Recurrent Neural Networks

Richard Dirauf, M.Sc.
Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
MLTS Exercise, 09.01.2025

MLTS Exercise – Organization

~~Introduction (31.10.2024)~~

~~Bayesian Linear Regression (07.11.2024)~~

~~Bayesian Linear Regression (14.11.2024)~~

~~Kalman Filter (21.11.2024)~~

~~Kalman Filter (28.11.2024)~~

~~Dynamic Time Warping (05.12.2024)~~

~~Dynamic Time Warping (12.12.2024)~~

~~No exercise planned (19.12.2024)~~

Holiday

RNN + LSTM (09.01.2025)

RNN + LSTM (16.01.2025)

Transformers (23.01.2025)

Transformers (30.01.2025)

Motivation

- How can we utilize neural networks to process sequential data?
- Passing complete sequences to a conventional network is computationally very expensive
  - long training
  - gradients converge very slowly
  - Temporal context gets lost
- **Solution**: Adapt the time related characteristic into the architecture of the net

Name Entity Recognition

Input: Harry Potter invented a new spell.

Dictionary:

$$\begin{pmatrix} \text{a} \\ \text{aaron} \\ \ldots \\ \text{harry} \\ \ldots \\ \text{potter} \\ \ldots \\ \text{spell} \\ \ldots \\ \text{zoo} \end{pmatrix} \quad \begin{array}{l} 1 \\ 2 \\ \ldots \\ 2{,}039 \\ \ldots \\ 6{,}453 \\ \ldots \\ 8{,}940 \\ \ldots \\ 10{,}000 \end{array}$$

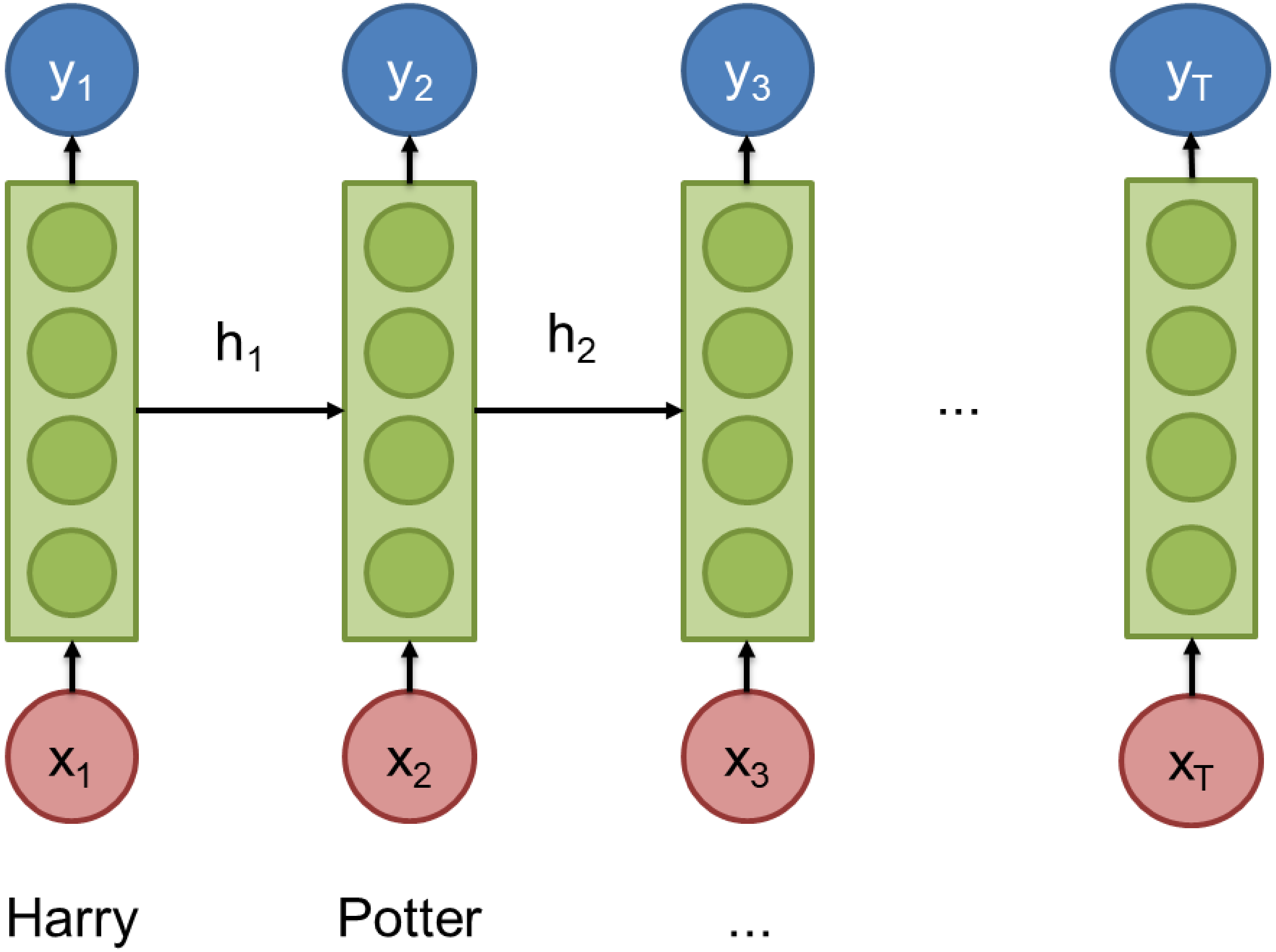
Name Entity Recognition

Input: Harry Potter invented a new spell.

Dictionary:

$$\begin{pmatrix} \text{a} \\ \text{aaron} \\ \ldots \\ \text{harry} \\ \ldots \\ \text{potter} \\ \ldots \\ \text{spell} \\ \ldots \\ \text{zoo} \end{pmatrix} \begin{array}{l} 1 \\ 2 \\ \ldots \\ 2{,}039 \\ \ldots \\ 6{,}453 \\ \ldots \\ 8{,}940 \\ \ldots \\ 10{,}000 \end{array} \qquad x^1: \begin{pmatrix} 0 \\ \ldots \\ 0 \\ 1 \\ 0 \\ \ldots \\ \ldots \\ \ldots \\ \ldots \\ 0 \end{pmatrix} \qquad x^6: \begin{pmatrix} 0 \\ \ldots \\ \ldots \\ \ldots \\ \ldots \\ \ldots \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

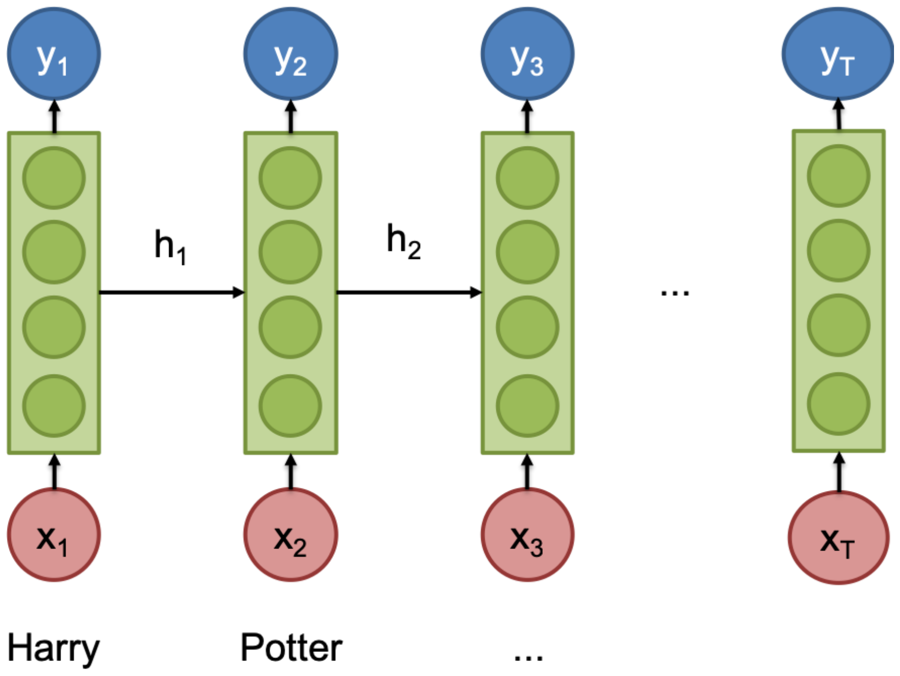
Recurrent Neural Networks (RNN)



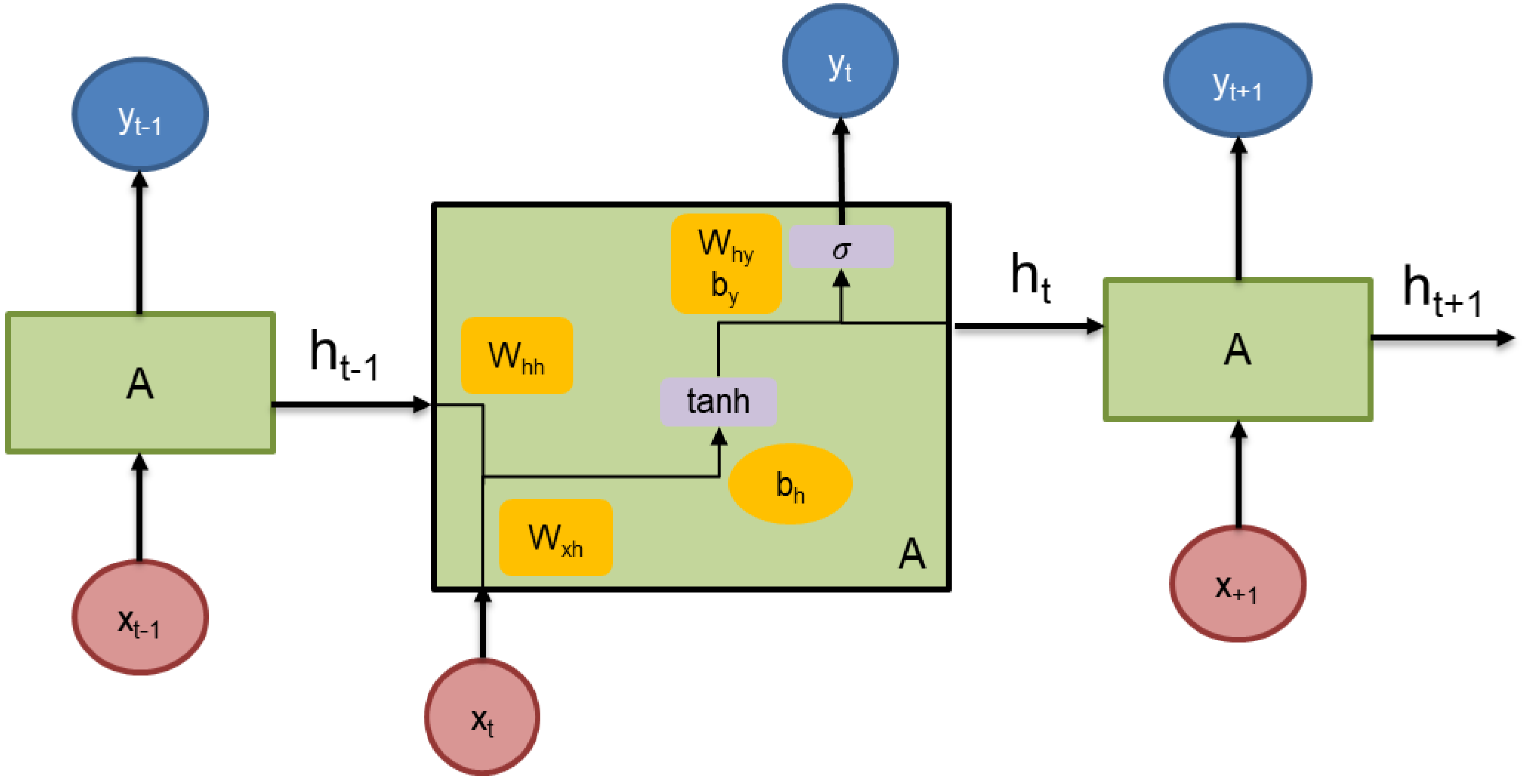
RNN Forward Propagation

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$
$$y_t = \sigma(W_{hy}h_t + b_y)$$

σ: sigmoid / softmax



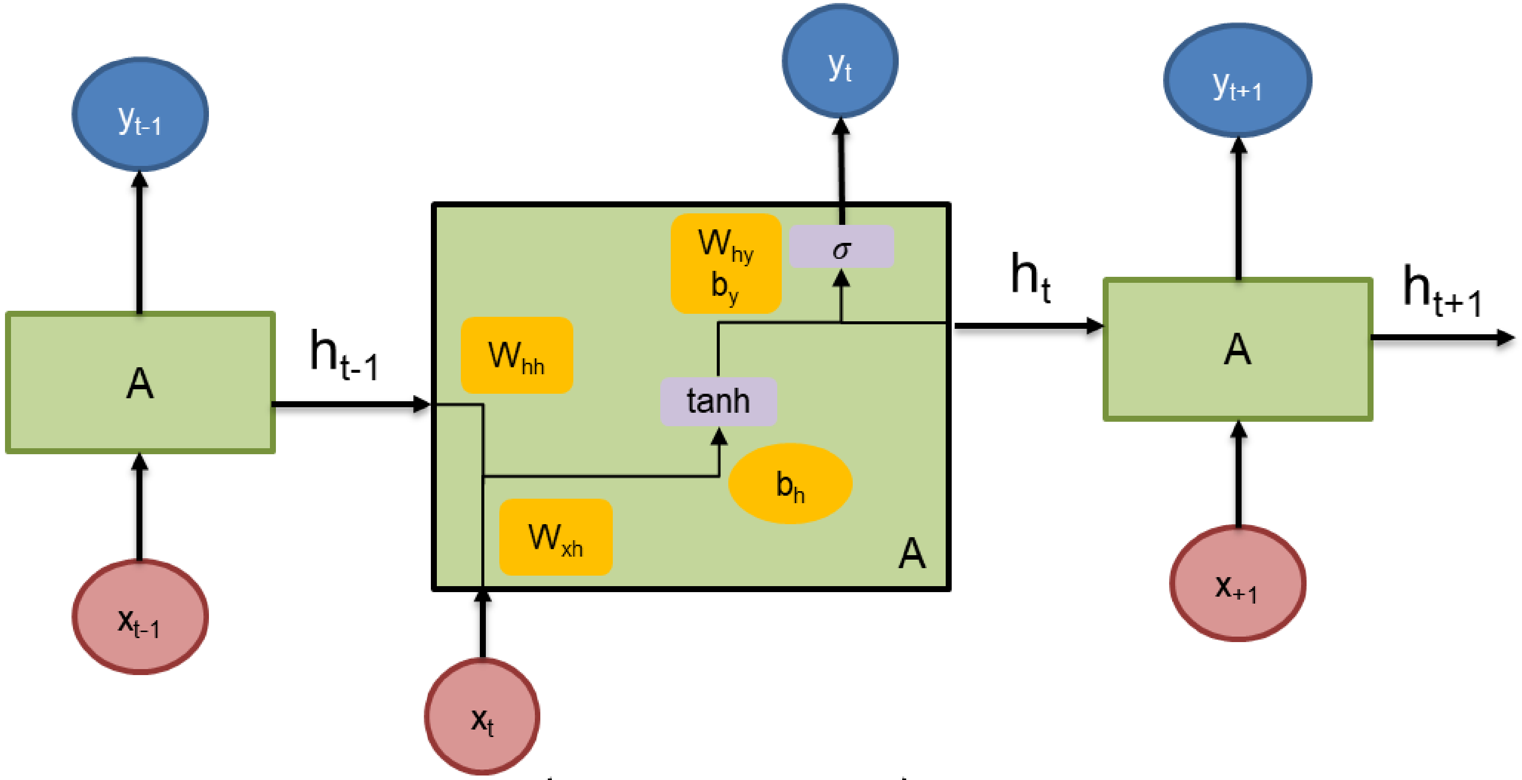
The RNN Cell Up Close



Hidden state: $h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$

W_{hh}: Weights of previous hidden state
W_{xh}: Weights of the input
b_h: Bias for state update

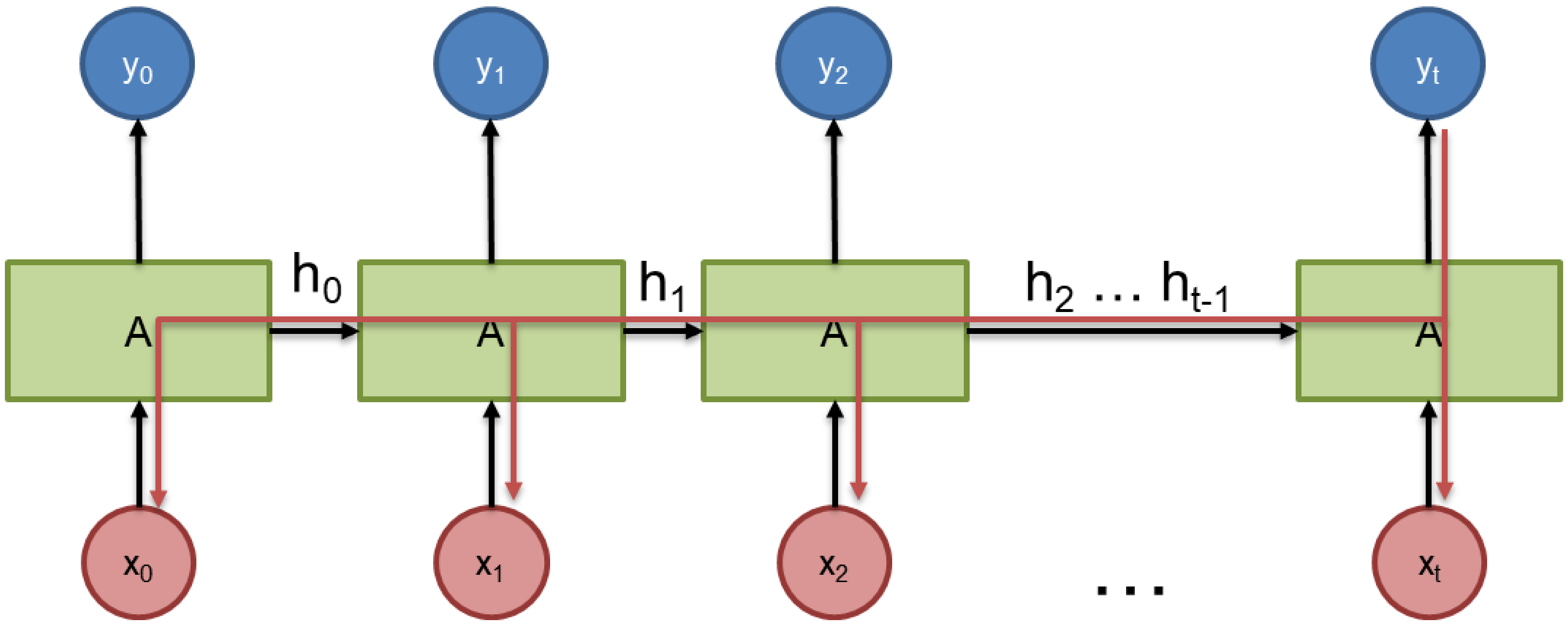
The RNN Cell Up Close



Output: $$y_t = \sigma(W_{hy}h_t + b_y)$$

W_{hy}: Weights of current hidden state
b_y: Bias for output update

Backpropagation Through Time (BPTT)



- The backward pass takes activations off the stack to compute the error derivatives at each time step
- Backpropagate through time all the way to the initial states to get the gradient of the cost function with respect to each initial state

(BPTT) Calculate The Cost

Total cost is the sum of losses at each time step

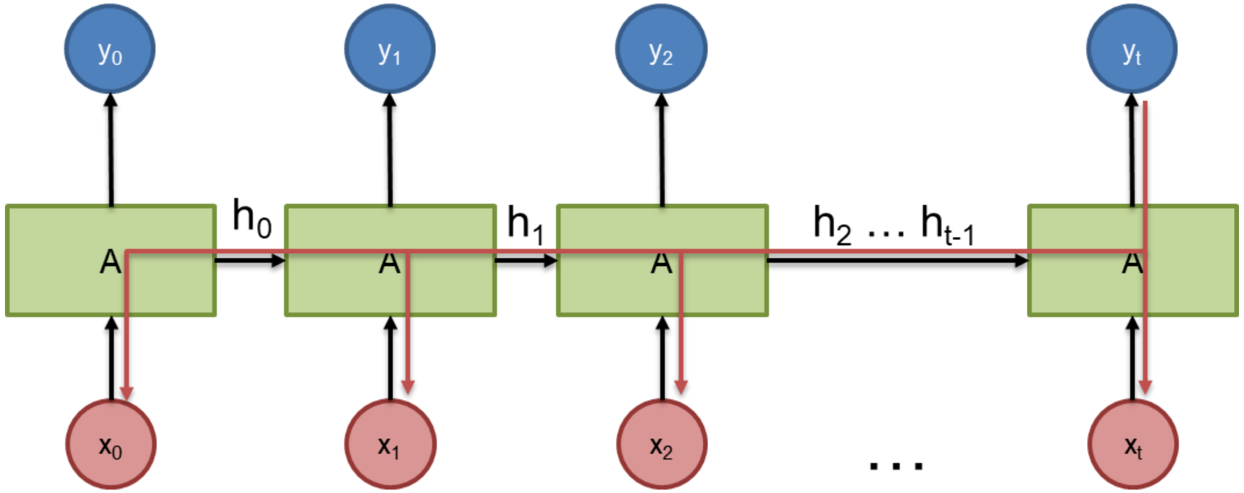
$$C(y, \hat{y}) = \sum_t C_t$$

$$C_t = \frac{1}{2} ||y_t - \hat{y}_t||^2$$

Compute the gradient of the loss

$$\nabla C = [\nabla W_{xh}, \nabla W_{hh}, \nabla W_{hy}, \nabla b_h, \nabla b_y, \nabla h]$$

Backpropagation Through Time (BPTT)



$$h_t = tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

$$tanh'(x) = 1 - tanh(x)^2$$

$$\nabla W_{xh} = tanh'(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \cdot x_t^T$$

$$\nabla W_{hh} = tanh'(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \cdot h_{t-1}^T$$

$$\nabla b_h = \sum_{batch} tanh'(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

$$\nabla x_t = W_{hx}^T \cdot tanh'(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

$$\nabla h_{t-1} = W_{hh}^T \cdot tanh'(W_{hx}x_{t-1} + W_{hh}h_{t-1} + b_h)$$

(BPTT) Update Weights and Biases

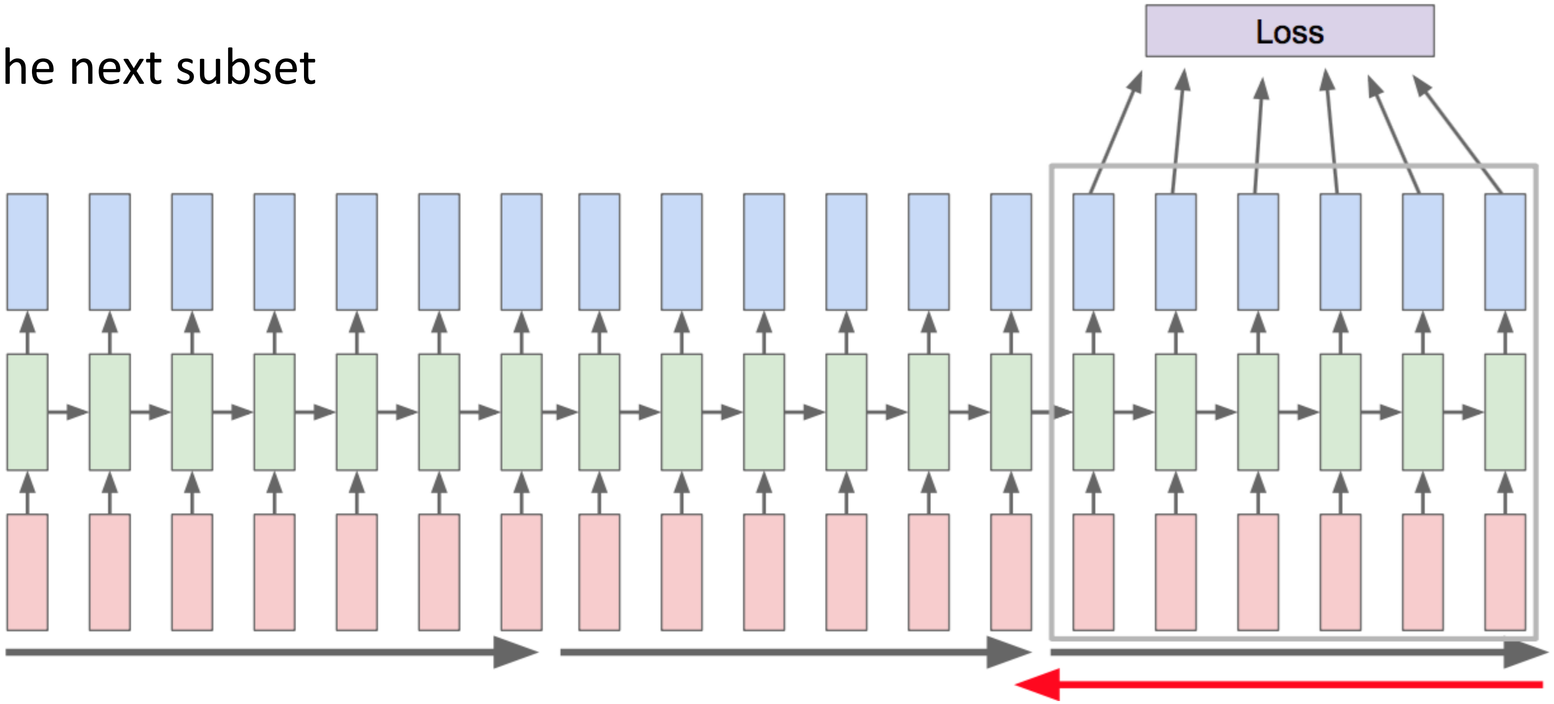
- Given the learning rate η update the weights and biases in the matrix W in the negative gradient direction

$$W = W - \eta \nabla C$$

- **Problem**: backpropagating through the whole sequence the parameter update is computational expensive
- **Solution**: truncated backpropagation through time (TBPTT)

Truncated Backpropagation Through Time

- Instead of passing through the complete sequence, forward pass through a subset
- Backpropagate through the subset
- Continue with the next subset



Adapted from https://tjmachinelearning.com/lectures/guest/rnnadv/rnnadv.html

Friedrich-Alexander-Universität
Technische Fakultät

Machine Learning
Data Analytics

Long Short-Term Memory Network

The Problem With Long-Term Dependency

Backward pass is completely linear:

- If weights are large (> 1), the gradient grows exponentially
- If weights are small (< 1), the gradients shrink exponentially

If we train on long sequences (> 100 time steps) the gradient may easily *explode* or *vanish*

The cat, which already ate ..., was/were full.

The Problem With Long-Term Dependency

Backward pass is completely linear:

- If weights are large (> 1), the gradient grows exponentially
- If weights are small (< 1), the gradients shrink exponentially

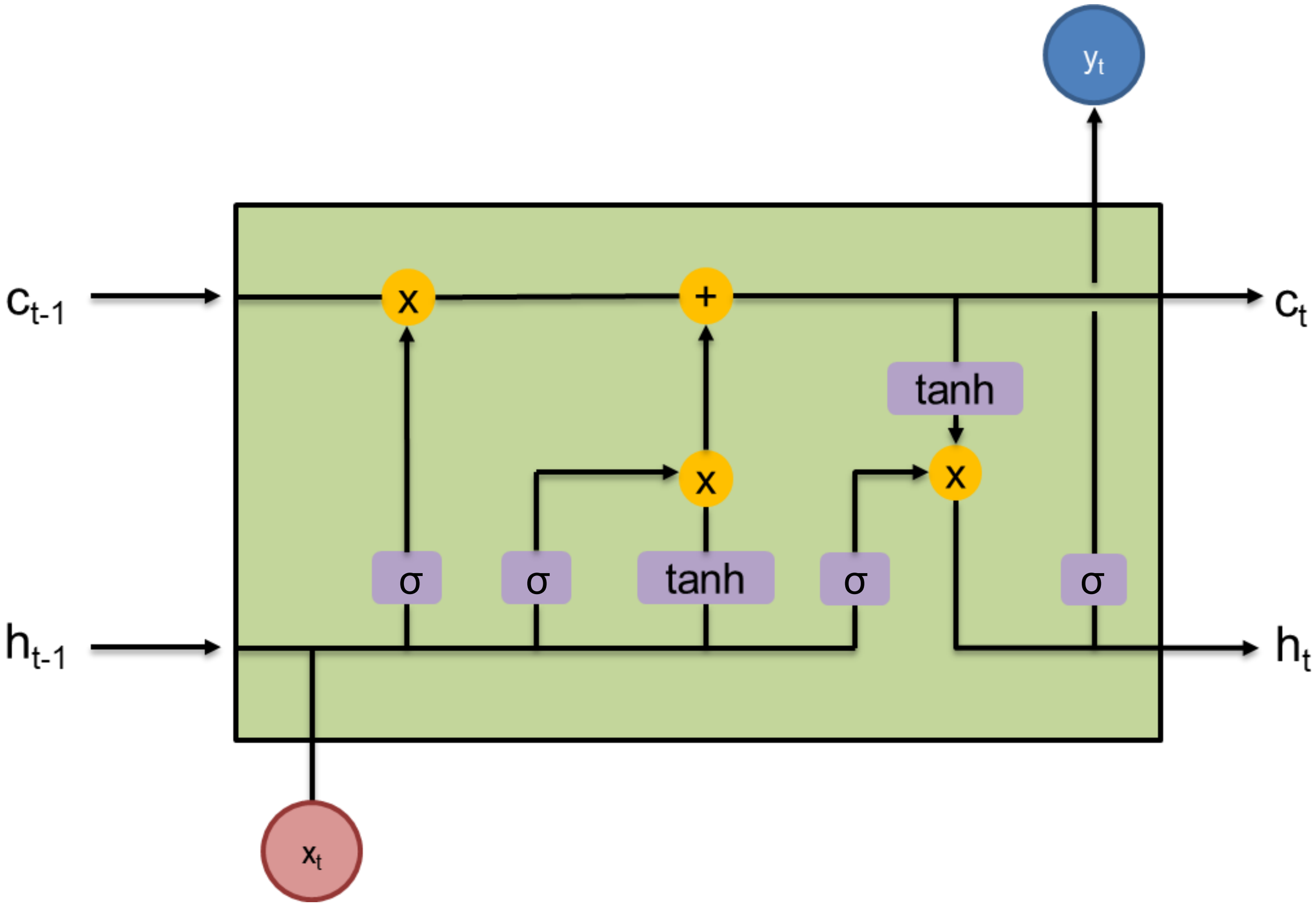
If we train on long sequences (> 100 time steps) the gradient may easily *explode* or *vanish*

Solution: Change RNN architecture

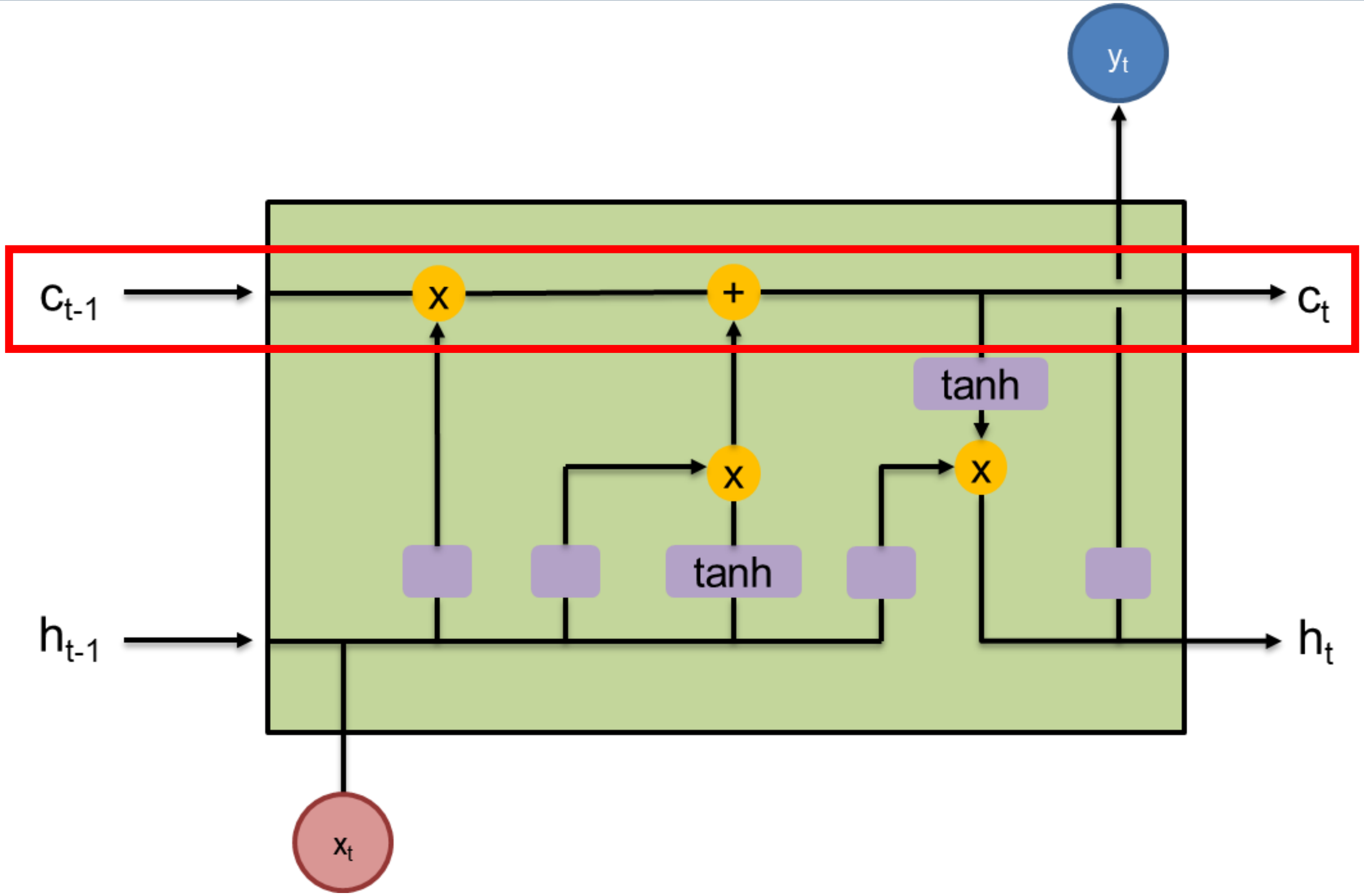
- e.g.: Long Short Term Memory Networks (LSTMs)

Long Short-Term Memory Network

LSTM unit:



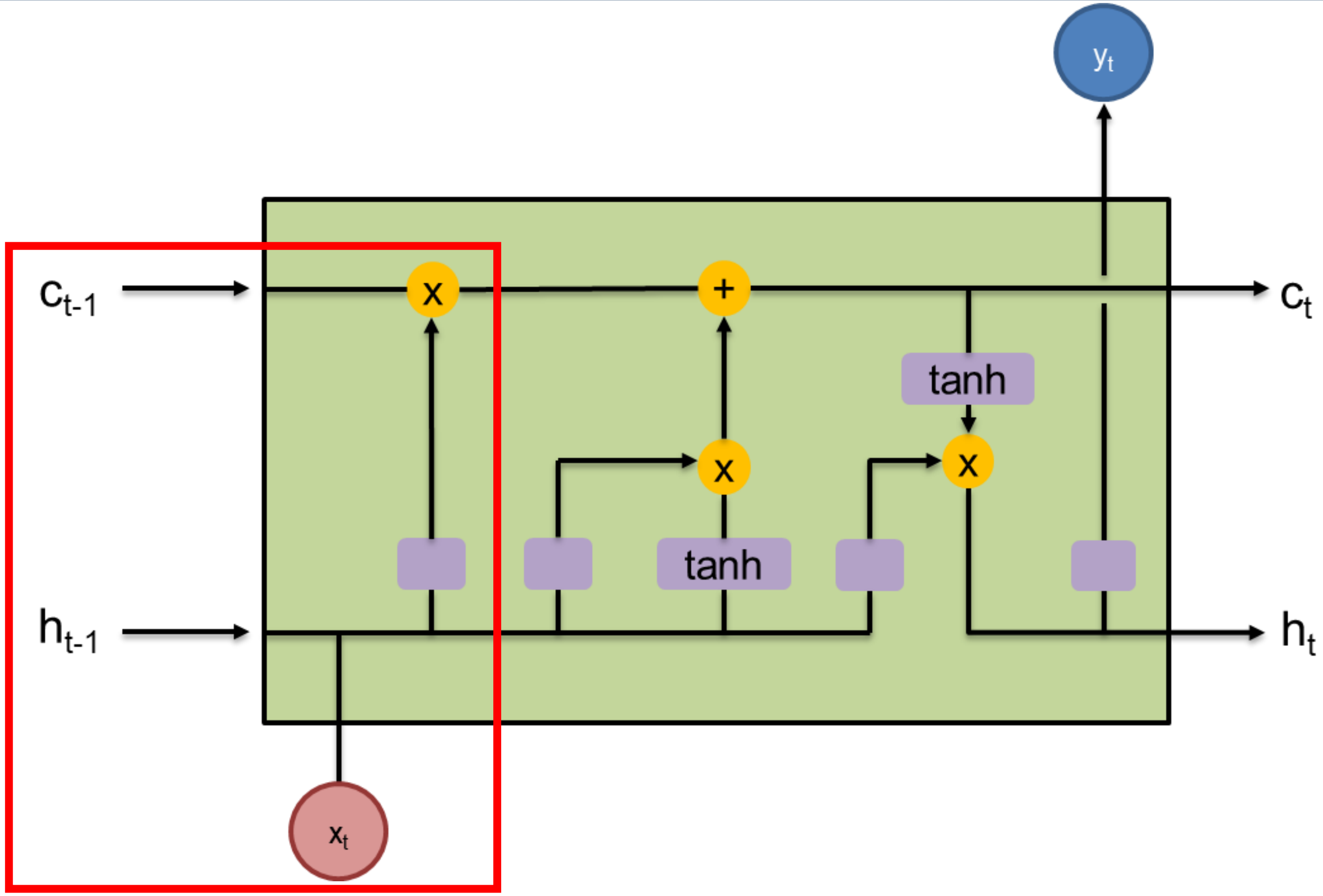
Long Short-Term Memory Network



Cell state:

- Only minor linear interactions
- Easy flow of information, relatively unchanged
- Adding or removing information to the cell state is regulated by gates

Long Short-Term Memory Network



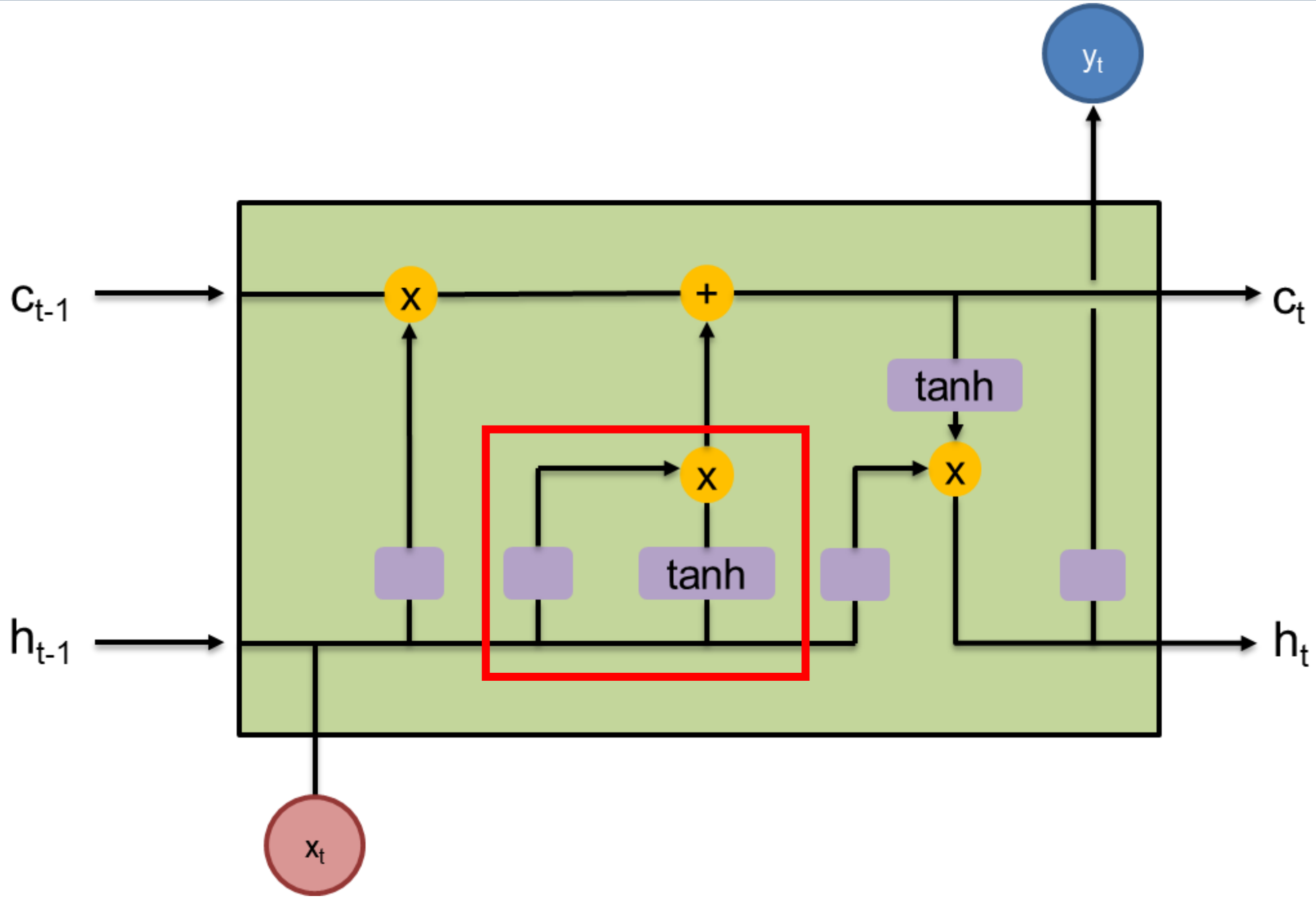
Forget gate:

Decide how much of the previous cell state will be forgotten

Sigmoid layer squashes h_{t-1} and x_t between 0 and 1

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

Long Short-Term Memory Network



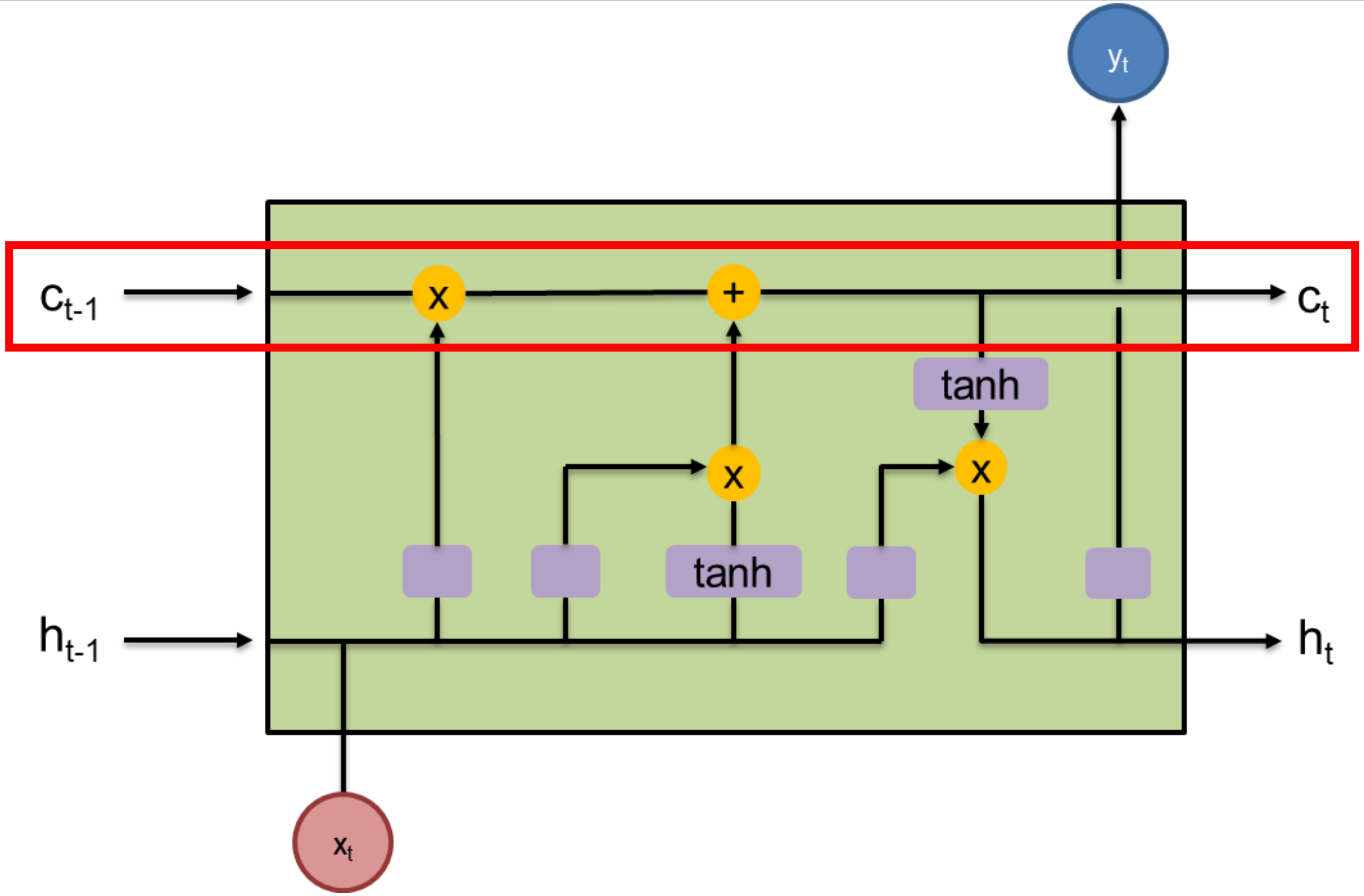
Input gate:

Decide what information we are going to store in the cell state

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

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

Long Short-Term Memory Network

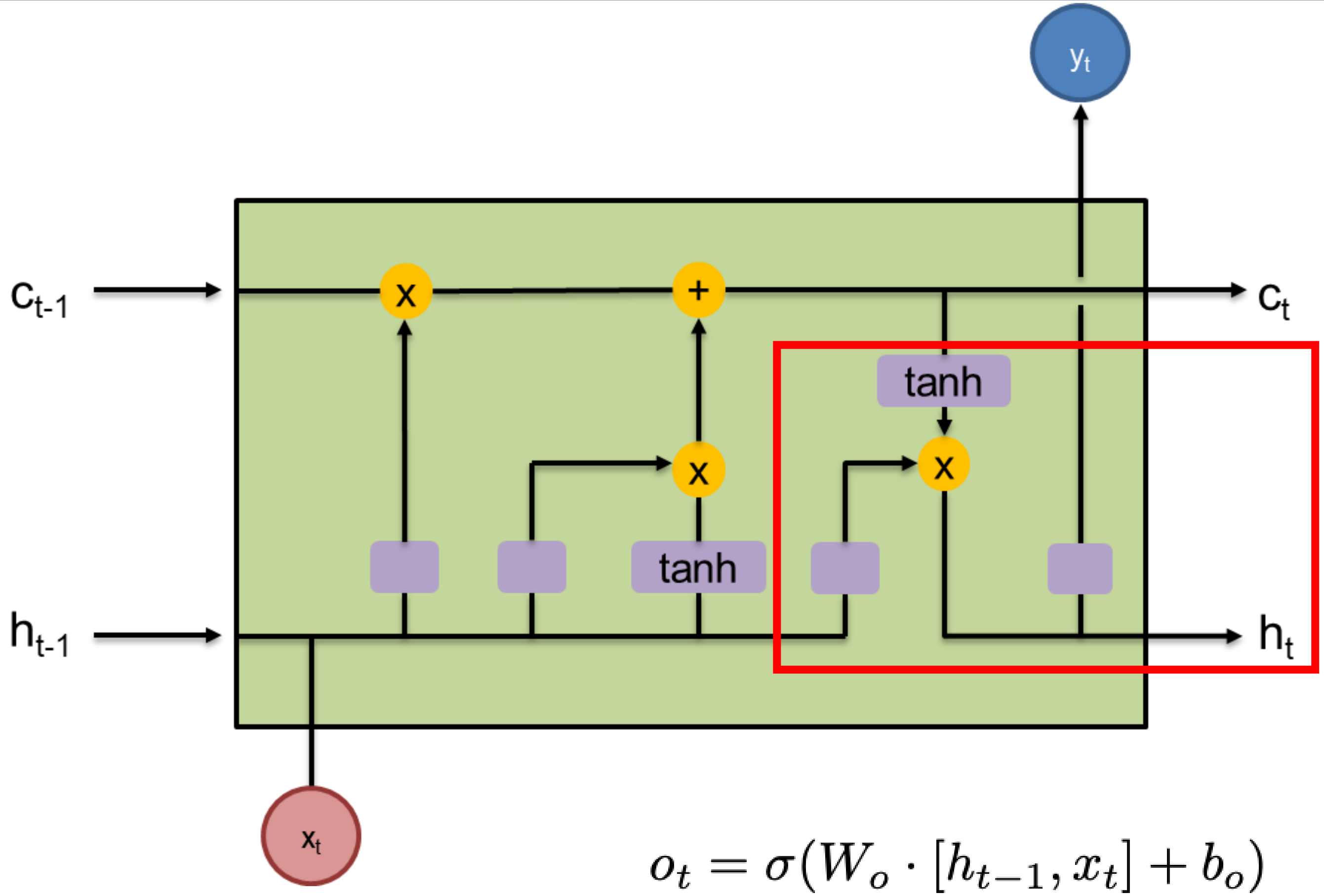


Combining values:

Update the old cell state c_{t-1} into c_t

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

Long Short-Term Memory Network



Output gate:

Define output based on the cell state

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot tanh(c_t)$$
$$y_t = \sigma(W_y \cdot h_t + b_y)$$

Gated Recurrent Units

LSTMs are great in order to avoid vanishing gradients

BUT: Loads of weights and biases which need to be optimized → difficult and slow training

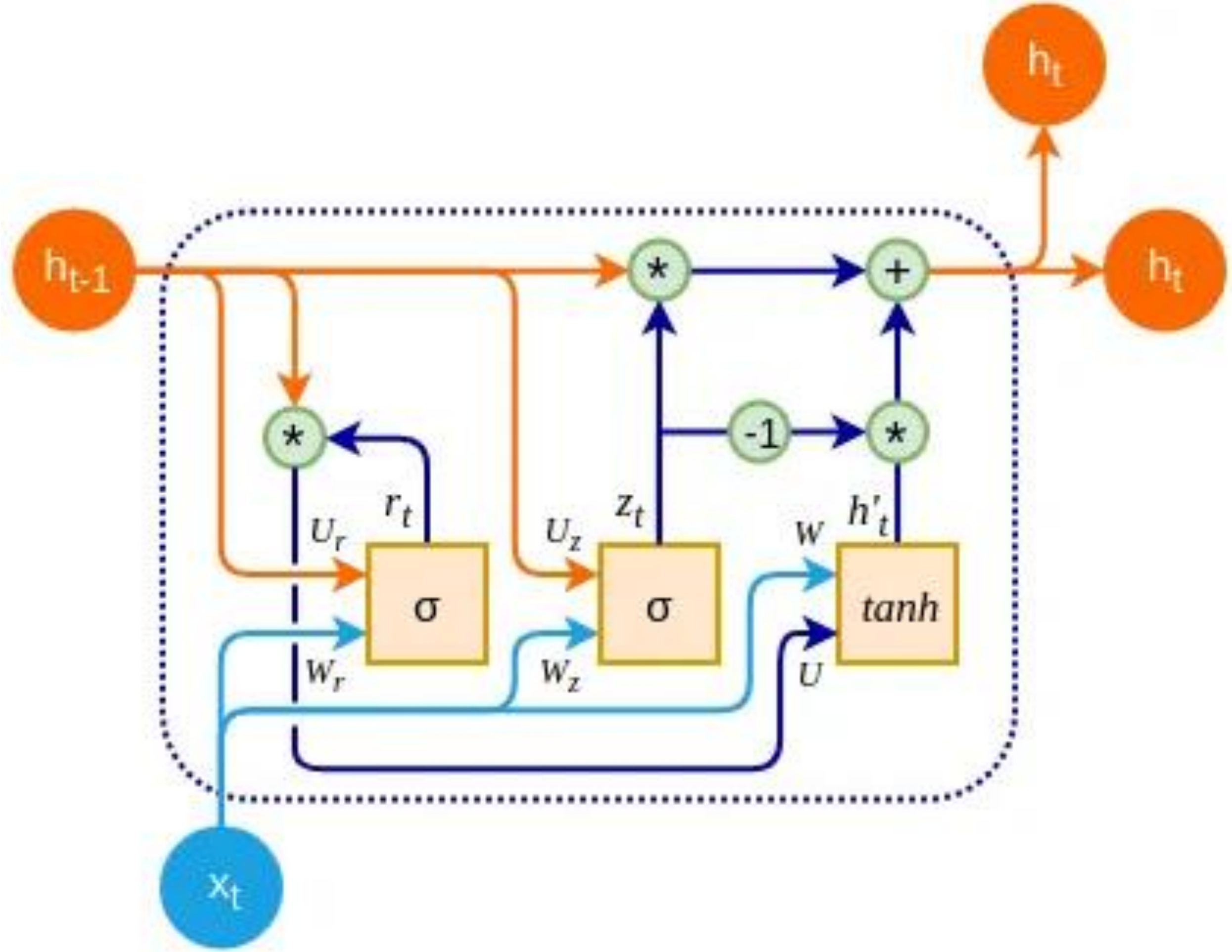
Gated Recurrent Units (GRU) reduce the number of parameters to simplify training

Basically, hidden state and cell state are combined into a single parameter

- Forget and input gate are merged into an update gate

Concept was first presented by Cho et al. in 2014

Gated Recurrent Units



https://medium.com/@anishnama20/understanding-gated-recurrent-unit-gru-in-deep-learning-2e54923f3e2

Any questions?