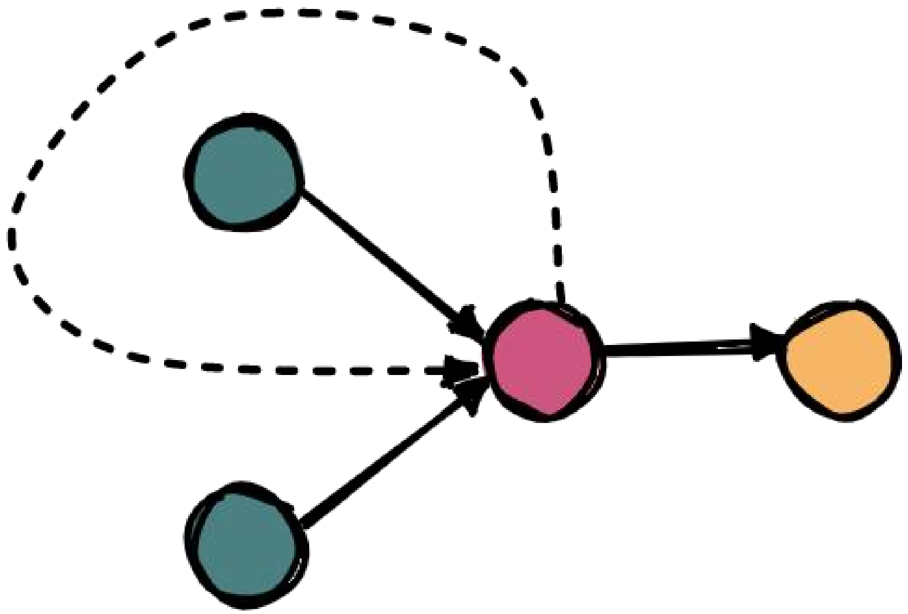
# Recurrent neural networks



Week 19

Middlesex University Dubai; CST4050 Fall21;
Instructor: Dr. Ivan Reznikov

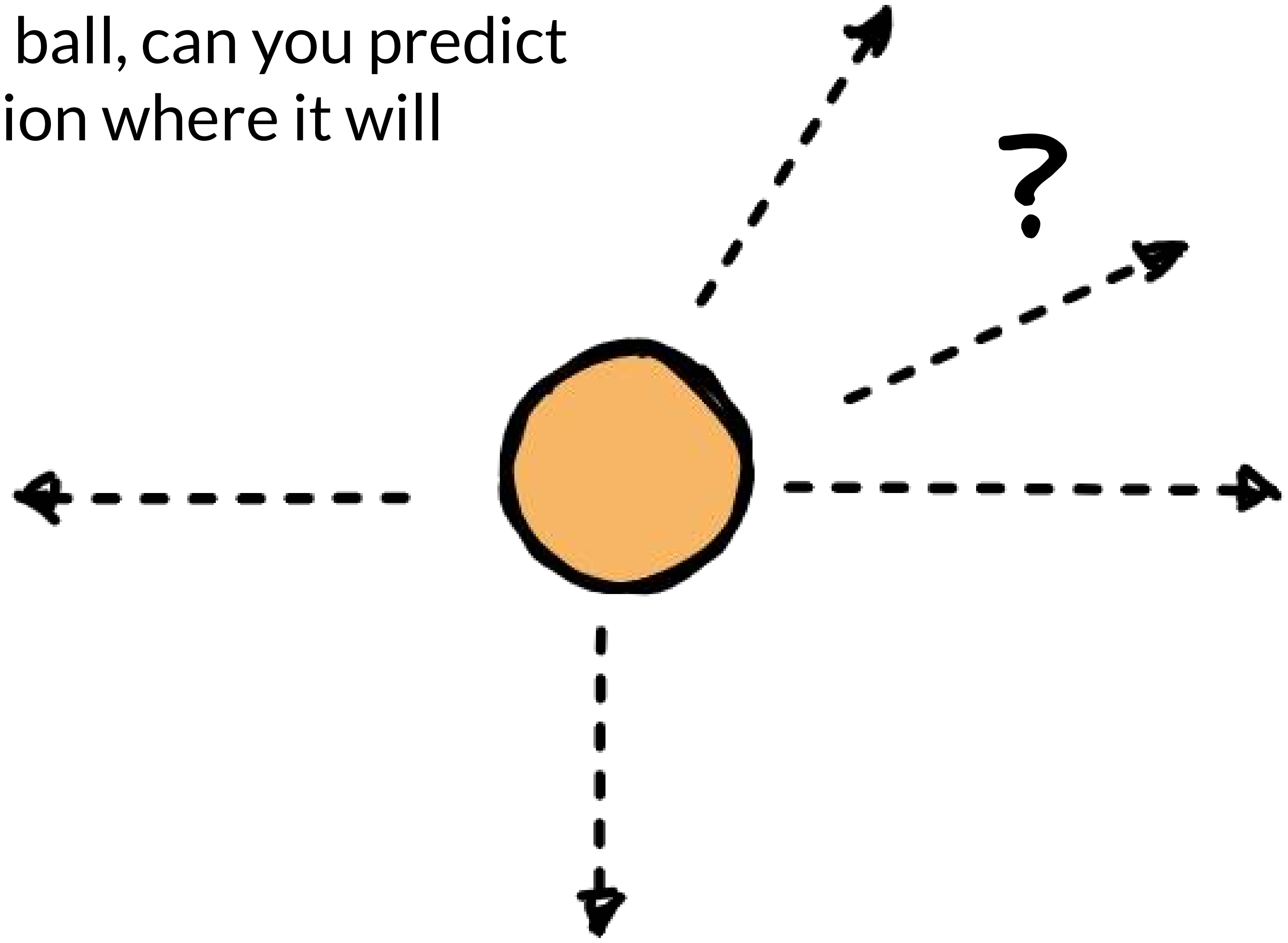
v1.1

# Plan

- Sequential Data
- Challenges with sequences
- Recurrent neural networks
- Long-short term memory
- Gated recurrent units

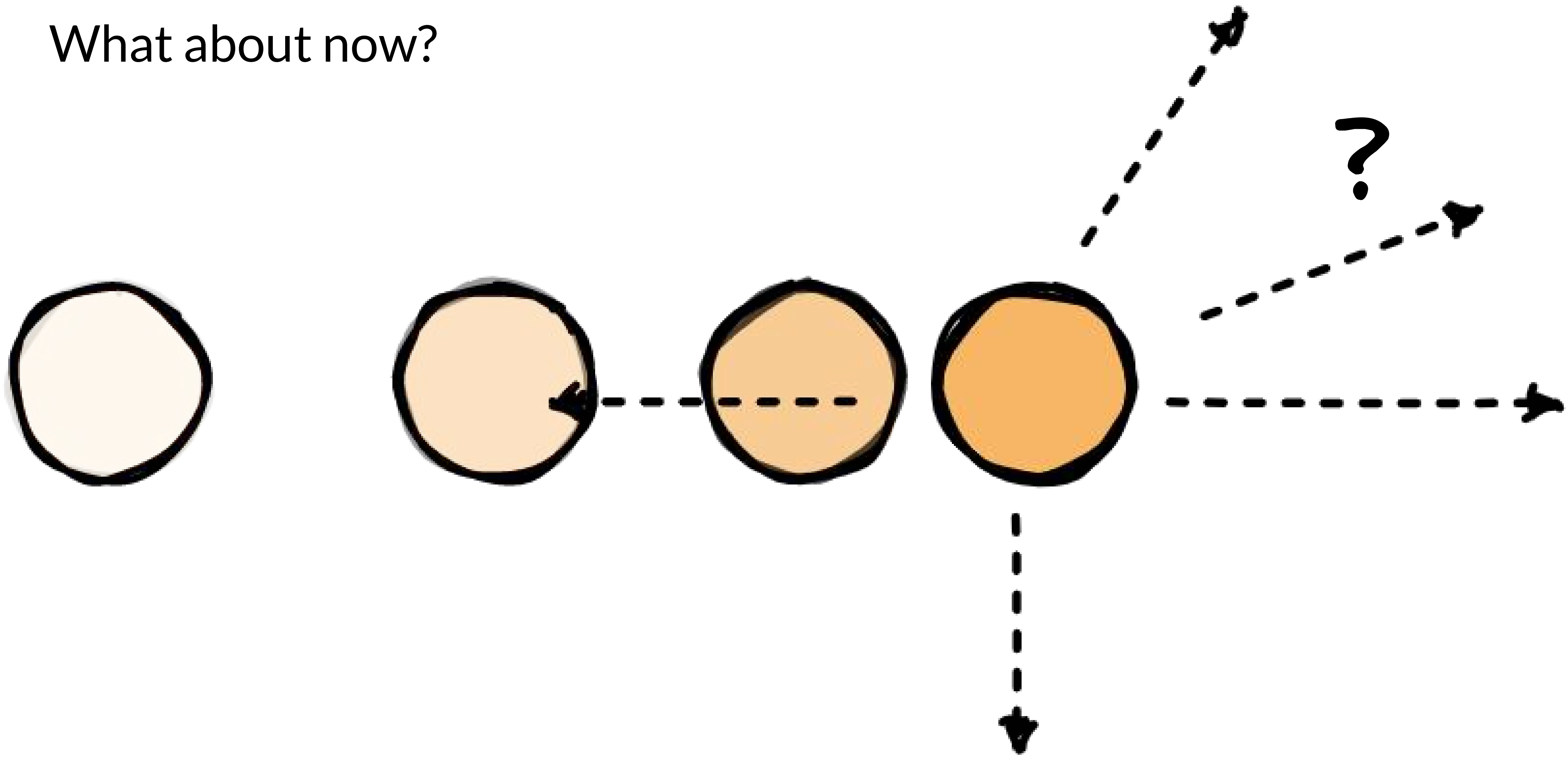
# Ball position prediction

Given the image of the ball, can you predict the direction and location where it will move next?



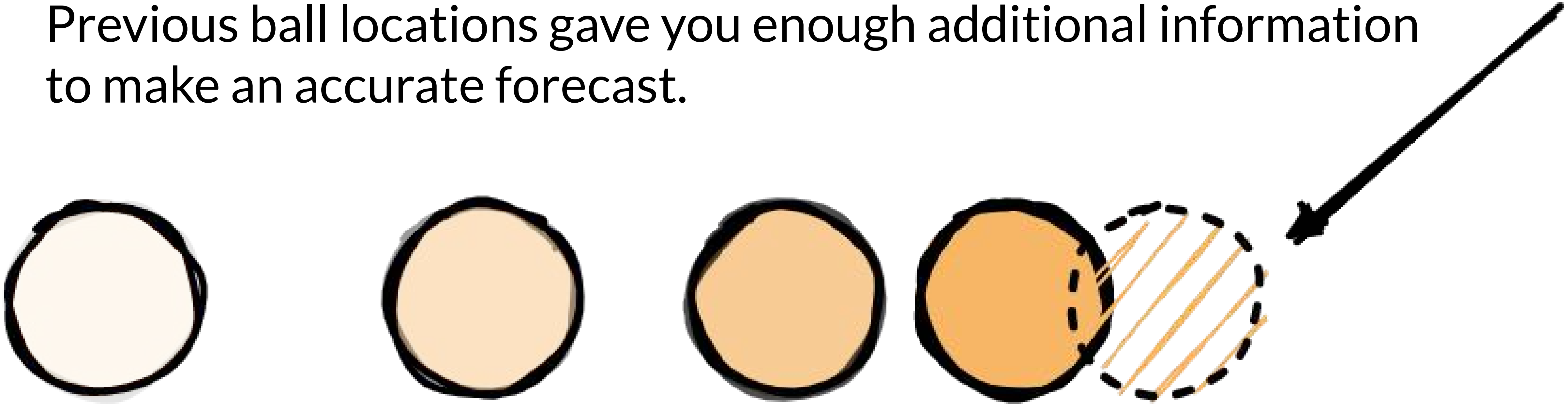
# Ball position prediction

What about now?



# Ball position prediction

High chance that you correctly the exact location and direction. Previous ball locations gave you enough additional information to make an accurate forecast.



# Sequences

Try saying all the numbers in order from 0 to 11 as fast as you can.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

# Sequences

What if we randomize the order of the numbers?
A bit slower, right?
The order matters.

6, 9, 4, 7, 5, 2, 11, 8, 10, 1, 0, 3

# Sequences

What if we start from 4?
Pretty much fast as the first time

4, x, x, x, x, x, x, 11

# Sequences

Now let's do the same exercise with the alphabet

ABCDEFGHIJKLMNOPQRSTUVWXYZ

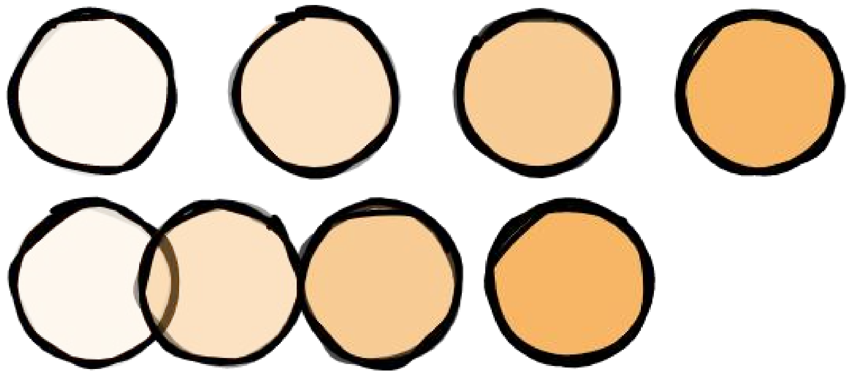
# Sequences

Now start with the letter F.
It takes a while to pick up the pace.

F. . . . . . . . . . . . . . . . . . . Z

# Why is that?

One of the reasons may be "more structured" sequenced:



?

?

5, 10, 15, ...

1, 3, 9, 27, ...

The official website for Game of Thrones on HBO, ___

Formula1 is the highest class of international racing ___

# Predict next word

The fact that Batman is so reliant on tech is his ...

train data

to be predicted

Problem 0: How to push text to a neural network, if the length of the sequence may vary?

# Predict next word

The fact that Batman is so reliant on tech is his ...

Solution a: Fixed small window

[ 00001 00010 ] → prediction

is his

Problem a: Long-distance relationships

London is my home city. This is the reason I fluently speak English

# Predict next word

The fact that Batman is so reliant on tech is his ...

Solution b: Fixed wide window

[ 00001 00010 01011 10110 11000 01110 11010 01110 10101 11000 11100]

The fact that Batman is so reliant on tech is his

→ prediction

Problem b: Lose of sense if met in different part of sequence

11010 01110 10101

reliant on tech

# Predict next word

The fact that Batman is so reliant on tech is his ...

Solution c: Sequence as set of counts
(bag-of-words)

so Joker a in

[ 1 0 0 0 1 0 0 1 0 1 0 2 0 0 1 1 0 0 0 1 0 1 0 0 0 1 0 ... ]
→ prediction

Problem c: Lose of sense if met in different part of sequence

Batman is reliant on tech == tech is reliant on Batman

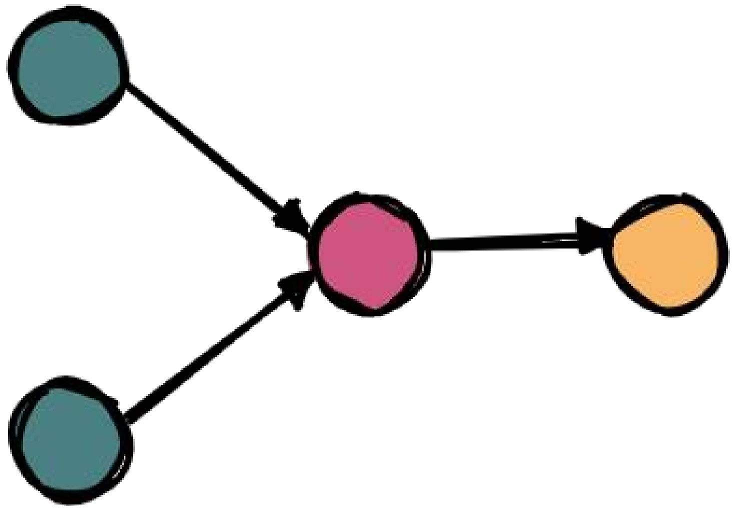
# Model criteria

Requirements:

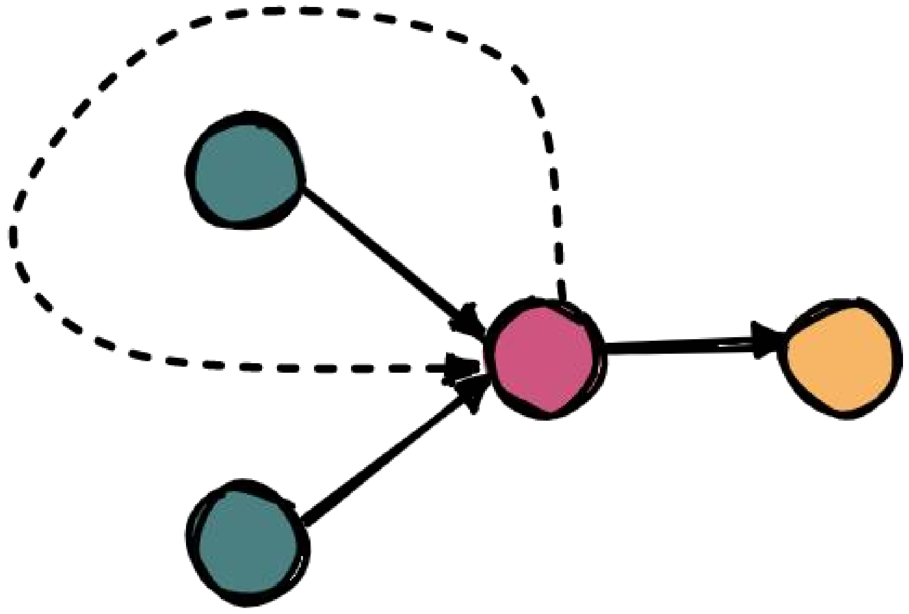
1. Handle sequences of different length
2. Track long-term dependencies
3. Share parameters across sequence
4. Save order information

# Saving memories

Having memory may be pretty important when we deal with time-series events. A regular perceptron has two inputs from the input or previous layer. But what if we could pass the "memory" – the last value of the neuron? This is called a **recurrent** perceptron. The "memory" is called **hidden state**.



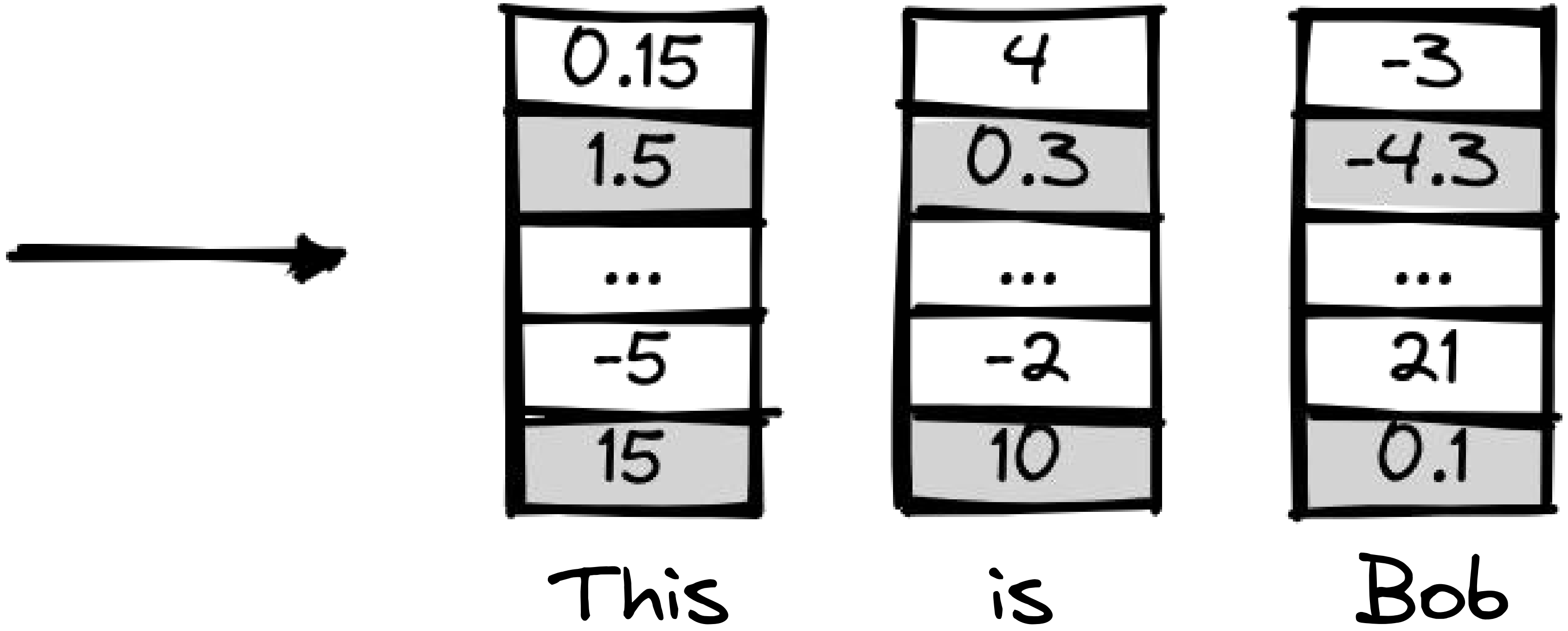
regular perceptron

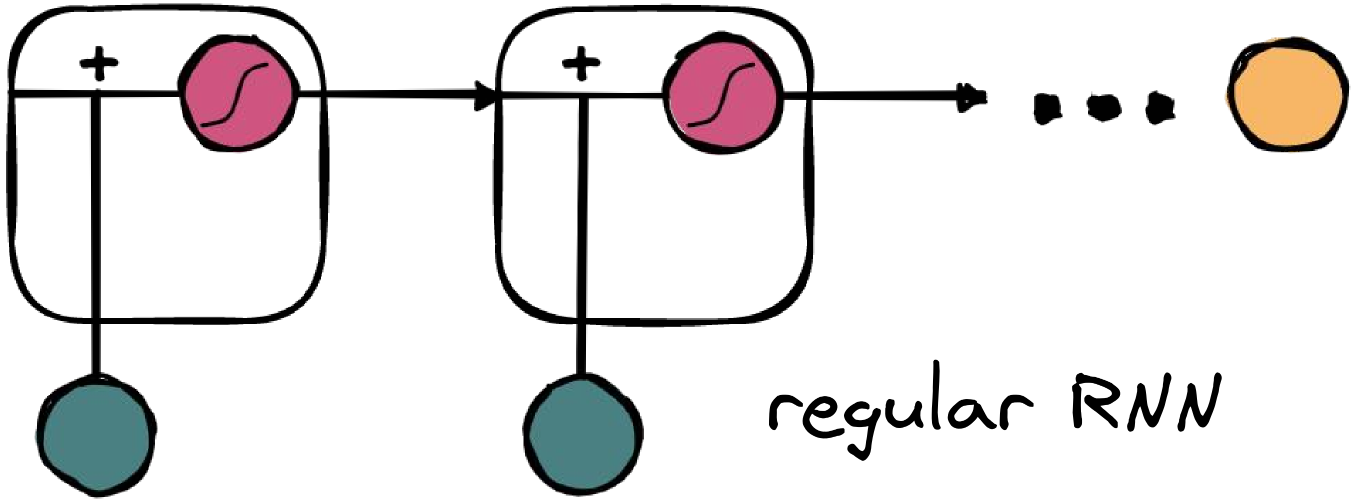


recurrent perceptron

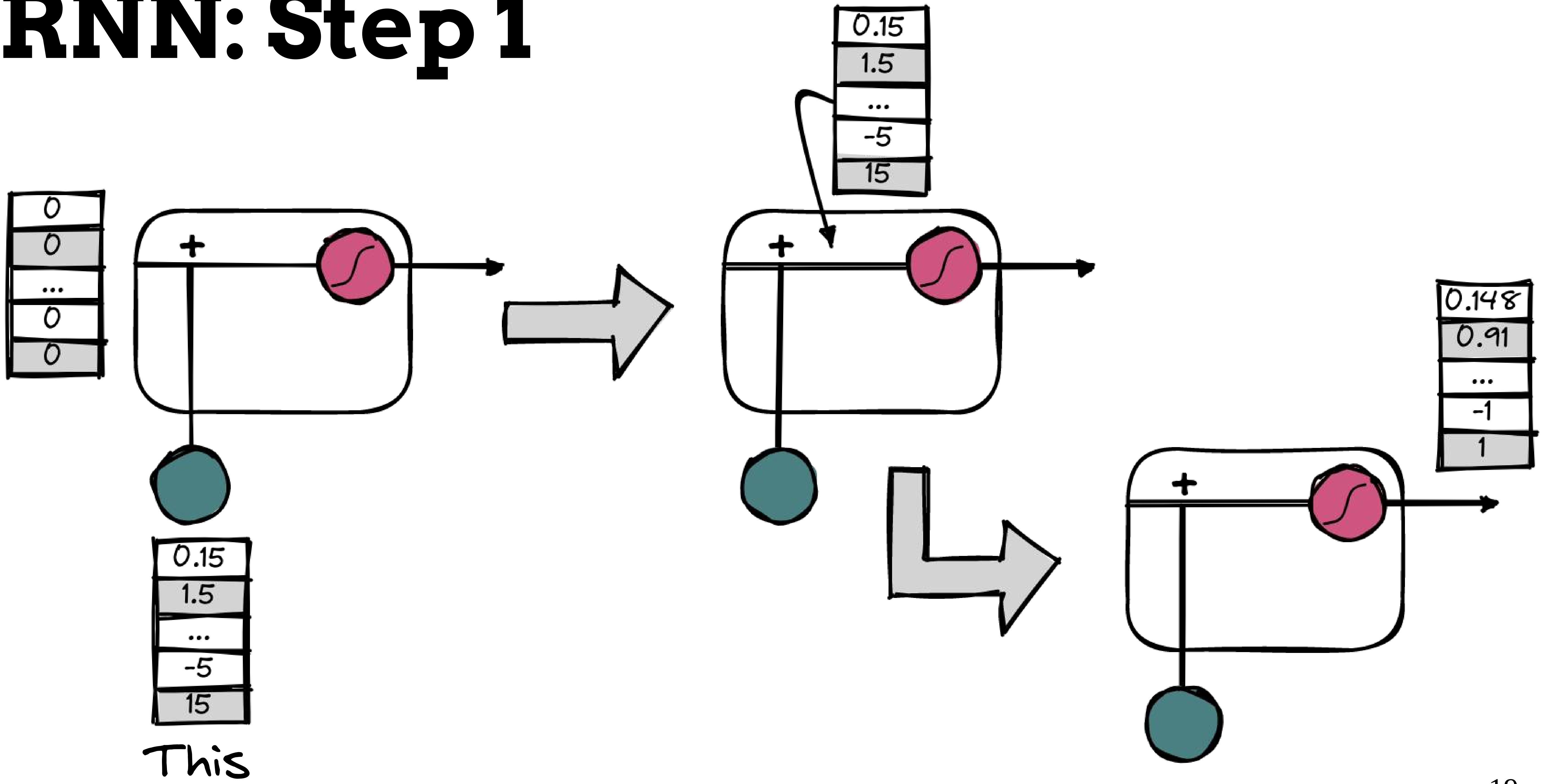
# RNN: Case1

"This is Bob"

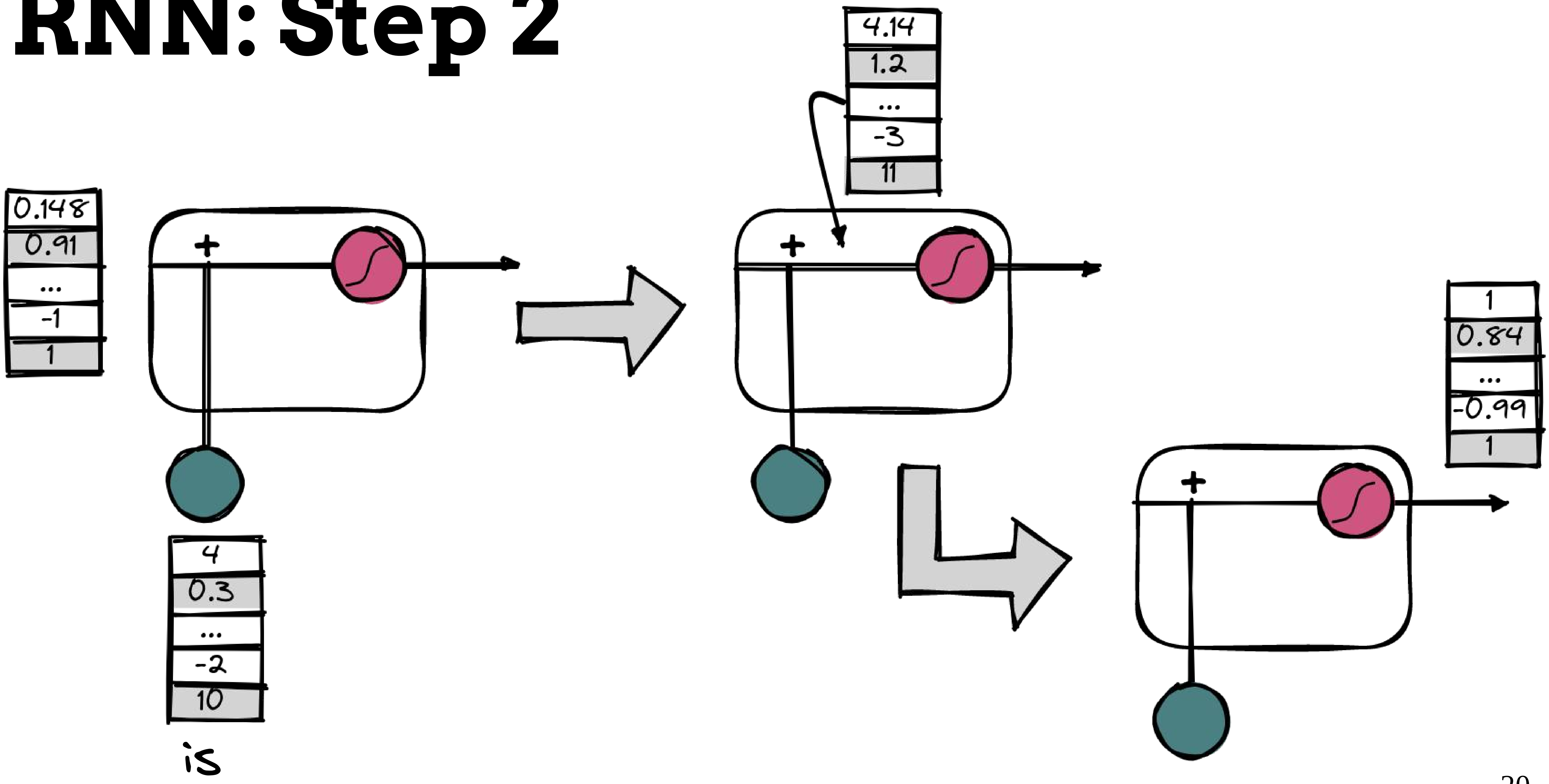


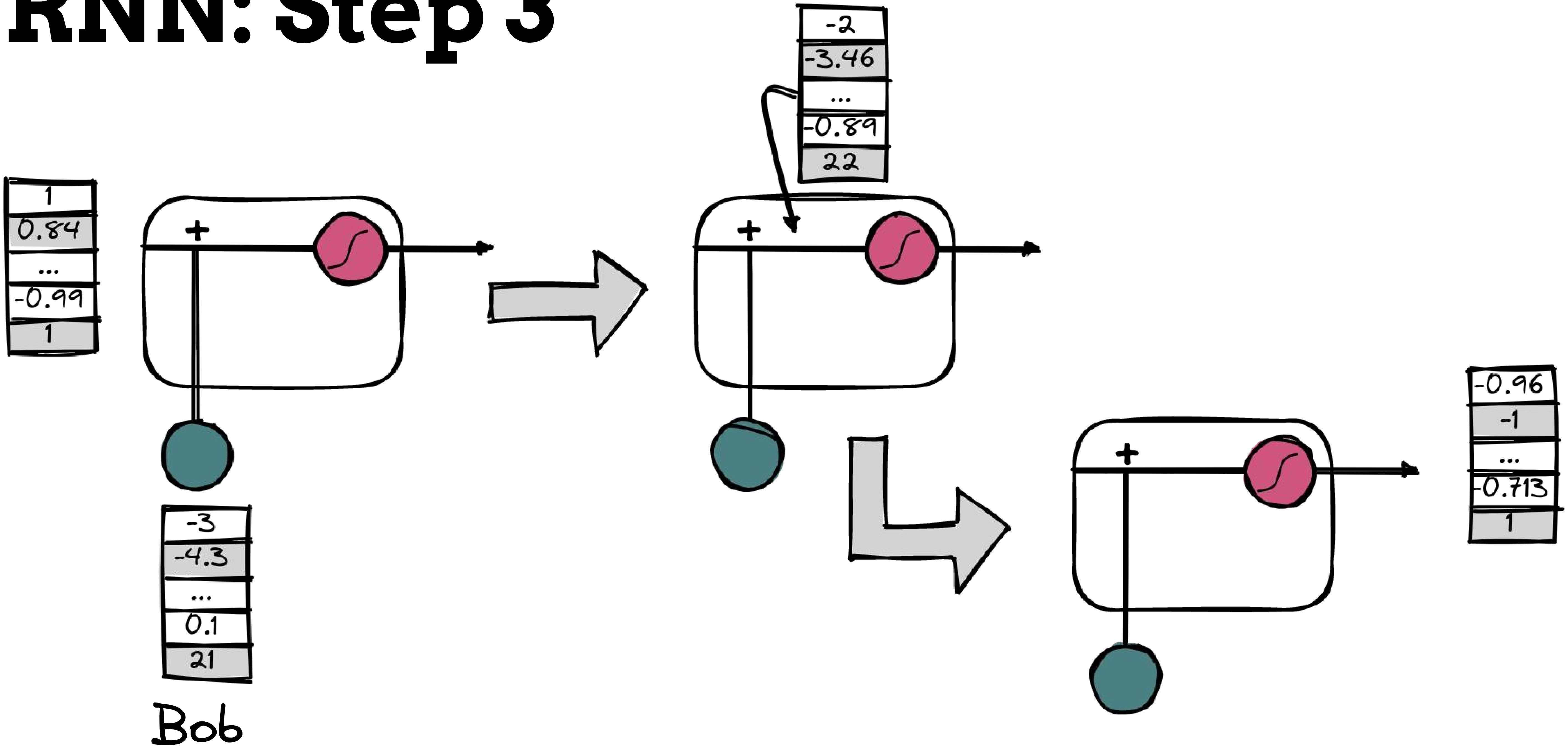
# RNN: Step 1



# RNN: Step 2



# RNN: Step 3



# RNN: Formula

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

function with parameters W

updated state

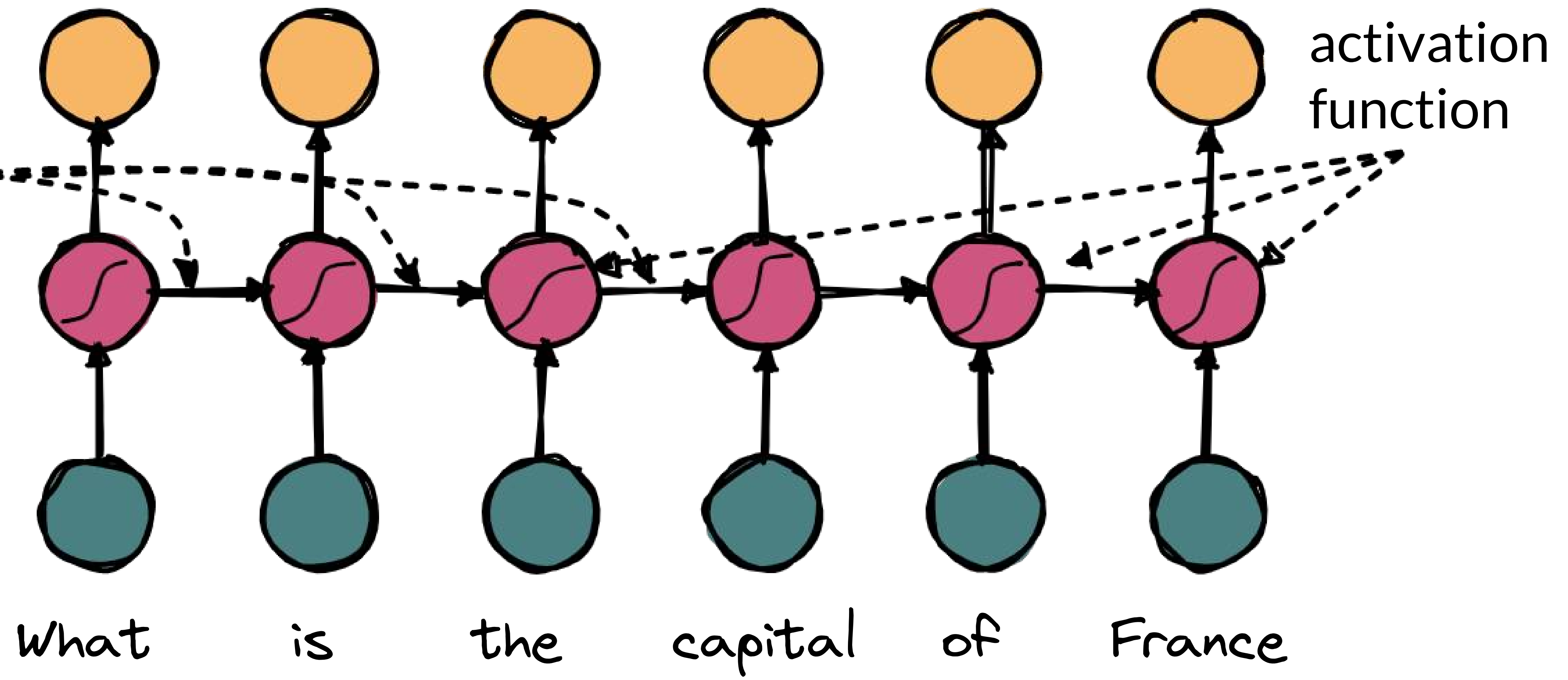
old state

input vector at time *t*

# RNN: Case2

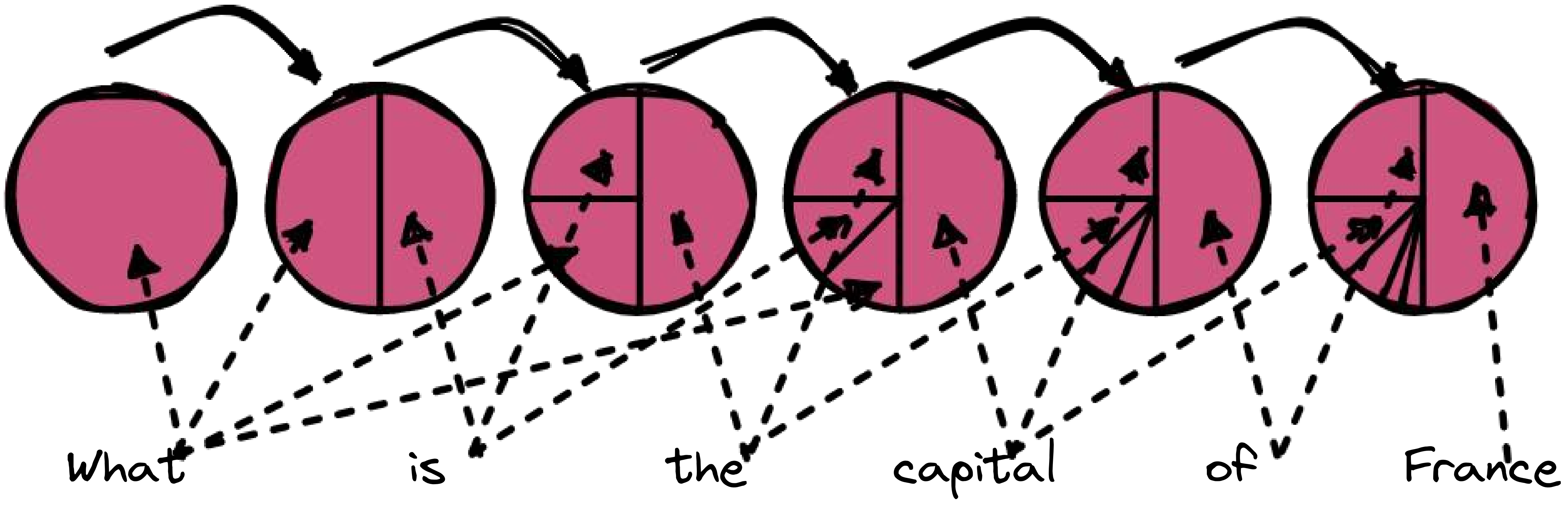
Let's take another query example:
"What is the capital of France?"

Passing information from hidden states. Hidden states are connected as a chain, and their impact lowers across time

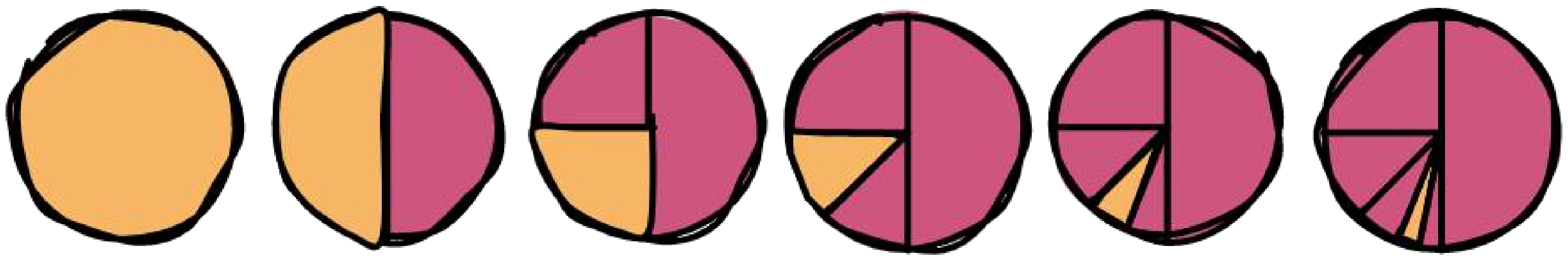


# Vanishing gradient

Let's take a look at how importance of the word changes along the chain:



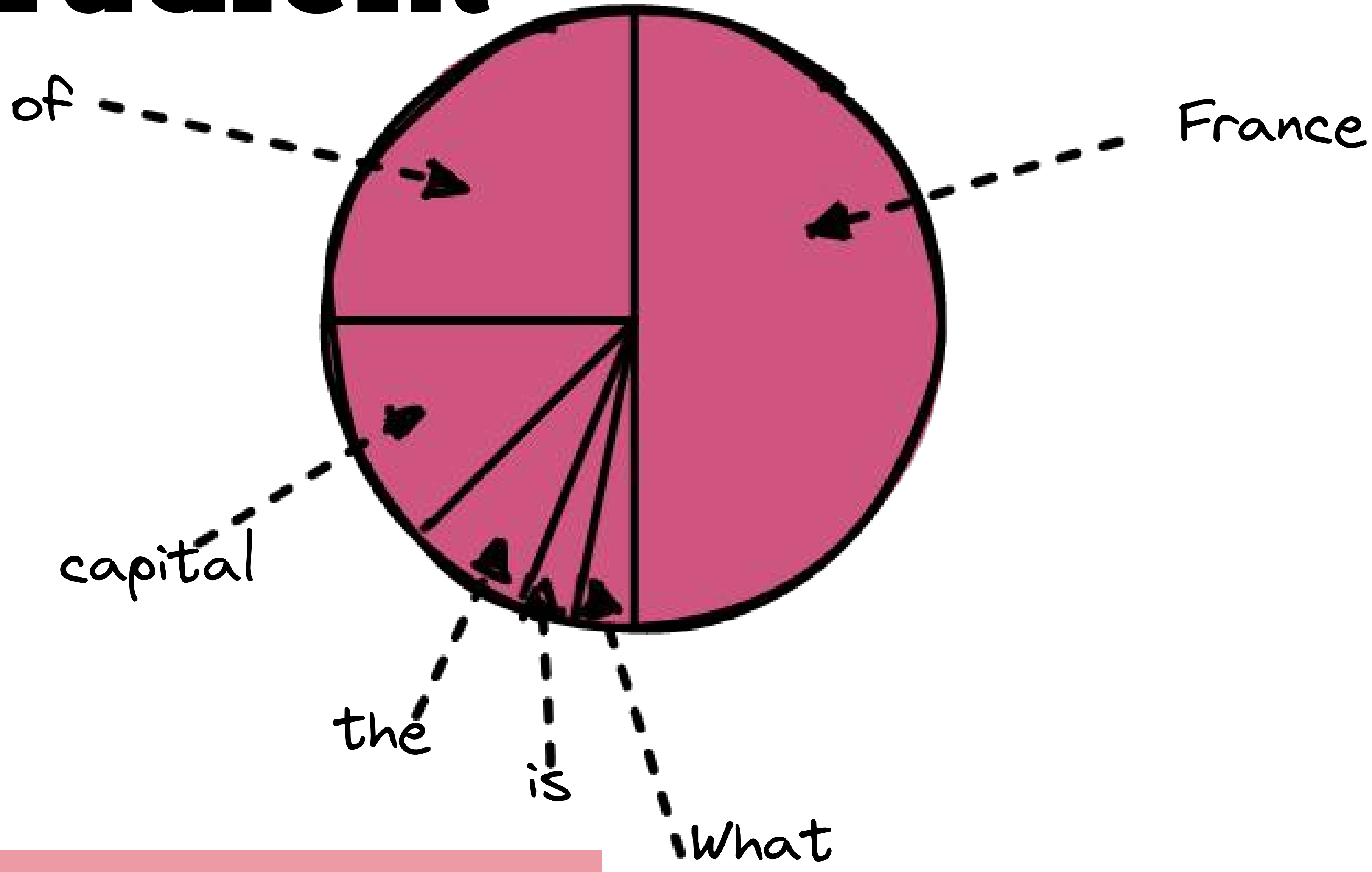

Impact of the word "What" :



# Vanishing gradient

As one can notice, the impact of recently met words is much higher than the ones from the beginning of the sequence.

This is an example of the vanishing gradient problem.

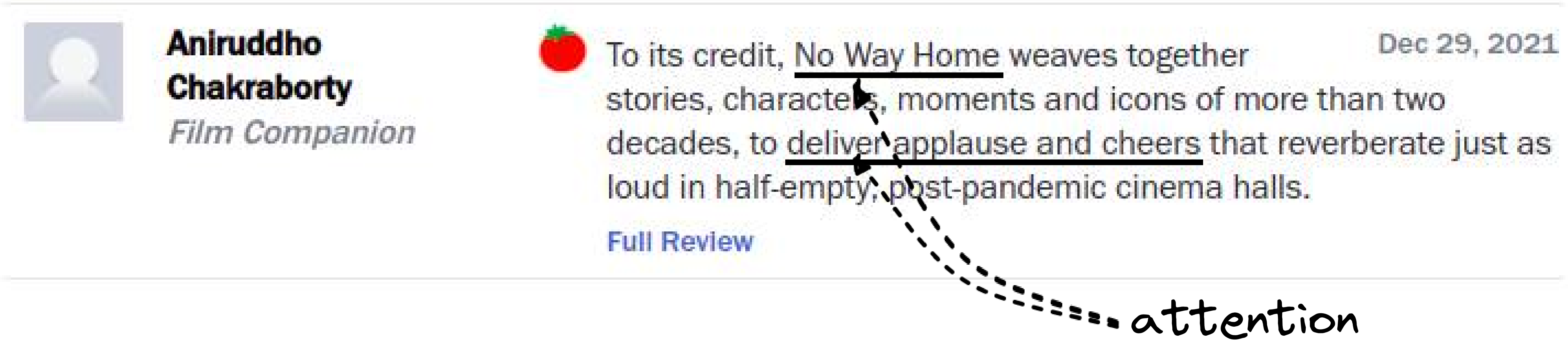



What is the capital of France

# Long-term dependencies

Solution: Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU).

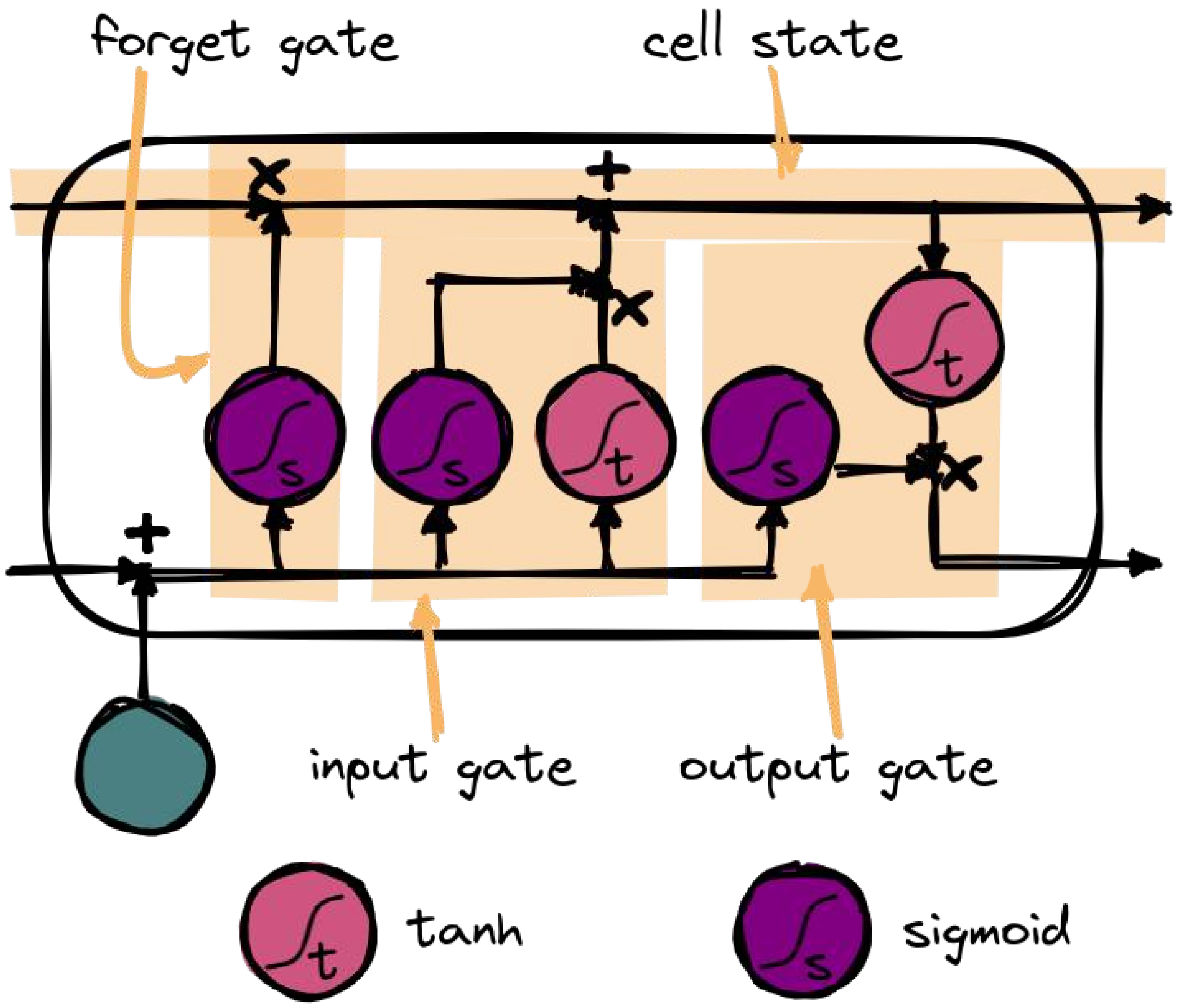
LSTMs and GRUs are like regular RNNs, but they're capable of understanding long-term dependencies. They achieve it by using "gates", that are responsible for learning what information to add or remove to the hidden state.



# Long-short term memory (LSTM)

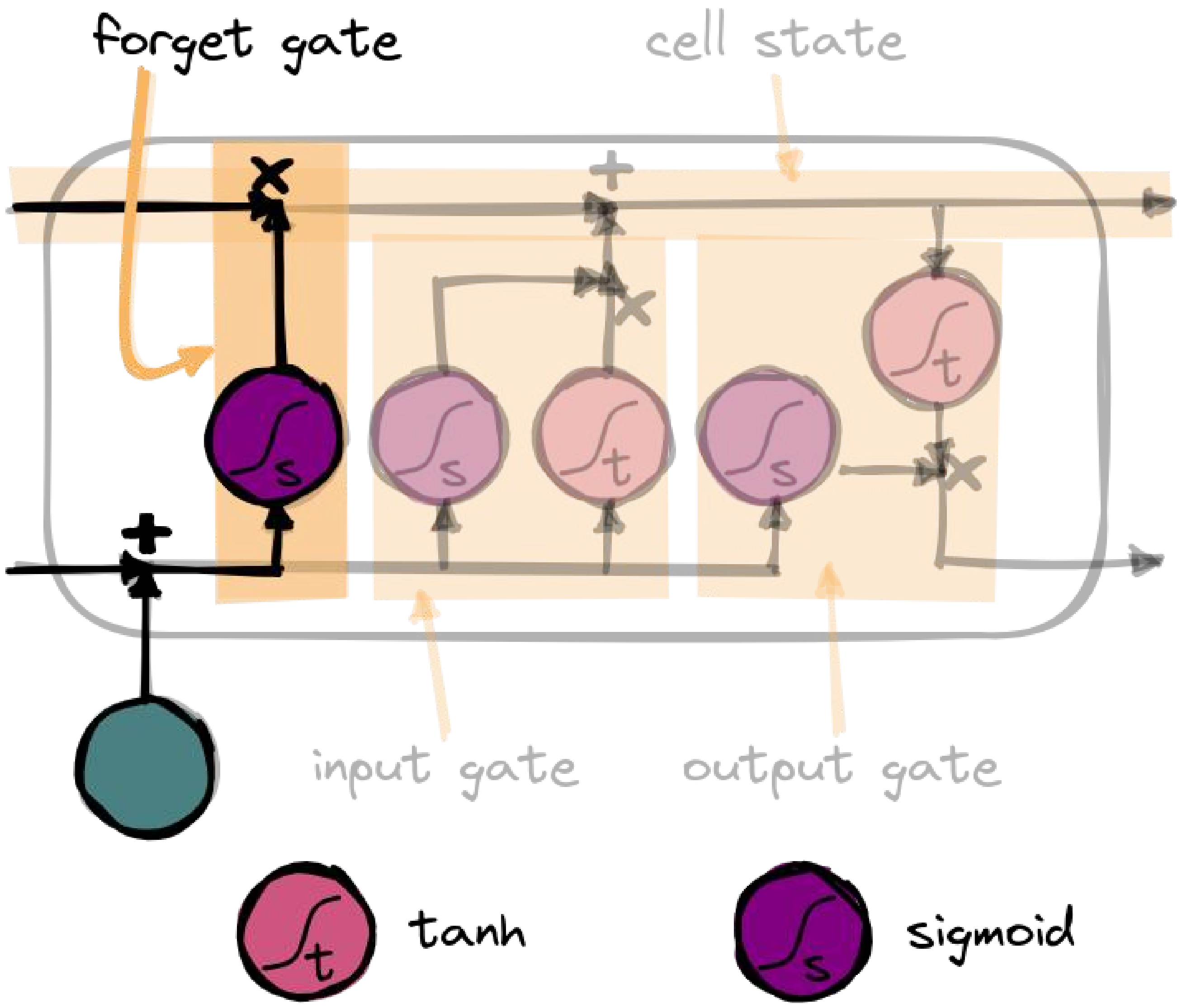
One of the solution to overcome the vanishing gradient problem is using LSTM cell, that is shown on the right.

The LSTM contains of several components:
- forget gate
- input gate
- output gate
- cell state



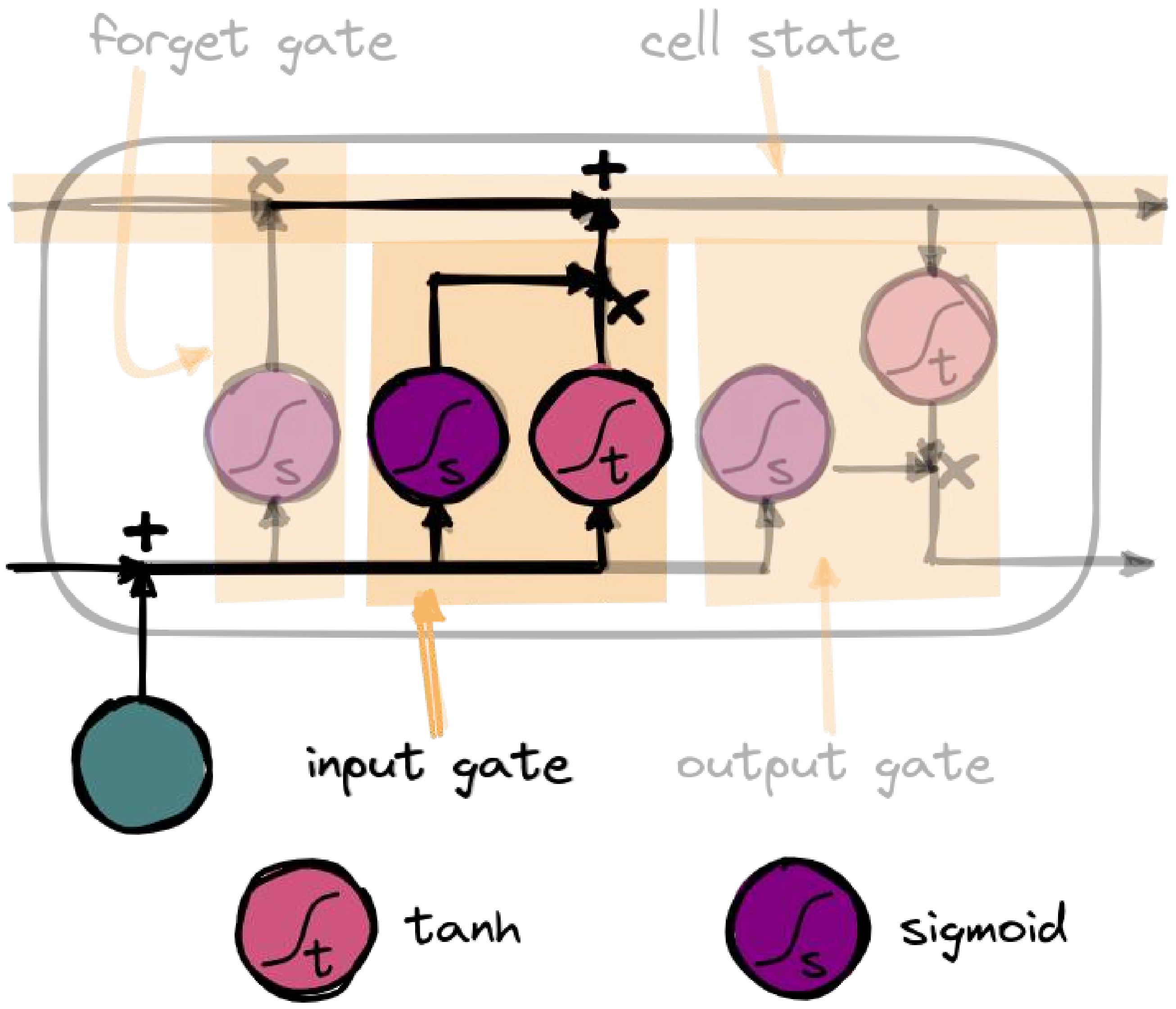
# LSTM: Forget gate

This gate decides what pieces of information should be thrown away or kept. Data from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1 — the closer to 0 means to forget, and the closer to 1 means to keep.



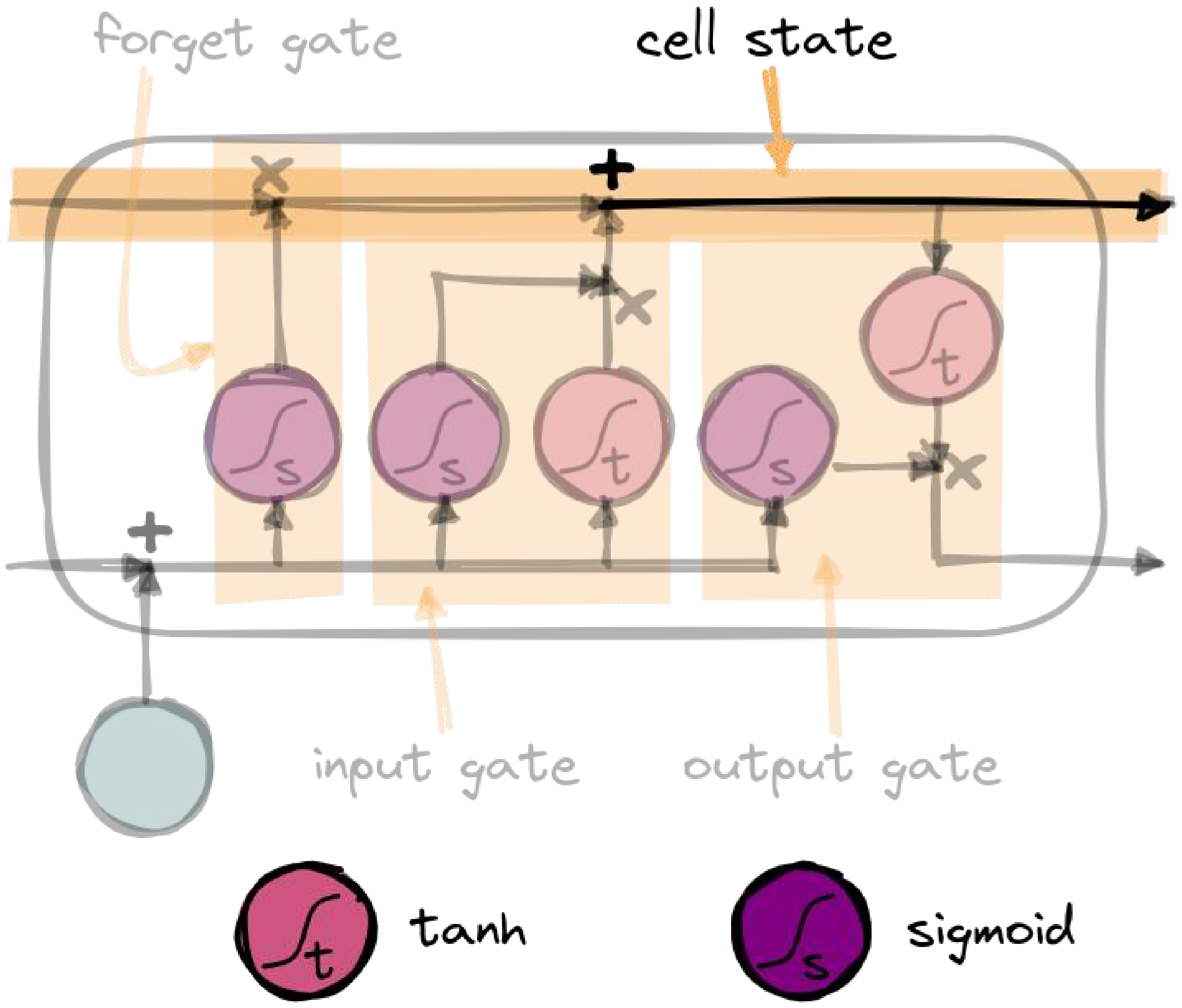
# LSTM: Input gate

To update the cell state, we have the input gate. First, pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values between 0 (not important) and 1 (important). Also, pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. You multiply the tanh output with the sigmoid output. The sigmoid output will decide which information to keep from the tanh output.



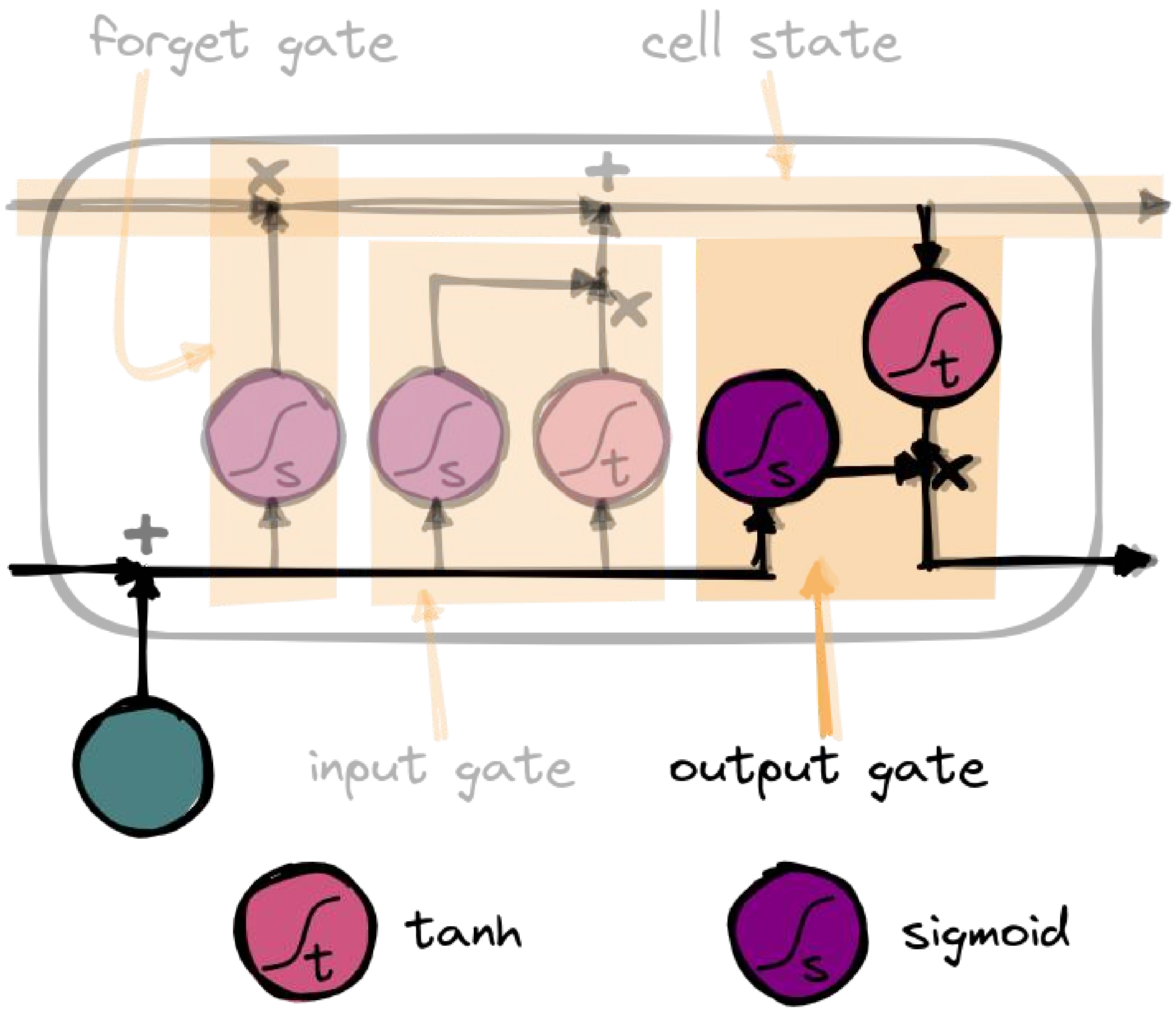
# LSTM: Cell state

We now have enough information to calculate the cell state. First, the cell state is pointwise multiplied with the forget gate output. This drops values in the cell state if it is multiplied by values near 0. Further, we pointwise add the result with the input gate. The update is the new cell state that represents relevant information. That gives us our new cell state.



# LSTM: Output gate

Last we have the output gate. The output gate decides what the next hidden state should be. The hidden state contains information on previous inputs and is used for predictions. First, pass the last hidden state and the current input into a sigmoid function. Then pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide the hidden state's information. The result is the new hidden state. The new cell state and the new hidden are then carried over to the next step.



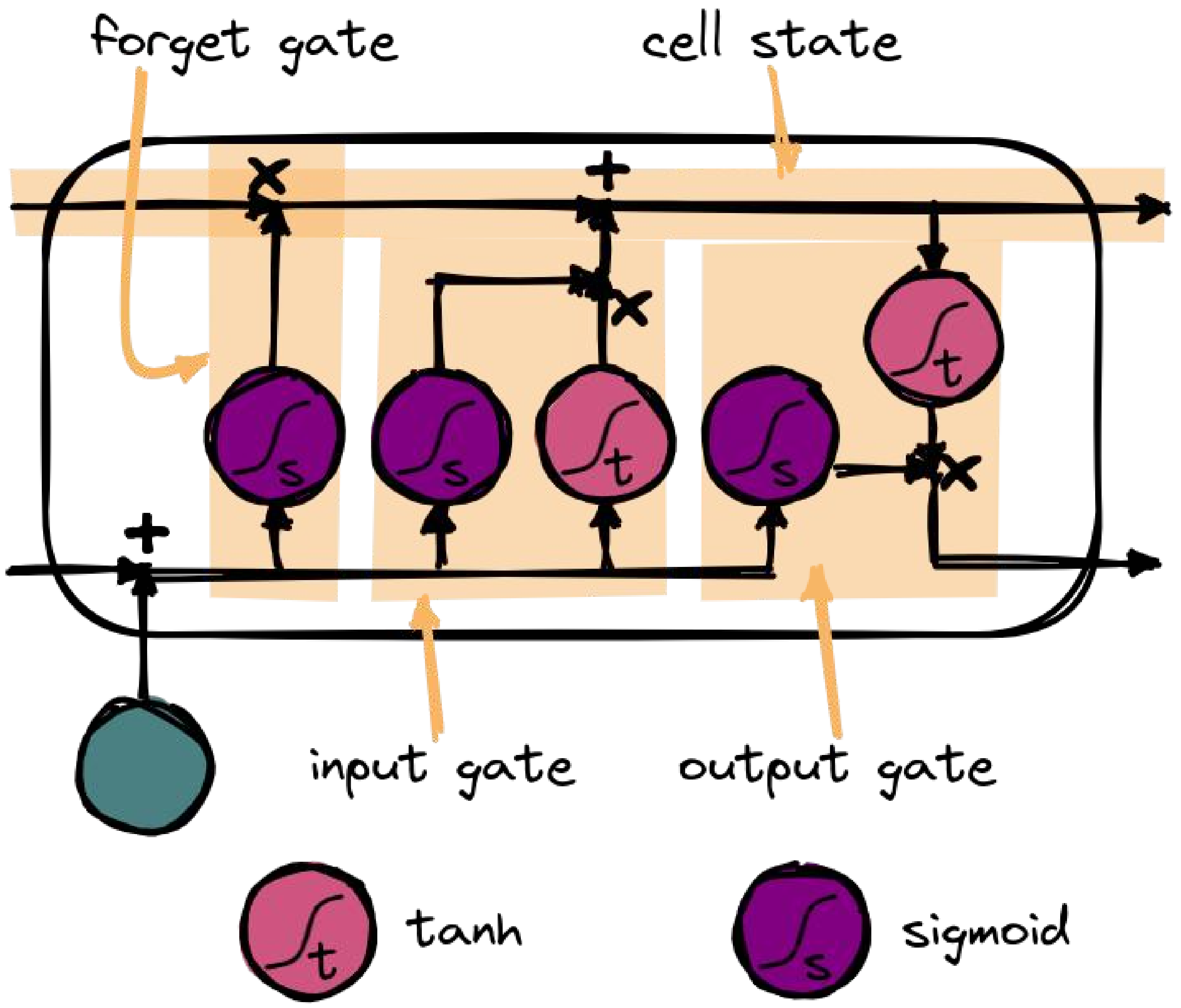
# LSTM: Overall

The forget gate decides what is relevant to keep from previous steps.

The input gate decides what information is relevant to add from the current step.

The output gate determines what the next hidden state should be.

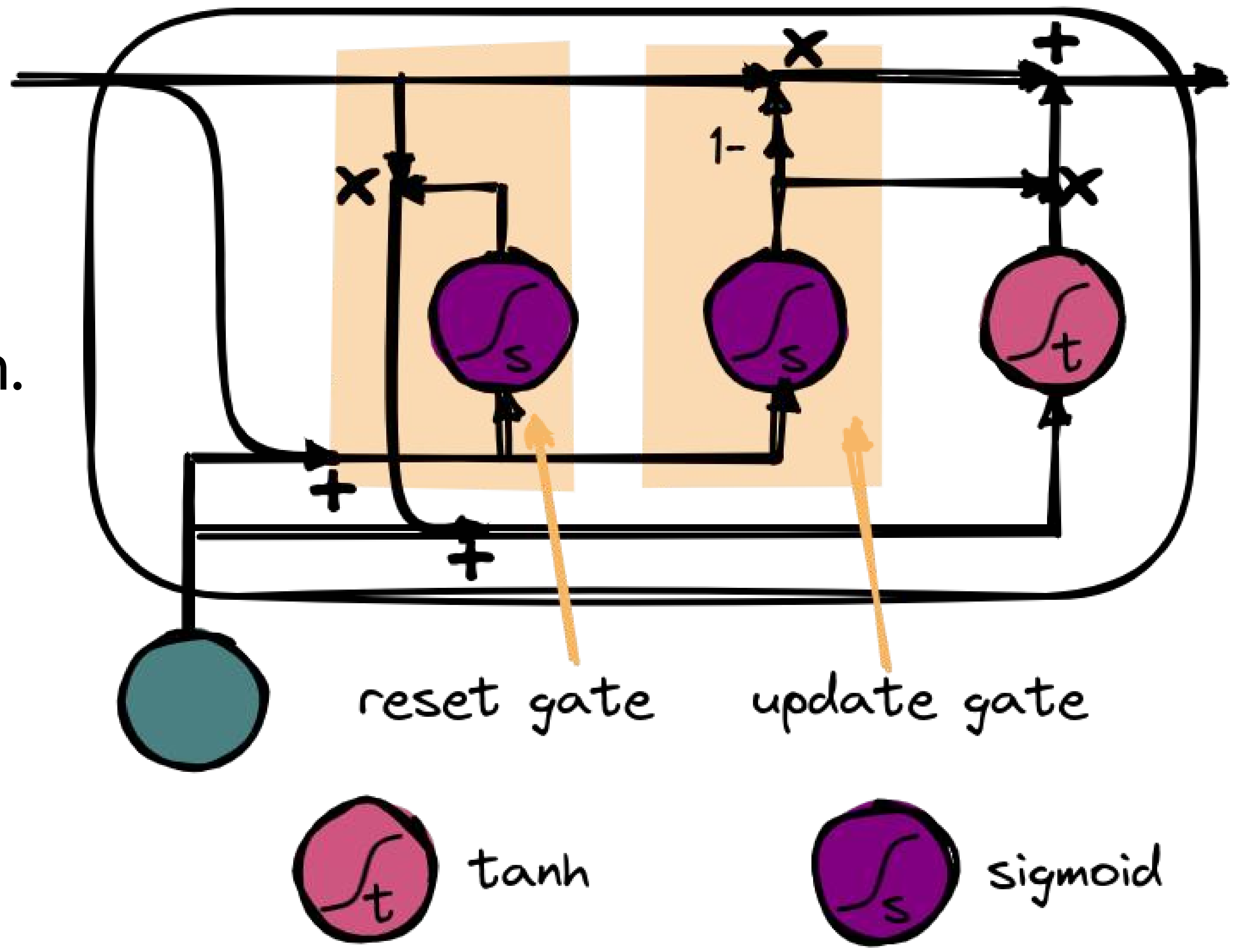
The cell state carries "important memory".



# Gated Recurrent Units (GRU)

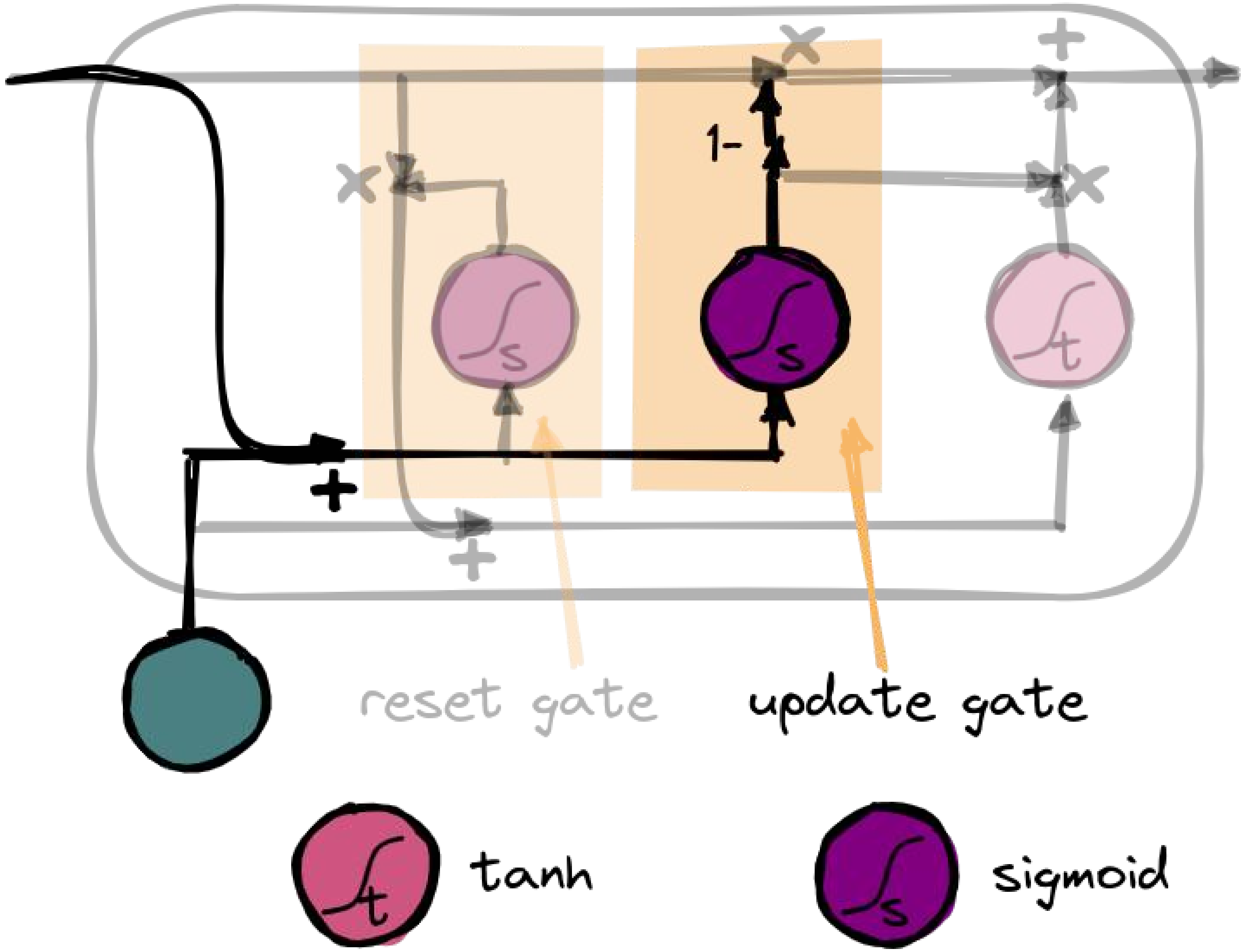
GRU is the newer generation of RNNs and is pretty similar to an LSTM.

GRUs got rid of the cell state and used the hidden state to transfer information. It also has two gates, a reset, and an update gate.



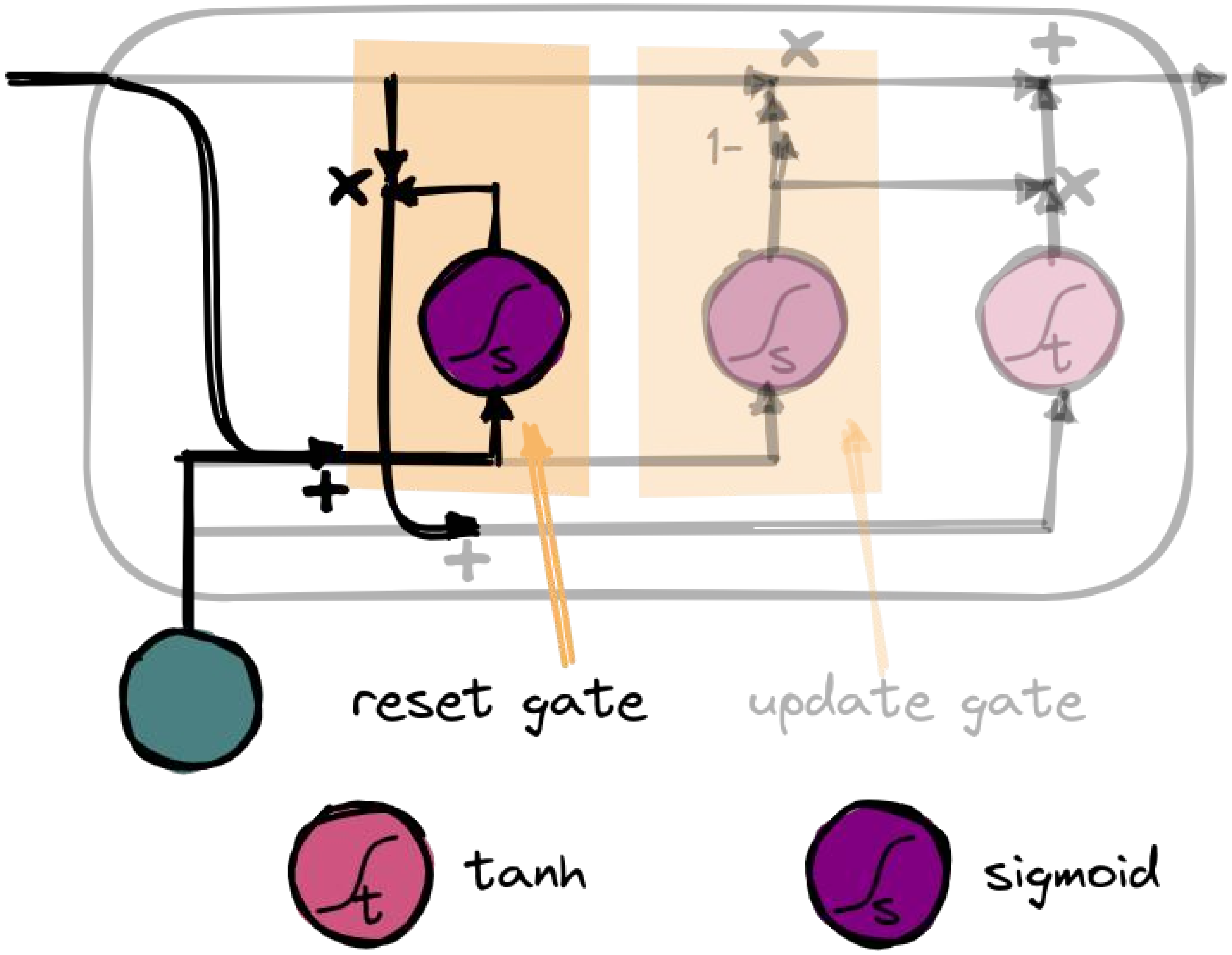
# GRU: Update gate

Update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.



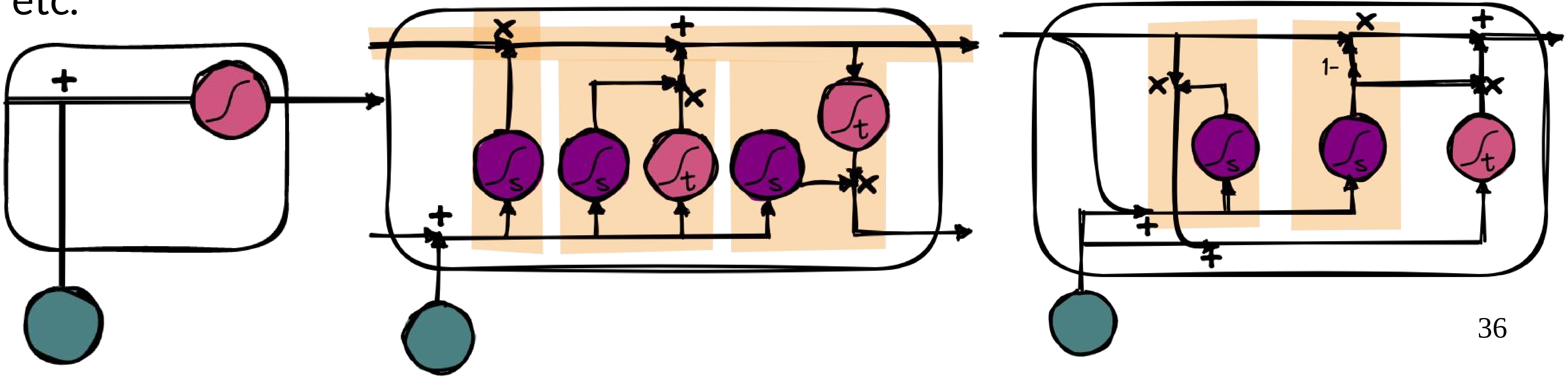
# GRU: Reset gate

The reset gate is another gate used to decide how much past information to forget.



# RNN: Summary

RNNs are suitable for processing sequence data for predictions but suffer from short-term memory. LSTMs and GRUs were created to mitigate short-term memory using mechanisms called gates. Gates are just neural networks that regulate the flow of information flowing through the sequence chain. LSTMs and GRUs are used in the state of the art deep learning applications like speech recognition, speech synthesis, natural language understanding, etc.



# RNN vs HMM

- RNNs have greater representational power

- RNNs do not make the Markov assumption and so, in theory, take into account long-term dependencies when modeling sequences

- RNNs may require an embedding layer for certain vector representations of their inputs (e.g., words encoded as one-hot vectors)