Sequence Modeling

CSE 4237 - Soft Computing

Mir Tafseer Nayeem
Faculty Member, CSE AUST
tafseer.nayeem@gmail.com

Example of Sequence

Audio

Audio can be split up into sequence of sound waves

Audio

Example of Sequence

Text can be split up into sequence of characters or words.

Sequence of Words

This is a short sentence

Sequence of Characters

T h i s i s a s h o r t s e n t e n c e

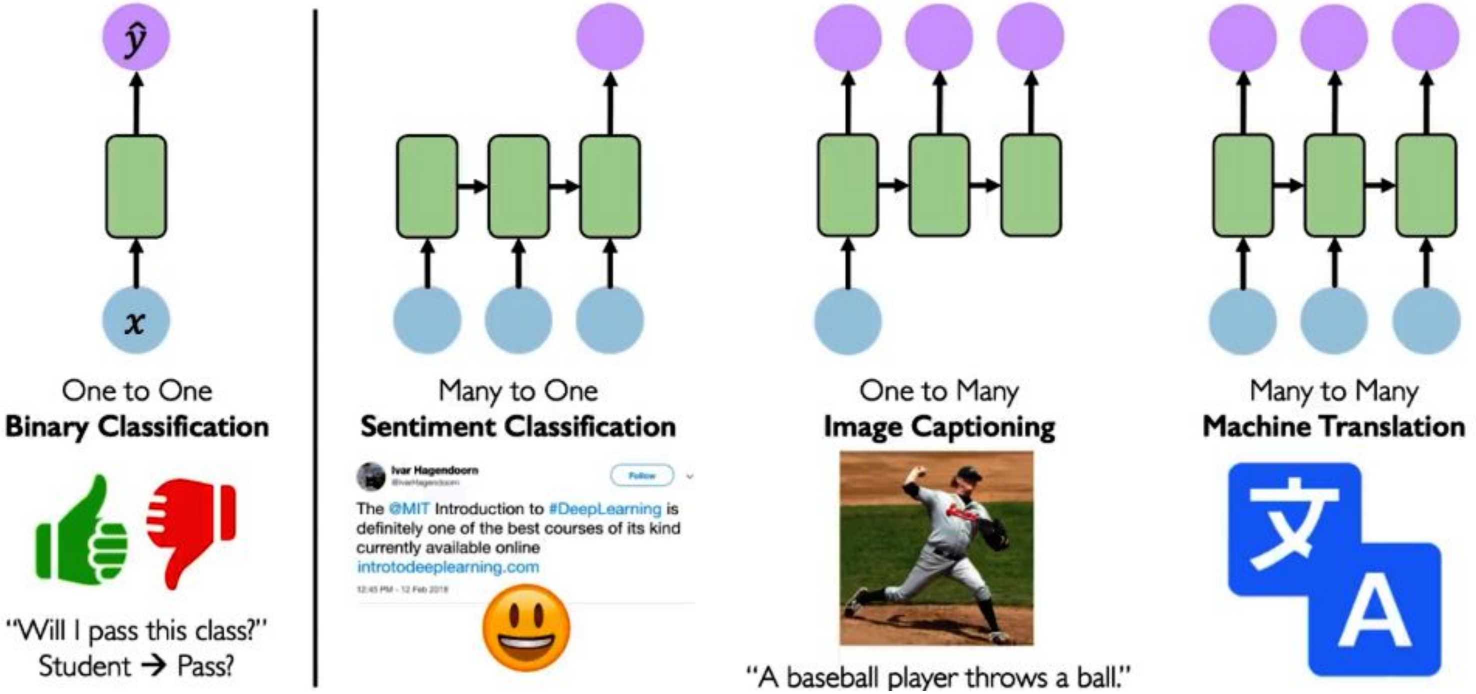
Example of Sequence

DNA sequence analysis

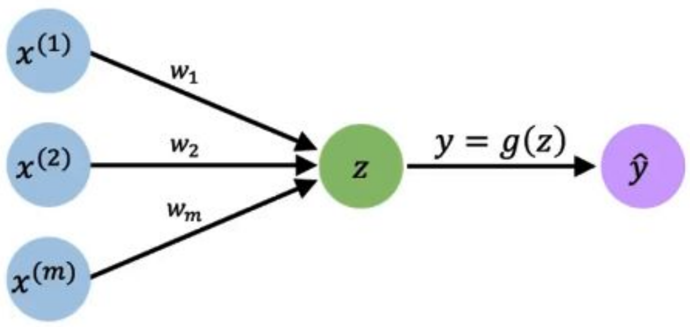
AGCCCCTGTGAGGAACTAG

Video activity recognition

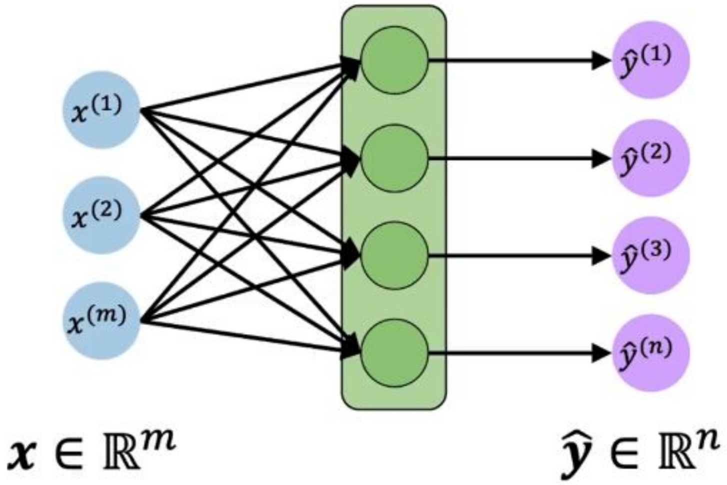
Sequence Modeling Applications



Feed-Forward Neural Network Revisited



Single Output

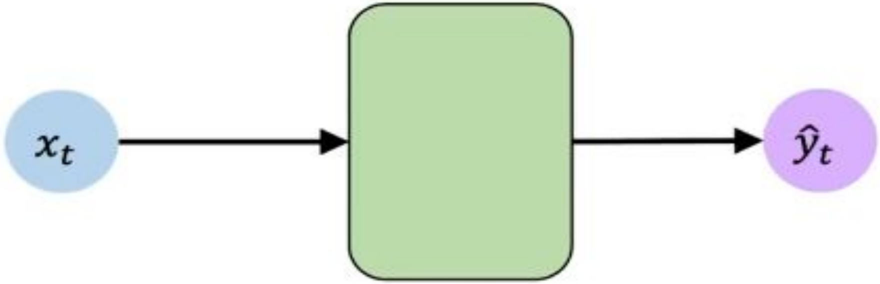


Multiple Output

It doesn’t have a notion of time or sequence. Our inputs and our outputs from a fixed time step

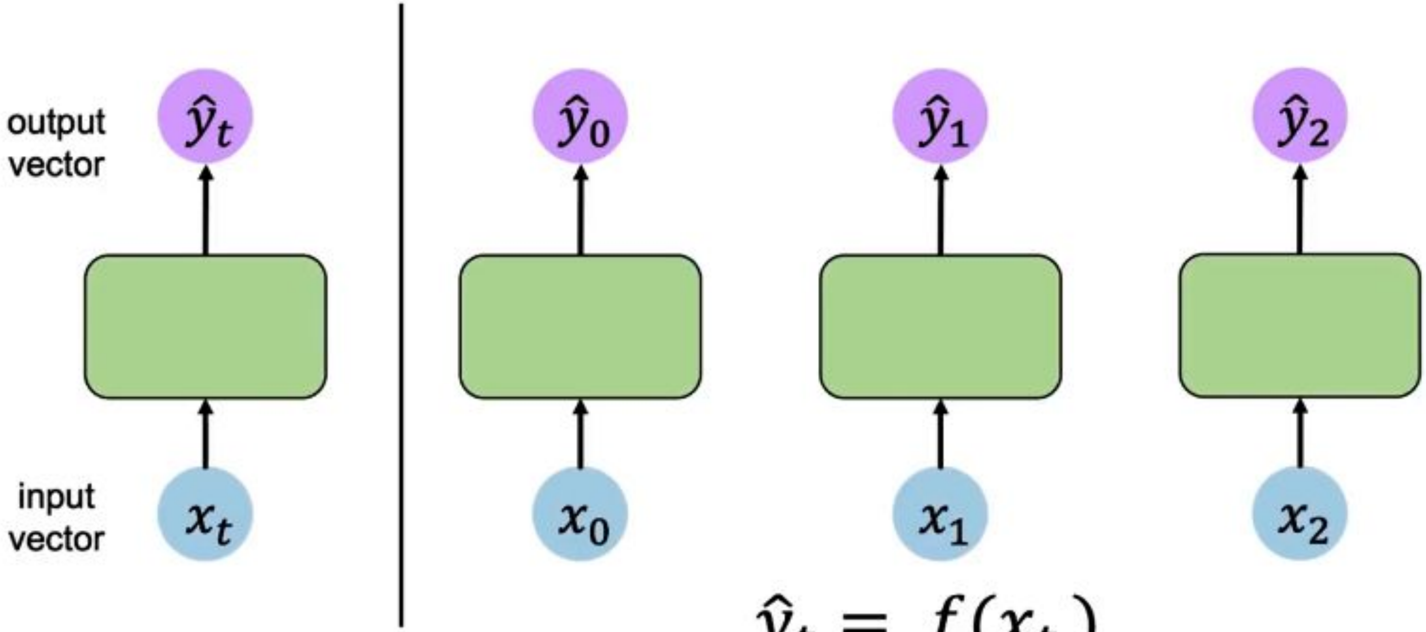
Feed-Forward Neural Network Revisited

Simplified Representation



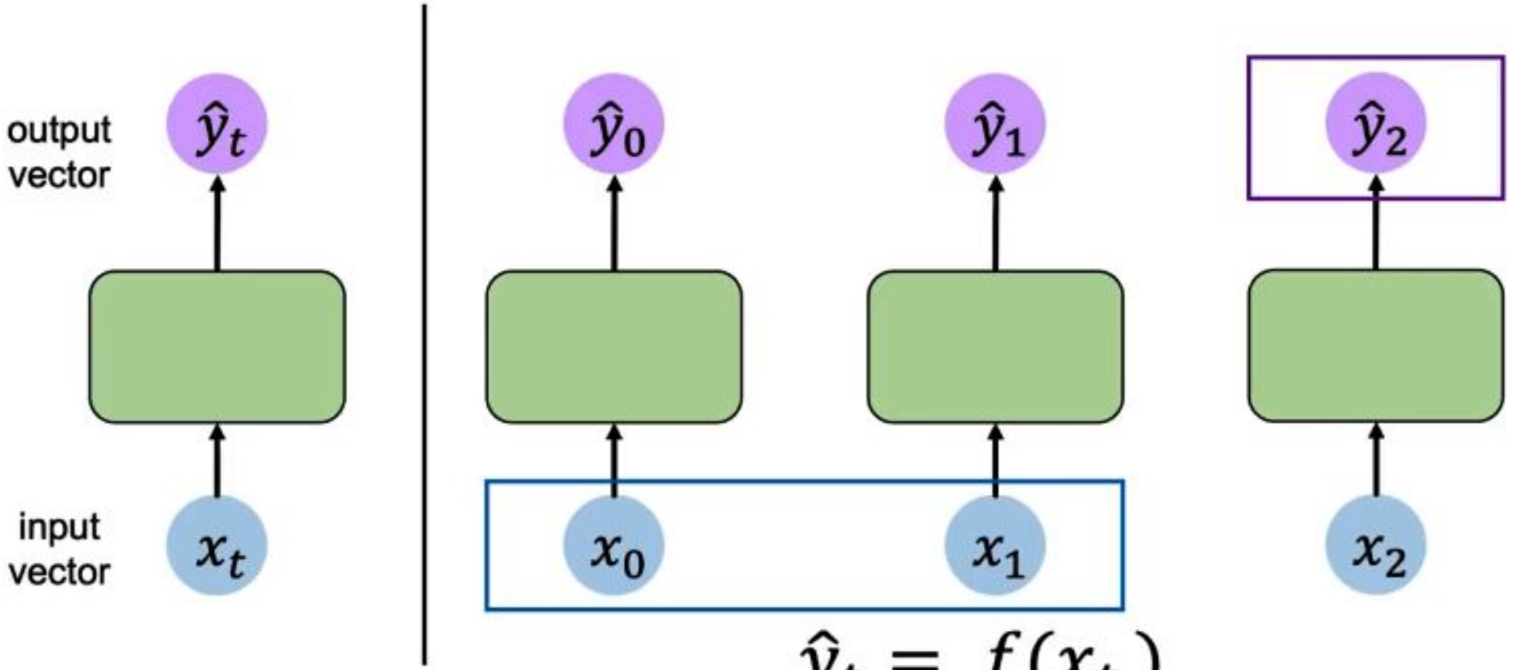
$$\boldsymbol{x_t} \in \mathbb{R}^m \qquad \boldsymbol{\hat{y}}_t \in \mathbb{R}^n$$

Handling Individual Time Steps



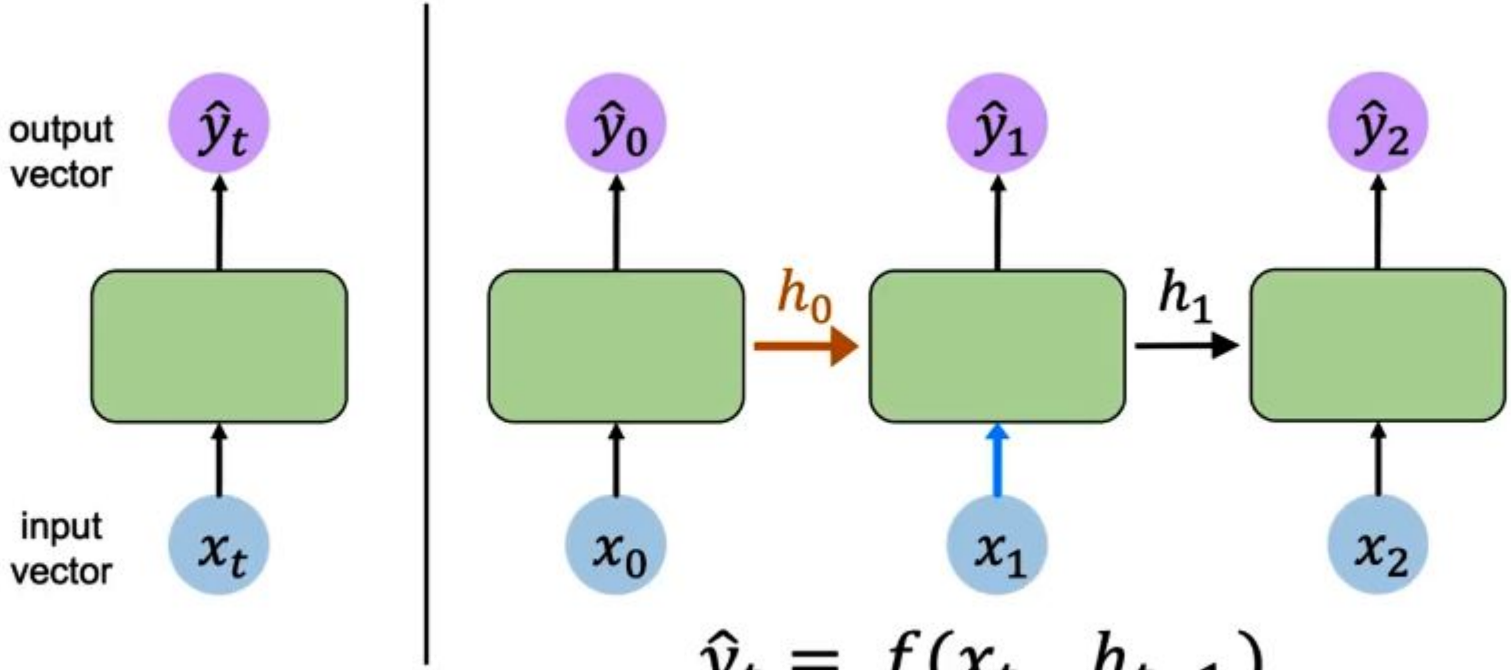
$$\hat{y}_t = f(x_t)$$

Handling Individual Time Steps



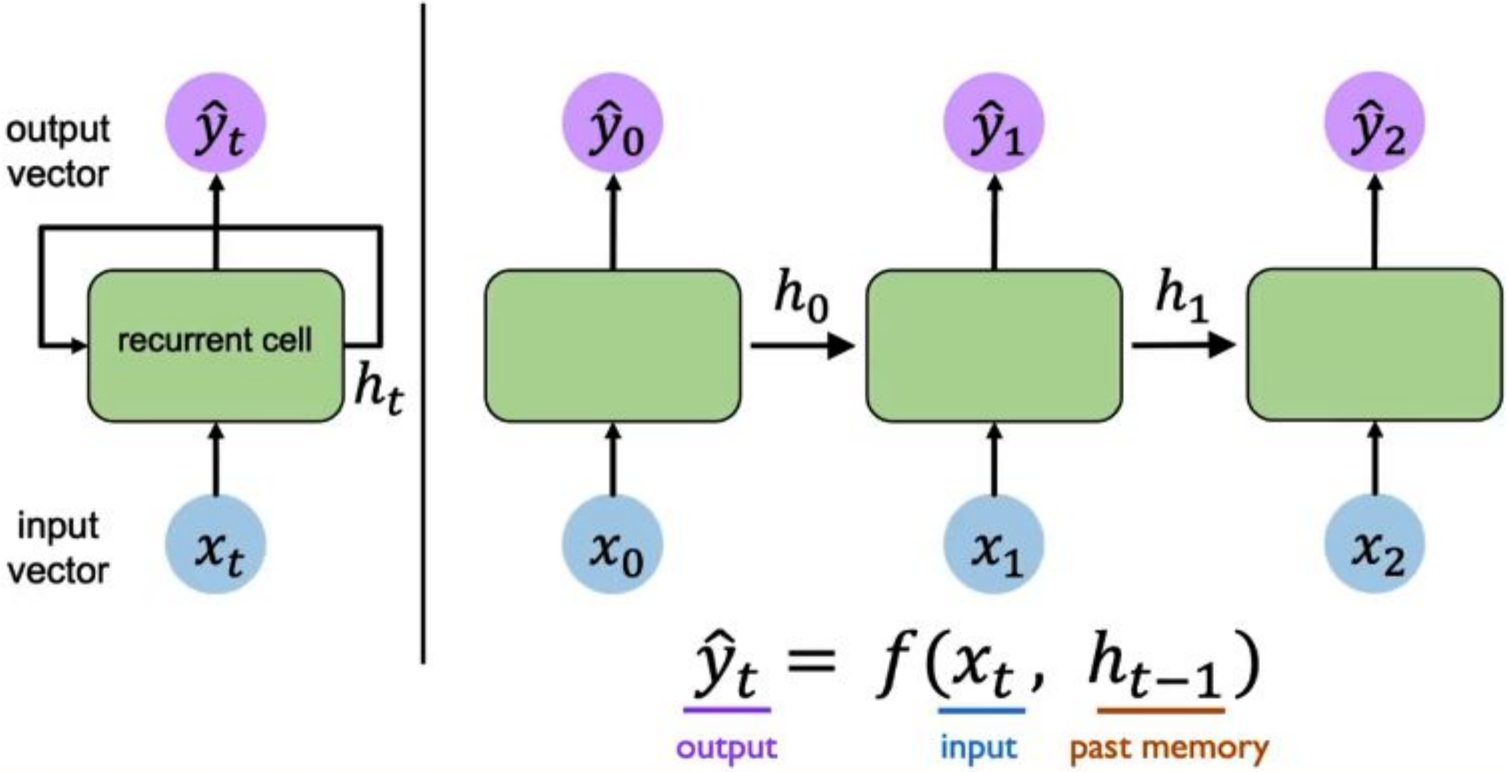
$$\hat{y}_t = f(x_t)$$

Neurons with Recurrence

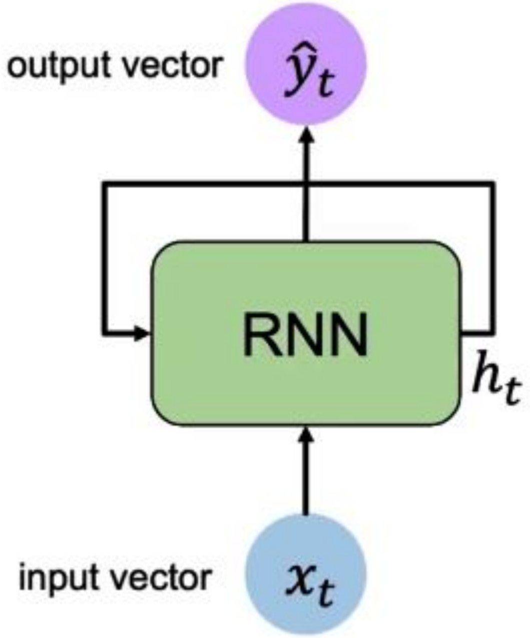


$$\underbrace{\hat{y}_t}_{\text{output}} = f(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}})$$

Neurons with Recurrence



Recurrent Neural Networks (RNNs)



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

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

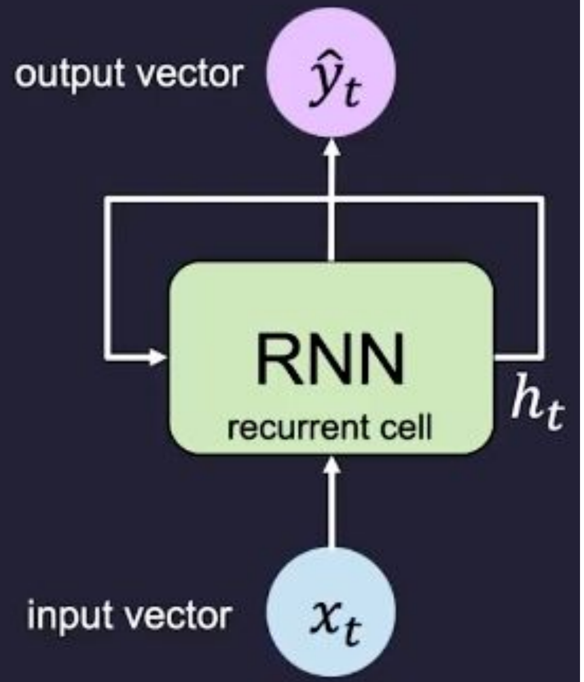
cell state | function with weights W | input | old state

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

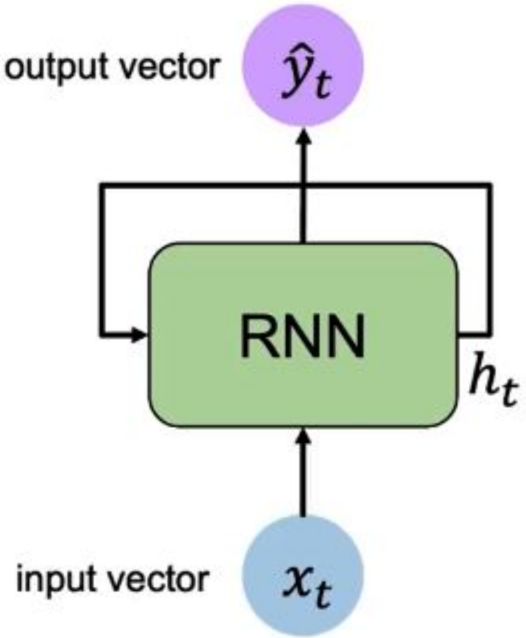
RNNs have a **state**, h_t, that is updated **at each time step** as a sequence is processed

RNN Intuition

```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!"
```



RNN State Update and Output



Output Vector

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

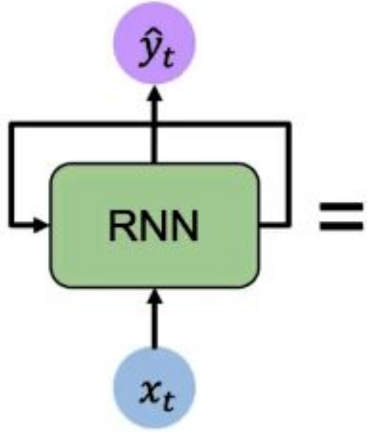
Update Hidden State

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

Input Vector

$$x_t$$

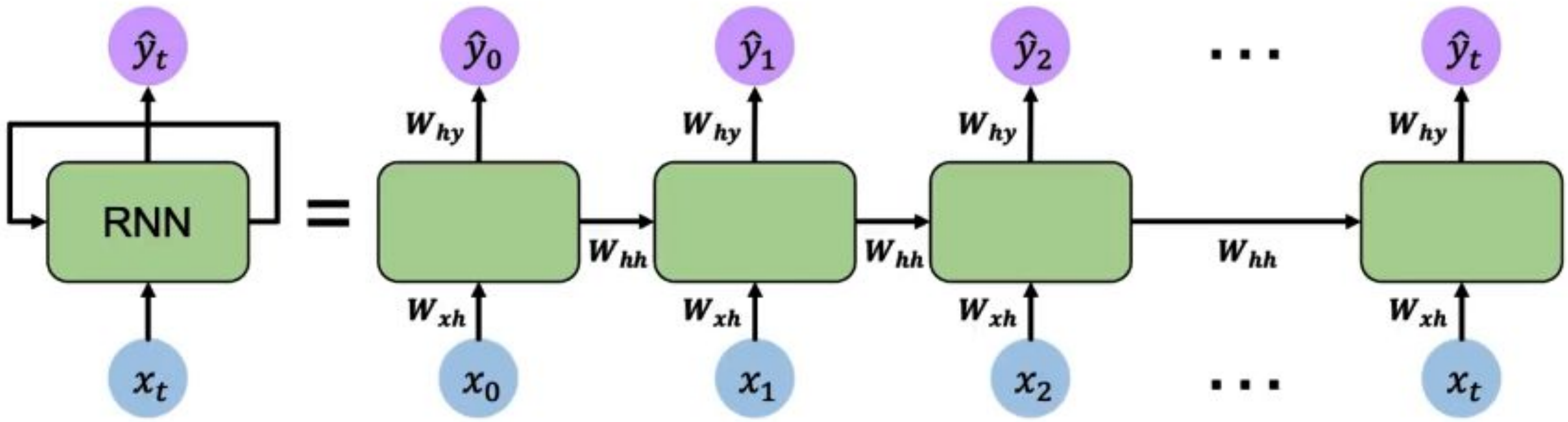
RNNs: Computational Graph Across Time



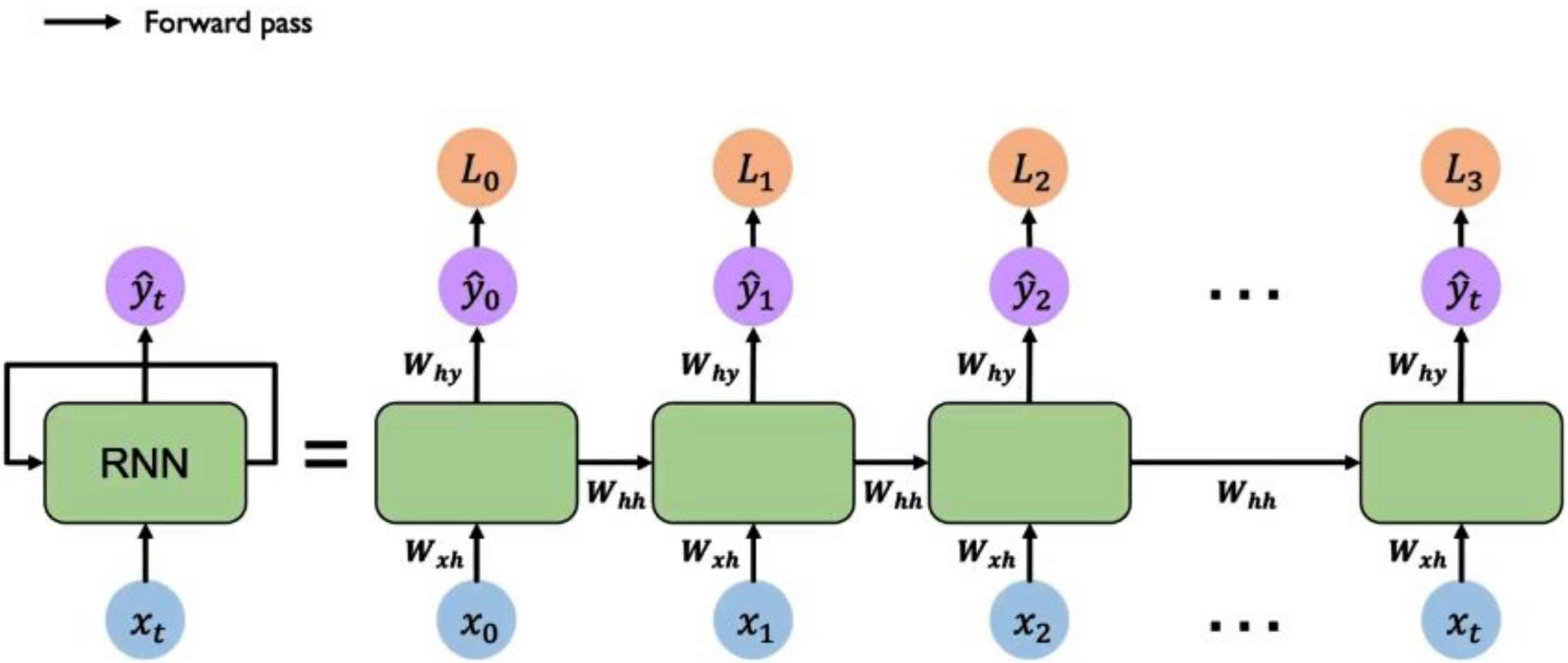
Represent as computational graph unrolled across time

RNNs: Computational Graph Across Time

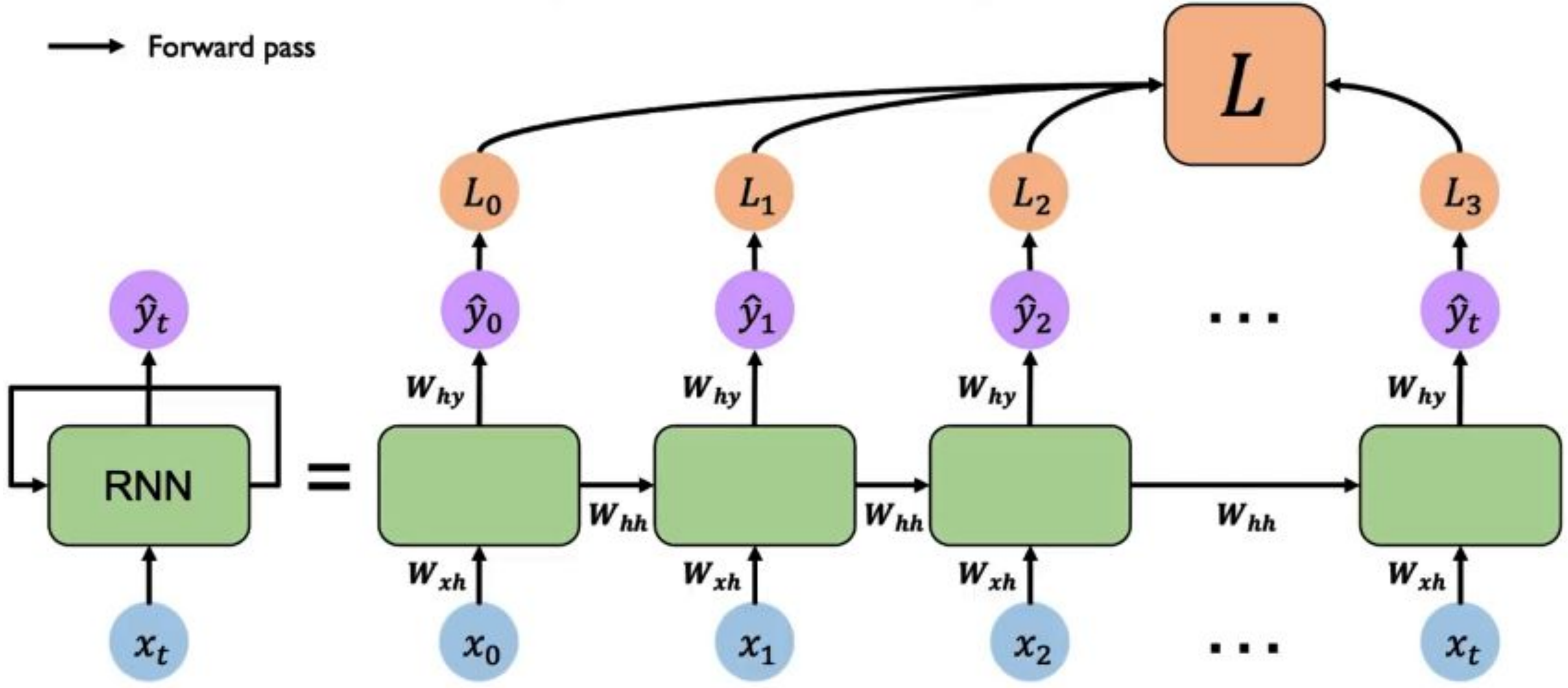
Re-use the **same weight matrices** at every time step



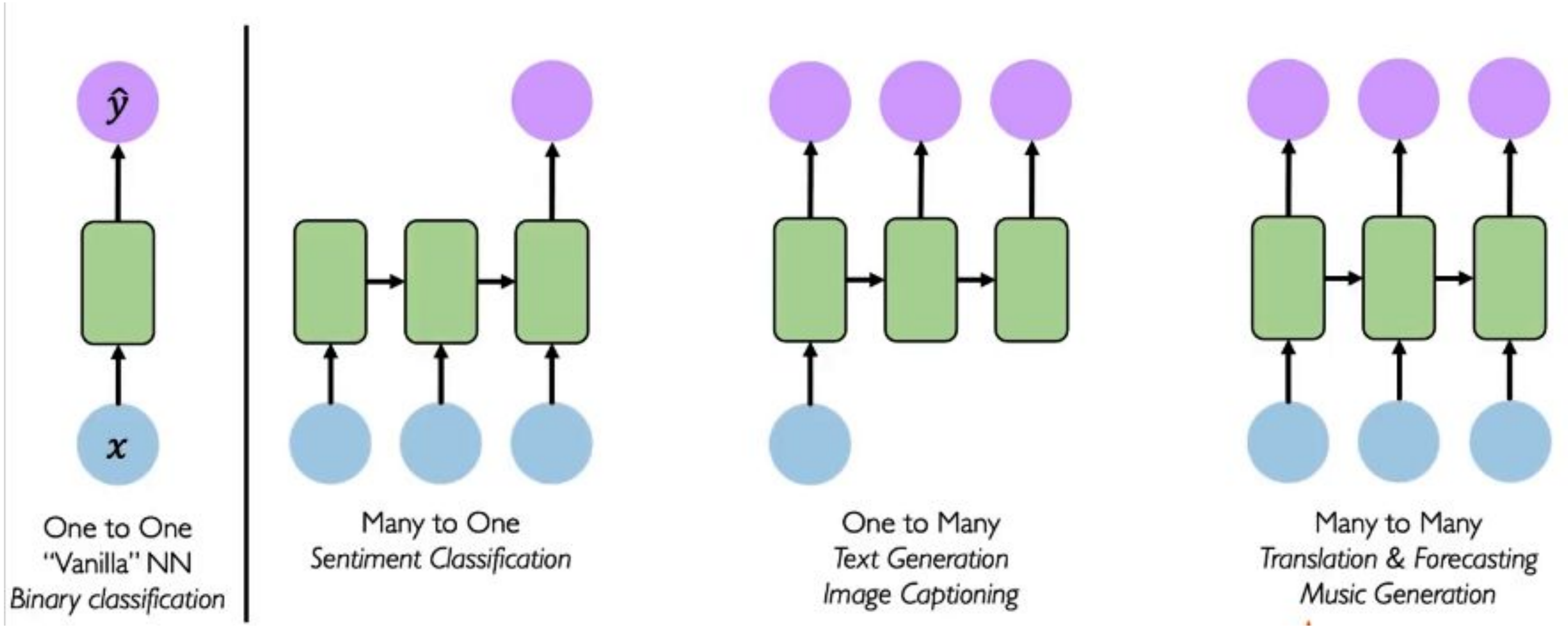
RNNs: Computational Graph Across Time



RNNs: Computational Graph Across Time



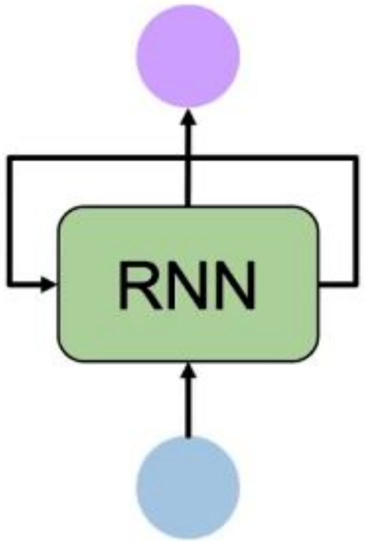
RNNs for Sequence Modeling



Sequence Modeling: Design Criteria

To model sequences, we need to:

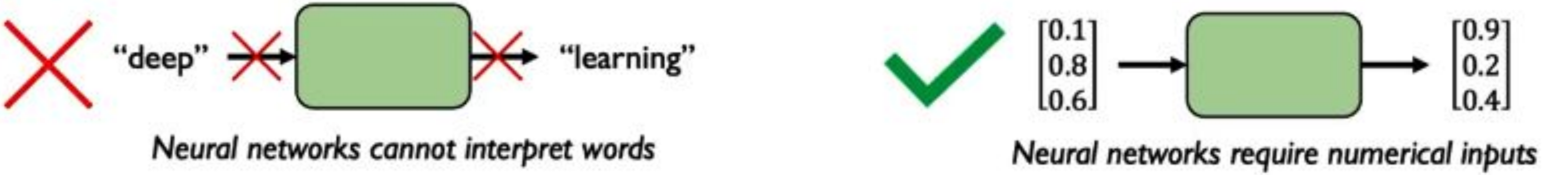
1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



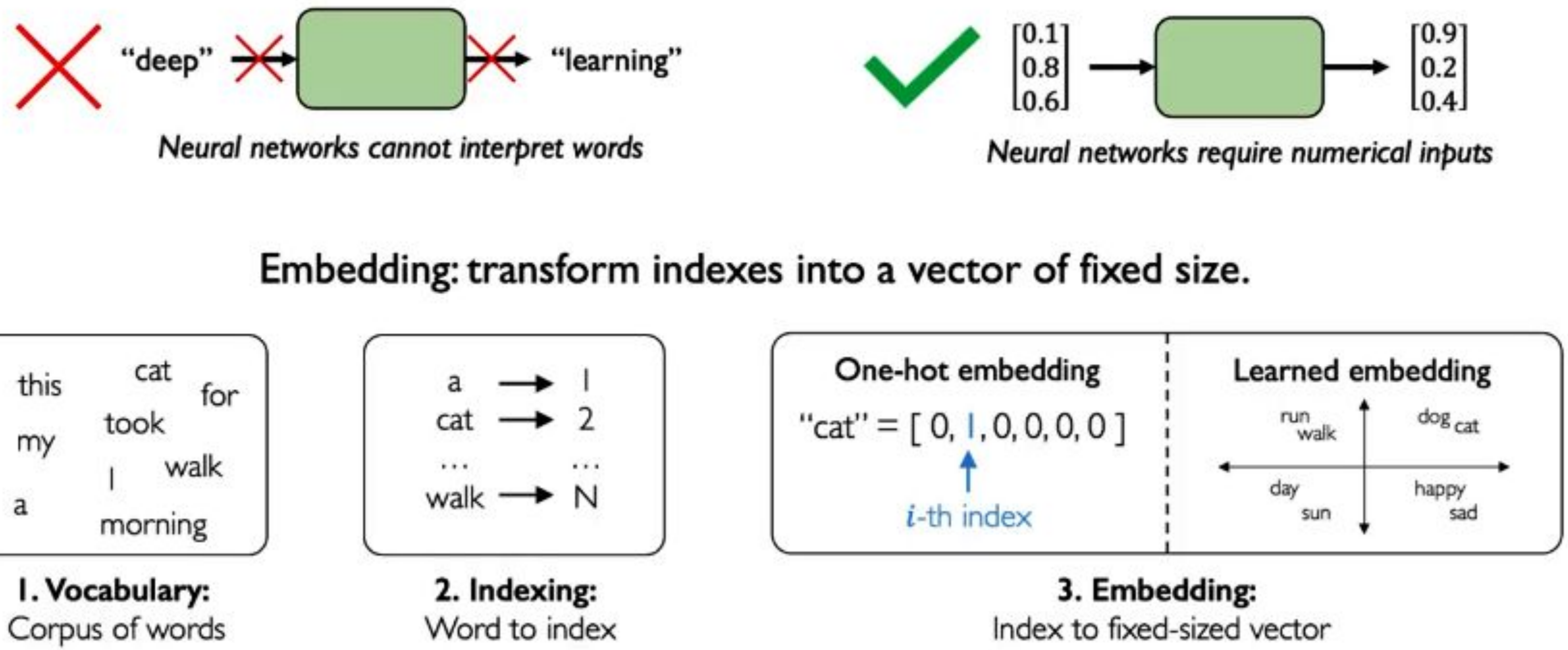
Recurrent Neural Networks (RNNs) meet
these sequence modeling design criteria

A Sequence Modeling Problem: Predict the Next Word

Representing Language to a Neural Network



Encoding Language for a Neural Network



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

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.

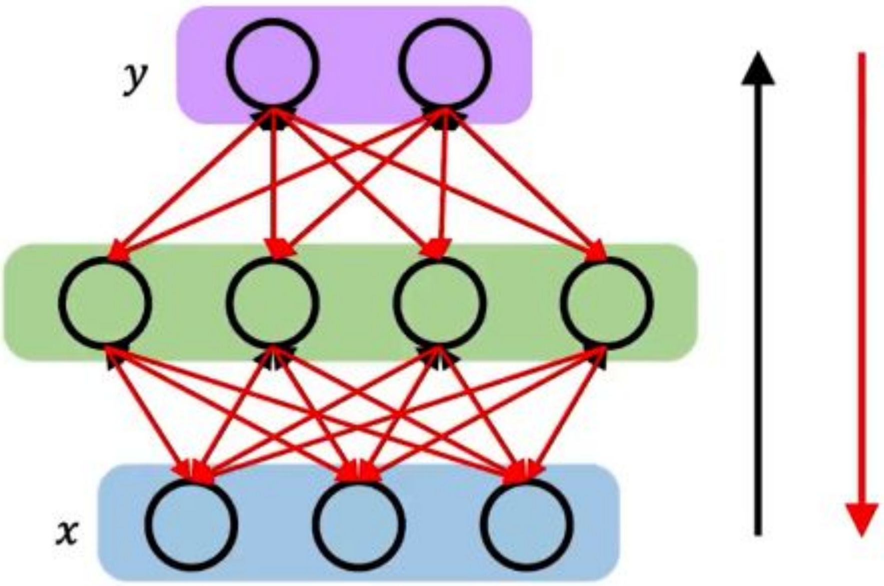
Capture Differences in Sequence Order

The food was good, not bad at all.

vs.

The food was bad, not good at all.

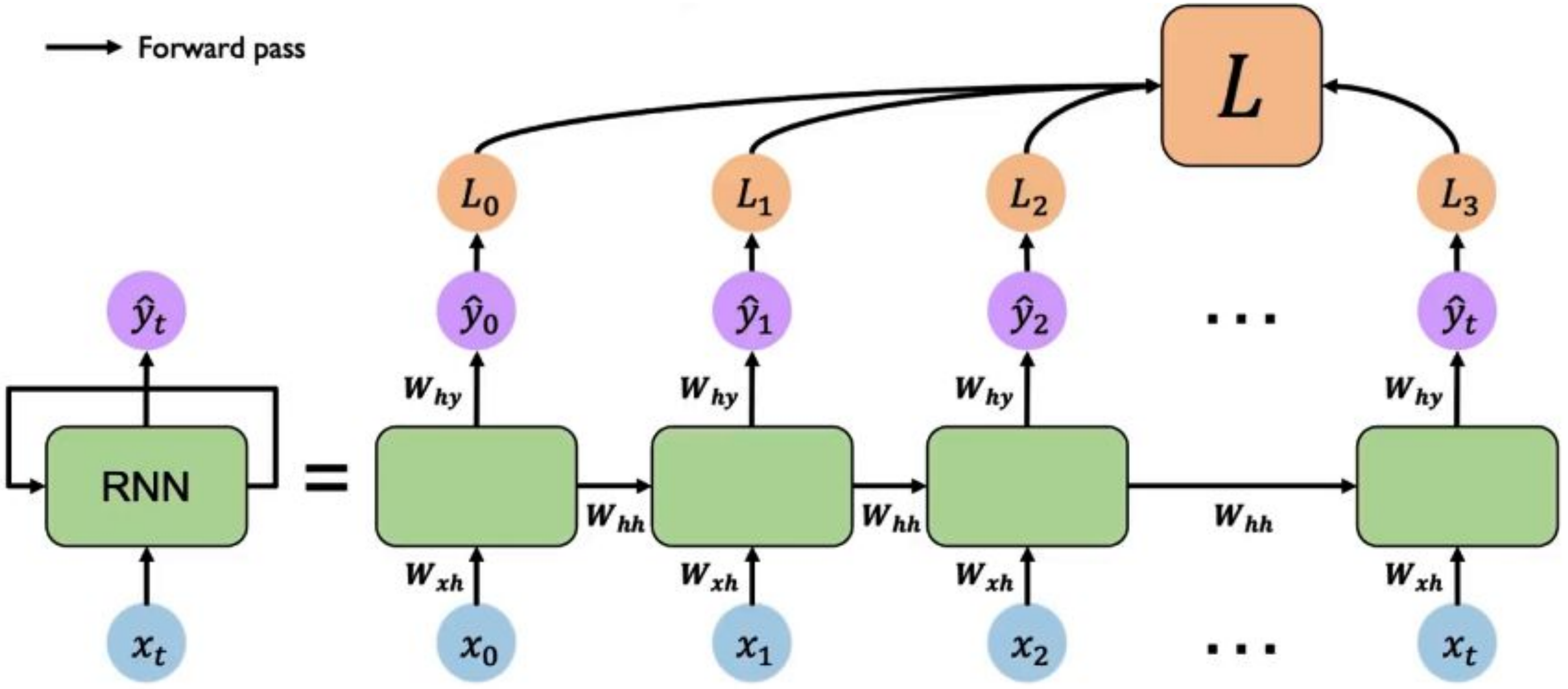
Recall: Backpropagation in Feed Forward Models



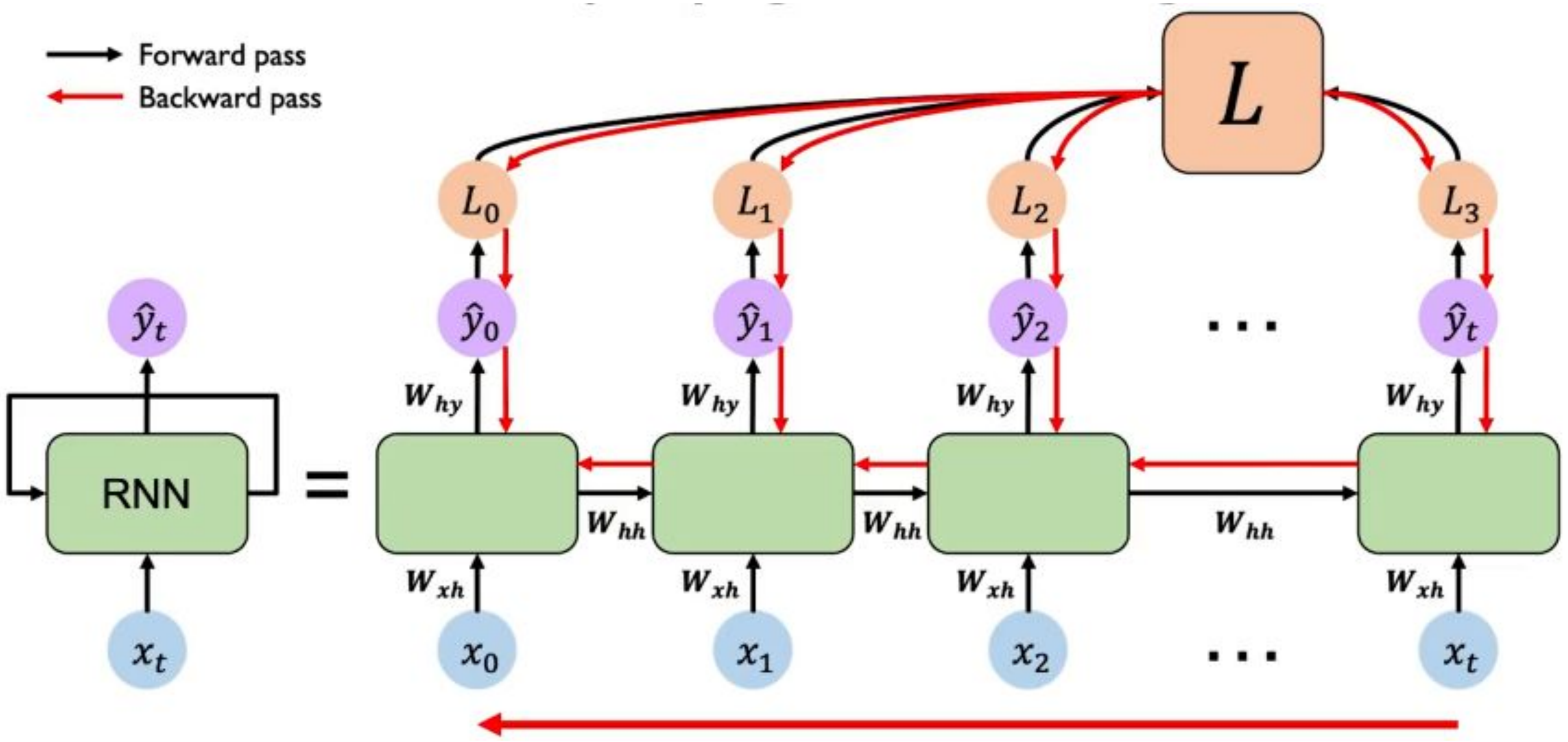
Backpropagation algorithm:

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

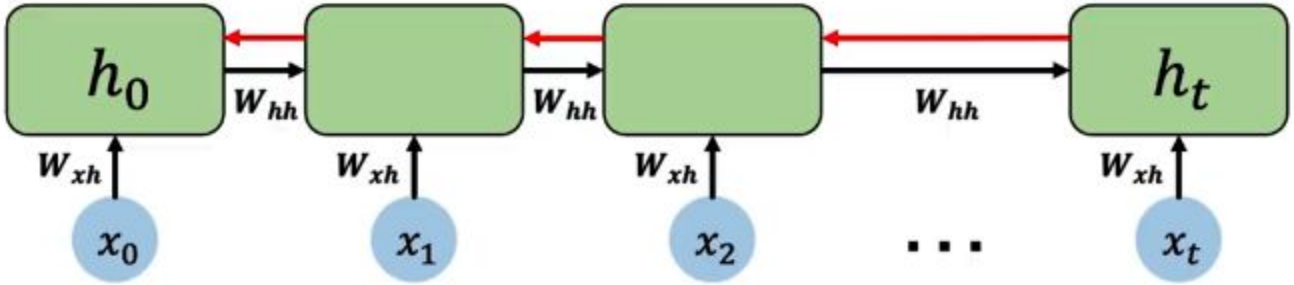
RNNs: Backpropagation Through Time (BTT)



RNNs: Backpropagation Through Time (BTT)

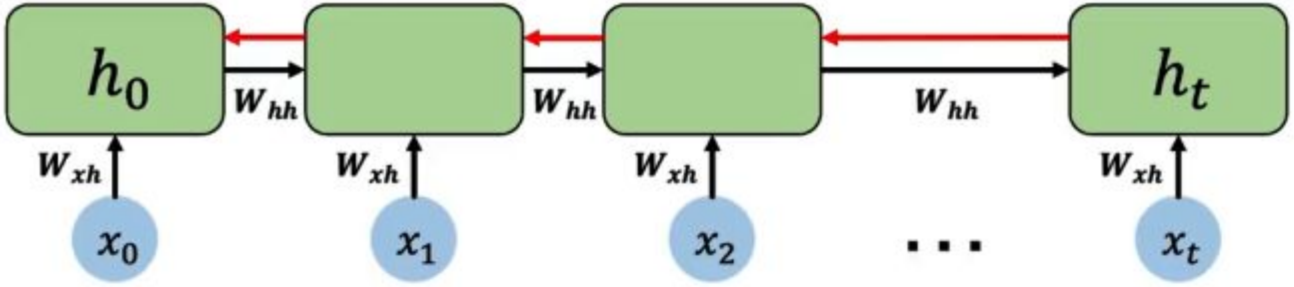


Standard RNN Gradient Flow



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

Standard RNN Gradient Flow: Exploding Gradients

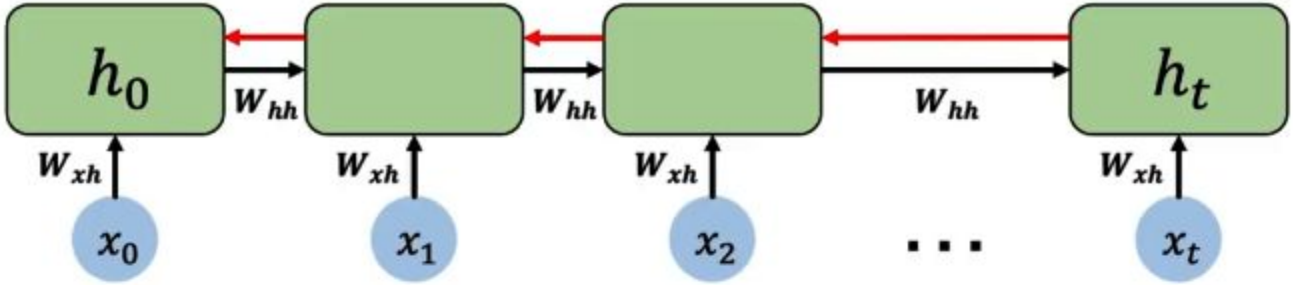


Computing the gradient wrt h_0 involves **many factors of W_{hh} + repeated gradient computation**!

Many values > 1:
exploding gradients

Gradient clipping to scale big 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

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

Problem of Long Term Dependencies

Why are vanishing gradients a problem?

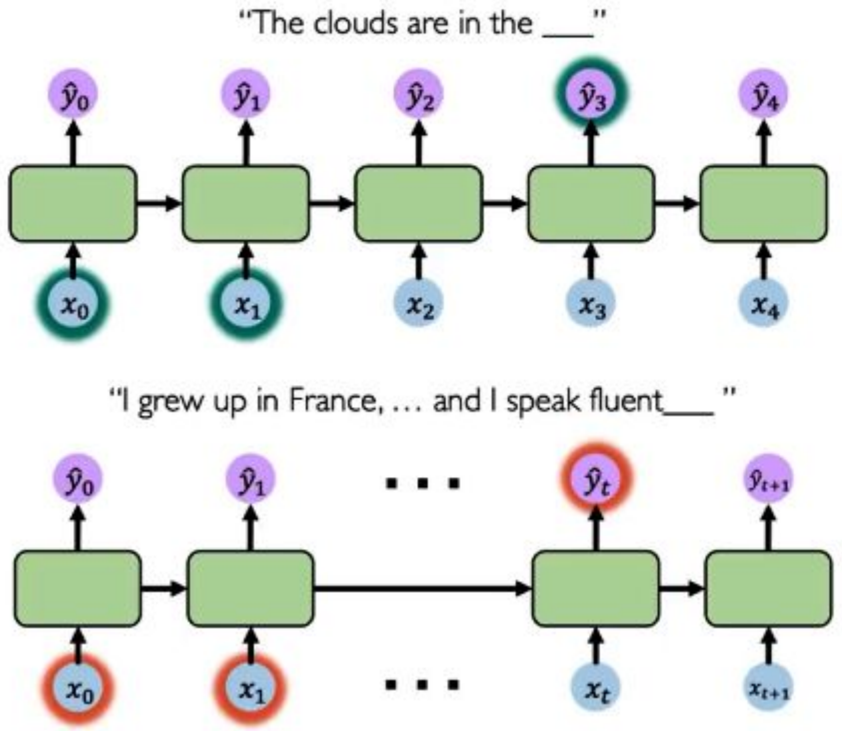
Multiply many **small numbers** together

↓

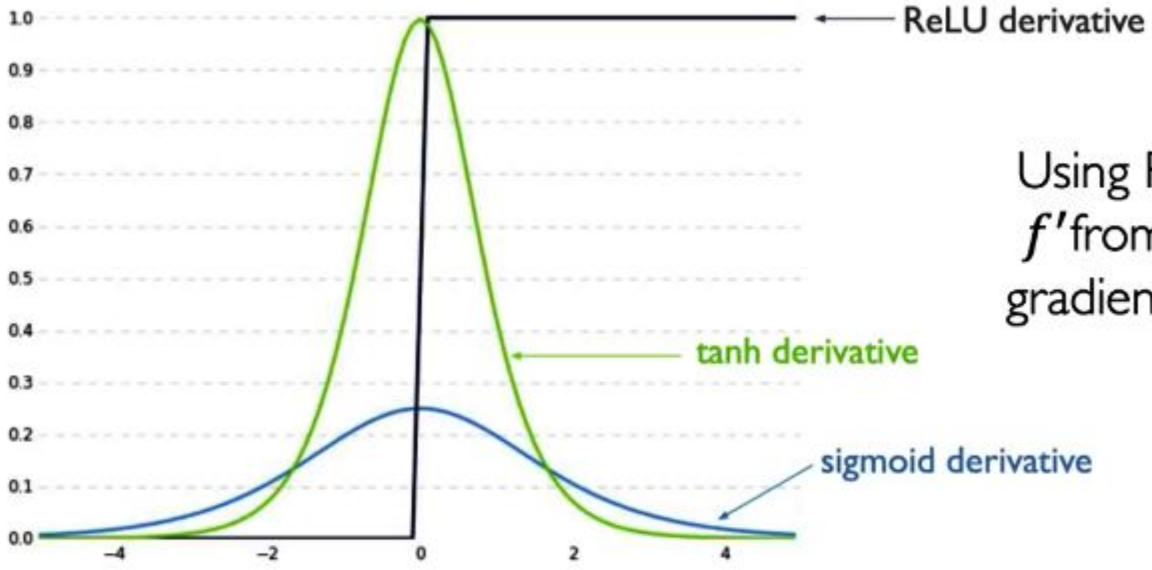
Errors due to further back time steps have smaller and smaller gradients

↓

Bias parameters to capture short-term dependencies



Trick #1: Activation Function



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

Trick #2: Parameter Initialization

Initialize **weights** to identity matrix

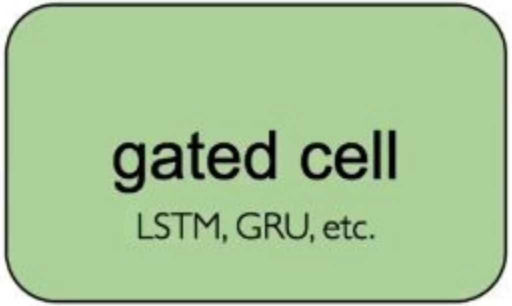
Initialize **biases** to zero

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

This helps prevent the weights from shrinking to zero.

Trick #3: Gated Cells

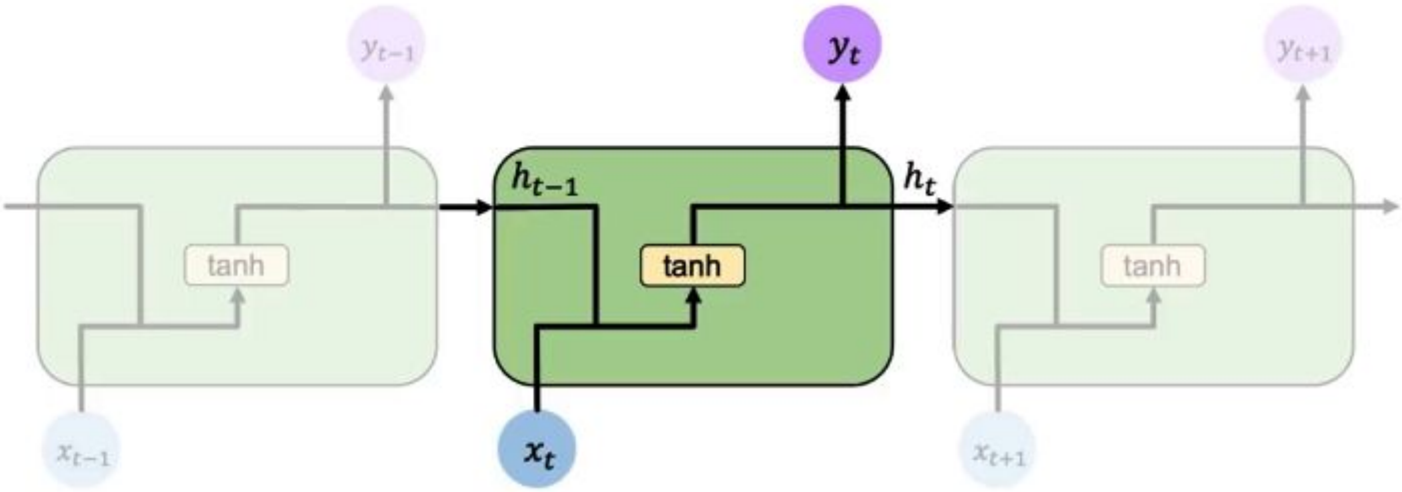
Idea: use a more **complex recurrent unit with gates** to control what information is passed through



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

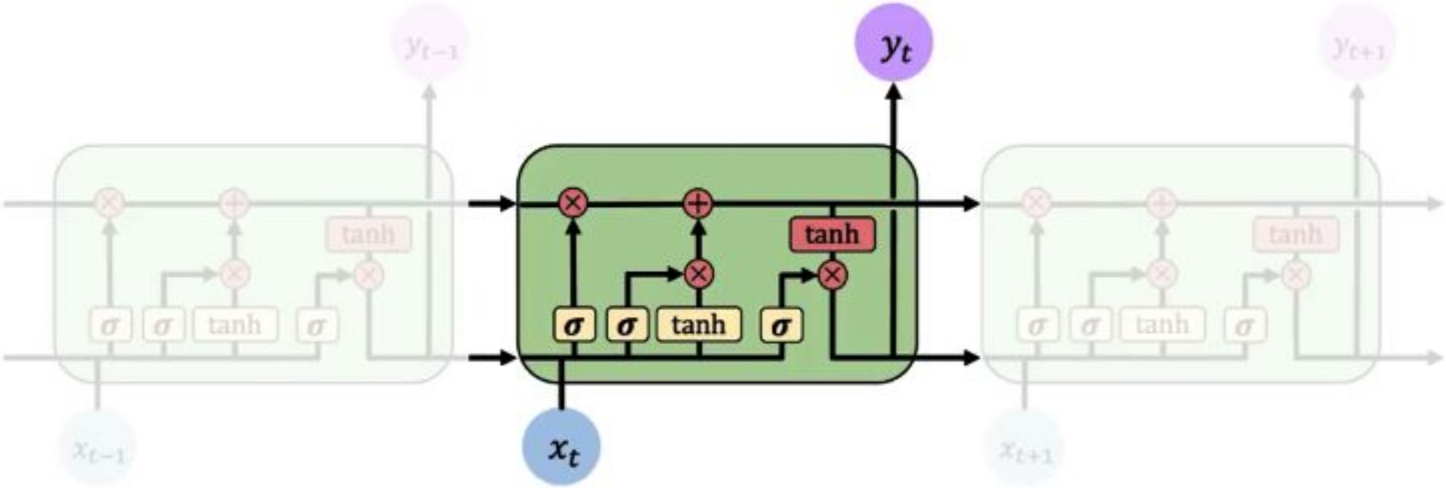
Standard RNN

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



Long Short Term Memory (LSTMs)

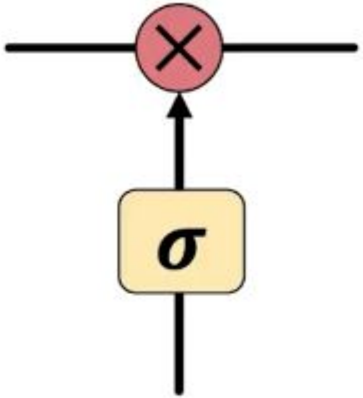
LSTM modules contain **computational blocks** that **control information flow**



LSTM cells are able to track information throughout many timesteps

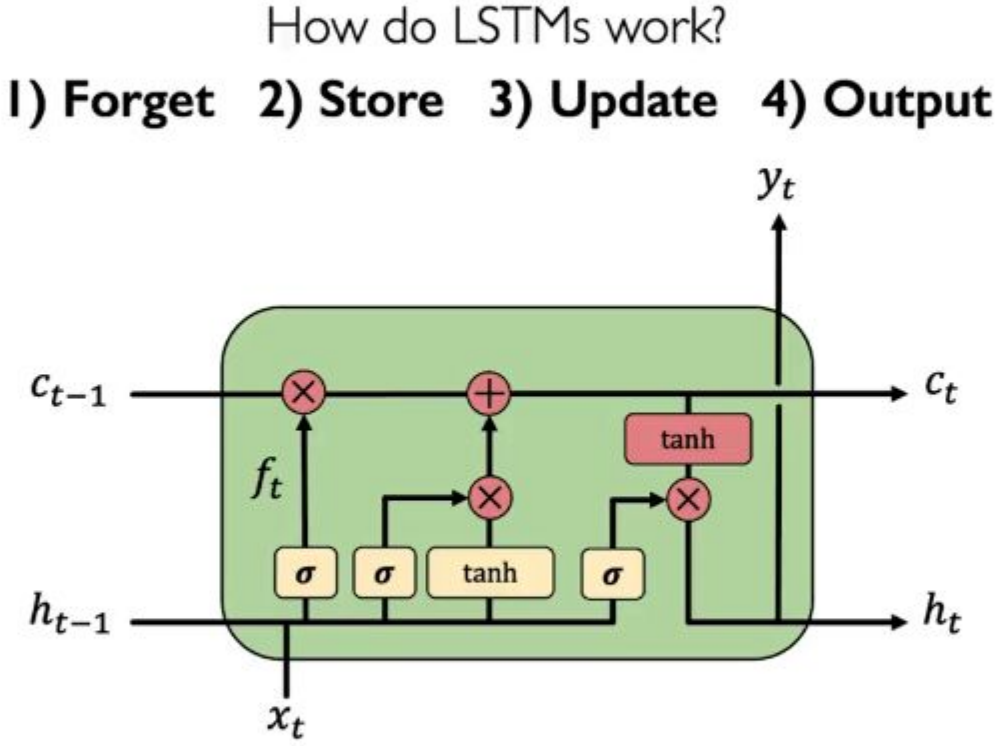
Long Short Term Memory (LSTMs)

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



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

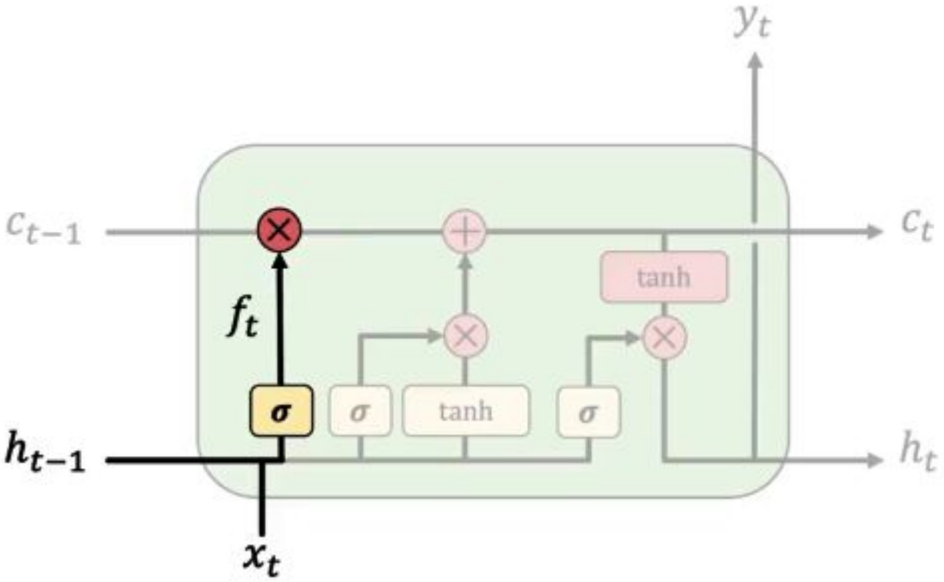
Long Short Term Memory (LSTMs)



Long Short Term Memory (LSTMs)

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

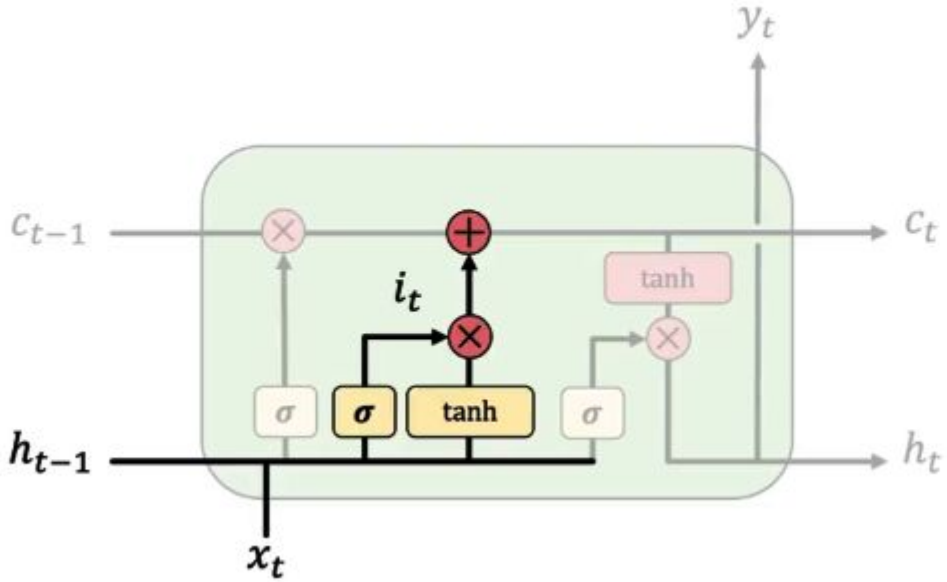
LSTMs **forget irrelevant** parts of the previous state



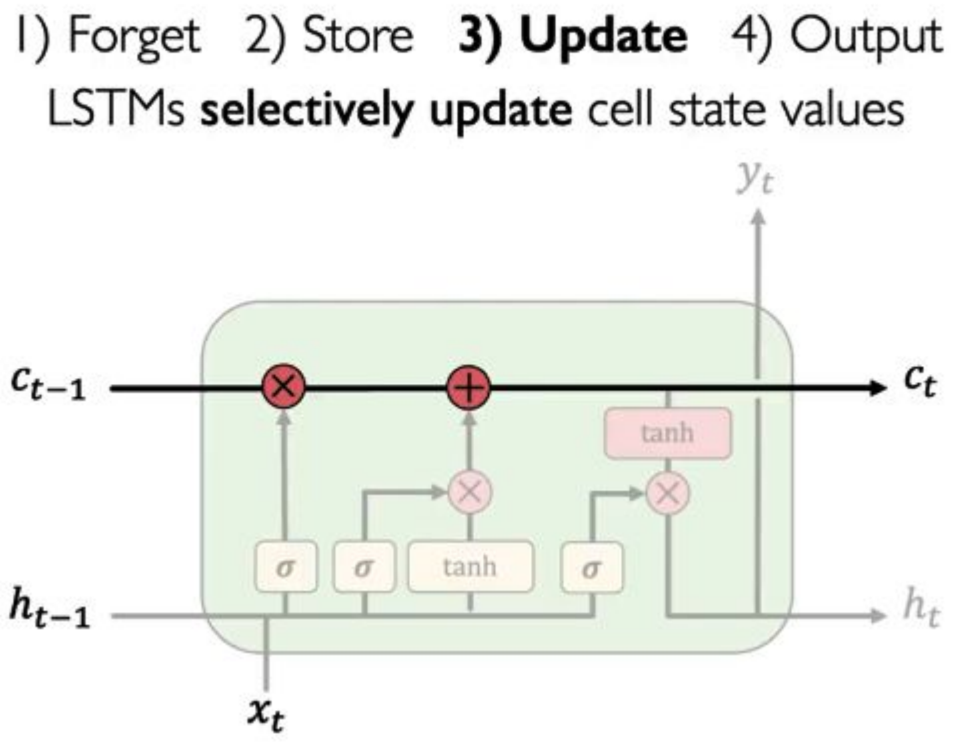
Long Short Term Memory (LSTMs)

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

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



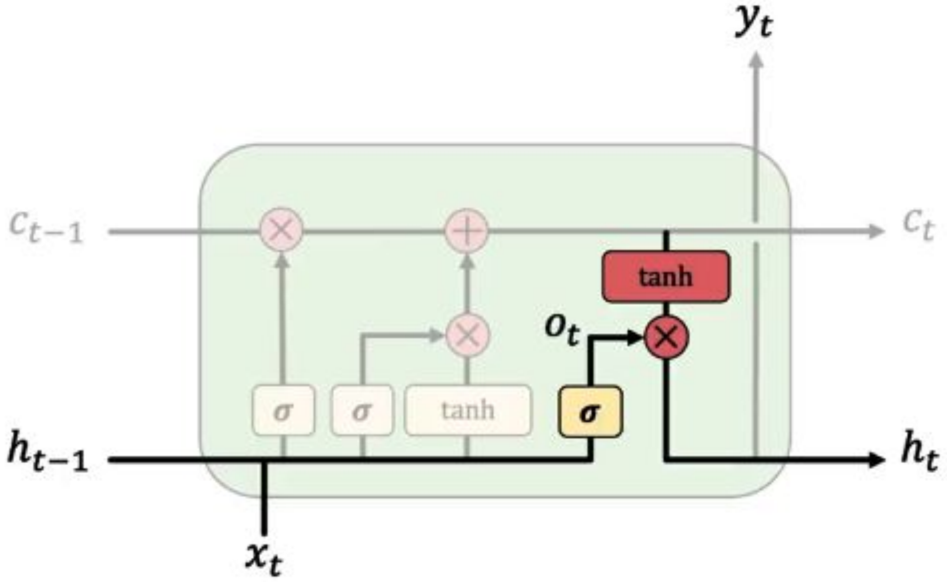
Long Short Term Memory (LSTMs)



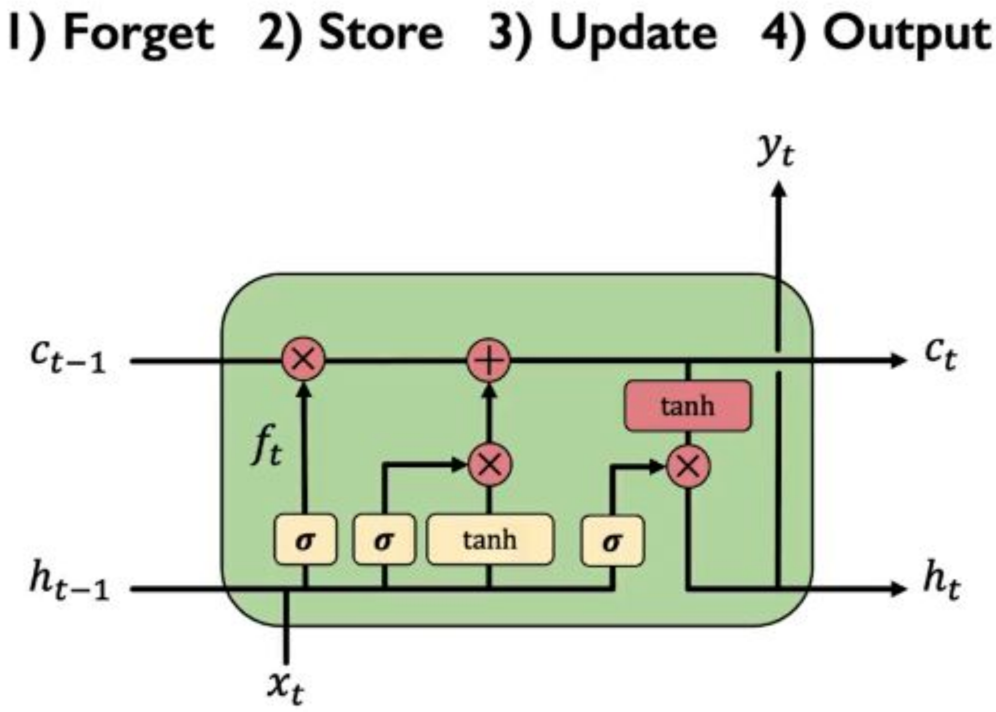
Long Short Term Memory (LSTMs)

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

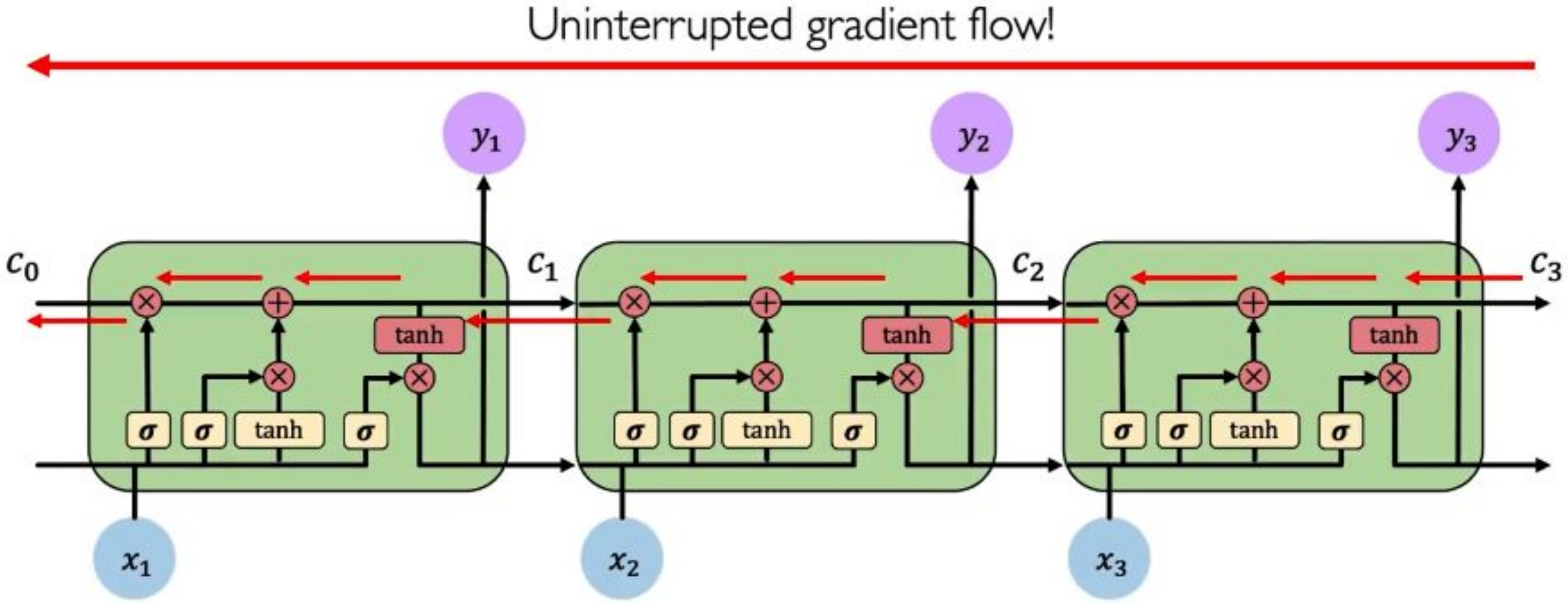
The **output gate** controls what information is sent to the next time step



Long Short Term Memory (LSTMs)



LSTM Gradient Flow



LSTMs: Key Concepts

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

Pros & Cons of LSTMs

- Advantages
  - They are able to model long-term sequence dependencies.
  - They are more robust to the problem of short memory than ‘Vanilla’ RNNs
- Disadvantages
  - They increase the computing complexity compared to the RNN with the introduction of more parameters to learn.
  - The memory required is higher than the one of ‘Vanilla’ RNNs due to the presence of several memory cells.

END