Previous lecture/practical

Any question?

Previous lecture/practical

Any question?

Recap: you will have to implement, train and evaluate CNN models during the final project —> Important to understand the underlying mechanisms…

Previous lecture/practical

Any question?

Recap: you will have to implement, train and evaluate CNN models during the final project —> Important to understand the underlying mechanisms…

I have updated the lecture/practical page with solutions to the different parts.

Parallel training in Deep Learning

Training batches

In the last practical you implemented train, val, and test **dataloaders**. An hyperparameter you had to pick was the **batch size,** i.e. the **number of samples to train on at each iteration**.

An **input tensor** from the data loader had the following shape: **(B, C, H, W)**
B: batch size
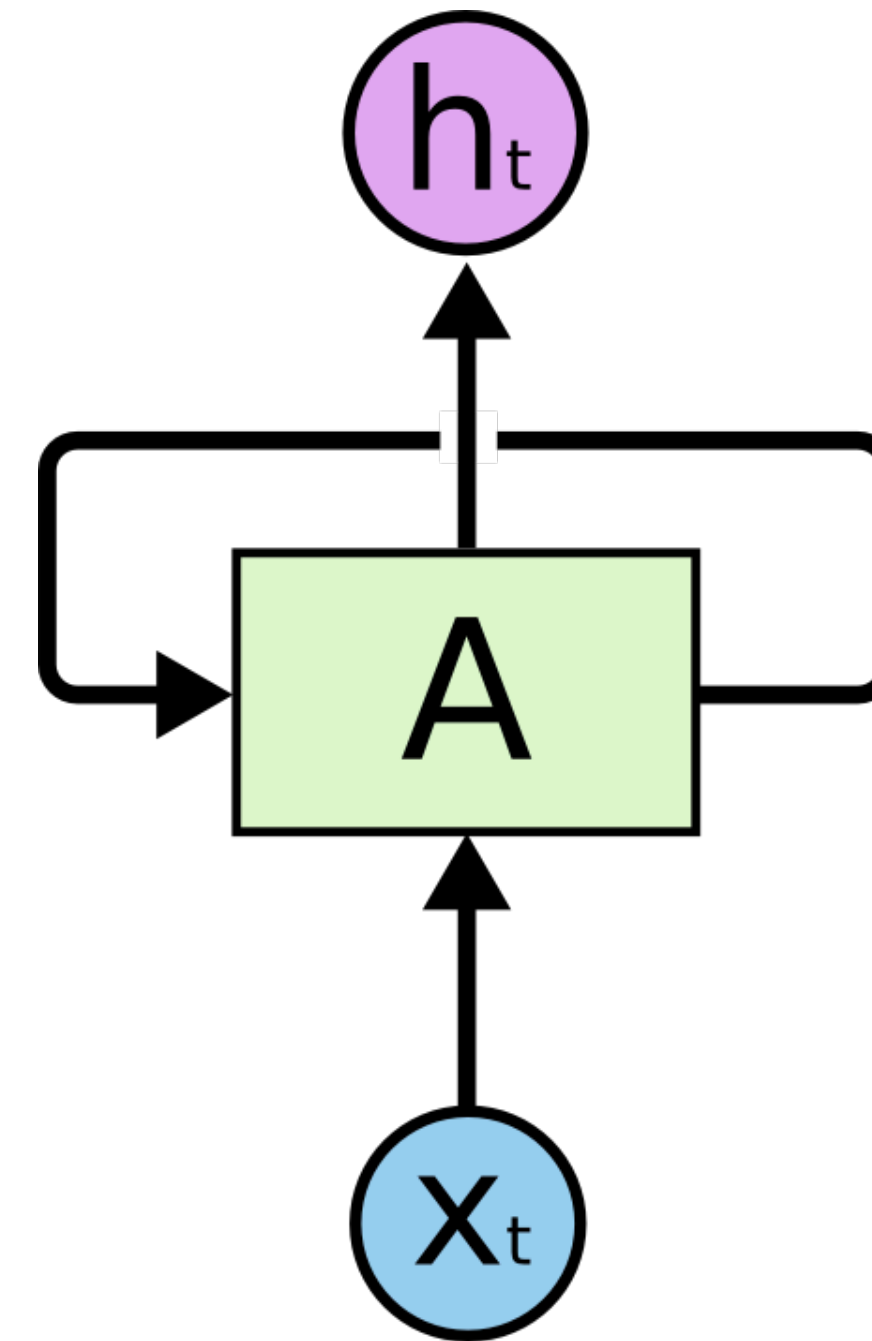C: number of channels (C=3 for an image)
H: tensor height
W: tensor width

If you input this tensor to your CNN, **all B inputs will be processed in parallel in a single forward pass: (B, C, H, W) —> CNN —> (B, C', H', W')** where C', H', W' are output feature map dimensions.

Introduction to Deep Learning

Lecture 2: Recurrent Neural Networks

Course Overview

1. Convolutional Neural Networks (1h lecture + 3h practical)

2. Recurrent Neural Networks (1h lecture + 3h practical)

3. Transformers and Attention (1h lecture + 3h practical)

4. Project (12h)

Course Overview

1. Convolutional Neural Networks (1h lecture + 3h practical)

2. <u>Recurrent Neural Networks</u> (1h lecture + 3h practical)

3. Transformers and Attention (1h lecture + 3h practical)

4. Project (12h)

Useful resources about RNNs

- A great blog post: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Hochreiter et al., Long short-term memory, Neural computation 1997

- Cho et al., Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, arXiv 2014

Why do we need memory?

When dealing with **temporal data** (text, videos, robotics, etc.), keeping track of the past becomes important...
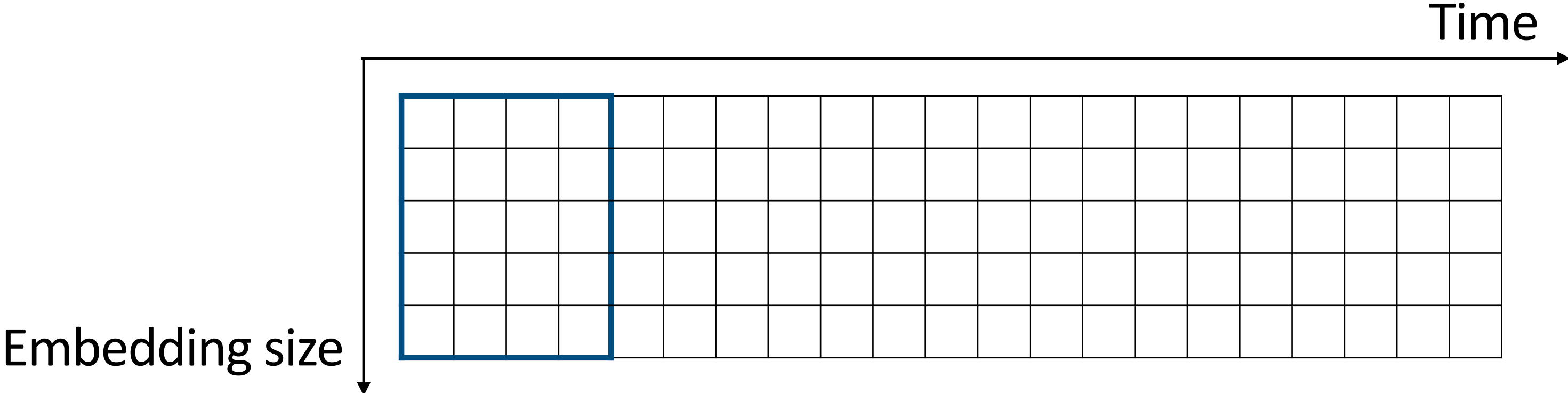
Inductive bias

When dealing with **temporal/sequential** data, **keeping track of essential information inside a vectorial memory** seems to be a good idea!

This is the main idea behind recurrent neural networks (RNNs)... We call this memory the **hidden state**.

Processing temporal information with convolutions

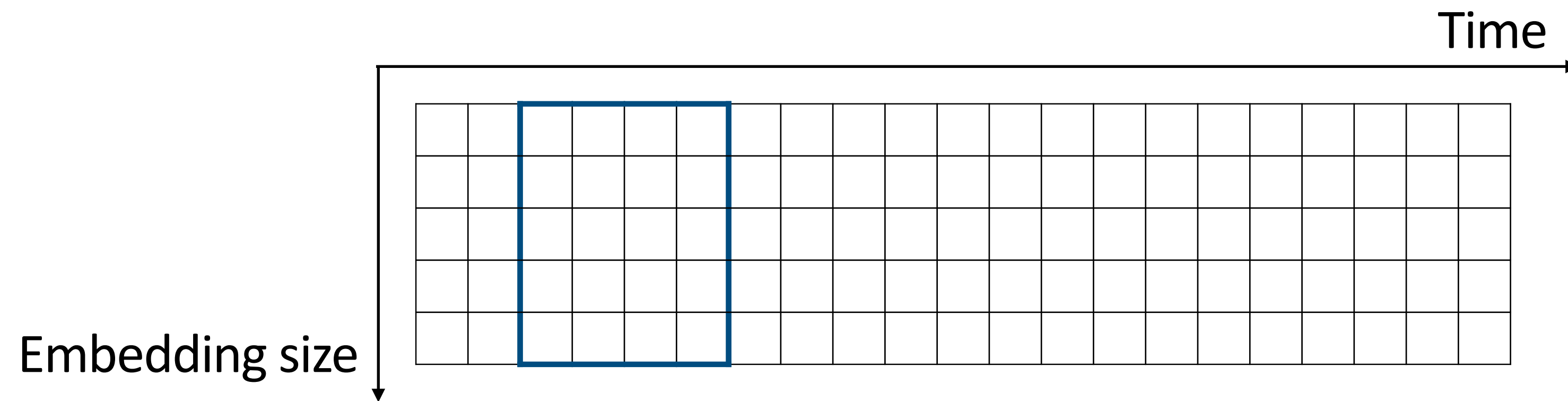In the previous lecture, we were sliding convolution kernels along image dimensions.

Can we **slide along time dimension**? **Yes**, we can!

Processing temporal information with convolutions

In the previous lecture, we were sliding convolution kernels along image dimensions.

Can we **slide along time dimension**? **Yes**, we can!

Processing temporal information with convolutions

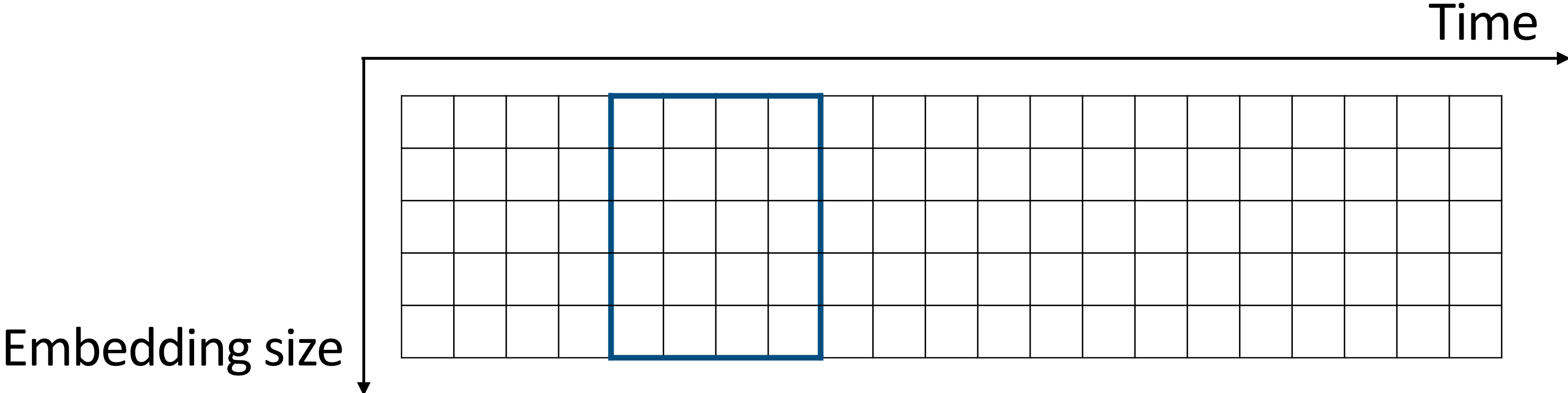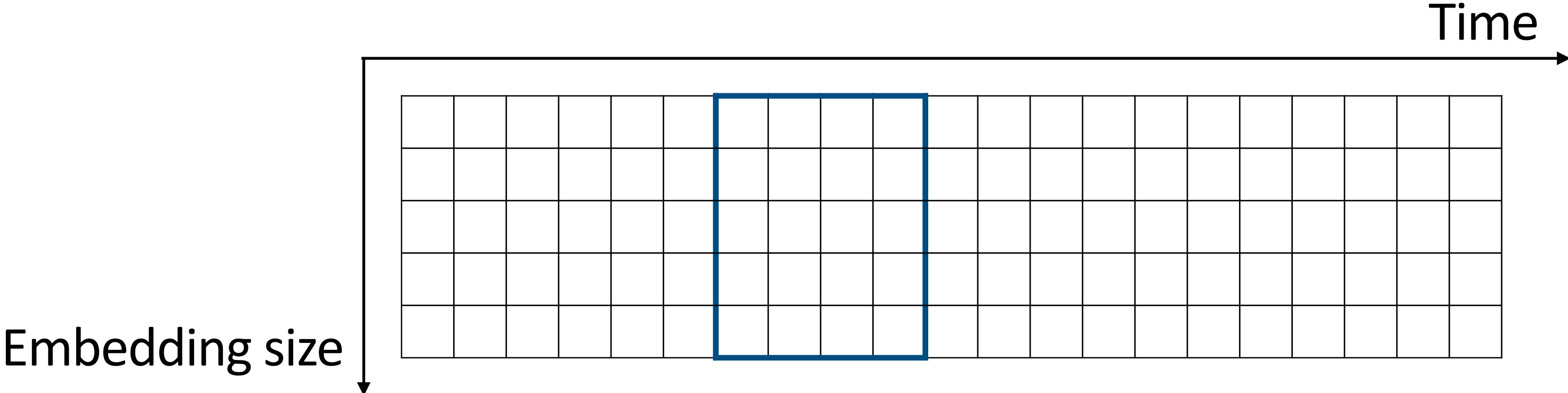In the previous lecture, we were sliding convolution kernels along image dimensions.

Can we **slide along time dimension**? **Yes**, we can!

Processing temporal information with convolutions

In the previous lecture, we were sliding convolution kernels along image dimensions.

Can we **slide along time dimension**? **Yes**, we can!
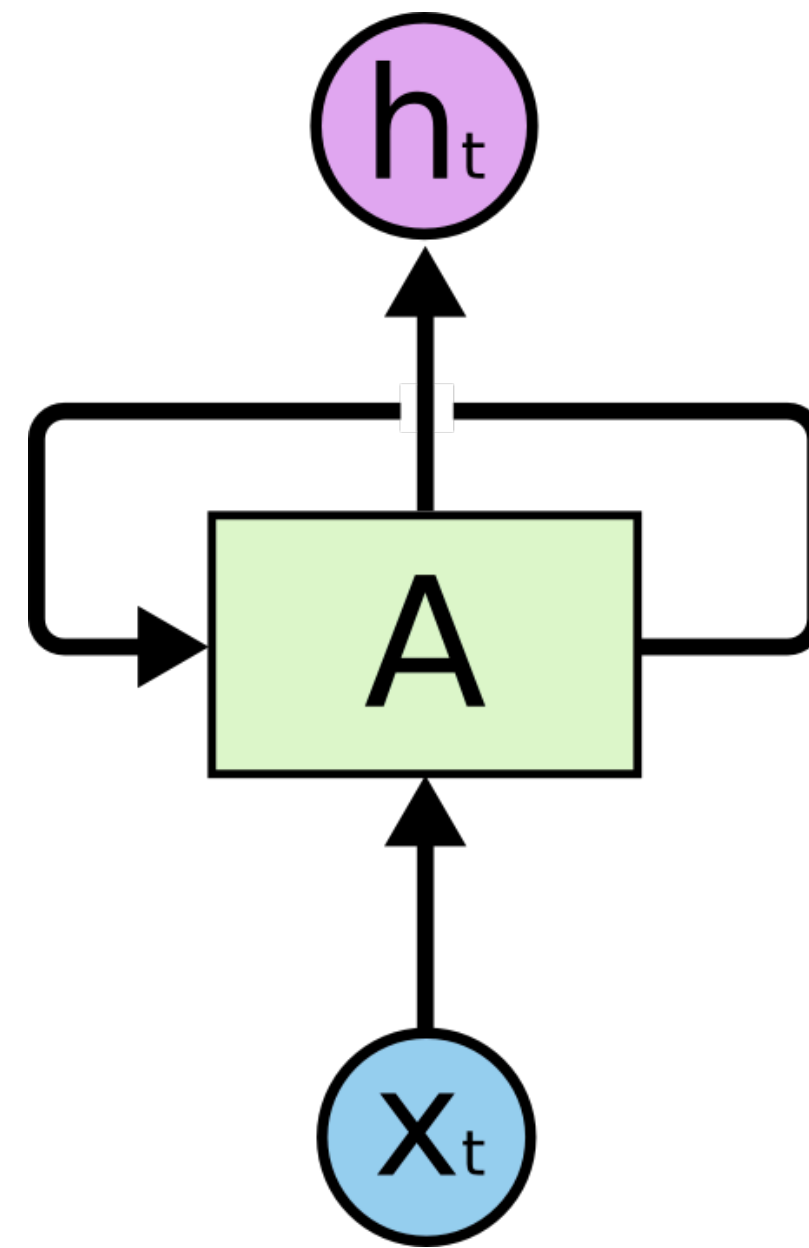
Time

Embedding size

Recurrent Neural Networks

All figures in the next slides will be taken from the **excellent** blog post by Chris Olah: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
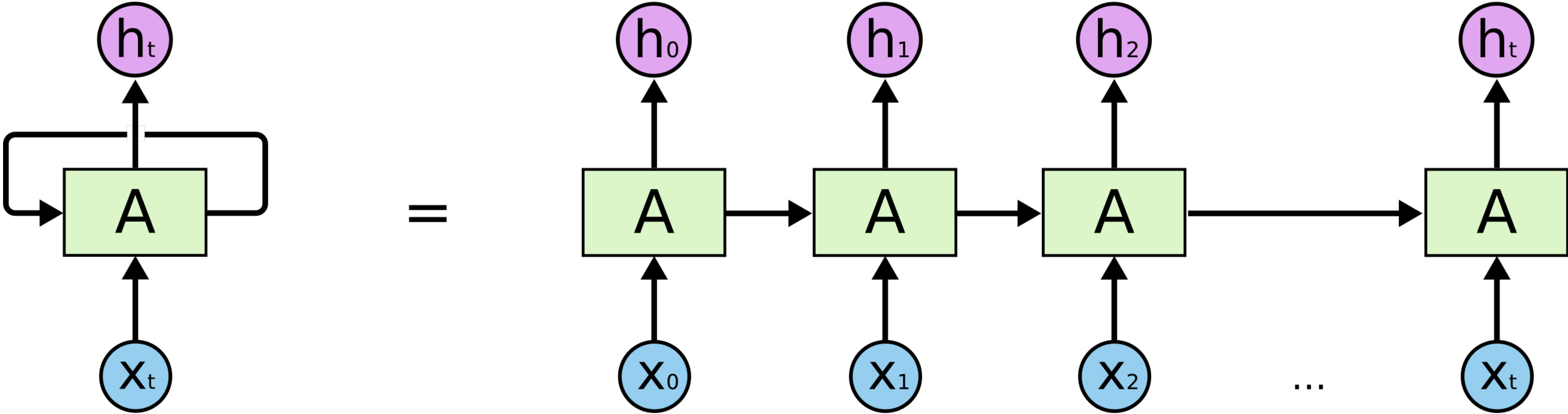
Recurrent Neural Networks — Recurrent Unit

Our unit A deals with **sequential** data: at time t, it is fed with input X_t and outputs a latent representation h_t, but **not only**: **it is also fed with an internal representation from the past step** $t - 1$.
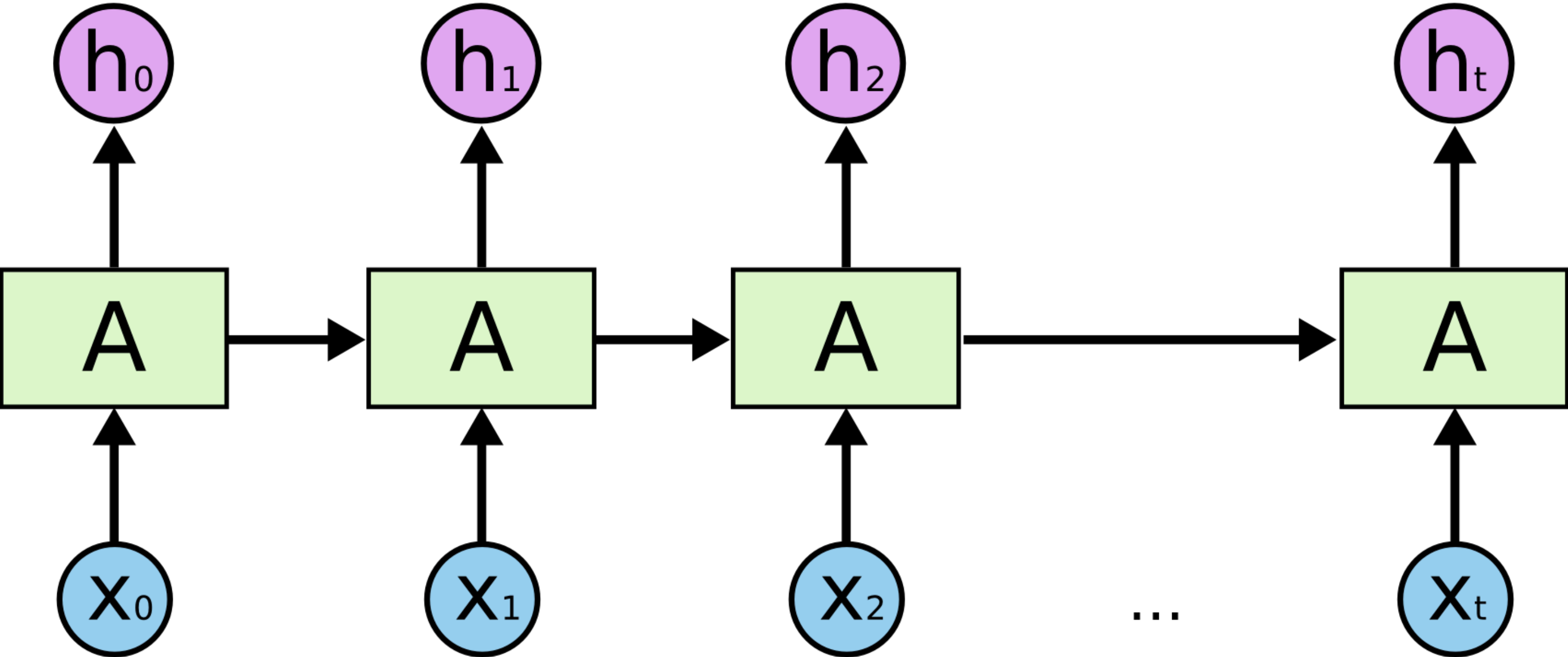
Recurrent Neural Networks — Recurrent Unit

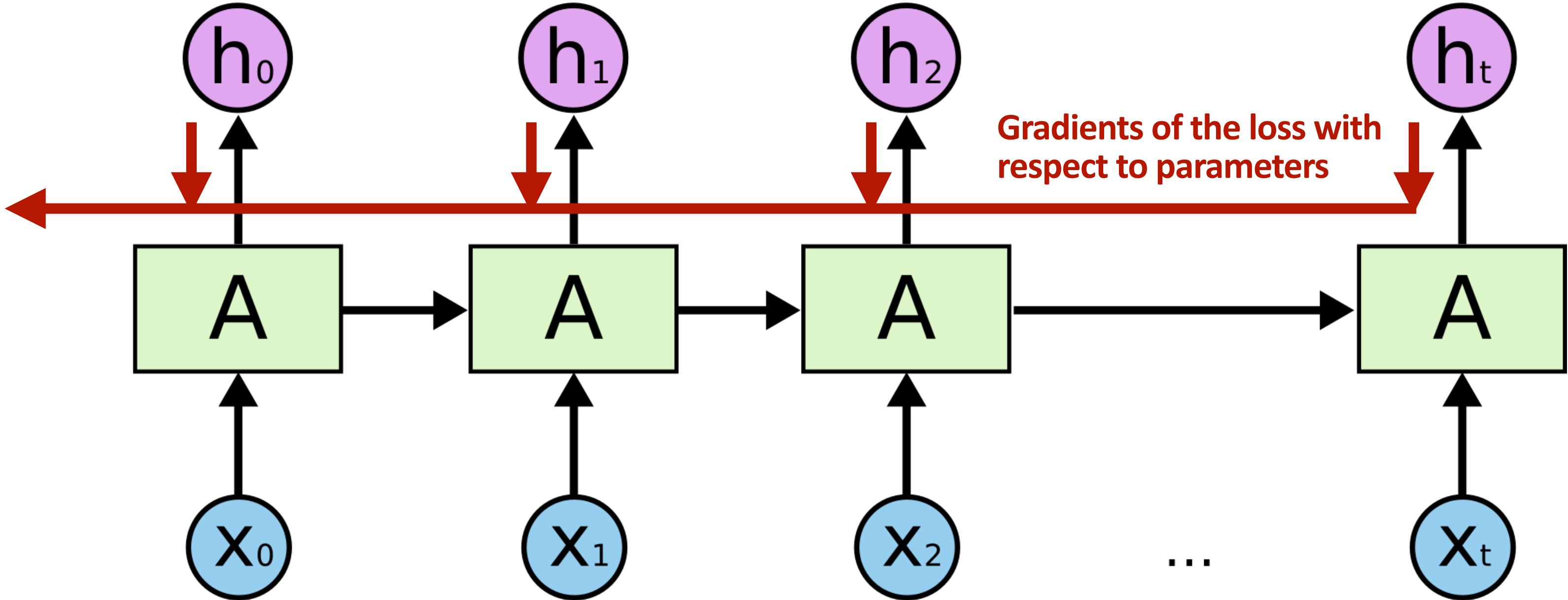Let's **unroll** the process **along the time axis**!

RNNs are trained with Backpropagation Through Time (BPTT):

RNNs are trained with Backpropagation Through Time (BPTT):

Gradients of the loss with respect to parameters

Recurrent Neural Networks — Issues with standard RNNs

Vanilla RNNs tend to be **hard to train** and suffer from **shortcomings**:

- In practice, RNNs **struggle to memorise long-term context**, i.e. information that appeared long time ago in the sequence.

- **Vanishing** and/or **exploding gradients**: small gradients vanish and high gradients explode respectively over long time ranges.

Recurrent Neural Networks — Issues with standard RNNs

Vanilla RNNs tend to be **hard to train** and suffer from **shortcomings**:

- In practice, RNNs **struggle to memorise long-term context**, i.e. information that appeared long time ago in the sequence.

- **Vanishing** and/or **exploding gradients**: small gradients vanish and high gradients explode respectively over long time ranges.

➡ New approaches based on gating mechanisms were introduced.

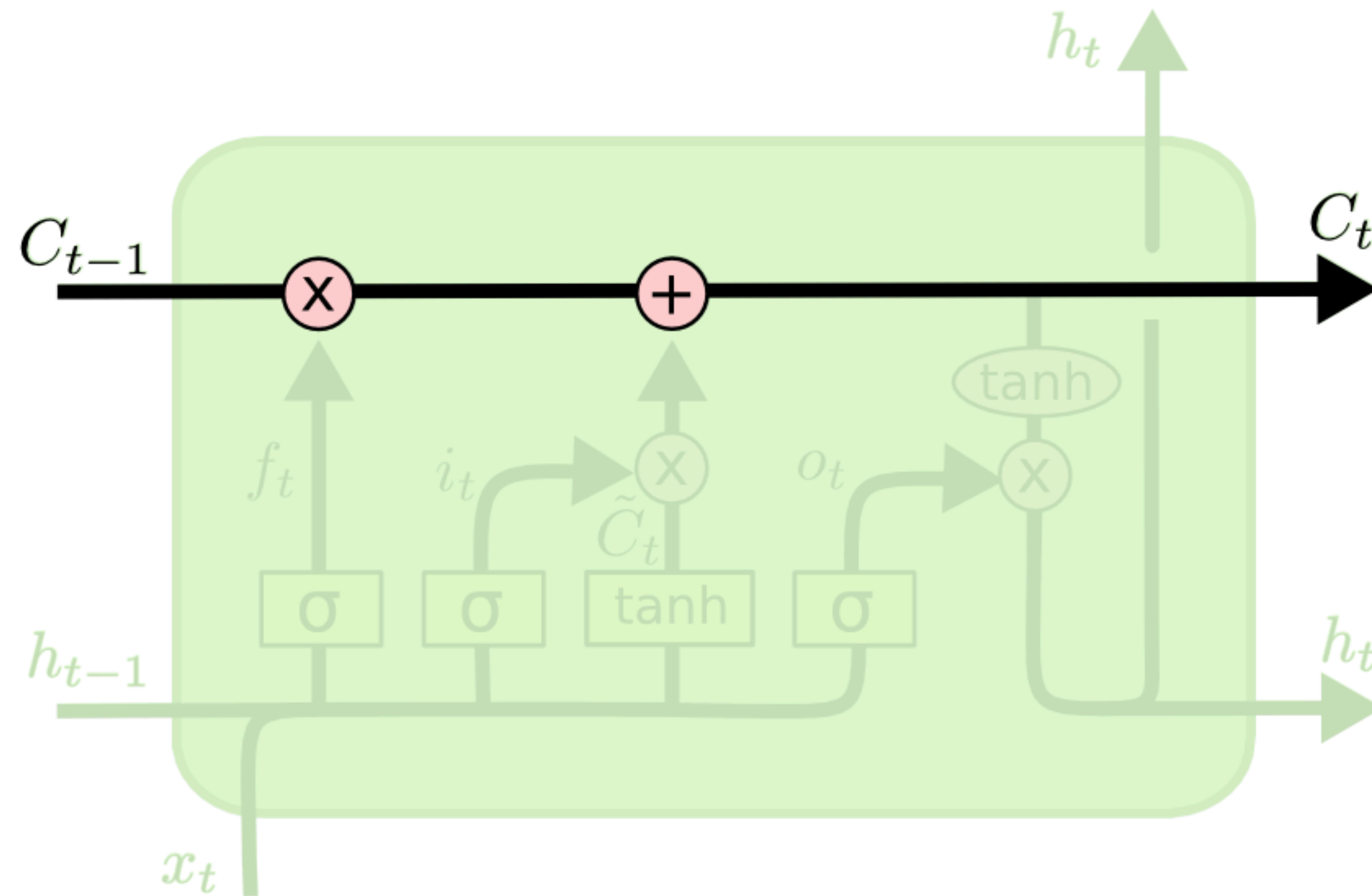# LSTM — Starting from a standard RNN...
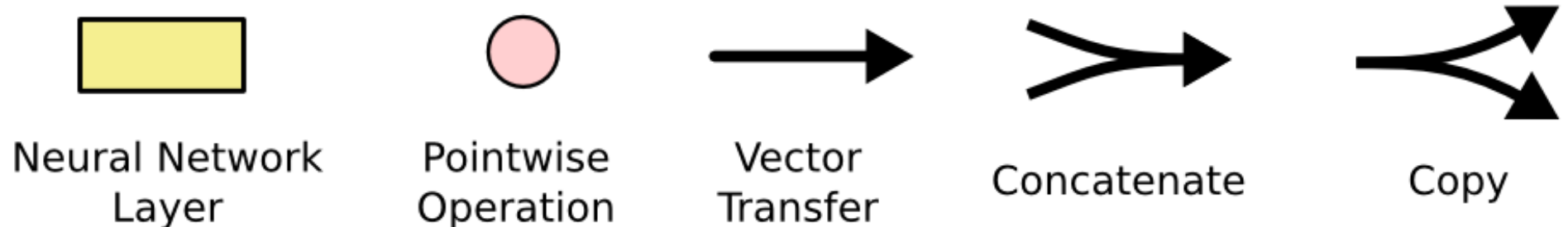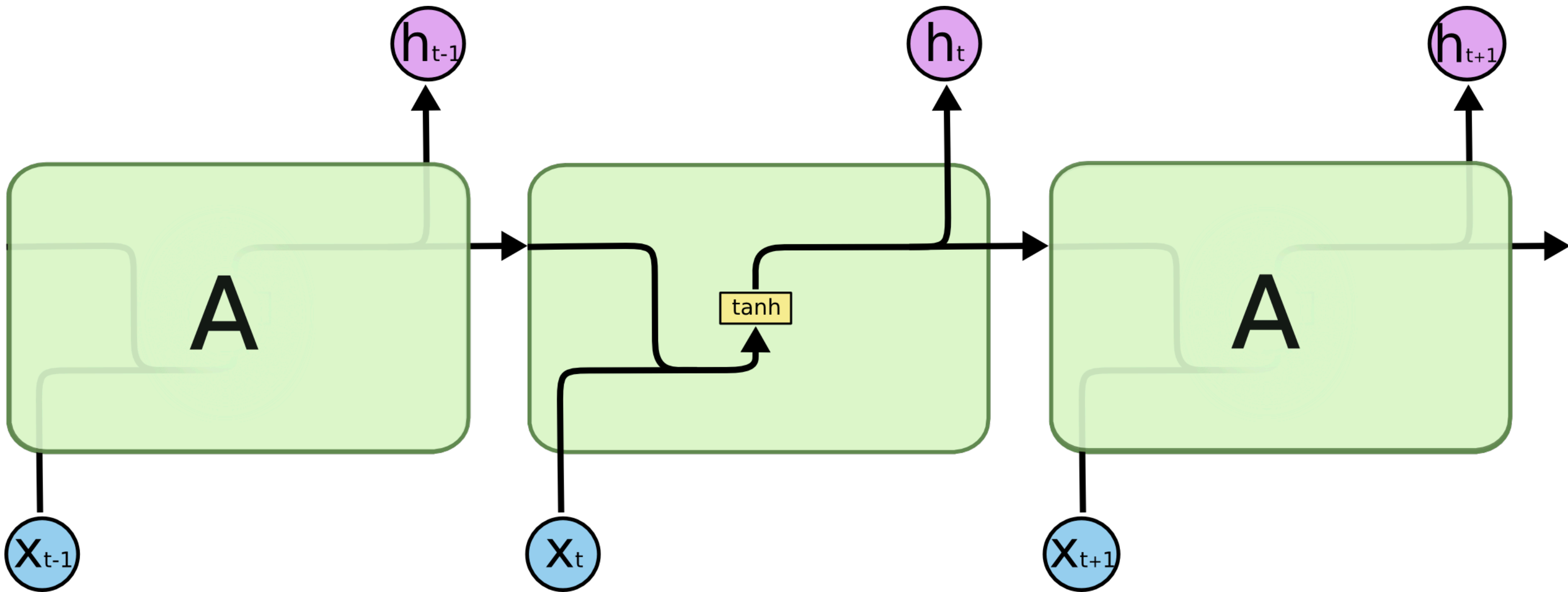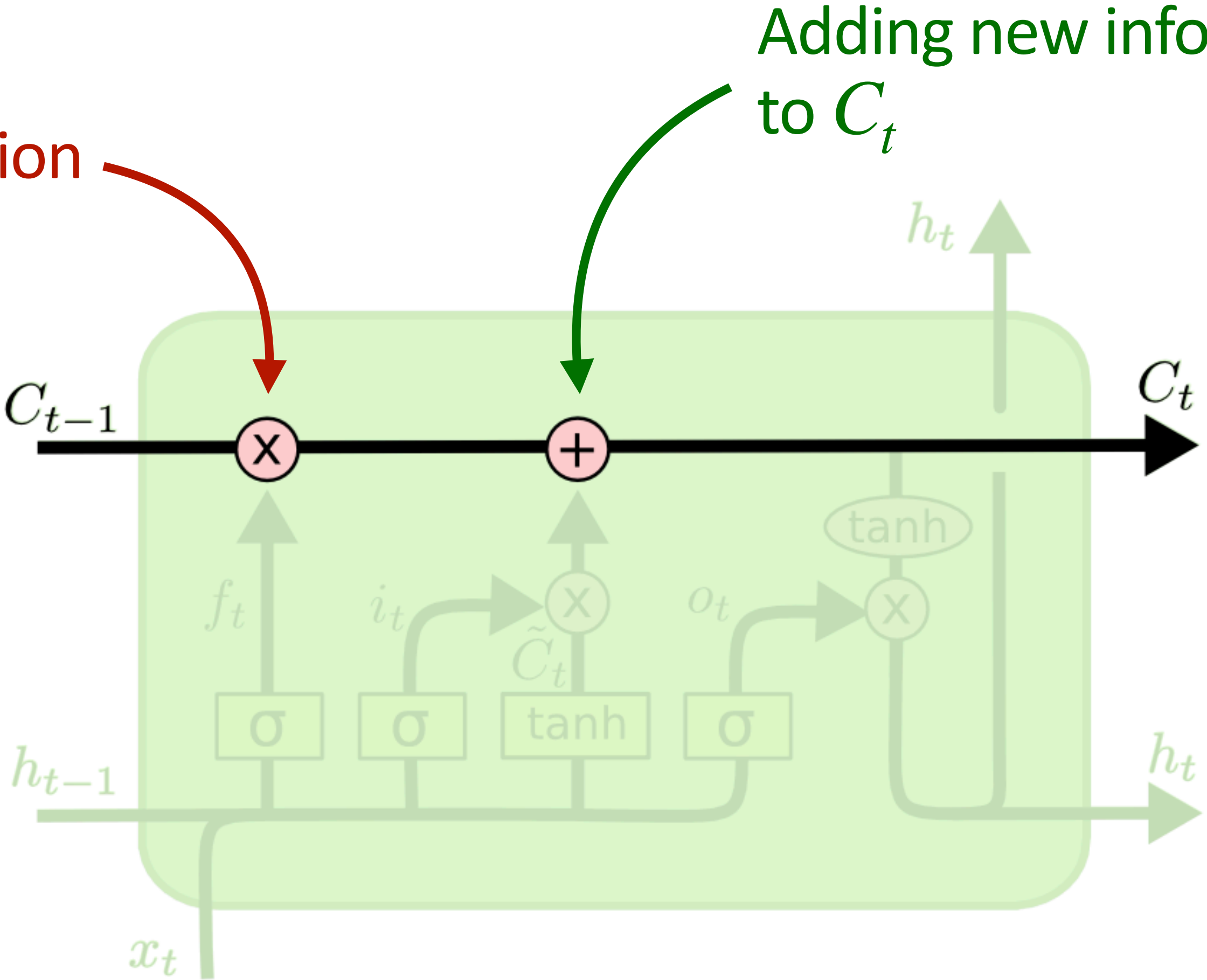
LSTM — ... to the LSTM architecture

Neural Network Layer · Pointwise Operation · Vector Transfer · Concatenate · Copy
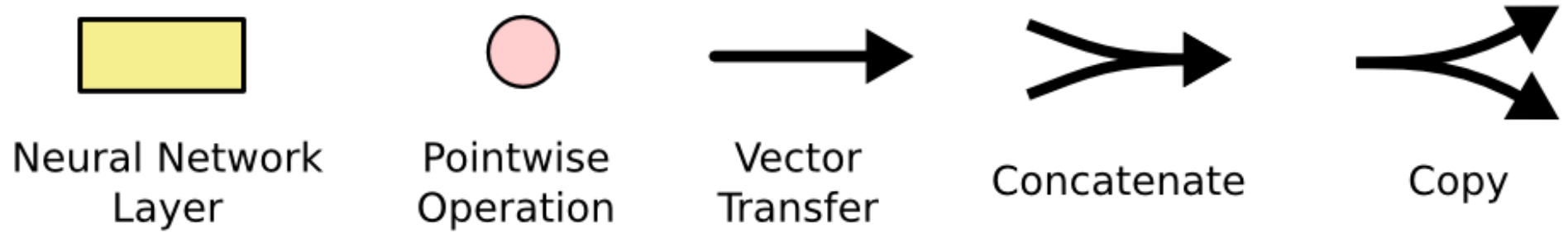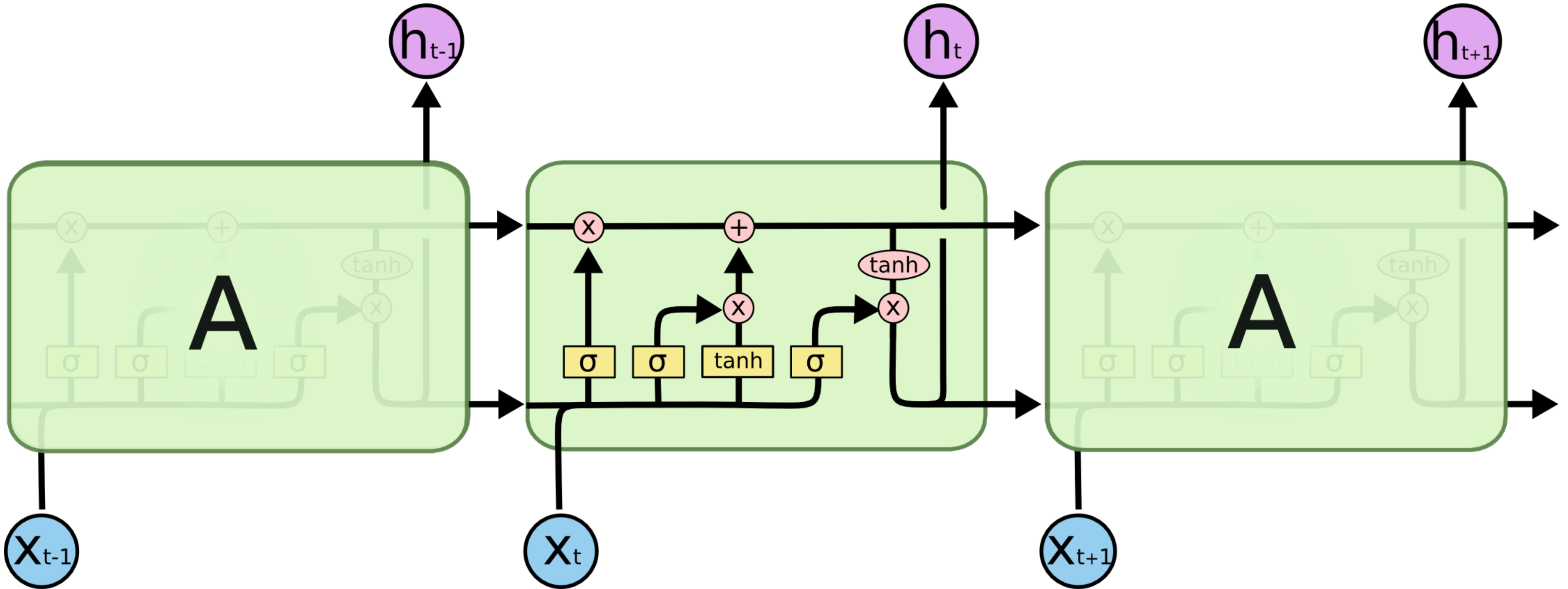
The Cell State C_t **stores the information we want to remember**.

LSTM — The Cell State
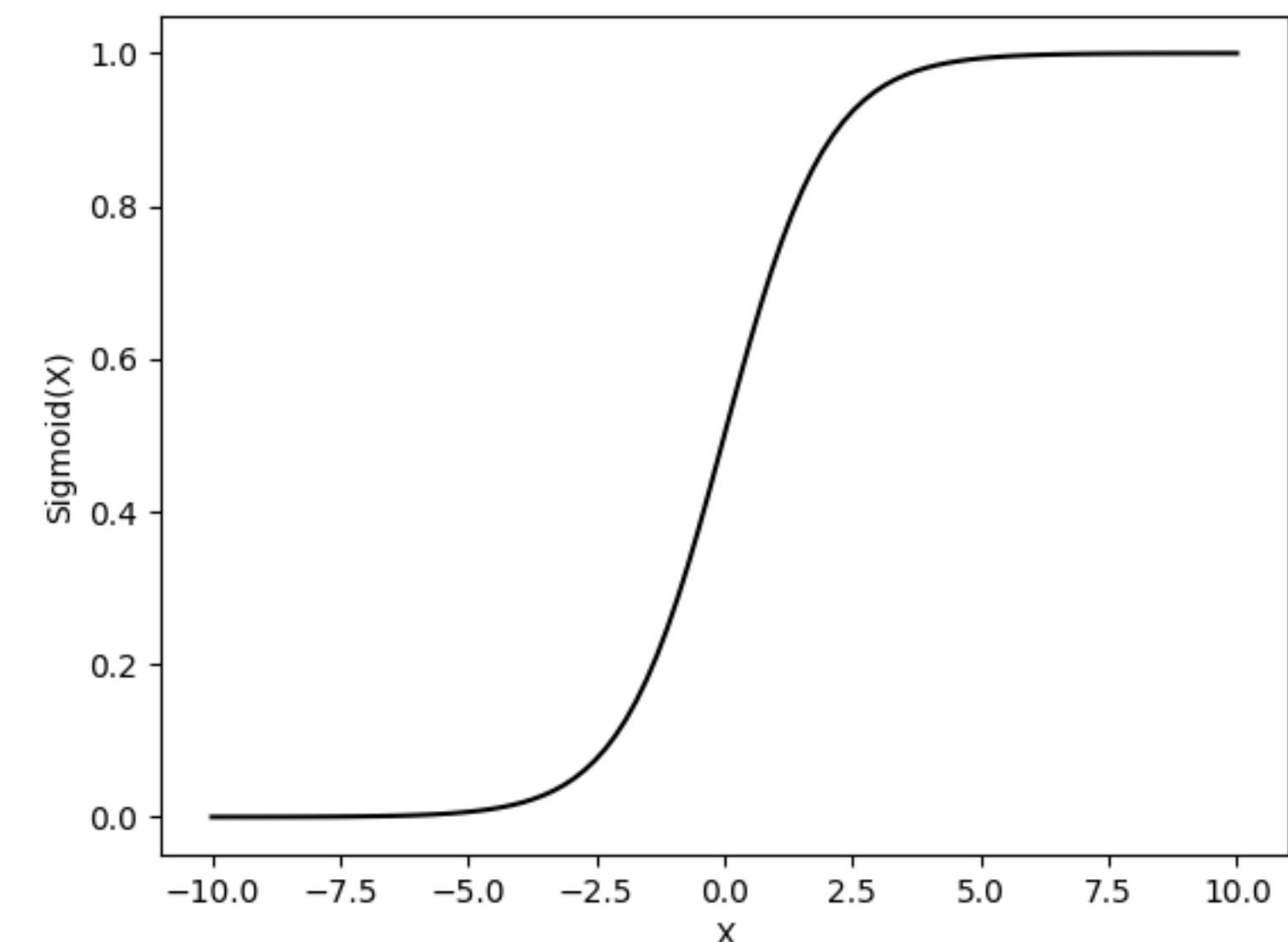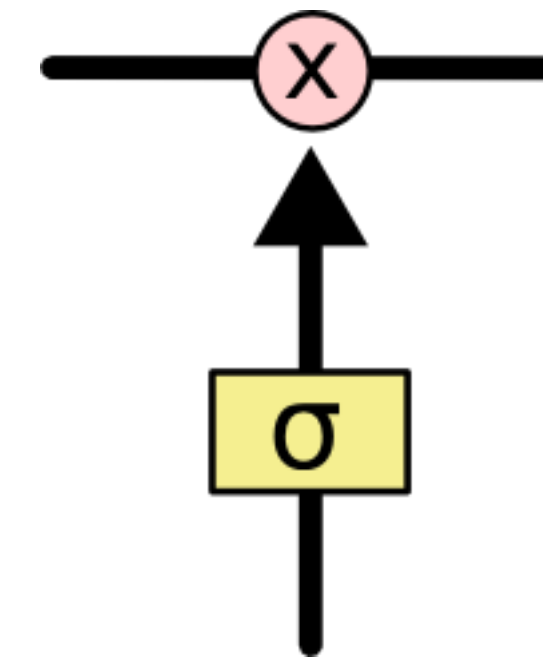
LSTM — Gates

A **gate** is a mechanism for deciding whether or not to let information go through. It is composed of a **sigmoid** function and a **pointwise multiplication** operation.

The sigmoid outputs a value between 0 and 1:
0 —> "**don't let any information** go through"
1 —> "**let all information** go through"

LSTM — The Forget Gate: What do we want to forget?

The Forget Gate **decides what information to forget from the Cell State**. From $h_{t-1} \in \mathbb{R}^d$ and $x_t \in \mathbb{R}^d$, it predicts **a scalar between 0 and 1 for each dimension of the Cell State**. The whole vector is $f_t \in \mathbb{R}^d$.
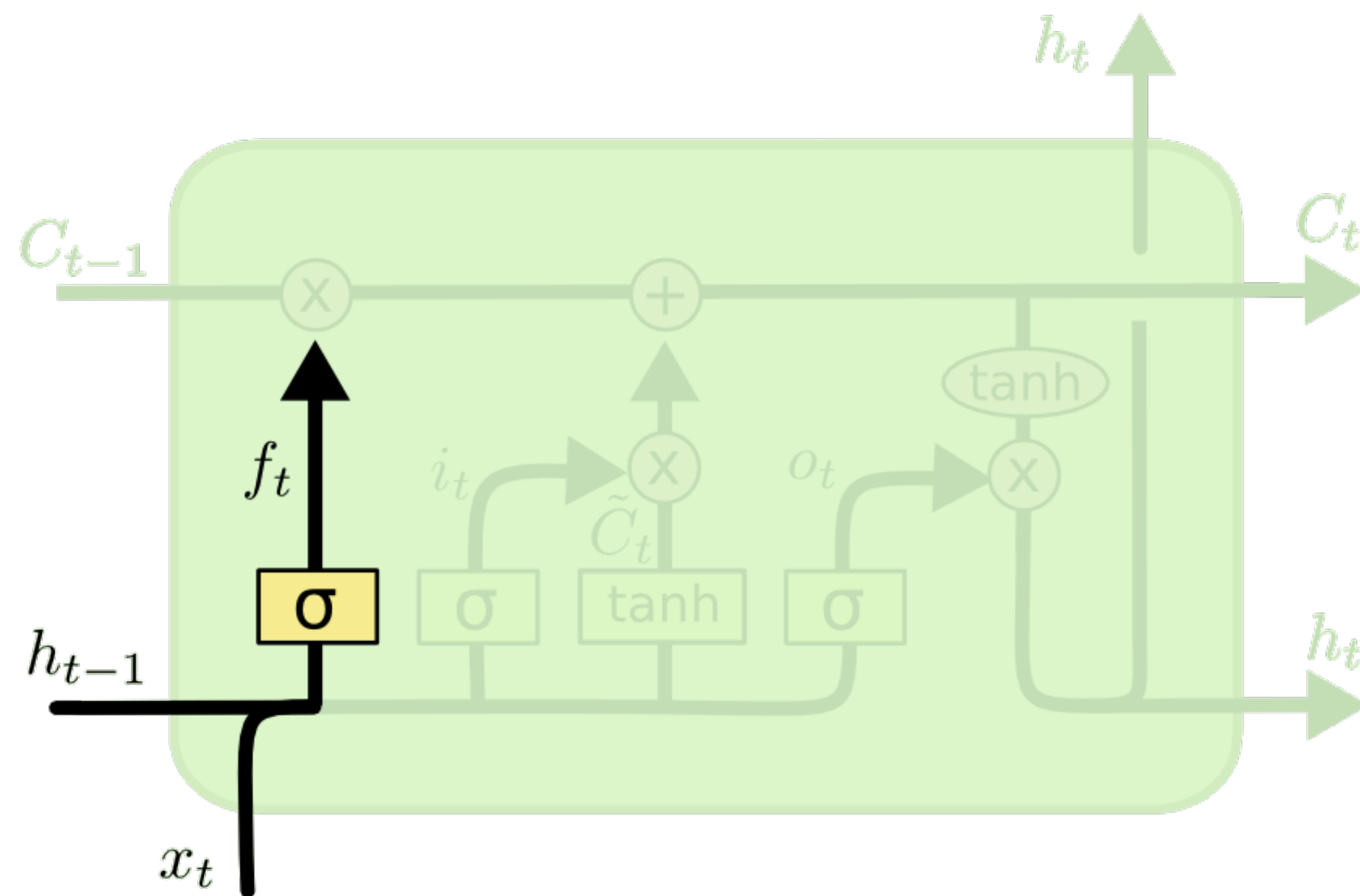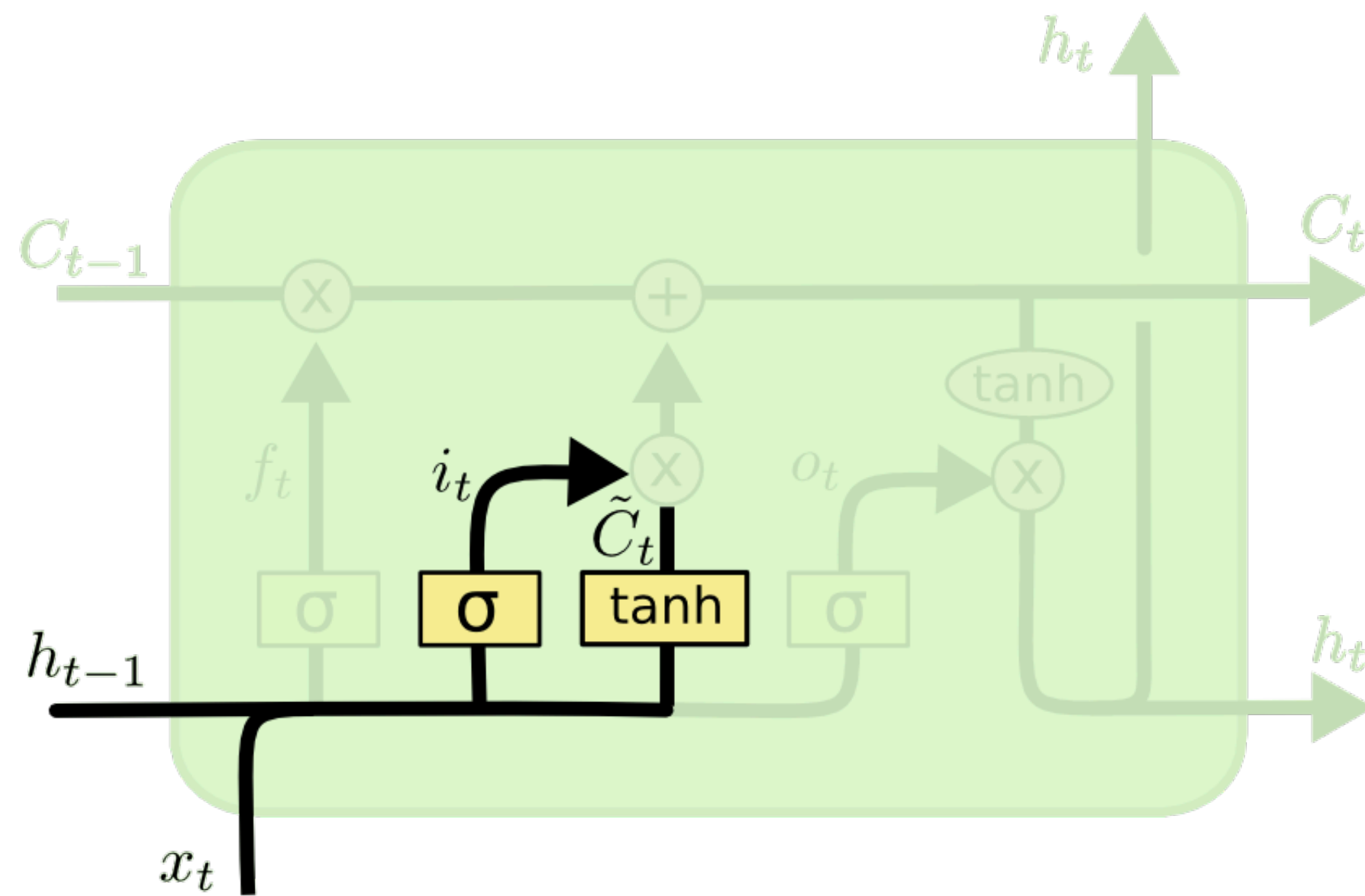

Weights of the linear layer

Bias term

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

Sigmoid activation function

The Input Gate **decides what channels in the Cell State to update** by predicting the i_t vector. Another linear layer followed by a tanh activation function outputs **update candidates** $\tilde{C}_t$.
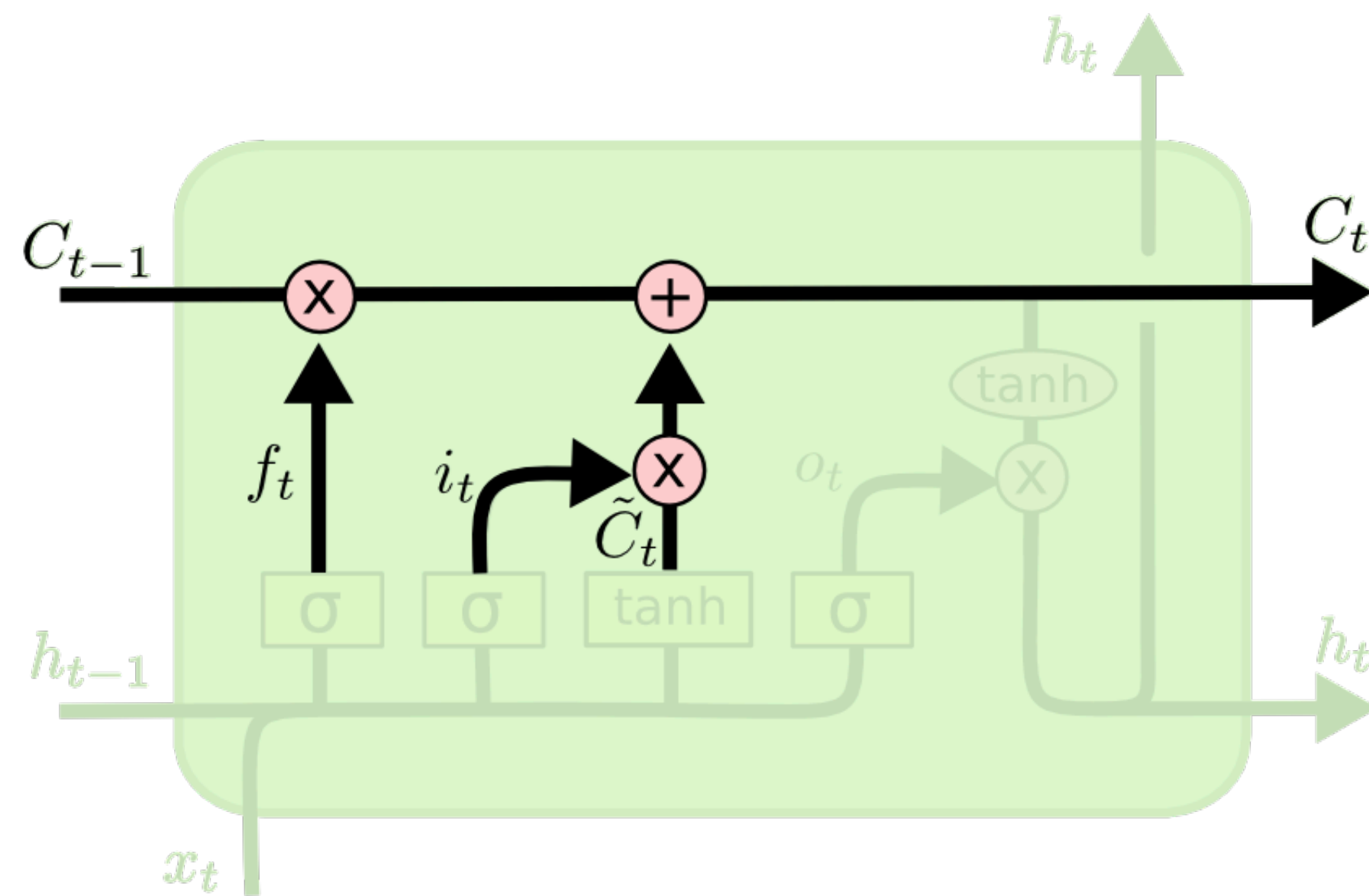
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

LSTM — Modifying the Cell State

We multiply C_{t-1} with f_t to forget channels we selected with the Forget Gate.

We then add $i_t \tilde{C}_t$, i.e. new candidates scaled by how much to update them, as decided by the Input Gate.

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

The output is a **filtered version of** C_t. **Another gate** takes h_{t-1} and x_t as inputs, and outputs the vector o_t. The latter **selects channels in** C_t that was previously passed through a tanh function.
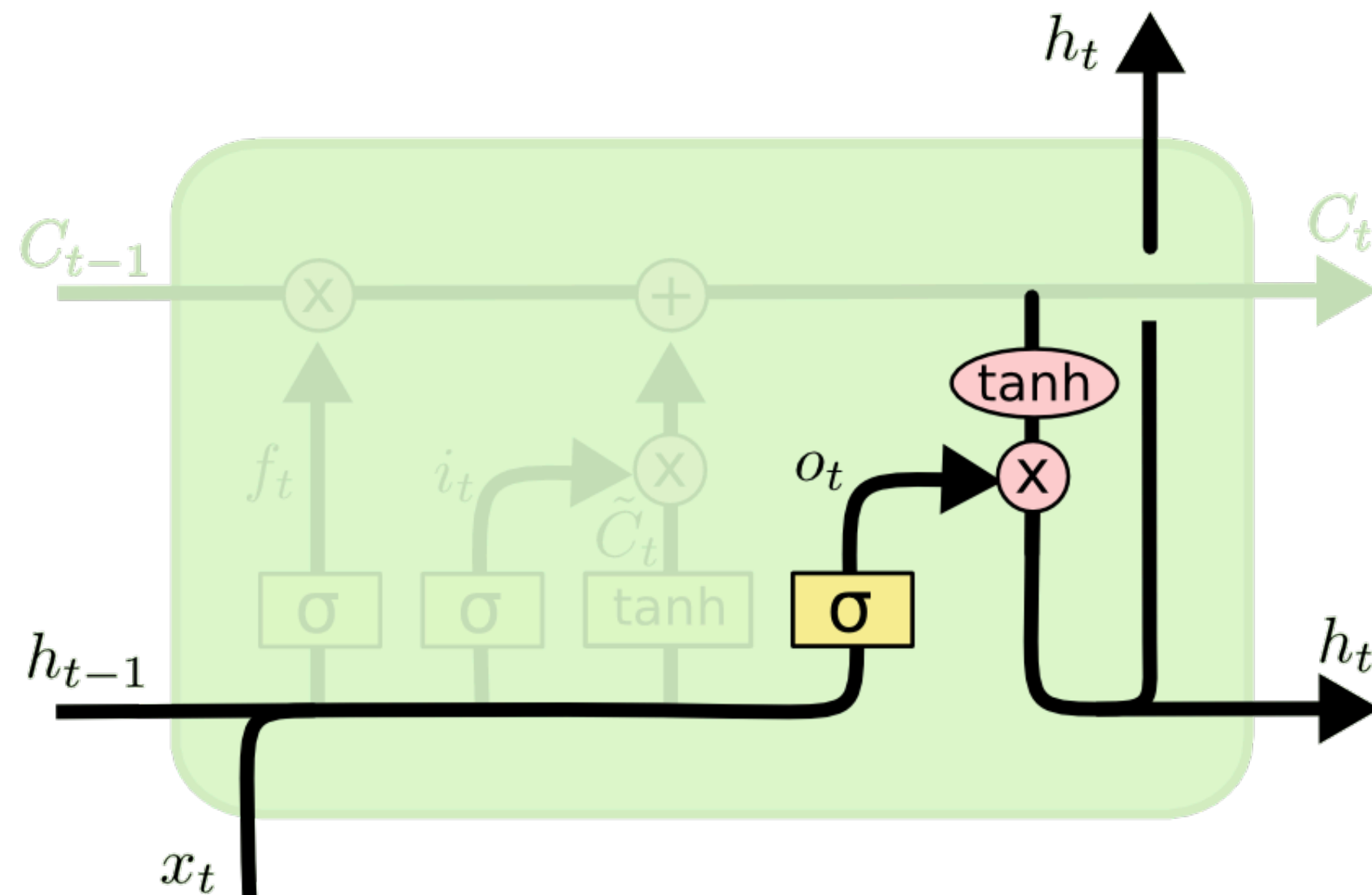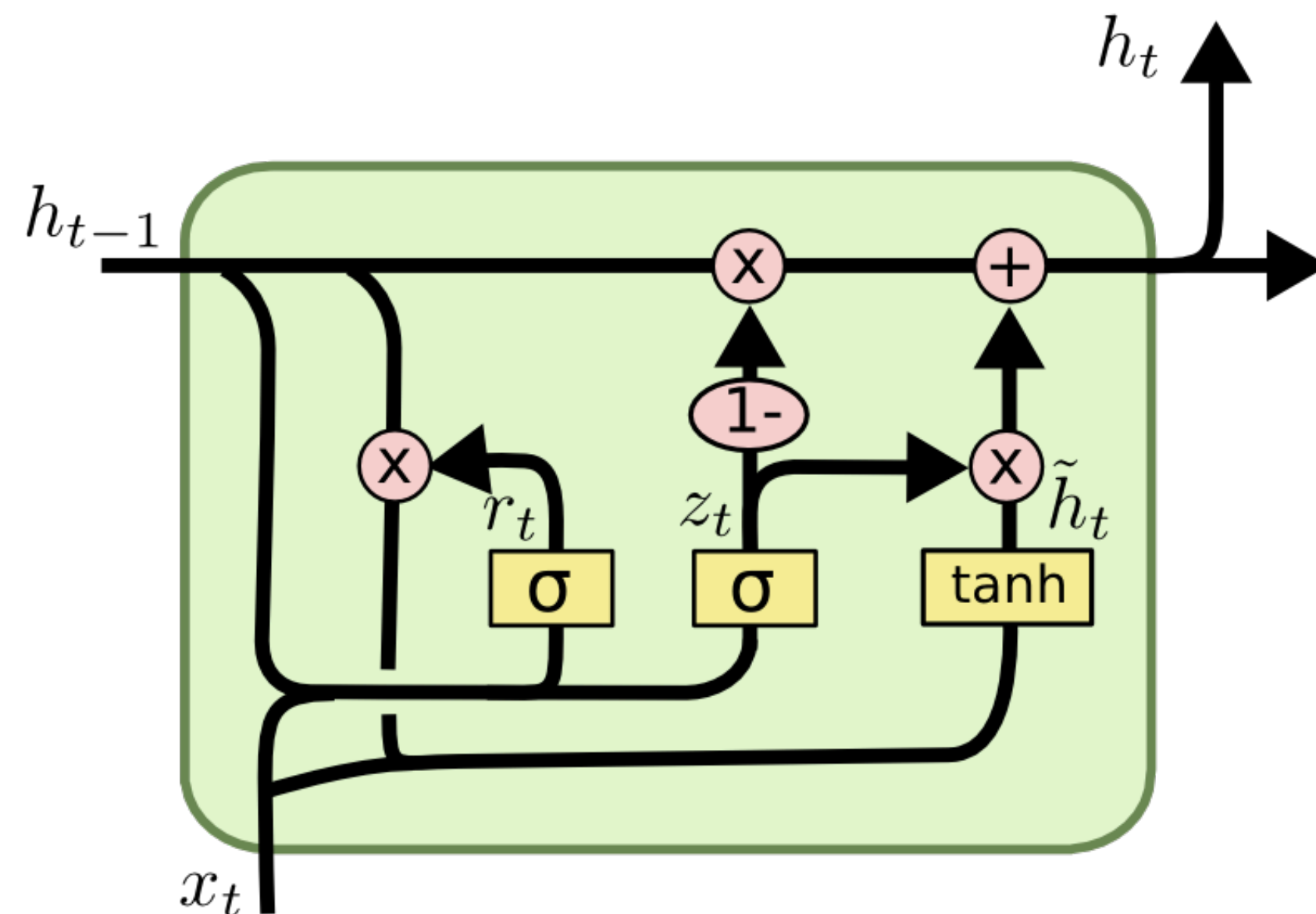

$$o_t = \sigma \left(W_o \left[h_{t-1}, x_t \right] + b_o \right)$$

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

GRU — A simpler variant of the LSTM

There have been new methods building on top of the LSTM.

One of them is the **Gated Recurrent Unit** (GRU), where the **Forget and Input gates are merged into a single Update Gate**.

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

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

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

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

Recurrent Neural Networks — Practical

Goals:

1. Implementing a Recurrent Neural Network from scratch
2. Understanding the involved computations
3. Building a full Deep Learning pipeline in PyTorch to train a model on a given dataset