Friedrich-Alexander-Universität
Technische Fakultät

Machine Learning
Data Analytics

FAU

Machine Learning for Time Series (MLTS)

Lecture 10: Deep Learning for Time Series - Recurrent models

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
19.12.2024

Topics overview

1. Time series fundamentals and definitions (Part 1)
2. Time series fundamentals and definitions (Part 2)
3. Bayesian Inference and Gaussian Processes
4. State space models (Kalman Filters)
5. State space models (Particle Filters)
6. Autoregressive models
7. Data mining on time series
8. Deep Learning (DL) for Time Series (Introduction to DL)
9. DL – Convolutional models (CNNs)
10. DL – Recurrent models (RNNs and LSTMs)
11. DL – Attention-based models (Transformers)
12. DL – From BERT to ChatGPT
13. DL – New Trends in Time Series processing
14. Time series in the real world

Topics overview

1. Time series fundamentals and definitions (Part 1)
2. Time series fundamentals and definitions (Part 2)
3. Bayesian Inference and Gaussian Processes
4. State space models (Kalman Filters)
5. State space models (Particle Filters)
6. Autoregressive models
7. Data mining on time series
8. Deep Learning (DL) for Time Series (Introduction to DL)
9. DL – Convolutional models (CNNs)
10. DL – Recurrent models (RNNs and LSTMs)
11. DL – Attention-based models (Transformers)
12. DL – From BERT to ChatGPT
13. DL – New Trends in Time Series processing
14. Time series in the real world

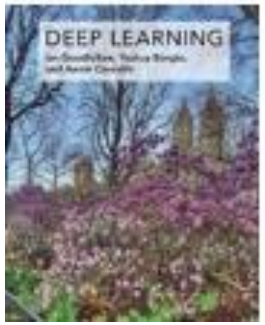
In this lecture…

1. *Concepts recap*
2. *Recurrent Neural Networks (RNNs)*
3. *Backpropagation Trough Time (BPTT)*
4. *Long-term dependencies*
5. *Long Short-term Memory (LSTMs) Networks*
6. *Recap*

References

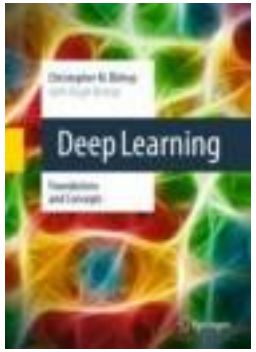
Deep Learning

by Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016)



Deep Learning: Foundations and Concepts

by C. Bishop, H. Bishop (2024)



Deep learning for Time Series – Recurrent models

Concepts recap

Multilayer perceptron (MLP)

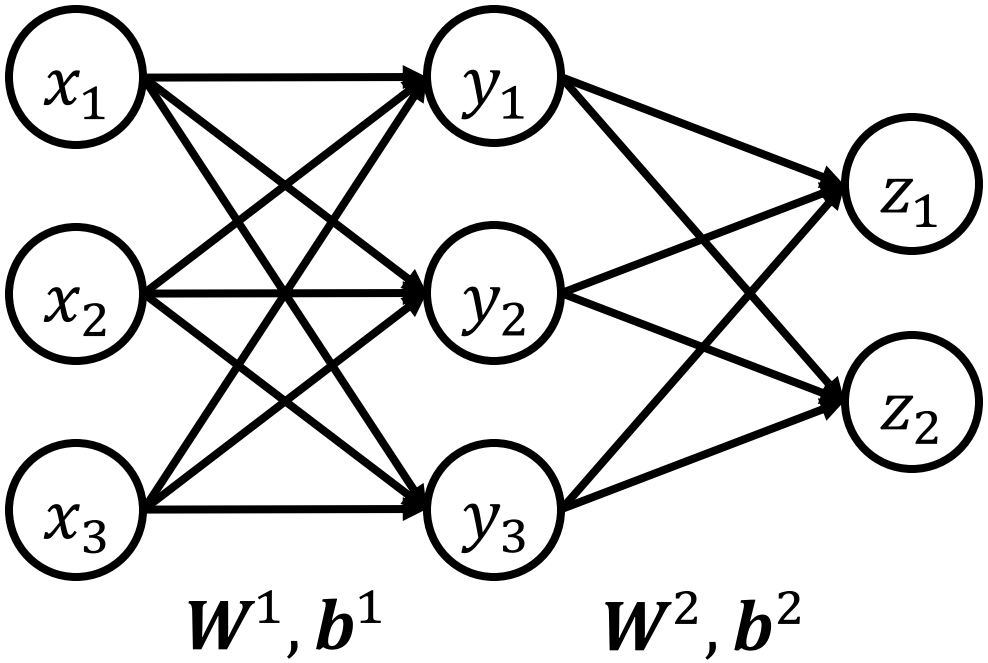
We can chain multiple layers, with each output being the input of the next:

$$y = \sigma(W^1 \cdot x + b^1)$$

$$z = \sigma(W^2 \cdot y + b^2)$$

Combining these leads us to:

$$z = \sigma(W^2 \cdot \sigma(W^1 \cdot x + b^1) + b^2)$$



- Each layer has its own set of parameters (weights $\boldsymbol{W^i}$ and bias $\boldsymbol{b^i}$)
- The underlying computation is a matrix multiplication described by

$$\boldsymbol{y^{i+1} = \sigma(W^i \cdot y^i + b^i)}$$

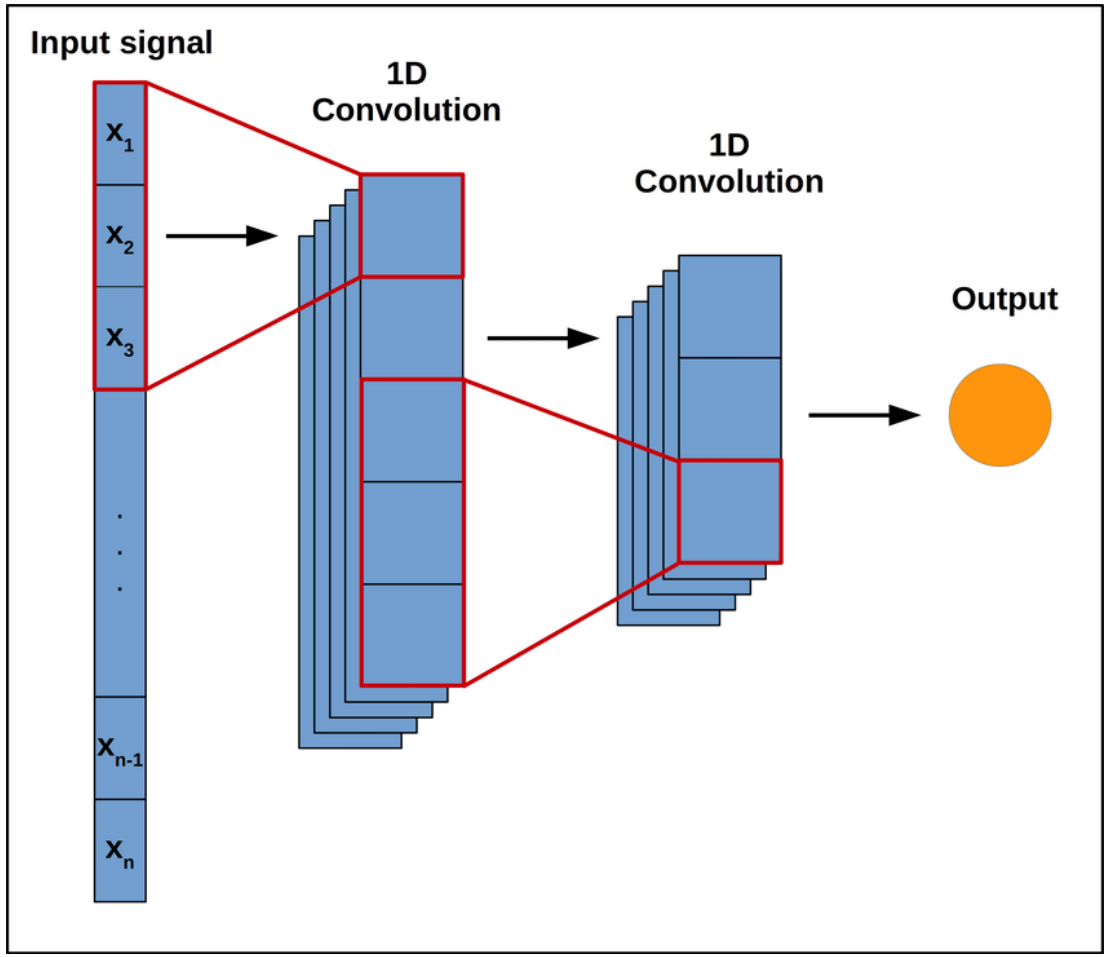
Backpropagation (BP): algorithmic view

1. Calculate the forward pass and store results for $\hat{y}$, a_j^k, and o_j^k.
2. Calculate the backward pass and store results for $\frac{\partial E}{\partial w_{ij}^k}$, proceeding from the last layer:
    a) Evaluate the error terms for the last layer δ_j^m
    b) Backpropagate the error term for the computation of δ_j^k
    c) Proceed to all previous layers
3. Combine the individual gradients $\forall j$ (simple average)
4. Update the weights according to a lerning rate λ

1-D Convolutional Neural Networks

CNNs can also be used to learn **temporal dependencies** on time series data.

- The convolution is applied along a single dimension, i.e., the **temporal** dimension.
- The resulting model is generally referred to as 1-D CNN.



Picture from Shenfield, Alex & Howarth, Martin. (2020). A Novel Deep Learning Model for the Detection and Identification of Rolling Element-Bearing Faults. Sensors (Basel, Switzerland). 20. 10.3390/s20185112.

Temporal Convolutional Networks (TCNs)

The **causal convolution** is best suited to model causality in the data.

- For images it can be implemented by “masked convolutions”, i.e., a tensor mask is applied before the actual convolution takes place.
- For 1D data, e.g., audio processing, it can be more easily implemented by shifting the output of a normal convolution by a few timesteps.

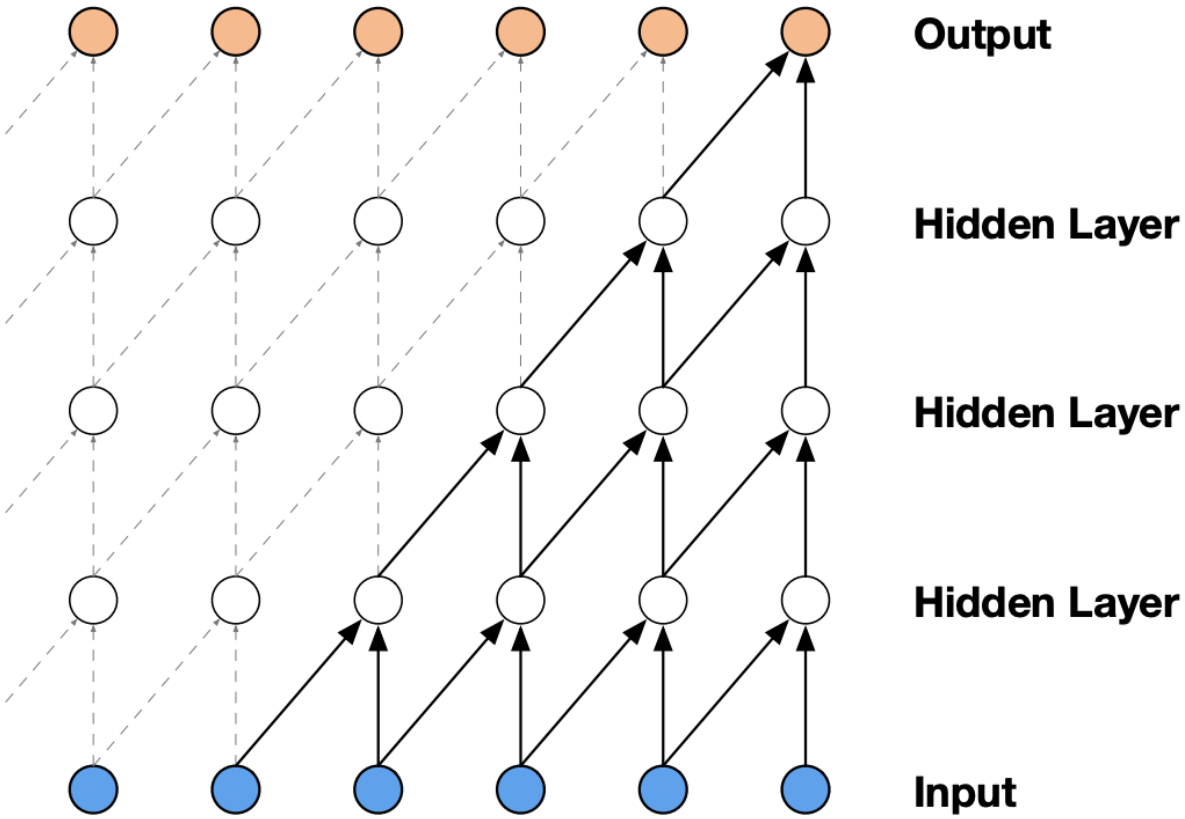


Image from Oord, A. V. D. (2016). WaveNet: A Generative Model for Raw Audio..

Deep learning for Time Series – Recurrent models

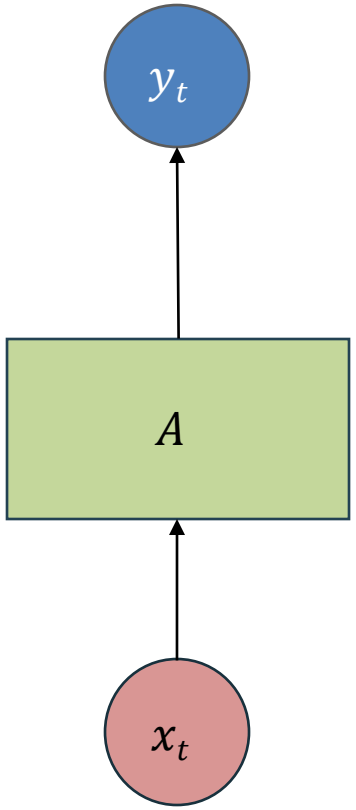
Recurrent Neural Networks (RNNs)

Limitations of NN for time series data

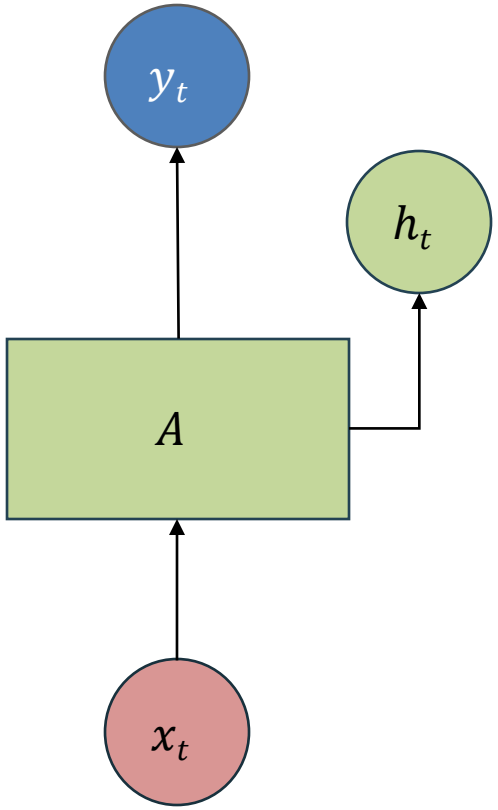
Feed-forward and Convolutional neural networks present some **disadvantages**, when applied to sequential data:

1. Cannot work online (sequence has to be fed all at once)
2. Consider only the current input
    - Canot memorize previous time steps
3. Cannot handle directly sequences of different lengths

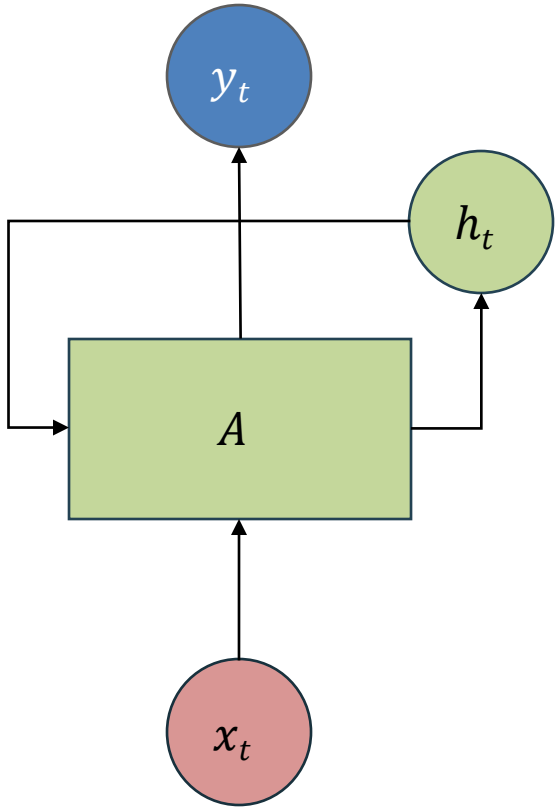
Recurrent Neural Network (RNN)



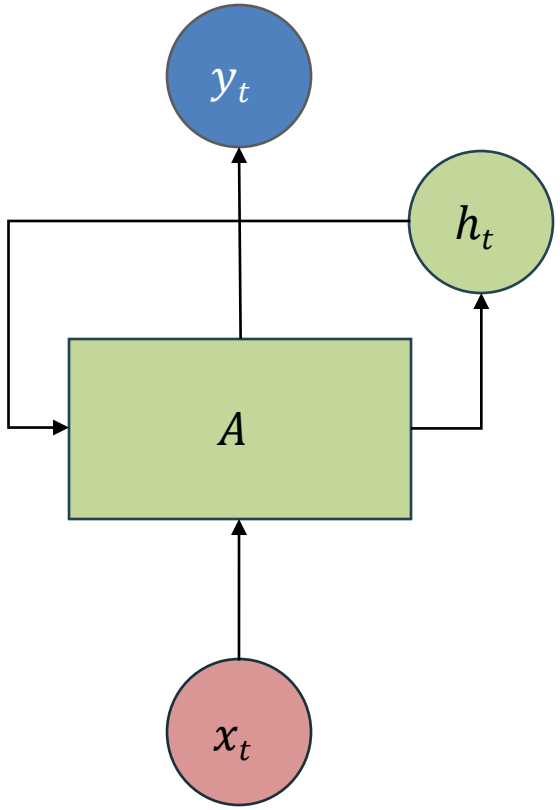
Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)



A **recurrent neural network (RNN)** is a neural network that contains feed-back connections.

- Activations can flow in a loop
- It allows for temporal processing

An RNN is composed by:

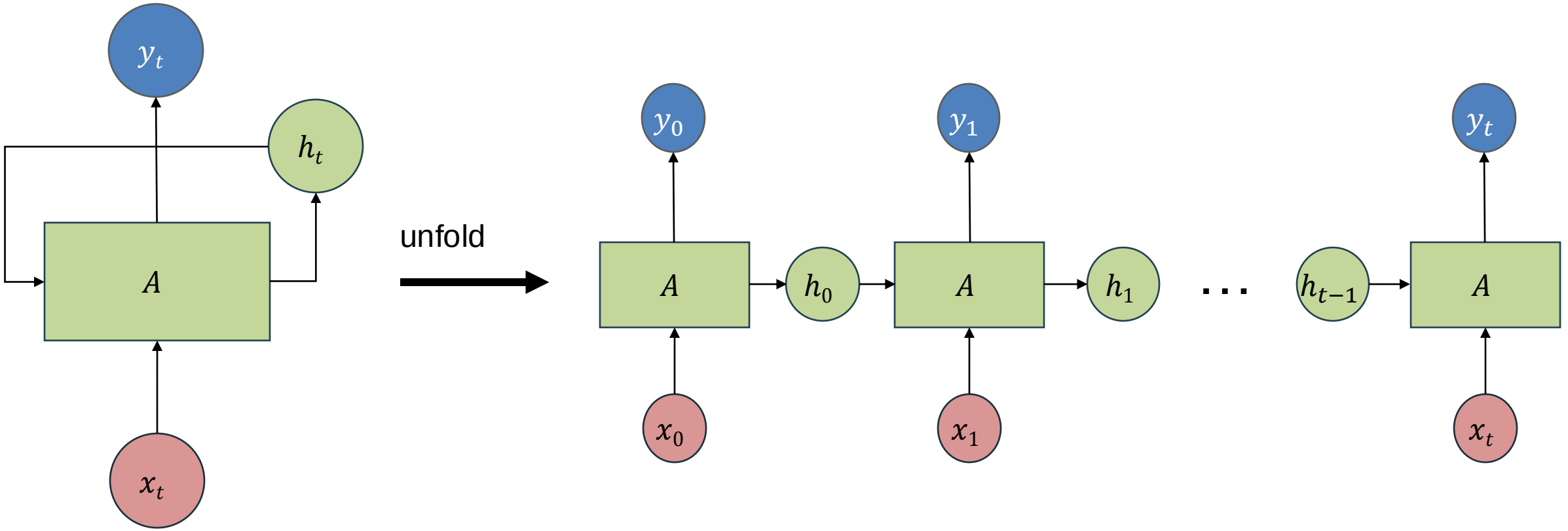
x_t: input at time t

A: neural network

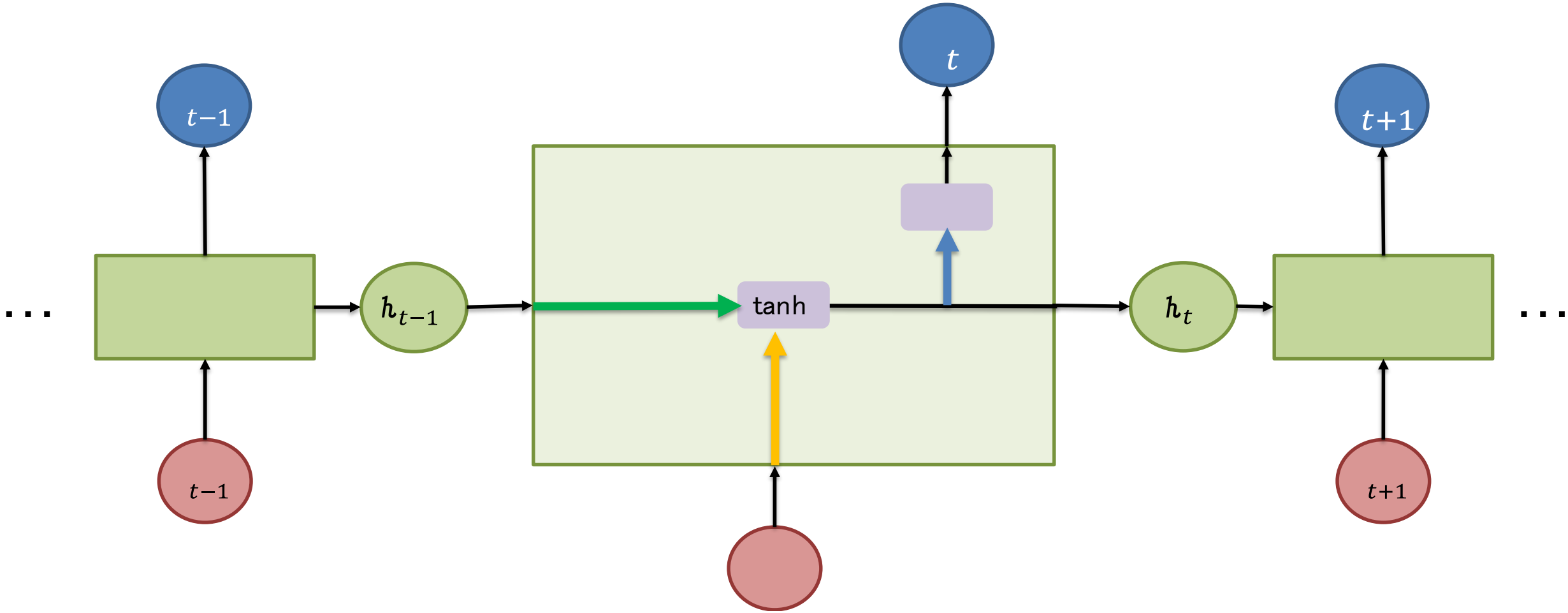
h_t: hidden state

y_t: output at time t

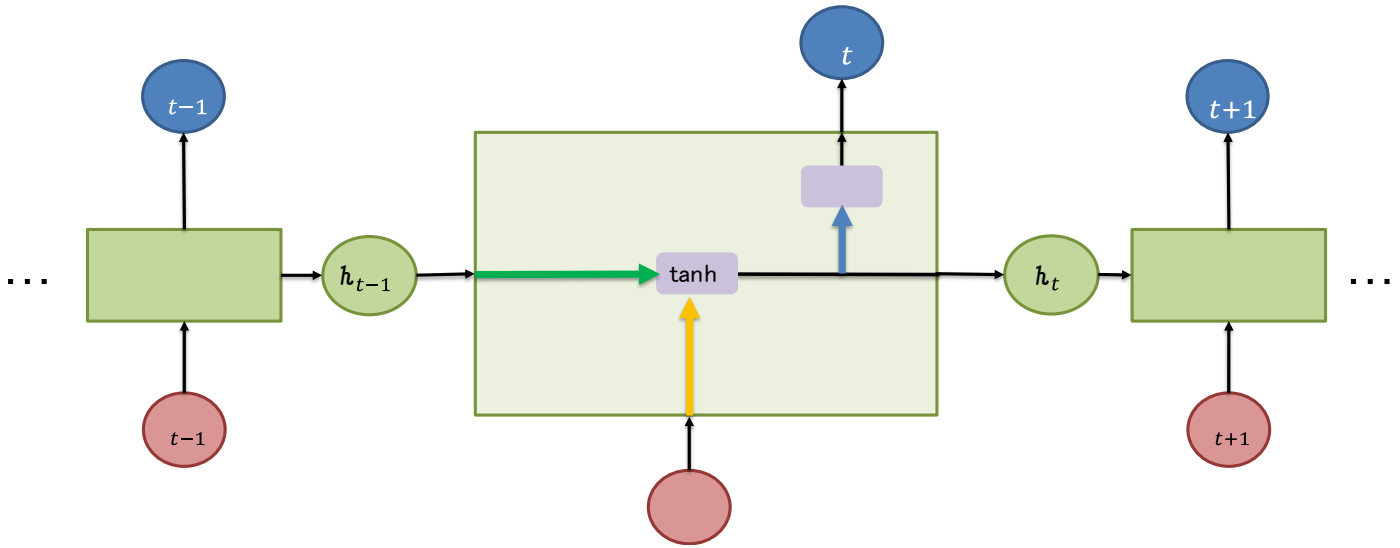
RNN Unfolding



Mathematical formulation



Mathematical formulation



The behaviour of the RNN can be described as a **dynamical system** by the pair of non-linear matrix equations:

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

$$y_t = \sigma(\boldsymbol{W_{hy}} h_t)$$

The order of the dynamical system corresponds to the dimensionality of the state h_t.

Mathematical formulation

In general, the neural network A can represent any function. We can write a more general formulation of the system as:

$$h_t = f_h(h_{t-1}, x_t; \theta_h)$$

$$y_t = f_y(h_t; \theta_y)$$

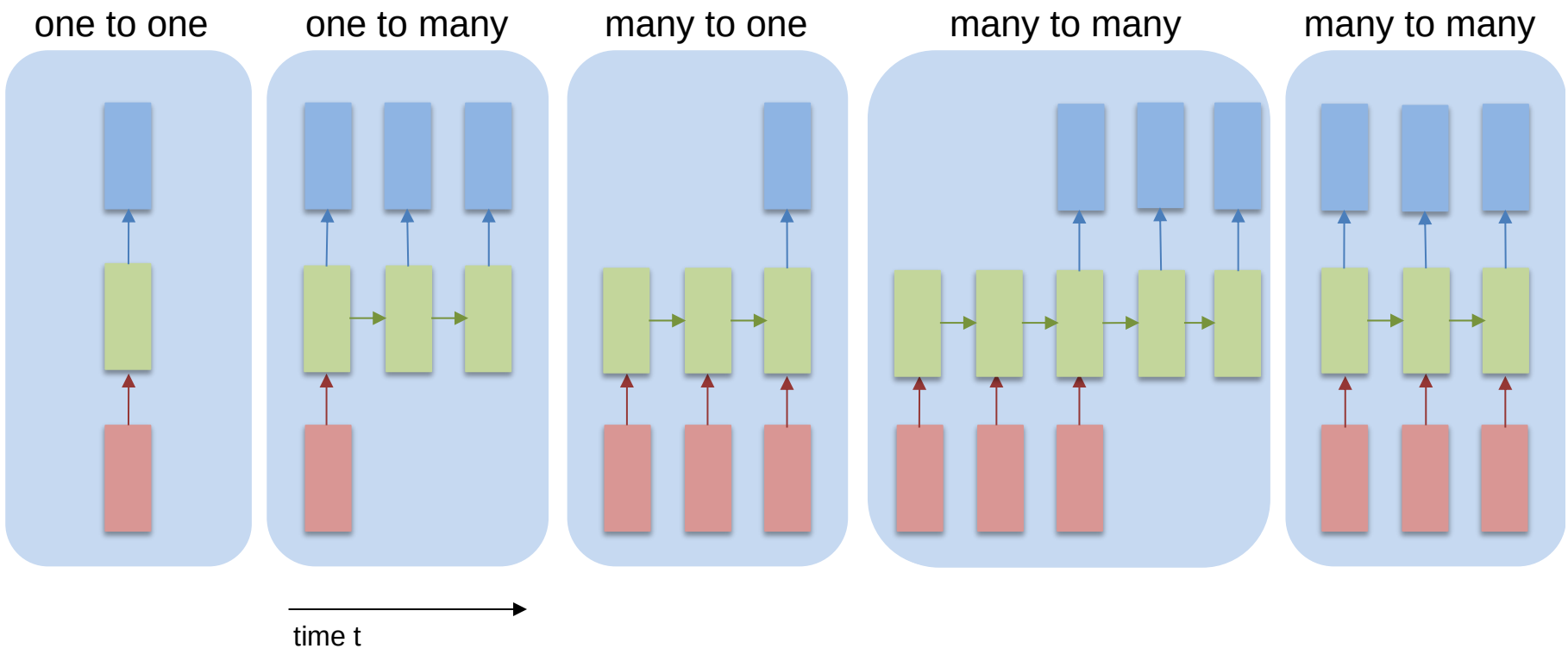
RNNs are universal approximators

The Universal Approximation Theorem states that:

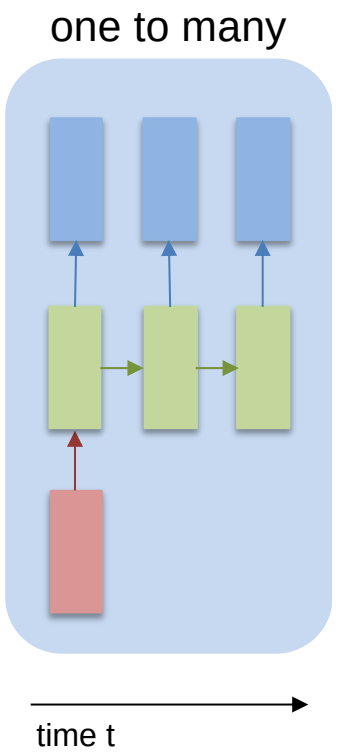
> *„Any non-linear dynamical system can be approximated to any accuracy by a recurrent neural network, with no restrictions on the compactness of the state space, provided that the network has enough sigmoidal hidden units.“*

- Knowing that RNNs are universal approximators does not explain how to learn such dynamical system from data.
- Since we can think about RNNs in terms of dynamical systems, we can also investigate their properties like stability, controllability and observability.

RNNs architecture

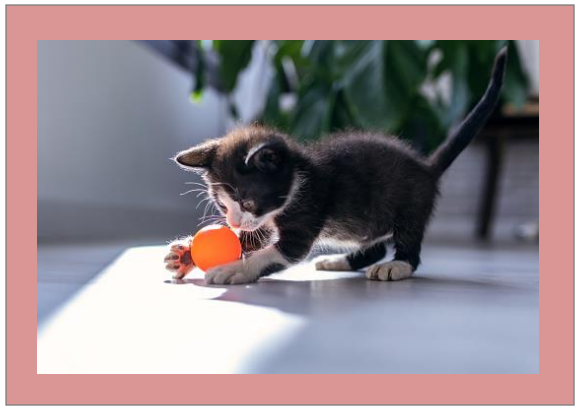


Example: one to many



A typical example of a one to many problem is that of **image captioning.**

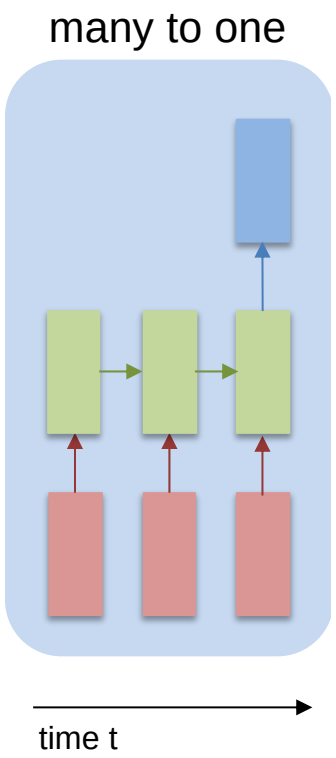
Input:



Output:

A cat playing with a ball

Example: many to one



A typical example of a many to one problem is that of **sentiment analysis.**

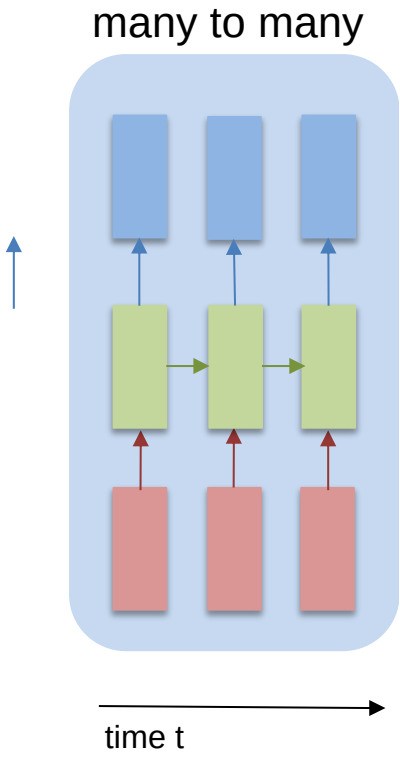
Input:

Horrible	service	the	room	was	dirty

Output:

("negative")

Example: many to many



A typical example of a many to many problem is that of name entity recognition.

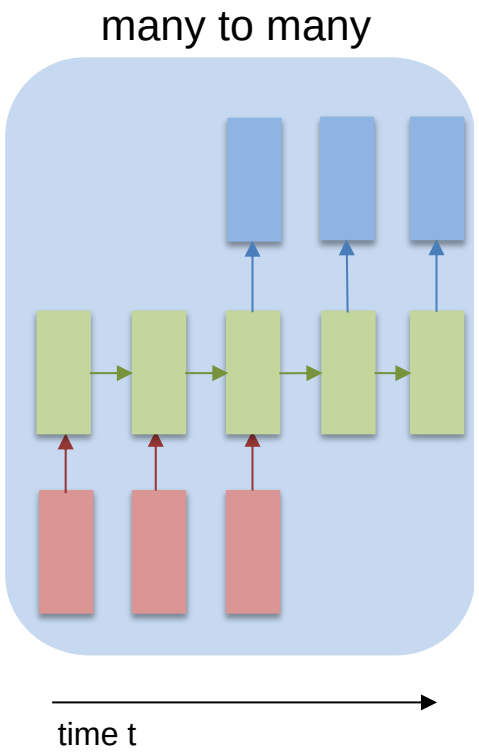
Input:

Harry	Potter	and	Hermione	invented	a	new	spell

Output:

1	1	0	1	0	0	0	0

Example: many to many



Another example of a many to many problem is that of **machine translation.**

Input:

Horrible	service	the	room	was	dirty

Output:

Un	servizio	orribile	la	camera	era	sporca

Deep learning for Time Series – Recurrent models

Backpropagation Through Time (BPTT)

Backpropagation Through Time (BPTT)

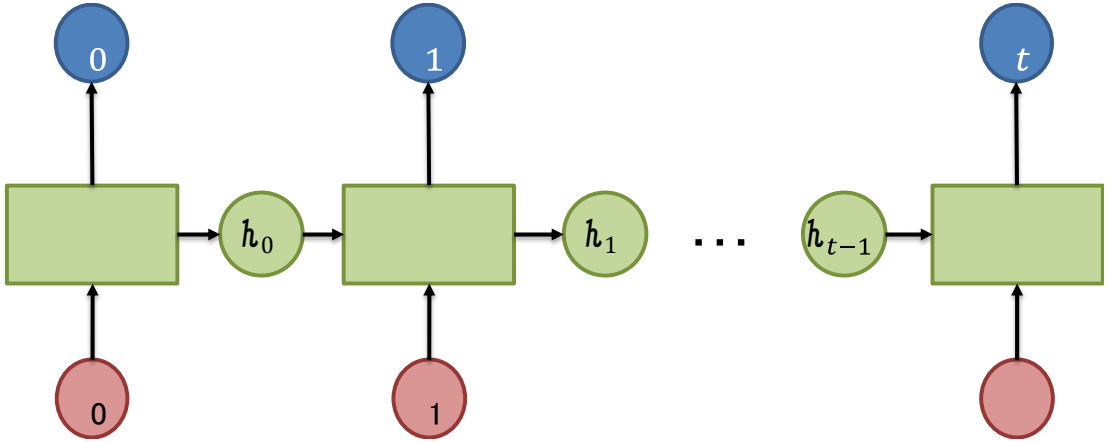
Backpropagation Through Time (BPTT) learning algorithm is a natural extension of the standard backpropagation that performs gradient descent on a complete unfolded RNN.

General idea:

1. Feed training examples through the network
2. Calculate the loss for the whole sequence
3. Get the gradients of all weights and update them according to the learning rate

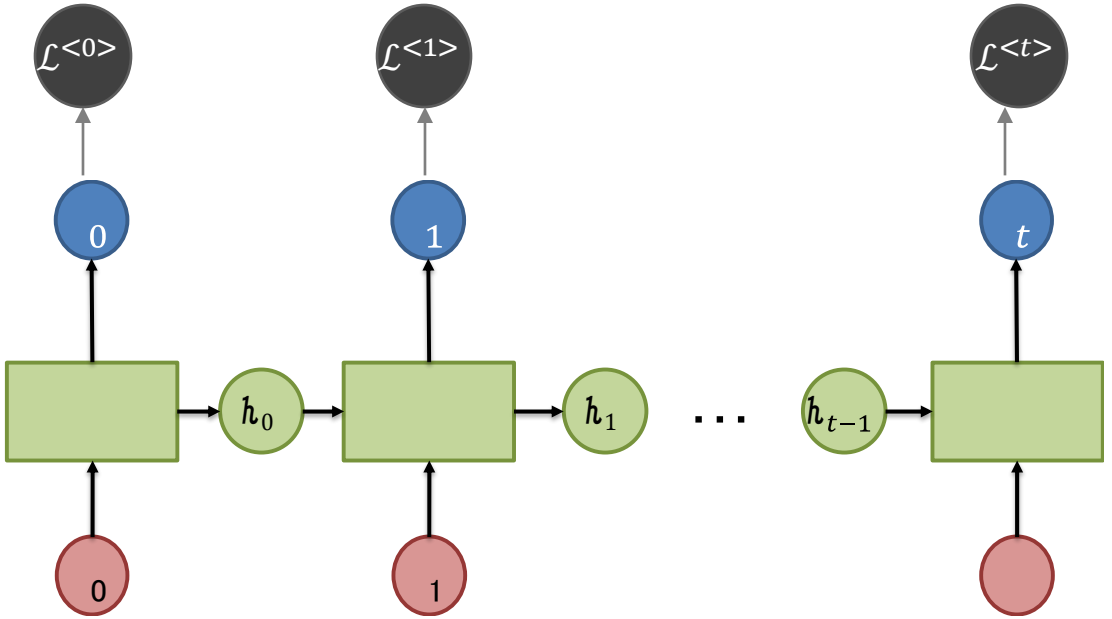
BPTT: Forward propagation

1. We feed the network with the whole sequence and compute all activations and outputs



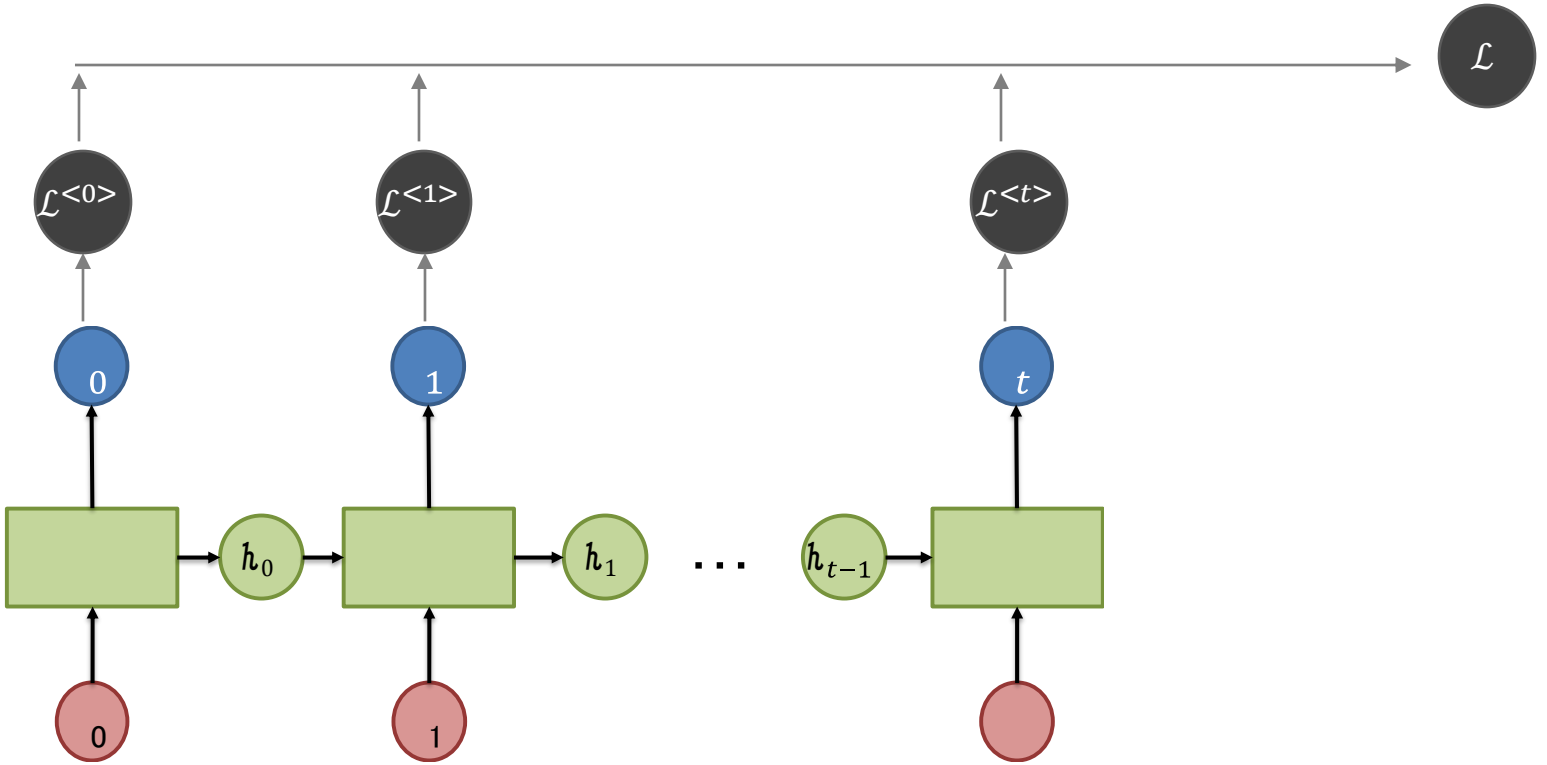
BPTT: Loss computation

2. We compute the overall loss of our prediction $\hat{y}$ w.r.t. the true sequence y.



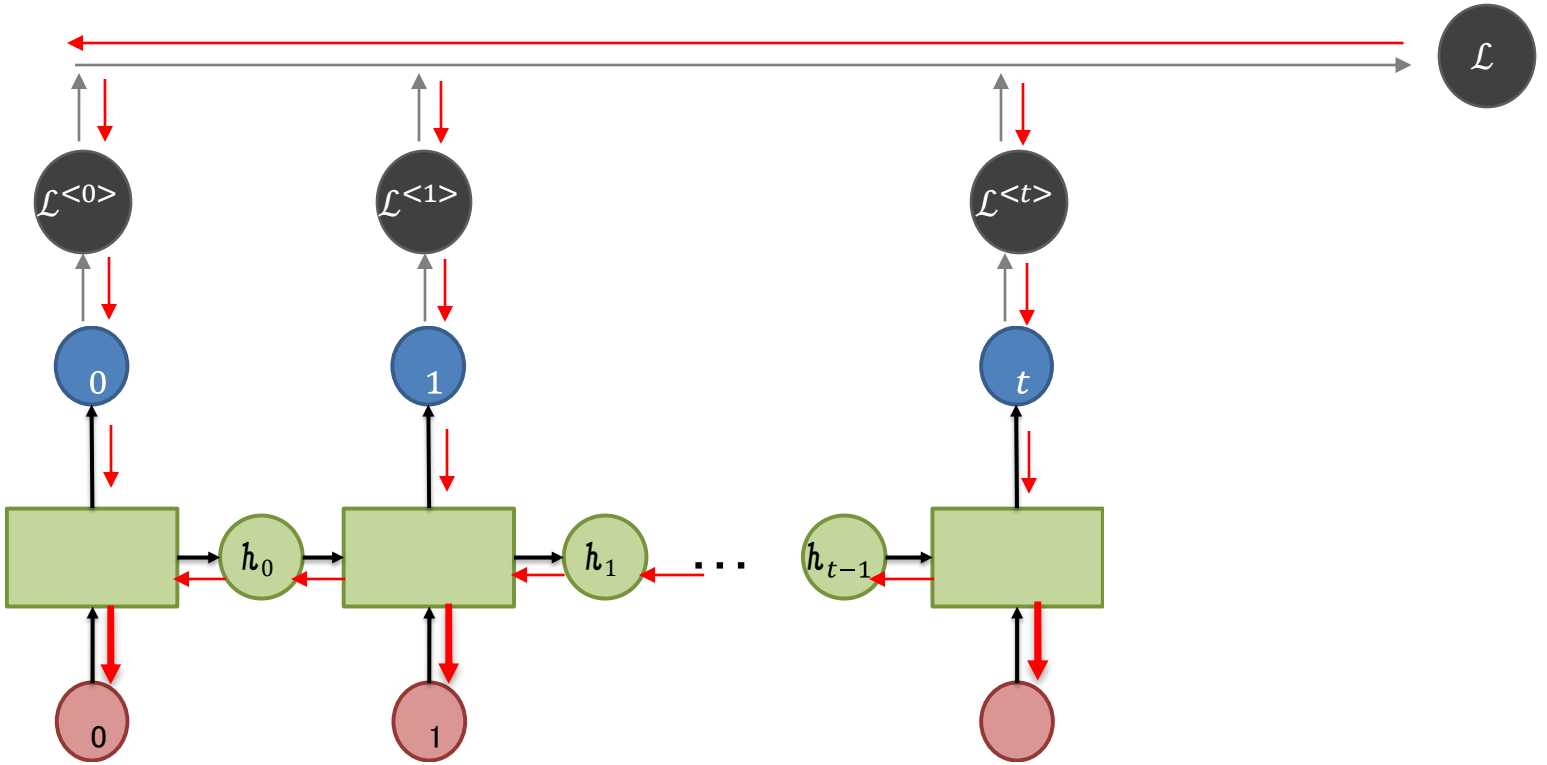
BPTT: Loss computation

2. We compute the overall loss of our prediction $\hat{y}$ w.r.t. the true sequence y.



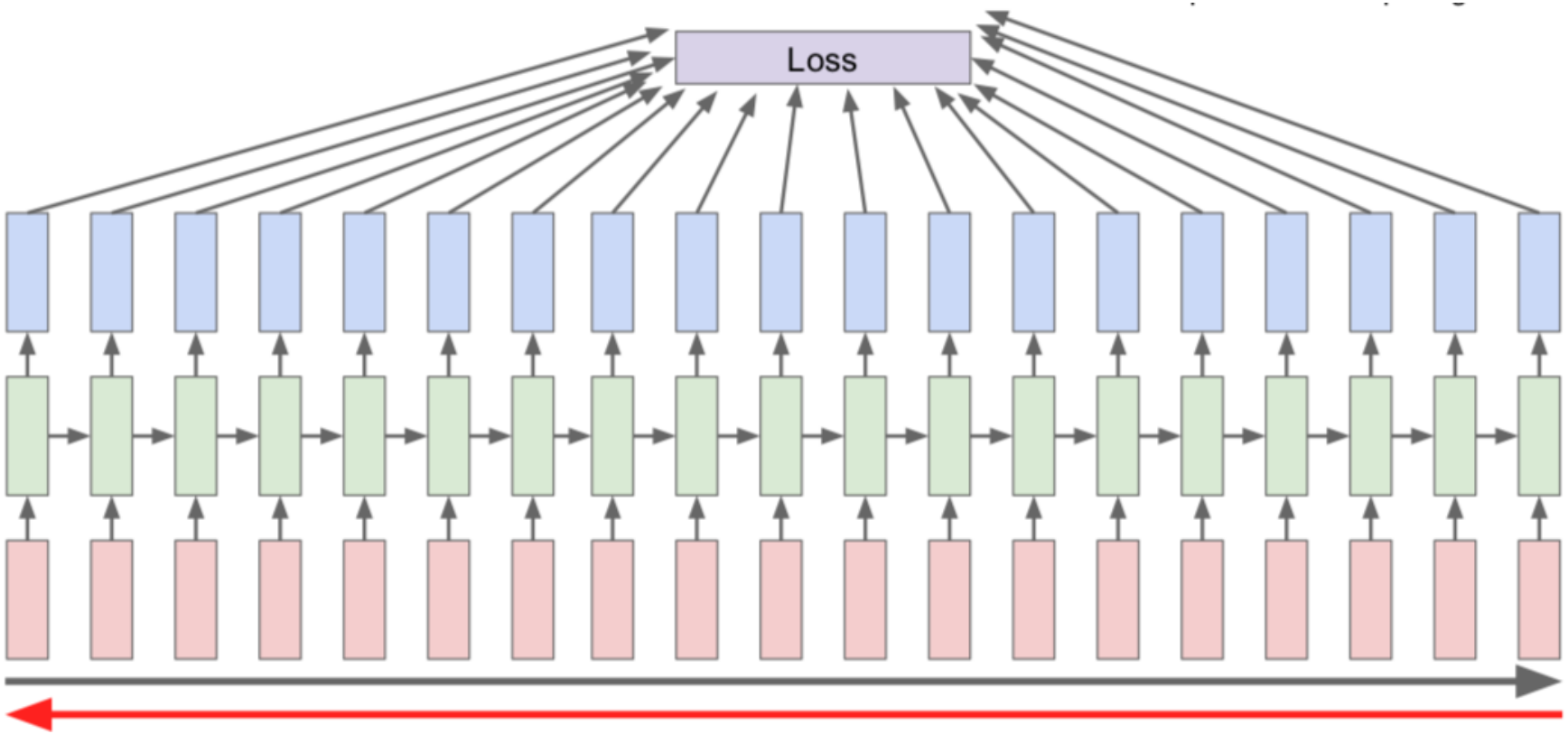
BPTT: Backpropagation

3. Get the gradients for all weights, and update the matrices using gradient descent.



BPTT: Limitations

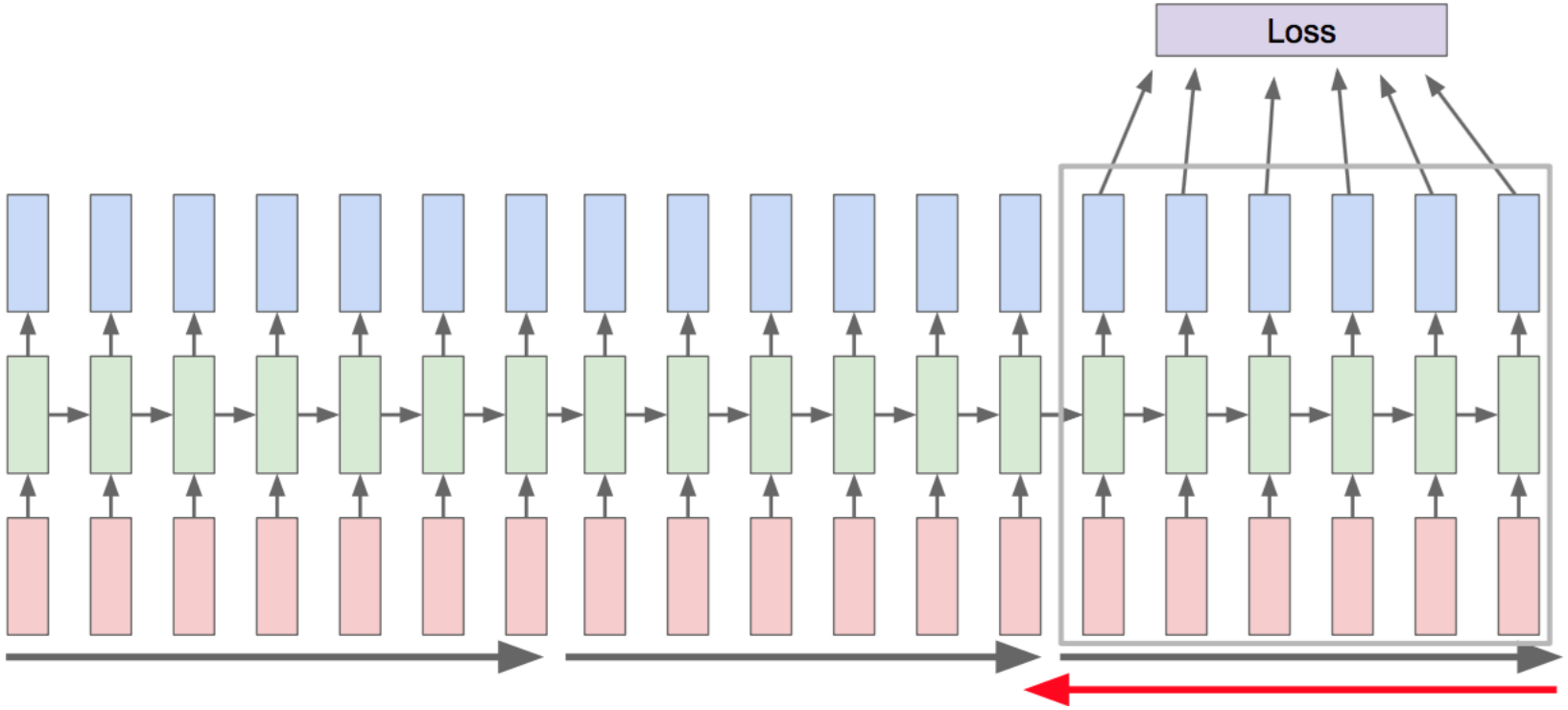
BPTT can be computationally very expensive as a lot of partial derivatives have to be computed, depending on the complexity of the network.



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

Truncated Backpropagation Through Time (Trunc-BPTT)

With the Truncated Backpropagation Through Time **(Trunc-BPTT)**, instead of passing the whole sequence, we perform the forward and backpard pass on a subset.



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

Deep Learning for Time Series – Recurrent models

Long-term dependencies

Long-term dependencies

Definition. A task is displaying **long-term dependencies** if the prediction of the desired output at time t depends on input presented at an earlier time $\tau \ll t$.

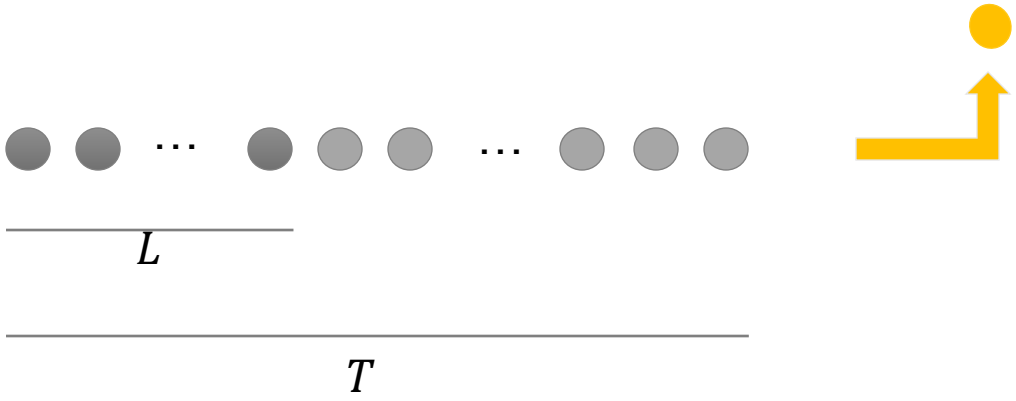
- In presence of long-term dependencies, RNNs can outperform a static FF-NN, although they appear **more difficult to train** optimally.
- Generally, their parameters tend to settle in **sub-optimal solutions that take into account short-term dependences**, but not long-term dependencies.

The symbol „≪” means “much less than”.

Example: minimal task with long-term dependencies

The problem can be solved only if the system is capable of storing the information about the **initial input values** for an arbitrary duration.

The values $(s_{L+1}, \dots, s_T)$ are irrelevant and can be regarded as *noise*. They can have the effect of erasing the internal information about the initial values of the input.



Vanishing/Exploding gradient problem

Exploding gradient. When the gradients of the loss function with respect to the network parameters are very large.

→ Learning is unstable

→ A possible solution is **gradient clipping,** i.e., cutting off gradients which are greater than a threshold during backpropagation.

Vanishing gradient. When the gradients of the loss function with respect to the network parameters are very small.

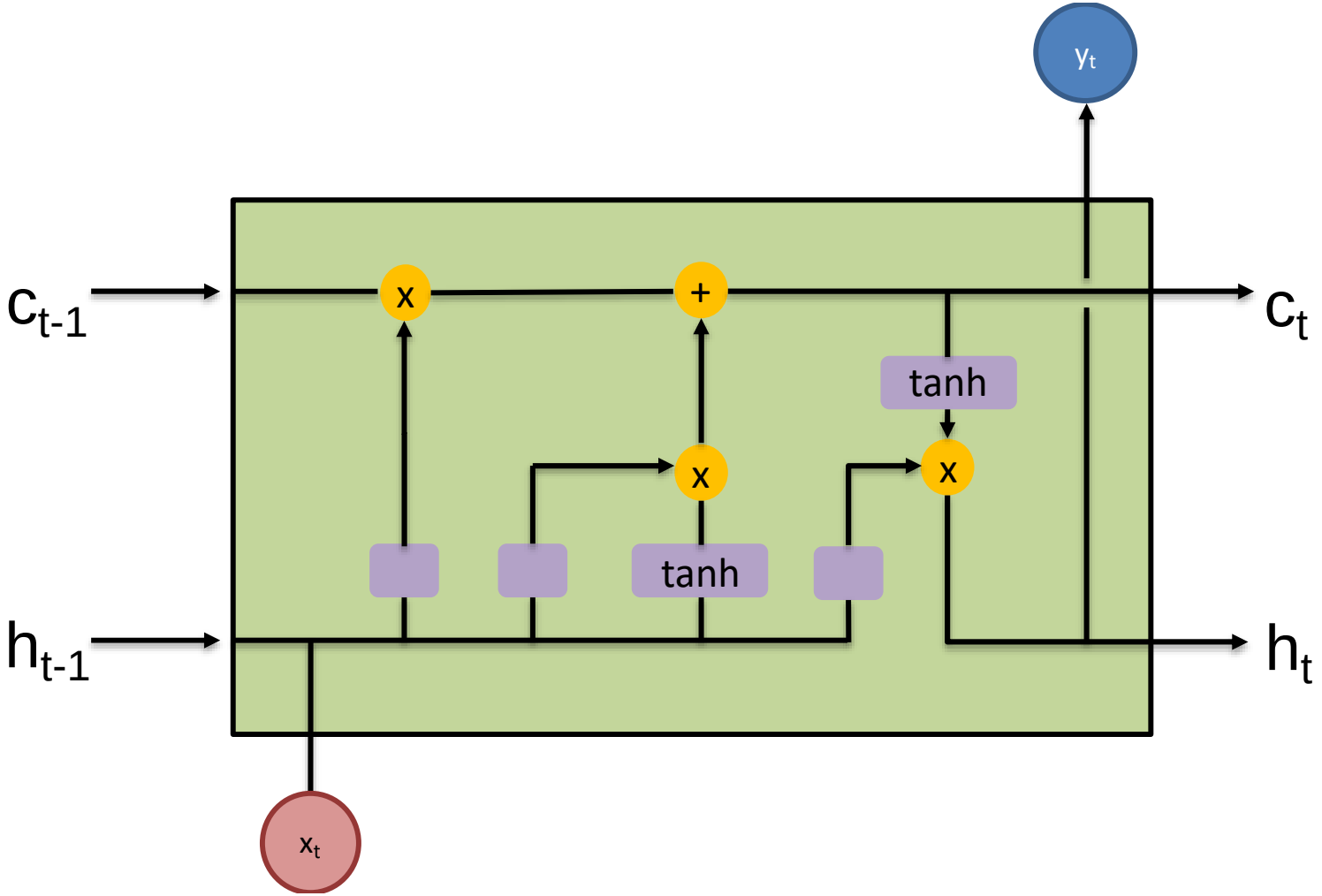
→ Learning is slow

→ A possible solution is to introduce „shortcuts“ in the architecture, e.g., **gates**.

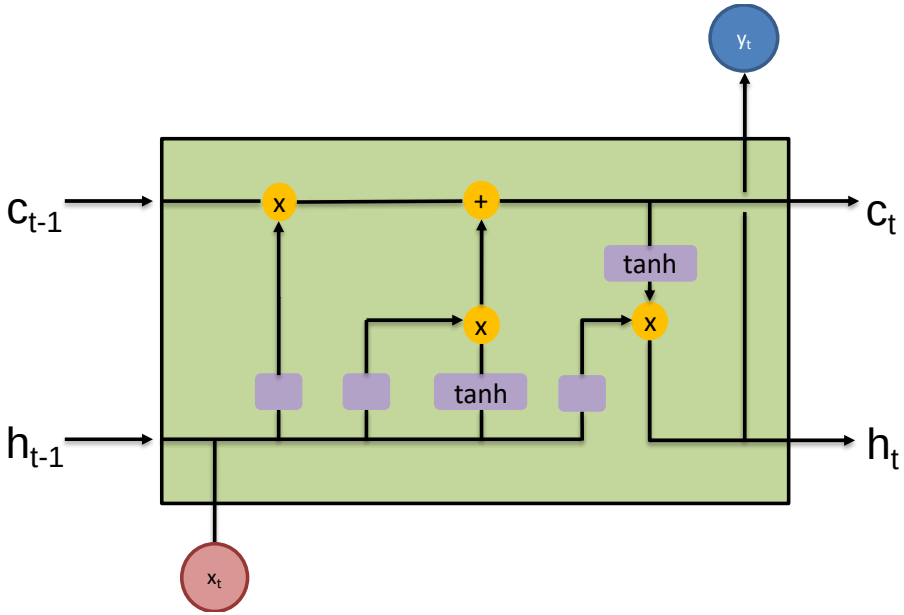
Deep Learning for Time Series – Recurrent models

Long short-term Memory (LSTMs) Networks

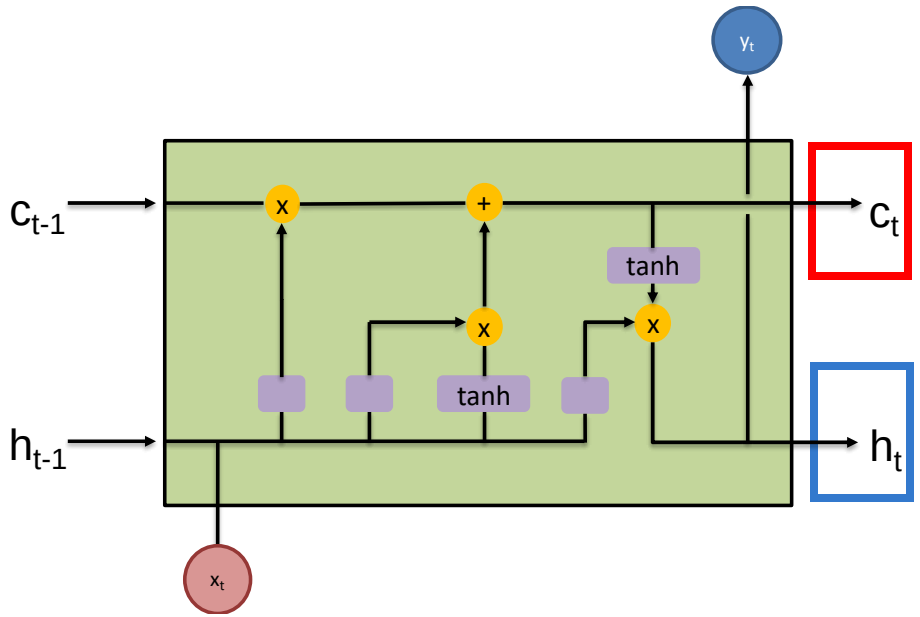
Long-short term memory networks (LSTMs)



Long-short term memory networks (LSTMs)



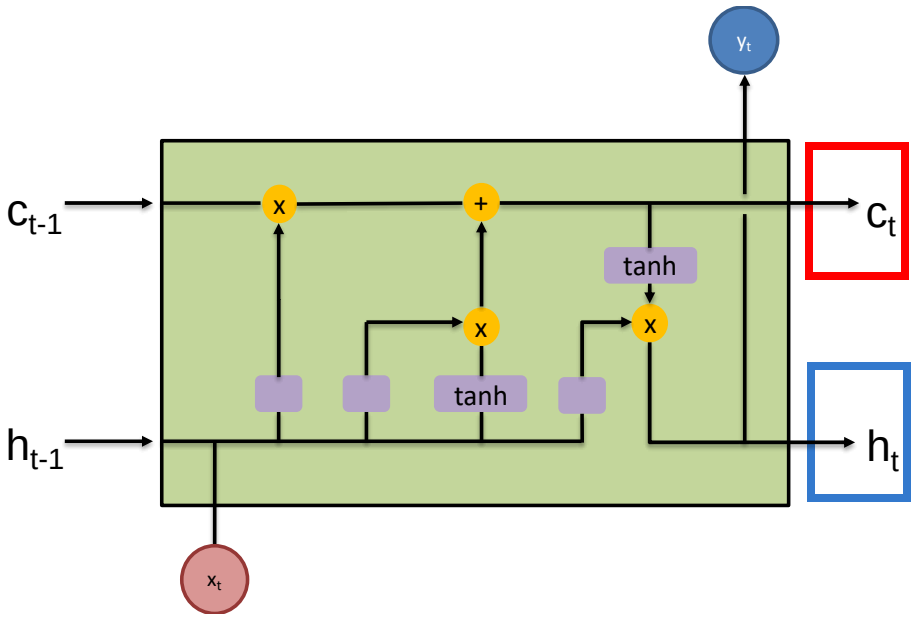
Long-short term memory networks (LSTMs)



Observations:

- Compared to RNNs, additionally to the **hidden state** h_t, we have another state, c_t, said the **cell state**.

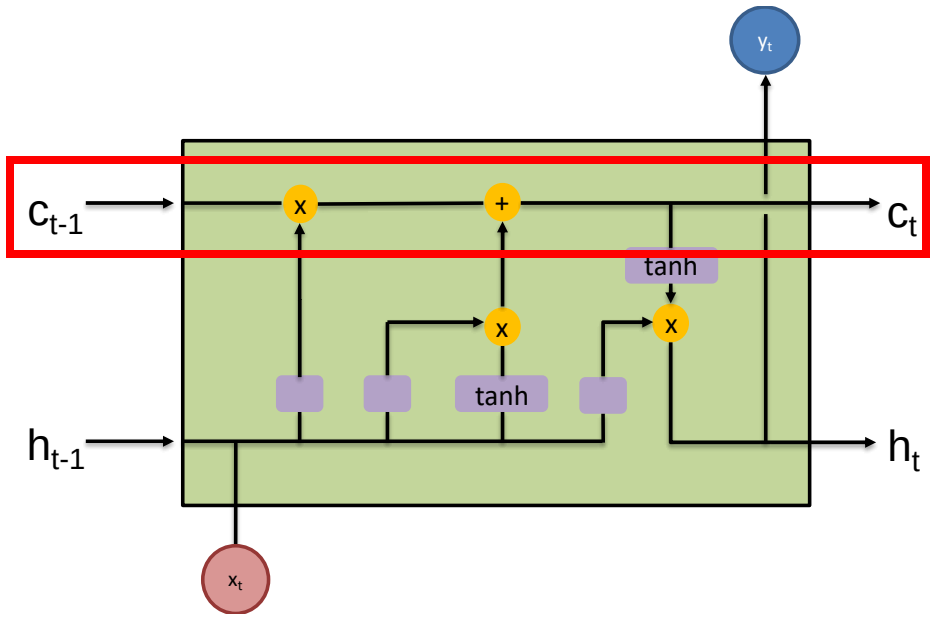
Long-short term memory networks (LSTMs)



Observations:

- Compared to RNNs, additionally to the **hidden state** h_t, we have another state, c_t, said the **cell state**.
- The information flow is regulated by **three gates**: the forget gate, the input gate and the output gate.

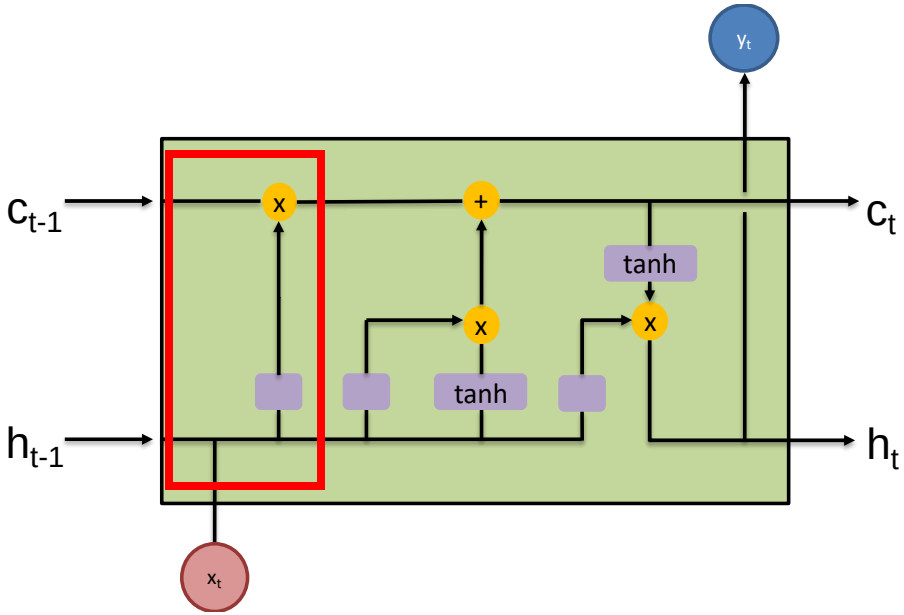
Long-short term memory networks (LSTMs)



The cell state has:

- Only minor interactions
- Simple information flow
- Other gates regulates weather it is preserved/not-preserved or updated.

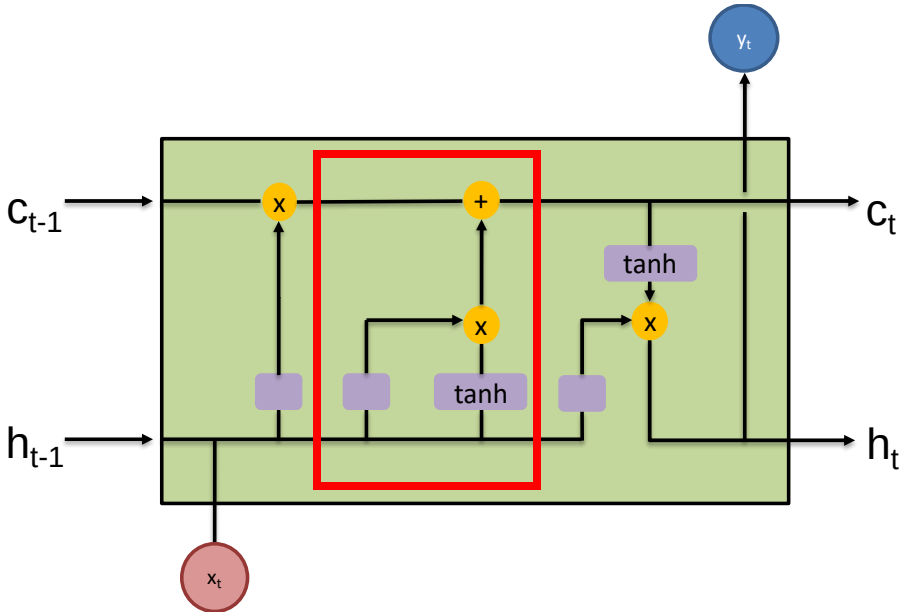
Long-short term memory networks (LSTMs)



The **forget gate** decides how much information to retain from the previous cell state.

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

Long-short term memory networks (LSTMs)

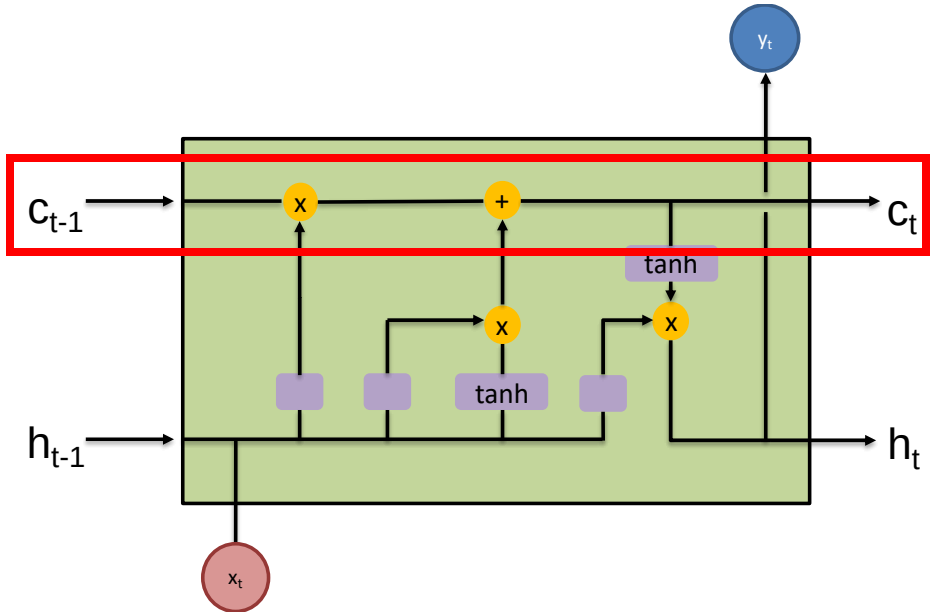


The **input gate** decides the information to be added to the cell state, based on the current input.

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

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

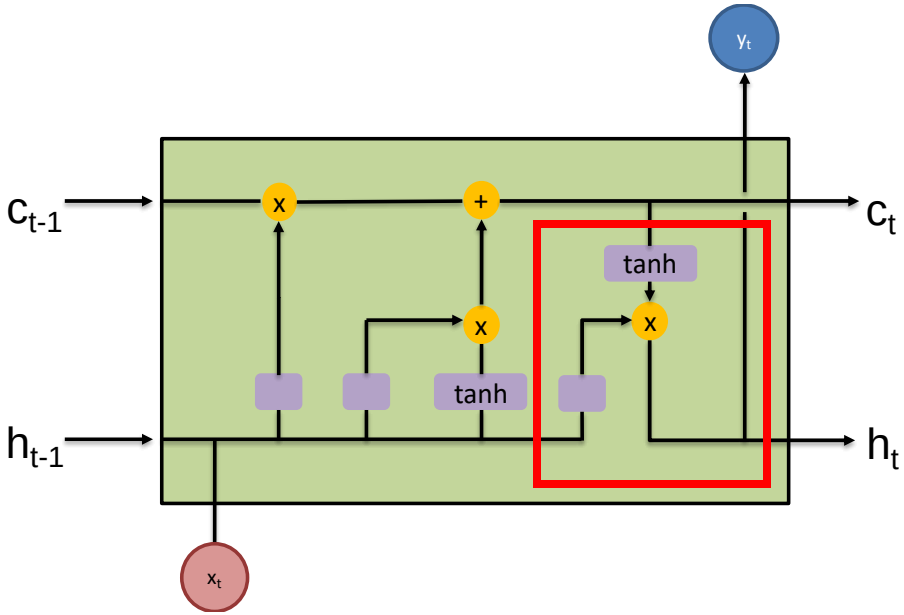
Long-short term memory networks (LSTMs)



The values can be combined to update the cell state

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

Long-short term memory networks (LSTMs)

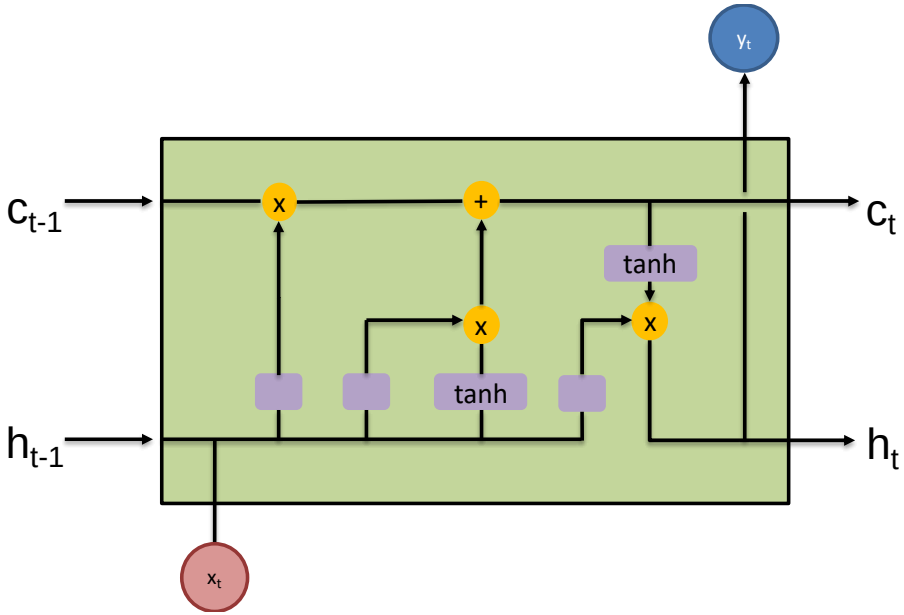


The **output gate** generates the output of the current LSTM cell, based on the current input, the previous output, and the updated cell state.

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

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

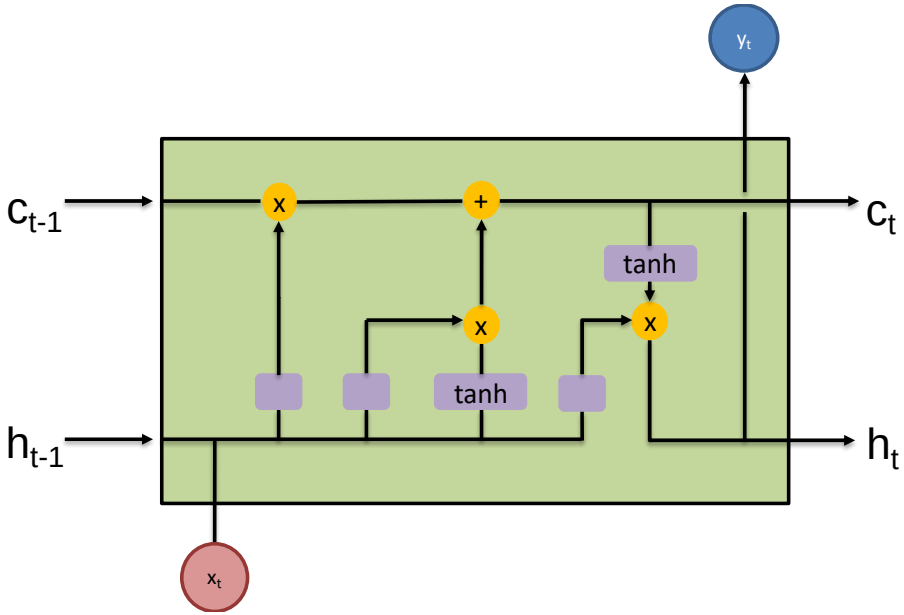
Long-short term memory networks (LSTMs)



To summarize, the LSTM cell is described by the following equations:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\mathrm{g_t} = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \mathrm{g_t}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(c_t)$$

Long-short term memory networks (LSTMs)



From the original publication [1]:

"*Each memory cell's internal architecture guarantees constant error flow within its constant error carrousel CEC… This represents the basis for bridging very long time lags. Two gate units learn to open and close access to error flow within each memory cell's CEC. The multiplicative input gate affords protection of the CEC from perturbation by irrelevant inputs. Likewise, the multiplicative output gate protects other units from perturbation by currently irrelevant memory contents.*"

[1] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

Deep Learning for Time Series – Recurrent models

Recap

In this lecture

- **RNNs**
- **BPTT**
- **Long-term dependencies**
- **LSTMs**

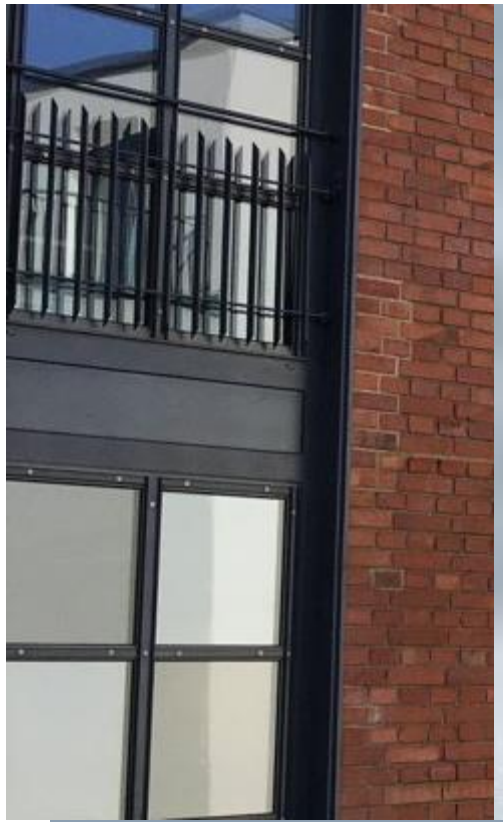
Recurrent models: Pros and Cons

Pros of recurrent models:

- Regardless of the sequence length, the learned model always has the same input size
    - They perform better on dataset with sequences of variable-length.
- It is possible to use same transition for all time steps.

Cons of recurrent models:

- Vanishing/Exploding gradient
  → Attenuated in LSTMs




Happy
Holidays

Alternatives to LSTMs

We can define different gated recurrent neural network architectures, e.g.,

→ Fit a specific problem

→ Reduce computations

→ Adapt to different data characteristics

Alternatives to LSTMs:

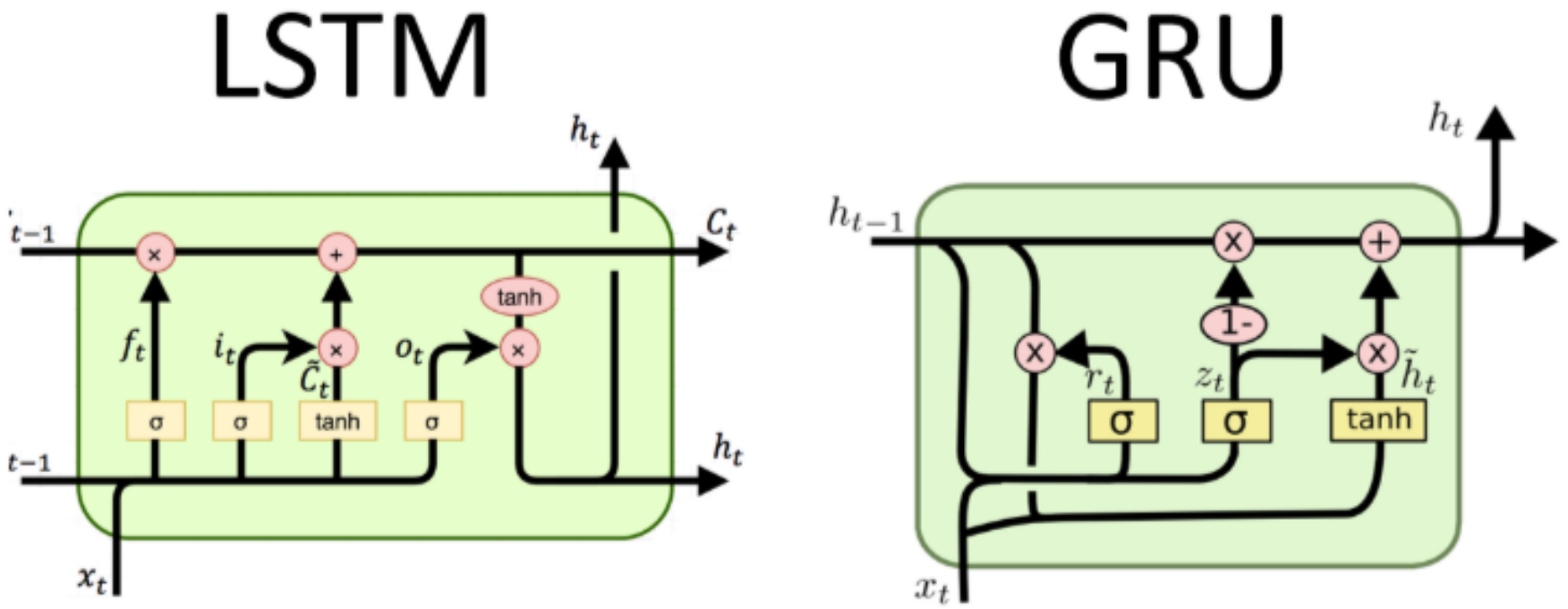
- Gated recurrent unit (GRU)
- Phased-LSTM

Gated recurrent units (GRUs)

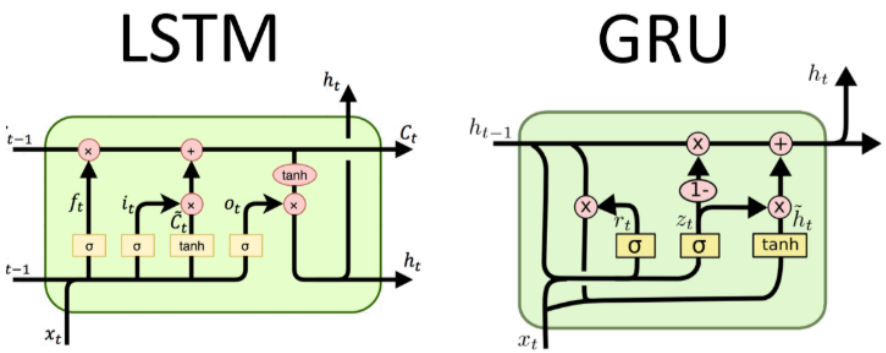
Gated recurrent units (GRUs) are newer (published in 2014) architecture and can be seen a simplification of LSTM (published in 1997).

- GRU is composed of only two gates: the reset gate and the update gate.
- It propagates a single state
- It's, then, the represented by fewer equations
  - → Less parameters to learn
  - → Faster to train

GRU vs LSTM



GRU vs LSTM



When do I prefer GRU over LSTM?

- It depends on the data
- It needs empirical evaluation

In this empirical study [2], GRU outperformed the LSTM on all tasks with the exception of language modelling.

[2] An Empirical Exploration of Recurrent Network Architectures. Jozefowic et al. (2015)

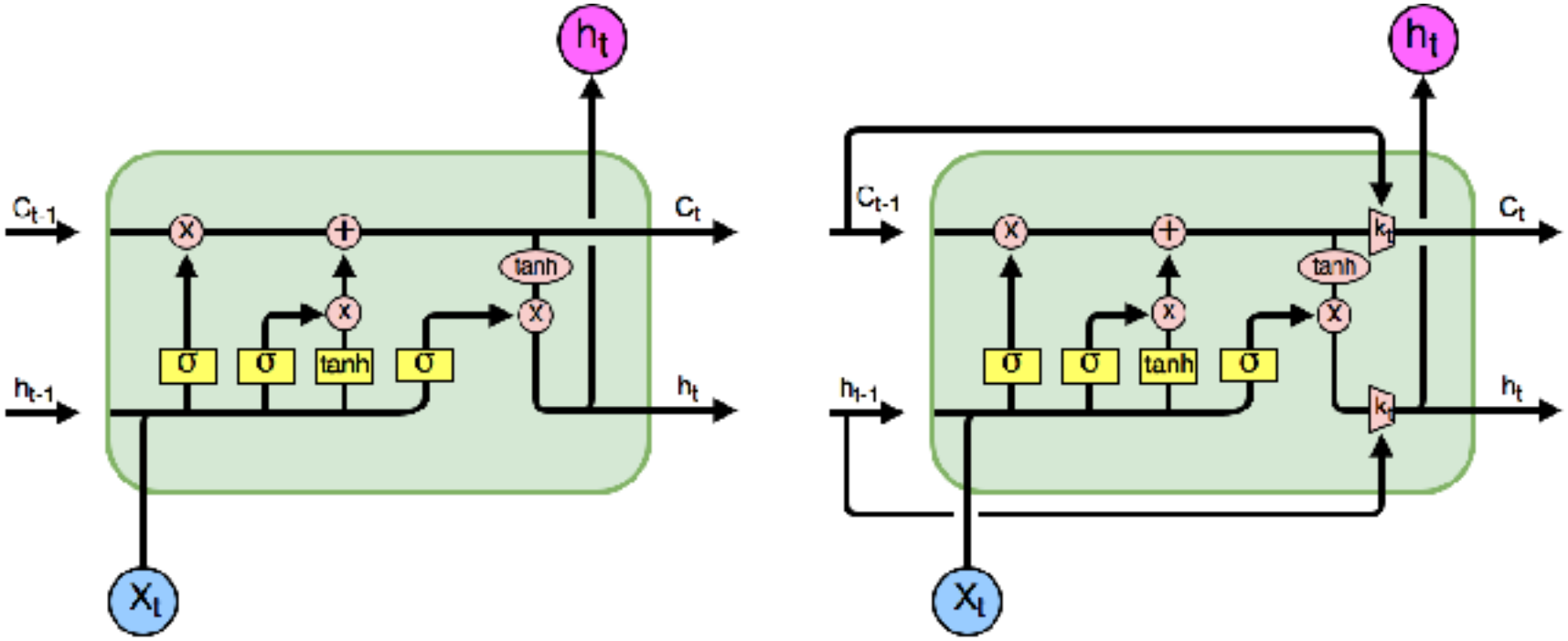
Phased-LSTM

Phased-LSTM [3] is designed to deal with input sampled at asynchronous times.

- Extends the LSTM unit by adding a new time gate.
- This gate is controlled by a parametrized oscillation with a frequency range that produces updates of the memory cell only during a small percentage of the cycle.
- The model naturally integrates inputs from sensors of arbitrary sampling rates.

[3] Neil, D., Pfeiffer, M., & Liu, S. C. (2016). Phased lstm: Accelerating recurrent network training for long or event-based sequences. *arXiv preprint arXiv:1610.09513*.

Phased-LSTM vs LSTM



(a) Standard LSTM

(b) Phased LSTM