Deep Learning

CSCI 1470

Eric Ewing

Monday, 3/10/25

Day 20: LSTMs and intro to seq2seq

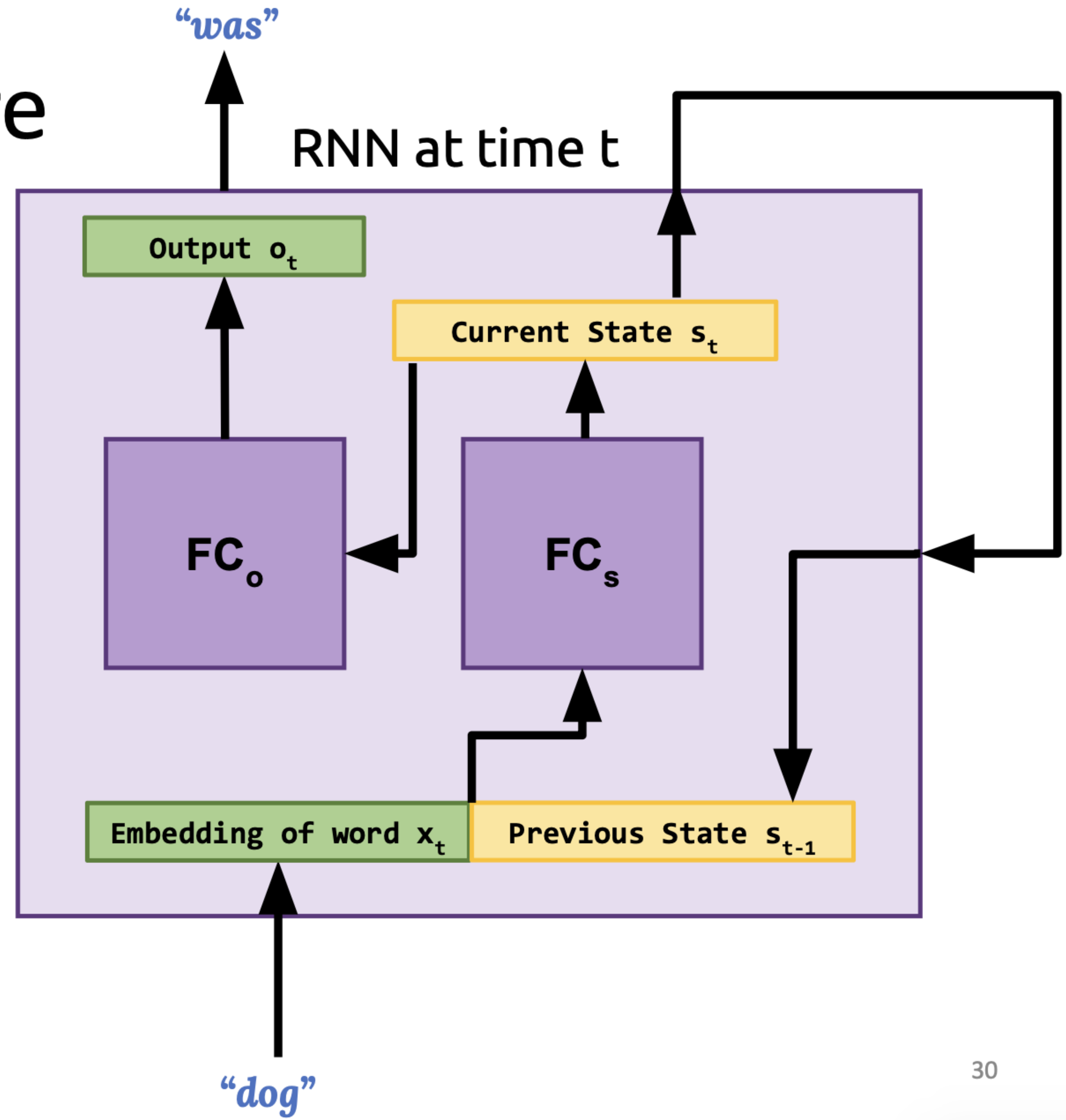
Logistics

- Keep thinking about final project groups and projects, use Edstem if you're looking for partners!
  - Max group size of 4
- ~60 people have not yet cloned the CNNs stencil
  - You should probably get started…
- Don't forget about workshops/SRC discussions
  - You have to attend 2 of each before the end of the semester

RNN Cell Architecture

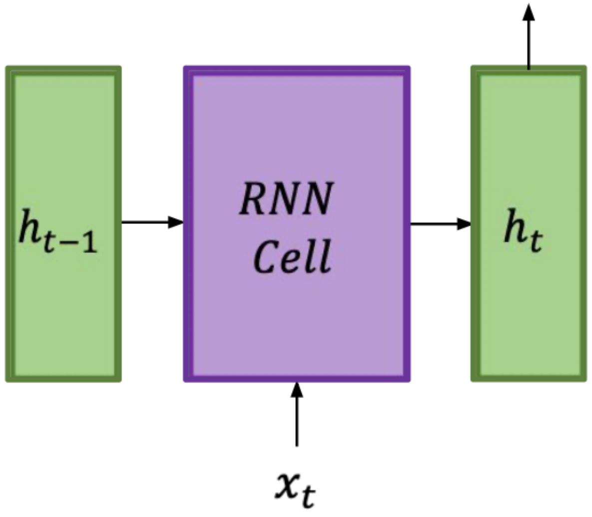
RNNs are bad at **"long term memory"**

Information at time t, s_t, is repeatedly fed through a fully connected layer and needs to remain relatively unchanged until it is needed.

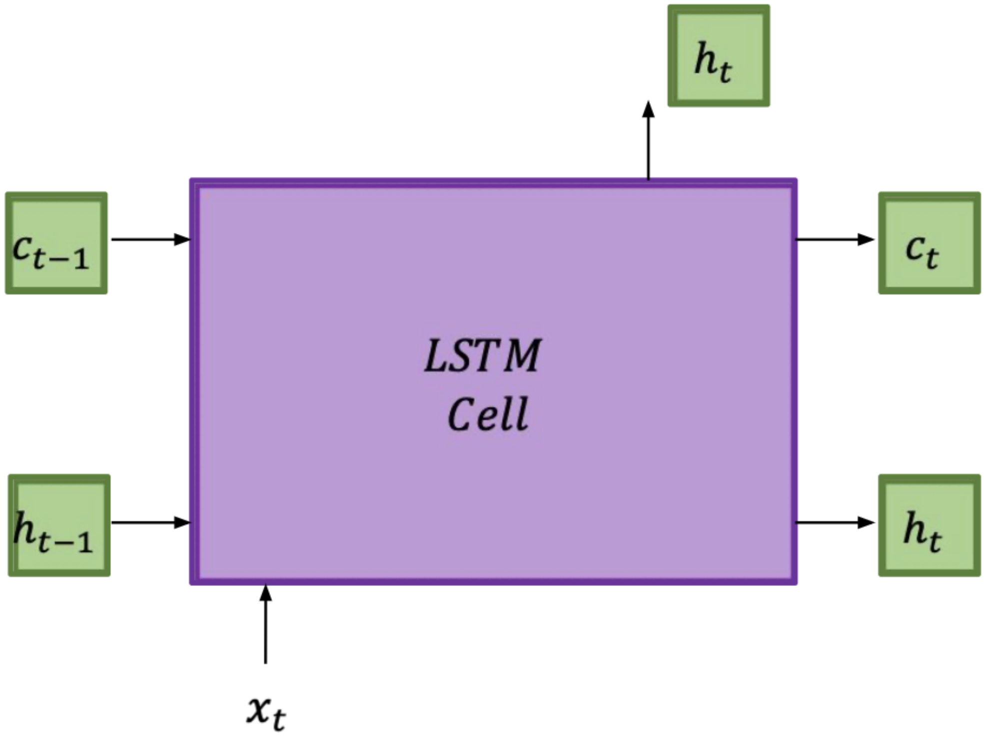


What is different?

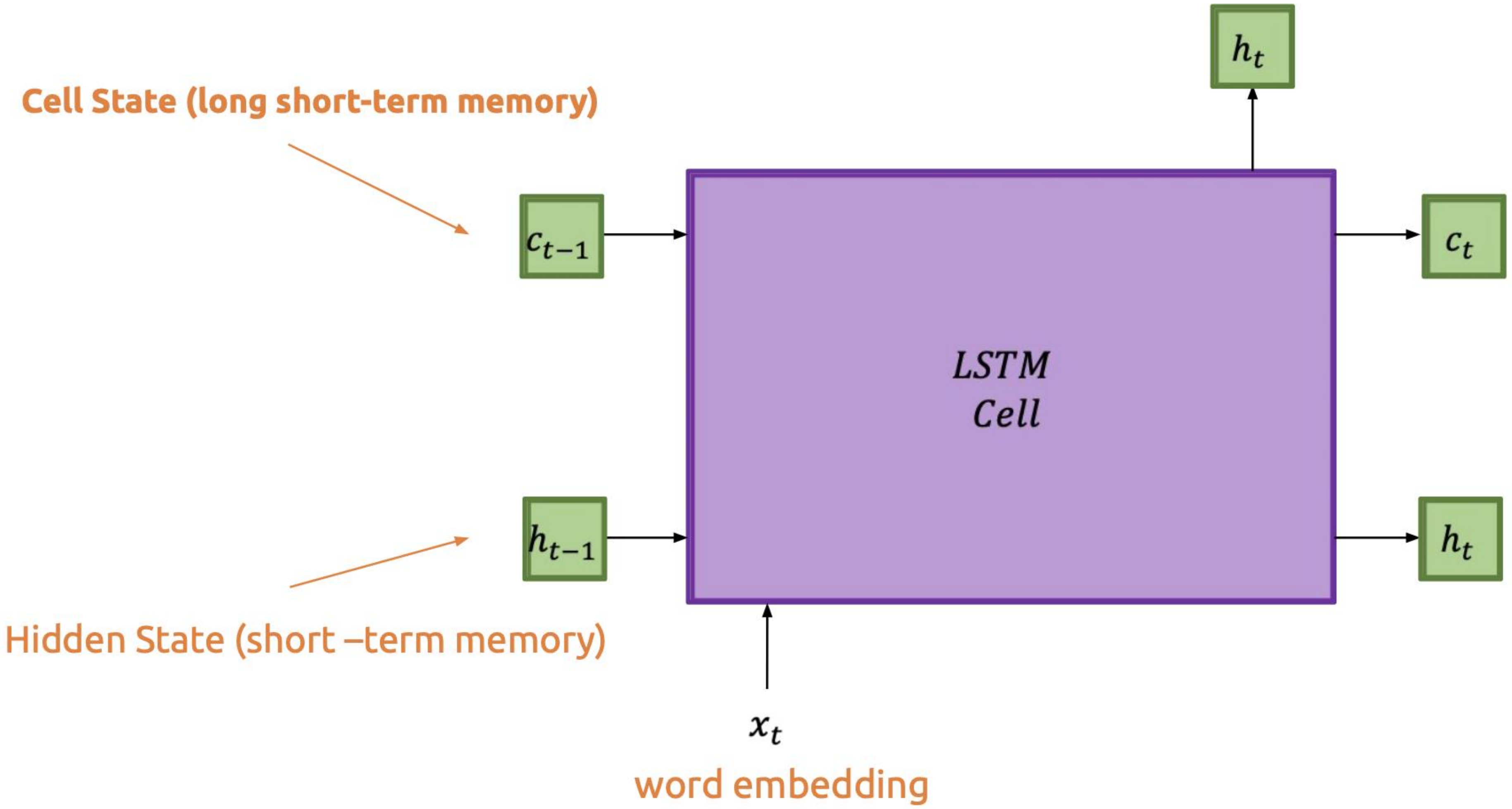
Vanilla RNN



LSTM



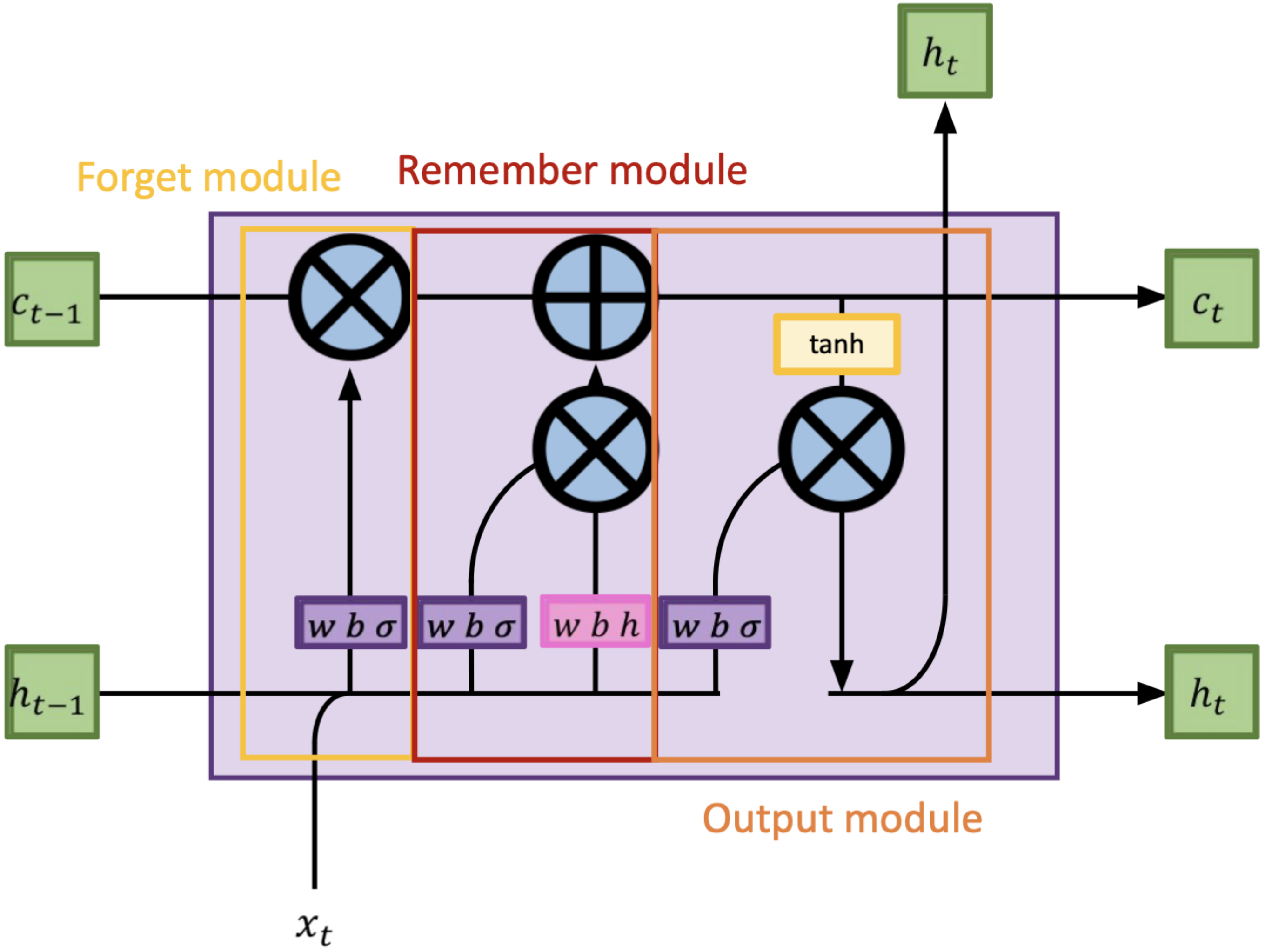
LSTM


Cell State (long short-term memory)
h_t
c_{t-1}
c_t
LSTM
Cell
h_{t-1}
h_t
Hidden State (short –term memory)
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”*

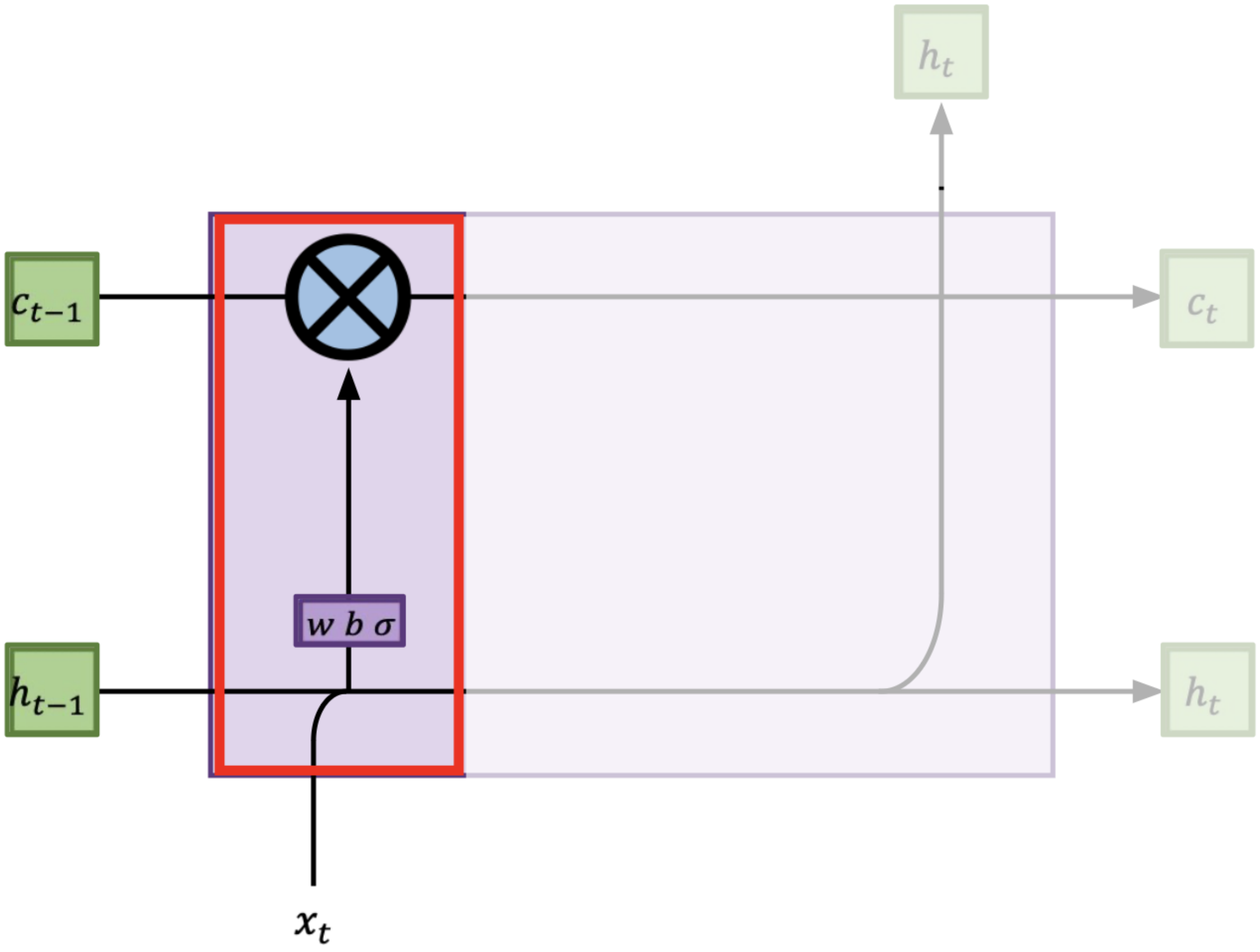
Forget Module

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

Forget Module

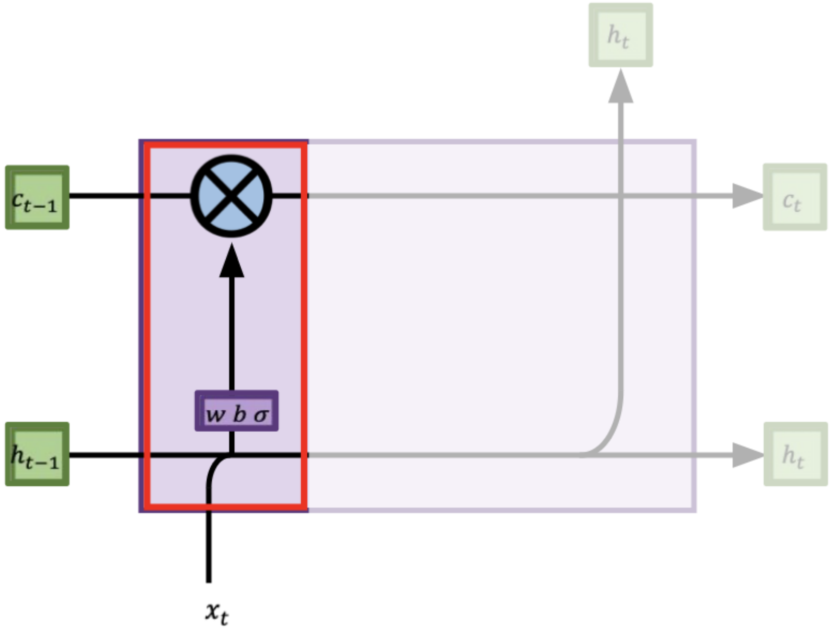
= fully connected layer with sigmoid

= pointwise multiplication



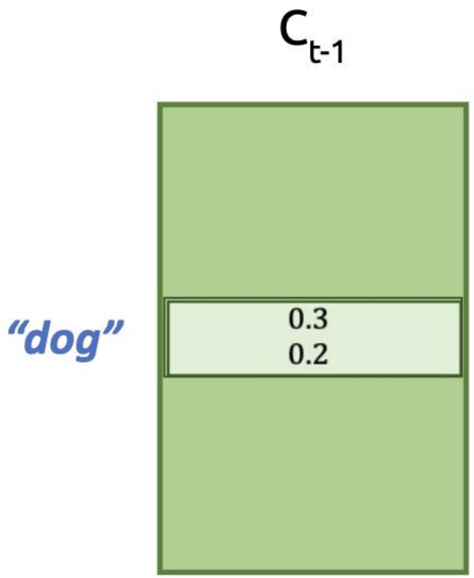
Forget Module

- 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

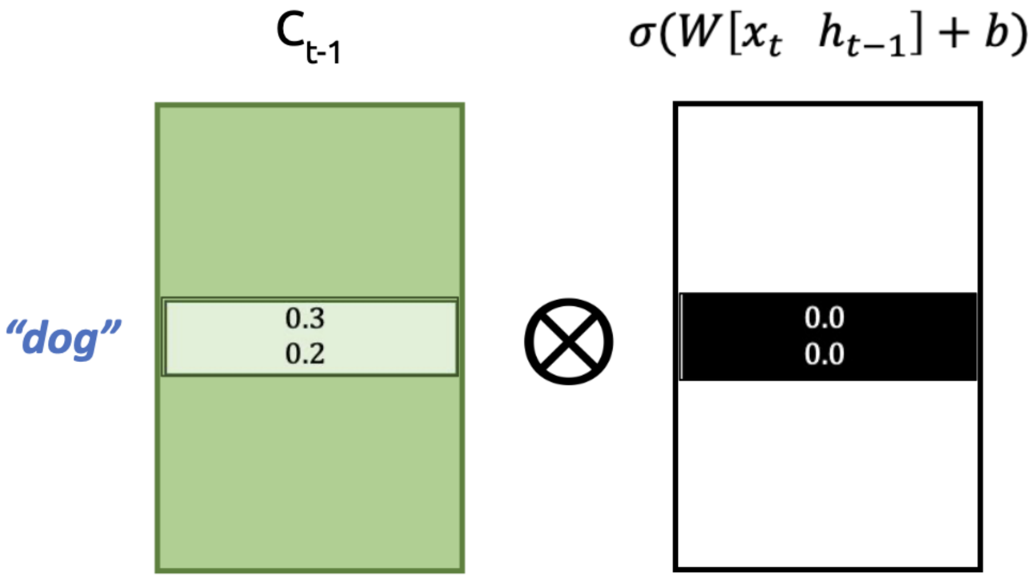
- Use pointwise multiplication by a **mask vector** to forget information
  - What do we want to forget from last cell state?



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

Forgetting information

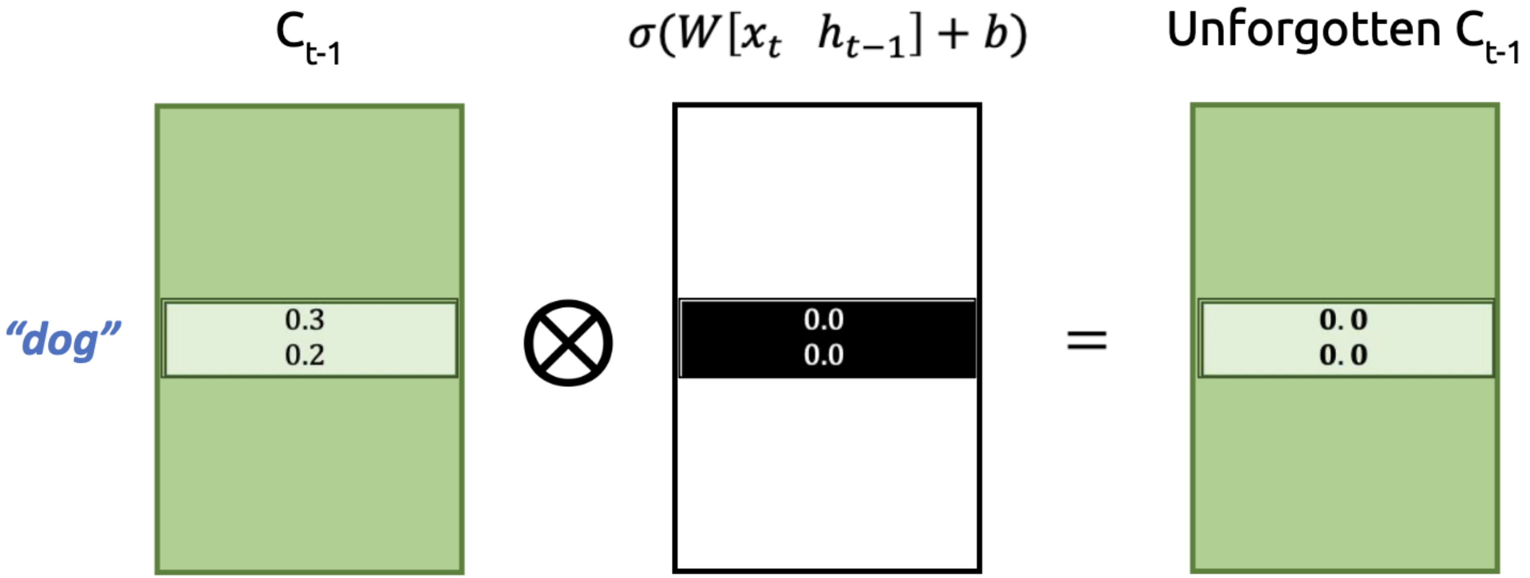
- Use pointwise multiplication by a **mask vector** to forget information
  - What do we want to forget from last cell state?
  - Output of fully connected + sigmoid is what we want to forget



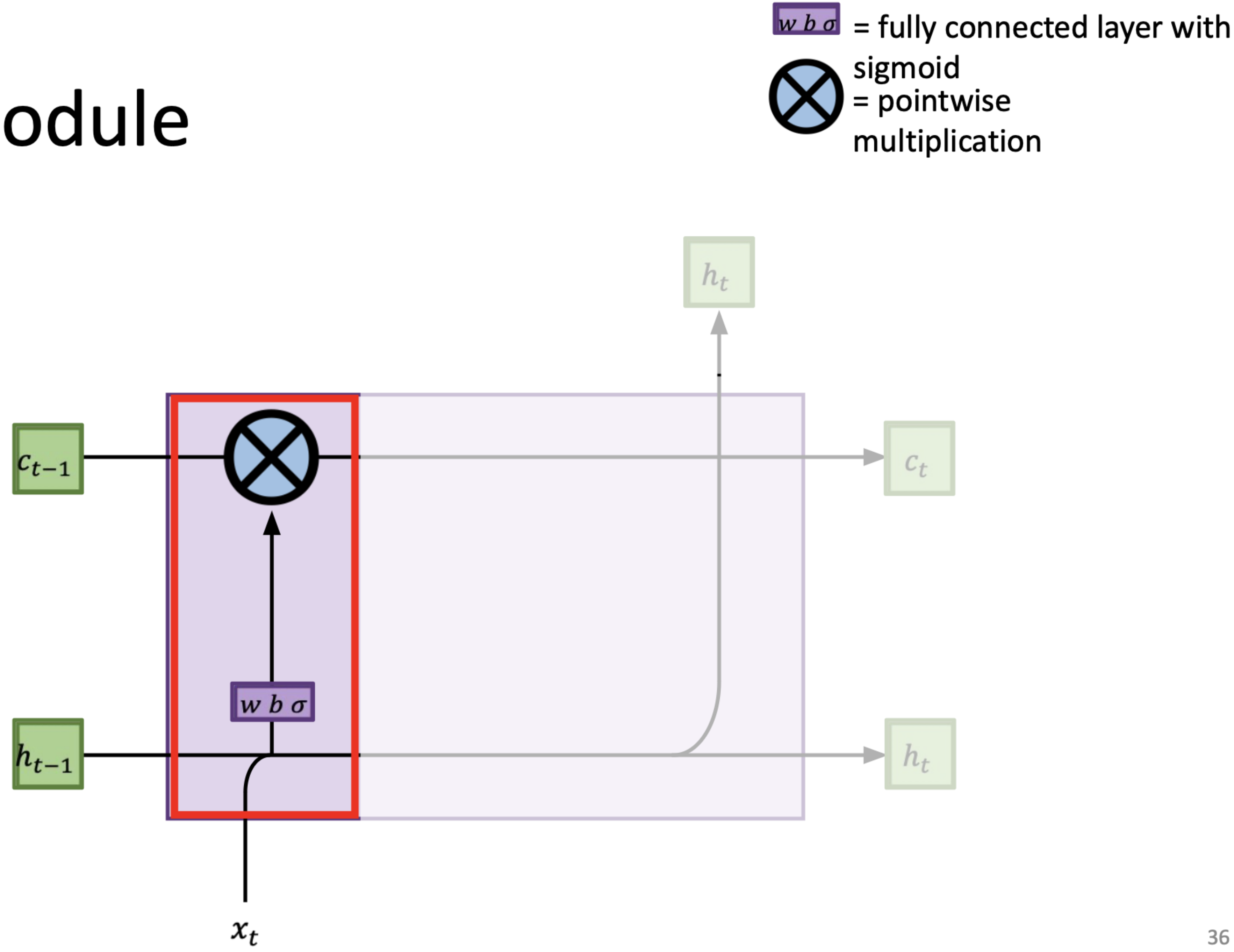
My dog has a fluffy tail. ***I love my dog***

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
  - "Zeros out" a part of the cell state
  - Pointwise multiplication by a learned mask vector is known as ***gating***



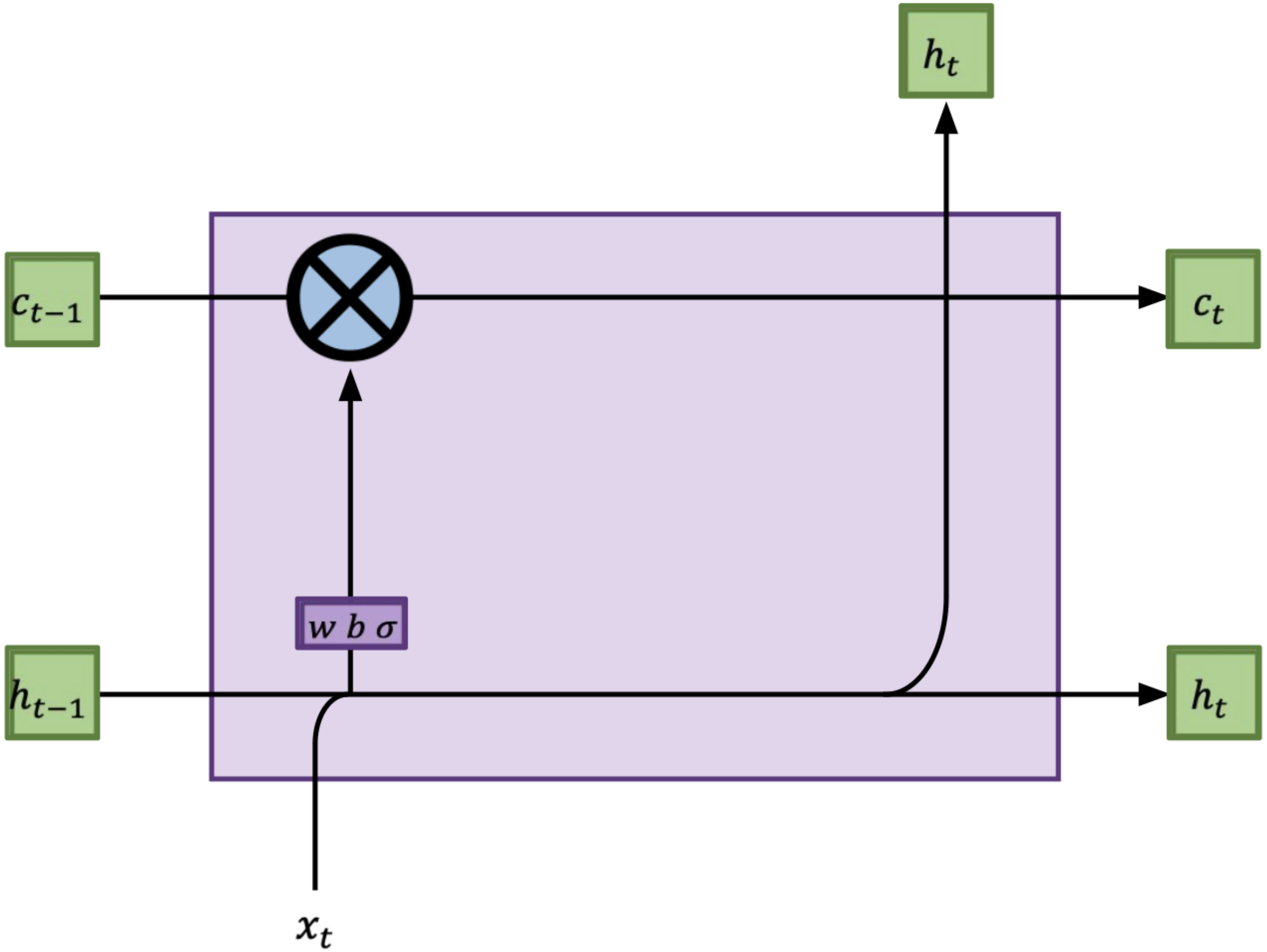
Forget Module



What's next?

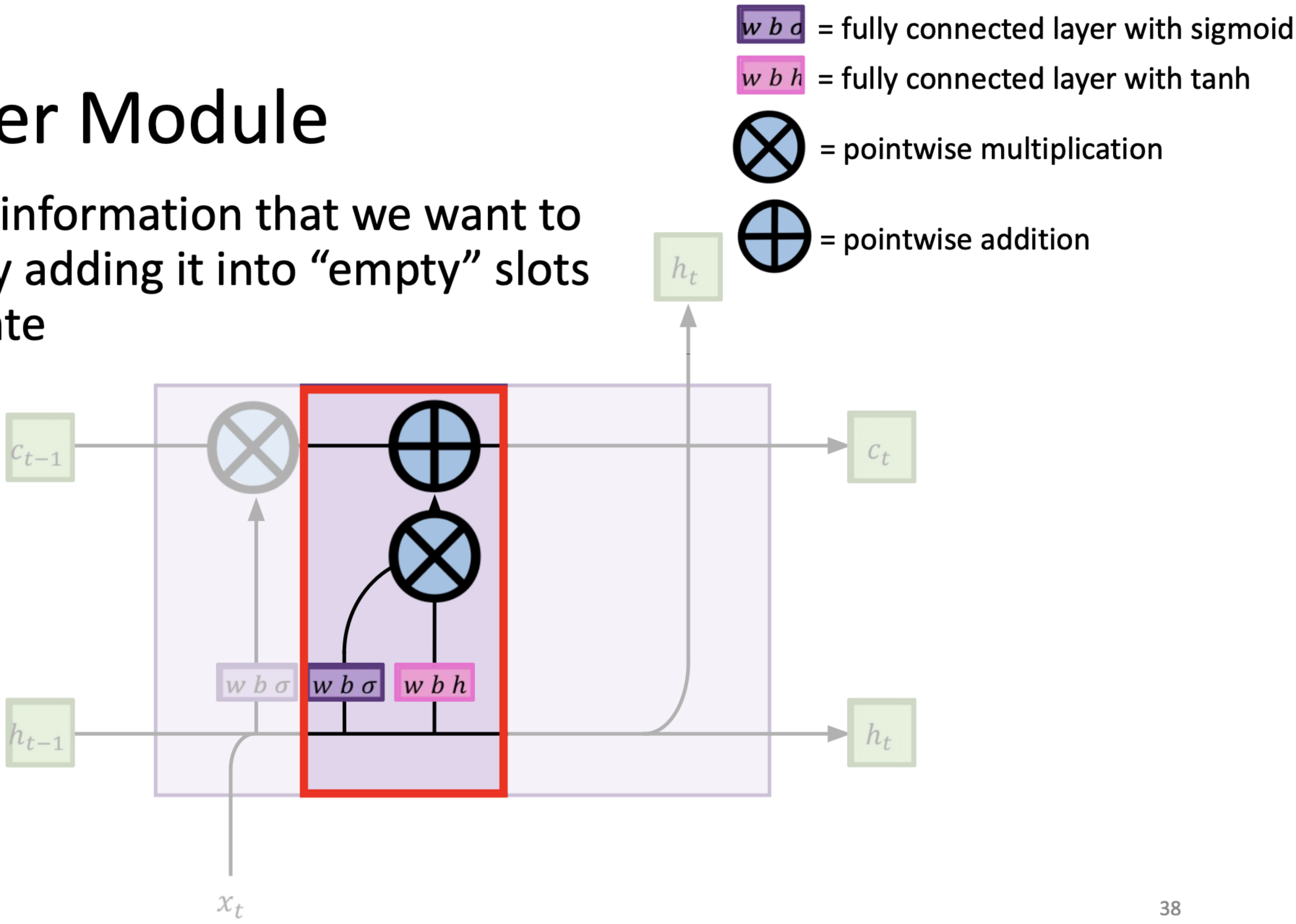
$w\ b\ \sigma$ = fully connected layer with sigmoid

= pointwise multiplication



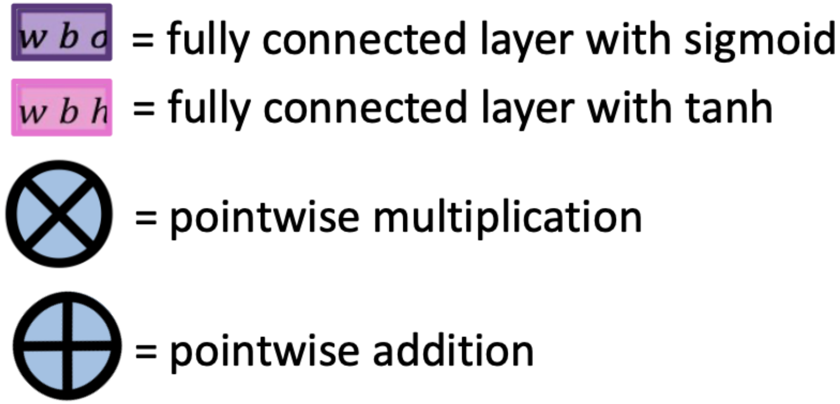
Remember Module

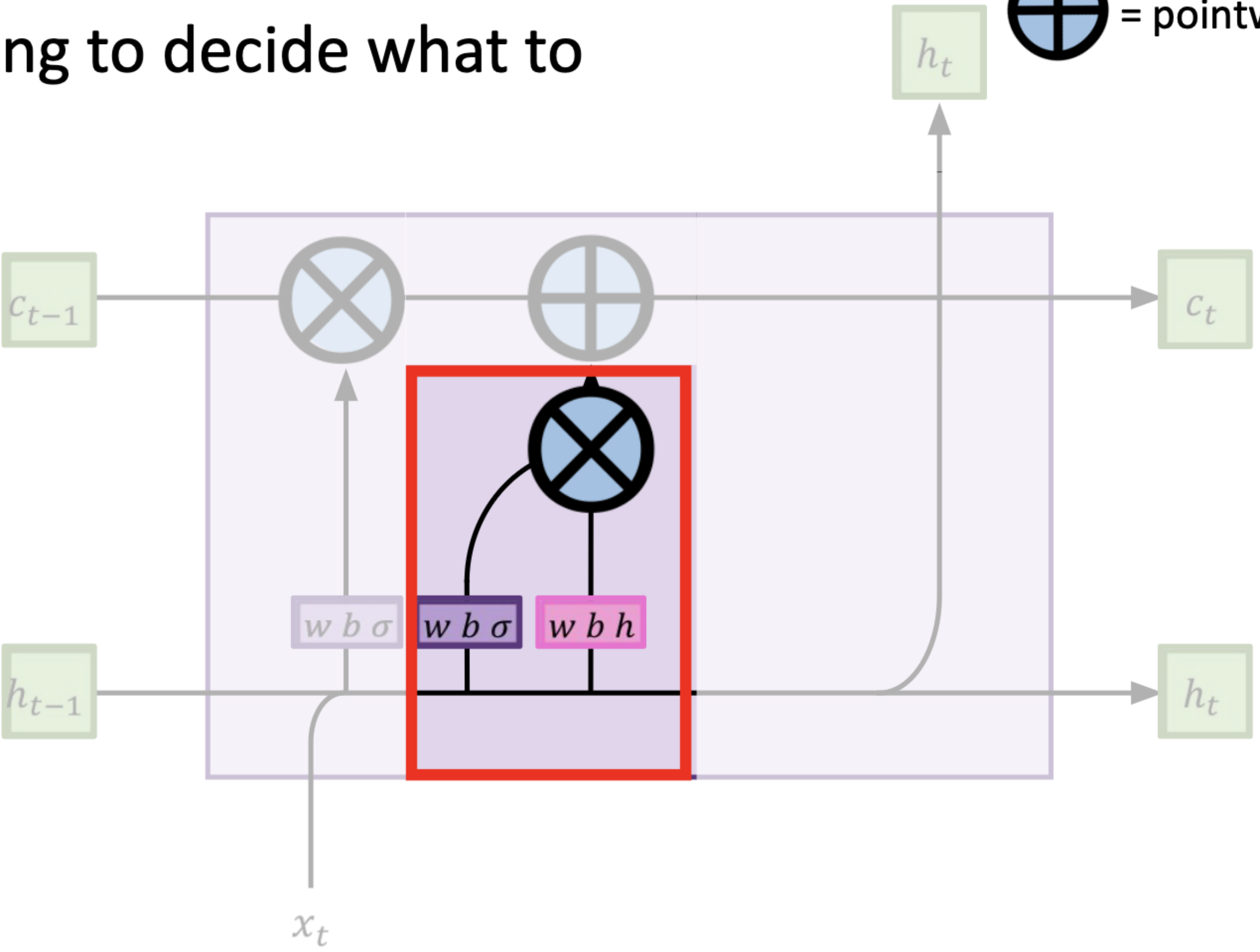
- We can save information that we want to remember by adding it into “empty” slots in the cell state



Remember Module

- First: use gating to decide what to remember



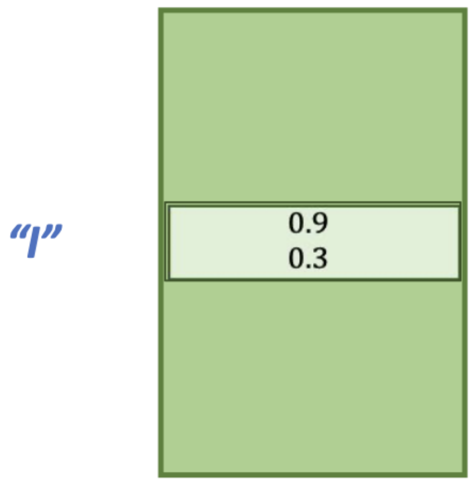


My dog has a fluffy tail. ***I love my dog***

Gating for ‘selective memory’

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

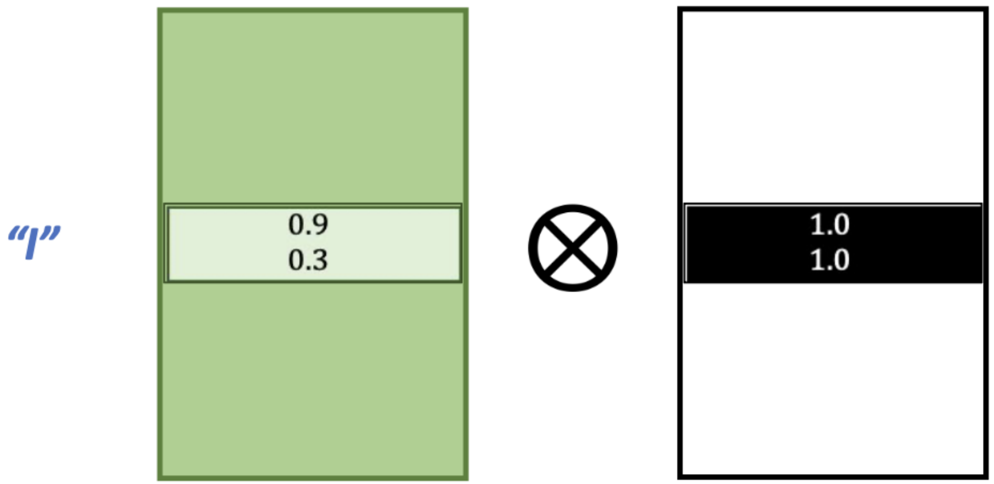
$\text{tanh}(W_1[x_t \;\; h_{t-1}] + b_1)$



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

$\tanh(W_1[x_t \quad h_{t-1}] + b_1)$ $\qquad$ $\sigma(W_2[x_t \quad h_{t-1}] + b_2)$

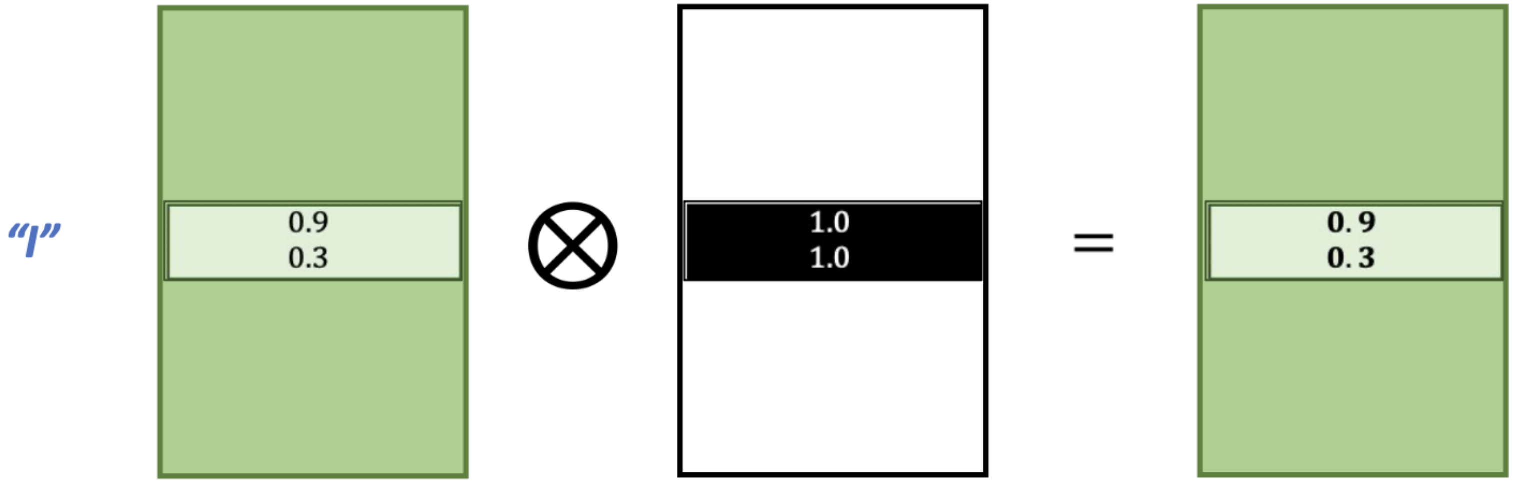


My dog has a fluffy tail. ***I love my dog***

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

$\tanh(W_1[x_t \quad h_{t-1}] + b_1)$ $\quad$ $\sigma(W_2[x_t \quad h_{t-1}] + b_2)$ $\quad$ Selected Memory



= fully connected layer with sigmoid

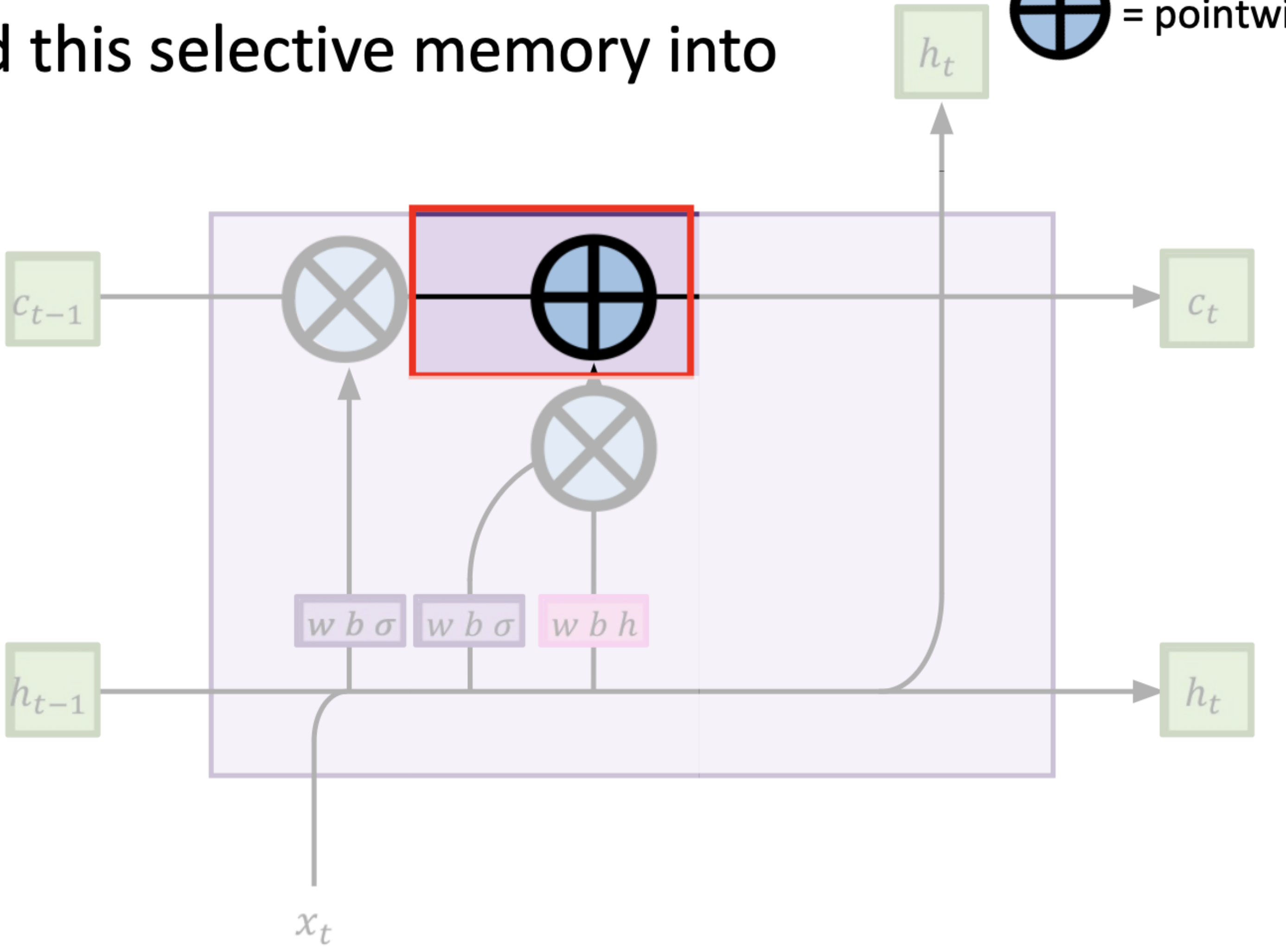
= fully connected layer with tanh

= pointwise multiplication

= pointwise addition

Remember Module

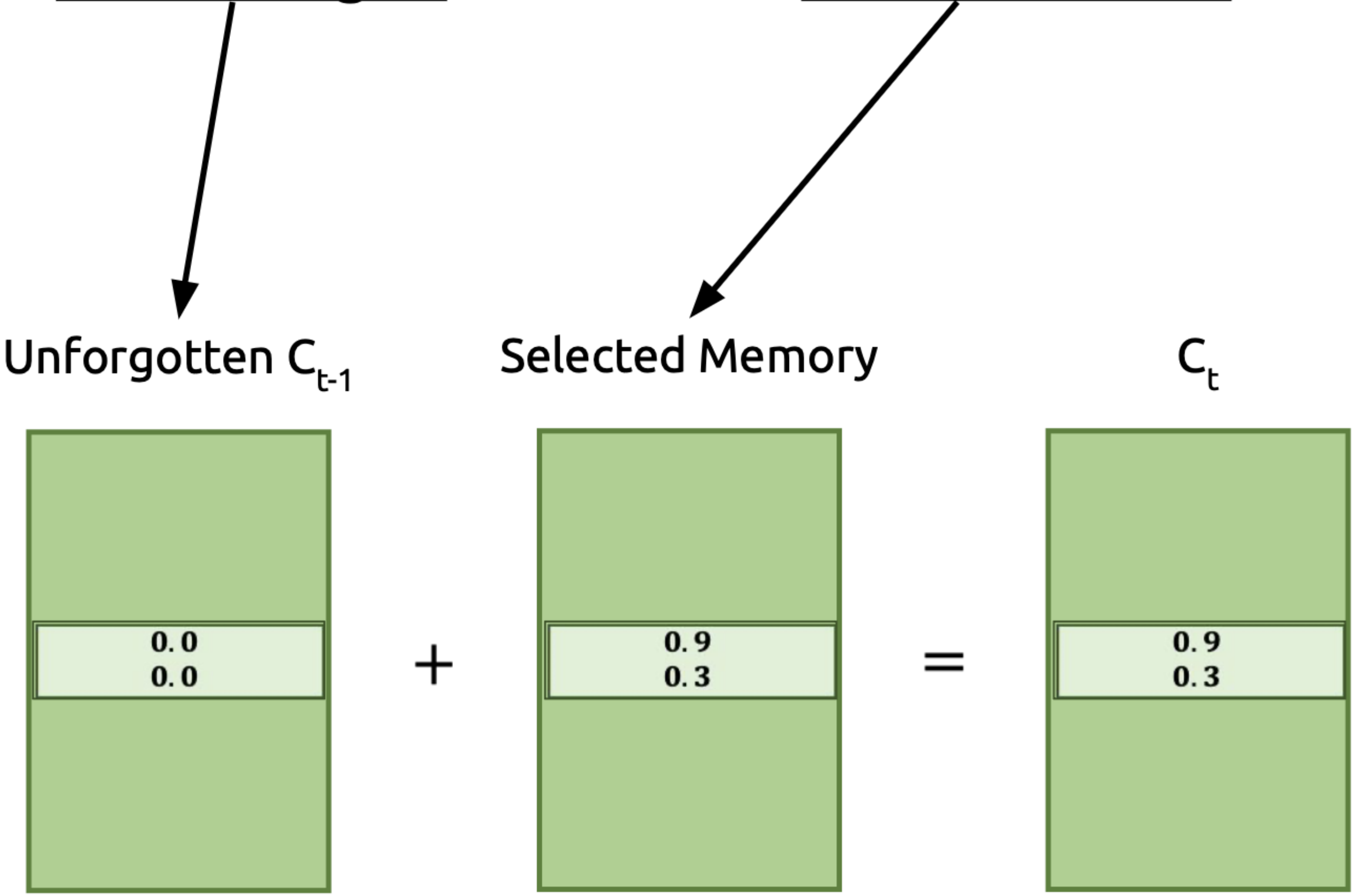
- Then: we add this selective memory into the cell state



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

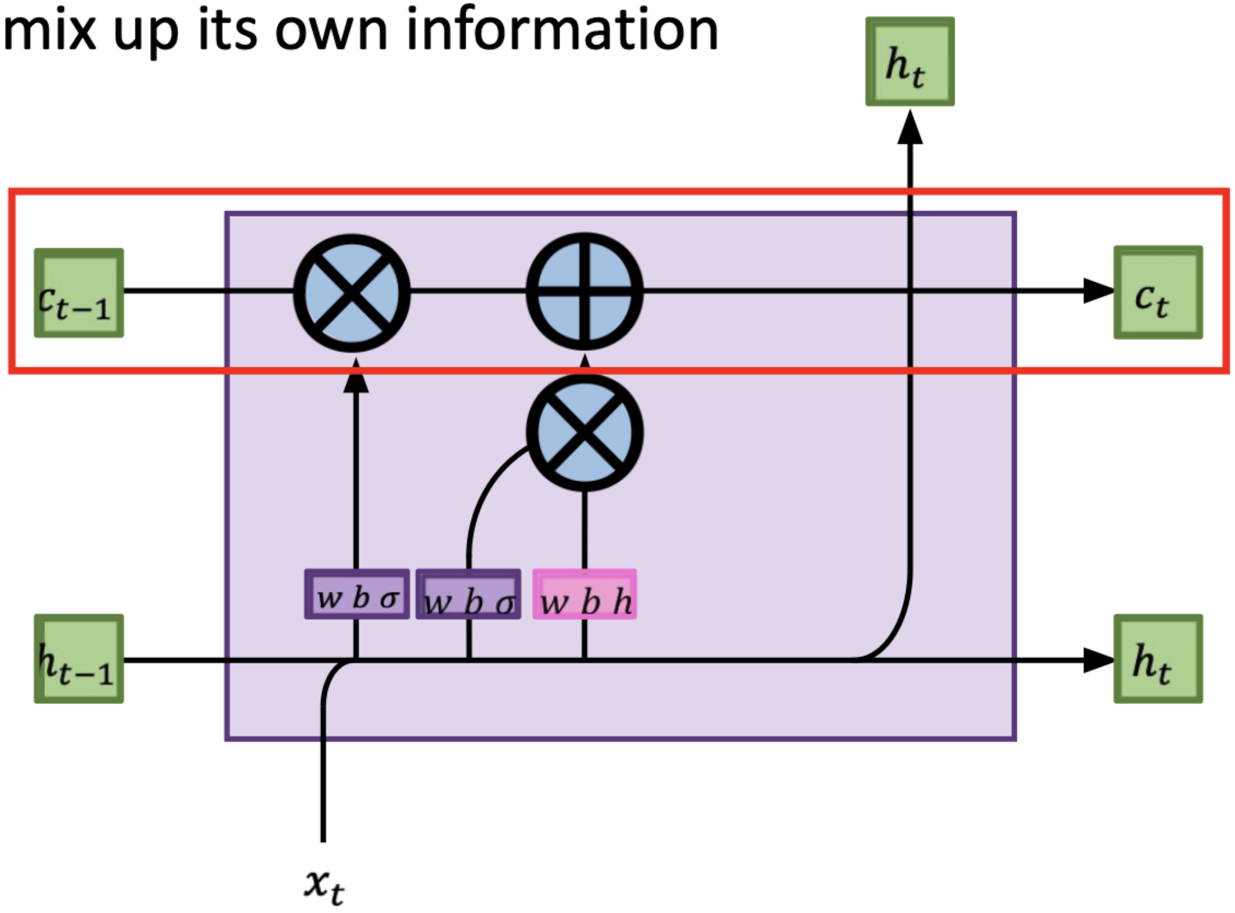
Remembering information

- Add what we <u>didn't forget</u> to what we <u>did remember</u>

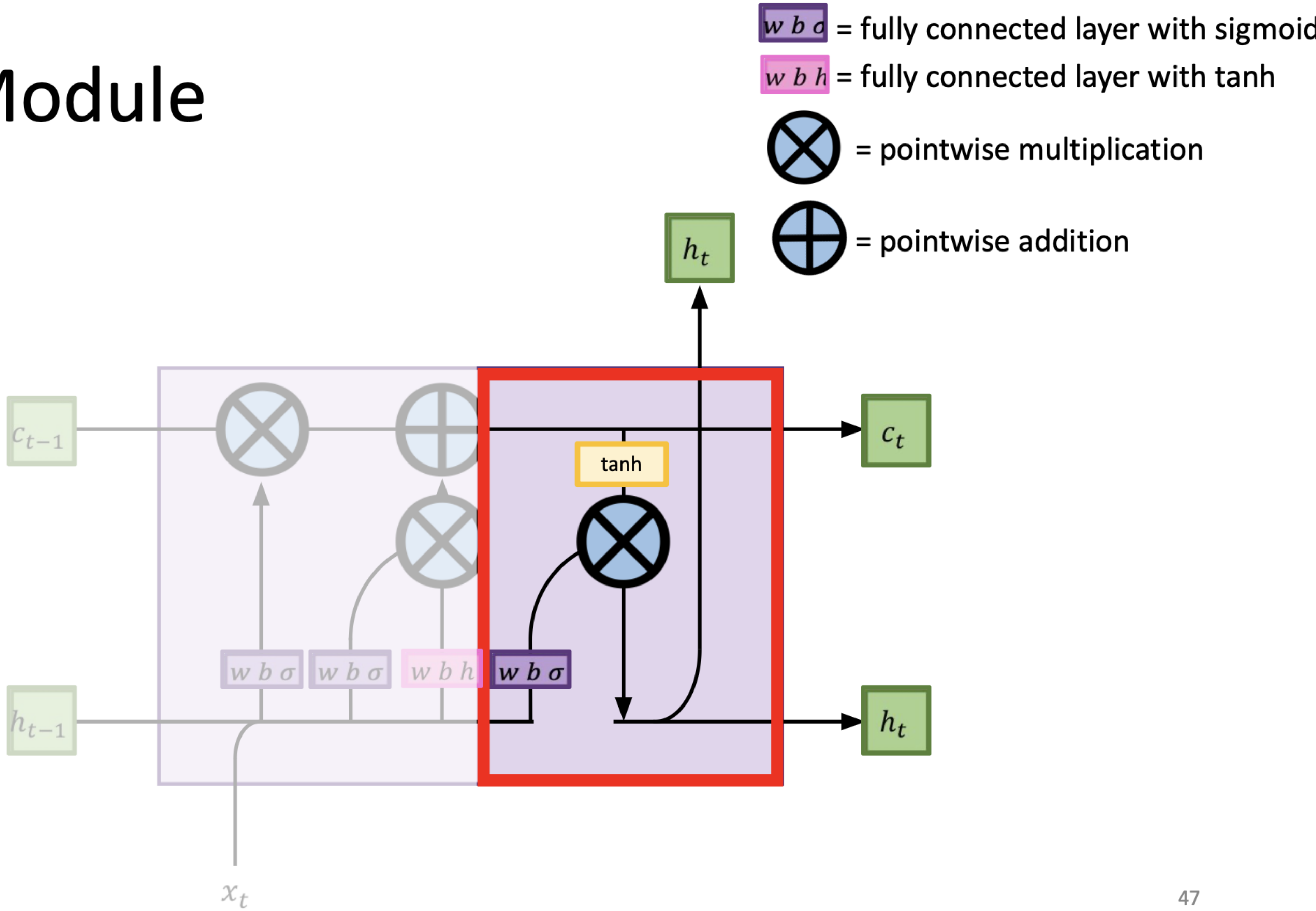


Why does this solve our problem?

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

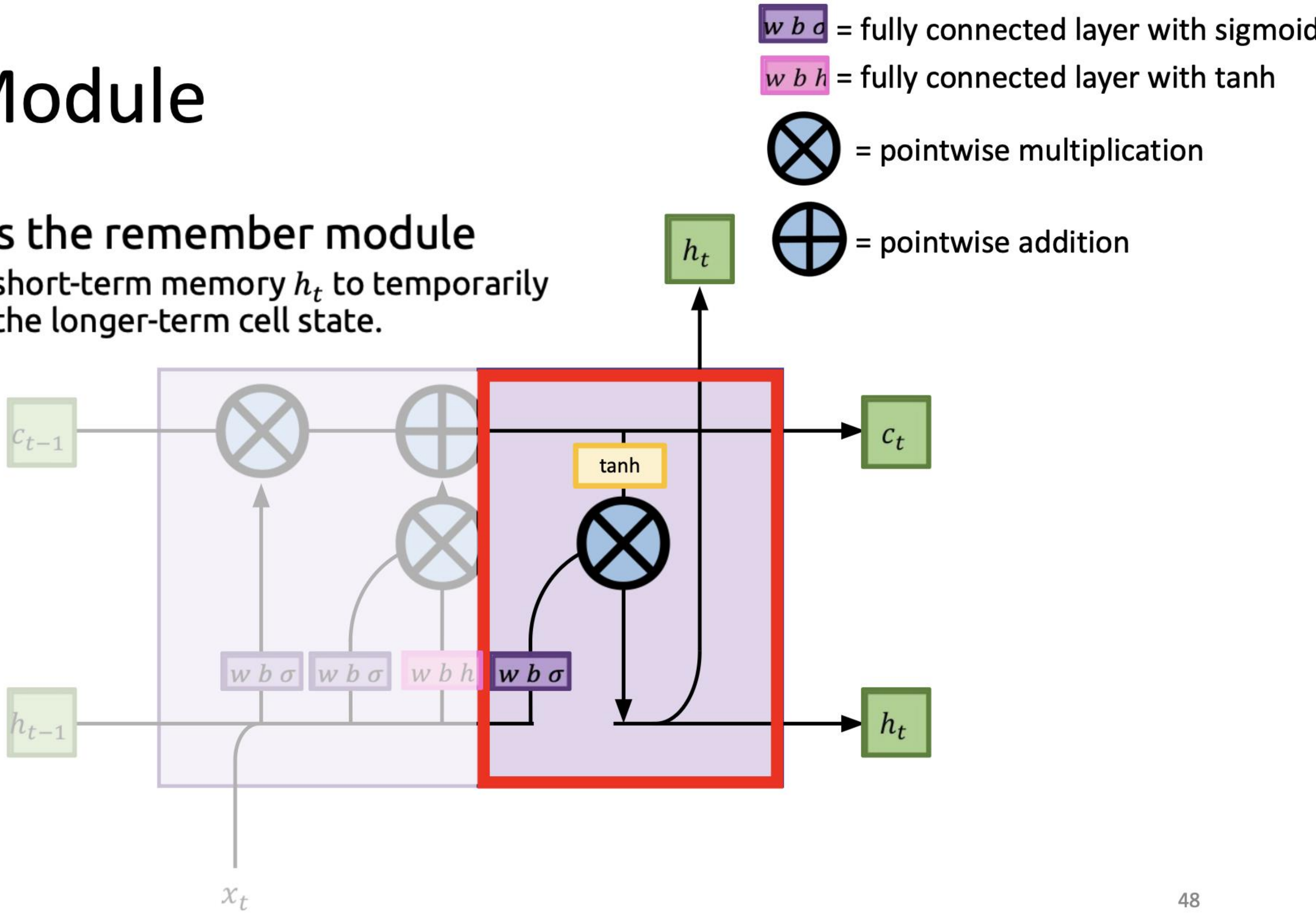


Output Module



Output Module

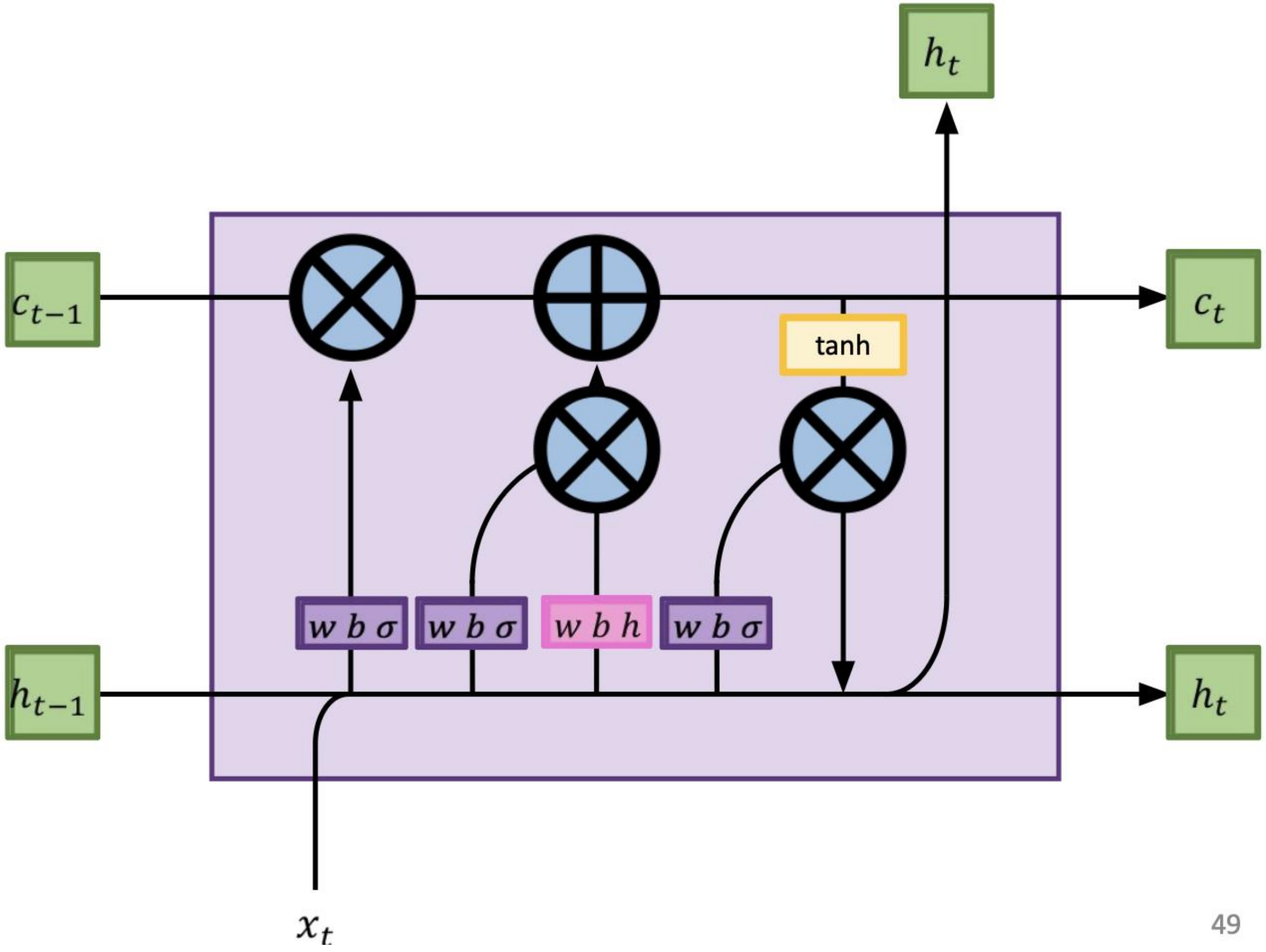
- Same structure as the remember module
  - Provides path for short-term memory h_t to temporarily acquire info from the longer-term cell state.



The Complete LSTM

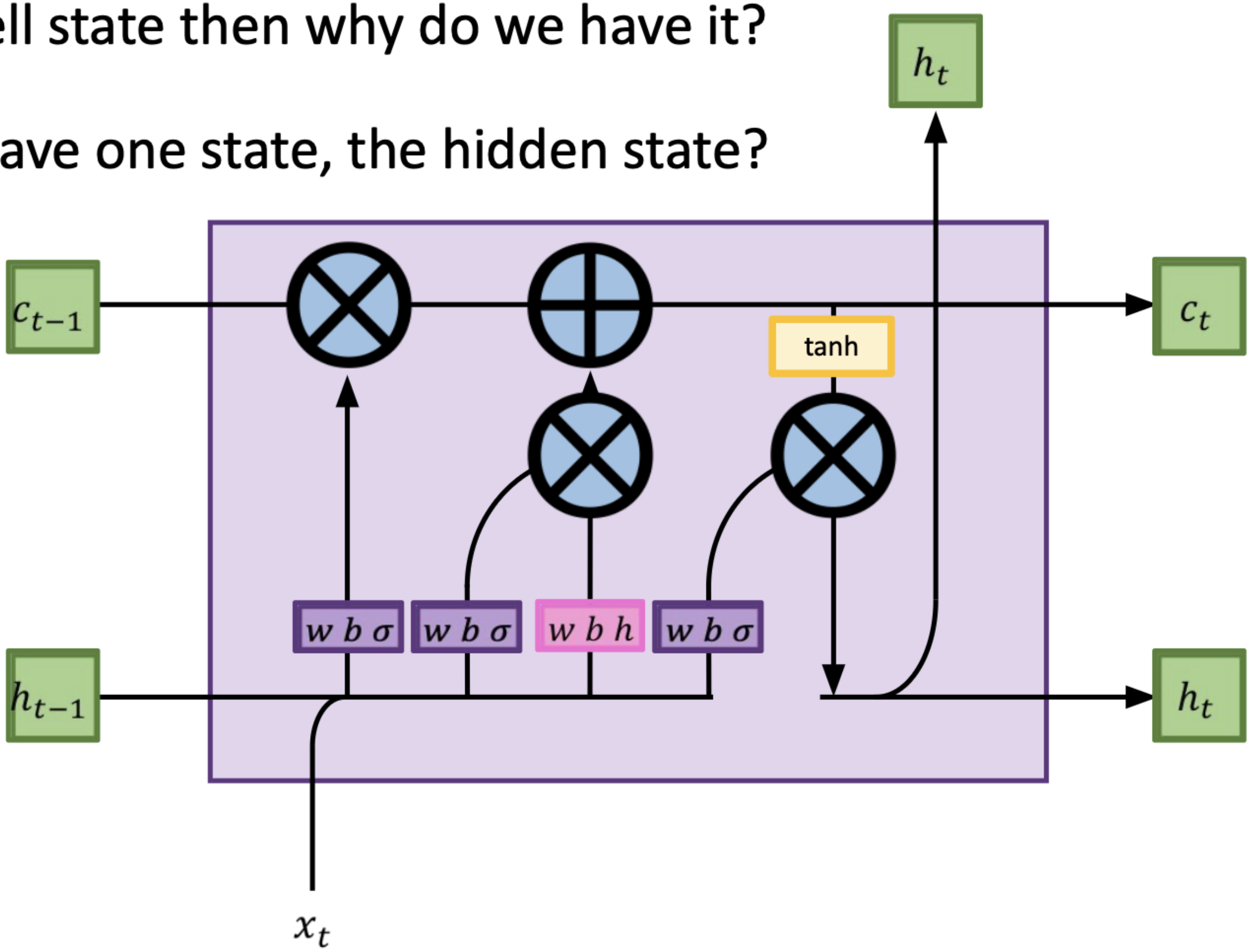
Any questions?

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



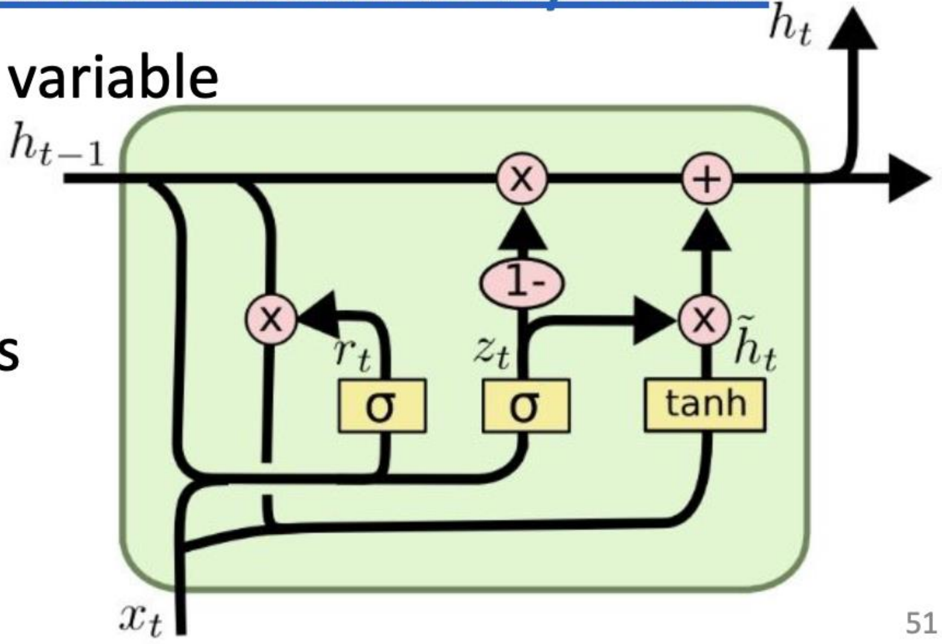
The Complete LSTM

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



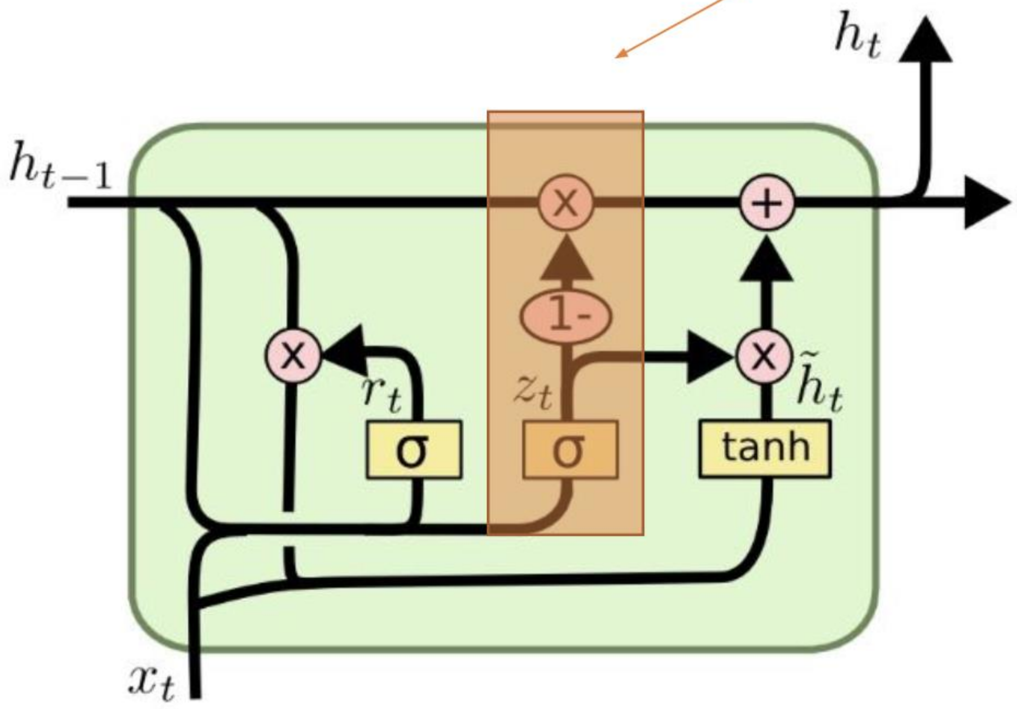
GRU

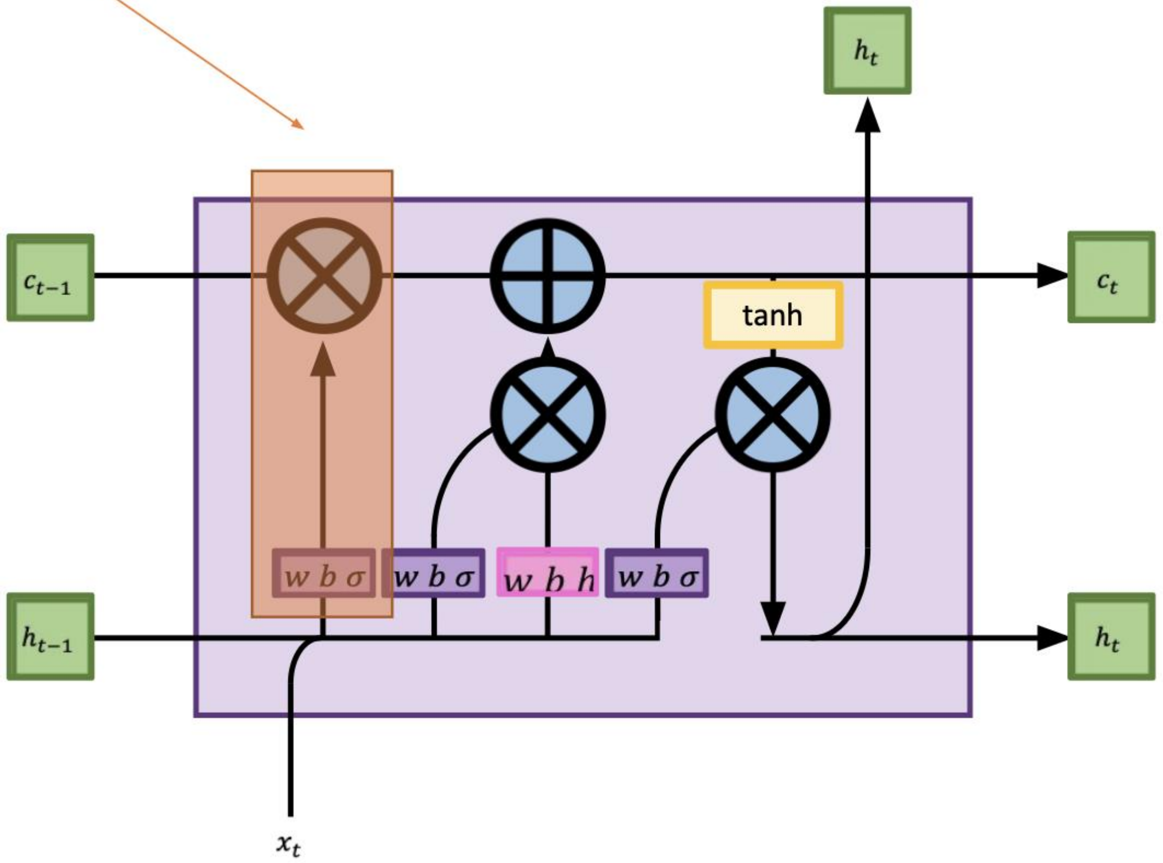
- ***G**ated **R**ecurrent **U**nit*
- In practice, similar performance and may train faster
  - Removes cell state, computationally more efficient and less complex
- In theory, weaker than LSTMs since it cannot unboundedly count
  - Counting: track increment or decrement of variable
  - e.g. Validate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



GRU vs LSTM

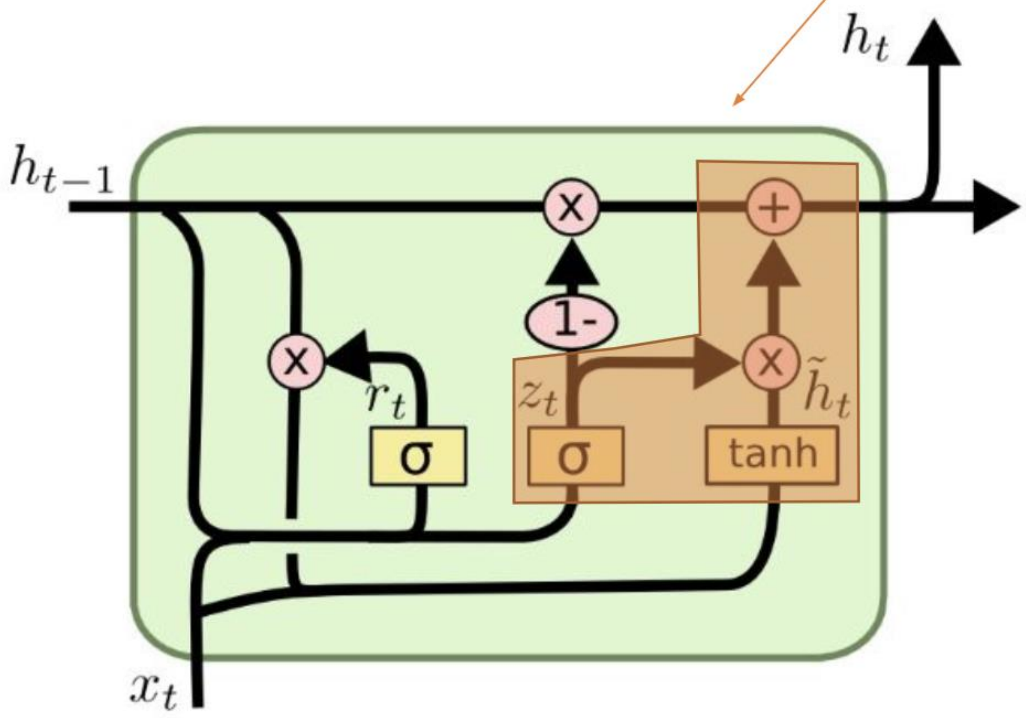
Forget module

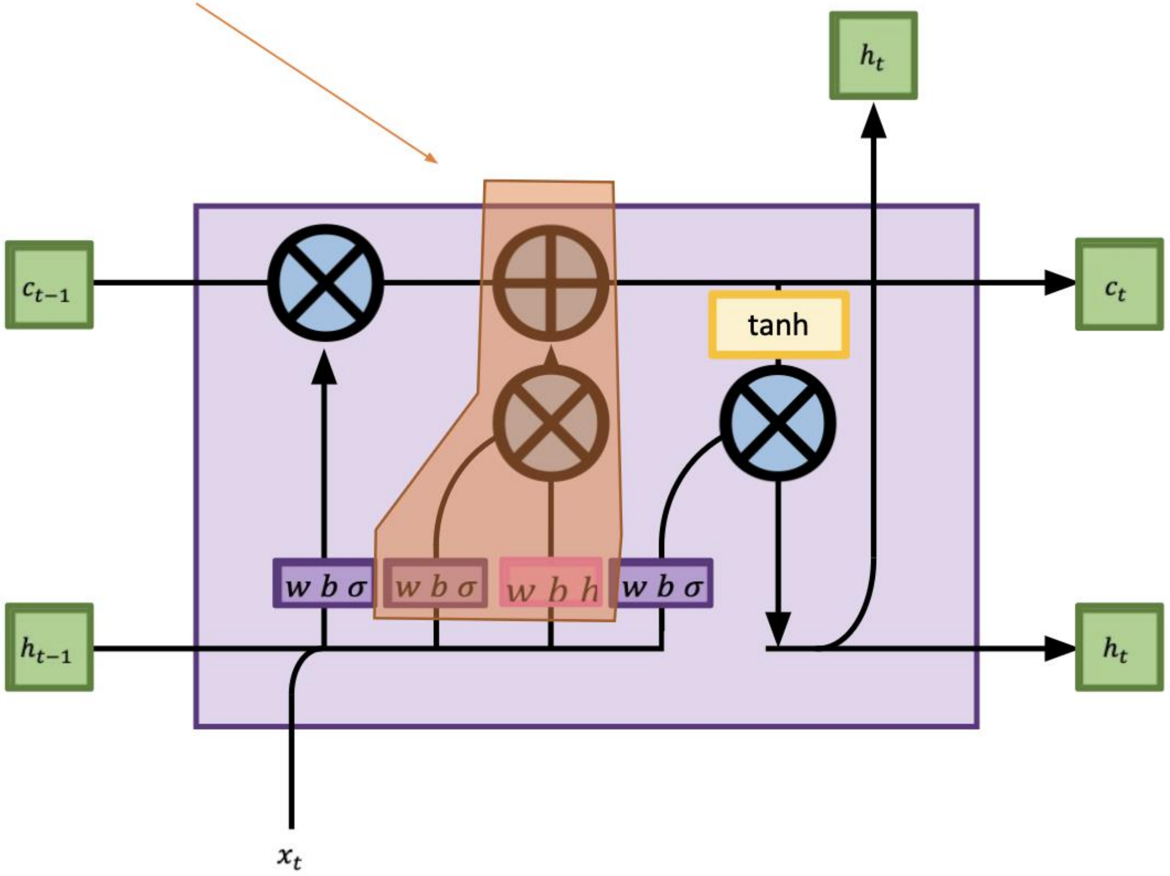




GRU vs LSTM

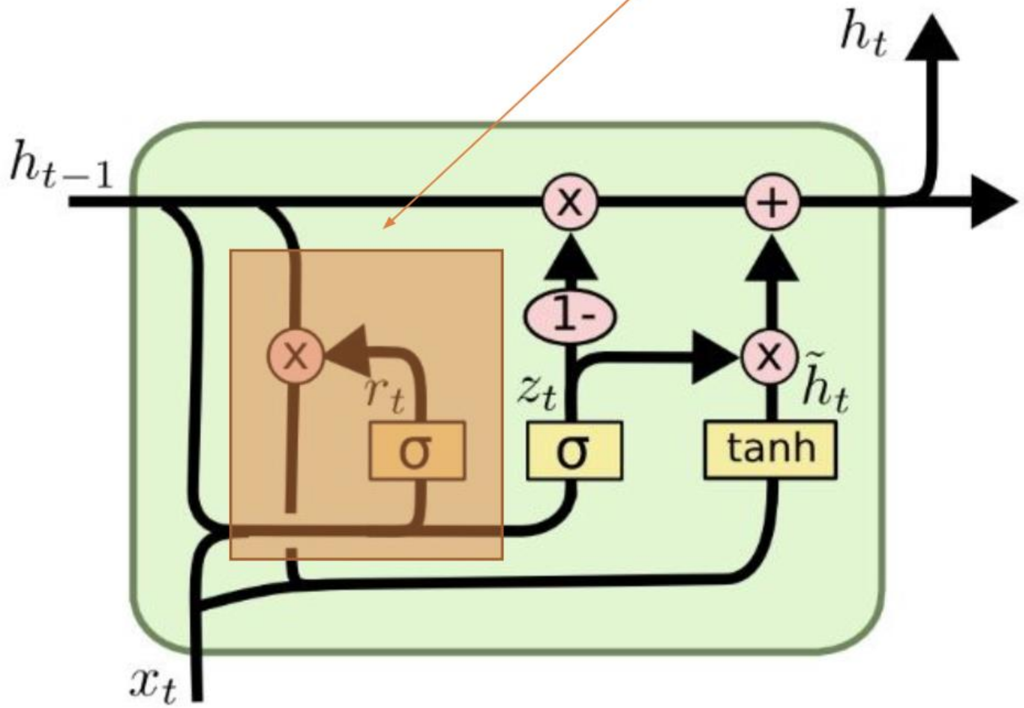
Remember module

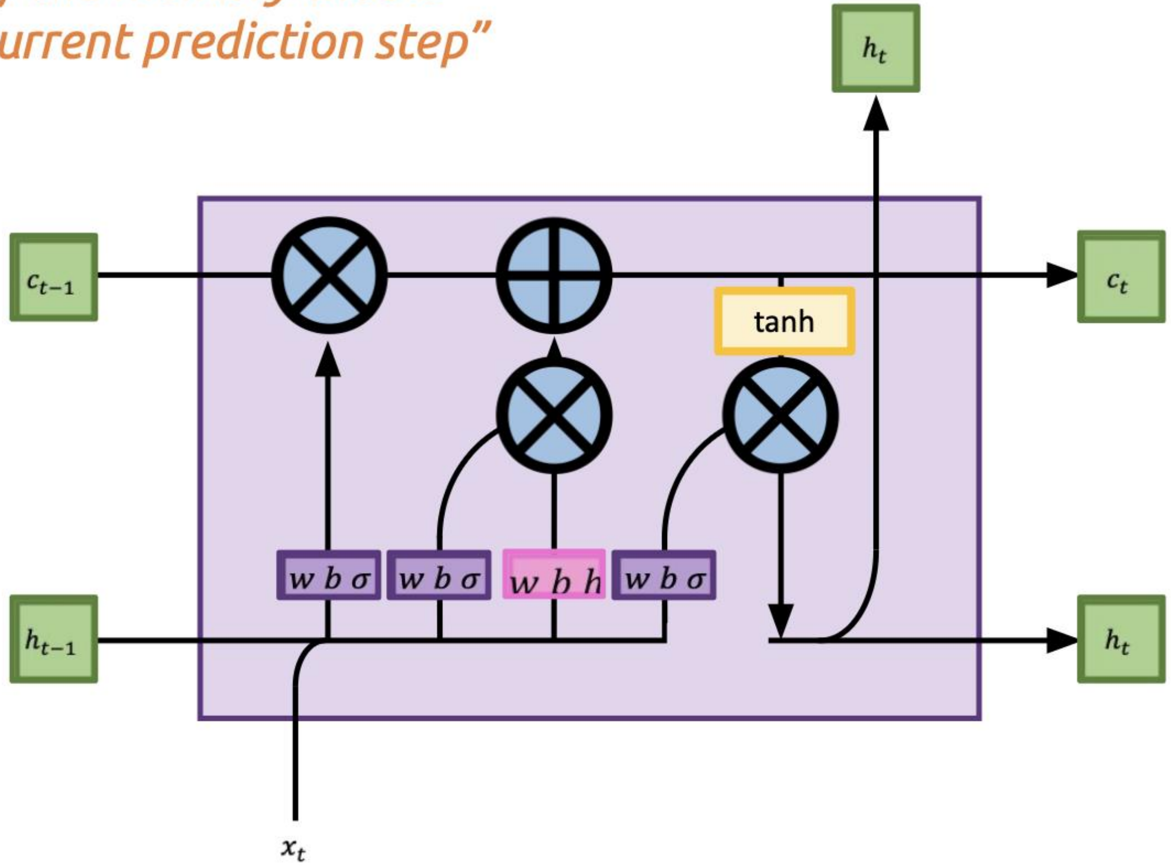




GRU vs LSTM

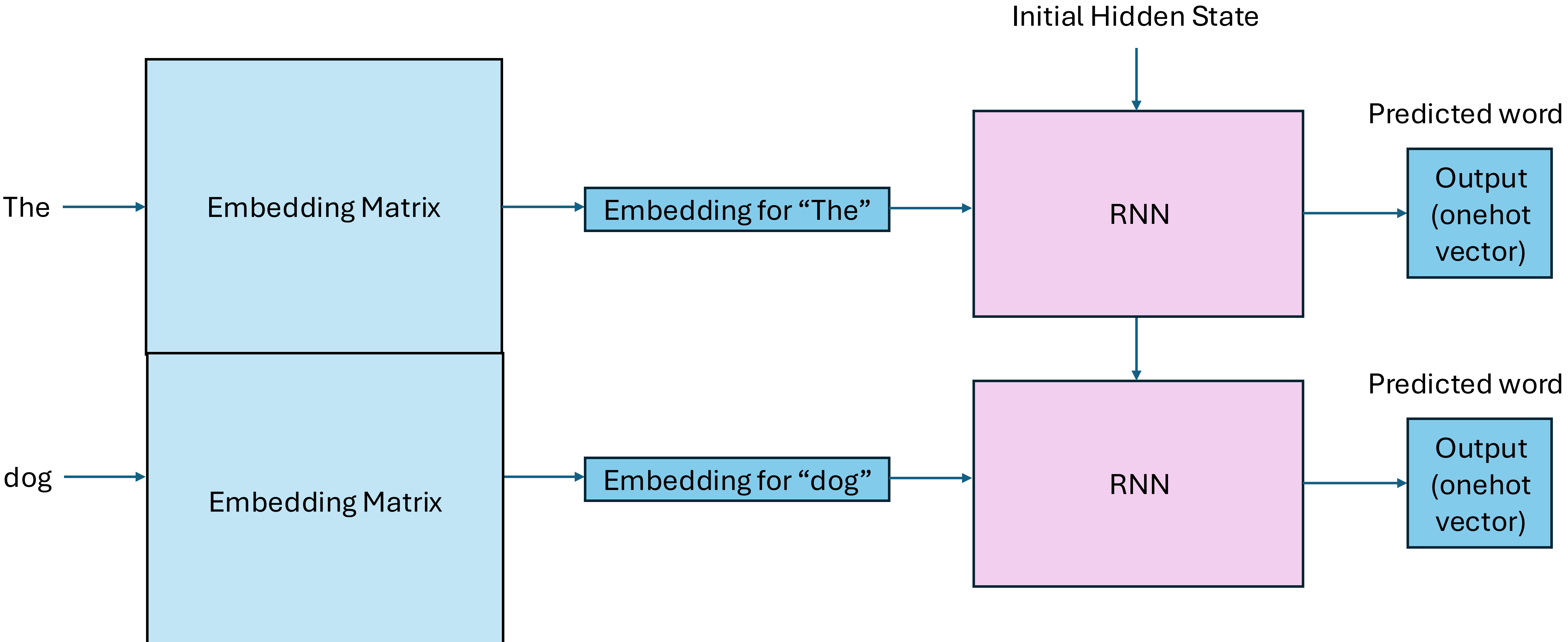
No direct analogue in LSTM
"Select the part of the memory that is relevant for the current prediction step"





Overview of RNN Sequence Prediction

The dog barked at ____



When to compute loss

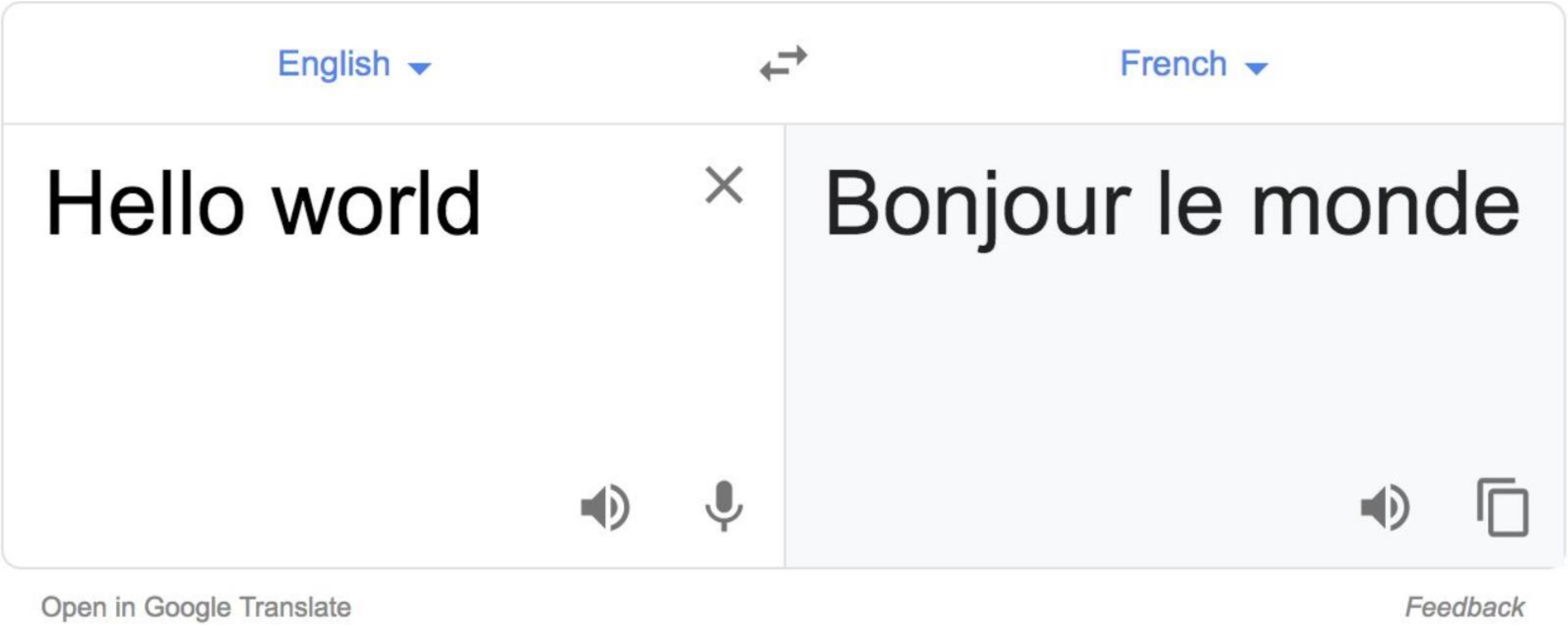
We have predictions and ground truths at every step, why not compute the loss after every word and backprop?

Should your model be penalized equally for incorrect predictions?

1) The ___
2) The dog barked at ___

Machine Translation

Software that translates one language to another



Why is this an interesting problem to solve?

- **Complex:** languages evolve rapidly and don’t have a clear and well-defined structure
  - Example of language change: “awful” originally meant “full of awe”, but is now strictly negative

- **Important**: billions per year spent on translation services
  - >CA$2.4 billion spent per year by Canadian government
  - >£100 million spent per year by UK government

Parallel Corpora

- We need pairs of equivalent sentences in two languages, called ***parallel corpora***

Canadian Hansards

- Hansards are transcripts of parliamentary debates
- Canada’s official languages are English and French, so everything said in parliament is transcribed in both languages

Canadian Hansards: Examples

English	French
What a past to celebrate.	Nous avons un beau passé à célébrer.
We are about to embark on a new era in health research in this country.	Le Canada est sur le point d'entrer dans une nouvelle ère en matière de recherche sur la santé.

Canadian Hansards

- We can use this as a dataset for MT!
- Not perfect:
  - *Translations aren’t literal*: in the example, “this country” is translated to “Le Canada”
  - *Biased in style*: not everyone speaks like politicians in parliamentary debate
  - *Biased in content*: some topics are never discussed in parliament

Other parallel corpora

- Europarl, a parallel corpus of 21 languages used in the European Parliament
- EUR-Lex, a parallel corpus of 24 languages used in EU law and public documents
- Japanese-English Bilingual Corpus of Wikipedia's Kyoto Articles

Problems with parallel corpora

Any questions?

- Expensive to produce
- Tend to be biased towards particular types of text – e.g. government documents containing formal language
- Translations aren't necessarily literal - e.g. "this country" -> "Le Canada"
- Parallel corpora are necessary, **but never perfect**

LM approach

- Language modelling works on a word-by-word basis, taking only previous words as input

$$P\left(w_{t,i}\right) = P\left(w_{t,i} \mid w_{s,i-1}, w_{s,i-2}, \dots, w_{s,0}\right)$$

- Where $w_{t,i}$ is the i^{th} word in the target sentence, and $w_{s,\,i}$ is the i^{th} word in the source sentence

Will it work for MT task?

Why our LM approach doesn't work for MT

- Language modelling works on a word-by-word basis, taking only previous words as input

$$P(w_{t,i}) = P(w_{t,i} \mid w_{s,i-1}, w_{s,i-2}, \ldots, w_{s,0})$$

- Where $w_{t,i}$ is the i^{th} word in the target sentence, and $w_{s,i}$ is the i^{th} word in the source sentence
- However, **it is not a given that the information we need comes in the preceding words**
- The order and length of the source and target sentences are not necessarily equal

Example from Hansards

- For example, take the first entry in Hansard's:

edited hansard number 1

hansard révisé numéro 1

What should we do?

Further examples

French: “Londres me manque”

Naive translation: “London I miss”

Correct translation: “I miss London”

French: “Je viens de partir”

Naive translation: “I come of to go”

Correct translation: “I just left”

Sequence to Sequence (seq2seq)

Thus, we cannot simply use the previous words – we need to ***summarize the source sentence first***

This is called sequence to ***sequence to sequence learning***, or ***seq2seq***

Sequence to Sequence (seq2seq)

- Instead of:

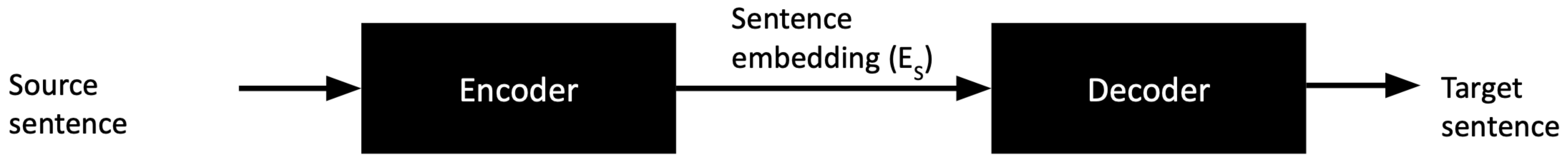
$$P(w_{i,t}) = P(w_{i,t} \mid w_{i-1,s}, w_{i-2,s}, \dots, w_{0,s})$$

Let's do:

$$P(w_{i,t}) = P(w_{i,t} \mid E_S, w_{i-1,t}, w_{i-2,t}, \dots, w_{0,t})$$

Where E_S is a summary, or ***embedding***, of the sentence taken from the source language, and w_i is the i^{th} word of the sentence in the target language

What will the neural net look like?



Origin of the encoder/decoder terminology: information theory

- The encoder “compresses” the source sentence into a compact “code”
- The decoder recovers the sentence (but in the target language) from this code

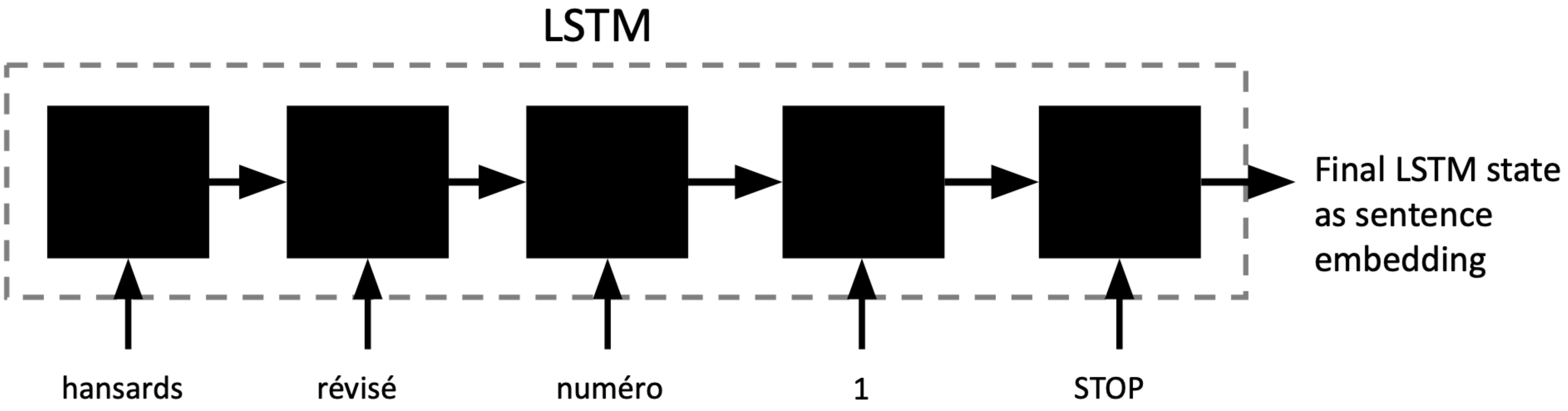
What will the neural net look like?

Any ideas?

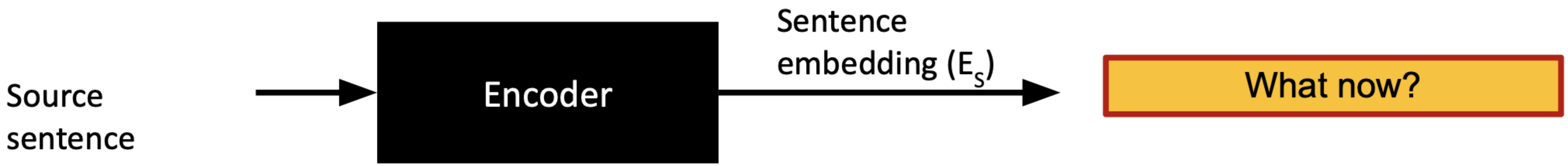
Encoder

- To generate the sentence embedding, we need an encoder
- Use an LSTM
- Feed in the source sentence
- Take the final LSTM state as the sentence embedding
- This will be a ***language-agnostic*** representation of the sentence
    - i.e. it will represent the *meaning* of the sentence without being tied to any particular language

Encoder architecture

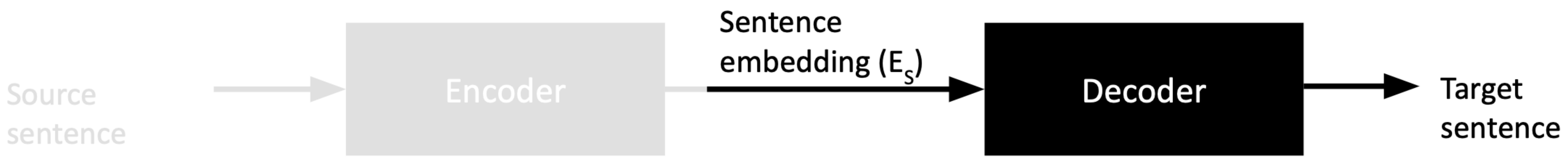


What will the neural net look like?



What will the neural net look like?

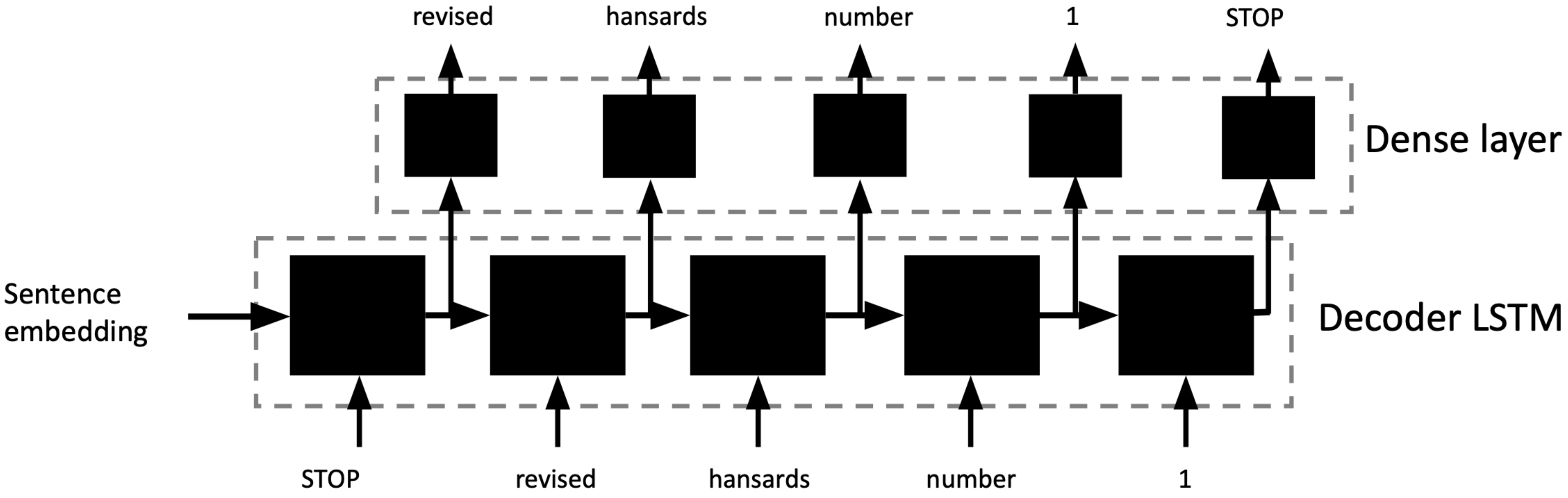
Any ideas?



Decoder

- We now have a sentence embedding representing the meaning of the source sentence
- Now, let's generate a sentence in the target language with the same meaning
- Use an LSTM again, **with the sentence embedding** as its initial hidden state
- The rest is just like language modeling:
    - Input to the LSTM is the previous word from the target sentence
    - Take each LSTM output and put it through a fully connected layer
    - Softmax to convert to probability distribution over next word in target language

Decoder architecture



Any questions?