CSCI 1470/2470
Spring 2024

Ritambhara Singh

March 04, 2024
Monday

Deep Learning

Long Short Term Memory (LSTM)

Project check-in#1 starts today!
Proposal due on March 15.

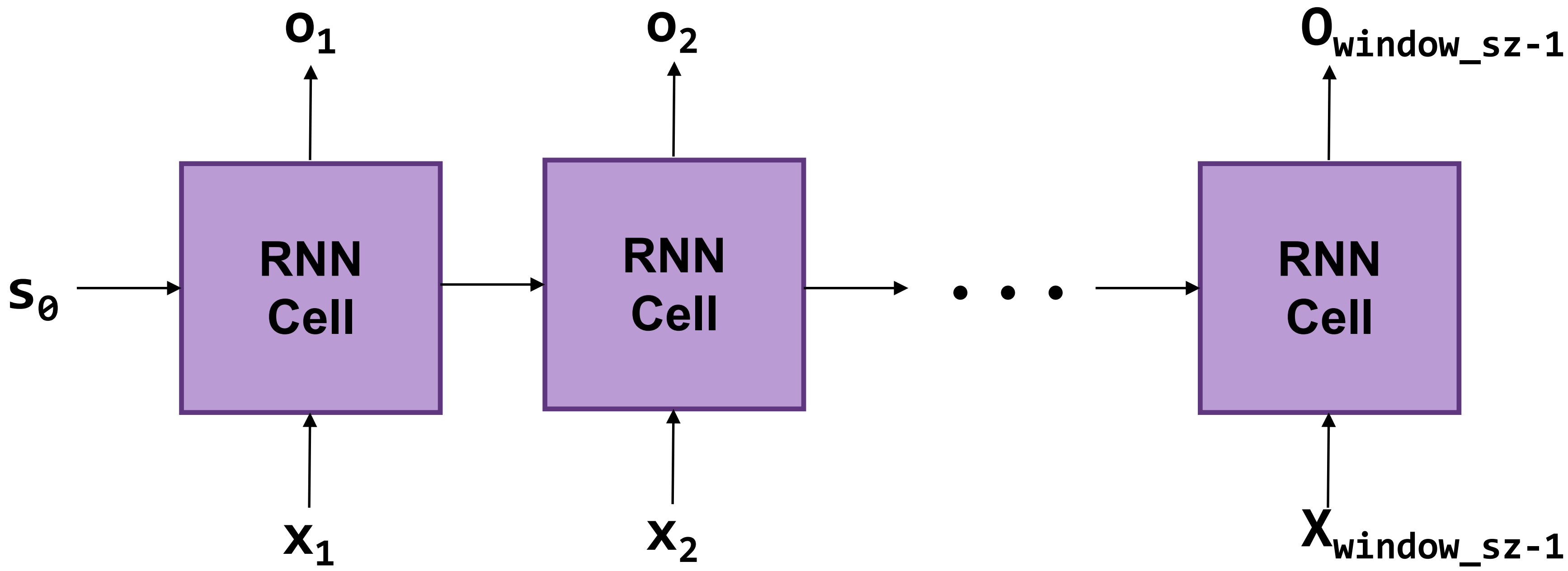
ChatGPT prompt “minimalist landscape painting of a deep underwater scene with a blue tang fish in the bottom right corner”

Review: RNN

Recurrent Neural Networks are networks in the form of a directed ***cyclic*** graph.

They pass previous *state* information from previous computations to the next.

They can be used to process sequence data with relatively low model complexity when compared to feed forward models.



RNN

Goal of RNNs: remember information from the past

RNN

"The dog that my family had when I was a child had a fluffy _____."

RNN

"The dog that my family had when I was a child had a fluffy _____."

Want: *"tail"*

RNN Weaknesses

But….RNNs are not very good at remembering things ***far*** in the past.

RNN Weaknesses

"The dog that my family had when I was a child had a fluffy ______."

- To predict "tail" RNN needs to remember the subject of the sentence - "dog"

RNN Weaknesses

"The dog that my family had when I was a child had a fluffy ______."

- To predict "tail" RNN needs to remember the subject of the sentence - "dog"
- "dog" and predicted word are separated by **12** words
  - On the outer limit of what a vanilla RNN would be able to remember.

Review: RNN update rule

$$s_t = \rho\big((e_t, s_{t-1})W_r + b_r\big)$$

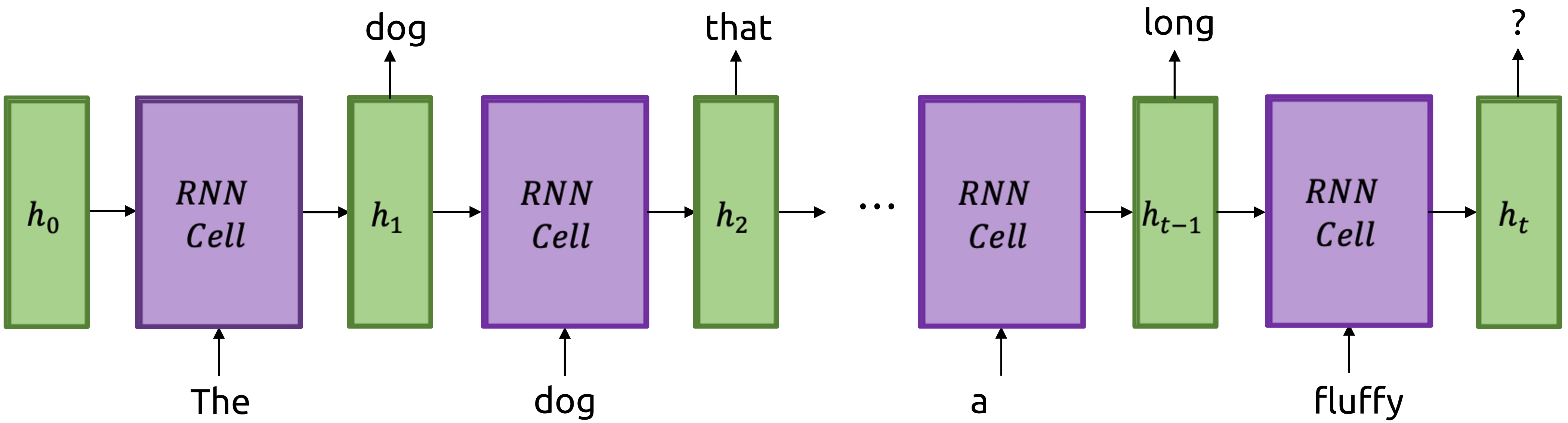
Activation function

Embedding of word t

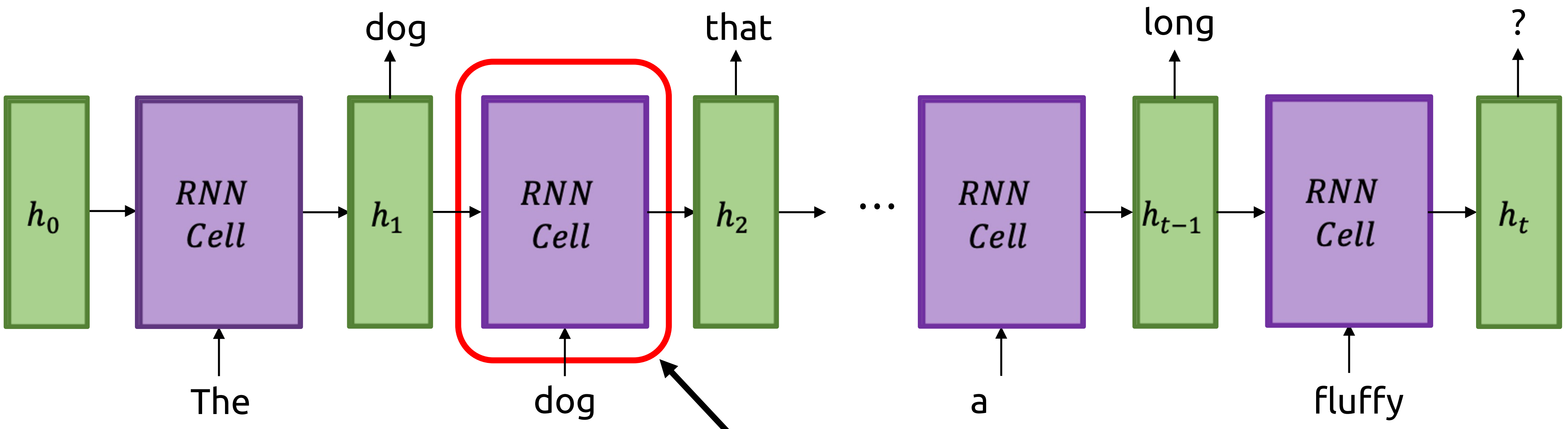
$$o_t = \sigma(s_t W_o + b_o)$$

Can call it a
hidden state

An Illustrative Example:

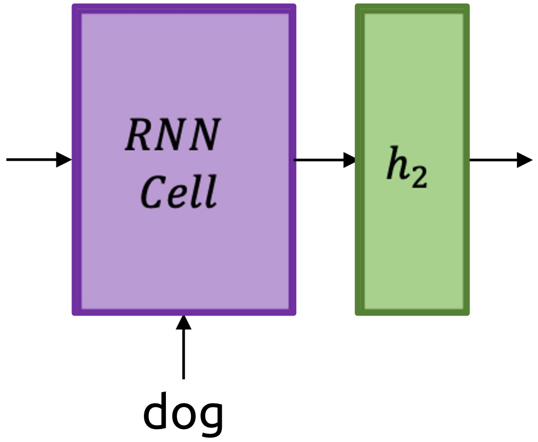


An Illustrative Example:



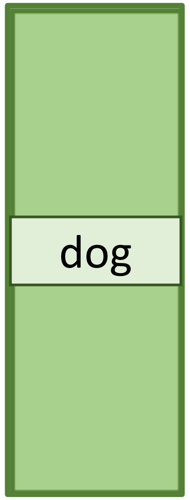
What happens to the information about "dog" as we continue through the network?

An Illustrative Example:



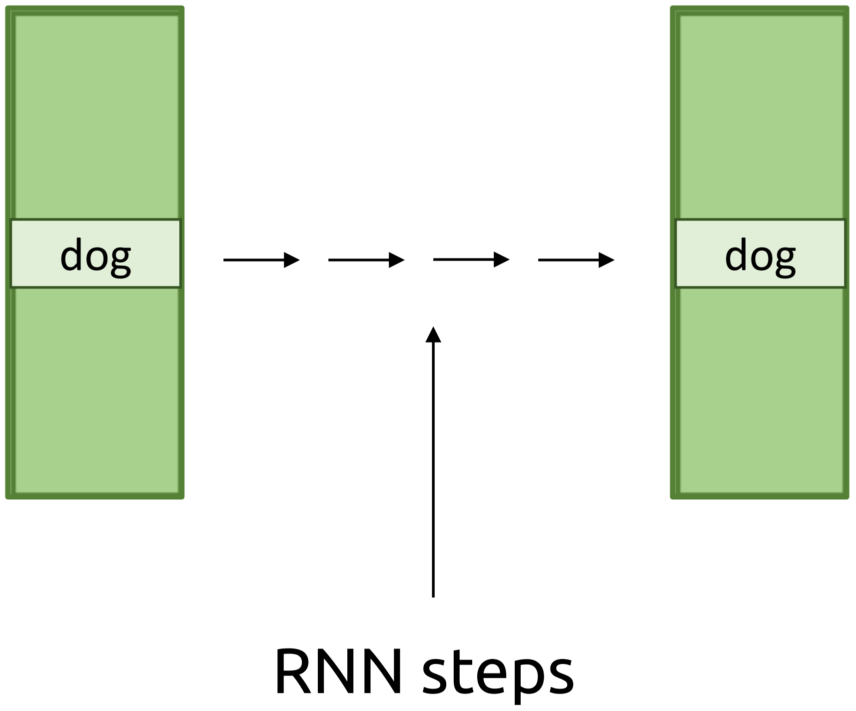
An Illustrative Example:

Can imagine that the information about *"dog"* is stored in some part of the RNN's hidden state vector



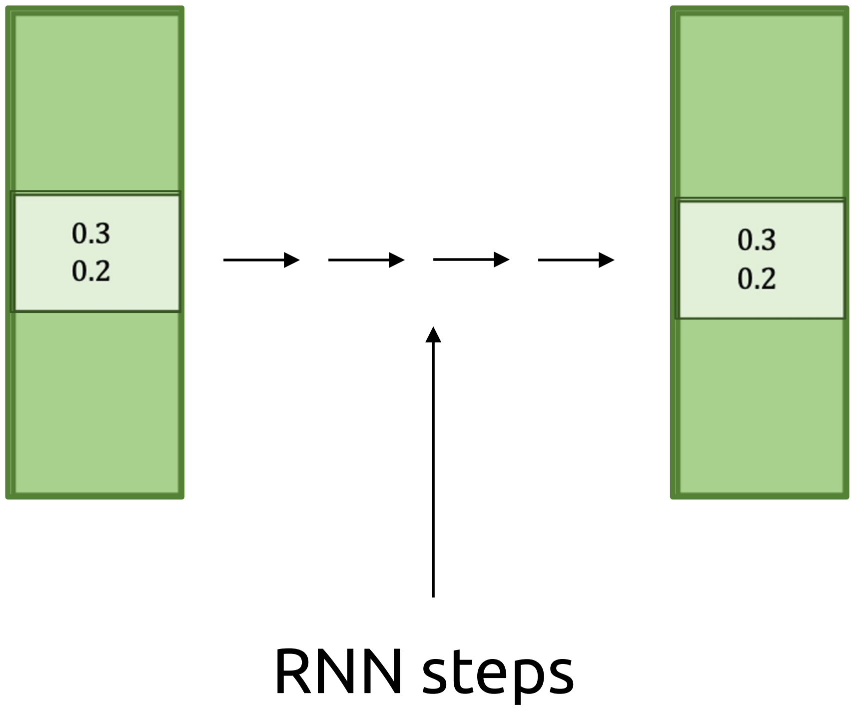
An Illustrative Example:

Through all subsequent RNN steps, we want *“dog”* to stay the same



An Illustrative Example:

If we think of *"dog"* as just a few entries in the vector…



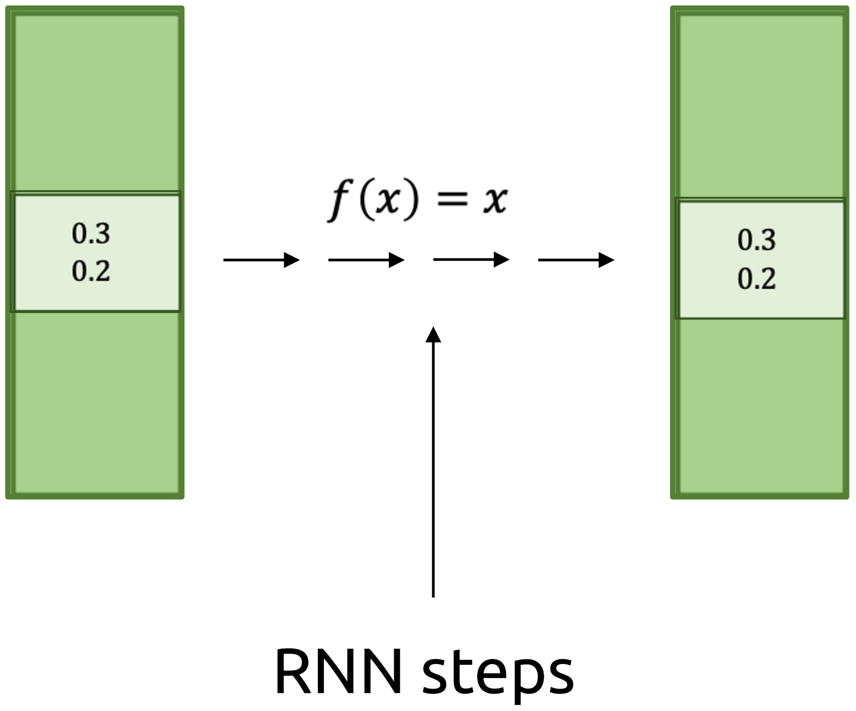
An Illustrative Example:

...to preserve *"dog"* , we need to compute the ***identity function*** over the part of the vector that stores it

But will that happen?

No

Why?



How does this affect the hidden state?

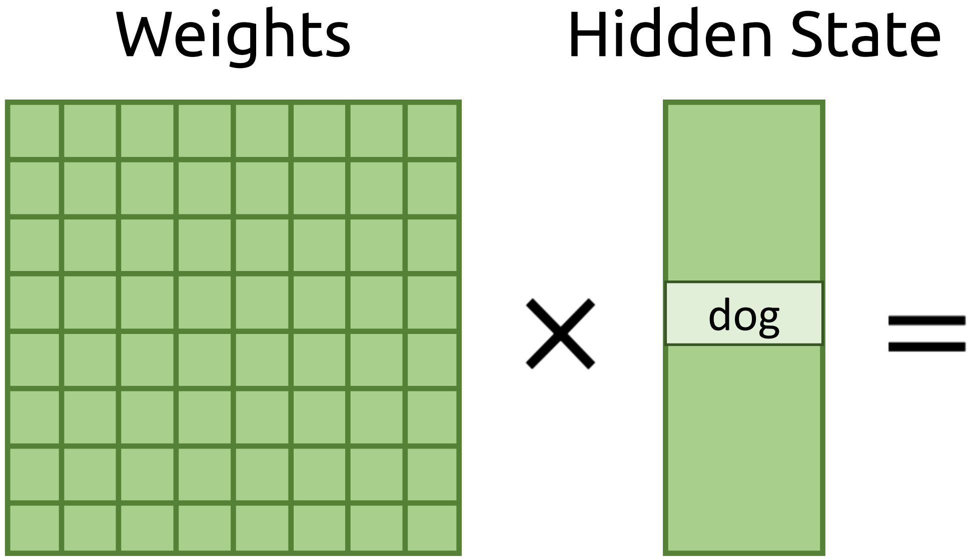
RNN update

$$h_t = \rho((e_t, h_{t-1})W_r + b_r)$$

The hidden state goes through a fully connected layer!

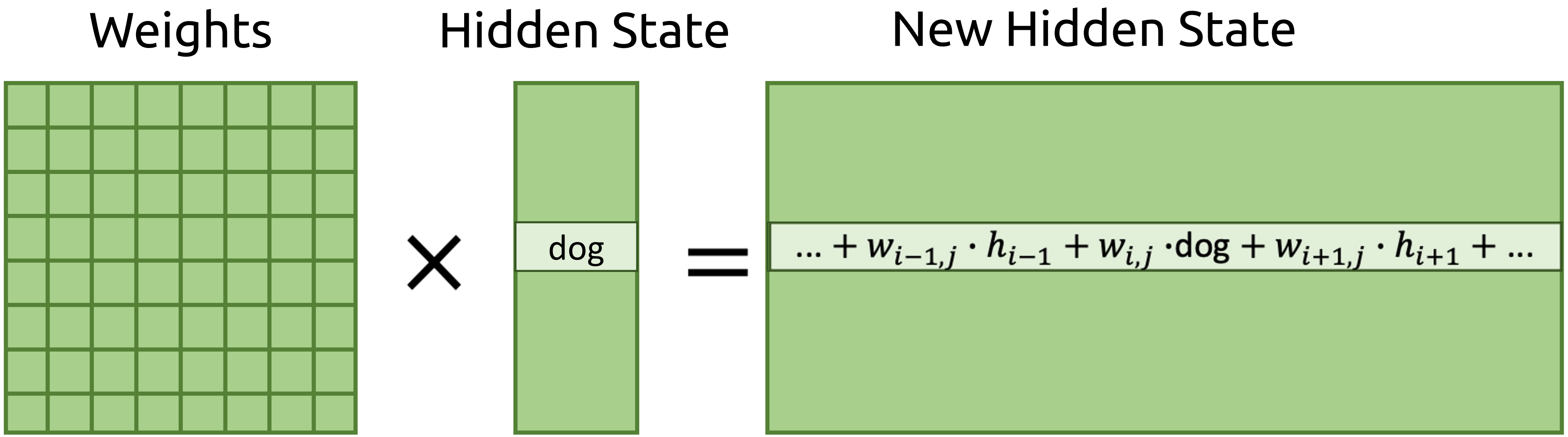
How does this affect the hidden state?

- What will happen to our dog after we multiply our weights by our hidden state?



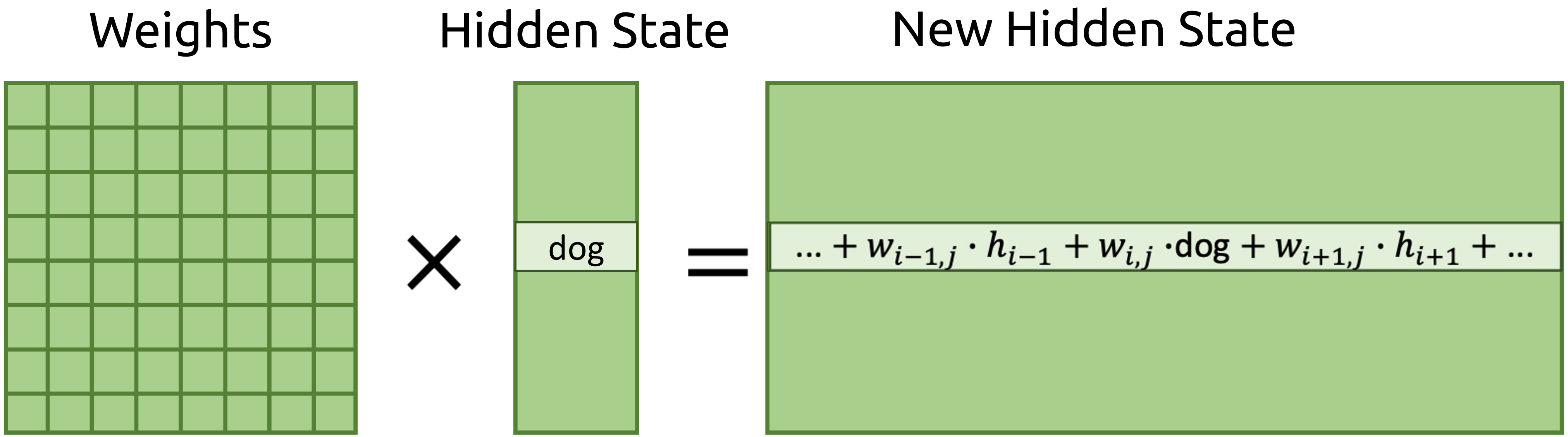
How does this affect the hidden state?

- What will happen to our dog after we multiply our weights by our hidden state?



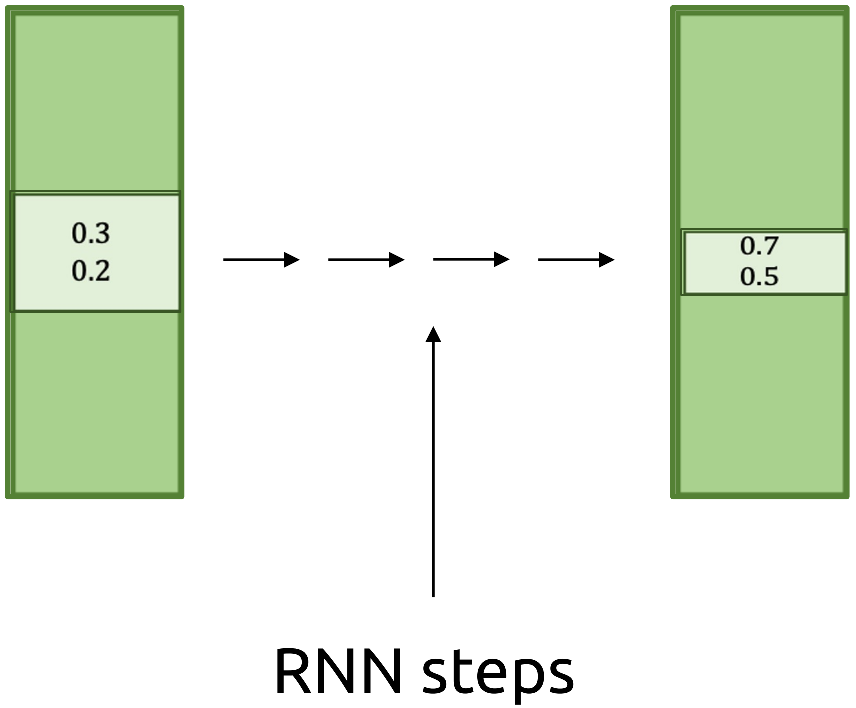
How does this affect the hidden state?

Dog gets lost in all the other information!



How does this affect the hidden state?

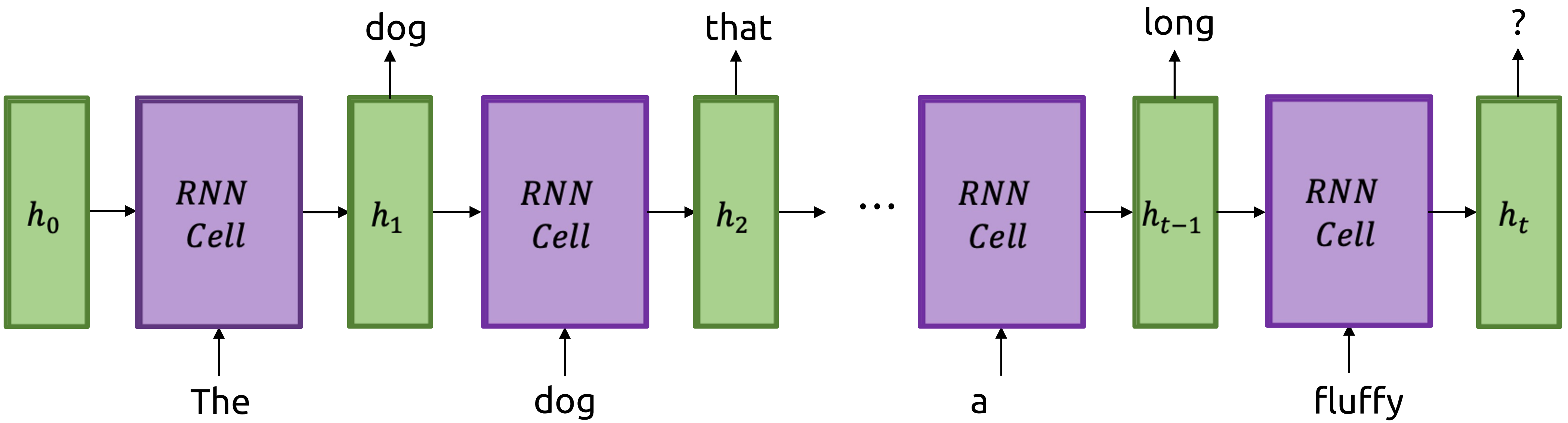
- “dog” in hidden state gets combined and mixed with rest of hidden state



RNN forgets about the dog after a certain time ☹

Any questions?

RNNs cannot learn "long term" dependency

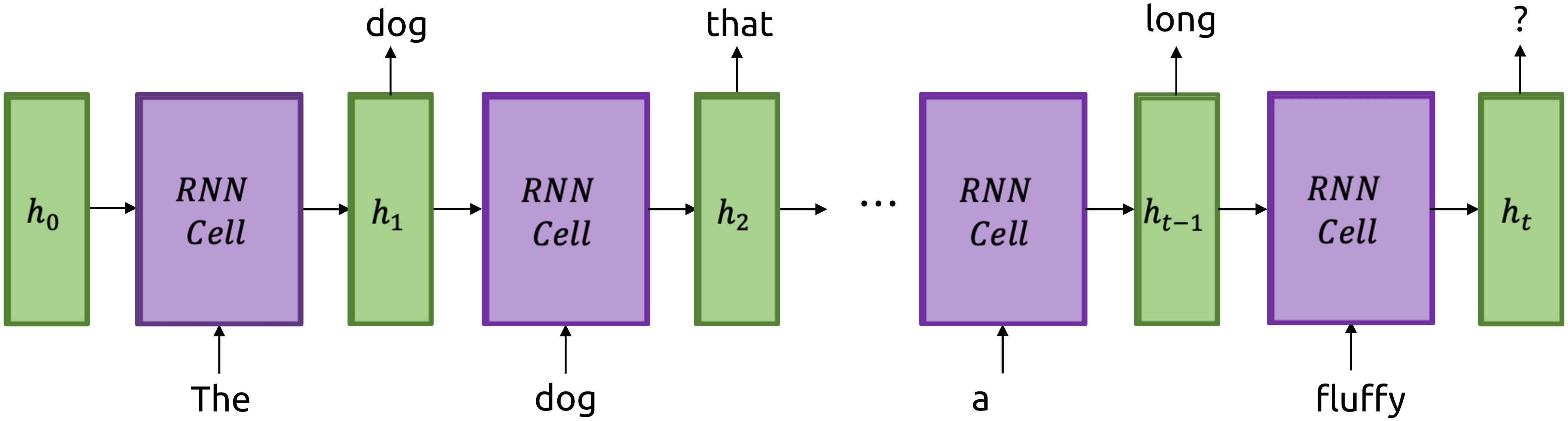


We need new way to update hidden state!

How?

An analogy to human (or computer) memory:

- RNN hidden state → "short term memory/RAM"
  - Like how you lose contents of RAM if you shut down a computer...
  - ...or how human short-term memory fades after time

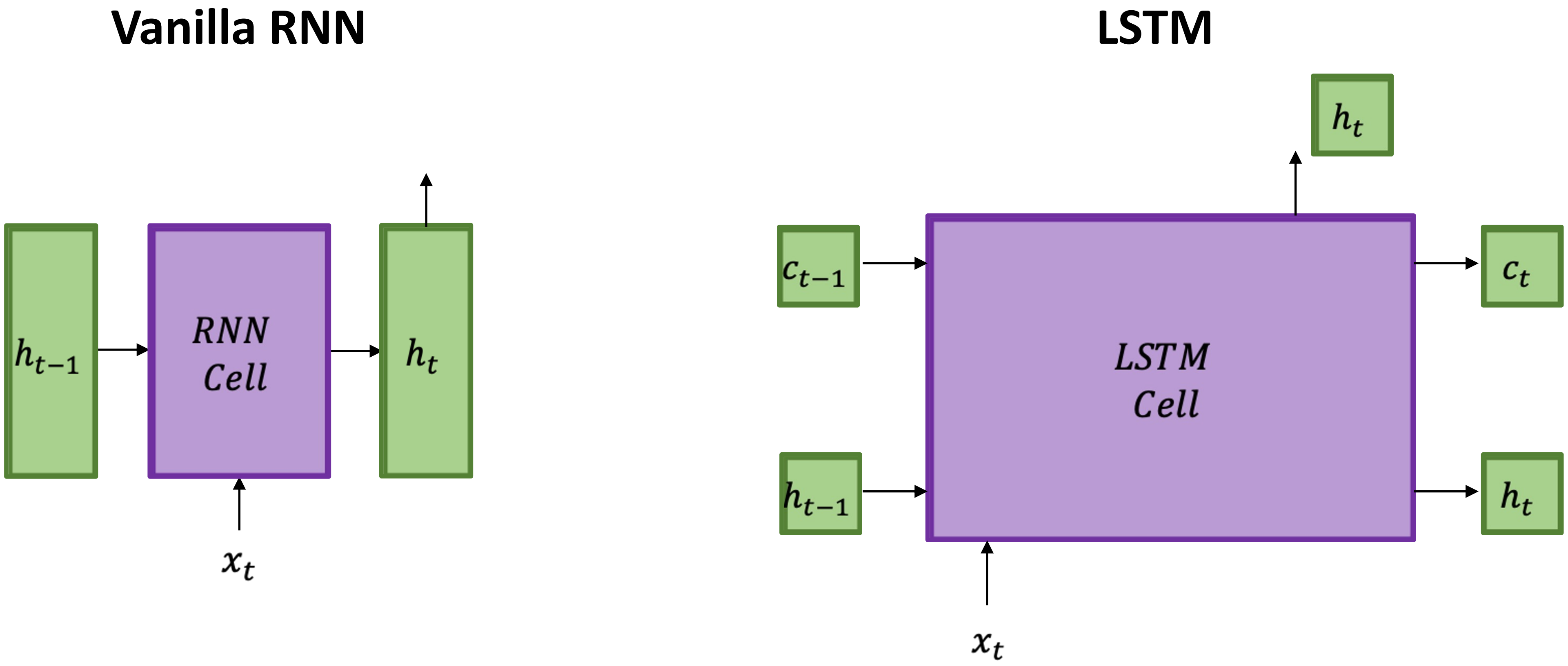


An analogy to human (or computer) memory:

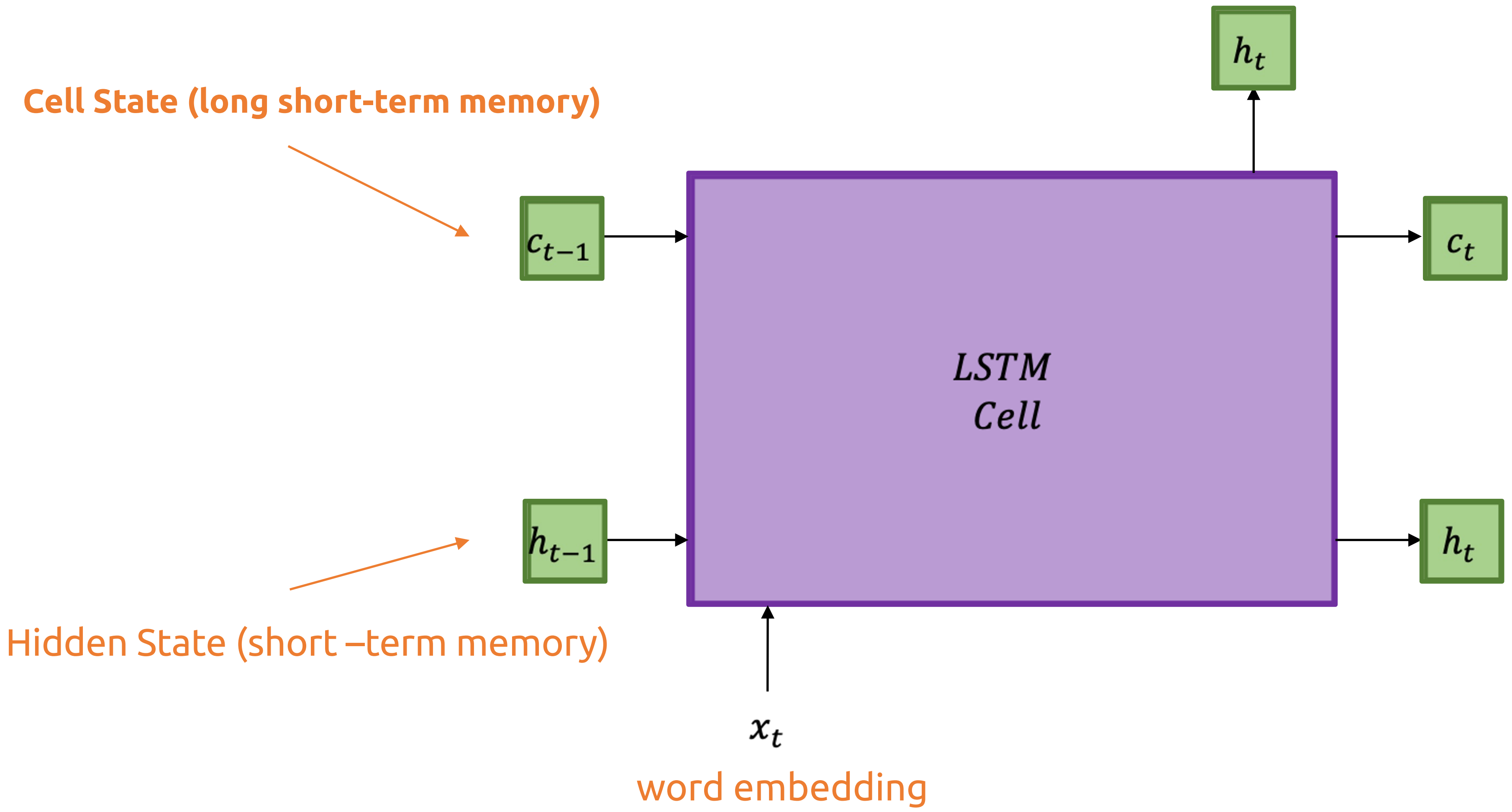
- RNN hidden state → "short term memory/RAM"
  - Like how you lose contents of RAM if you shut down a computer...
  - ...or how human short-term memory fades after time
- What we want → "long term memory/disk"
  - Some state representing knowledge that persists
  - Like how contents of disk persist across shut-downs...
  - ...or how sleep consolidates human memory into long-term memory

- ***<u>Long</u>*** Short Term Memory (LSTM)
  - "Short-term memory that persists over time"
  - i.e. "hidden states that remember information for longer"

What is different?



LSTM

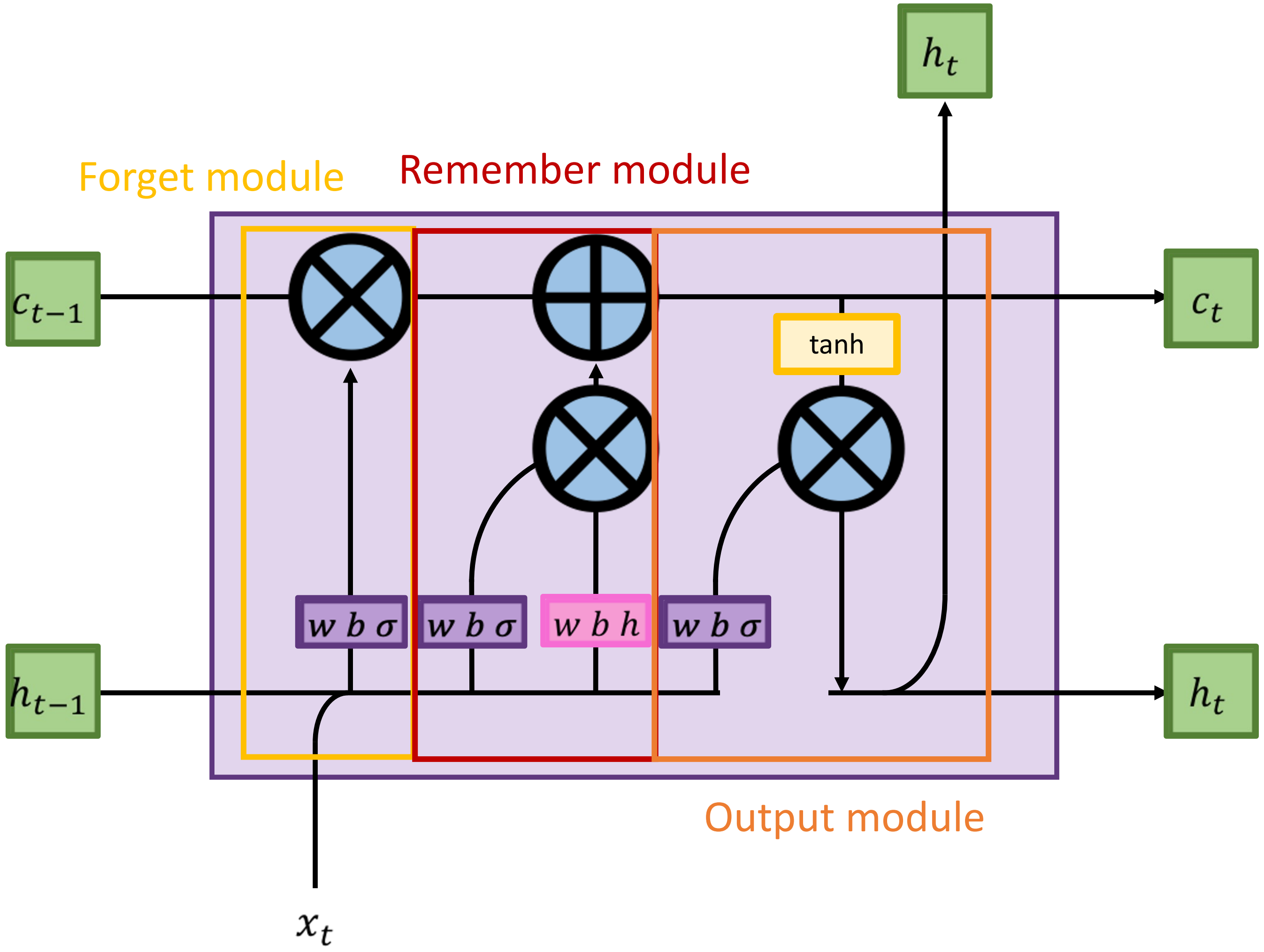


Cell State (long short-term memory)
c_{t-1}
h_{t-1}
Hidden State (short –term memory)
$LSTM$
$Cell$
h_t
c_t
h_t
x_t
word embedding

How an LSTM works

- An LSTM consists of 3 major modules:
  - Forget module
  - Remember module
  - Output module

The Complete LSTM



Forget Module

Say we just predicted *"tail"* in *"My dog has a fluffy _____."*

Next set of words: *"I love my dog"*

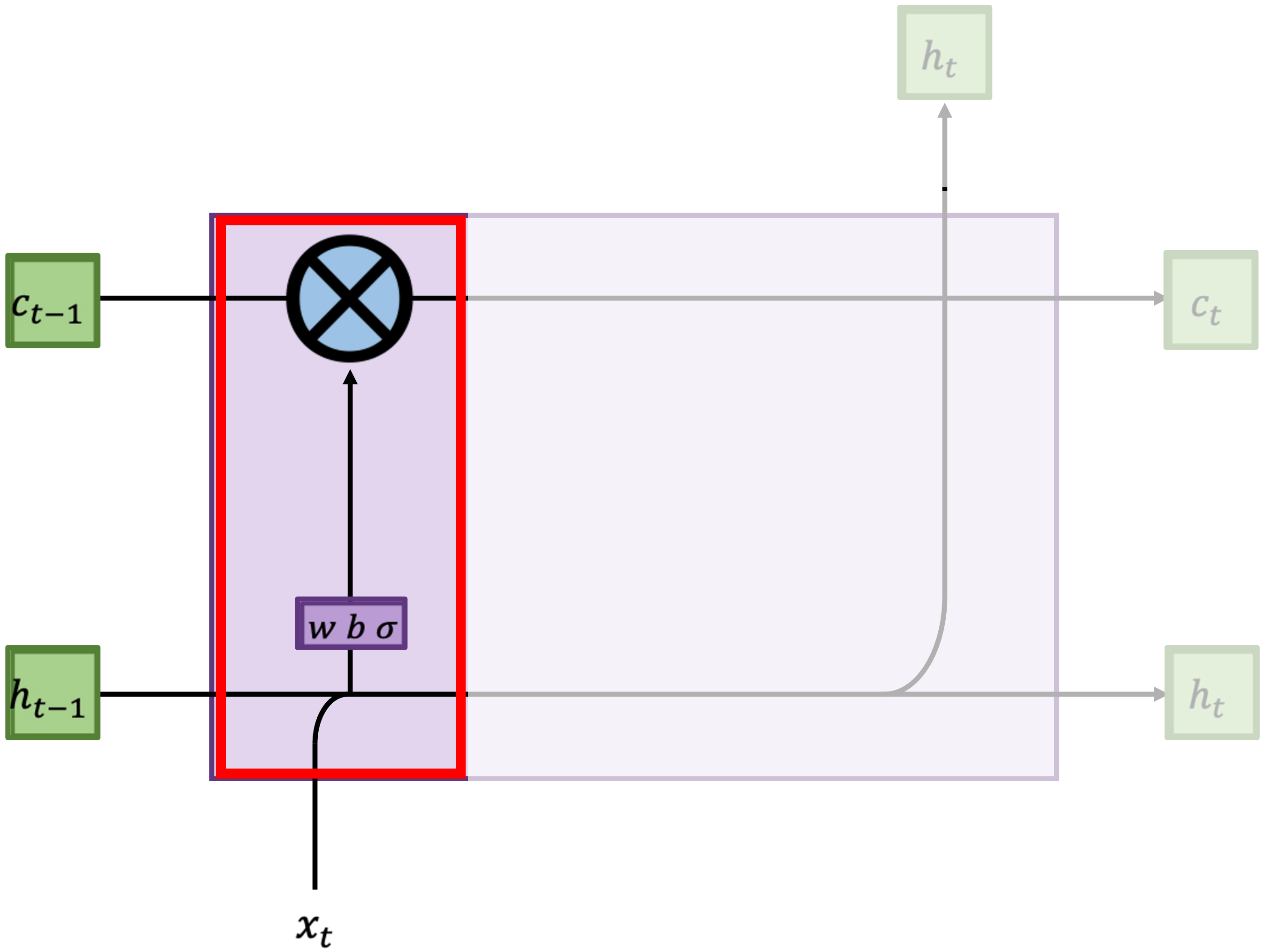
Forget Module

- Model no longer needs to know about ***"dog"***
- Ready to **delete** information about subject

Forget Module

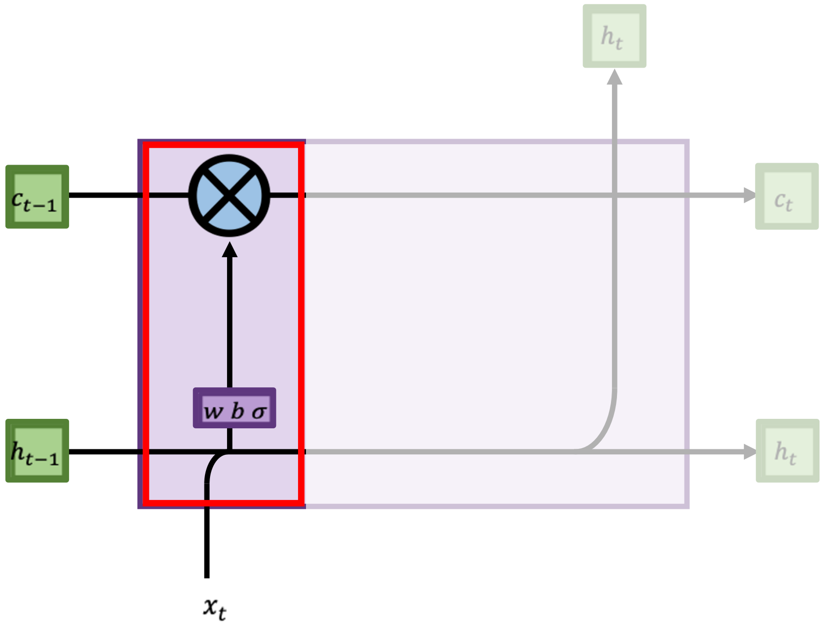
$w\ b\ \sigma$ = fully connected layer with sigmoid

$\otimes$ = pointwise multiplication



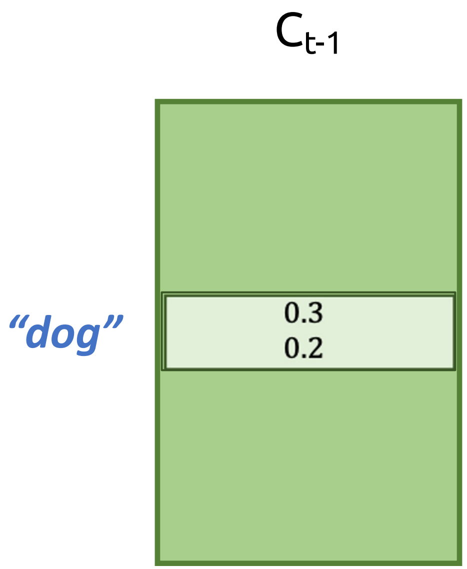
Forget Module

- Filters out what gets allowed into the LSTM cell from the last state
  - Example: If it's remembering gender pronouns, and a new subject is seen, it will forget the old gender pronouns
- Either lets parts of C_{t-1} pass through or not



Forgetting information

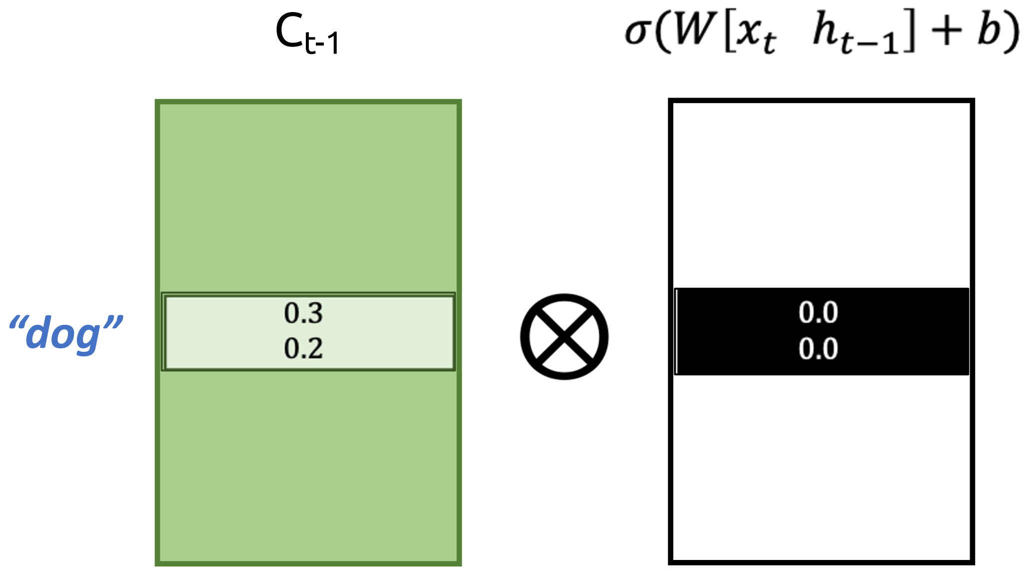
- Use pointwise multiplication by a **mask vector** to forget information
  - What do we want to forget from last cell state?



My dog has a fluffy tail. ***I love my dog***

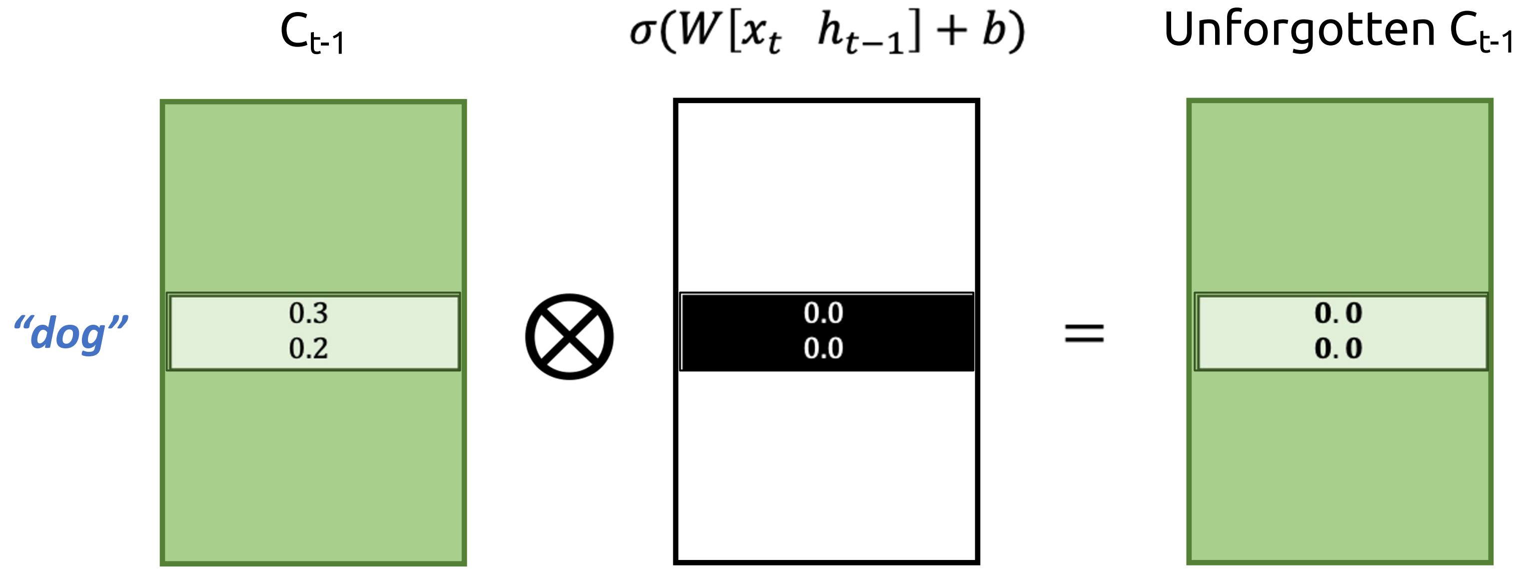
Forgetting information

- Use pointwise multiplication by a **mask vector** to forget information
  - What do we want to forget from last cell state?
  - Output of fully connected + sigmoid is what we want to forget



Forgetting information

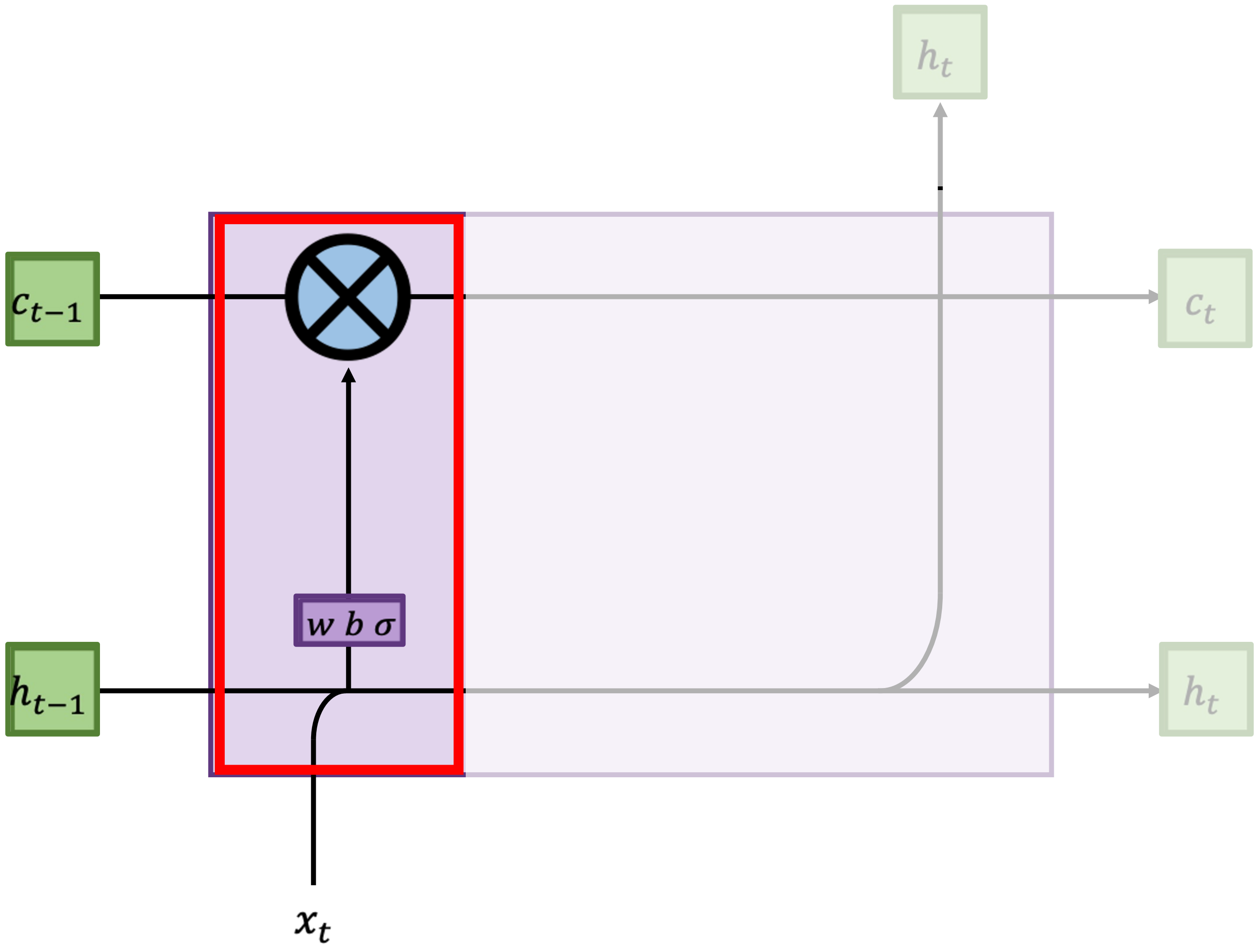
- Use pointwise multiplication by a **mask vector** to forget information
  - What do we want to forget from last cell state?
  - Output of fully connected + sigmoid is what we want to forget
  - “Zeros out” a part of the cell state
  - Pointwise multiplication by a learned mask vector is known as ***gating***



Forget Module

 = fully connected layer with sigmoid

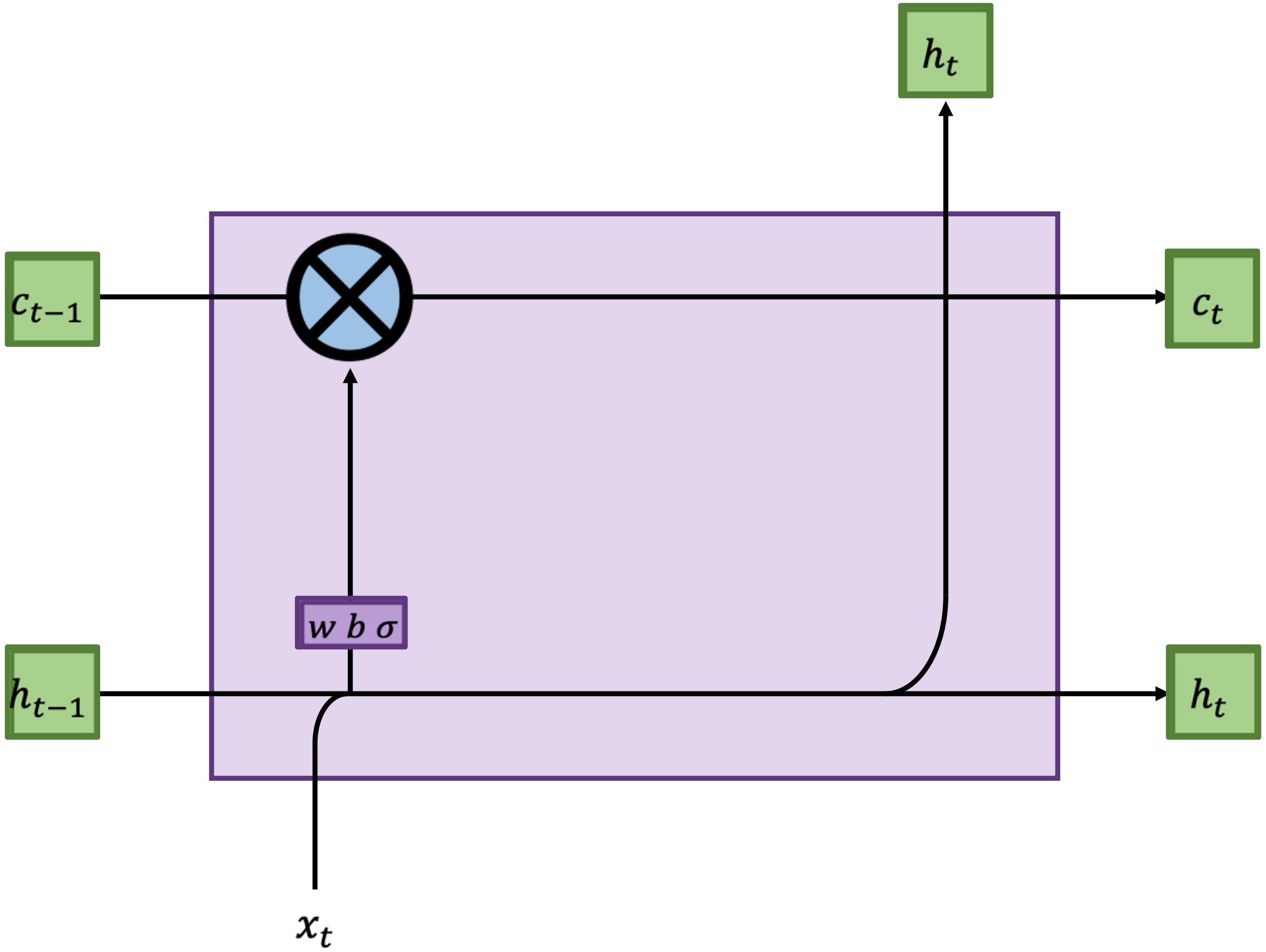
 = pointwise multiplication



What's next?

= fully connected layer with sigmoid


= pointwise multiplication



$w\ b\ \sigma$ = fully connected layer with sigmoid

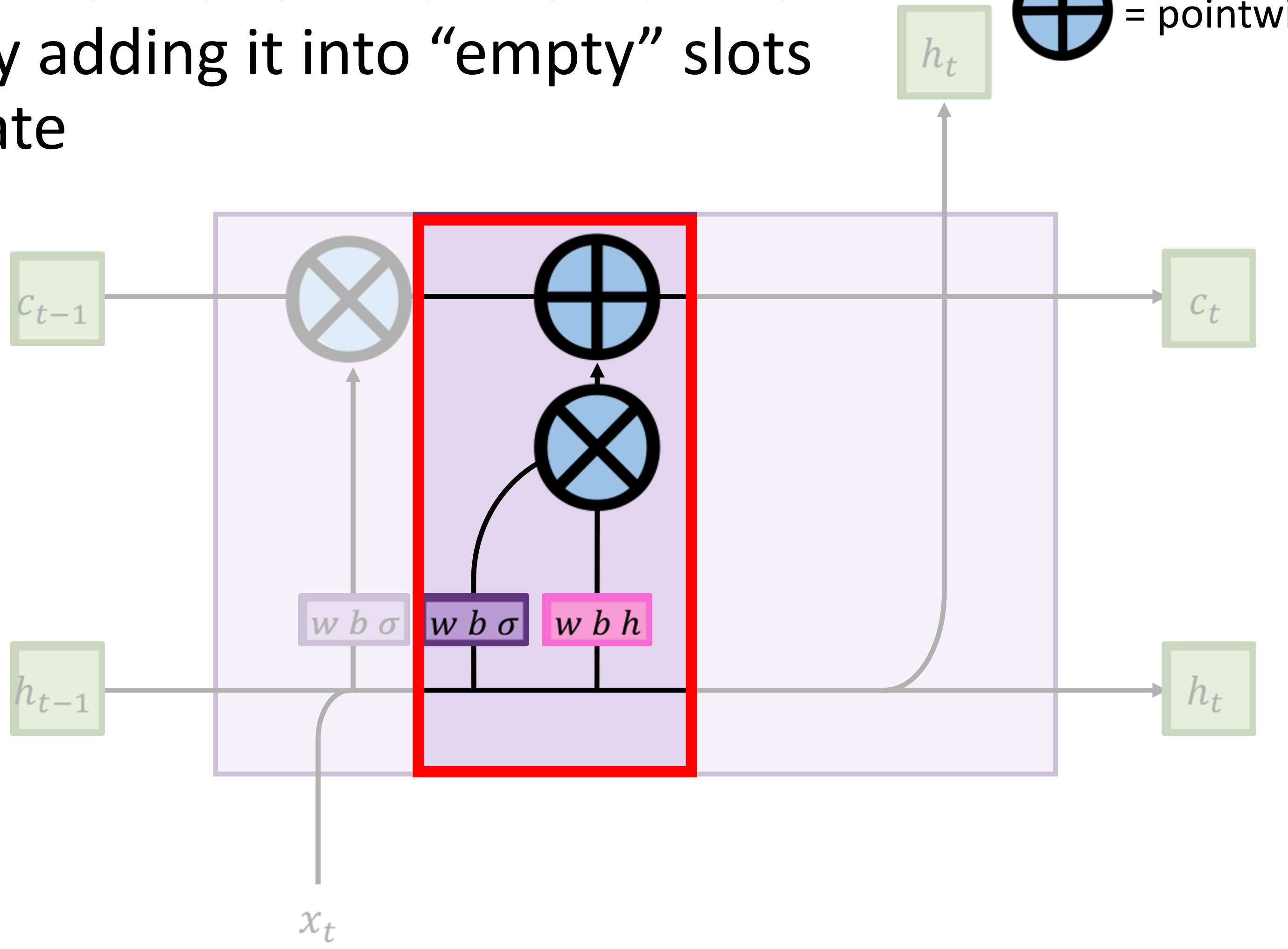
$w\ b\ h$ = fully connected layer with tanh

$\otimes$ = pointwise multiplication

$\oplus$ = pointwise addition

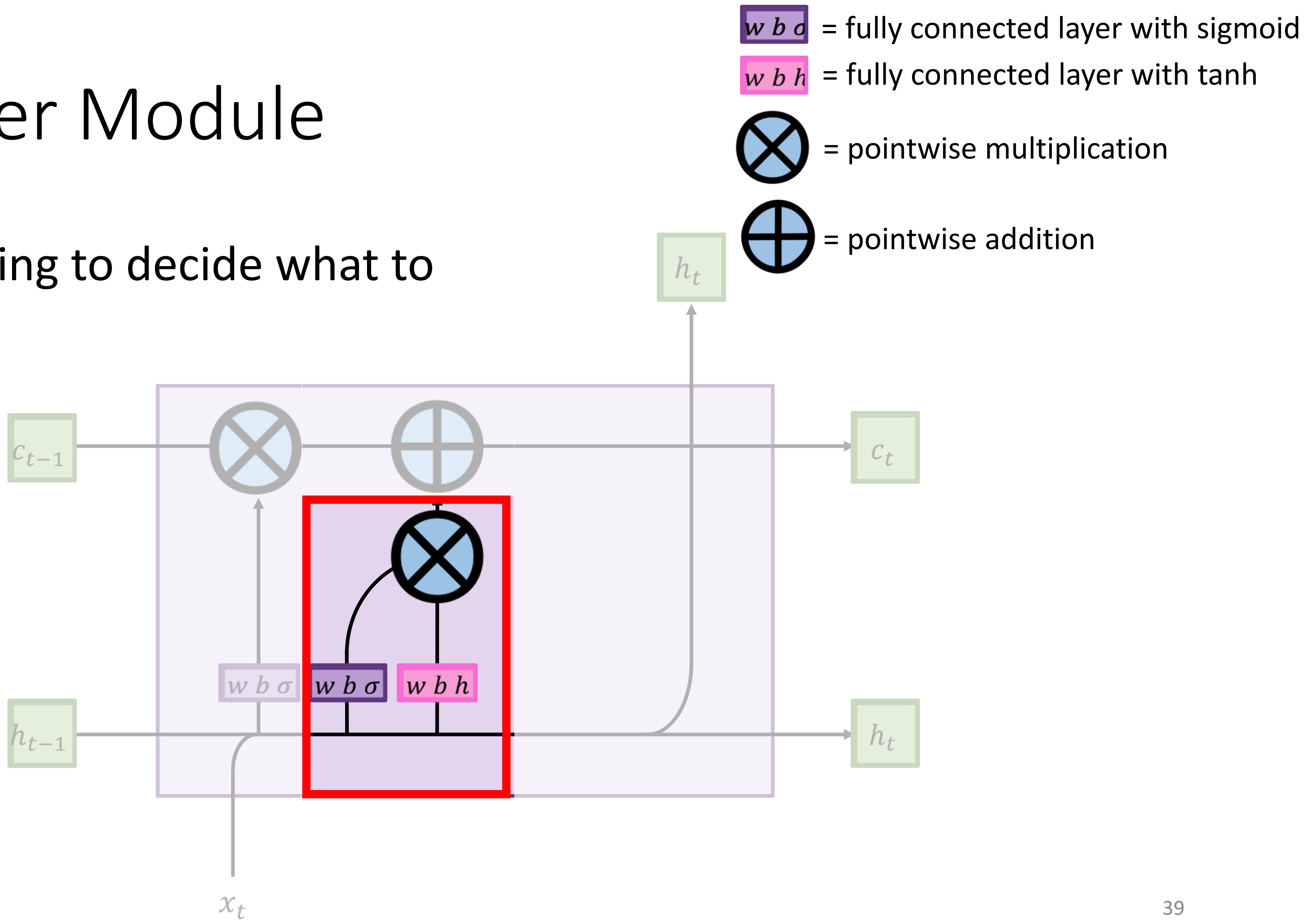
Remember Module

- We can save information that we want to remember by adding it into "empty" slots in the cell state



Remember Module

- First: use gating to decide what to remember

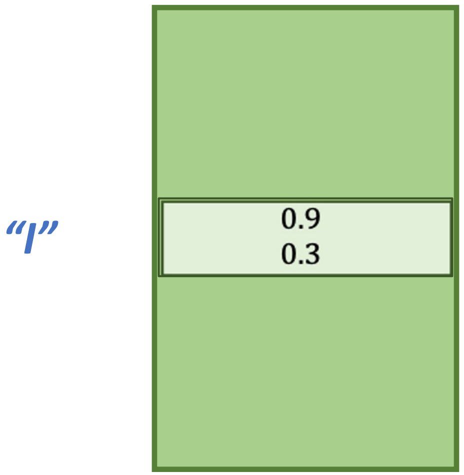


My dog has a fluffy tail. ***I love my dog***

Gating for ‘selective memory’

- A fully-connected + tanh on [input, memory] computes some new memory

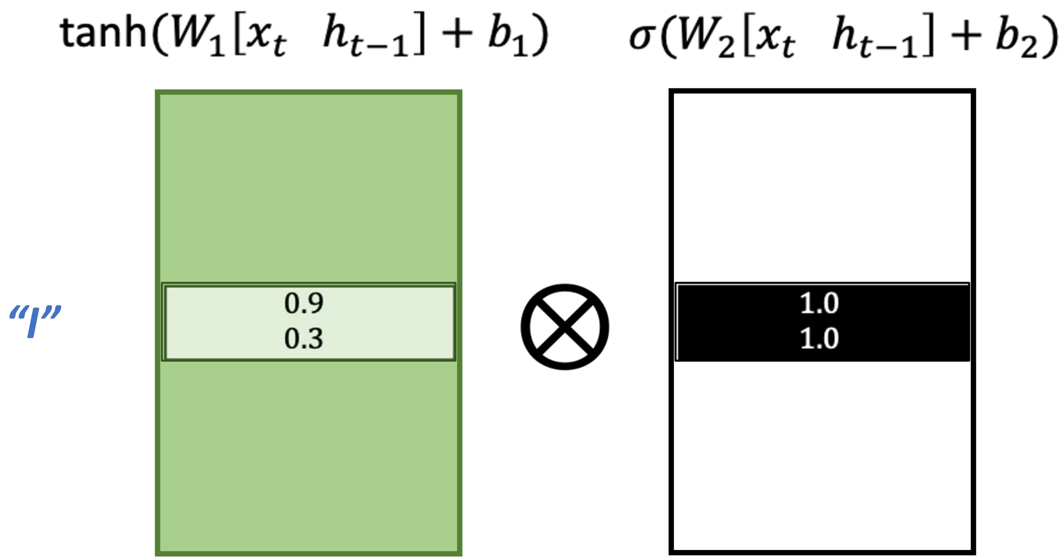
$$\tanh(W_1[x_t \quad h_{t-1}] + b_1)$$



My dog has a fluffy tail. ***I love my dog***

Gating for 'selective memory'

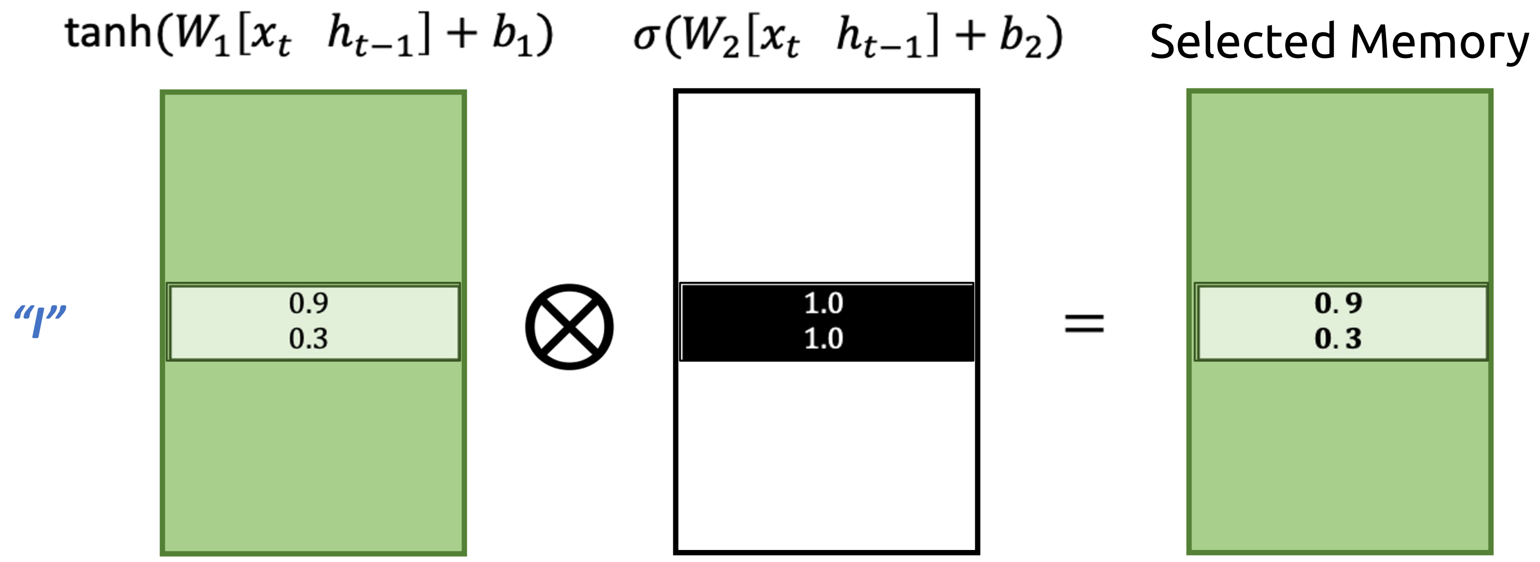
- A fully-connected + tanh on [input, memory] computes some new memory
- We gate this memory to decide what bits of it we want to remember long-term in the cell state



My dog has a fluffy tail. ***I love my dog***

Gating for 'selective memory'

- A fully-connected + tanh on [input, memory] computes some new memory
- We gate this memory to decide what bits of it we want to remember long-term in the cell state



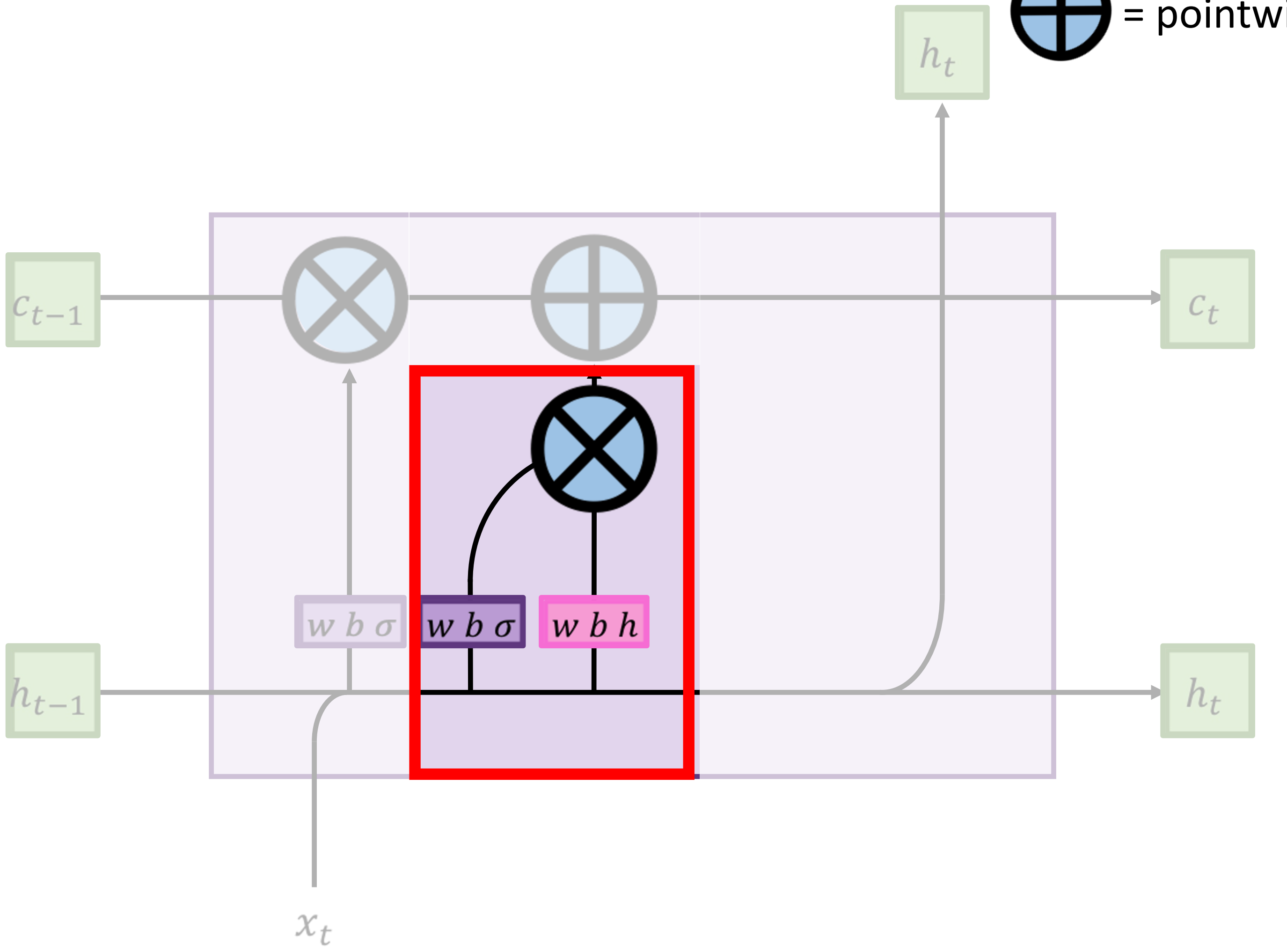
Remember Module

$w\ b\ \sigma$ = fully connected layer with sigmoid

$w\ b\ h$ = fully connected layer with tanh

⊗ = pointwise multiplication

⊕ = pointwise addition



Remember Module

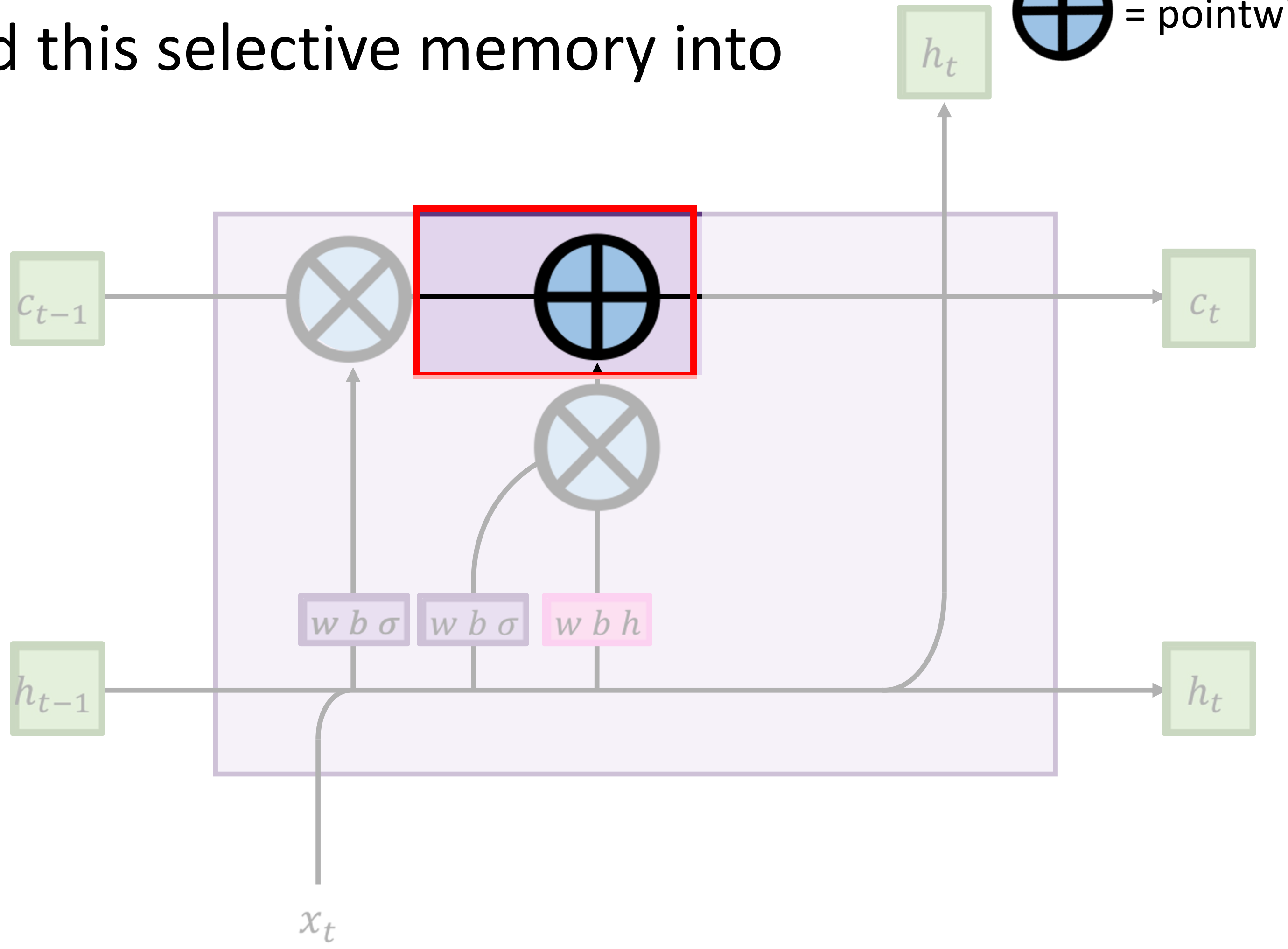
- Then: we add this selective memory into the cell state

$w\ b\ \sigma$ = fully connected layer with sigmoid

$w\ b\ h$ = fully connected layer with tanh

$\otimes$ = pointwise multiplication

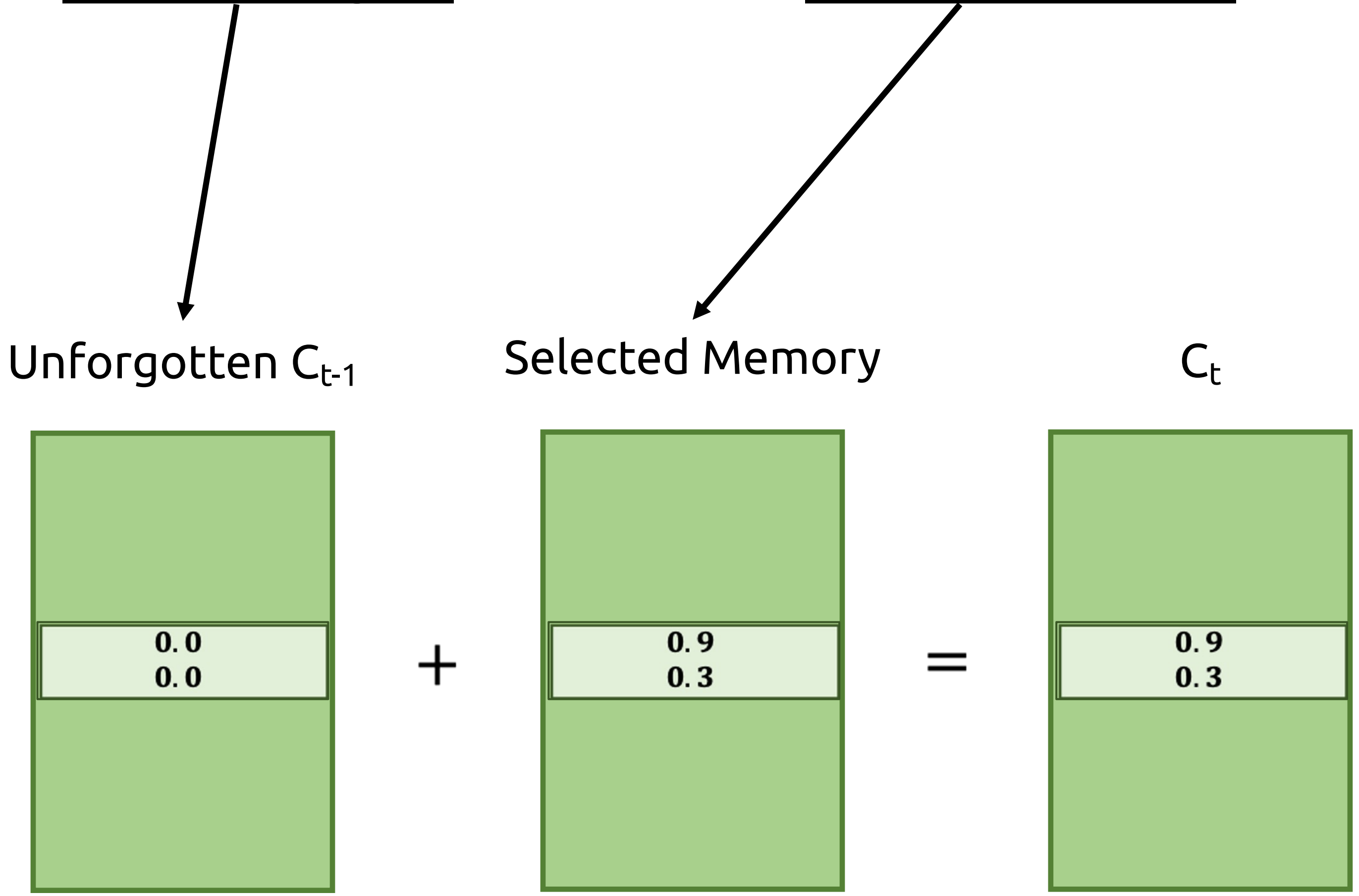
$\oplus$ = pointwise addition



My dog has a fluffy tail. ***I love my dog***

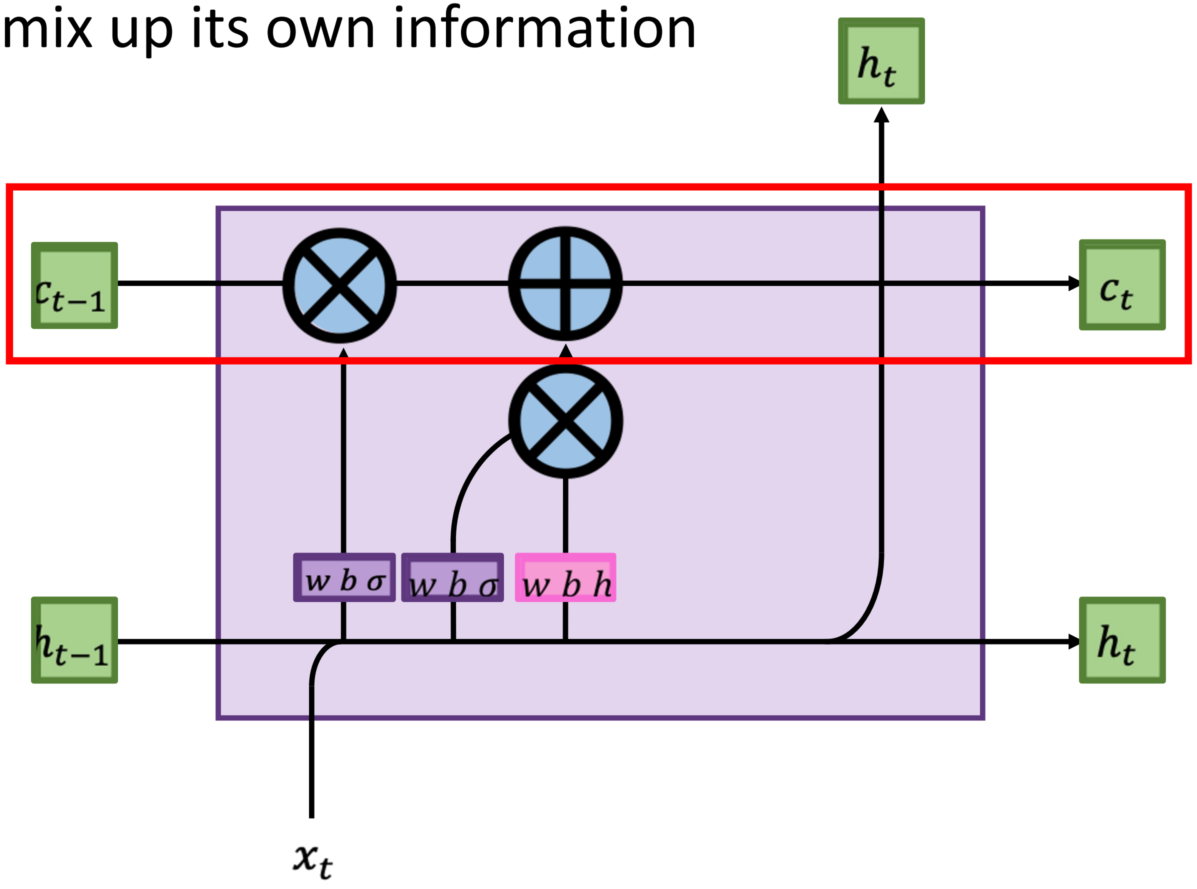
Remembering information

- Add what we didn't forget to what we did remember



Why does this solve our problem?

- Cell state never goes through a fully connected layer!
  - Never has to mix up its own information



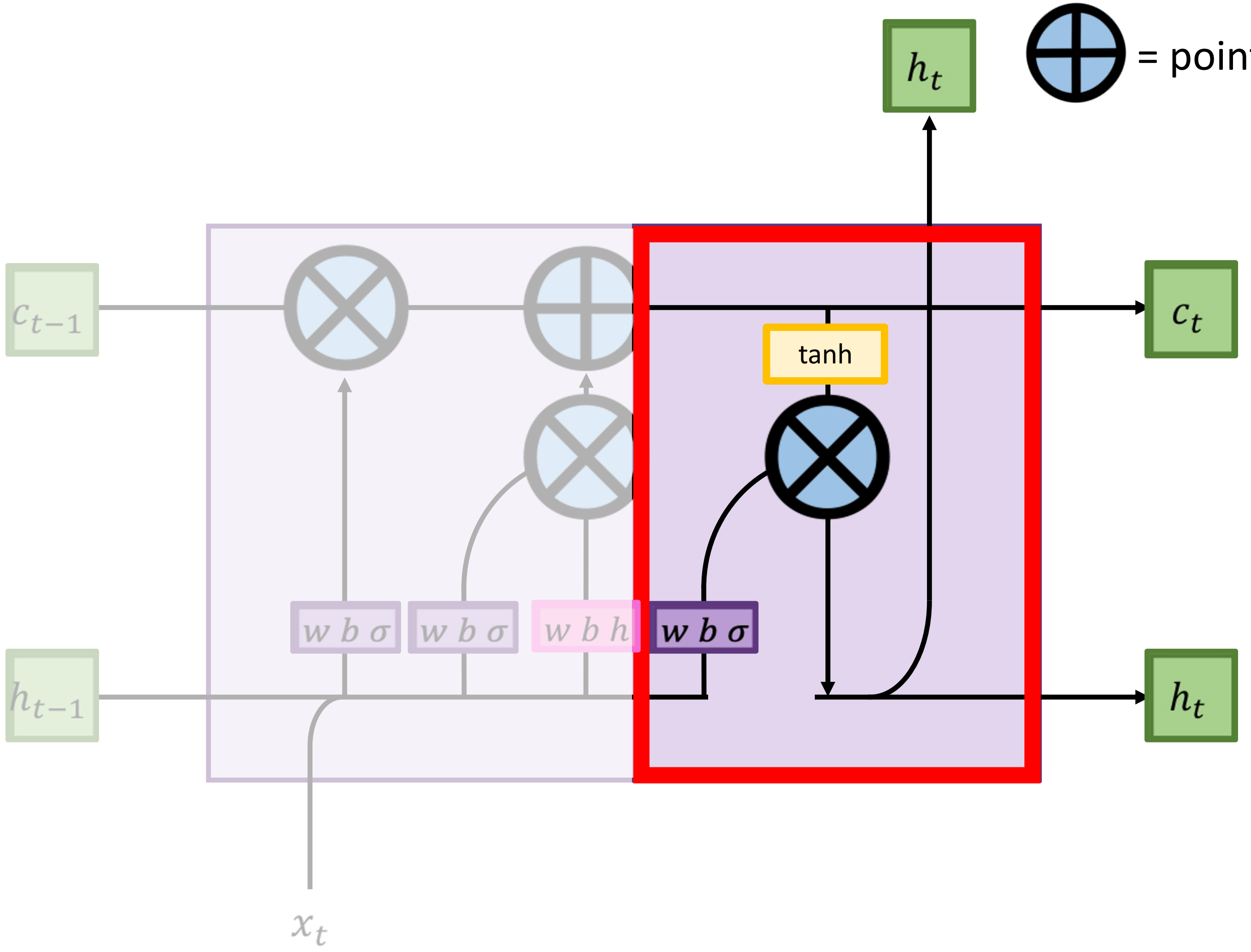
Output Module

$w\ b\ \sigma$ = fully connected layer with sigmoid

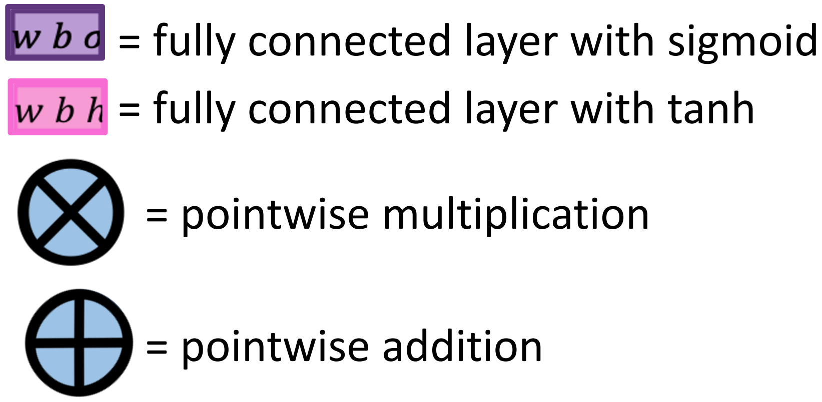
$w\ b\ h$ = fully connected layer with tanh

⊗ = pointwise multiplication

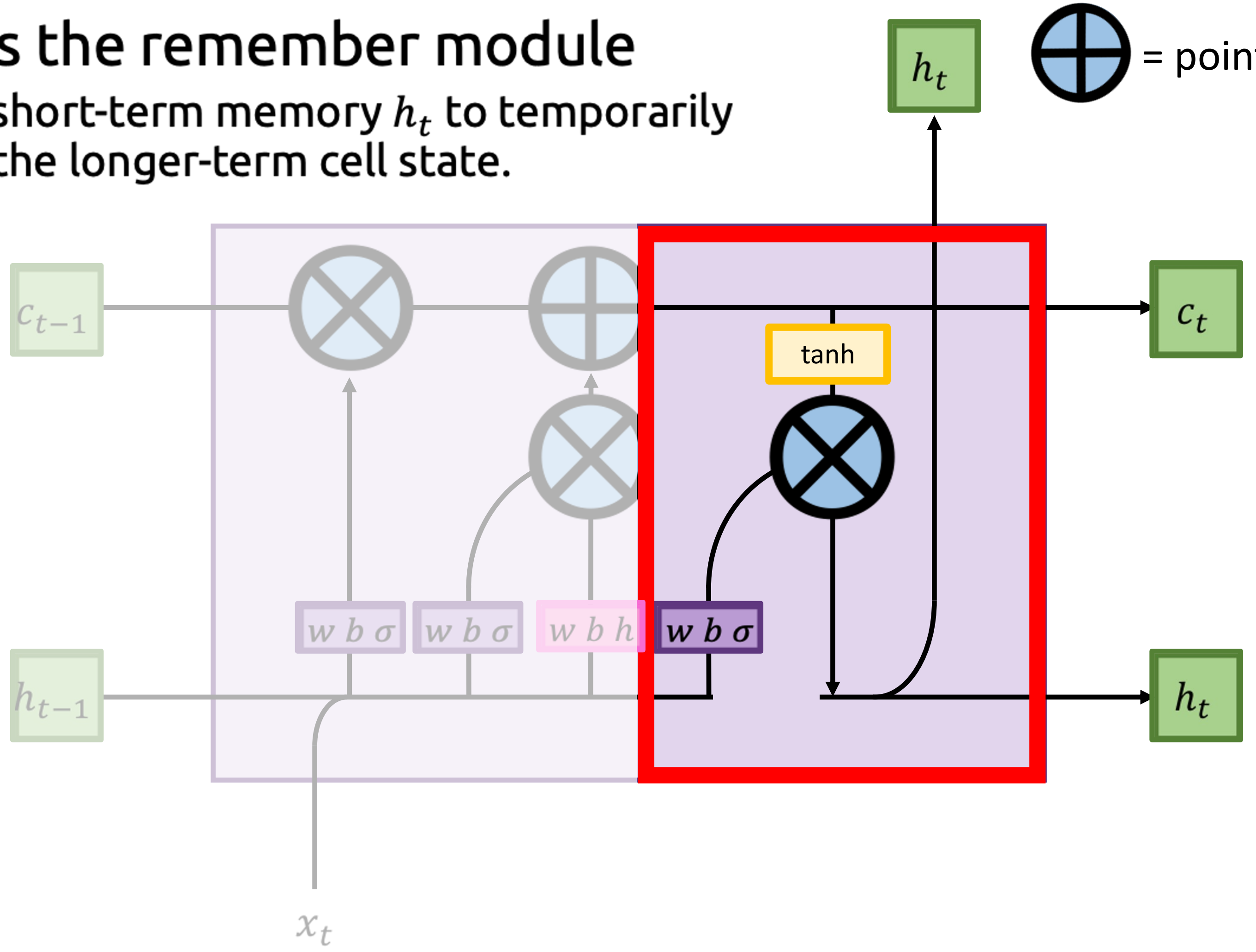
⊕ = pointwise addition



Output Module



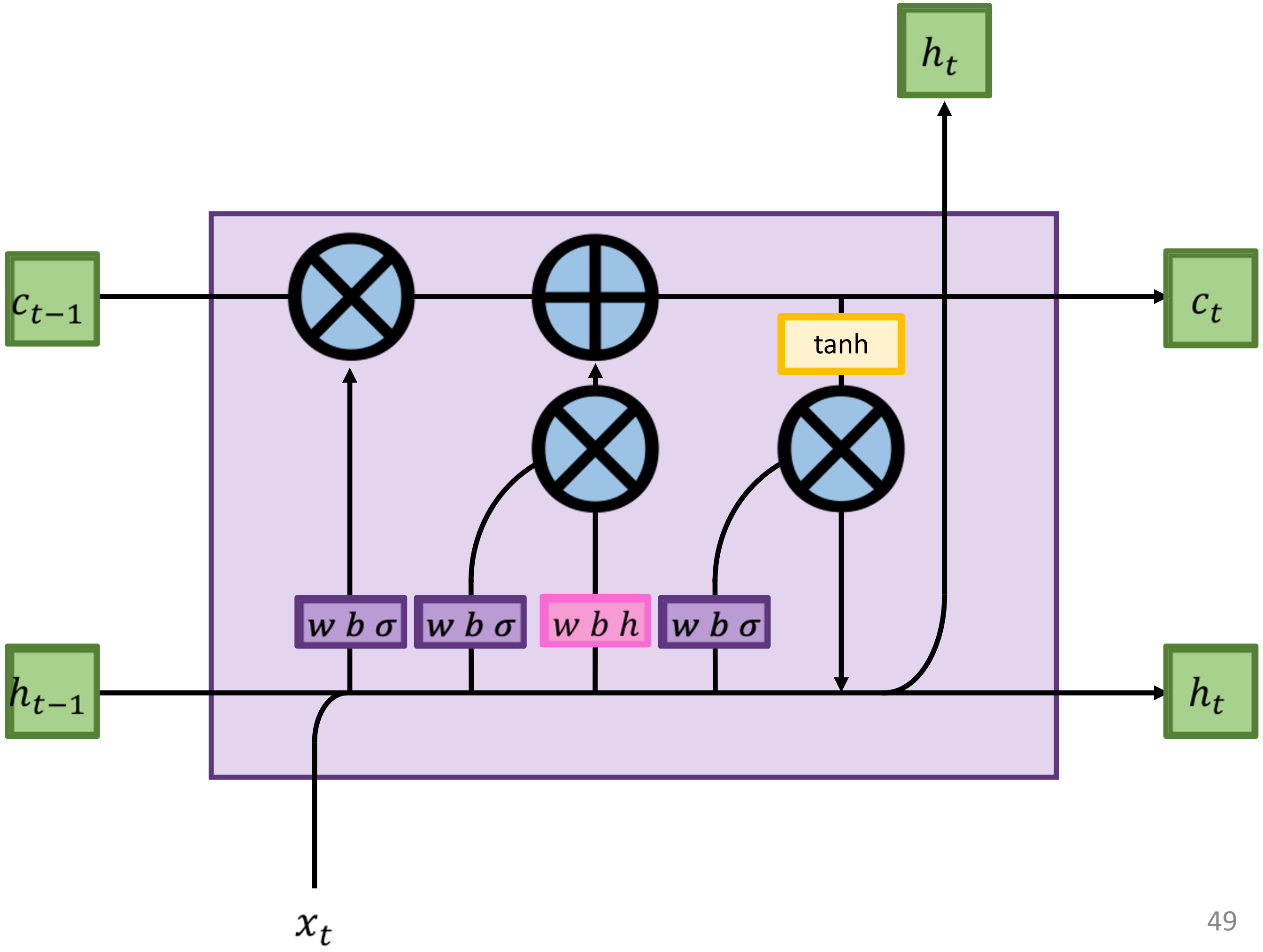
- **Same structure as the remember module**
  - Provides path for short-term memory h_t to temporarily acquire info from the longer-term cell state.



Any questions?

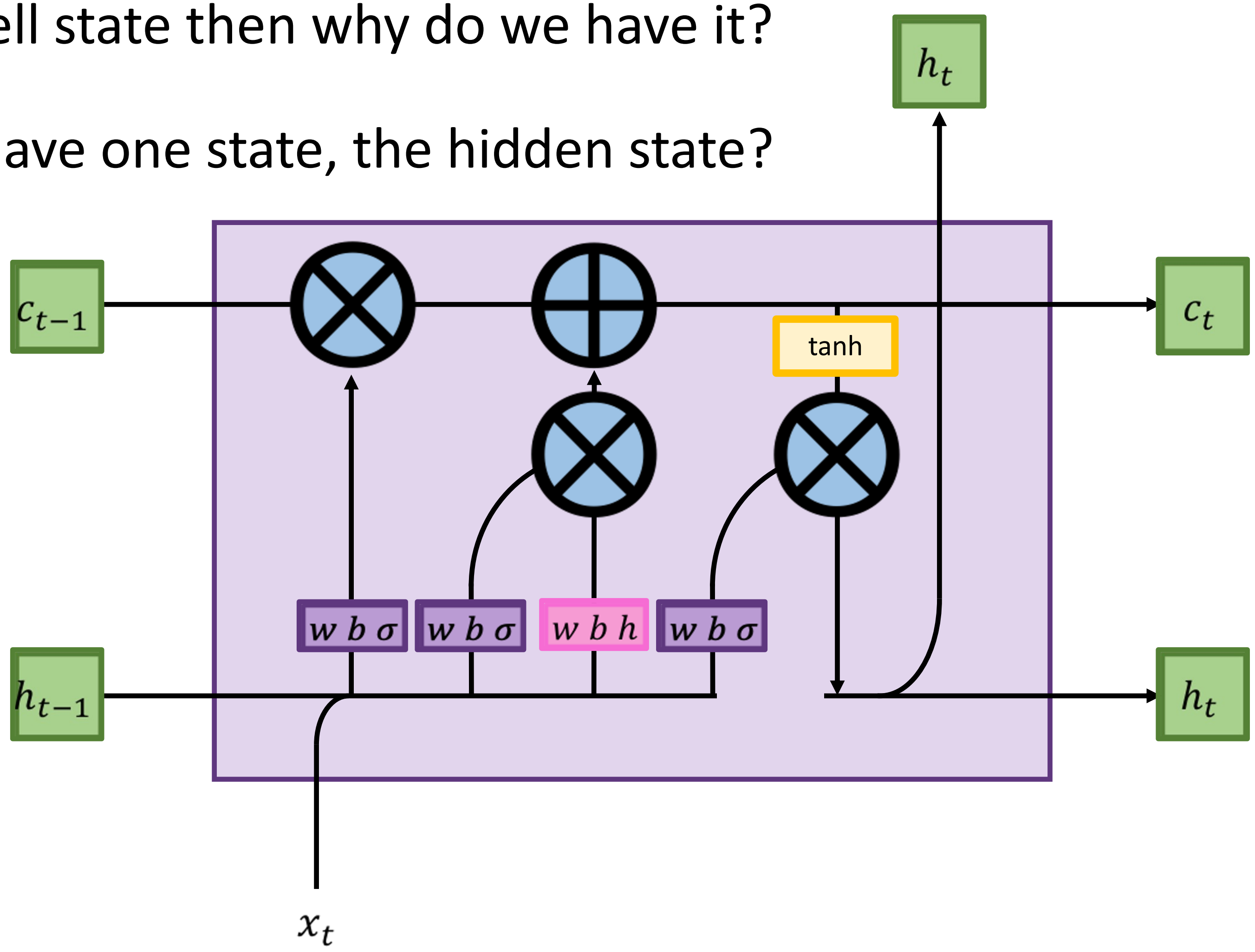
The Complete LSTM

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$\widetilde{c}_t = tanh(W h_{t-1} + U x_t + b)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$$
$$h_t = o_t \circ tanh(c_t)$$
$$y_t = h_t$$



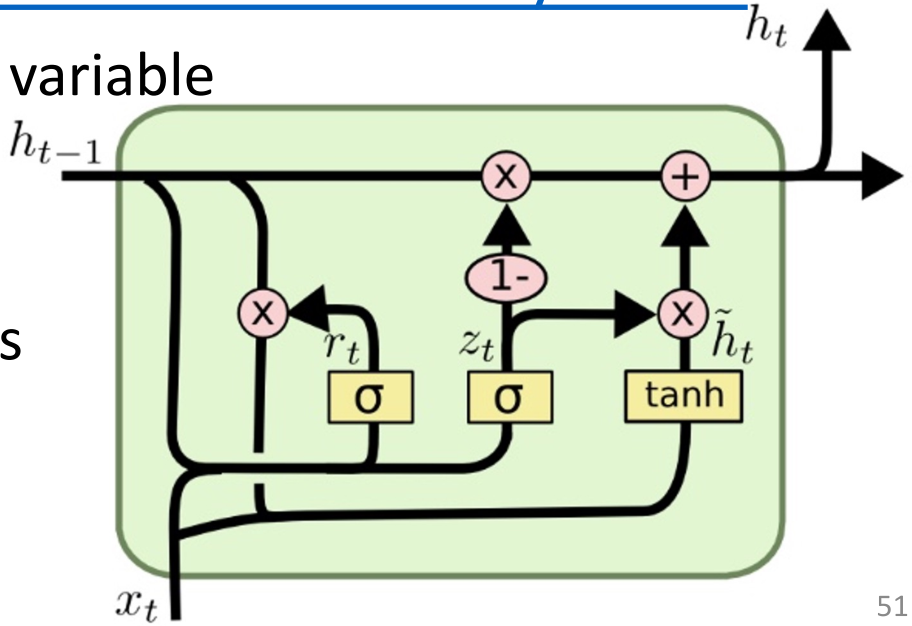
The Complete LSTM

- If we never output cell state then why do we have it?
- Is it possible to just have one state, the hidden state?



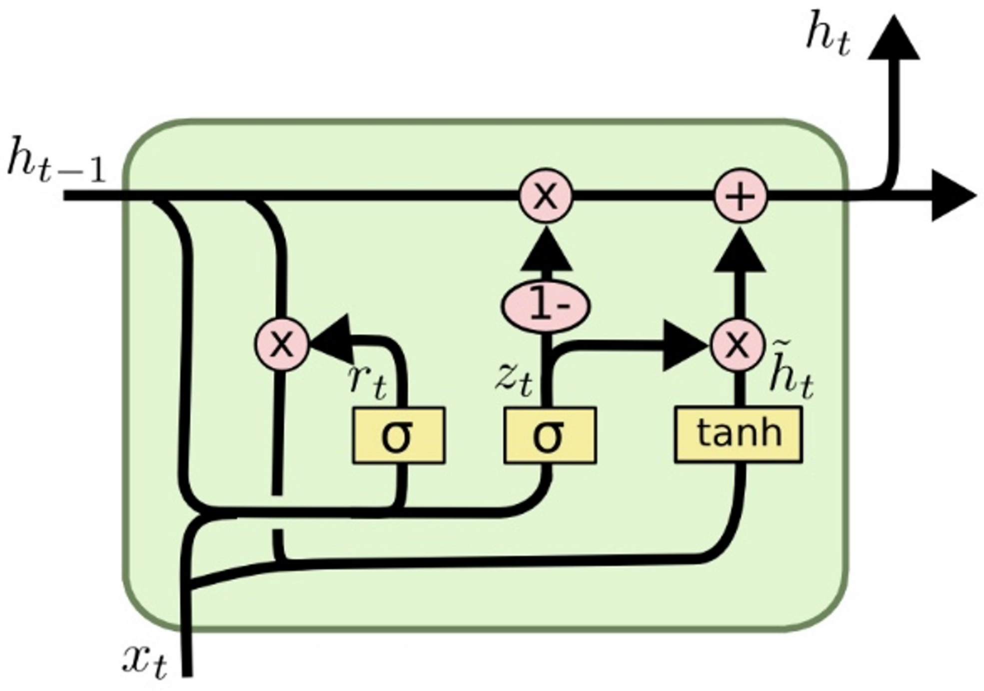
GRU

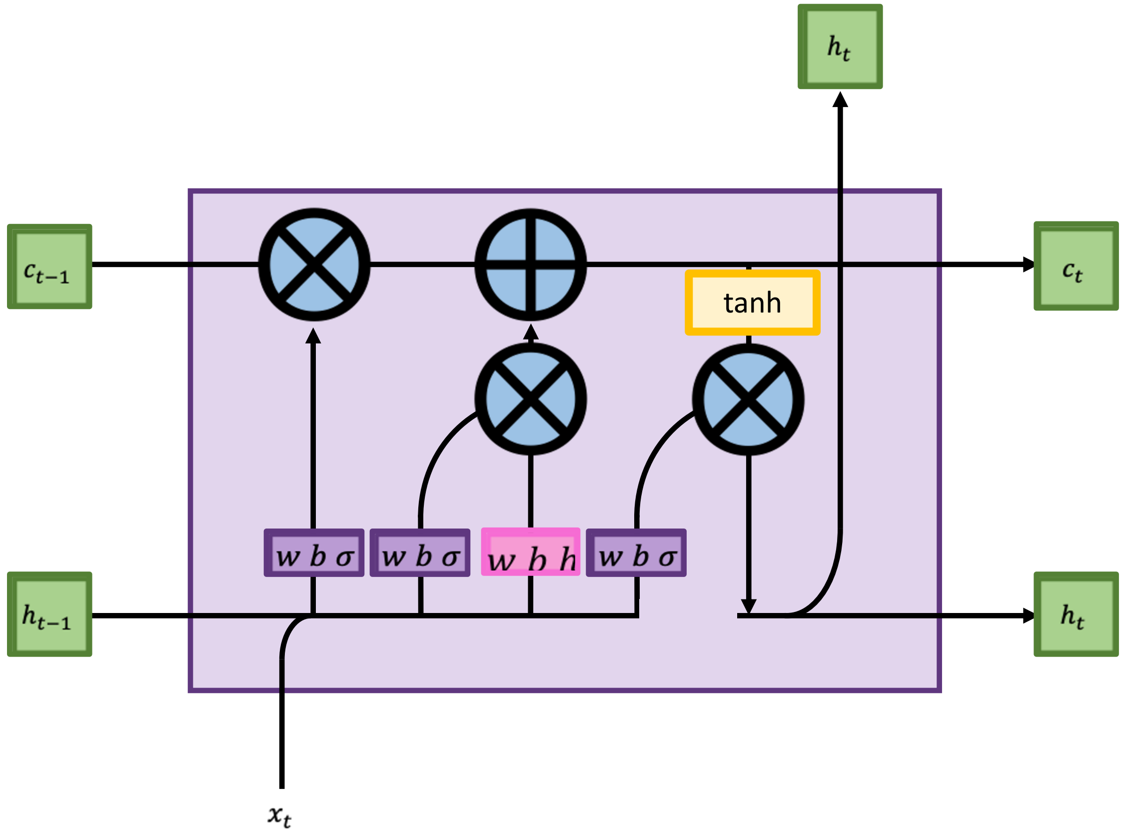
- ***G***ated ***R***ecurrent ***U***nit
- In practice, similar performance and may train faster
  - Removes cell state, computationally more efficient and less complex
- In theory, weaker than LSTMs since it [cannot unboundedly count]
  - Counting: track increment or decrement of variable
  - e.g. Validate brackets in code
    [… (… { … } …) …]
    Requires counting brackets & nesting levels



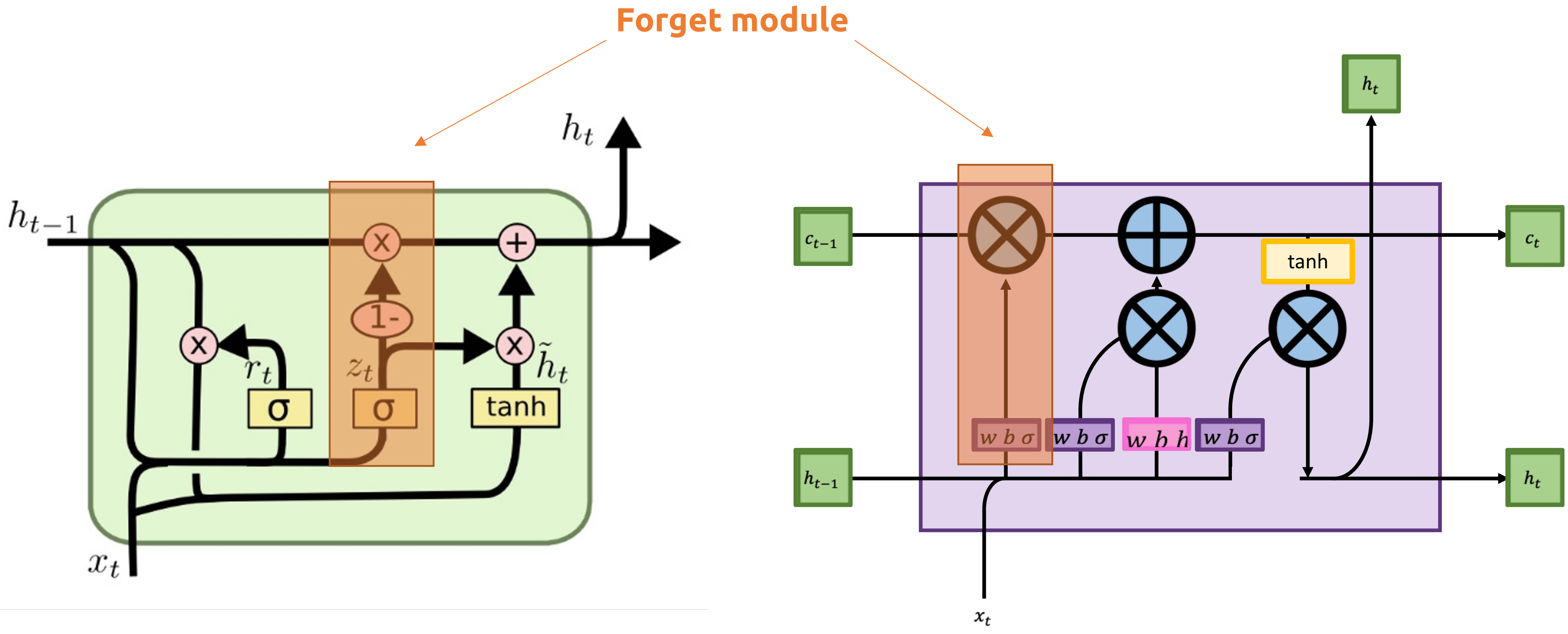
GRU vs LSTM

Can you guess which part is forget module, which one is remember module, and which one is output module in the GRU?



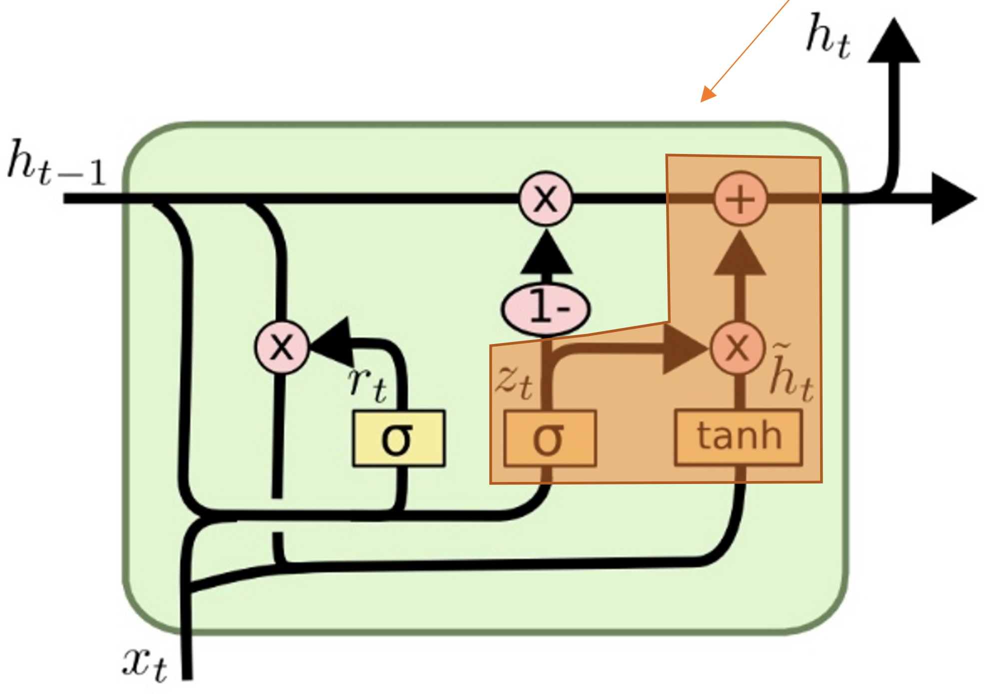


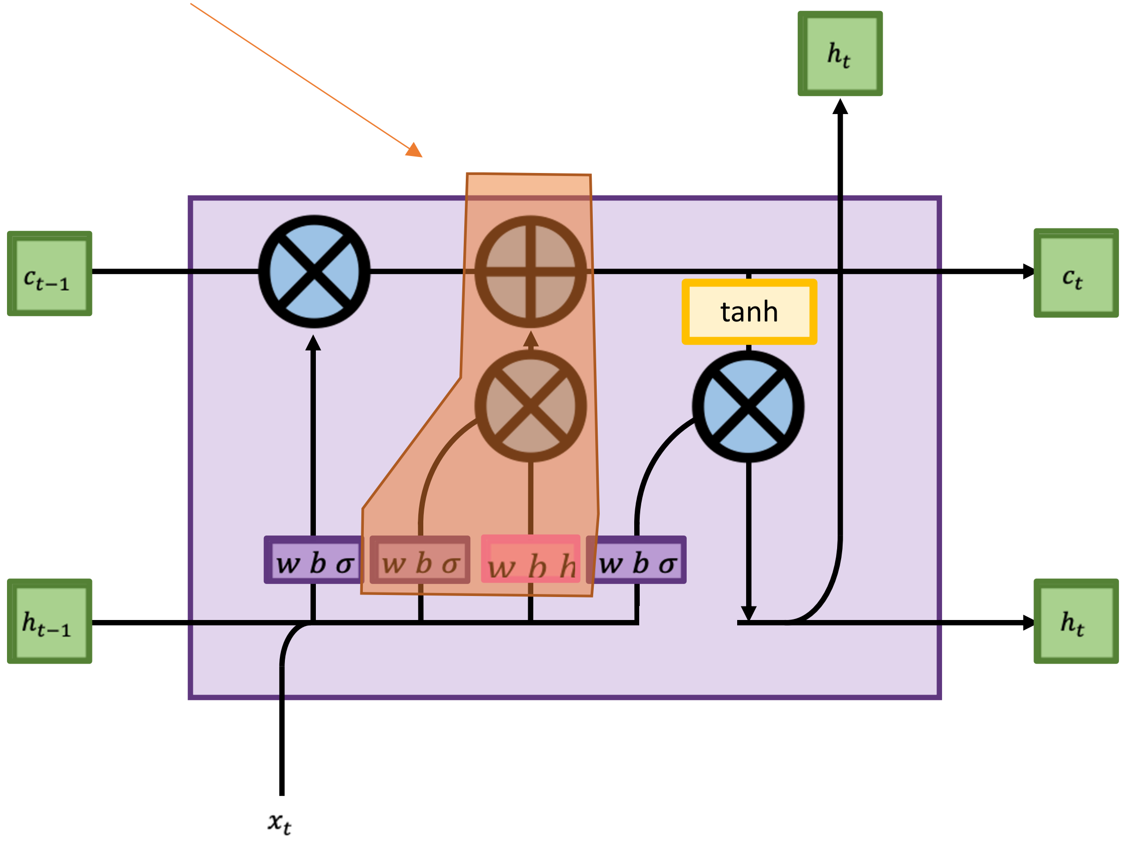
GRU vs LSTM


Forget module
h_t
h_{t-1}
1-
r_t
z_t
$\tilde{h}_t$
σ
σ
tanh
x_t
h_t
c_{t-1}
c_t
tanh
$w\ b\ \sigma$
$w\ b\ \sigma$
$w\ b\ h$
$w\ b\ \sigma$
h_{t-1}
h_t
x_t

GRU vs LSTM

Remember module

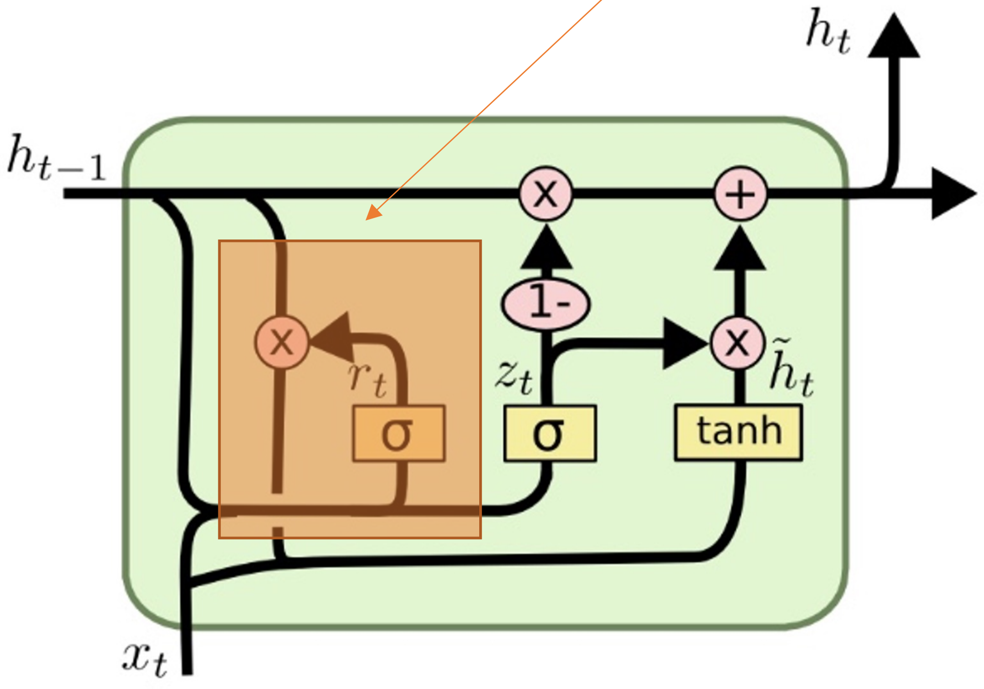


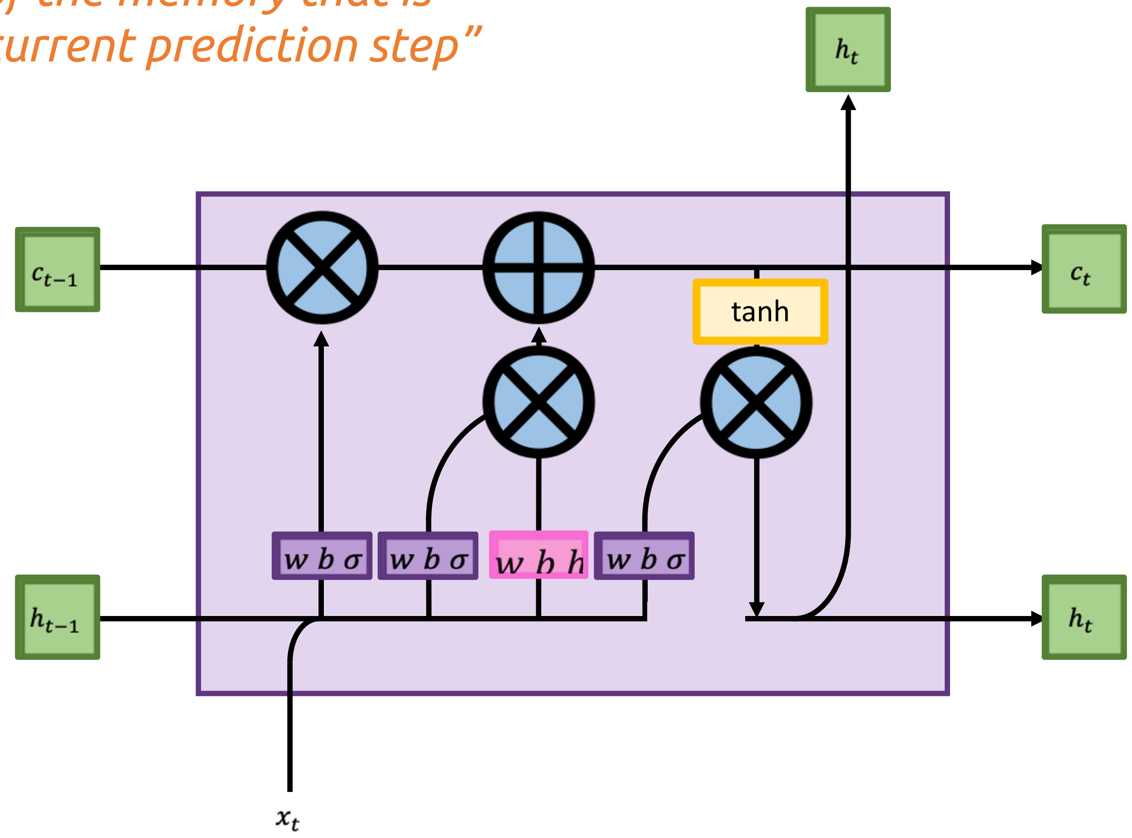


GRU vs LSTM

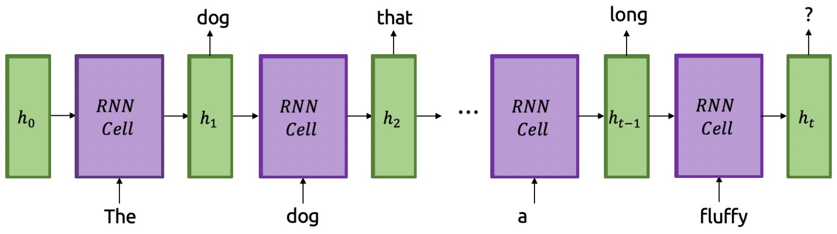
No direct analogue in LSTM

"Select the part of the memory that is relevant for the current prediction step"



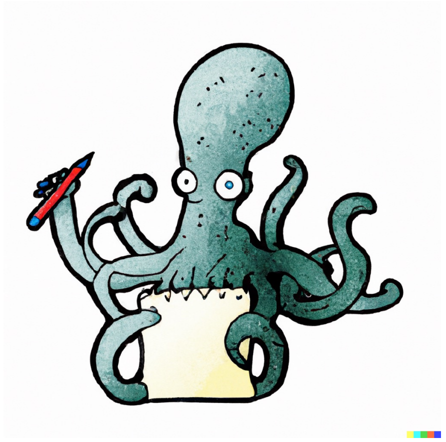


Recap



Limitations of RNNs

- RNN cell state → Hidden state
- Hidden state gets modified over time
- Cannot capture long-term dependency



LSTMS

- Cell state + Hidden state
- 3 modules
- GRU vs LSTM