Previous lecture/practical

Any question?

Parallel training in Deep Learning

Training batches

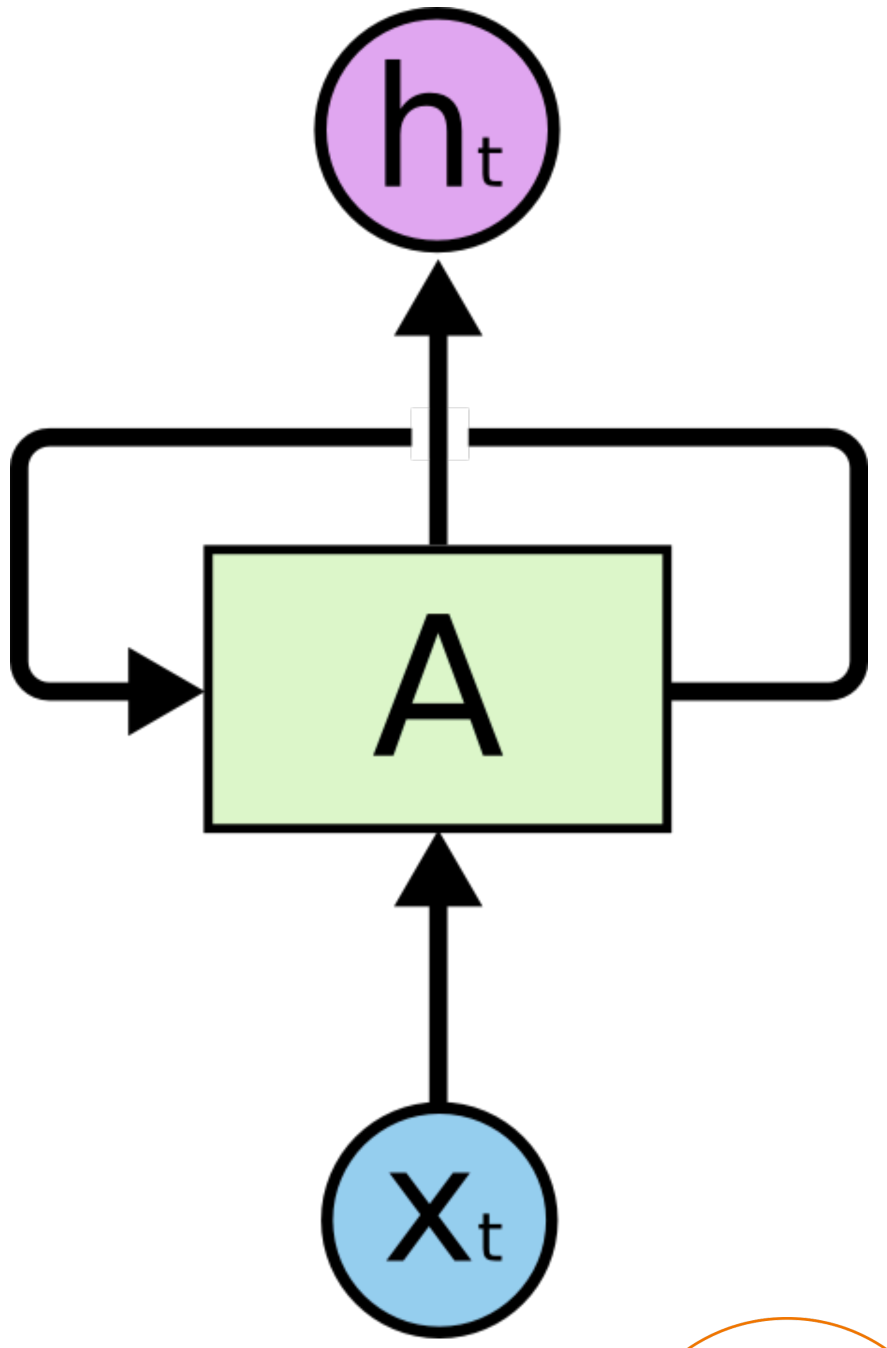
In the last practical you implemented train, val, and test **dataloaders**. An hyperparameter you had to pick was the **batch size,** i.e. the **number of samples to train on at each iteration**.

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.

Artificial Intelligence & Data Analysis

Lecture 2: Recurrent Neural Networks



Lyon 1
Pierre Marza

Course Overview

1. Convolutional Neural Networks (Lecture + practical)
2. Recurrent Neural Networks (Lecture + *optional* practical)
3. Reinforcement Learning 1 (Johan Peralez)
4. Reinforcement Learning 2 (Johan Peralez)
5. Project (15h)

Course Overview

1. Convolutional Neural Networks (Lecture + practical)
2. Recurrent Neural Networks (Lecture + *optional* practical)

3. Reinforcement Learning 1 (Johan Peralez)
4. Reinforcement Learning 2 (Johan Peralez)

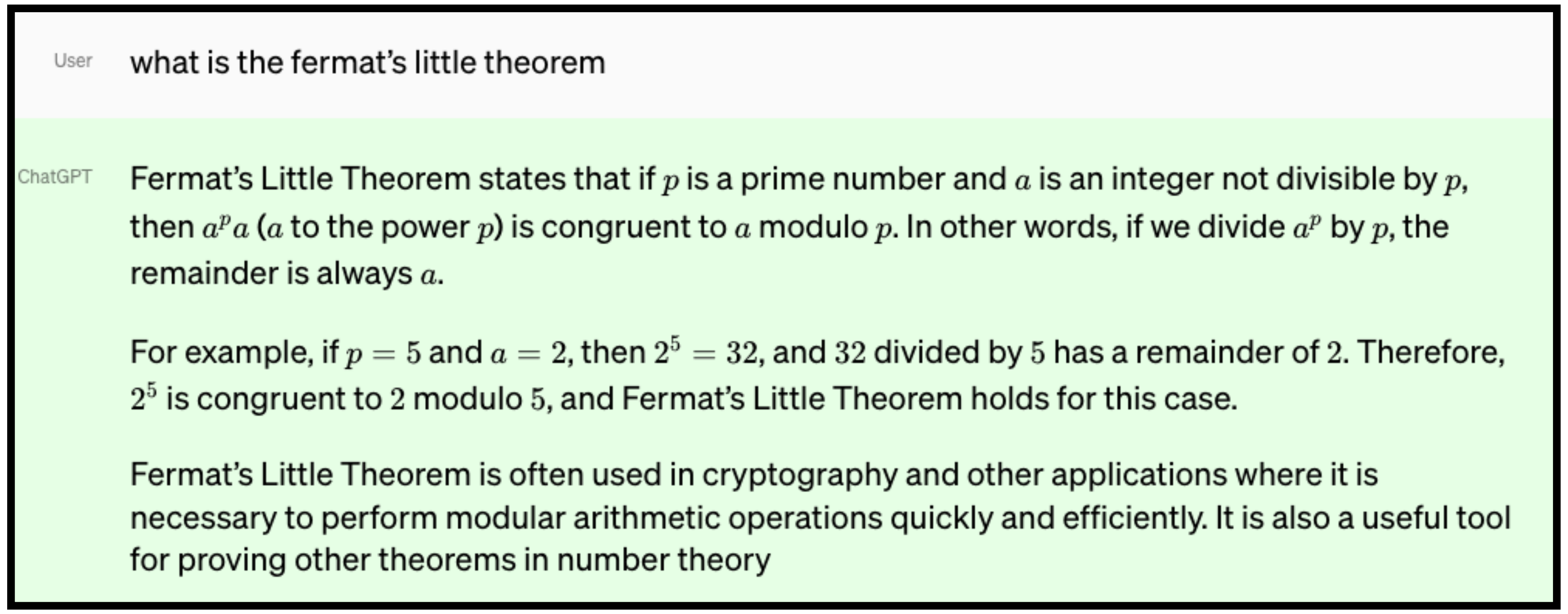
5. Project (15h)

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…


User what is the fermat's little theorem

ChatGPT Fermat's Little Theorem states that if p is a prime number and a is an integer not divisible by p, then $a^p a$ (a to the power p) is congruent to a modulo p. In other words, if we divide a^p by p, the remainder is always a.

For example, if $p = 5$ and $a = 2$, then $2^5 = 32$, and 32 divided by 5 has a remainder of 2. Therefore, 2^5 is congruent to 2 modulo 5, and Fermat's Little Theorem holds for this case.

Fermat's Little Theorem is often used in cryptography and other applications where it is necessary to perform modular arithmetic operations quickly and efficiently. It is also a useful tool for proving other theorems in number theory

Inductive bias

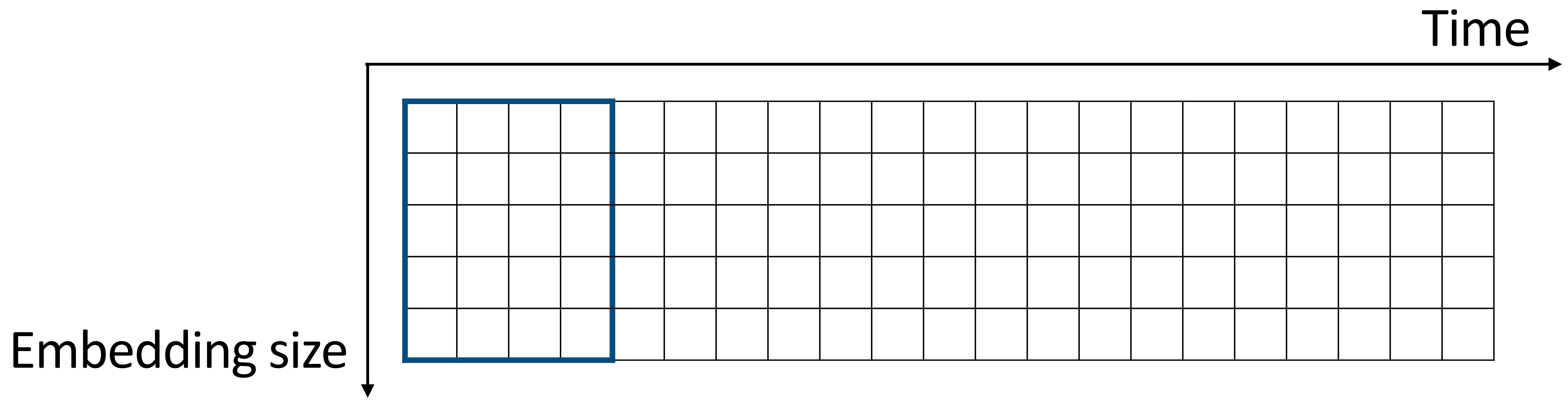
When dealing with **temporal/sequential** data, **keeping track of essential information inside a vectorial memory** seems to be a good idea!

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.

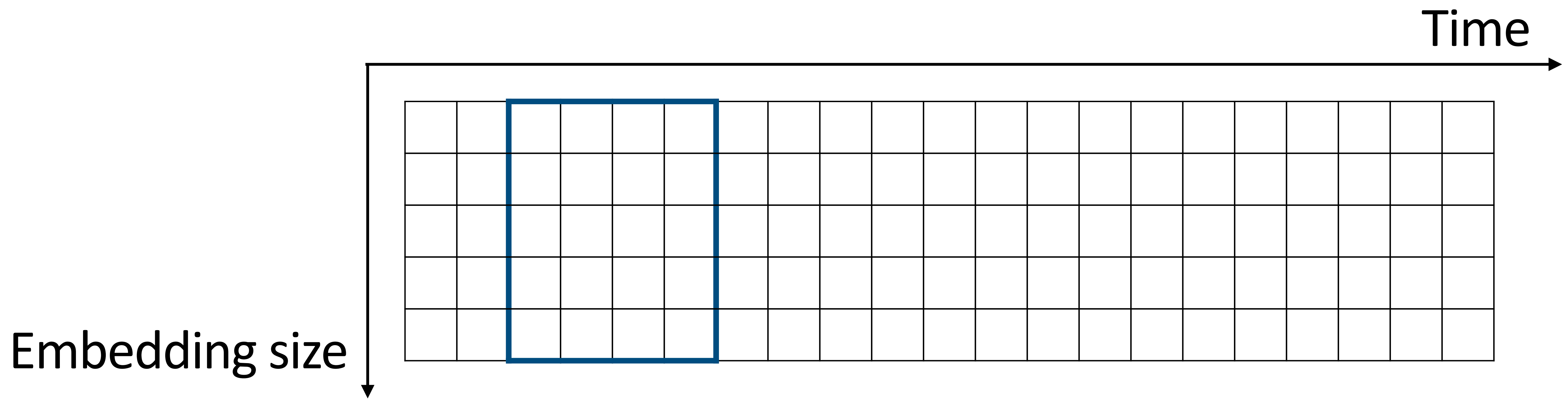
Can we **slide along time dimension**? **Yes**, we can!



Processing temporal information with convolutions

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

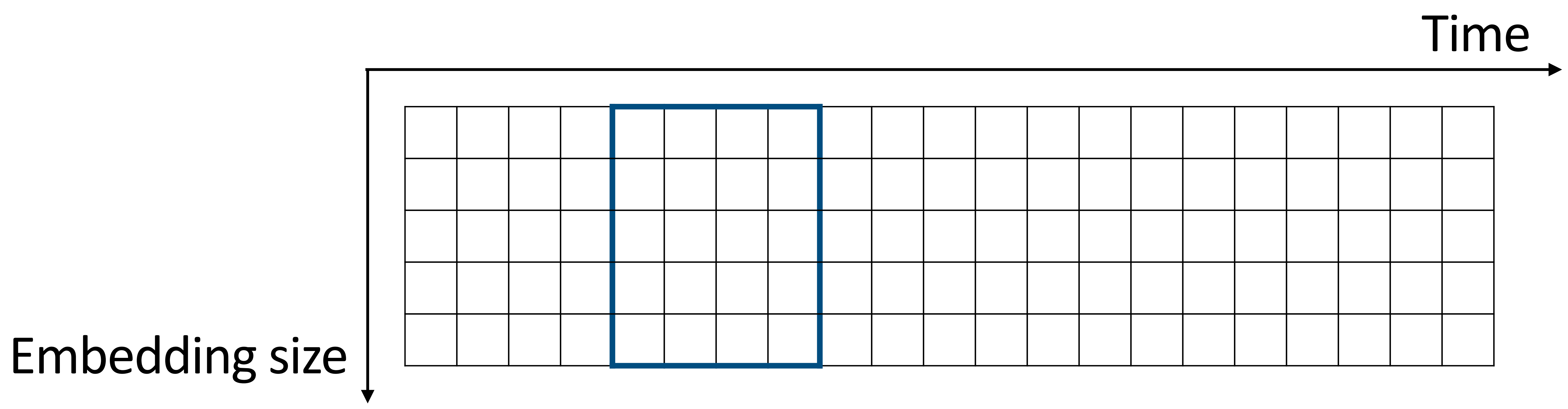
Can we **slide along time dimension**? **Yes**, we can!



Processing temporal information with convolutions

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

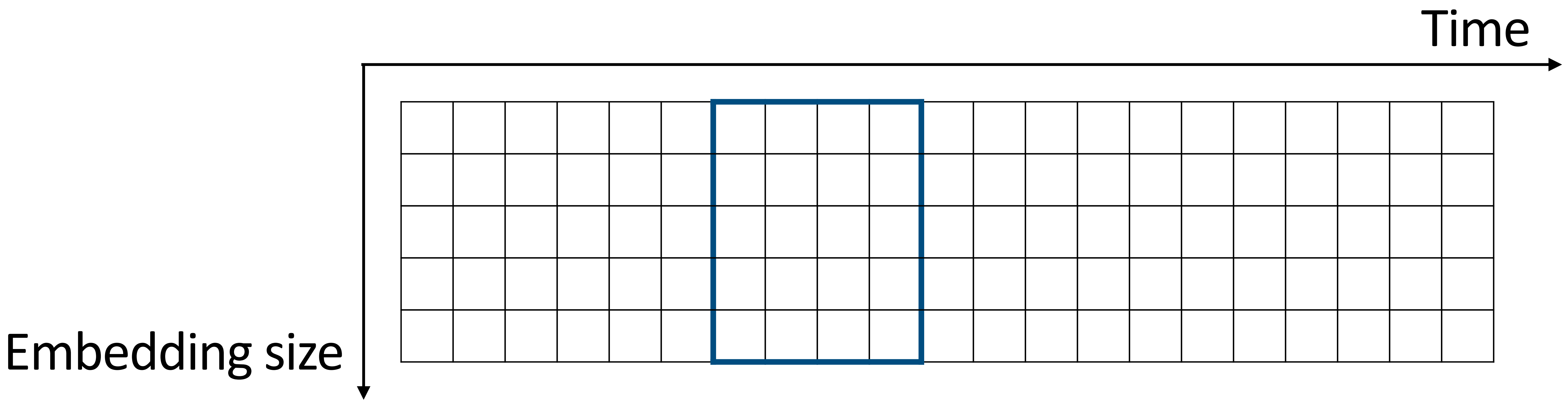
Can we **slide along time dimension**? **Yes**, we can!



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!

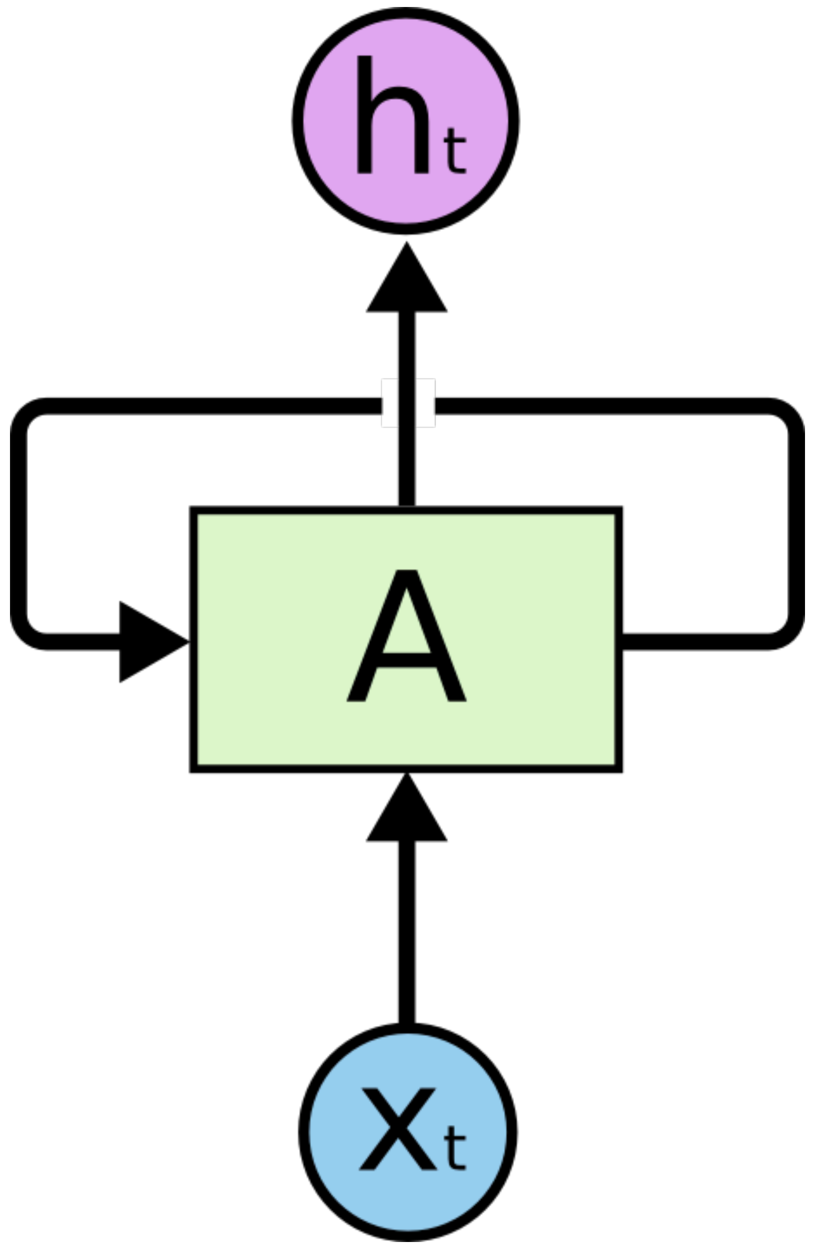


Recurrent Neural Networks

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

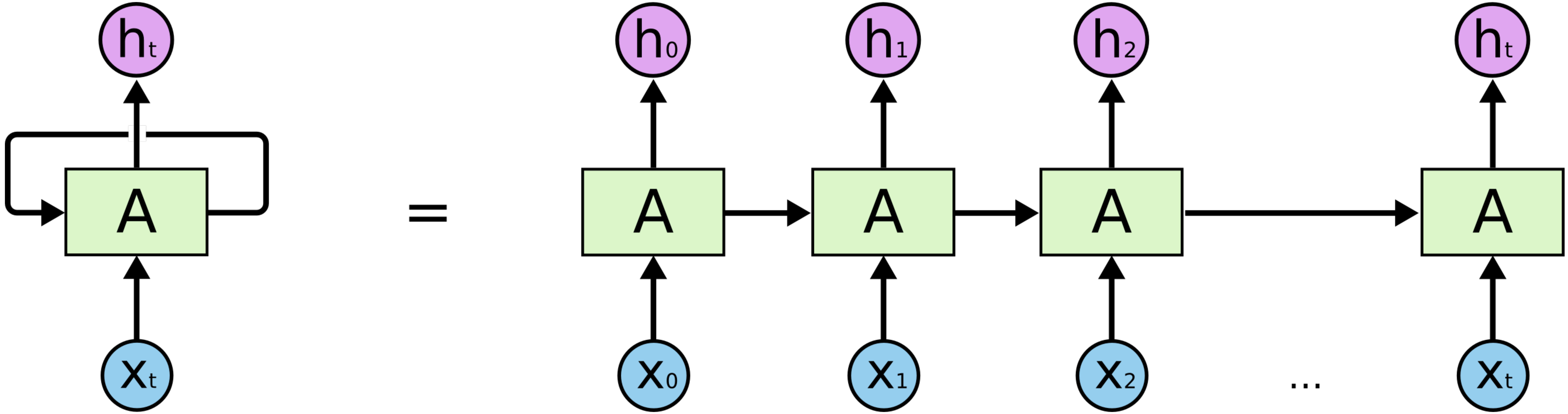
Our unit A deals with **sequential** data: at time t, it is fed with input X_t and outputs a latent representation h_t, but **not only**: **it is also fed with an internal representation from the past step $t-1$.**



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks — Recurrent Unit

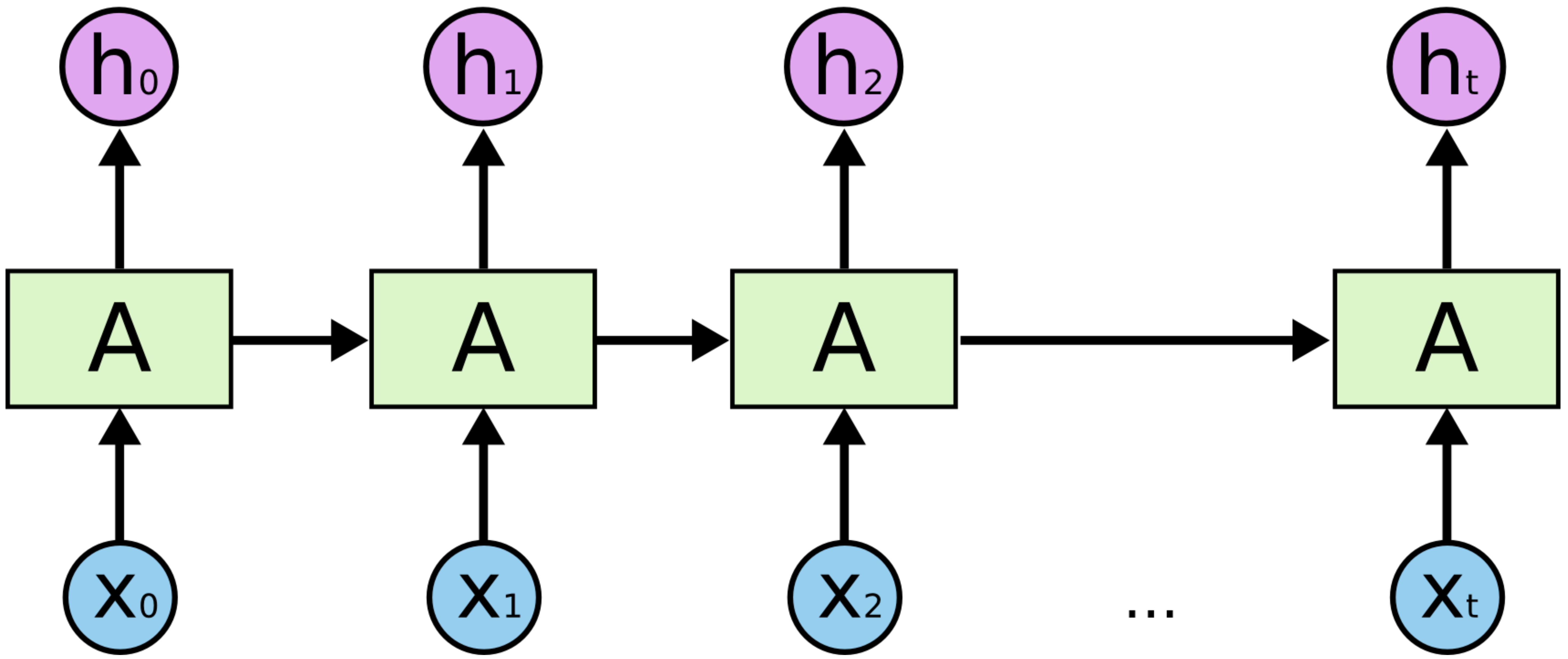
Let’s **unroll** the process **along the time axis**!



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks — Backpropagation Though Time

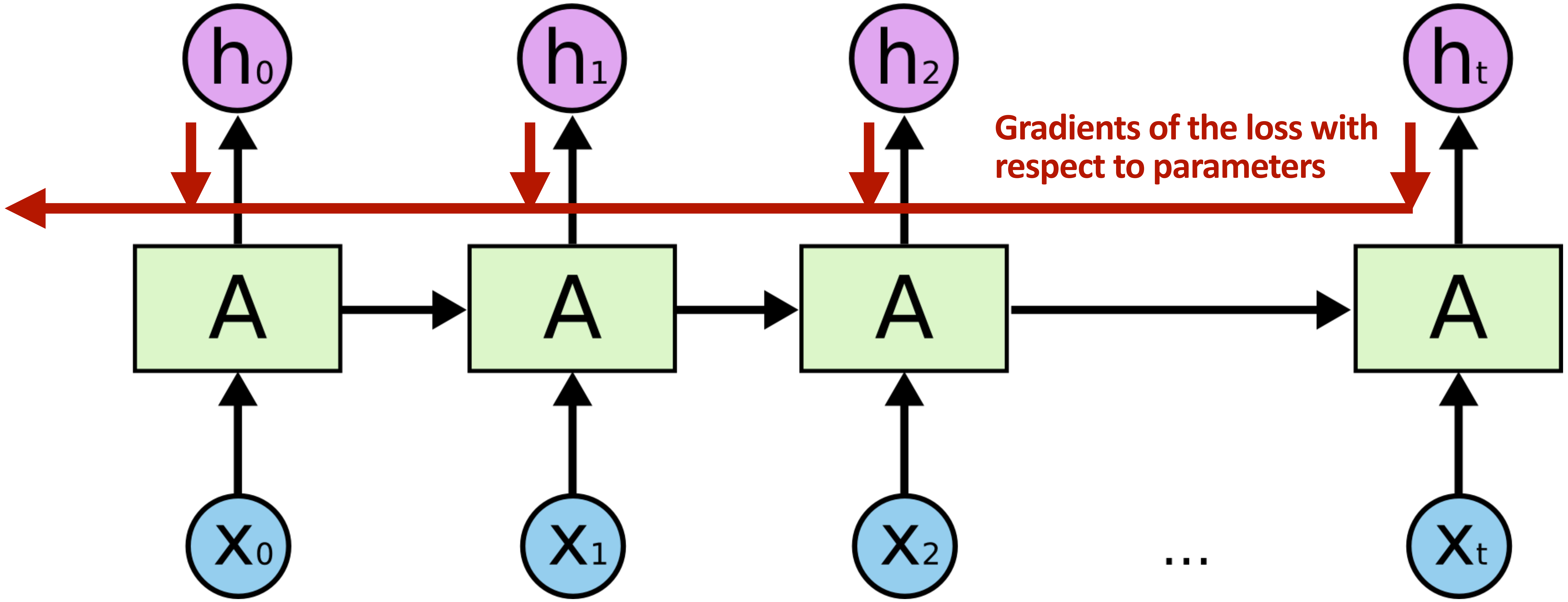
RNNs are trained with Backpropagation Through Time (BPTT):



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Networks — Backpropagation Though Time

RNNs are trained with Backpropagation Through Time (BPTT):



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

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

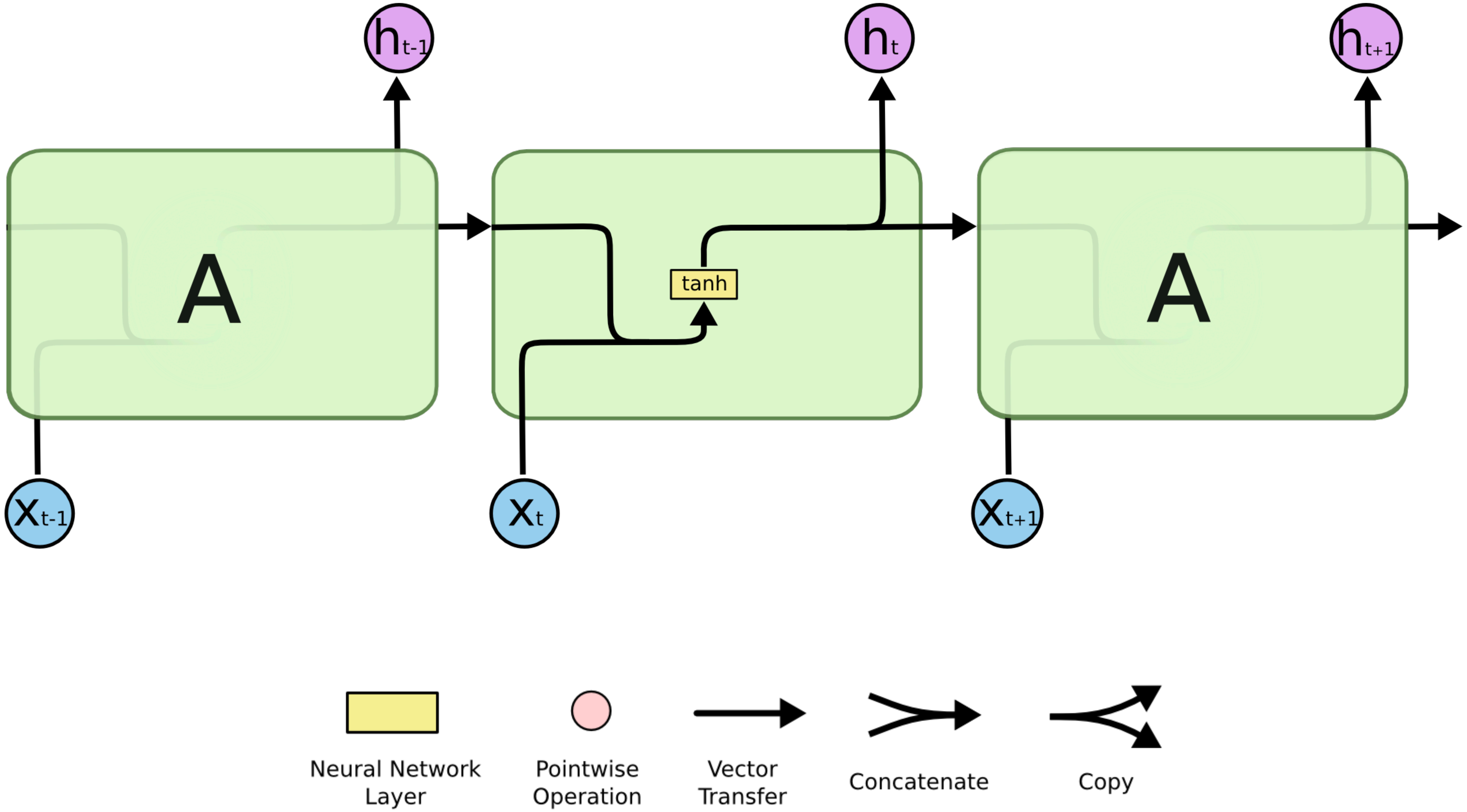
Vanilla RNNs tend to be **hard to train** and suffer from **shortcomings**:

- 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.

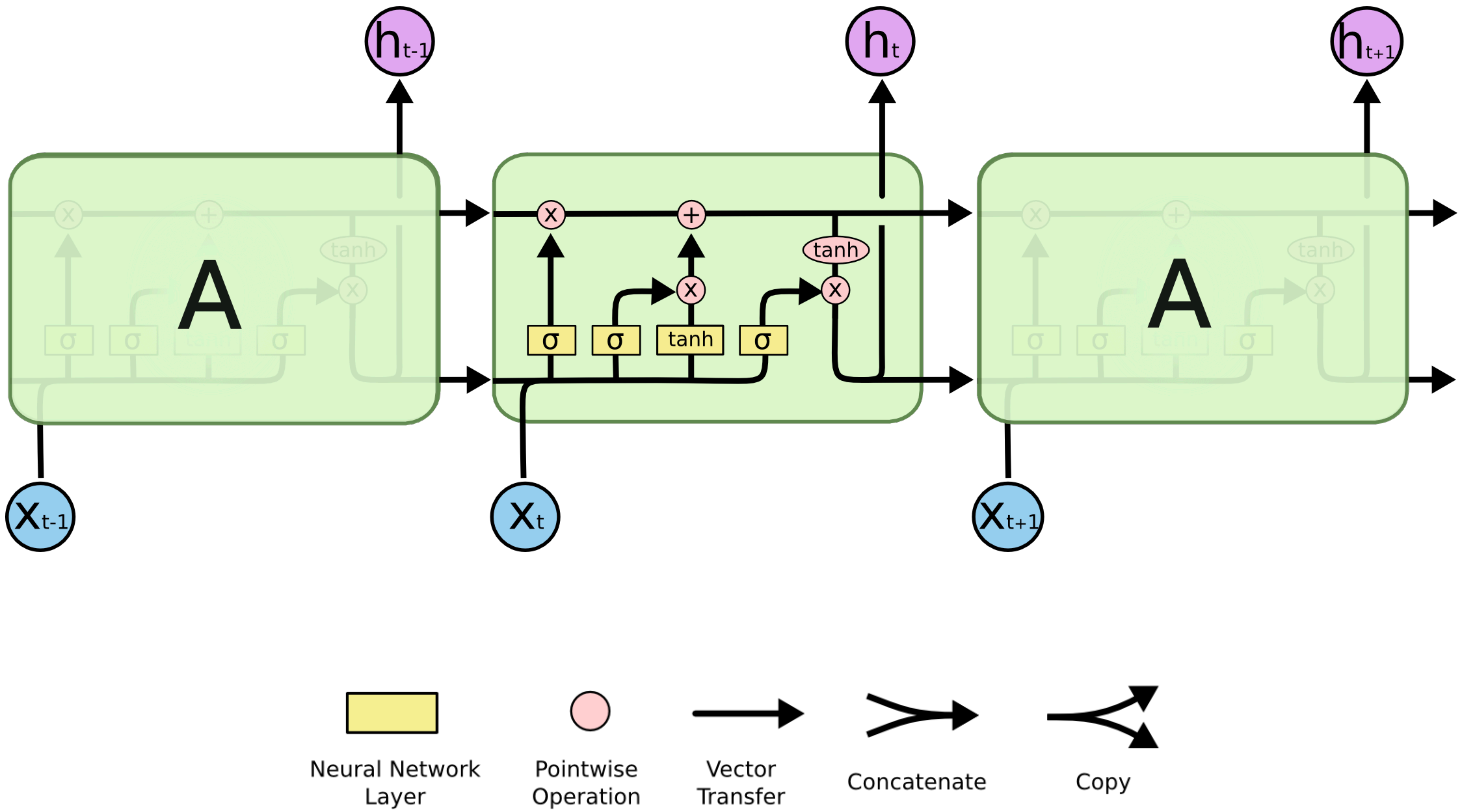
➡ New approaches based on gating mechanisms were introduced.

LSTM — Starting from a standard RNN…



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

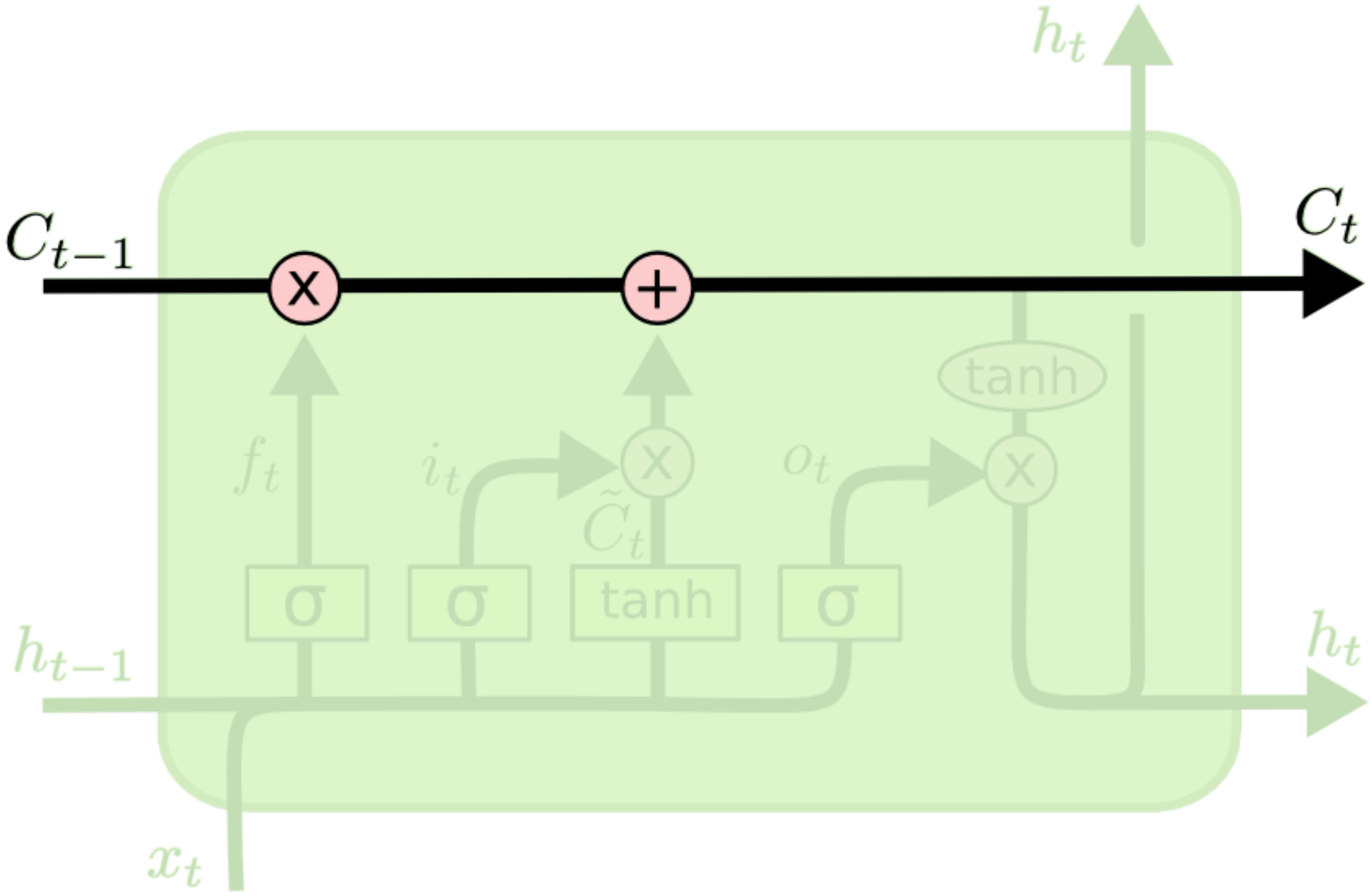
LSTM — … to the LSTM architecture



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM — The Cell State

The Cell State C_t **stores the information we want to remember**.

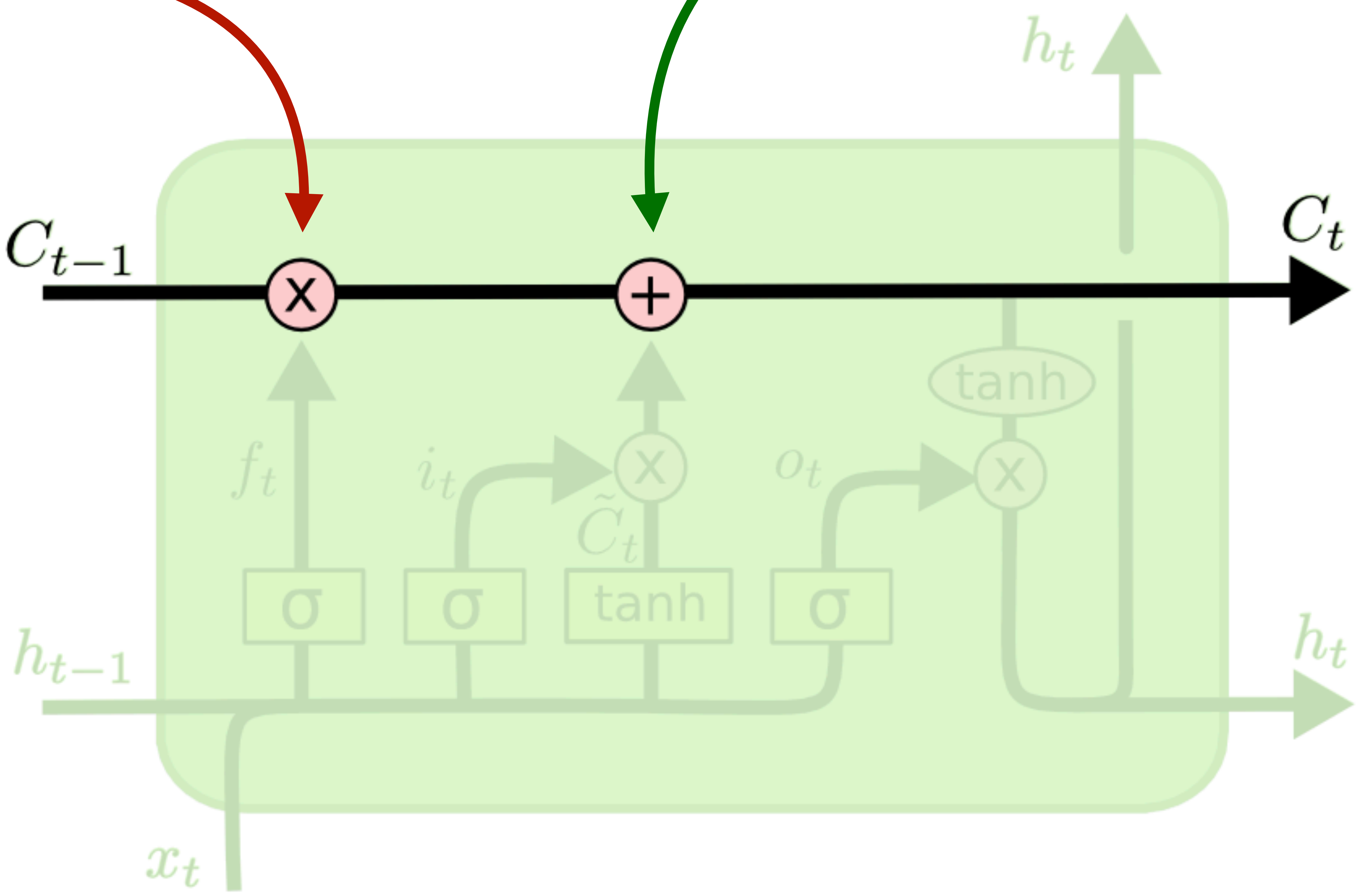


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM — The Cell State

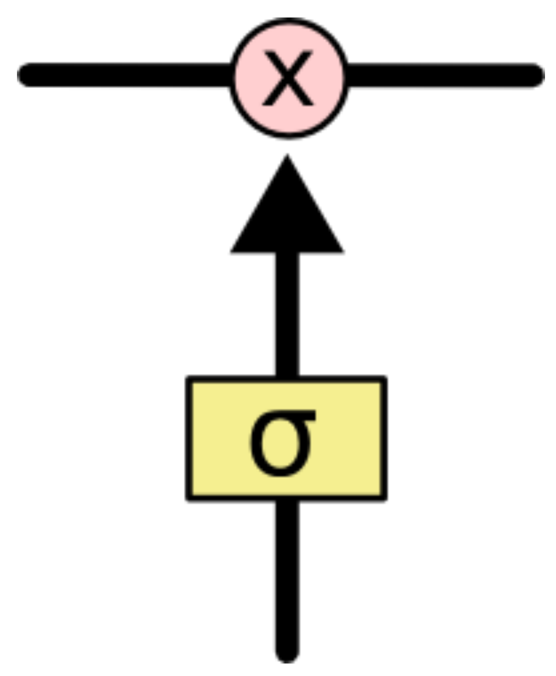
Adding new information to C_t

Removing old information from C_{t-1}

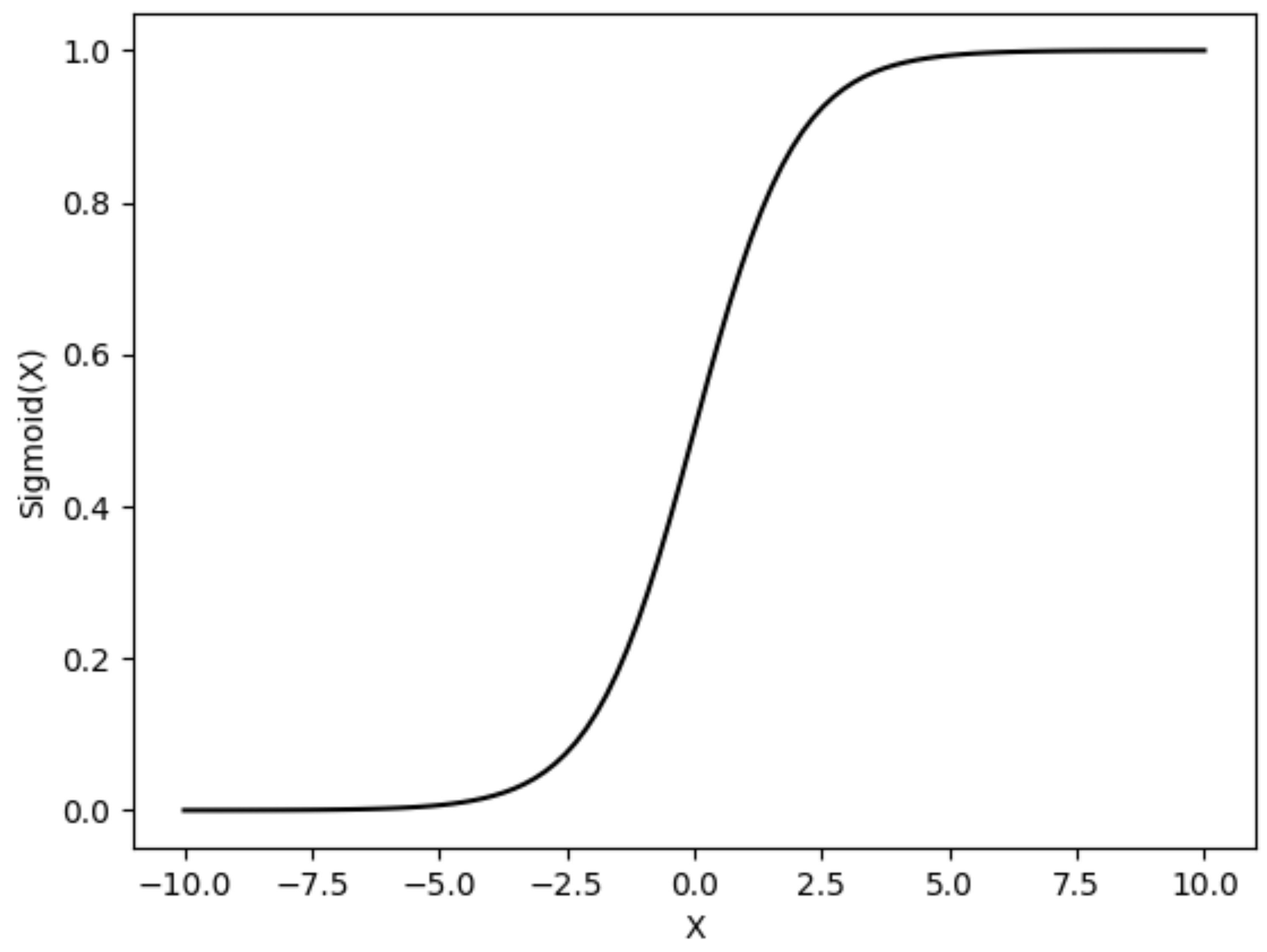


LSTM — Gates

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.



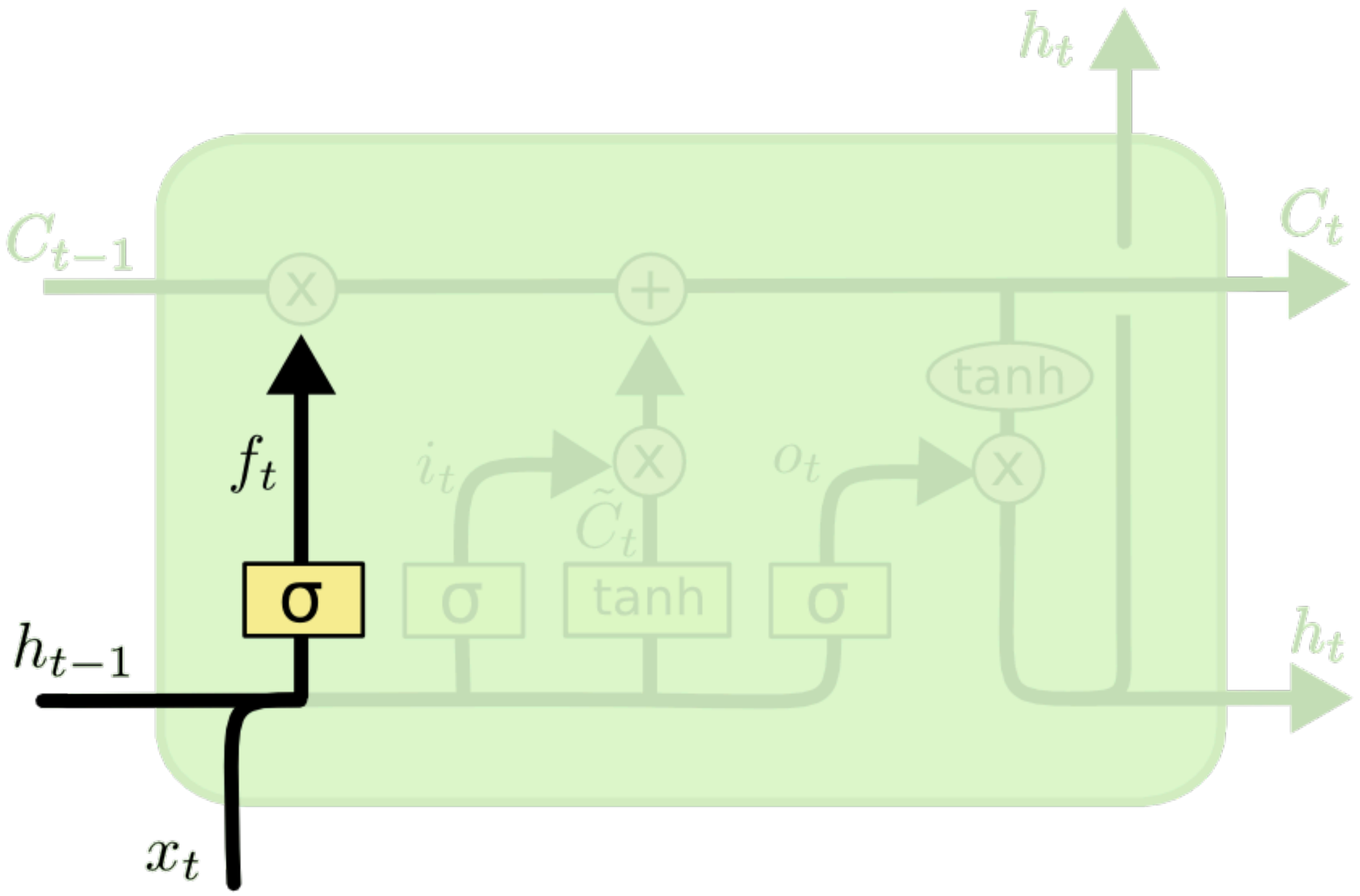
The sigmoid outputs a value between 0 and 1:
0 —> “**don’t let any information** go through”
1 —> “**let all information** go through”



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

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

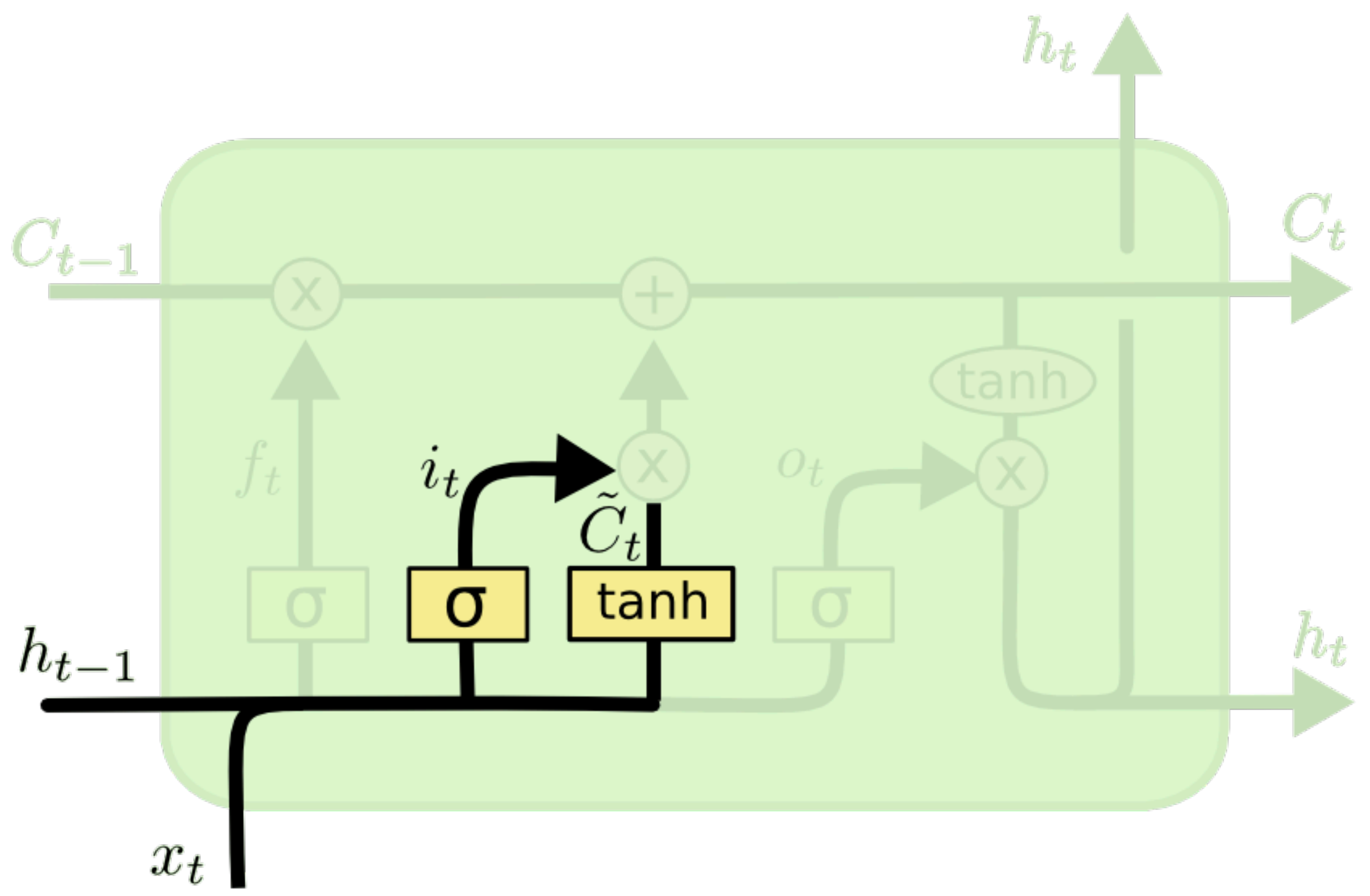
Bias term

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

Sigmoid activation function

LSTM — The Input Gate: What new information to remember?

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** $\tilde{C}_t$.



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

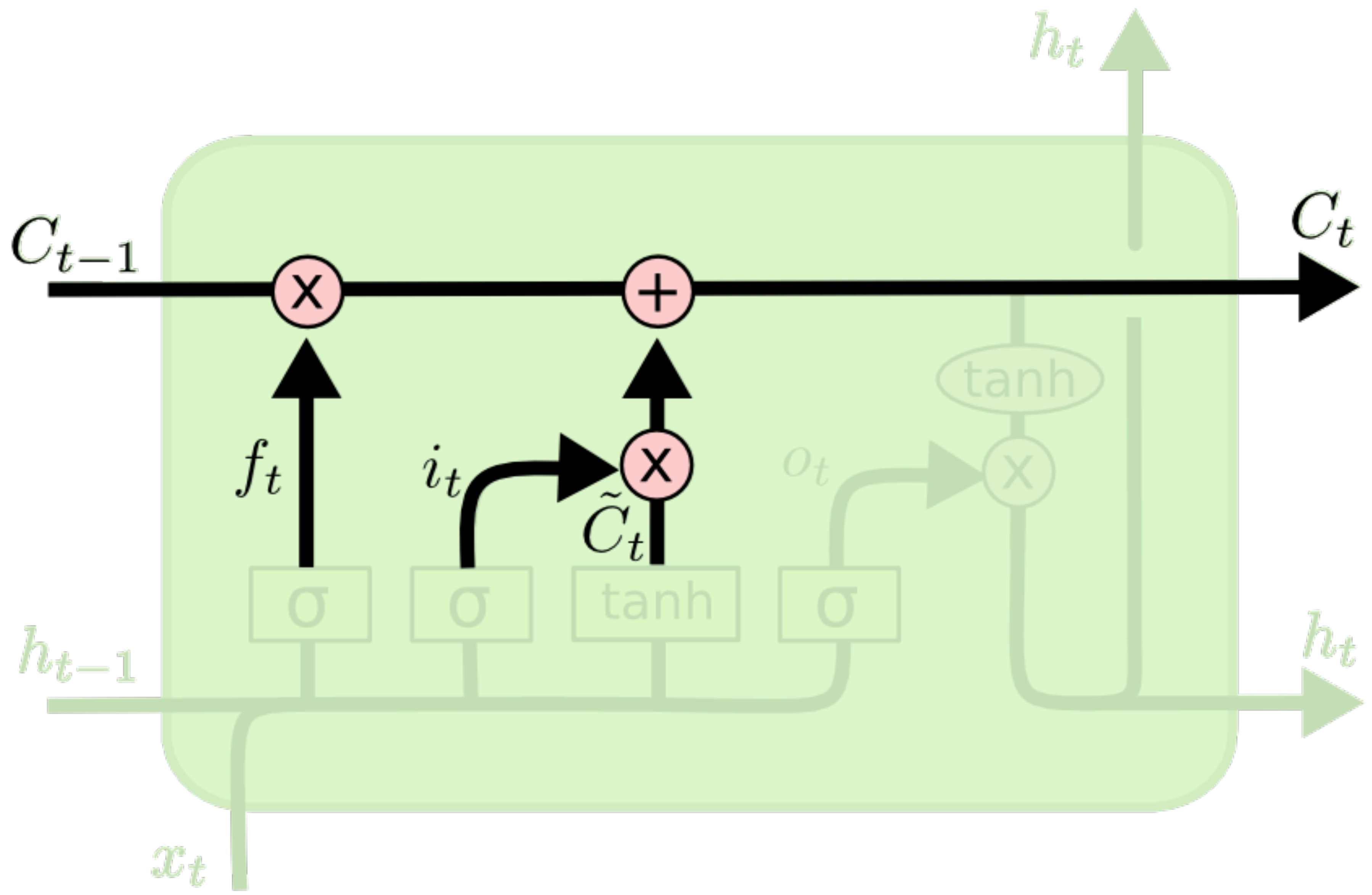
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

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.

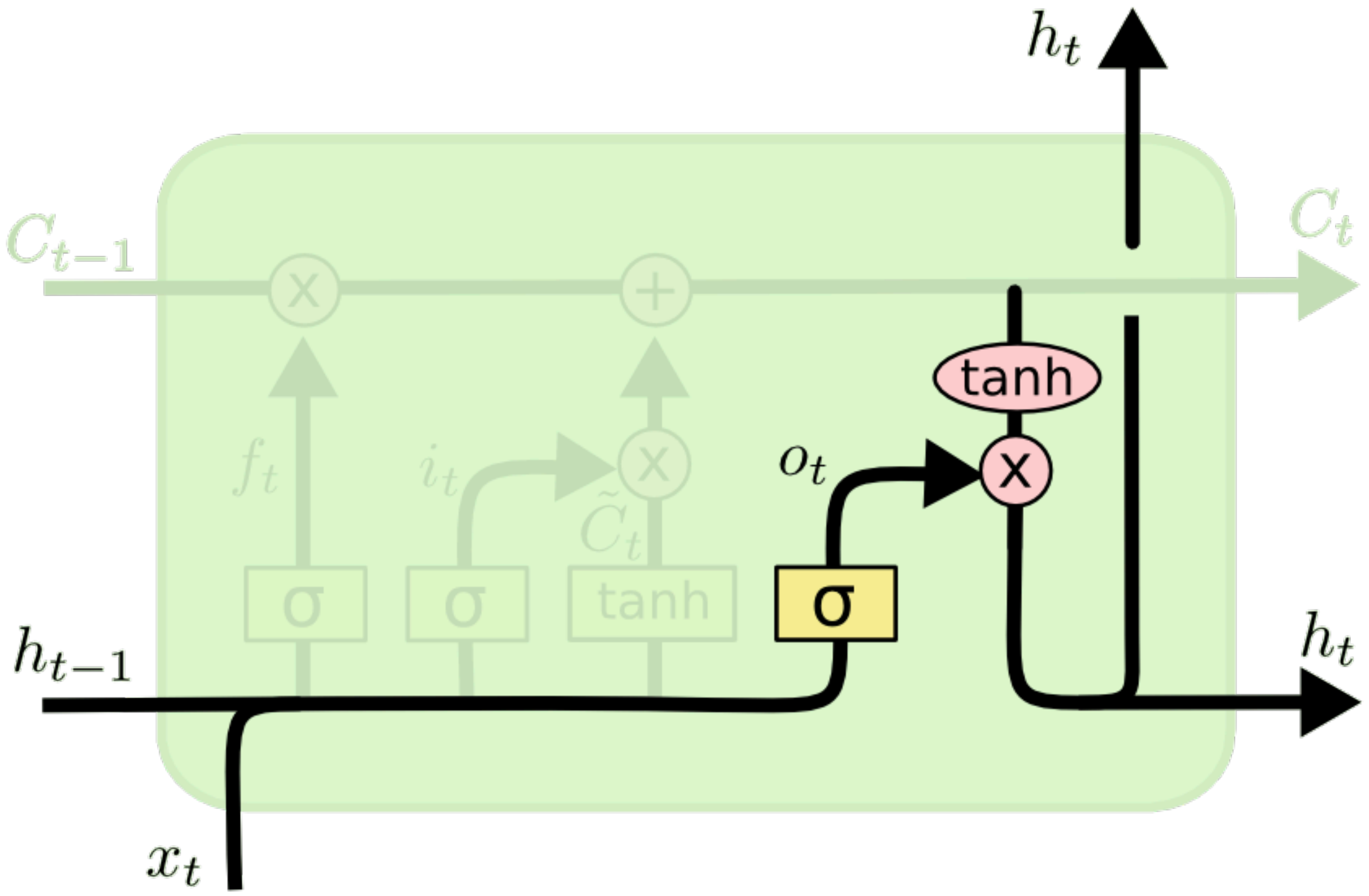


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM — What to output?

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.



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

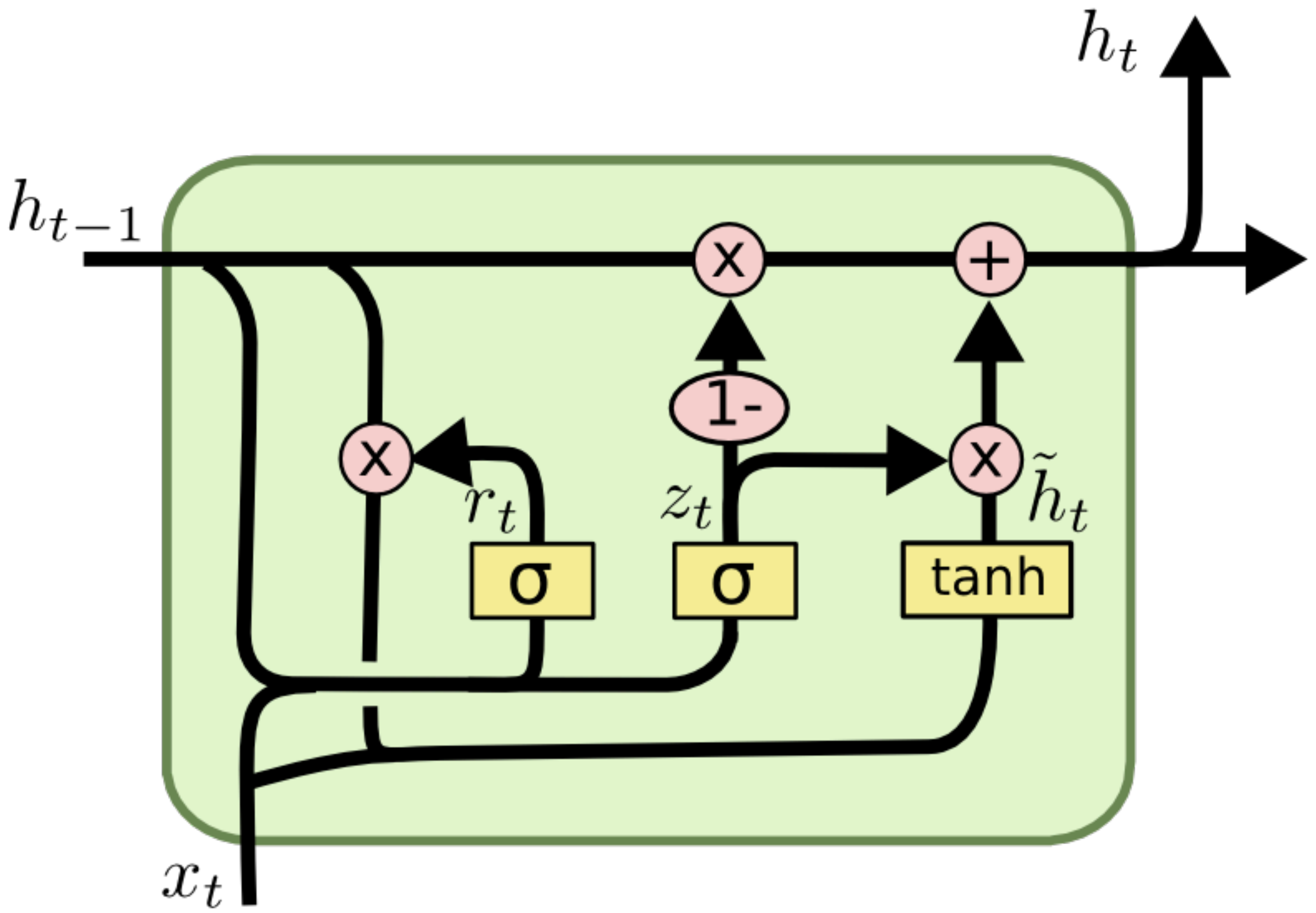
$$h_t = o_t * \tanh\left(C_t\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

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**.



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

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

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

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

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

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