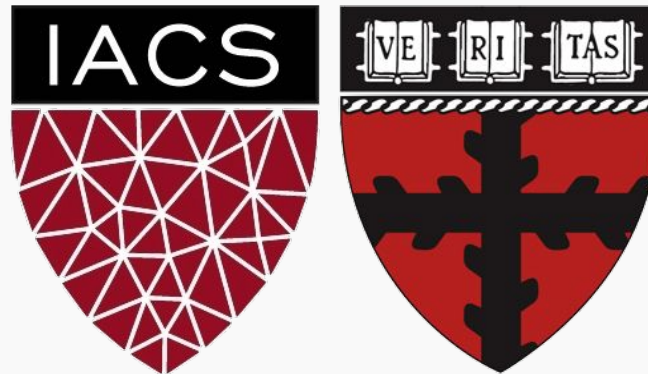


# Lecture 15: Recurrent Neural Networks dos

## CS109B Data Science 2

Pavlos Protopapas, Mark Glickman, and Chris Tanner



# Outline

---

Why RNNs

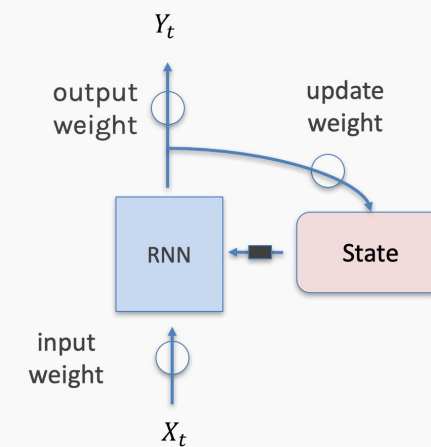
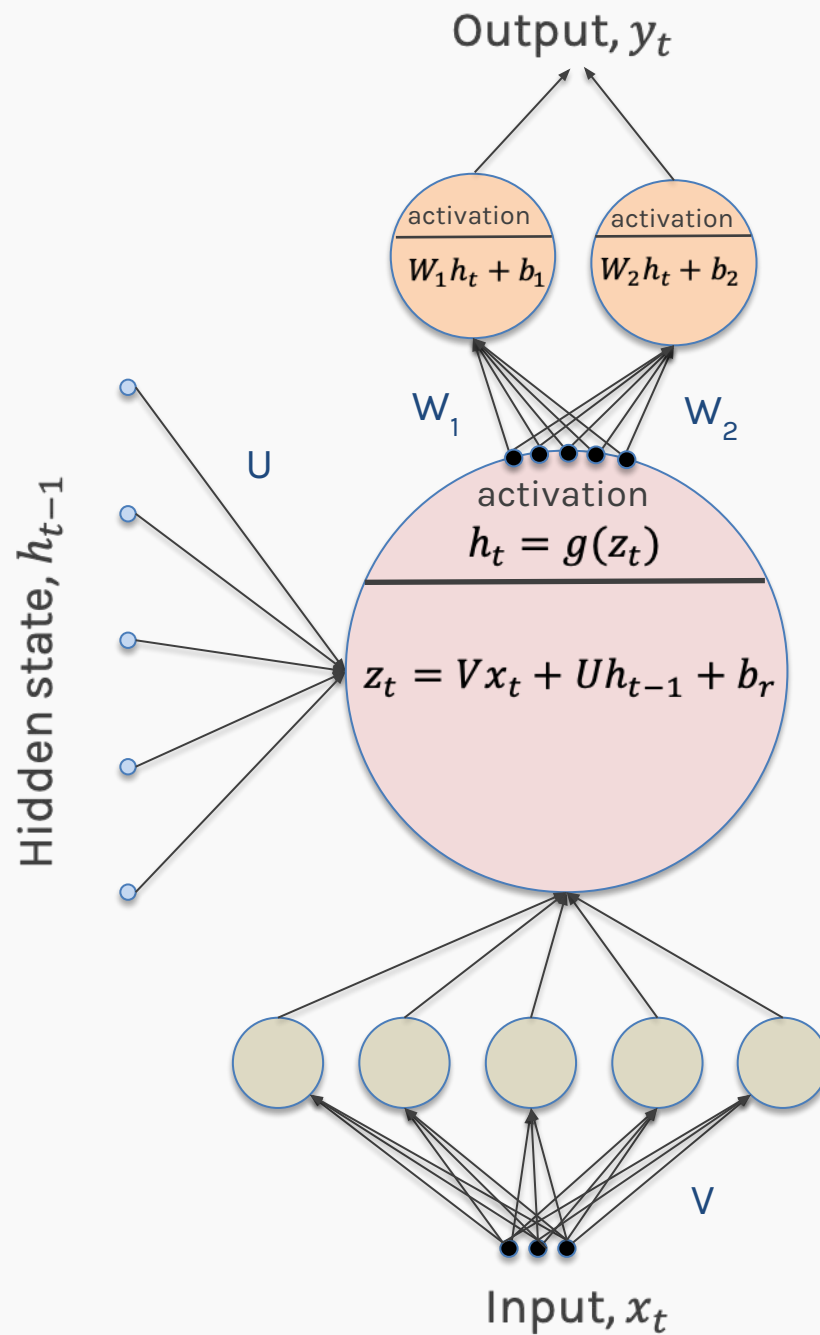
**Main Concept of RNNs (part 1)**

More Details of RNNs

RNN training

Gated RNN

# Anatomy of an RNN unit



# Long-term Dependencies

---

Unfolded networks can be very deep

Long-term interactions are given exponentially smaller weights than small-term interactions

Gradients tend to either *vanish* or *explode*

# Long Short-Term Memory

---

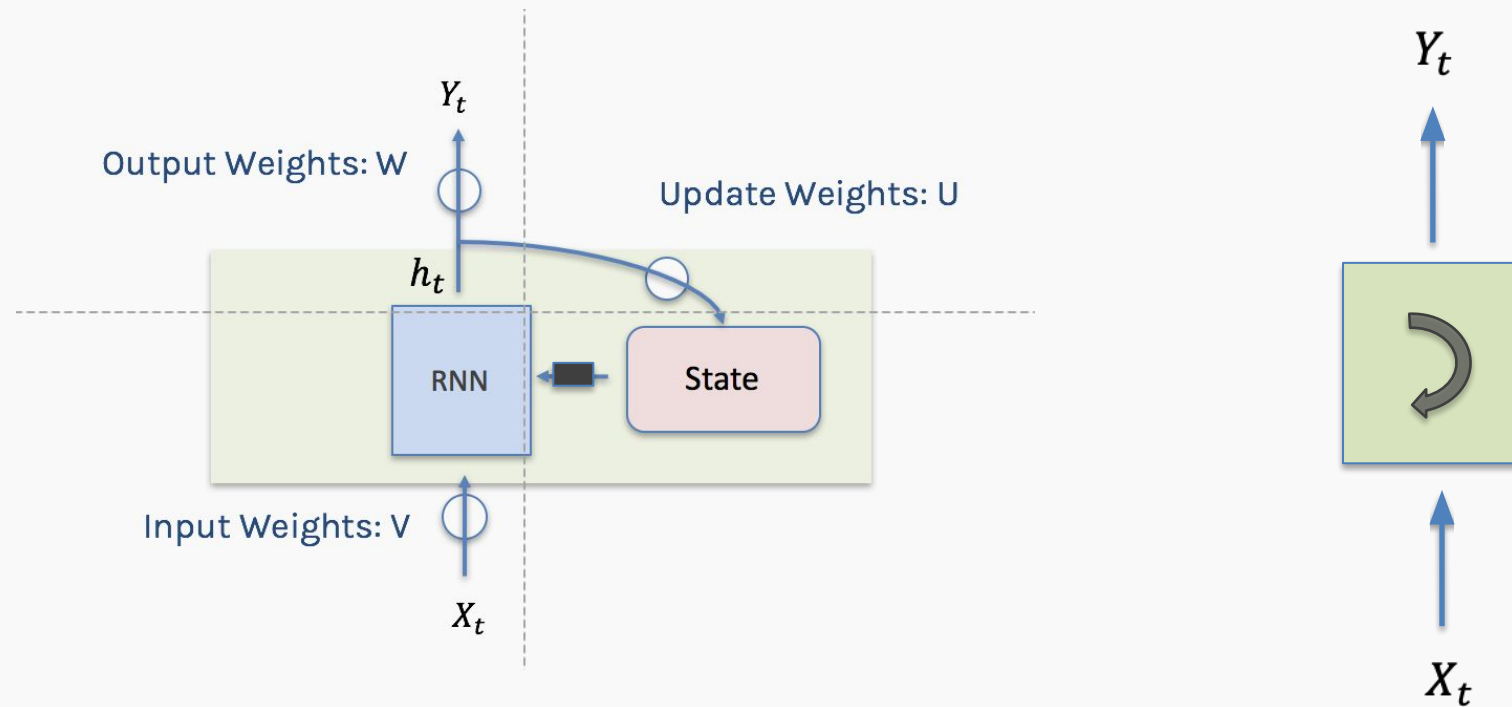
Handles long-term dependencies

Leaky units where weights on self-loop  $\alpha$  is context-dependent

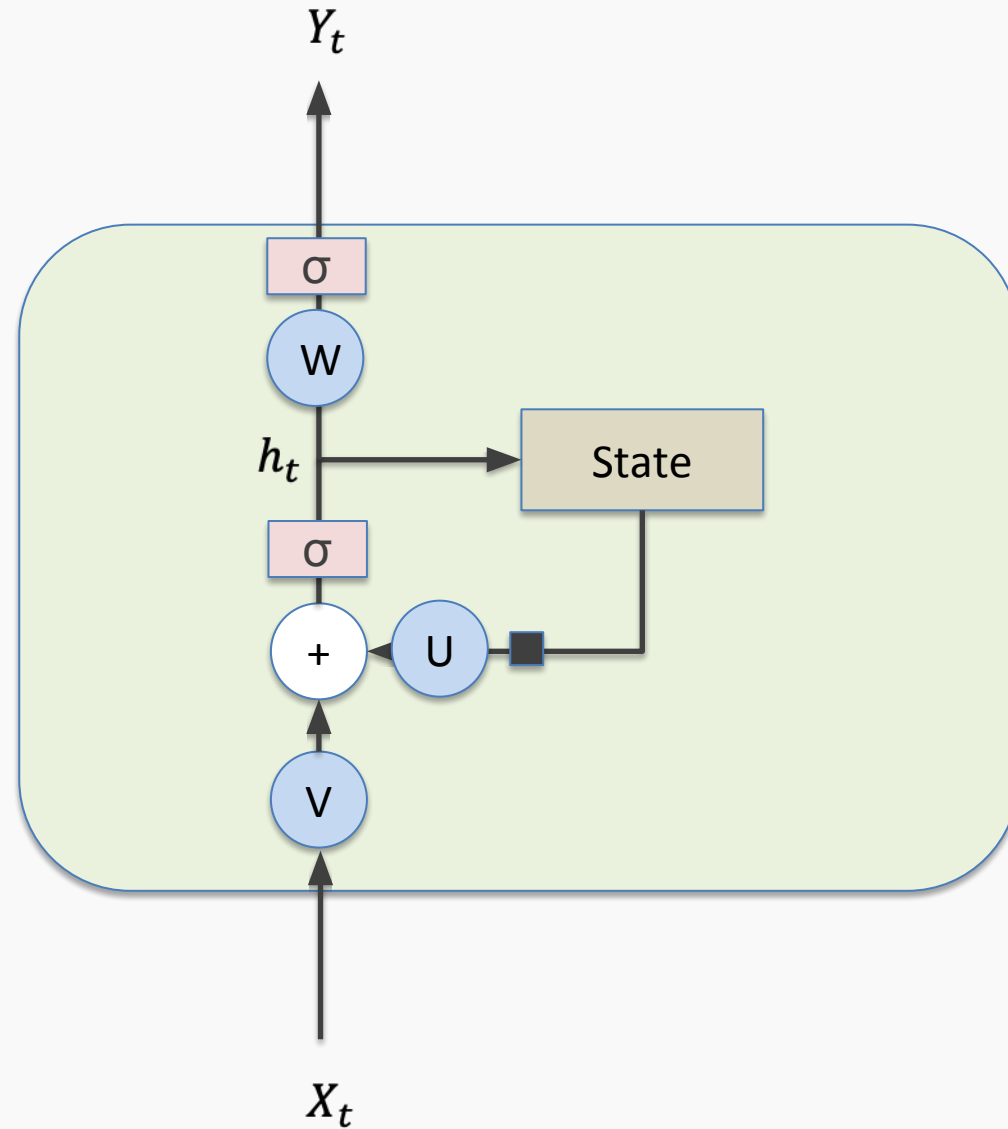
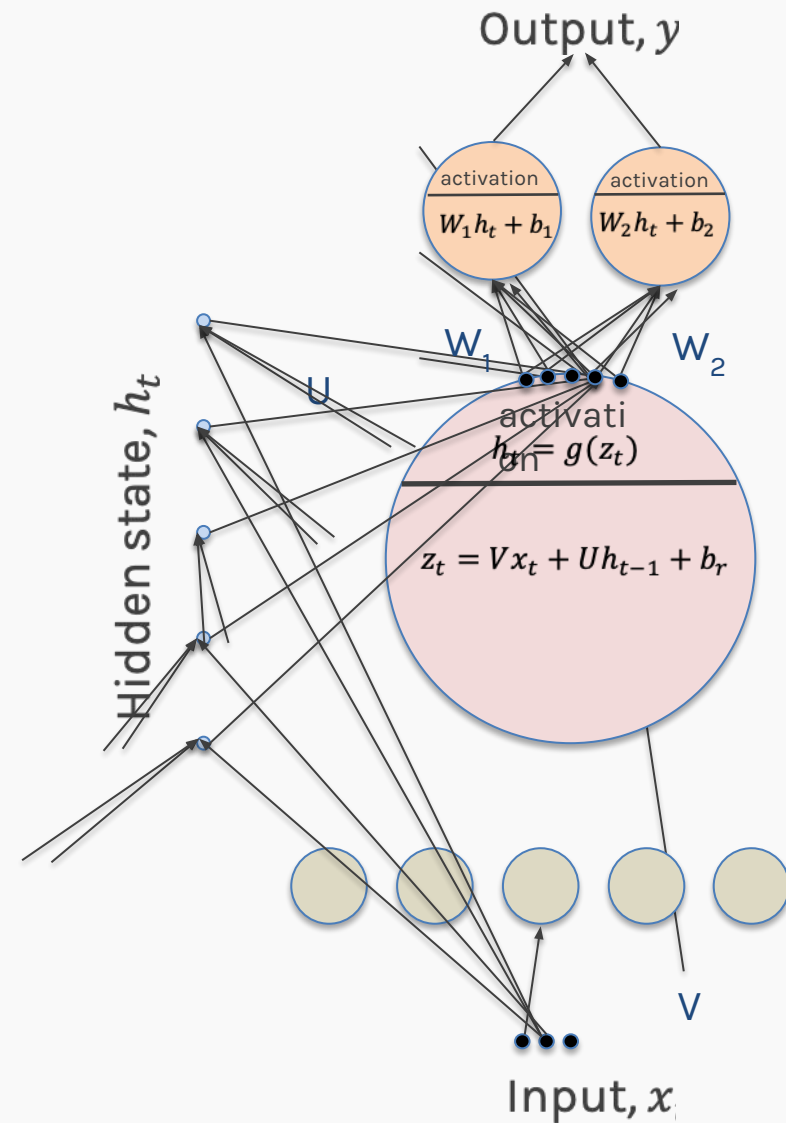
Allow network to decide whether to accumulate or forget past info

# Notation

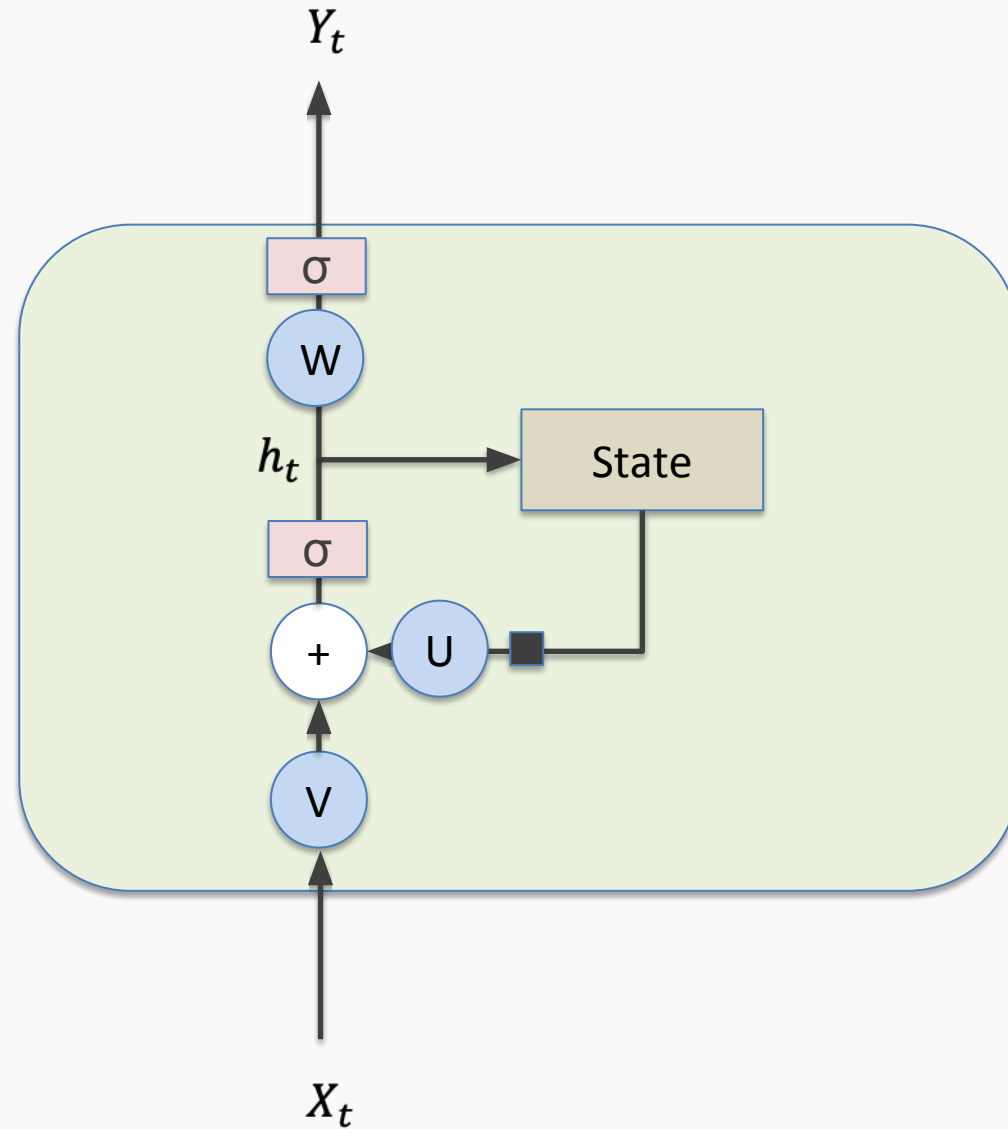
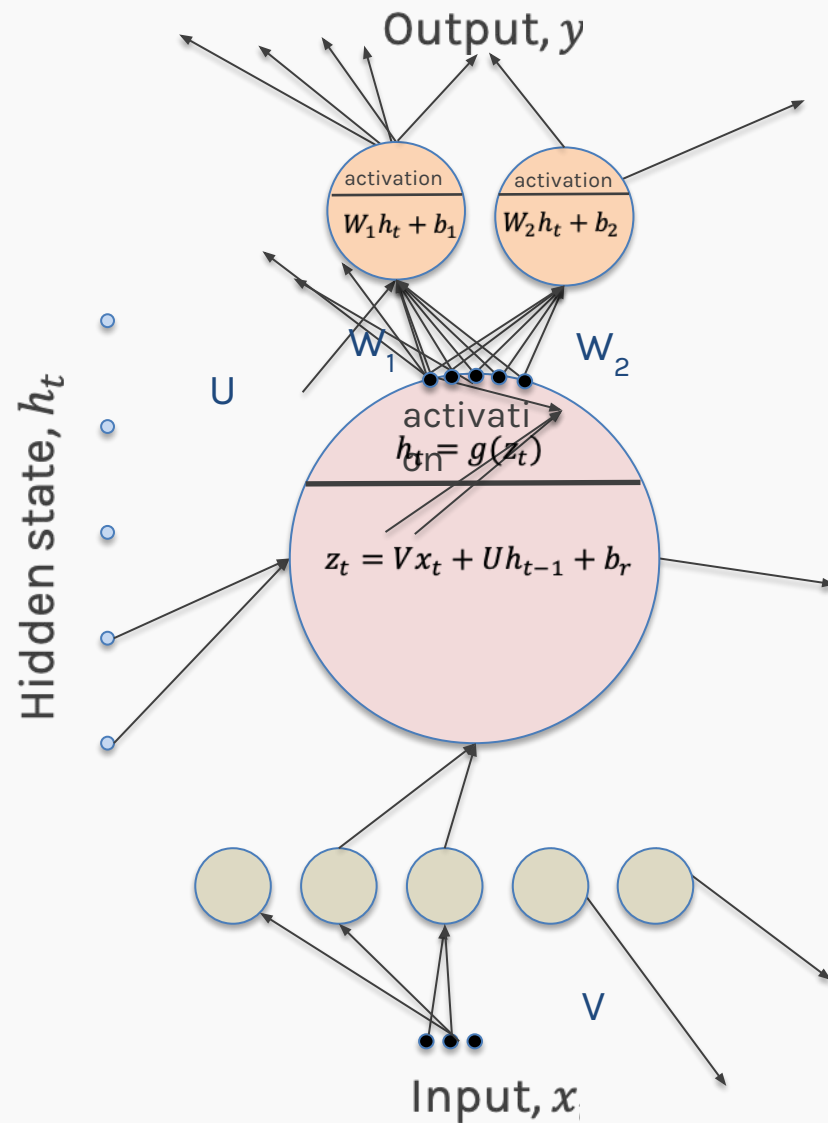
Using conventional and convenient notation



# Simple RNN again

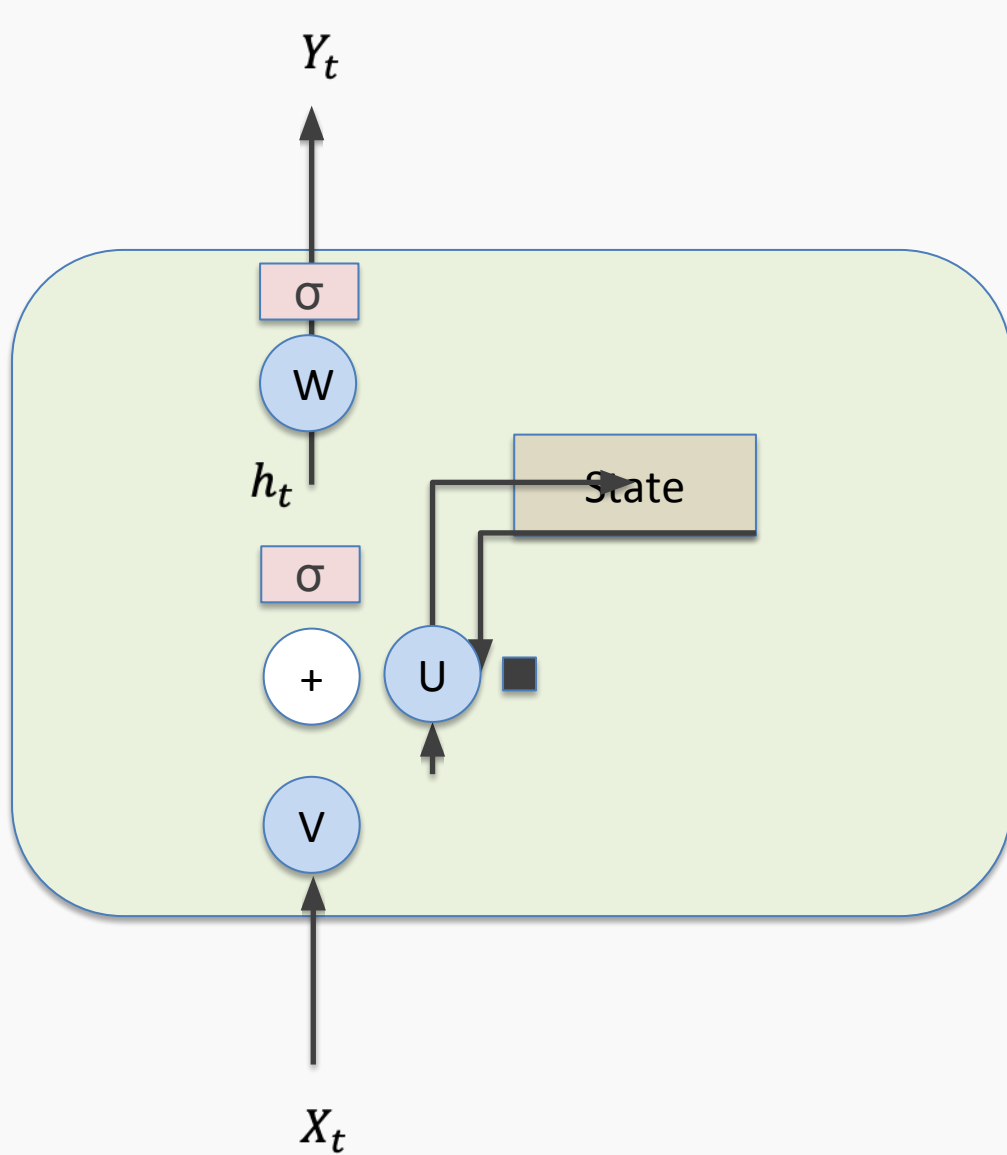


# Simple RNN again

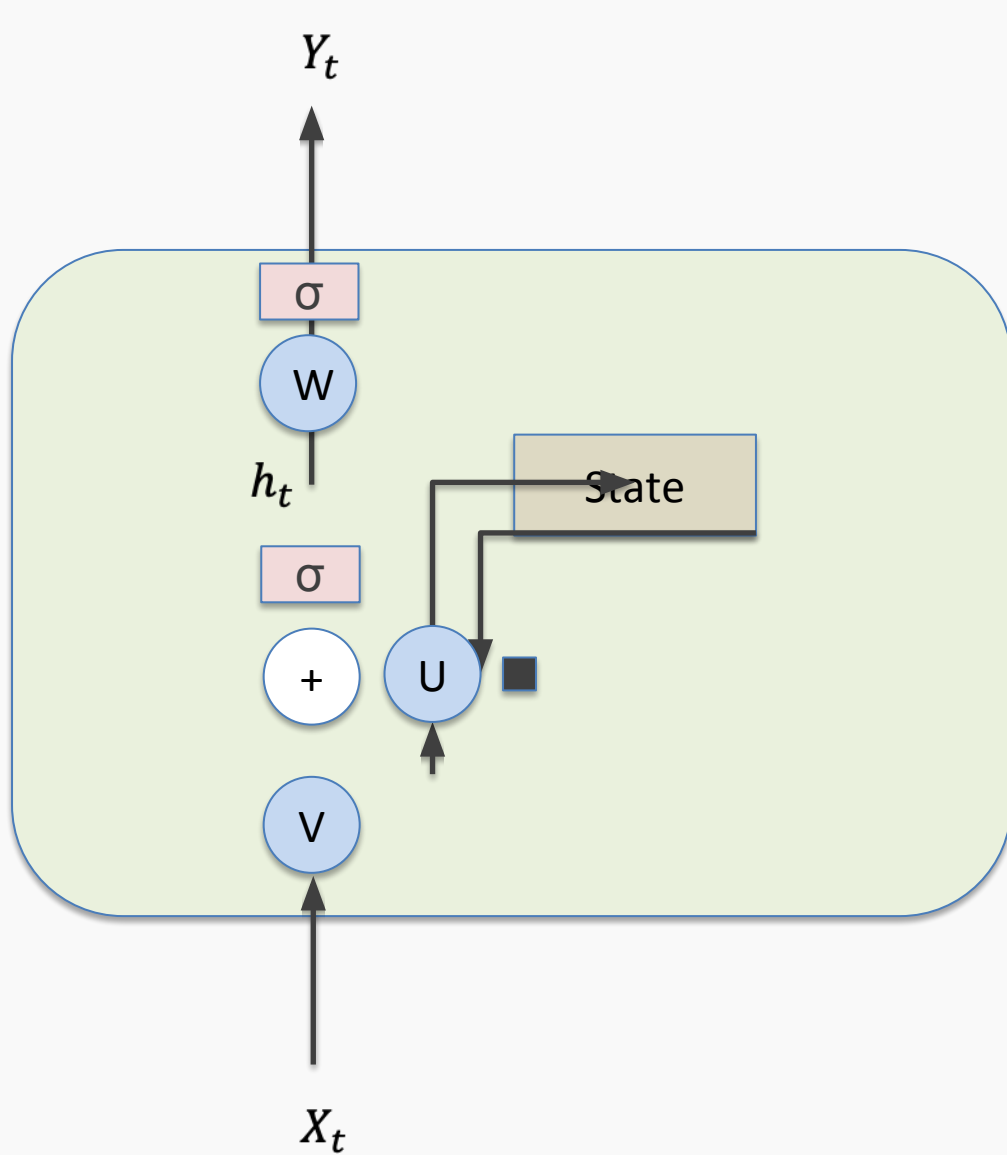




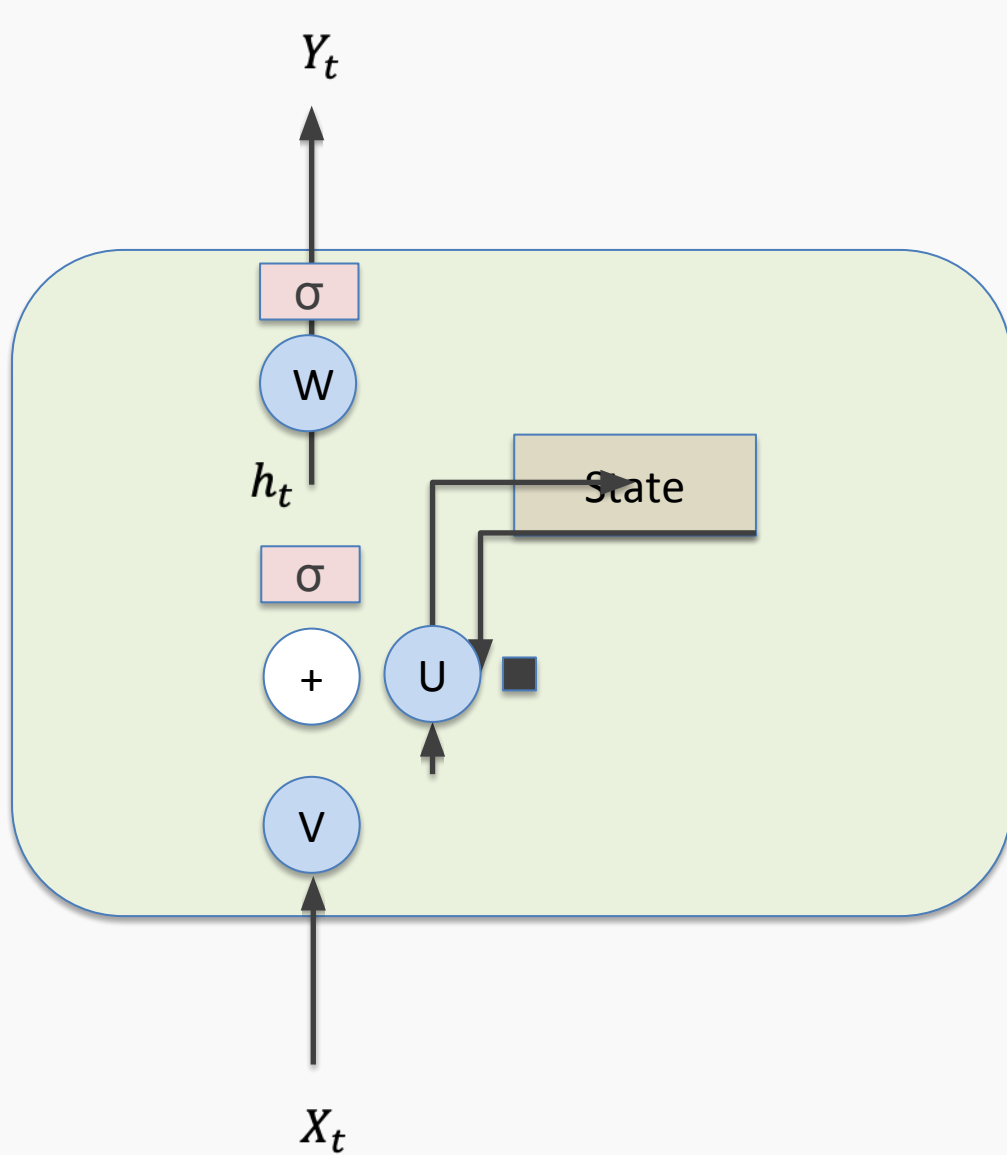
# Simple RNN again: **Memories**



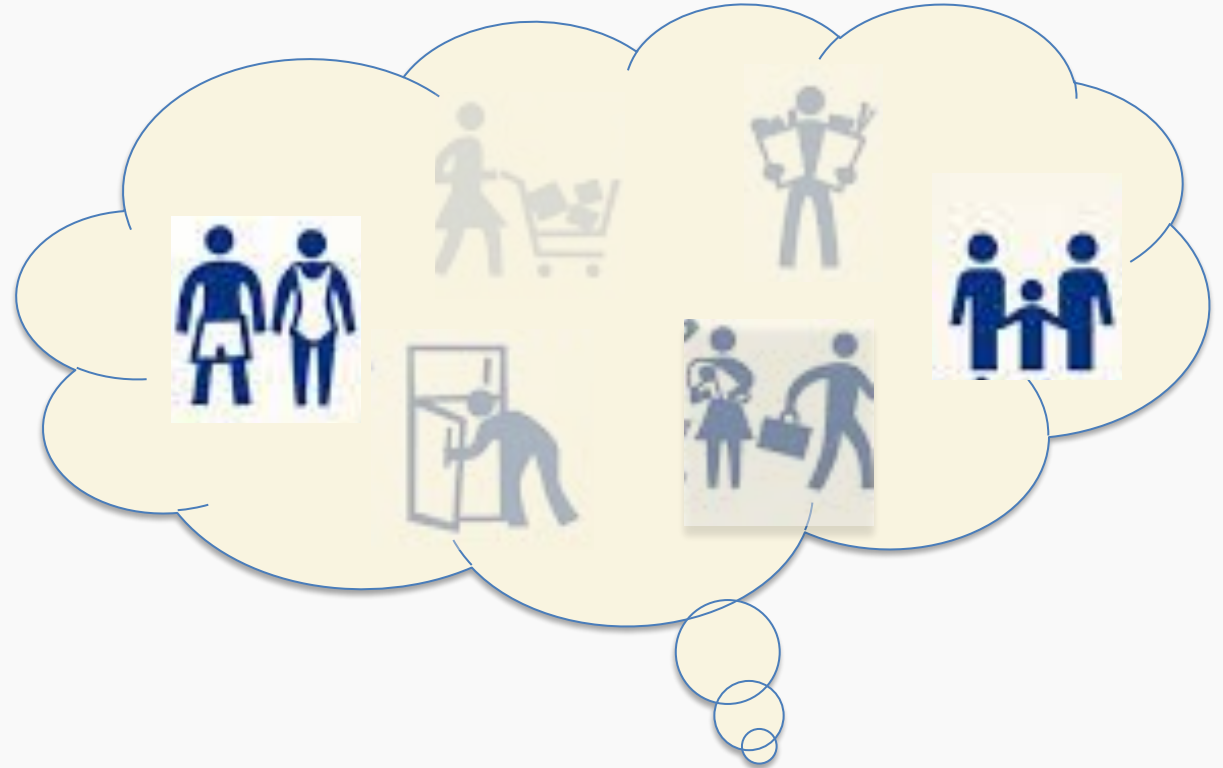
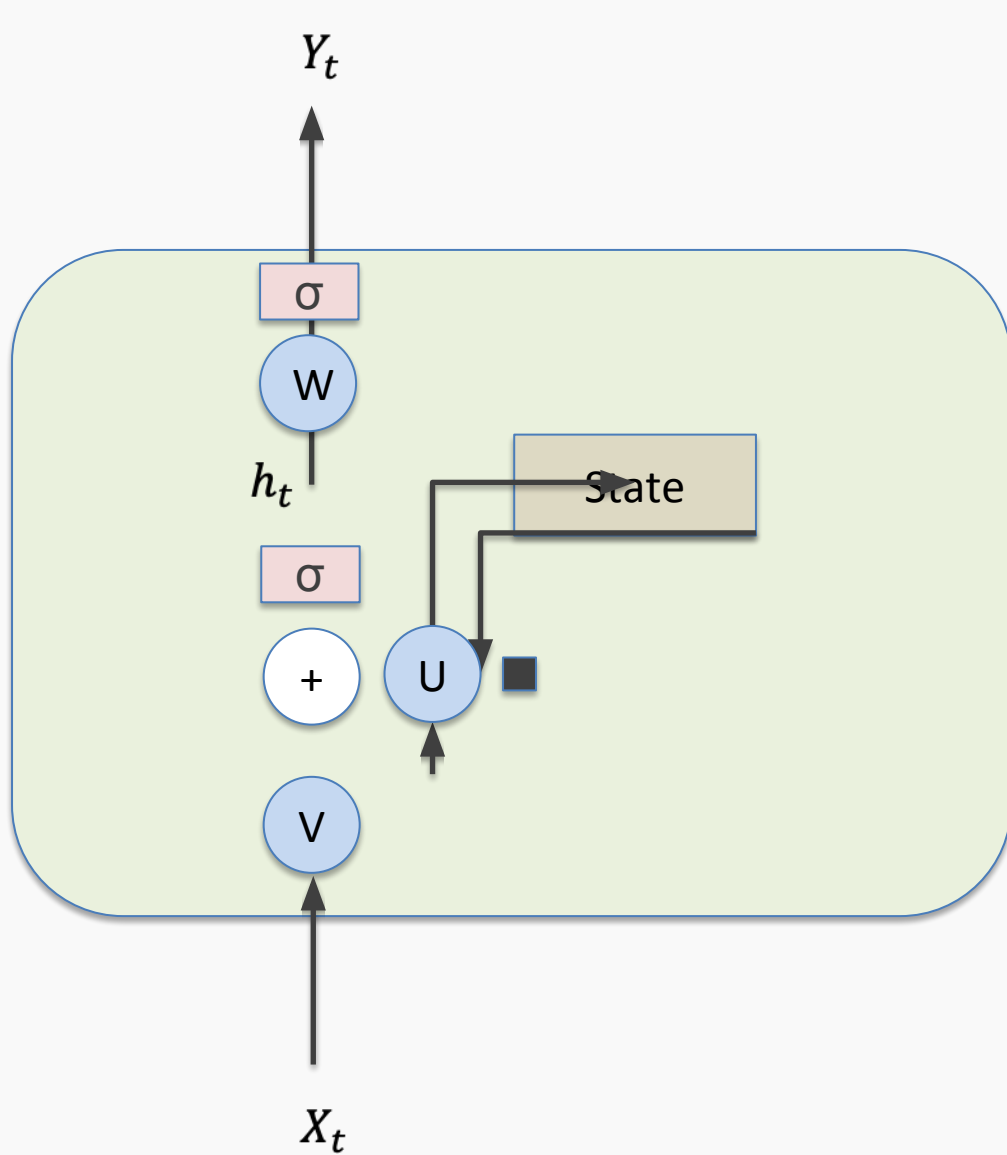
# Simple RNN again: Memories - Forgetting



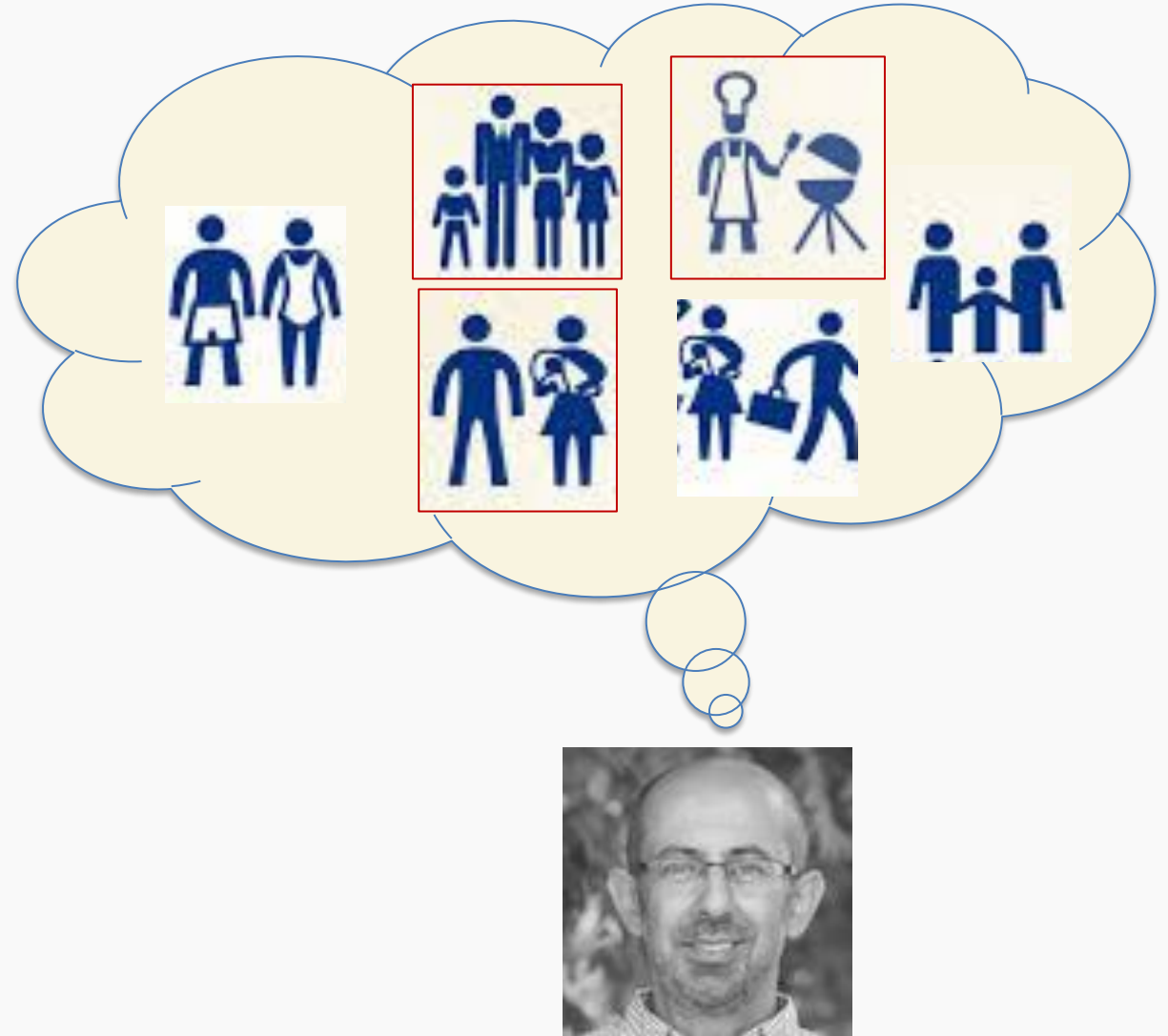
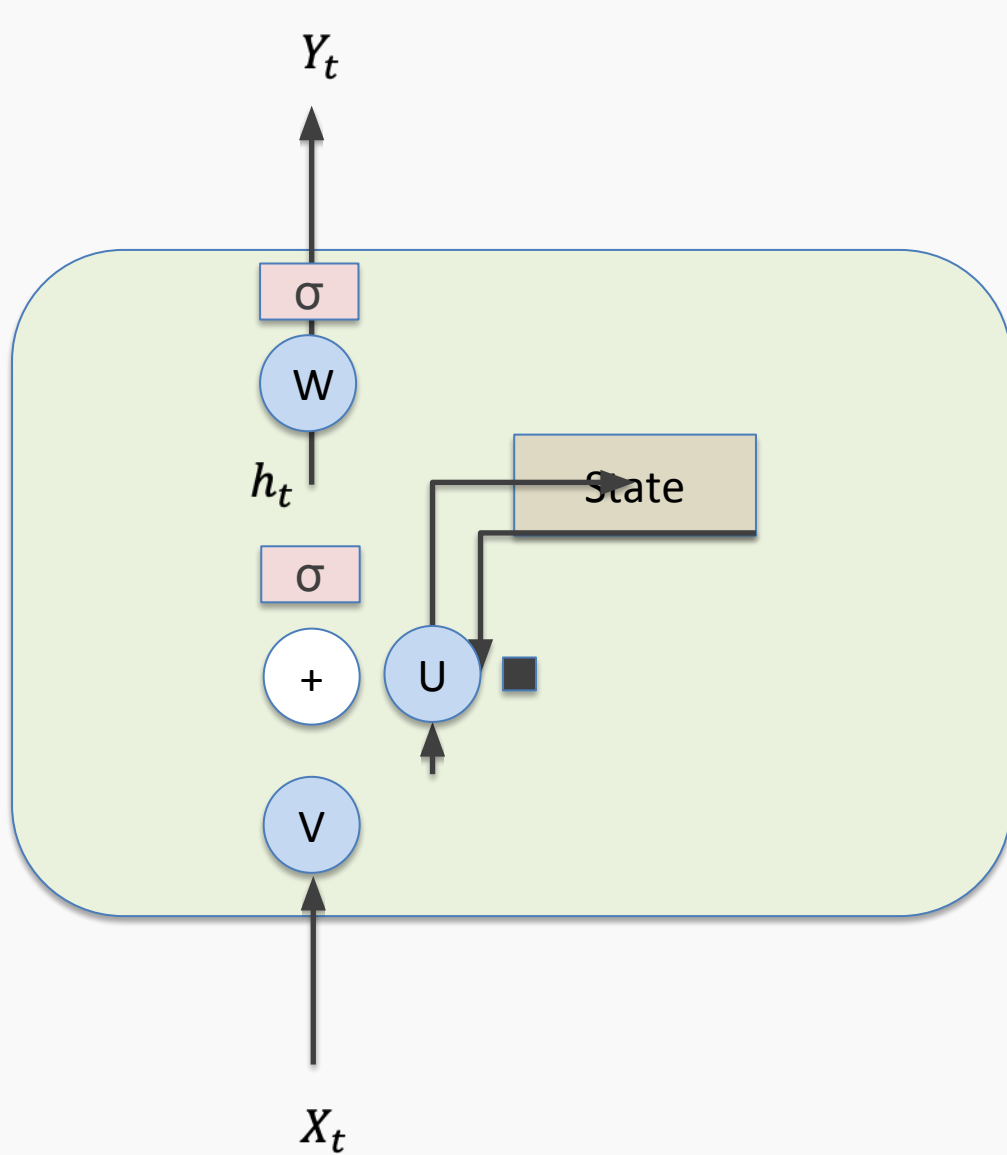
# Simple RNN again: **New Events**



# Simple RNN again: New Events Weighted



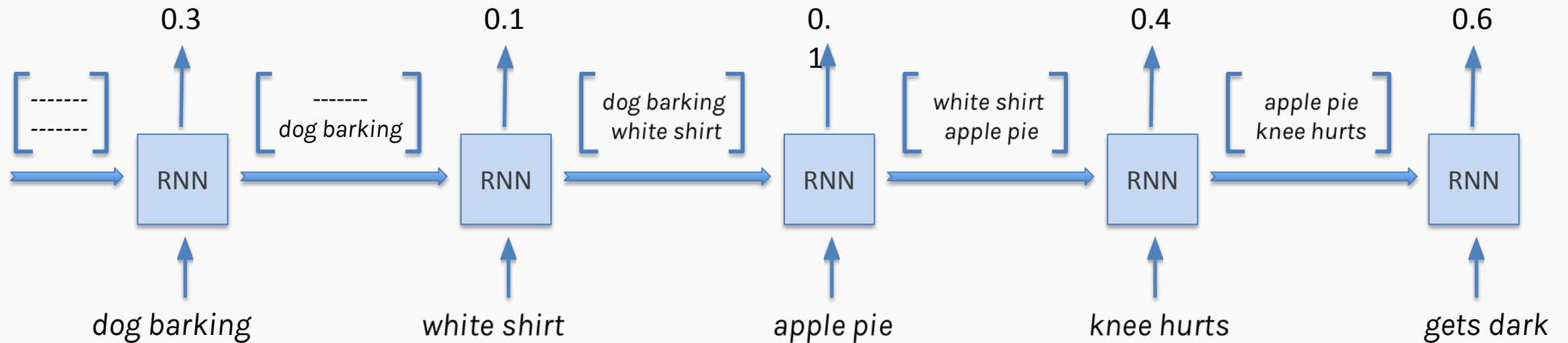
# Simple RNN again: Updated memories



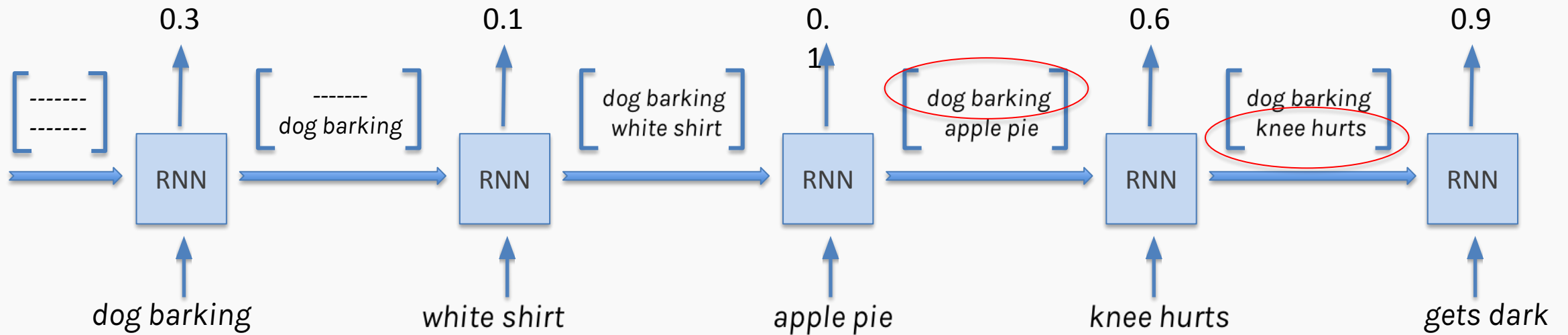
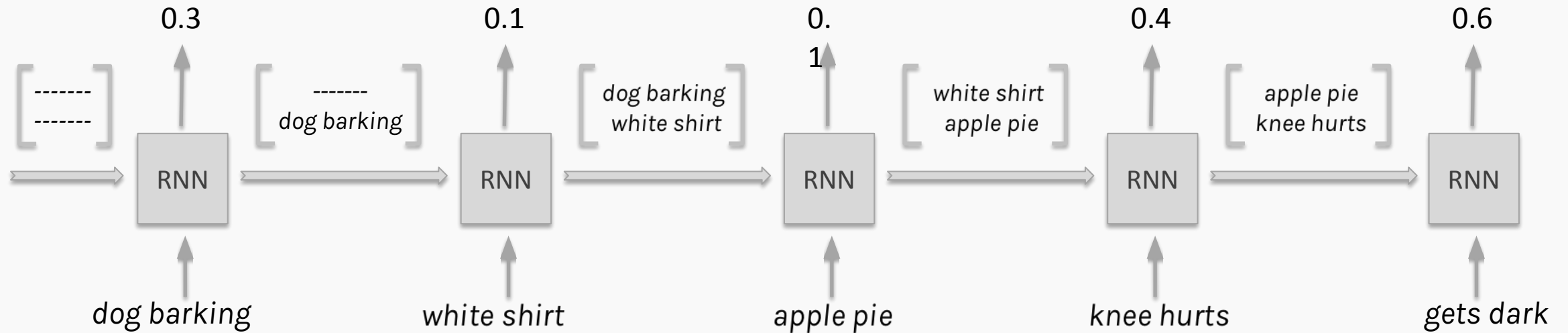
# RNN + Memory

A naïve way of remembering/forgetting and updating is to remember the latest  $n$  inputs.

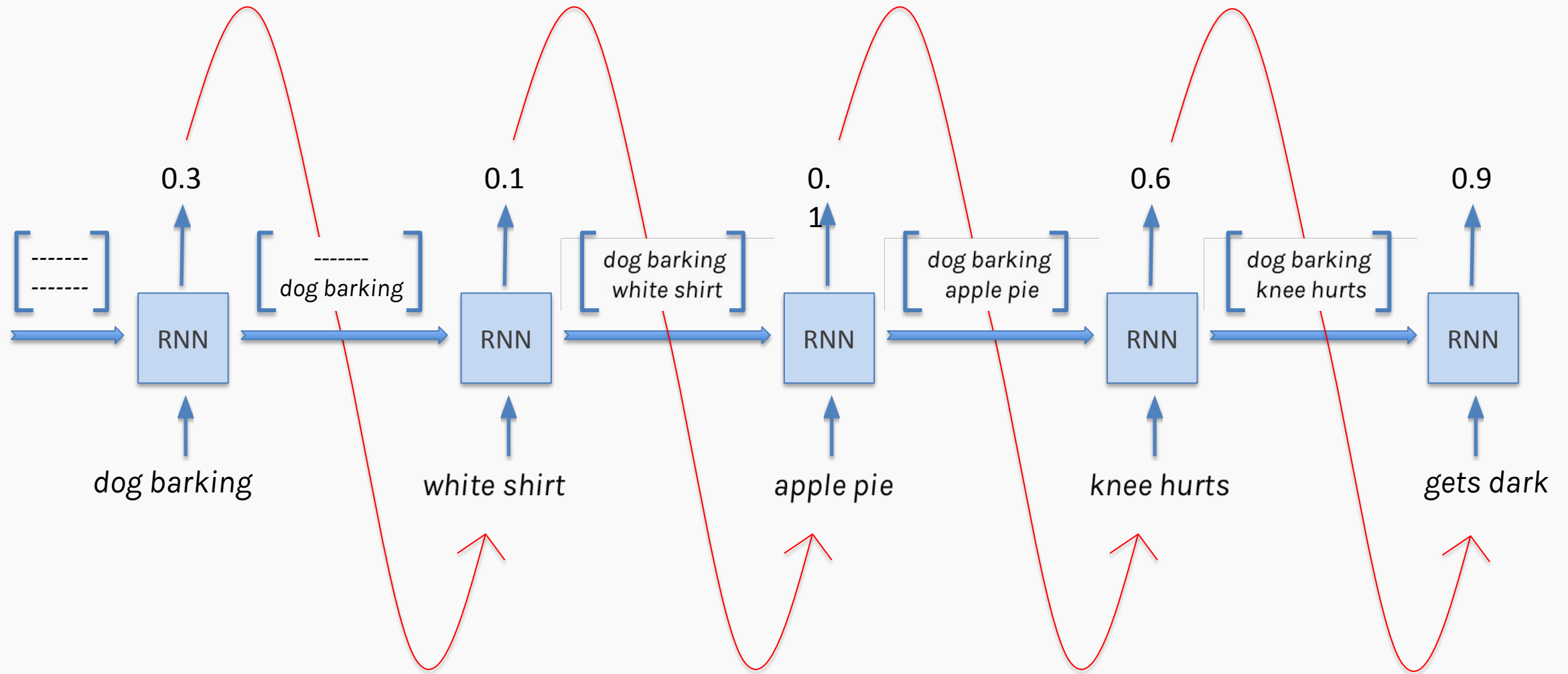
**Example:** Is it raining? Build an RNN to estimate the probability of raining:



## RNN + Memory (cont) – A better scheme



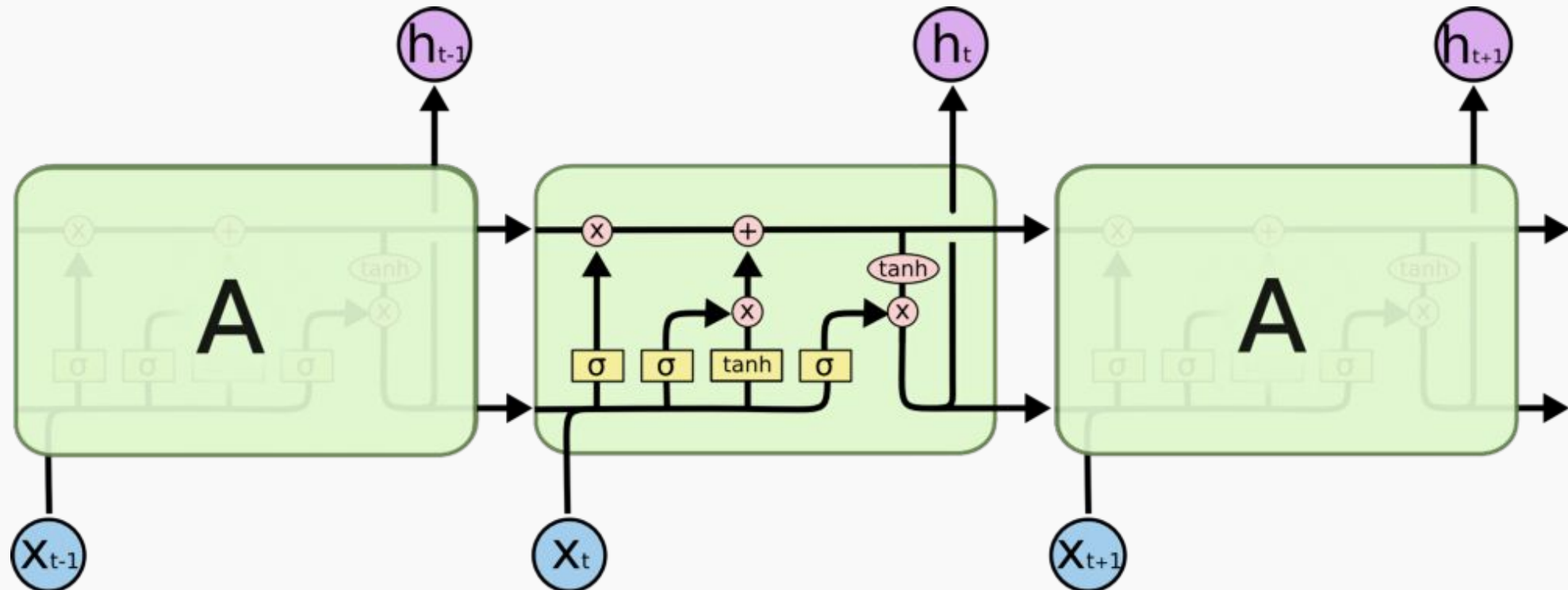
# RNN + Memory + Output





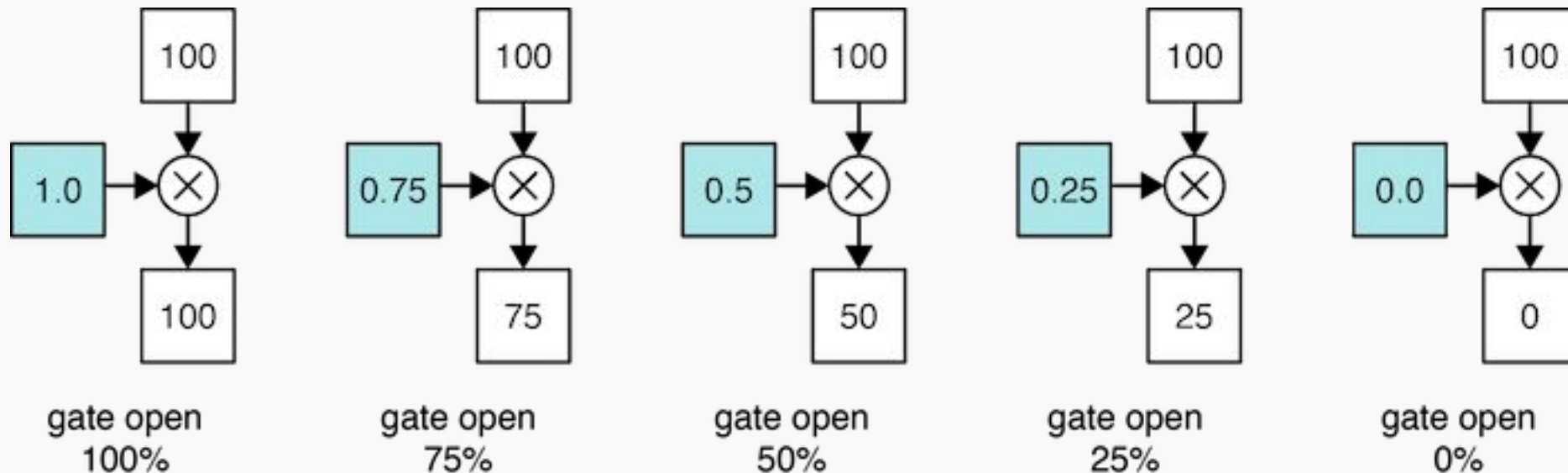
# LSTM: Long short term memory

It looks kind of complicated but we will **dissect** it.



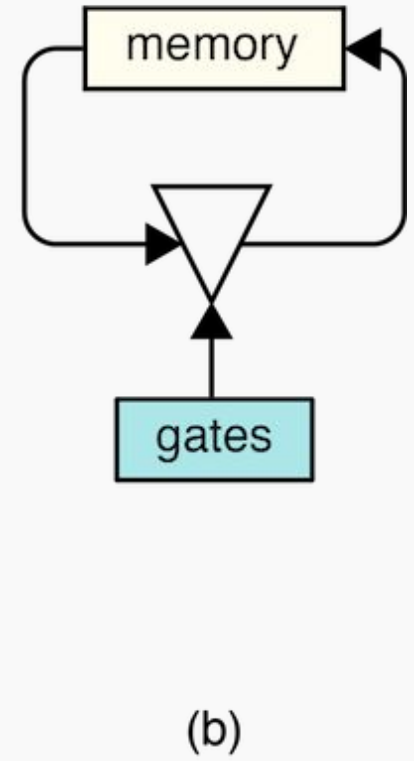
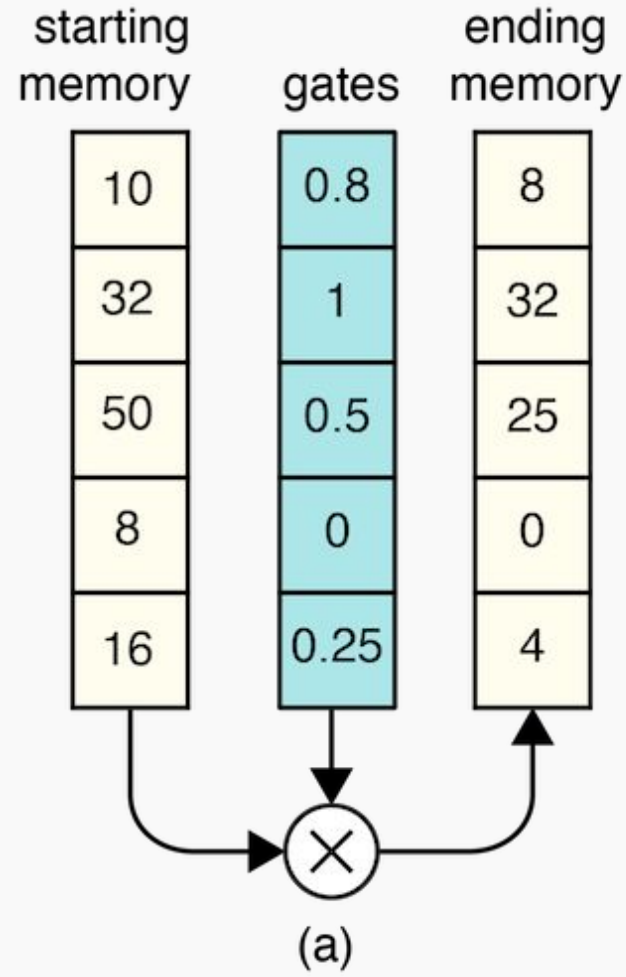
# Gates

A key idea in the LSTM is a mechanism called a gate.



# Forgetting

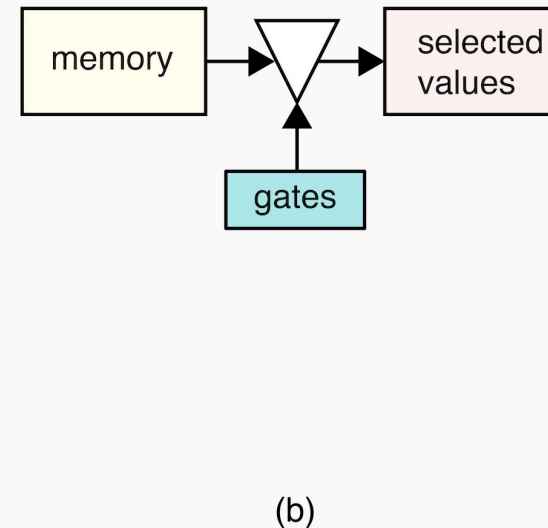
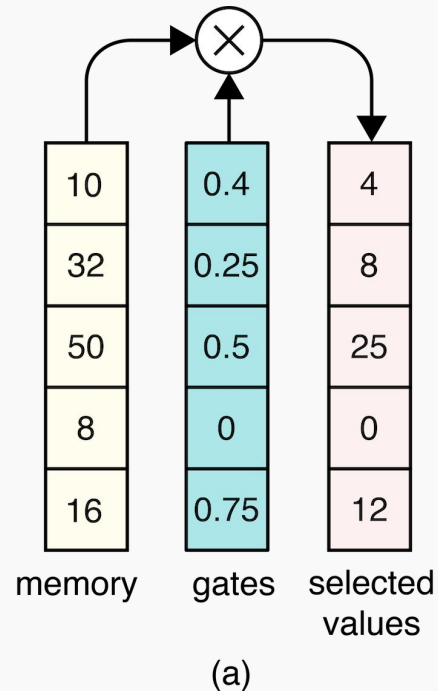
Each value is multiplied by a gate, and the result is stored back into the memory.



# Remembering

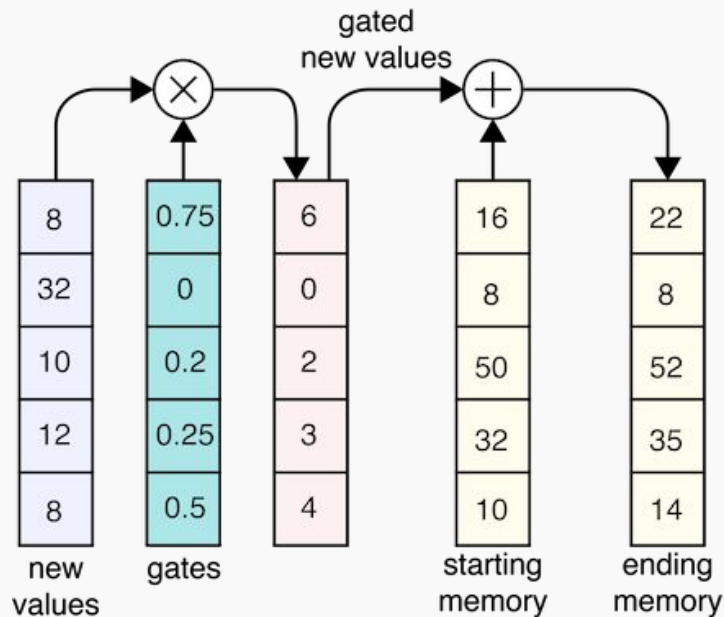
Remembering involves two steps.

1. We determine how much of each new value we want to remember and we use gates to control that.
2. Remember the gated values, we merely add them in to the existing contents of the memory.

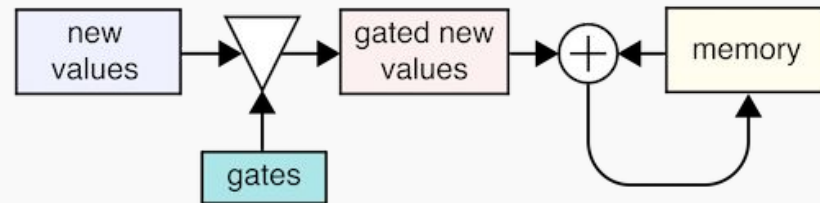


# Updating

To select from memory we just determine how much of each element we want to use, we apply gates to the memory elements, and the results are a list of scaled memories.



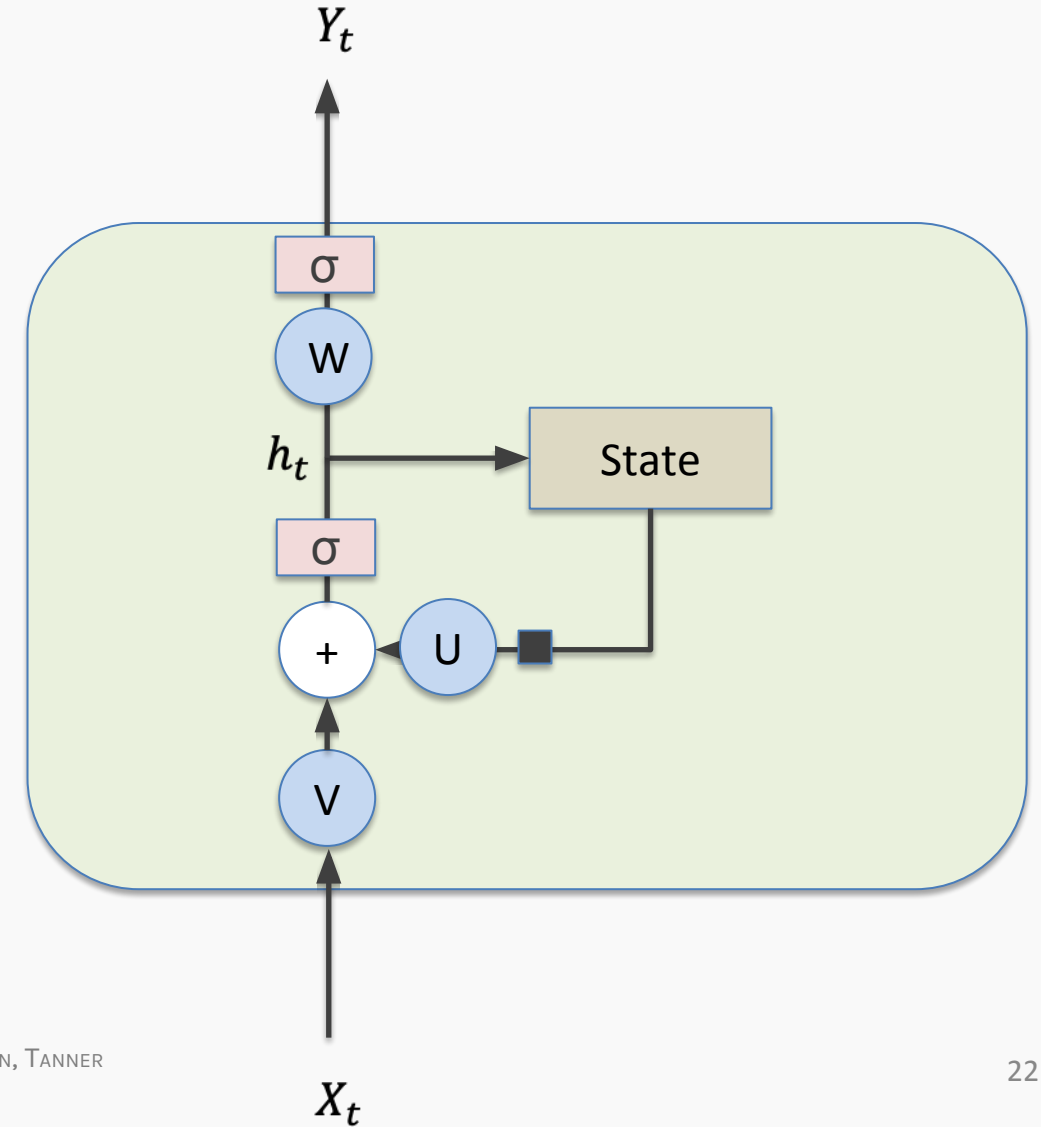
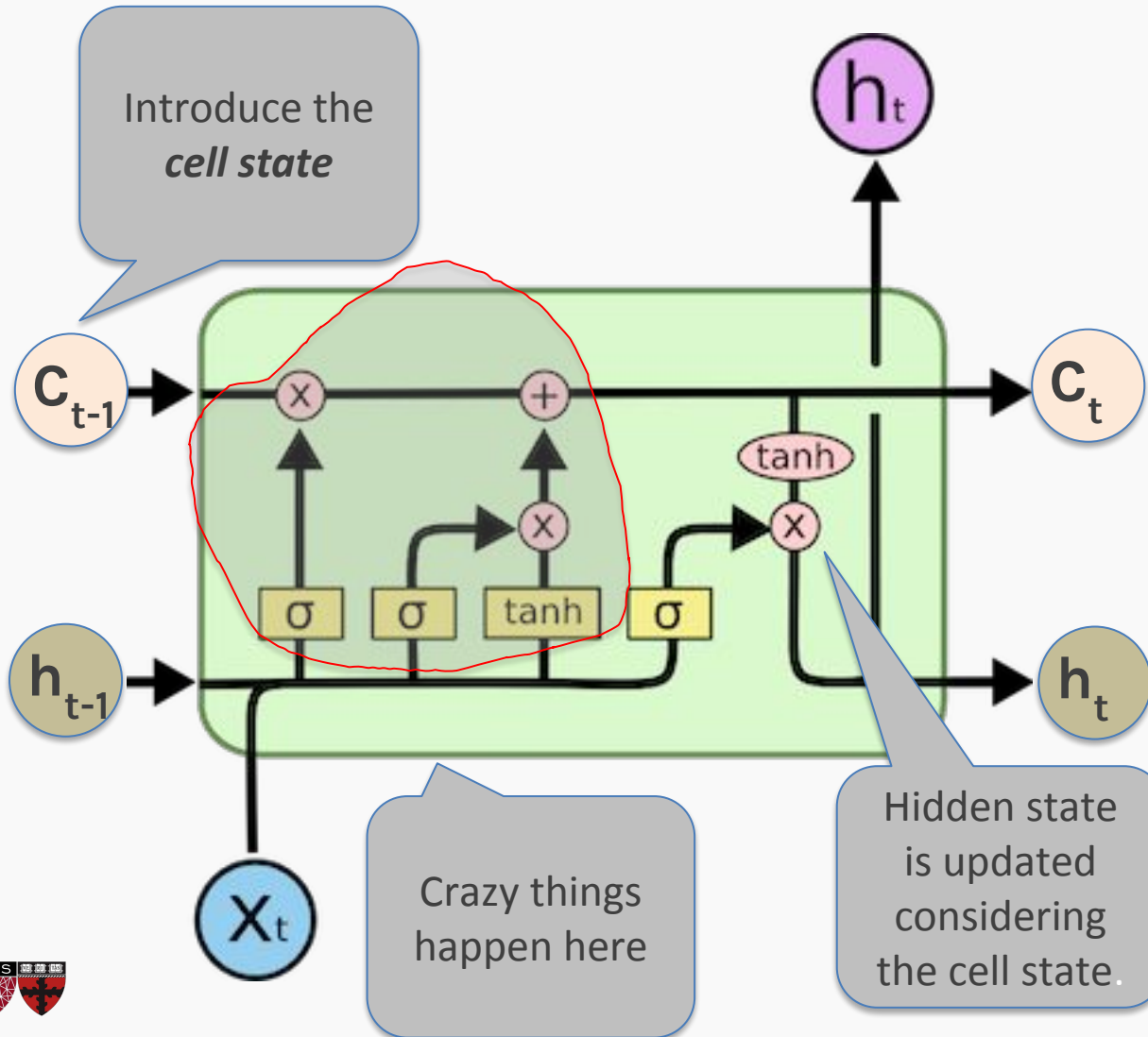
(a)



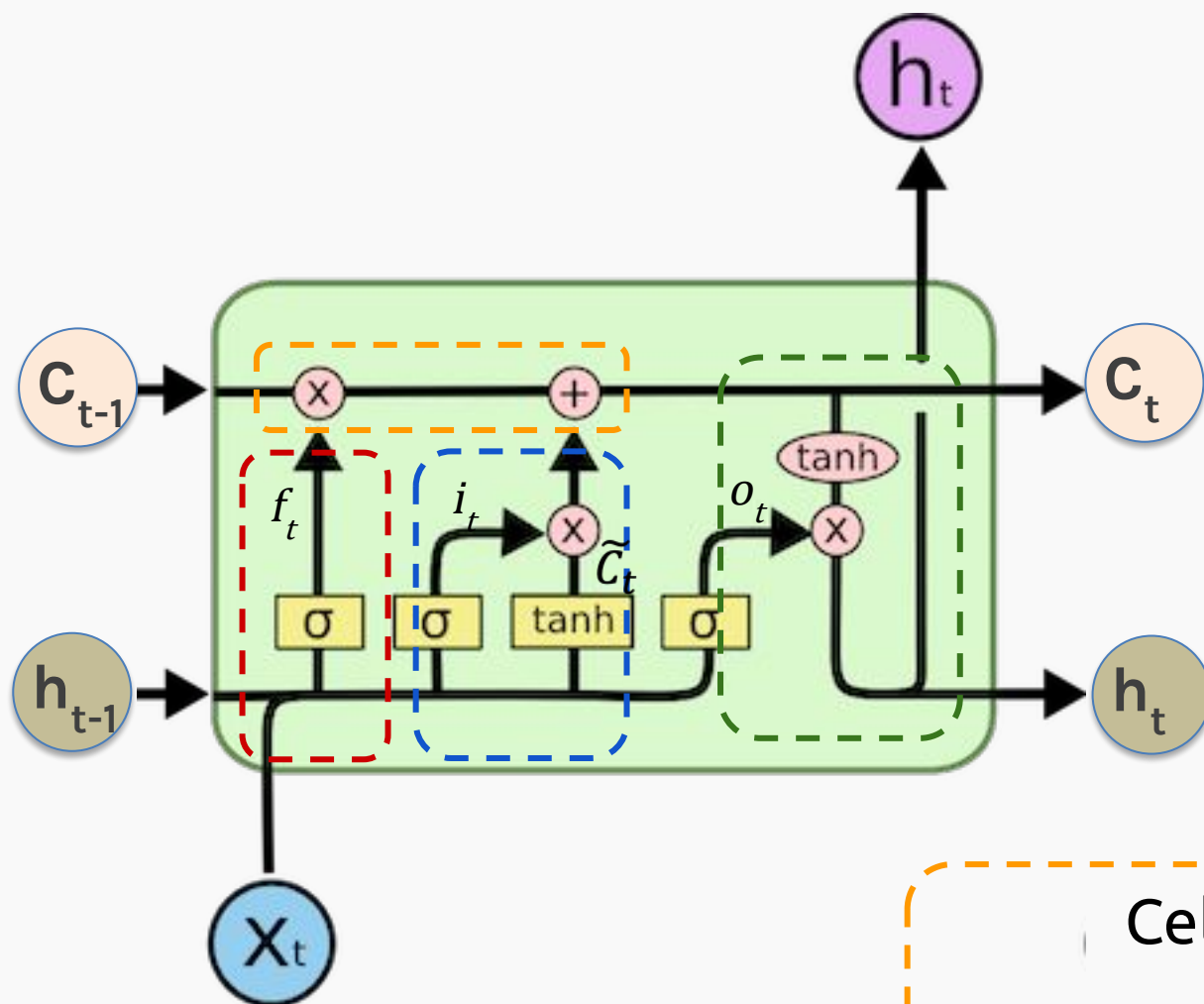
(b)

# LSTM

LSTM again. Find the differences compared to a simple RNN.



Before we dive in to really understand LSTM, lets see the big picture ...



### Forget Gate

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

### Output Gate

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

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

### Input Gate

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

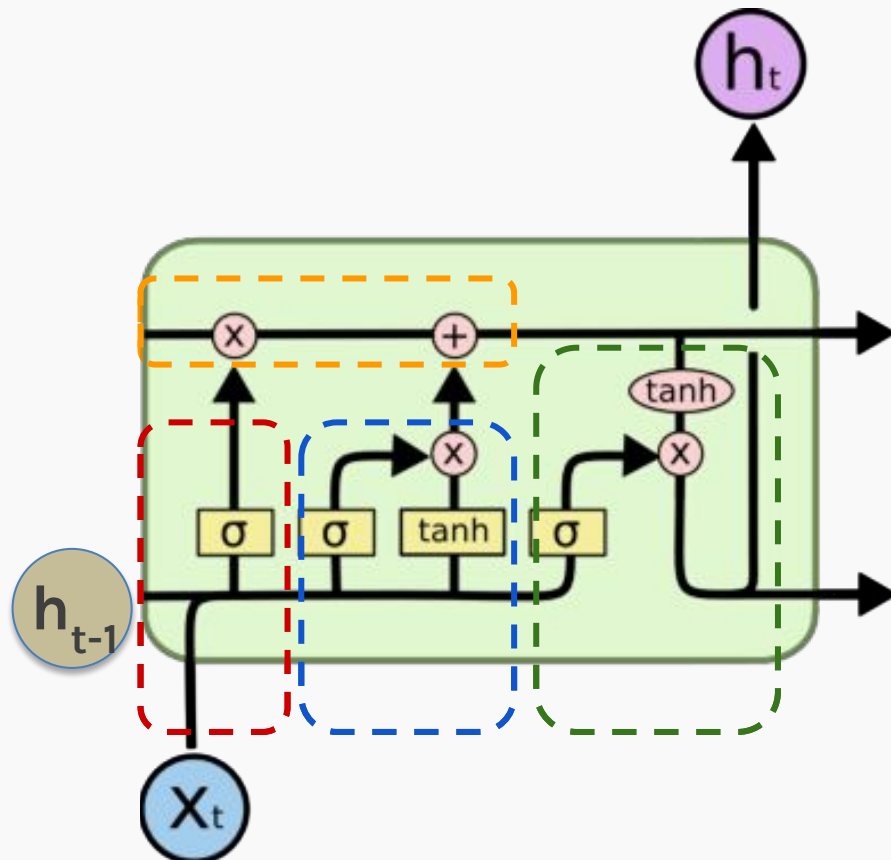
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

### Cell State

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

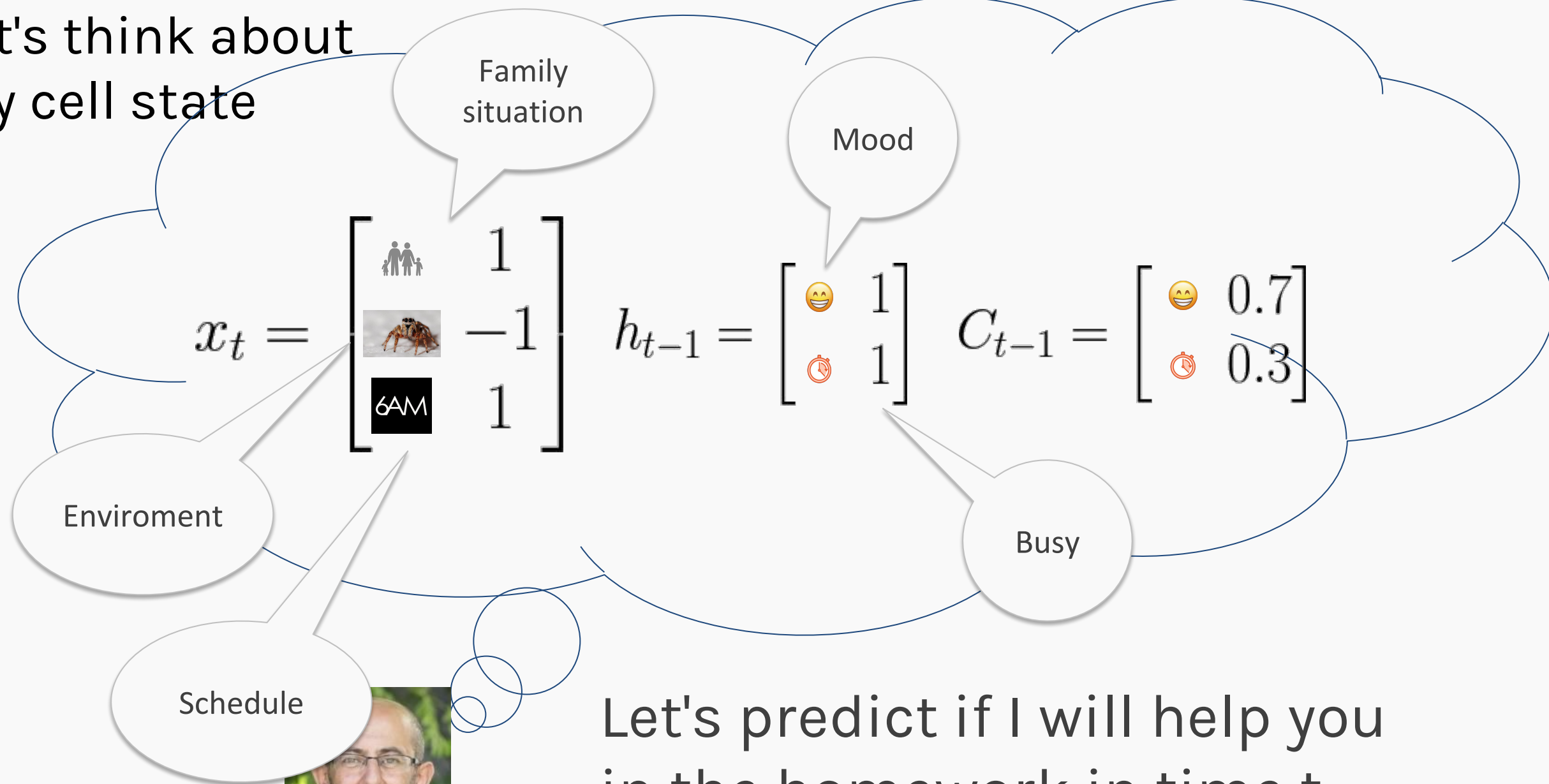
# LSTM: Big picture

1. LSTMs are recurrent neural network with a **cell** and a **hidden** state, both of these are updated in each step and can be thought as memories.
2. **Cell** states work as a **long term memory** and it is updated based on the relation between the hidden state at  $t-1$  ( $h_{t-1}$ ) and the input at  $t$  ( $x_t$ ).
3. The **hidden** state of the next step is a transformation of the cell state and the





# Let's think about my cell state

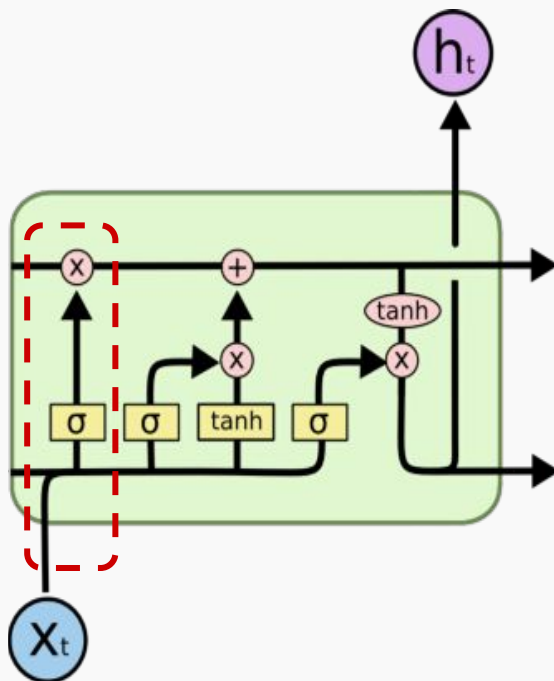


## Let's predict if I will help you in the homework in time t.



## Forget Gate

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



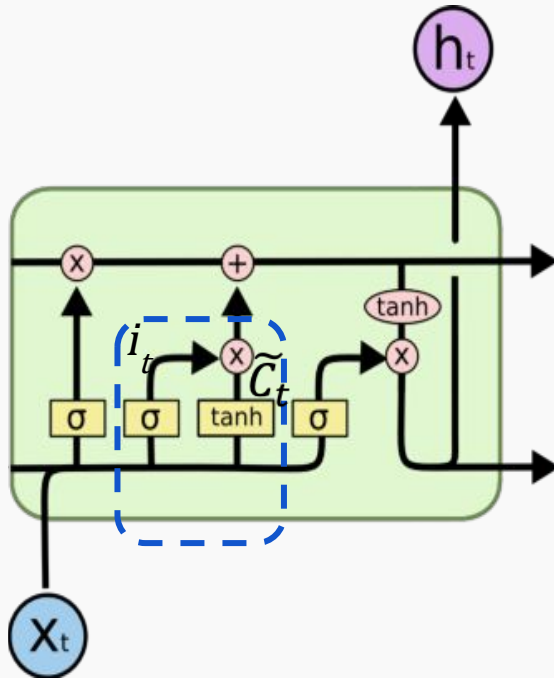
The forget gate tries to estimate what features of the cell state should be forgotten.

$$f_t = \sigma \left( \underbrace{\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \text{😊} & 1 \\ \text{🚫} & 1 \end{bmatrix}}_{\begin{bmatrix} 0 \\ 0 \end{bmatrix}} + \underbrace{\begin{bmatrix} 1 & 1 & -100 \\ 0.1 & 0.1 & -100 \end{bmatrix} \begin{bmatrix} \text{👨👩👧} & 1 \\ \text{🐘} & -1 \\ \text{6AM} & 1 \end{bmatrix}}_{\begin{bmatrix} -100 \\ -100 \end{bmatrix}} \right)$$

Erase  
everything!

## Input Gate

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_f) \end{aligned}$$



The input gate layer works in a similar way as the forget layer. The input gate layer estimates the degree of confidence of  $\tilde{C}_t$  and  $i_t \times \tilde{C}_t$  is a new estimation of the cell state

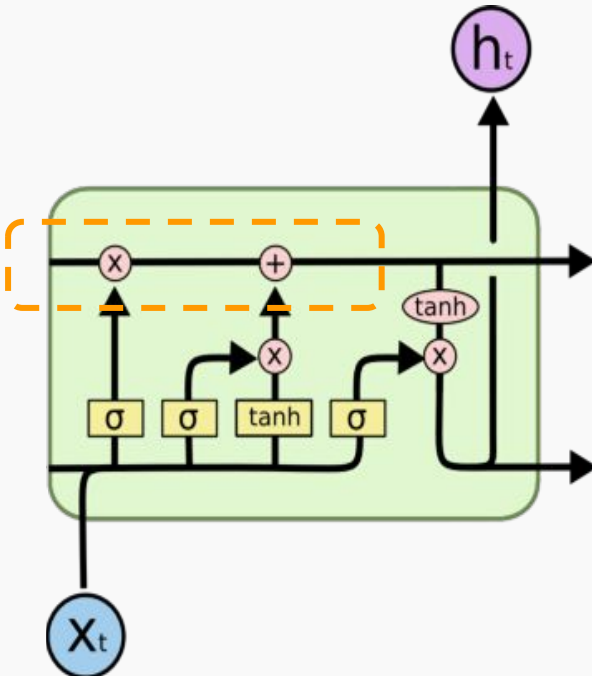
Let's say that my input gate estimation is:  $i_t = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$$\tilde{C}_t = \tanh \left( \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \text{😊} & 1 \\ \text{🕒} & 1 \end{bmatrix} + \begin{bmatrix} 10 & 1 & -1 \\ -1 & 1 & 10 \end{bmatrix} \begin{bmatrix} \text{👨👩👦} & 1 \\ \text{🦋} & -1 \\ \text{6AM} & 1 \end{bmatrix} \right)$$

$$\tilde{C}_t = \tanh \left( \underbrace{\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 10 \\ 10 \end{bmatrix}}_{\begin{bmatrix} \text{😊} & 1 \\ \text{🕒} & 1 \end{bmatrix}} \right)$$

## Cell state

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



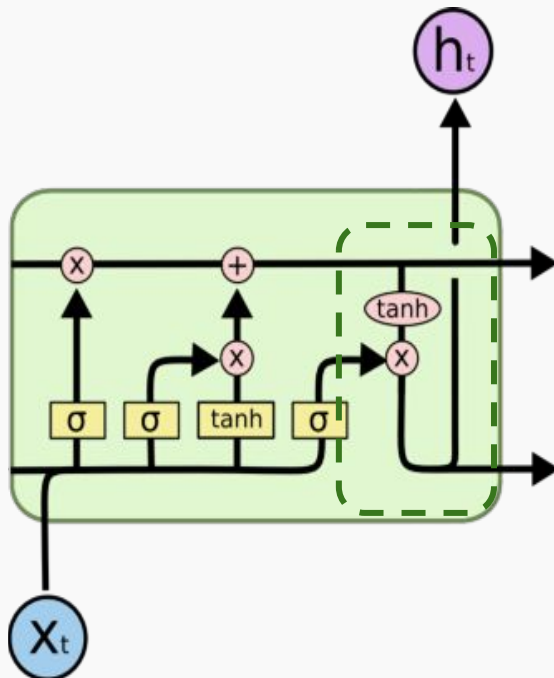
After the calculation of forget gate and input gate we can update our cell state.

$$C_t = \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix} * \begin{bmatrix} \text{😄} & 0.7 \\ \text{🕒} & 0.3 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} * \begin{bmatrix} \text{😄} & 1 \\ \text{🕒} & 1 \end{bmatrix}}_{C_t = \begin{bmatrix} \text{😄} & 1 \\ \text{🕒} & 1 \end{bmatrix}}$$

## Output gate

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

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



- The output gate layer is calculated using the information of the input at time  $t$  ( $x_t$ ) and hidden state of the last step ( $h_{t-1}$ ).
- It is important to notice that hidden state used in the next step is obtained using the output gate layer, which is usually the function that we optimize.

$$o_t = \sigma \left( \underbrace{\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} \text{😊} & 1 \\ \text{🕒} & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \text{👤} & 1 \\ \text{🐜} & -1 \\ \text{6AM} & 1 \end{bmatrix}}_{o_t \approx 0.9} \right)$$

$$h_t \approx 0.9 * \begin{bmatrix} \text{😊} & 1 \\ \text{🕒} & 1 \end{bmatrix} = \begin{bmatrix} \text{😊} & 0.9 \\ \text{🕒} & 0.9 \end{bmatrix}$$

wcct! = we can calculate this!

To optimize my parameters we basically need to do:  
Let's calculate all the derivatives in some time t!

$$W = W - \eta \frac{\partial \mathcal{L}}{\partial W}$$

$$\frac{\partial \mathcal{L}}{\partial W_f} = \frac{\partial \mathcal{L}}{\partial C^t} \underbrace{\frac{\partial C^t}{\partial f^t}}_{C^{t-1}} \underbrace{\frac{\partial f^t}{\partial W_f}}_{\text{wcct!}}$$

$$\frac{\partial \mathcal{L}}{\partial W_c} = \frac{\partial \mathcal{L}}{\partial C^t} \underbrace{\frac{\partial C^t}{\partial (i^t \odot \hat{C}^t)}}_1 \underbrace{\frac{\partial (i^t \odot \hat{C}^t)}{\partial W_c}}_{\text{wcct!}}$$

$$\frac{\partial \mathcal{L}}{\partial W_i} = \frac{\partial \mathcal{L}}{\partial C^t} \underbrace{\frac{\partial C^t}{\partial (i^t \odot \hat{C}^t)}}_1 \underbrace{\frac{\partial (i^t \odot \hat{C}^t)}{\partial W_i}}_{\text{wcct!}}$$

$$\frac{\partial \mathcal{L}}{\partial W_o} = \frac{\partial \mathcal{L}}{\partial h^t} \underbrace{\frac{\partial h^t}{\partial o^t}}_{\tanh C^t} \underbrace{\frac{\partial o^t}{\partial W_o}}_{\text{wcct!}}$$

Let's calculate the cell state and the hidden state

$$\frac{\partial \mathcal{L}}{\partial h^{t-1}} = \frac{\partial \mathcal{L}}{\partial C^t} \left( \frac{\partial C^t}{\partial f^t} \frac{\partial f^t}{\partial h^t} + \frac{\partial C^t}{\partial (i^t \odot \hat{C}^t)} \frac{\partial (i^t \odot \hat{C}^t)}{\partial h^t} \right) + \frac{\partial \mathcal{L}}{\partial h^t} \frac{\partial h^t}{\partial o^t} \frac{\partial o^t}{\partial h^{t-1}}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial C^t} &= \underbrace{\frac{\partial \mathcal{L}}{\partial (f^{t+1} \odot C^t + i^{t+1} \odot \hat{C}^t)}}_{\left( \frac{\partial \mathcal{L}}{\partial C^{t+1}} + \frac{\partial \mathcal{L}}{\partial h^{t+1}} \frac{\partial h^{t+1}}{\partial C^{t+1}} \right) \odot f^{t+1}} \underbrace{\frac{\partial (f^{t+1} \odot C^t + i^{t+1} \odot \hat{C}^t)}{\partial C^t}}_{\odot f^{t+1}} \end{aligned}$$

**when your basic RNN isn't  
capable of catching long-term  
dependencies**





# GRU

---

A variant of the LSTM is called the Gated Recurrent Unit, or GRU.

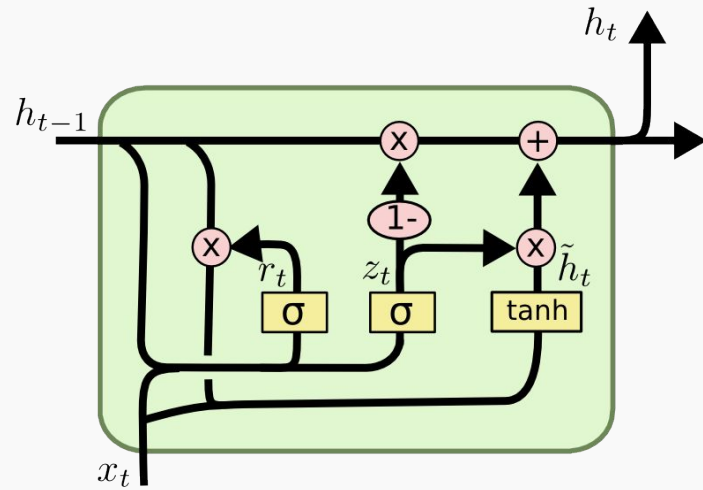
The GRU is like an LSTM but with some simplifications.

1. **The forget and input gates are combined into a single gate**
2. **No cell state**

Since there's a bit less work to be done, a GRU can be a bit faster than an LSTM. It also usually produces results that are similar to the LSTM.

**Note:** Worthwhile to try both the LSTM and GRU to see if either provides more accurate results for a data set.

# GRU (cont)

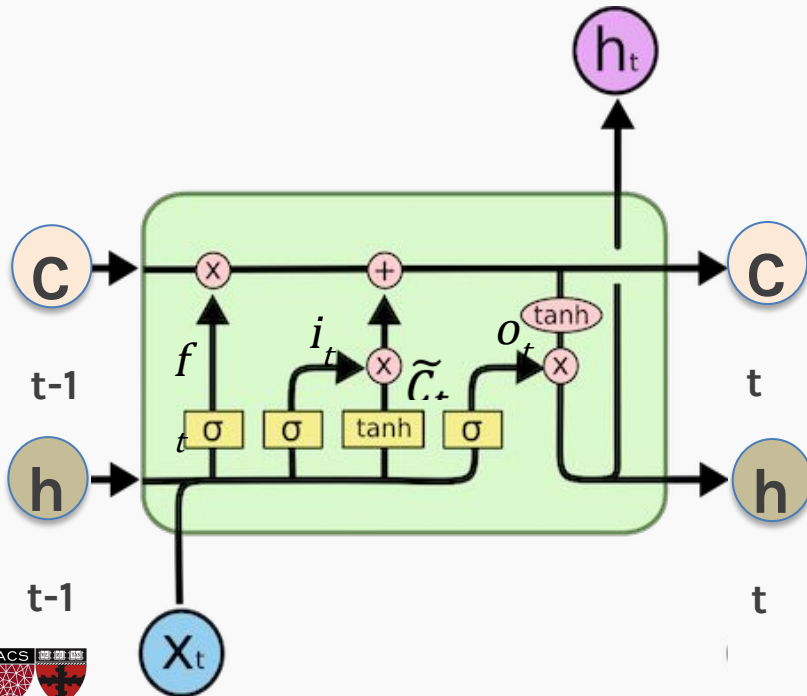


$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

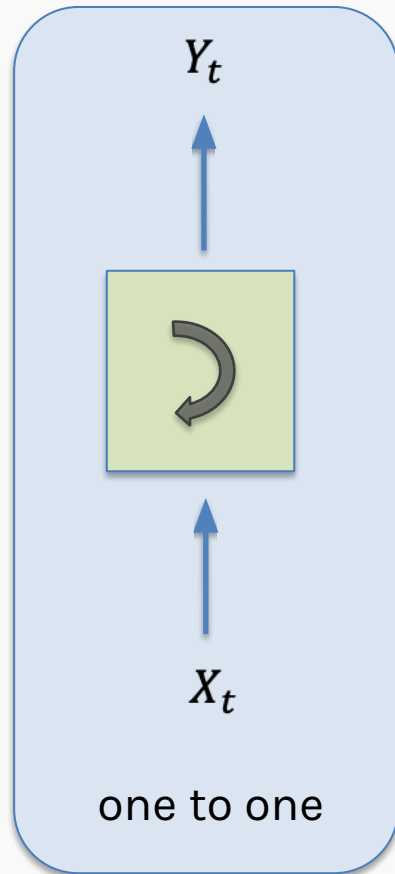
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

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

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

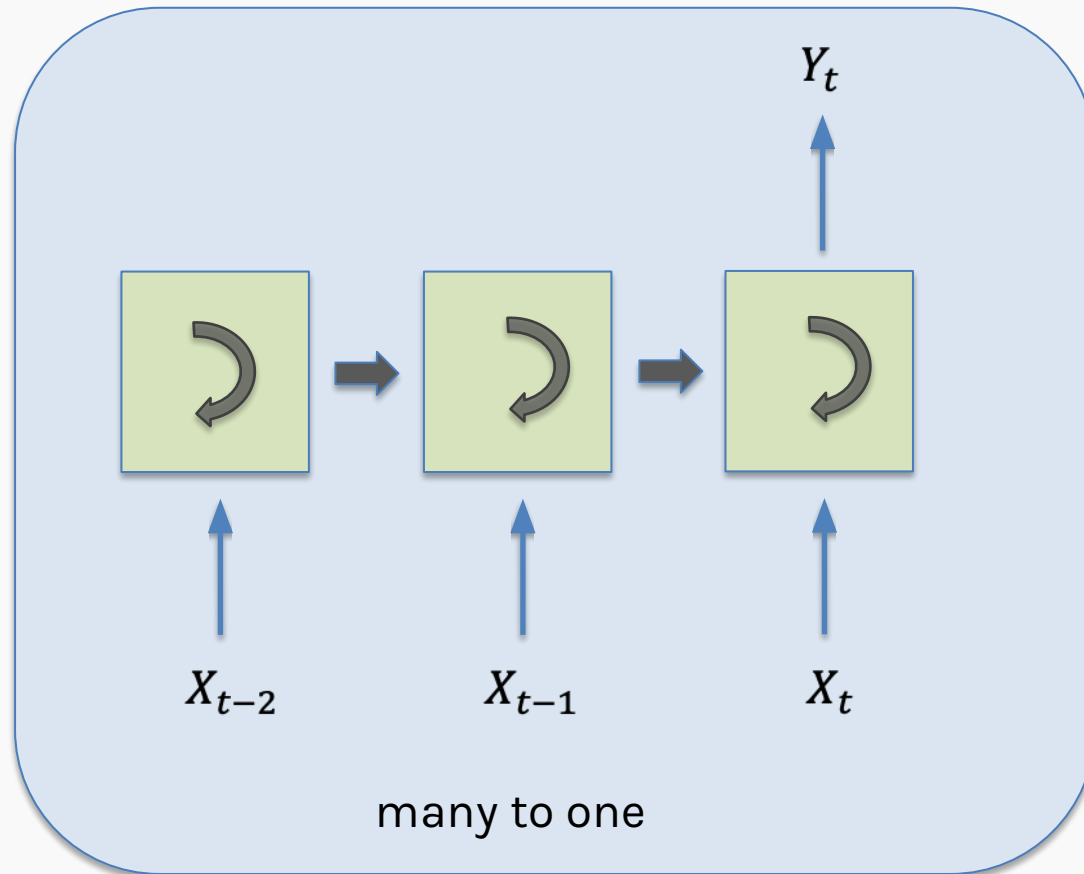


# RNN Structures



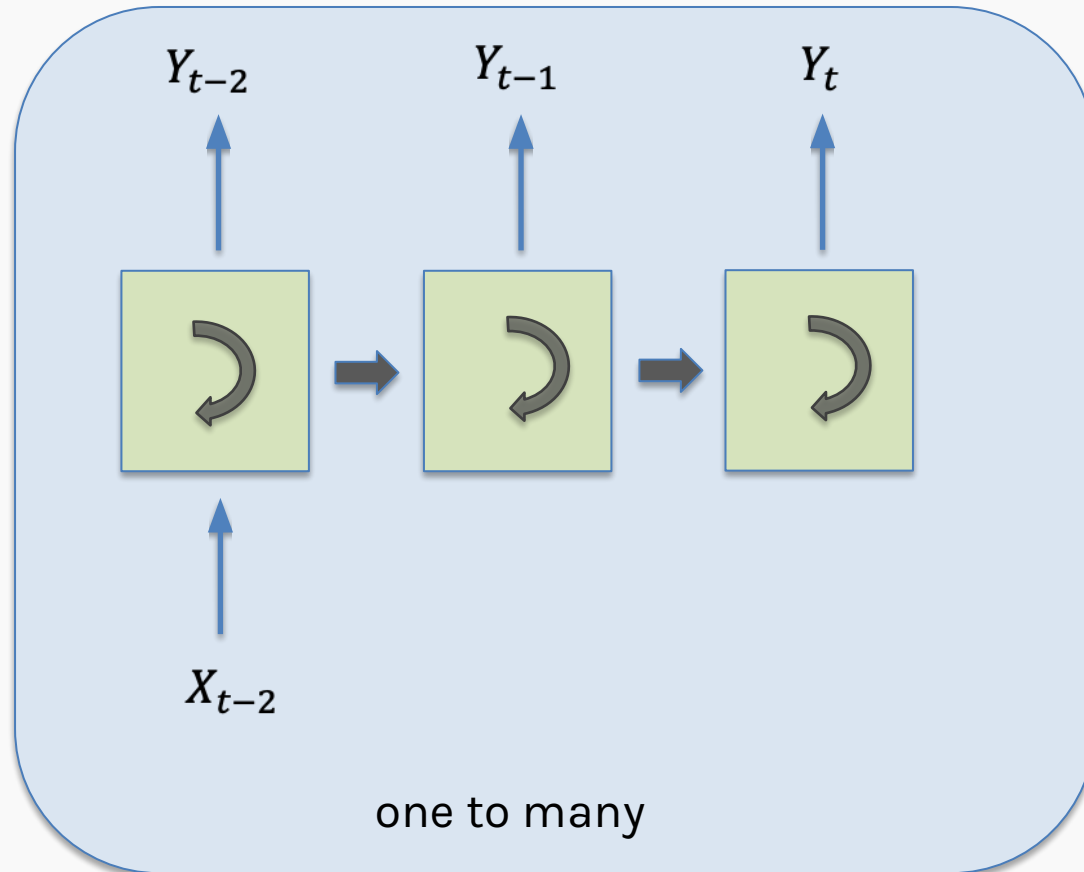
- The **one to one** structure is useless.
- It takes a single input and it produces a single output.
- Not useful because the RNN cell is making little use of its unique ability to remember things about its input sequence

# RNN Structures (cont)



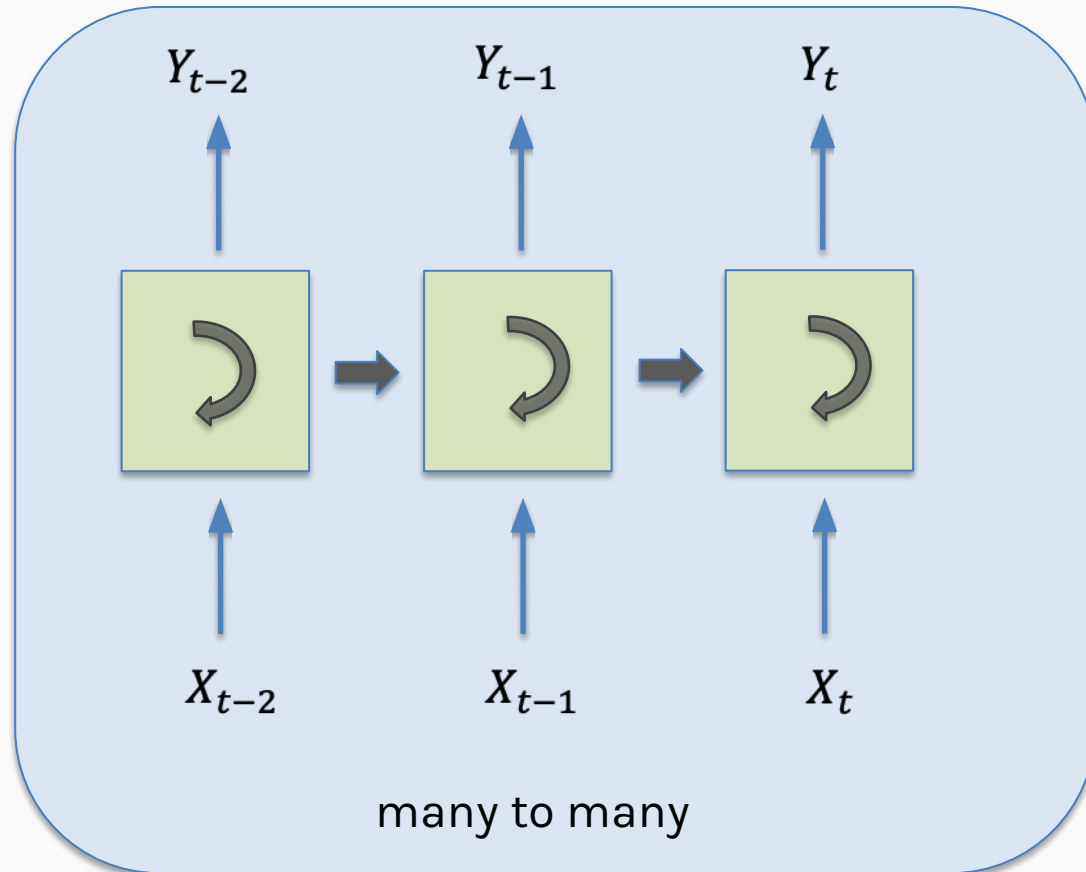
The **many to one** structure reads in a sequence and gives us back a single value. Example: Sentiment analysis, where the network is given a piece of text and then reports on some quality inherent in the writing. A common example is to look at a movie review and determine if it was positive or negative.

# RNN Structures (cont)



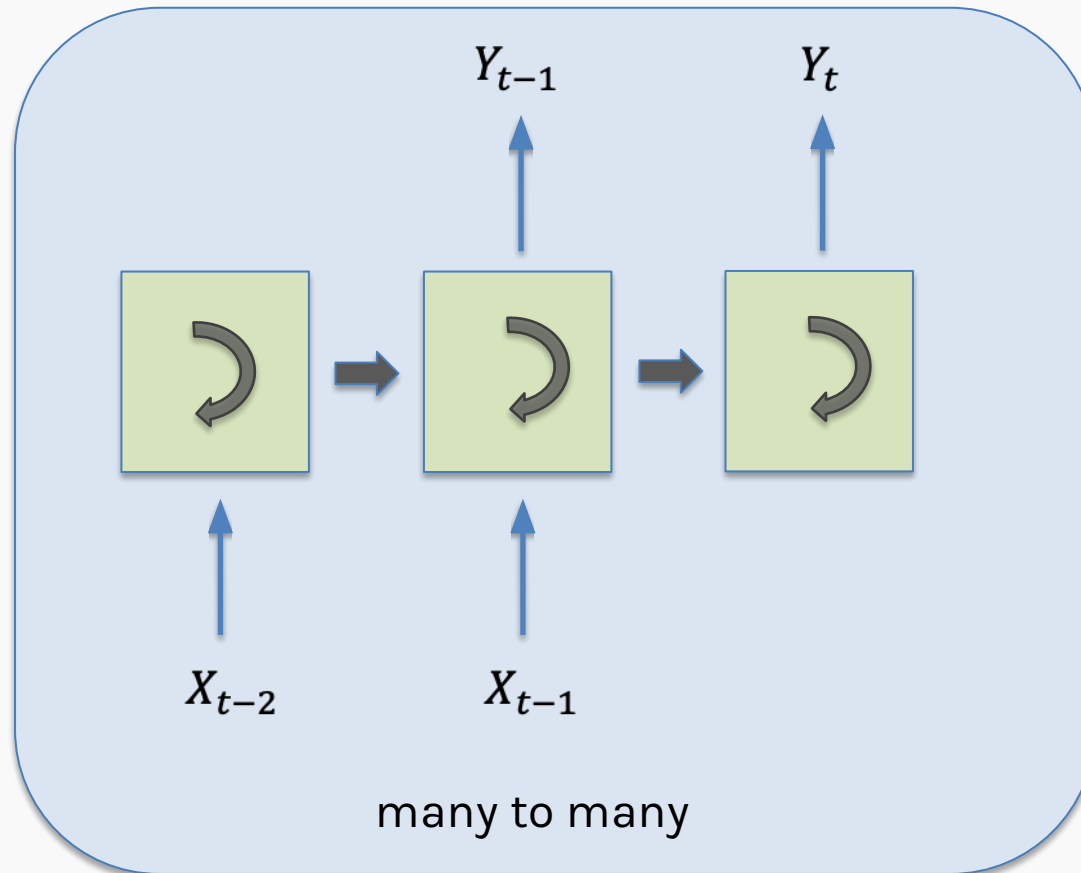
The **one to many** takes in a single piece of data and produces a sequence.  
For example we give it the starting note for a song, and the network produces the rest of the melody for us.

# RNN Structures (cont)



The **many to many** structures are in some ways the most interesting. used for machine translation.  
Example: Predict if it will rain given some inputs.

# RNN Structures (cont)



This form of **many to many** can be used for machine translation.

For example, the English sentence:  
**“The black dog jumped over the cat”**  
In Italian as:

“Il cane nero saltò sopra il gatto”  
In the Italia, the adjective “nero” (black) follows the noun “cane” (dog), so we need to have some kind of buffer so we can produce the words in their proper English.

# Bidirectional

LSTM and RNN are designed to analyze sequence of values.

For example: *Patrick said he needs a vacation.*

*he* here means *Patrick* and we know this because *Patrick* was before the word *he*.

However consider the following sentence:

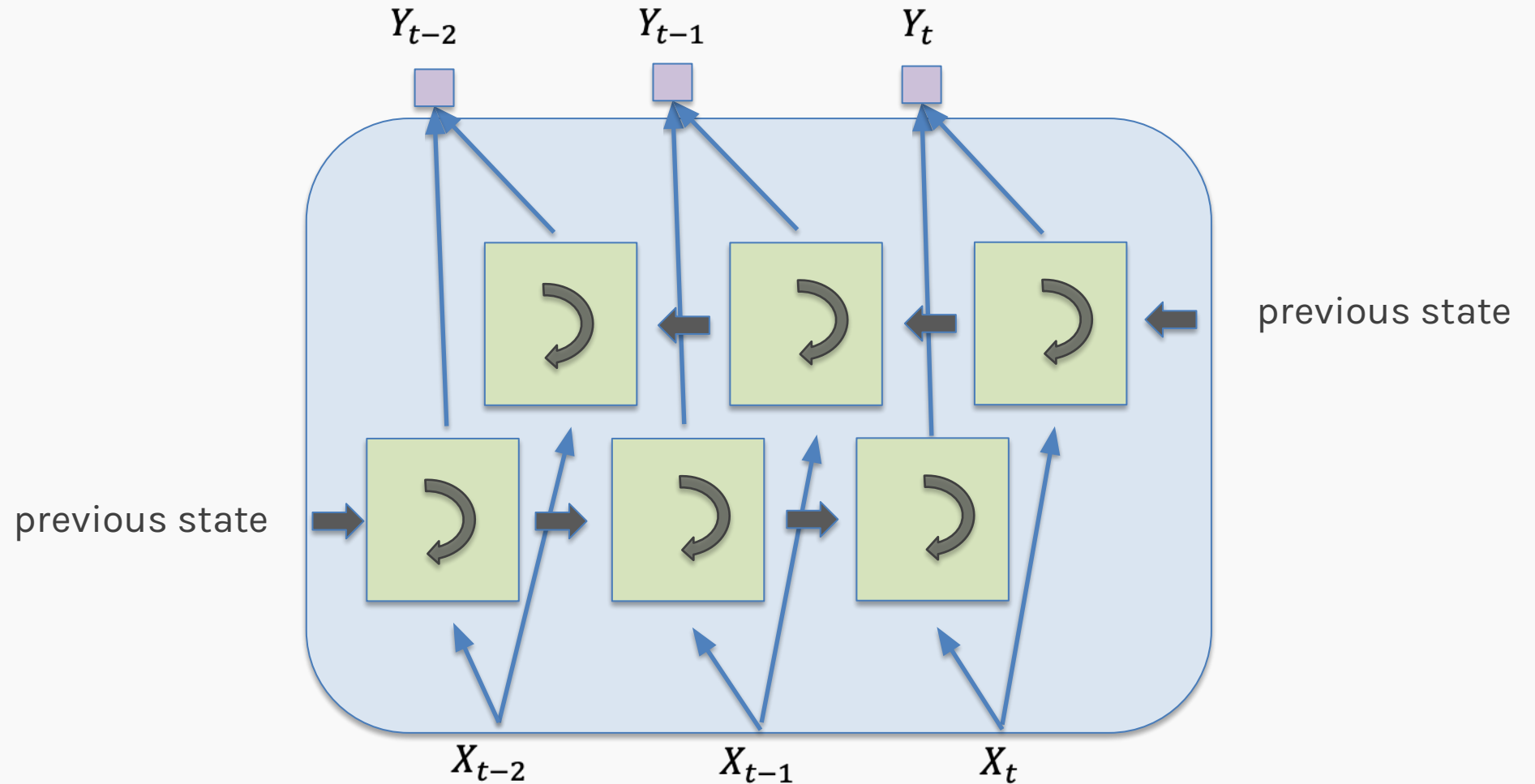
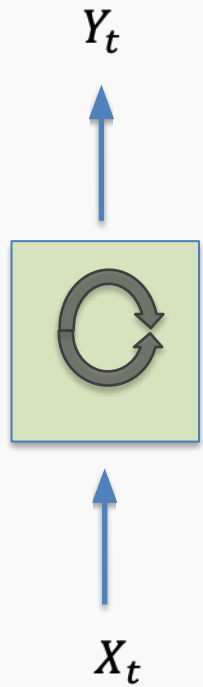
*He needs to work more, Pavlos said about Patrick.*

Bidirectional RNN or BRNN or bidirectional LSTM or BLSTM when using LSTM units.



# Bidirectional (cond)

symbol for a BRNN

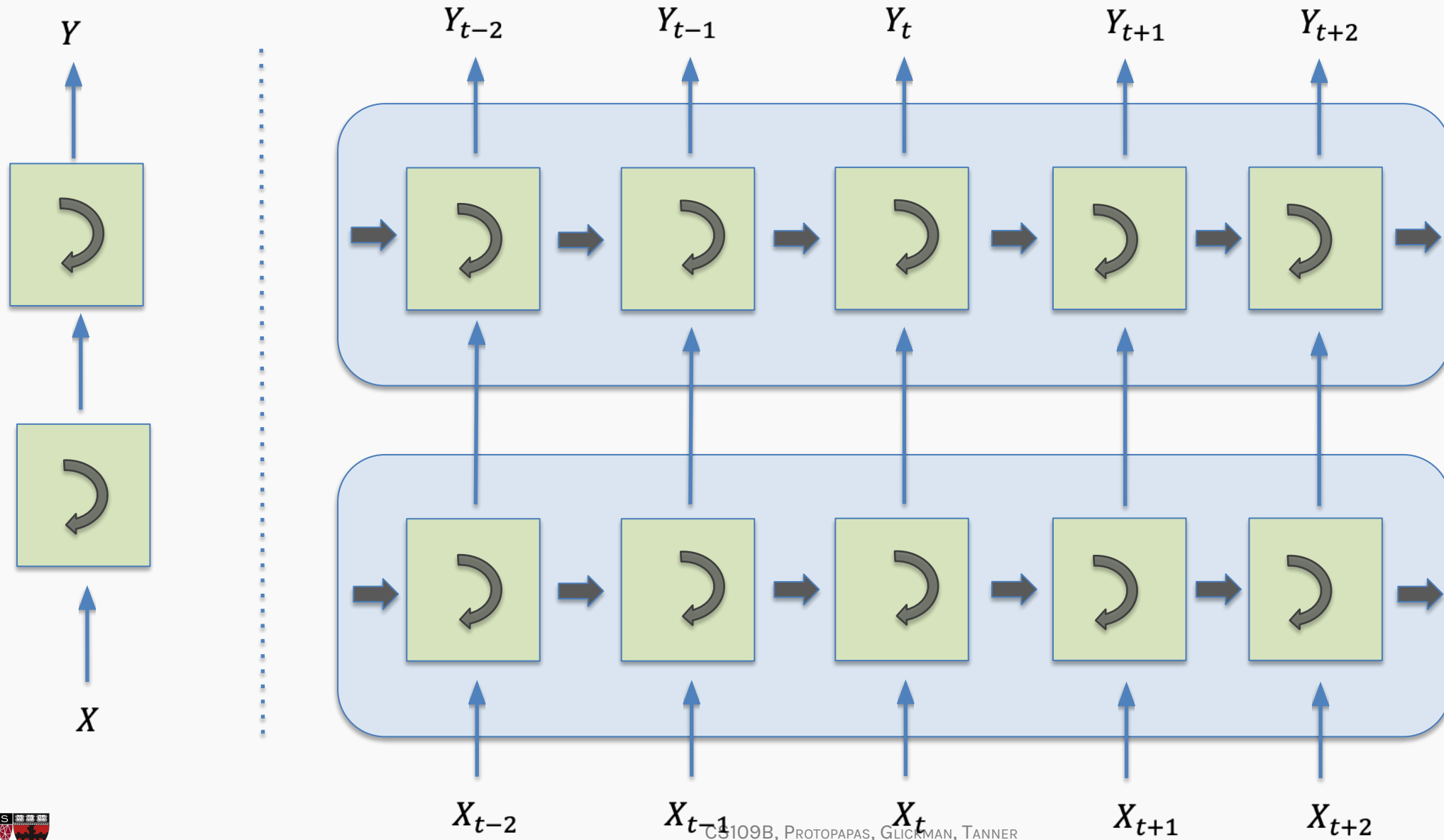


# Deep RNN

LSTM units can be arranged in layers, so that each the output of each unit is the input to the other units. This is called **a deep RNN**, where the adjective “deep” refers to these multiple layers.

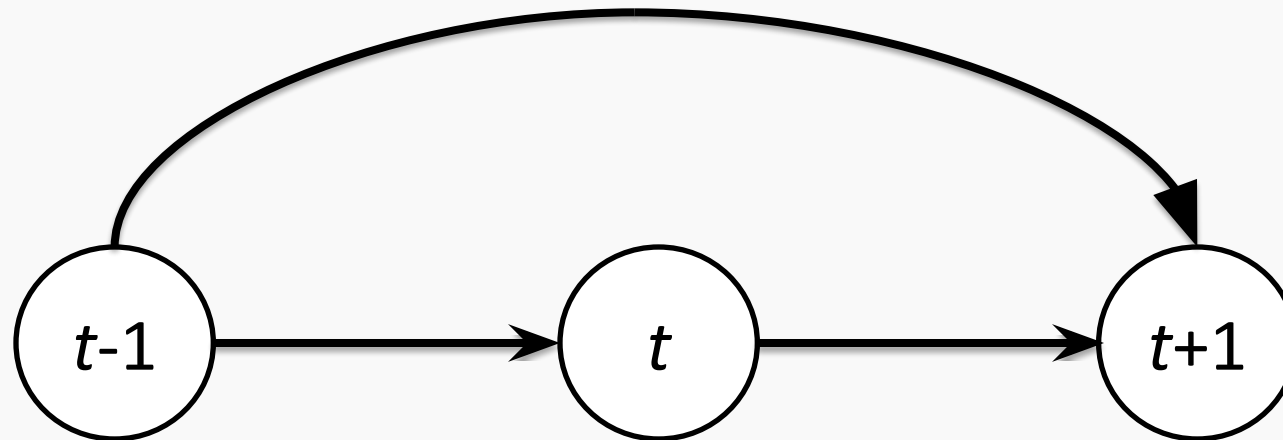
- Each layer feeds the LSTM on the next layer
- First time step of a feature is fed to the first LSTM, which processes that data and produces an output (and a new state for itself).
- That output is fed to the next LSTM, which does the same thing, and the next, and so on.
- Then the second time step arrives at the first LSTM, and the process repeats.

# Deep RNN



# Skip Connections

Add additional **connections between units  $d$  time steps apart**  
Creating paths through time where gradients neither vanish or explode



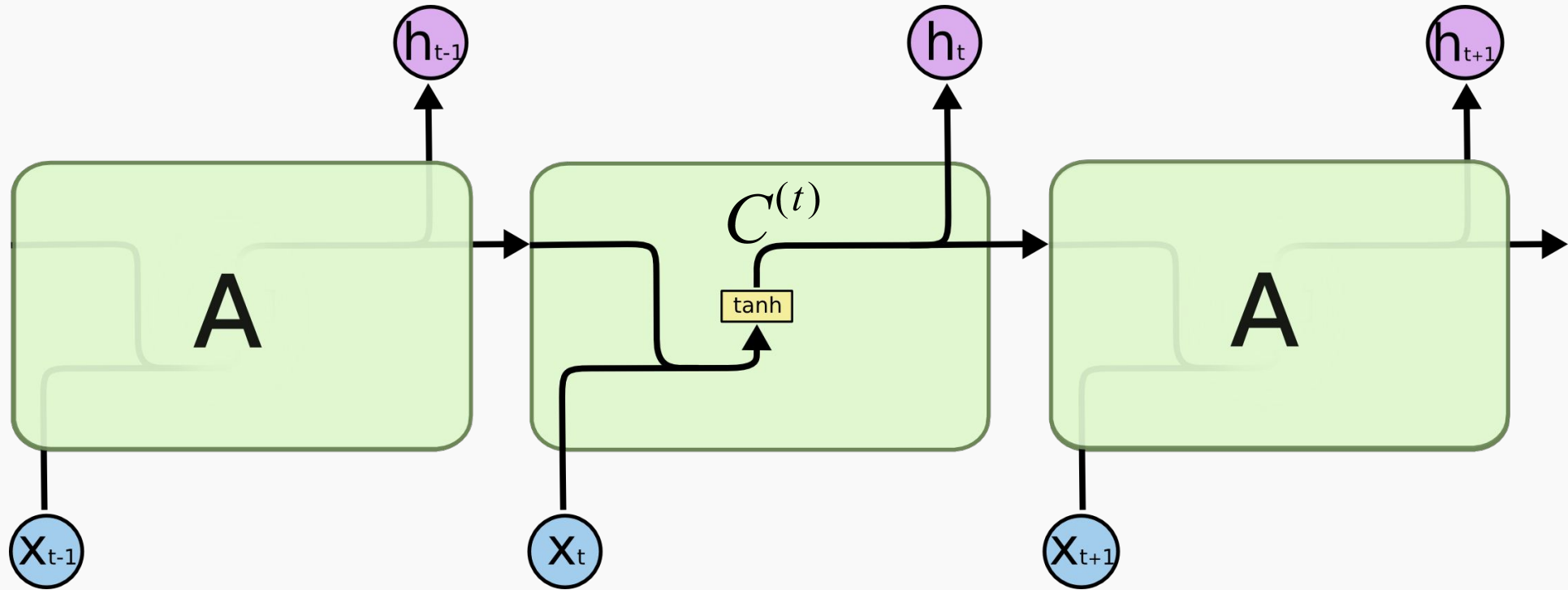
# Leaky Units

---

Linear self-connections

Maintain **cell state**: running average of past hidden activations

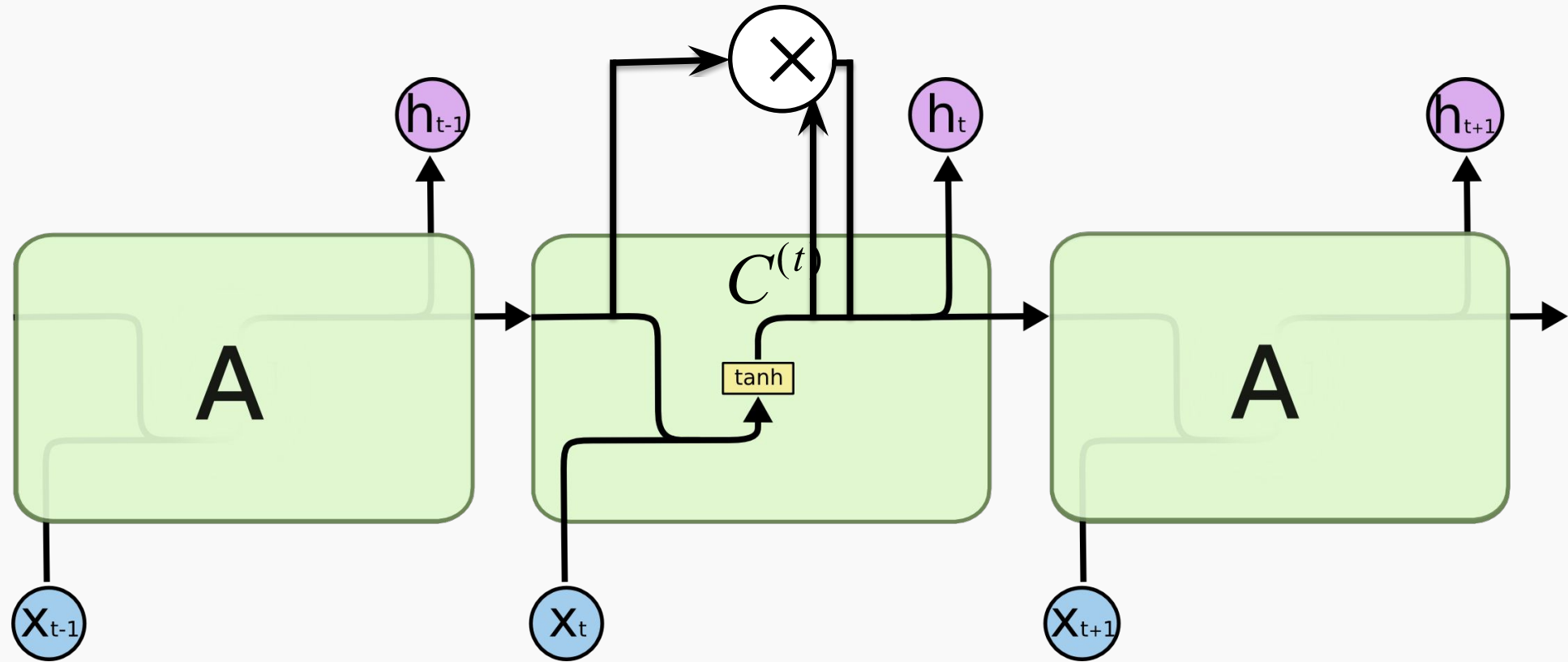
# Standard RNN



$$C^{(t)} = \tanh(\mathbf{W}h^{(t-1)} + \mathbf{U}x^{(t-1)})$$

$$h^{(t)} = C^{(t)}$$

# Leaky Unit



$$C^{(t)} = \tanh(\mathbf{W}h^{(t-1)} + \mathbf{U}x^{(t-1)})$$

$$h^{(t)} = \alpha h^{(t-1)} + (1-\alpha)C^{(t)}$$

# Sequence to Sequence

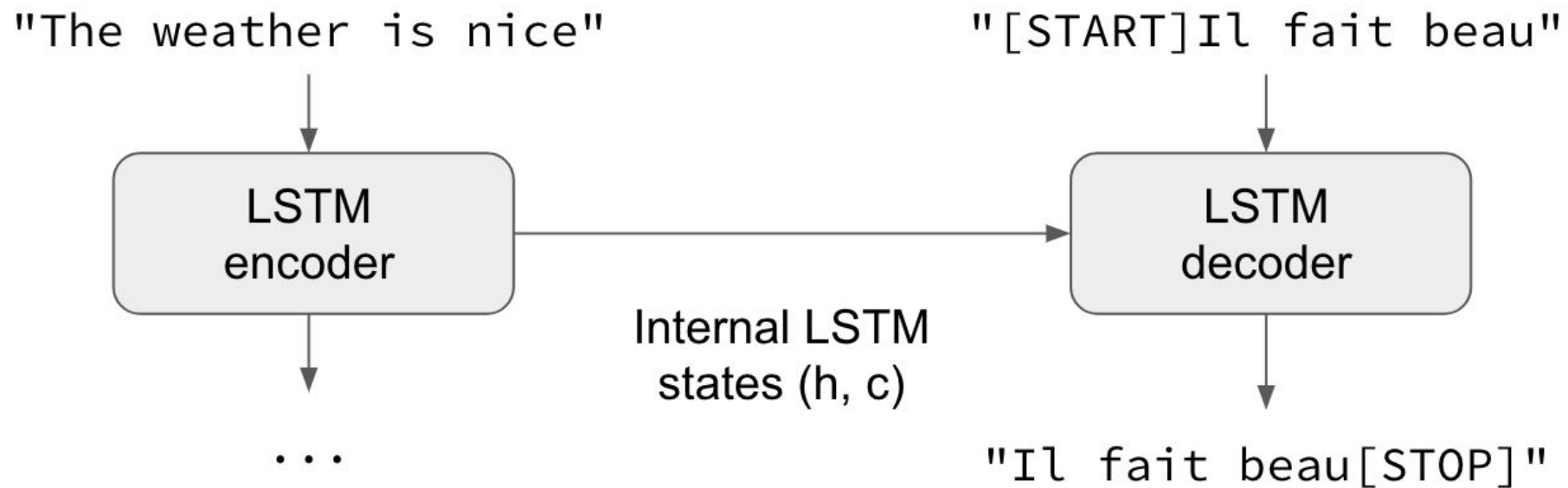
---

**Sebastien lived in France.**

Seq2seq model learns from variable sequence input fixed length sequence output. It uses two LSTM model, one learns vector representation from input sequence of fixed dimensionality and another LSTM learns to decode from this input vector to target sequence.



# Sequence to Sequence (cont)



# What is Teacher Forcing (cont)

*Models that have recurrent connections from their outputs leading back into the model may be trained with teacher forcing.*

— Page 372, [Deep Learning](#), 2016.

*Teacher forcing is a procedure [...] in which during training the model receives the ground truth output  $y(t)$  as input at time  $t + 1$ .*

— Page 372, [Deep Learning](#), 2016.

# What is Teacher Forcing (cont)

Given the following input sequence:

“The wheels on the bus go round and round.”

In this task we want to train a model to generate the next word in the sequence given the previous sequence of words.

We add a token to signal the start of the sequence and another to signal the end of the sequence. We will use “[**START**]” and “[**END**]” respectively.

“[**START**] The wheels on the bus go round and round [**END**]”

# What is Teacher Forcing (cont)

Imagine the model generates the word “A”, but of course, we expected “The”.

The model is off track and is going to get punished for every subsequent word it generates. This makes learning slower and the model unstable.

**Instead, we can use teacher forcing.**

In the first example when the model generated “A” as output, we can discard this output after calculating error and feed in “The” as part of the input on the subsequent time step.

# What is Teacher Forcing (cont)

In the first example when the model generated “**A**” as output, we can discard this output after calculating error and feed in “**The**” as part of the input on the subsequent time step.

[START], ?

[START], **The**, wheels ?

[START], **The**, wheels, on , ?

[START], **The**, wheels, on , **the**, ?

...

**REMEMBER: ONLY IN TRAINING TIME**



# LiveSlides web content

To view

**Download the add-in.**

[liveslides.com/download](https://liveslides.com/download)

**Start the presentation.**

# Attention models

---

Sebastien lived in France.

Back in Sebastien's days at France, he lived in the city of Paris, a city of great beauty and filled with love and beautiful art, and he spoke French, a language with great history and the national language of France.

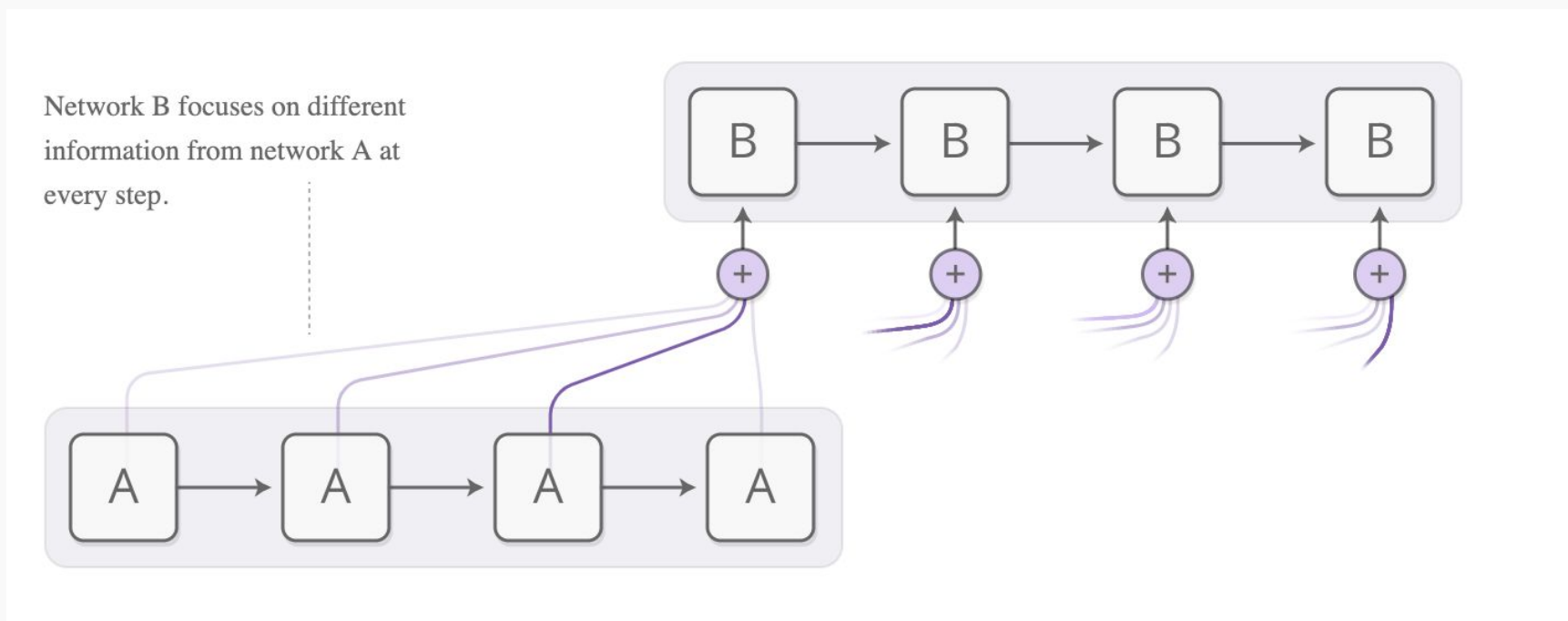
Back in Sebastien's days at France, =>  
he lived in the city of Paris, =>  
a city of great beauty and filled with love and beautiful art, =>  
and he spoke French, a language with great history and the national language of France.

# Attention models

When translating a sentence, you pay attention to the word that is presently translated.

When transcribing an audio recording, you listen carefully to the segment you are actively writing down.

To describe the room you are in, you describe the objects in that room.





# Attention models (cont)

---

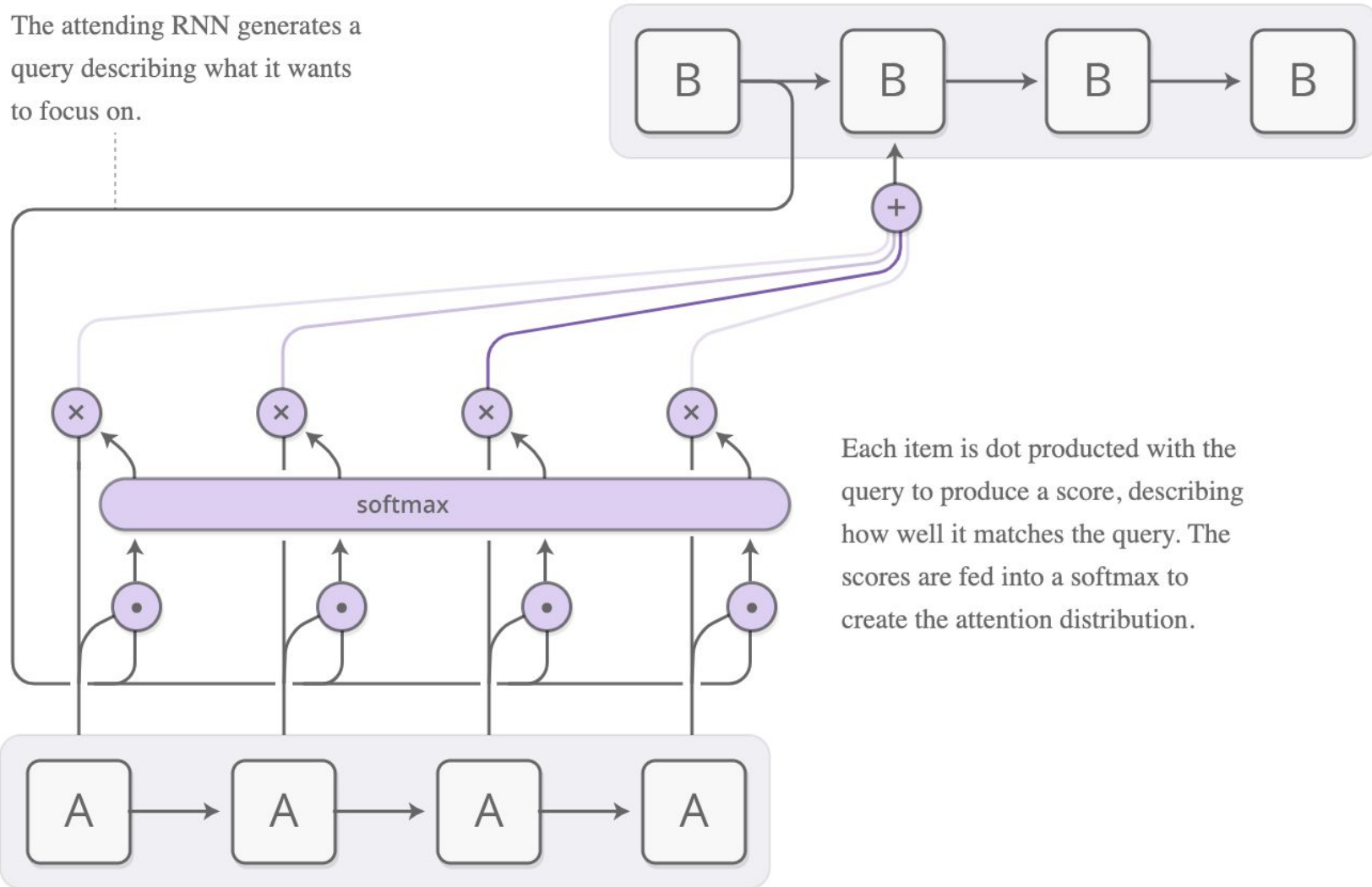
This attention is generated with content-based attention. The attending RNN generates a query describing what it wants to focus on. Each item is multiplied (dot product) with the query to produce a score, describing how well it matches the query. The scores are fed into a softmax to create the attention distribution.

Source

– <https://github.com/google/seq2seq>

# Attention models (cont)

The attending RNN generates a query describing what it wants to focus on.



Each item is dot producted with the query to produce a score, describing how well it matches the query. The scores are fed into a softmax to create the attention distribution.



# LiveSlides web content

To view

**Download the add-in.**

[liveslides.com/download](https://liveslides.com/download)

**Start the presentation.**