



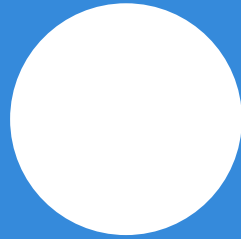
Deep Sequence Modeling

MIT 6.S191

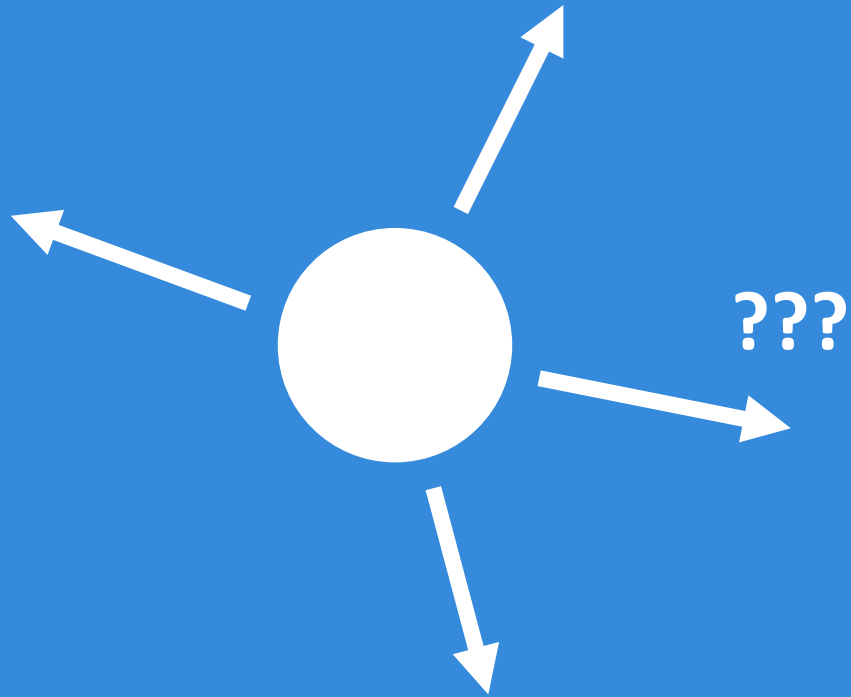
Ava Soleimany
January 28, 2019



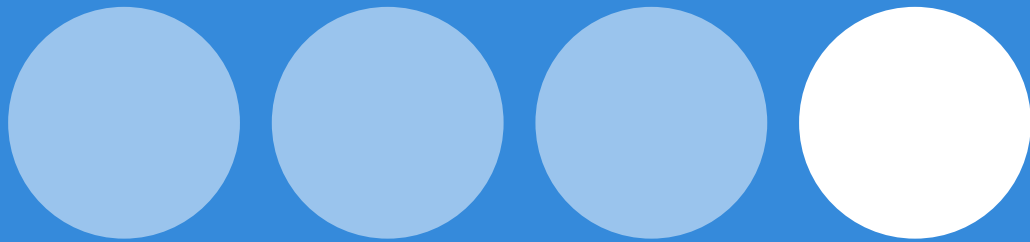
Given an image of a ball,
can you predict where it will go next?



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.”

Adapted from H. Suresh, 6.S191 2018

A sequence modeling problem: predict the next word

“This morning I took my cat for a walk.”

given these words

Adapted from H. Suresh, 6.S191 2018

A sequence modeling problem: predict the next word

“This morning I took my cat for a walk.”

given these words

predict the
next word

Adapted from H. Suresh, 6.S191 2018

Idea #1: use a fixed window

“This morning I took my cat for a walk.”

given these
two words

predict the
next word

Adapted from H. Suresh, 6.S191 2018

Idea #1: use a fixed window

“This morning I took my cat **for a** walk.”

given these predict the
two words next word

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

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

for

a

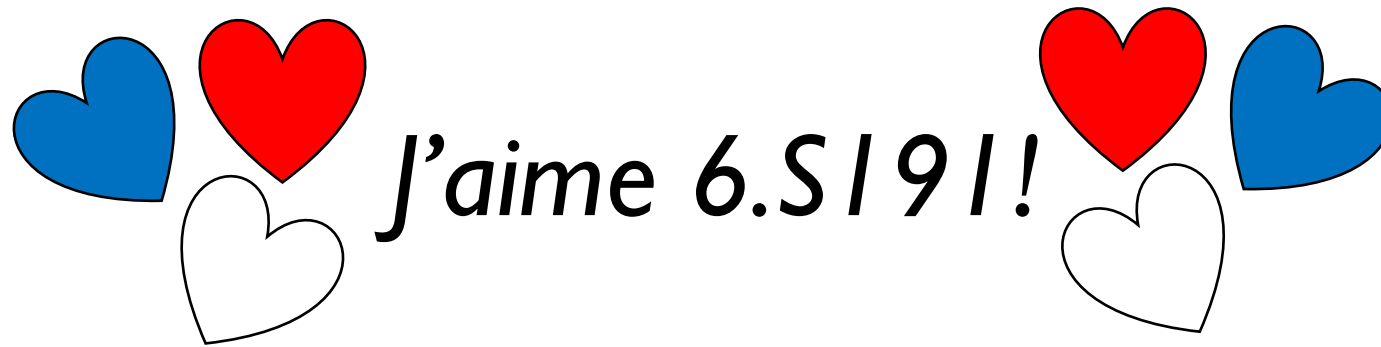


prediction

Adapted from H. Suresh, 6.S191 2018

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.

Adapted from H. Suresh, 6.S191 2018

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

Adapted from H. Suresh, 6.S191 2018

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

Adapted from H. Suresh, 6.S191 2018

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 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 0 1 0 0 0 0 0 0 0 0 0 1 ...]
this morning

Adapted from H. Suresh, 6.S191 2018

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 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 0 1 0 0 0 0 0 0 0 0 0 1 ...]
this morning

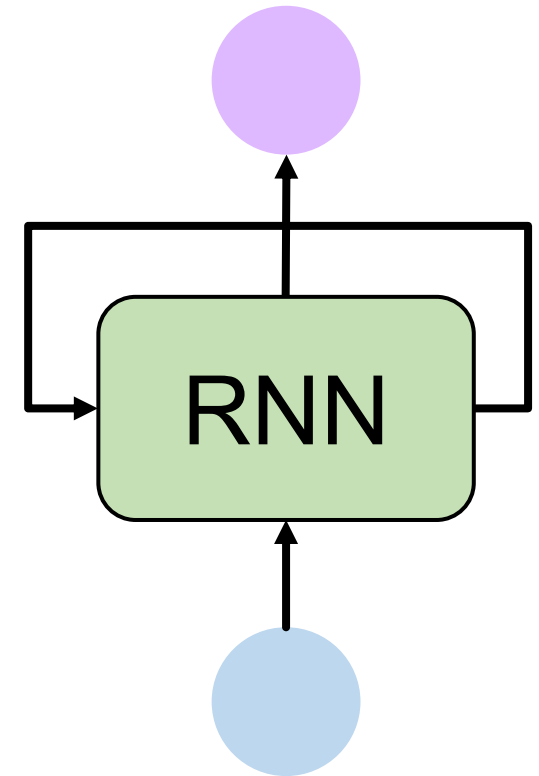
Things we learn about the sequence **won't transfer** if they appear **elsewhere** in the sequence.

Adapted from H. Suresh, 6.S191 2018

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

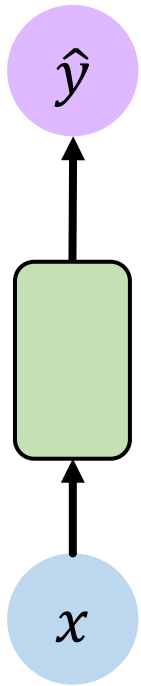


Today: **Recurrent Neural Networks (RNNs)** as an approach to sequence modeling problems

Adapted from H. Suresh, 6.S191 2018

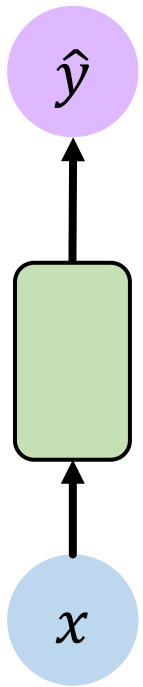
Recurrent Neural Networks (RNNs)

Standard feed-forward neural network

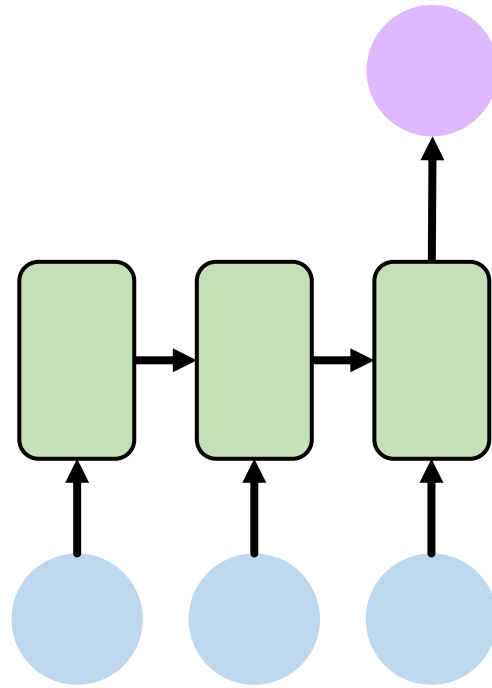


One to One
“Vanilla” neural network

Recurrent neural networks: sequence modeling

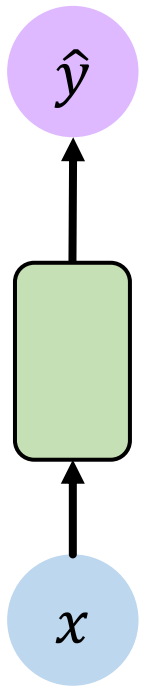


One to One
"Vanilla" neural network

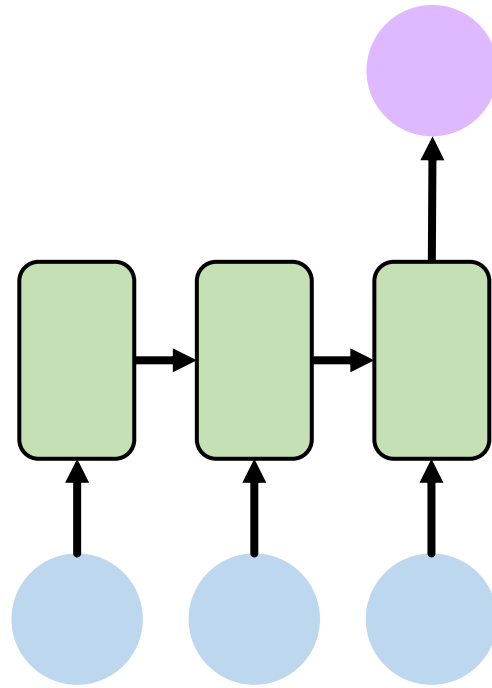


Many to One
Sentiment Classification

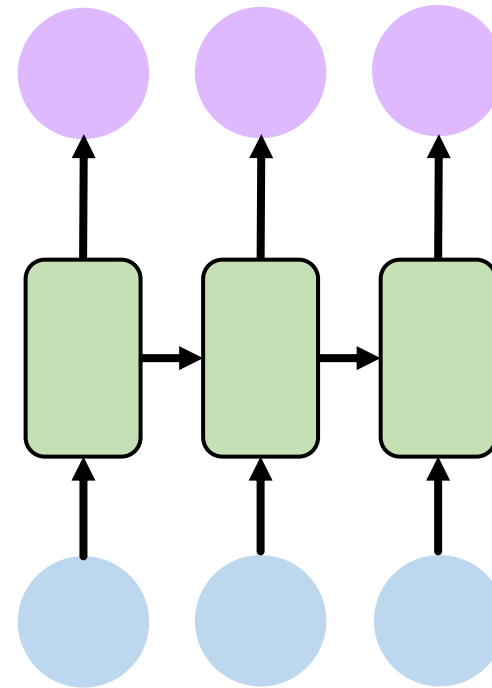
Recurrent neural networks: sequence modeling



One to One
"Vanilla" neural network



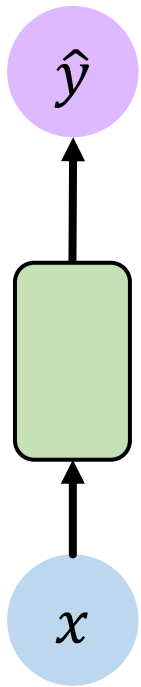
Many to One
Sentiment Classification



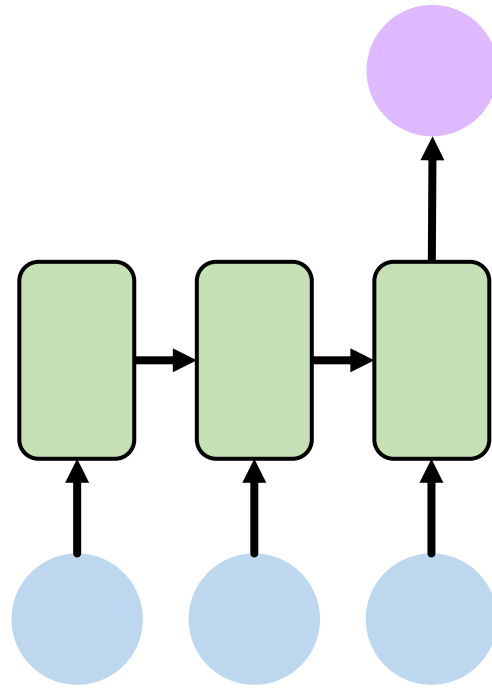
Many to Many
Music Generation



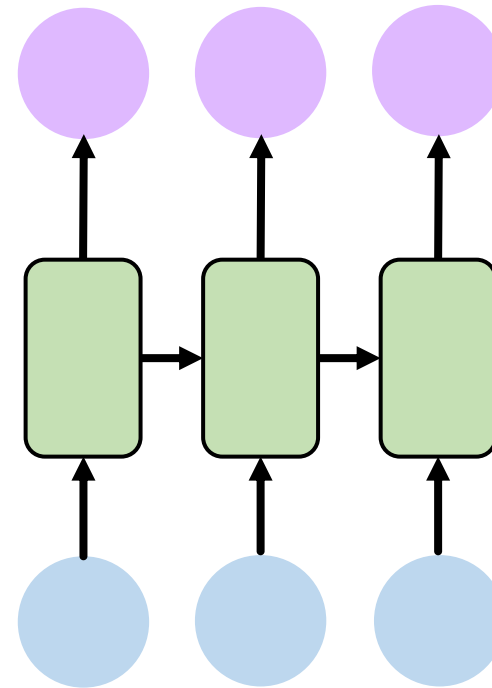
Recurrent neural networks: sequence modeling



One to One
"Vanilla" neural network



Many to One
Sentiment Classification

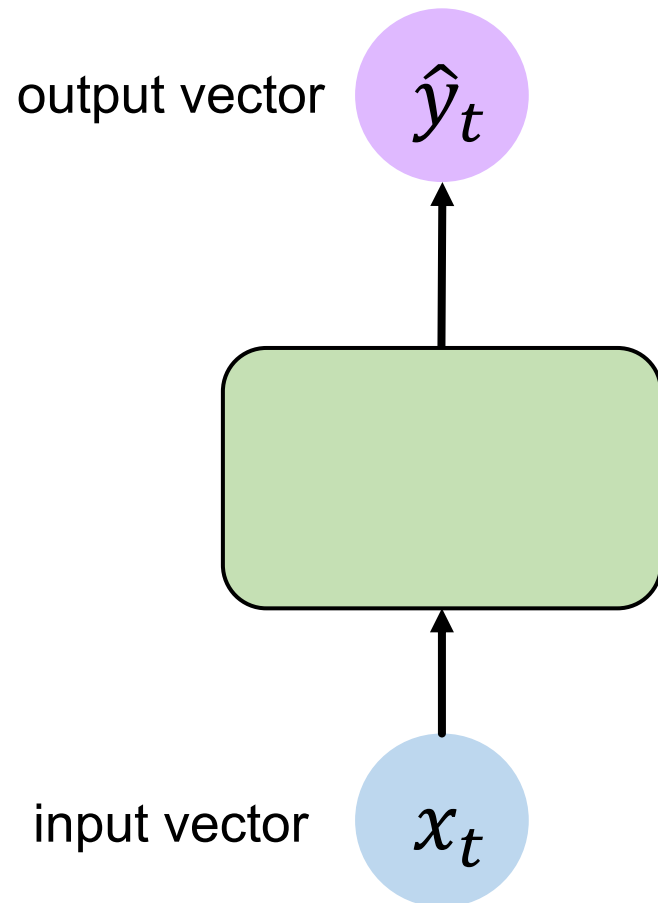


Many to Many
Music Generation

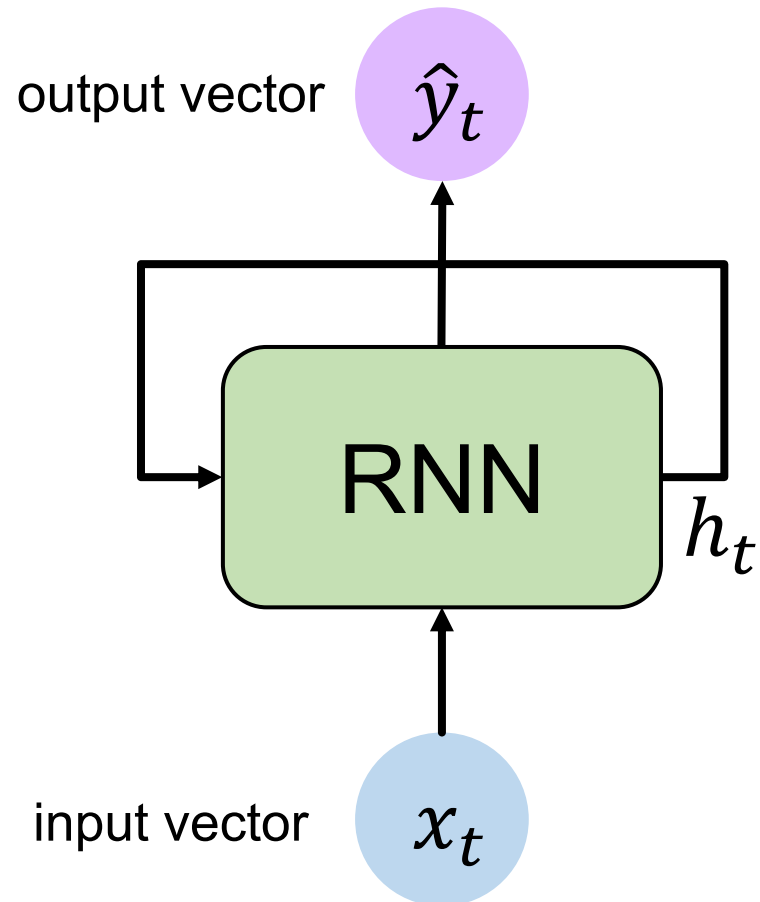


... and many other architectures and applications

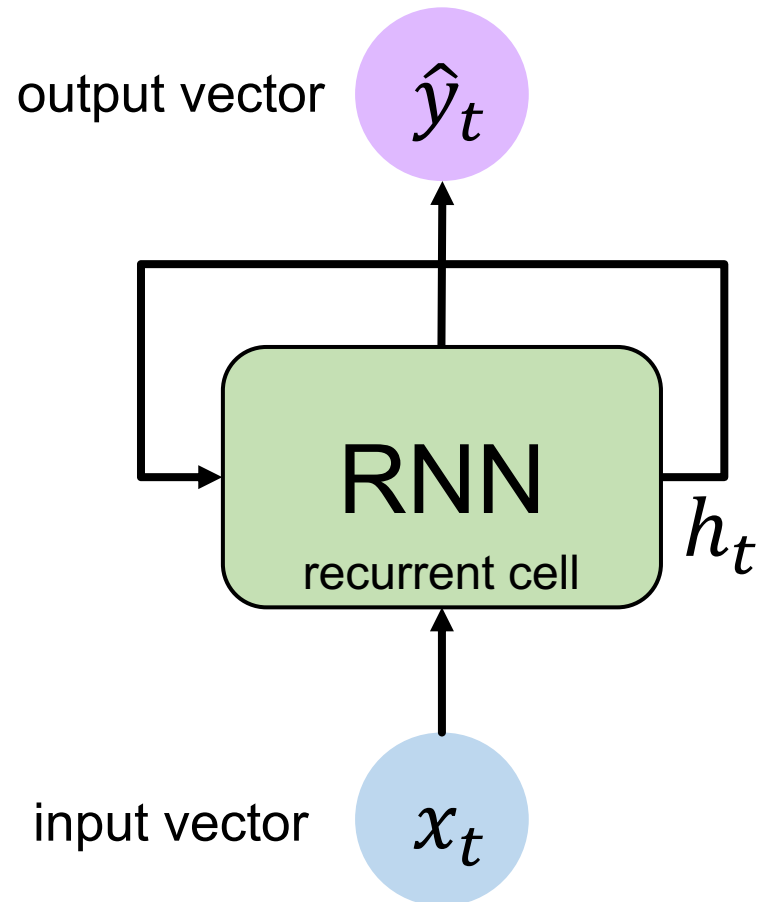
A standard “vanilla” neural network



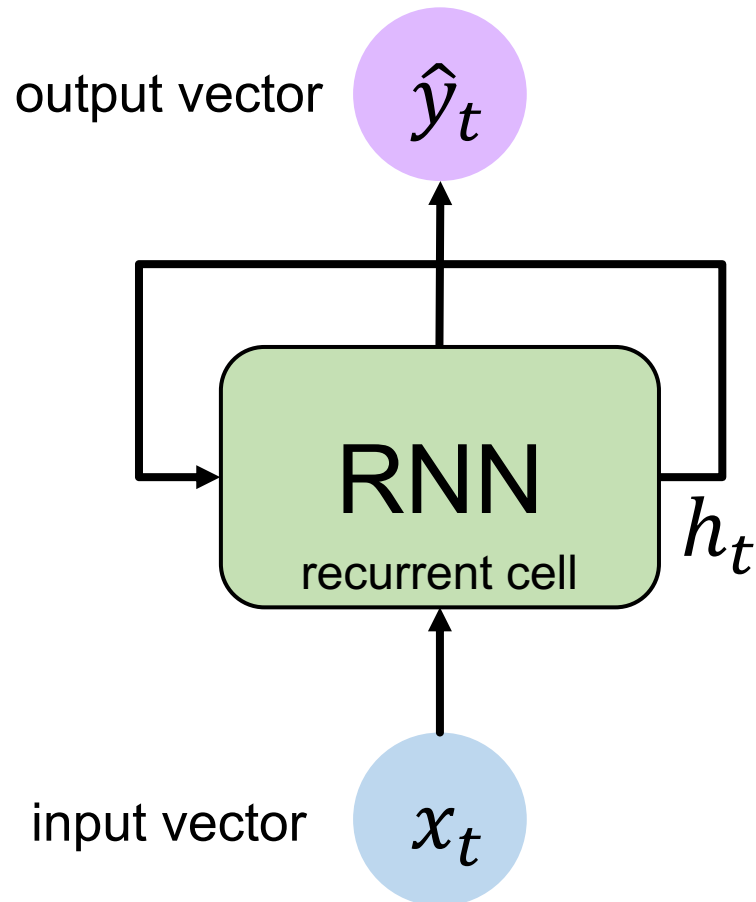
A recurrent neural network (RNN)



A recurrent neural network (RNN)

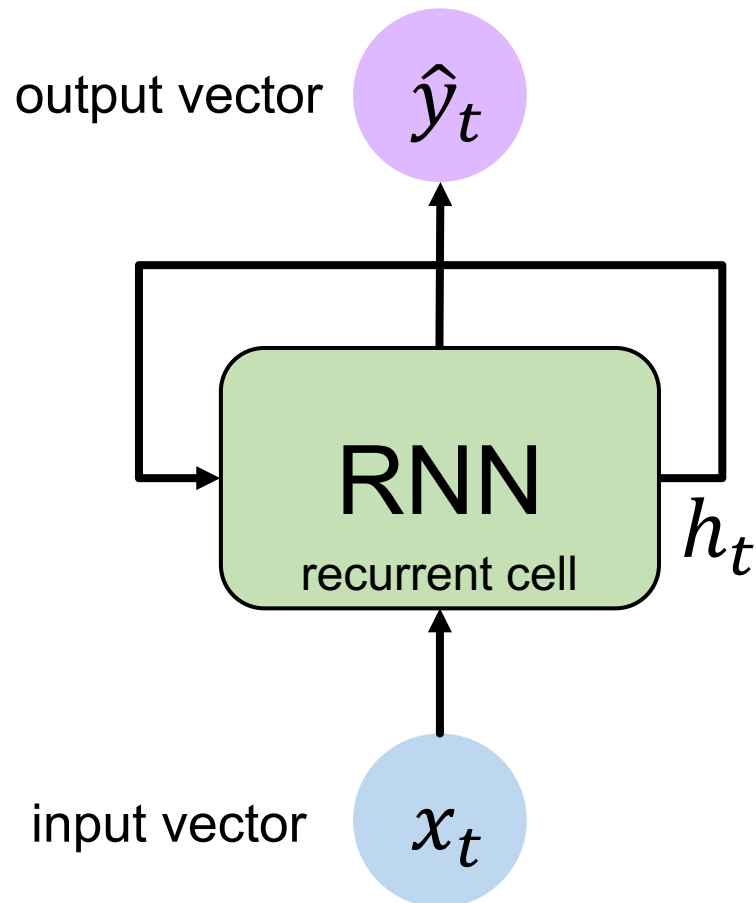


A recurrent neural network (RNN)



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

A recurrent neural network (RNN)

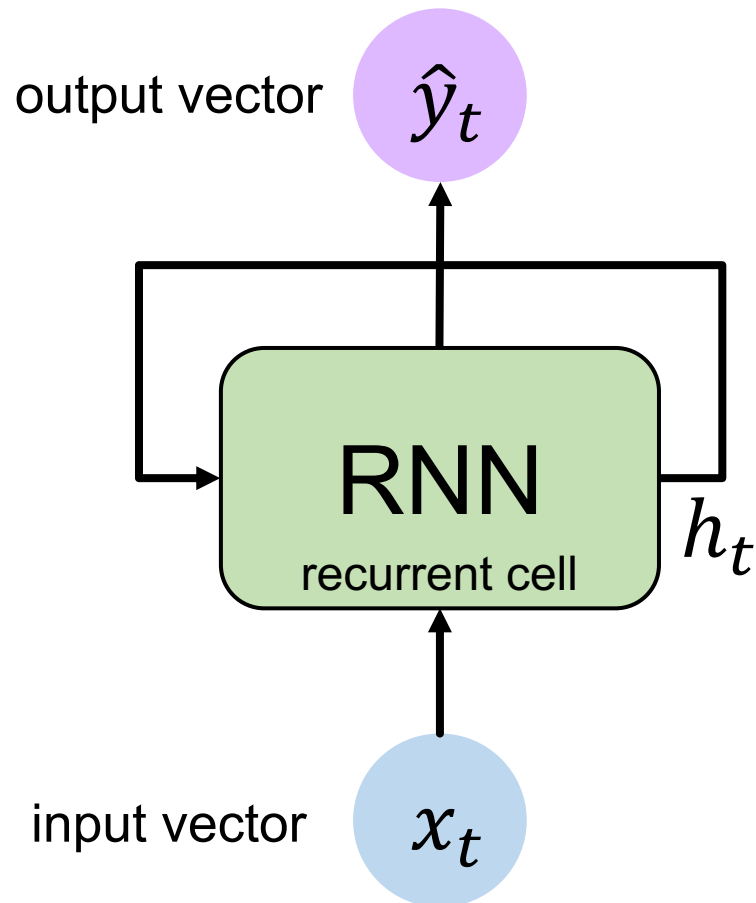


Apply a **recurrence relation** at every time step to process a sequence:

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

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

A recurrent neural network (RNN)



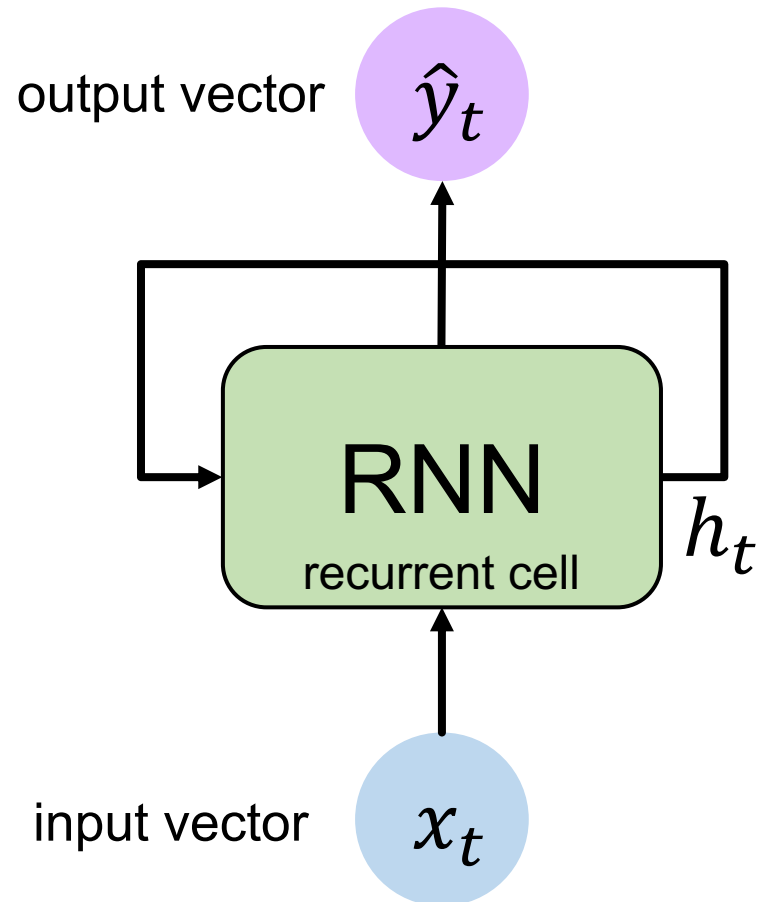
Apply a **recurrence relation** at every time step to process a sequence:

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

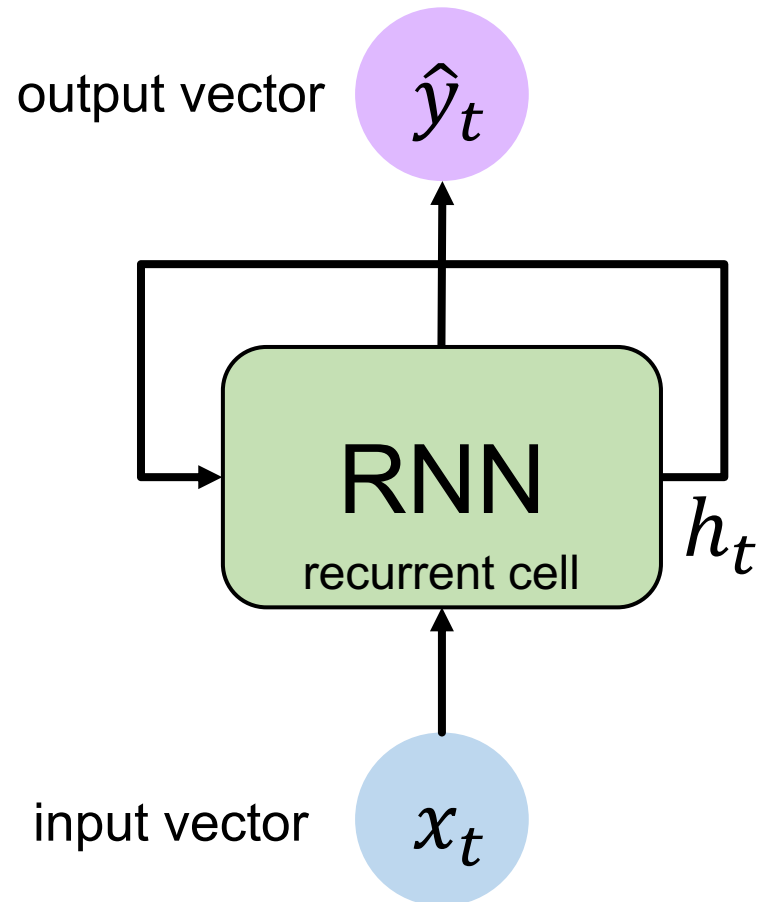
new state function parameterized by W old state input vector at time step t

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

RNN state update and output

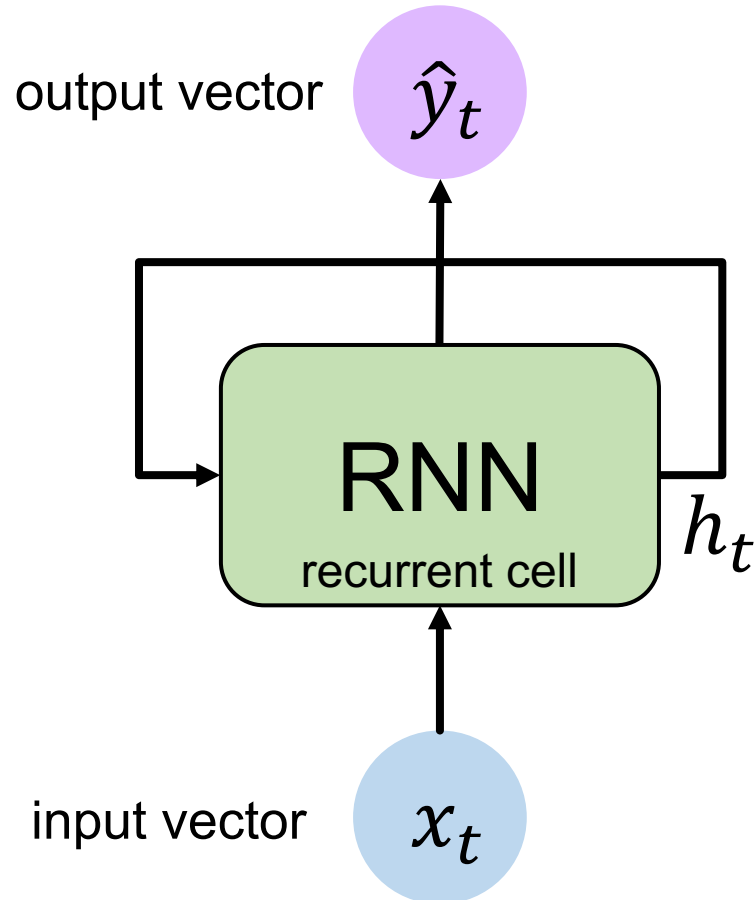


RNN state update and output



Input Vector

RNN state update and output

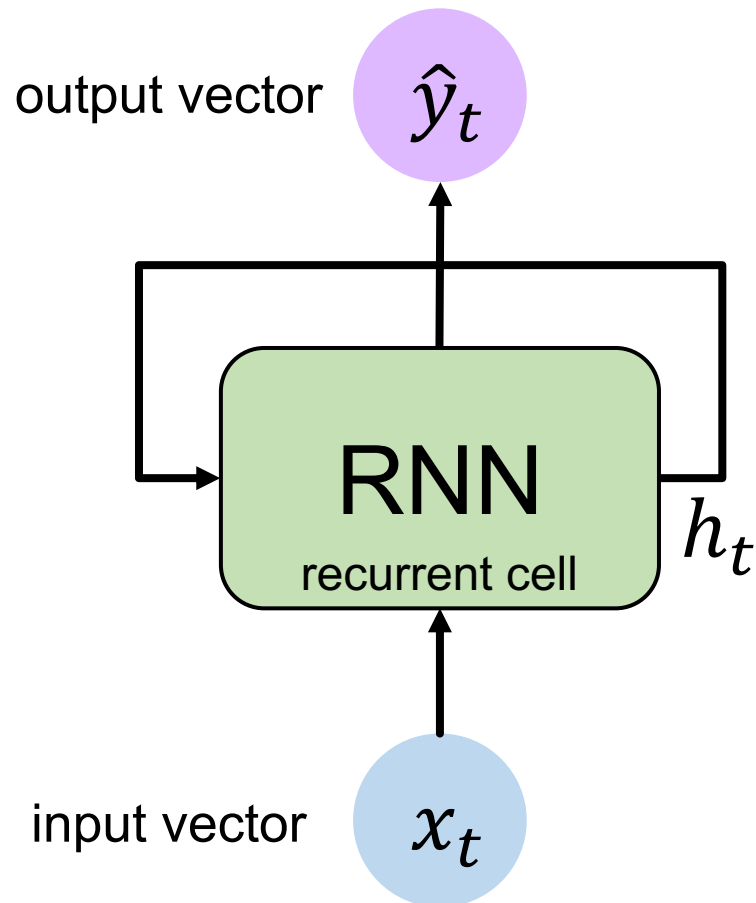


Update Hidden State

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

Input Vector

RNN state update and output



Output Vector

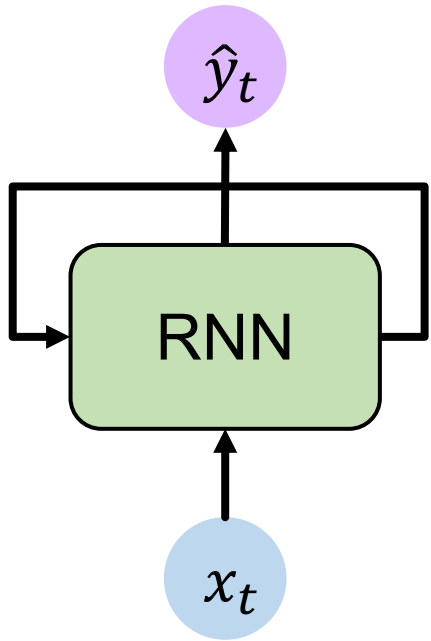
$$\hat{y}_t = \mathbf{W}_{hy}h_t$$

Update Hidden State

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

Input Vector

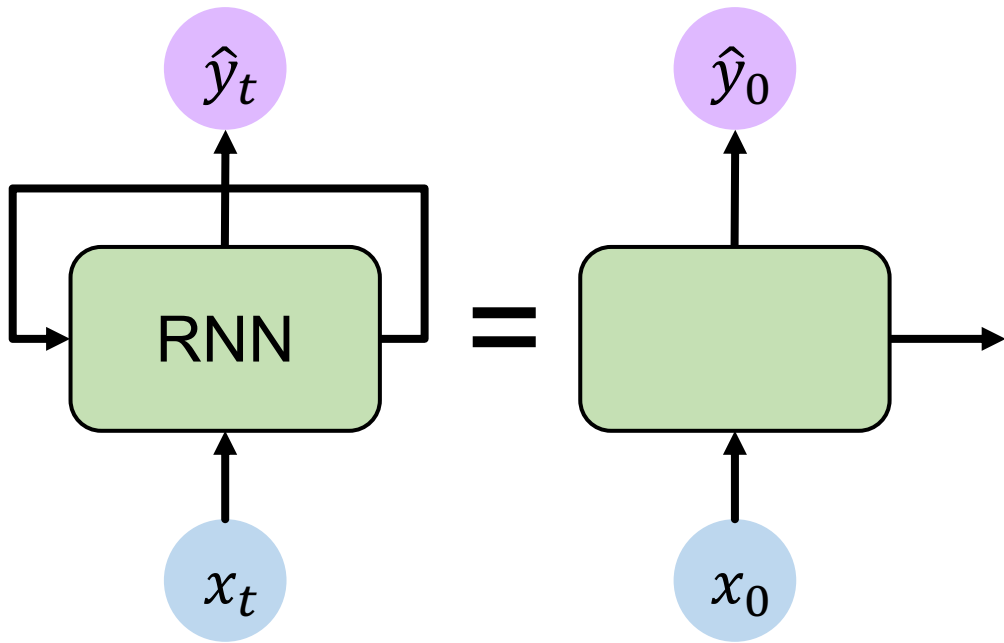
RNNs: computational graph across time



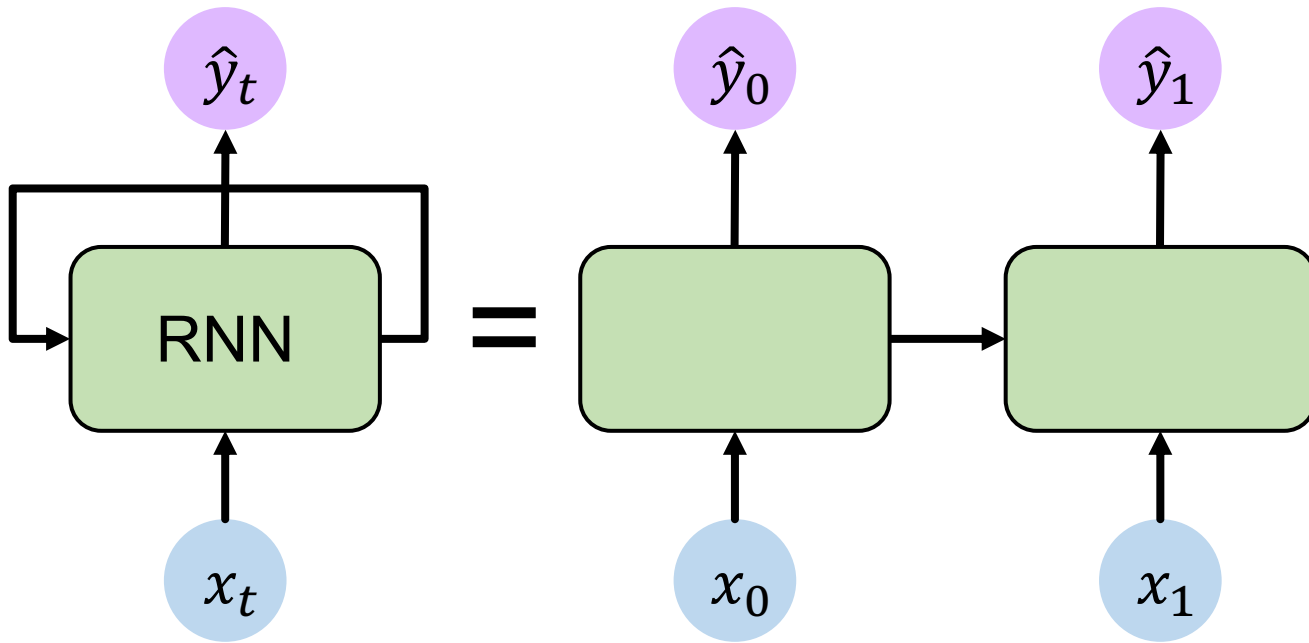
=

Represent as computational graph unrolled across time

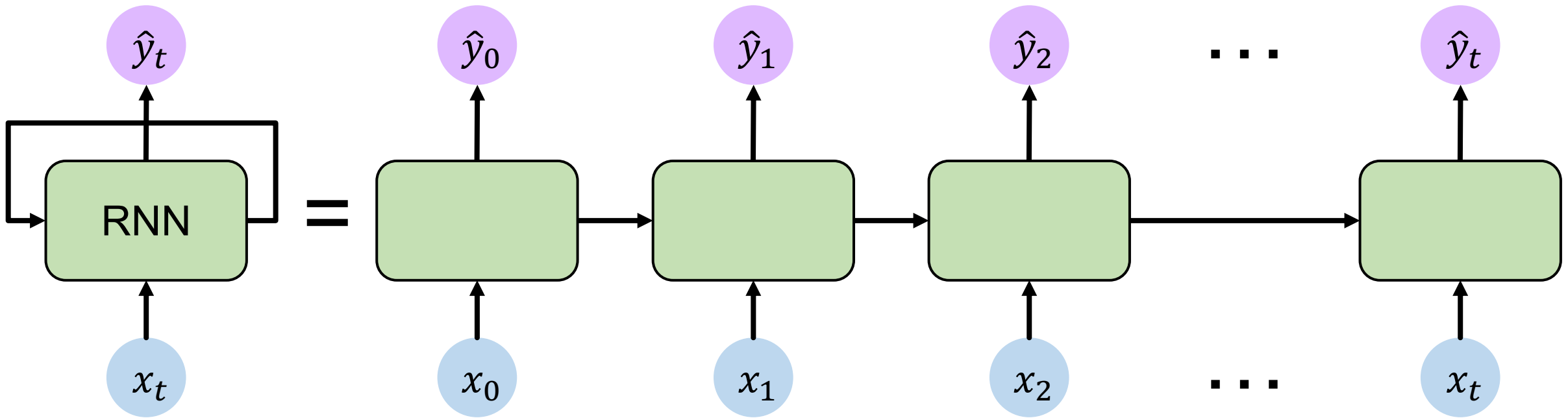
RNNs: computational graph across time



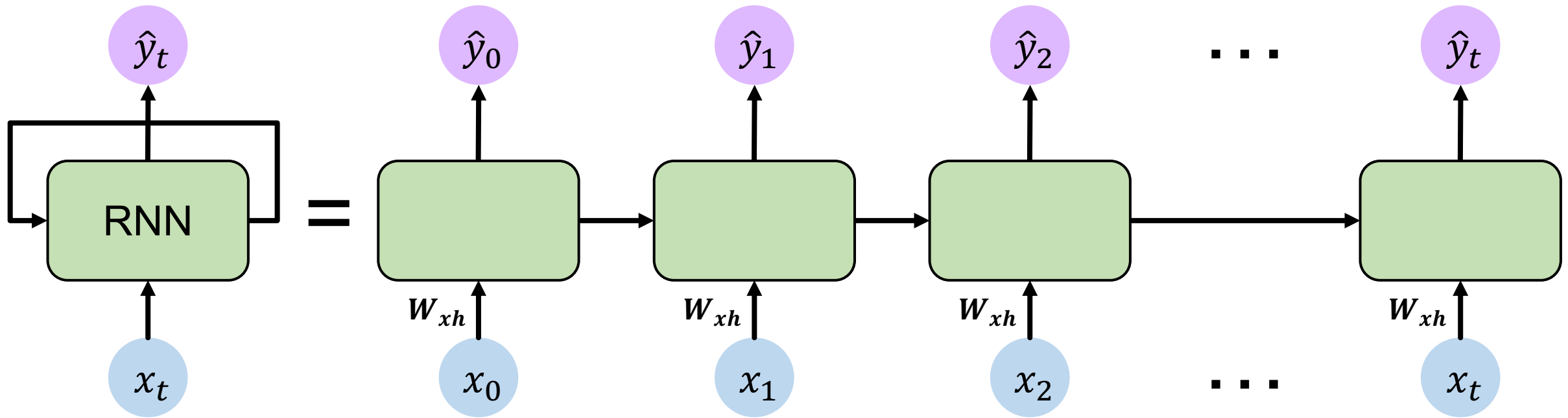
RNNs: computational graph across time



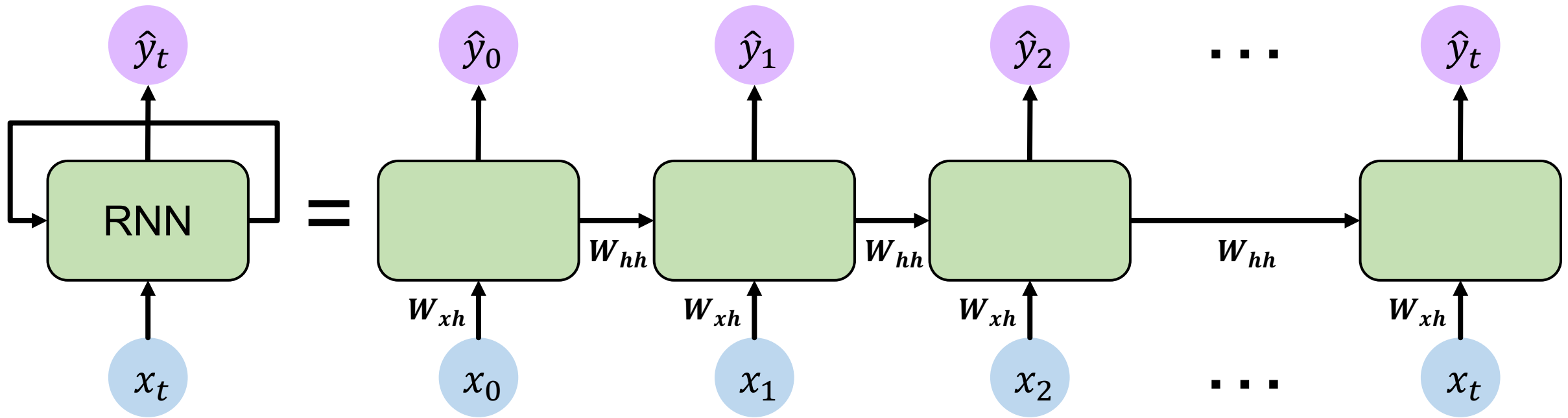
RNNs: computational graph across time



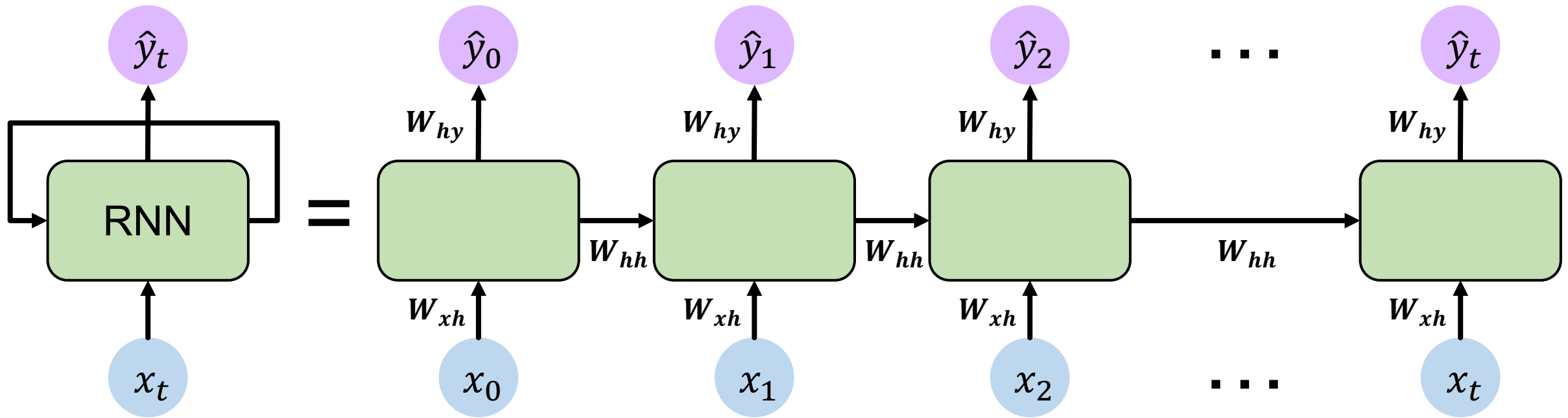
RNNs: computational graph across time



RNNs: computational graph across time

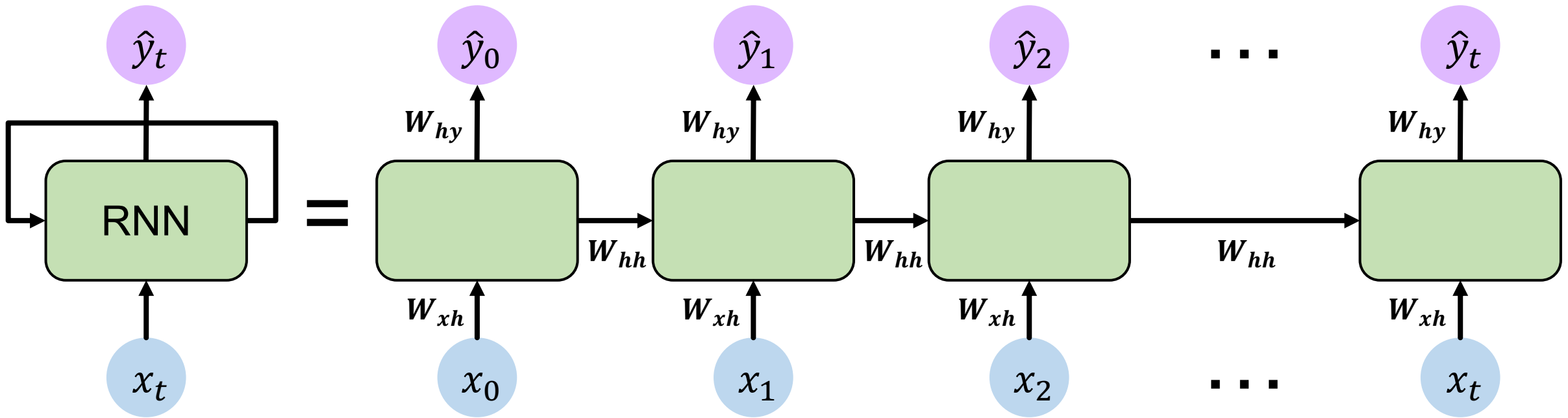


RNNs: computational graph across time



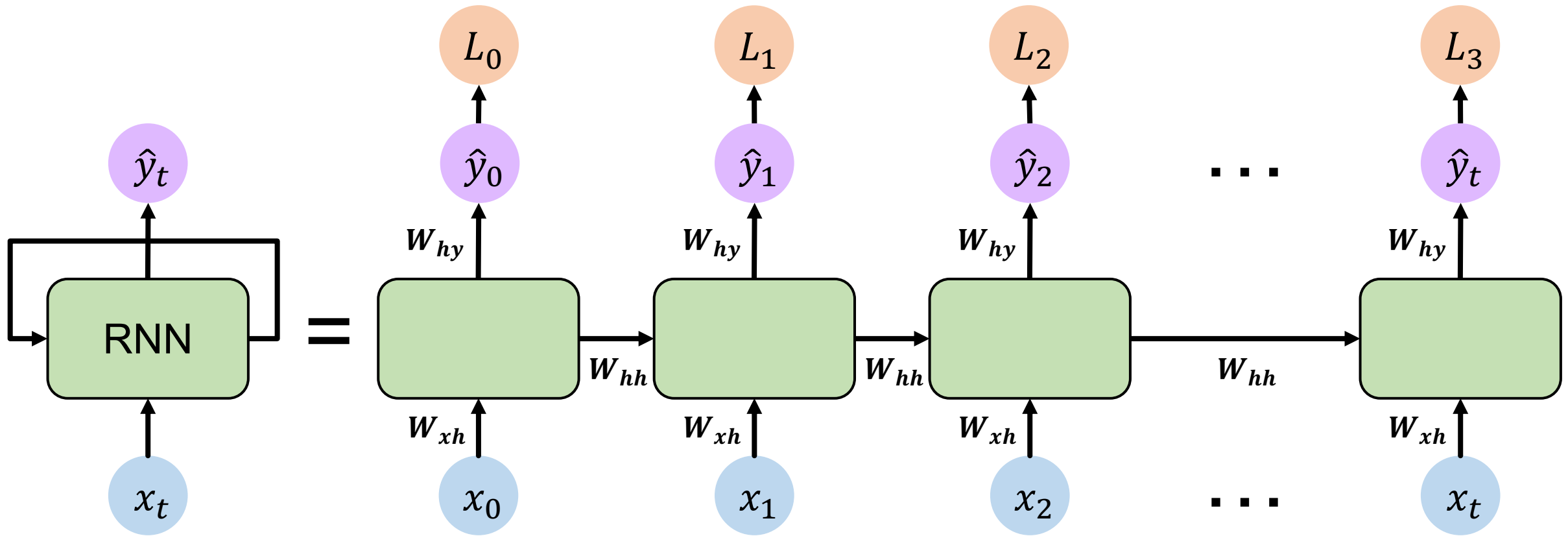
RNNs: computational graph across time

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

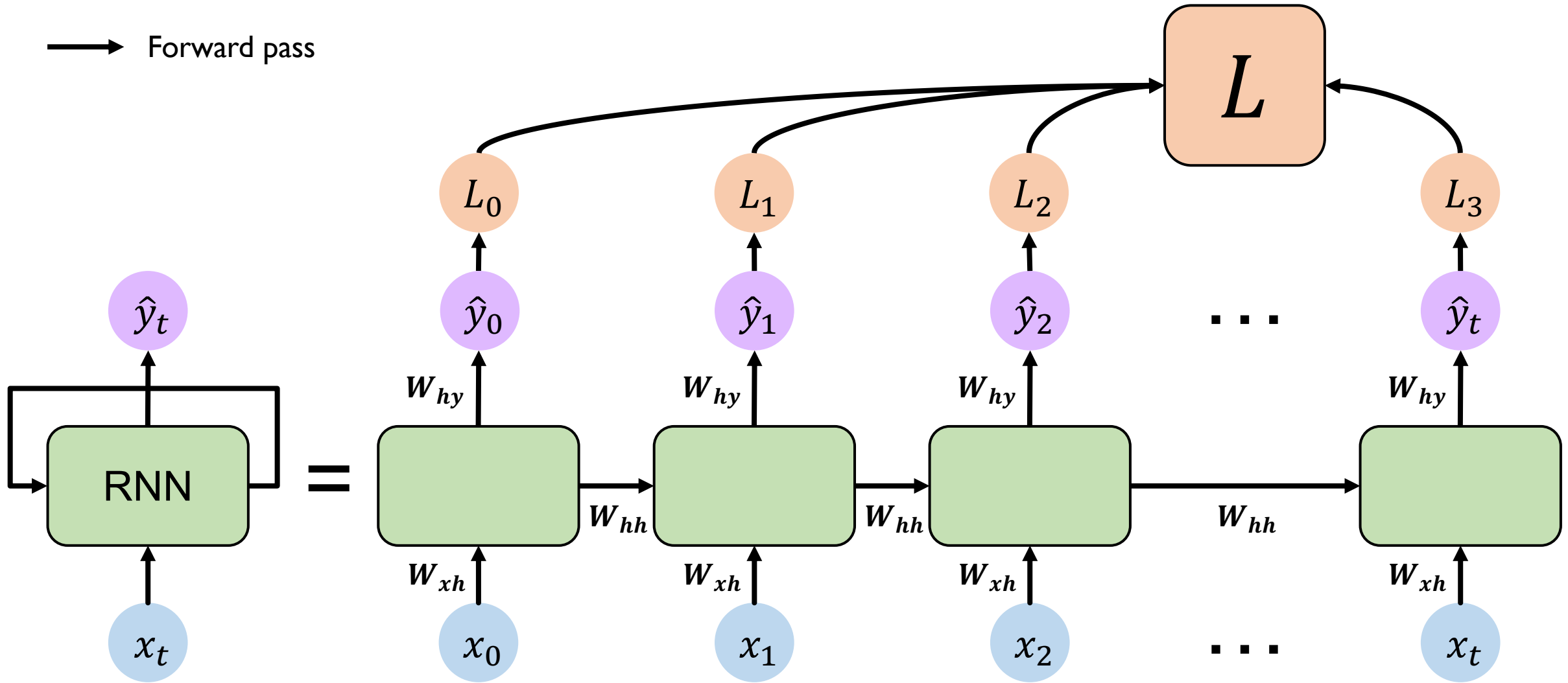


RNNs: computational graph across time

→ Forward pass

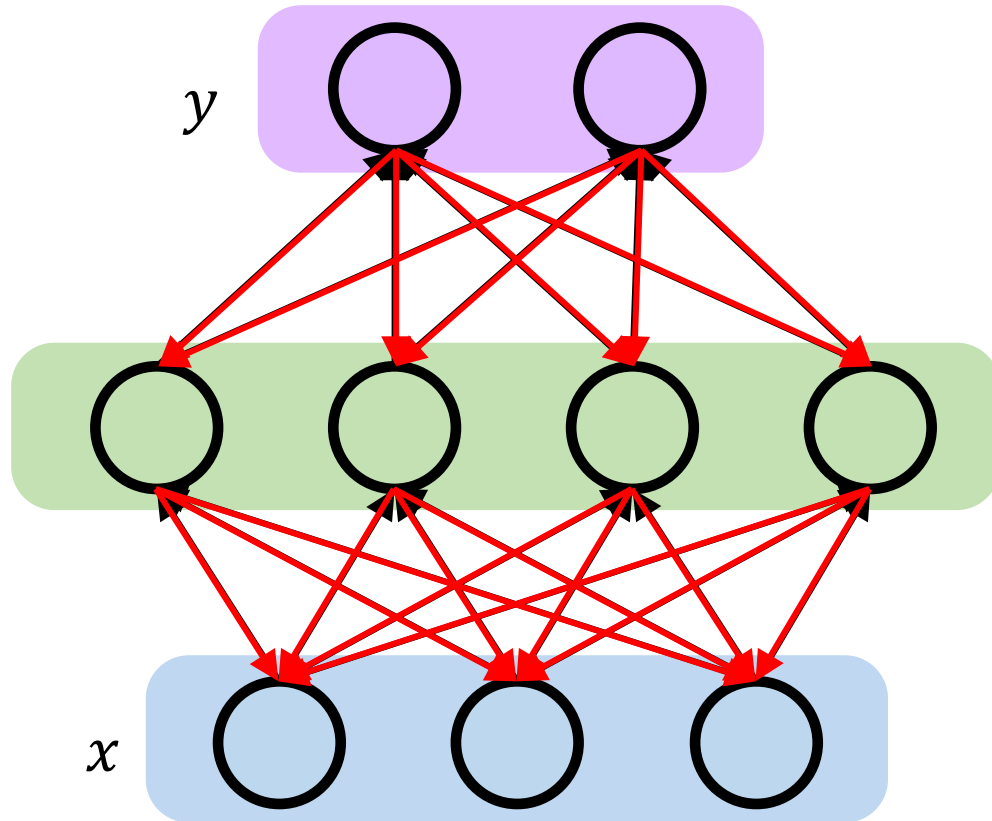


RNNs: computational graph across time



Backpropagation Through Time (BPTT)

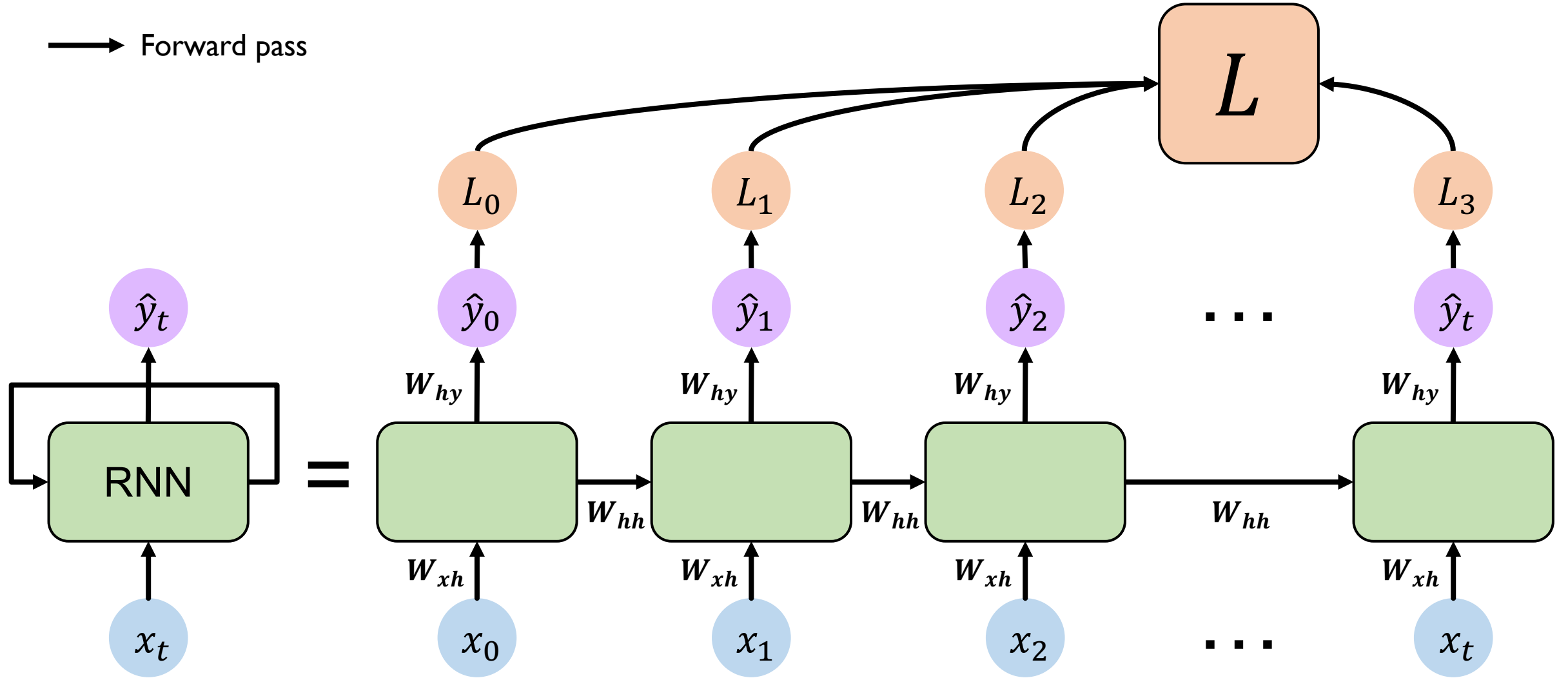
Recall: backpropagation in feed forward models



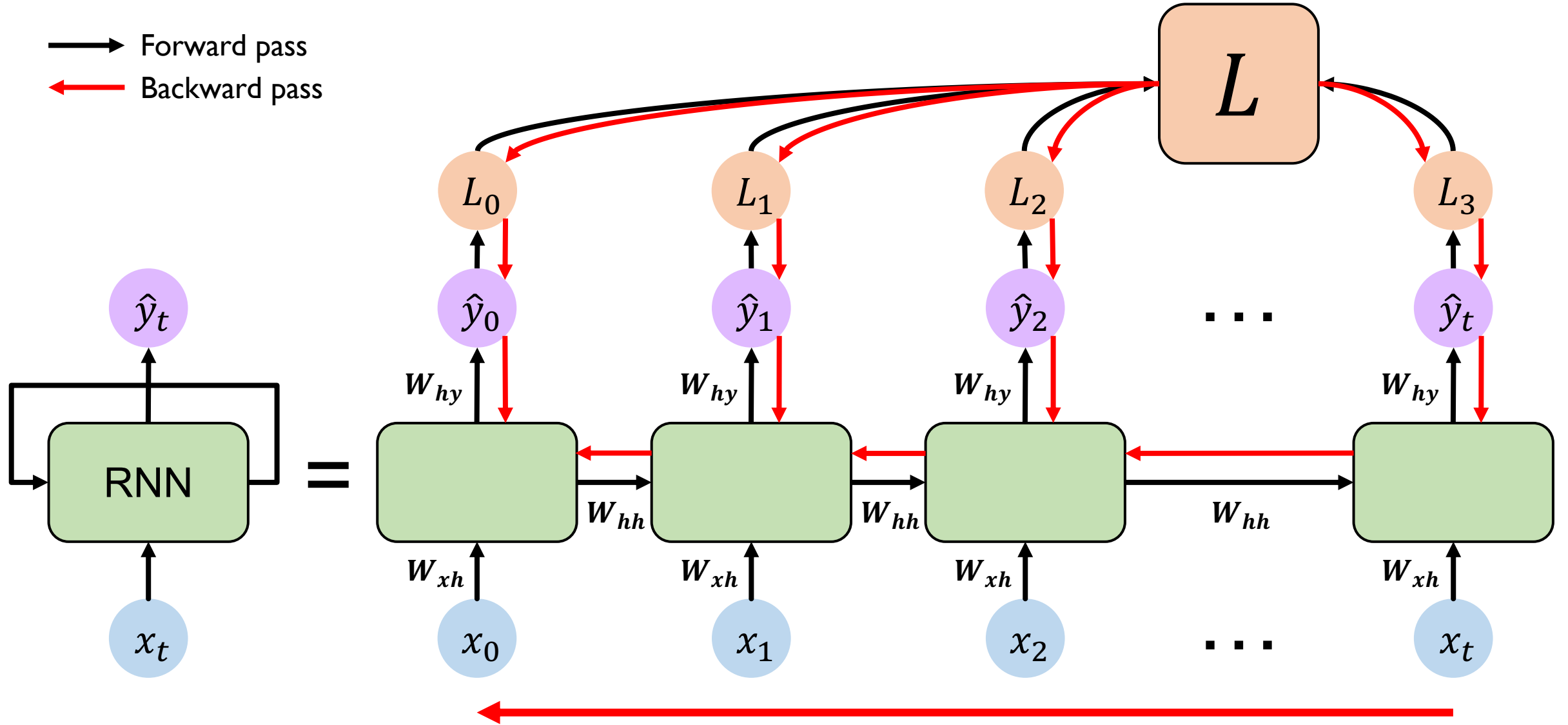
Backpropagation algorithm:

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

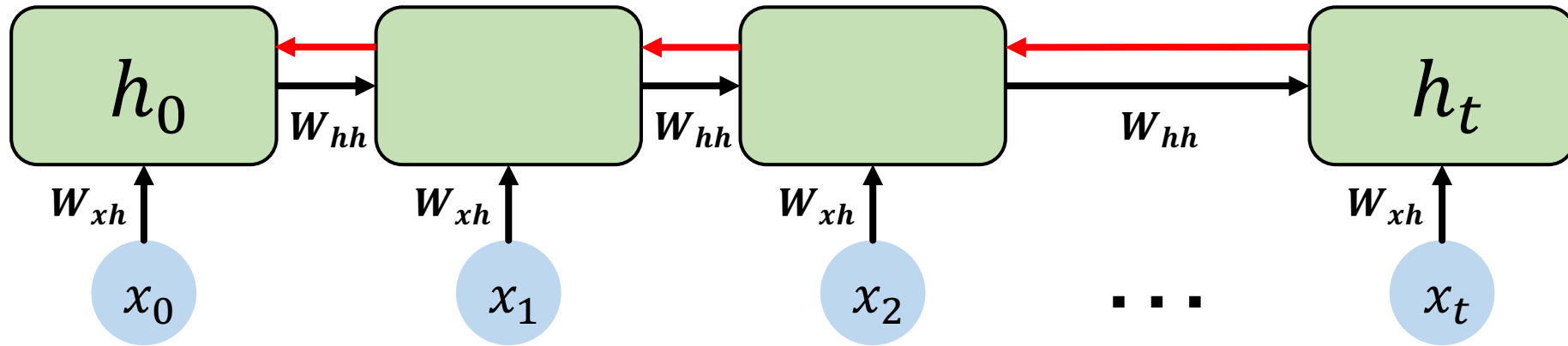
RNNs: backpropagation through time



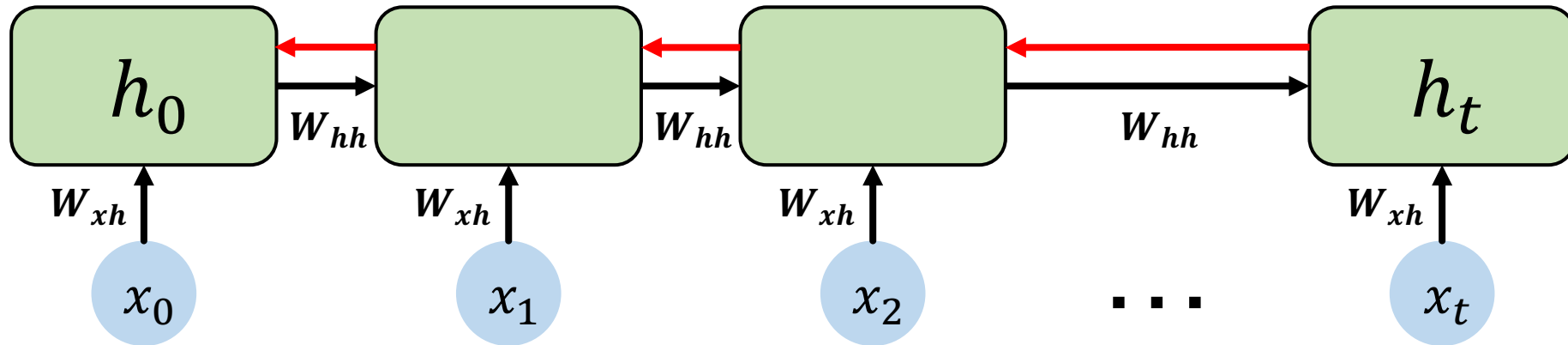
RNNs: backpropagation through time



Standard RNN gradient flow

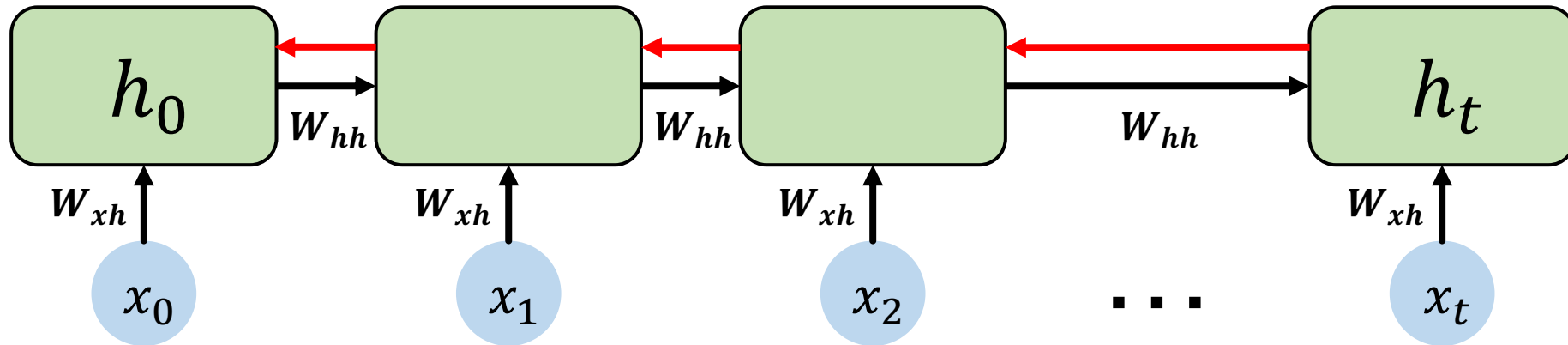


Standard RNN gradient flow



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

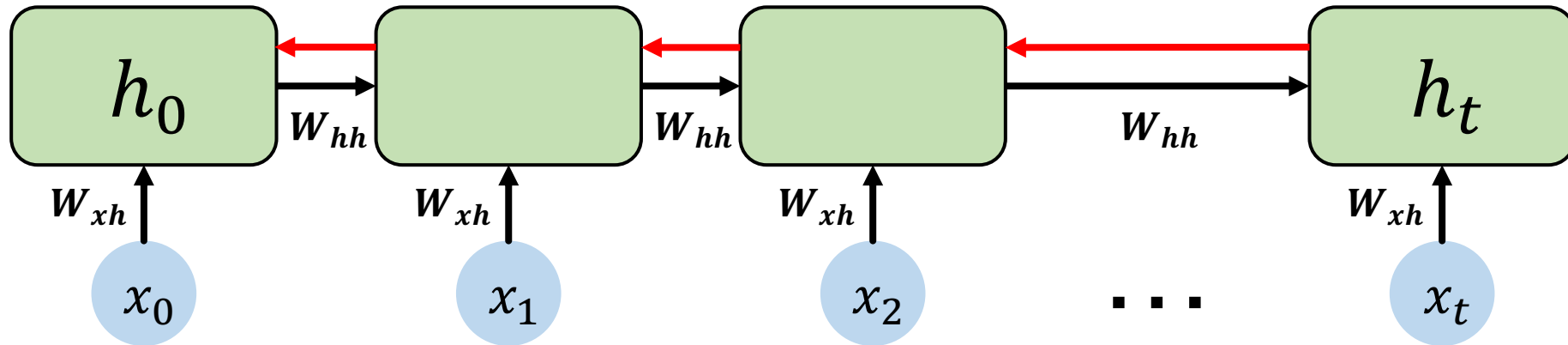
Standard RNN gradient flow: exploding gradients



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

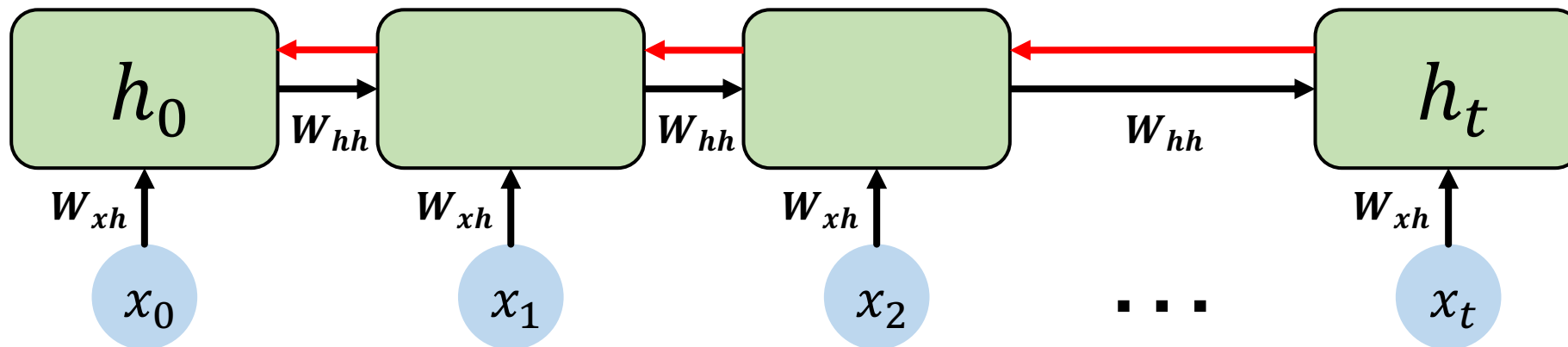


Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f' !)

Many values > 1 :
exploding gradients

Gradient clipping to
scale big gradients

Standard RNN gradient flow: vanishing gradients



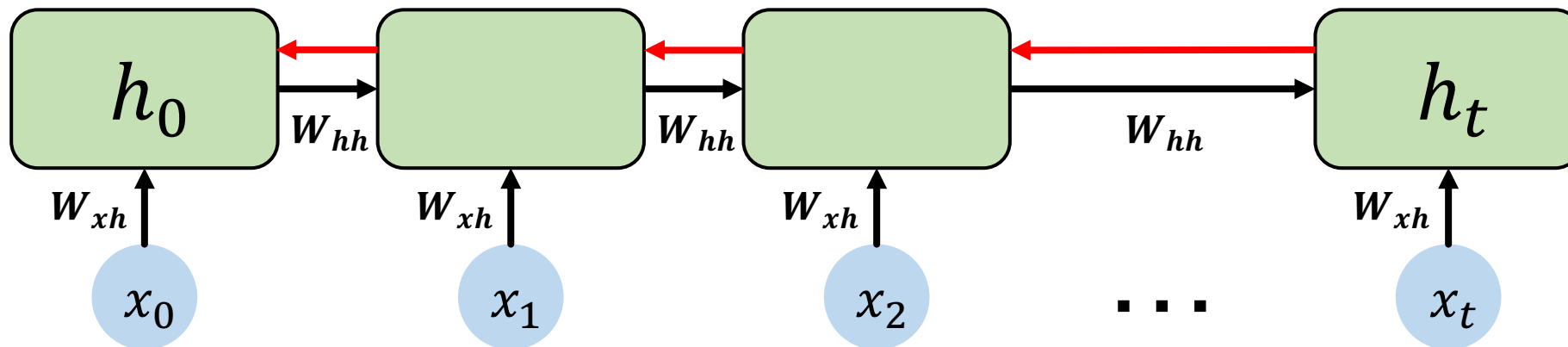
Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f' !)

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

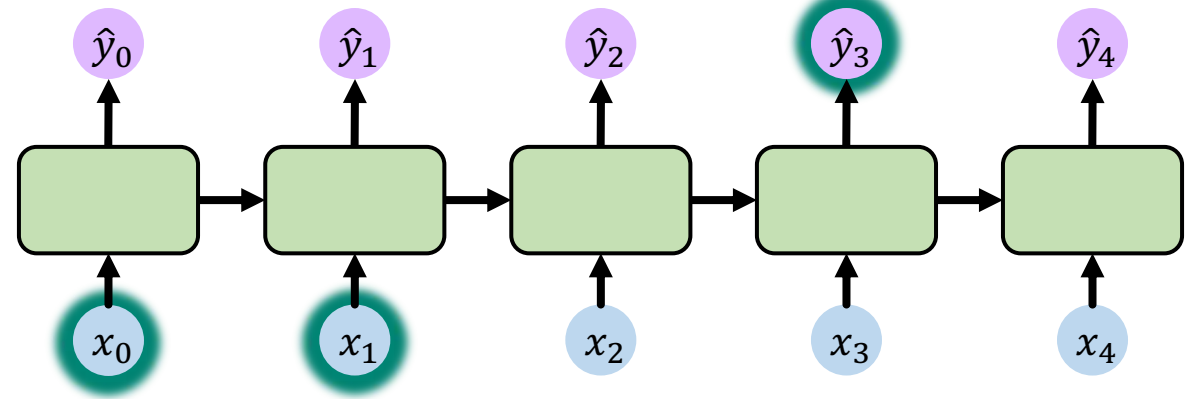


Errors due to further back time steps have smaller and smaller gradients



Bias parameters to capture short-term dependencies

“The clouds are in the ____”



The problem of long-term dependencies

Why are vanishing gradients a problem?

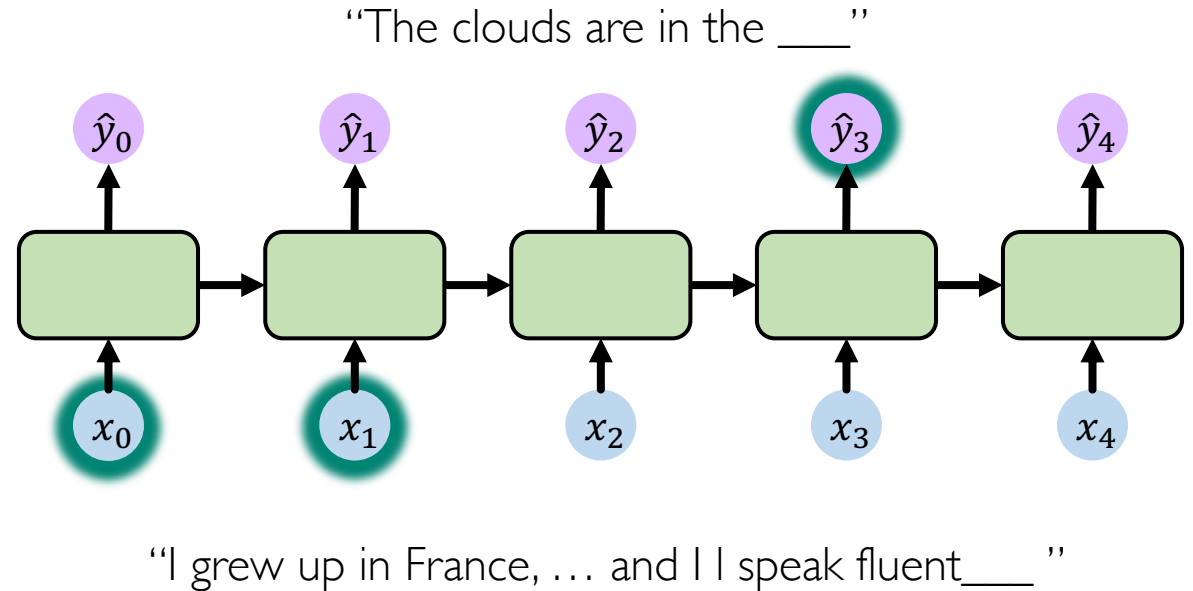
Multiply many **small numbers** together



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



Bias parameters to capture short-term dependencies



The problem of long-term dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

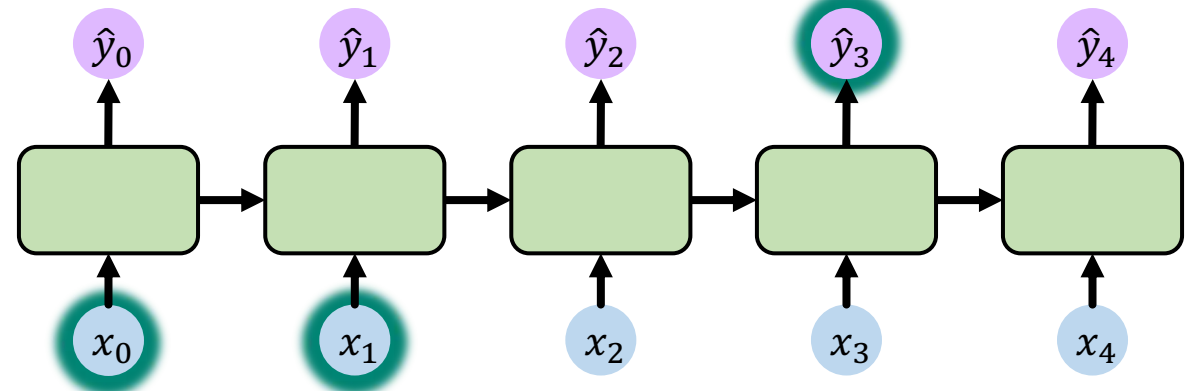


Errors due to further back time steps have smaller and smaller gradients

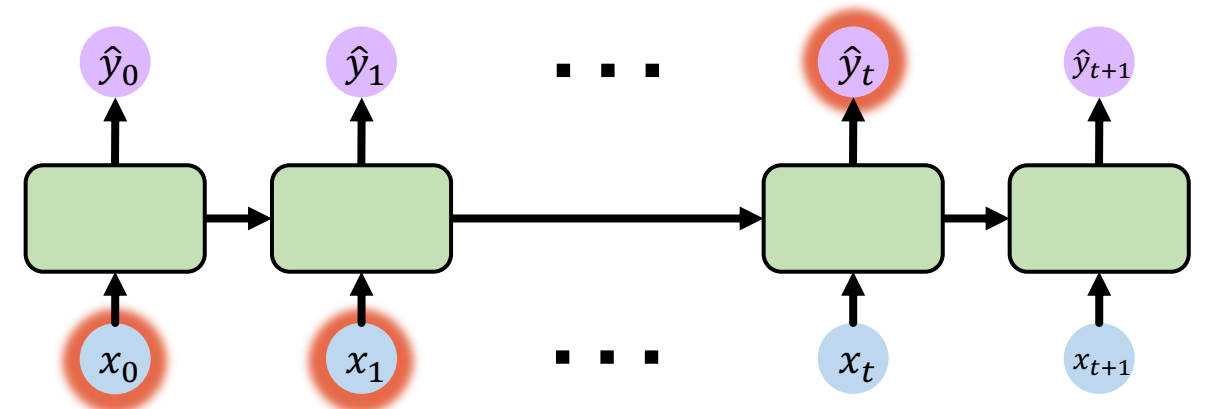


Bias parameters to capture short-term dependencies

“The clouds are in the ____”

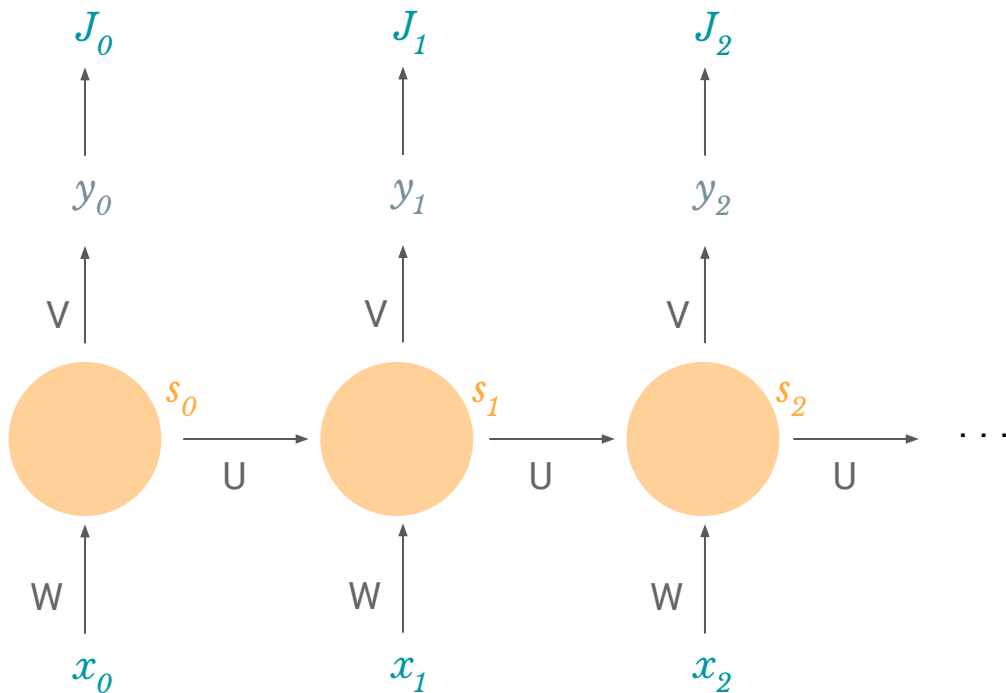


“I grew up in France, ... and I I speak fluent ____”



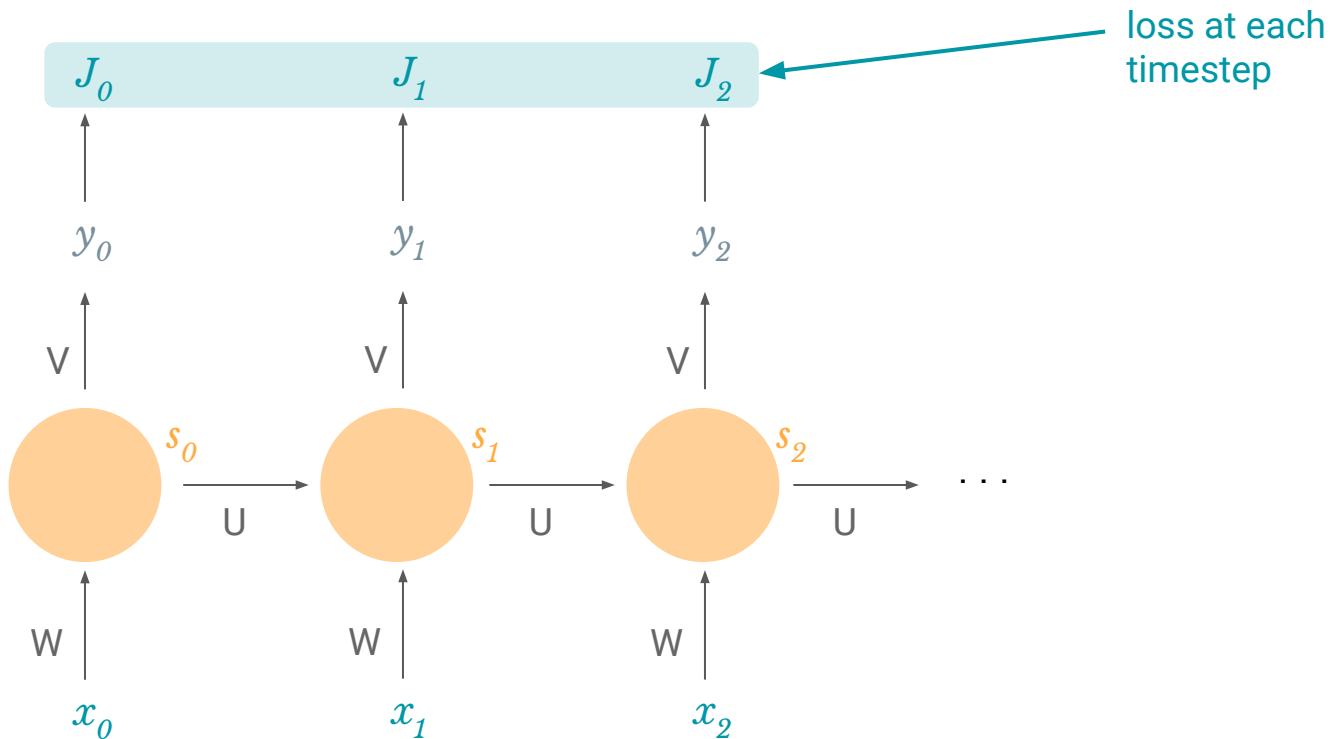
we have a **loss at each timestep**:

(since we're making a prediction at each timestep)



we have a **loss at each timestep**:

(since we're making a prediction at each timestep)



we **sum the losses** across time:

$$\text{loss at time } t = J_t(\Theta)$$

Θ = our
parameters, like
weights



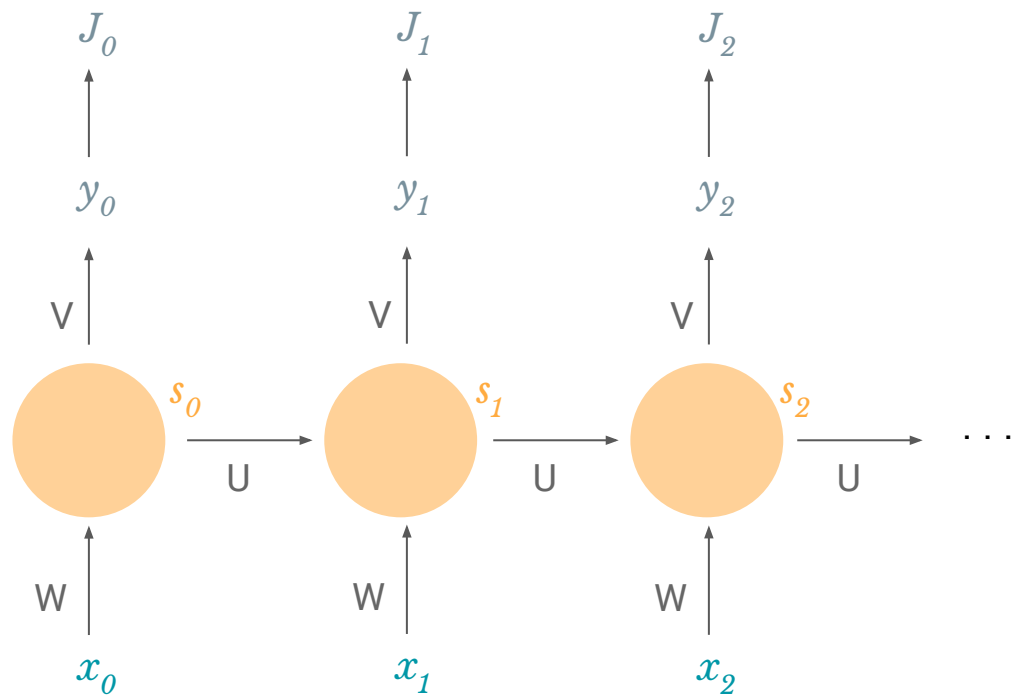
$$\text{total loss} = J(\Theta) = \sum_t J_t(\Theta)$$

what are our **gradients**?

we sum gradients across time for each parameter P :

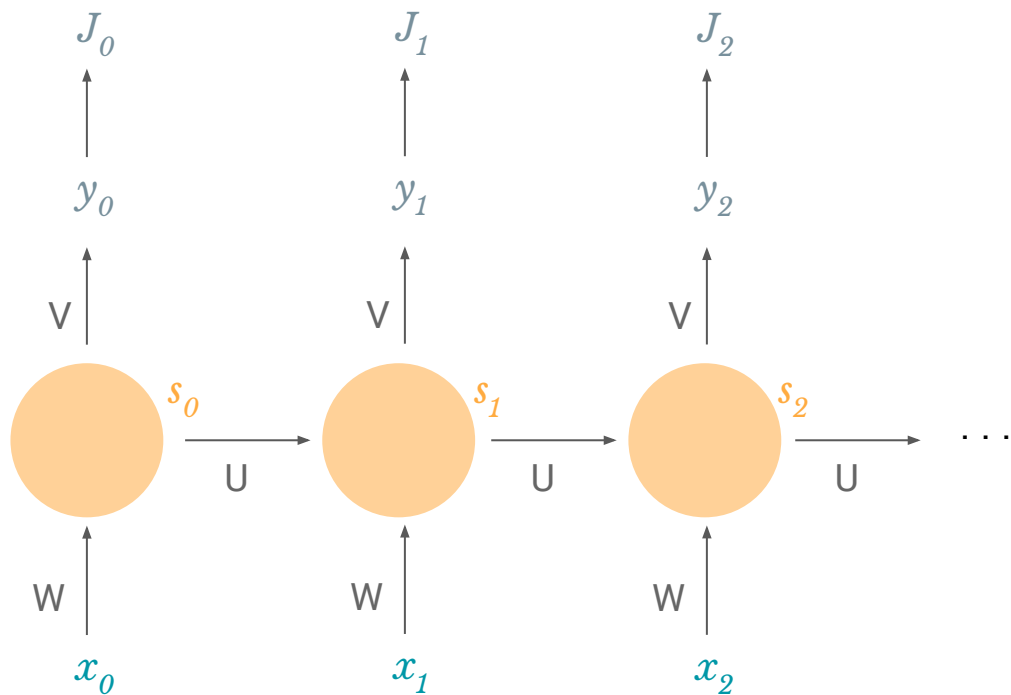
$$\frac{\partial J}{\partial P} = \sum_t \frac{\partial J_t}{\partial P}$$

let's try it out for W with the **chain rule**:



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

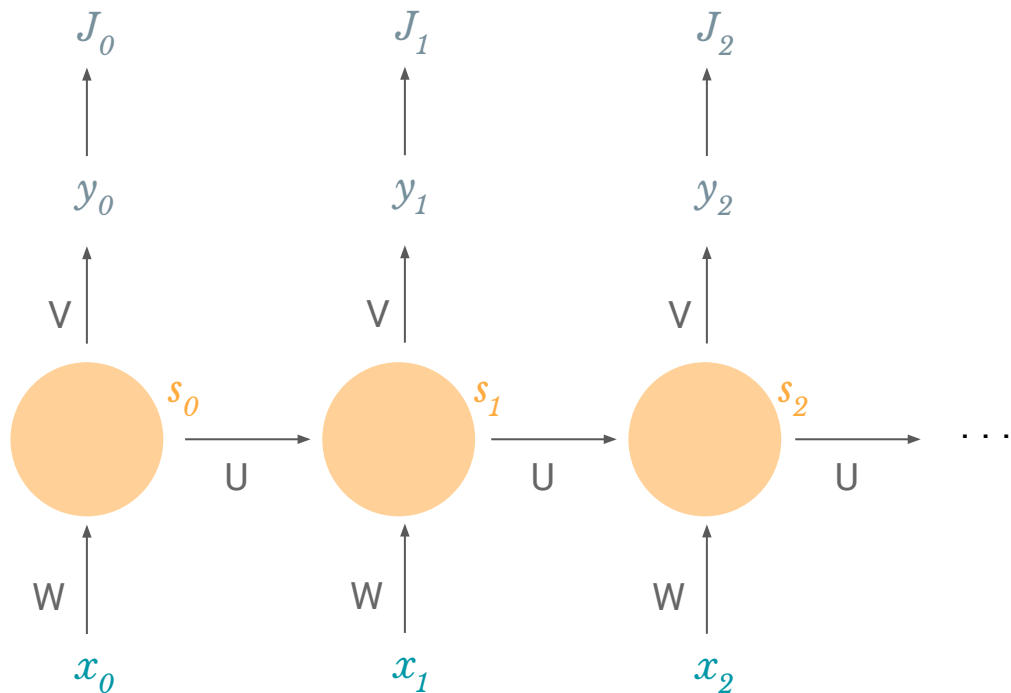
let's try it out for W with the **chain rule**:



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t :

let's try it out for W with the **chain rule**:

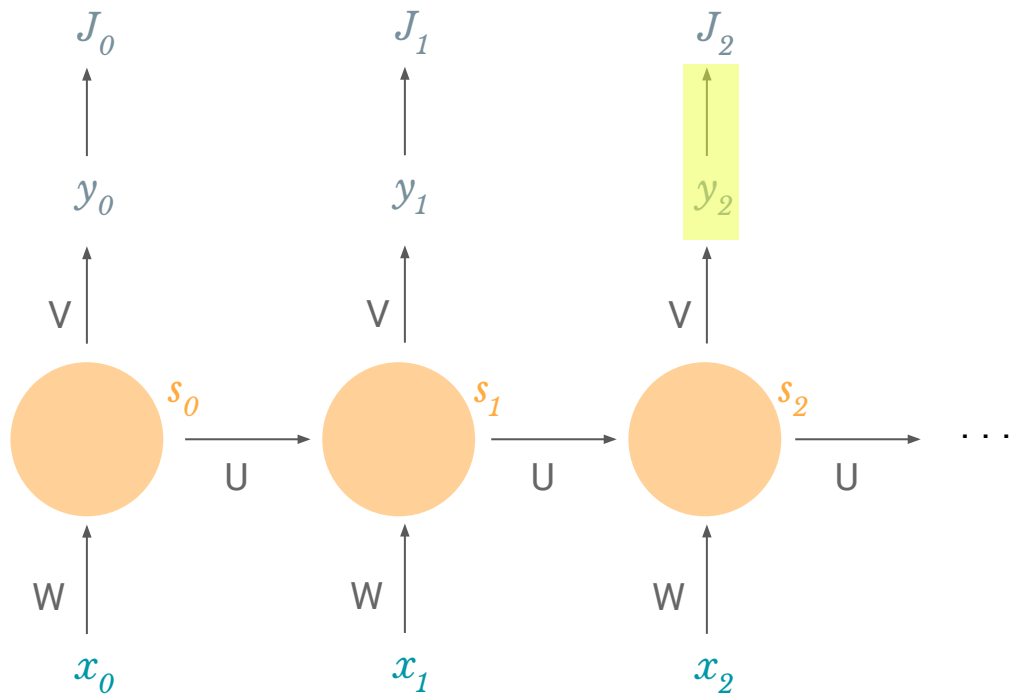


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t :

$$\frac{\partial J_2}{\partial W}$$

let's try it out for W with the **chain rule**:

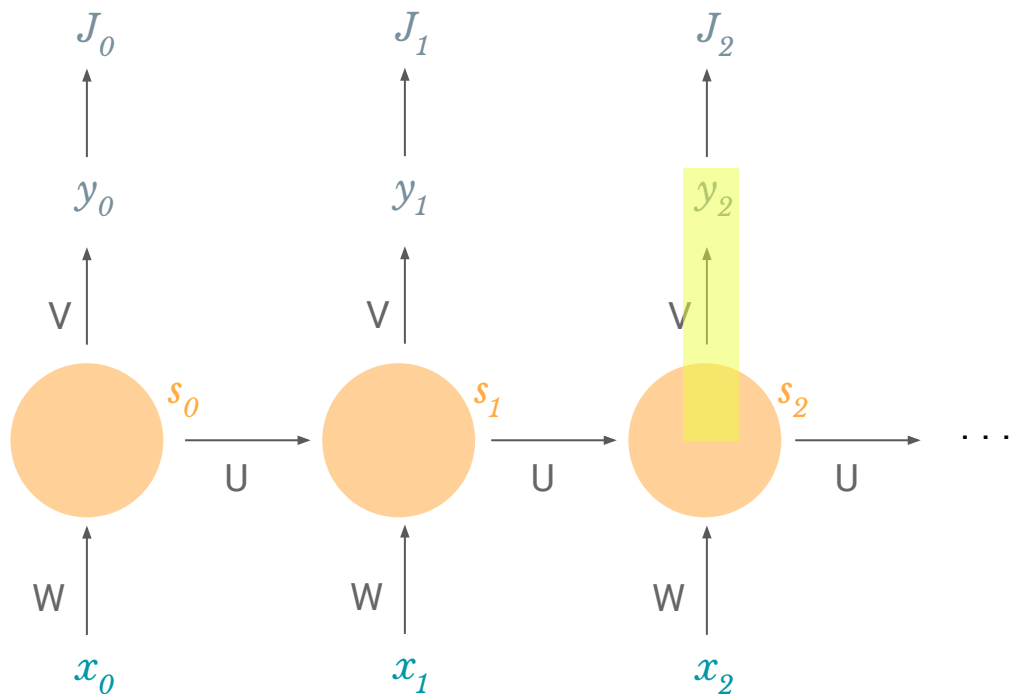


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t :

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}$$

let's try it out for W with the **chain rule**:

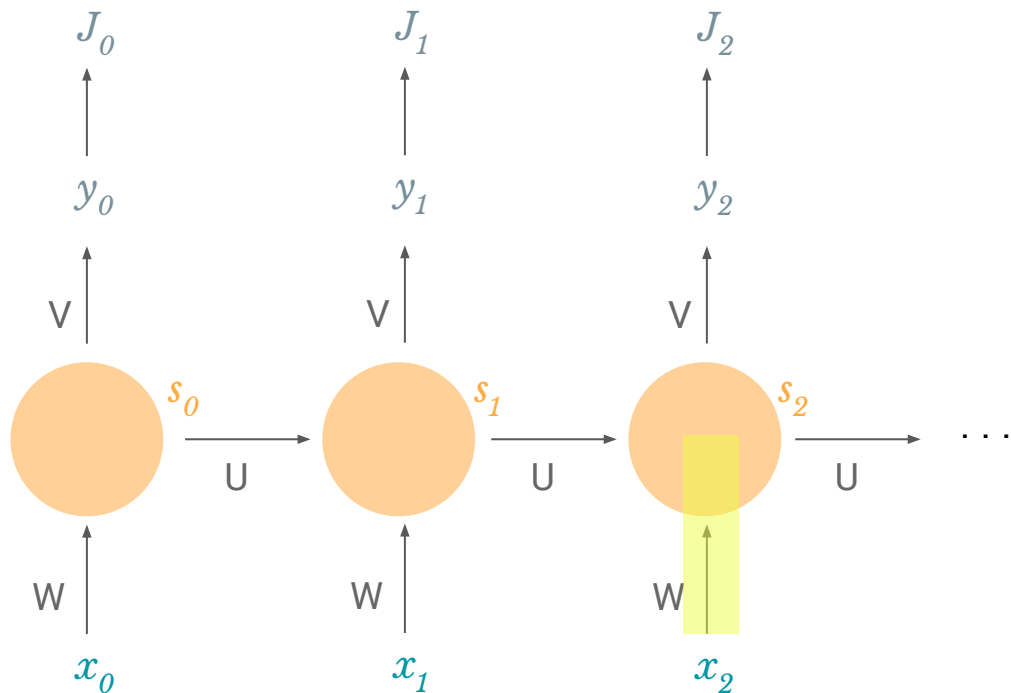


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t :

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2}$$

let's try it out for W with the **chain rule**:

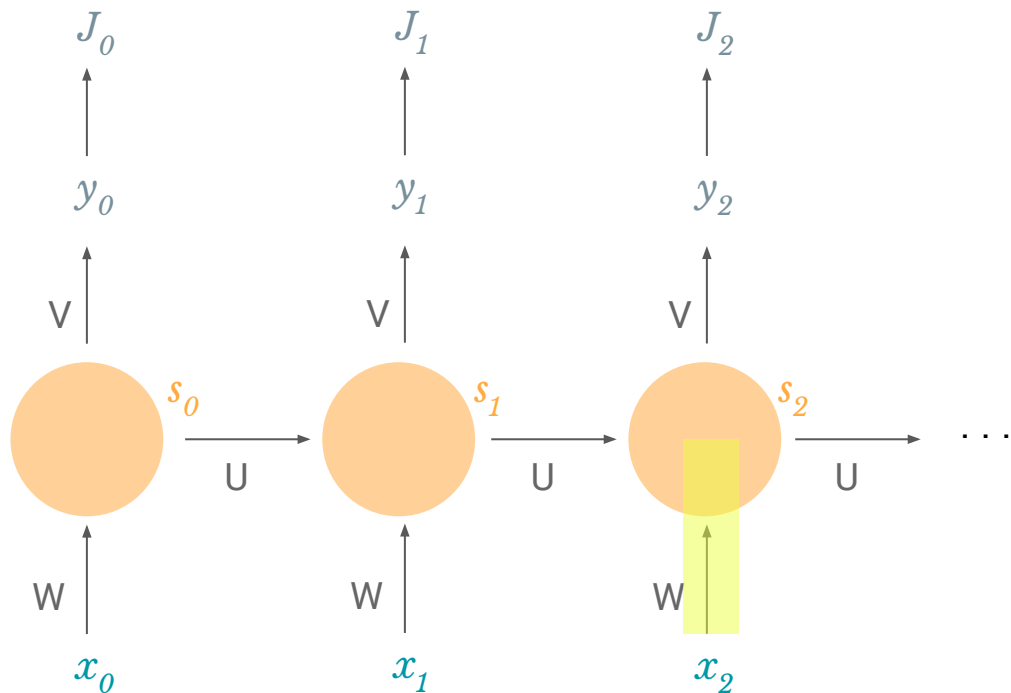


$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t :

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial W}$$

let's try it out for W with the **chain rule**:



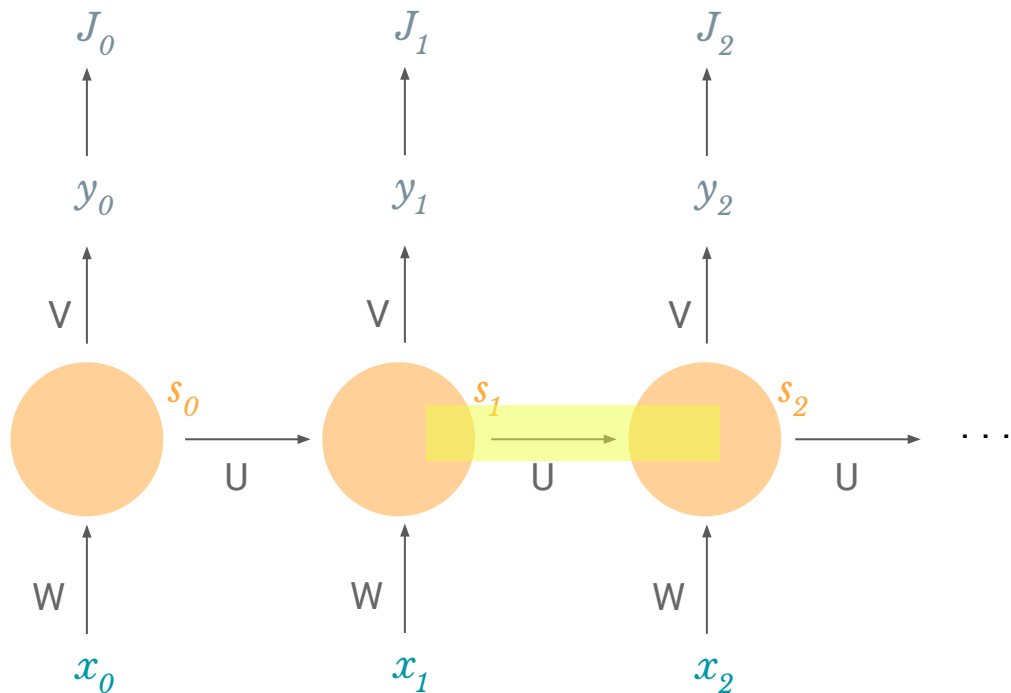
$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t :

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial W}$$

but wait...

let's try it out for W with the **chain rule**:



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

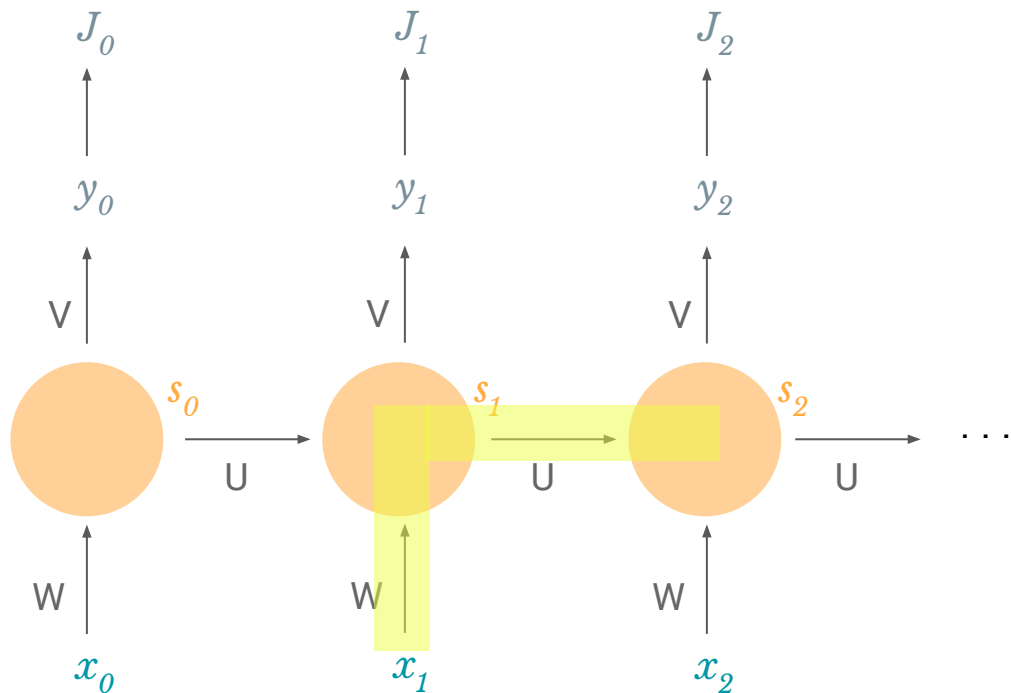
so let's take a single timestep t :

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial W}$$

but wait...

$$s_2 = \tanh(U s_1 + W x_2)$$

let's try it out for W with the **chain rule**:



$$\frac{\partial J}{\partial W} = \sum_t \frac{\partial J_t}{\partial W}$$

so let's take a single timestep t :

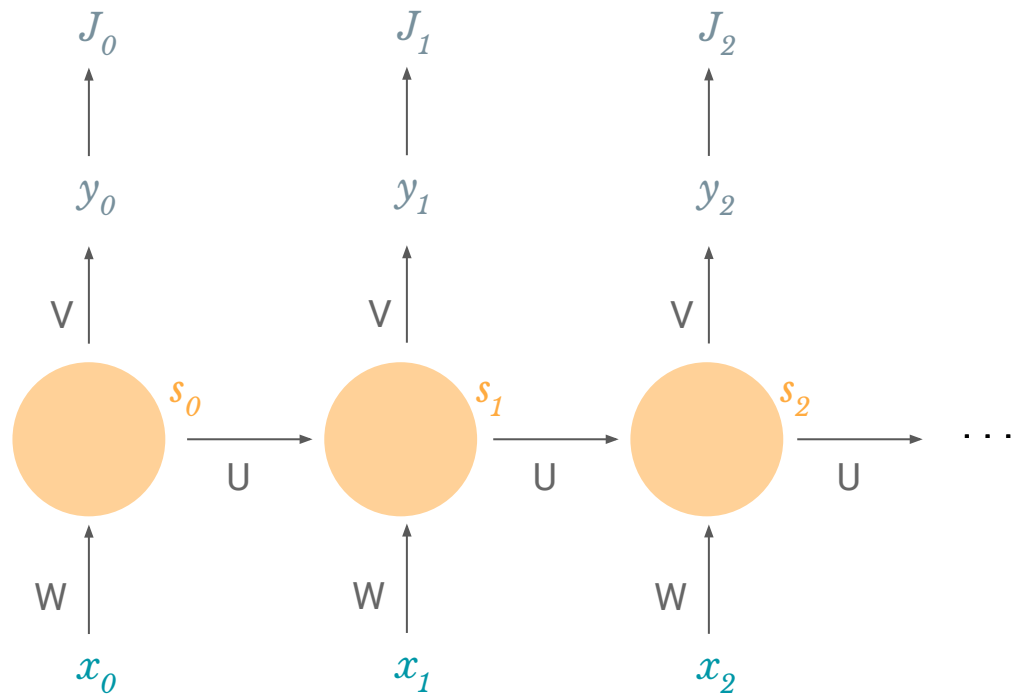
$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial W}$$

but wait...

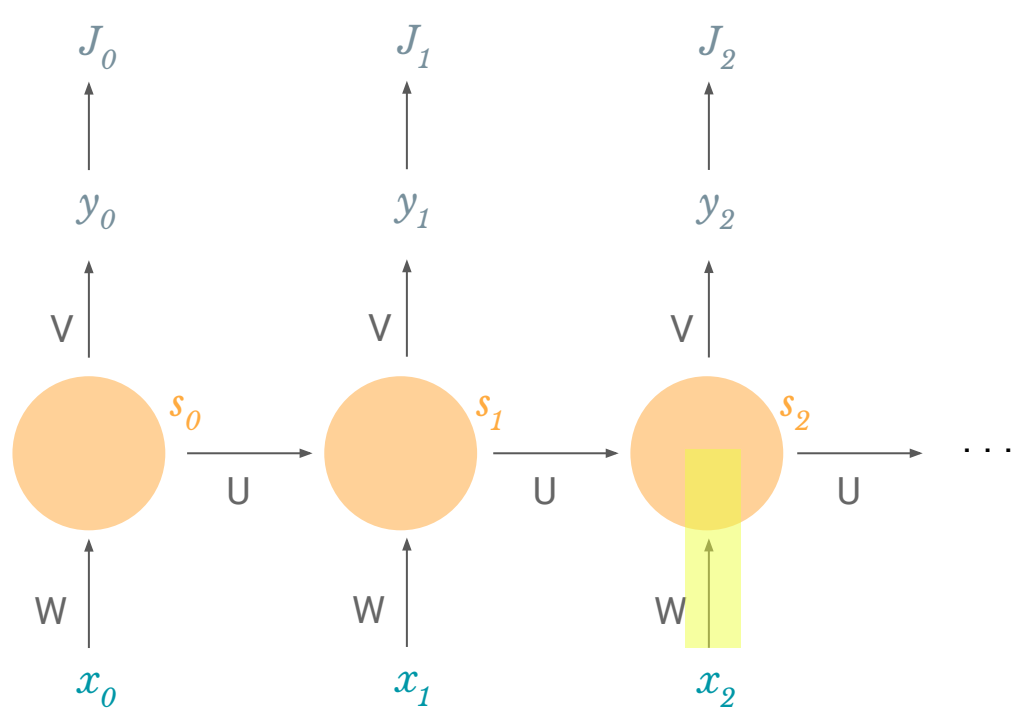
$$s_2 = \tanh(U s_1 + W x_2)$$

s_1 also depends on W so we can't just treat $\frac{\partial s_2}{\partial W}$ as a constant!

how does s_2 depend on W ?

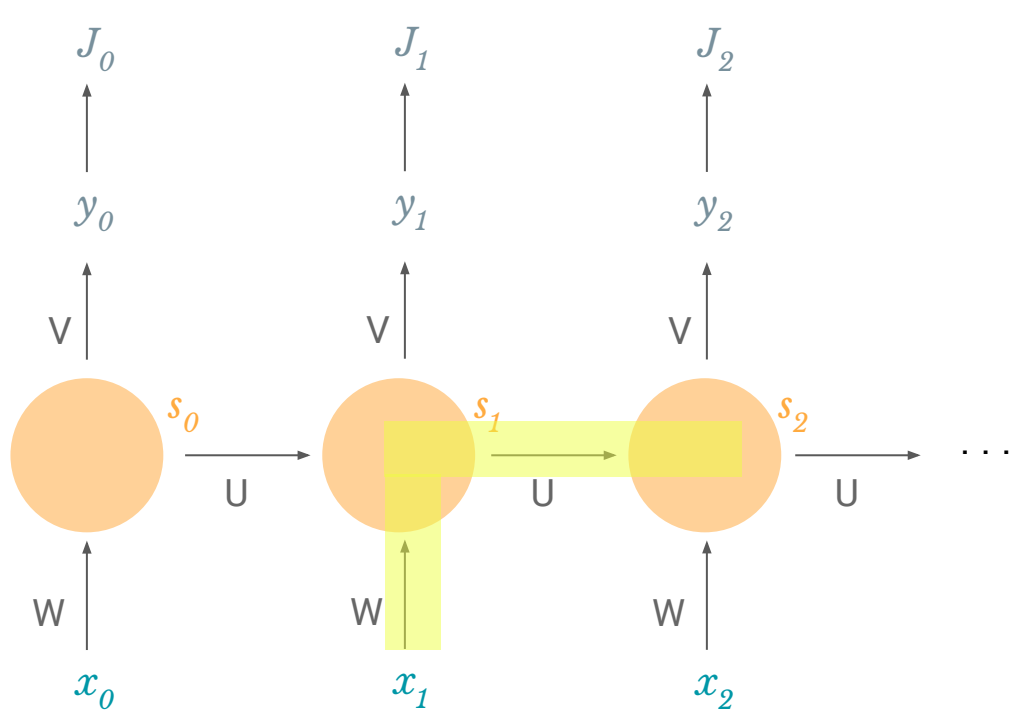


how does s_2 depend on W ?



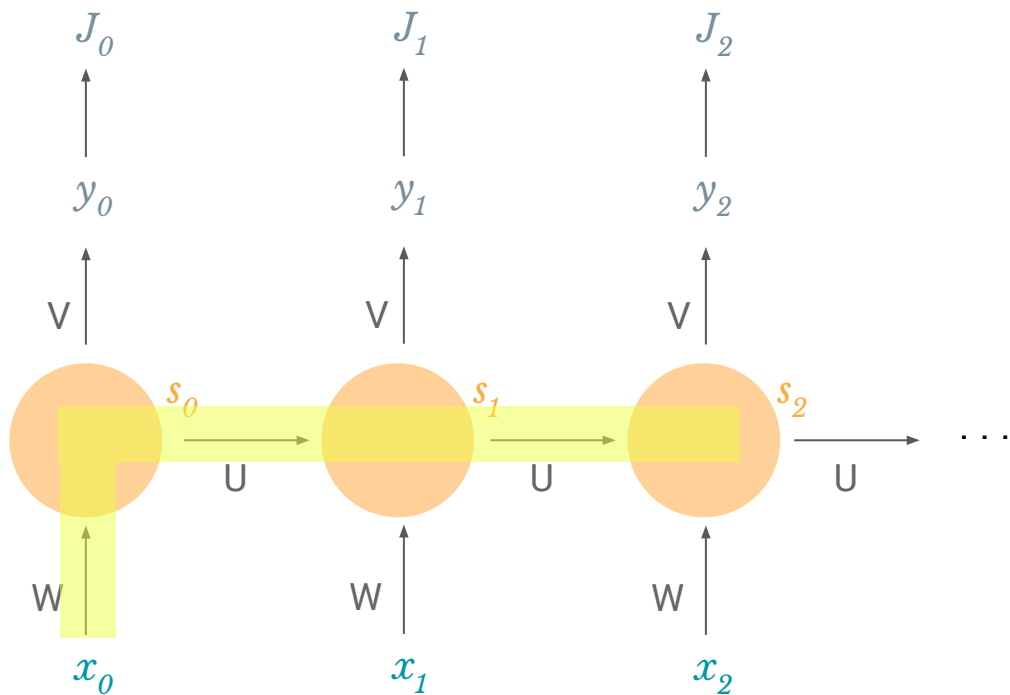
$$\frac{\partial s_2}{\partial W}$$

how does s_2 depend on W ?



$$\frac{\partial s_2}{\partial W} + \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W}$$

how does s_2 depend on W ?



$$\begin{aligned} & \frac{\partial s_2}{\partial W} \\ & + \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W} \\ & + \frac{\partial s_2}{\partial s_0} \frac{\partial s_0}{\partial W} \end{aligned}$$

backpropagation through time:

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^2 \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \underbrace{\frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}}_{\text{Contributions of } W \text{ in previous timesteps to the error at timestep } t}$$

Contributions of W in previous timesteps to the error at timestep t

backpropagation through time:

$$\frac{\partial J_t}{\partial W} = \sum_{k=0}^t \frac{\partial J_t}{\partial y_t} \frac{\partial y_t}{\partial s_t} \underbrace{\frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W}}_{\text{Contributions of } W \text{ in previous timesteps to the error at timestep } t}$$

Contributions of W in previous timesteps to the error at timestep t

why are RNNs **hard to train**?

problem: vanishing gradient

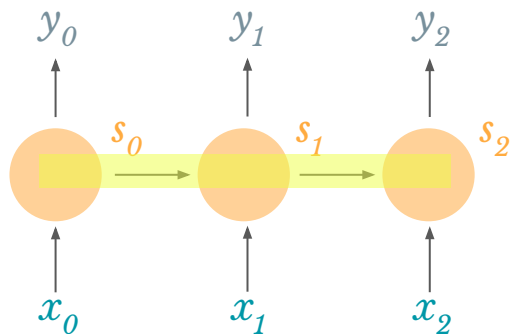
$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^2 \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}$$

problem: vanishing gradient

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^2 \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}$$

problem: vanishing gradient

$$\frac{\partial J_2}{\partial W} = \sum_{k=0}^2 \frac{\partial J_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W}$$



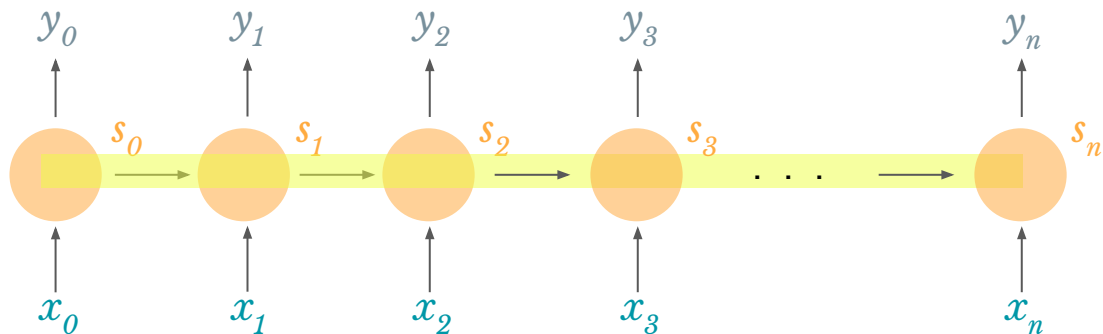
at $k = 0$:

$$\frac{\partial s_2}{\partial s_0} = \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

problem: vanishing gradient

$$\frac{\partial J_n}{\partial W} = \sum_{k=0}^n \frac{\partial J_n}{\partial y_n} \frac{\partial y_n}{\partial s_n} \frac{\partial s_n}{\partial s_k} \frac{\partial s_k}{\partial W}$$

$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$

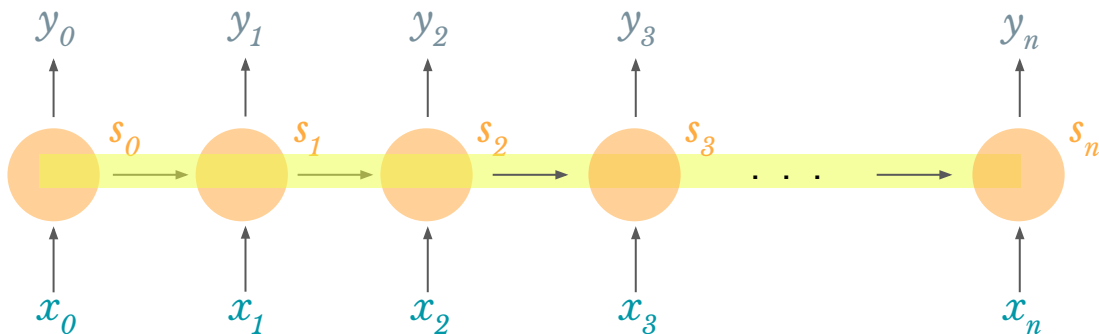


problem: vanishing gradient

$$\frac{\partial J_n}{\partial W} = \sum_{k=0}^n \frac{\partial J_n}{\partial y_n} \frac{\partial y_n}{\partial s_n} \frac{\partial s_n}{\partial s_k} \frac{\partial s_k}{\partial W}$$

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

as the gap between timesteps gets bigger, this product gets longer and longer!



problem: vanishing gradient

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdot \dots \cdot \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

problem: vanishing gradient

what are each of these terms?



$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdot \cdot \cdot \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

problem: vanishing gradient

what are each of these terms? →

$$\frac{\partial s_n}{\partial s_{n-1}} = W^T \text{diag}[f'(W s_{j-1} + U x_j)]$$

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

W = sampled from
standard normal
distribution = mostly < 1

f = tanh or sigmoid so $f' < 1$

problem: vanishing gradient

what are each of these terms? →

$$\frac{\partial s_n}{\partial s_{n-1}} = W^T \text{diag}[f'(W s_{j-1} + U x_j)]$$

$$\frac{\partial s_n}{\partial s_{n-1}} \frac{\partial s_{n-1}}{\partial s_{n-2}} \cdots \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

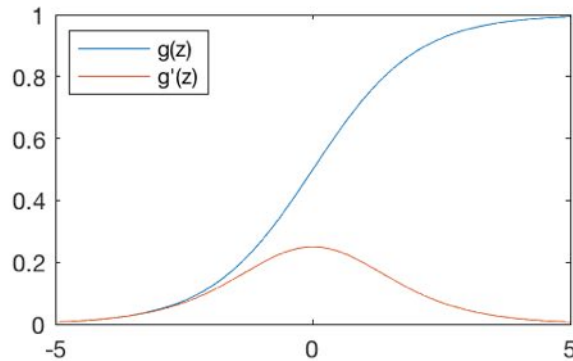
W = sampled from
standard normal
distribution = mostly < 1

f = tanh or sigmoid so $f' < 1$

we're multiplying a lot of **small numbers** together.

Common Activation Functions

Sigmoid Function

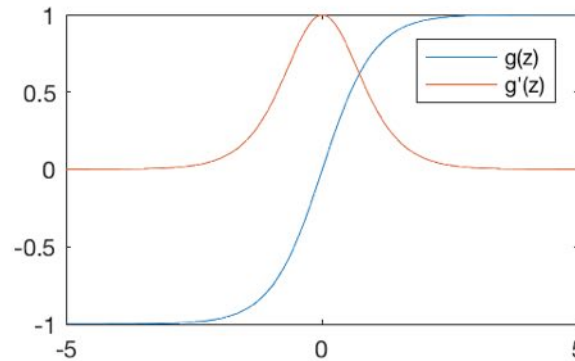


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

 `tf.nn.sigmoid(z)`

Hyperbolic Tangent

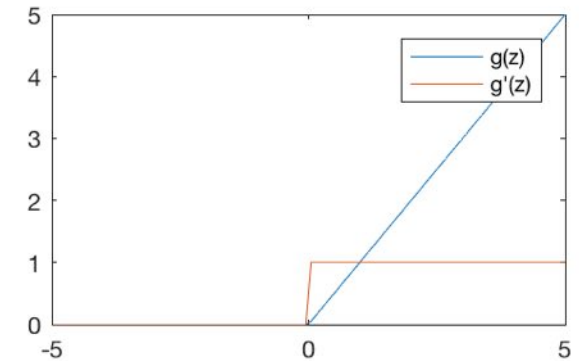


$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

 `tf.nn.tanh(z)`

Rectified Linear Unit (ReLU)



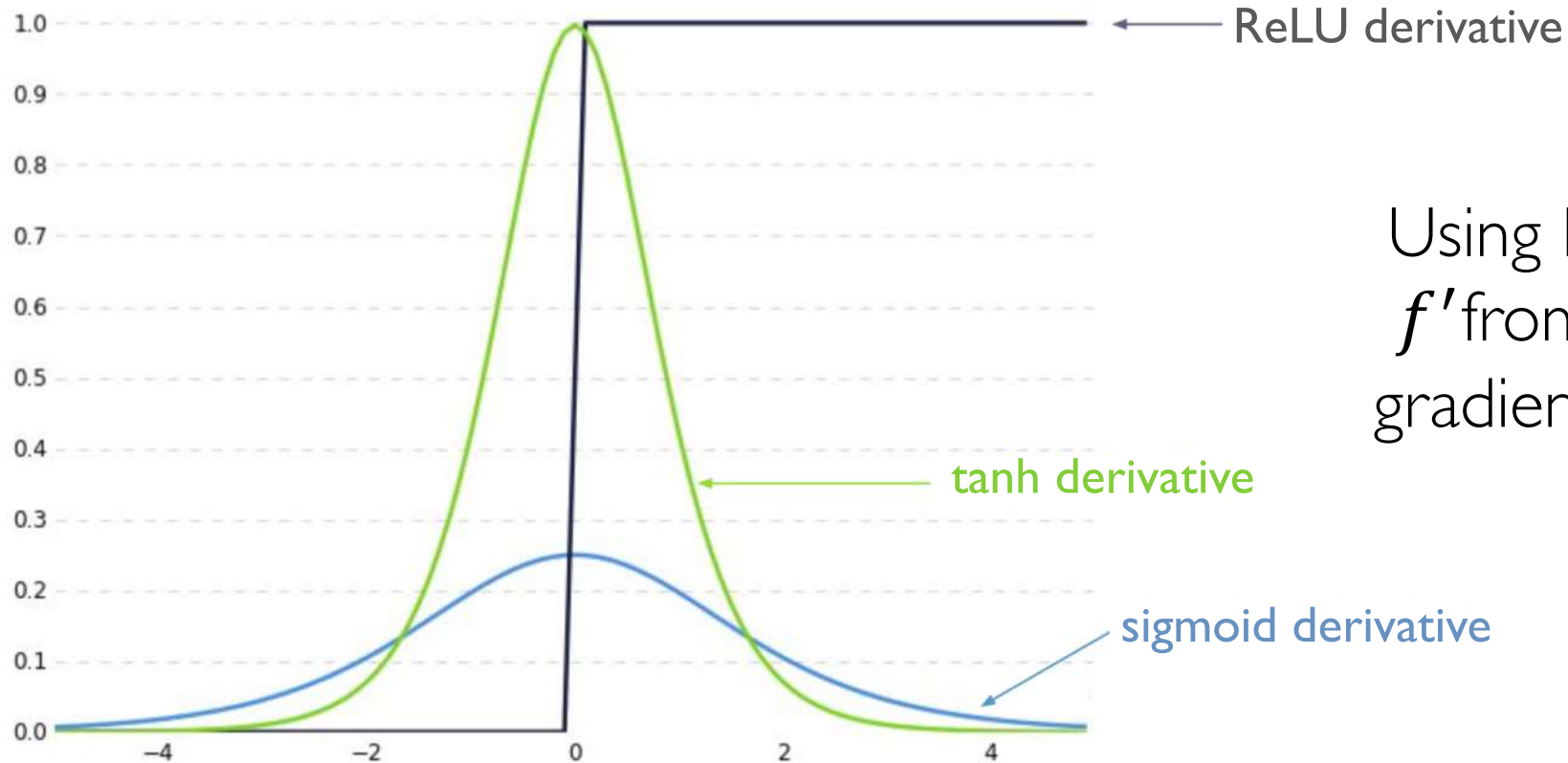
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

 `tf.nn.relu(z)`

NOTE: All activation functions are **non-linear**

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

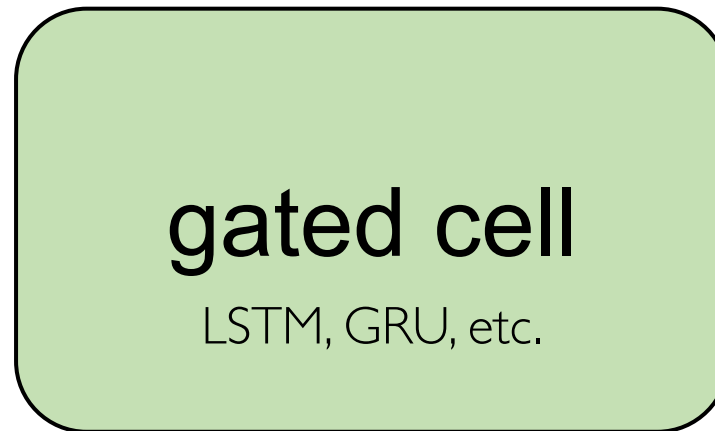
Initialize **biases** to zero

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

This helps prevent the weights from shrinking to zero.

Solution #3: gated cells

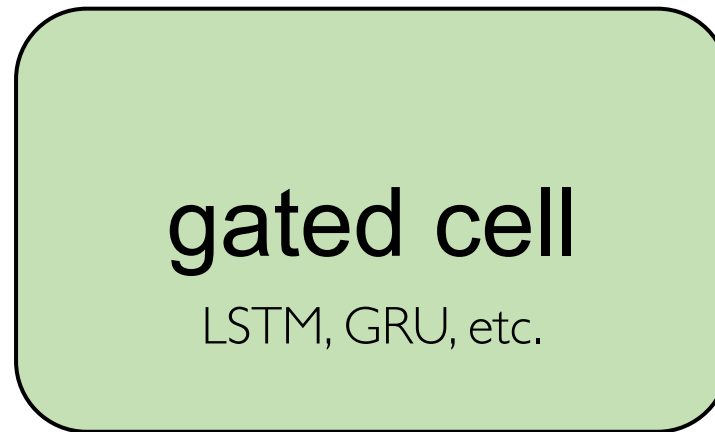
Idea: use a more **complex recurrent unit with gates** to control what information is passed through



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



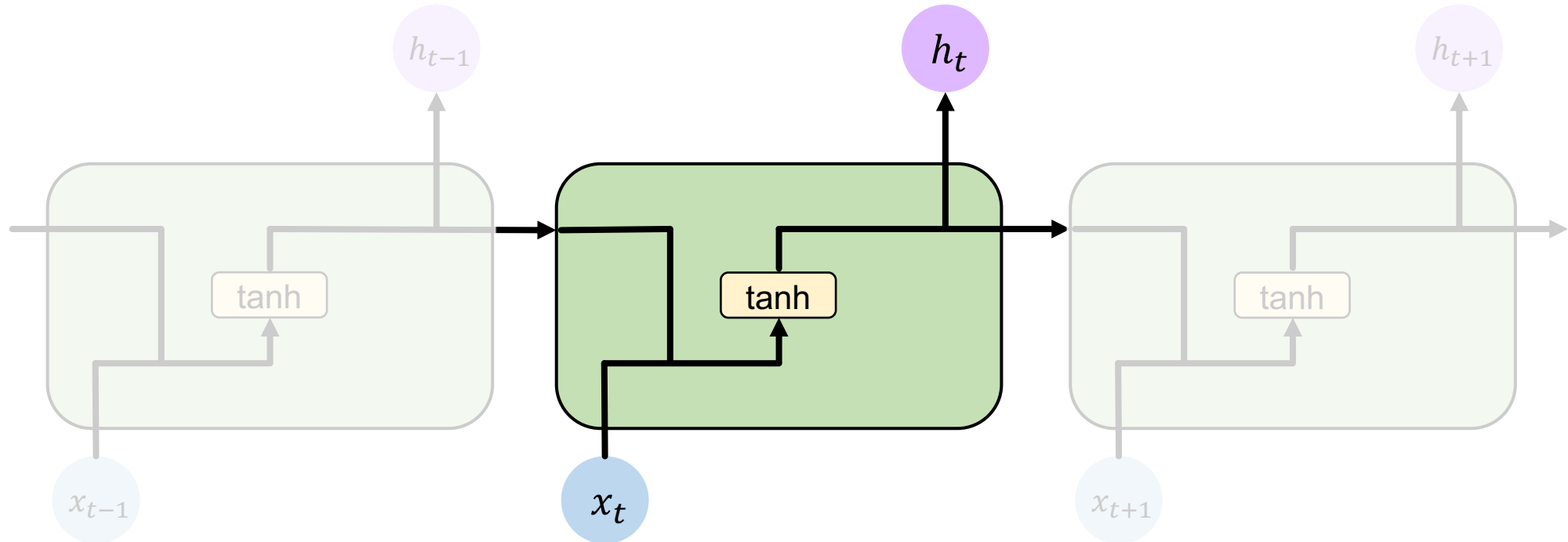
Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

Adapted from H. Suresh, 6.S191 2018

Long Short Term Memory (LSTM) Networks

Standard RNN

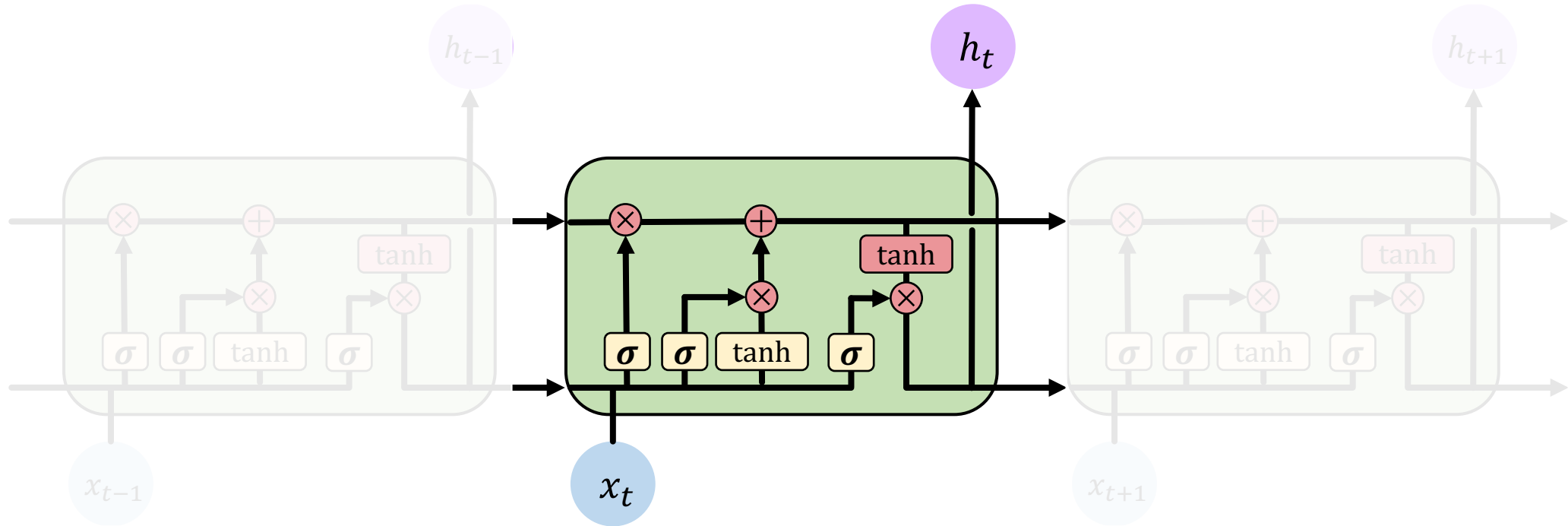
In a standard RNN, repeating modules contain a **simple computation node**



[2]

Long Short Term Memory (LSTMs)

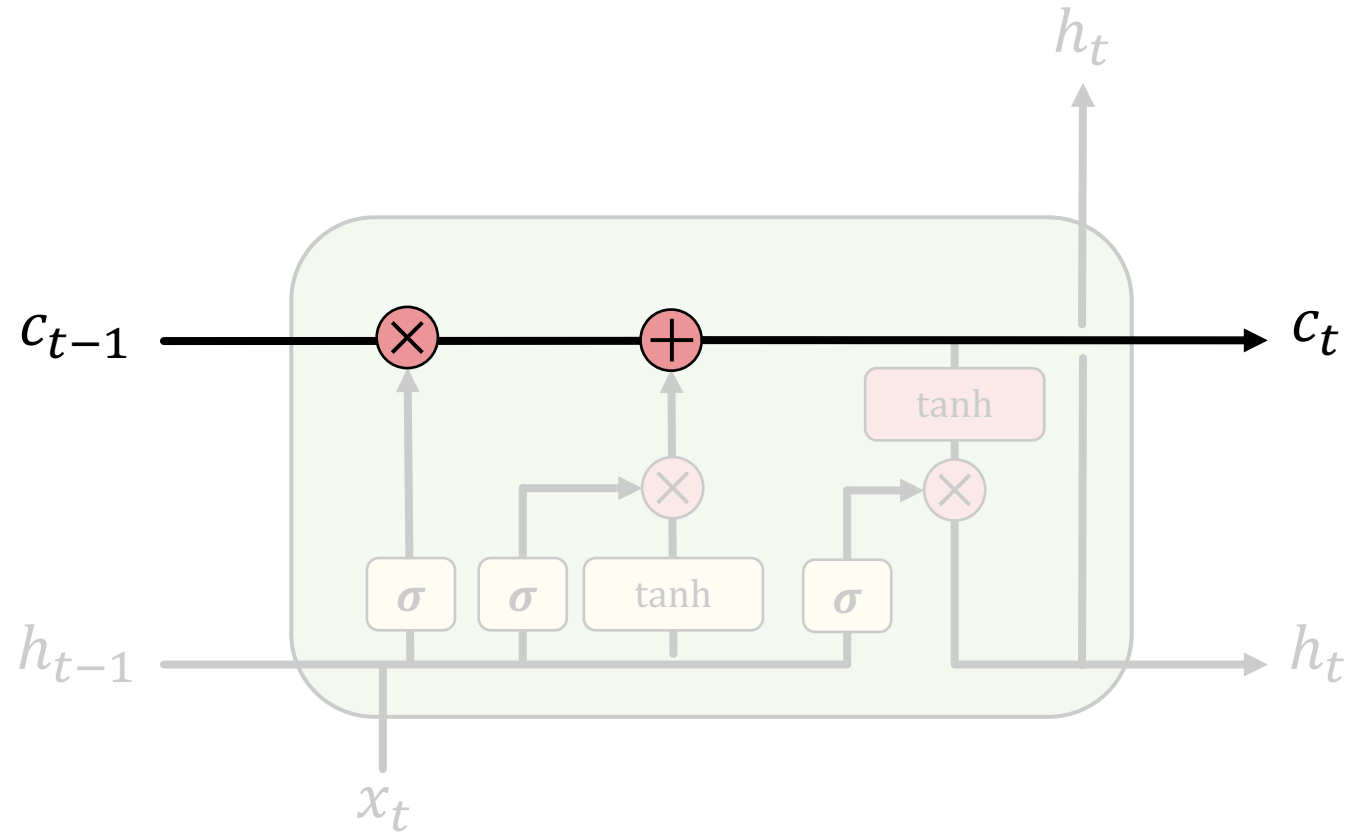
LSTM repeating modules contain **interacting layers** that **control information flow**



LSTM cells are able to track information throughout many timesteps

Long Short Term Memory (LSTMs)

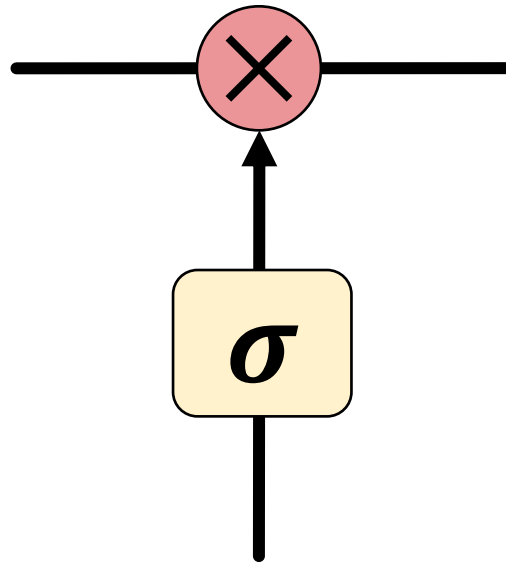
LSTMs maintain a **cell state** c_t where it's easy for information to flow



[2, 5]

Long Short Term Memory (LSTMs)

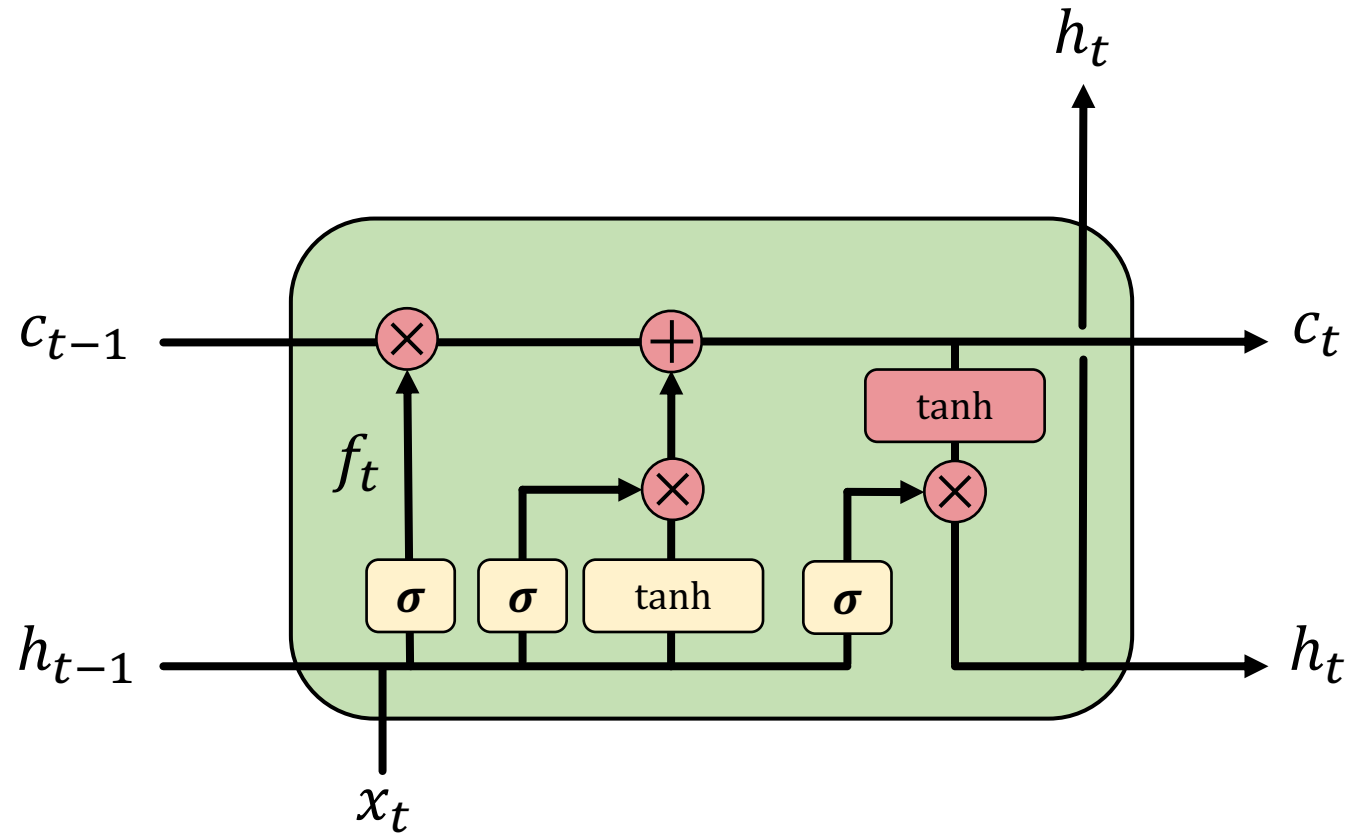
Information is **added** or **removed** to cell state through structures called **gates**



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

Long Short Term Memory (LSTMs)

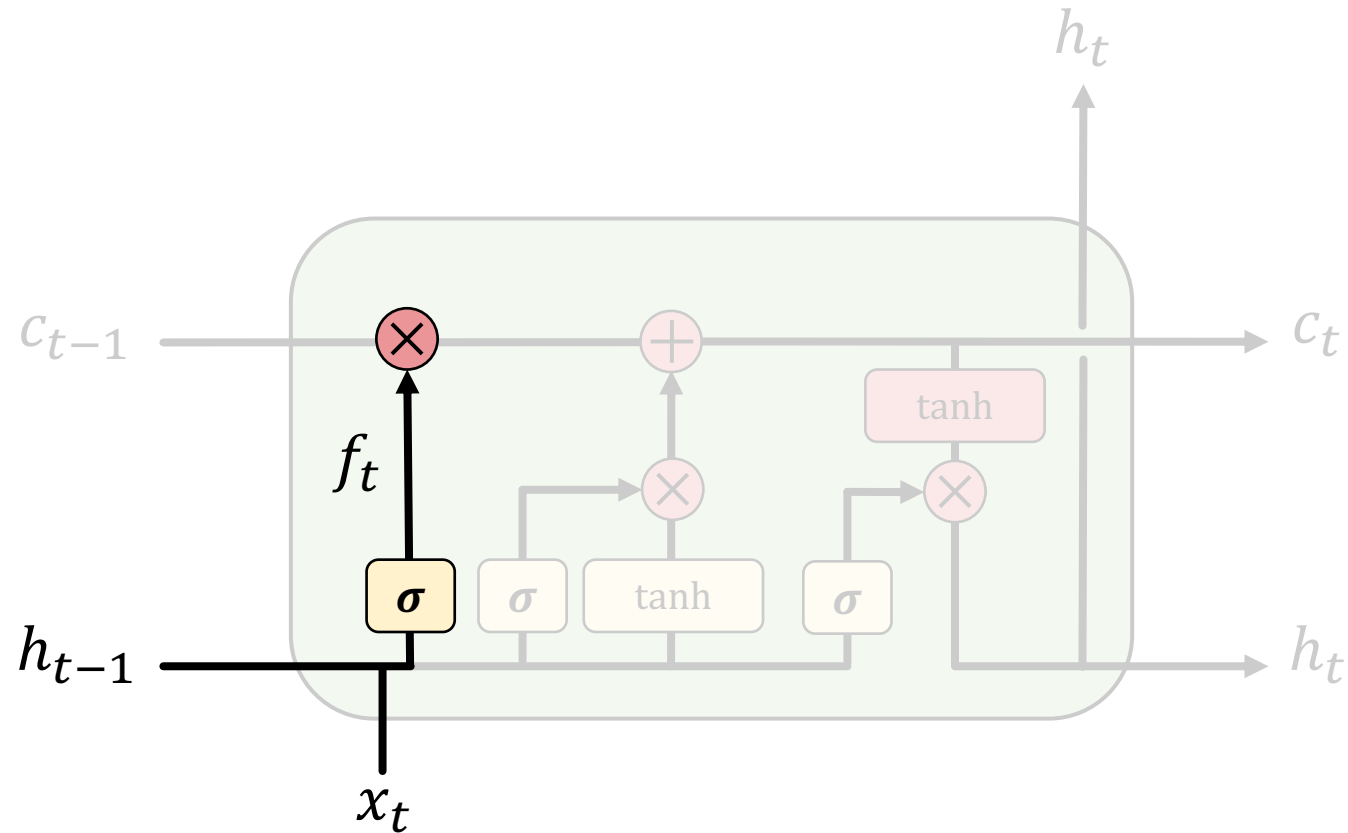
How do LSTMs work?



[2, 5]

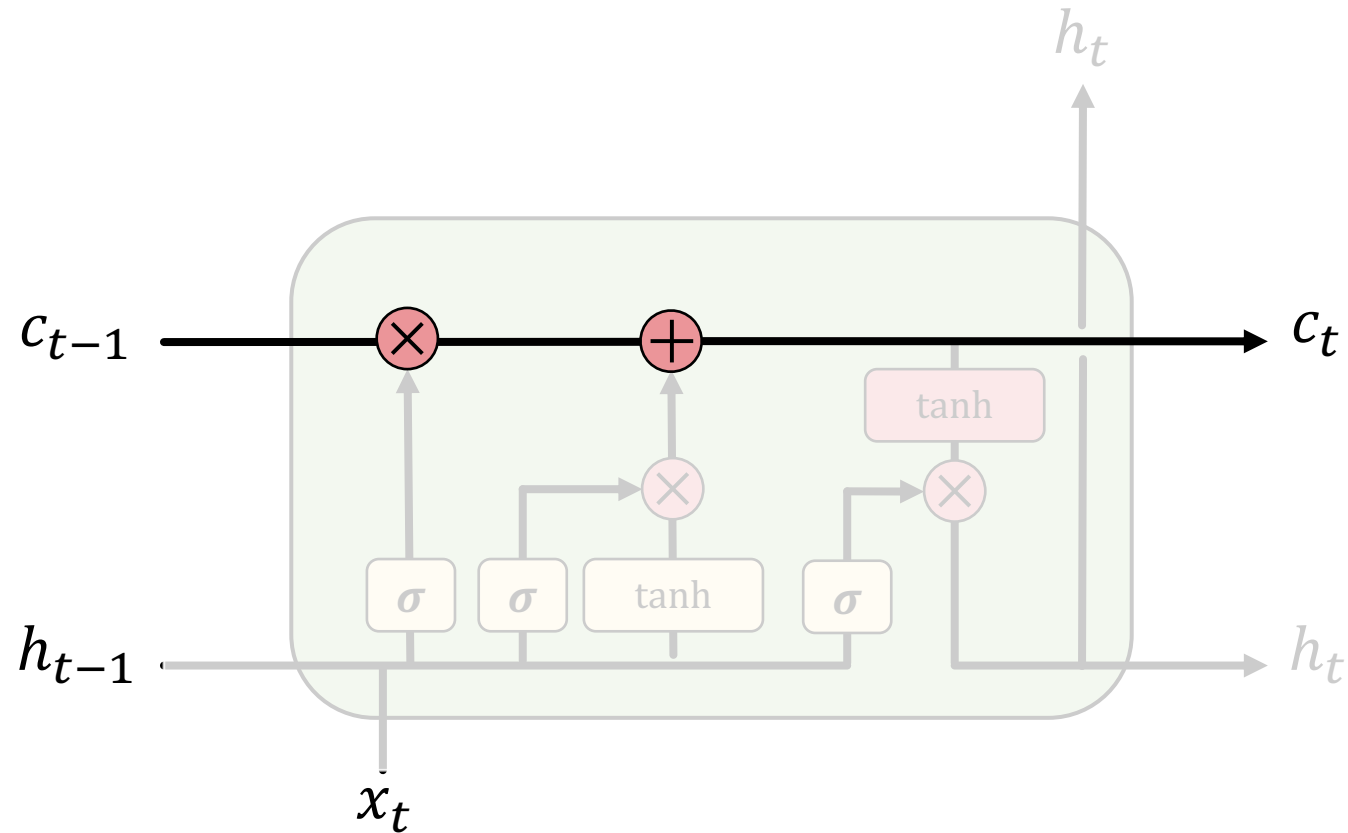
Long Short Term Memory (LSTMs)

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



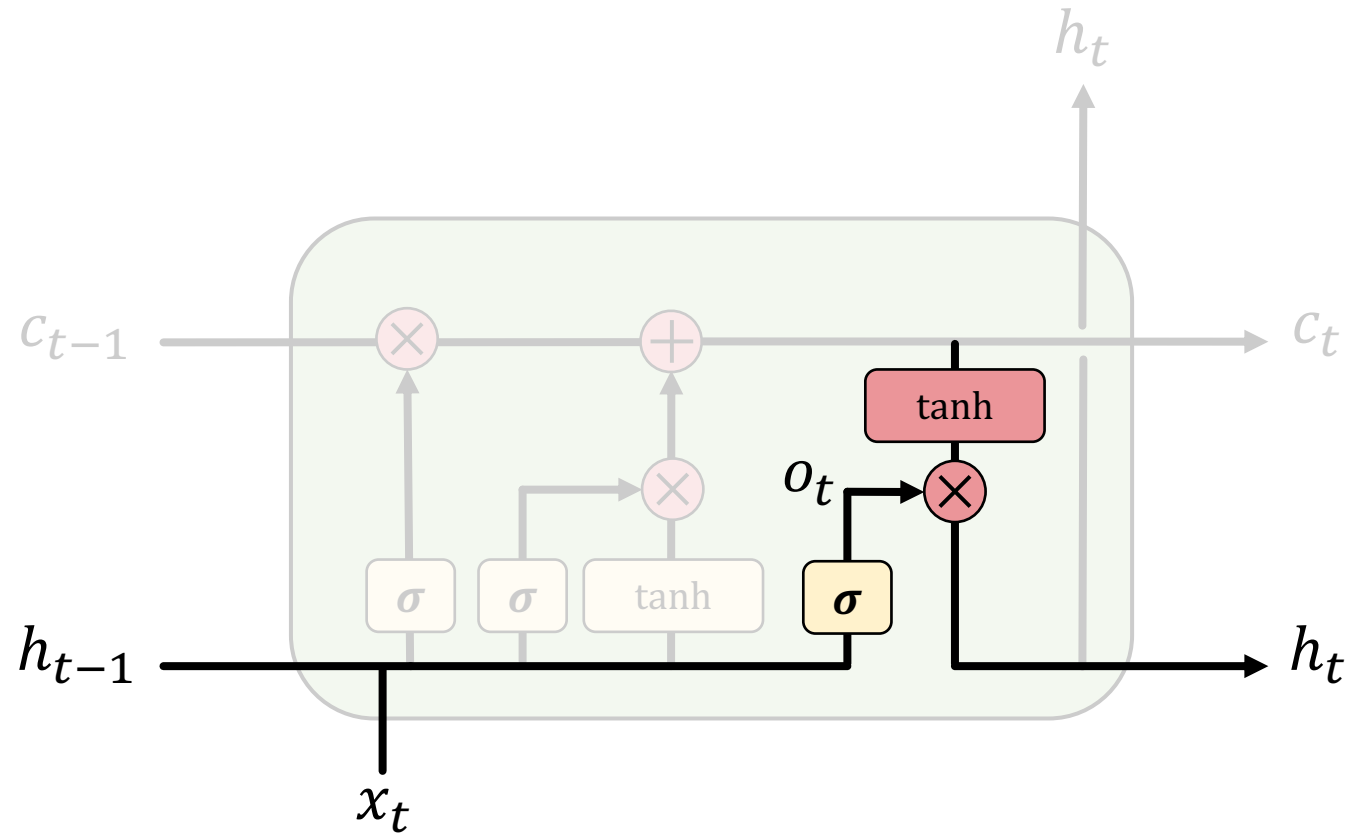
Long Short Term Memory (LSTMs)

LSTMs **selectively update** cell state values



Long Short Term Memory (LSTMs)

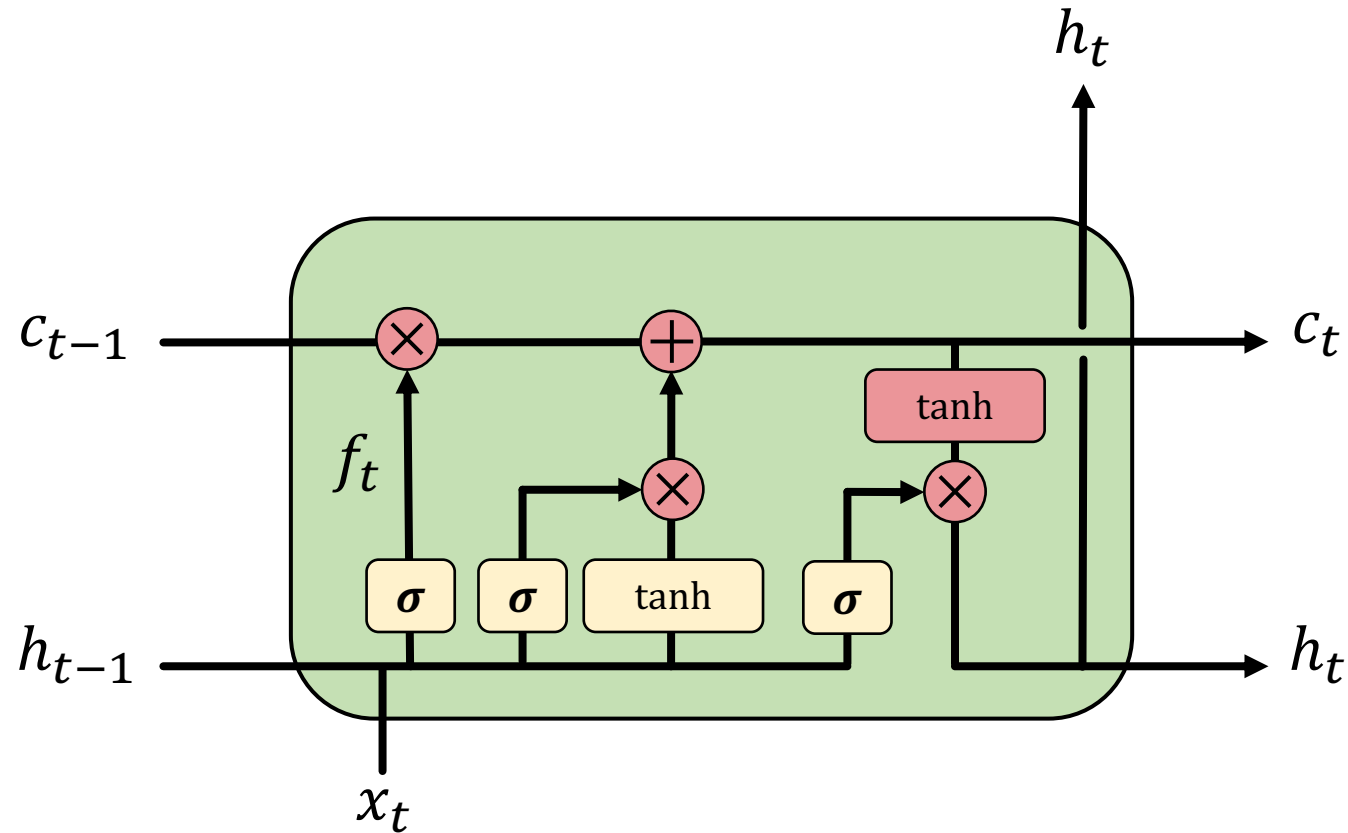
LSTMs use an **output gate** to output certain parts of the cell state



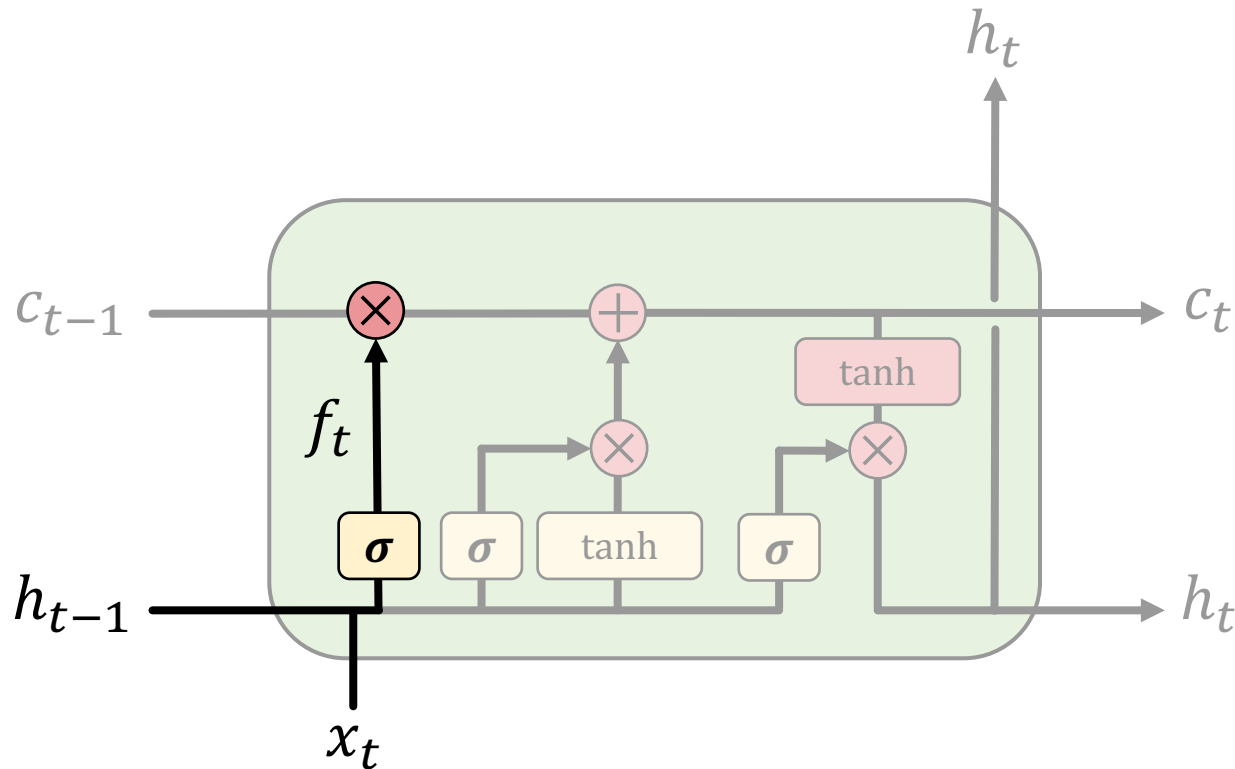
Long Short Term Memory (LSTMs)

How do LSTMs work?

1) Forget 2) Update 3) Output



LSTMs: forget irrelevant information

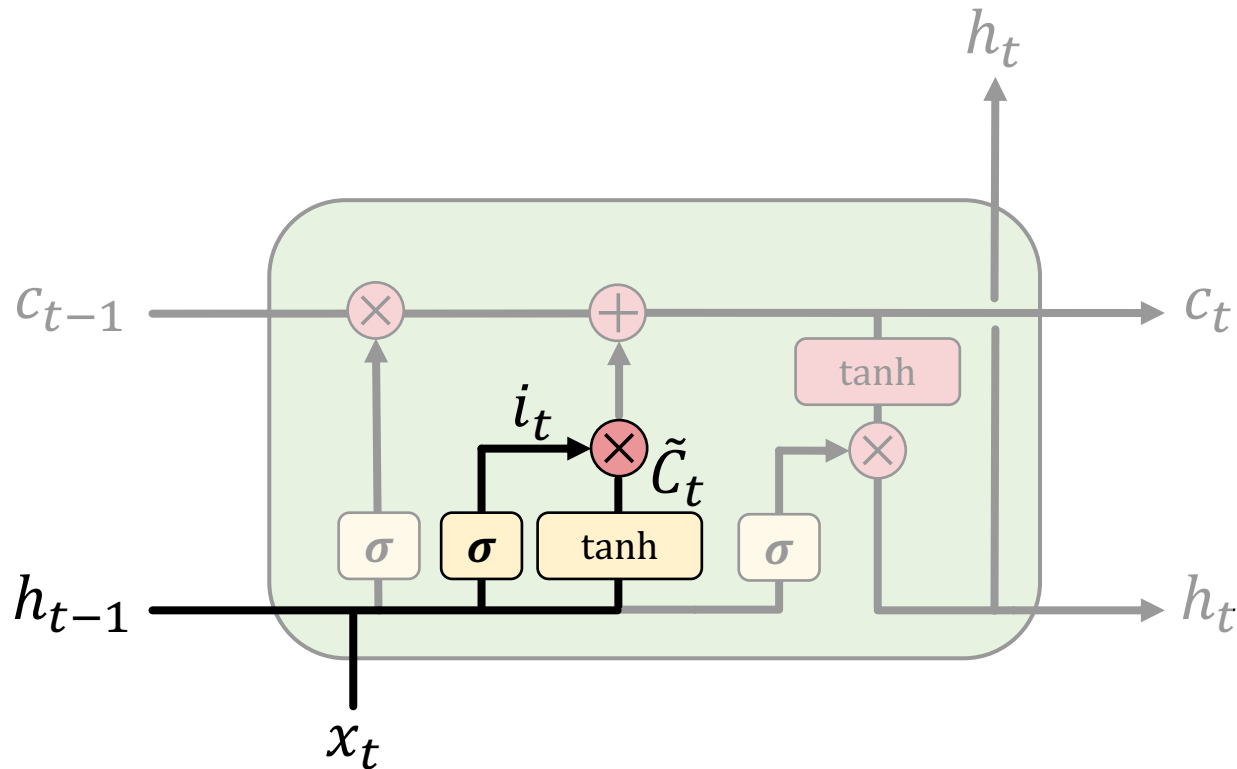


$$f_t = \sigma(\mathbf{W}_i[h_{t-1}, x_t] + b_f)$$

- 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

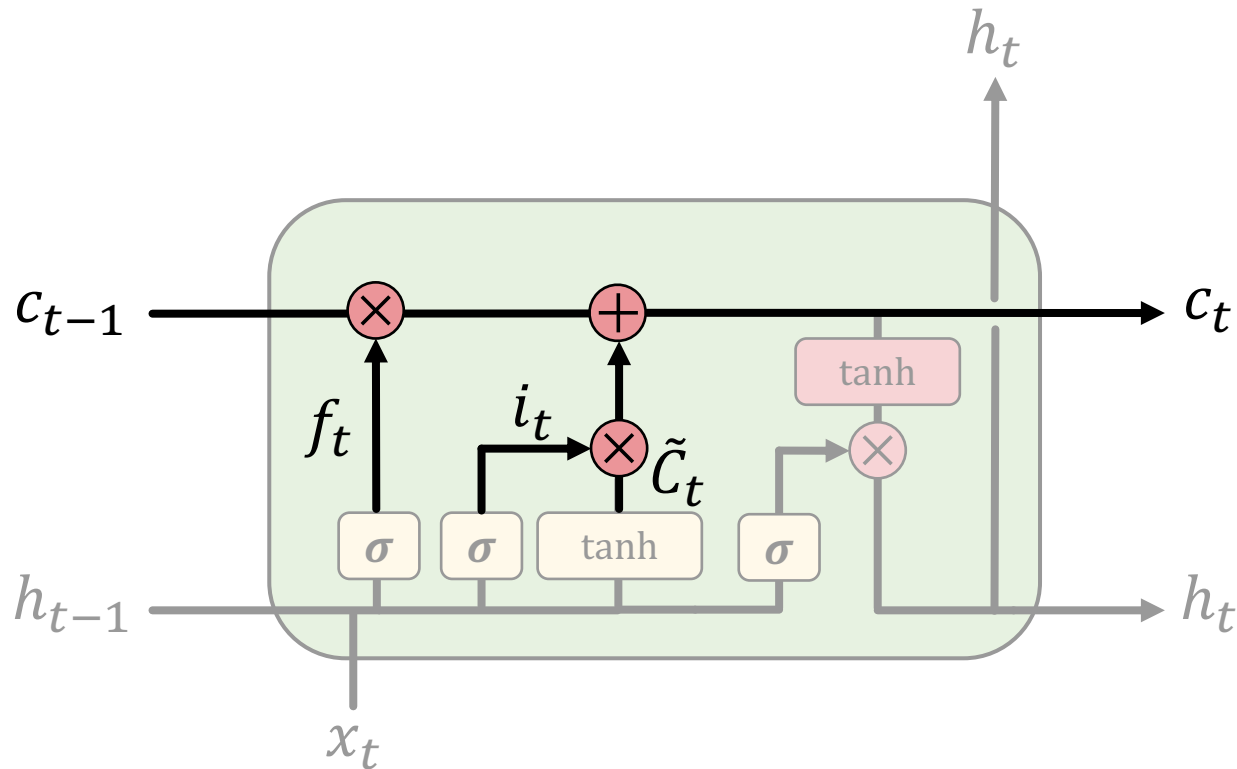


$$i_t = \sigma(\mathbf{W}_i [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(\mathbf{W}_c [h_{t-1}, x_t] + b_c)$$

- 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

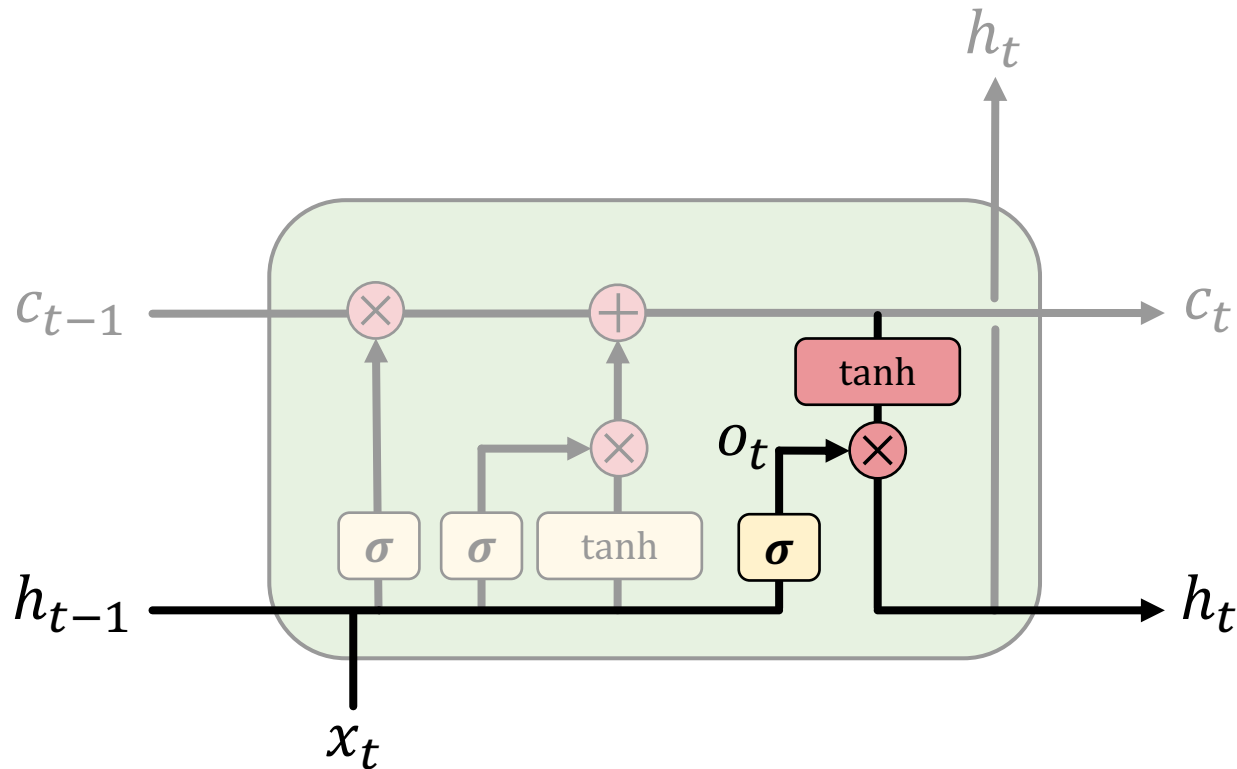


$$c_t = f_t * c_{t-1} + i_t * \tilde{C}_t$$

- 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



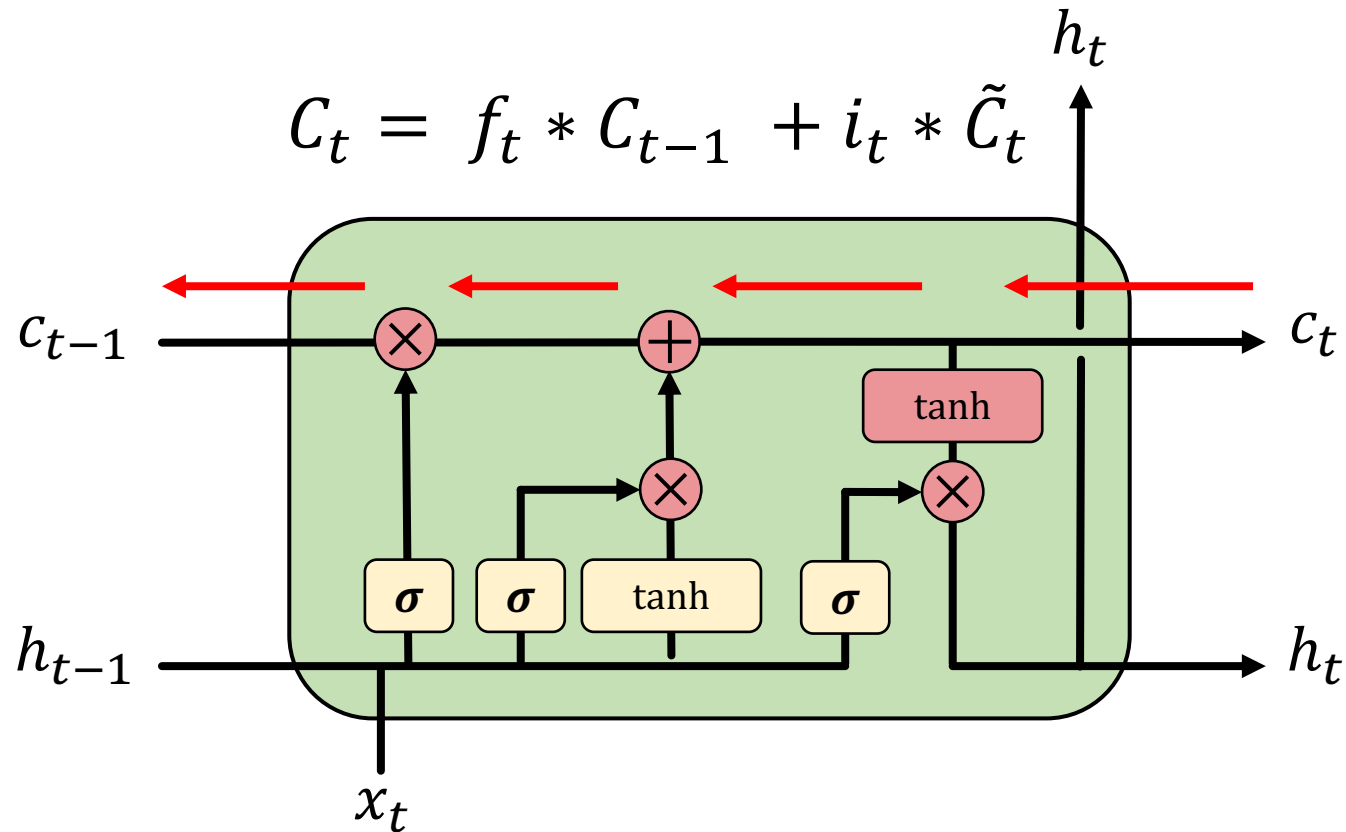
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

- 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 \rightarrow avoid vanishing gradient problem.



[2, 5]

LSTMs: key concepts

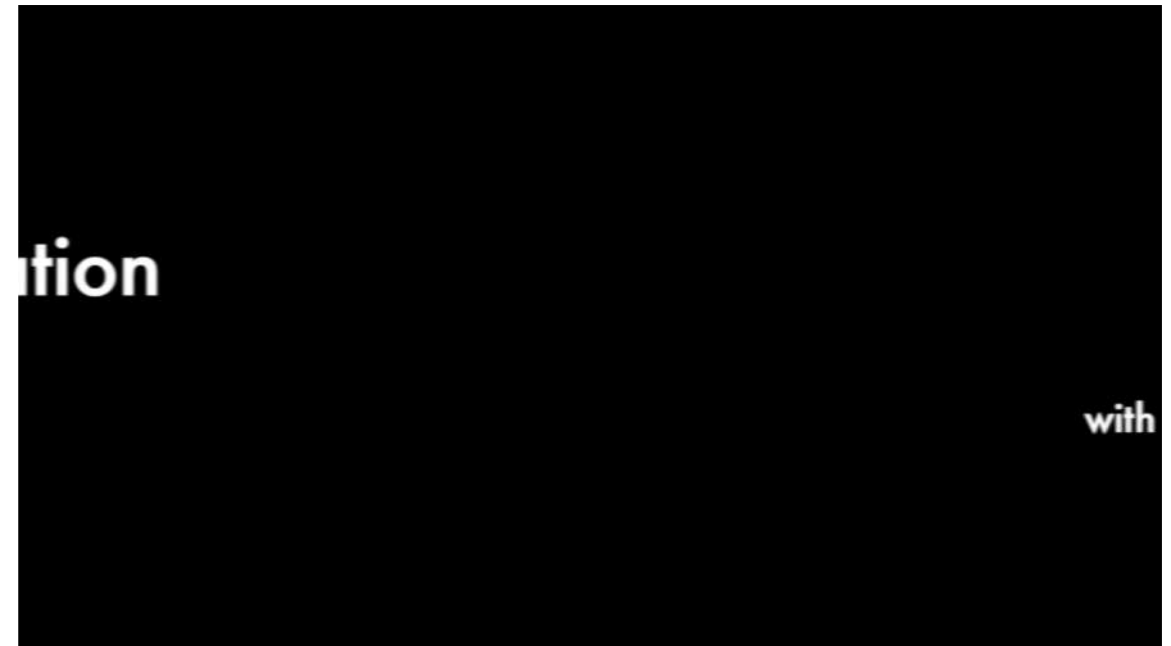
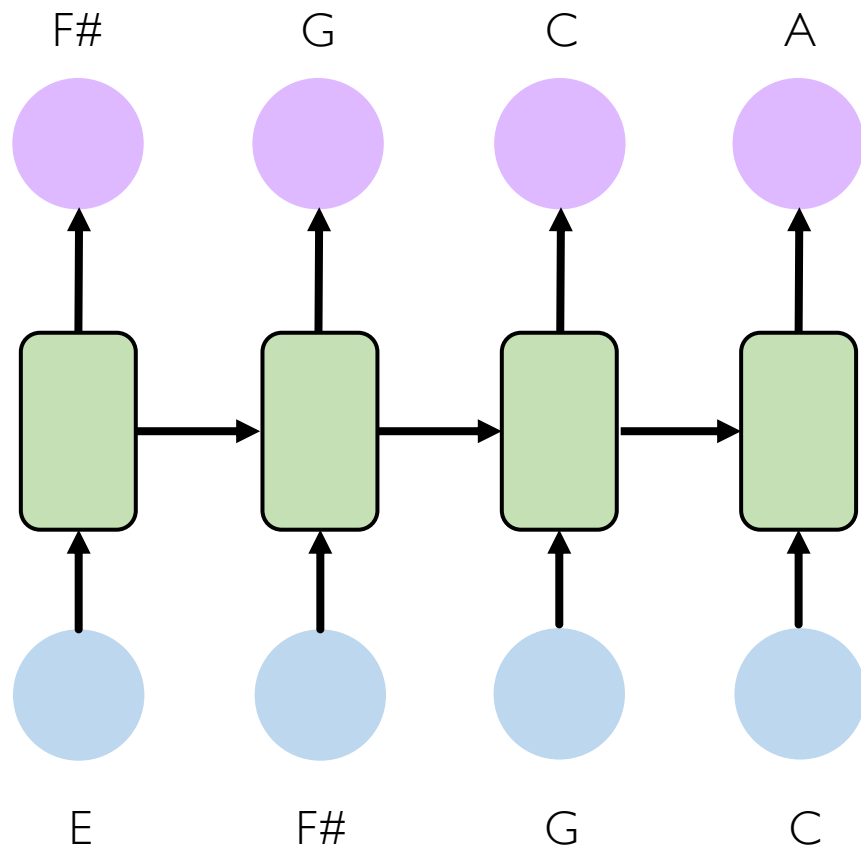
1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
 - Forget gate gets rid of irrelevant information
 - Selectively update cell state
 - Output gate returns a filtered version of the cell state
3. Backpropagation from c_t to c_{t-1} doesn't require matrix multiplication:
uninterrupted gradient flow

RNN Applications

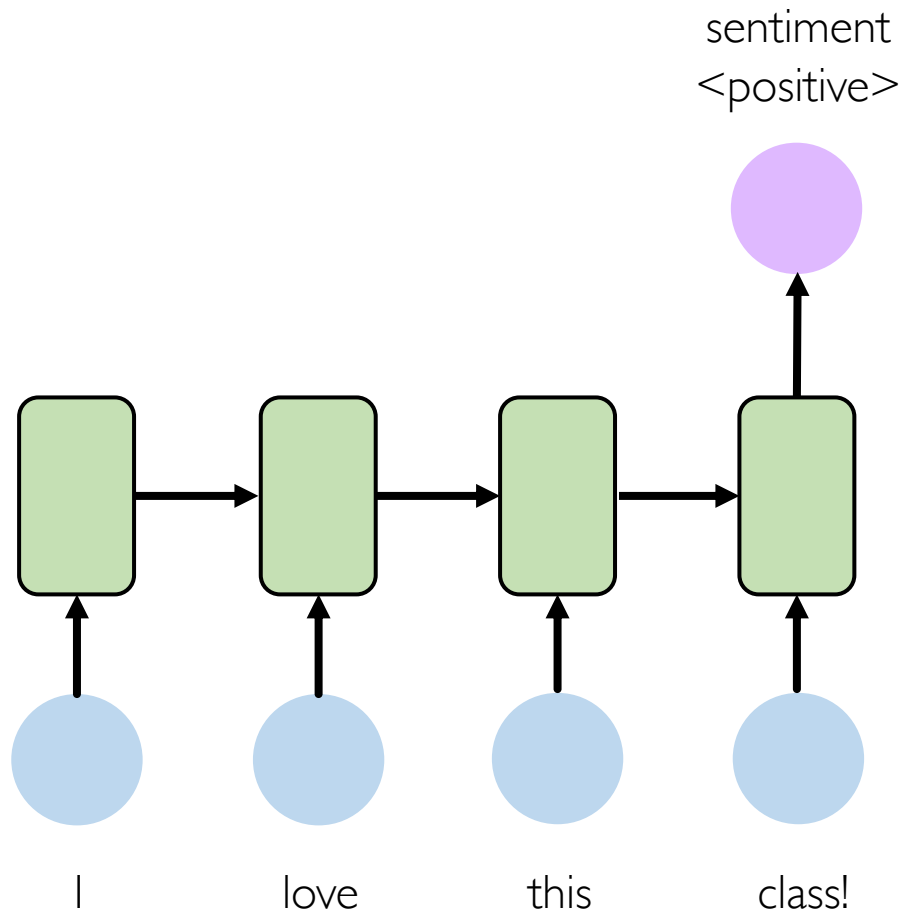
Example task: music generation

Input: sheet music

Output: next character in sheet music




Example task: sentiment classification

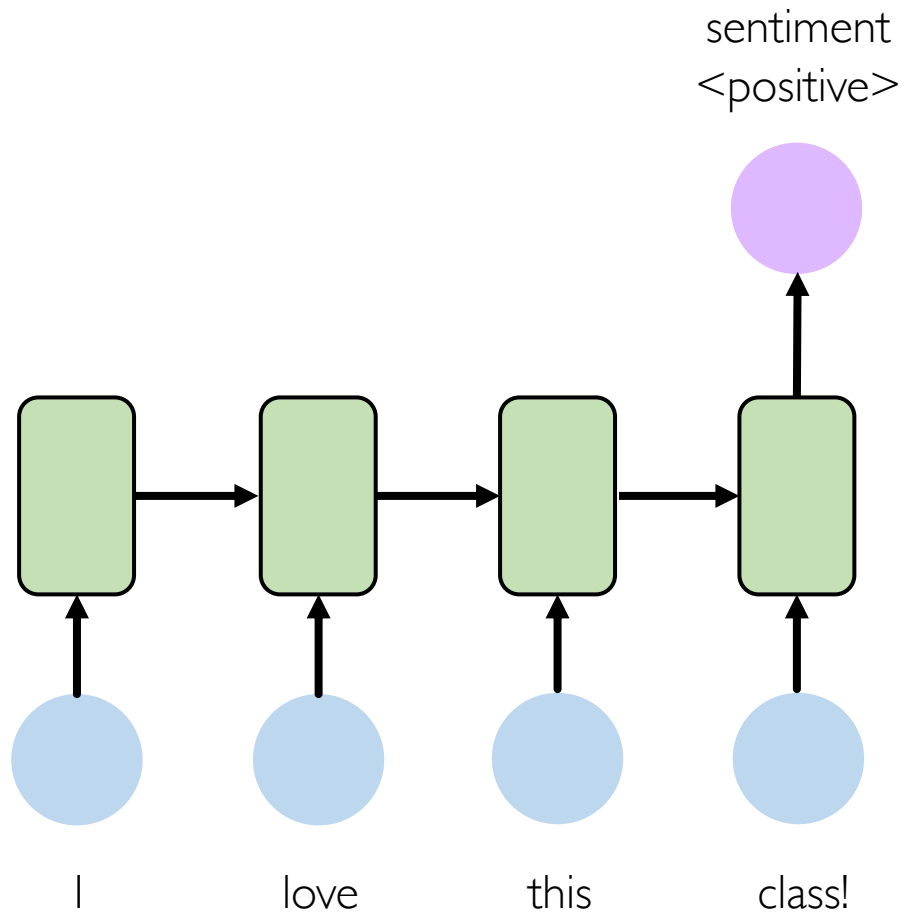


Input: sequence of words

Output: probability of having positive sentiment

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

Example task: sentiment classification

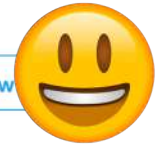


Tweet sentiment classification



Ivar Hagendoorn
@IvarHagendoorn

Follow



The @MIT Introduction to #DeepLearning is definitely one of the best courses of its kind currently available online introtodeeplearning.com

12:45 PM - 12 Feb 2018



Angels-Cave
@AngelsCave

Follow

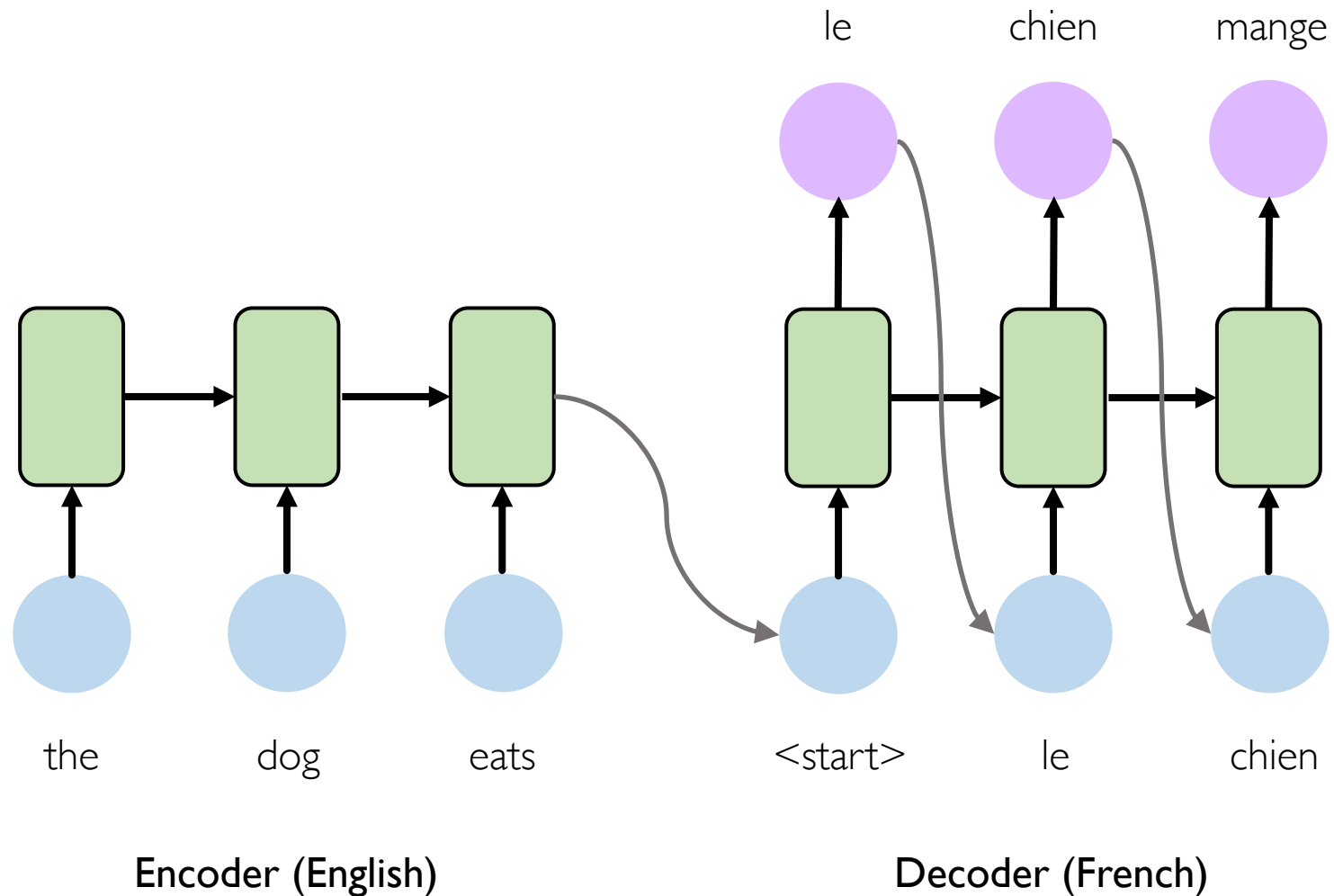


Replying to @Kazuki2048

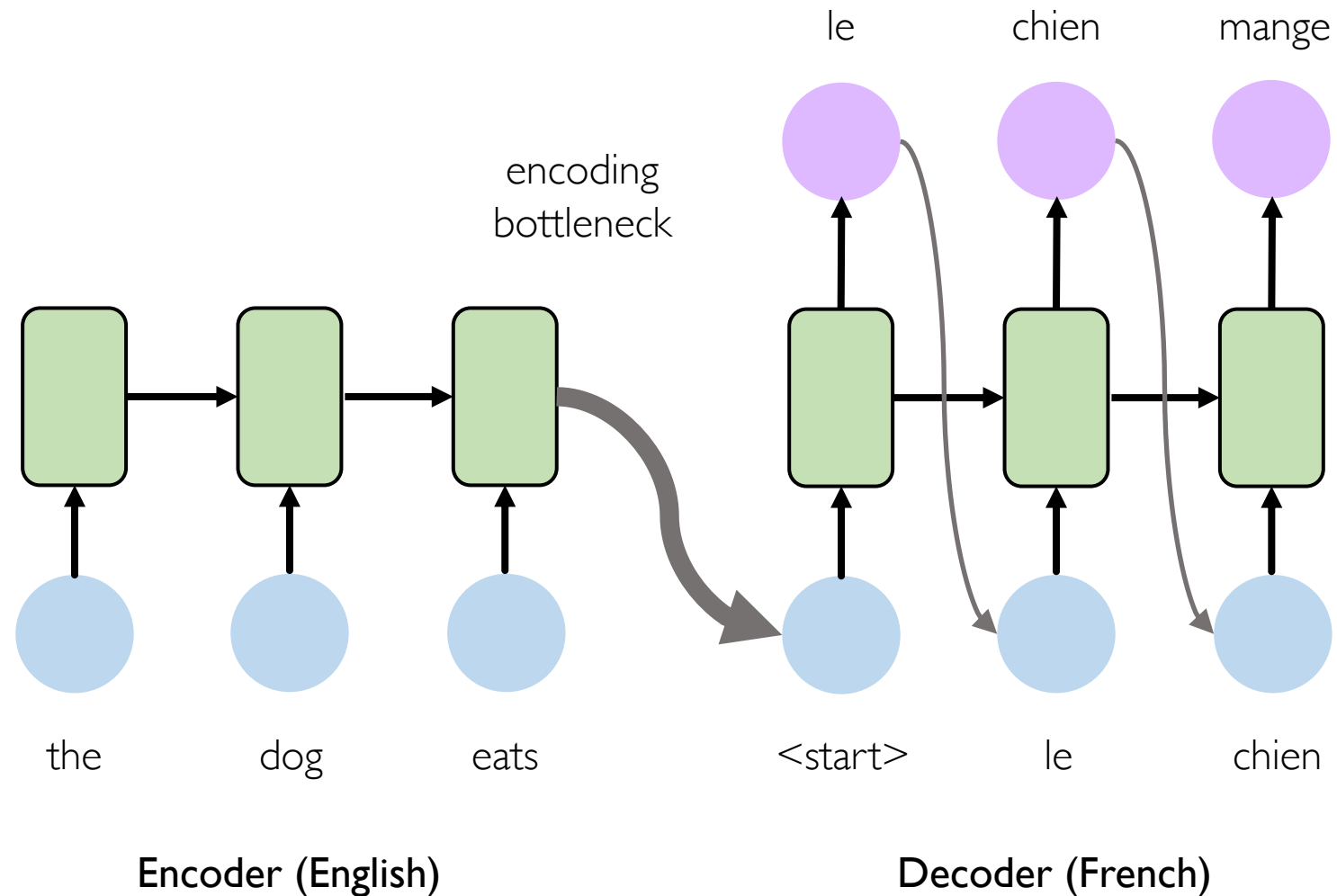
I wouldn't mind a bit of snow right now. We haven't had any in my bit of the Midlands this winter! :(

2:19 AM - 25 Jan 2019

Example task: machine translation



Example task: machine translation

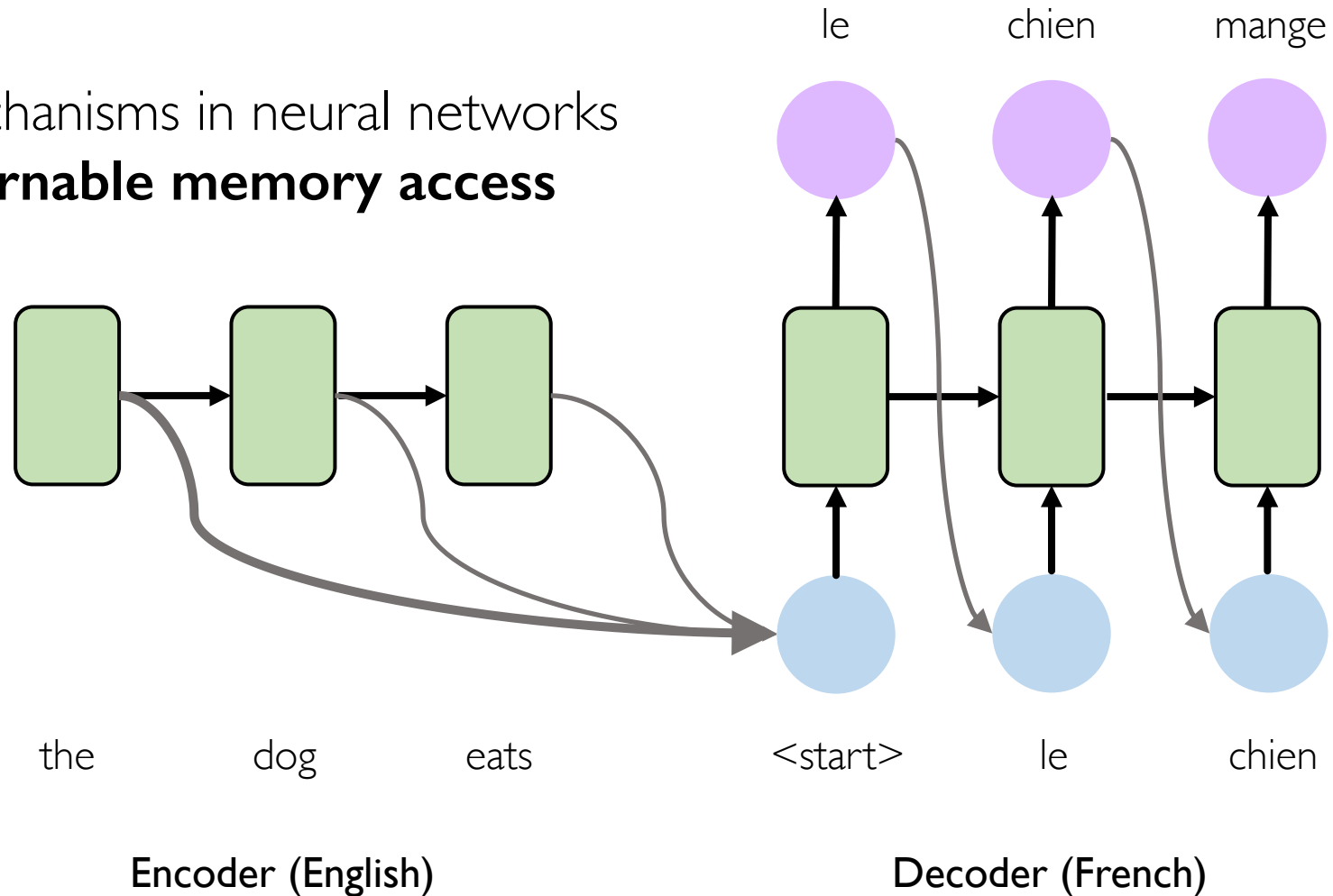


Adapted from H. Suresh, 6.S191 2018

[8,9]

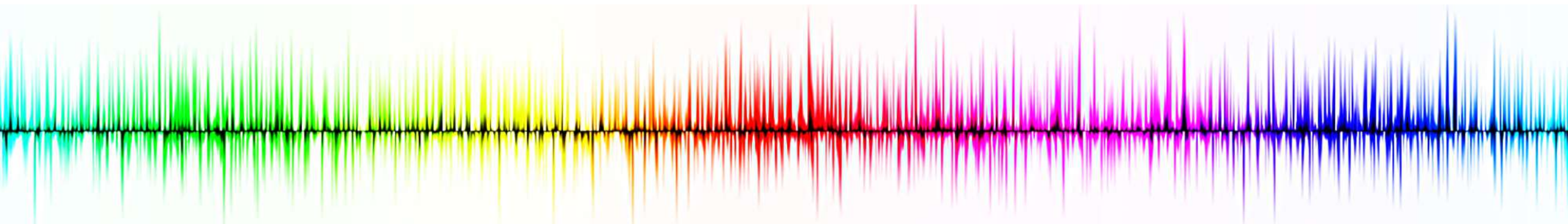
Attention mechanisms

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



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



References:
goo.gl/hbLkF6

End of Slides