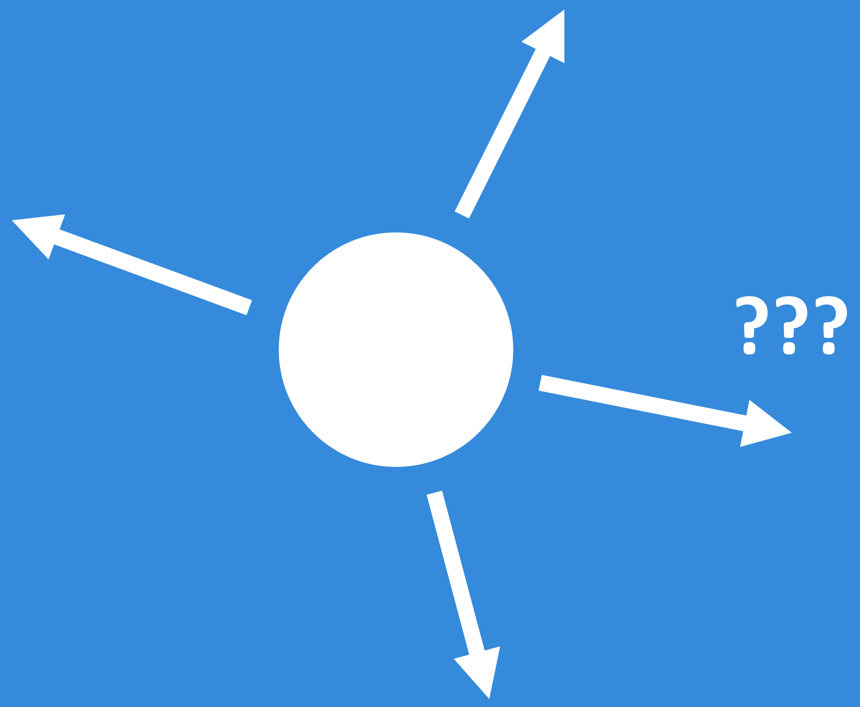
Deep Sequence Modeling
MIT 6.S191

Ava Soleimany

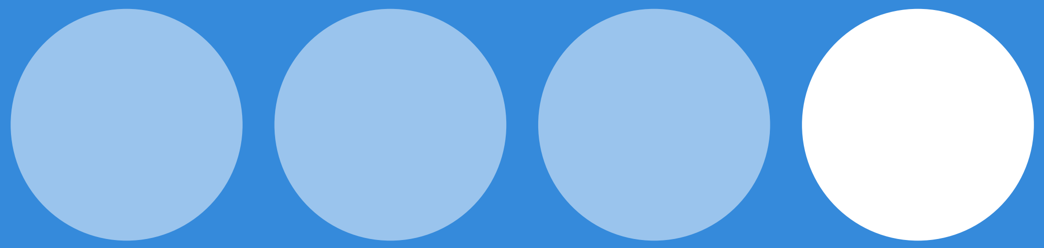
January 28, 2019

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

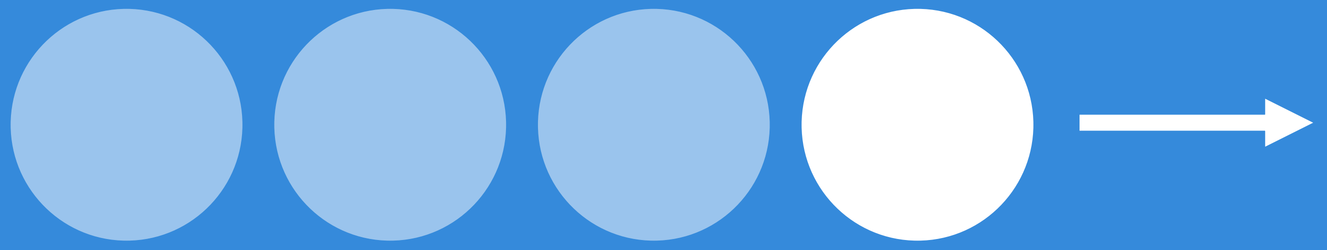
Given an image of a ball, can you predict where it will go next?



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

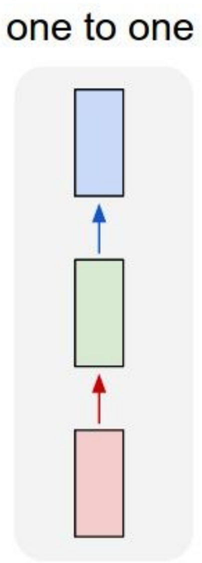
word:

Text

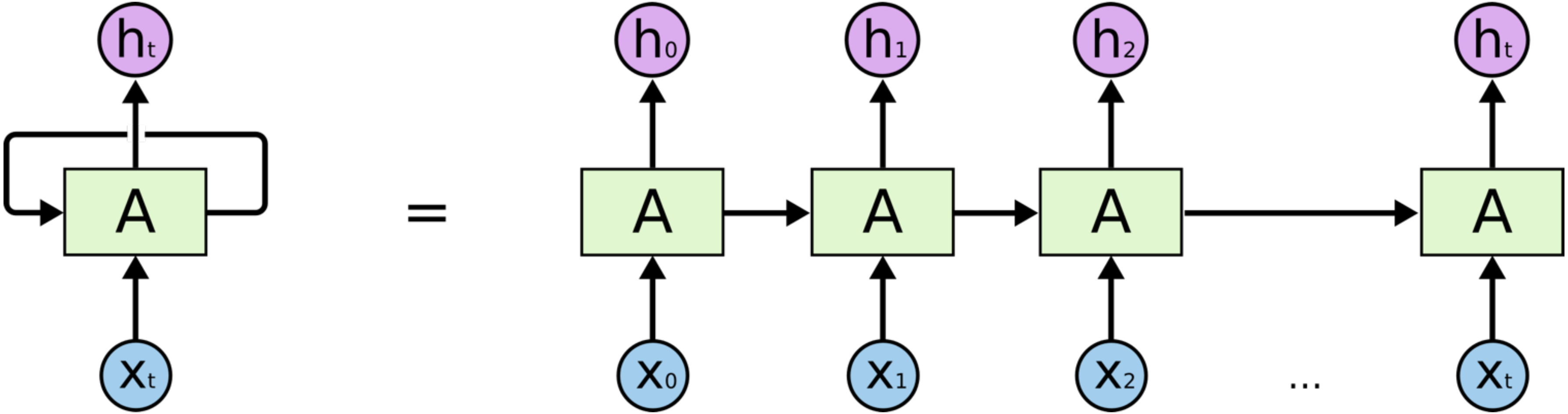
A Sequence Modeling Problem: Predict the Next Word

Sequence data

- We don’t understand one word only
- We understand based on the previous words + this word. (time series)
- NN/CNN cannot do this



Neural Networks for Sequence data



http://colah.github.io/posts/2015-08-Understanding-LS

RNN Applications

- Language Modeling
- Speech Recognition
- Machine Translation
- Conversation Modeling/Question Answering
- Image/Video Captioning
- Image/Music/Dance Generation

https://github.com/TensorFlowKR/awesome_tensorflow_implementations

http://jiwonkim.org/awesome-rnn/

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 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

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 ___."

J'aime 6.S191!

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.

Adapted from H. Suresh, 6.S191 2018

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 1 0 ...]

this morning took the cat

Each of these inputs has a **separate parameter:**

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 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

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 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

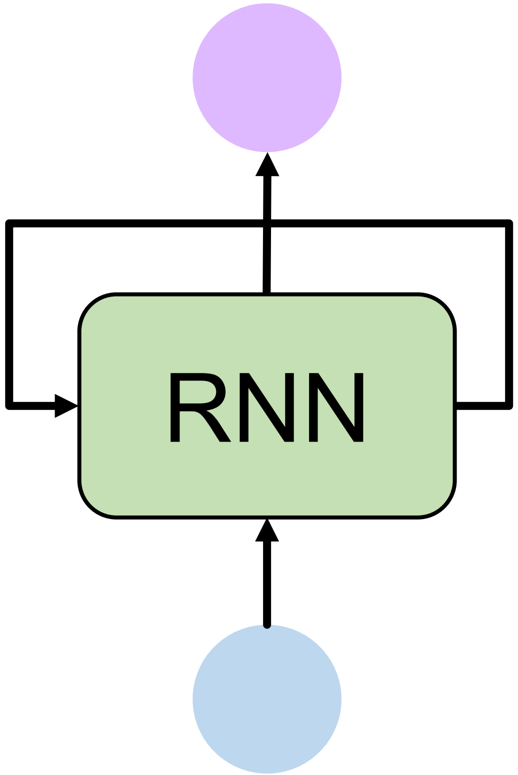
Things we learn about the sequence **won't transfer** if they appear **elsewhere** in the sequence.

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

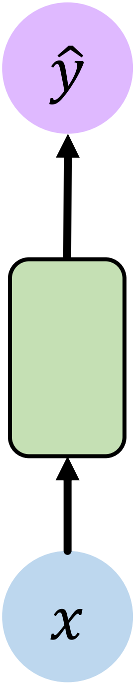


Today: **Recurrent Neural Networks (RNNs)** as an approach to sequence modeling problems

Adapted from H. Suresh, 6.S191 2018

Recurrent Neural Networks (RNNs)

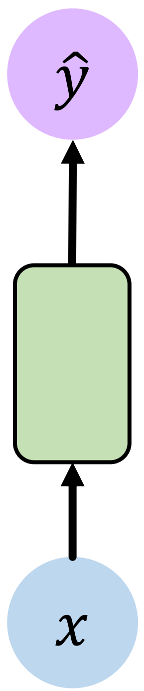
Standard feed-forward neural network



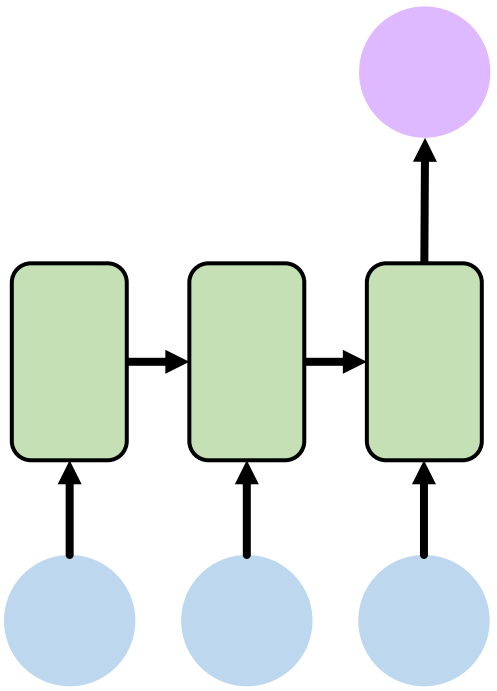
One to One
"Vanilla" neural network

[1]

Recurrent neural networks: sequence modeling



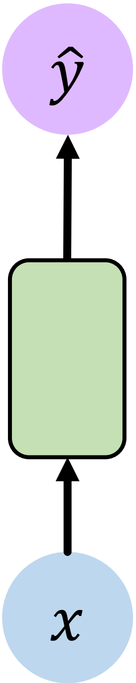
One to One
"Vanilla" neural network



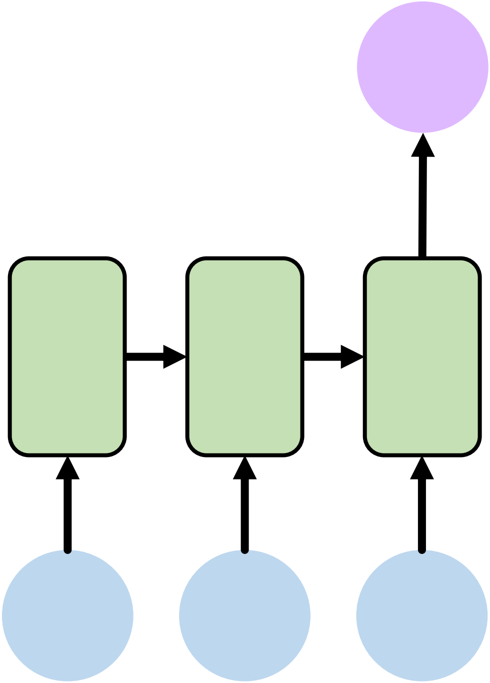
Many to One
Sentiment Classification

[1]

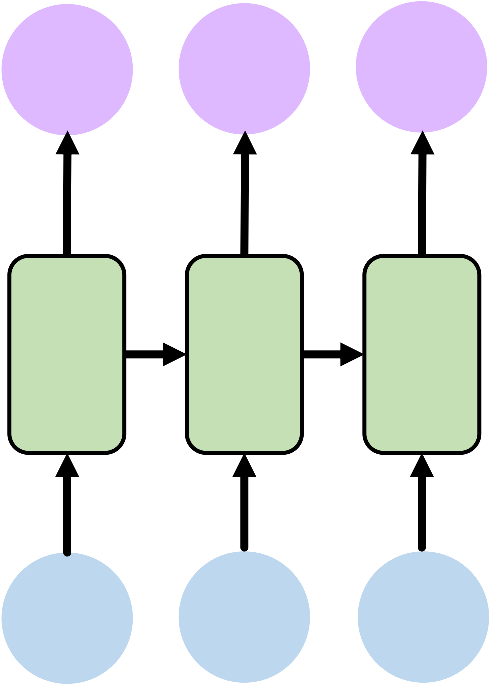
Recurrent neural networks: sequence modeling



One to One
"Vanilla" neural network



Many to One
Sentiment Classification

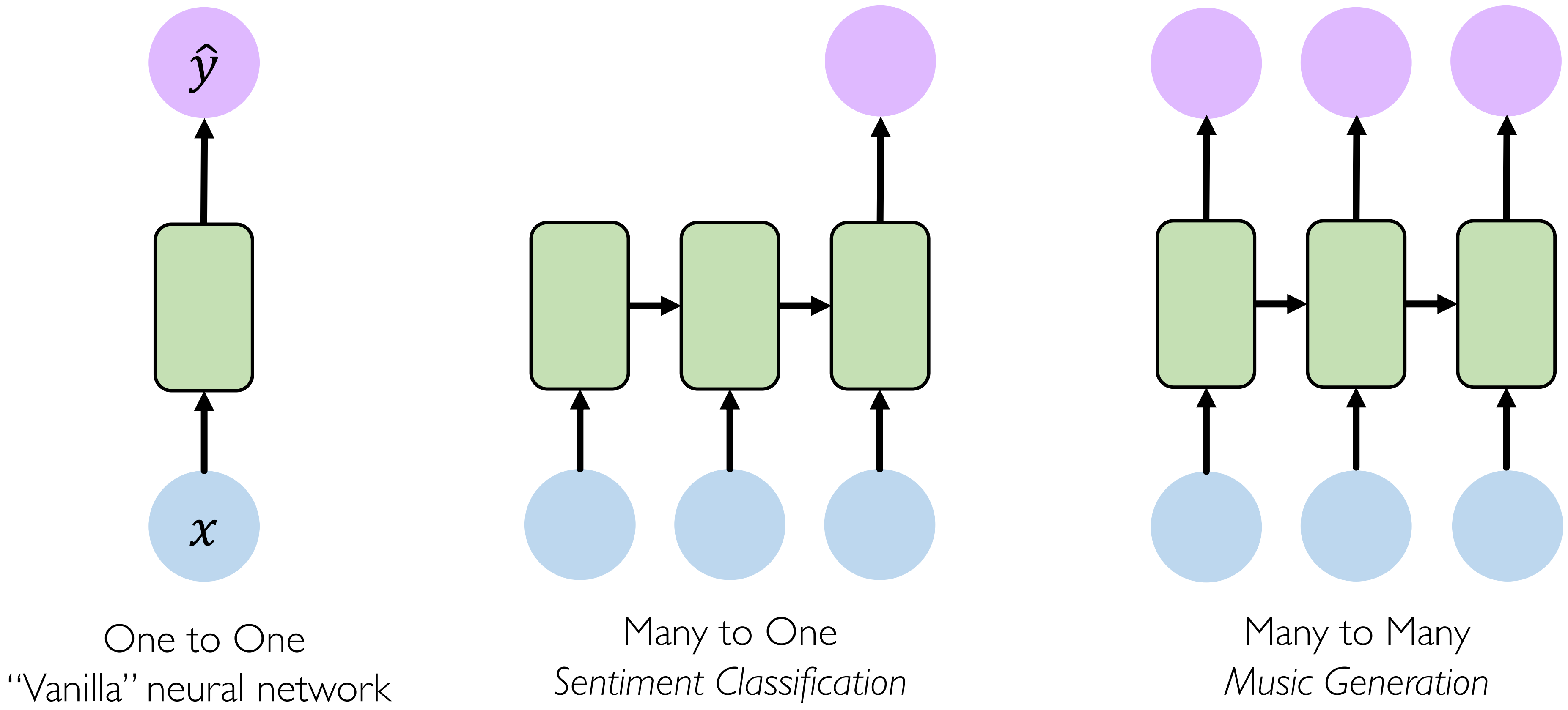


Many to Many
Music Generation

6.S191 Lab!

[1]

Recurrent neural networks: sequence modeling

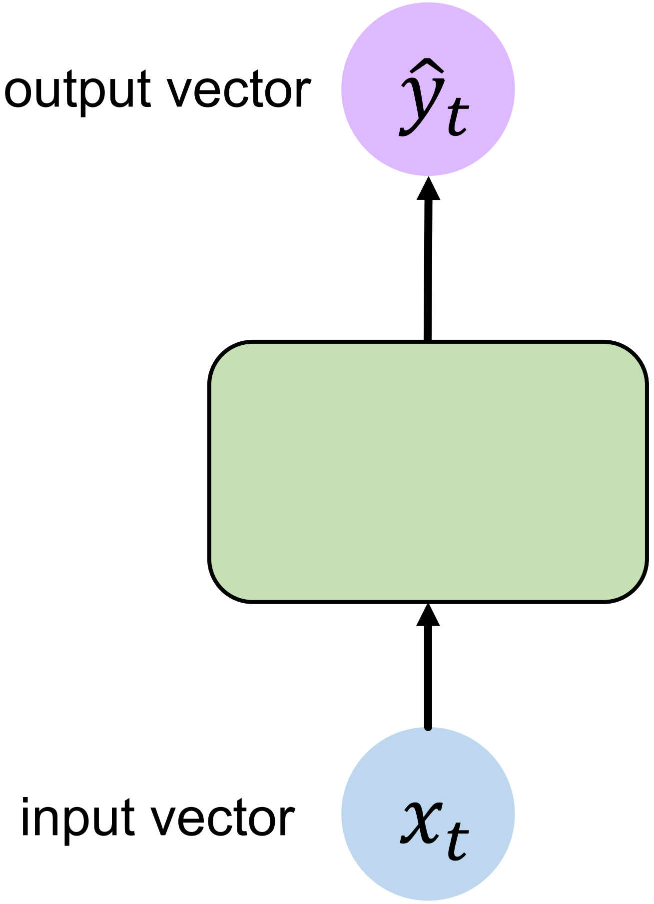


6.S191 Lab!

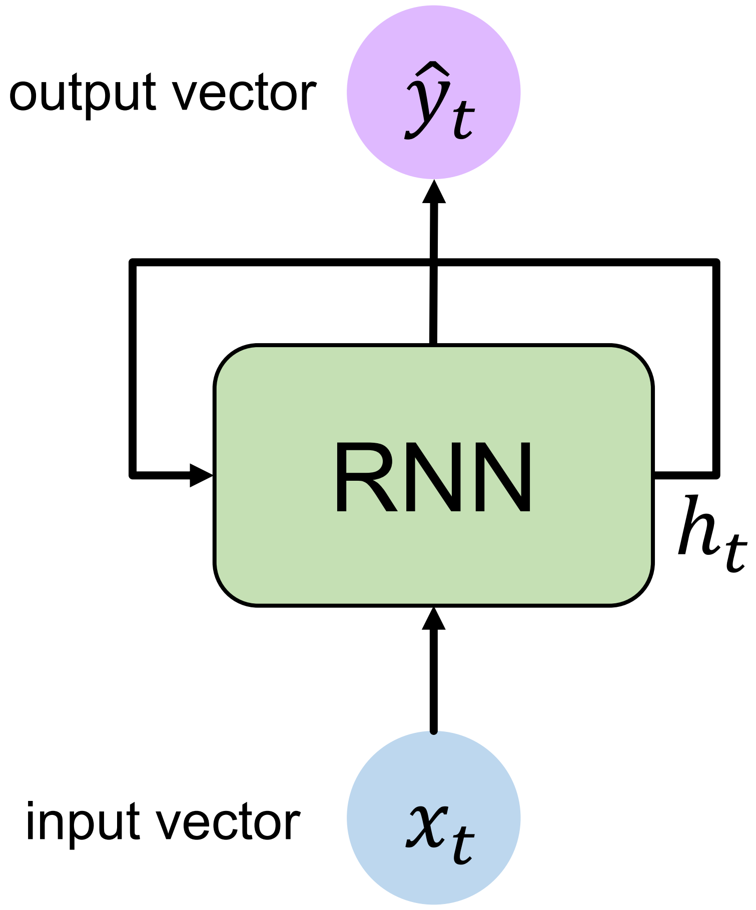
... and many other architectures and applications

[1]

A standard "vanilla" neural network

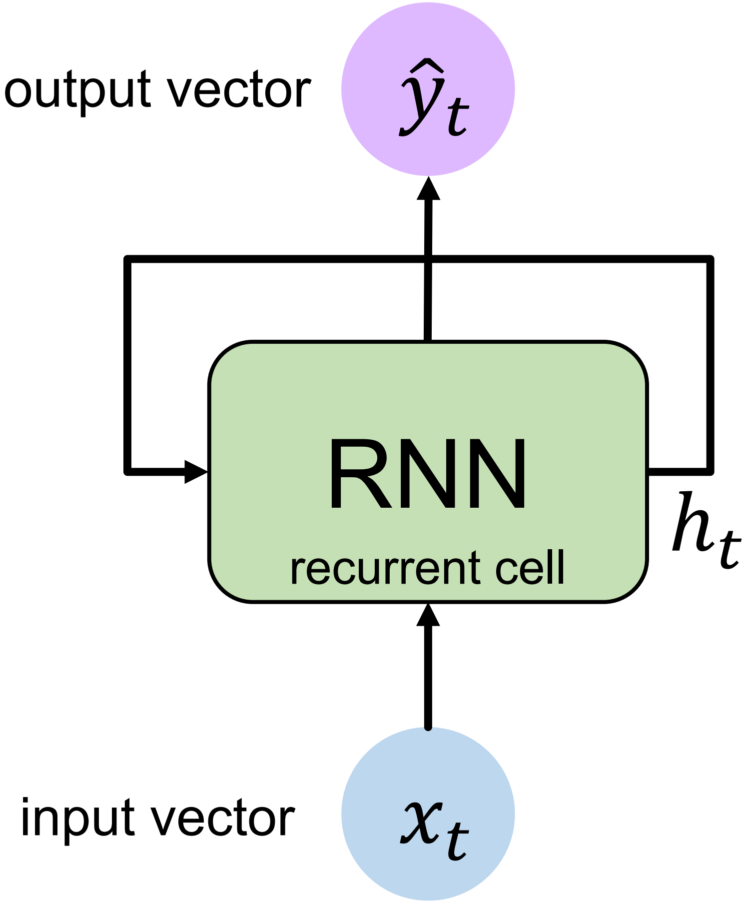


A recurrent neural network (RNN)

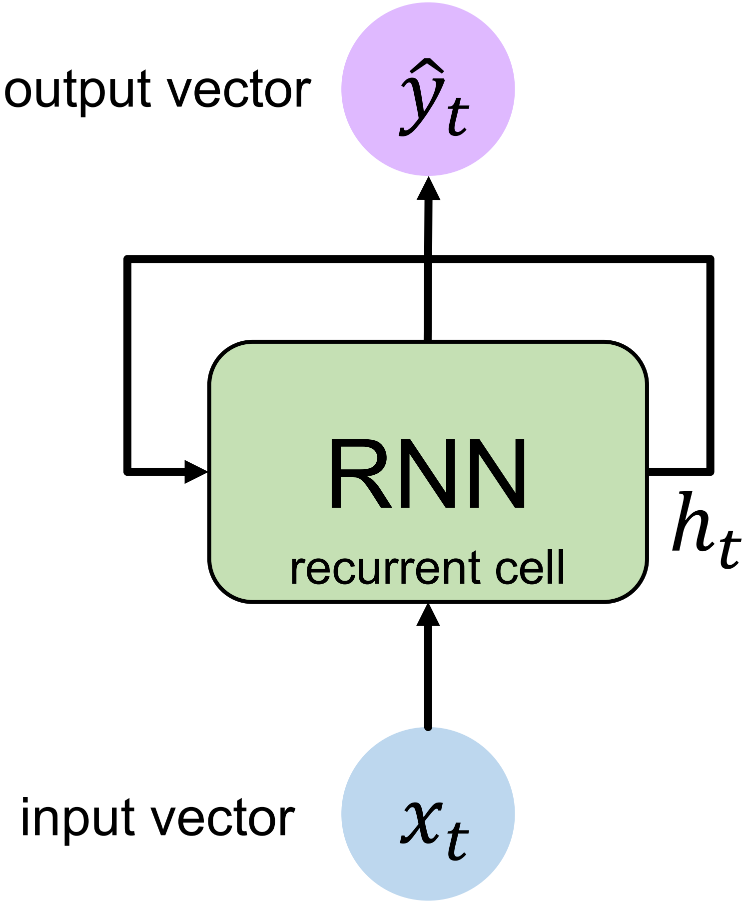


[2]

A recurrent neural network (RNN)

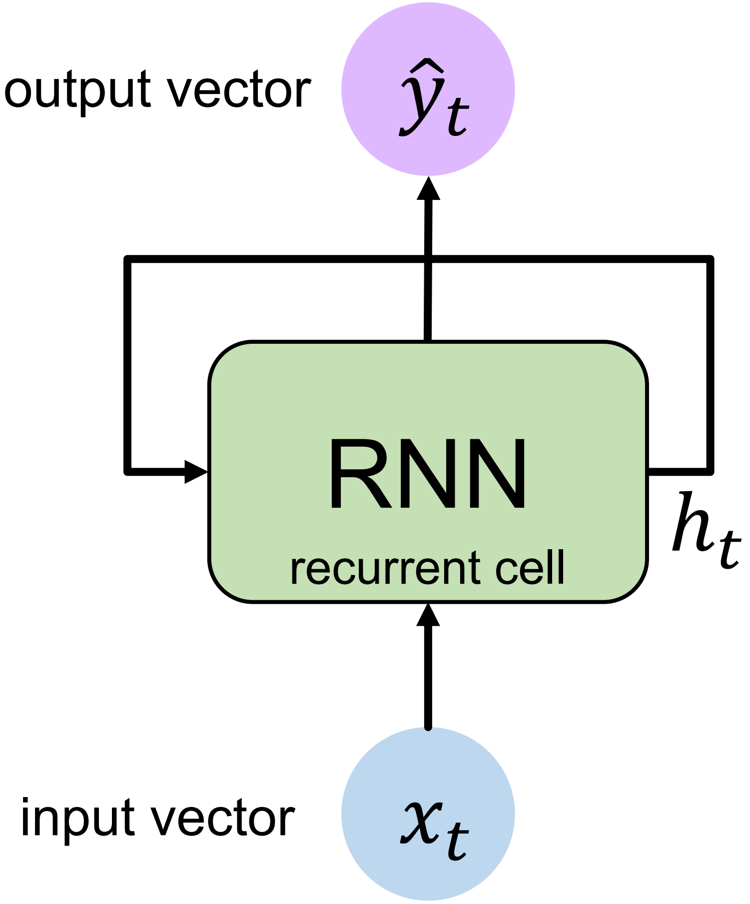


A recurrent neural network (RNN)



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:

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

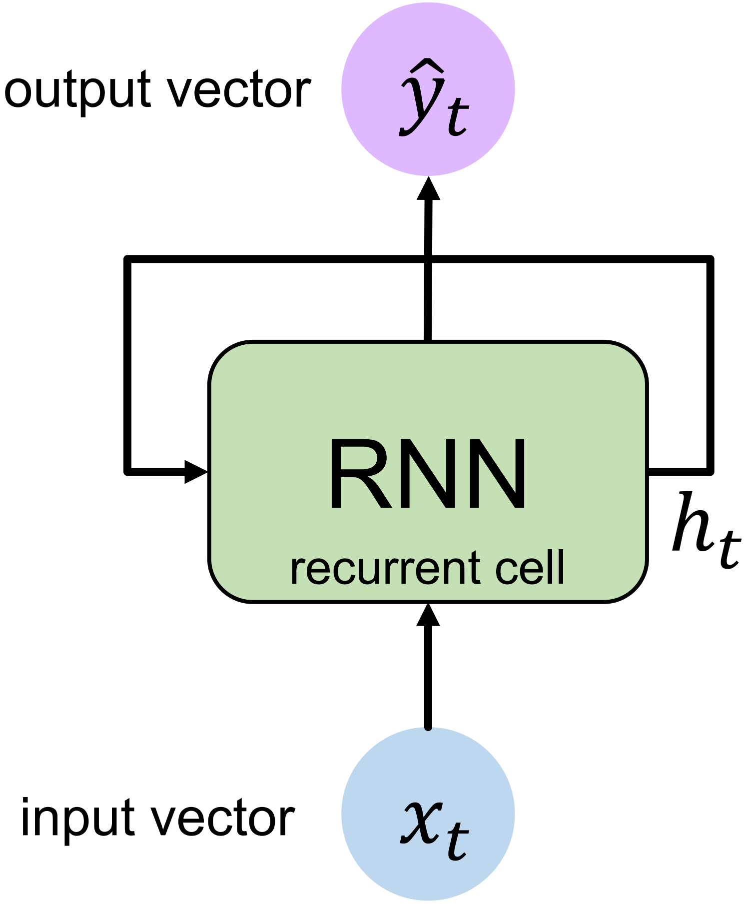
cell state

function parameterized by W

old state

input vector at time step *t*

A recurrent neural network (RNN)



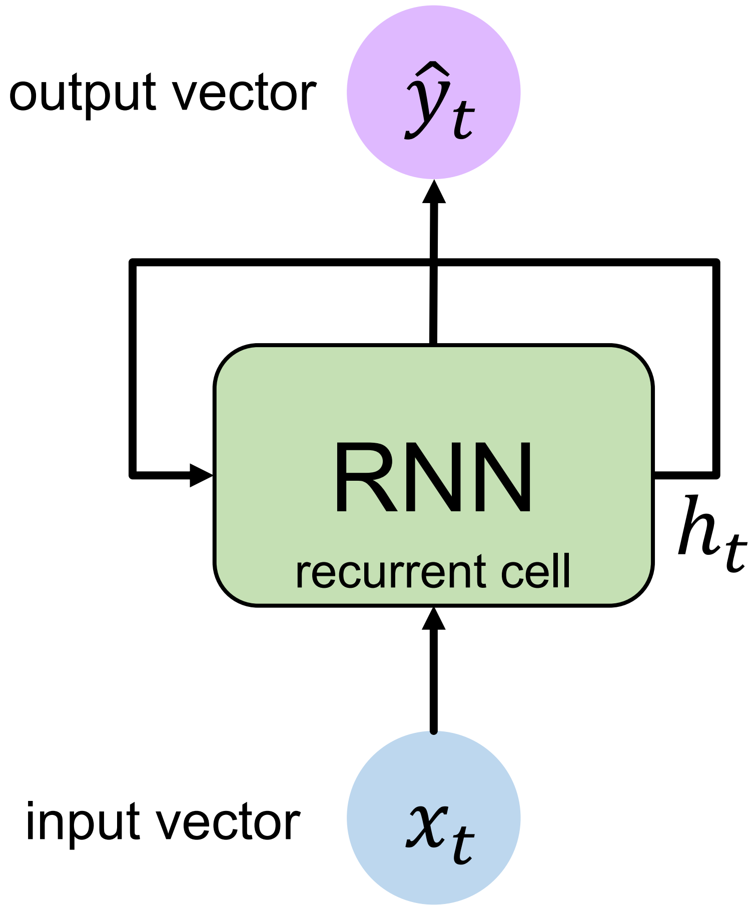
Apply a **recurrence relation** at every time step to process a sequence:

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

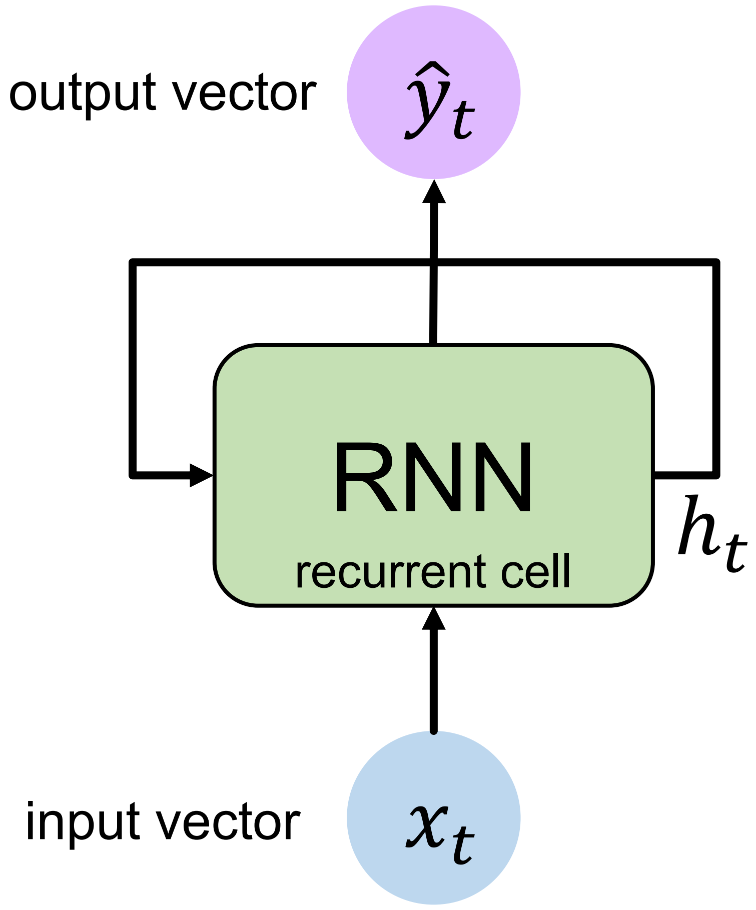
new state — function parameterized by W — old state — input vector at time step *t*

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

RNN state update and output

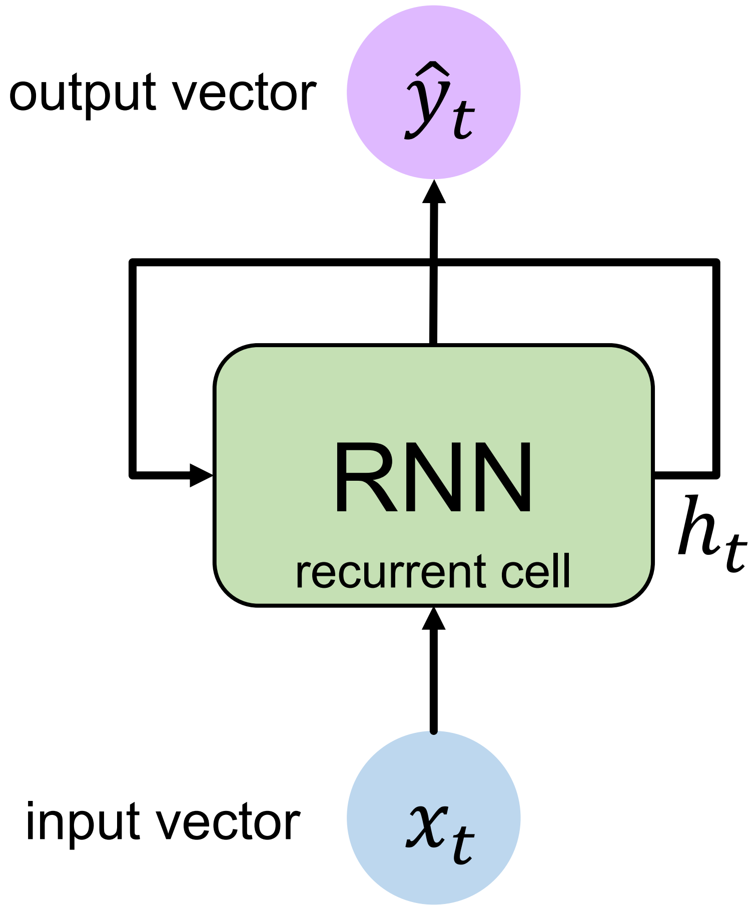


RNN state update and output



Input Vector

RNN state update and output

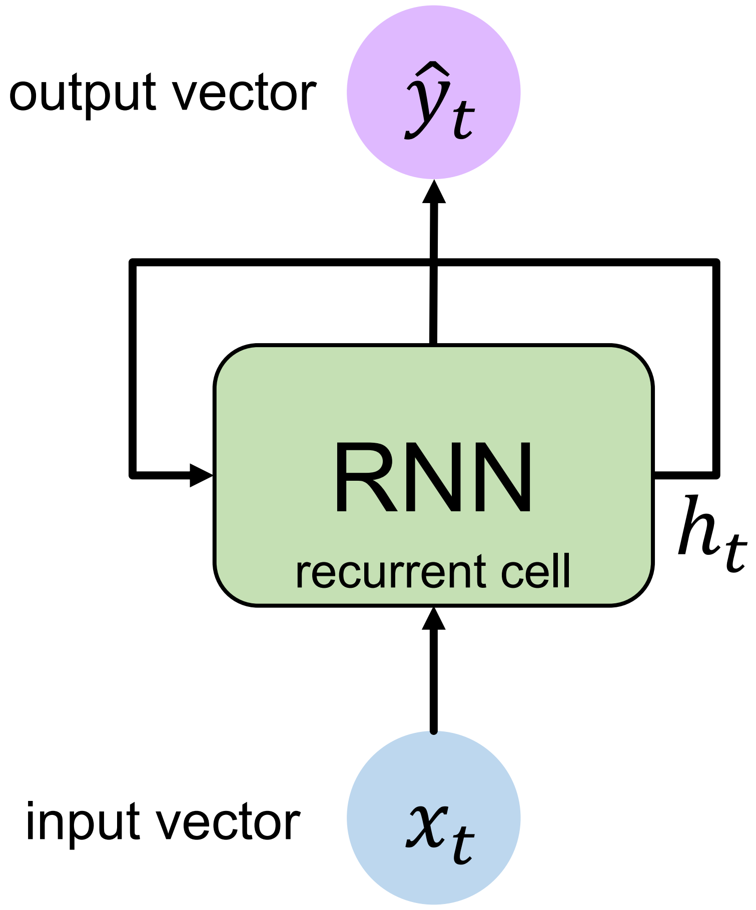


Update Hidden State

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

Input Vector

RNN state update and output



Output Vector

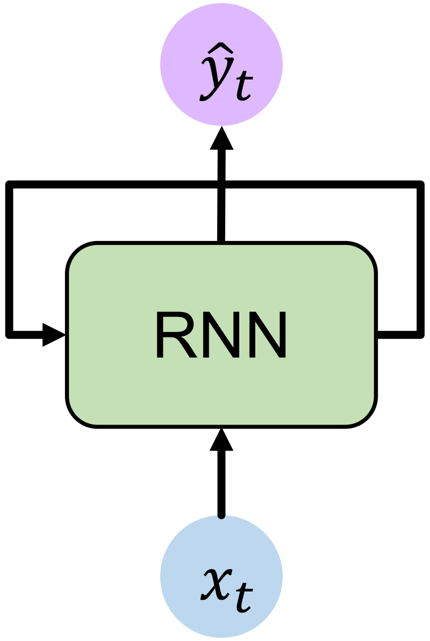
$$\hat{y}_t = \boldsymbol{W_{hy}} h_t$$

Update Hidden State

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

Input Vector

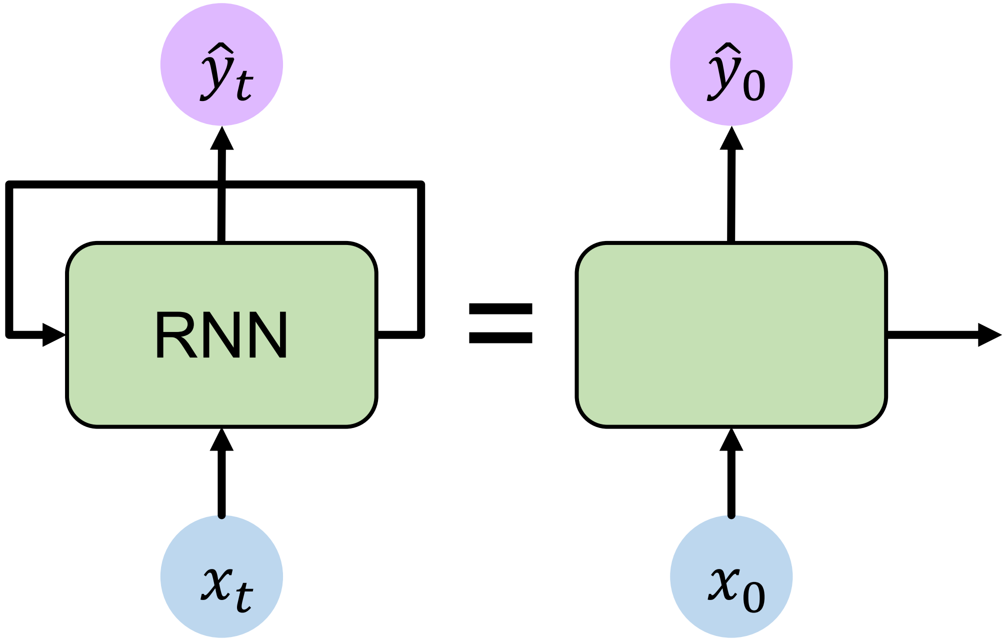
RNNs: computational graph across time



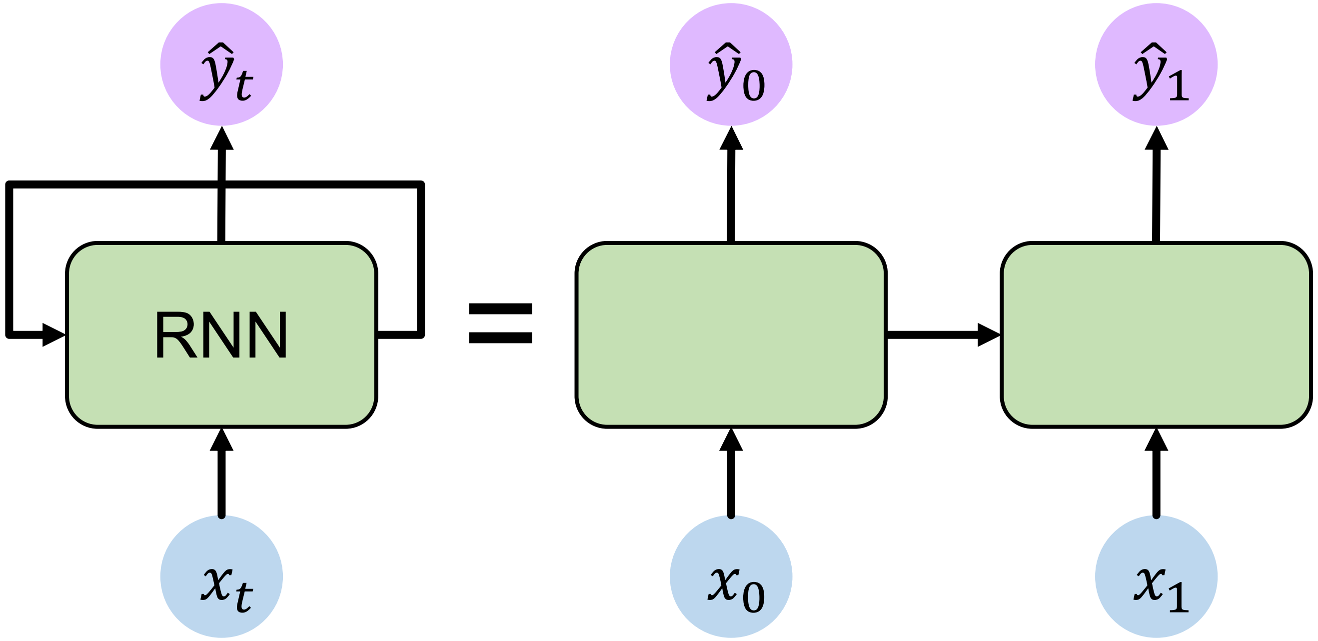
=

Represent as computational graph unrolled across time

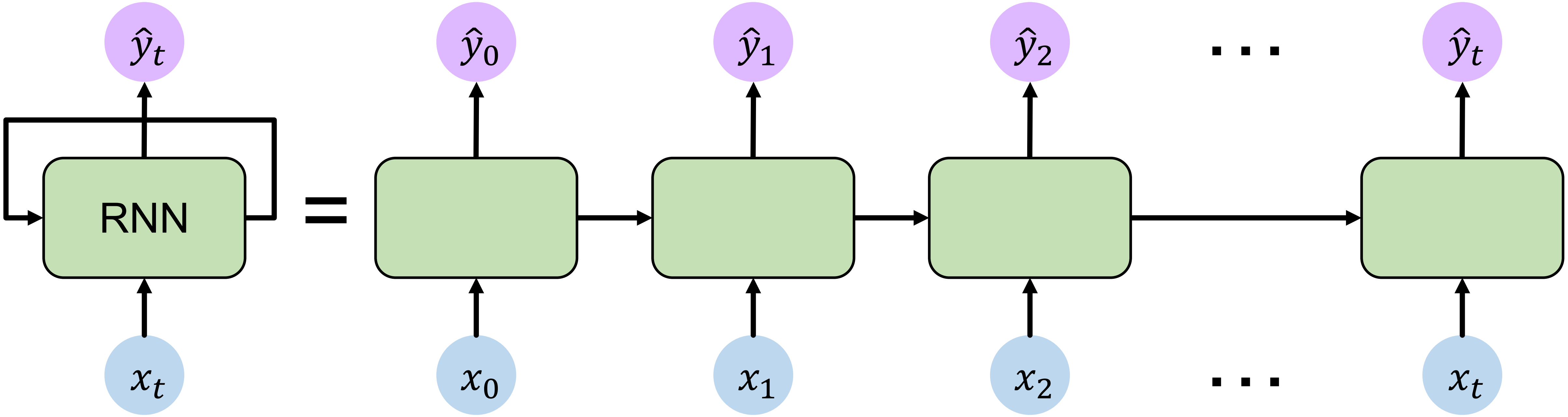
RNNs: computational graph across time



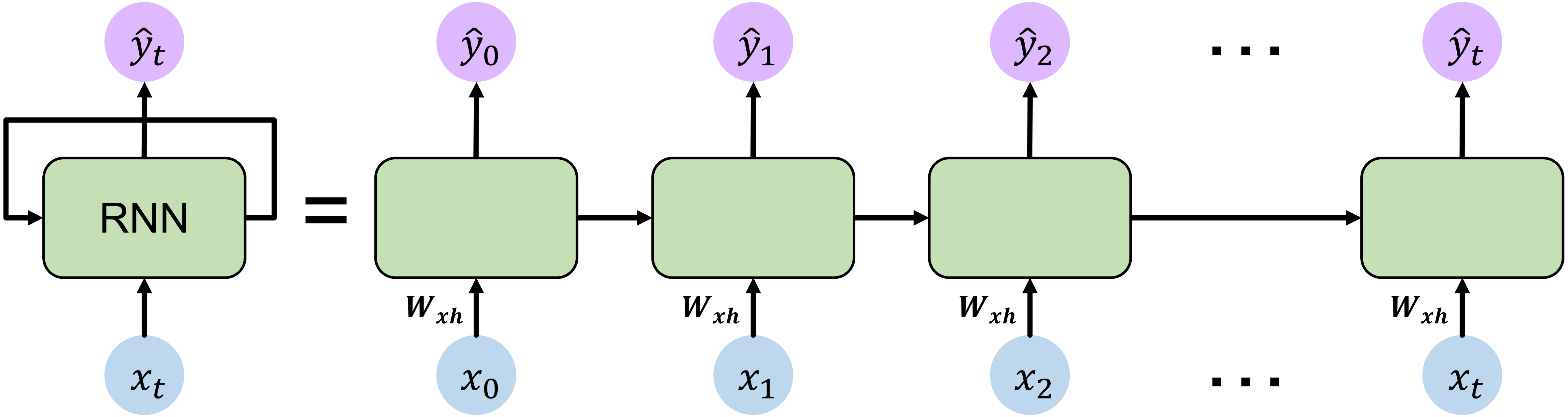
RNNs: computational graph across time



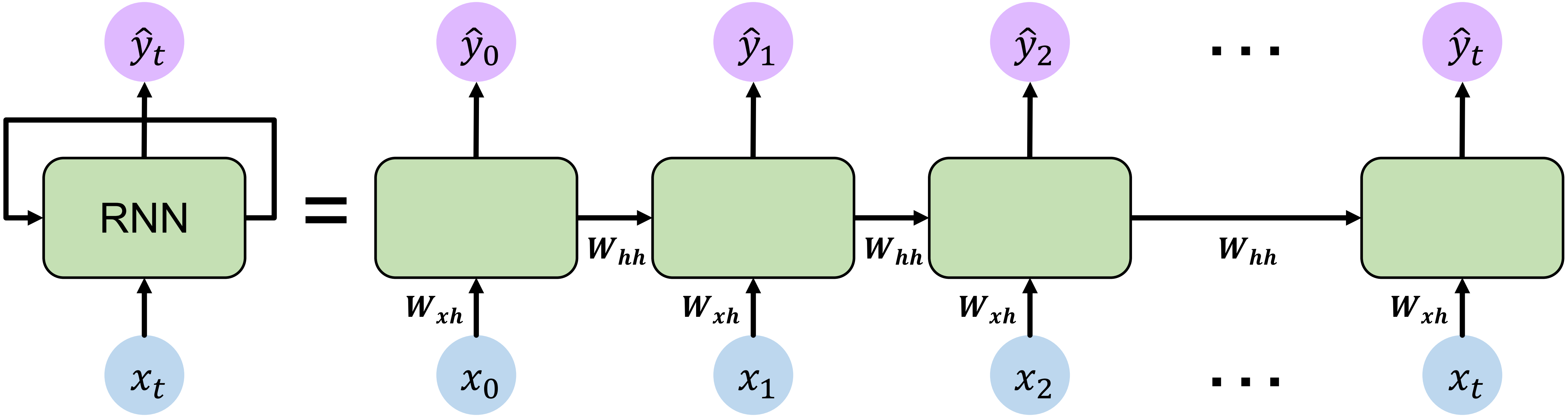
RNNs: computational graph across time



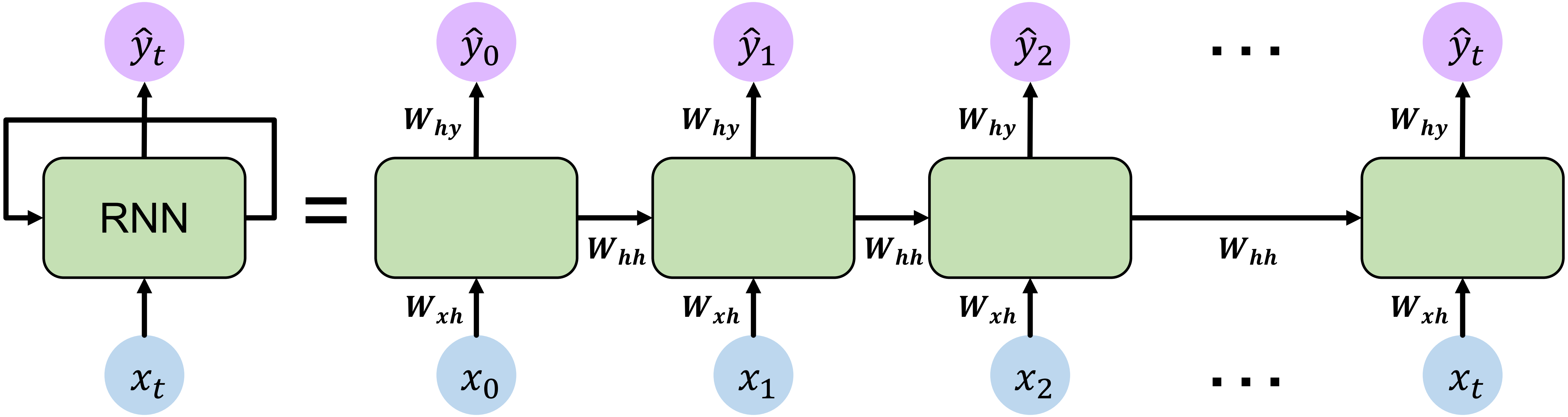
RNNs: computational graph across time



RNNs: computational graph across time

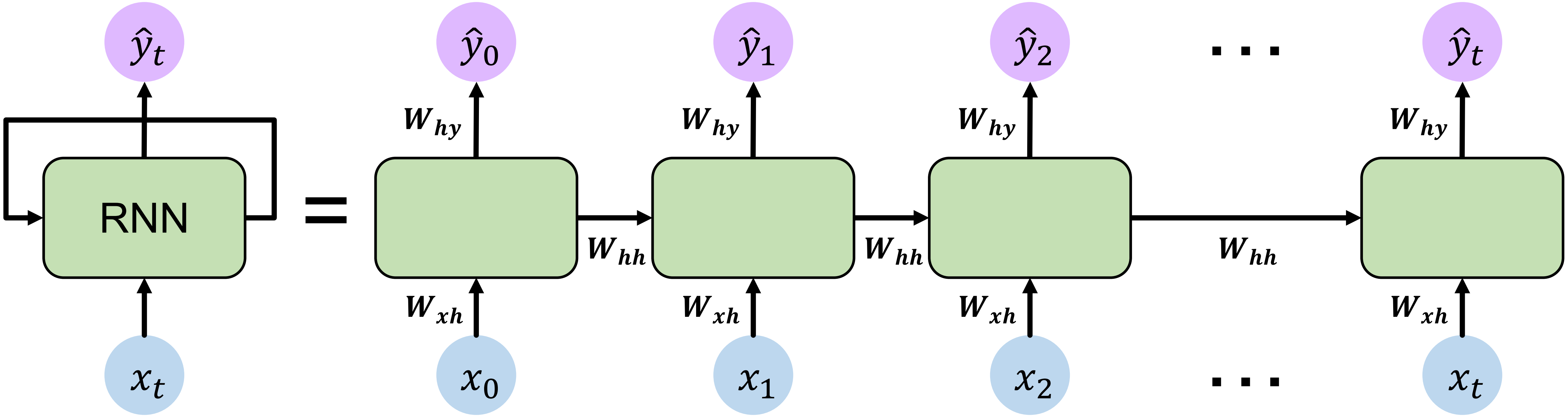


RNNs: computational graph across time

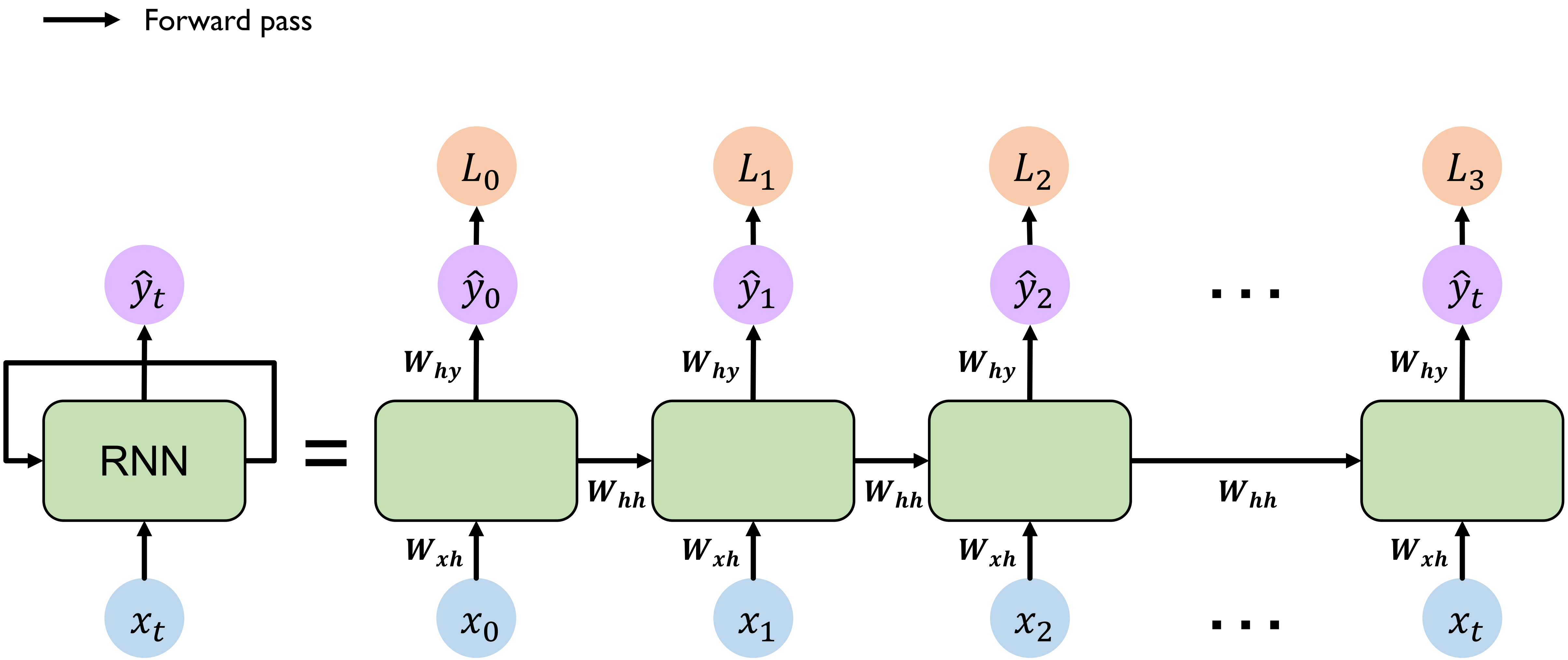


RNNs: computational graph across time

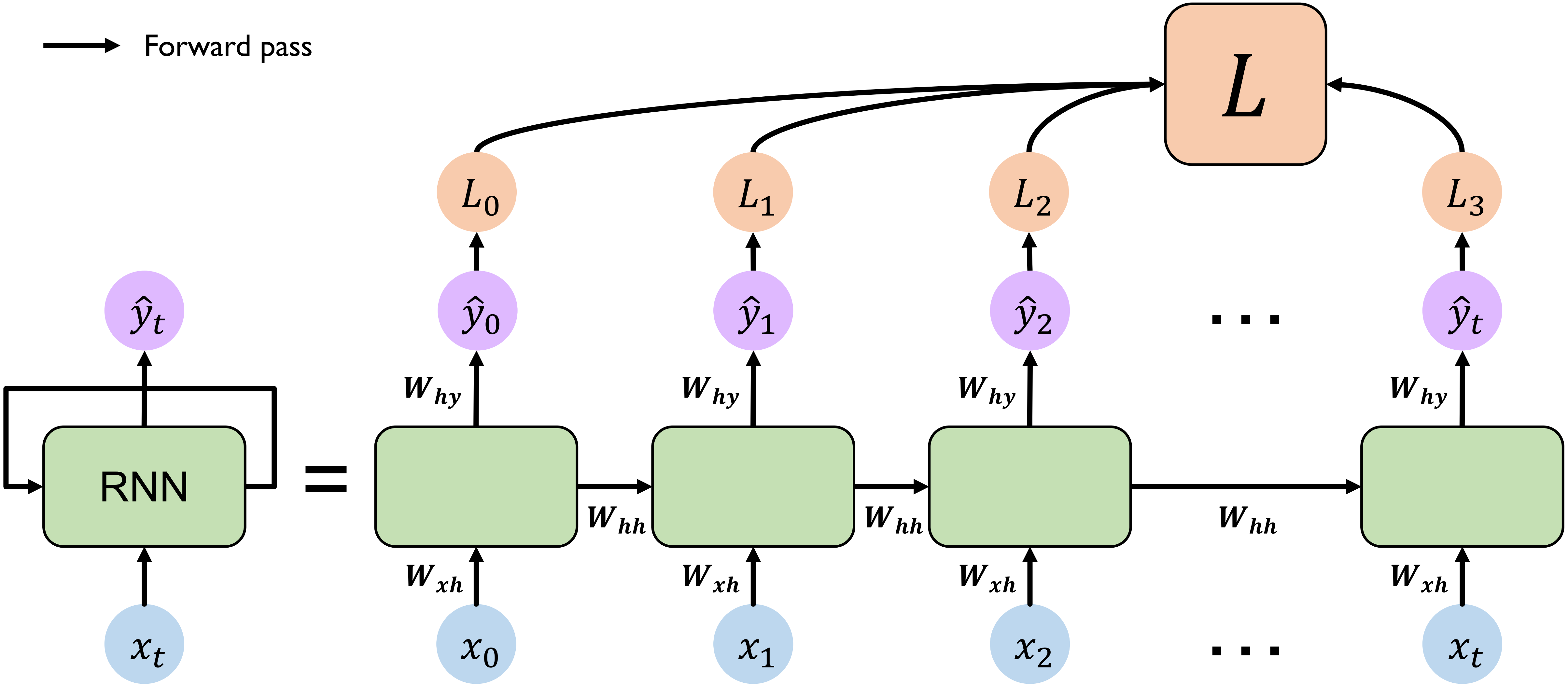
Re-use the **same weight matrices** at every time step



RNNs: computational graph across time

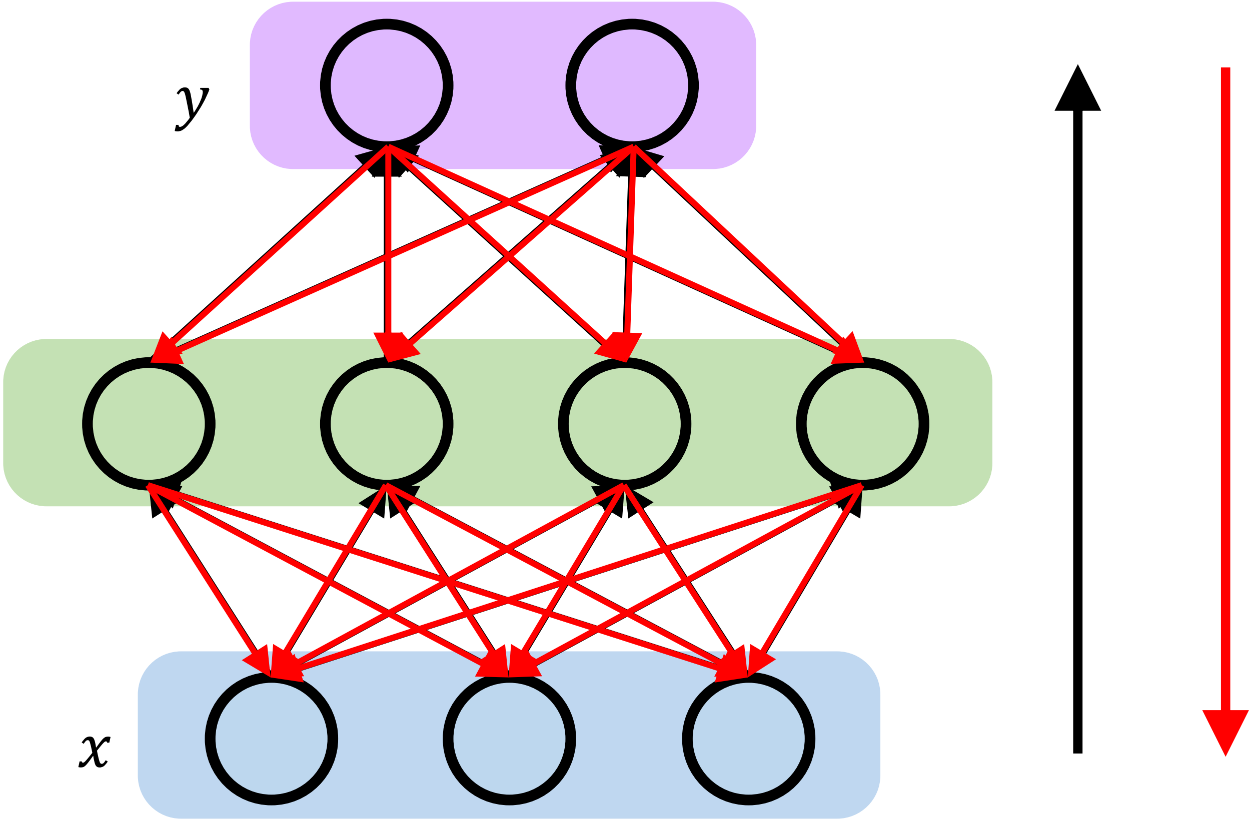


RNNs: computational graph across time



Backpropagation Through Time (BPTT)

Recall: backpropagation in feed forward models

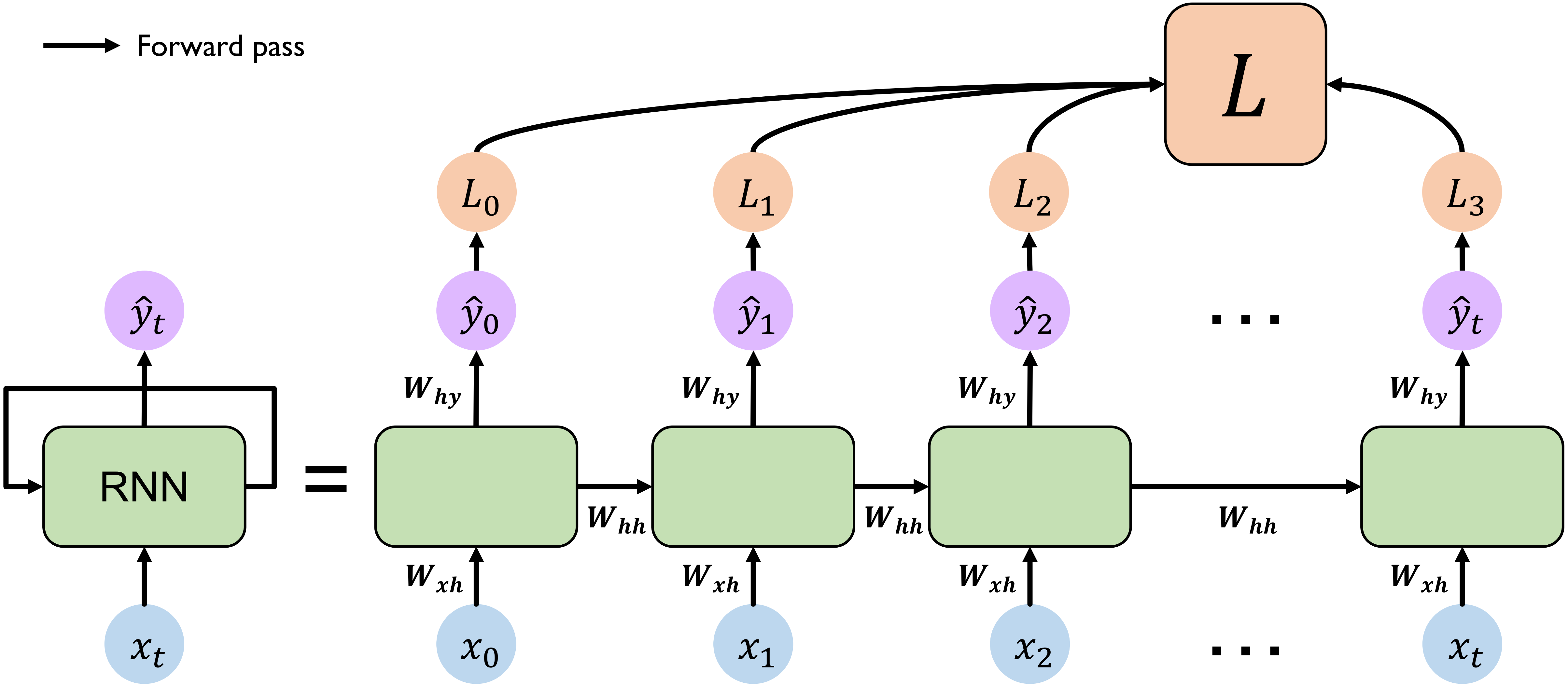


Backpropagation algorithm:

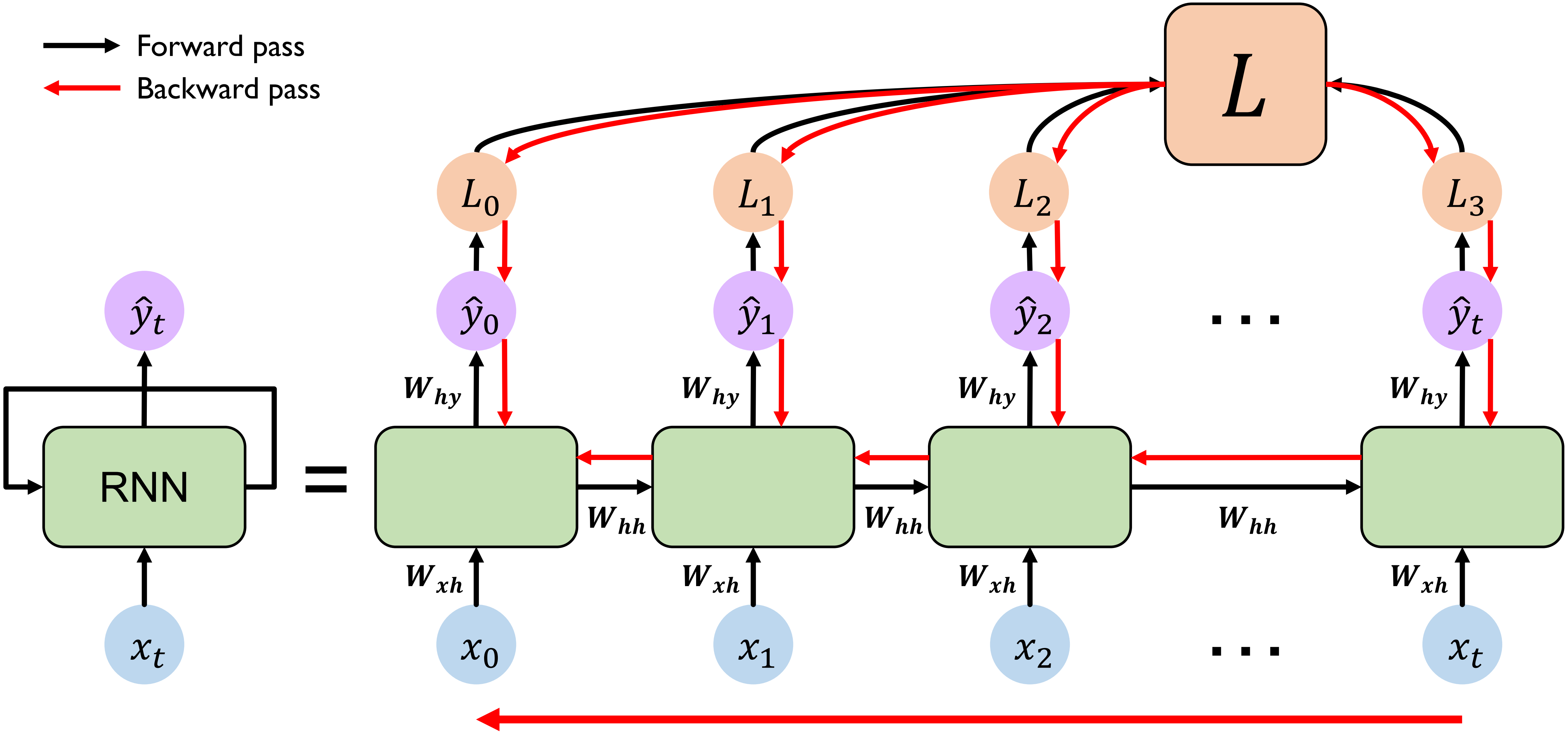
1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

[3]

RNNs: backpropagation through time

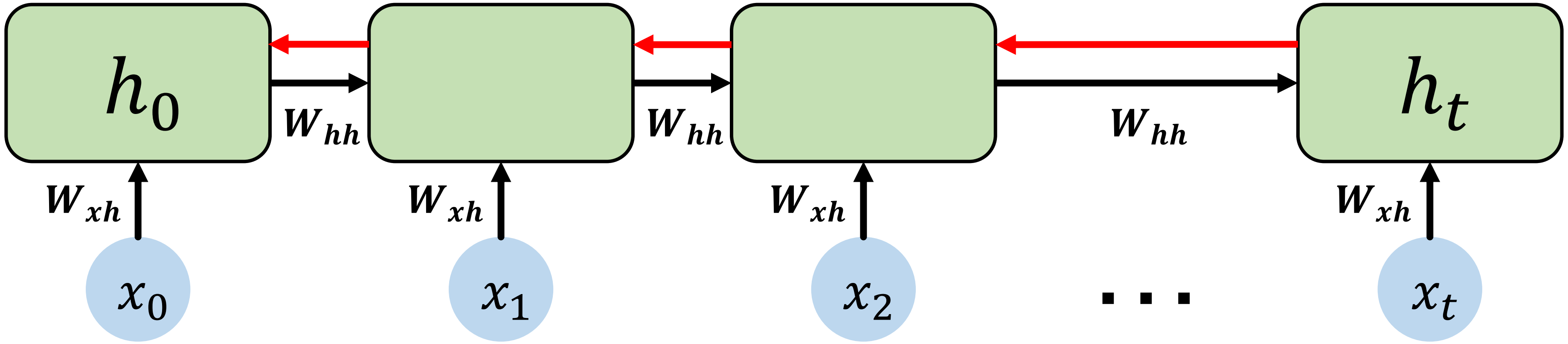


RNNs: backpropagation through time



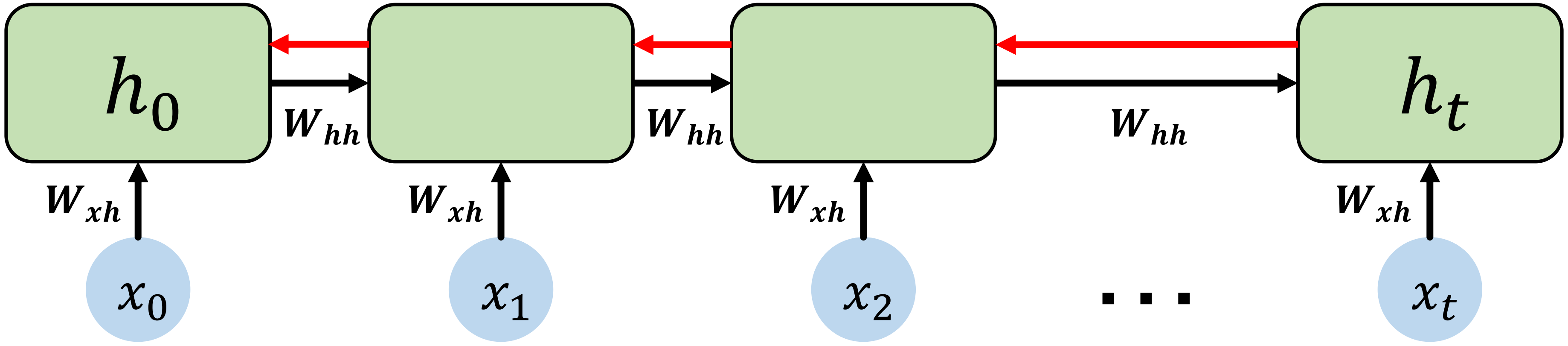
[4]

Standard RNN gradient flow



[1]

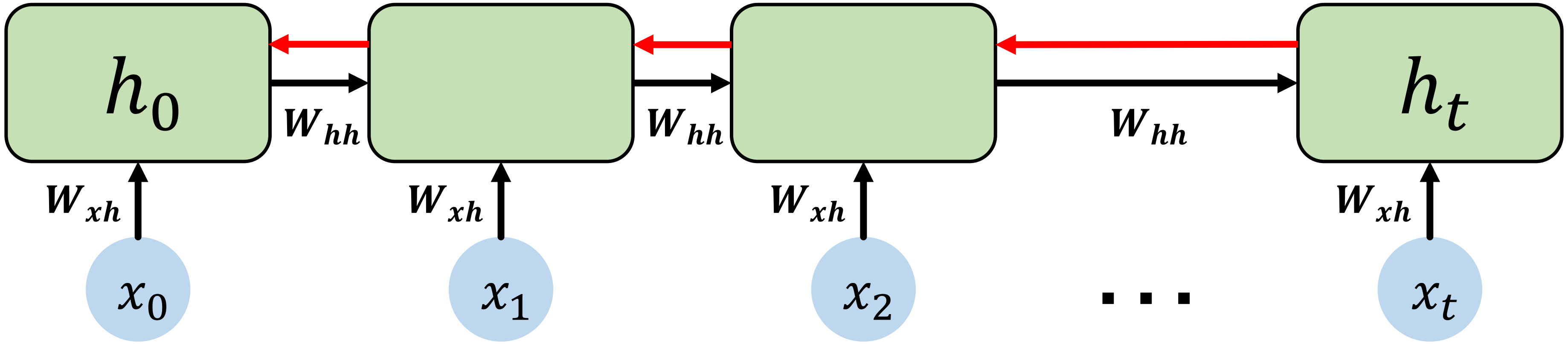
Standard RNN gradient flow



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

[1]

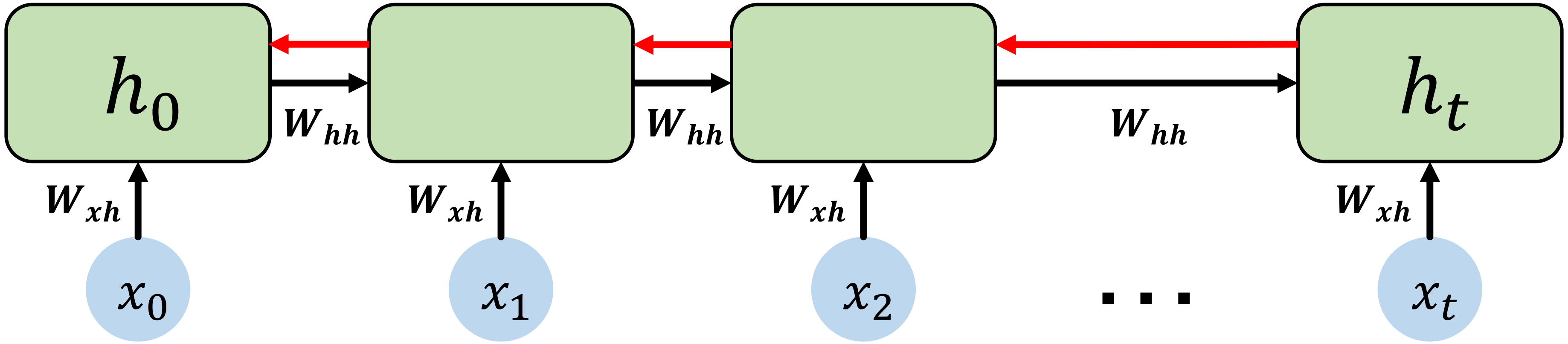
Standard RNN gradient flow: exploding gradients



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



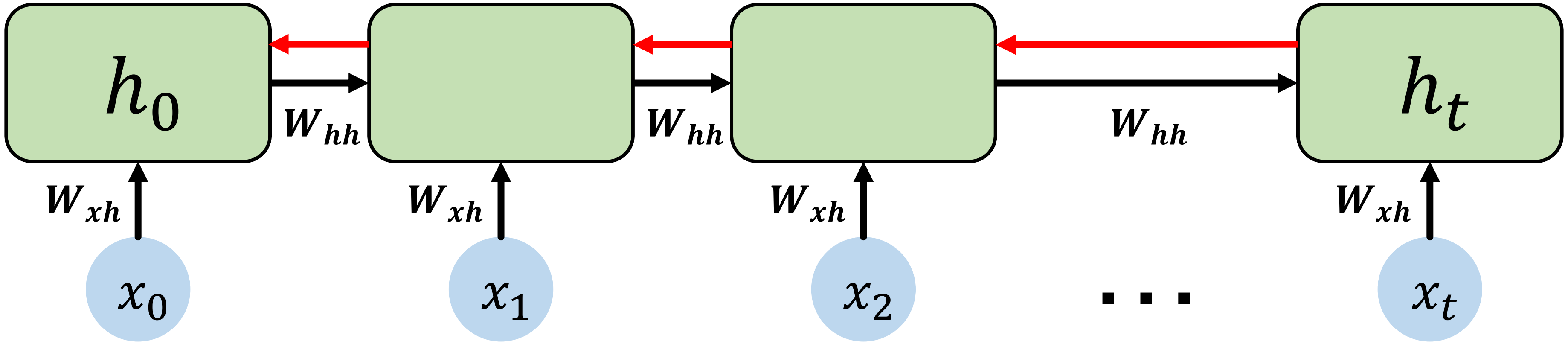
Computing the gradient wrt h_0 involves **many factors of W_{hh}** (and repeated f'!)

Many values > 1:
exploding gradients

Gradient clipping to
scale big gradients

[1]

Standard RNN gradient flow: vanishing gradients



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

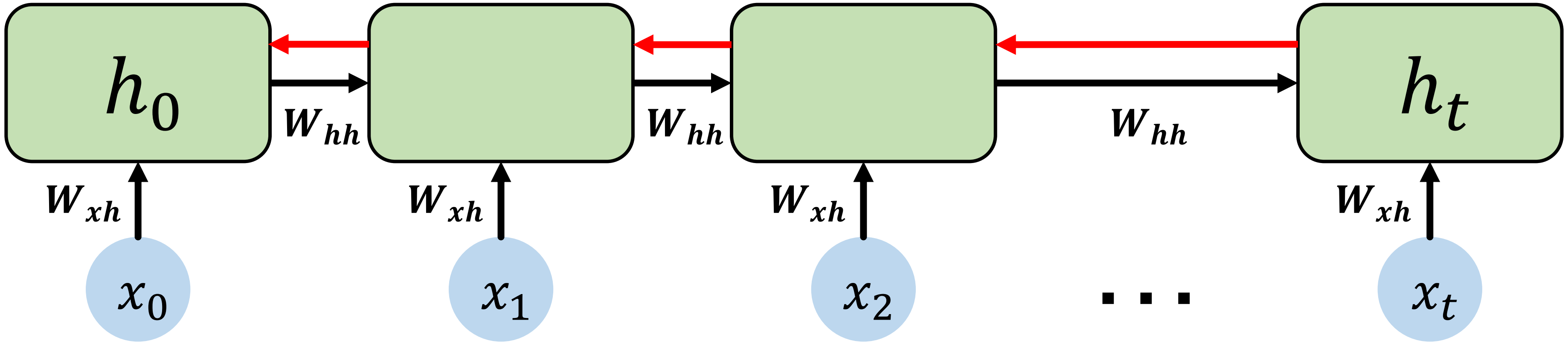
Many values > 1:
exploding gradients

Gradient clipping to scale big gradients

Many values < 1:
vanishing gradients

[1]

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

[1]

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?

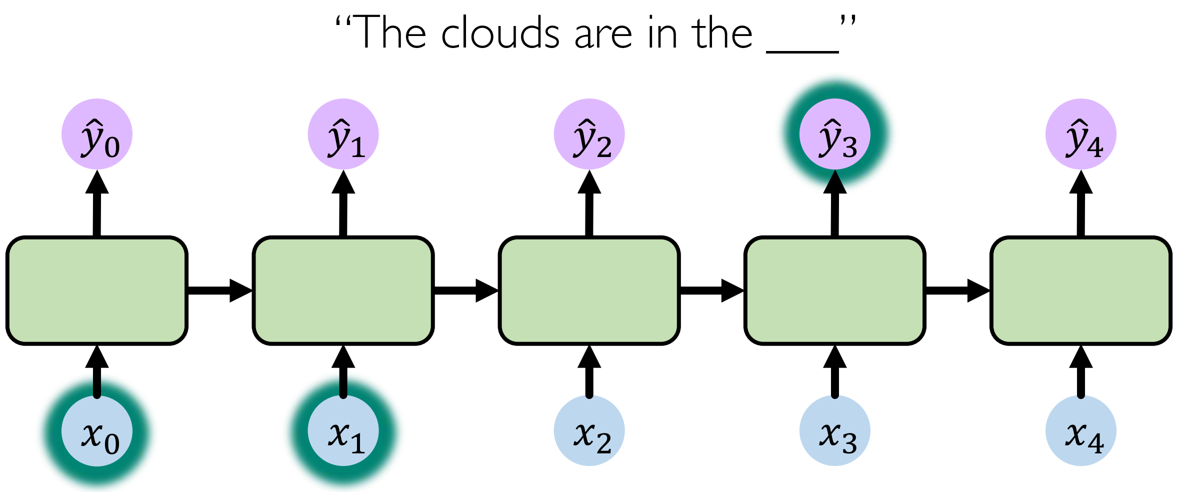
Multiply many **small numbers** together

↓

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

↓

Bias parameters to capture short-term dependencies



The problem of long-term dependencies

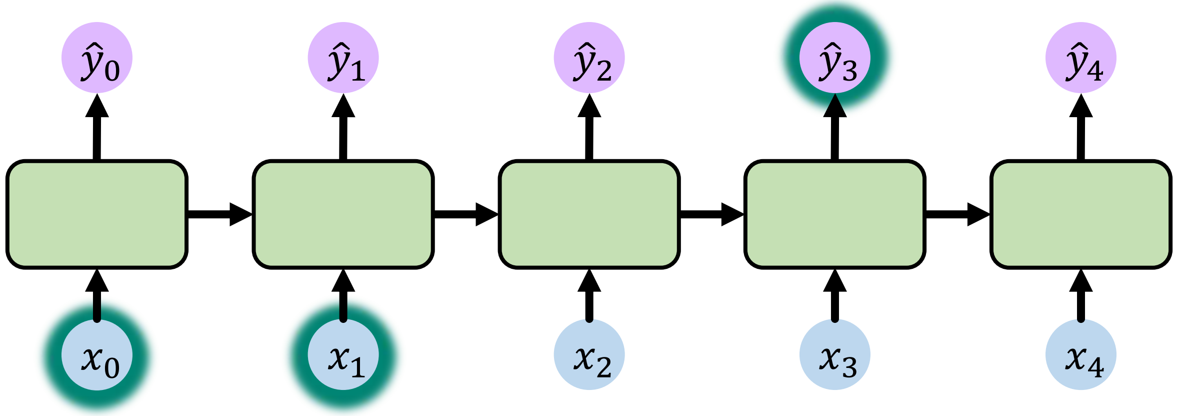
Why are vanishing gradients a problem?

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?

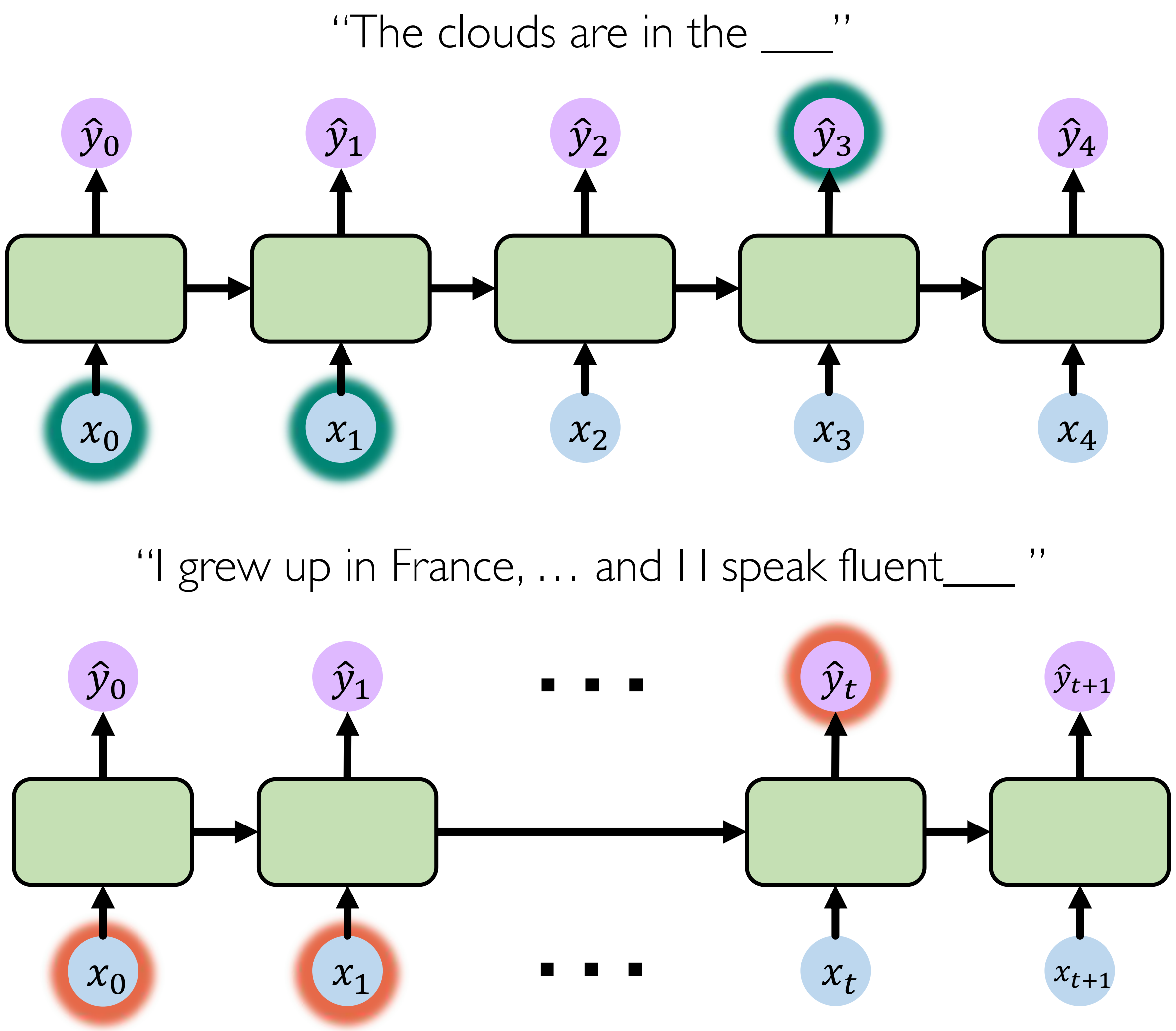
Multiply many **small numbers** together

↓

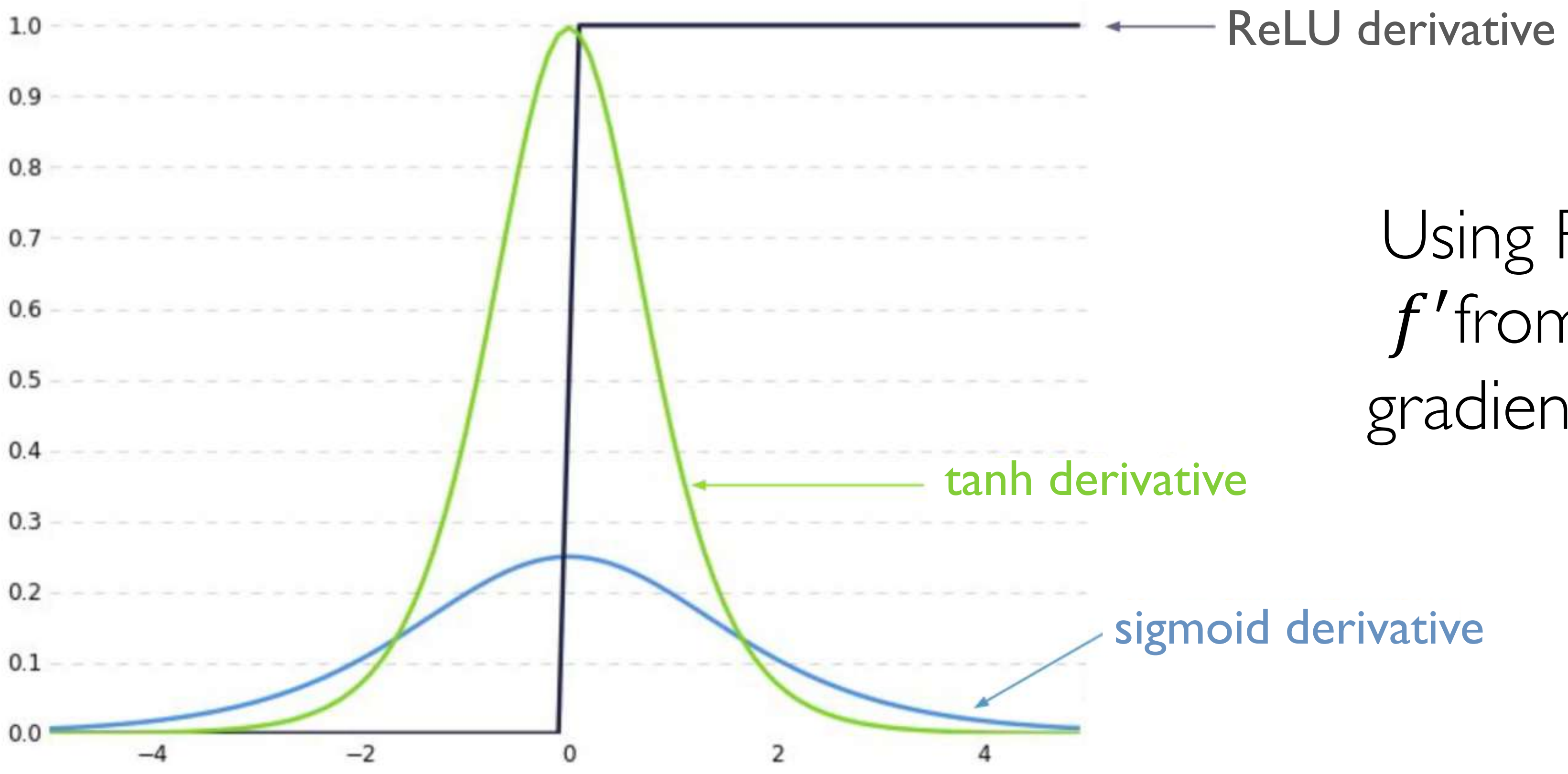
Errors due to further back time steps have smaller and smaller gradients

↓

Bias parameters to capture short-term dependencies



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

gated cell

LSTM, GRU, etc.

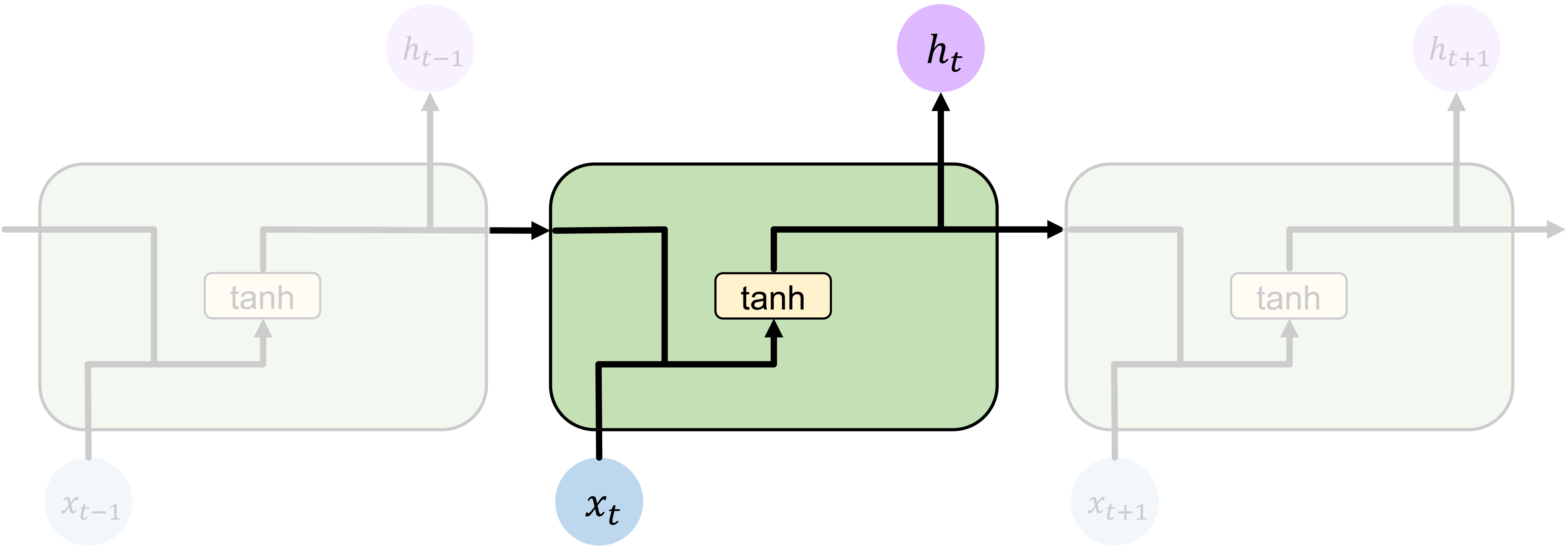
Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

Adapted from H. Suresh, 6.S191 2018

Long Short Term Memory (LSTM) Networks

Standard RNN

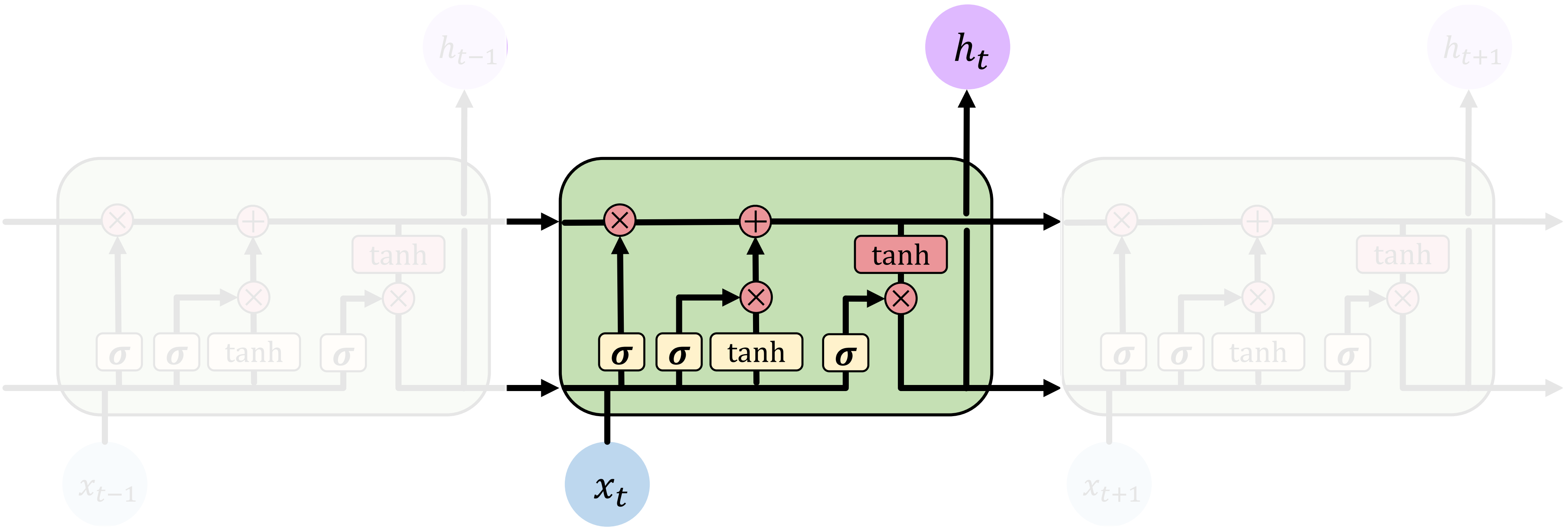
In a standard RNN, repeating modules contain a **simple computation node**



[2]

Long Short Term Memory (LSTMs)

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

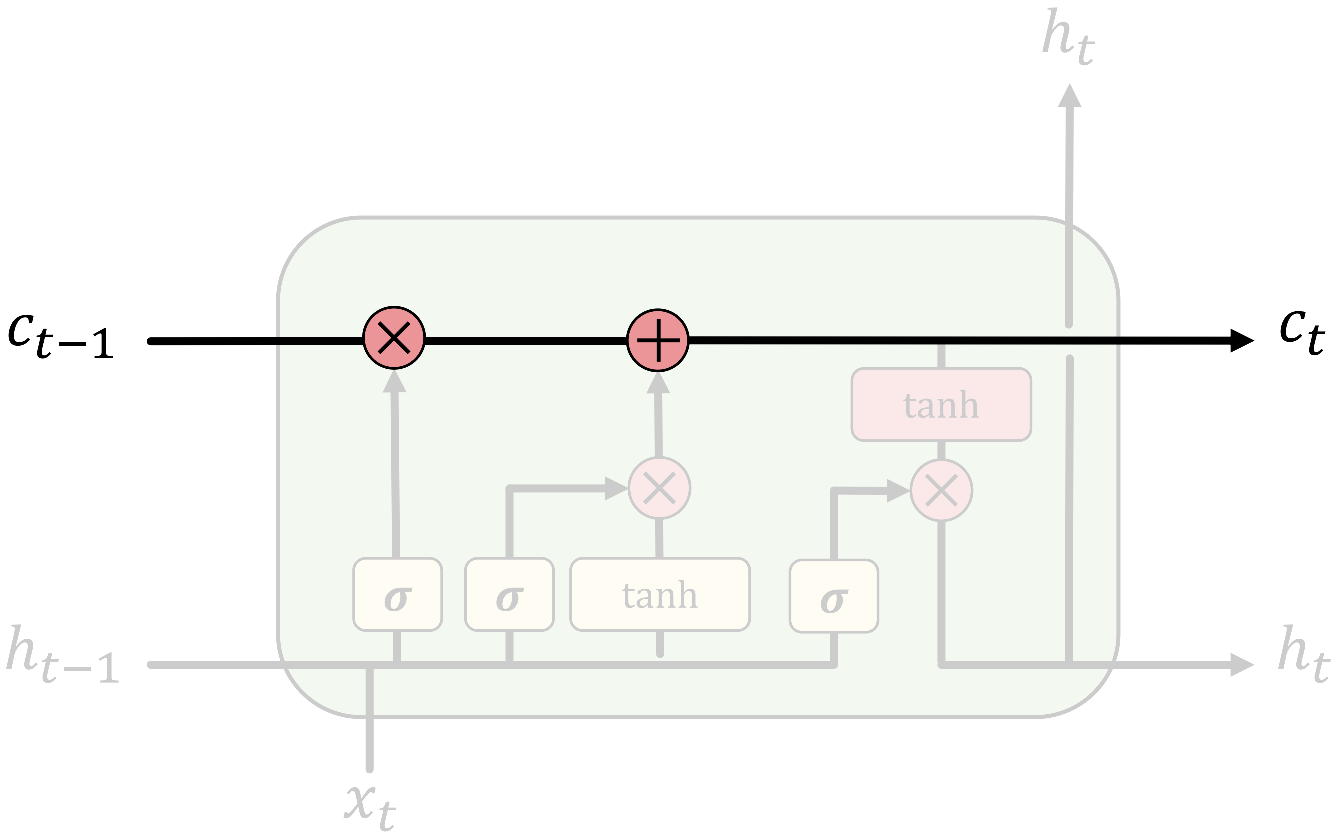


LSTM cells are able to track information throughout many timesteps

Hochreiter & Schmidhuber, 1997 [2, 5]

Long Short Term Memory (LSTMs)

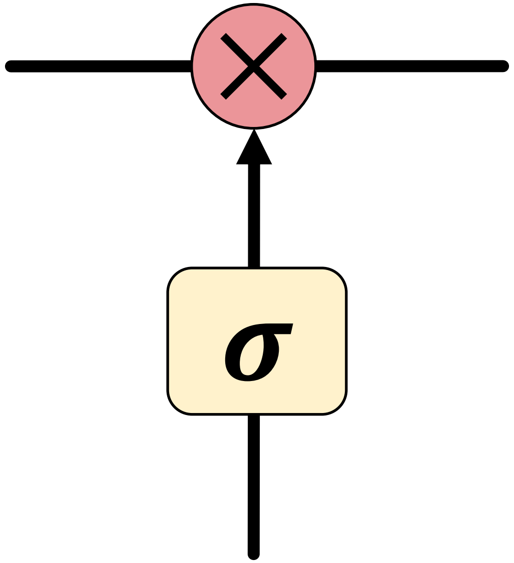
LSTMs maintain a **cell state** c_t where it's easy for information to flow



[2, 5]

Long Short Term Memory (LSTMs)

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

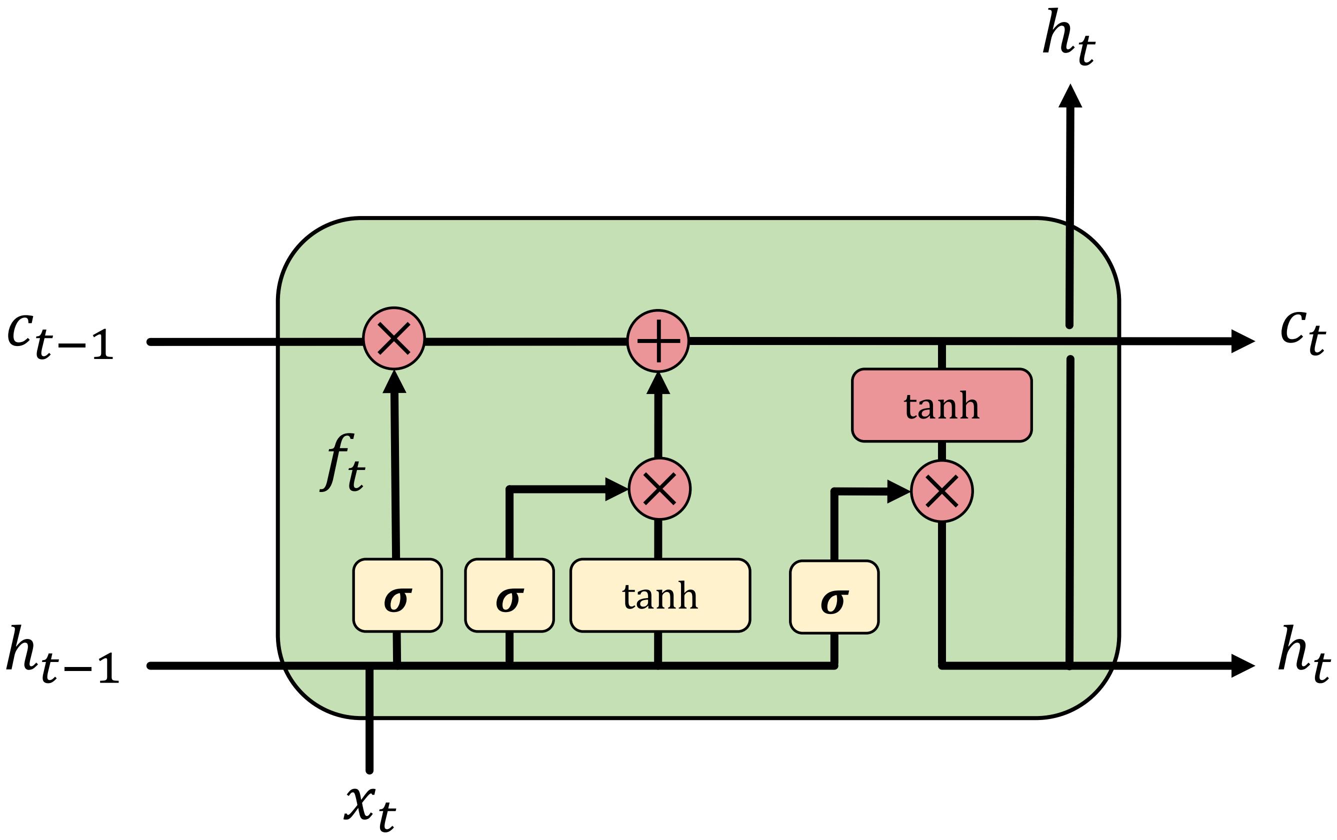


Gates optionally let information through, via a sigmoid neural net layer and pointwise multiplication

[2, 5]

Long Short Term Memory (LSTMs)

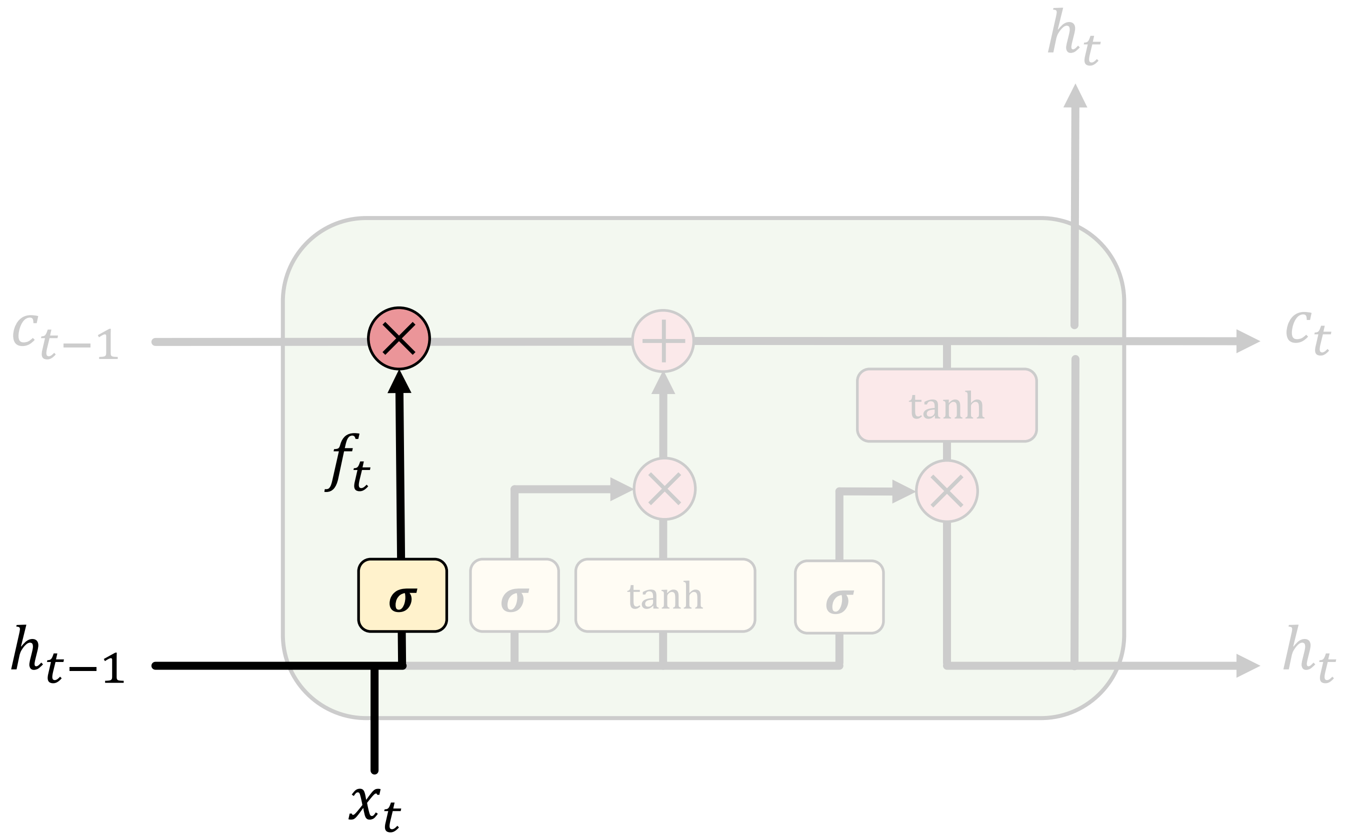
How do LSTMs work?



[2, 5]

Long Short Term Memory (LSTMs)

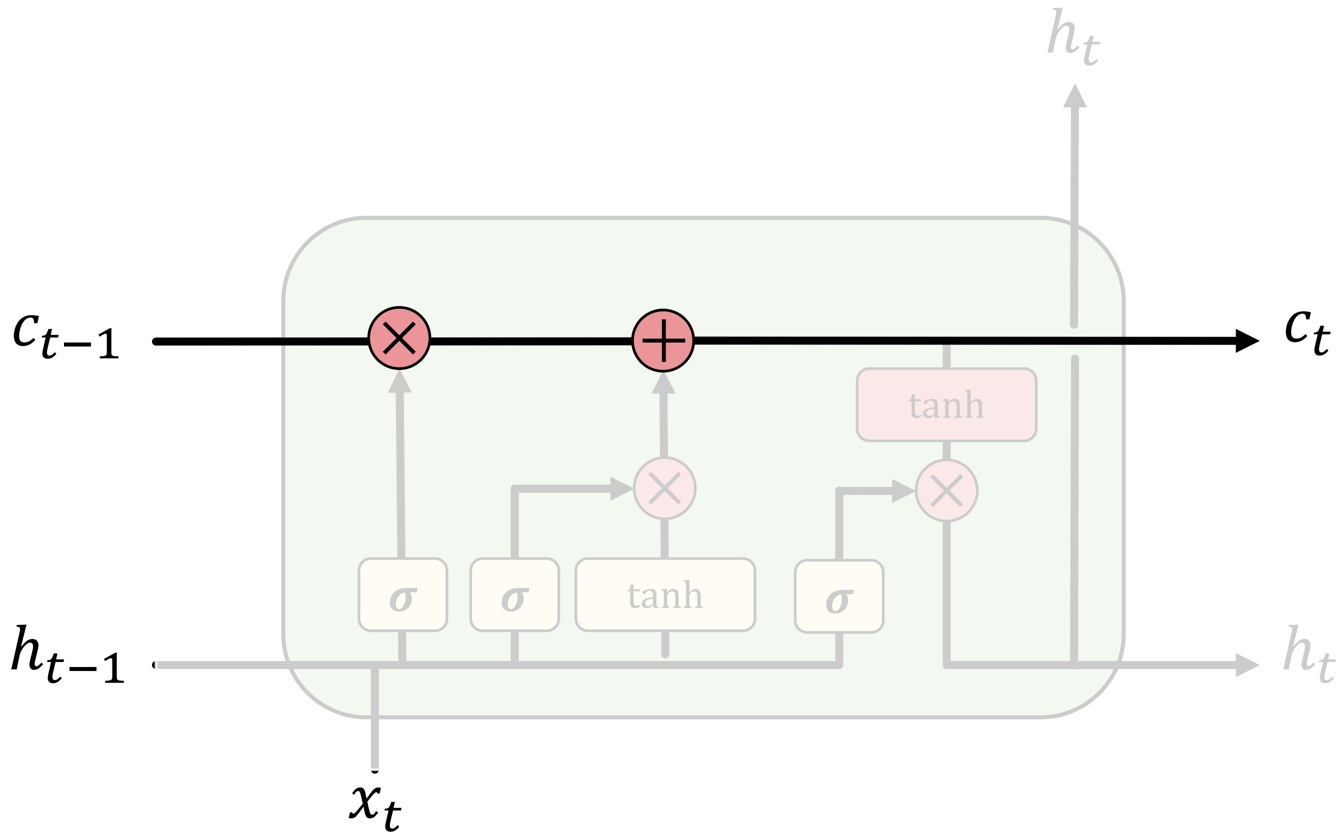
LSTMs **forget irrelevant** parts of the previous state



[2, 5]

Long Short Term Memory (LSTMs)

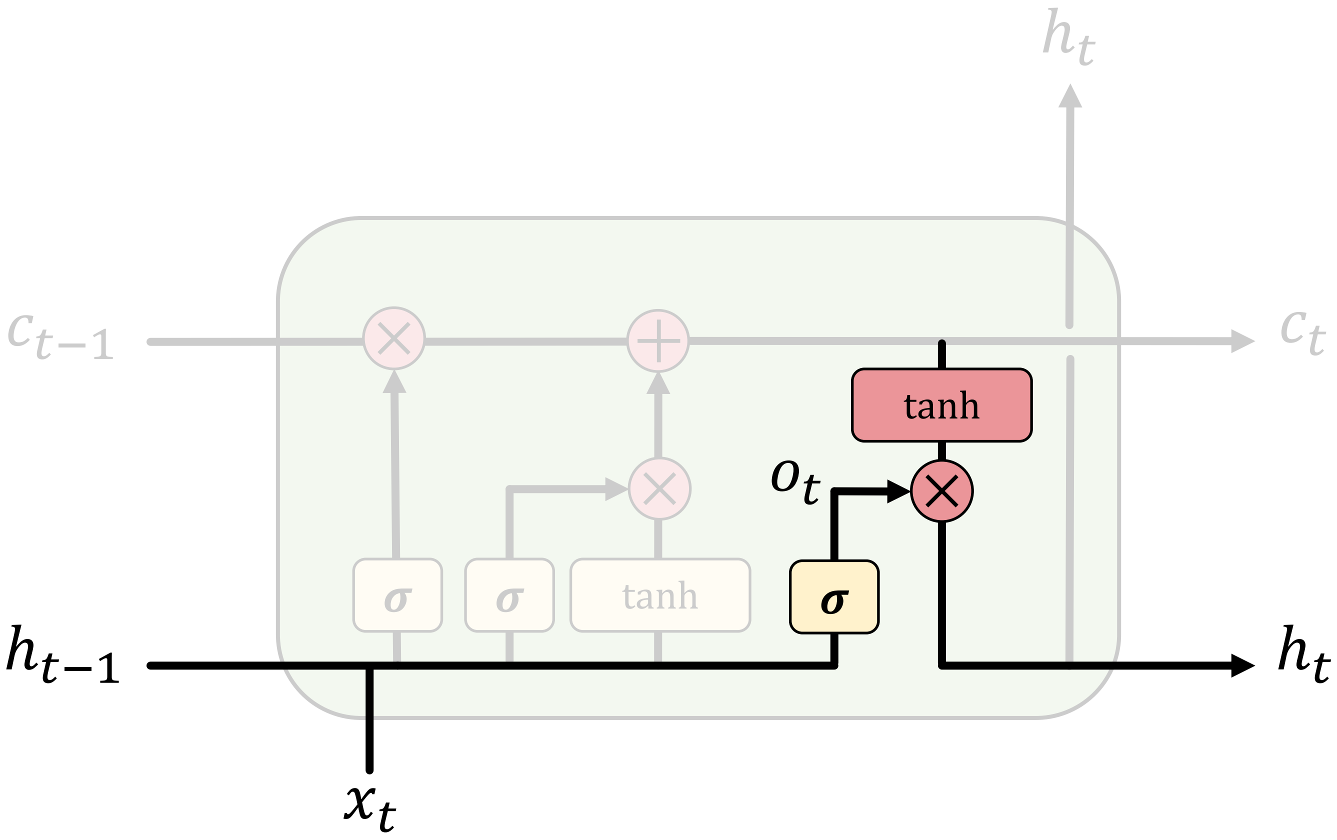
LSTMs **selectively update** cell state values



[2, 5]

Long Short Term Memory (LSTMs)

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

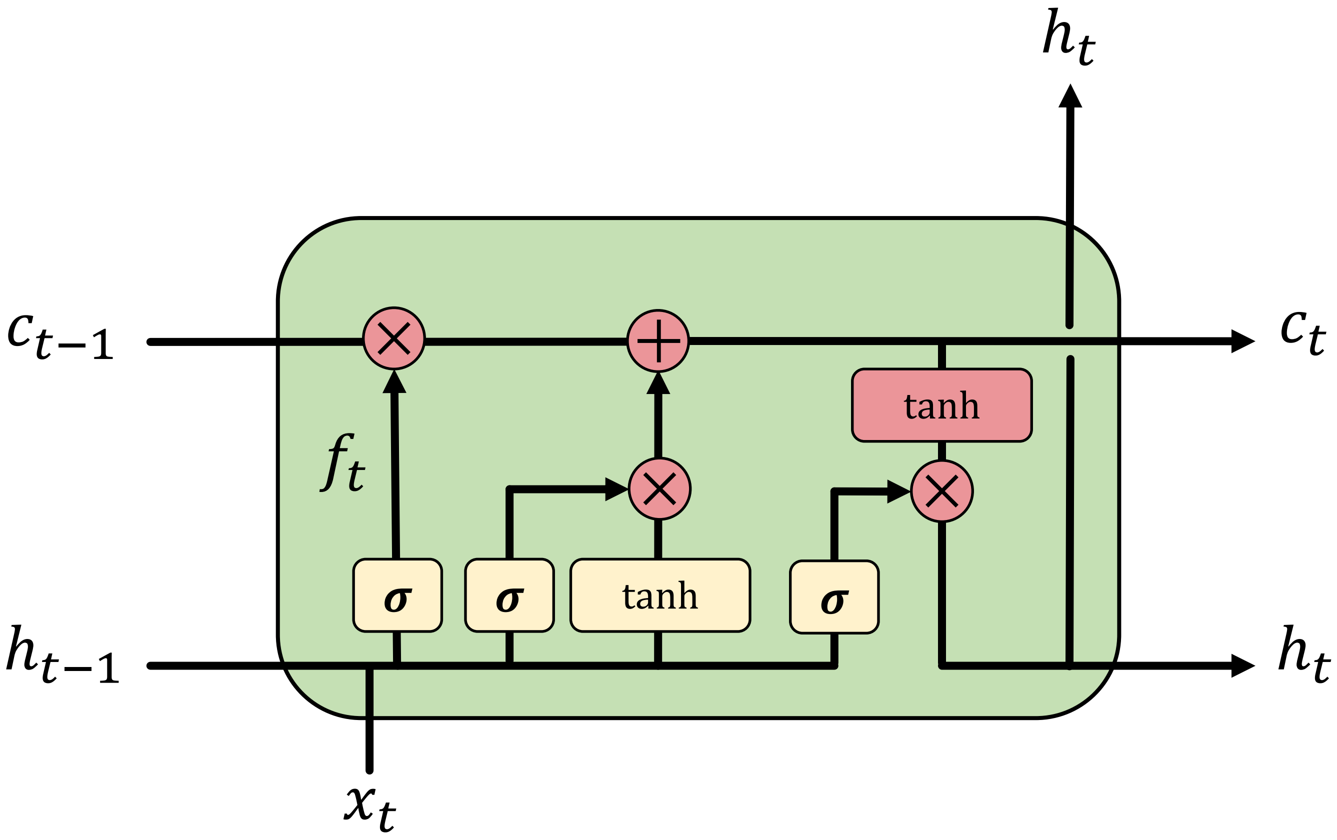


[2, 5]

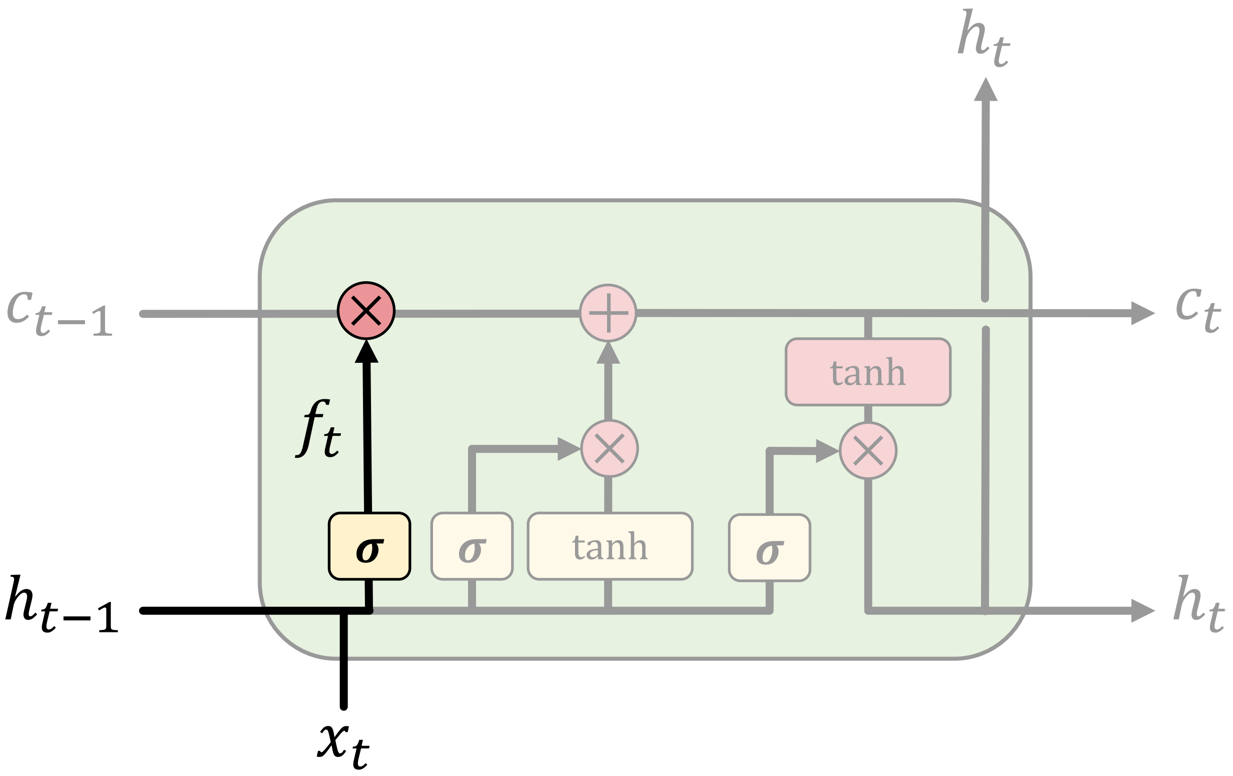
Long Short Term Memory (LSTMs)

How do LSTMs work?

1) Forget 2) Update 3) Output



LSTMs: forget irrelevant information



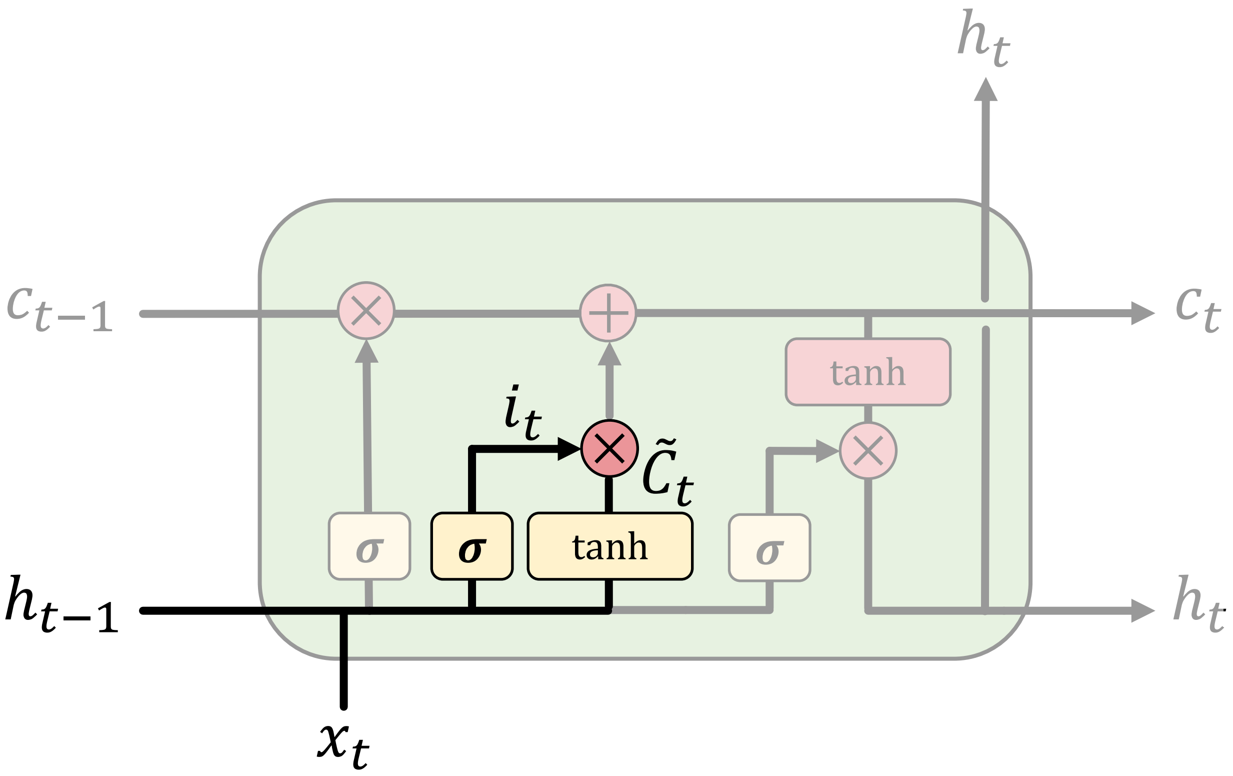
$$f_t = \sigma(\boldsymbol{W_i}[\,h_{t-1}, x_t\,] + b_f)$$

- 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.

[2, 5]

LSTMs: identify new information to be stored



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

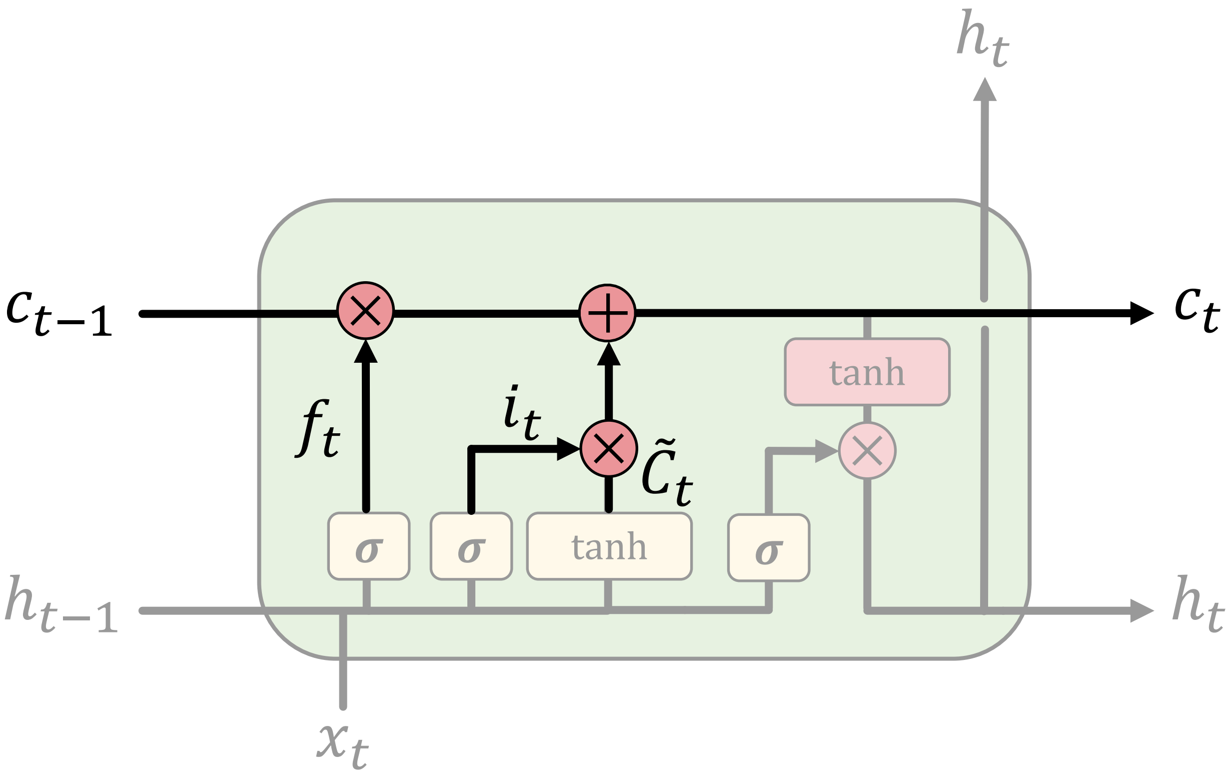
$$\tilde{C}_t = \tanh(\boldsymbol{W}_C[\, h_{t-1}, x_t \,] + b_C)$$

- 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.

[2, 5]

LSTMs: update cell state



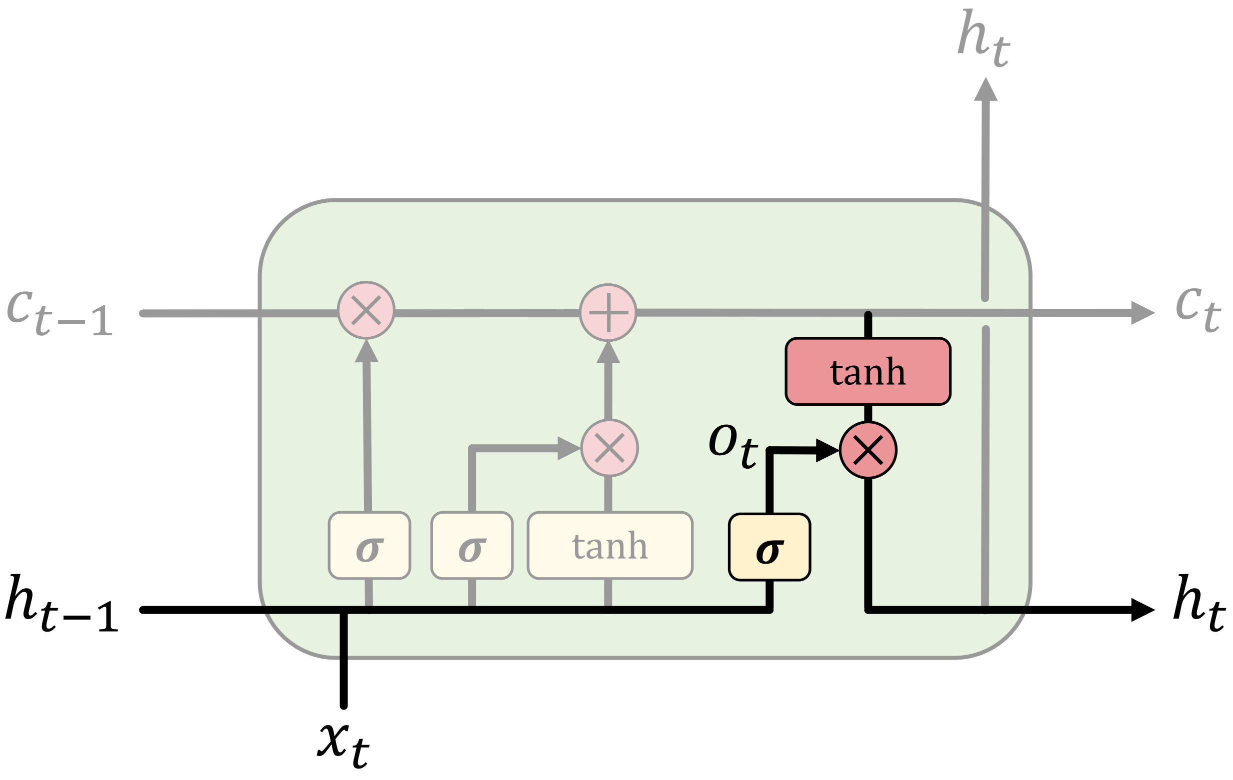
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- 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.

[2, 5]

LSTMs: output filtered version of cell state



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

$$h_t = o_t * \tanh(C_t)$$

- 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

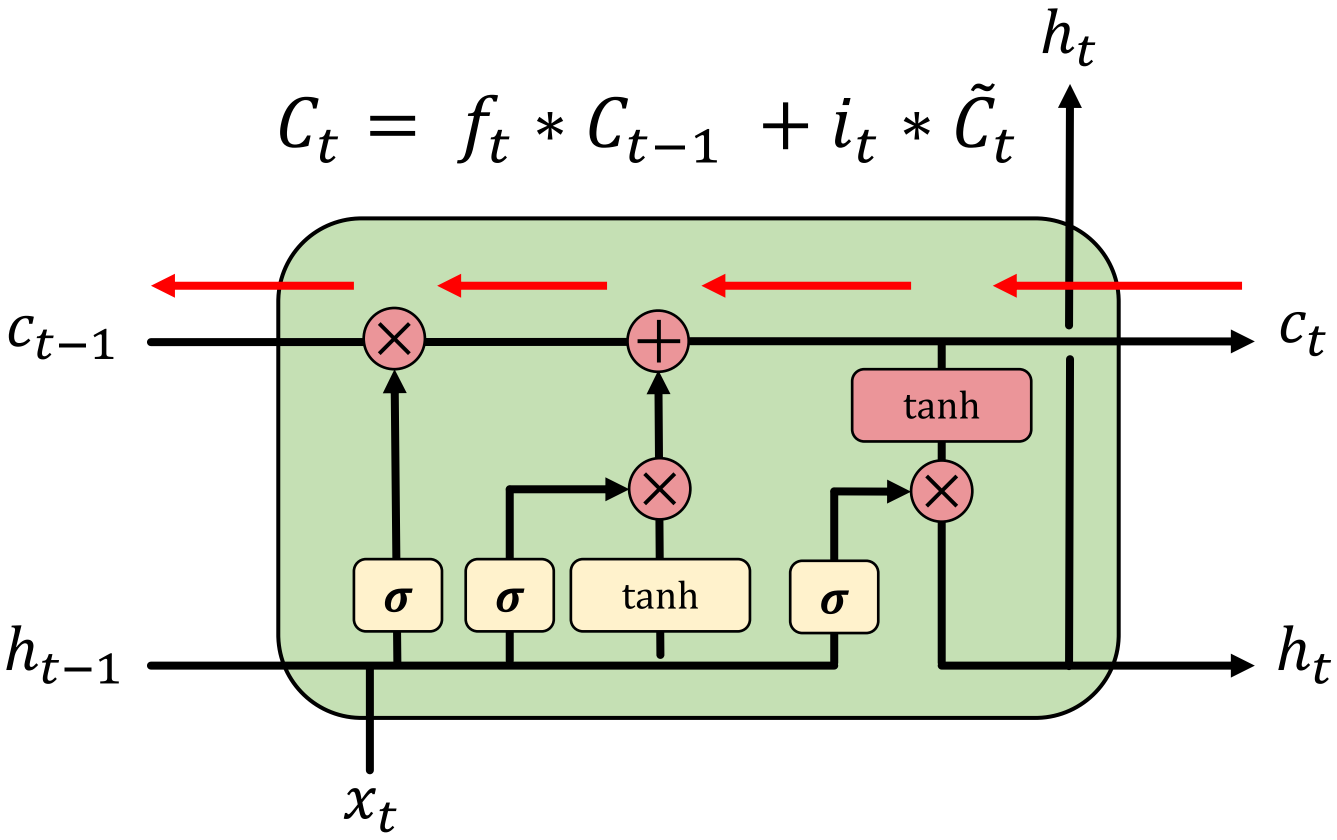
ex: Having seen a subject, may output information relating to a verb.

[2, 5]

LSTM gradient flow

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

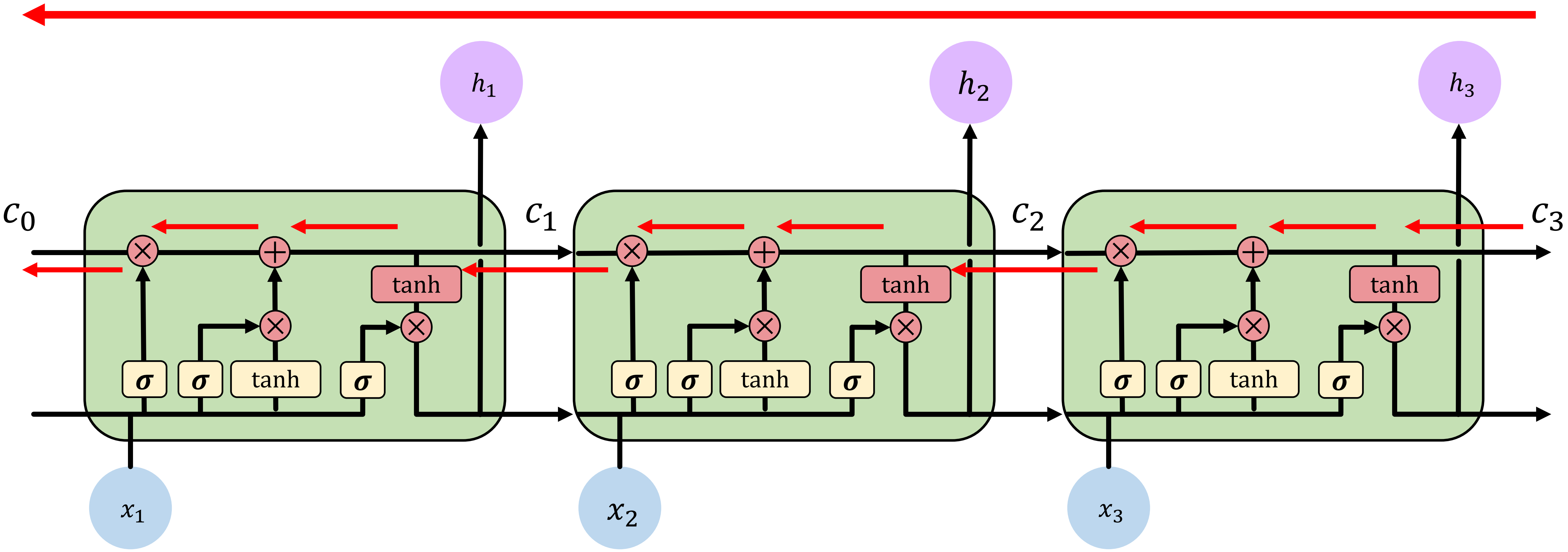
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



[2, 5]

LSTM gradient flow

Uninterrupted gradient flow!

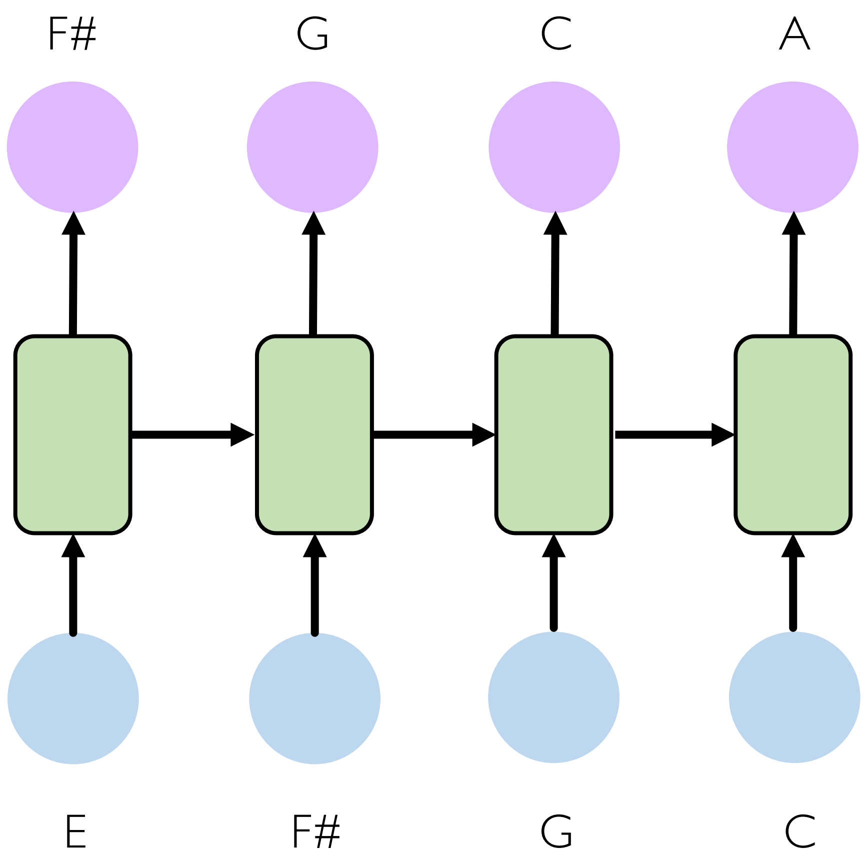


LSTMs: key concepts

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

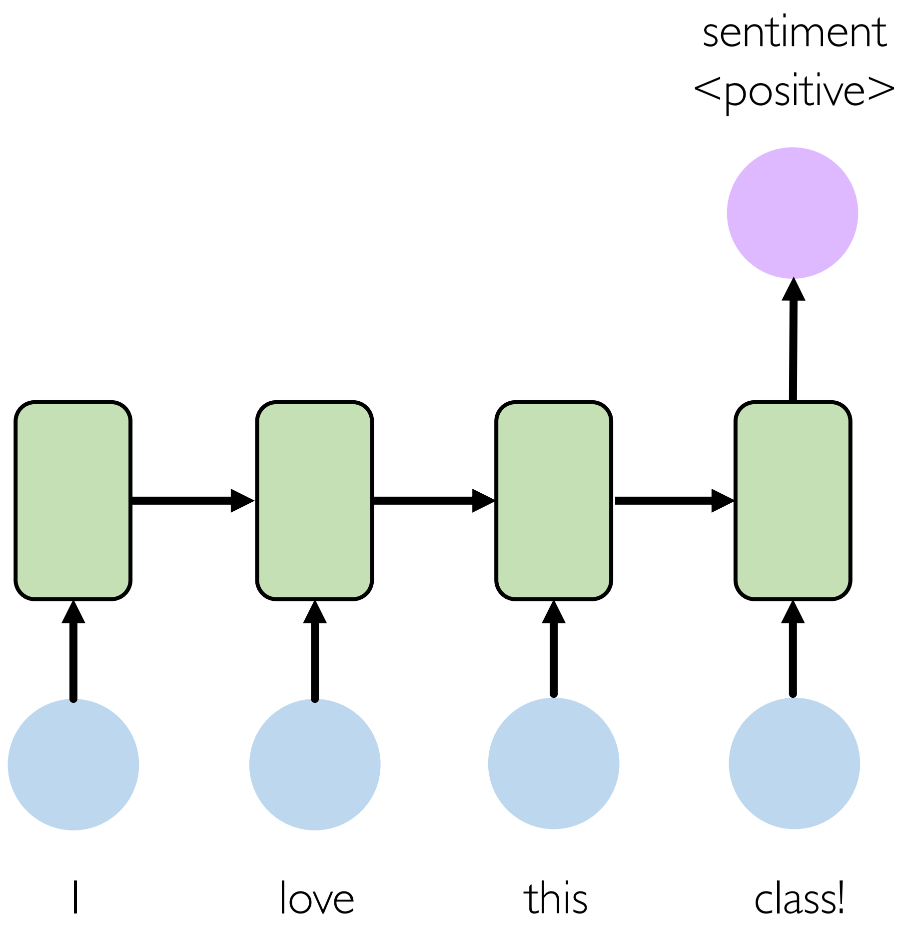
Output: next character in sheet music

6.S191 Lab!

Adapted from H. Suresh, 6.S191 2018

[6]

Example task: sentiment classification



Input: sequence of words

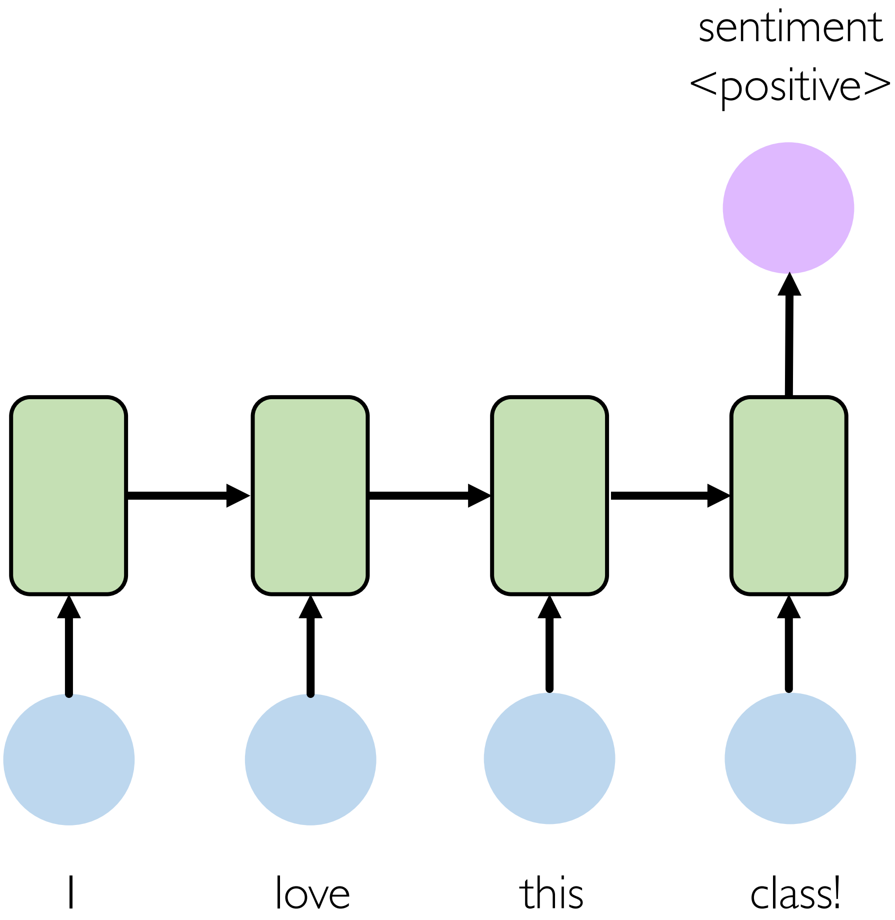
Output: probability of having positive sentiment

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

Adapted from H. Suresh, 6.S191 2018

[7]

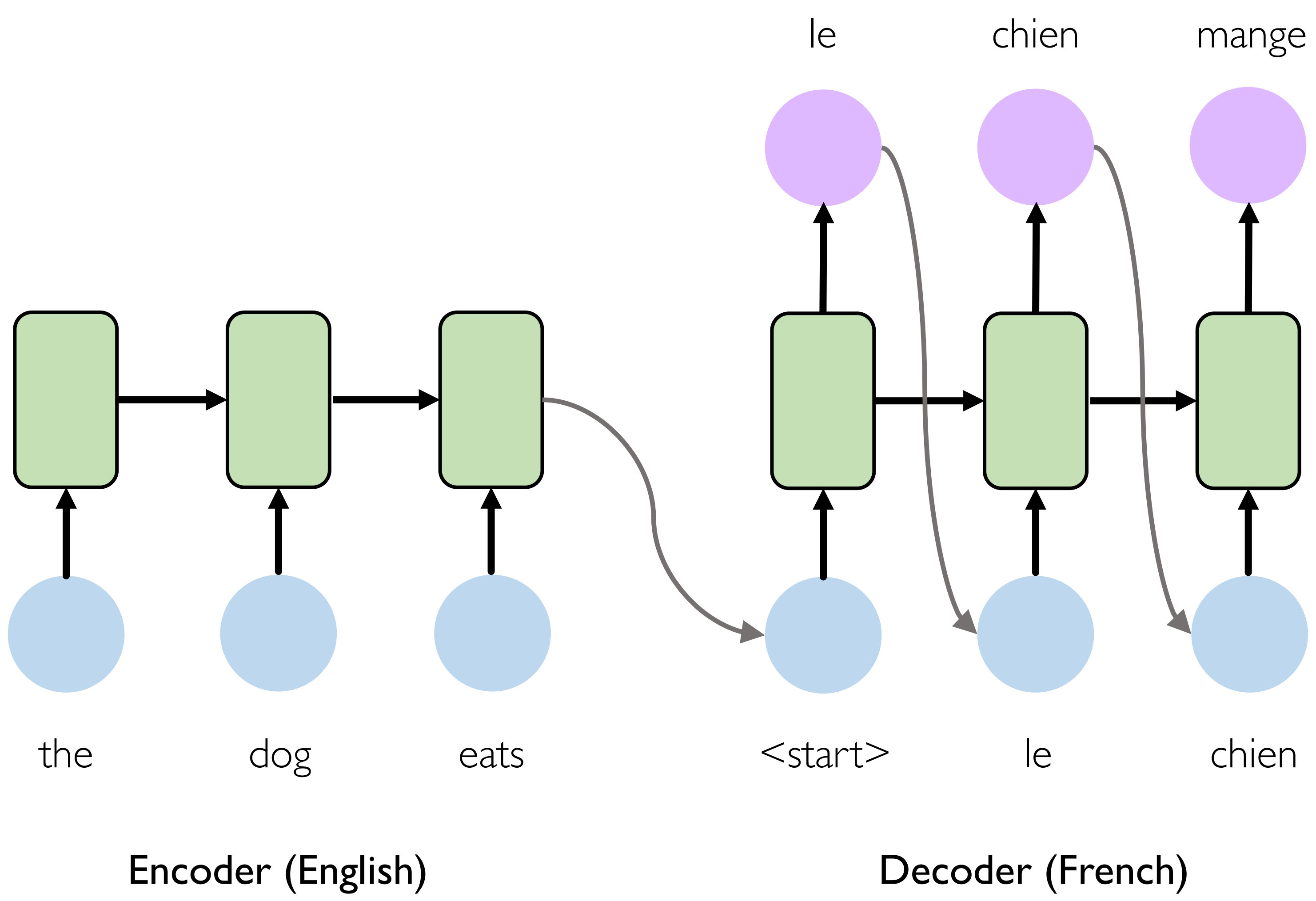
Example task: sentiment classification



Tweet sentiment classification

Adapted from H. Suresh, 6.S191 2018

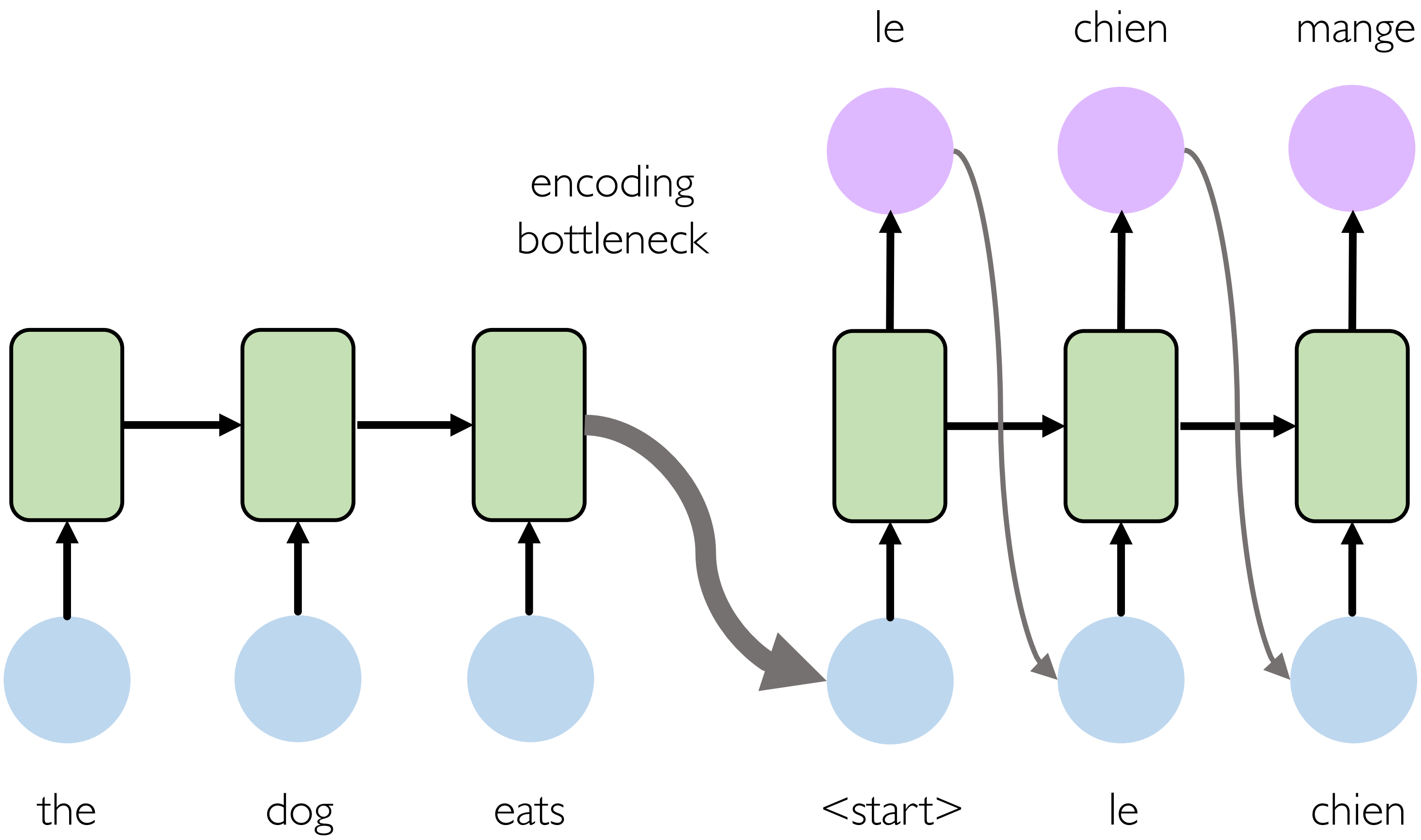
Example task: machine translation



Adapted from H. Suresh, 6.S191 2018

[8,9]

Example task: machine translation



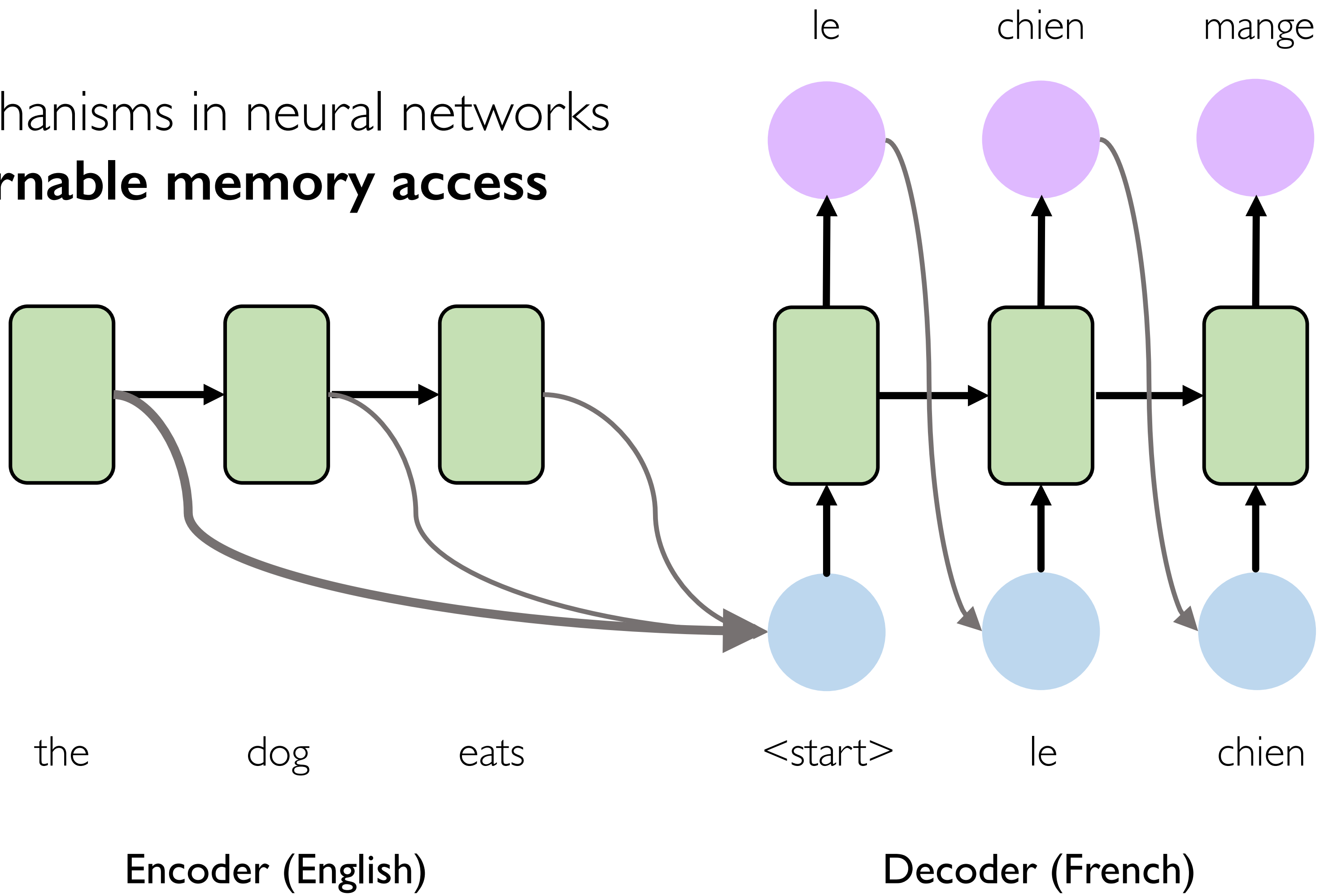
Encoder (English) **Decoder (French)**

Adapted from H. Suresh, 6.S191 2018

[8,9]

Attention mechanisms

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

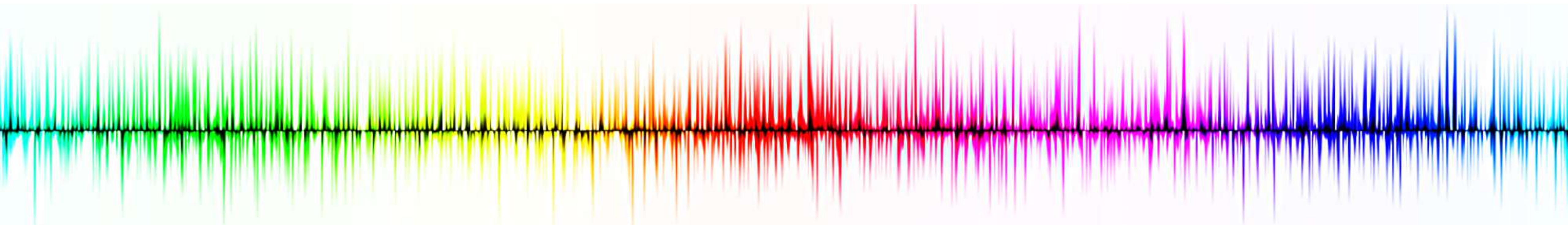


Adapted from H. Suresh, 6.S191 2018

[8,9]

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



Natural Language Processing

- Typical preprocessing steps of text data
  - ➢ Form vocabulary of words that maps words to a unique ID
  - ➢ Different criteria can be used to select which words are part of the vocabulary
  - ➢ Pick most frequent words and ignore uninformative words from a user-defined short list (ex.: “ the ”, “ a ”, etc.)
  - ➢ All words not in the vocabulary will be mapped to a special “out-of-vocabulary’
- Typical vocabulary sizes will vary between 10,000 and 250,000

http://www.cs.cmu.edu/~rsalakhu/10707/

Vocabulary

- Example:

Word	w
"the"	1
"and"	2
"dog"	3
"."	4
"OOV"	5

"the"	1
"cat"	5
"and"	2
"the"	1
"dog"	3
"play"	5
"."	4

- We will note word IDs with the symbol w
  - ➢ we can think of w as a categorical feature for the original word
  - ➢ we will sometimes refer to w as a word, for simplicity

One-Hot Encoding

- From its word ID, we get a basic representation of a word through the one-hot encoding of the ID

  - the one-hot vector of an ID is a vector filled with 0s, except for a 1 at the position associated with the ID
  - For vocabulary size D=10, the one-hot vector of word ID w=4 is:

$$e(w) = [\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ ]$$

  - A one-hot encoding makes no assumption about word similarity
  - This is a natural representation to start with, though a poor one

One-Hot Encoding

- The major problem with the one-hot representation is that it is very high-dimensional
  - ➢ the dimensionality of $e(w)$ is the size of the vocabulary
  - ➢ a typical vocabulary size is ≈100,000
  - ➢ a window of 10 words would correspond to an input vector of at least 1,000,000 units!
- This has 2 consequences:
  - ➢ vulnerability to overfitting (millions of inputs means millions of parameters to train)
  - ➢ computationally expensive

Continuous Representation of Words

- word embeddings(word vectors, word representations)

  Each word w is associated with a real-valued vector C(w)

- Ex : Word2Vec - a method for learning word vectors

Word	w	$C(w)$
" the "	1	[0.6762, -0.9607, 0.3626, -0.2410, 0.6636]
" a "	2	[0.6859, -0.9266, 0.3777, -0.2140, 0.6711]
" have "	3	[0.1656, -0.1530, 0.0310, -0.3321, -0.1342]
" be "	4	[0.1760, -0.1340, 0.0702, -0.2981, -0.1111]
" cat "	5	[0.5896, 0.9137, 0.0452, 0.7603, -0.6541]
" dog "	6	[0.5965, 0.9143, 0.0899, 0.7702, -0.6392]
" car "	7	[-0.0069, 0.7995, 0.6433, 0.2898, 0.6359]
...	...	...

References

- http://jiwonkim.org/awesome-rnn/ see references !!
- Klaus Greff, Rupesh Kumar Srivastava, Jan Koutnik, Bas R. Steunebrink, Jurgen Schmidhuber, LSTM: A Search Space Odyssey, arXiv:1503.04069
- Zachary C. Lipton, A Critical Review of Recurrent Neural Networks for Sequence Learning, arXiv:1506.00019
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, Vol. 9, No. 8, pp. 1735–1780, 1997.