Advanced Deep Learning Architectures

COMP 5214 & ELEC 5680

Instructor: Dr. Qifeng Chen
https://cqf.io

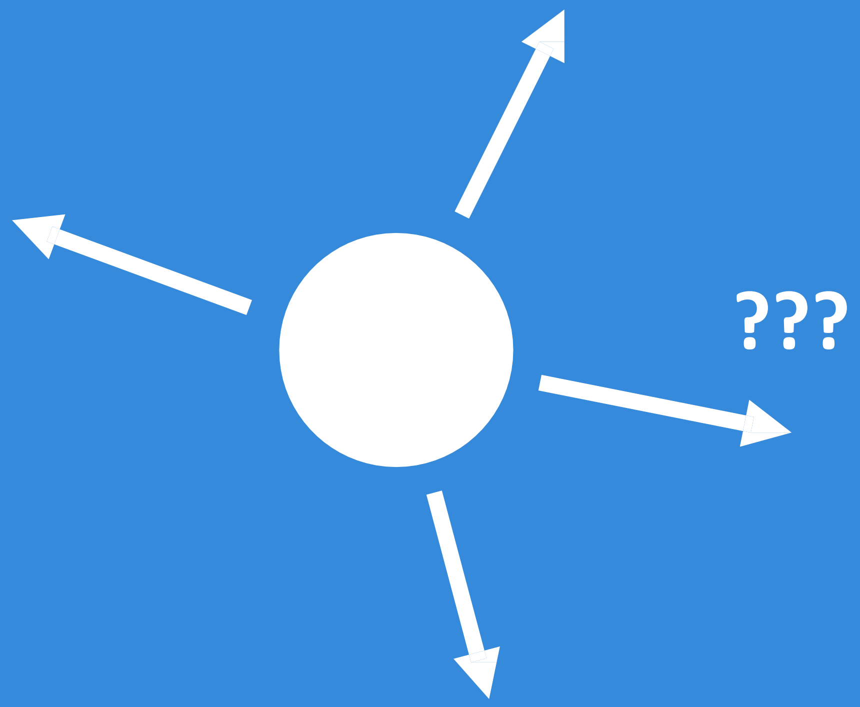
Credit: http://introtodeeplearning.com/

Logistics

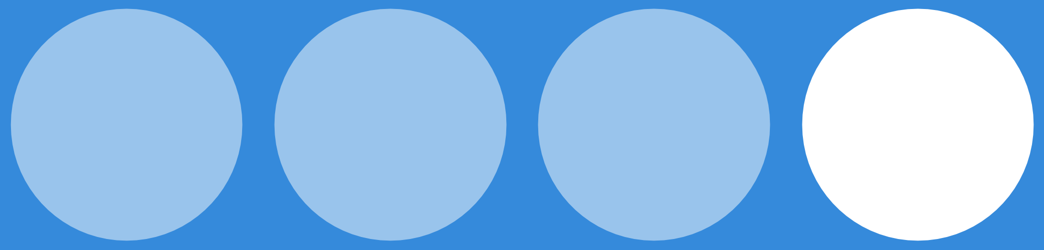
- Assignment 1 is released today
- Pay attention to project proposal

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

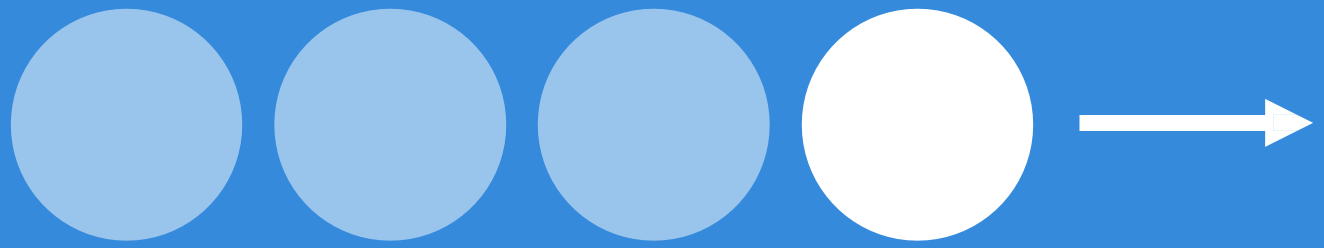
Given an image of a ball,
can you predict where it will go next?



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

character:

6.S191 Introduction to Deep Learning

word:

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 two words

predict the next word

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

Problem #1: can’t model long-term dependencies

“France is where I grew up, but I now live in Boston. I speak fluent ____.”

J’aime 6.S191!

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

morning I took this cat

prediction

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:

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

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 ...]

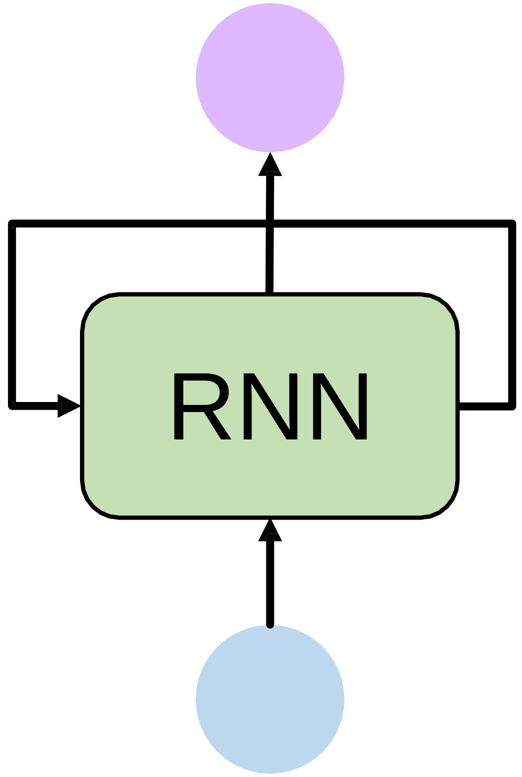
this morning

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

Sequence modeling: design criteria

To model sequences, we need to:

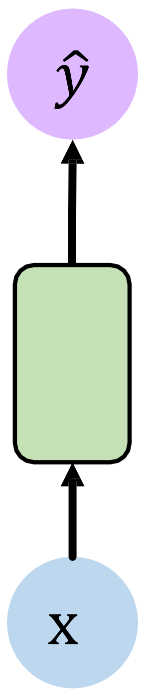
1. Handle variable-length sequences
2. Track long-term dependencies
3. Maintain information about order
4. Share parameters across the sequence



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

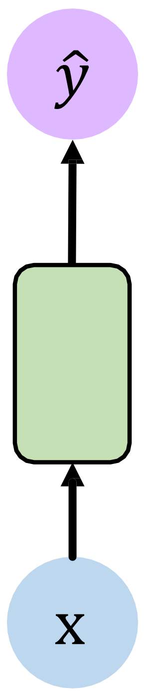
Recurrent Neural Networks (RNNs)

Standard feed-forward neural network

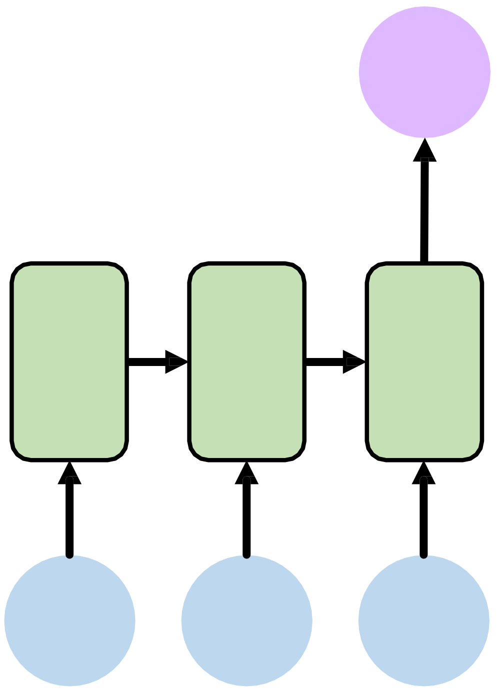


One to One
"Vanilla" neural network

Recurrent neural networks: sequence modeling

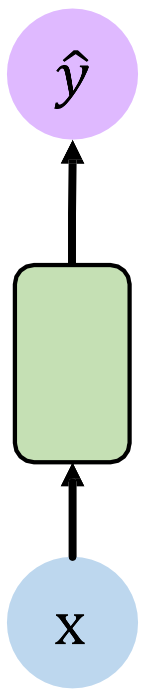


One to One
"Vanilla" neural network

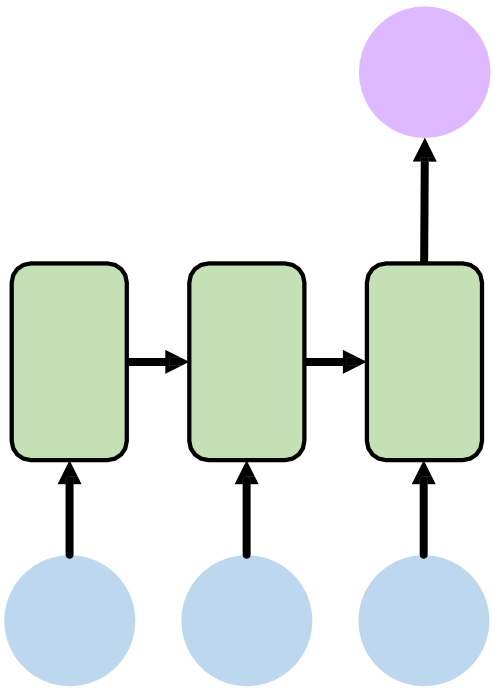


Many to One
Sentiment Classification

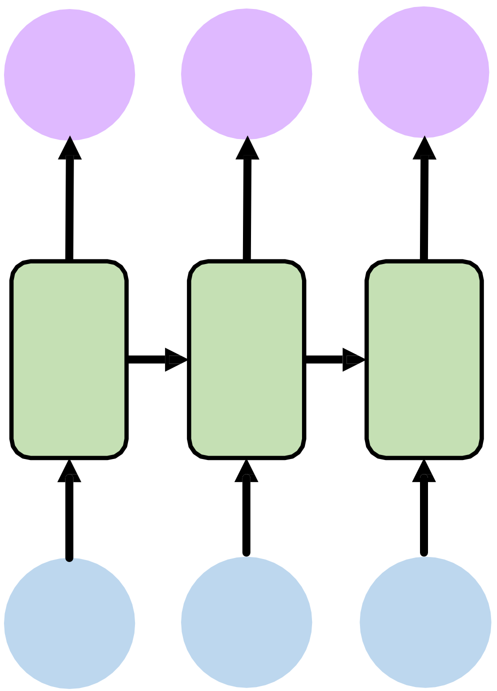
Recurrent neural networks: sequence modeling



One to One
"Vanilla" neural network



Many to One
Sentiment Classification

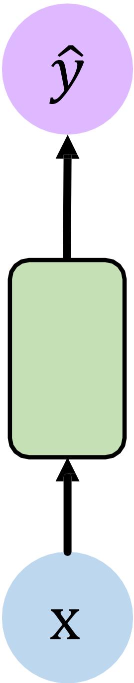


Many to Many
Music Generation

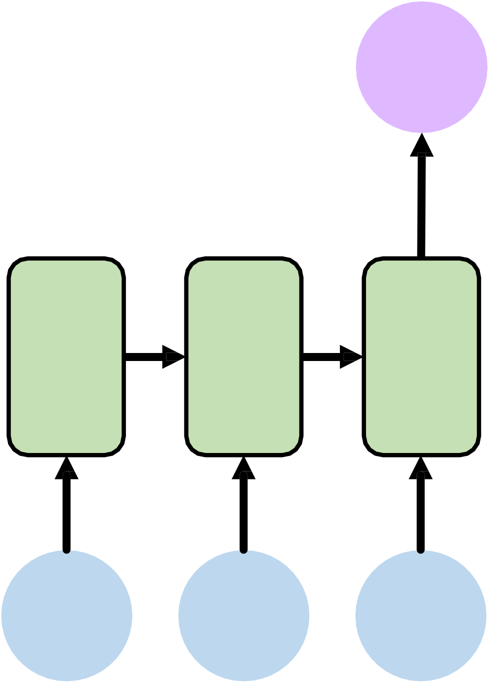
6.S191 Lab!

[1]

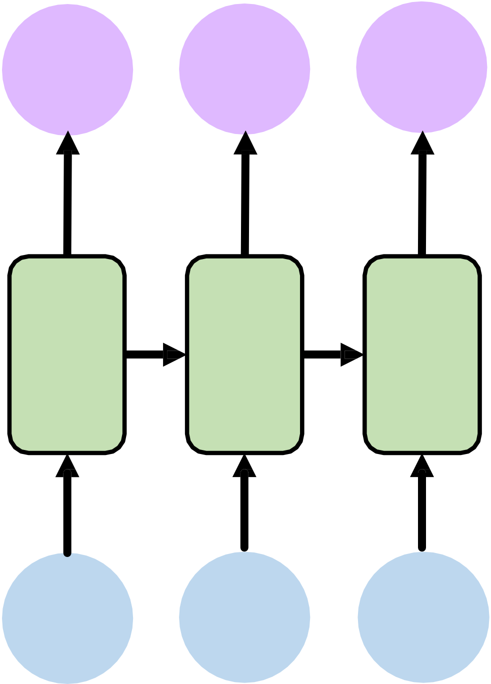
Recurrent neural networks: sequence modeling



One to One
"Vanilla" neural network



Many to One
Sentiment Classification

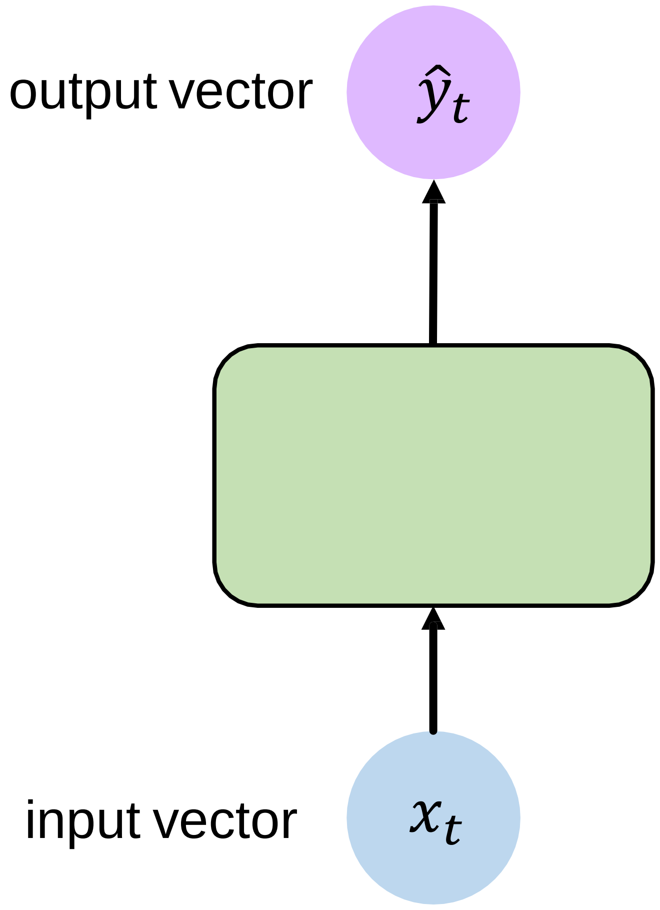


Many to Many
Music Generation

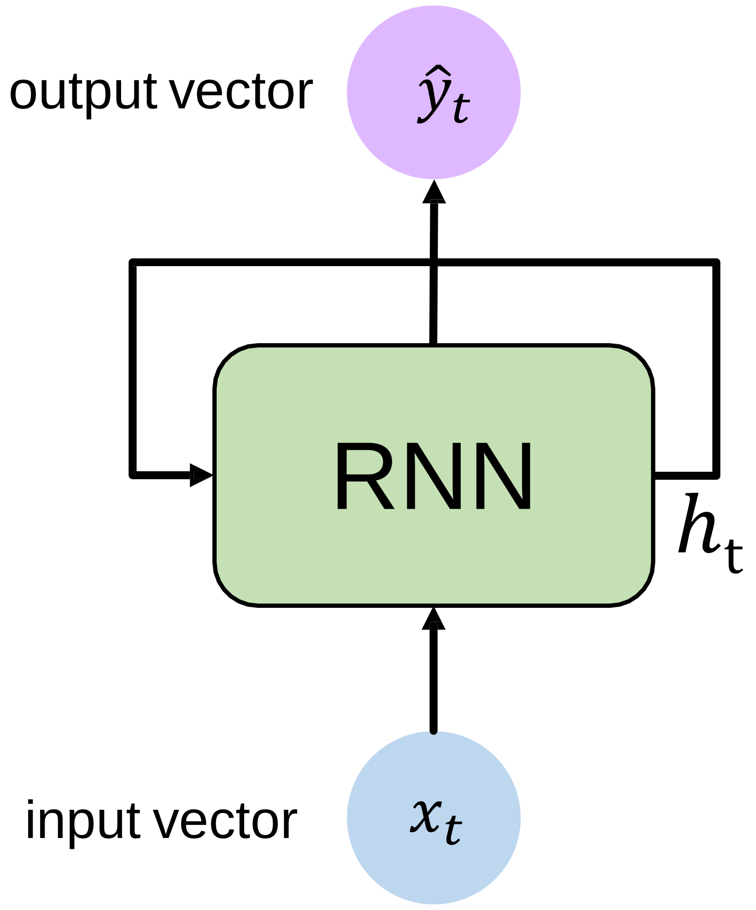
6.S191 Lab!

… and many other architectures and applications

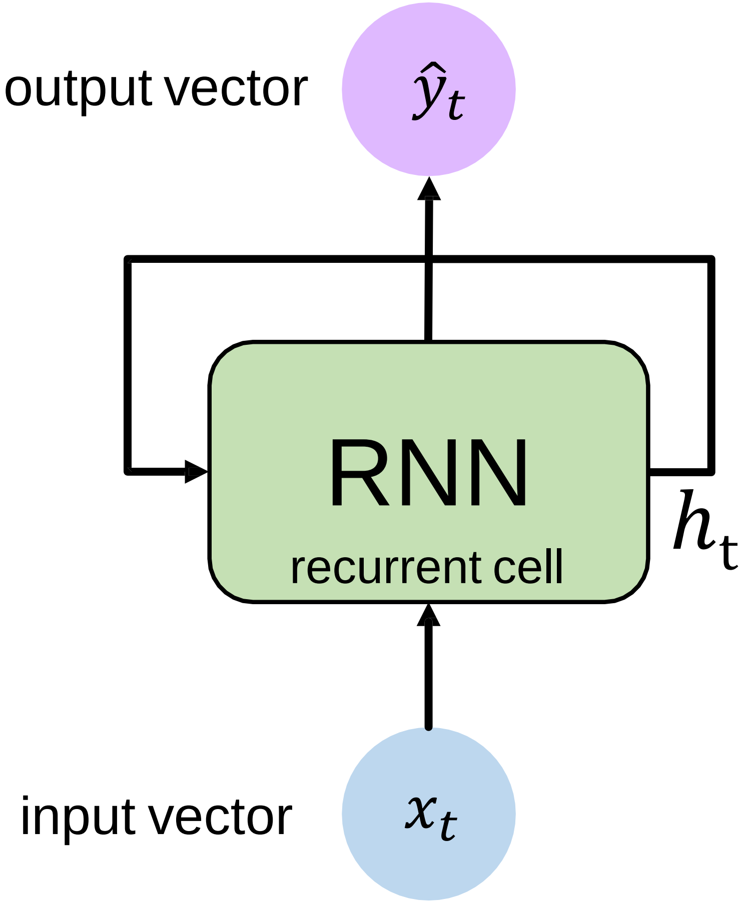
A standard “vanilla” neural network



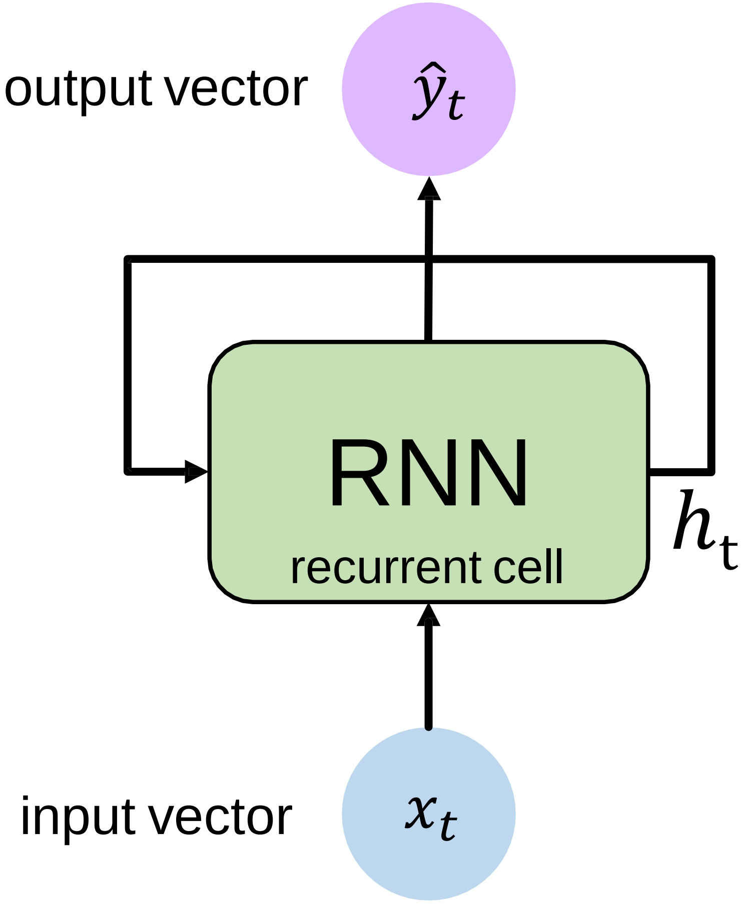
A recurrent neural network (RNN)



A recurrent neural network (RNN)

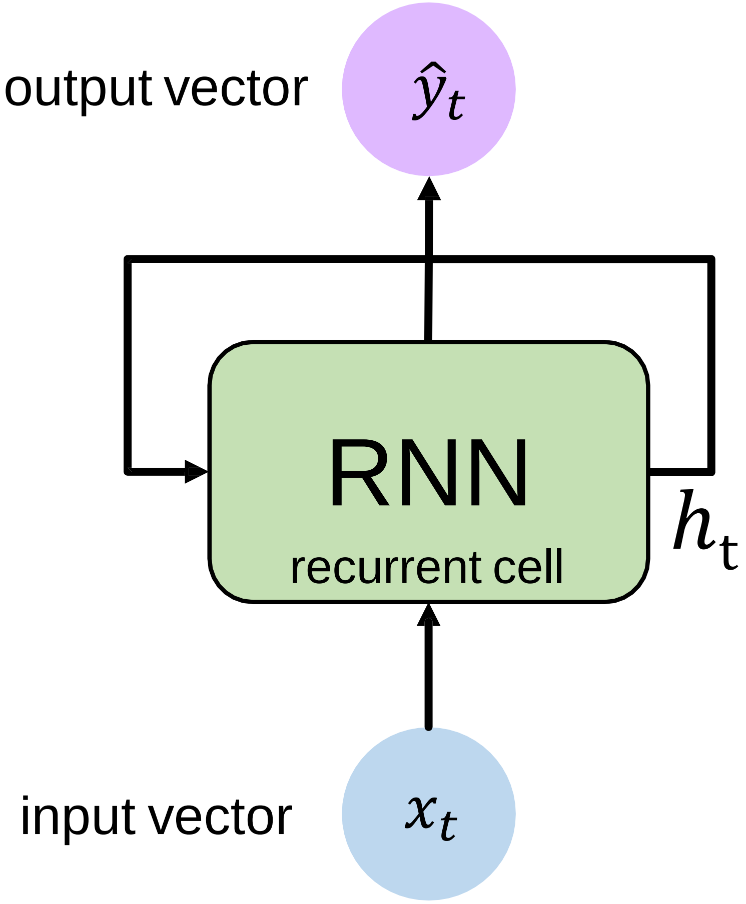


A recurrent neural network (RNN)



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

A recurrent neural network (RNN)

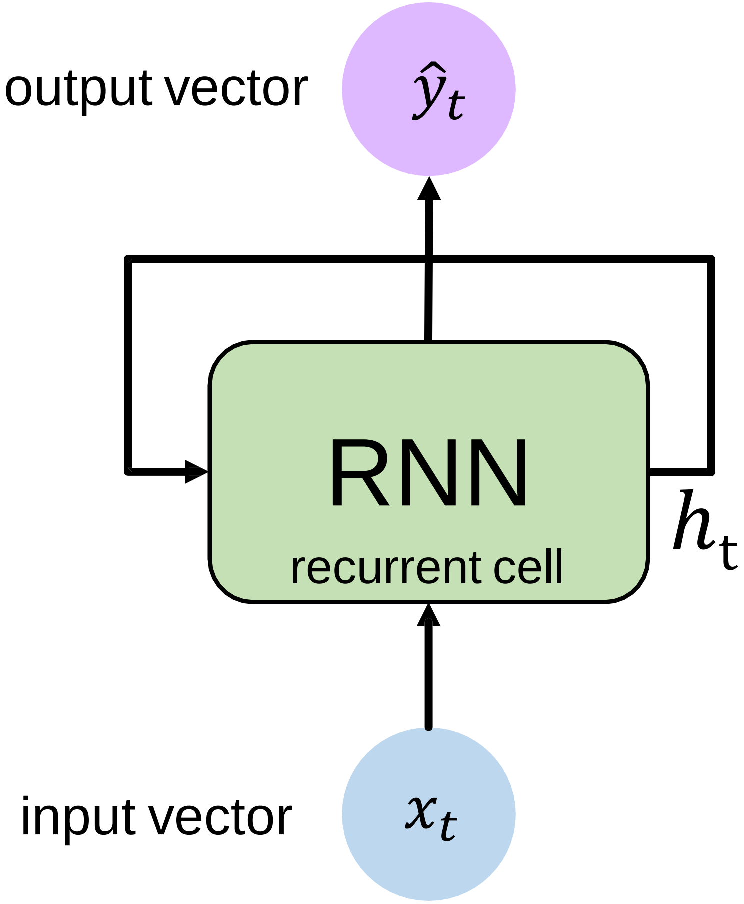


Apply a recurrence relation at every time step to process a sequence:

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

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

A recurrent neural network (RNN)

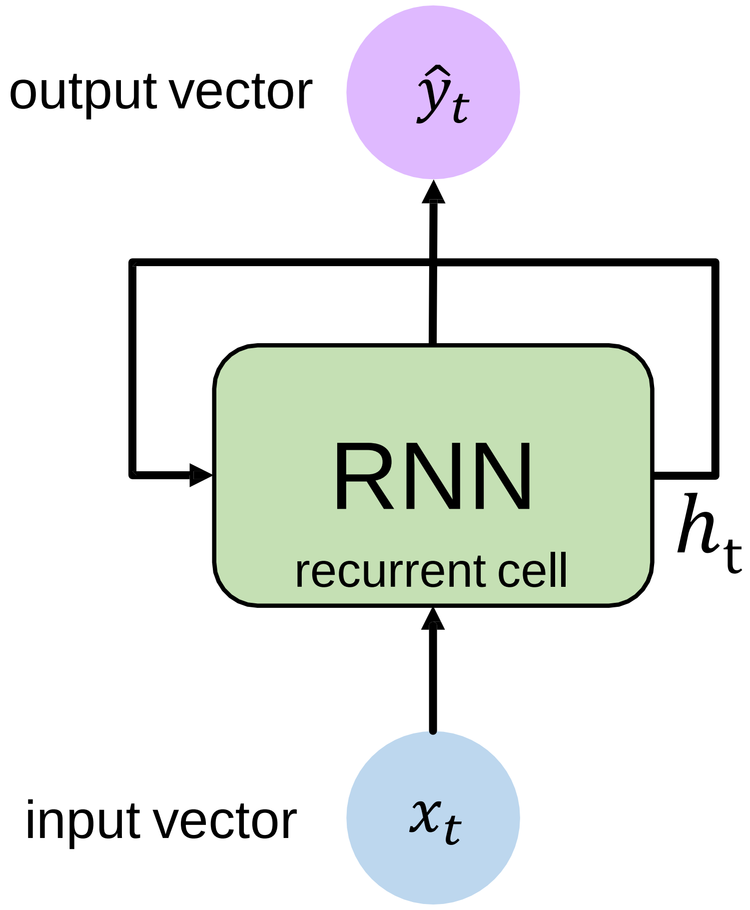


Apply a recurrence relation at every time step to process a sequence:

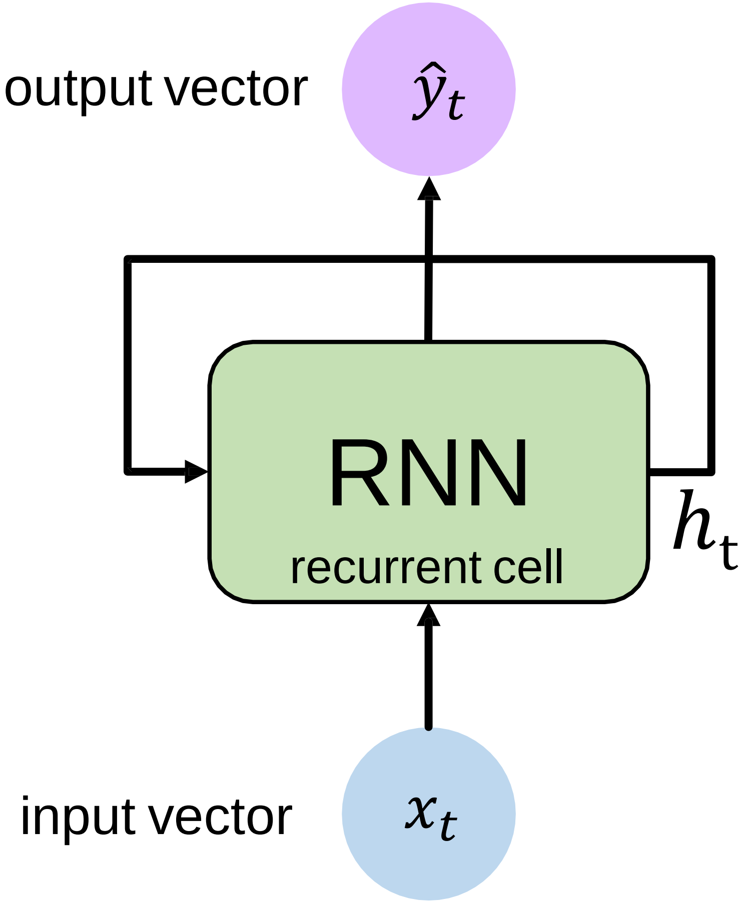
$$\underbrace{\boxed{h_t}}_{\text{cell state}} = \underbrace{\boxed{f_W}}_{\text{function parameterized by W}}(\underbrace{\boxed{h_{t-1}}}_{\text{old state}}, \underbrace{\boxed{x_t}}_{\text{input vector at time step } t})$$

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

RNN state update and output

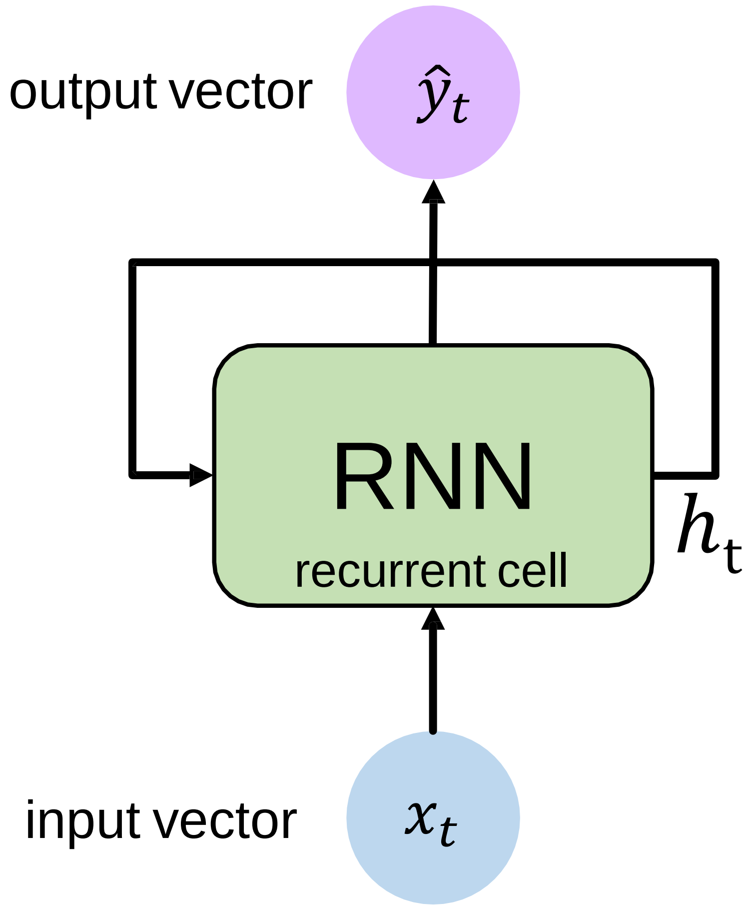


RNN state update and output



InputVector

RNN state update and output

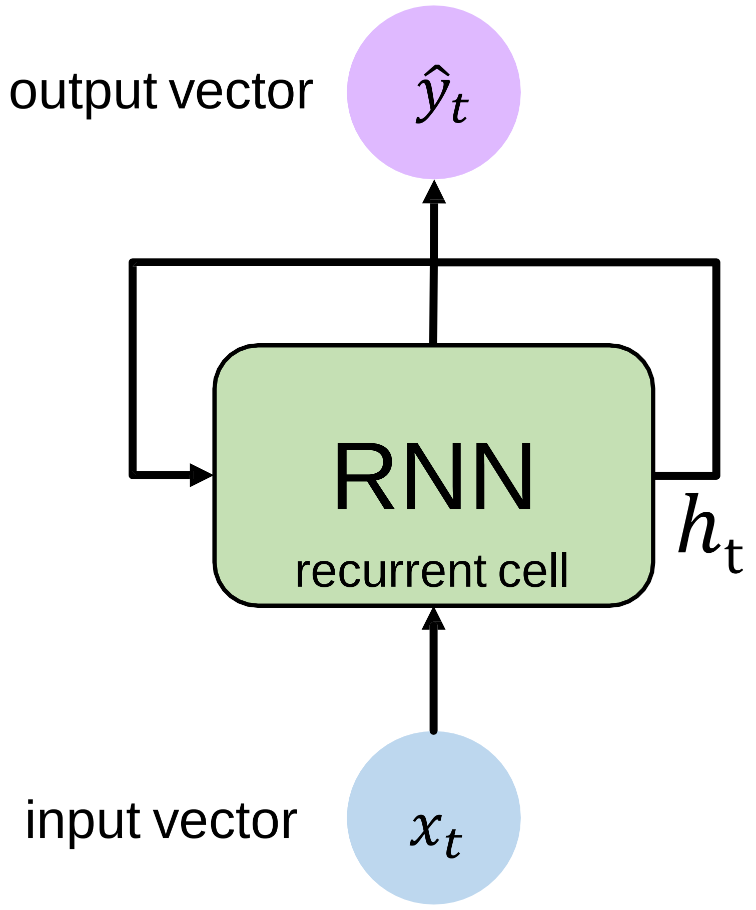


Update Hidden State

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

InputVector

RNN state update and output



Output Vector

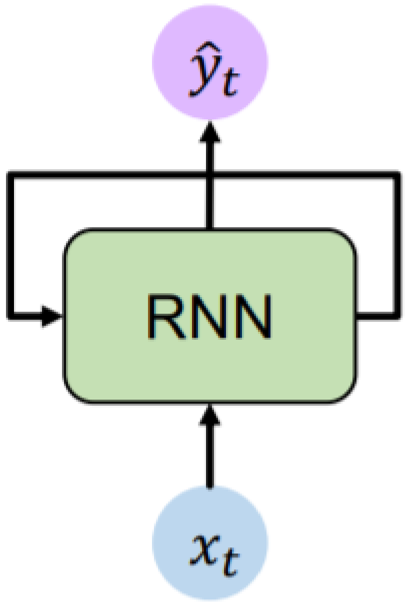
$$\hat{y}_t = \boldsymbol{W}_{\boldsymbol{hy}} h_t$$

Update Hidden State

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

InputVector

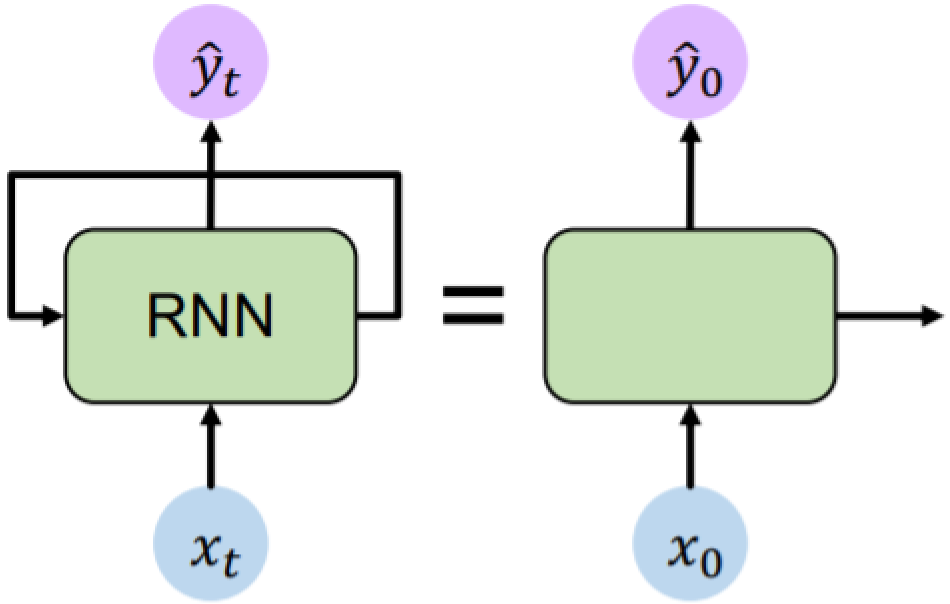
RNNs: computational graph across time



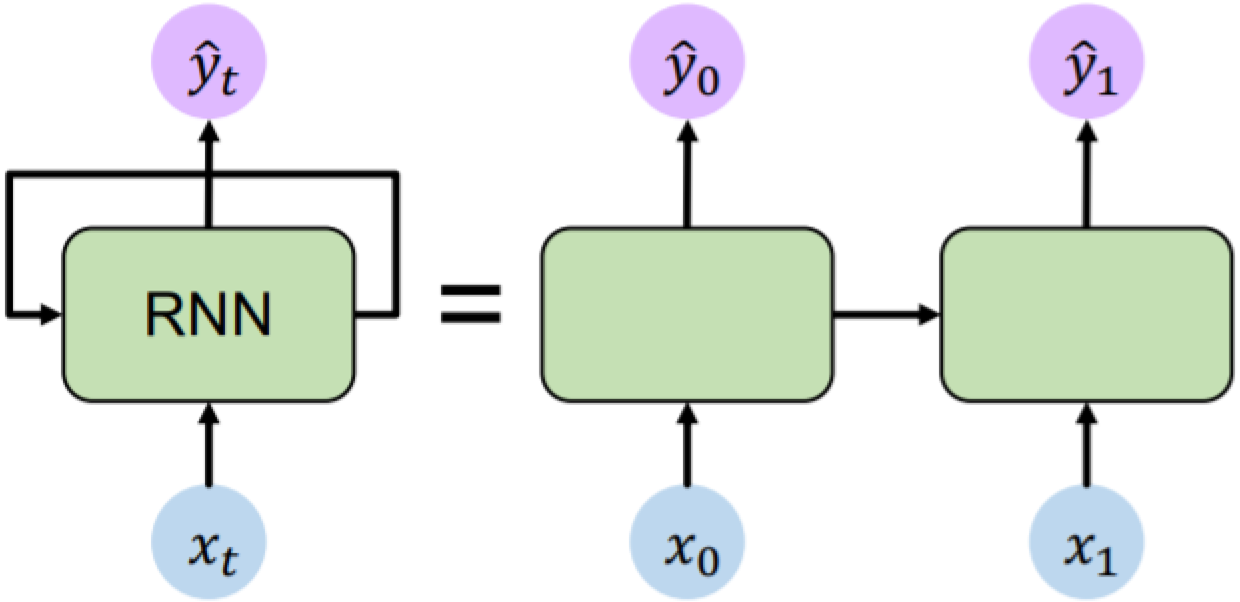
=

Represent as computational graph unrolled across time

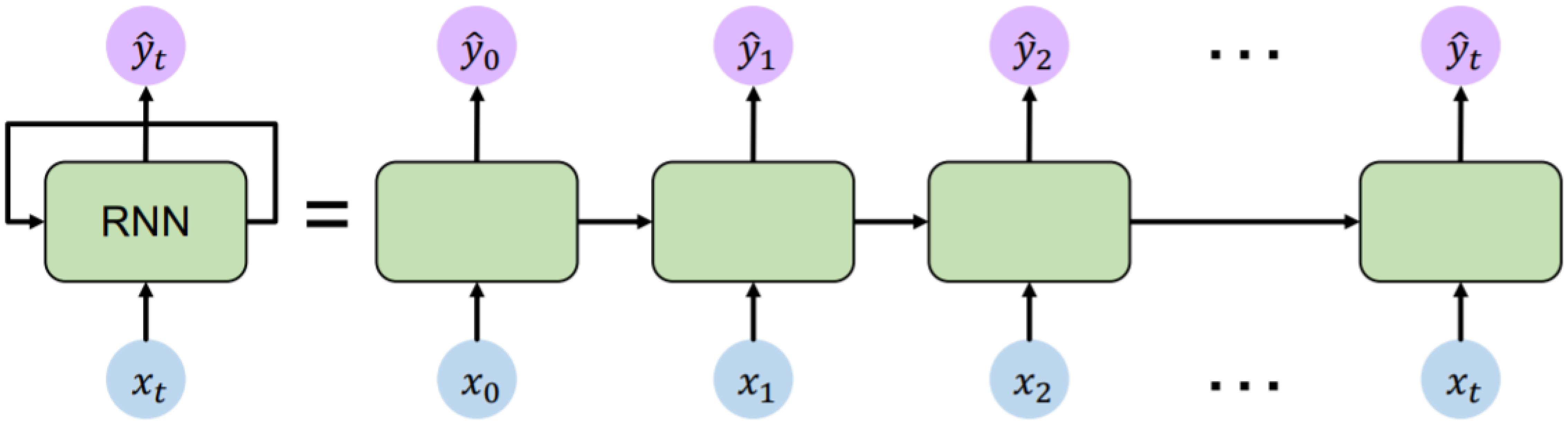
RNNs: computational graph across time



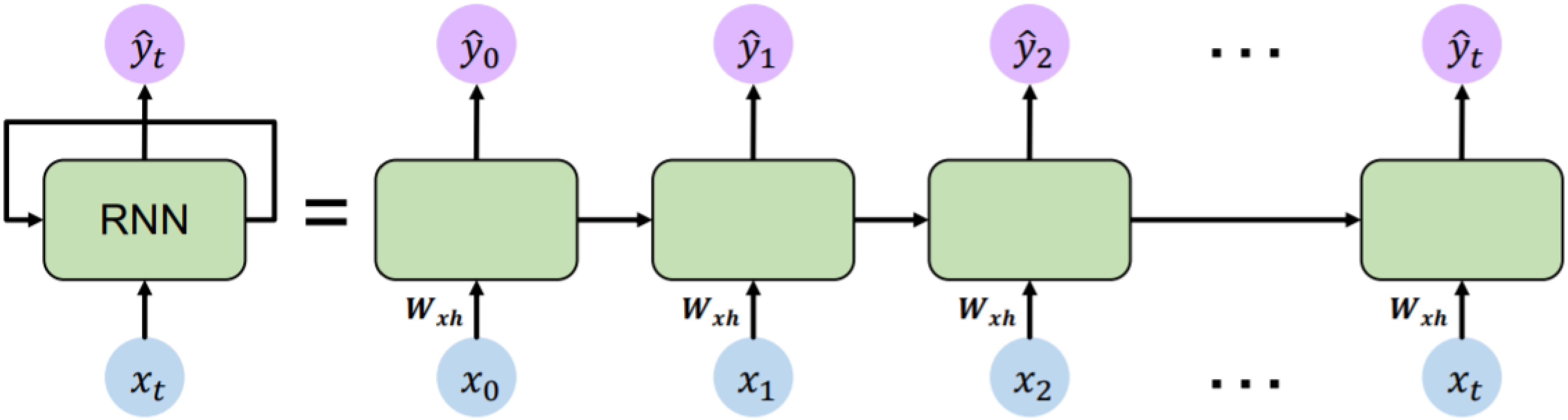
RNNs: computational graph across time



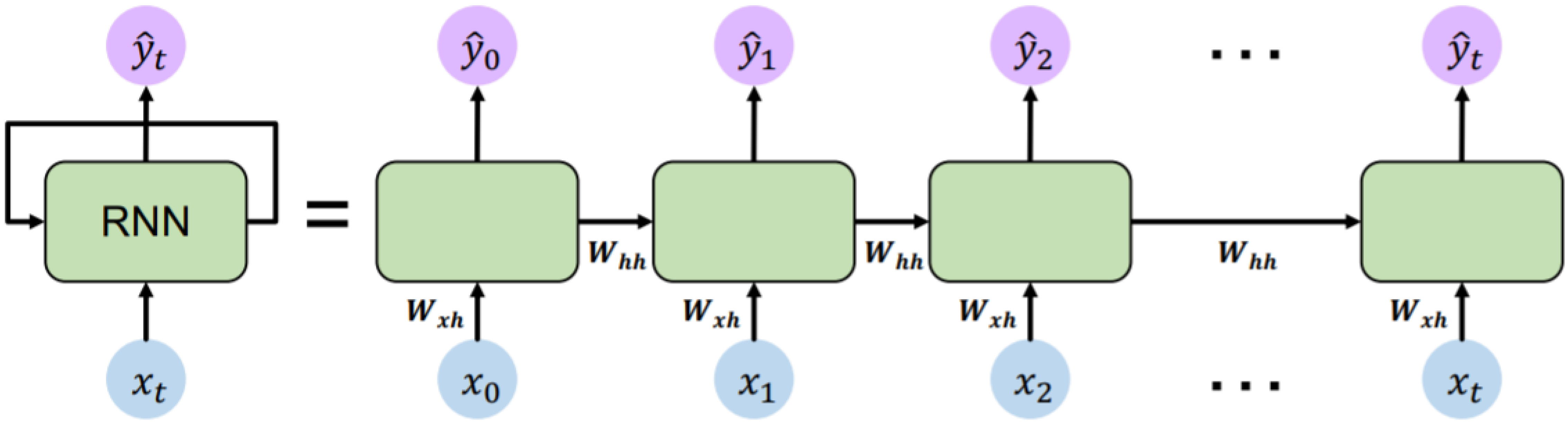
RNNs: computational graph across time



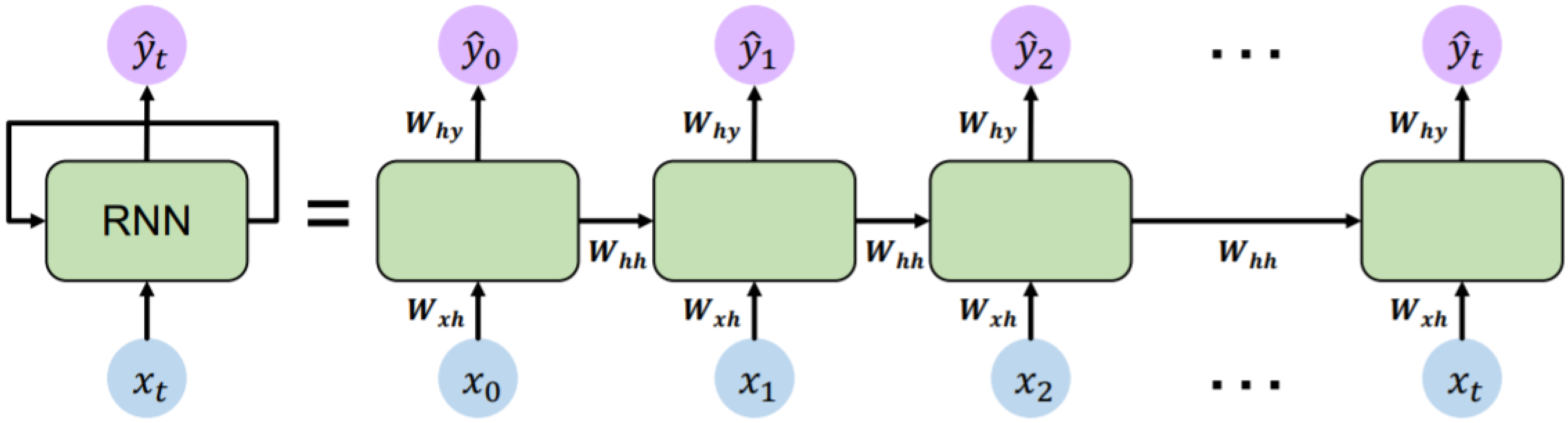
RNNs: computational graph across time



RNNs: computational graph across time

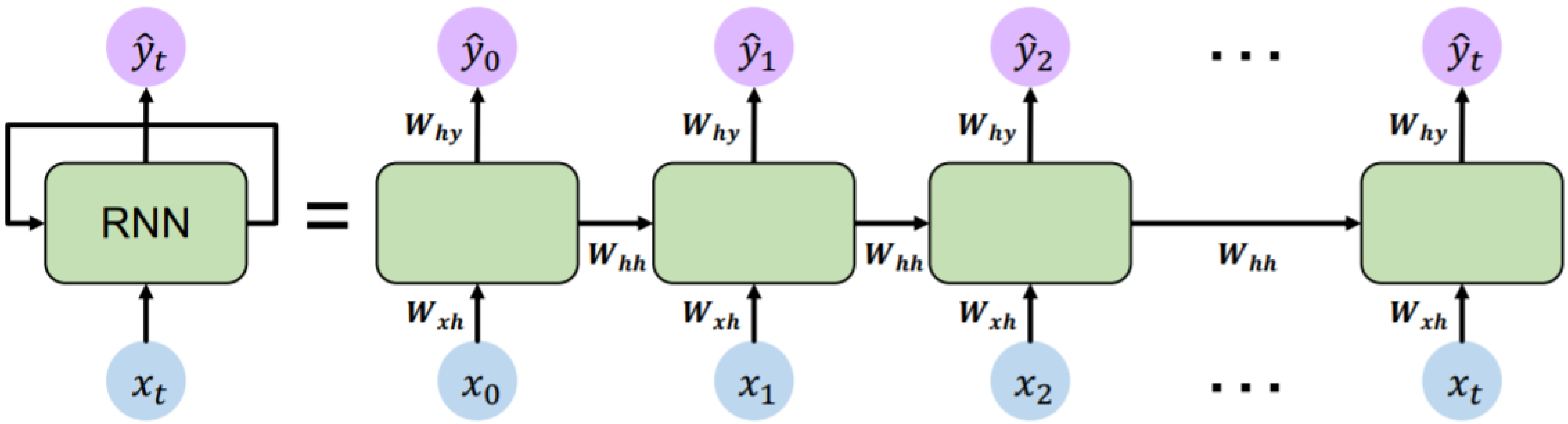


RNNs: computational graph across time



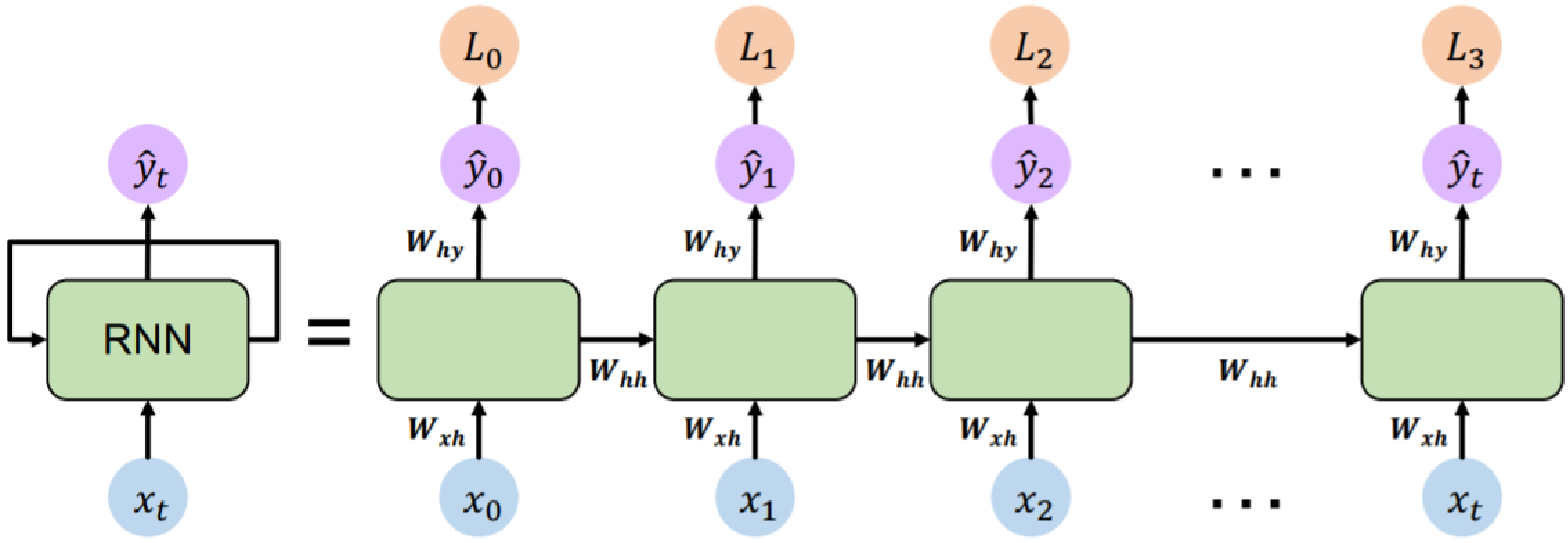
RNNs: computational graph across time

Re-use the same weight matrices at every time step

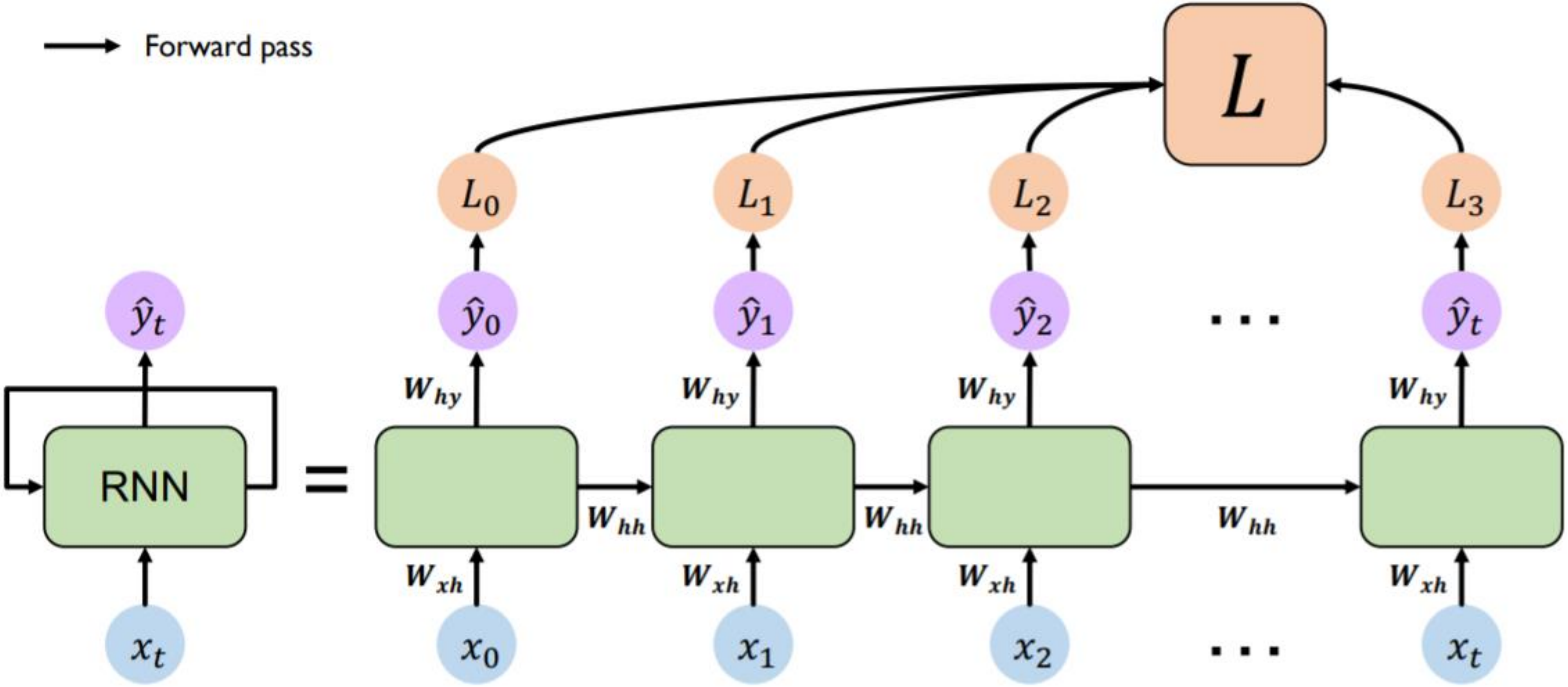


RNNs: computational graph across time

Forward pass

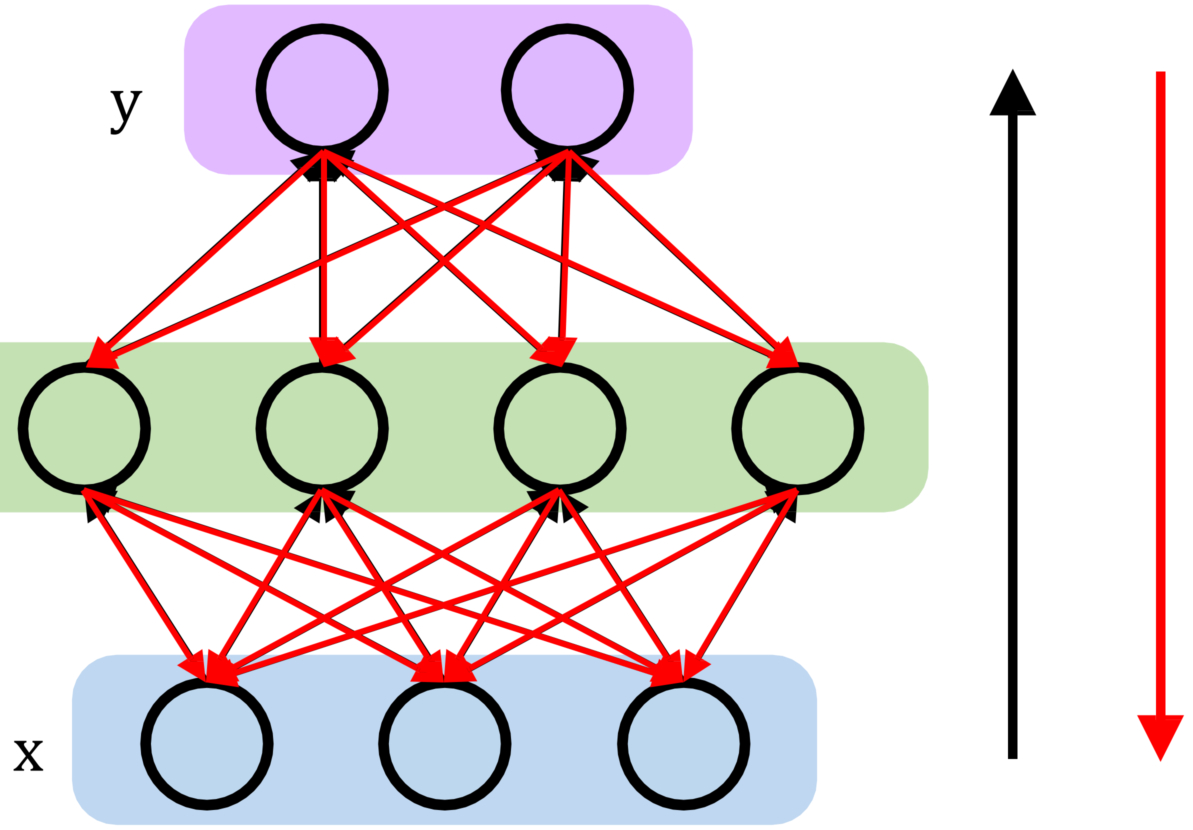


RNNs: computational graph across time



Backpropagation Through Time (BPTT)

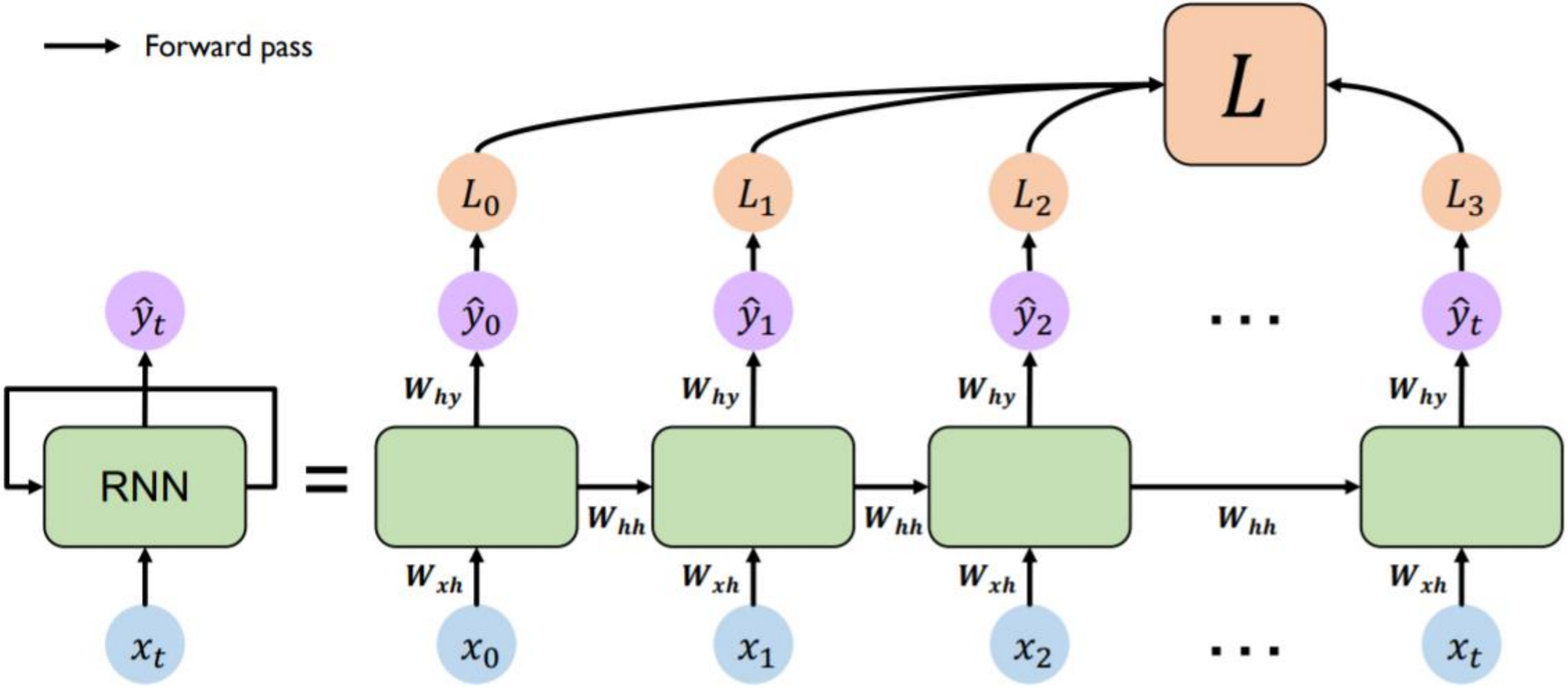
Recall: backpropagation in feed forward models



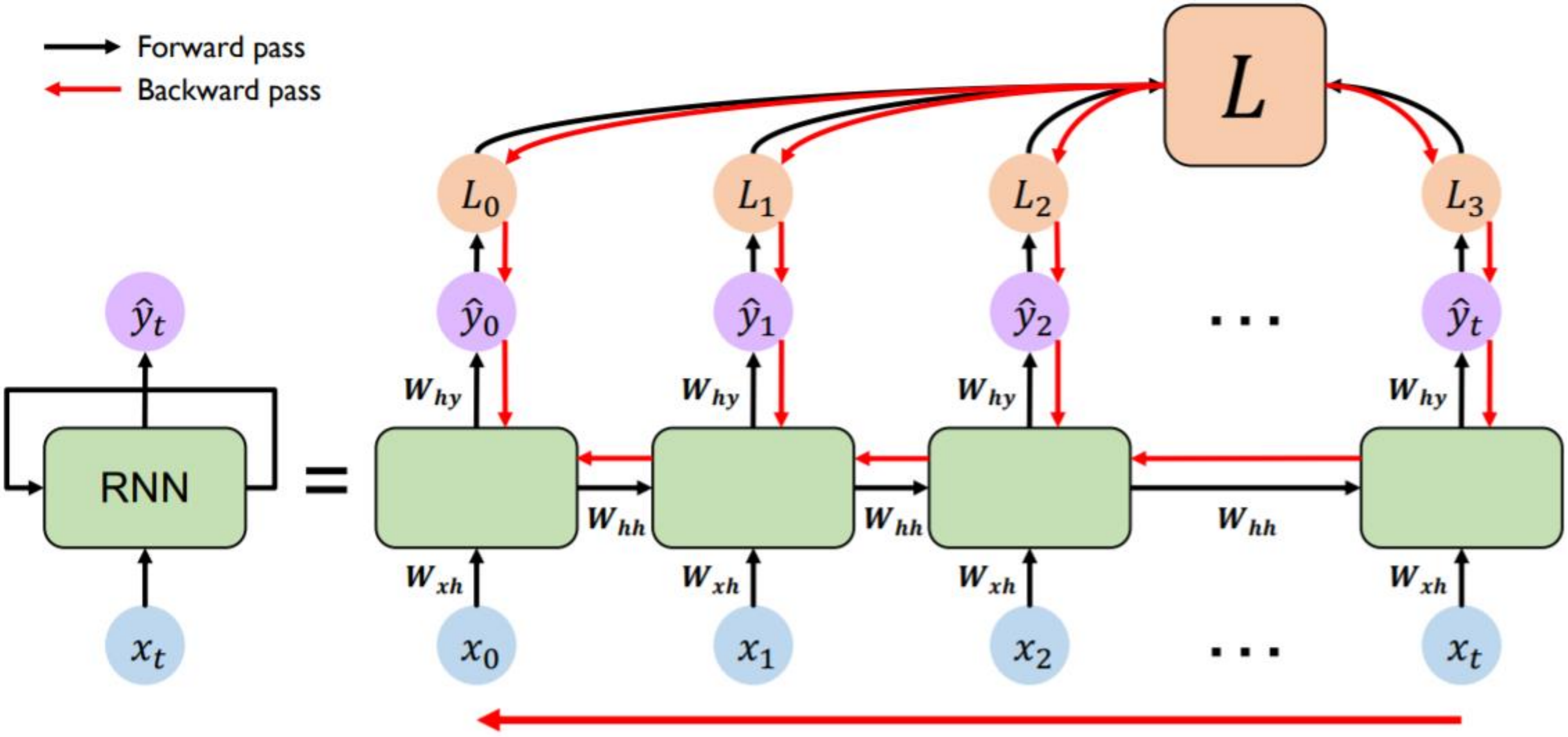
Backpropagation algorithm:

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

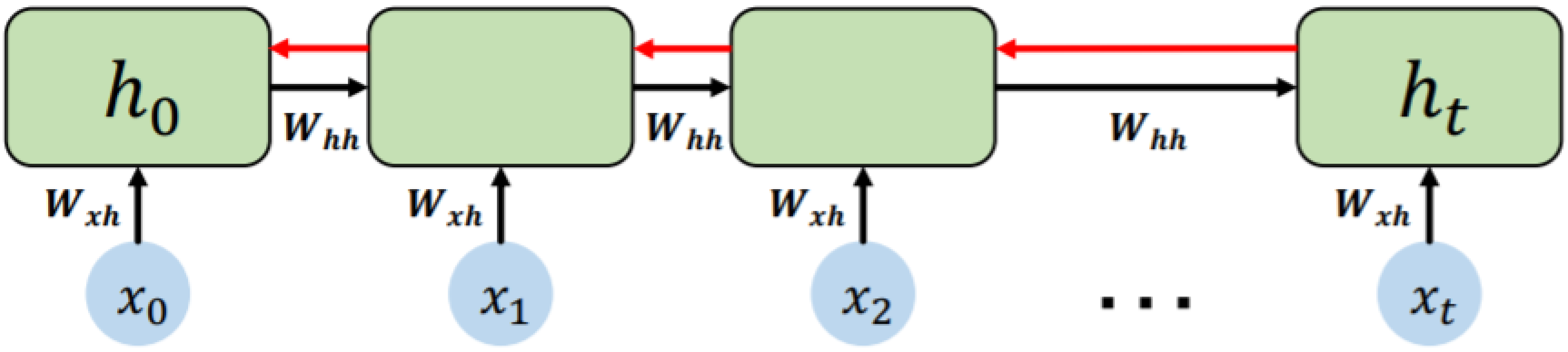
RNNs: computational graph across time



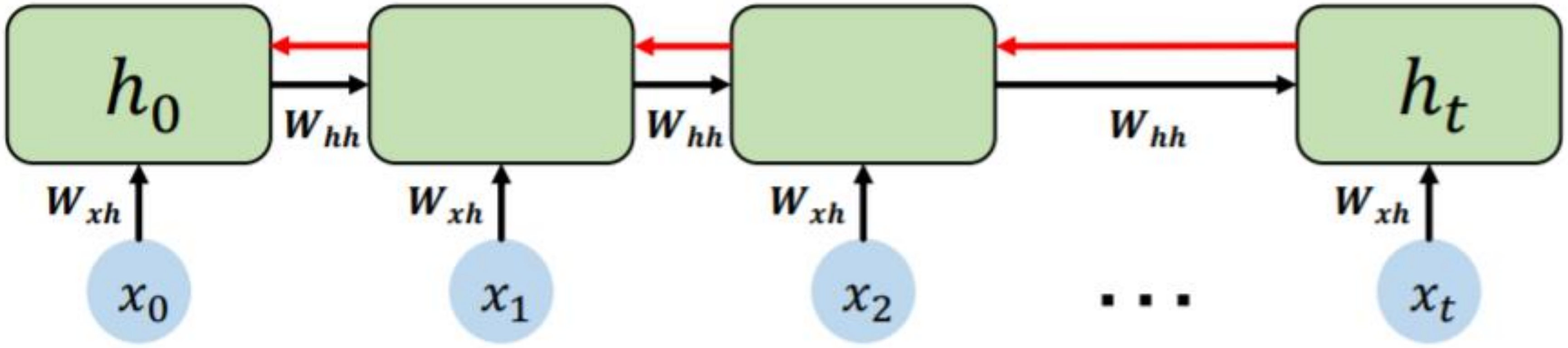
RNNs: backpropagation through time



Standard RNN gradient flow

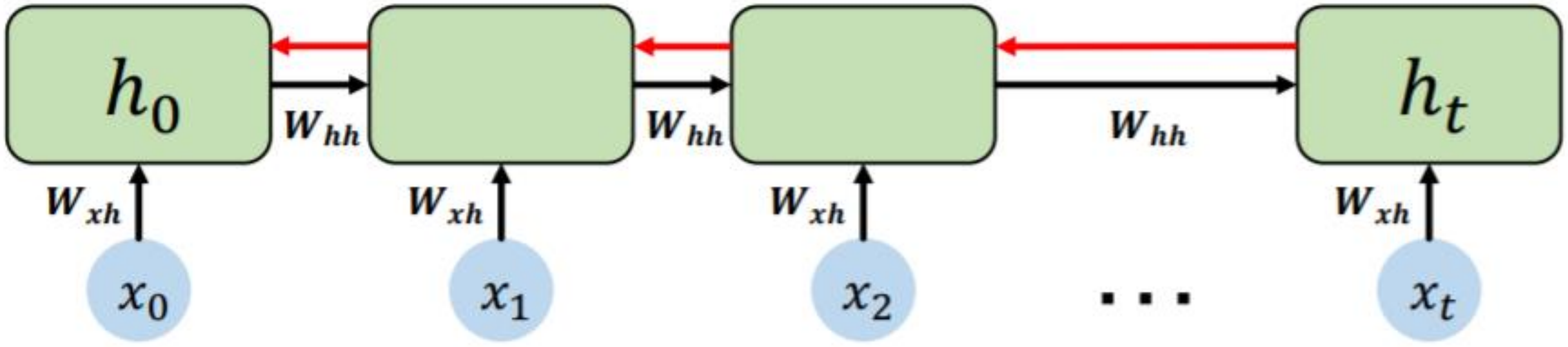


Standard RNN gradient flow



Computing the gradient wrt h_0 involves **many factors of** W_{hh} (and repeated f'!)

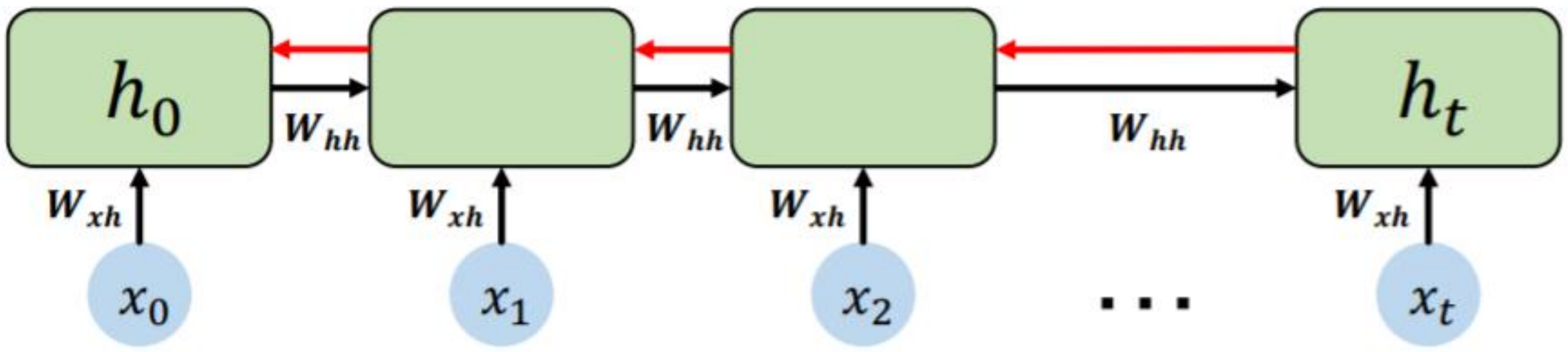
Standard RNN gradient flow: exploding gradients



Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f'!)

Many values > 1:
exploding gradients

Standard RNN gradient flow: exploding gradients

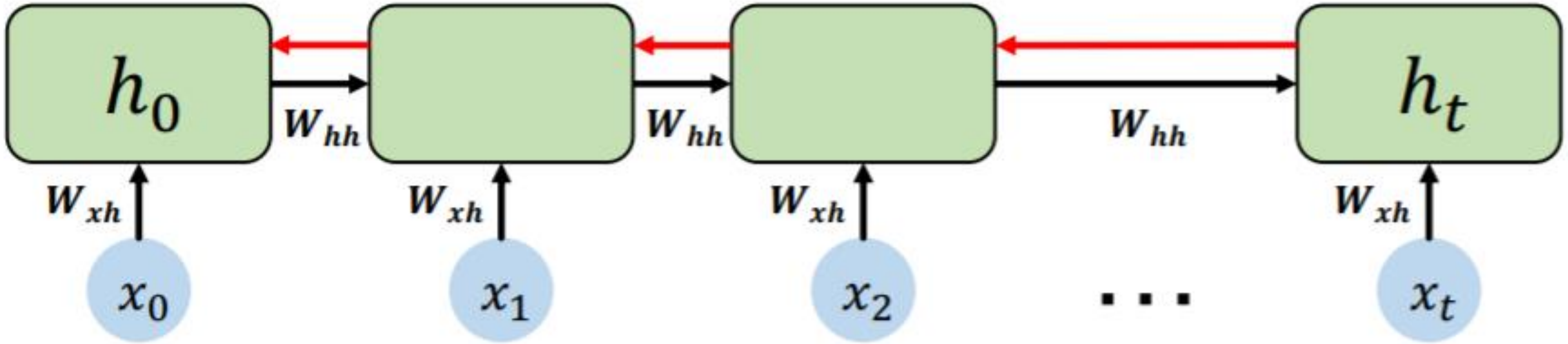


Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f'!)

Many values > 1:
exploding gradients

Gradient clipping to
scale big gradients

Standard RNN gradient flow: vanishing gradients



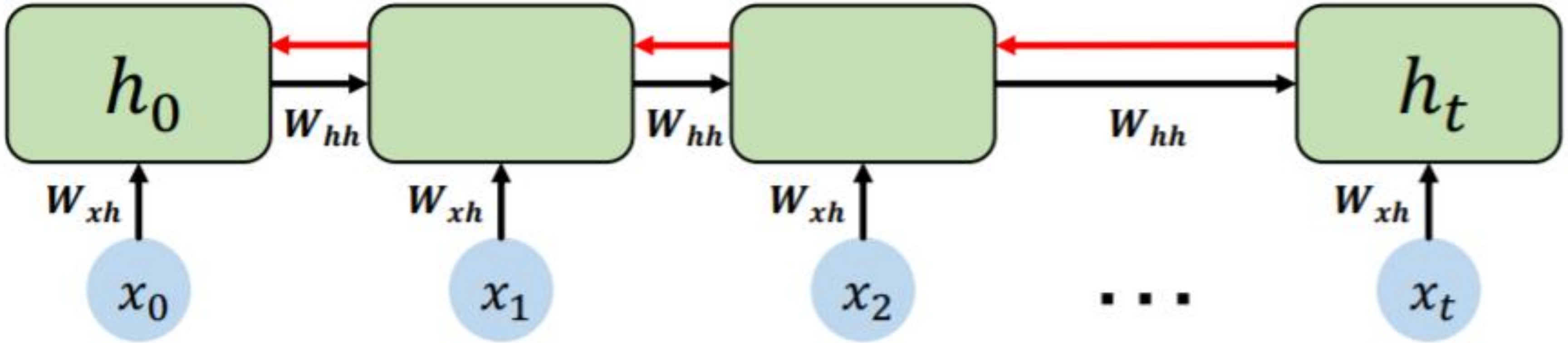
Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f'!)

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} (and repeated f'!)

Largest singular value > 1:
exploding gradients

Gradient clipping to scale big gradients

Largest singular value < 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 network 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 network to capture short-term dependencies

The problem of long-term dependencies

Why are vanishing gradients a problem?

Multiply many small numbers together

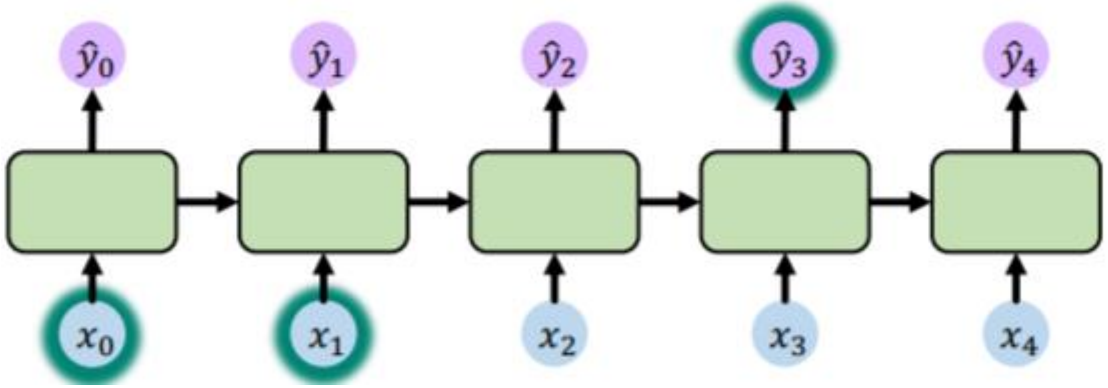
↓

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

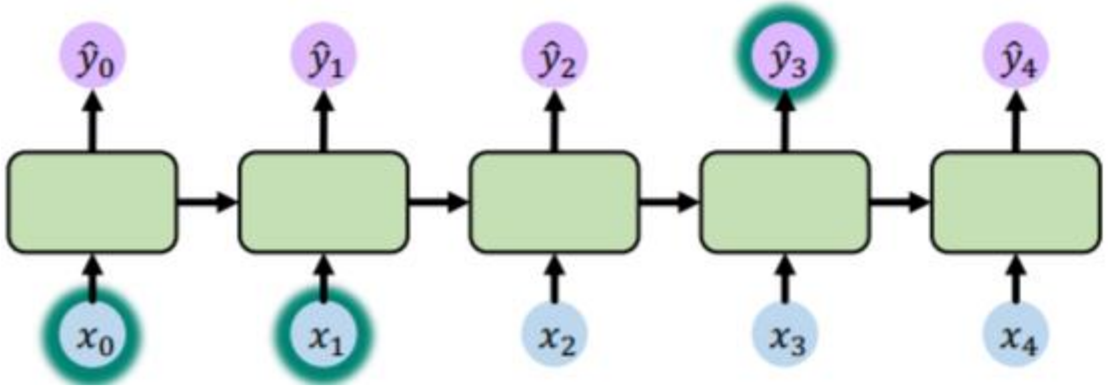
↓

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

↓

Bias parameters to capture short-term dependencies

“The clouds are in the ___”



“I grew up in France,… and I I speak fluent___ ”

The problem of long-term dependencies

Why are vanishing gradients a problem?

Multiply many small numbers together

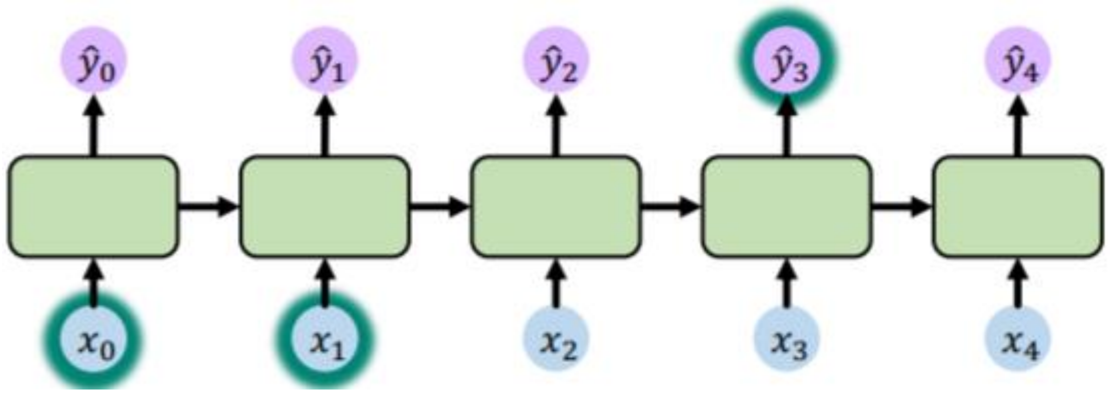
↓

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

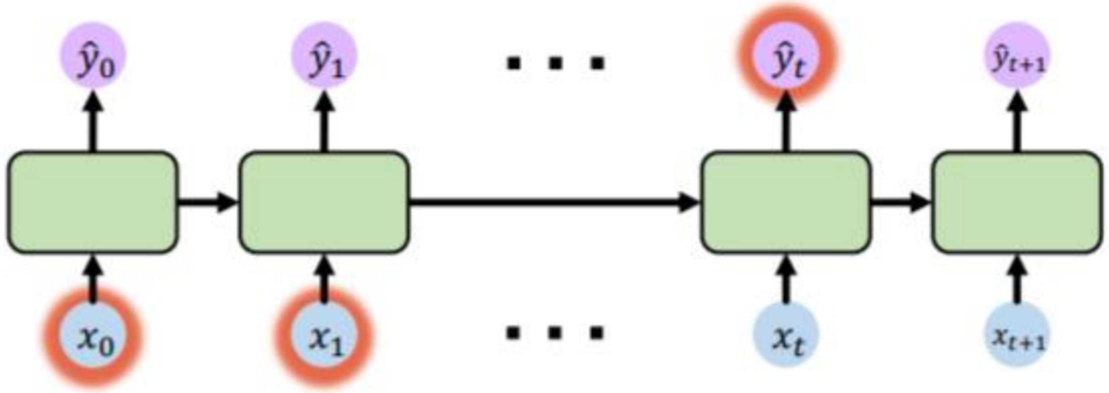
↓

Bias parameters to capture short-term dependencies

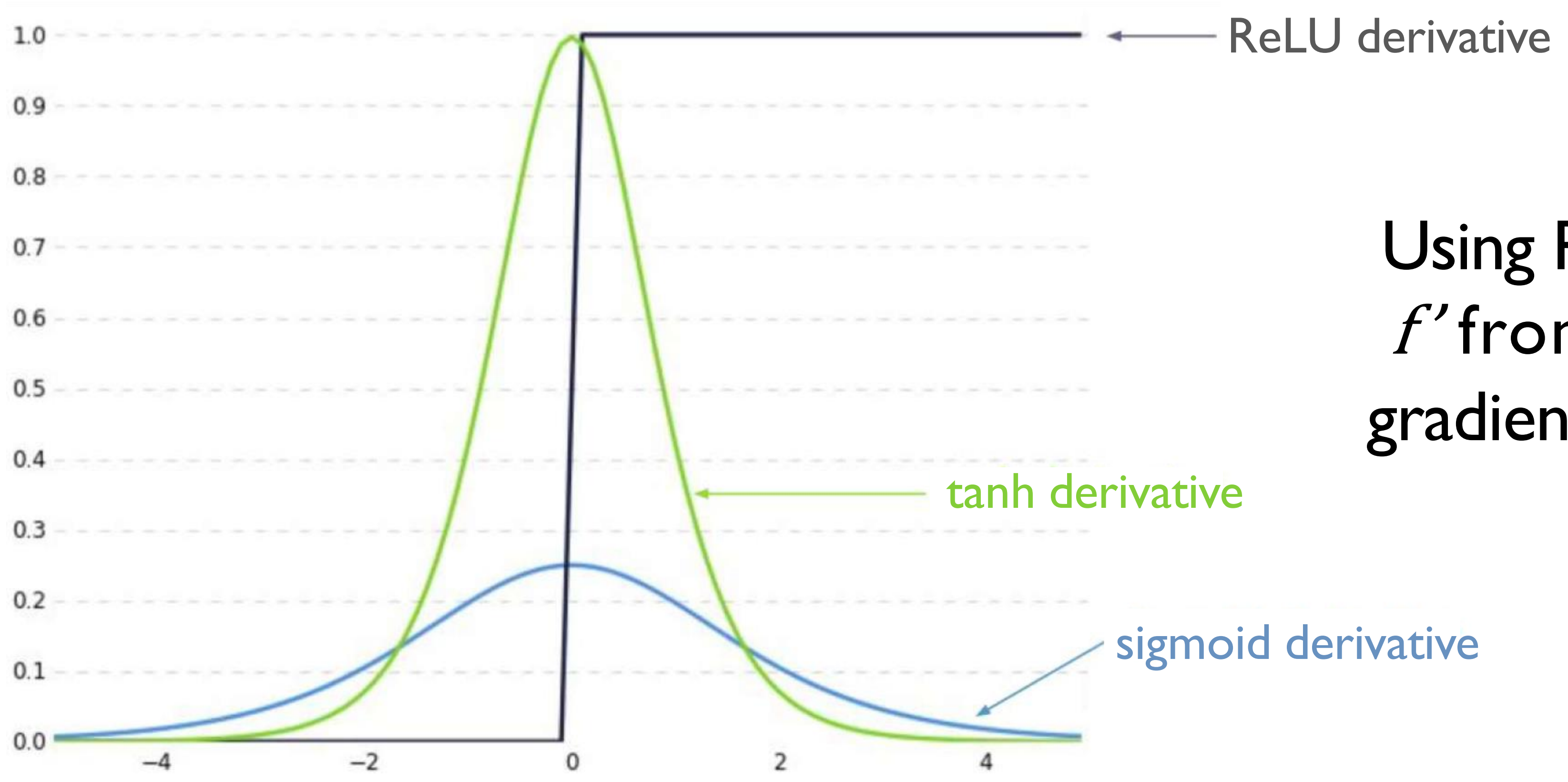
"The clouds are in the ___"



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



Trick #1: activation functions



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

Adapted from H. Suresh, 6.S191 2018

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.

Adapted from H. Suresh, 6.S191 2018

Solution #3: gated cells

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

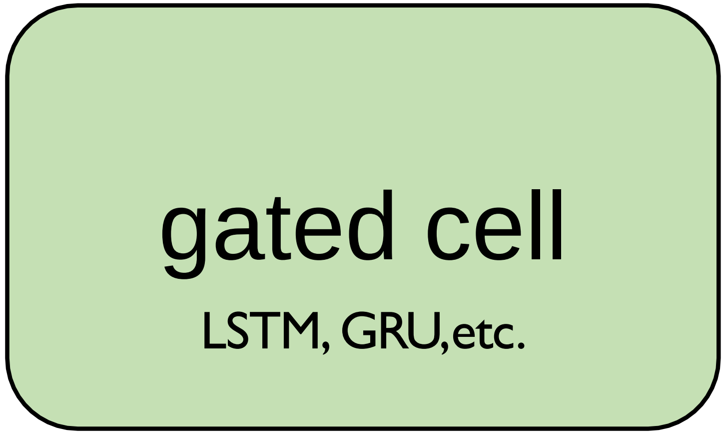
gated cell

LSTM, GRU, etc.

Adapted from H. Suresh, 6.S191 2018

Solution #3: gated cells

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



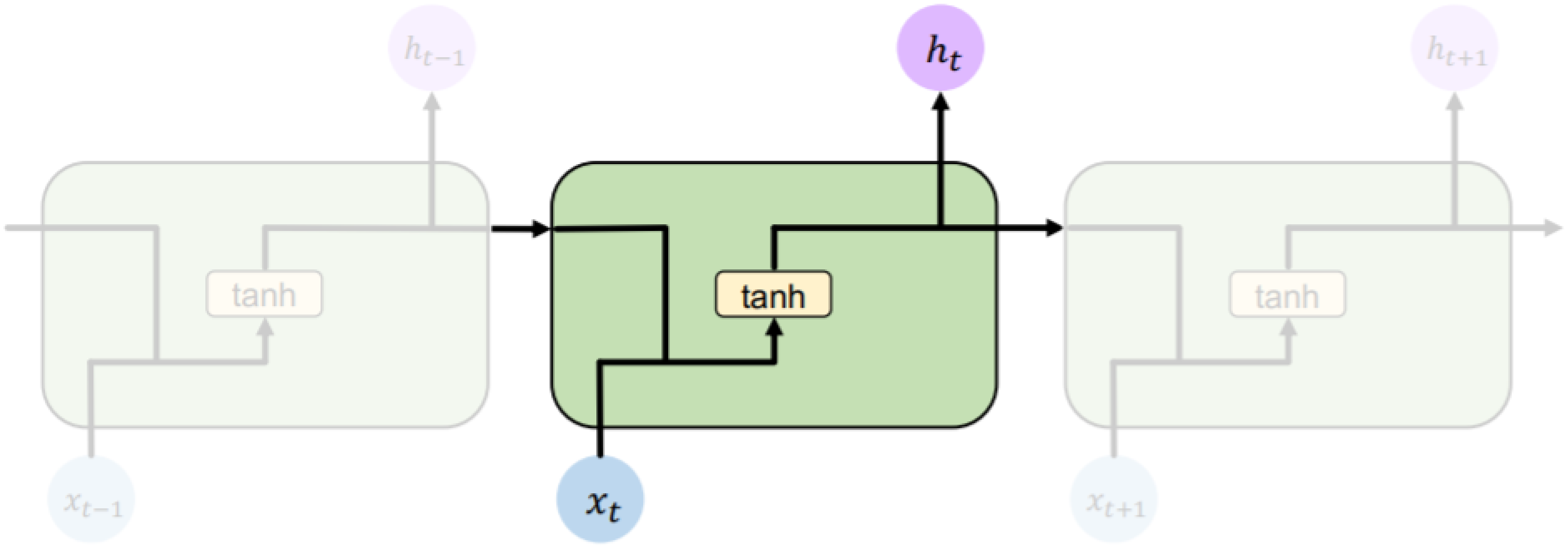
Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

Adapted from H. Suresh, 6.S191 2018

Long Short Term Memory (LSTM) Networks

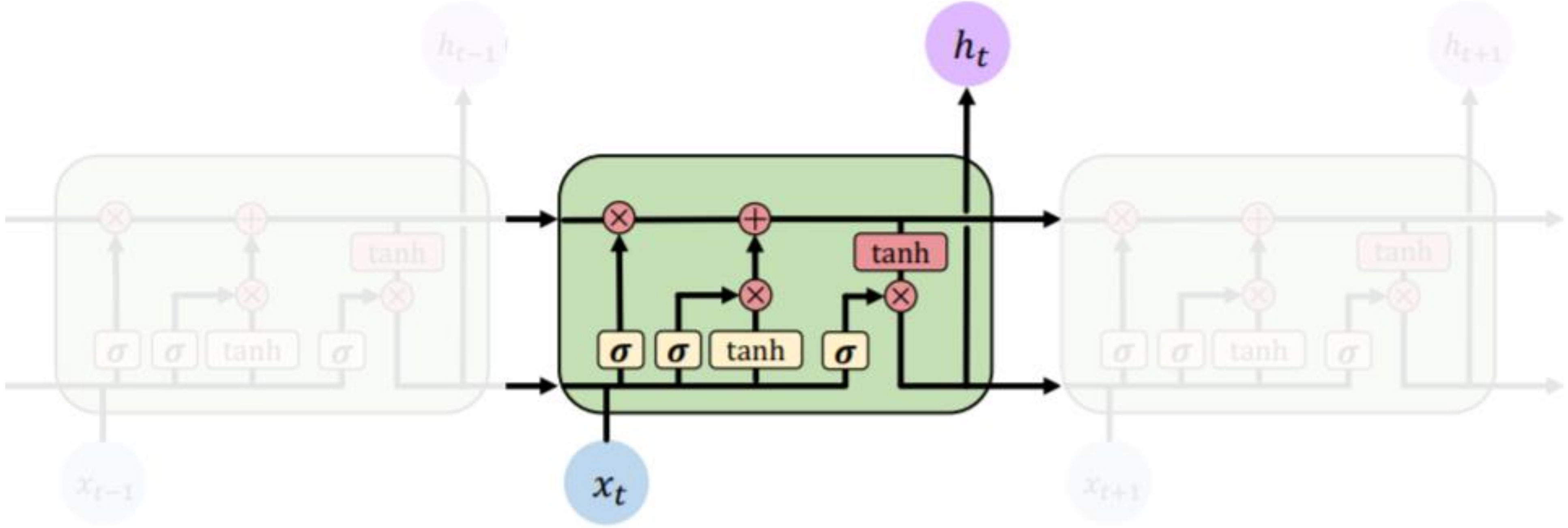
Standard RNN

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



Long Short Term Memory (LSTMs)

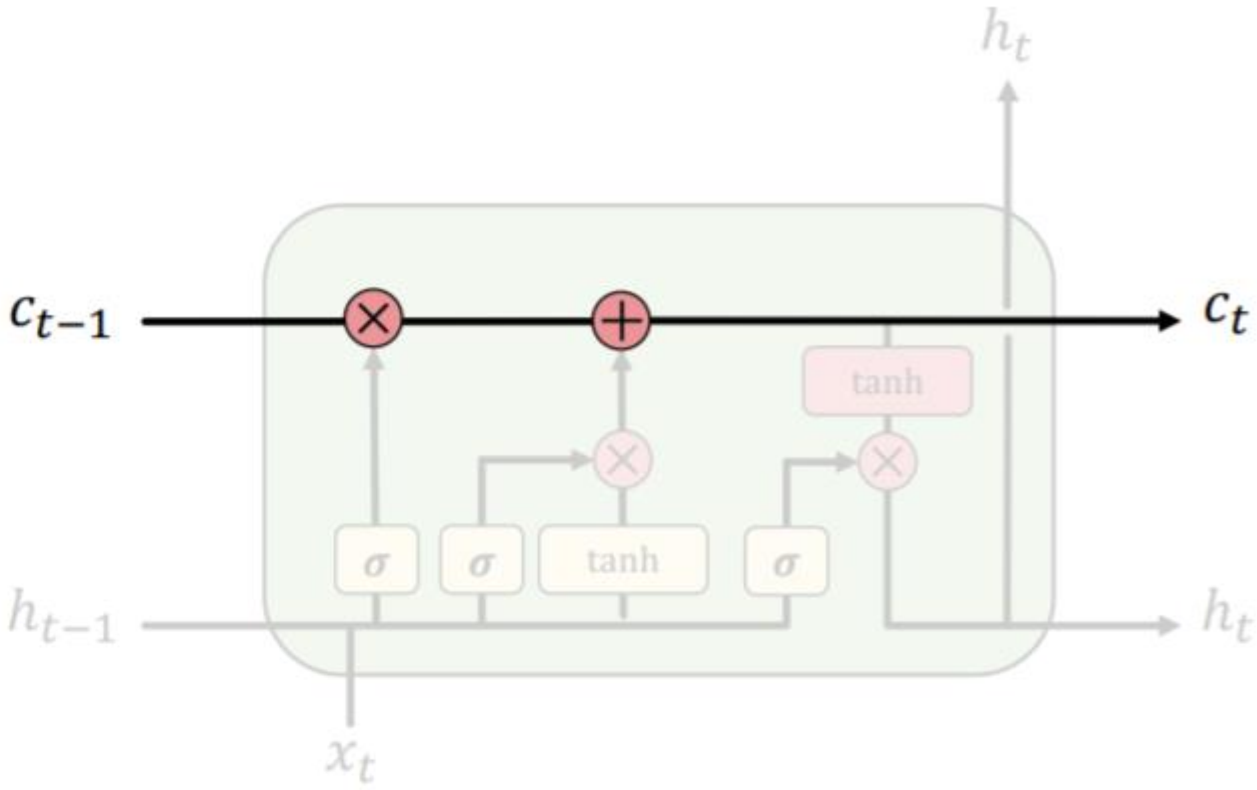
LSTM repeating modules contain interacting layers that control information flow



LSTM cells are able to track information throughout many timesteps

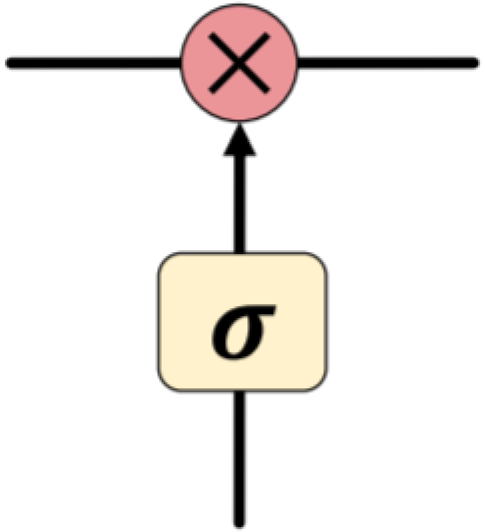
Long Short Term Memory (LSTMs)

LSTMs maintain a cell state c_t where it's easy for information to flow



Long Short Term Memory (LSTMs)

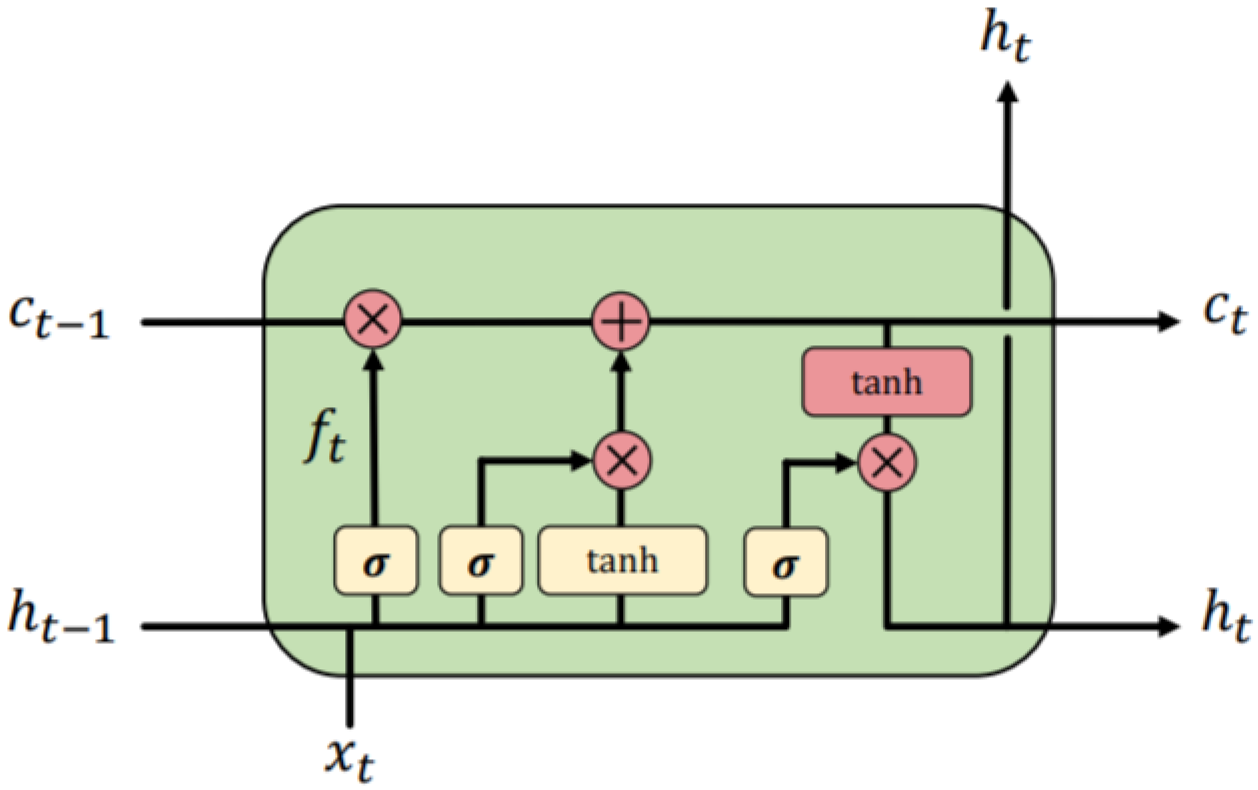
Information is added or removed to cell state through structures called gates



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

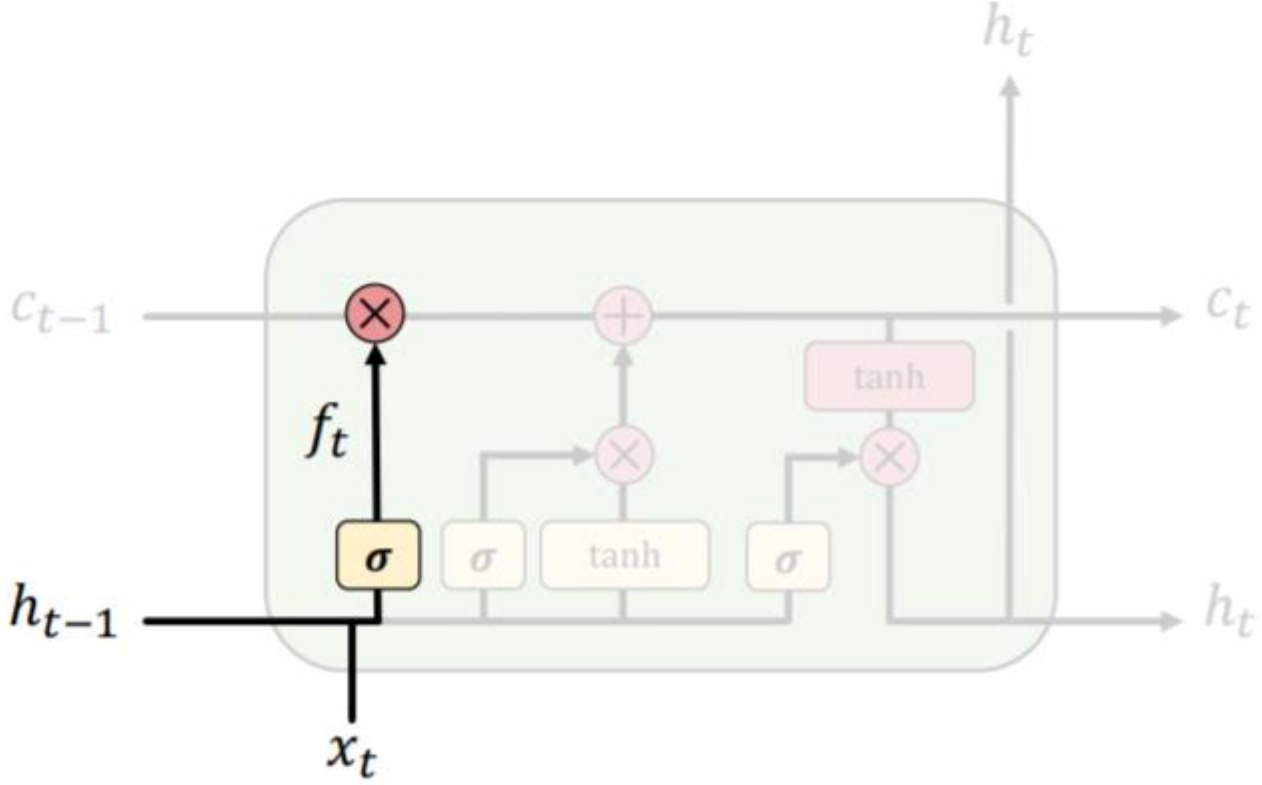
Long Short Term Memory (LSTMs)

How do LSTMs work?



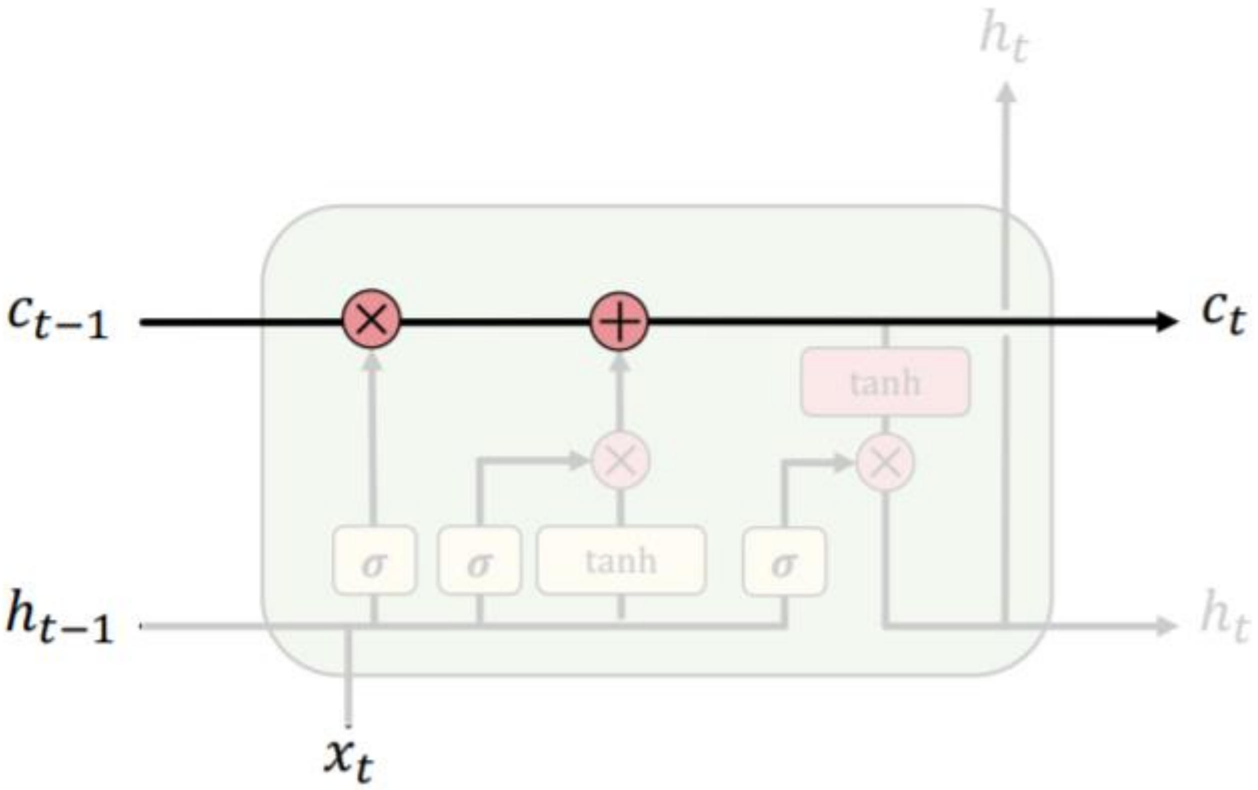
Long Short Term Memory (LSTMs)

LSTMs forget irrelevant parts of the previous state



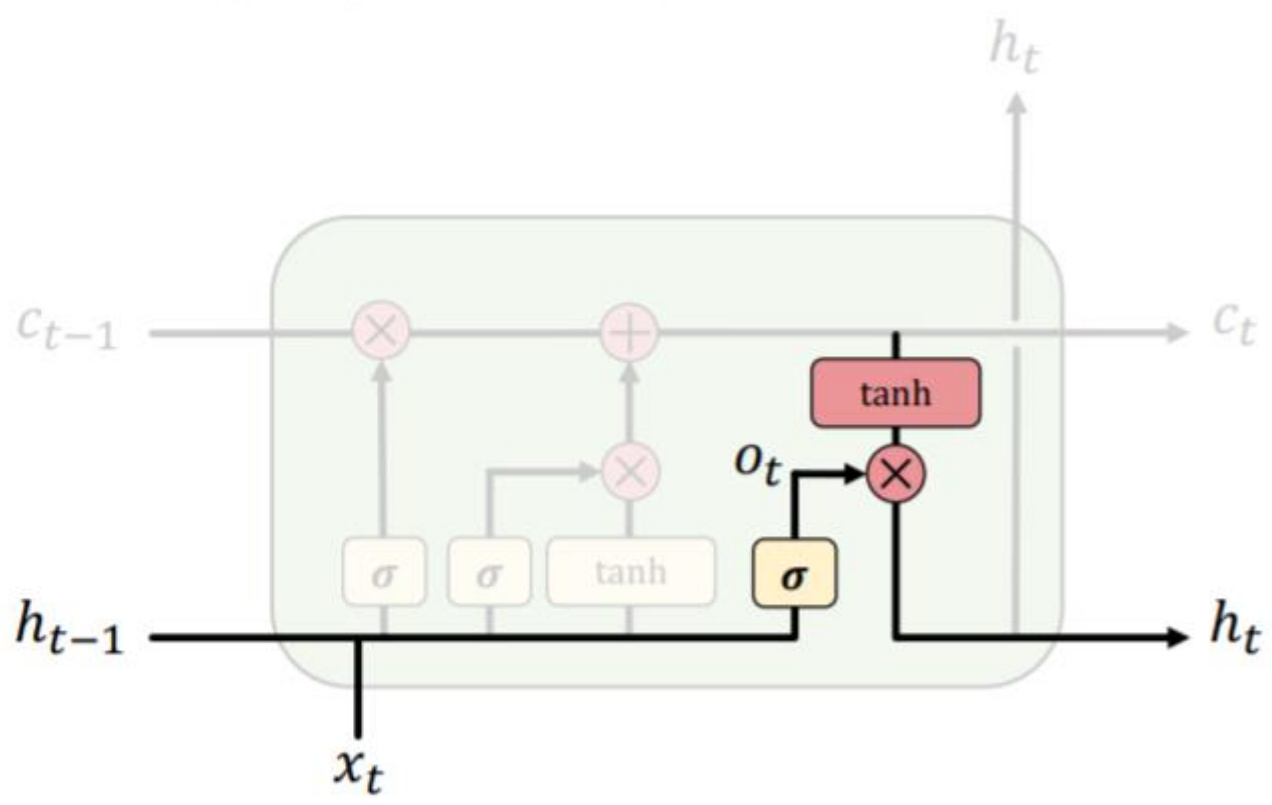
Long Short Term Memory (LSTMs)

LSTMs selectively update cell state values



Long Short Term Memory (LSTMs)

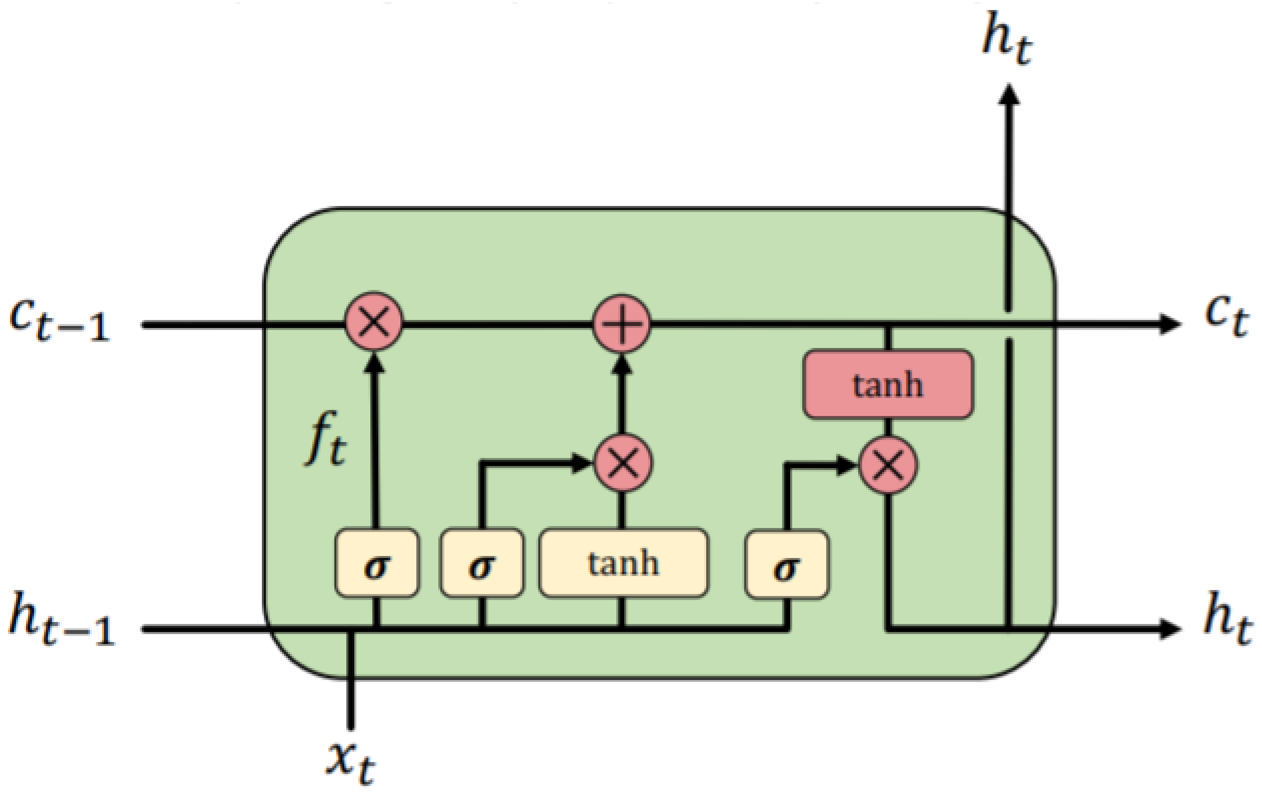
LSTMs use an output gate to output certain parts of the cell state



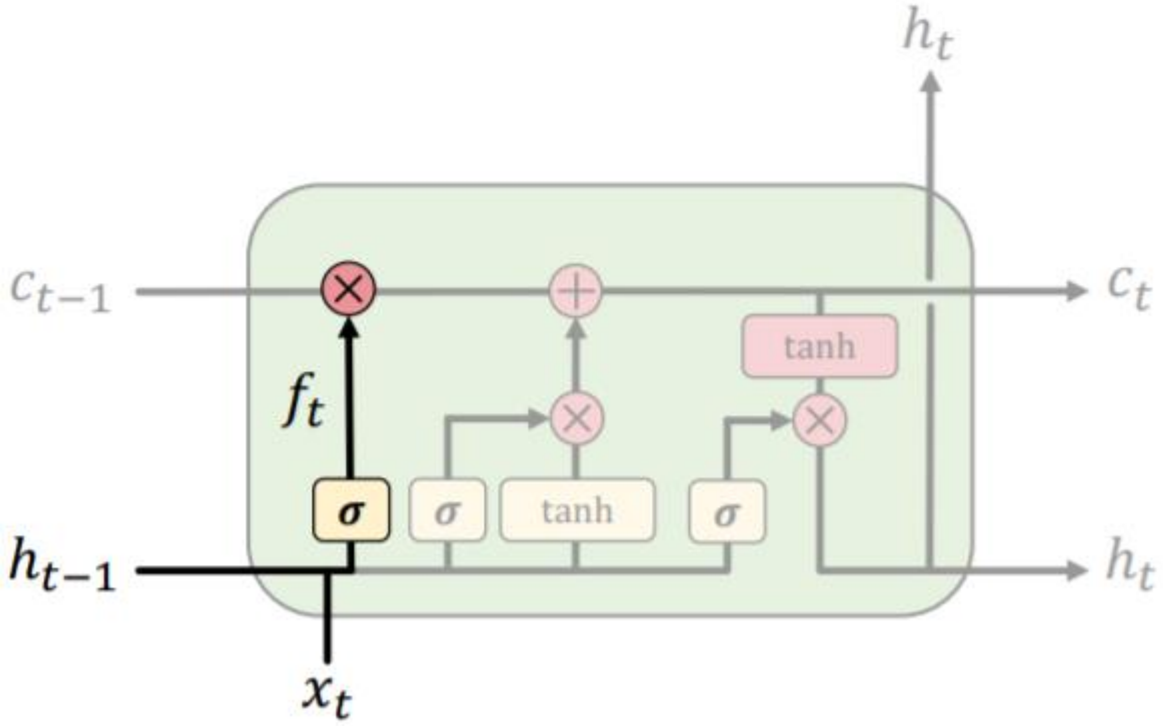
Long Short Term Memory (LSTMs)

How do LSTMs work?

1) Forget 2) Update 3) Output



LSTMs: forget irrelevant information

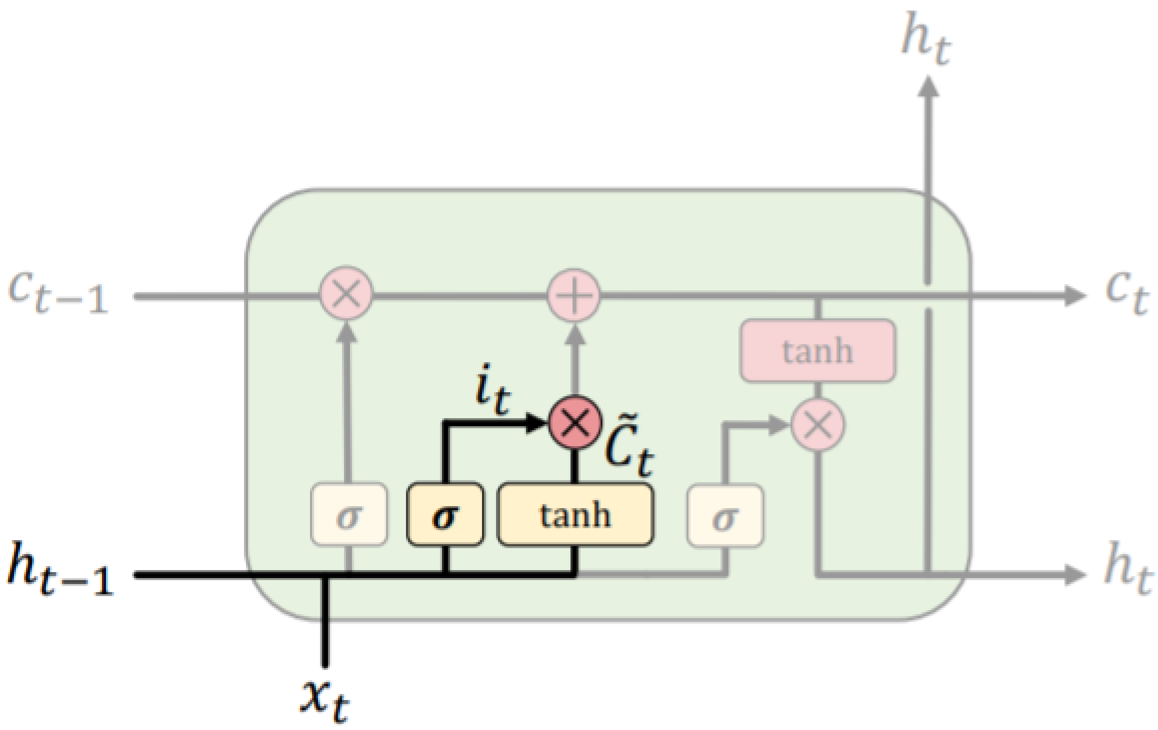


$$f_t = \sigma(\boldsymbol{W_i}[\, h_{t-1}, x_t \,] + b_f)$$

- Use previous cell output and input
- Sigmoid: value 0 and 1 – "completely forget" vs. "completely keep"

ex: Forget the gender pronoun of previous subject in sentence.

LSTMs: identify new information to be stored



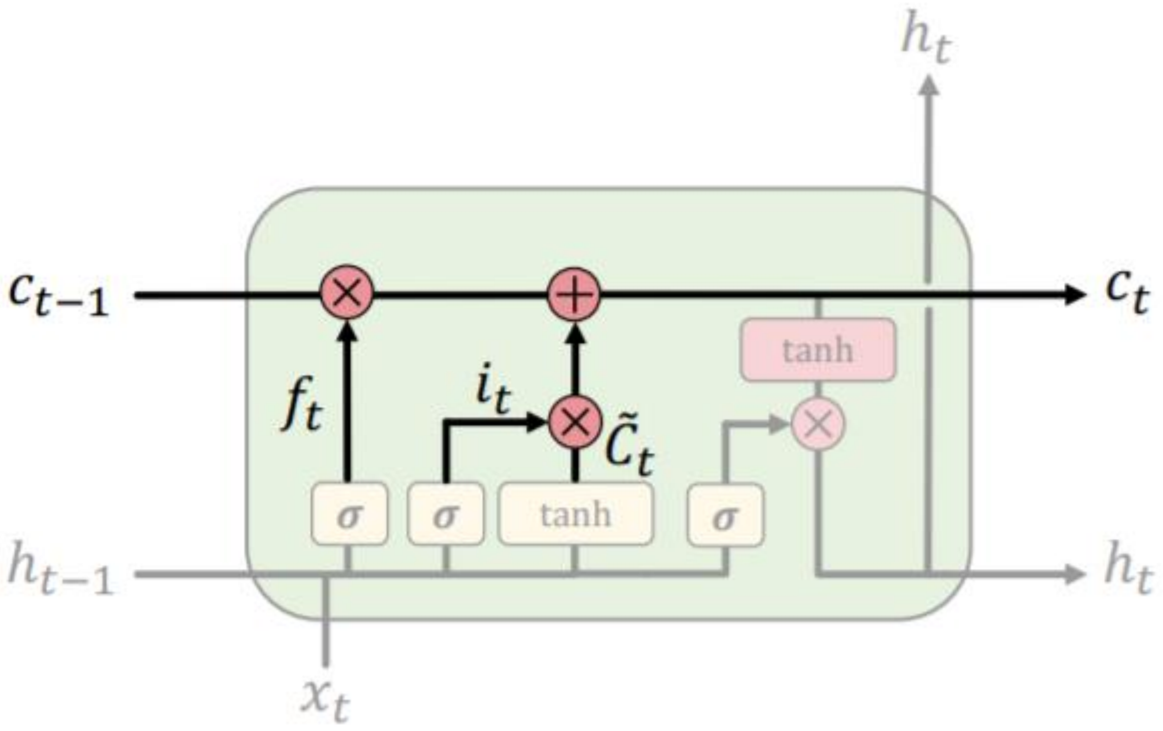
$$i_t = \sigma(\boldsymbol{W_i}[\, h_{t-1}, x_t \,] + b_i)$$

$$\tilde{C}_t = \tanh(\boldsymbol{W_C}[\, h_{t-1}, x_t \,] + b_C)$$

- Sigmoid layer: decide what values to update
- Tanh layer: generate new vector of "candidate values" that could be added to the state

ex: Add gender of new subject to replace that of old subject.

LSTMs: update cell state

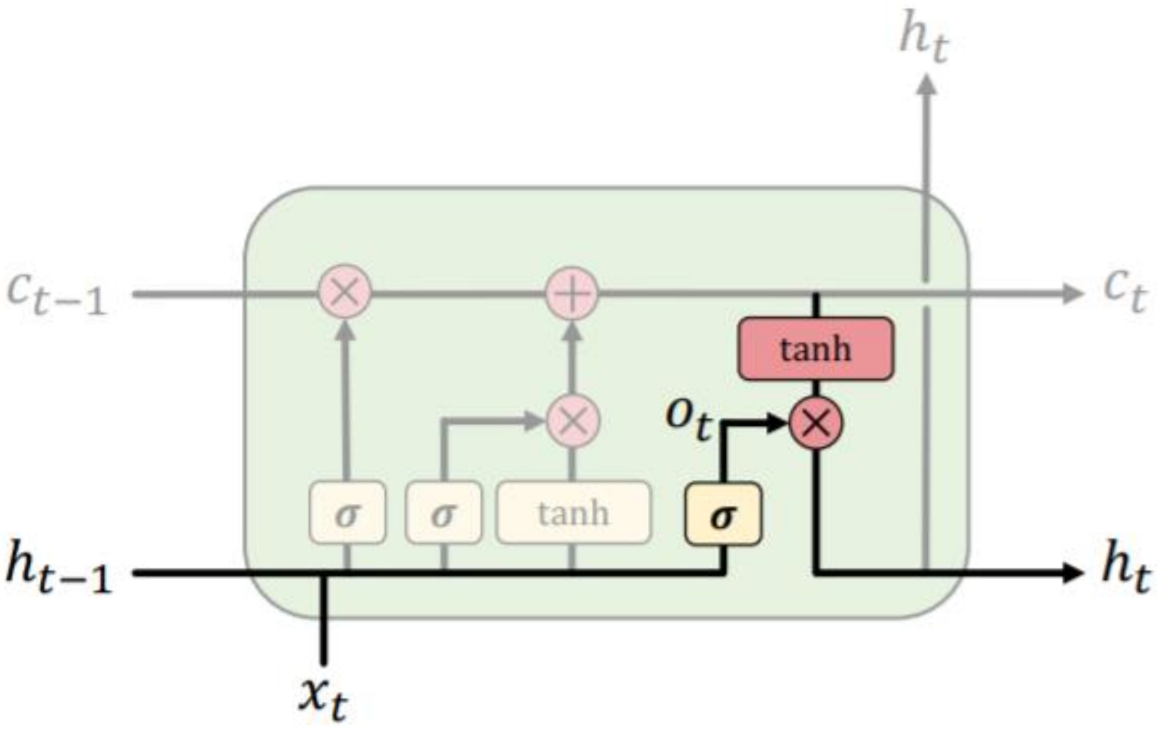


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Apply forget operation to previous internal cell state: $f_t * C_{t-1}$
- Add new candidate values, scaled by how much we decided to update: $i_t * \tilde{C}_t$

ex: Actually drop old information and add new information about subject's gender.

LSTMs: output filtered version of cell state



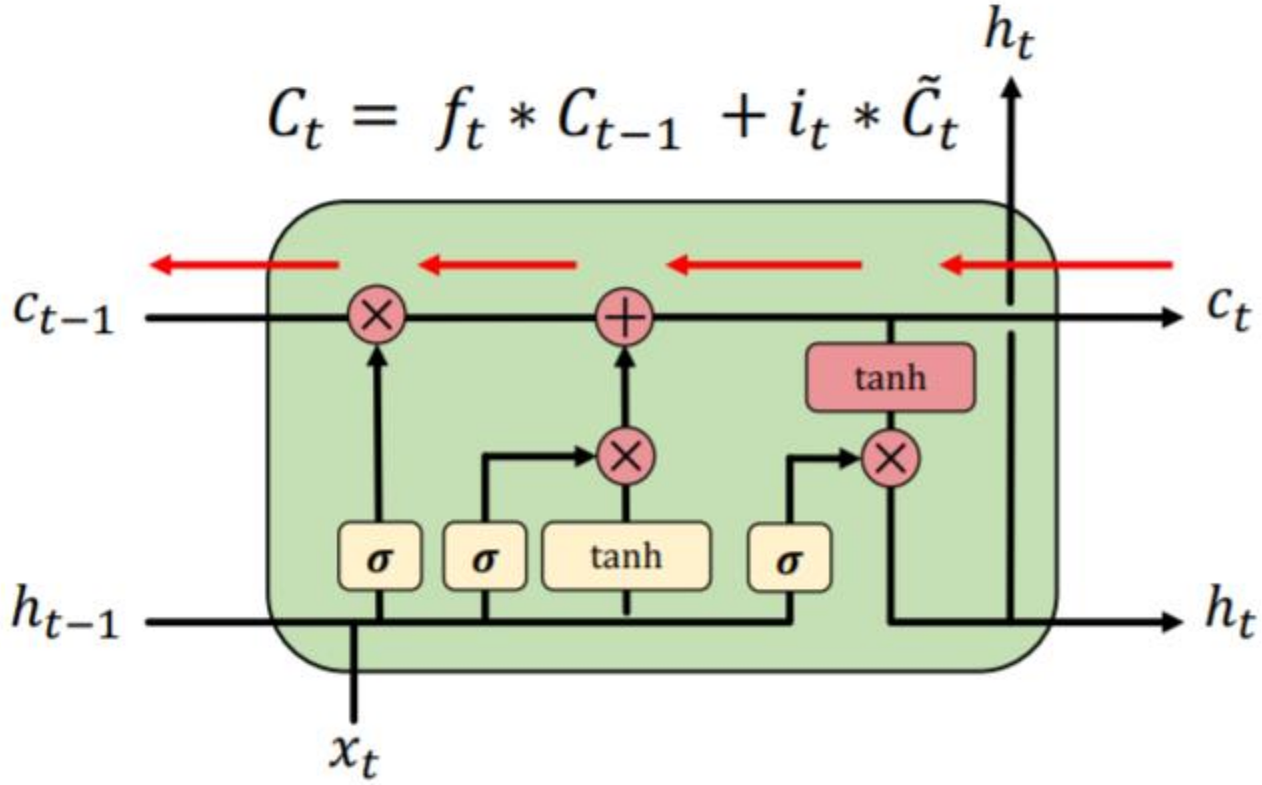
$$o_t = \sigma(W_o[\, h_{t-1}, x_t \,] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

- Sigmoid layer: decide what parts of state to output
- Tanh layer: squash values between -1 and 1
- $o_t * \tanh(C_t)$: output filtered version of cell state

ex: Having seen a subject, may output information relating to a verb.

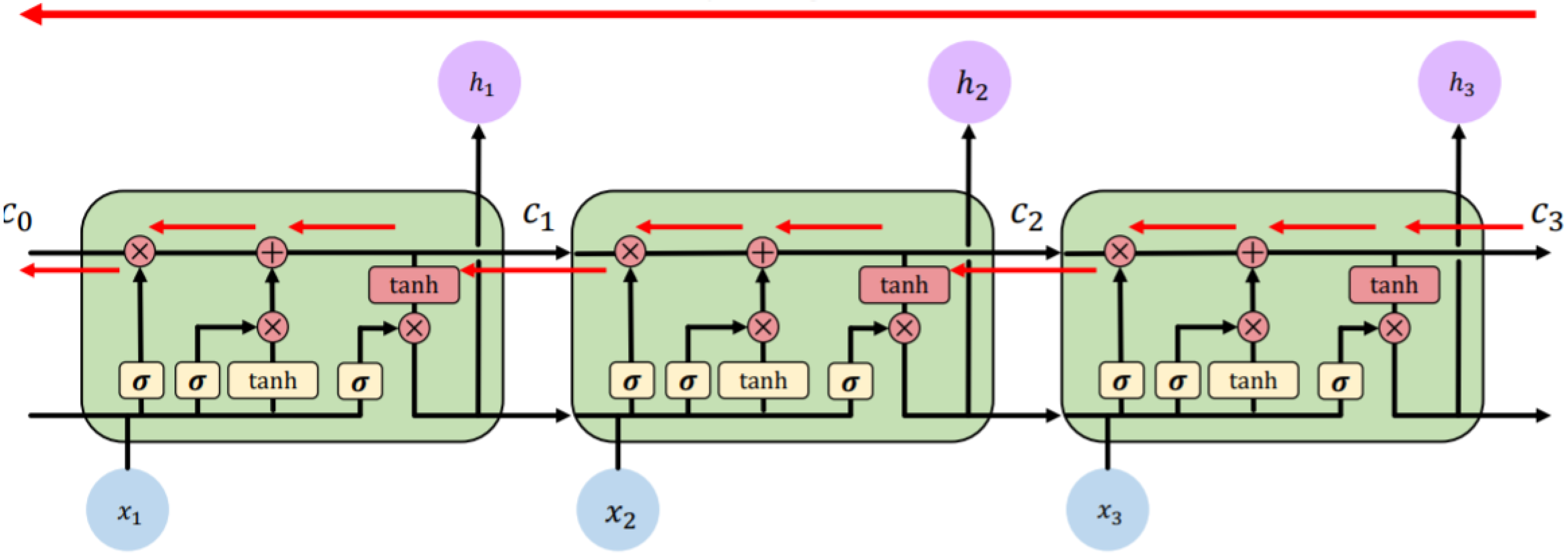
LSTM gradient flow

Backpropagation from C_t to C_{t-1} requires only elementwise multiplication!
No matrix multiplication → avoid vanishing gradient problem.



LSTM gradient flow

Uninterrupted gradient flow!

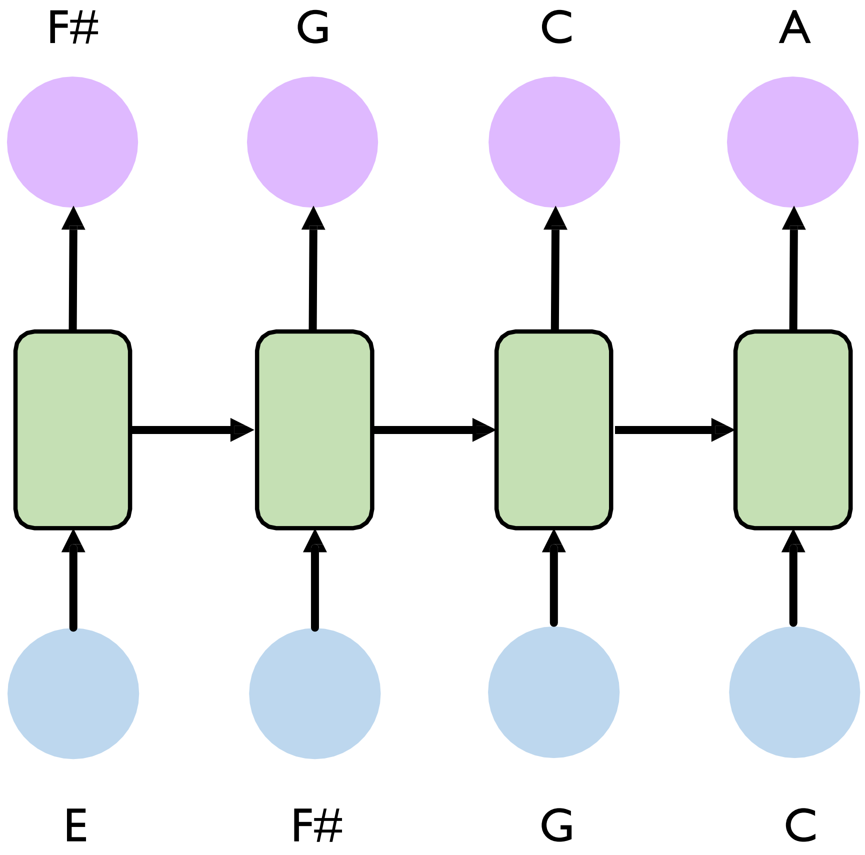


LSTMs: key concepts

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
   - Selectively update cell state
   - Output gate returns a filtered version of the cell state
3. Backpropagation from C_t to C_{t-1} doesn't require matrix multiplication: uninterrupted gradient flow

RNN Applications

Example task: music generation



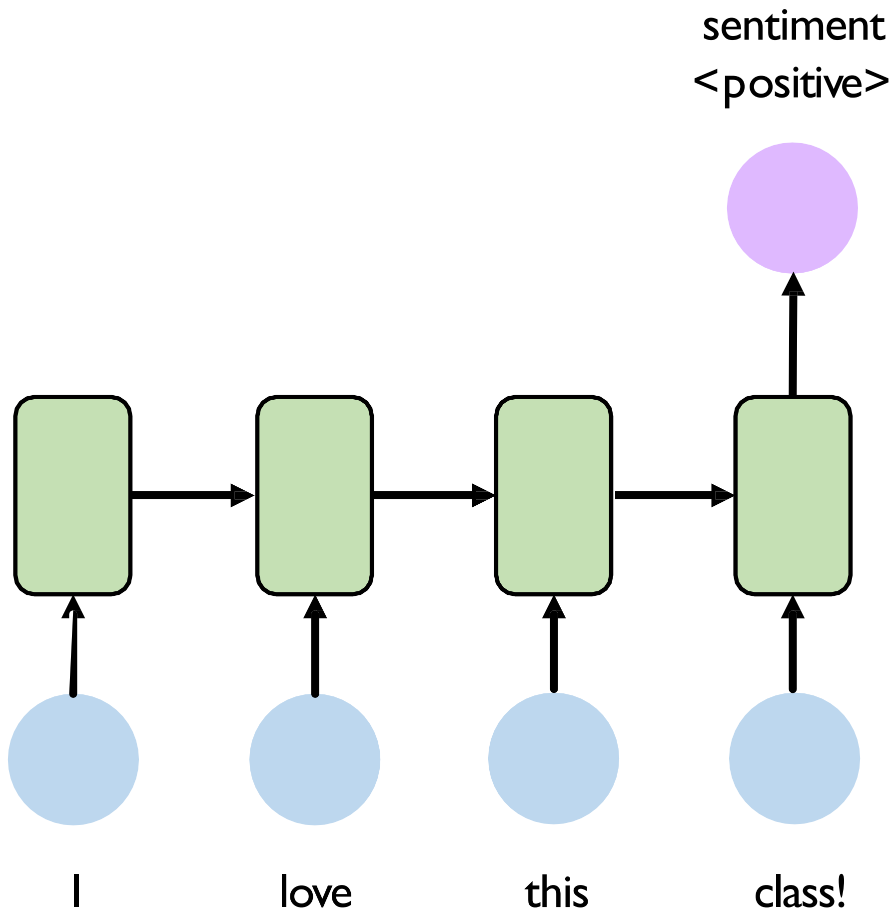
Input: sheet music

Output: next character in sheet music

6.S191 Lab!

Adapted from H. Suresh, 6.S191 2018

Example task: sentiment classification



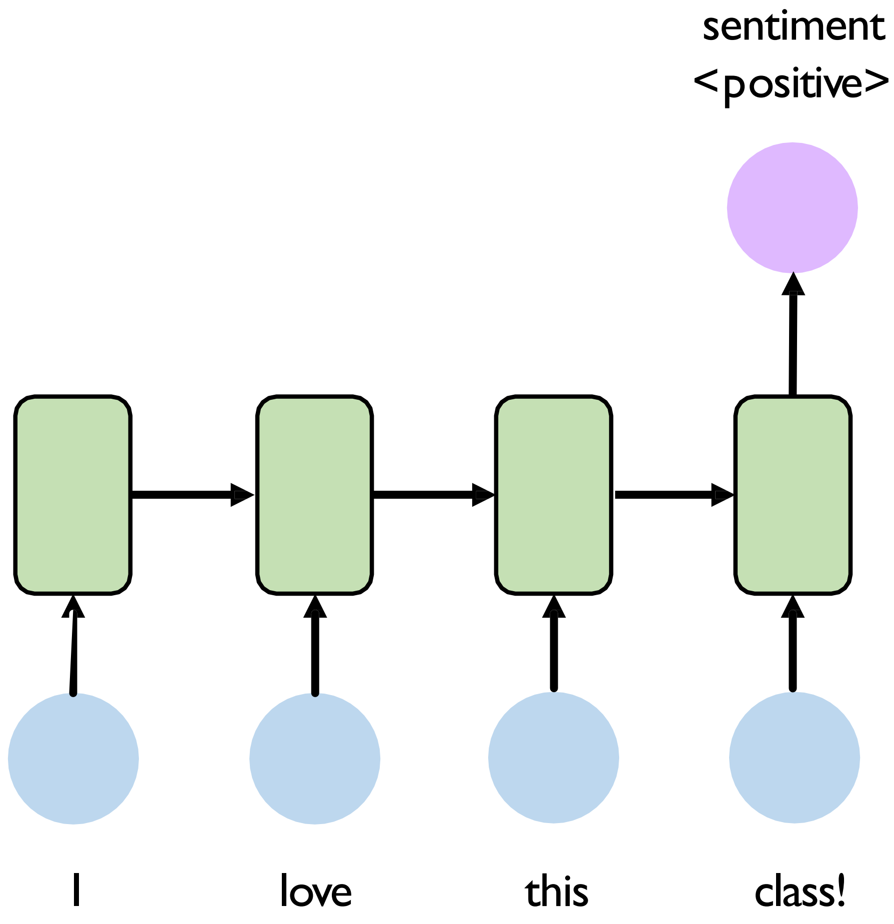
Input: sequence of words

Output: probability of having positive sentiment

```
loss = tf.nn.softmax_cross_entropy_with_logits(
    labels=model.y, logits=model.pred
)
```

Adapted from H. Suresh, 6.S191 2018

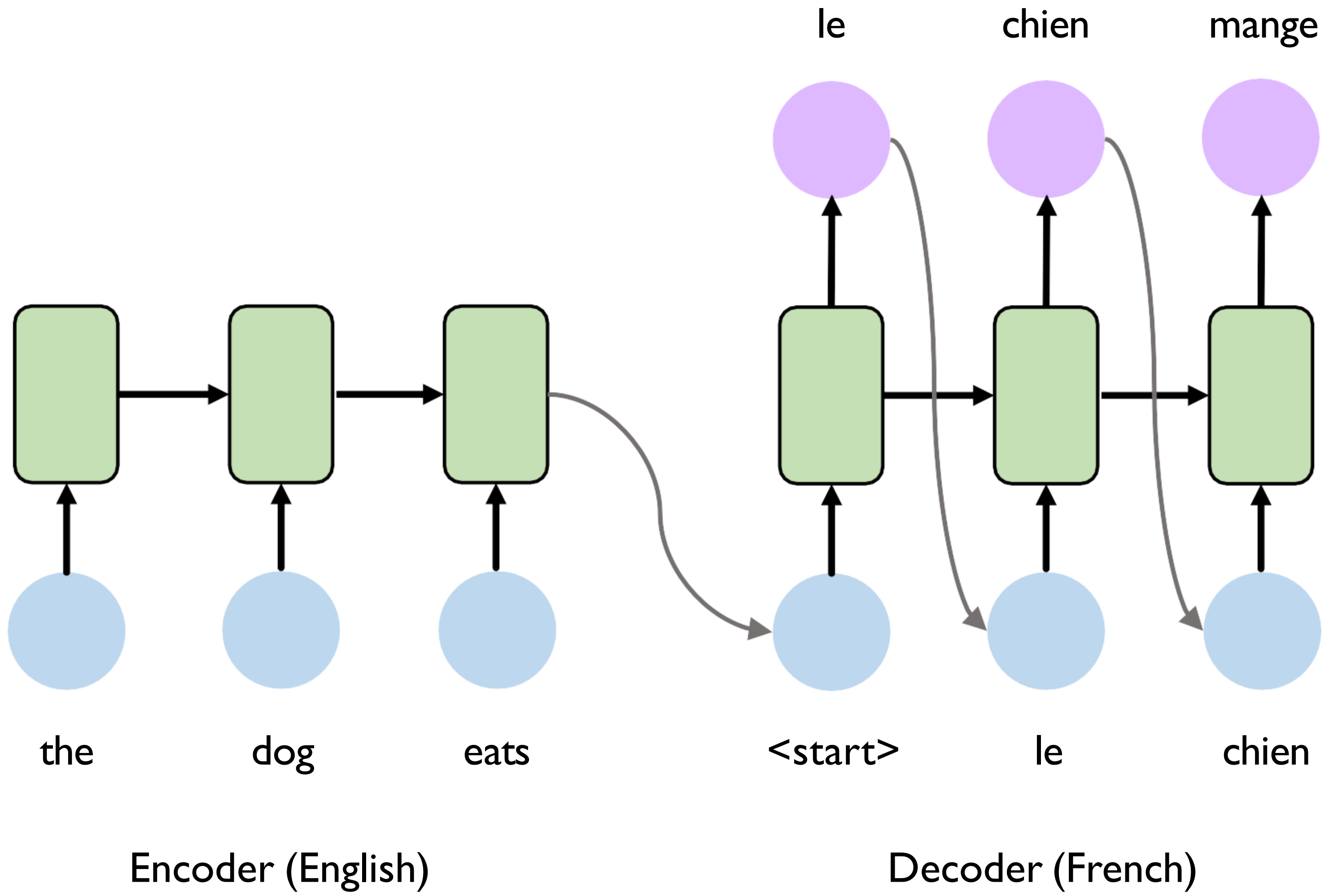
Example task: sentiment classification



Tweet sentiment classification

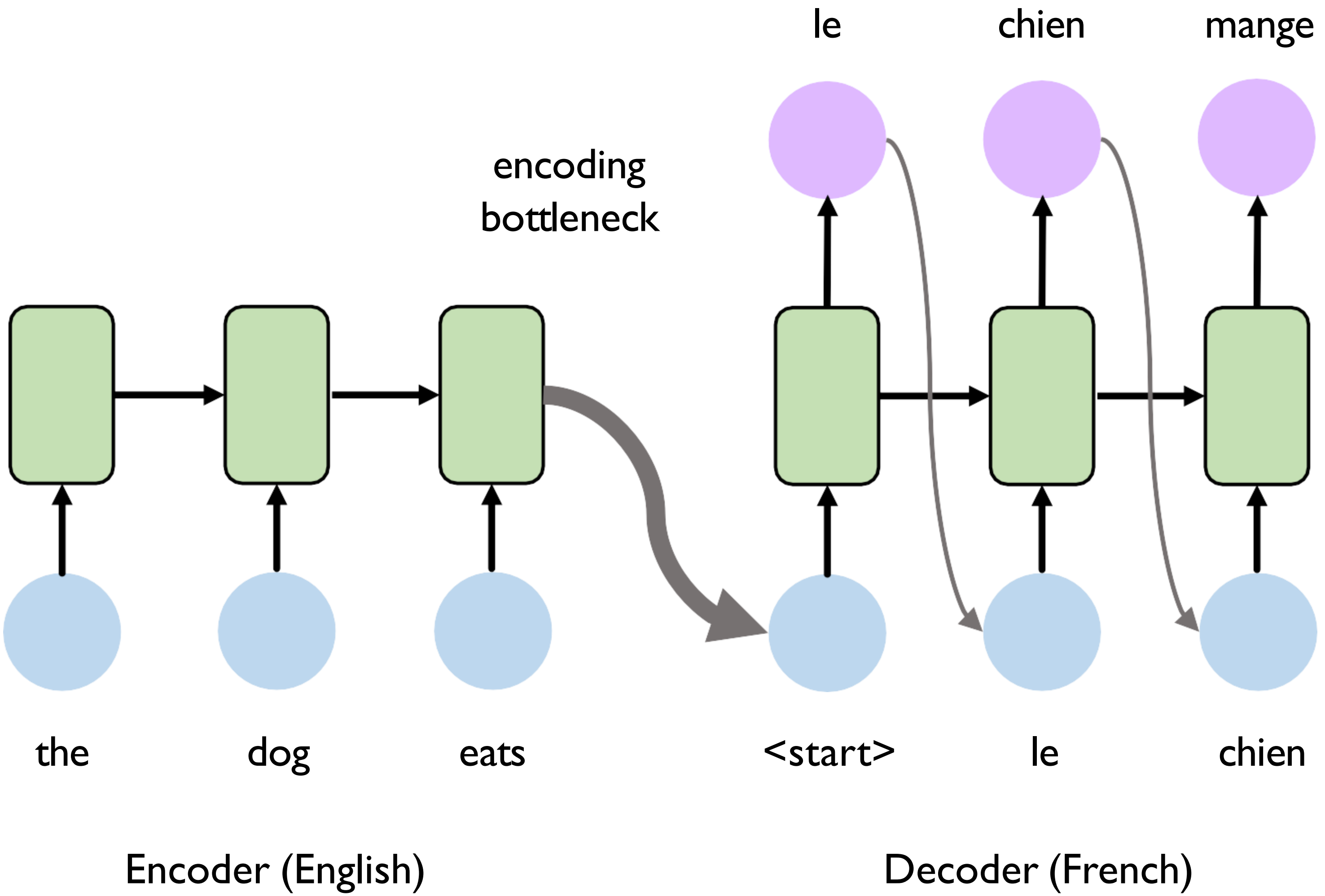
Adapted from H. Suresh, 6.S191 2018

Example task: machine translation



Adapted from H. Suresh, 6.S191 2018

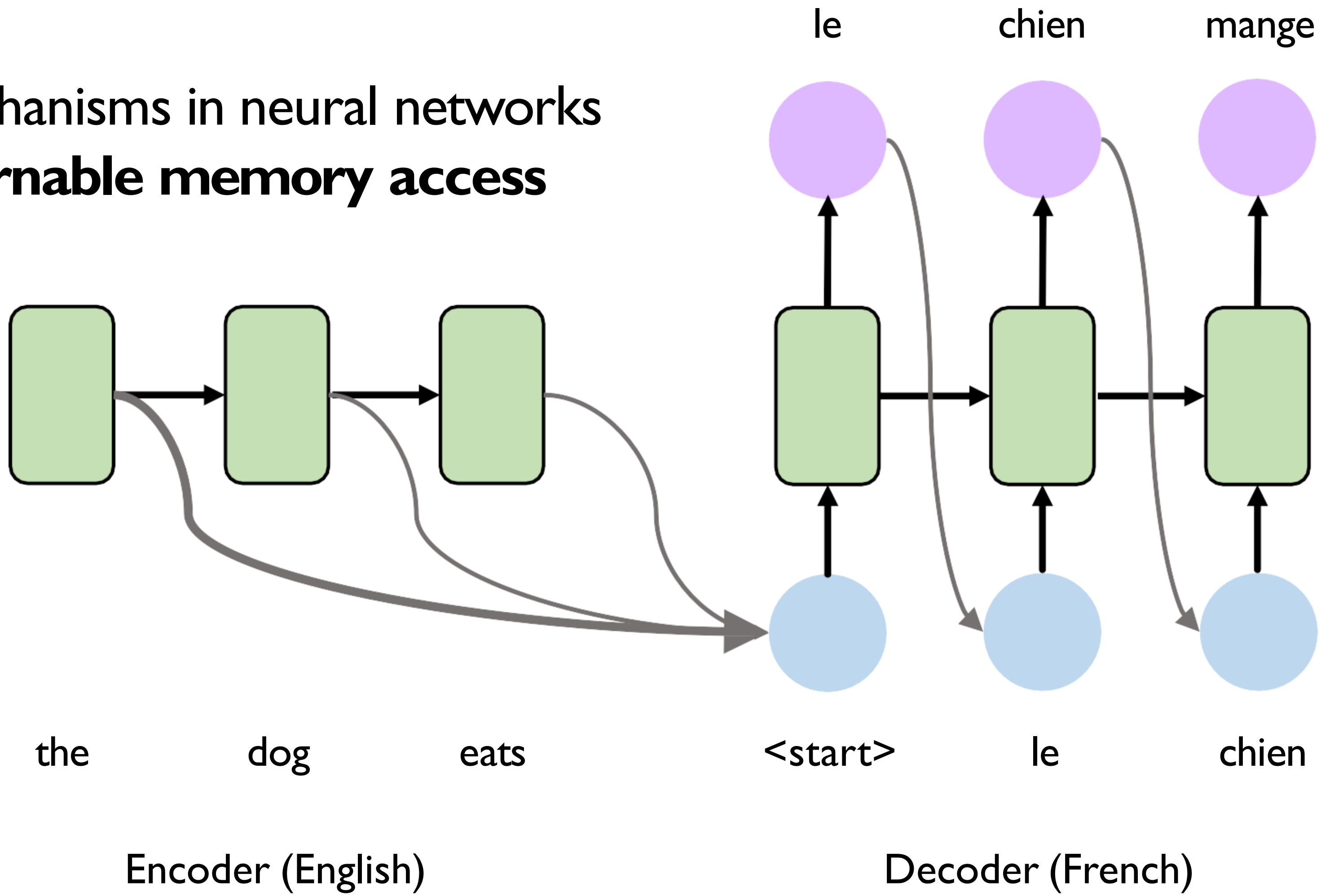
Example task: machine translation



Adapted from H. Suresh, 6.S191 2018

Attention mechanisms

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



Adapted from H. Suresh, 6.S191 2018

Gated Recurrent Unit (GRU)

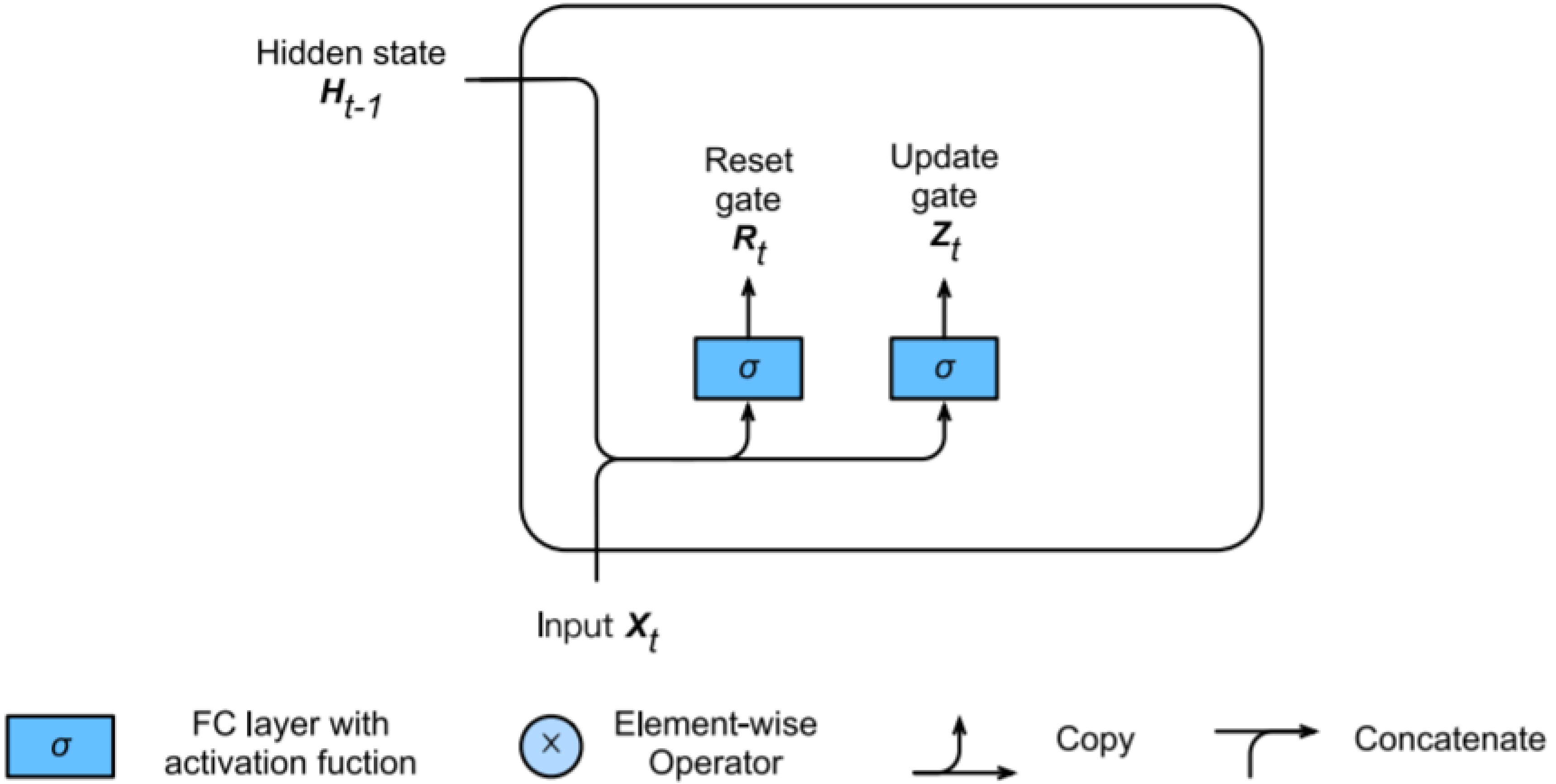
Paying attention to a sequence

- Not all observations are equally relevant

- Only remember the relevant ones
  - Need mechanism to **pay attention (update gate)**
  - Need mechanism to **forget (reset gate)**

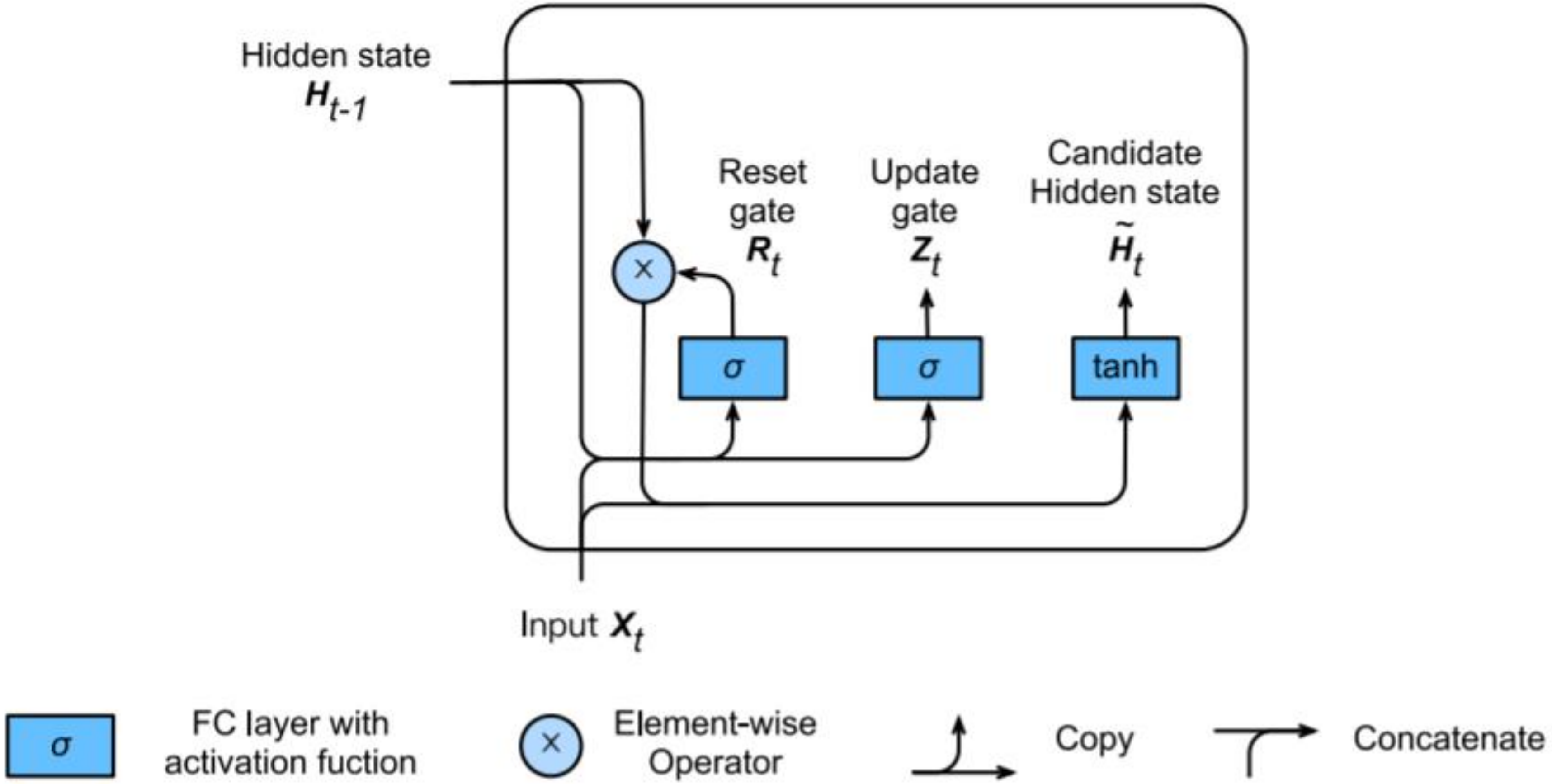
Gating

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$



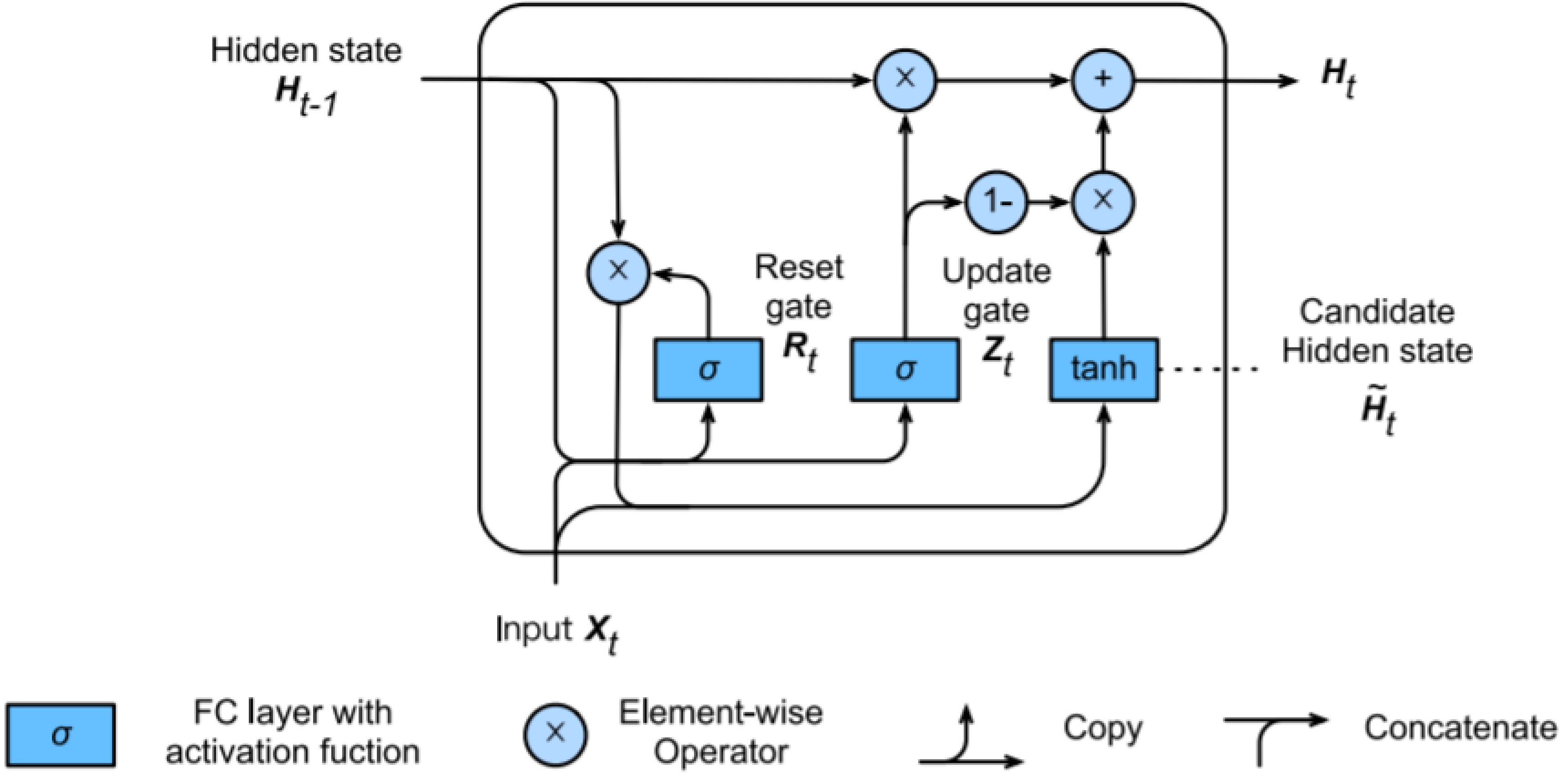
Candidate Hidden State

$$\tilde{H}_t = \tanh(X_t W_{xh} + \left(R_t \odot H_{t-1}\right) W_{hh} + b_h)$$



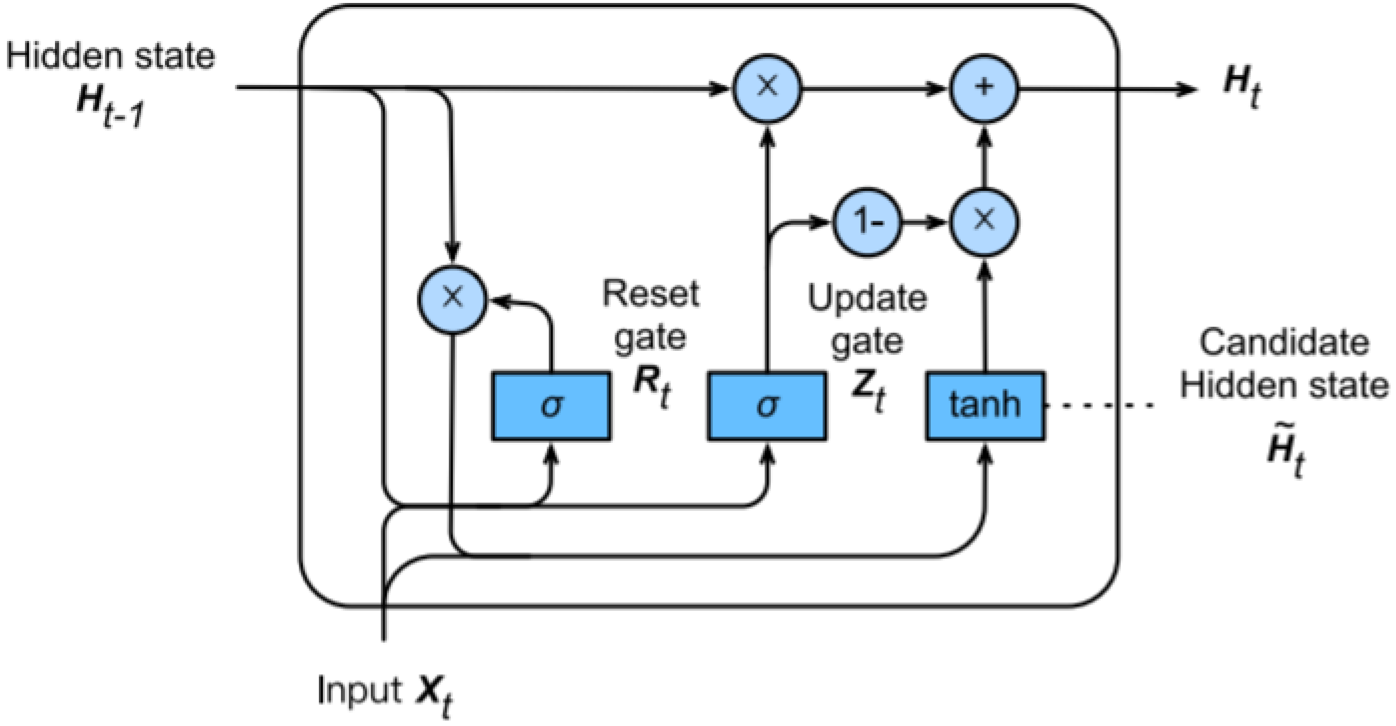
Hidden State

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$

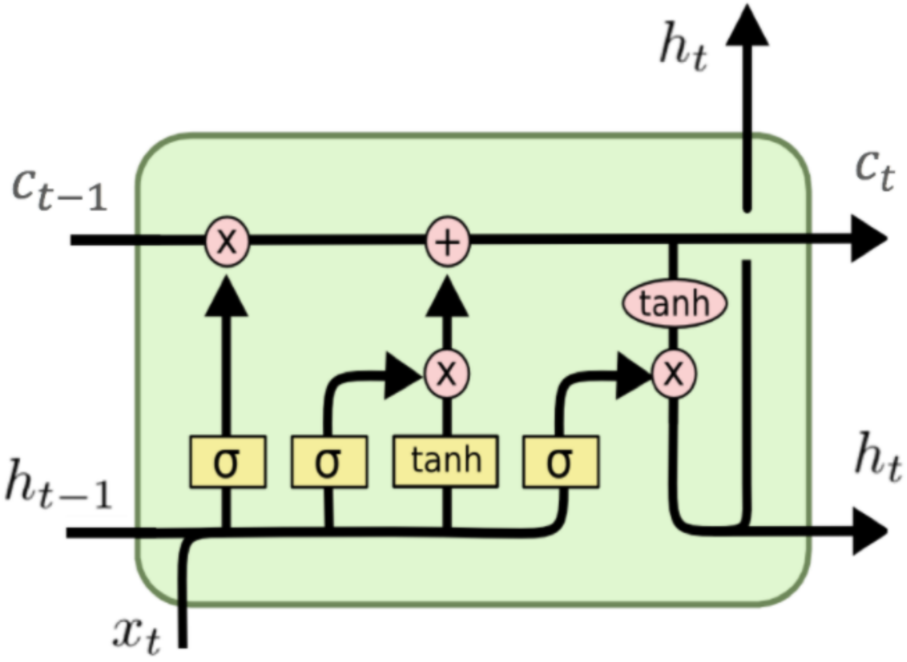


Summary

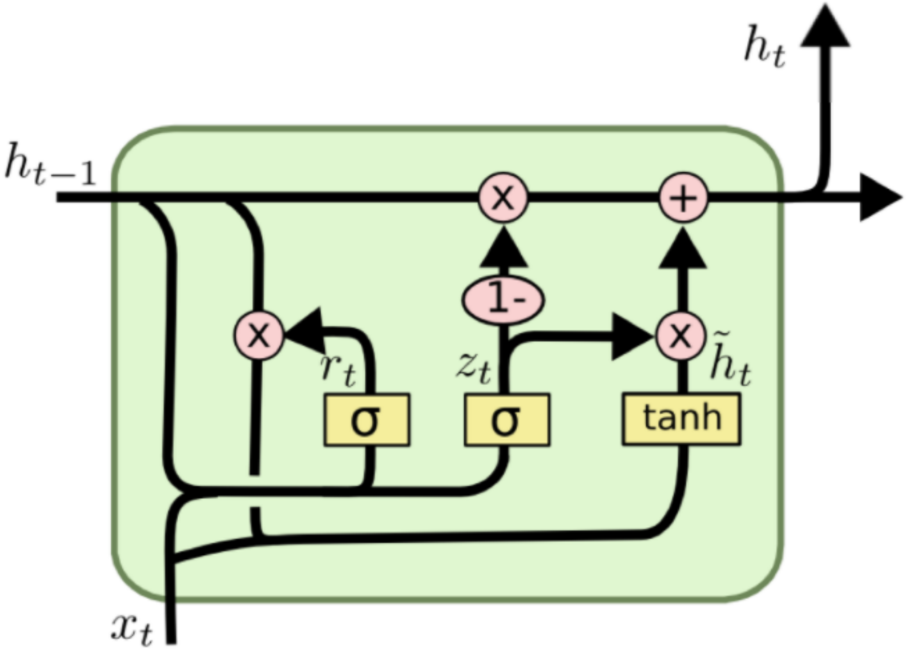
$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$
$$\tilde{H}_t = \tanh(X_t W_{xh} + \left(R_t \odot H_{t-1}\right) W_{hh} + b_h)$$
$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$



LSTM vs. GRU



LSTM
(Long-Short Term Memory)



GRU
(Gated Recurrent Unit)

The GRU is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate.

Recurrent neural networks (RNNs)

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 & GRUs let us model long-term dependencies
5. Models for music generation, classification, machine translation