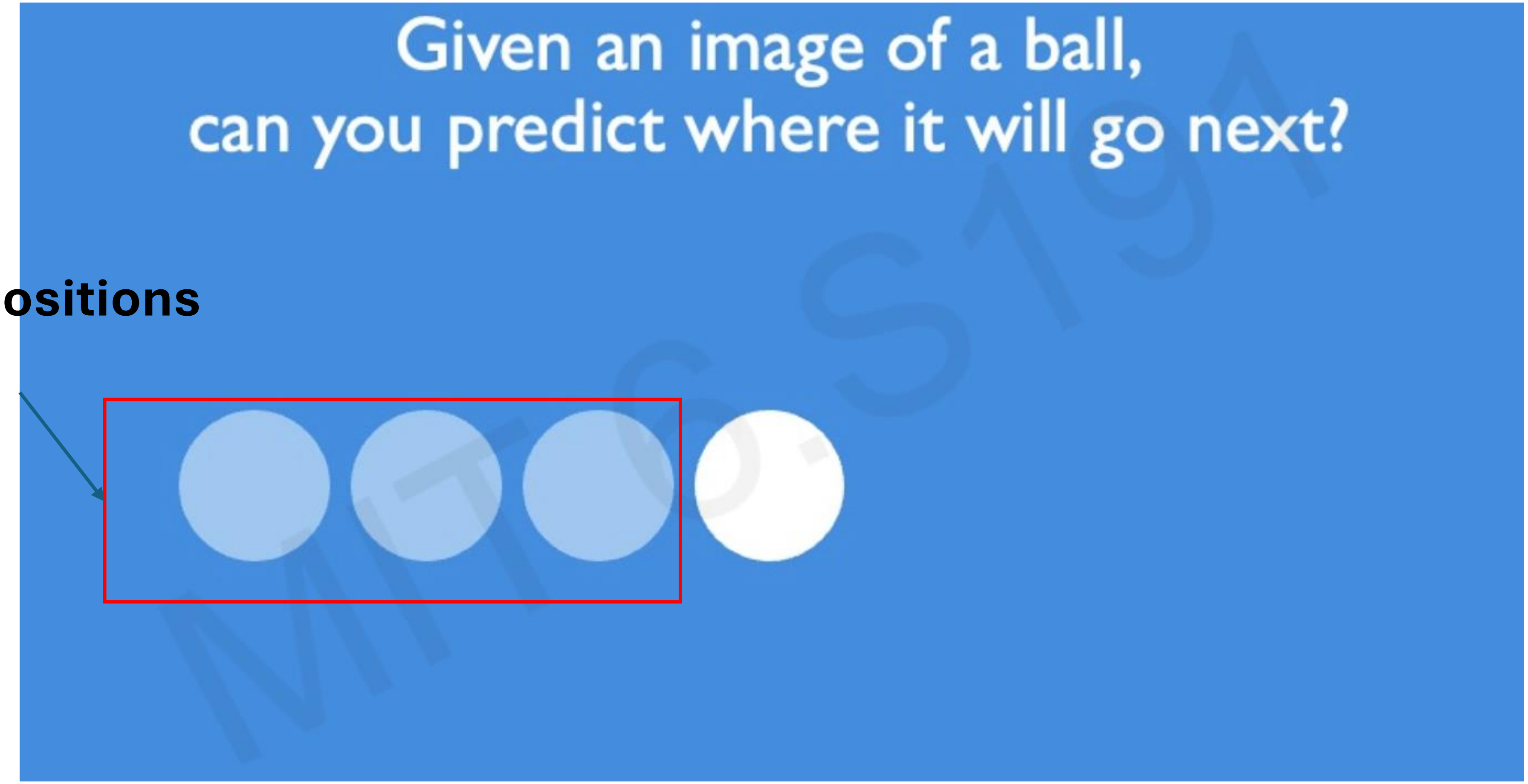
Why Sequence Modelling?



Why Sequence Modelling?

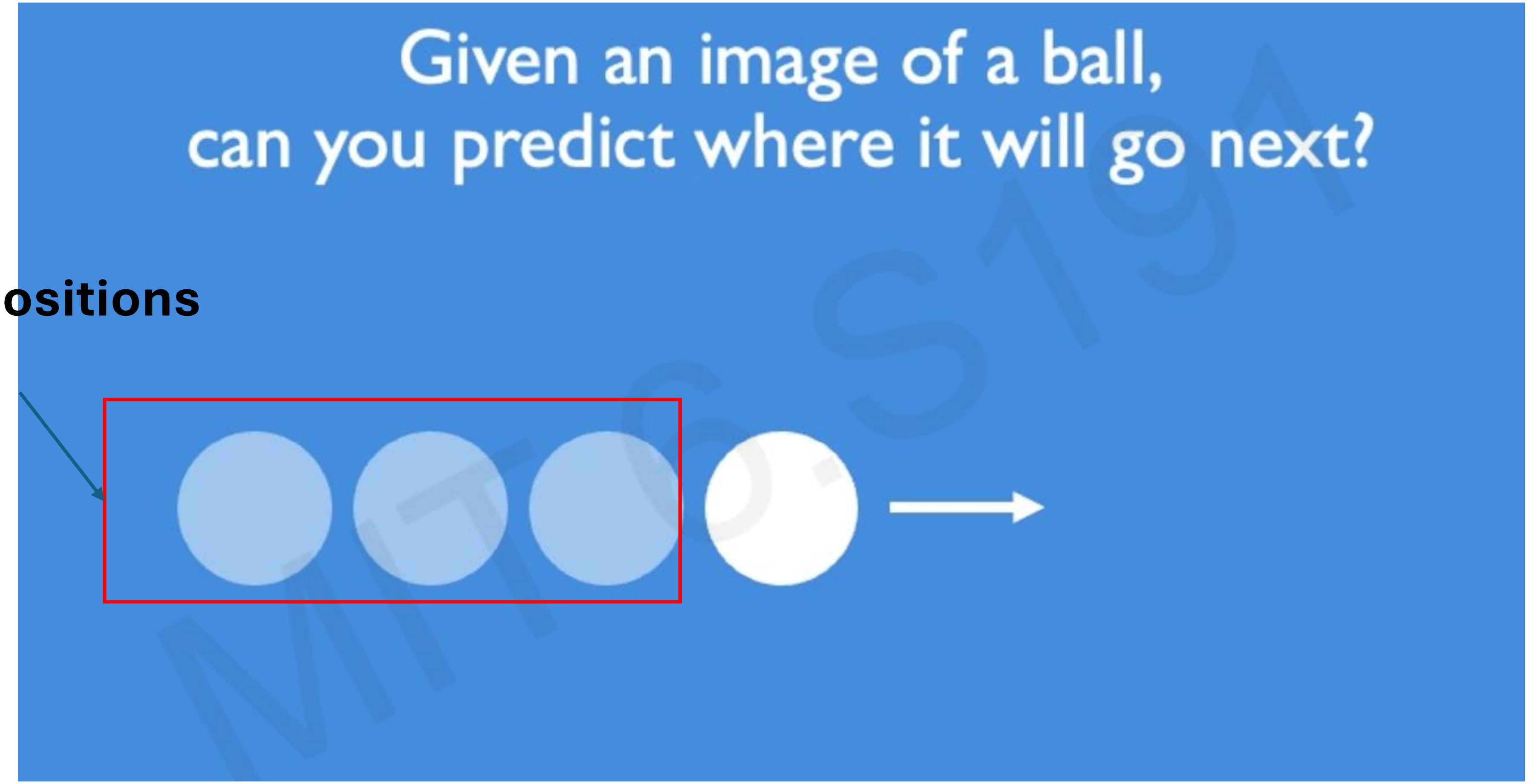


Why Sequence Modelling?



Previous positions

Why Sequence Modelling?

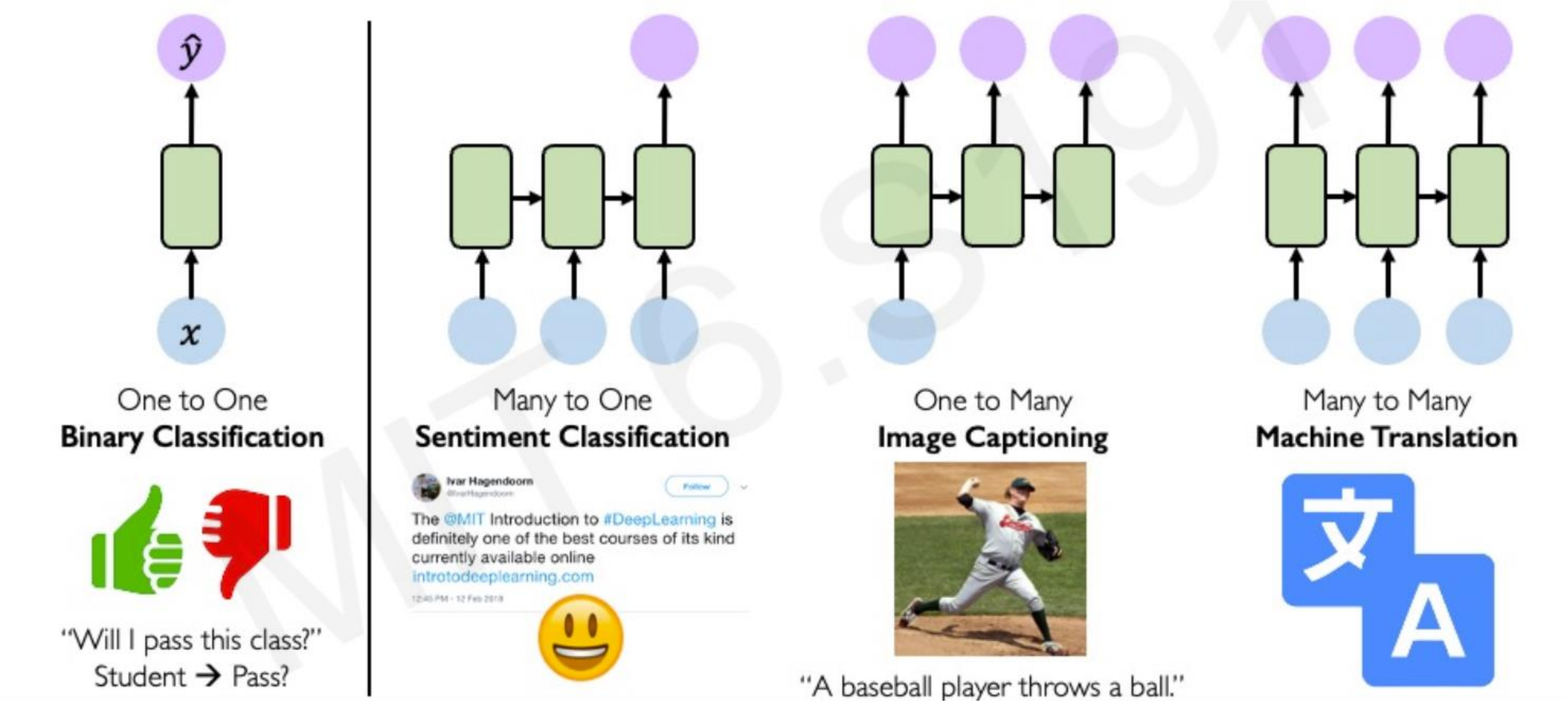


Previous positions

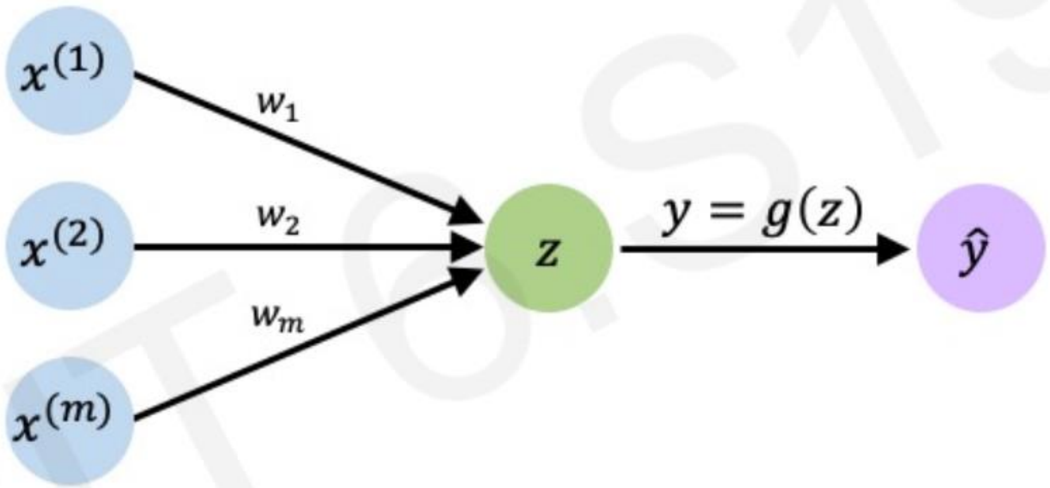
Other sequential data

Audio

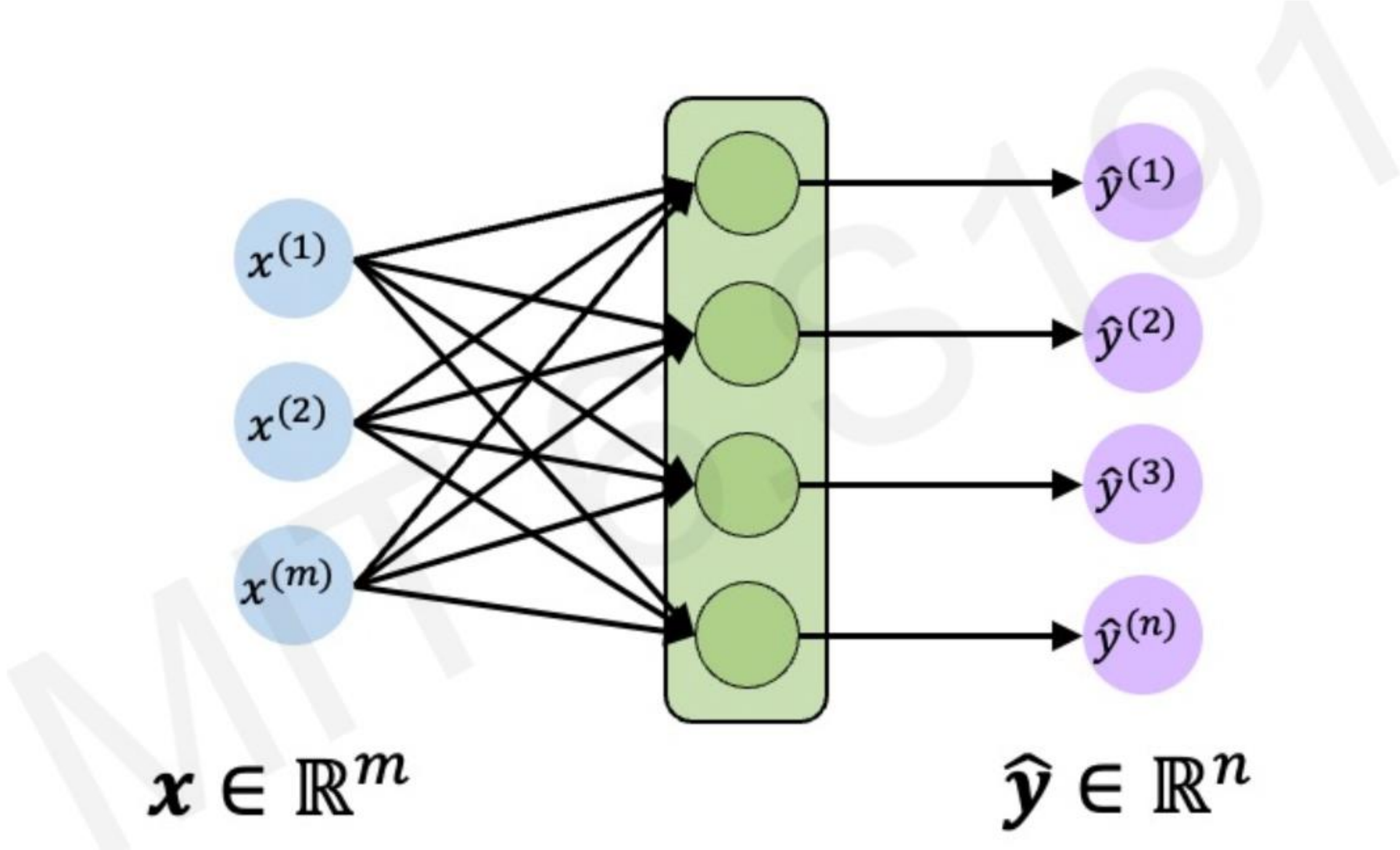
Sequences in Applications



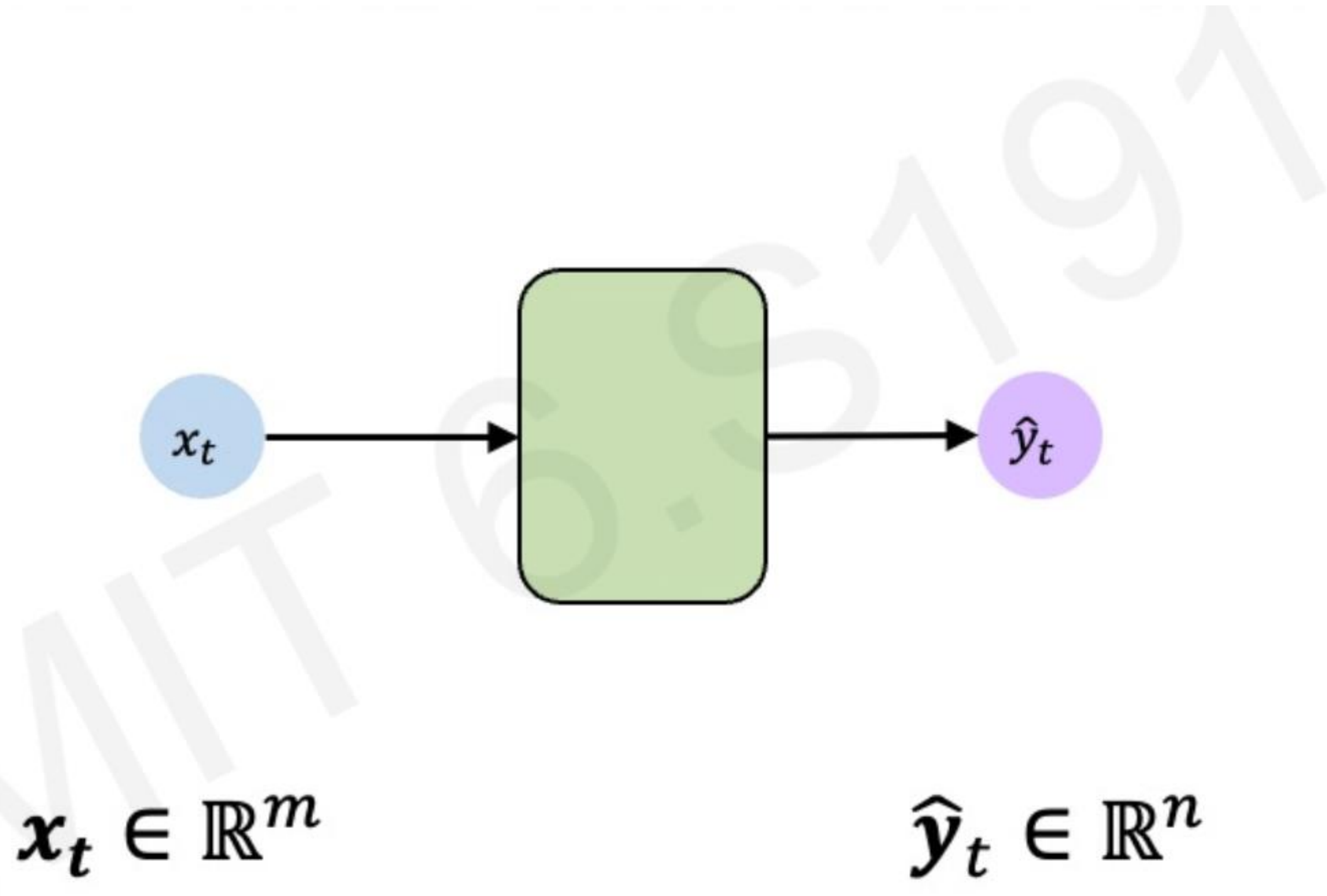
Perceptron



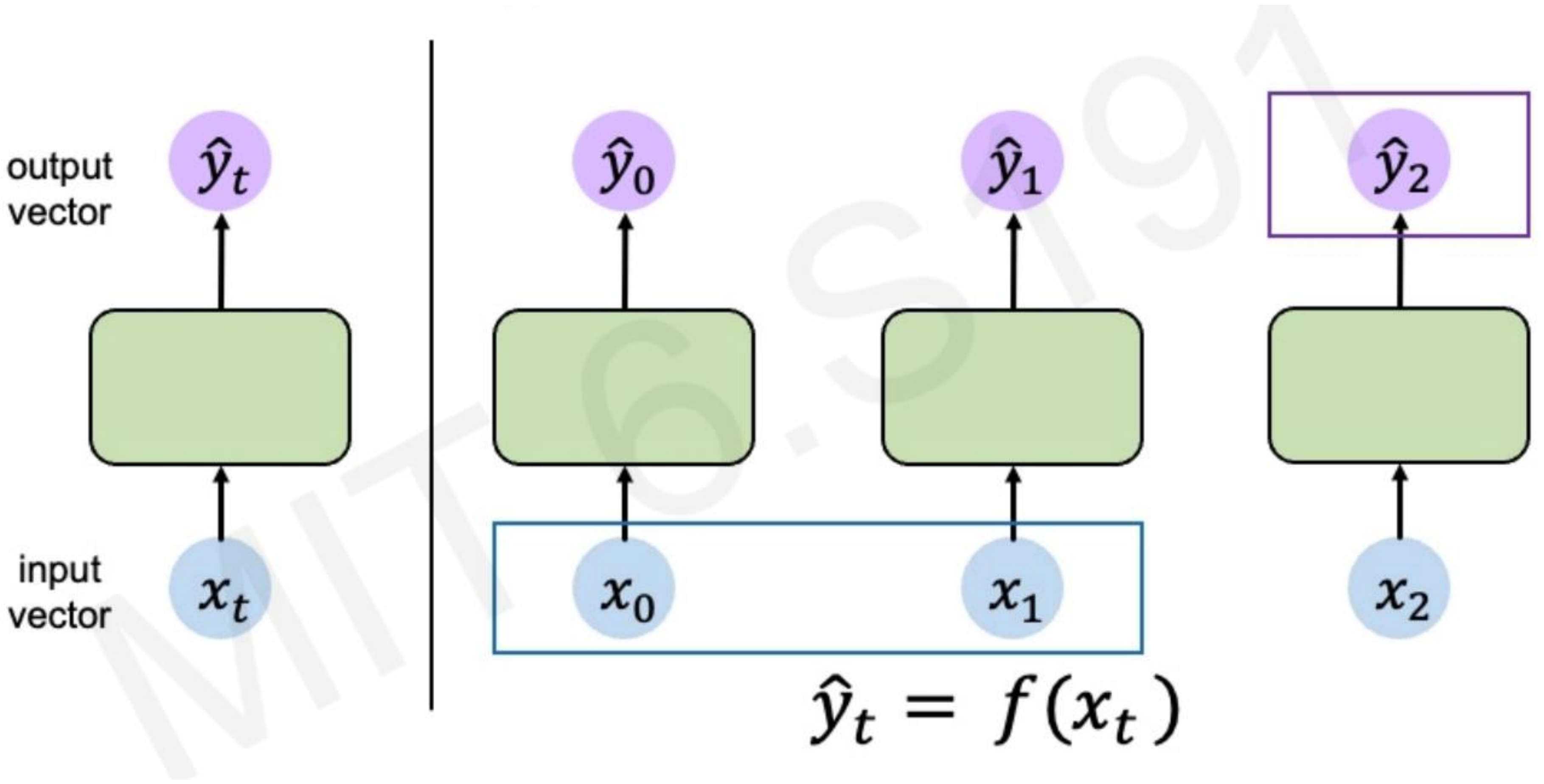
Simple Neural Networks



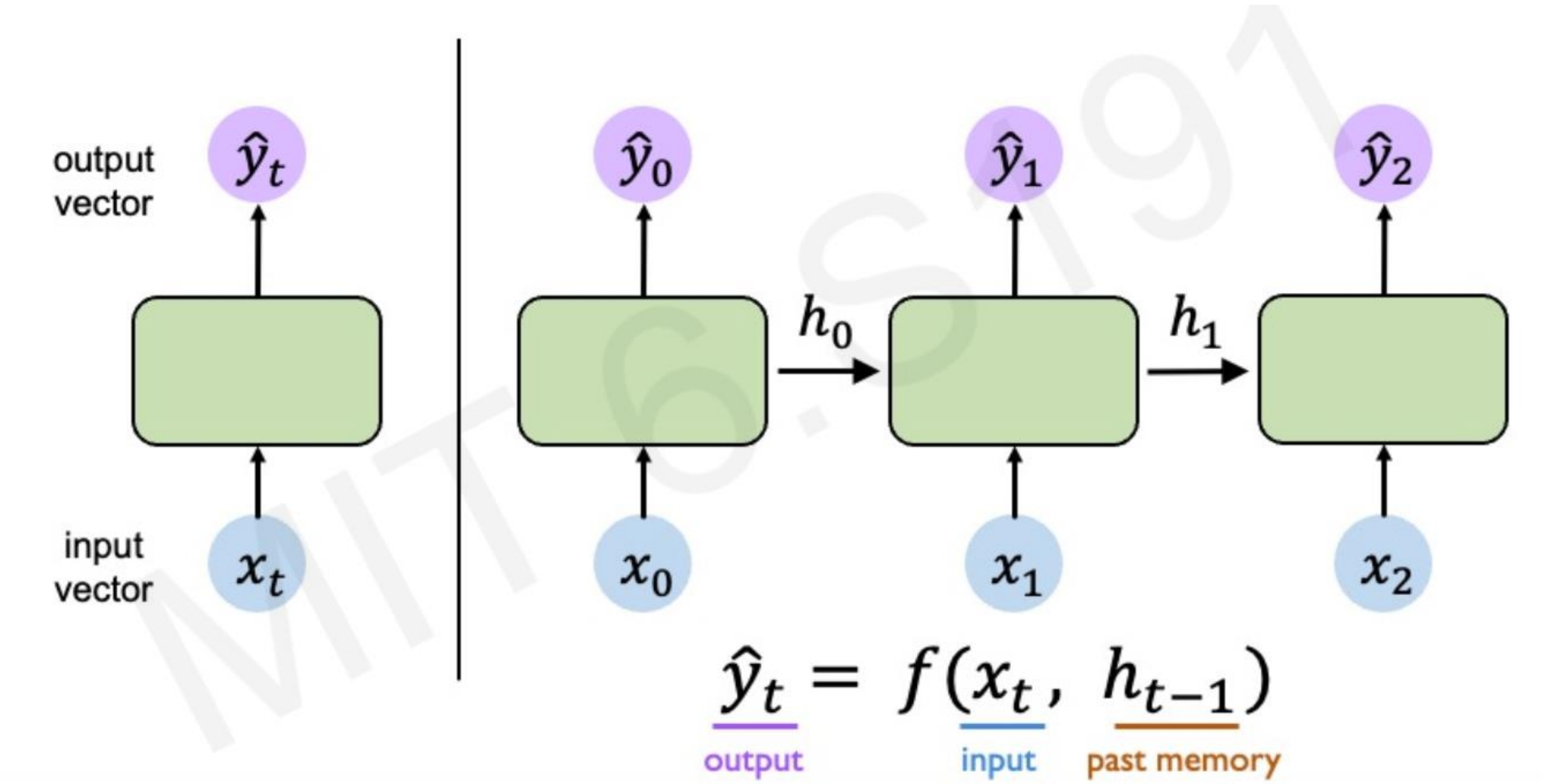
Simple Neural Networks



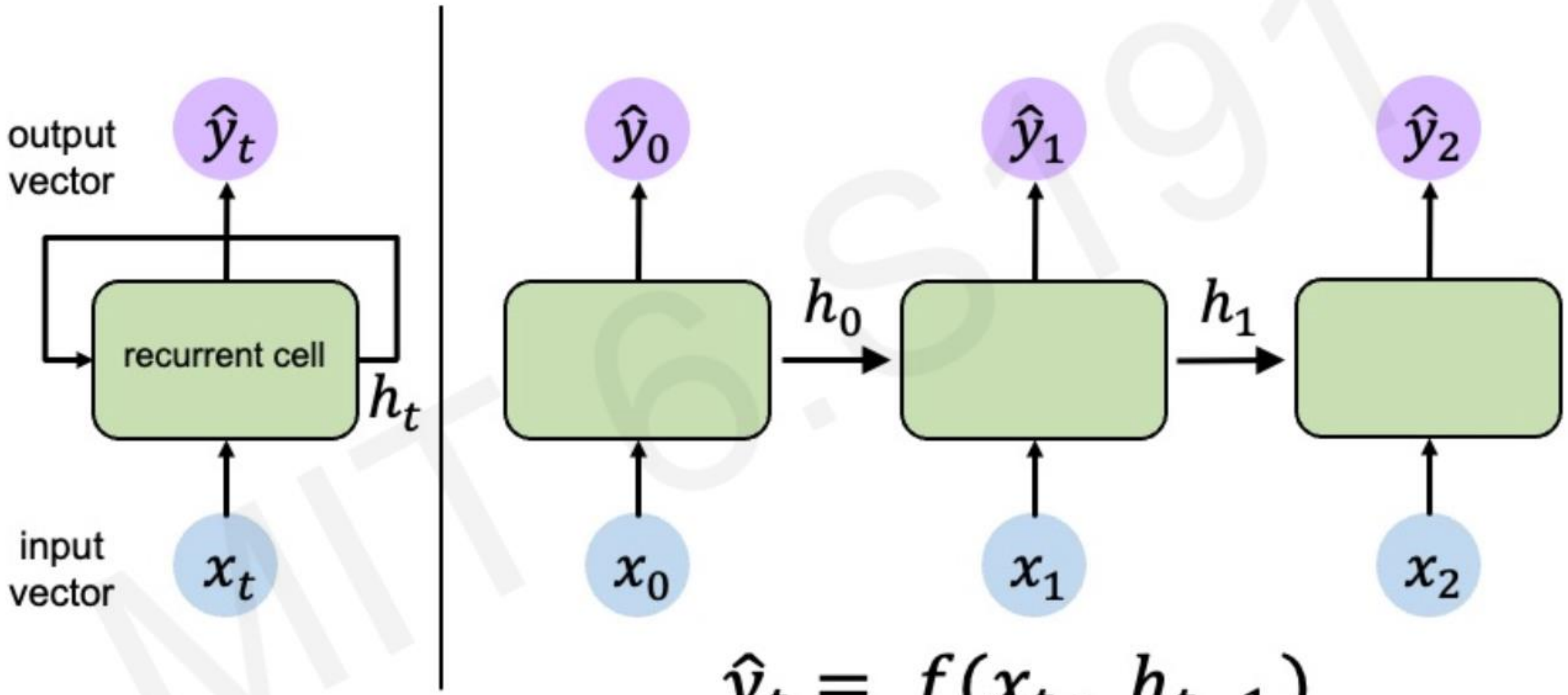
Handling Multiple Timesteps



Adding Context

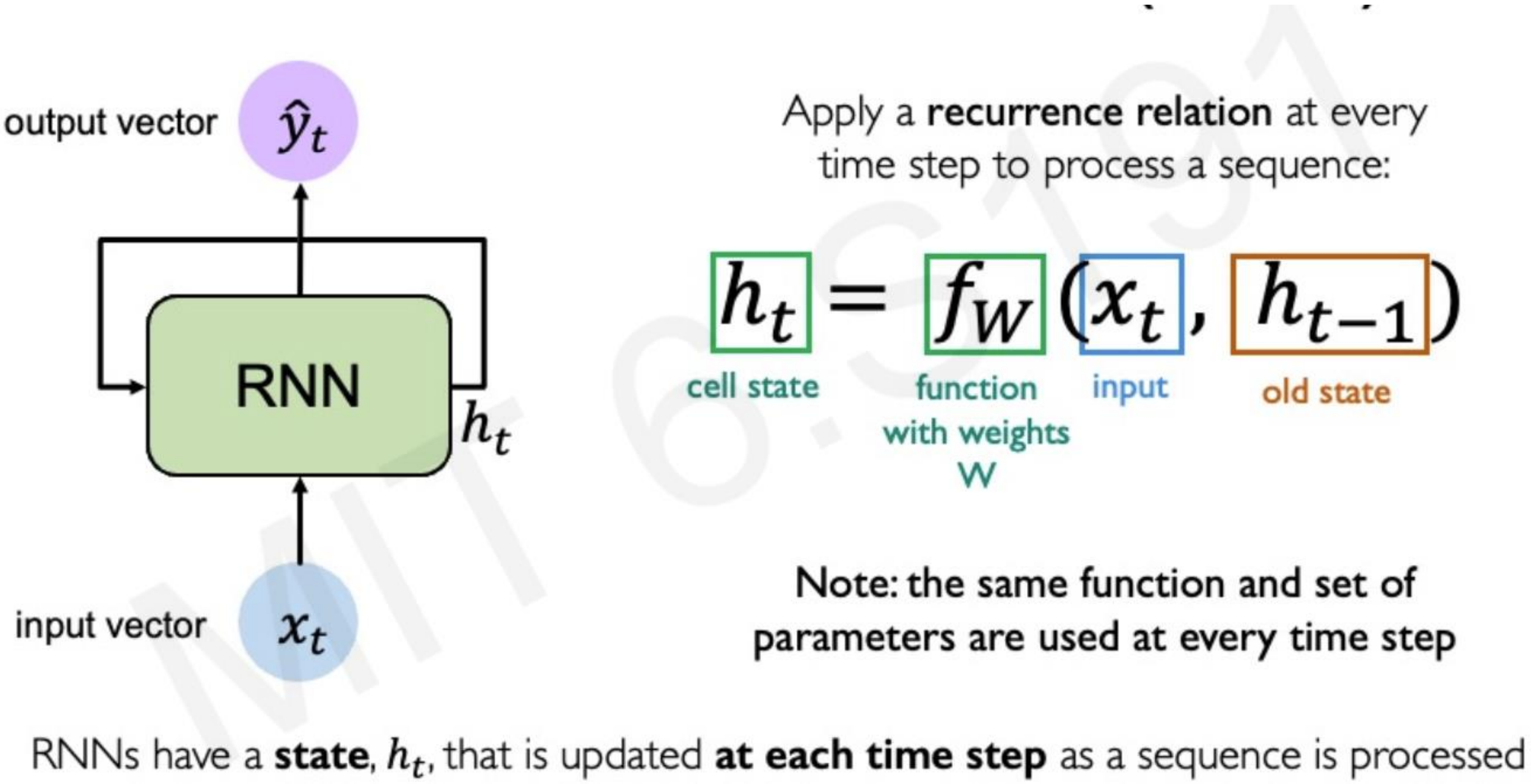


Recurrent Neurons



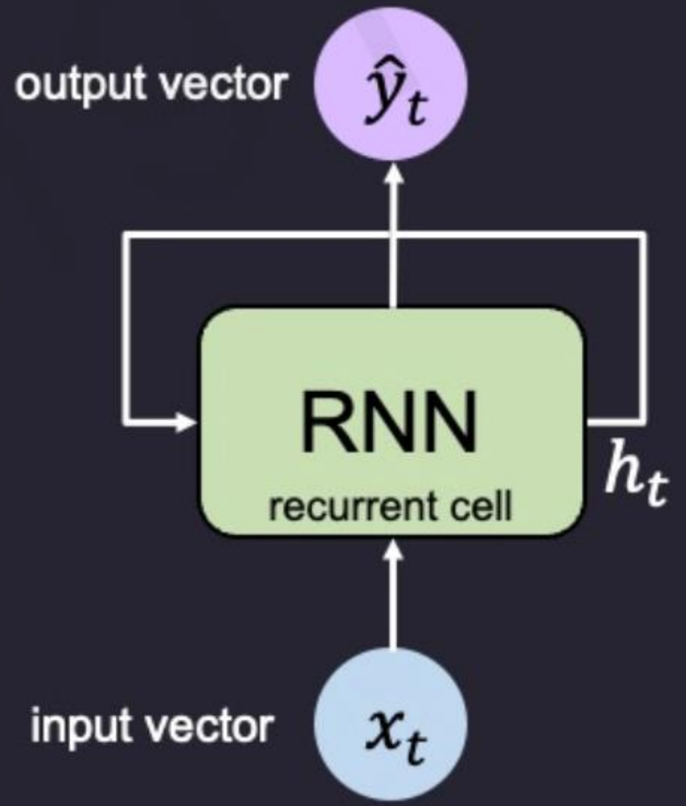
$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

Recurrent Neural Networks (RNN)

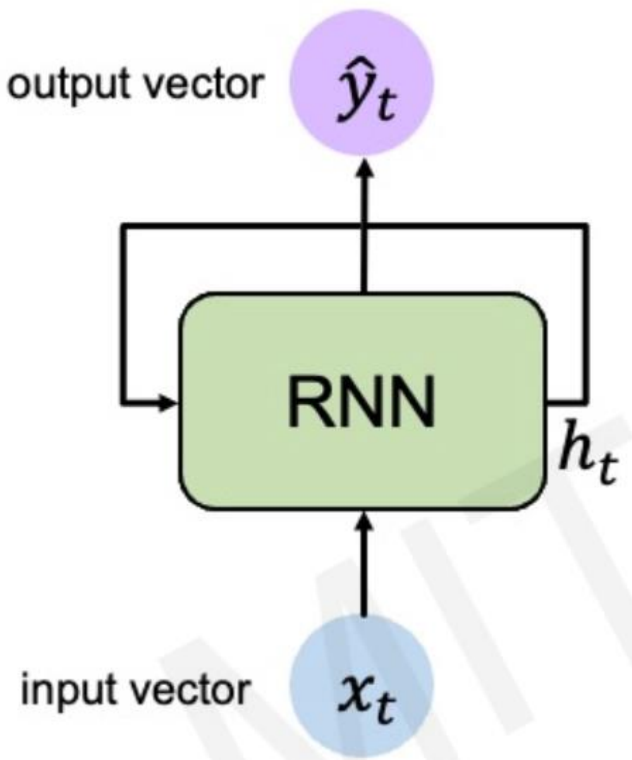


Recurrent Neural Networks

```python
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!"
```



Recurrent Neural Networks



Output Vector

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

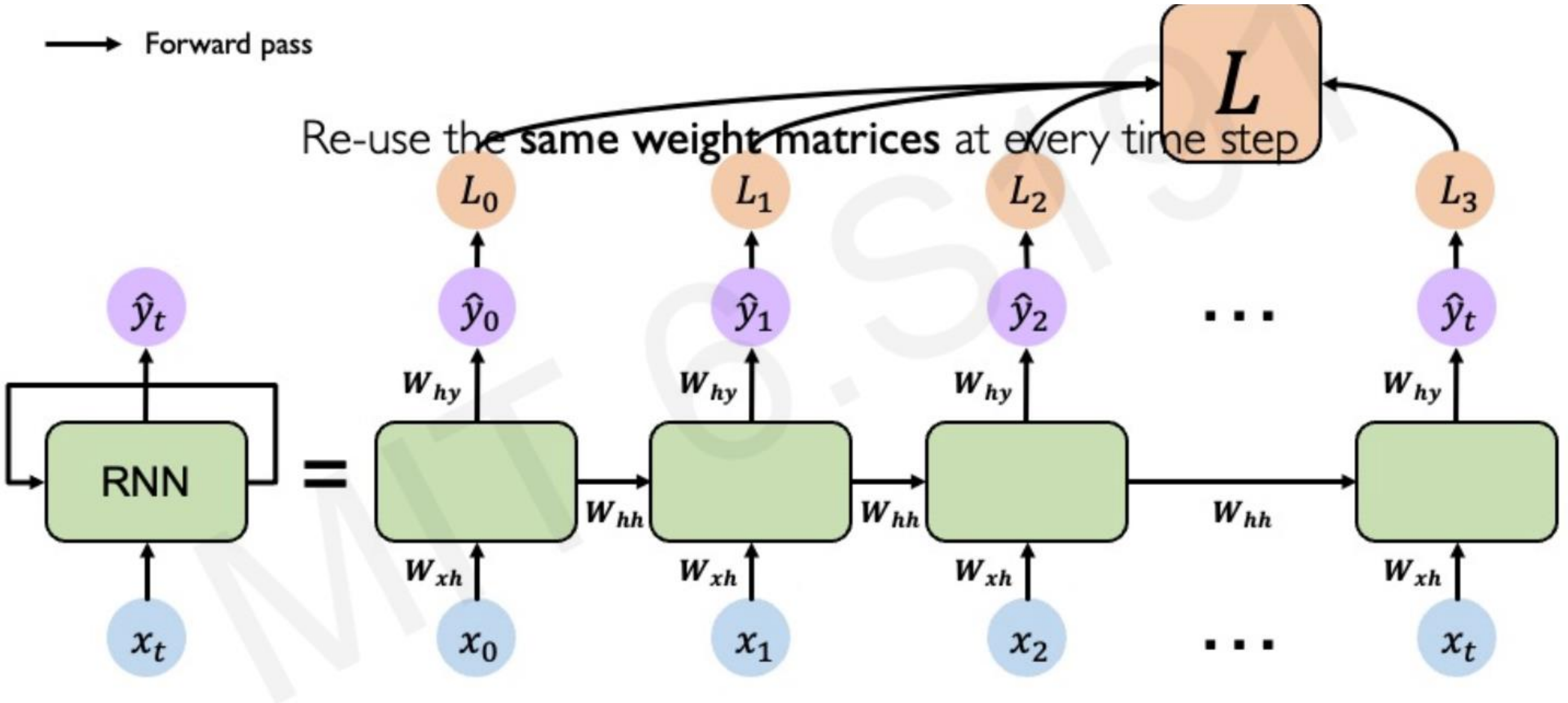
Update Hidden State

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

Input Vector

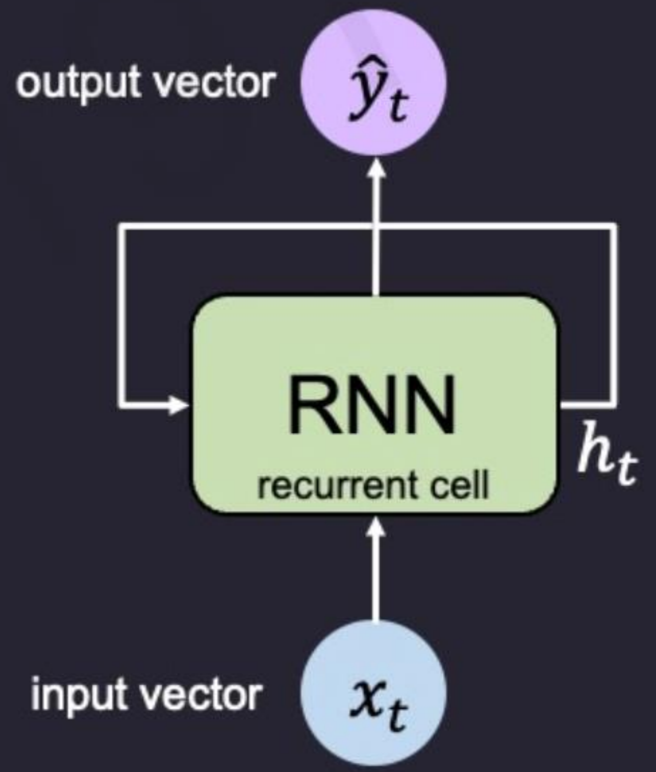
$$x_t$$

Recurrent Neural Networks



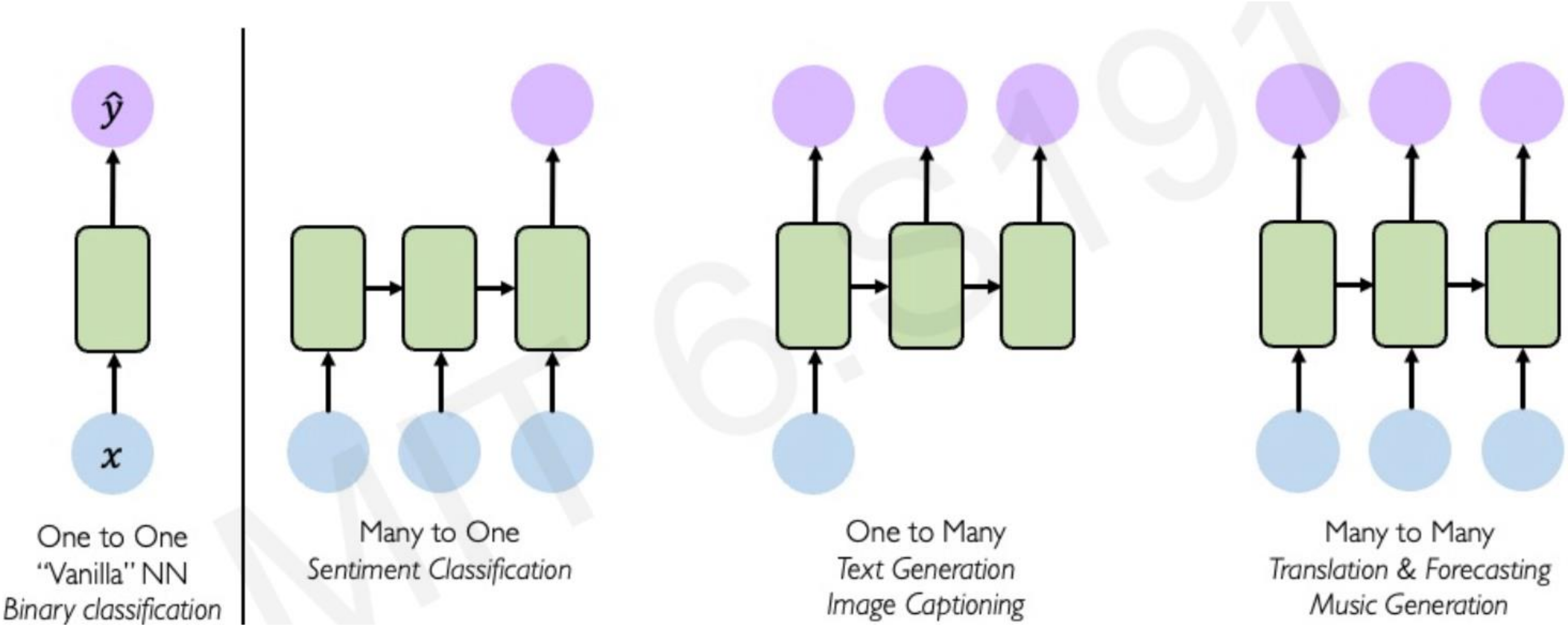
Recurrent Neural Networks

```
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
```



Recurrent Neural Networks

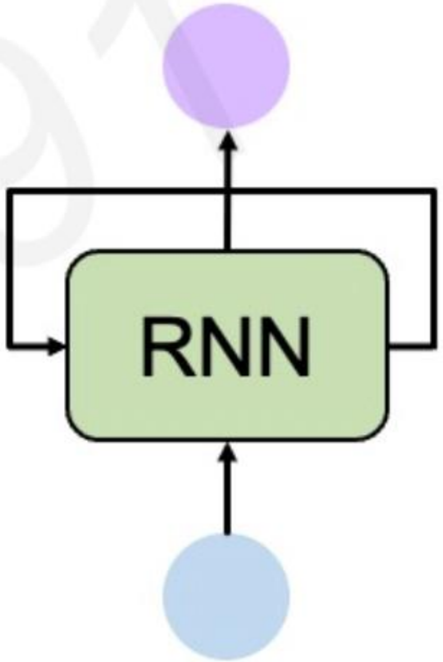
Recurrent Neural Networks



Design Factors

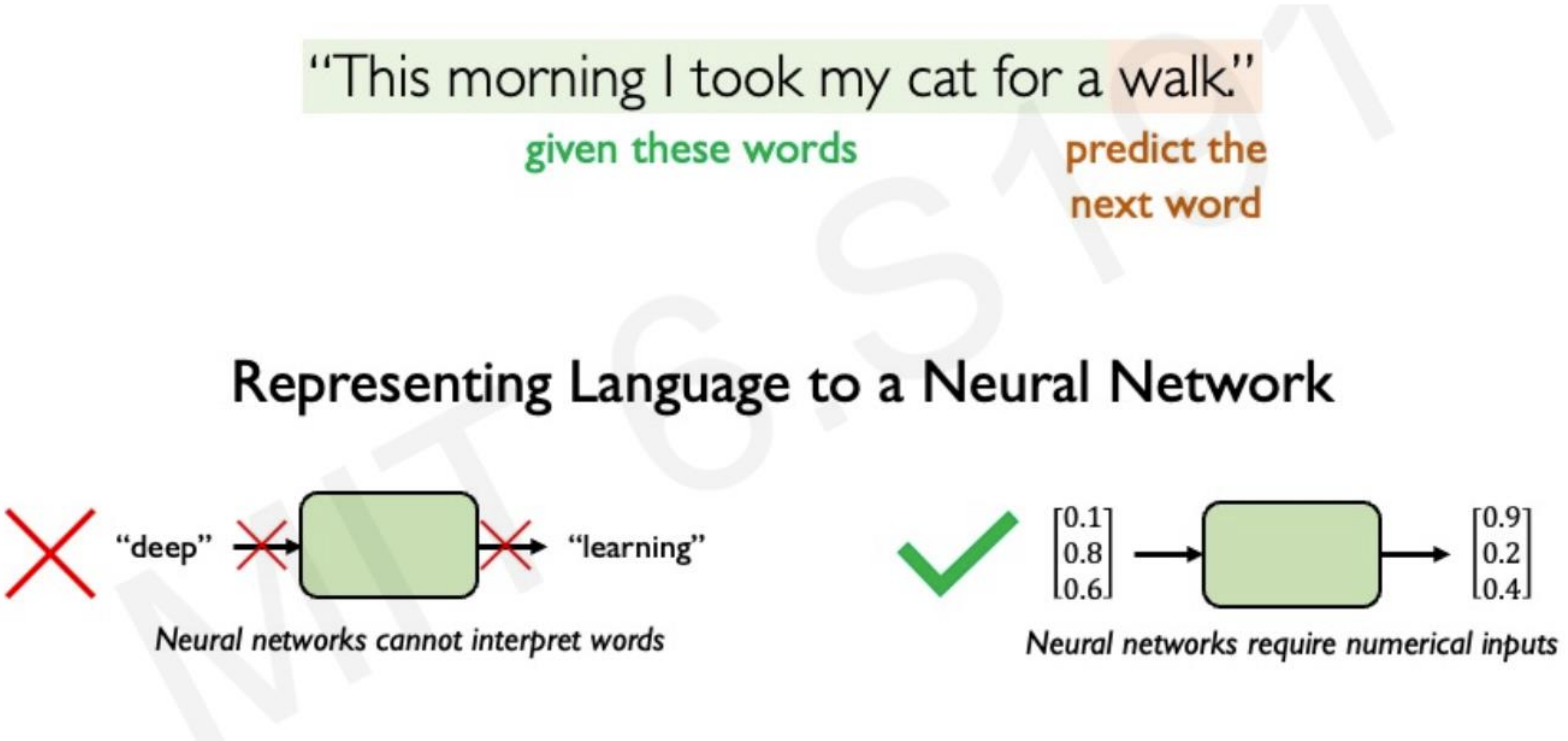
To model sequences, we need to:

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

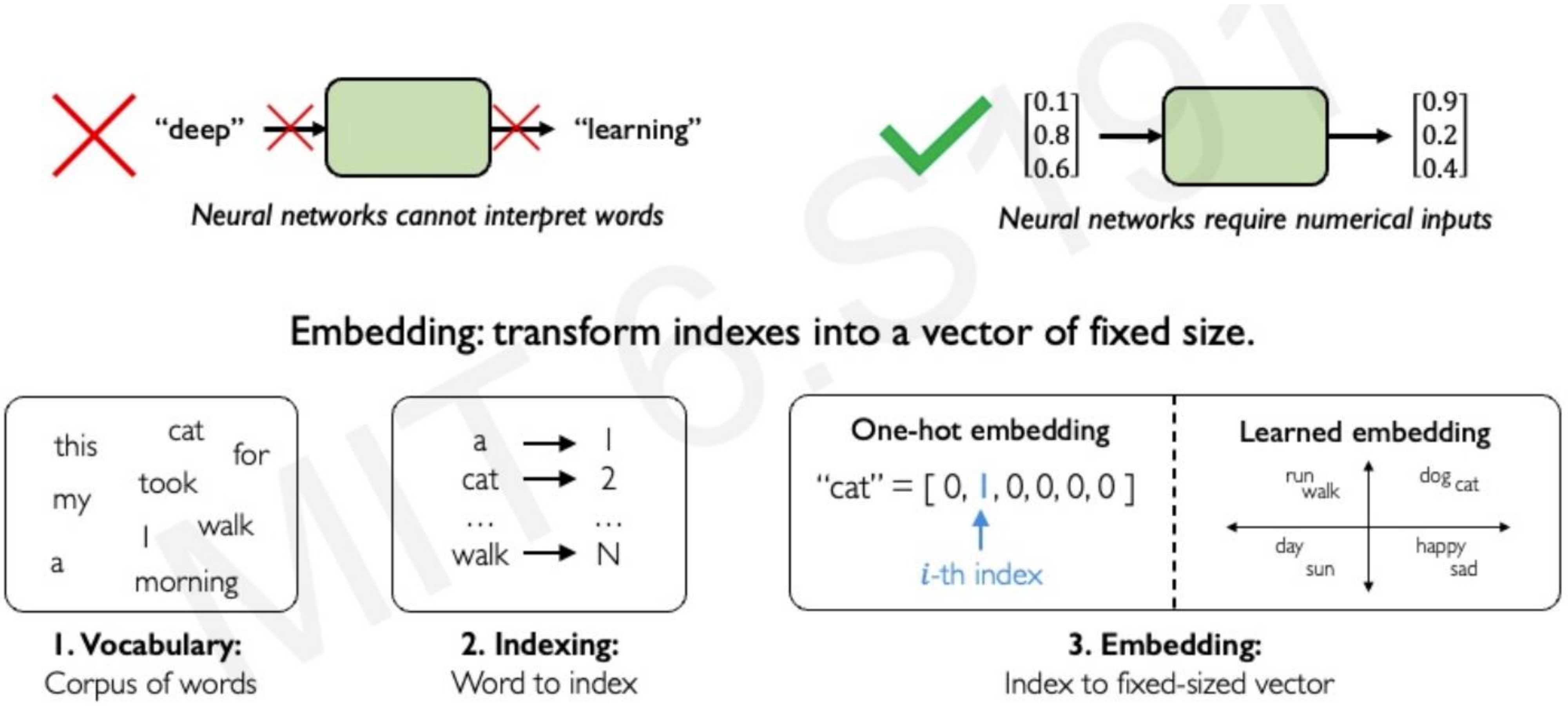


Recurrent Neural Networks (RNNs) meet
these sequence modeling design criteria

Next Word Prediction – GPT Task



Next Word Prediction – GPT Task



Variable Length

The food was great

vs.

We visited a restaurant for lunch

vs.

We were hungry but cleaned the house before eating

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.

Word Order

The food was good, not bad at all.

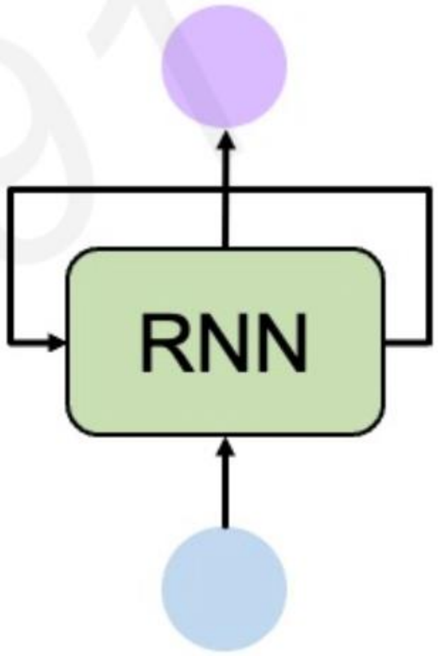
vs.

The food was bad, not good at all.

RNN

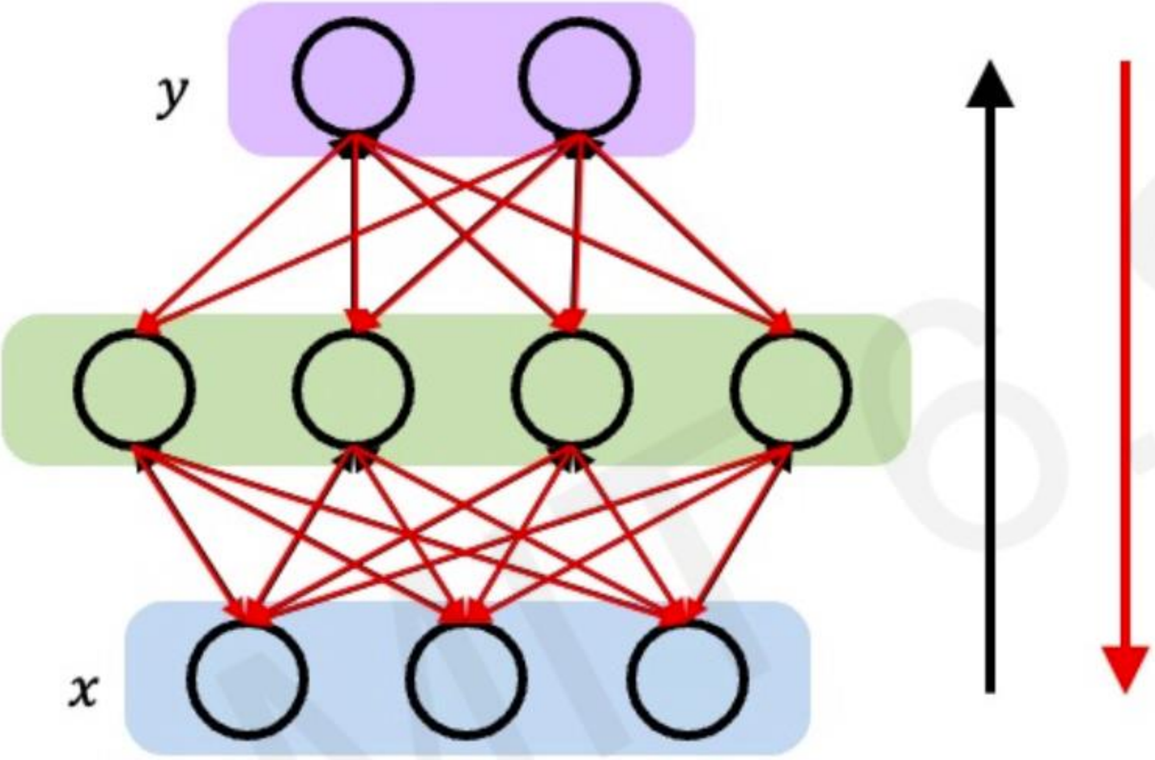
To model sequences, we need to:

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

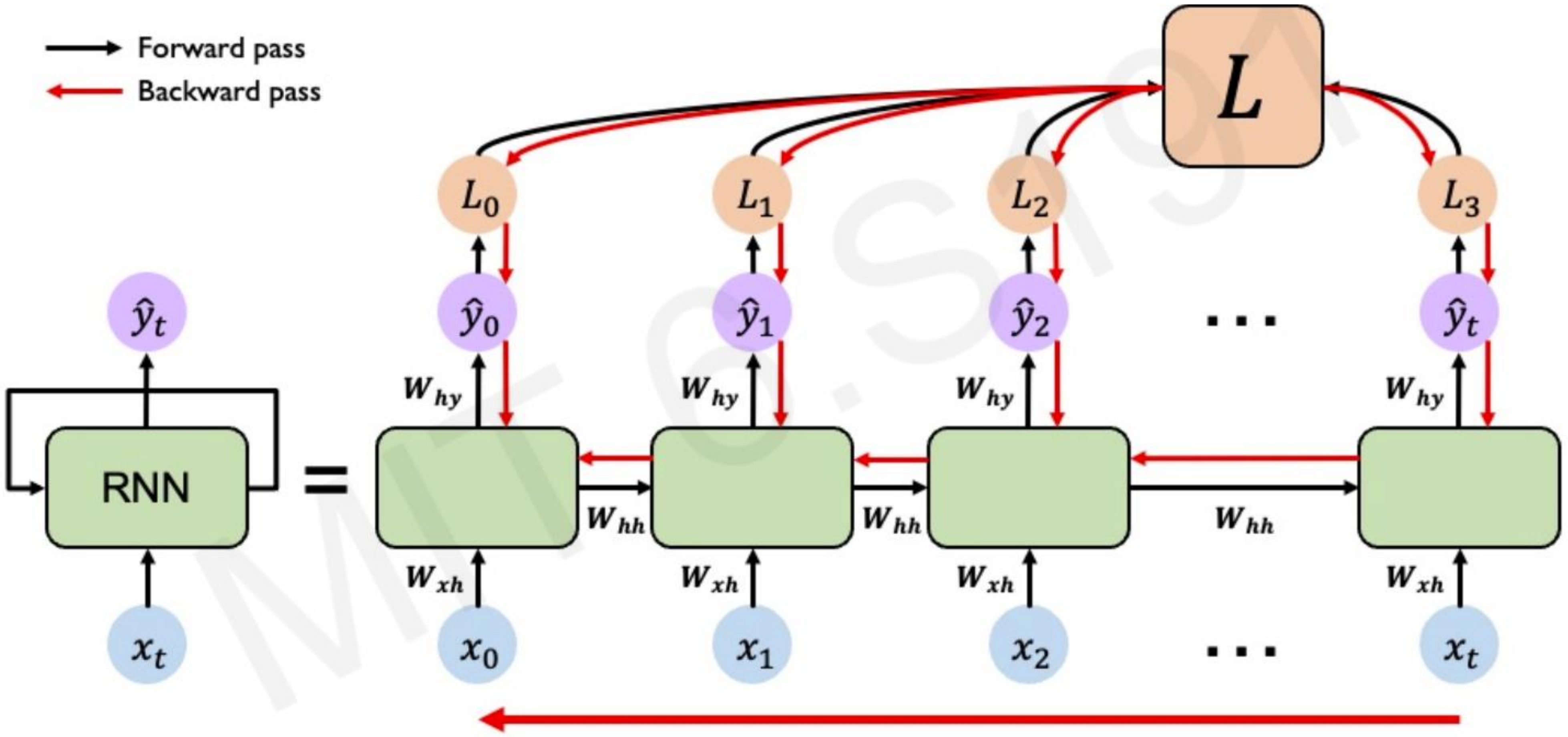
Gradient Backprop.



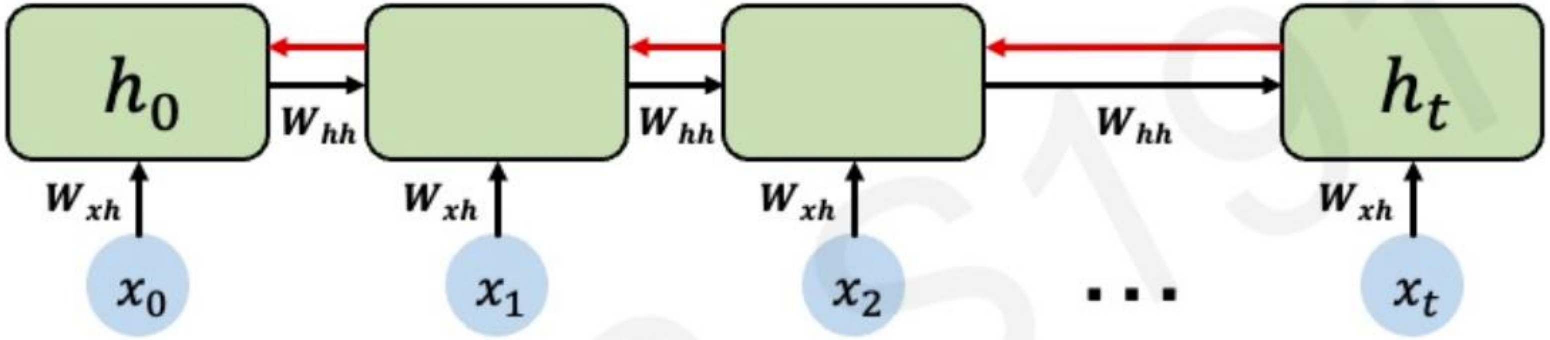
Backpropagation algorithm:

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

Time Backprop.



Exploding Gradient

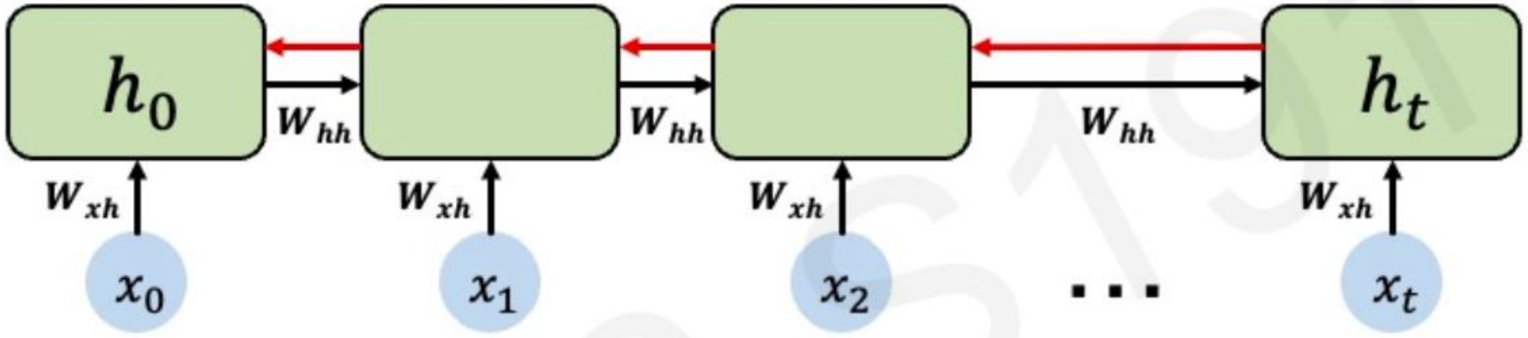


Computing the gradient wrt h_0 involves **many factors of W_{hh}** + **repeated gradient computation**!

Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

Vanishing Gradient



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

Many values > 1:
exploding gradients

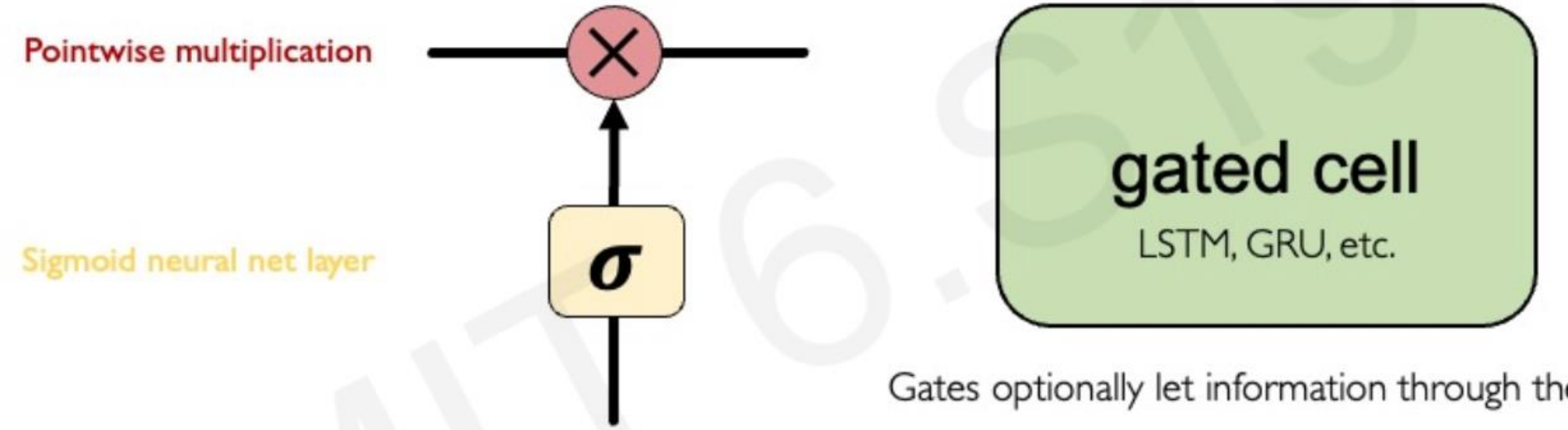
Gradient clipping to scale big gradients

Many values < 1:
vanishing gradients

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

LSTM

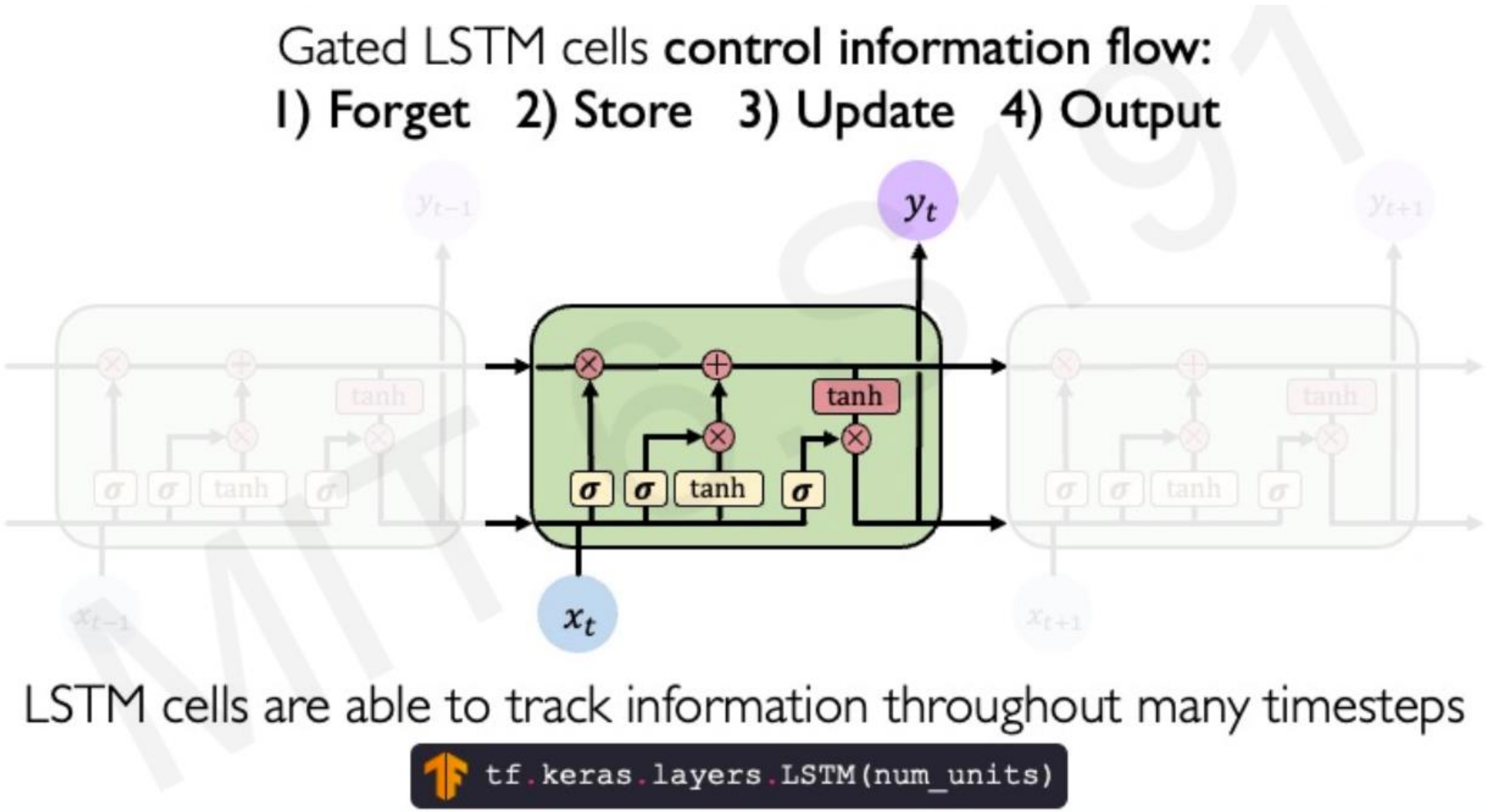
Idea: use **gates** to selectively **add** or **remove** information within **each recurrent unit with**



Gates optionally let information through the cell

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

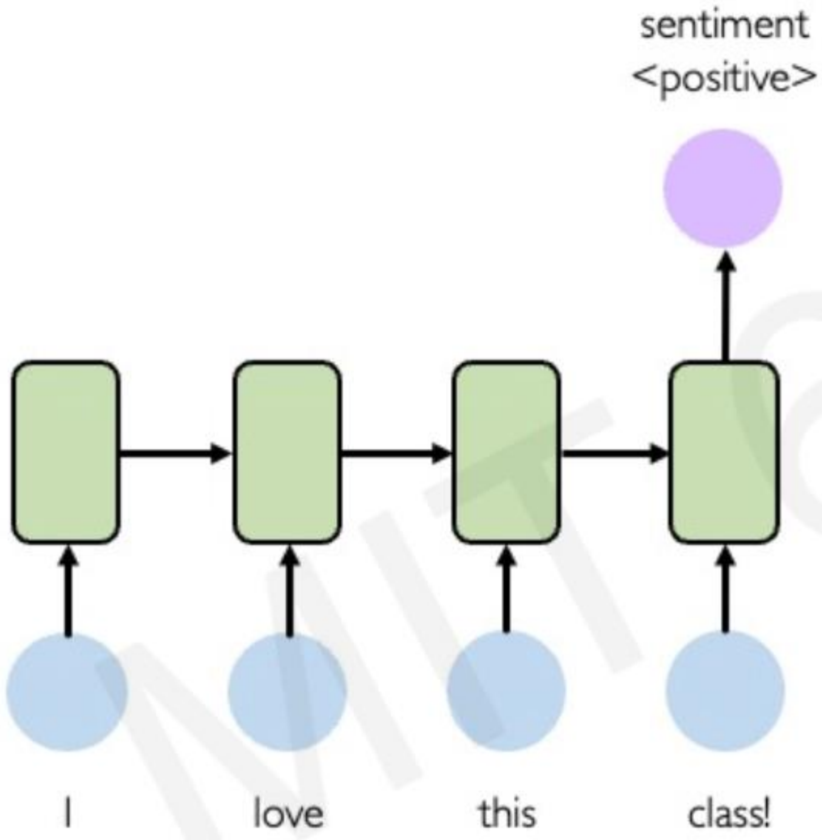
Long-term Dependencies



LSTM

1. Maintain a **cell state**
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 partially **uninterrupted gradient flow**

Sentiment Analysis

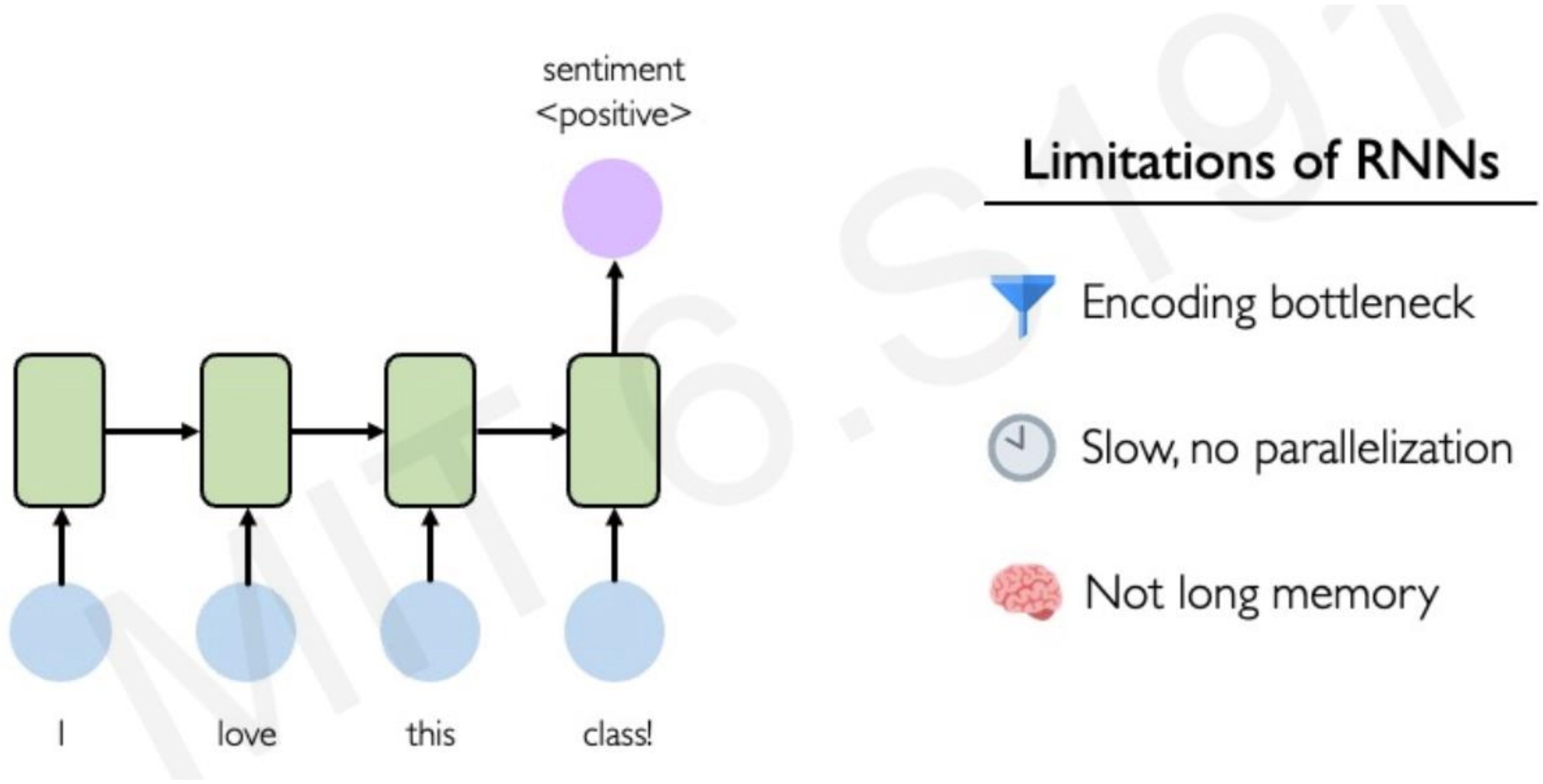


Input: sequence of words

Output: probability of having positive sentiment

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

Challenges with RNN Models



Attention is all you need!