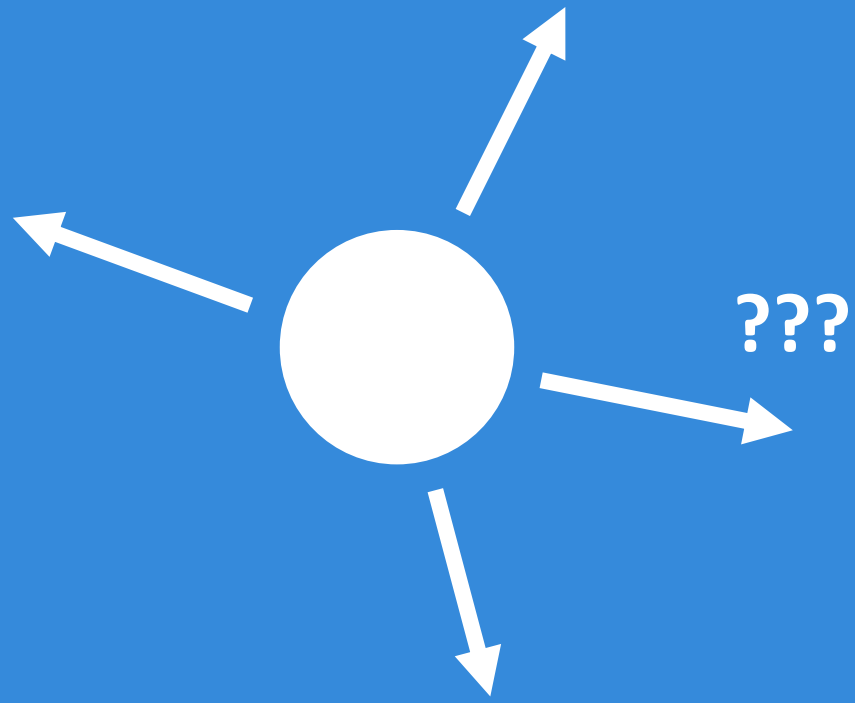
Sequence Modeling: Recurrent Neural Networks

References:

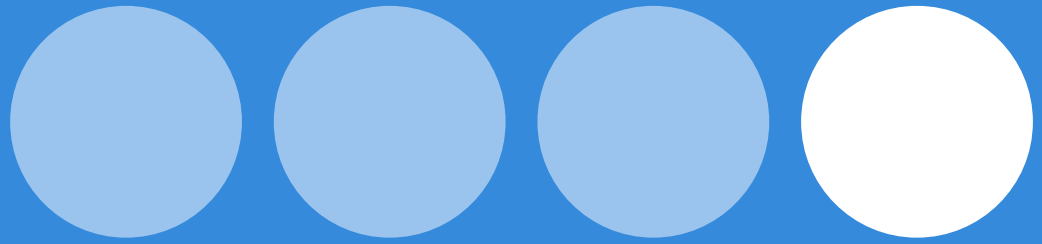
- introtodeeplearning.com
- Dive into Deep Learning, Chapter 8, https://d2l.ai/chapter_recurrent-neural-networks/index.html
- deeplearningbook.org
- deeplearningindaba.com

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

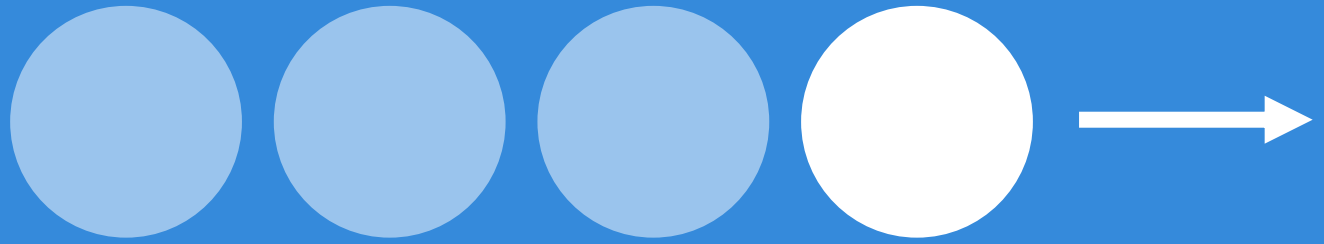
Given an image of a ball,
can you predict where it will go next?



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:

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 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 ___."

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 1 0 0 1 0 0 0 1 0 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 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:**

Adapted from H. Suresh, 6.S191 2018

Problem #3: no parameter sharing

[1 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 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 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 1 ...]

this morning

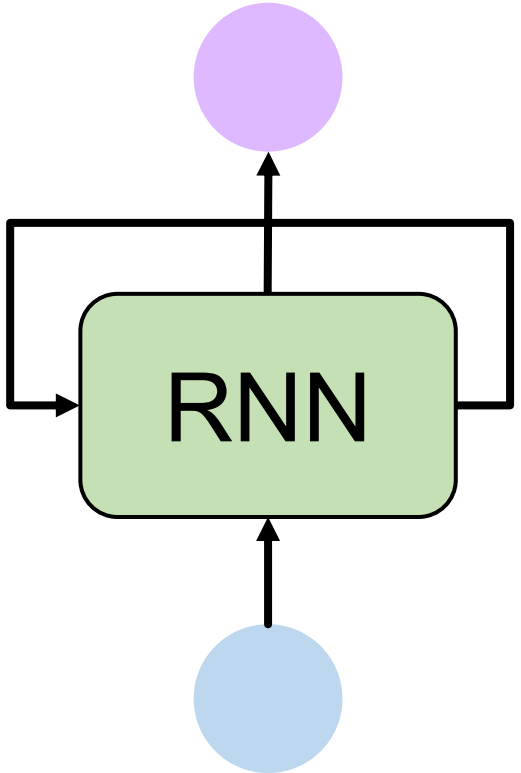
Things we learn about the sequence **won't transfer** if they appear **elsewhere** in the sequence.

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

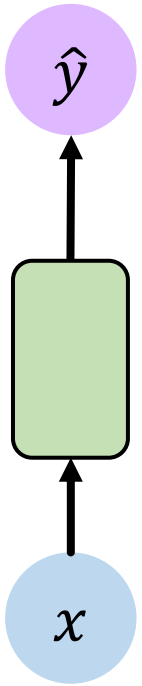


Today: **Recurrent Neural Networks (RNNs)** as an approach to sequence modeling problems

Adapted from H. Suresh, 6.S191 2018

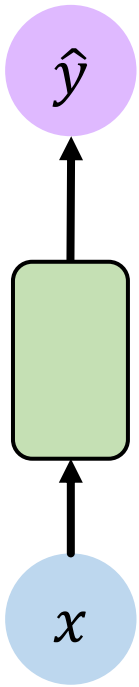
Recurrent Neural Networks (RNNs)

Standard feed-forward neural network

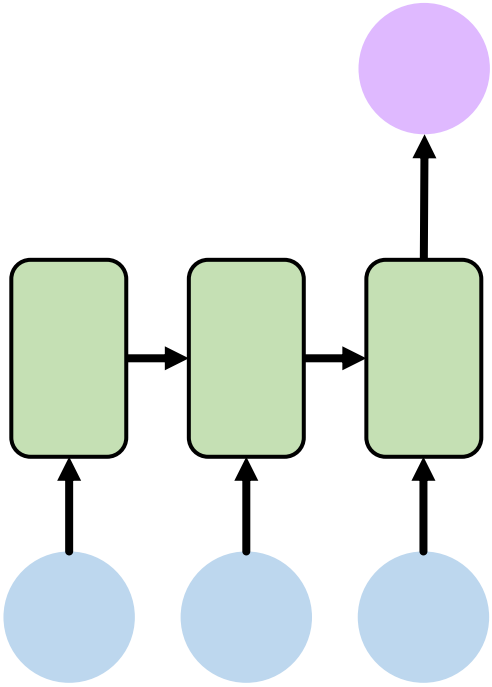


One to One
"Vanilla" neural network

Recurrent neural networks: sequence modeling

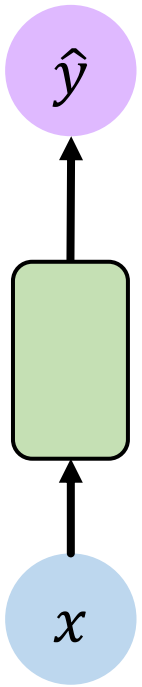


One to One
"Vanilla" neural network

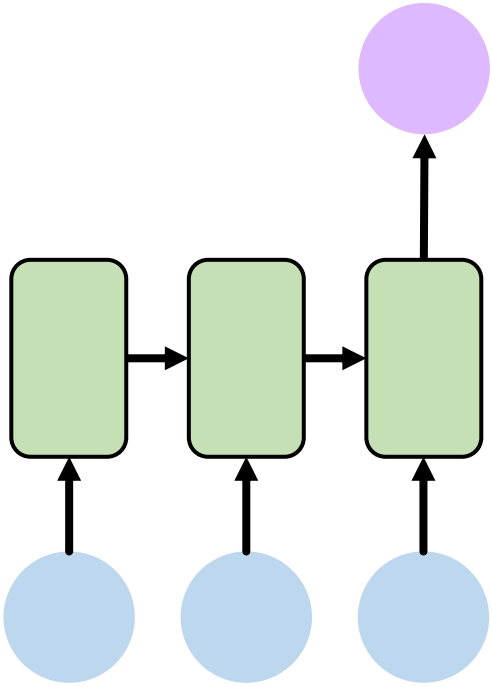


Many to One
Sentiment Classification

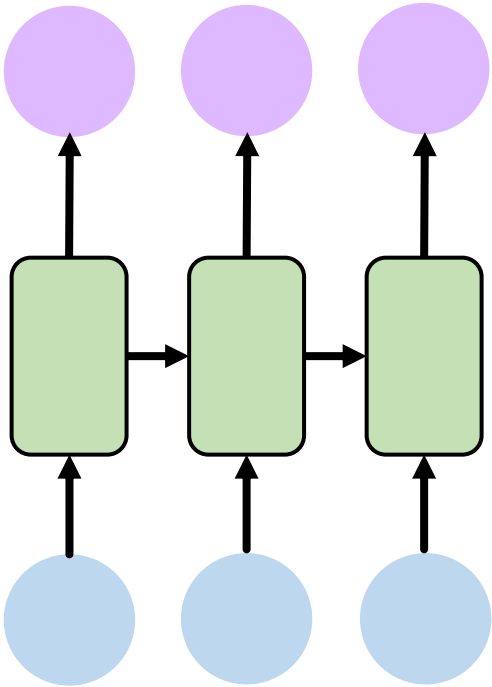
Recurrent neural networks: sequence modeling



One to One
"Vanilla" neural network

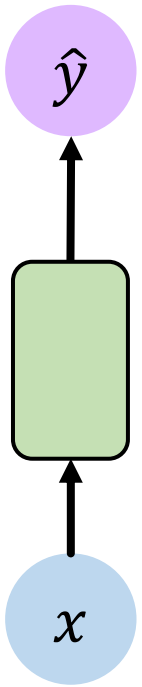


Many to One
Sentiment Classification

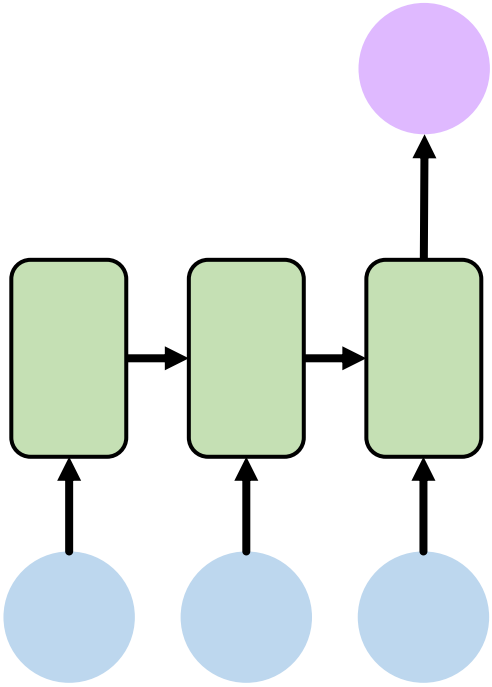


Many to Many
Music Generation

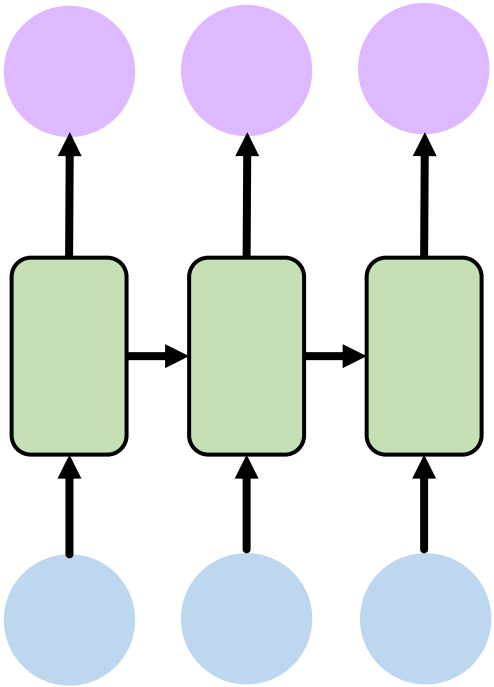
Recurrent neural networks: sequence modeling



One to One
"Vanilla" neural network



Many to One
Sentiment Classification

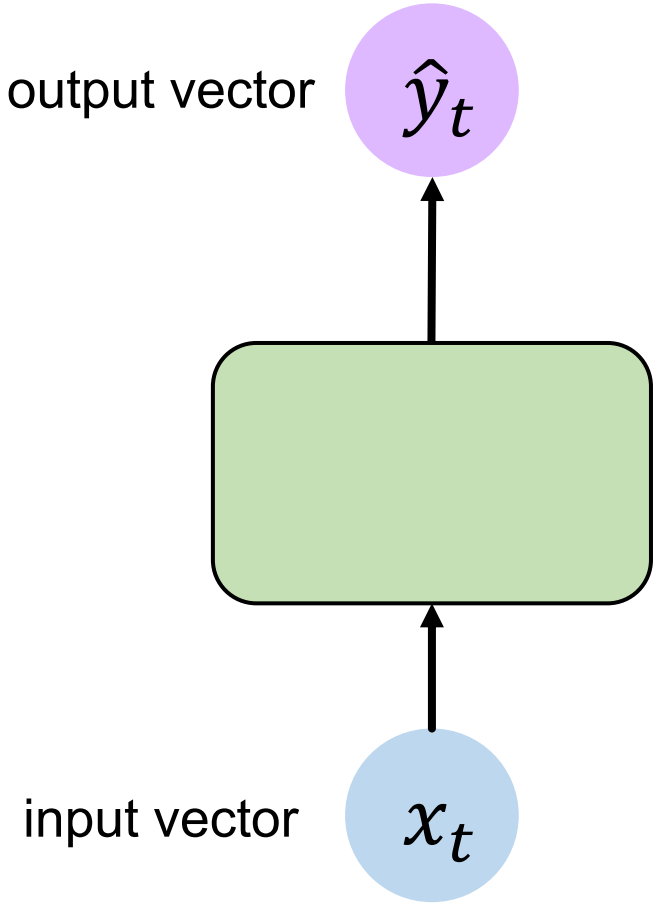


Many to Many
Music Generation

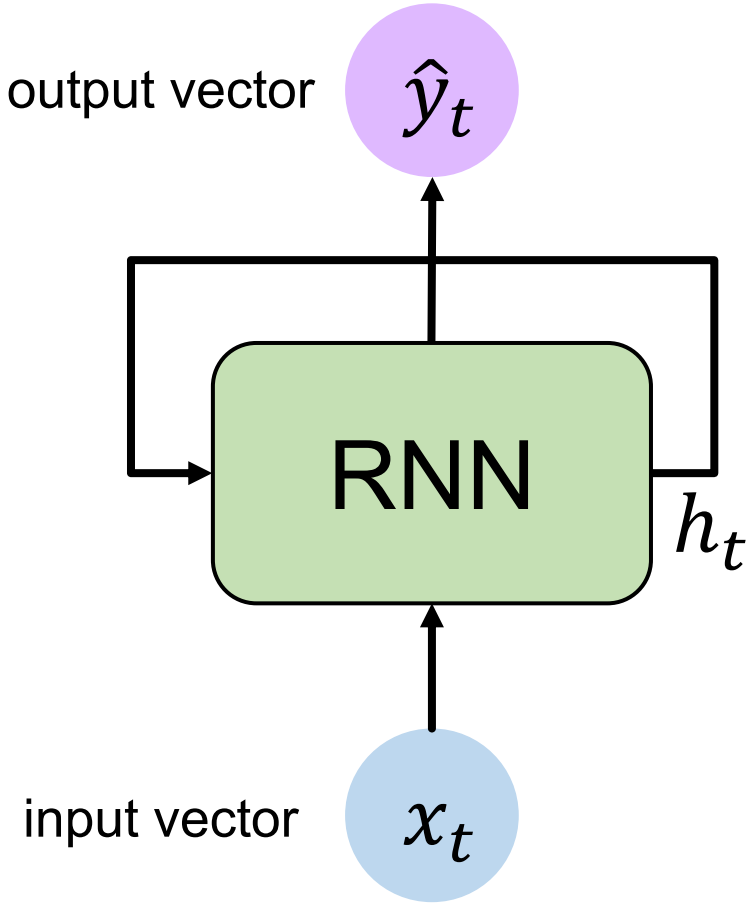
… and many other architectures and applications

[1]

A standard "vanilla" neural network

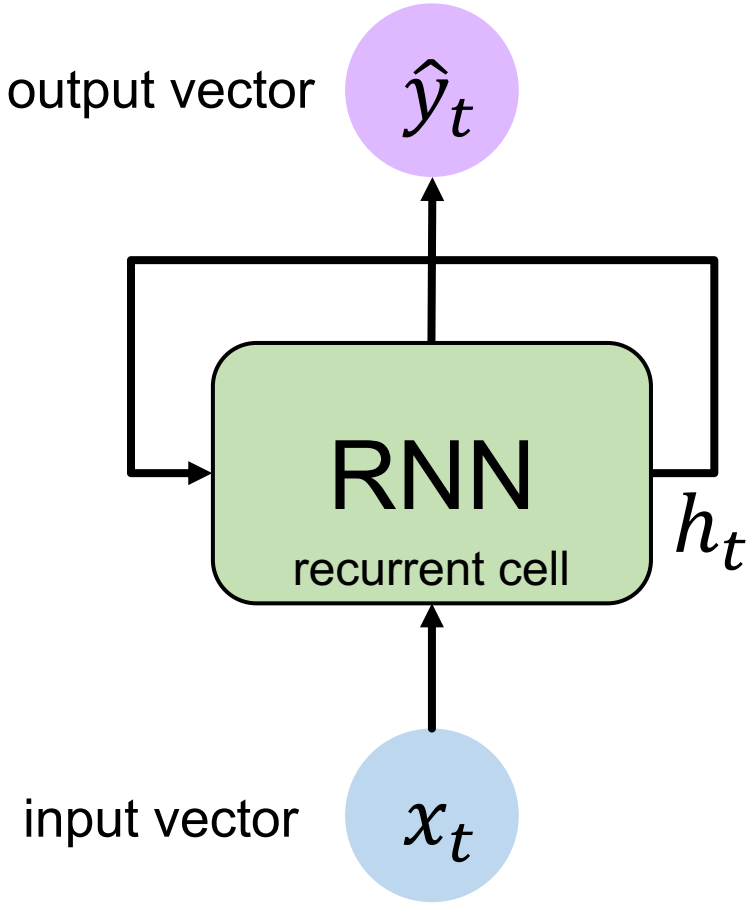


A recurrent neural network (RNN)

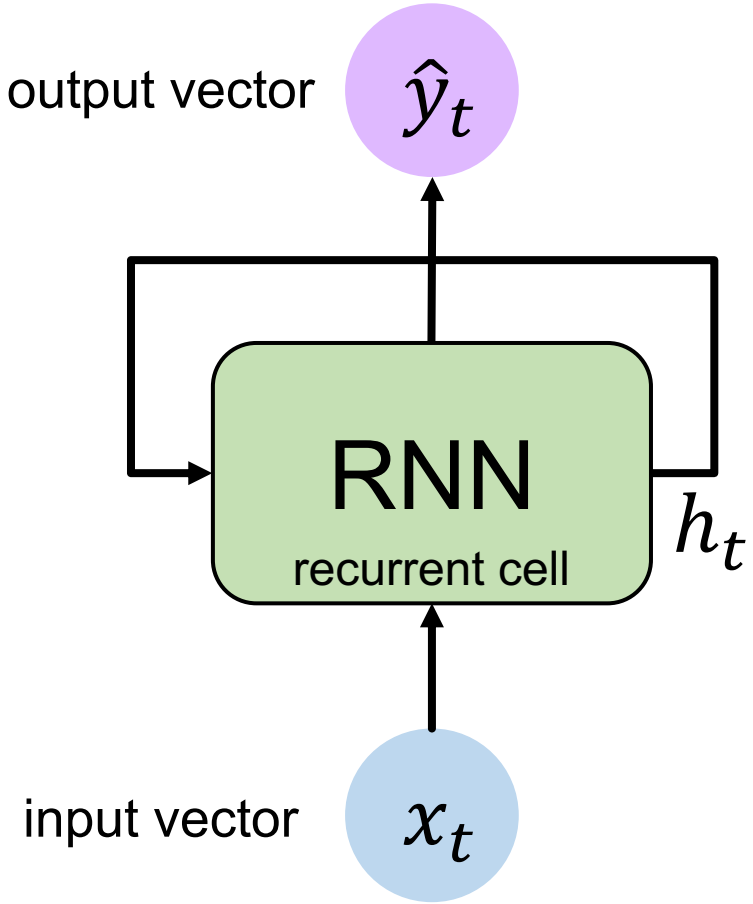


[2]

A recurrent neural network (RNN)

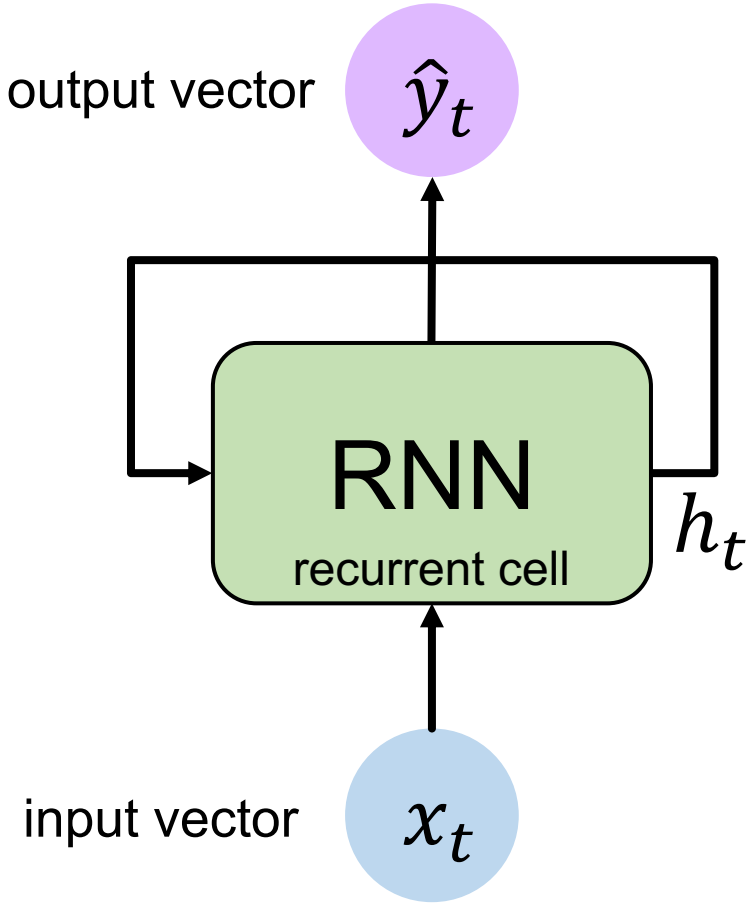


A recurrent neural network (RNN)



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:

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

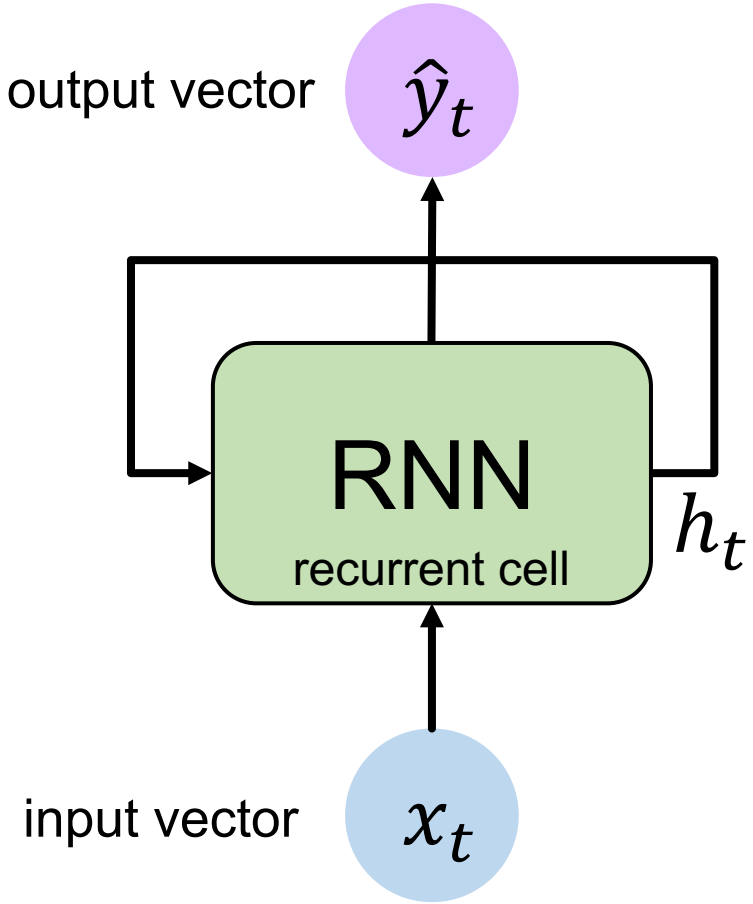
cell state

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:

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

new state

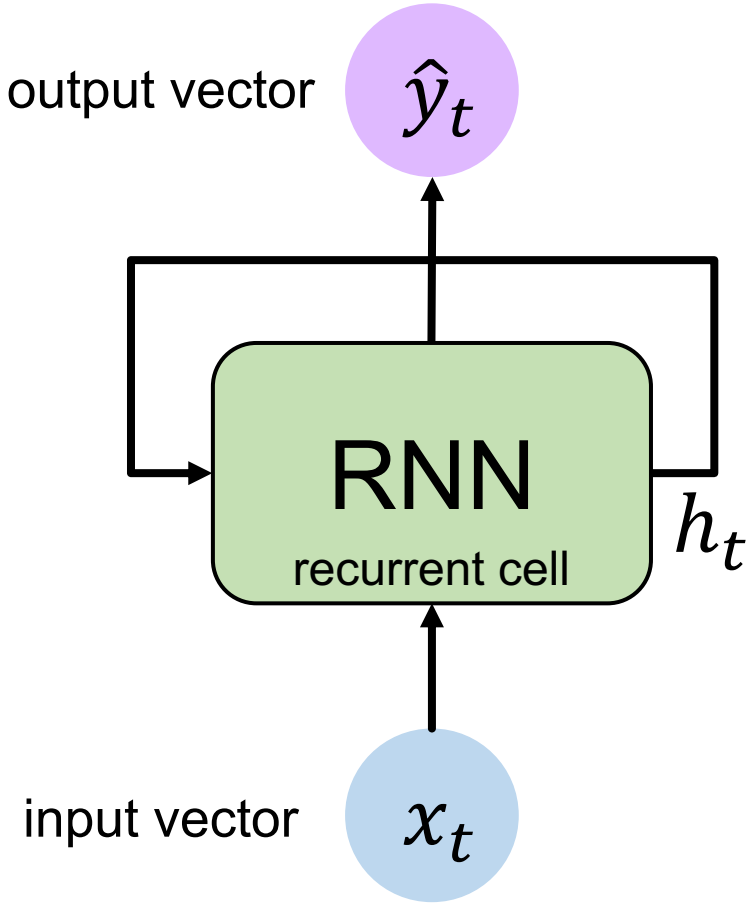
function parameterized by W

old state

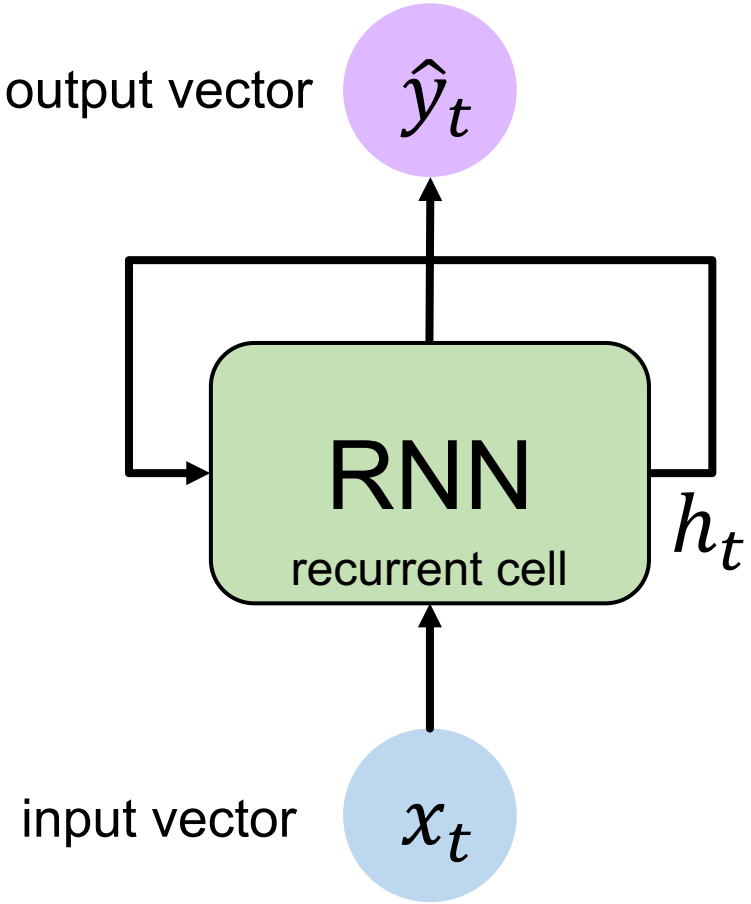
input vector at time step *t*

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

RNN state update and output

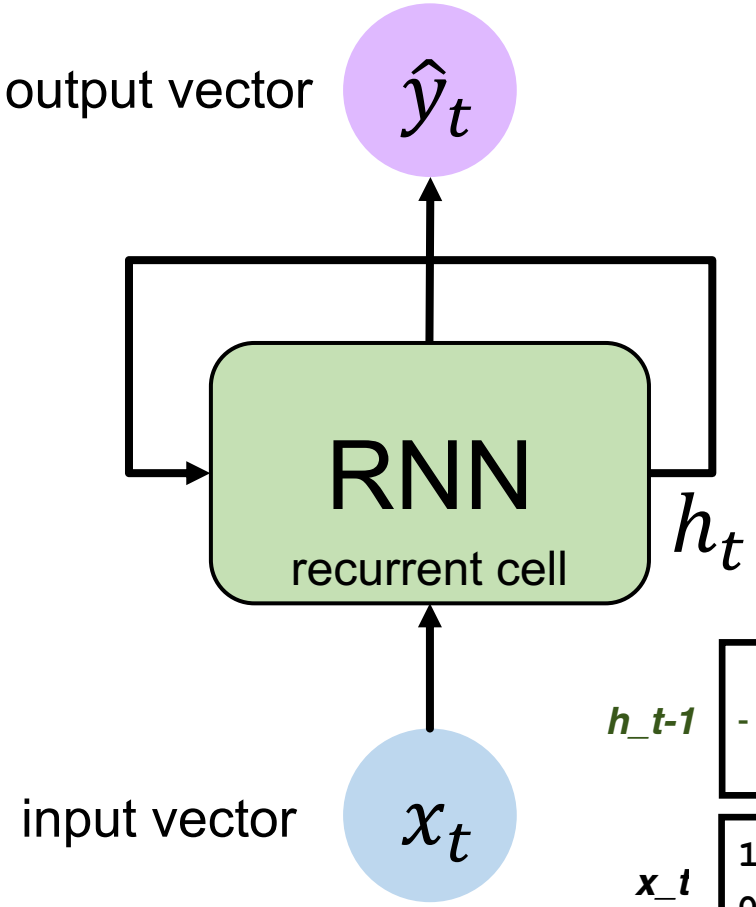


RNN state update and output



Input Vector

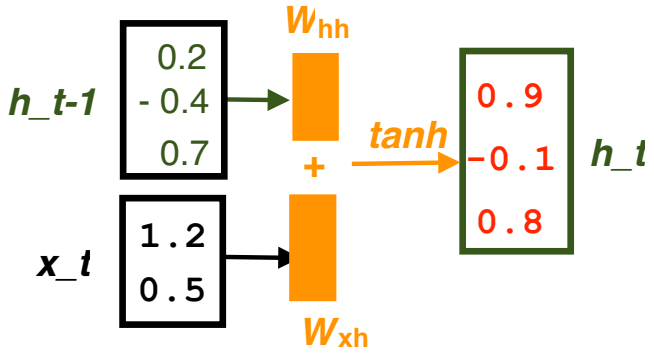
RNN state update and output



Update Hidden State

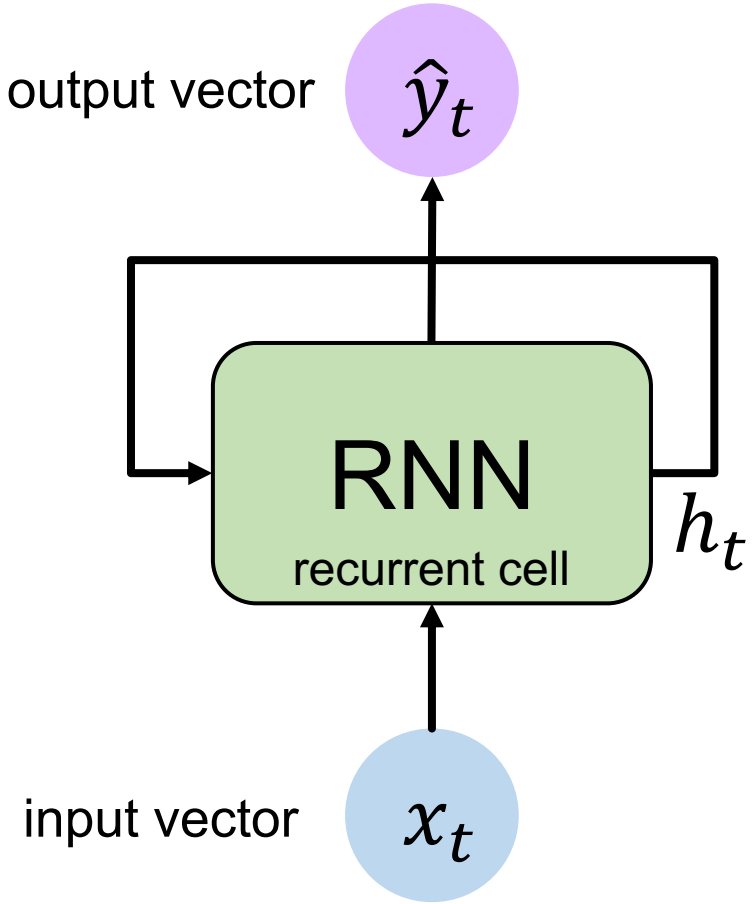
$$h_t = \tanh(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t)$$

$$= \tanh(w\,[h_{t-1}, x_t] + b)$$



Input Vector

RNN state update and output



Output Vector

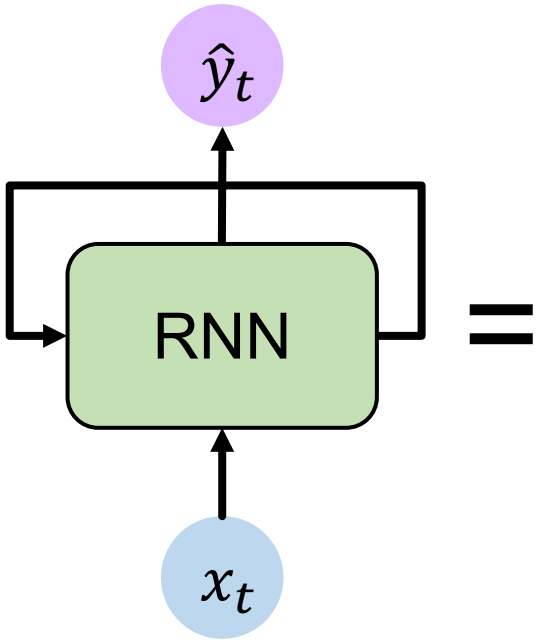
$$\hat{y}_t = \boldsymbol{W}_{\boldsymbol{hy}} h_t$$

Update Hidden State

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

Input Vector

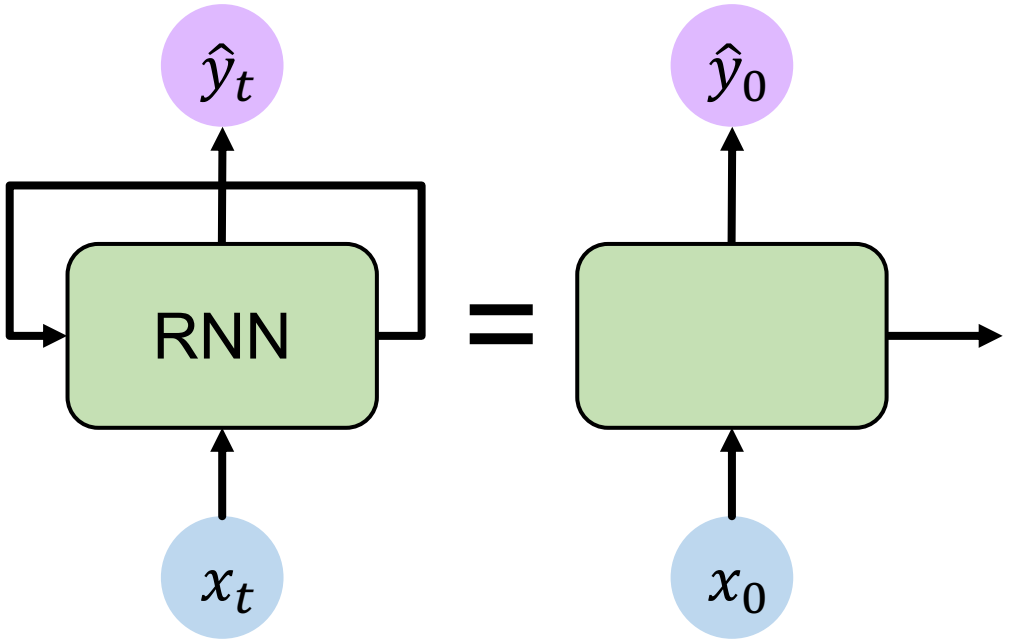
RNNs: computational graph across time



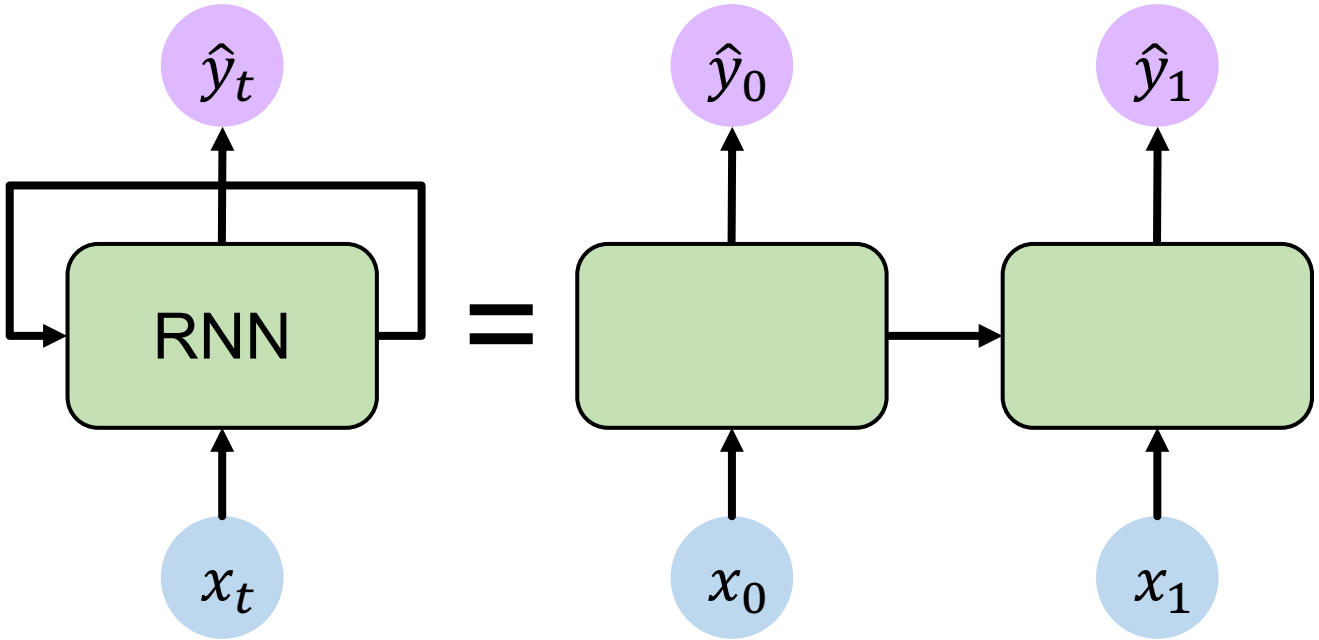
=

Represent as computational graph unrolled across time

RNNs: computational graph across time



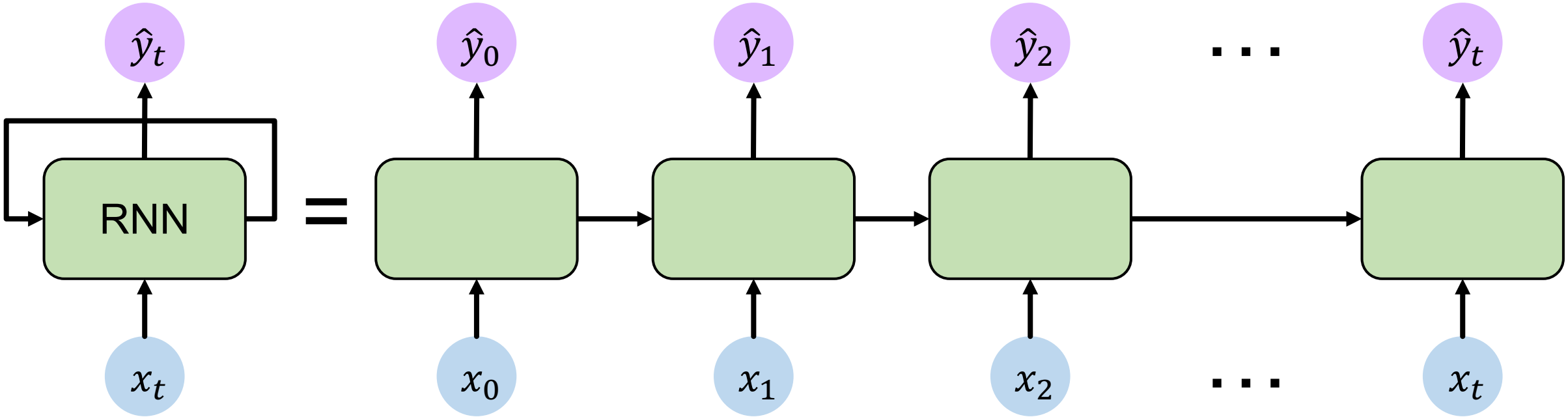
RNNs: computational graph across time



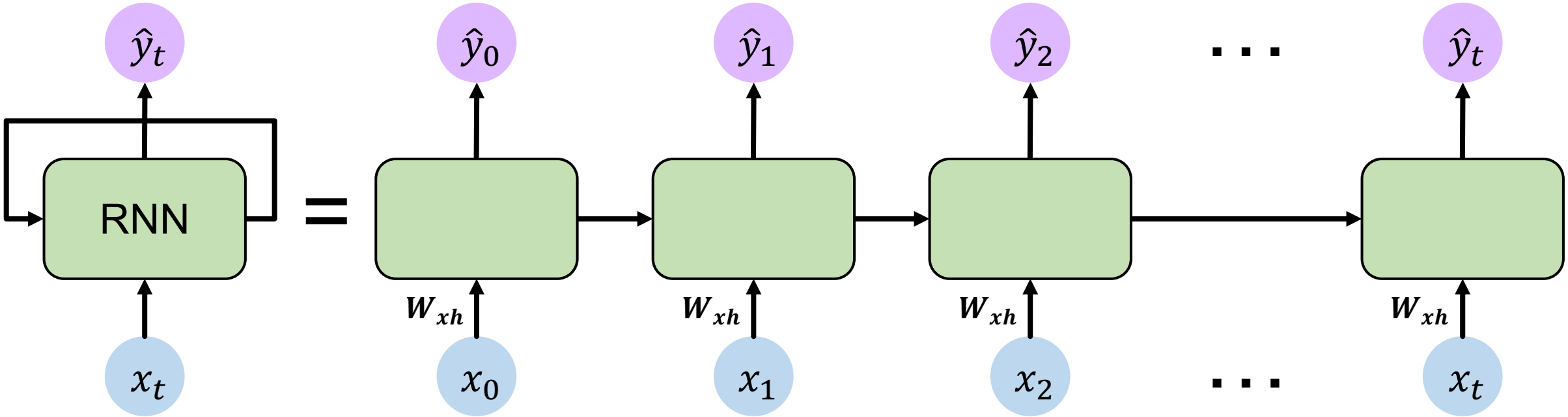
RNNs: computational graph across time

Eg.: France is where I grew up, but I now live in Boston. I speak fluent ___.

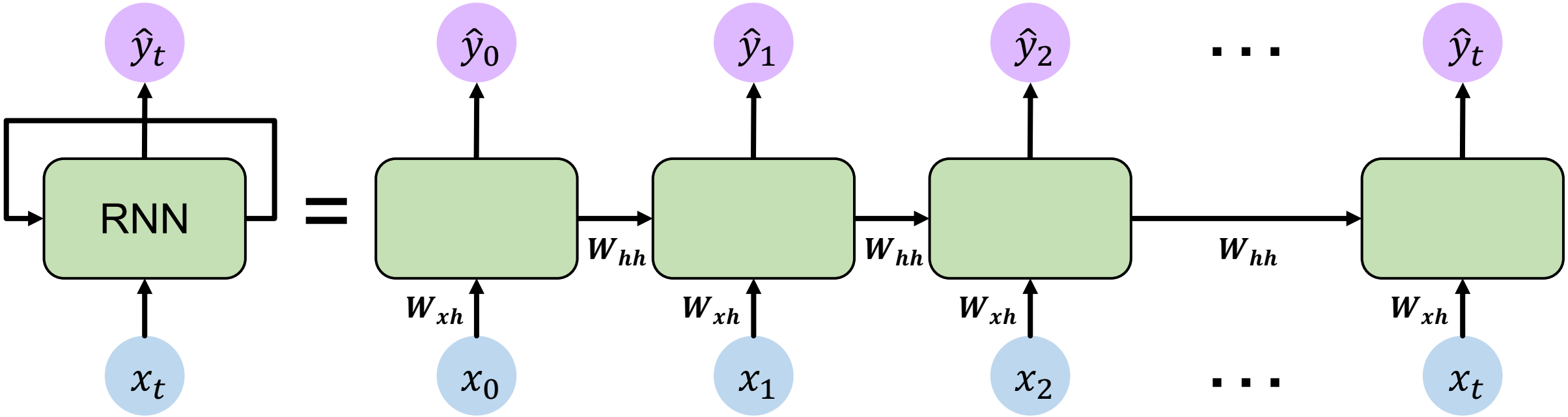
Discussion:
At time t, information from which words would be more likely stored in the memory?



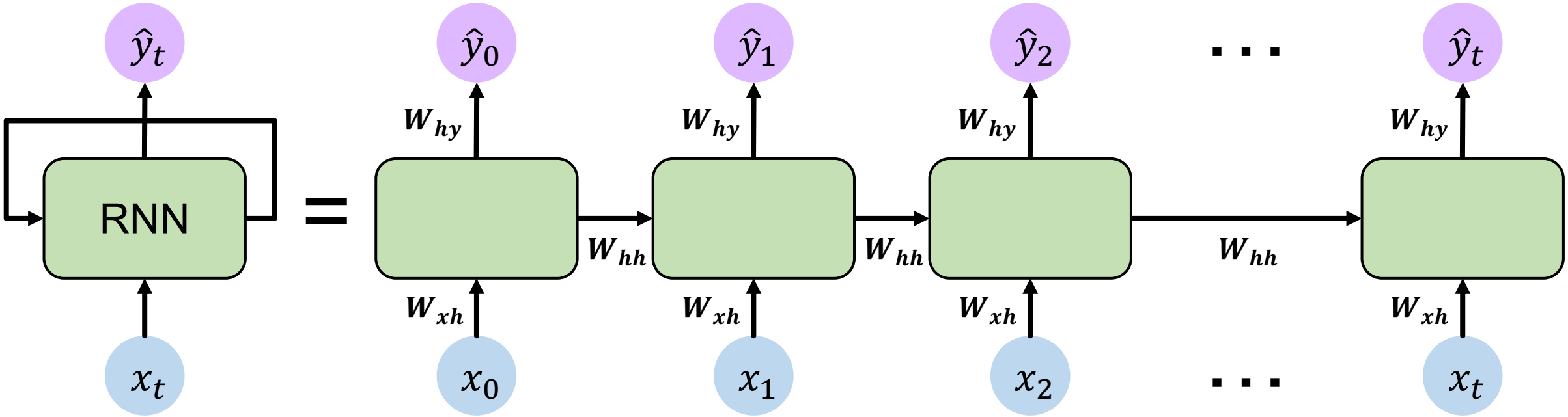
RNNs: computational graph across time



RNNs: computational graph across time

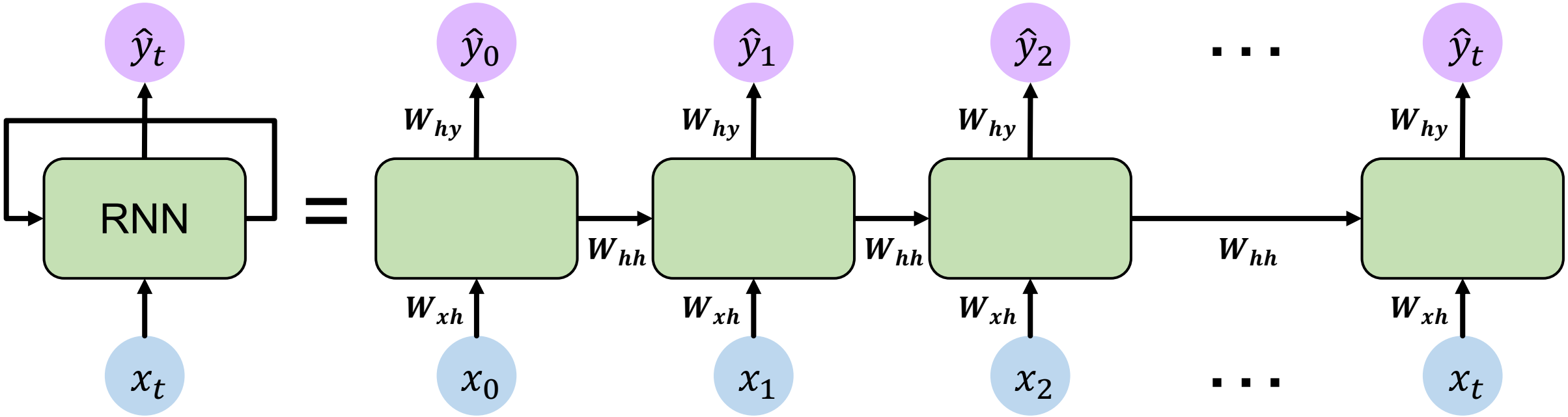


RNNs: computational graph across time

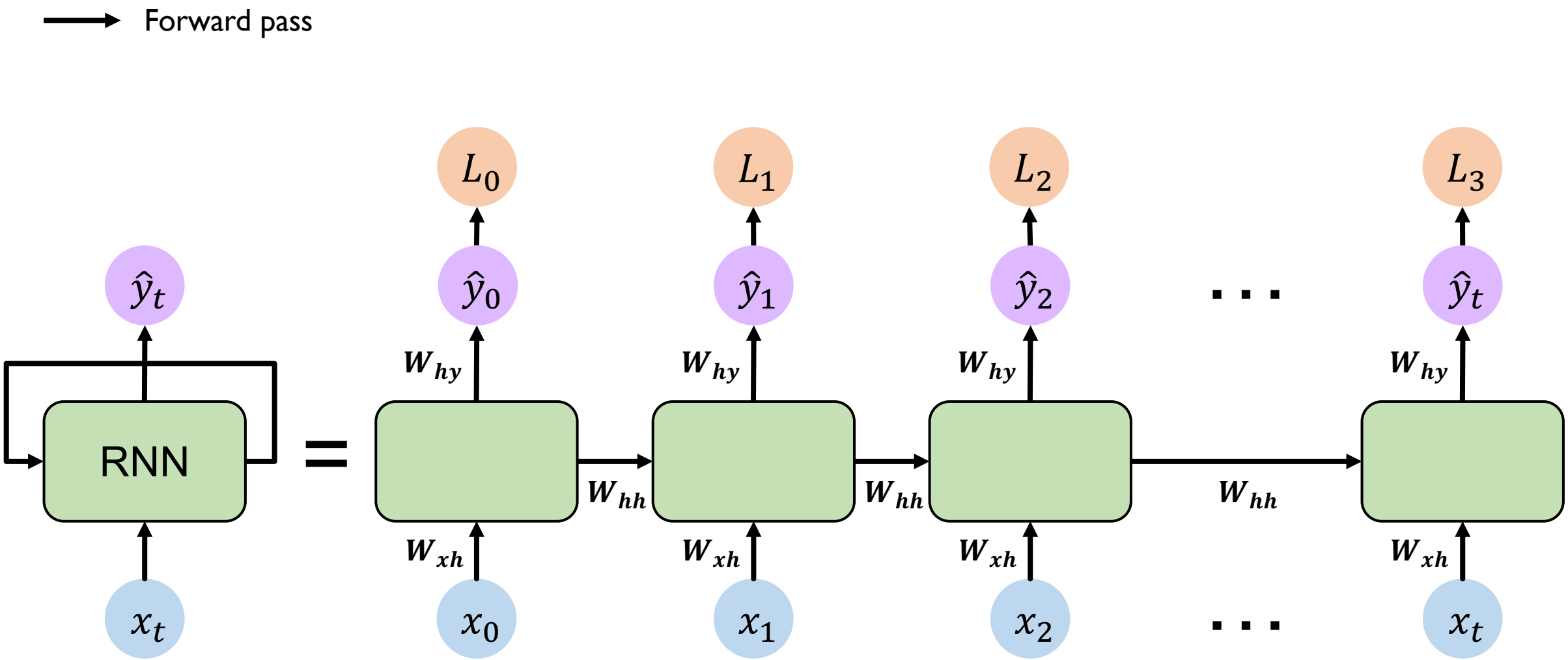


RNNs: computational graph across time

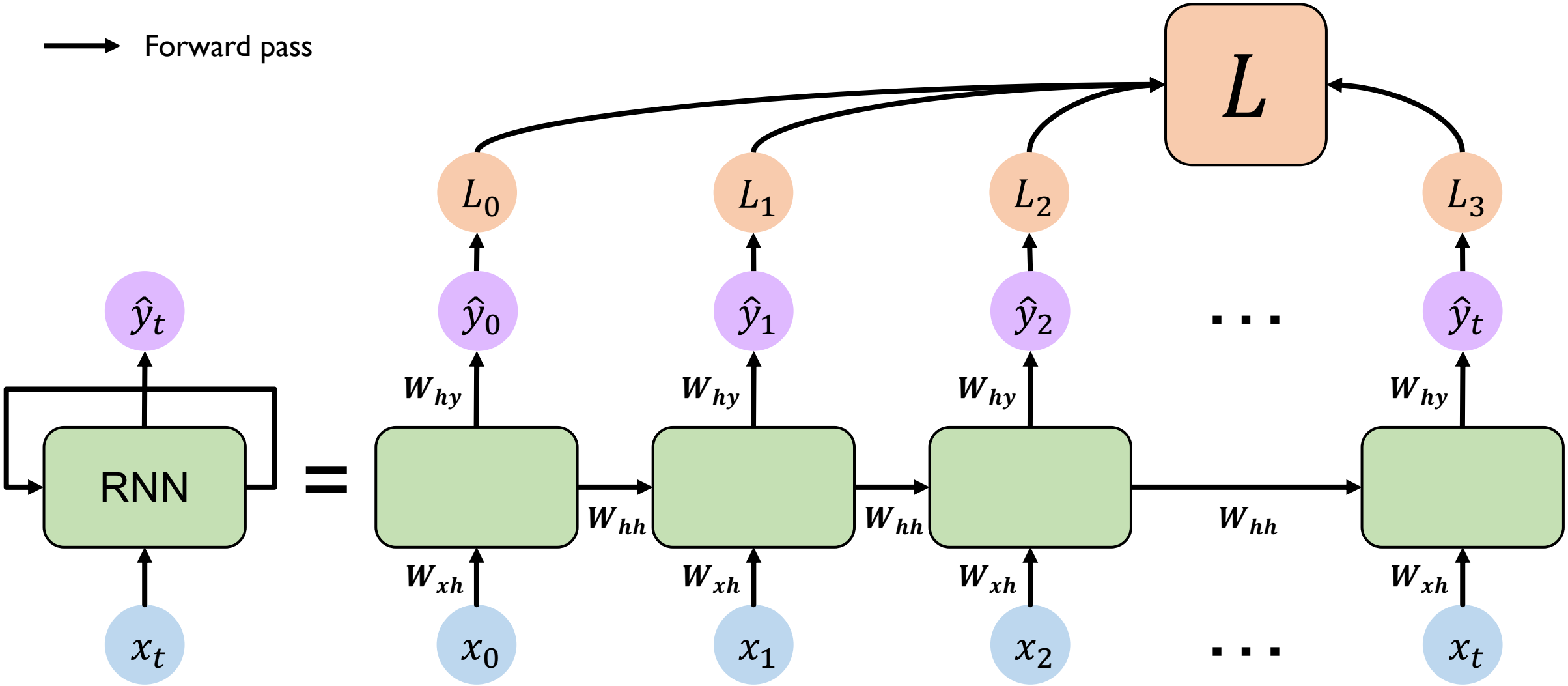
Re-use the **same weight matrices** at every time step



RNNs: computational graph across time

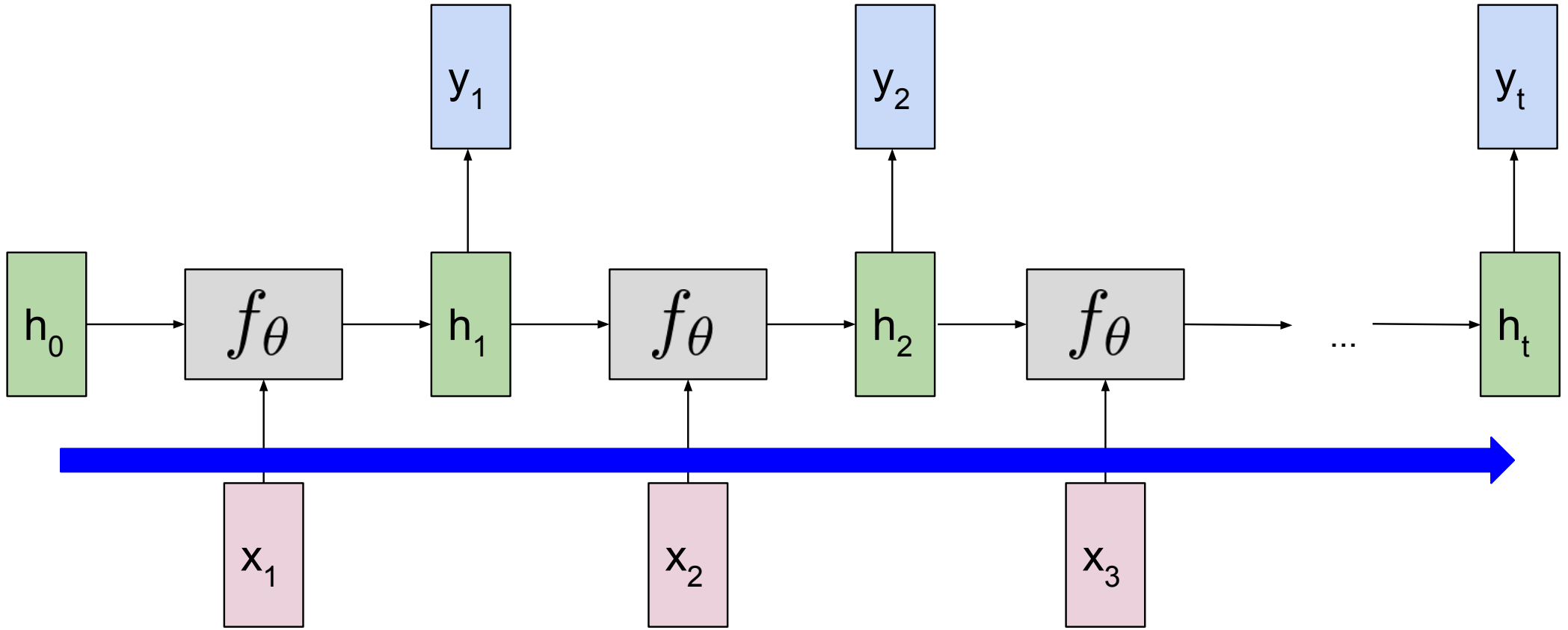


RNNs: computational graph across time



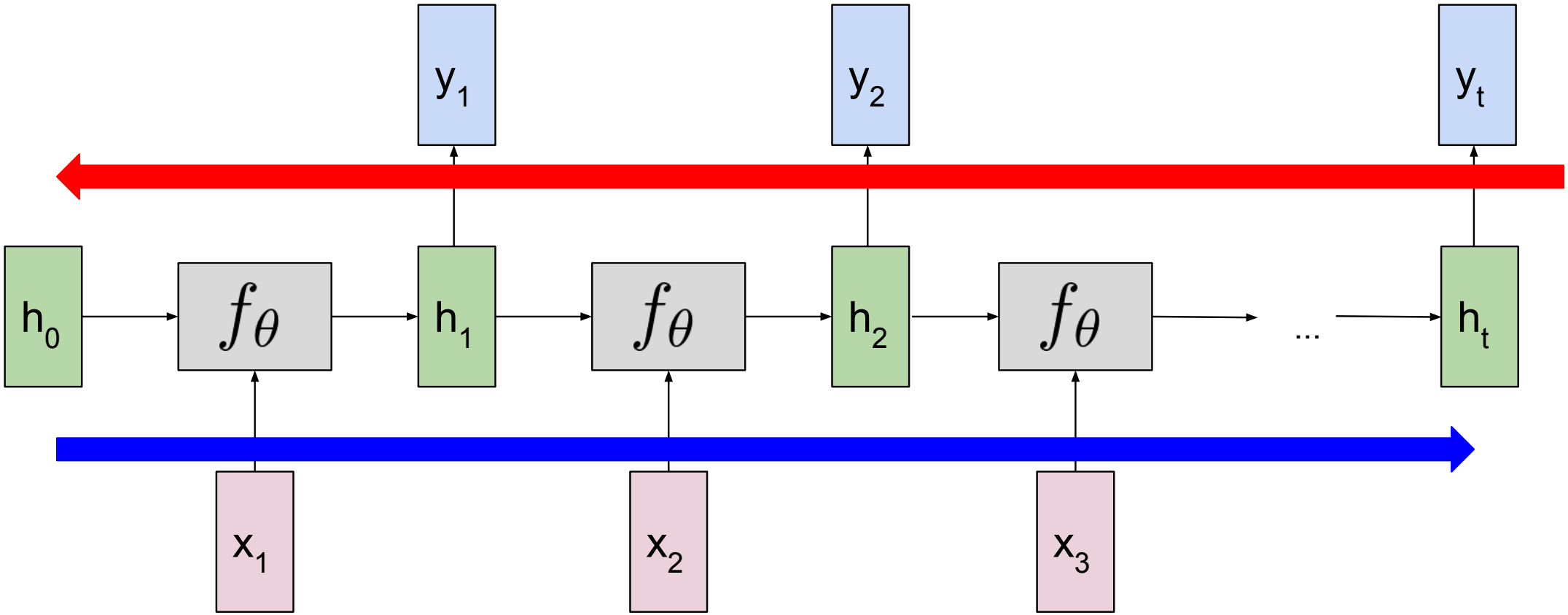
Backpropagation Through Time (BPTT)

Unrolling the RNN Computation Graph



“Unrolled” over *n* time-steps.

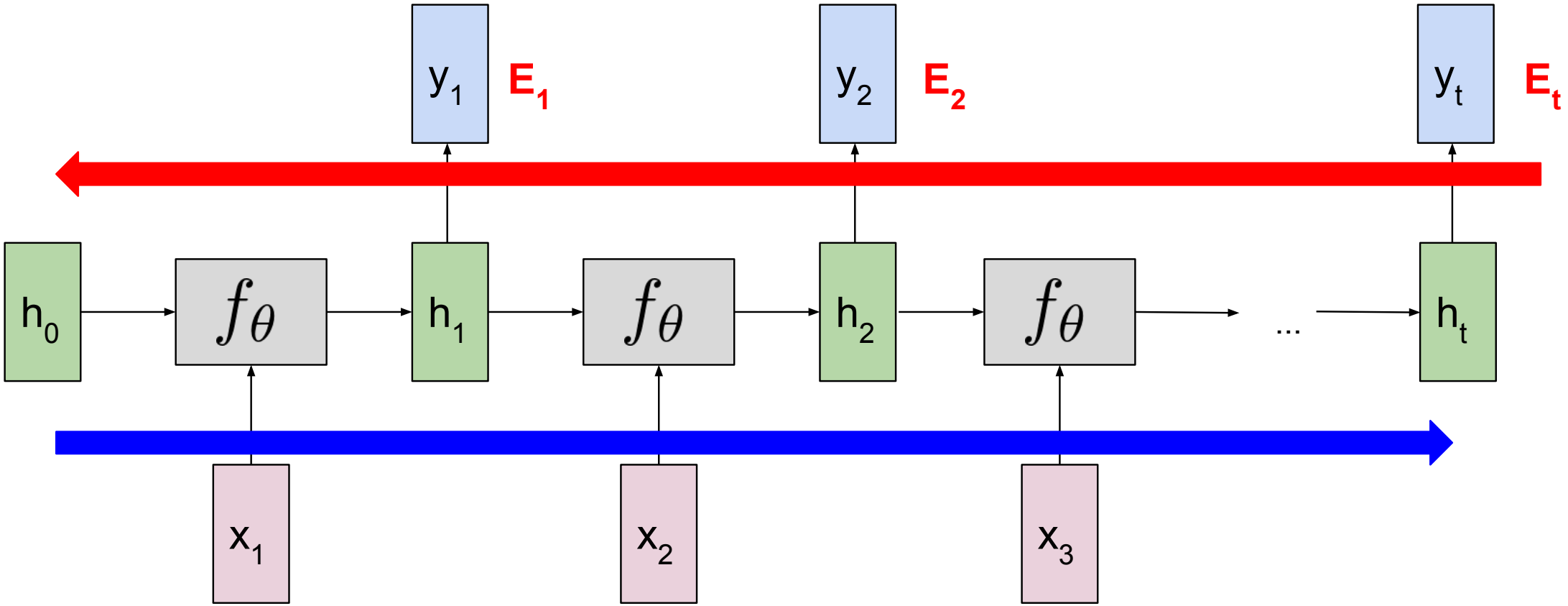
Unrolling the RNN Computation Graph



“Unrolled” over *n* time-steps.

Step 1: Compute all errors.

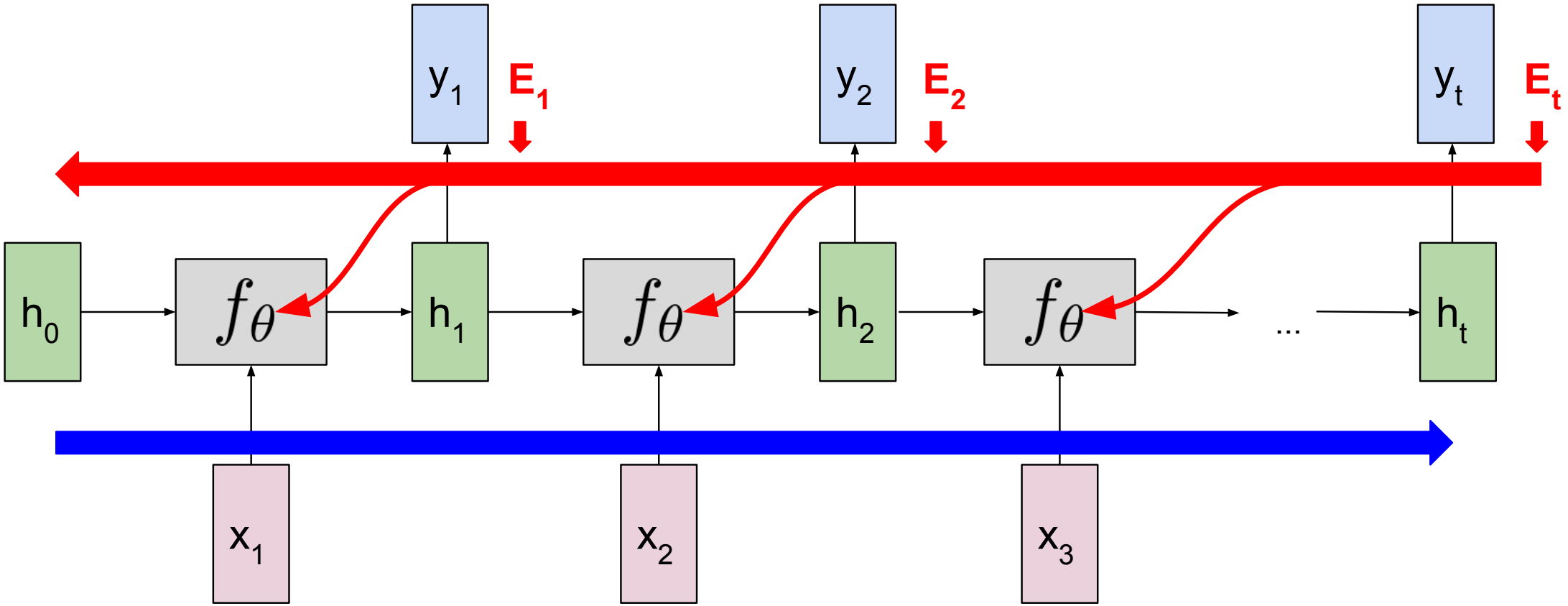
Unrolling the RNN Computation Graph



"Unrolled" over *n* time-steps.

Unrolling the RNN Computation Graph

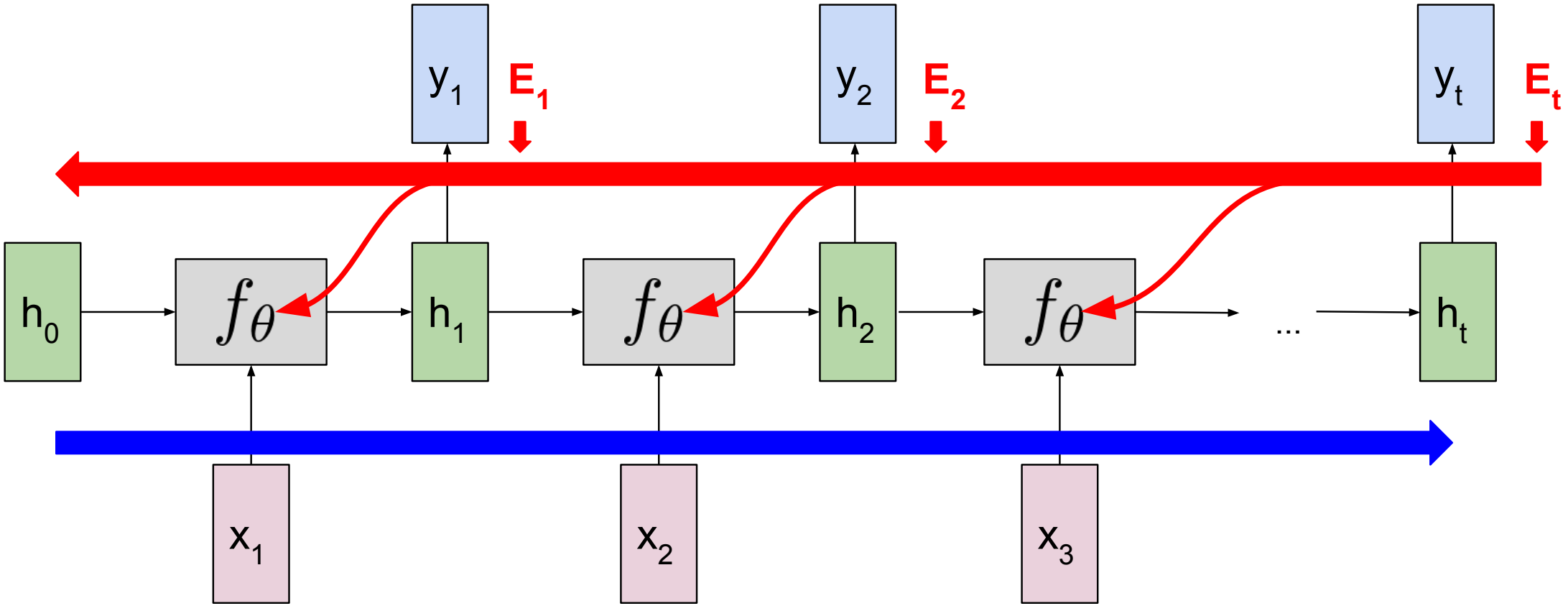
Step 1: Compute all errors.
Step 2: Pass error back for each time-step from n back to 1.



"Unrolled" over *n* time-steps.

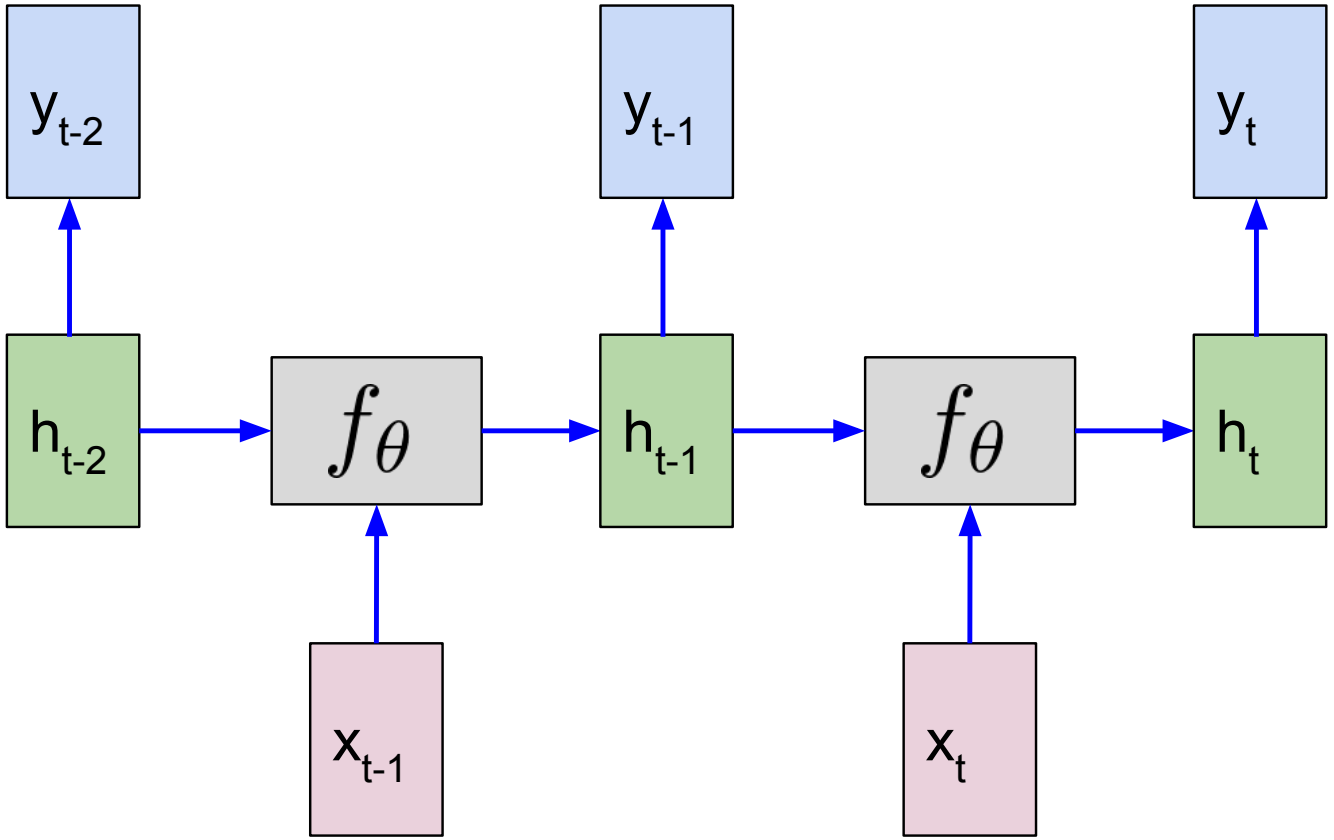
Unrolling the RNN Computation Graph

Step 1: Compute all errors.
Step 2: Pass error back for each time-step from n back to 1.
Step 3: Update weights.

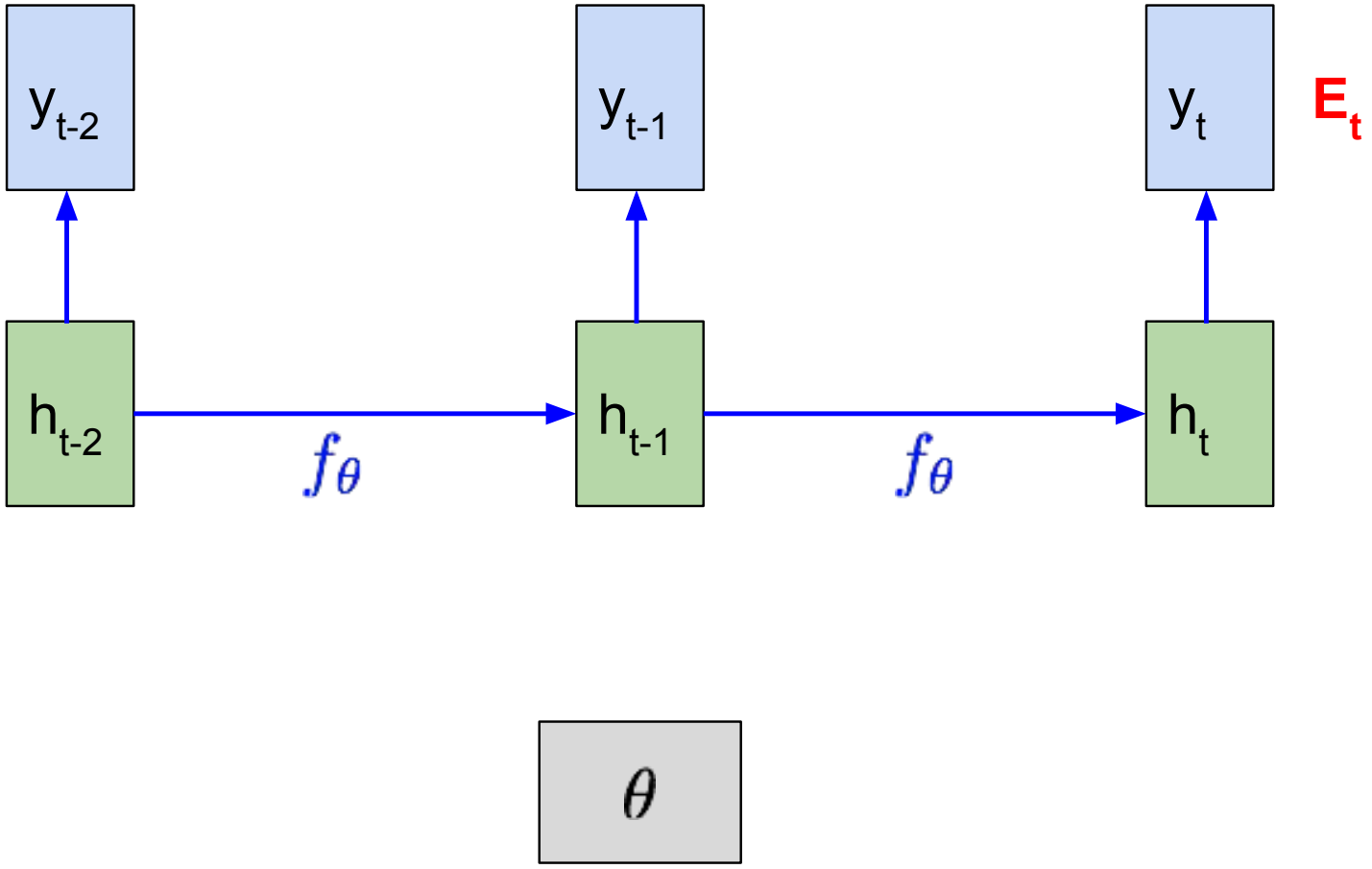


"Unrolled" over *n* time-steps.

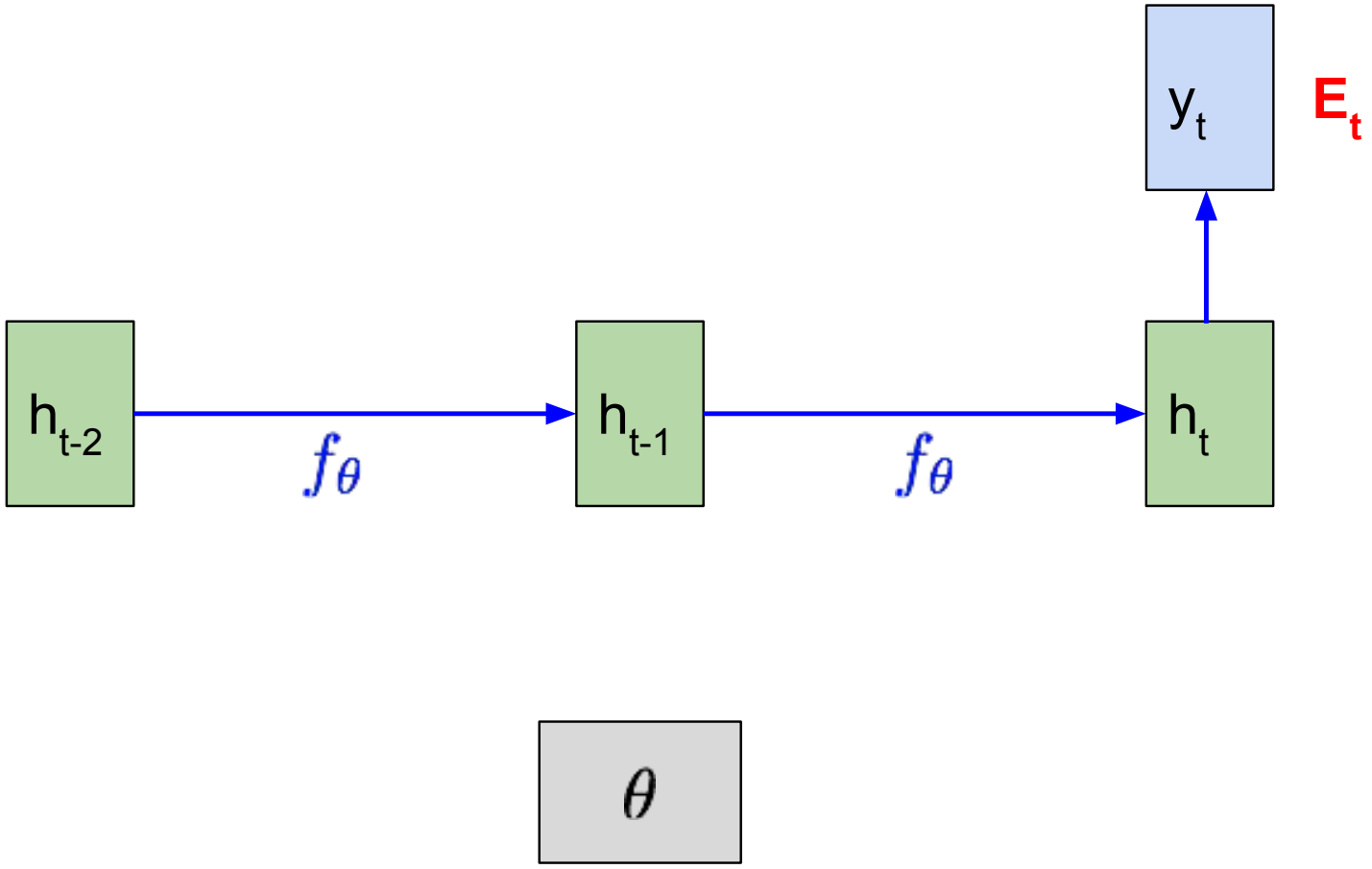
Unrolling the RNN Computation Graph



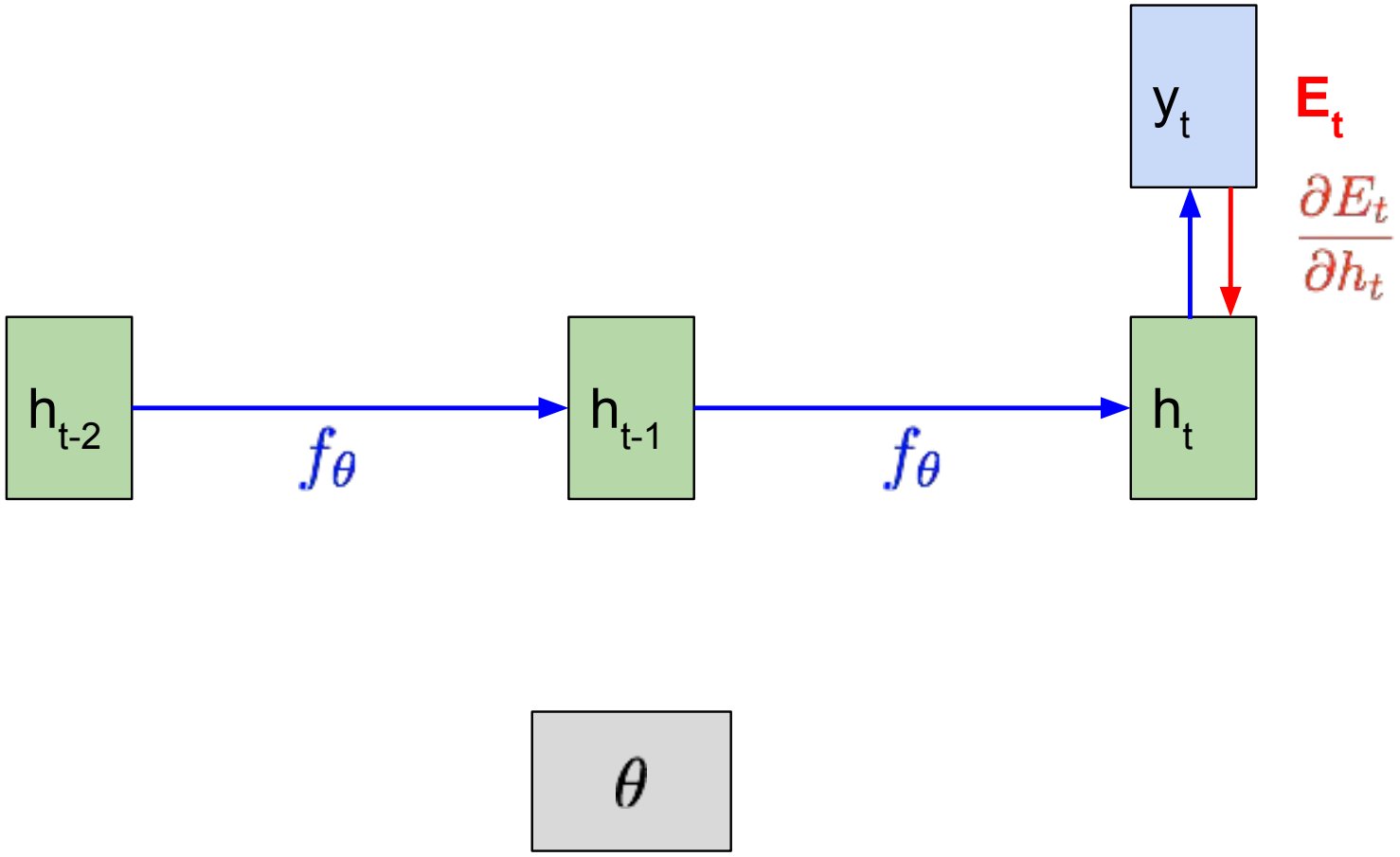
Unrolling the RNN Computation Graph



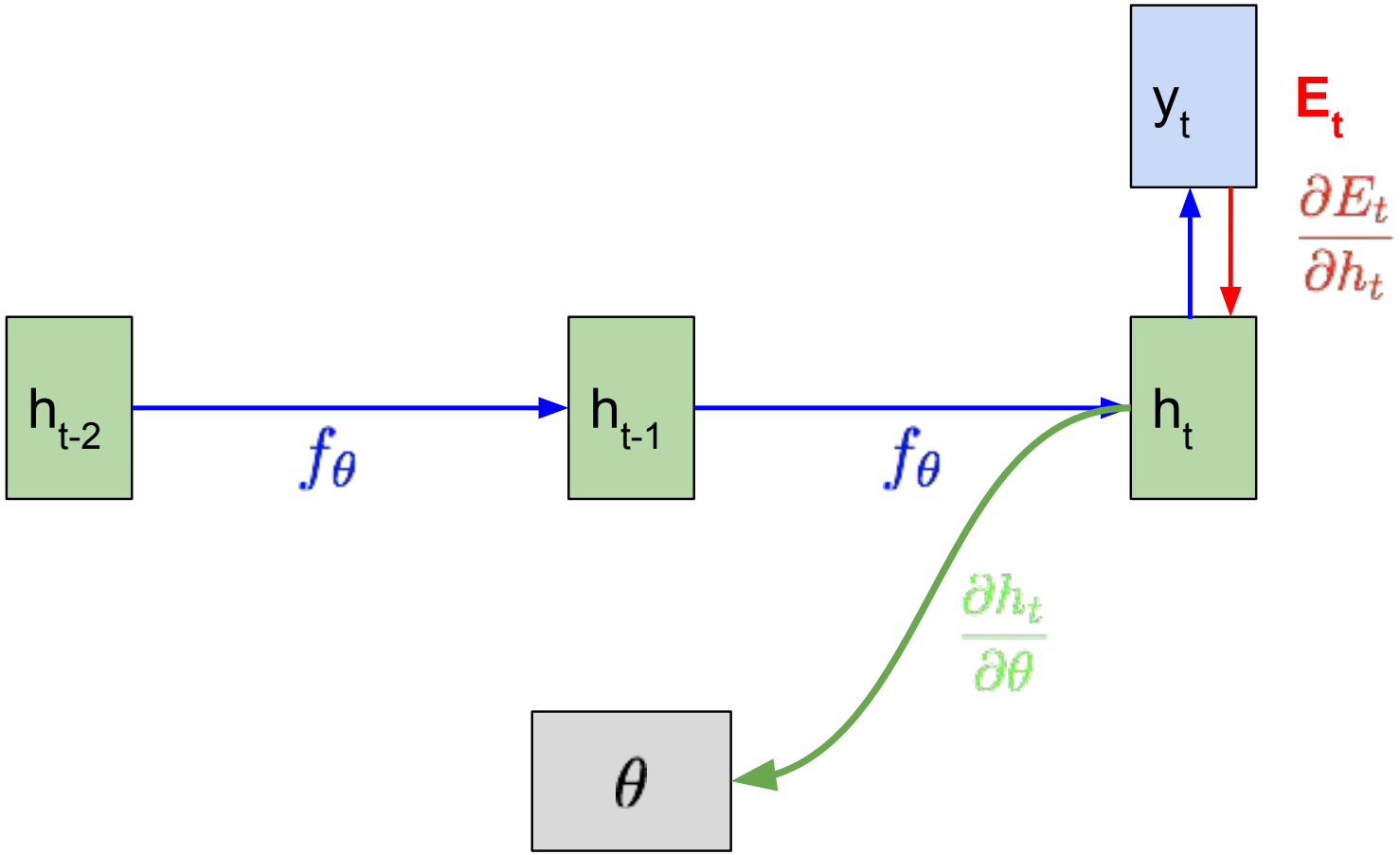
Unrolling the RNN Computation Graph



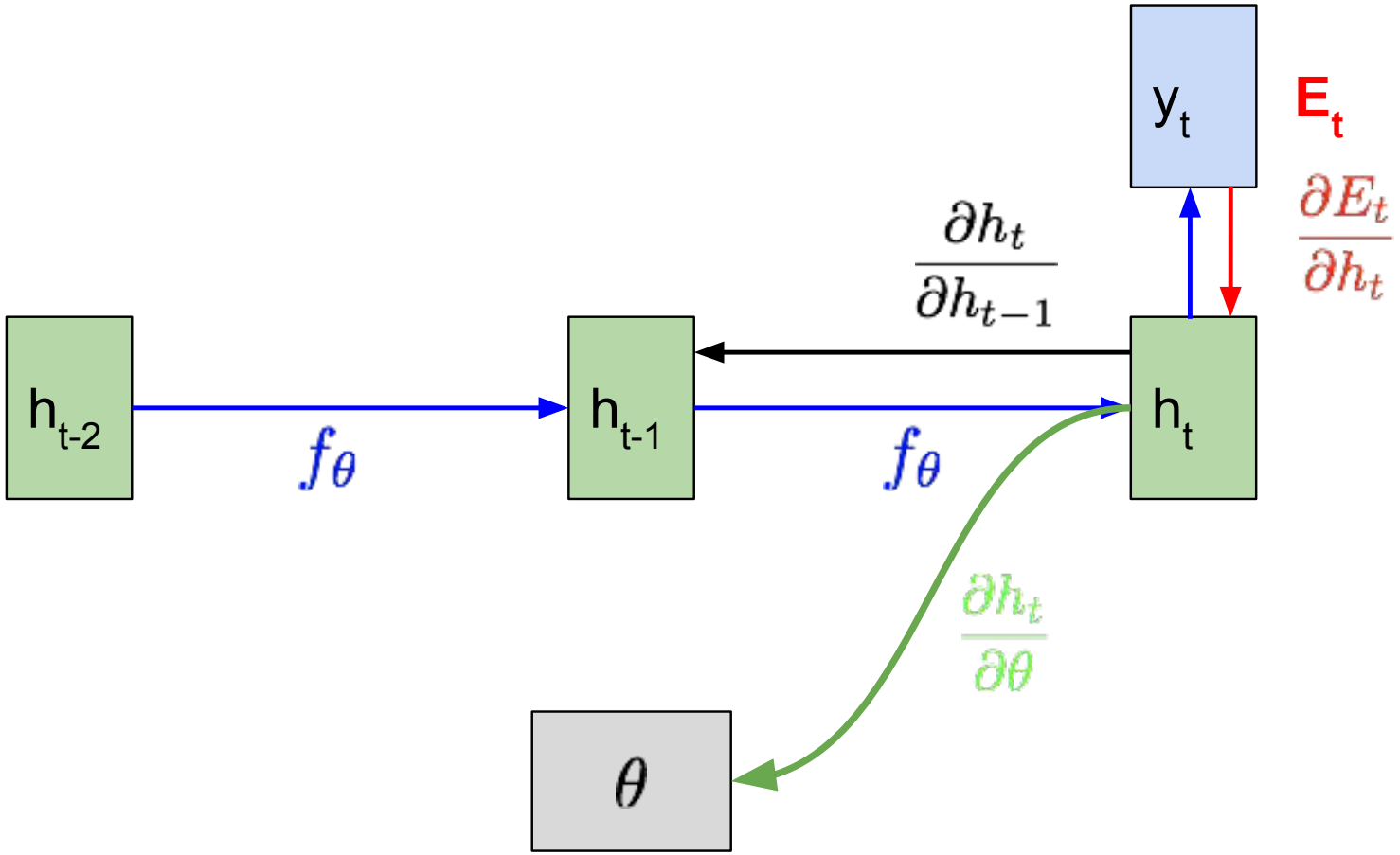
Unrolling the RNN Computation Graph



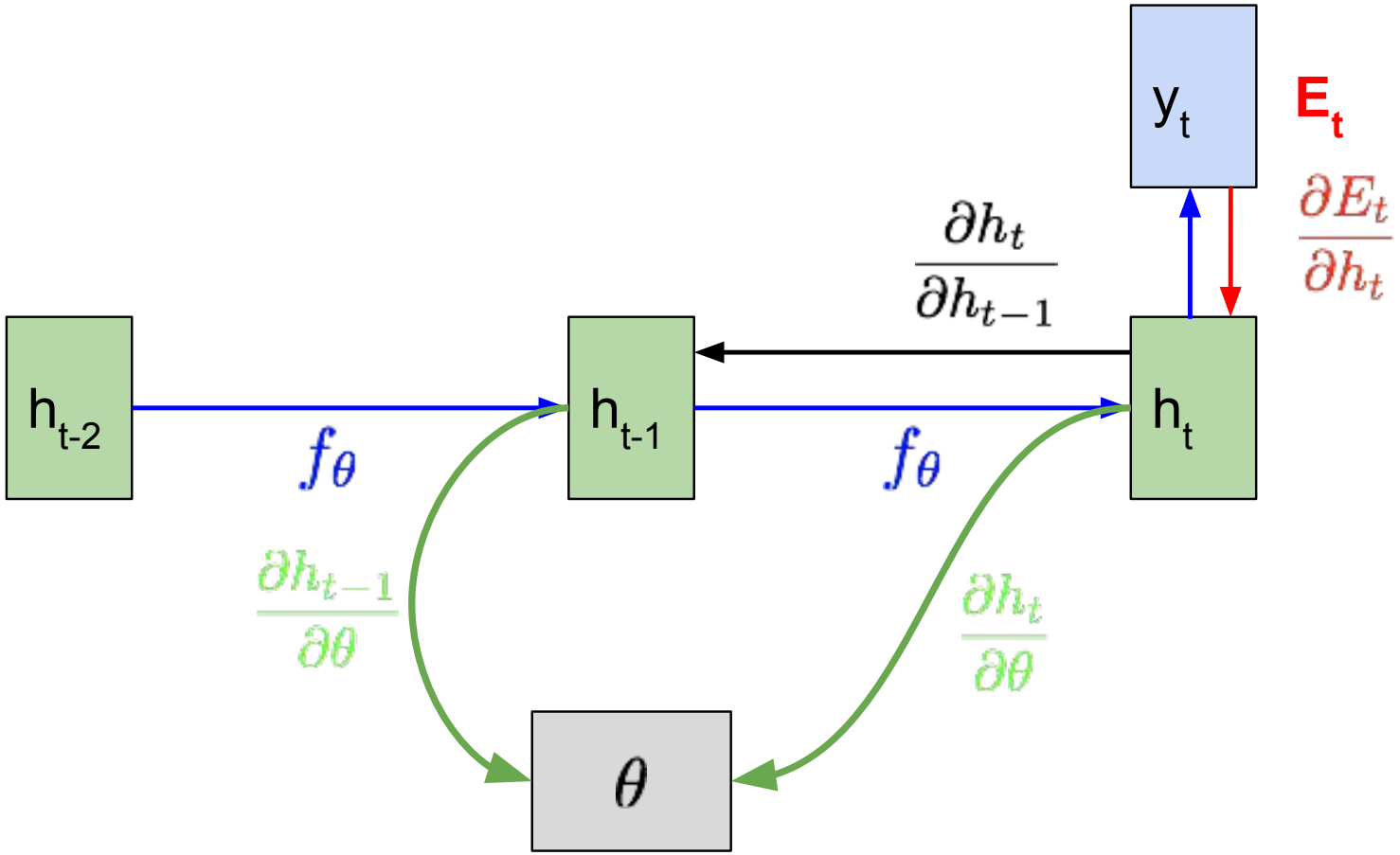
Unrolling the RNN Computation Graph



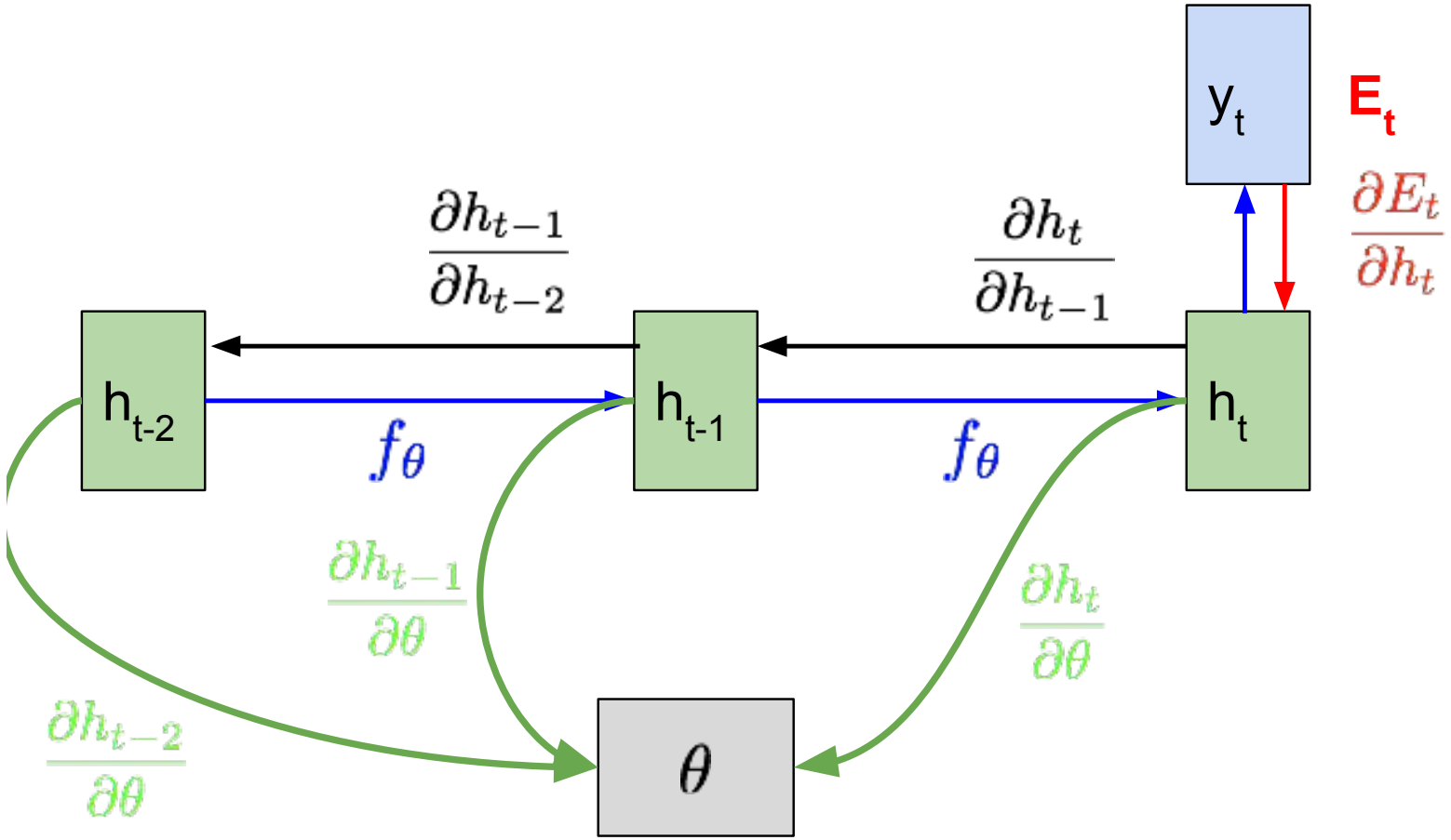
Unrolling the RNN Computation Graph



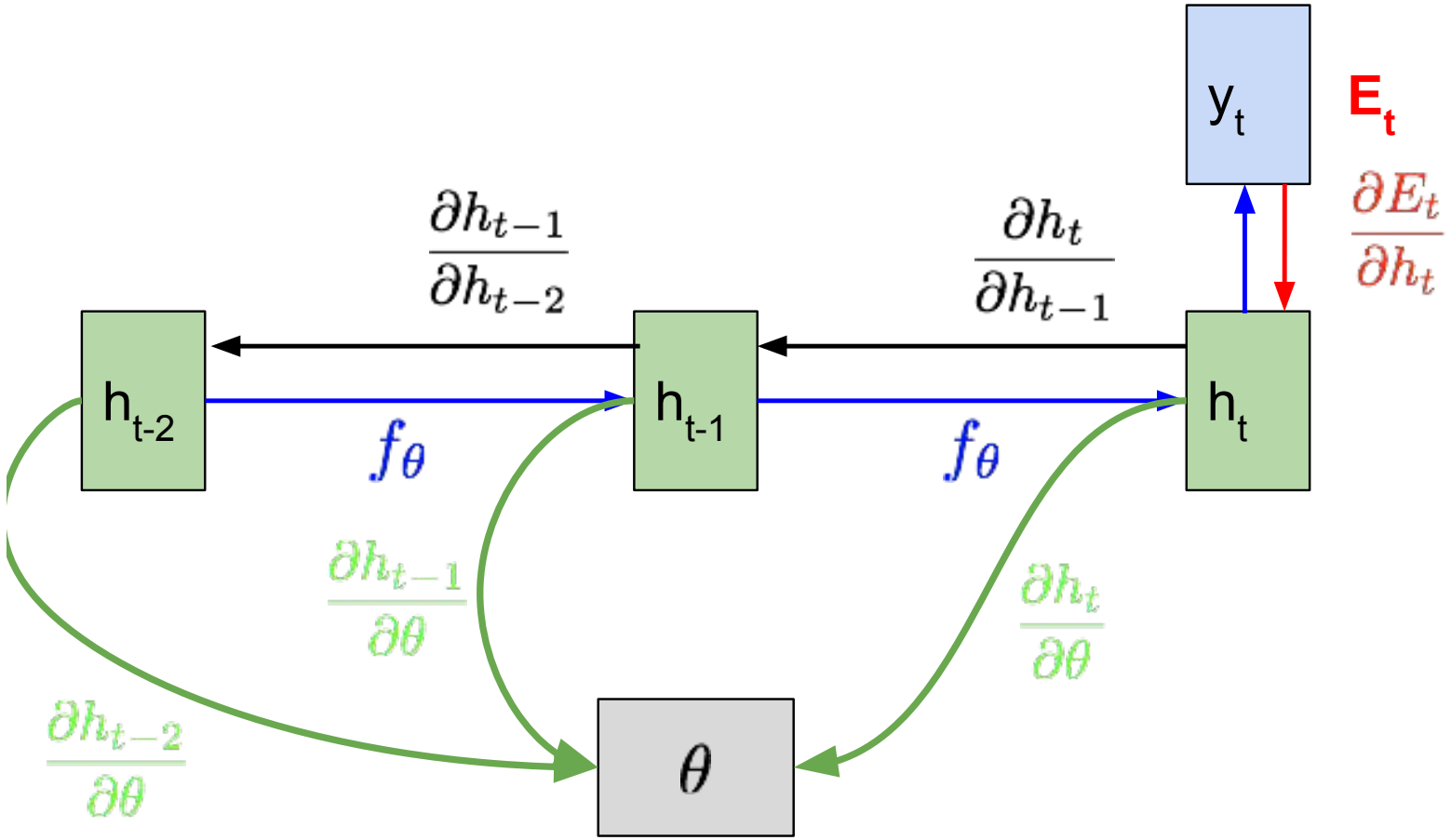
Unrolling the RNN Computation Graph



Unrolling the RNN Computation Graph



Unrolling the RNN Computation Graph

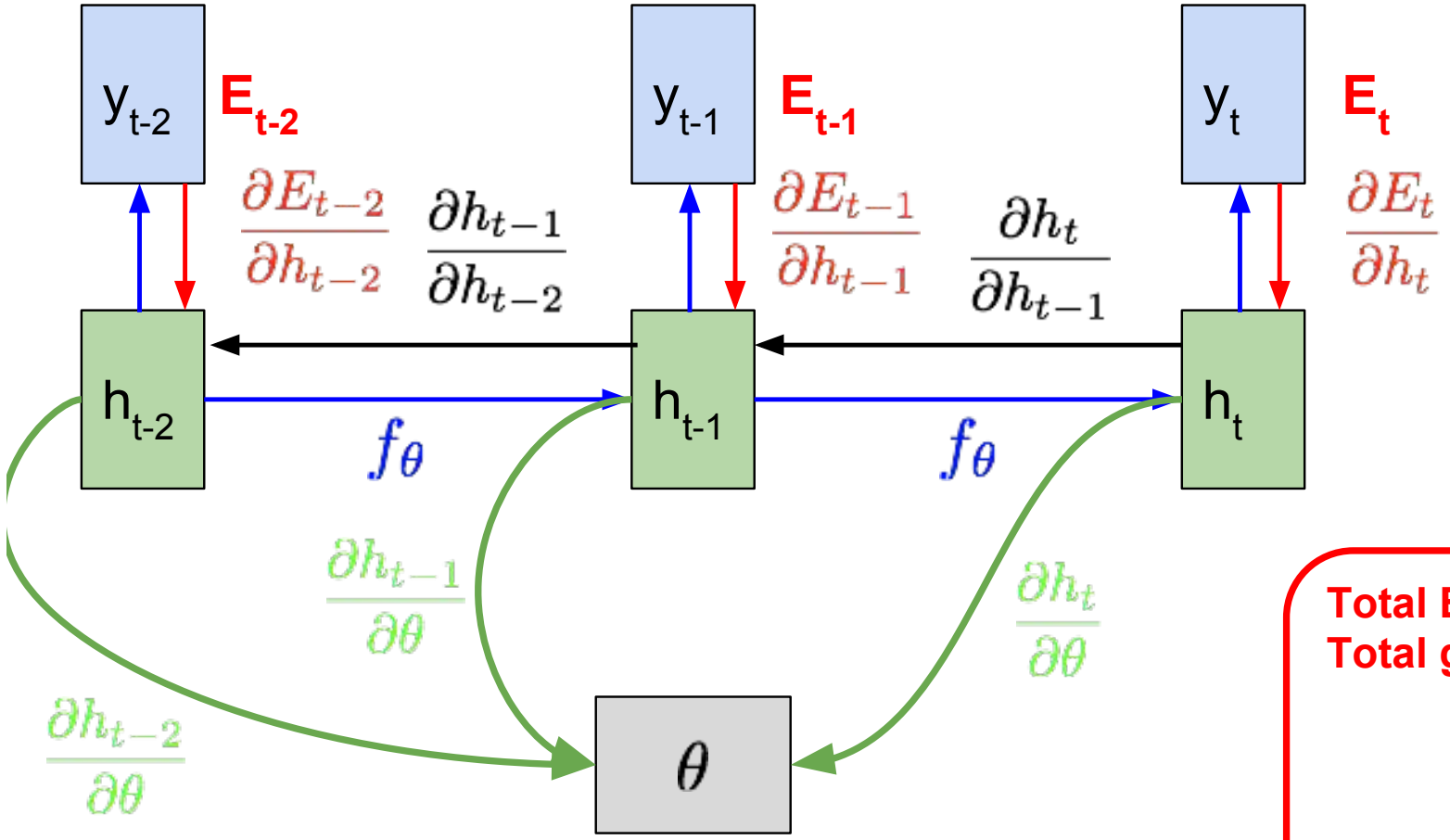


$$\frac{\partial E_t}{\partial \theta} = \sum_{t'=1}^{t} \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t'}} \frac{\partial h_{t'}}{\partial \theta}$$

where

$$\frac{\partial h_t}{\partial h_{t'}} = \Pi_{k=t'+1}^{t} \frac{\partial h_k}{\partial h_{k-1}}$$

Unrolling the RNN Computation Graph



$$\frac{\partial E_t}{\partial \theta} = \sum_{t'=1}^{t} \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t'}} \frac{\partial h_{t'}}{\partial \theta}$$

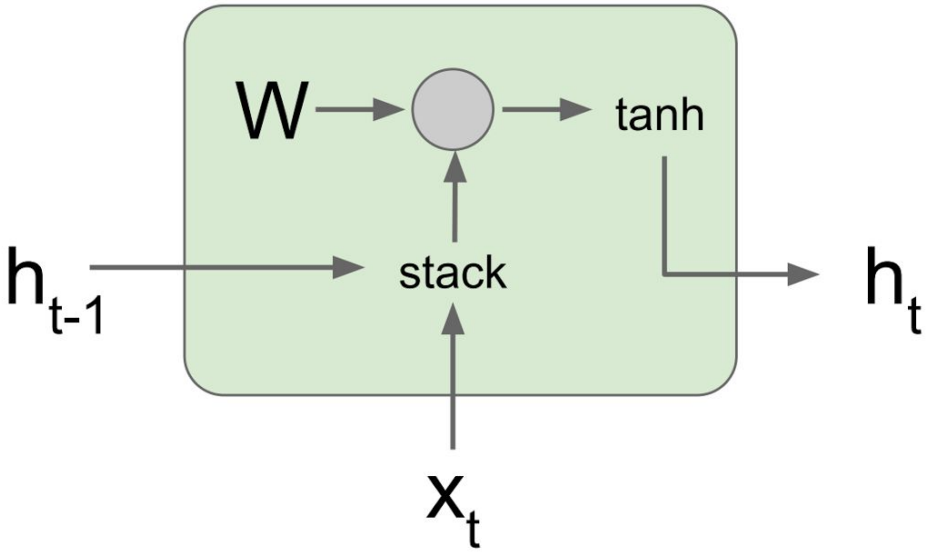
where

$$\frac{\partial h_t}{\partial h_{t'}} = \Pi_{k=t'+1}^{t} \frac{\partial h_k}{\partial h_{k-1}}$$

Total Error = $E_1 + E_2 + \ldots + E_t$
Total gradient = sum of all $dE_t/d\theta$'s

$$\frac{\partial E_{TOTAL}}{\partial \theta} = \frac{\partial \sum_t E_t}{\partial \theta} = \sum_t \frac{\partial E_t}{\partial \theta}$$

Vanilla RNN Gradient Flow

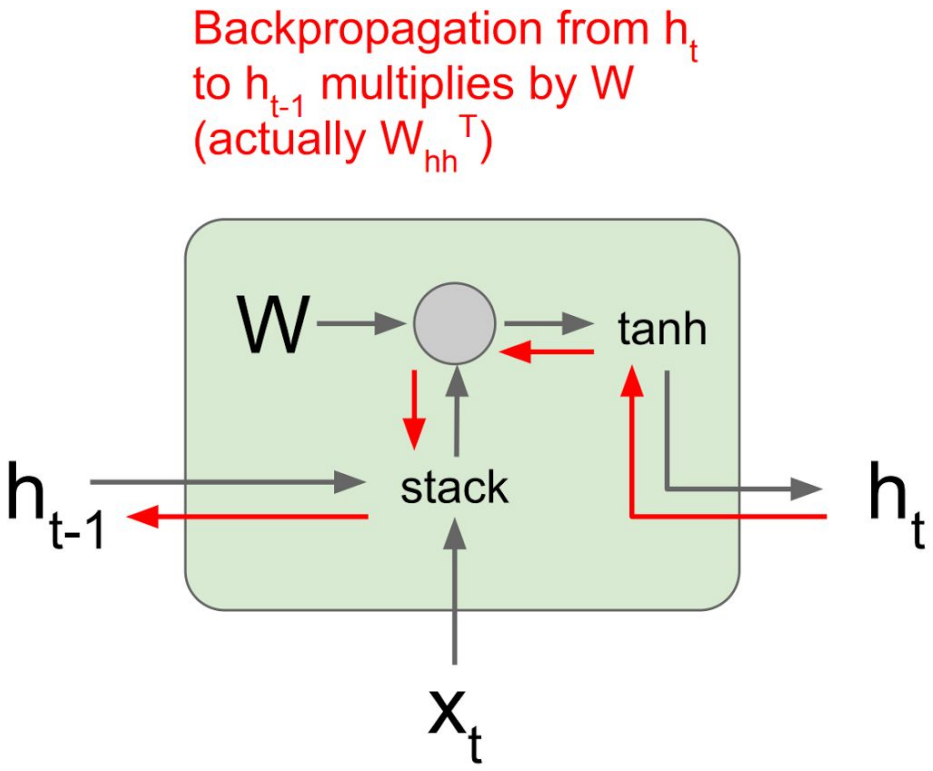


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

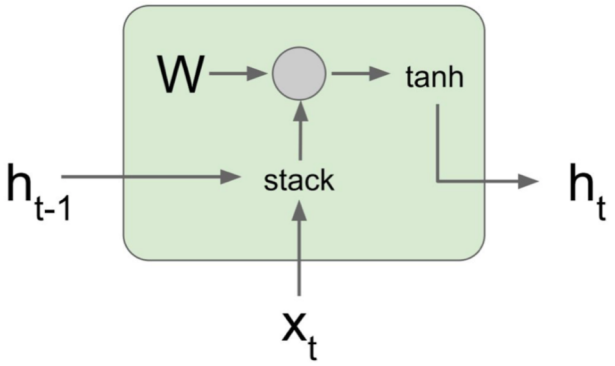
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow



$$
\begin{aligned}
h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{aligned}
$$

Vanilla RNN Gradient Flow



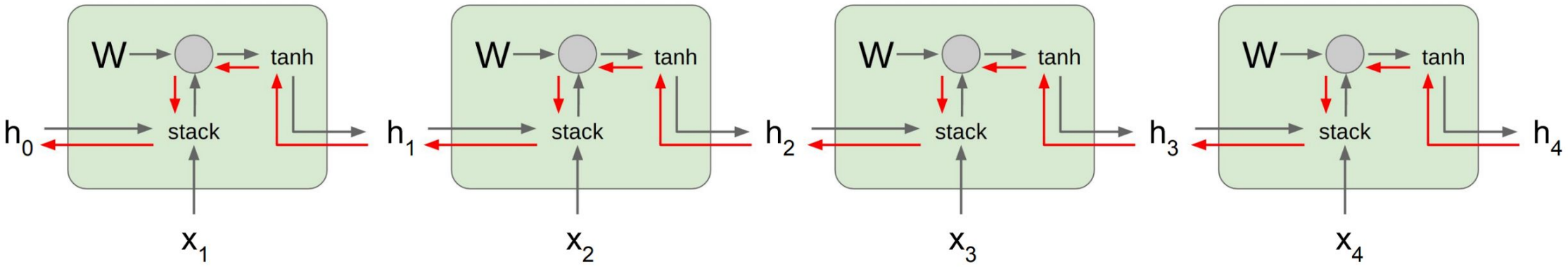
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'_{t-1} * W_{hh}$$

$$\frac{\partial h_{t+1}}{\partial h_1} = (\prod_{k=t}^{1} tanh'_k) * (W_{hh})^t$$

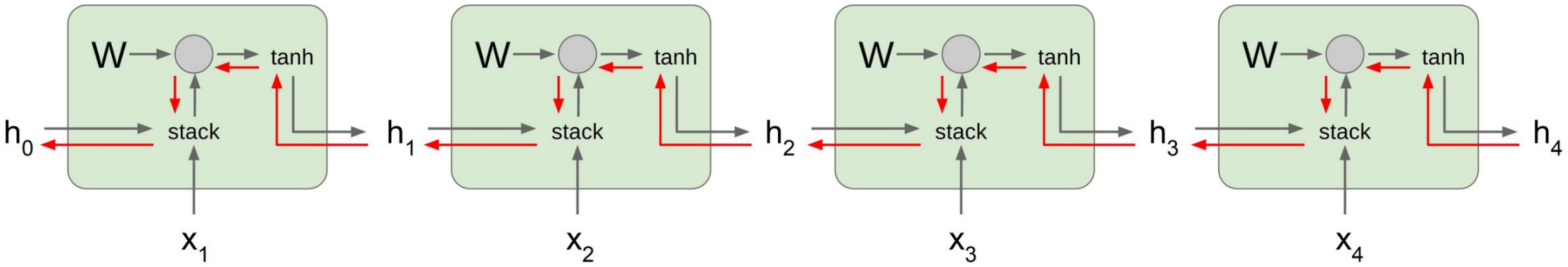
- Suppose that W_{hh} has an eigendecomposition, $W_{hh} = V\ diag(\lambda)\ V^{-1}$
- $(W_{hh})^t = V\ diag(\lambda)^t\ V^{-1}$, then gradients are scaled according to $diag(\lambda)^t$.
- For eigenvalue λ_i,
  - If λ_i >1, may lead to **gradient explosion**
  - If λ_i <1, gradient may **vanish**
- **Vanishing gradients** make it difficult to know which direction the parameters should move to improve the cost function, while **exploding gradients** can make learning unstable.

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Vanilla RNN Gradient Flow

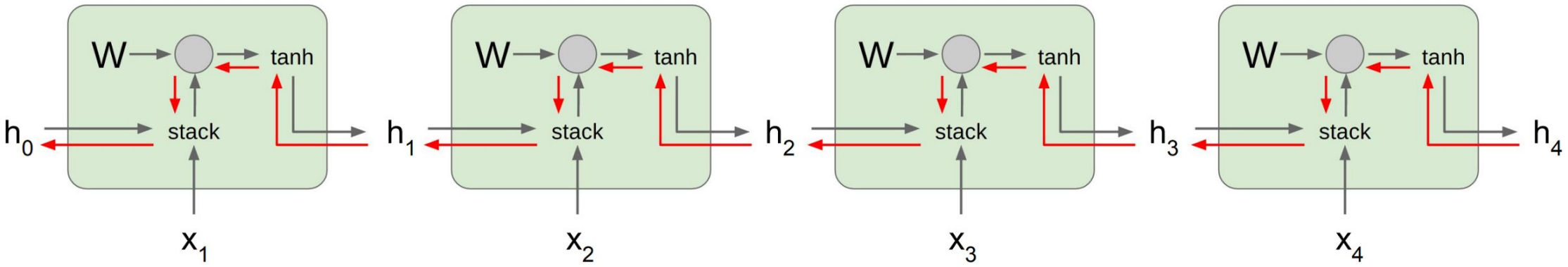


Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

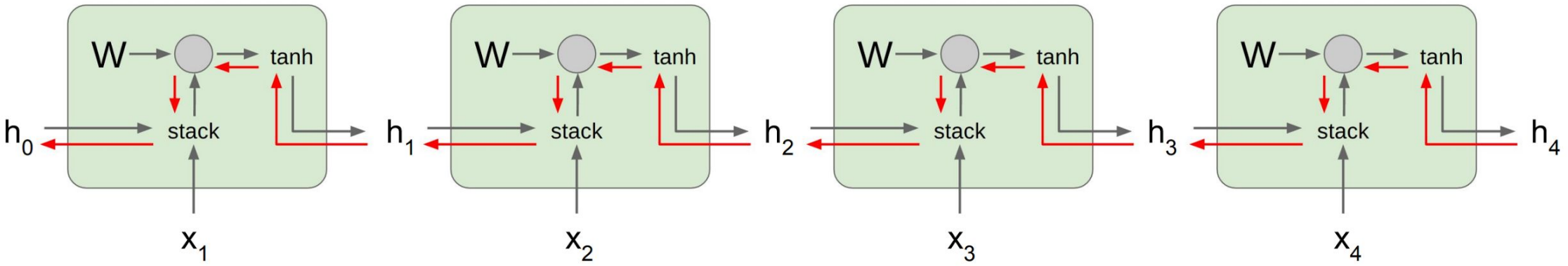
Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

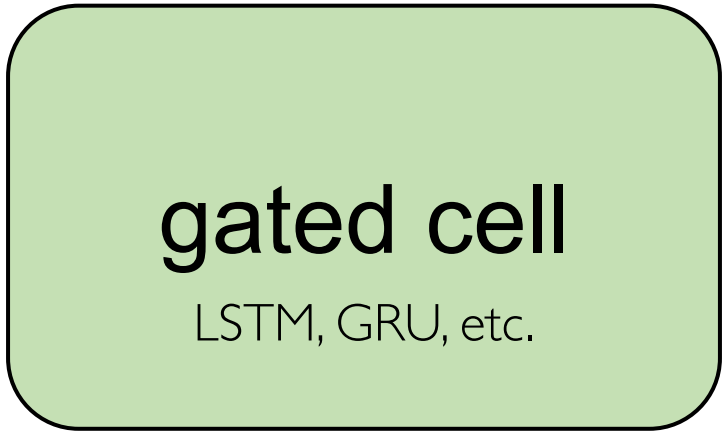
Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients → Change RNN architecture

Part 2!

Solution: Gated Cell

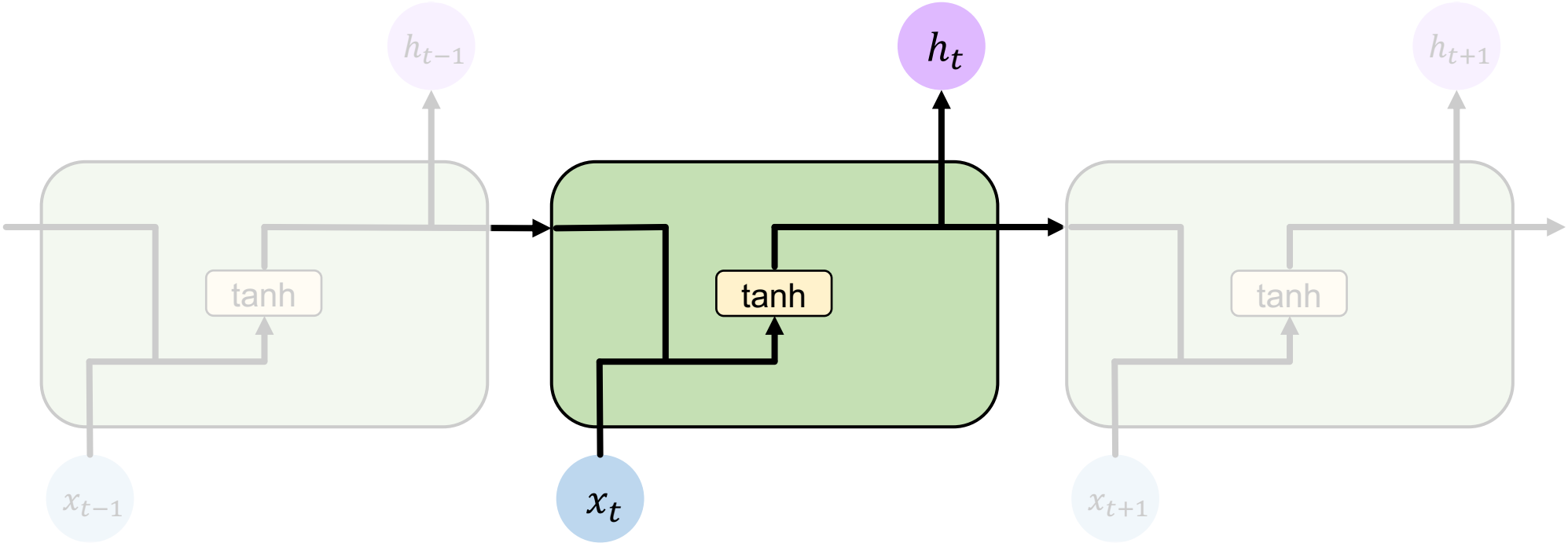
Idea: use a more **complex recurrent unit with gates** to control what information is passed through



Long Short Term Memory (LSTM) Networks

Standard RNN

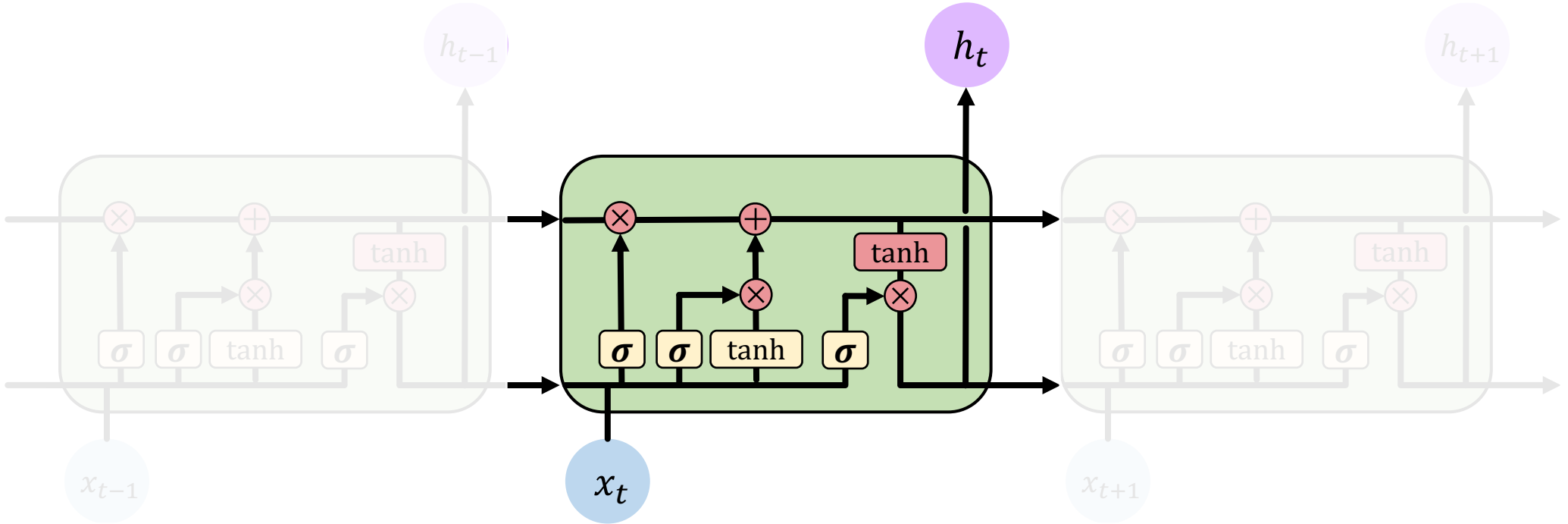
In a standard RNN, repeating modules contain a **simple computation node**



[2]

Long Short Term Memory (LSTMs)

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

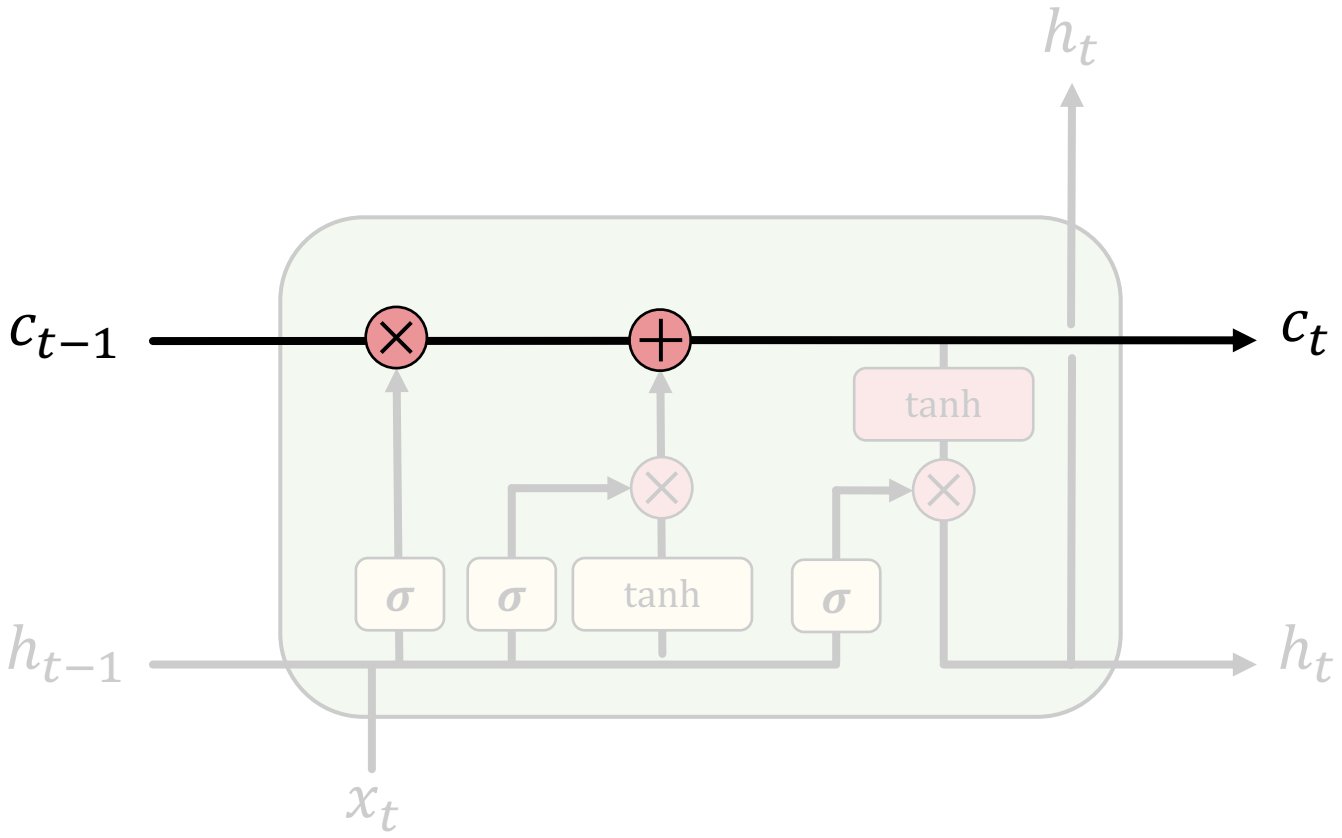


LSTM cells are able to track information throughout many timesteps

Hochreiter & Schmidhuber, 1997 [2, 5]

Long Short Term Memory (LSTMs)

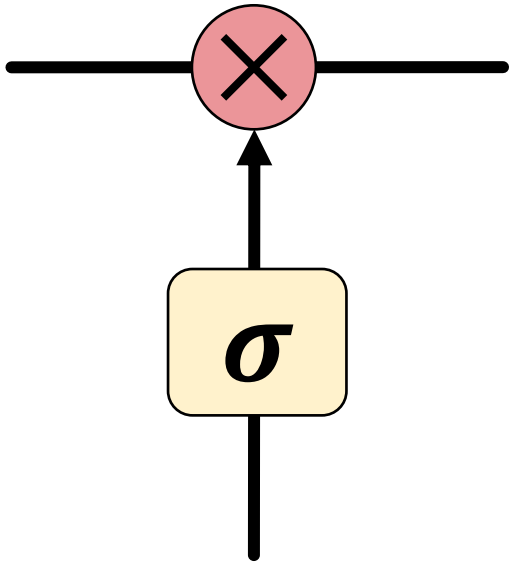
LSTMs maintain a **cell state** c_t where it's easy for information to flow



[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

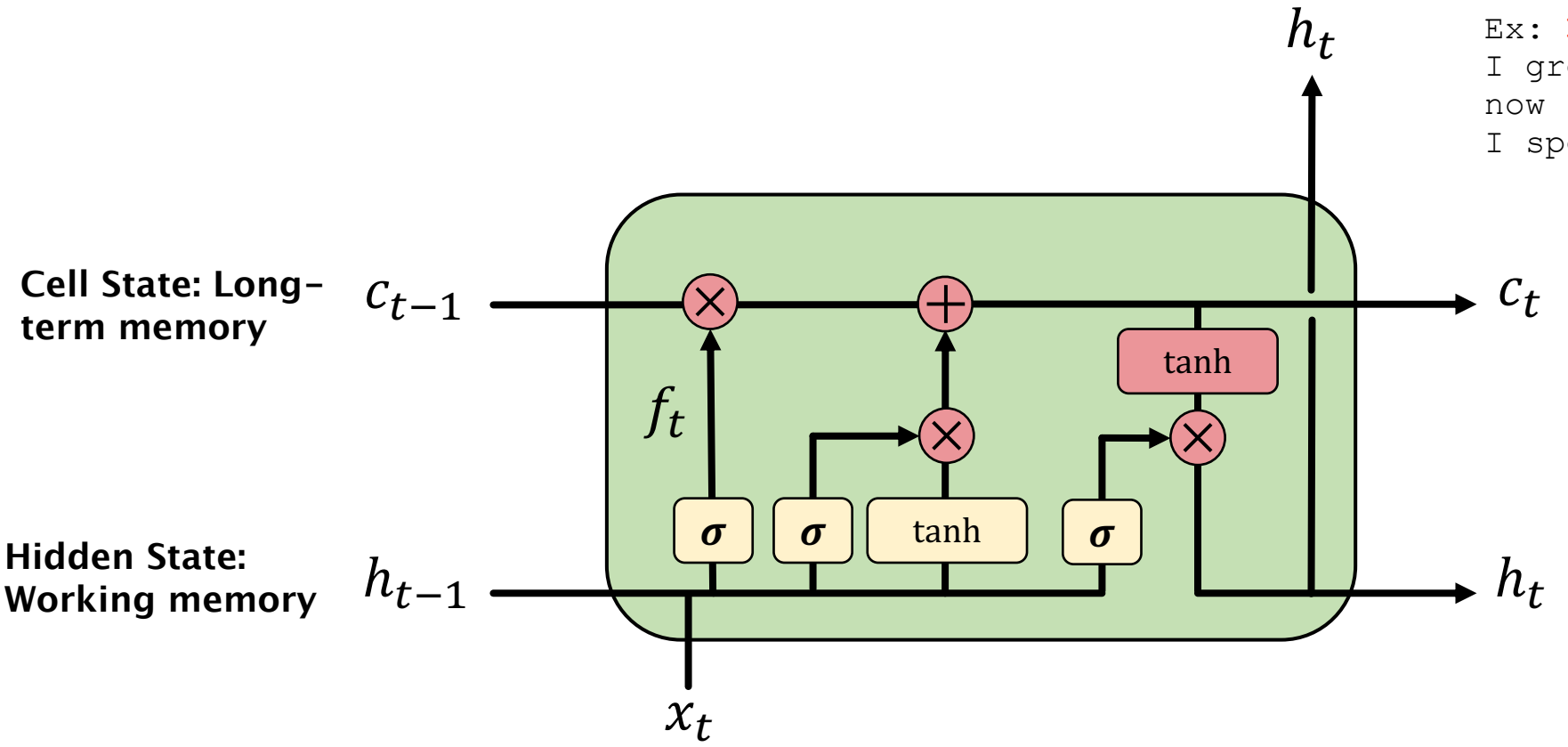
[2, 5]

Long Short Term Memory (LSTMs)

How do LSTMs work?

Why cell memory is needed?

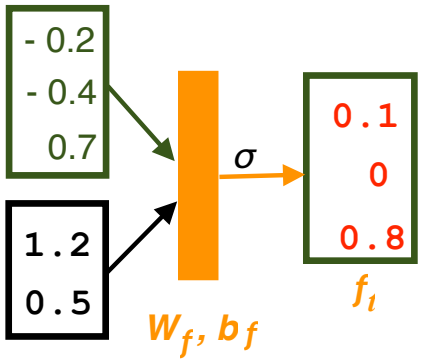
Ex: **France** is where I grew up, but I now live in Boston. I speak fluent ___.

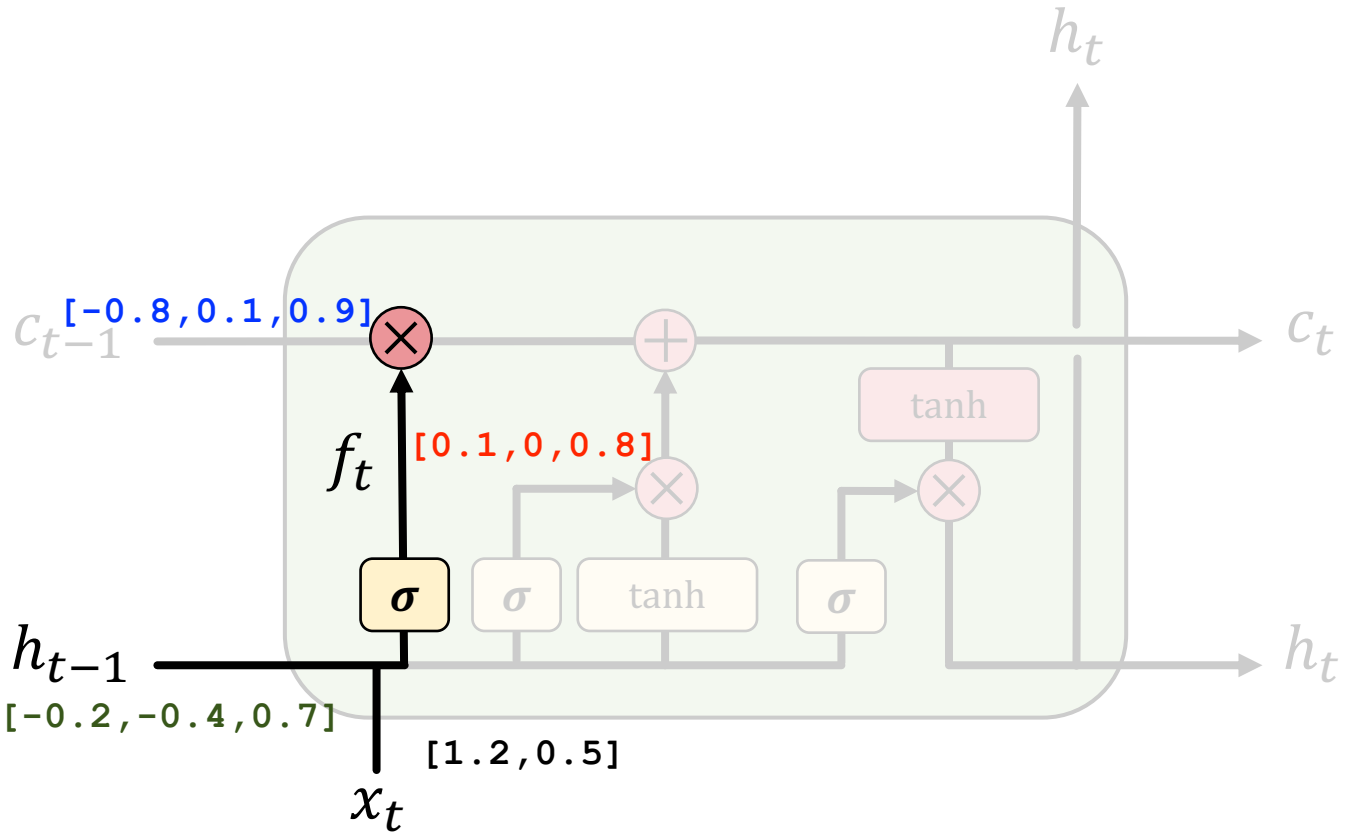


[2, 5]

Long Short Term Memory (LSTMs)

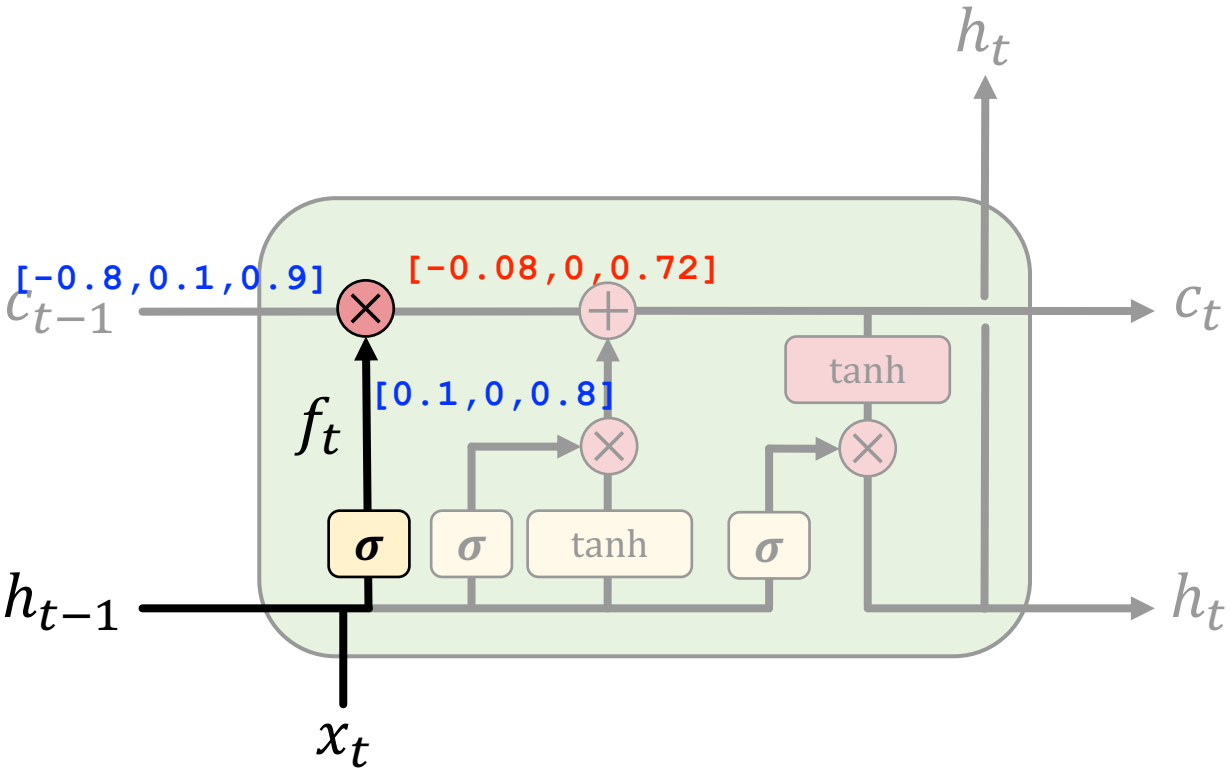
LSTMs **forget irrelevant** parts of the previous state

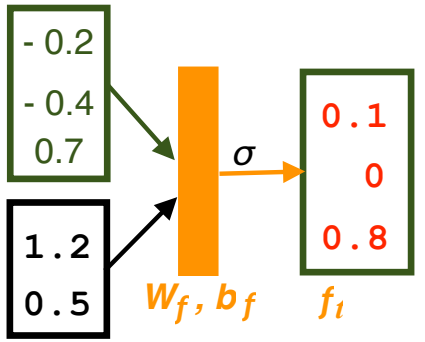




[2, 5]

LSTMs: forget irrelevant information





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

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

Ex: Mady and Monica walk in to the room together, later Richard walks in to the room. Mady said "hi" to ____??

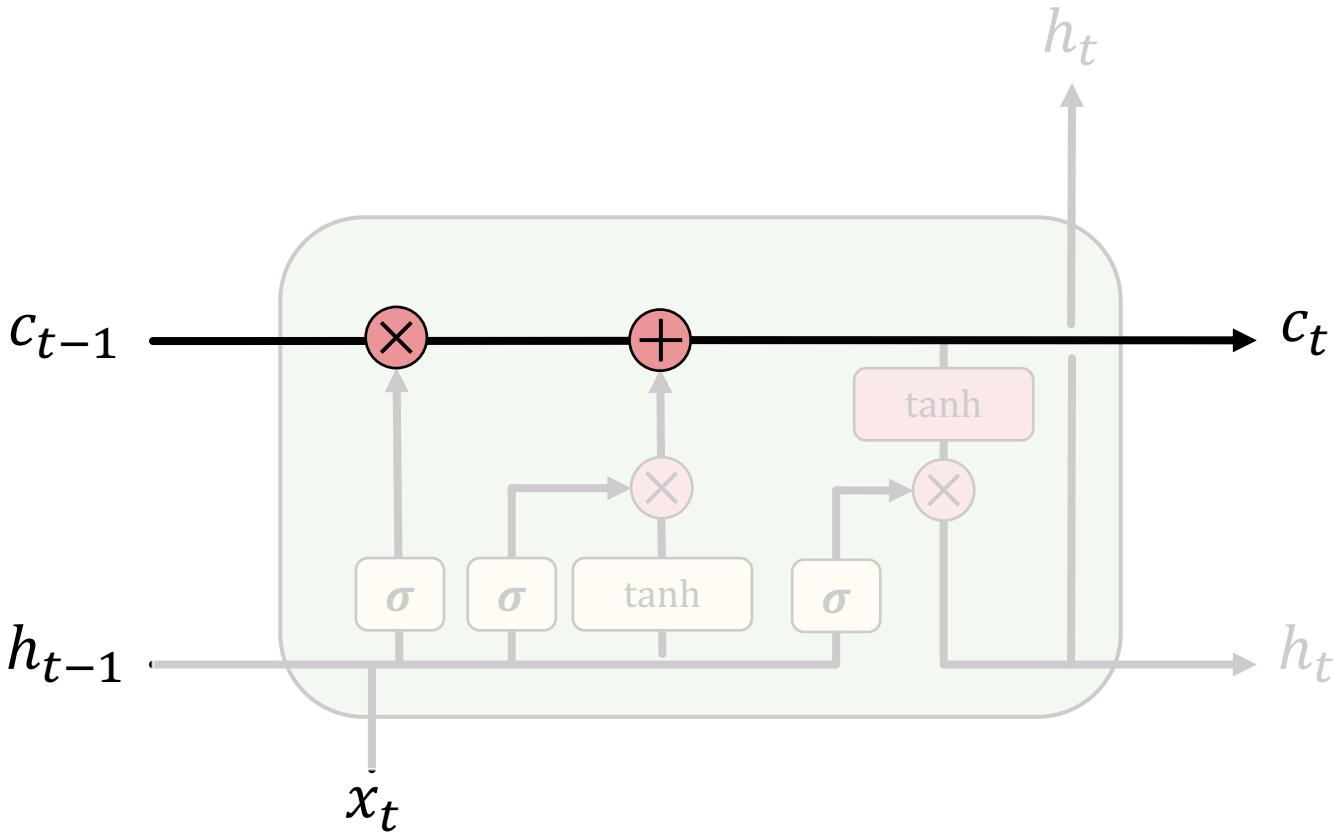
To answer this question, the cell state might change from Monica to Richard.

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

[2, 5]

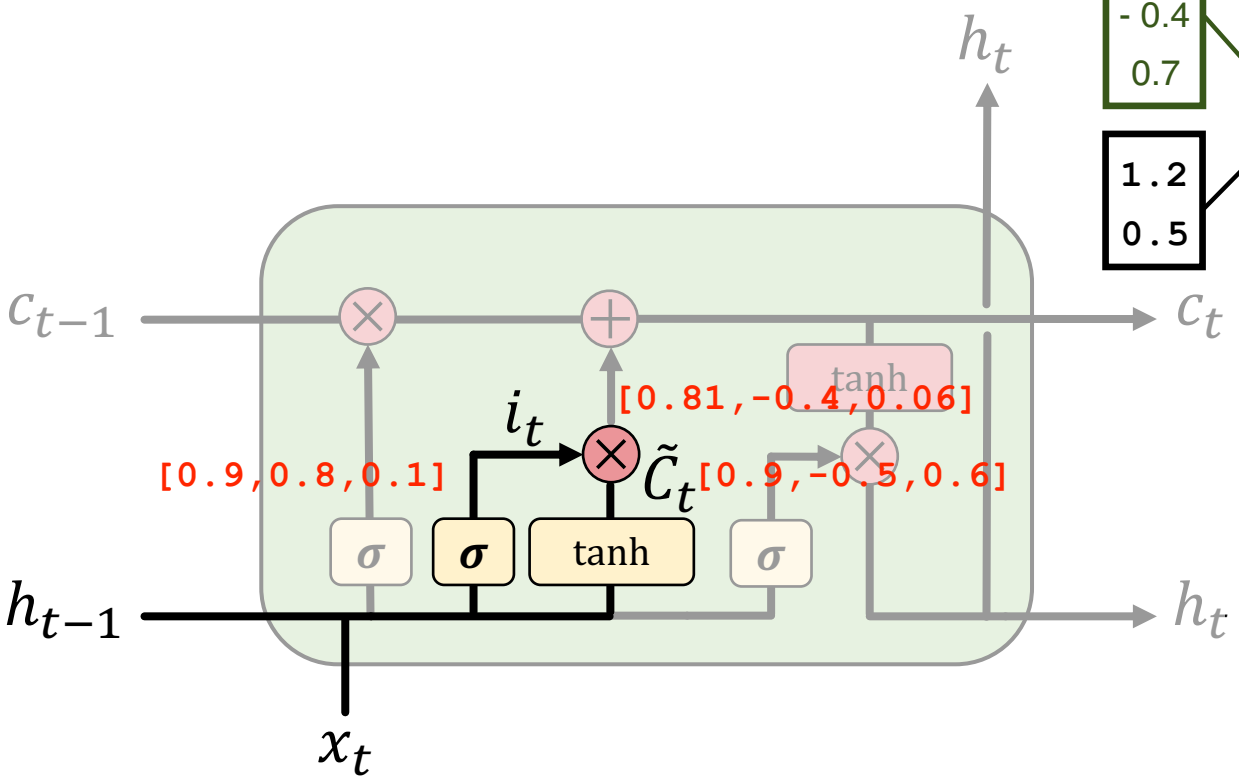
Long Short Term Memory (LSTMs)

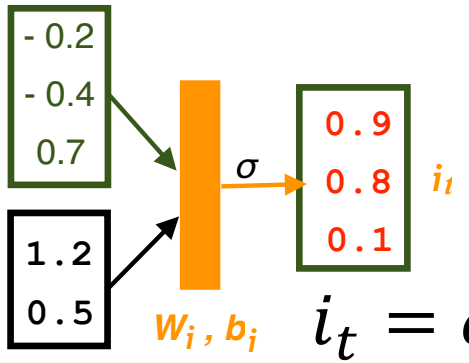
LSTMs **selectively update** cell state values

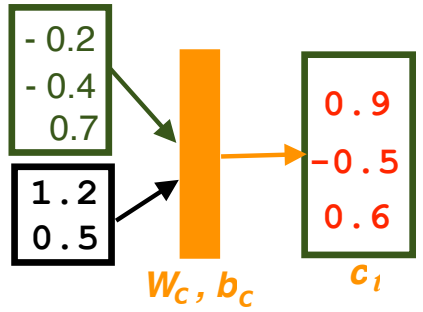


[2, 5]

LSTMs: identify new information to be stored







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

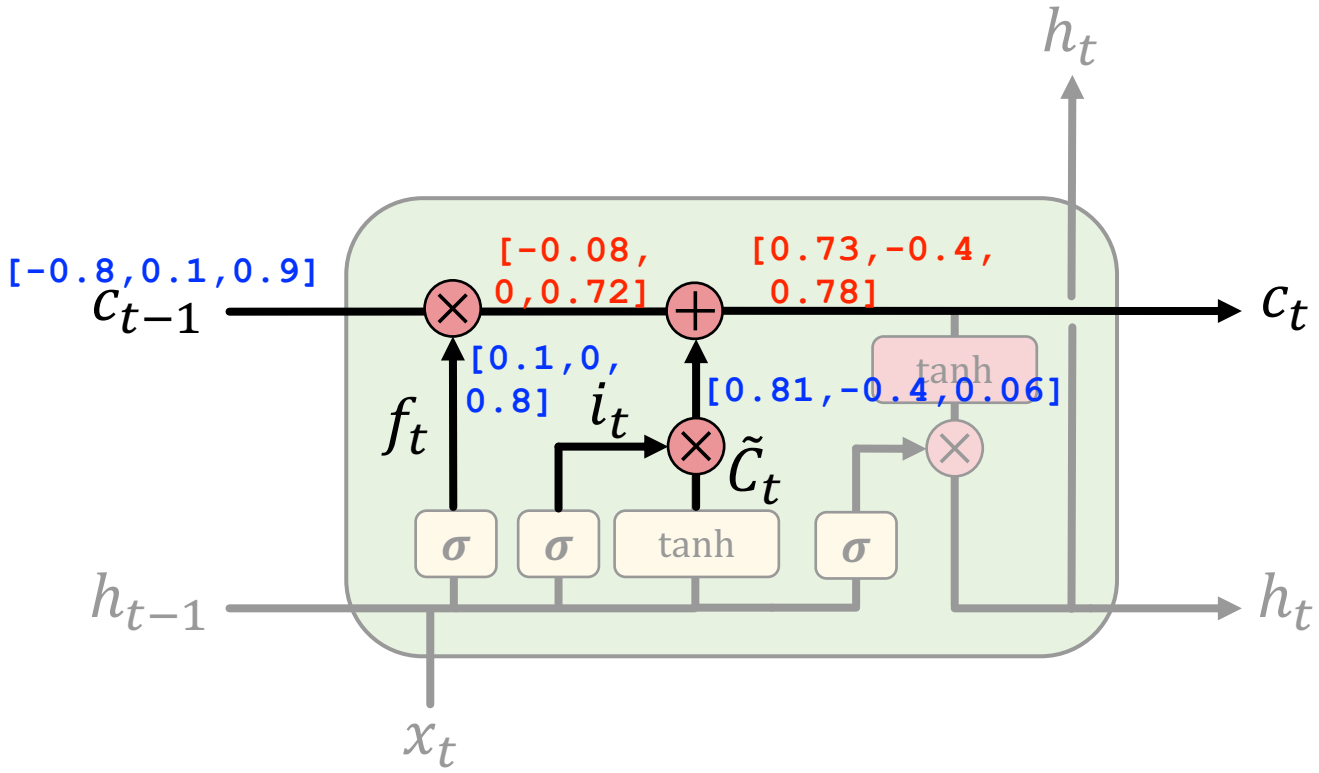
$$\tilde{C}_t = \tanh(\boldsymbol{W_C}[\, h_{t-1}, x_t \,] + b_C)$$

- Sigmoid layer: decide what values to update
- Tanh layer: generate new vector of "candidate values" that could be added to the state, **i.e., increases and decreases in the state.**

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

[2, 5]

LSTMs: update cell state



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

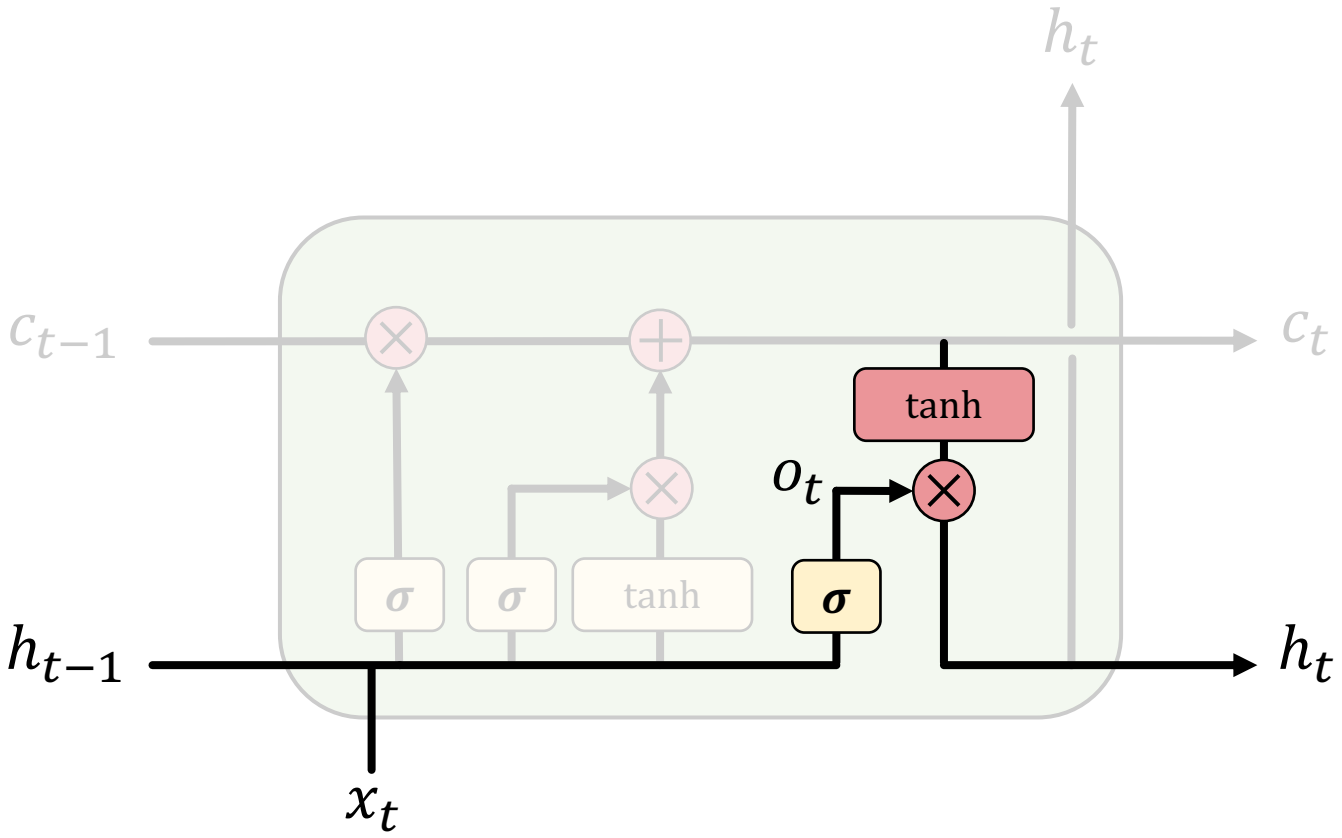
- Apply forget operation to previous internal cell state: $f_t * C_{t-1}$
- Add new candidate values, scaled by how much we decided to update: $i_t * \tilde{C}_t$

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

[2, 5]

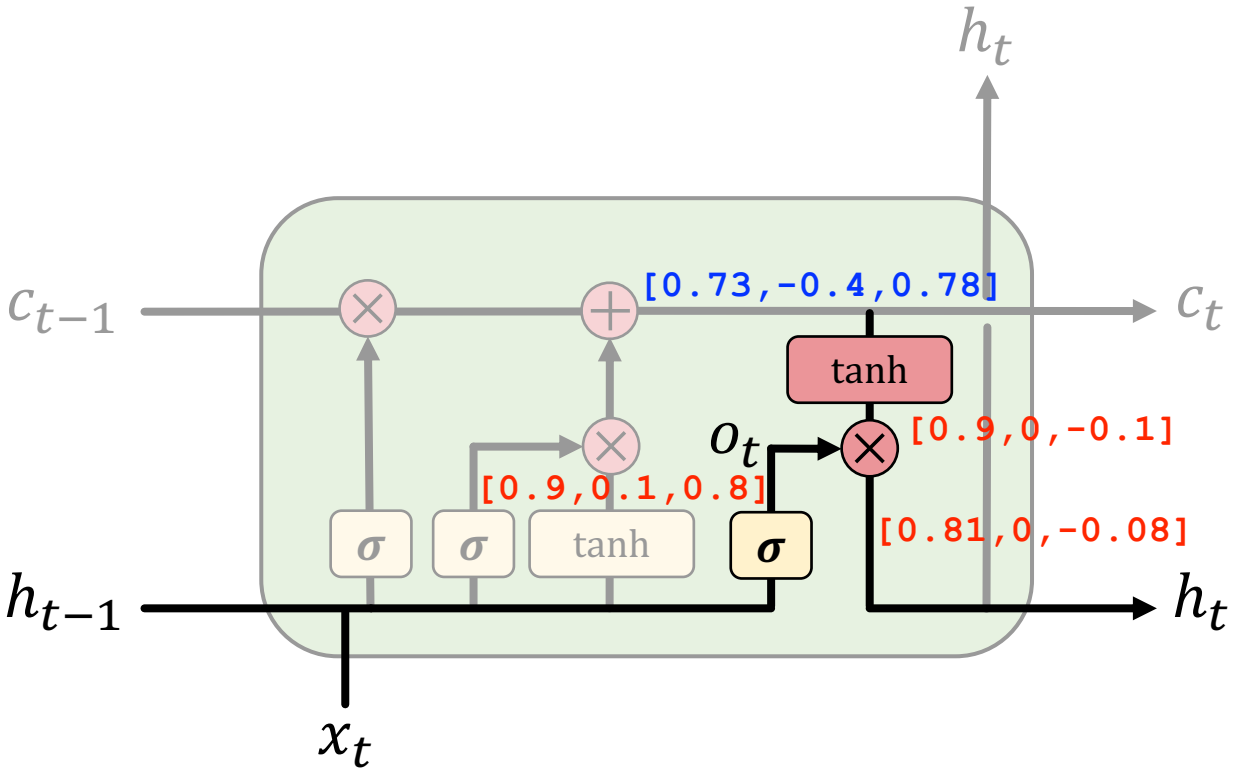
Long Short Term Memory (LSTMs)

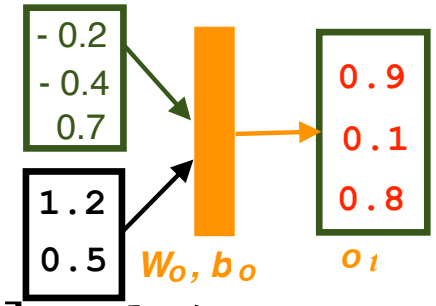
LSTMs use an **output gate** to output certain parts of the cell state



[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

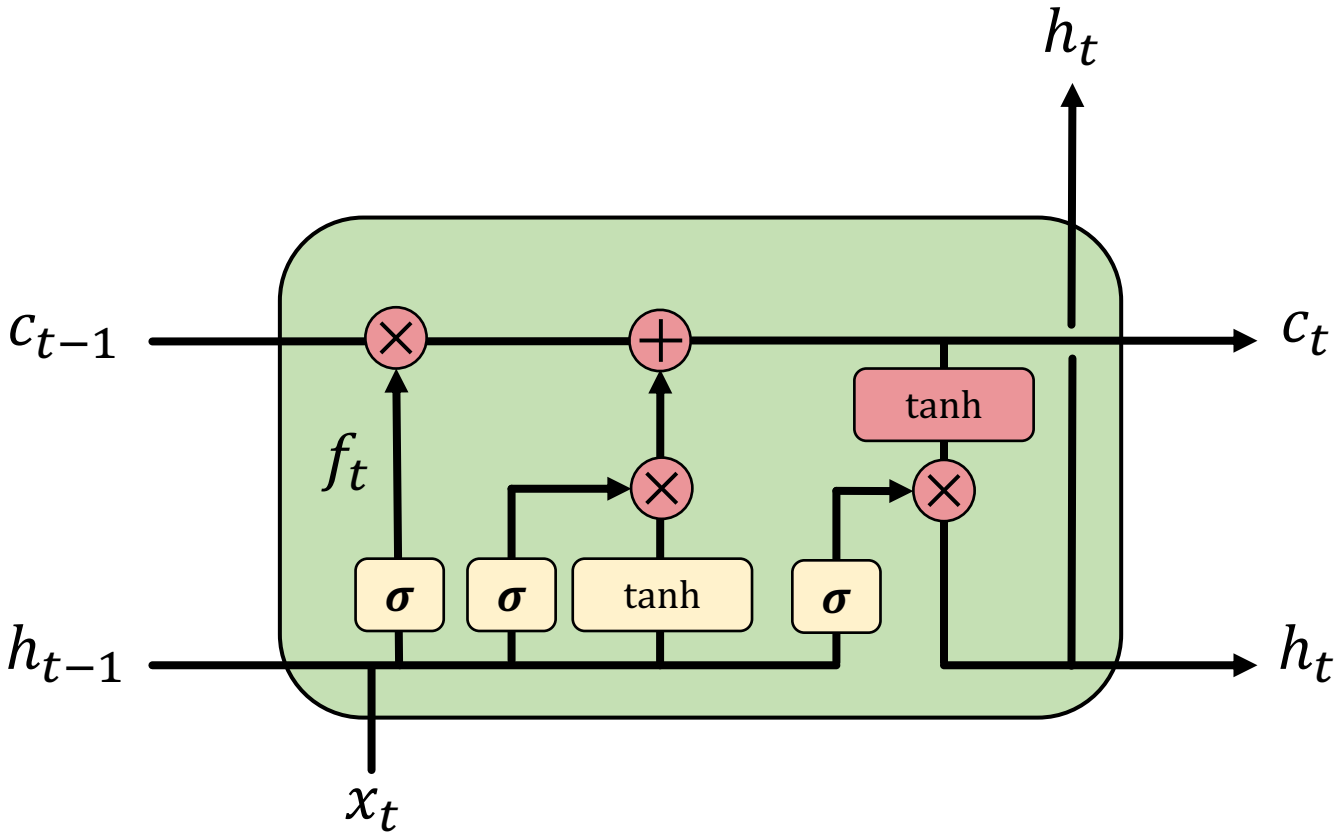
ex: Having seen a subject, may output information relating to a verb.

[2, 5]

Long Short Term Memory (LSTMs)

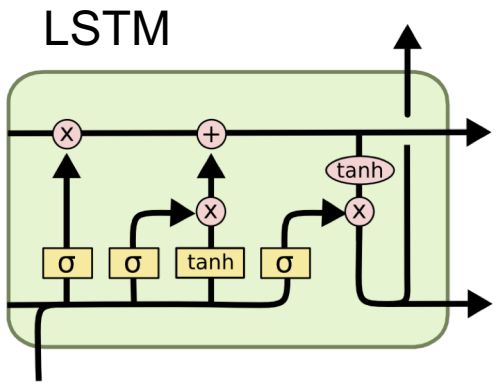
How do LSTMs work?

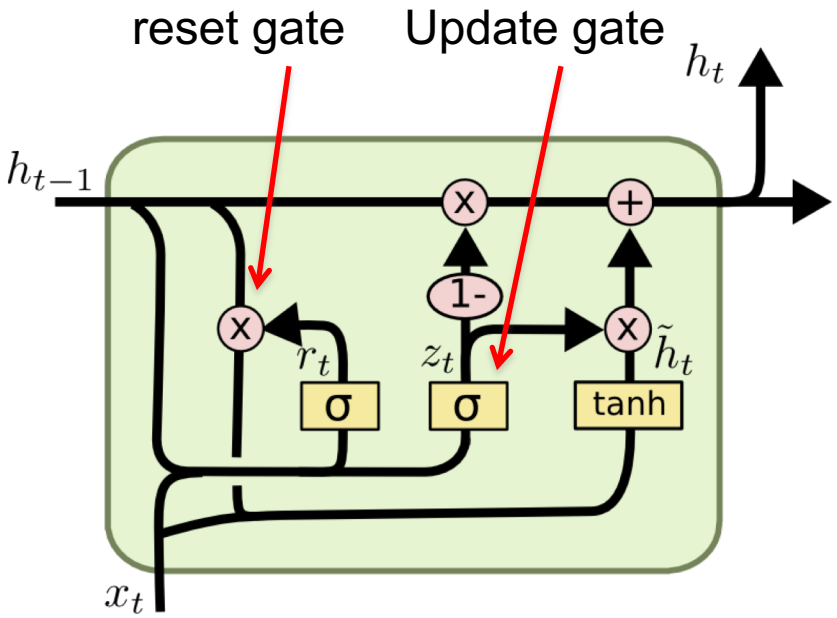
1) Forget 2) Update 3) Output



GRU – gated recurrent unit

(more compression)





$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

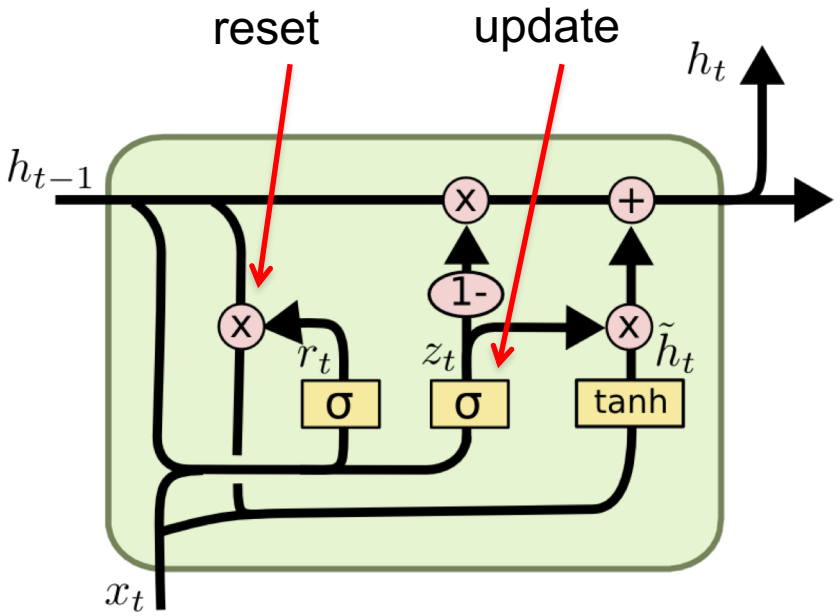
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- It combines the update and input into a single update gate; 2 gates vs. 3 gates;
- It also merges the cell state and hidden state; No need for the cell layer to pass values along.
- GRU is simpler than LSTM; training is more efficient

GRU & LSTM

GRU



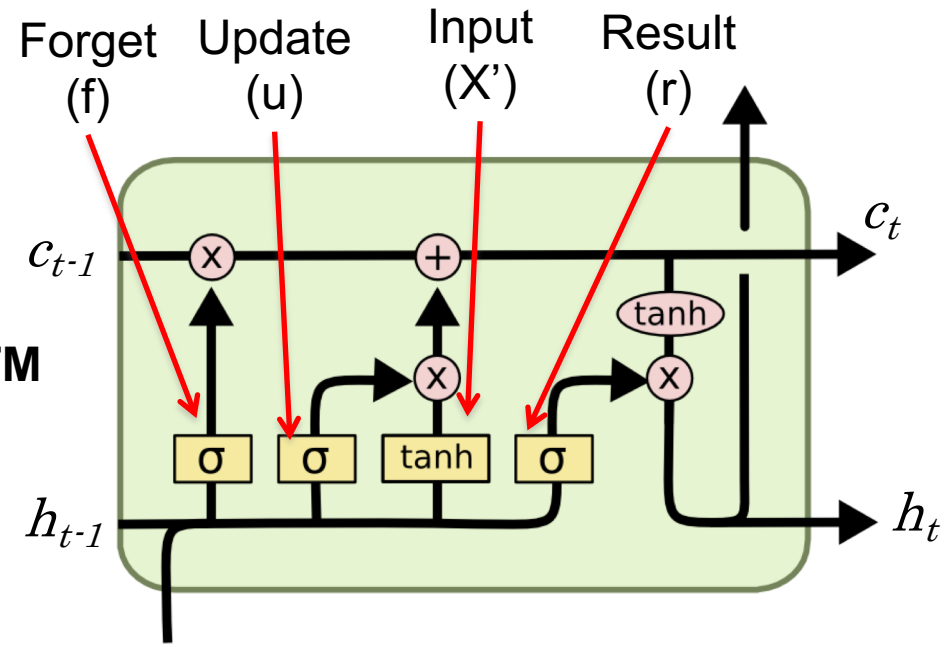
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM

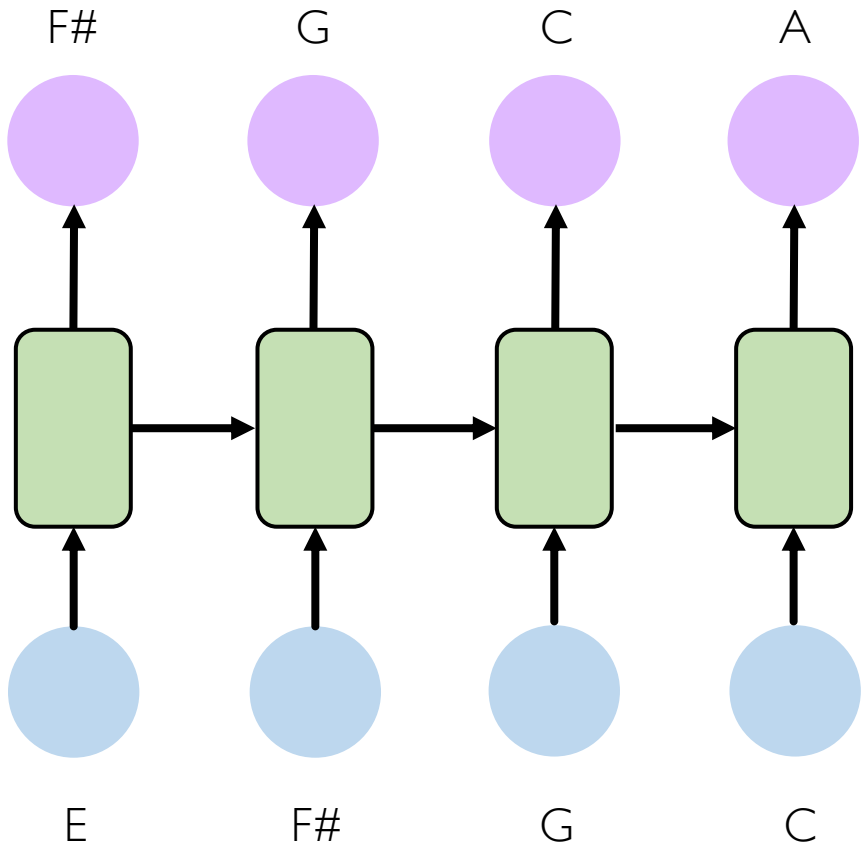


Question:
If input x has 2 units, h/c state has 3 units, how many parameters are there for each model? For simplicity, let's assume bias is not used.

$$f = \sigma(X.W_f + b_f)$$
$$u = \sigma(X.W_u + b_u)$$
$$r = \sigma(X.W_r + b_r)$$
$$X' = \tanh(X.W_c + b_c)$$
$$C_t = f * C_{t-1} + u * X'$$
$$H_t = r * \tanh(C_t)$$

RNN Applications

Example task: music generation



Input: sheet music

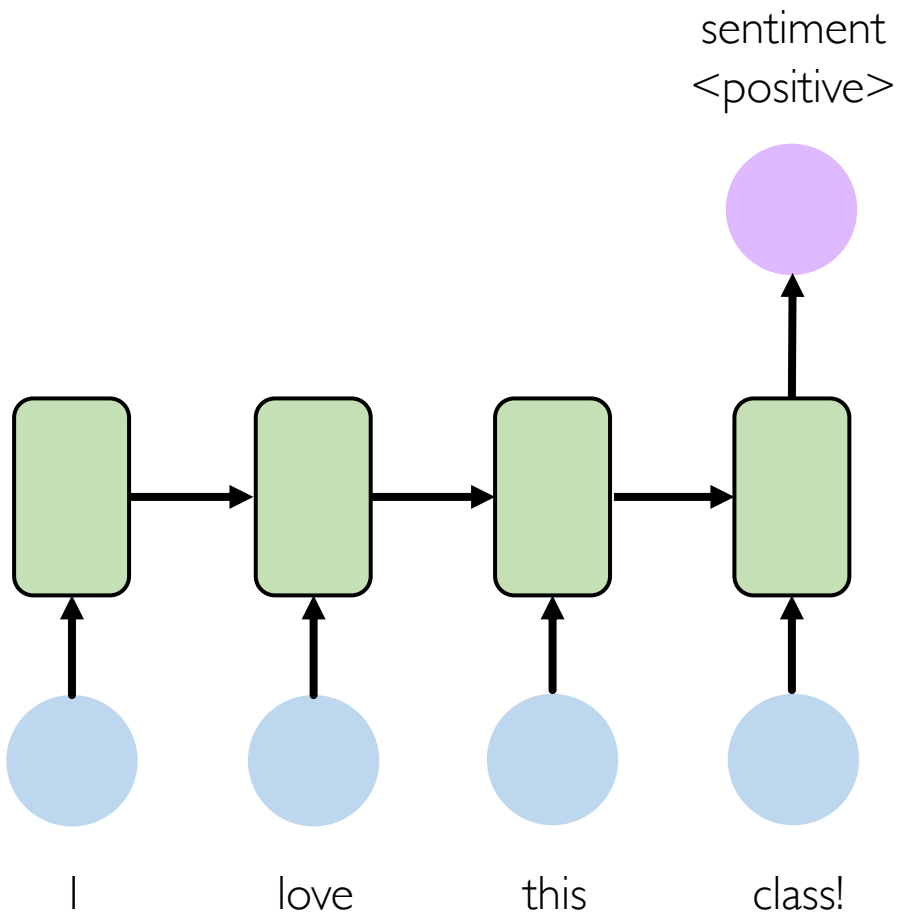
Output: next character in sheet music

6.S191 Lab!

Adapted from H. Suresh, 6.S191 2018

[6]

Example task: sentiment classification



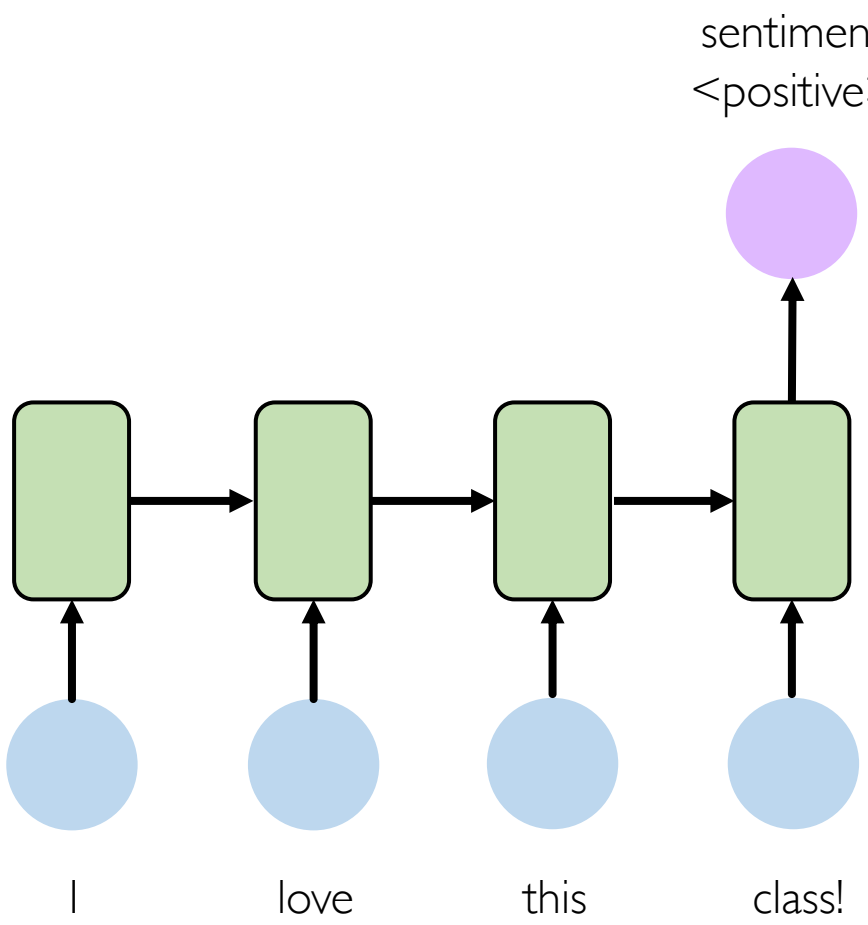
Input: sequence of words

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

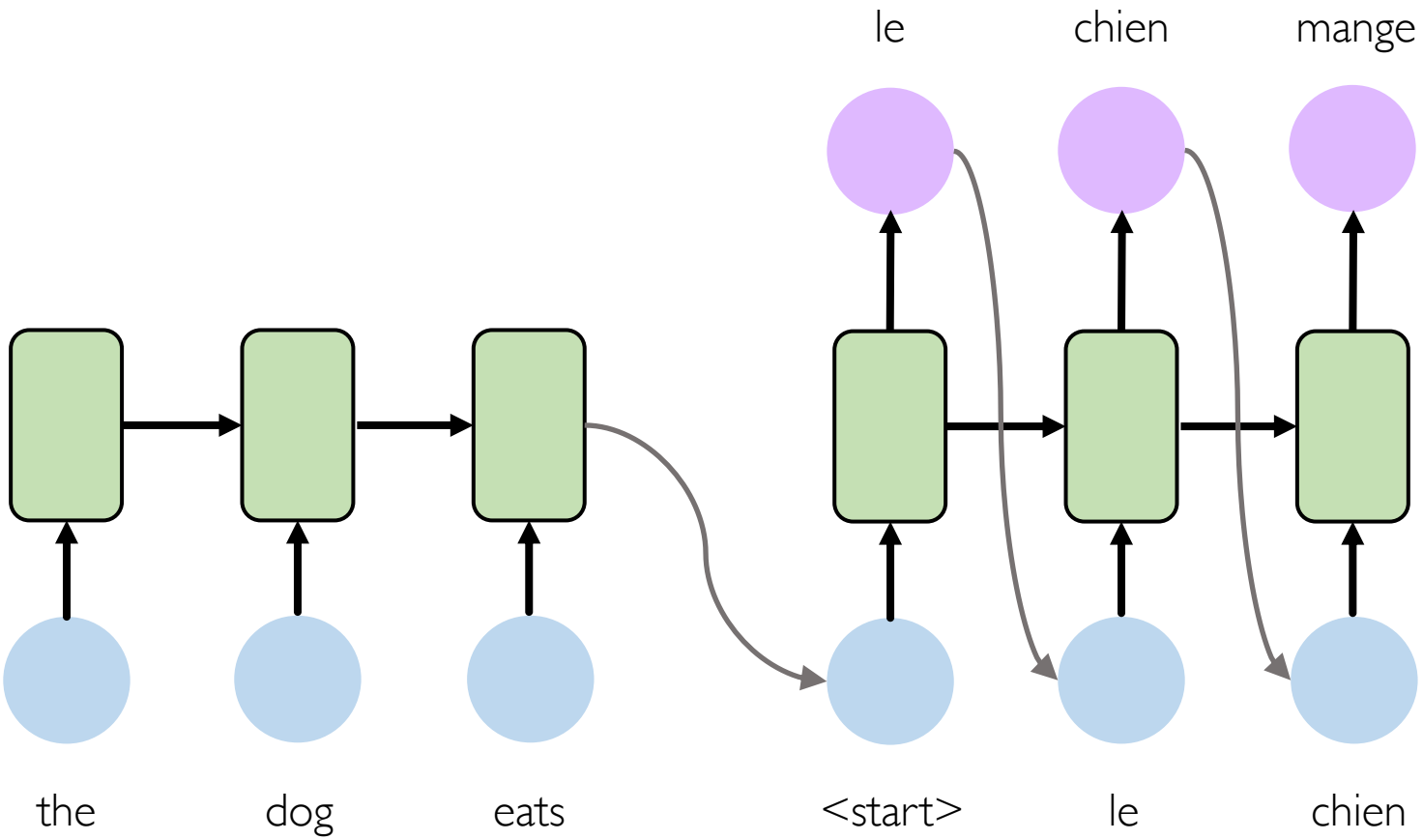
Example task: sentiment classification



Tweet sentiment classification

Adapted from H. Suresh, 6.S191 2018

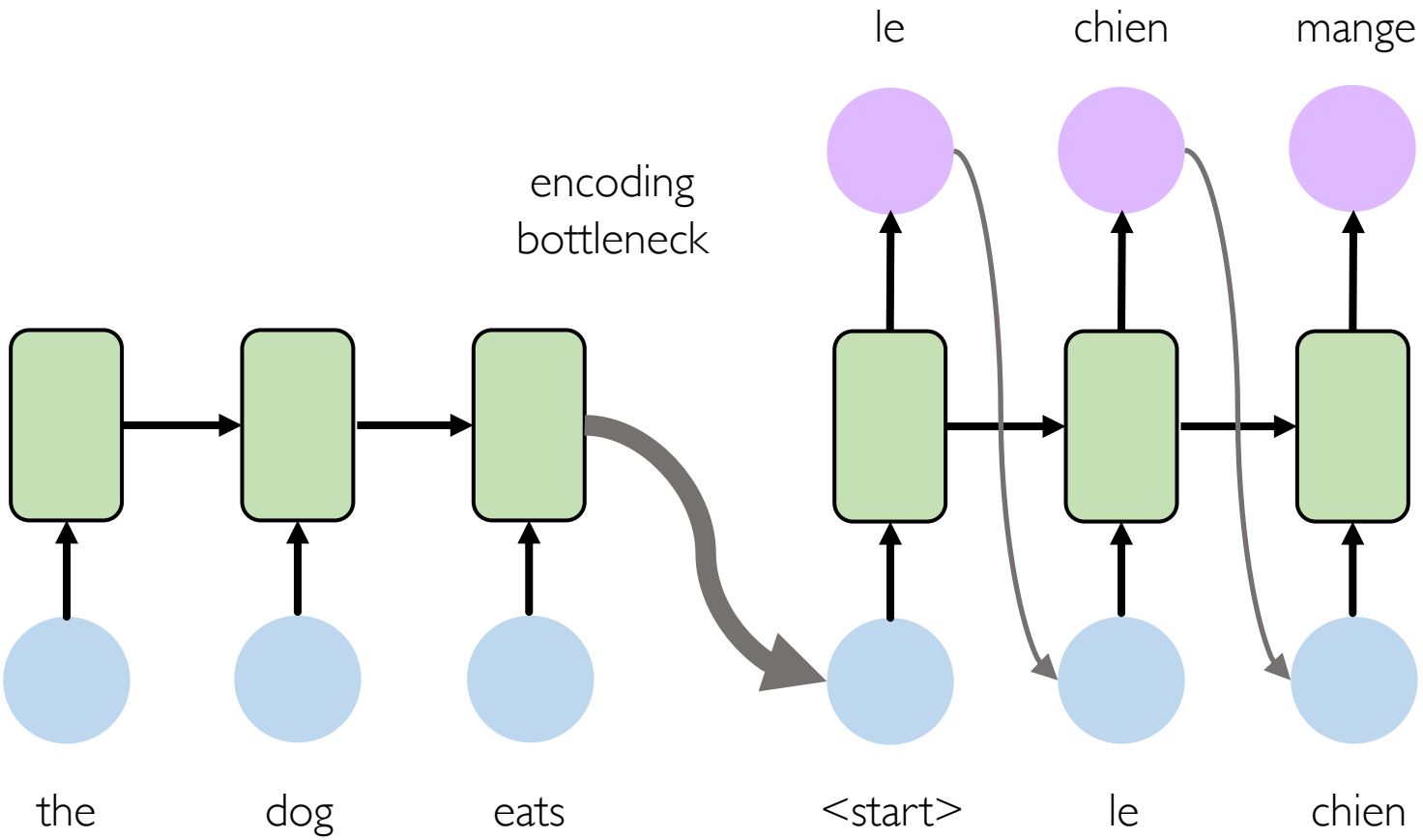
Example task: machine translation



Adapted from H. Suresh, 6.S191 2018

[8,9]

Example task: machine translation

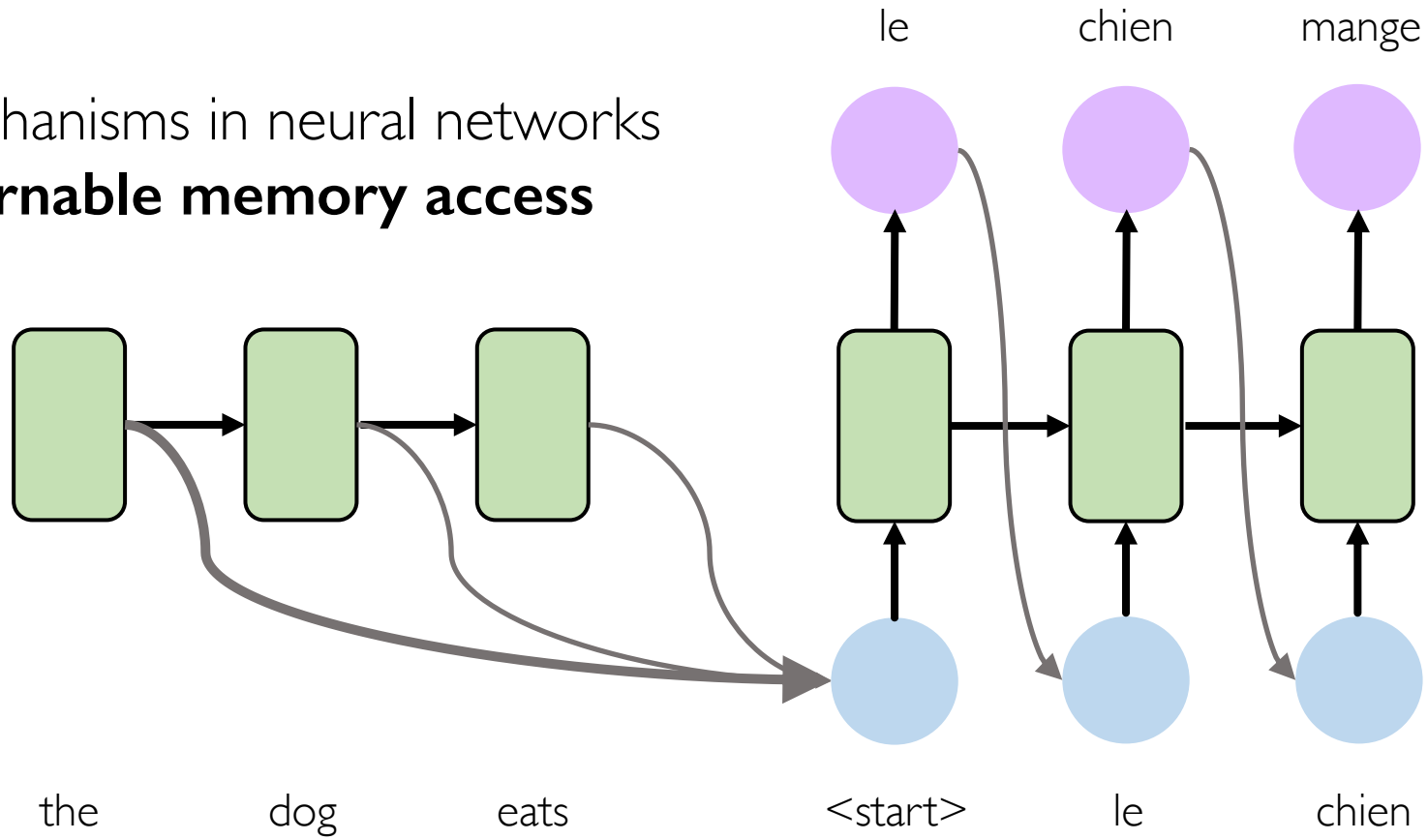


Adapted from H. Suresh, 6.S191 2018

[8,9]

Attention mechanisms

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

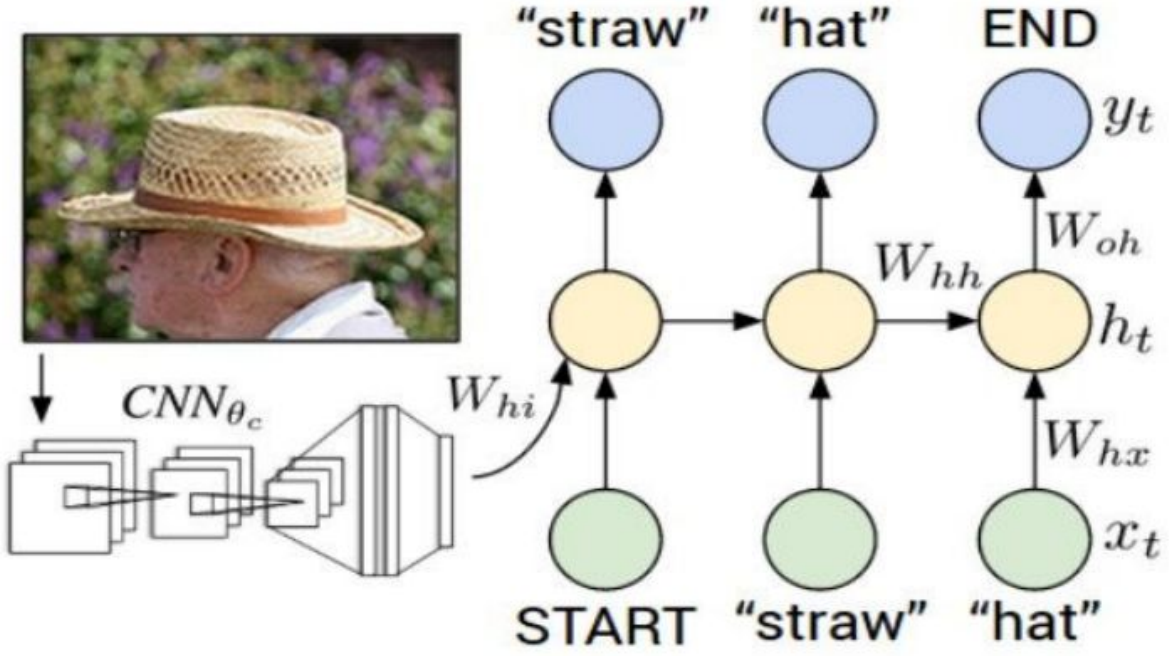


Adapted from H. Suresh, 6.S191 2018

[8,9]

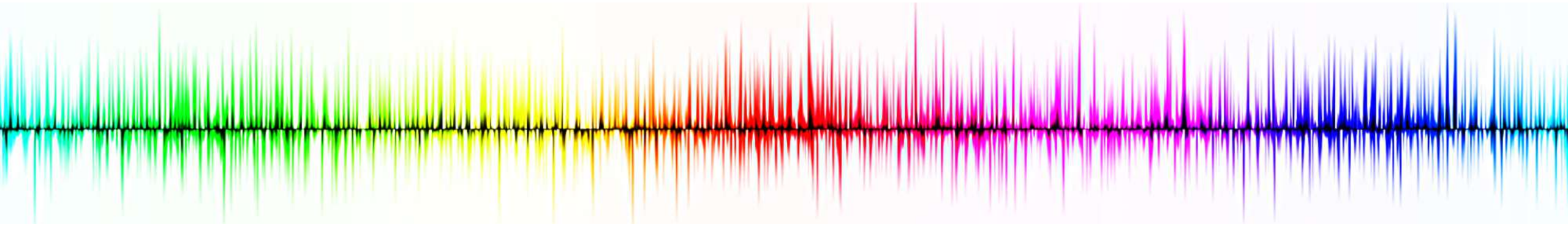
One-to-Many: Image captioning

GOAL: Given image, generate a sentence to describe its content.



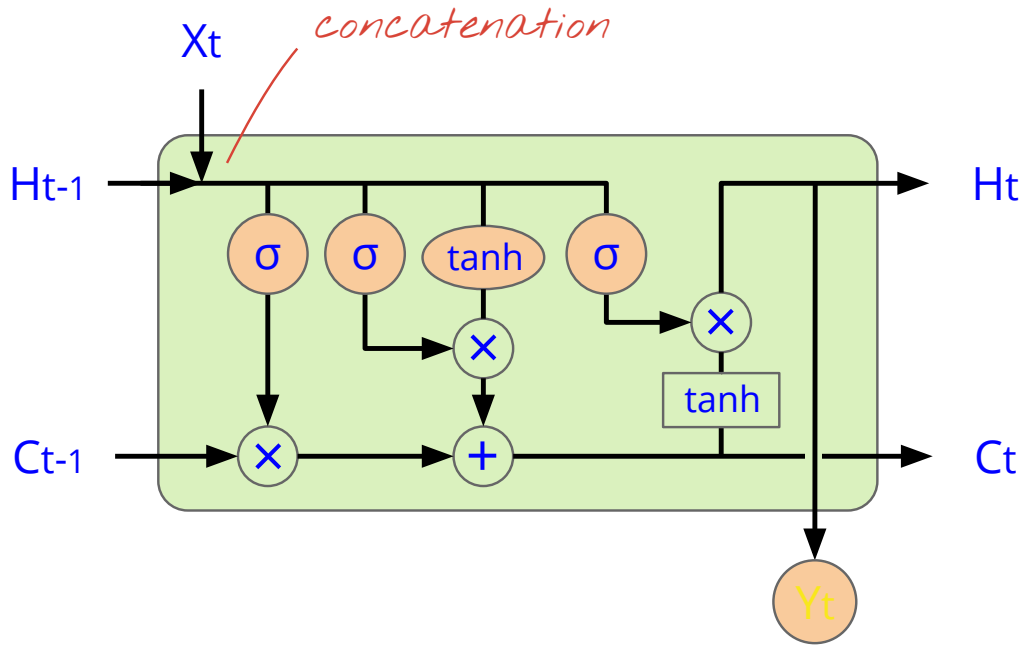
Recurrent neural networks (RNNs)

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



LSTM = Long Short Term Memory

LSTM



σ tanh Neural net. layers

× tanh Element-wise operations

$$X = X_t \oplus H_{t-1}$$

$$f = \sigma(X.W_f + b_f)$$

$$u = \sigma(X.W_u + b_u)$$

$$r = \sigma(X.W_r + b_r)$$

$$X' = \tanh(X.W_c + b_c)$$

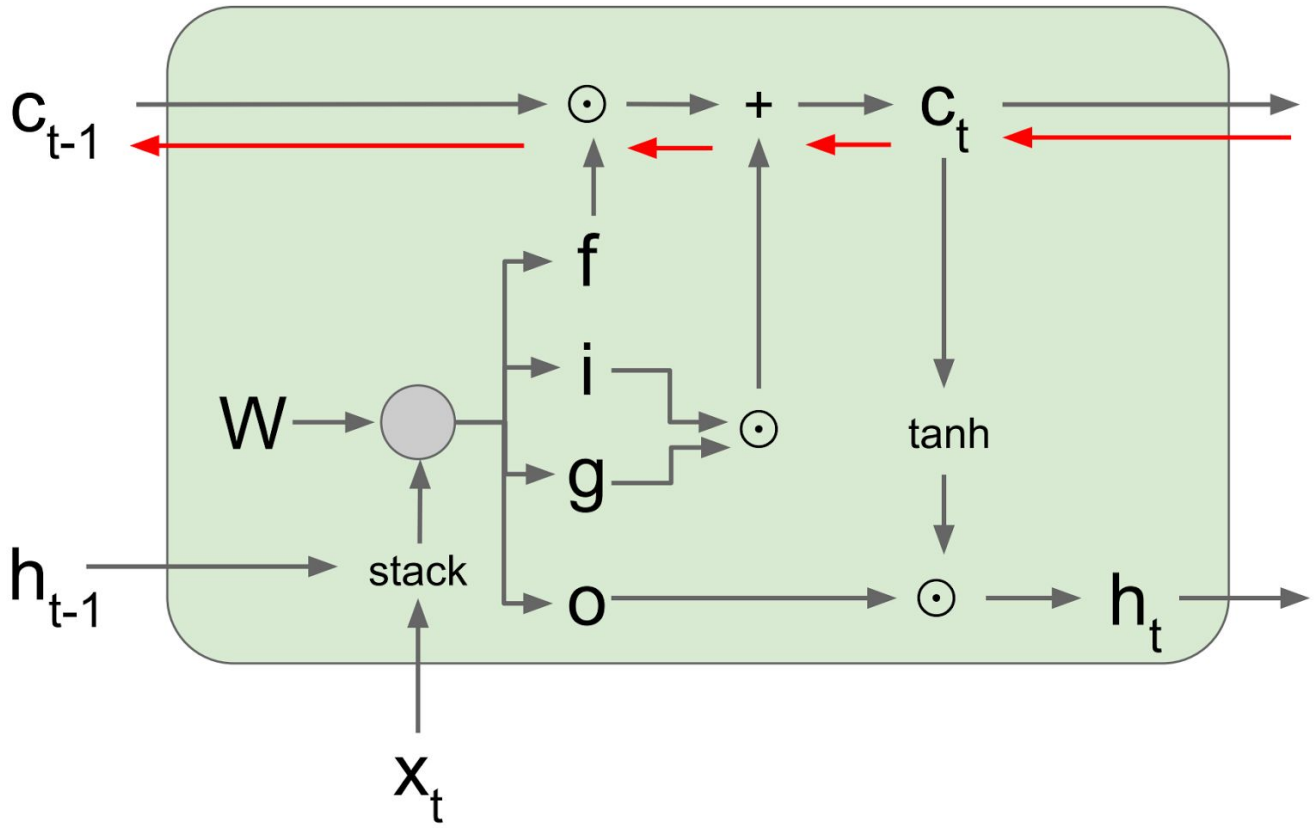
$$C_t = f * C_{t-1} + u * X'$$

$$H_t = r * \tanh(C_t)$$

$$Y_t = \text{softmax}(H_t.W + b)$$

Long Short-term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

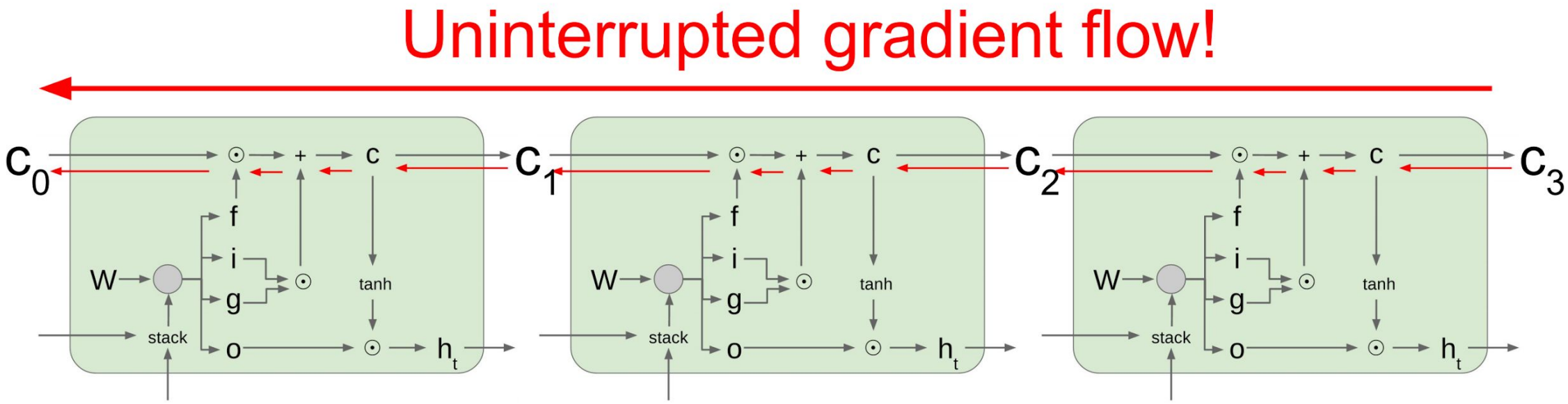
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Note: this slide uses different notation for “element-wise multiplication”

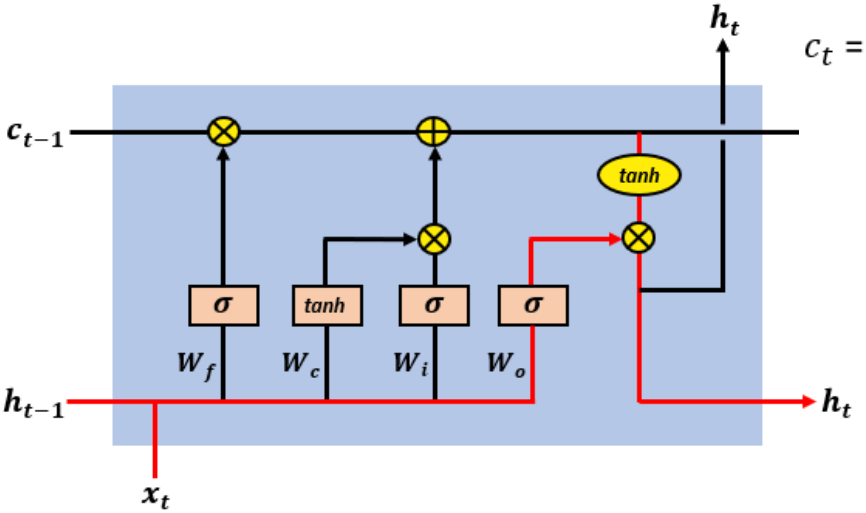
Long Short-term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



LSTM Gradient Flow

Feel free to skip the part!



$$c_t = c_{t-1} \otimes \sigma(W_f \cdot [h_{t-1}, x_t]) \oplus$$

$$tanh\,(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t \qquad (5)$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial}{\partial c_{t-1}}[c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t]$$

$$= \frac{\partial}{\partial c_{t-1}}[c_{t-1} \otimes f_t] + \frac{\partial}{\partial c_{t-1}}[\tilde{c}_t \otimes i_t]$$

$$= \frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t$$

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k} \cdots \frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial W}$$

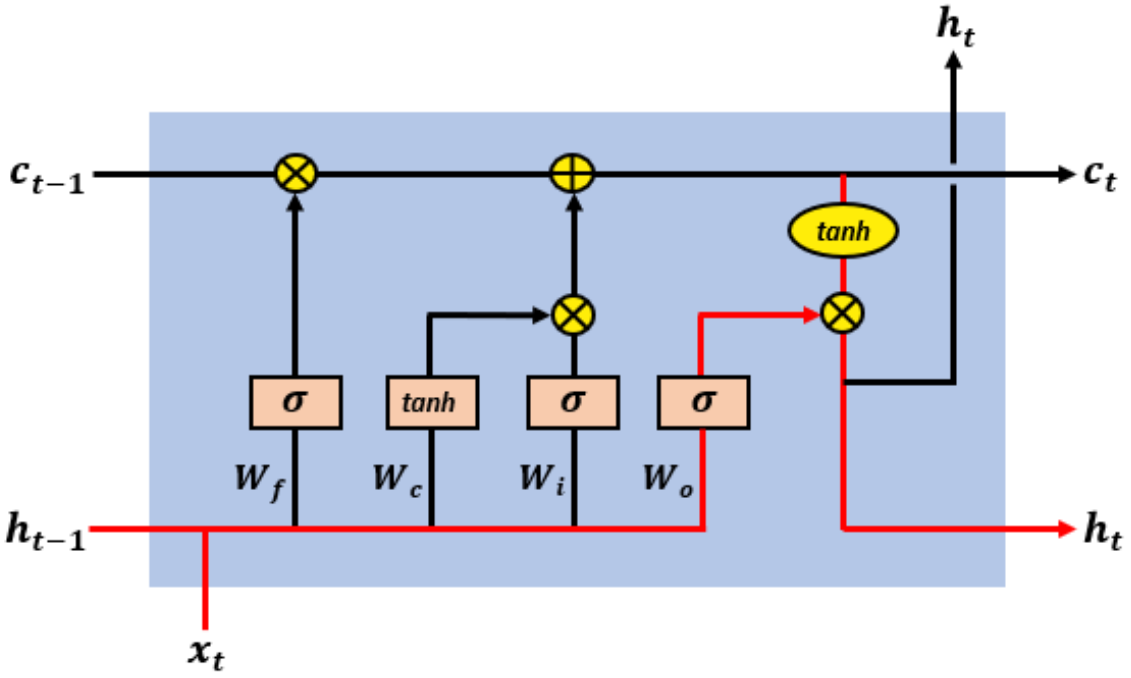
This term may cause gradient vanish

$$= \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k} \frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W} \qquad (4)$$

https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

LSTM Gradient Flow

Feel free to skip the part!



$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot c_{t-1} \quad A_t$$

$$+ f_t \quad B_t$$

$$+ \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot \tilde{c}_t \quad C_t$$

$$+ \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes tanh'(c_{t-1}) \cdot i_t \quad D_t$$

$$\frac{\partial c_t}{\partial c_{t-1}} = A_t + B_t + C_t + D_t \qquad (6)$$

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k} \cdots \frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial W}$$

$$= \frac{\partial E_k}{\partial h_k}\frac{\partial h_k}{\partial c_k}\left(\prod_{t=2}^{k} \frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W} \qquad (4)$$

- **The gradient contains the forget gate's vector of activations**
- This allows the LSTM to decide, at each time step, that certain information should not be forgotten
- **The cell state gradient is an additive function.** The LSTM updates and balances the values of the four components making it more likely the additive expression does not vanish

https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577