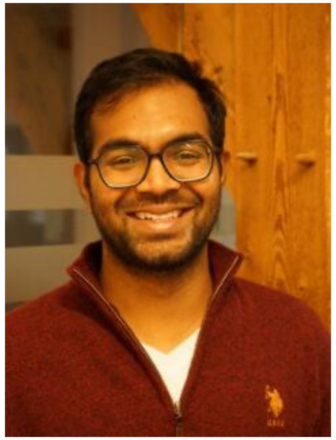
Seminar Advances in Deep Learning for Time Series (ADLTS)

Lecture 3: Deep Learning for Time Series

Dr. Dario Zanca
Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
17.10.2024

Team: FAU & PUCV

Dr. Dario Zanca (FAU)
dario.zanca@fau.de



Naga Venkata Sai Jitin Jami, M. Sc. (FAU)
jitin.jami@fau.de

Dr. Christtoffer Loeffler (PUCV)
christoffer.loffler@pucv.cl

&

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

Topics overview

Recorded Lectures

I. Introduction
II. The Tool Tracking dataset
III. DL for Time Series
IV. Time-aware models
V. XAI for Time Series - part 1
VI. Active Learning for Time Series - part 1
VII. Semi-supervised Learning
VIII. Domain-shifts, Ethics, and Bias
IX. XAI for Time Series - part 2
X. Active Learning for Time Series - part 2

Topics overview

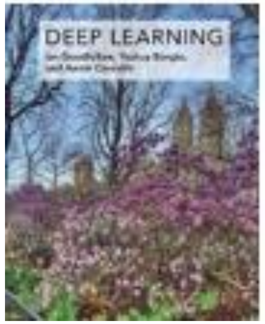
Recorded Lectures

I. Introduction
II. The Tool Tracking dataset
III. DL for Time Series
IV. Time-aware models
V. XAI for Time Series - part 1
VI. Active Learning for Time Series - part 1
VII. Semi-supervised Learning
VIII. Domain-shifts, Ethics, and Bias
IX. XAI for Time Series - part 2
X. Active Learning for Time Series - part 2

References

Deep Learning

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



Deep Learning: Foundations and Concepts

by C. Bishop, H. Bishop (2024)

Lecture outline

1. Introduction to Deep Learning
2. Convolutional Neural Networks (CNNs)
3. Recurrent models (RNNs and LSTMs)
4. Transformers

ADLTS \ DL for TS \
Introduction to Deep Learning

Why Deep Learning?

Previous method needed **handcrafted features**:

- MFCCs (speech processing) (1)
- I-Vector (speech processing) (2)
- Sift (scene alignment, videos) (3) → Needs expert knowledge about domain

(1) Mermelstein, P. (1976). Distance measures for speech recognition, psychological and instrumental. *Pattern recognition and artificial intelligence*, *116*, 374-388.

(2) V. Gupta, P. Kenny, P. Ouellet and T. Stafylakis, "I-vector-based speaker adaptation of deep neural networks for French broadcast audio transcription," *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 6334-6338, doi: 10.1109/ICASSP.2014.6854823.

(3) Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), 91-110.

Why Deep Learning?

What if we can not define generally applicable features?

- High dimensional data
- Hard to come up with generally applicable features

→ With deep learning, we can find features in a data driven way

→ Can help capture complex non-linear relationships

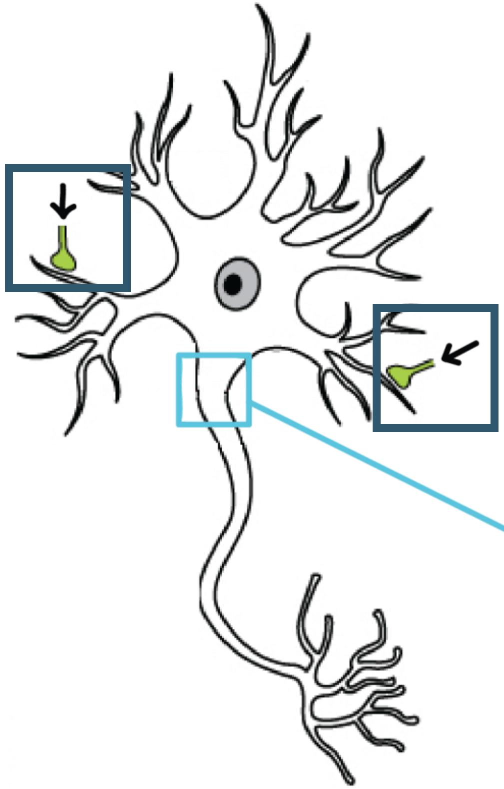
The Human Brain

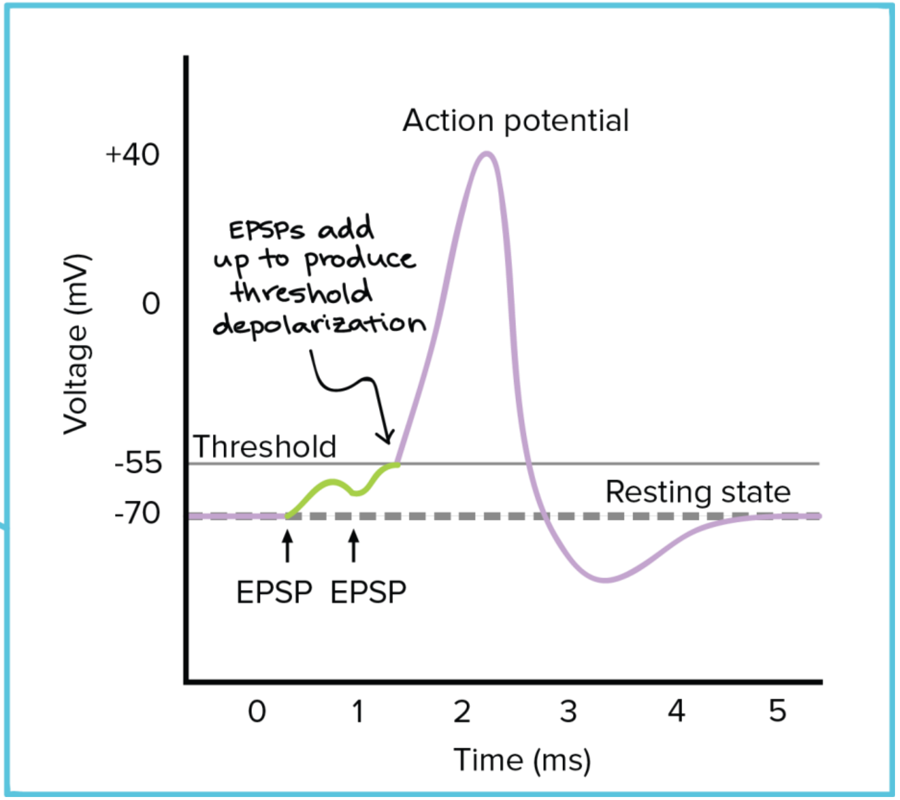
The human brain is our reference for an intelligent agent.

- It contains different areas specialized for some tasks (e.g., the **visual cortex**)
- It consists of neurons as the fundamental unit of “**computation**”

Image from https://ucresearch.tumblr.com/post/138868208574

The Brain's Neuron





EPSP = Excitatory postsynaptic potential

- Excitatory stimuli reach the neuron
- Threshold is reached
- Neuron fires and triggers action potential

The Perceptron – Computational Model of a Neuron

Let’s build the computational model step by step:

1. Show the input and output of our neuron (which depends on the data and task)

Input x_1

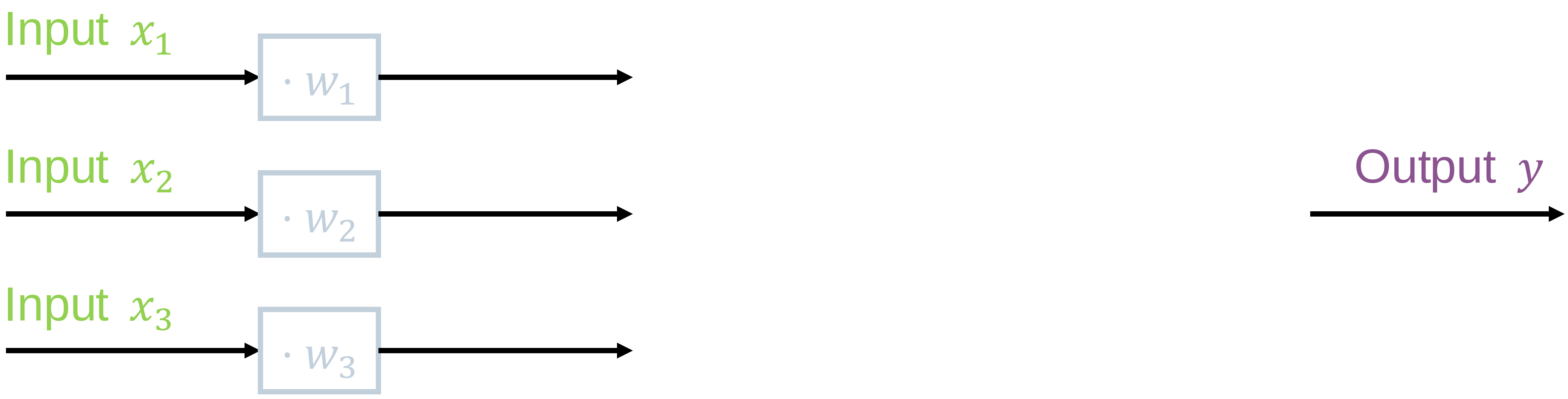
Input x_2

Output y

Input x_3

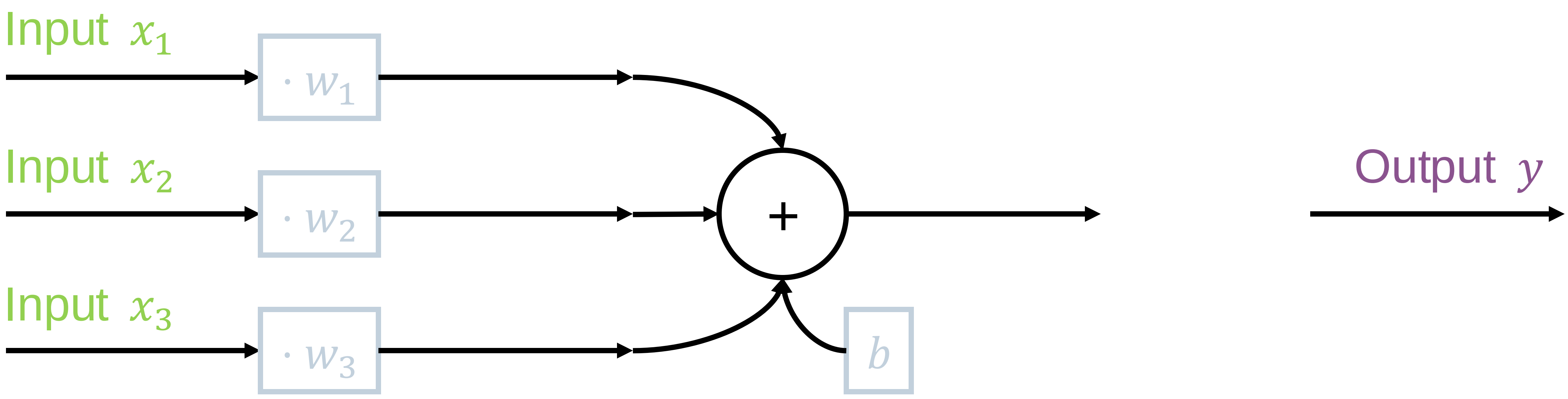
The Perceptron – Computational Model of a Neuron

2. Weights can “select” or “deselect” input channels (not all are relevant for subsequent computations)



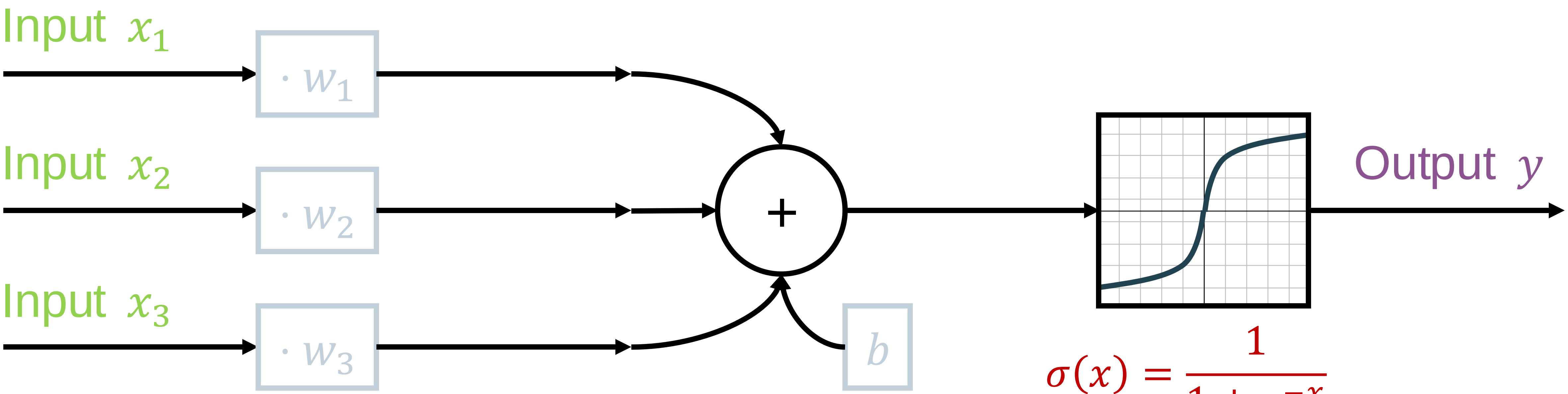
The Perceptron – Computational Model of a Neuron

3. We add up all the excitatory signals and the **resting potential (or bias)** to determine the current potential.



The Perceptron – Computational Model of a Neuron

4. A **threshold function (or activation function) σ** is applied to determine whether an action potential has to be sent in the output



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Activation Functions

The goal of an **activation function** is to activate (or not) neuron.

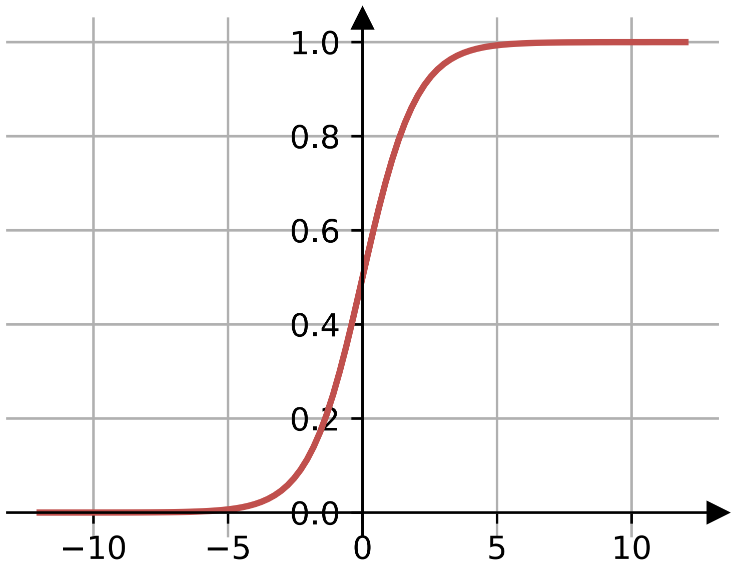
It brings **non-linearity** to a neural network:

→ The output is **not** a linear function of the input

Different activation functions can be considered, taking into account their:

- Output range
- Mean
- Gradient

Sigmoid

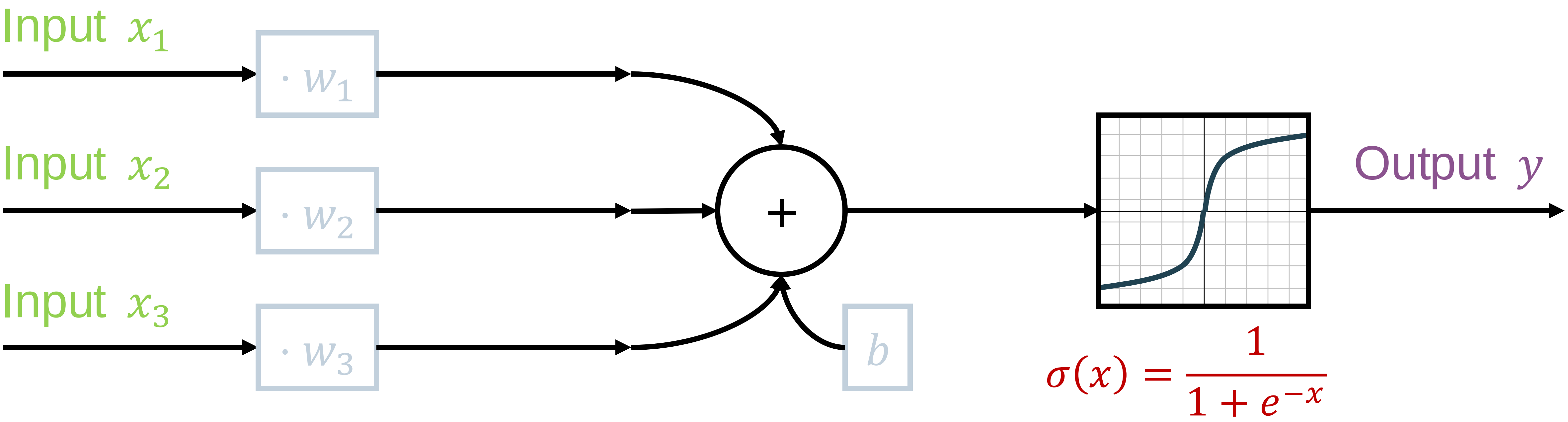


$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The Perceptron – Computational Model of a Neuron

5. We can write the perceptron mathematical model to map inputs x_1, x_2, x_3 to the output y_1 using channel **weights w_1, w_2, w_3, b**:

$$y \quad = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b) = \sigma(\sum_i w_i \cdot x_i + b)$$

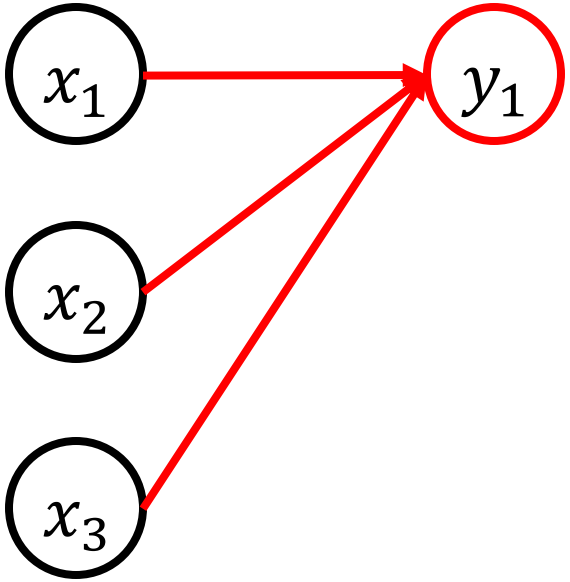


Single Layer Perceptron

We can combine multiple perceptrons to create a **layer.**

One perceptron has the following equation:

$$y_1 = \sigma(w_{11} \cdot x_1 + w_{12} \cdot x_2 + w_{13} \cdot x_3 + b)$$

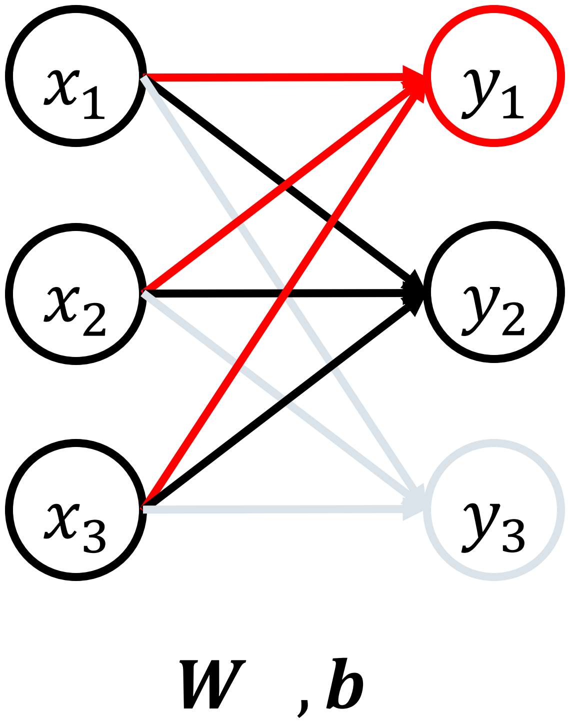


Single Layer Perceptron

We can combine multiple perceptrons to create a **layer.**

We can combine **three perceptrons** into one layer:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \underbrace{\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix}}_{W} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \underbrace{\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}}_{b}$$



Or in a more simplified form: $y = \sigma(W \cdot x + b)$

Multi Layer Perceptron (MLP)

We can chain **multiple layers**

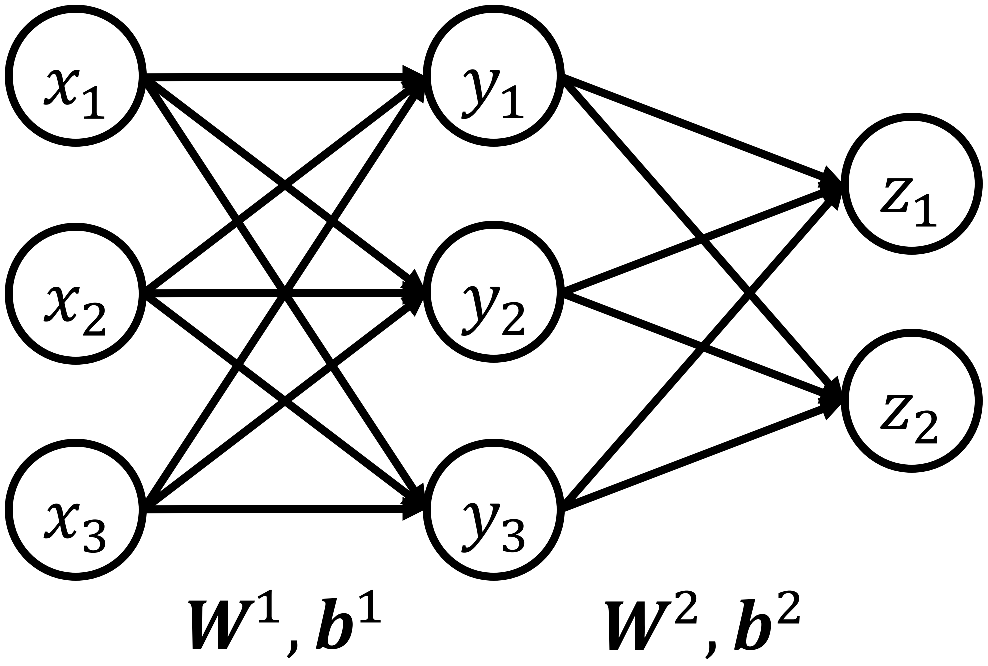
$$y = \sigma(W^1 \cdot x + b^1)$$

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

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

→ Each layer has its own set of parameters (weights $\boldsymbol{W}^i$ and bias $\boldsymbol{b}^i$)

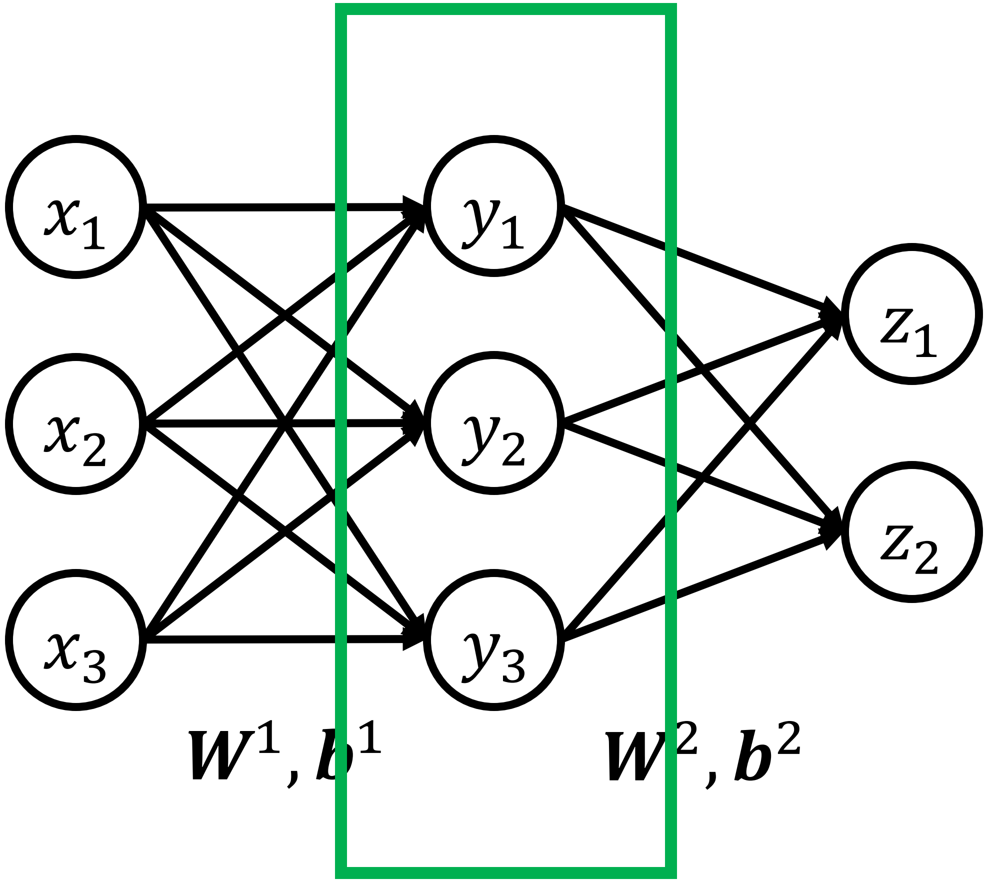
→ The activation function ensures that all layers do not collapse into one: it introduces **non-linearity**



Multilayer perceptron (MLP)

We call "**hidden layer**" any layer in between the input and the output layers.

For example, this neural network (image on the right) is a Multi-Layer-Perceptron (MLP) with a **single hidden layer** (highlighted by the green box).



→ The underlying computation is described by:

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

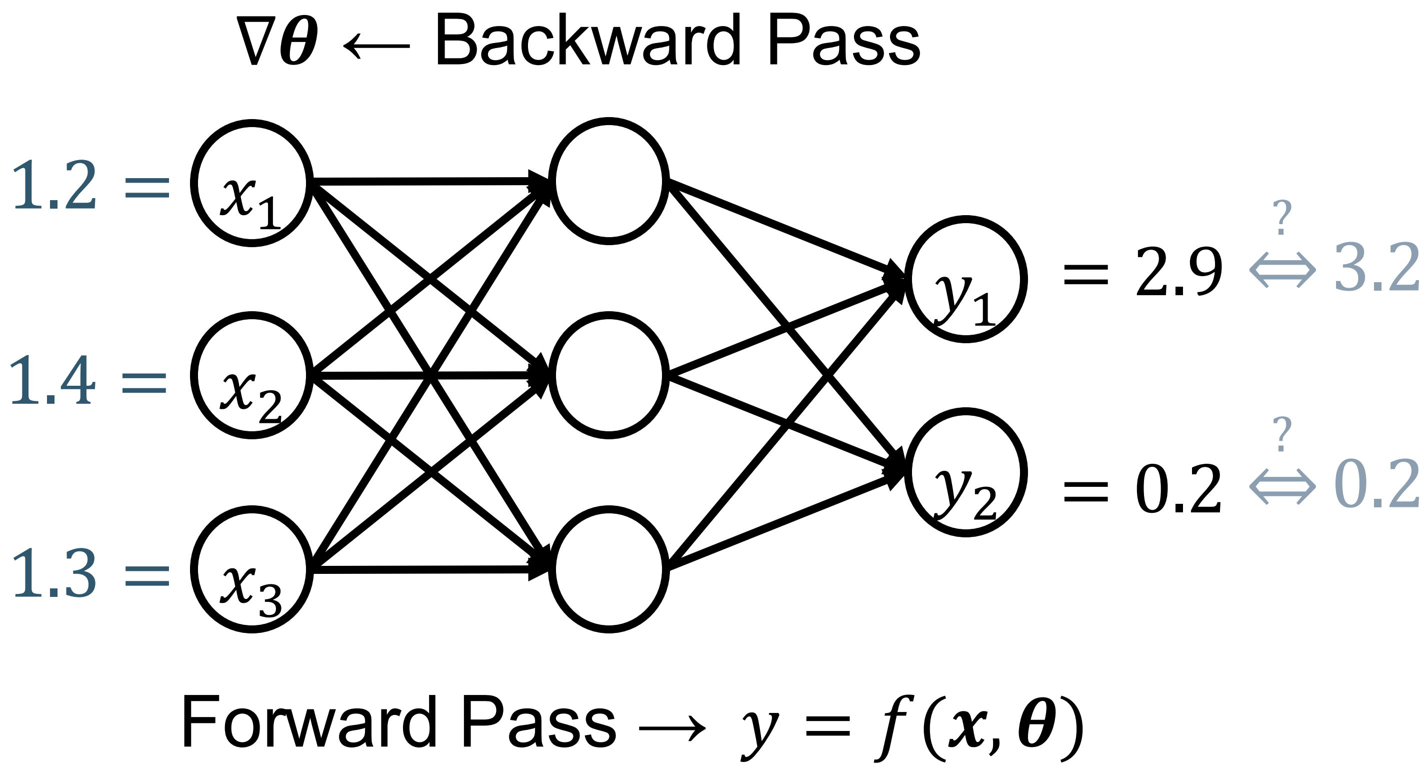
Also referred to feedforward networks (no feedback connection)

How are Model Parameters Learned?

1) **Forward propagation:** This phase refers to the computation of the output using input and parameters.

2) **Loss calculation:** The output and expected output are then compare using a loss function.

3) **Backward propagation**: During this phase, the model computes the gradients of the loss with respect to each parameter ($\boldsymbol{\theta}$).



Backpropagation Algorithm: Overview

1. Calculate the forward pass and **store** results for $\hat{y}$, a_j^k, and z_j^k.

2. Calculate the backward pass and **store** results for $\frac{\partial L}{\partial w_{ij}}$, proceeding from the last layer:

   a) **Evaluate** the error terms for the last layer δ_k

   b) **Backpropagate** the error term for the computation of δ_j

   c) **Iterate** to all previous layers

3. Combine the individual gradients (average)

4. Update the weights according to the **learning rate** λ

ADLTS \ DL for TS \
Convolutional Neural Networks (CNNs)

Motivation: The Human Visual Cortex

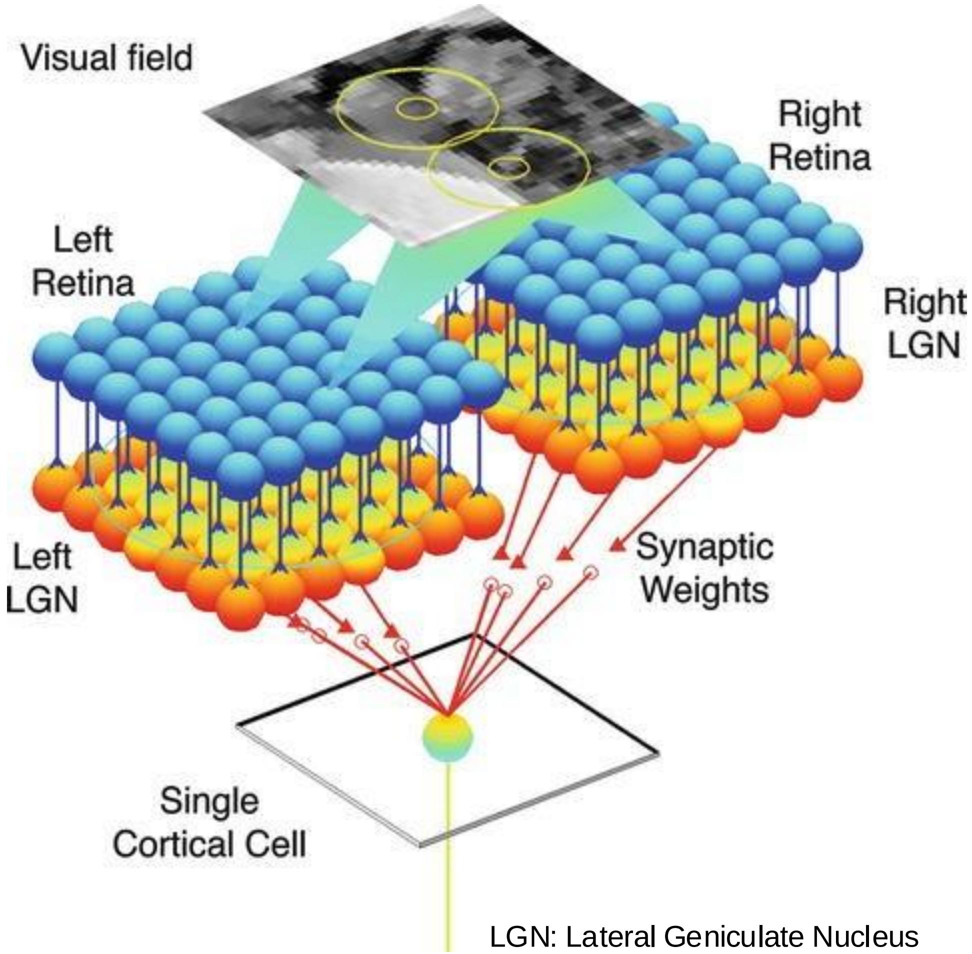
CNNs were inspired by a specialized brain area: the **visual cortex**.

1) The visual cortex processes visual information collected by the retinae in a **hierarchical** manner.

2) The **receptive field** of cortical neurons increases the later they are in this hierarchy.

   a) Small receptive fields are stimulated by high spatial frequencies - or fine details

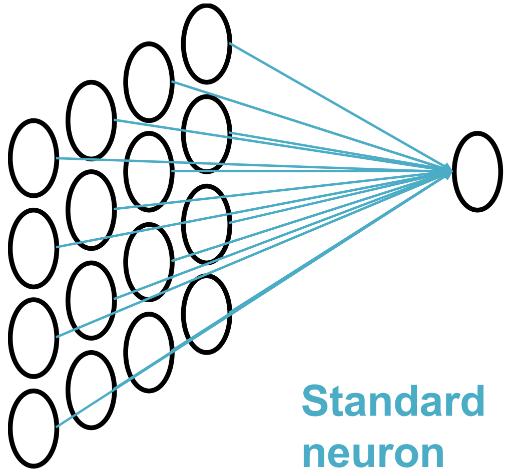
   b) Large receptive fields are stimulated by low spatial frequencies - or coarse details



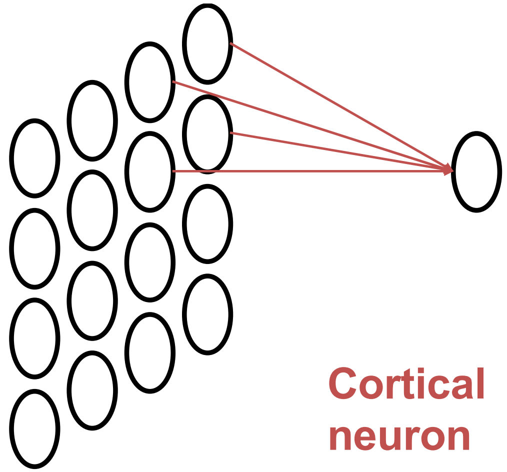
LGN: Lateral Geniculate Nucleus

Modeling Receptive Fields

- In standard **multilayer perceptrons** (MLP), layers are **fully-connected,** i.e. all units from one layer are connected to all units from the previous layer.



- Cortical neurons have small **receptive fields**: units from one layer have **sparse** and **localized** connection with units from the previous layer.



→ This is modeled by discrete convolutional operations.

Convolutional Operation

The **continuous convolution** is a operation on two functions f and g noted $f * g$ that produces a third function.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

It is the integral of the product of the two functions after one is reflected about the y-axis and shifted.

In deep learning, a **discrete** version of this operation is used.

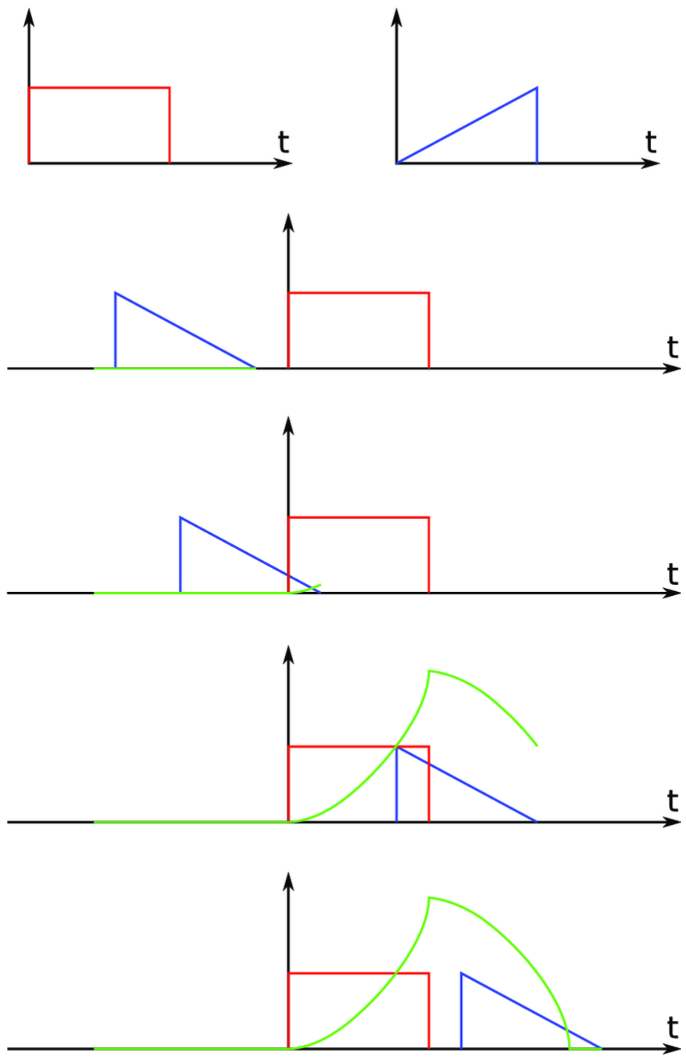


Image from Pihlajamäki, Tapani. Multi-resolution Short-time Fourier Transform Implementation of Directional Audio Coding.

Discrete Convolution

The (discrete) **convolution** is a mathematical operation on two functions. On our case, the **input** and a smaller **filter** produce a third function (the feature map).

$$(input * filter)[n] = \sum_{m=-\infty}^{+\infty} input[m] \cdot filter[n-m]$$

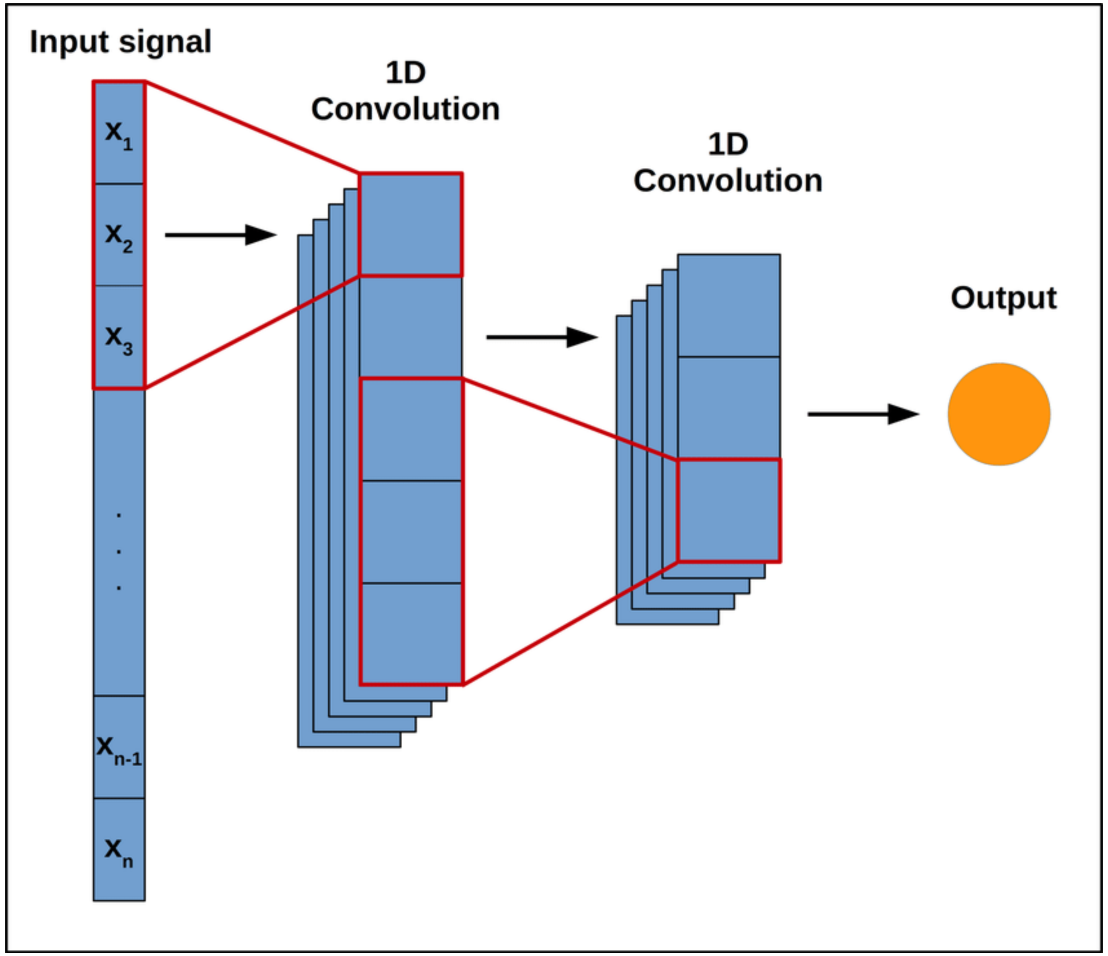
→ The sum is evaluated for all values of the shift.

→ The term convolution is used both to indicate the result function and process of computing it.

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.

1D Convolutional Operation

Like the 2D operation, the output computation only depends on a **subset** of the time series:

$$y_1 = w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$\begin{array}{|c|c|c|c|c|}\hline x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline\end{array} * \begin{array}{|c|c|c|}\hline w_1 & w_2 & w_3 \\ \hline\end{array} = \begin{array}{|c|c|c|}\hline y_1 & & \\ \hline\end{array}$$

1D Convolutional Operation

Like the 2D operation, the output computation only depends on a **subset** of the time series:

$$y_2 = w_1 x_2 + w_2 x_3 + w_3 x_4$$

$$\begin{array}{|c|c|c|c|c|} \hline x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline w_1 & w_2 & w_3 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline y_1 & y_2 & \\ \hline \end{array}$$

1D Convolutional Operation

Like the 2D operation, the output computation only depends on a **subset** of the time series:

$$y_3 = w_1 x_3 + w_2 x_4 + w_3 x_5$$

$$\begin{array}{|c|c|c|c|c|}\hline x_1 & x_2 & x_3 & x_4 & x_5 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline w_1 & w_2 & w_3 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline y_1 & y_2 & y_3 \\ \hline \end{array}$$

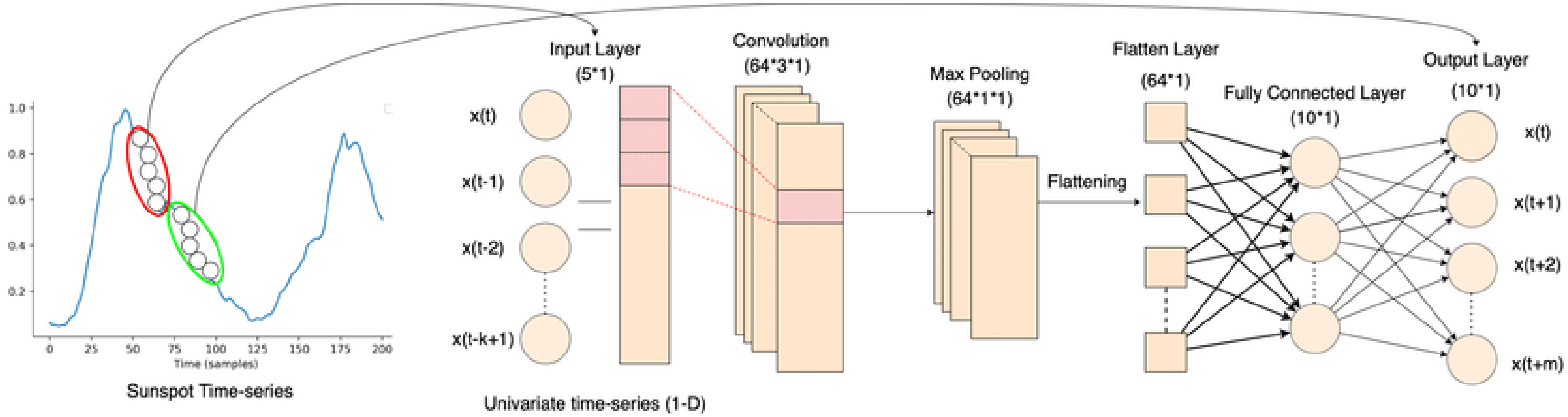
CNN Properties and Time Series

CNN properties make them very appropriate for **computer vision**. Those can also be useful for **time series**:

- Ability to extract **local patterns**: 1D CNNs can extract local patterns that occur in time series data (e.g. short term trends in stock prices, weather events)
- **Translation Equivariance**: 1D CNNs can identify those patterns regardless of their position, which is useful for tasks such as anomaly detection
- **Dimensionality reduction** of pooling layers: This can be useful when datasets are high dimensional.
- **Hierarchical** feature learning: Time series are composed of patterns at different time **scales**. (e.g. seasonal weather patterns vs sudden weather event)

CNN for Time Series Forecasting

Example of CNN for univariate time series **forecasting** [1].



Chandra, Rohitash & Goyal, Shaurya & Gupta, Rishabh. (2021). Evaluation of deep learning models for multi-step ahead time series prediction. 10.48550/arXiv.2103.14250.

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.

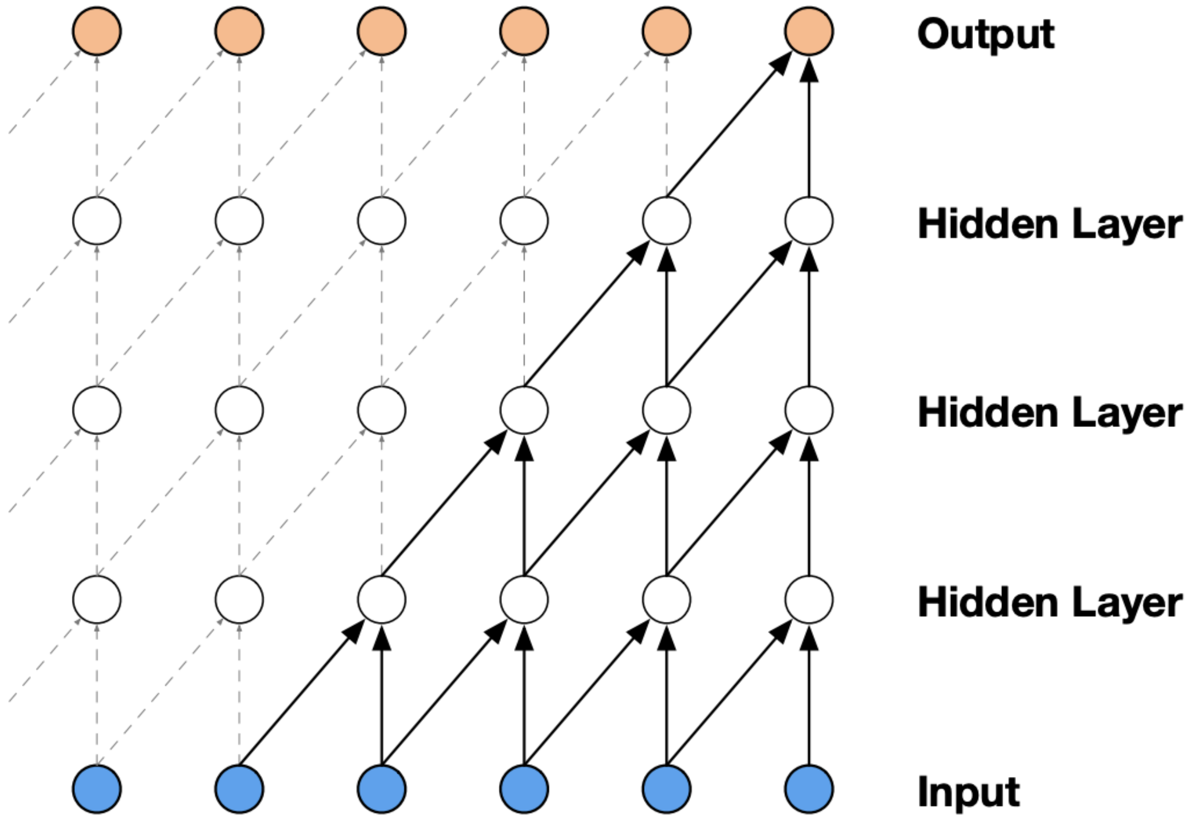


Image from Oord, A. V. D. (2016). WaveNet: A Generative Model for Raw Audio..

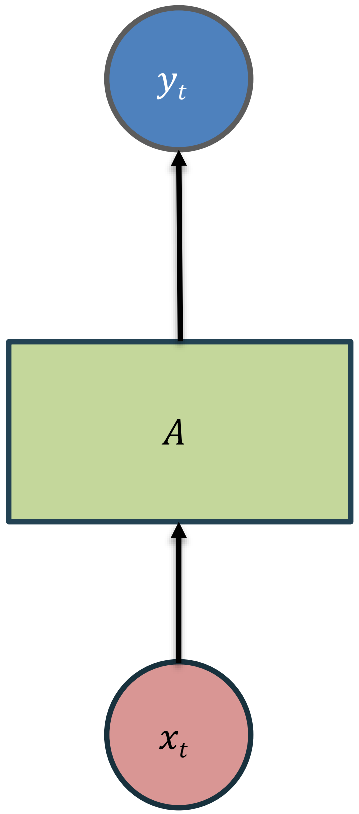
ADLTS \ DL for TS \
Recurrent models (RNNs and LSTMs)

Limitations of NN for time series data

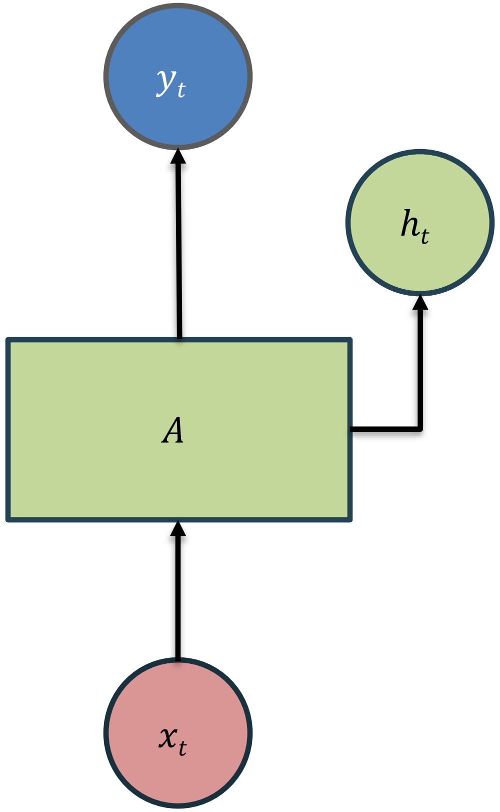
Feed-forward and Convolutional neural networks present some **disadvantages**, when applied to sequential data:

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

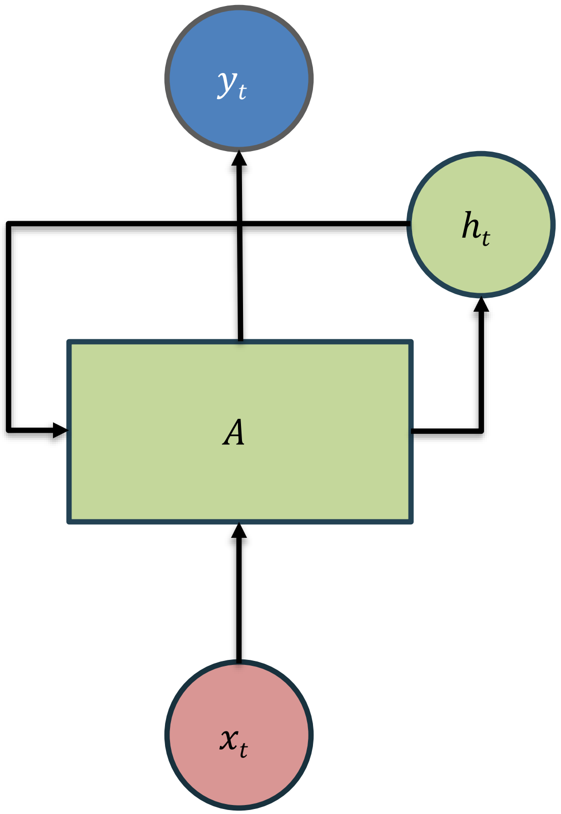
Recurrent Neural Network (RNN)



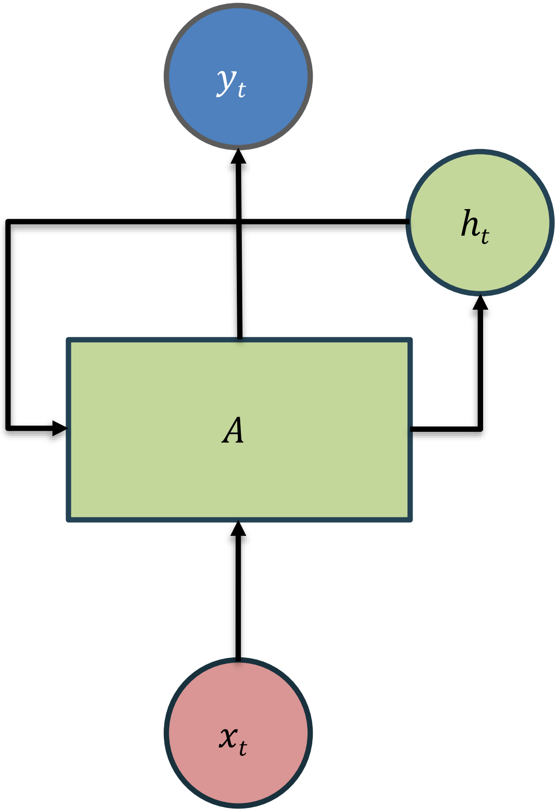
Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)



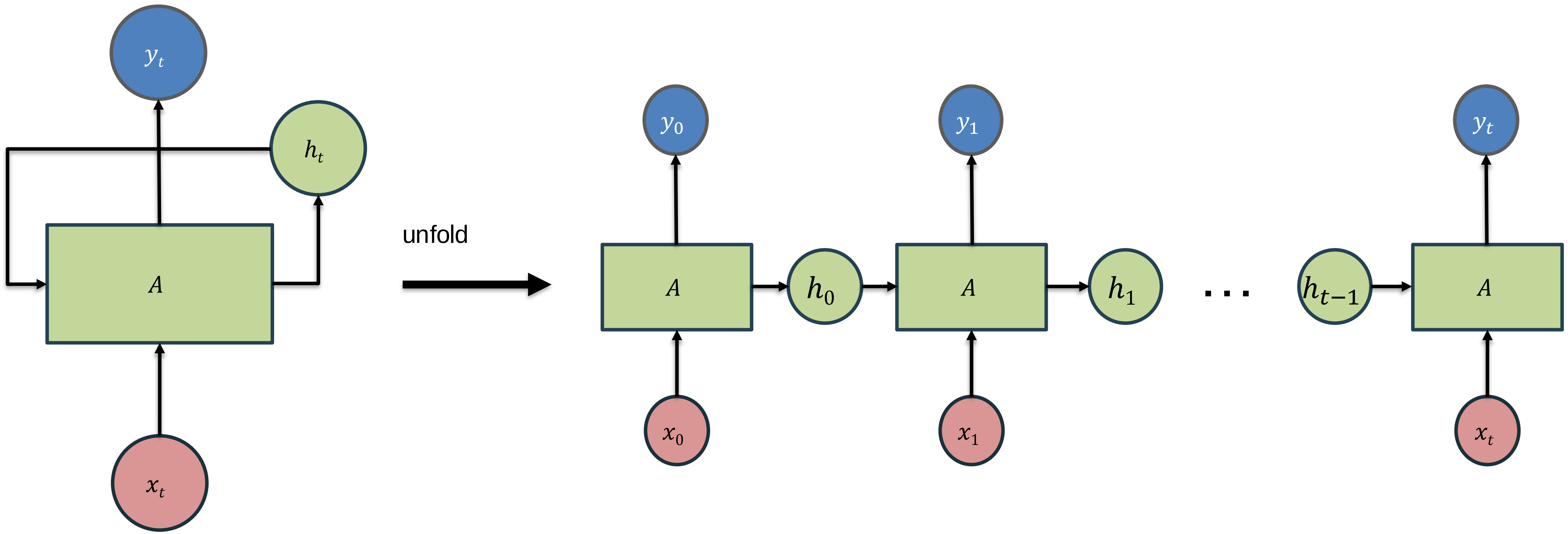
A **recurrent neural network (RNN)** is a neural network that contains feed-back connections.

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

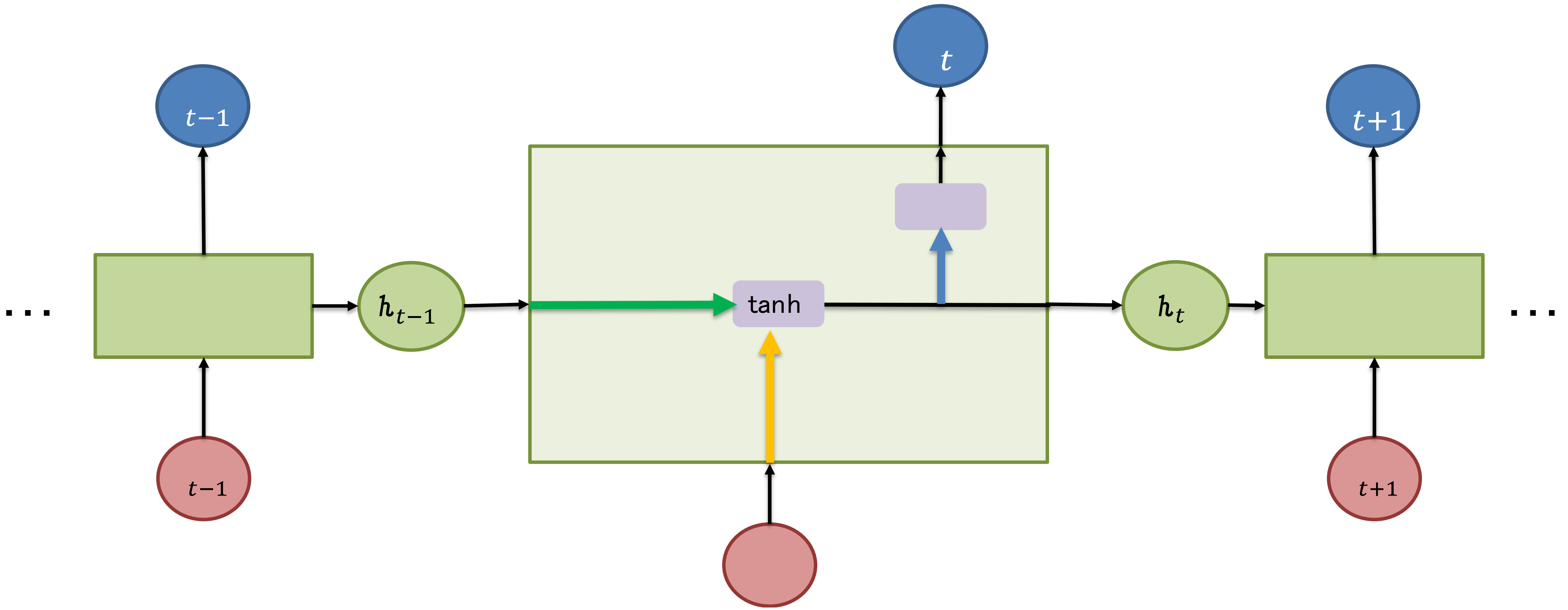
An RNN is composed by:

x_t: input at time t
A: neural network
h_t: hidden state
y_t: output at time t

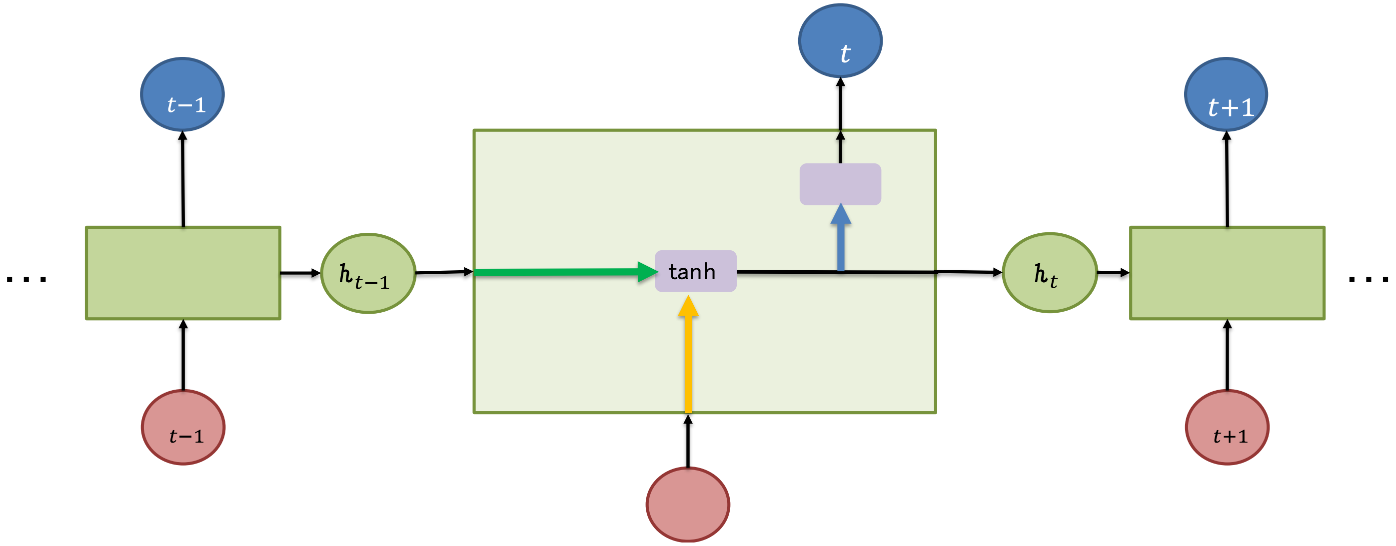
RNN Unfolding



Mathematical formulation



Mathematical formulation

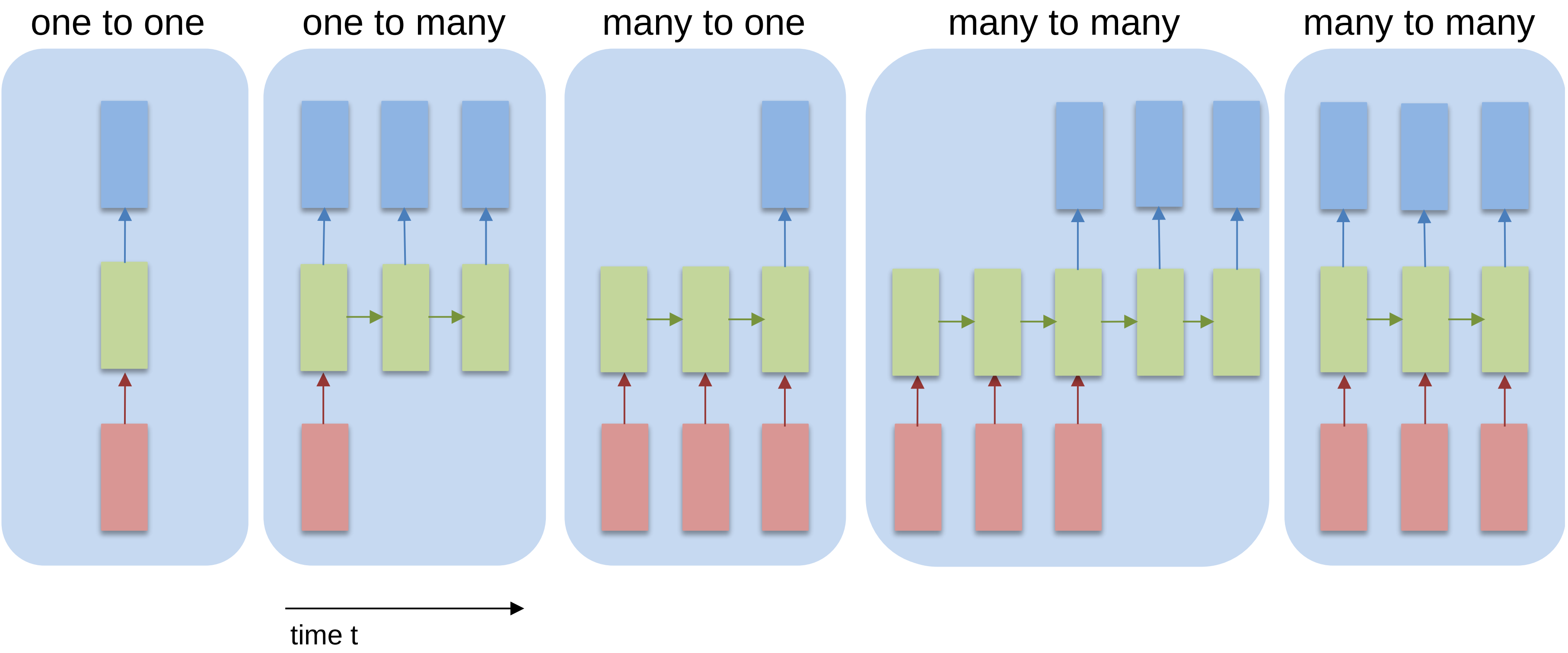


The behaviour of the RNN can be described as a **dynamical system** by the pair of non-linear matrix equations:

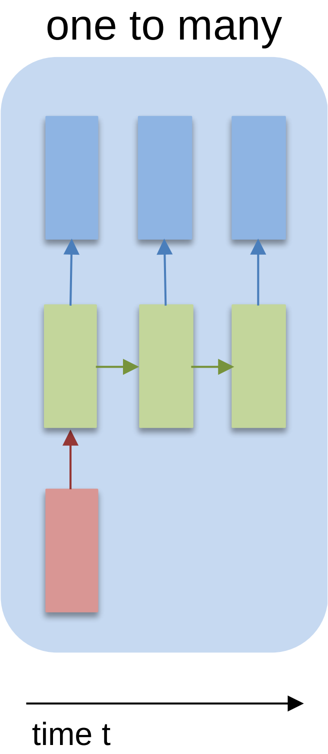
$$h_t = \tanh(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t)$$
$$y_t = \sigma(\boldsymbol{W_{hy}} h_t)$$

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

RNNs architecture

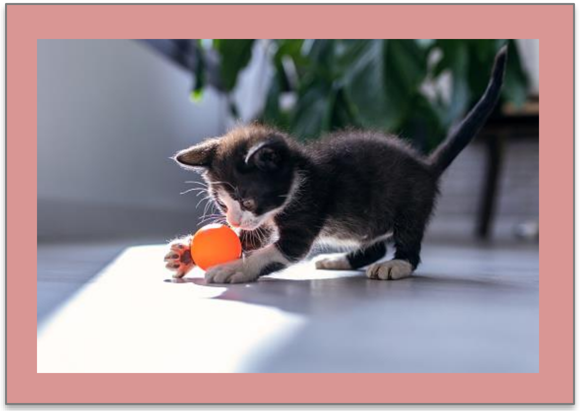


Example: one to many



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

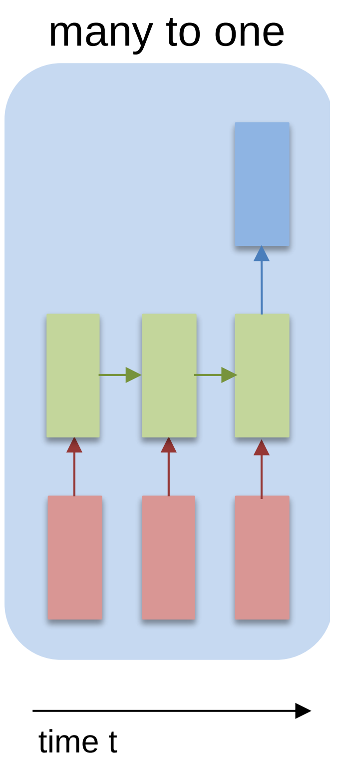
Input:



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.**

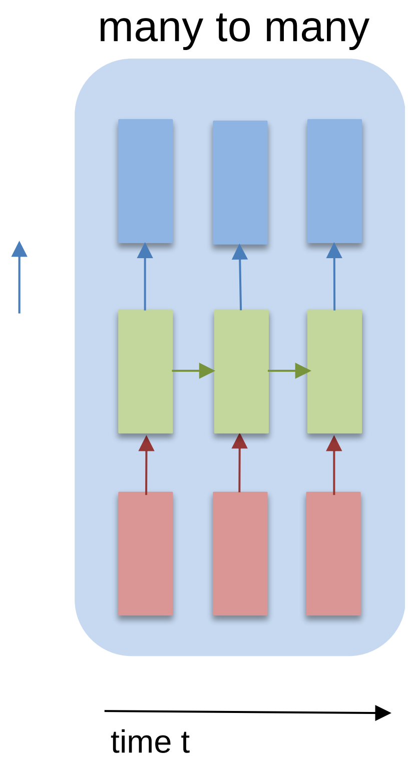
Input:

Horrible | service | the | room | was | dirty

Output:

("negative")

Example: many to many



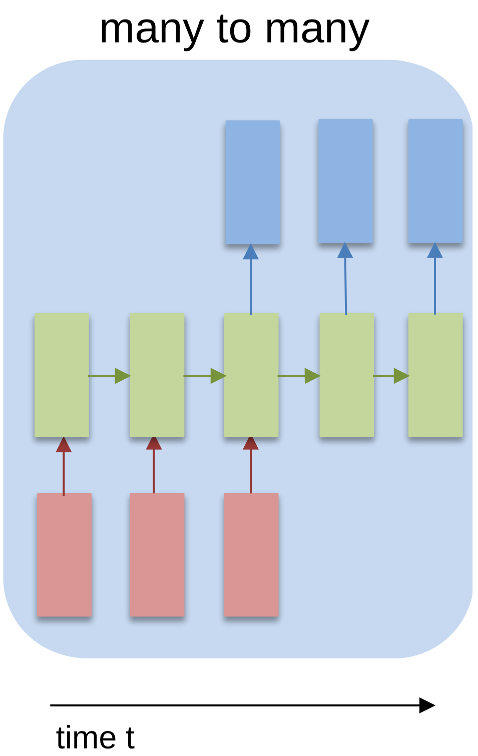
A typical example of a many to many problem is that of name entity recognition.

Input:

Harry	Potter	and	Hermione	invented	a	new	spell
1	1	0	1	0	0	0	0

Output:

Example: many to many



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

Input:

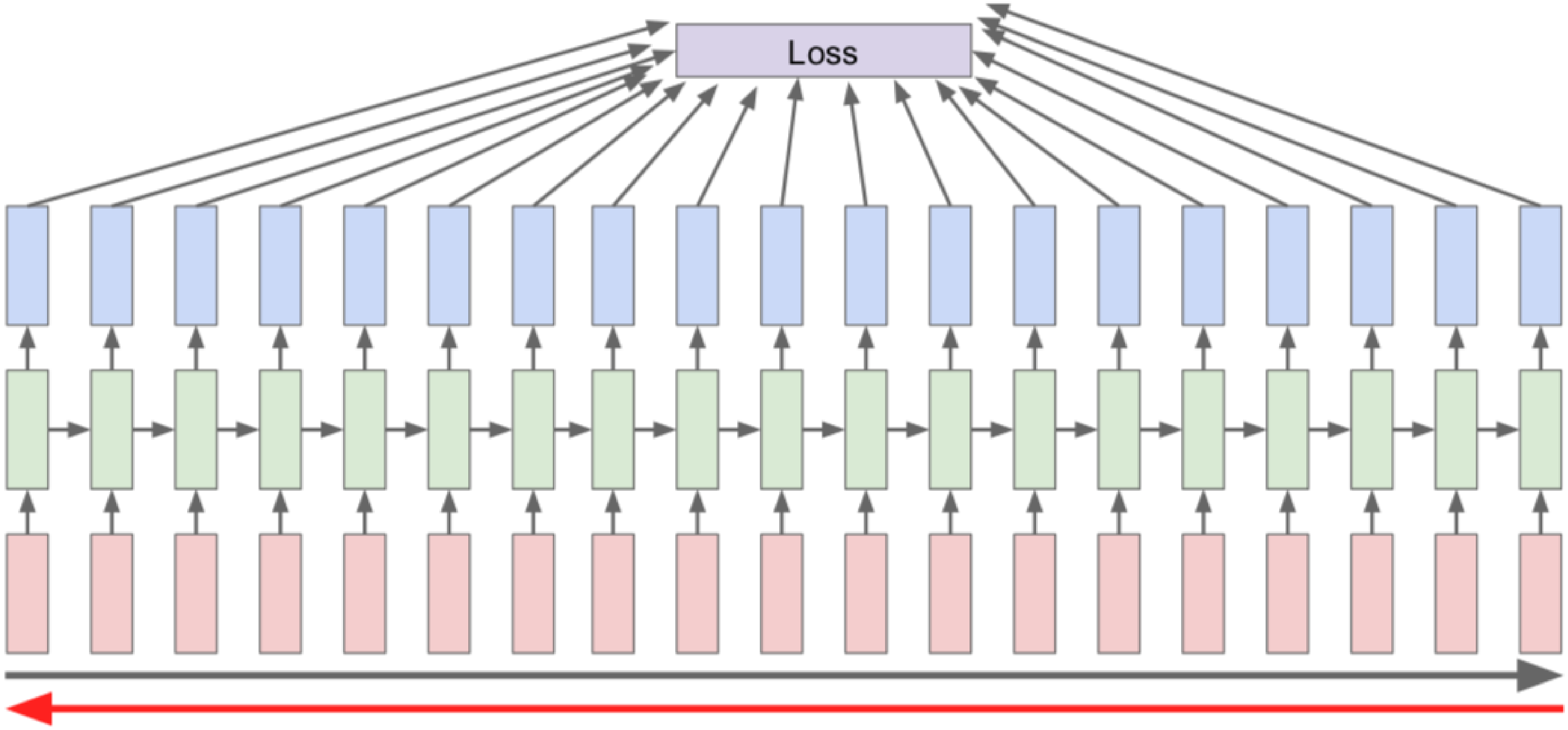
Horrible | service | the | room | was | dirty

Output:

Un | servizio | orribile | la | camera | era | sporca

BPTT: Limitations

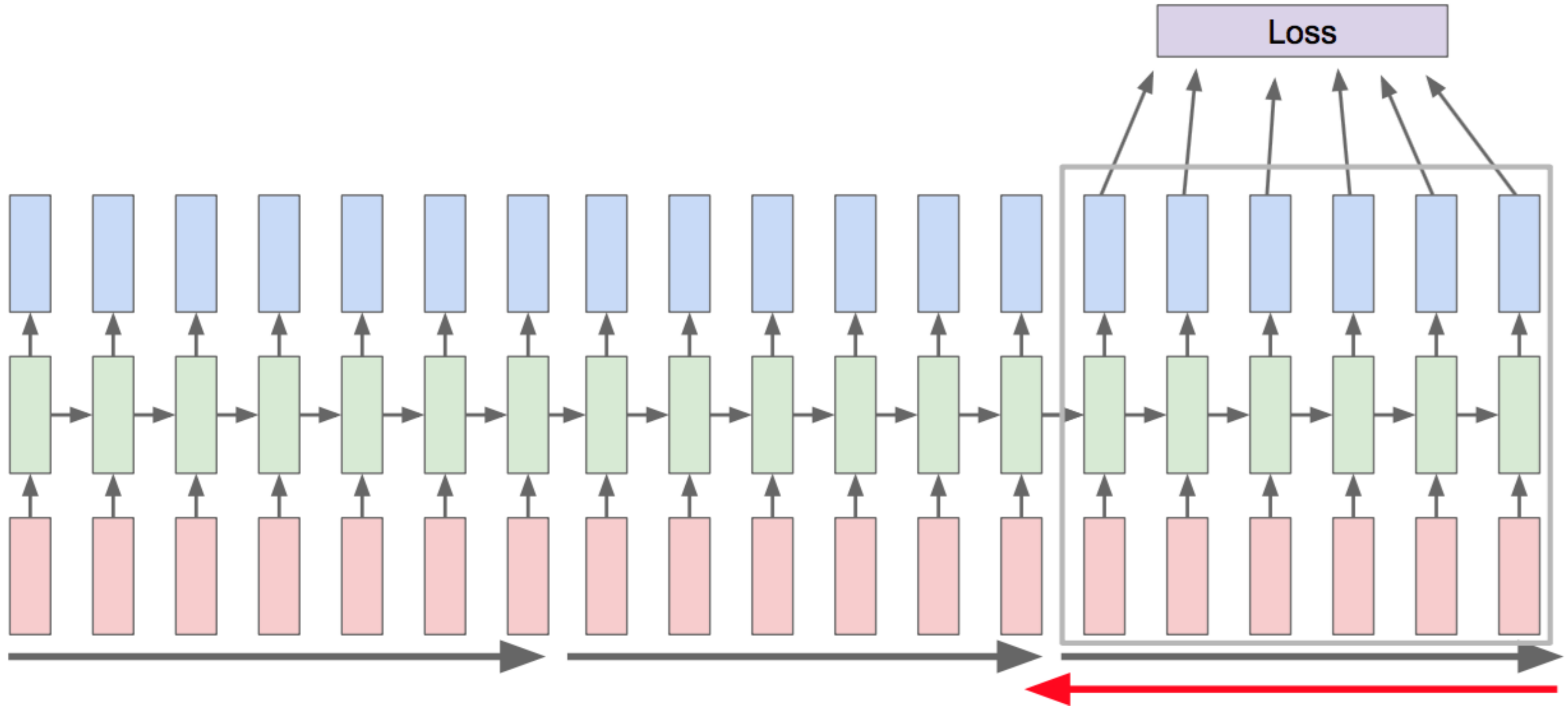
BPTT can be computationally very expensive as a lot of partial derivatives have to be computed, depending on the complexity of the network.



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

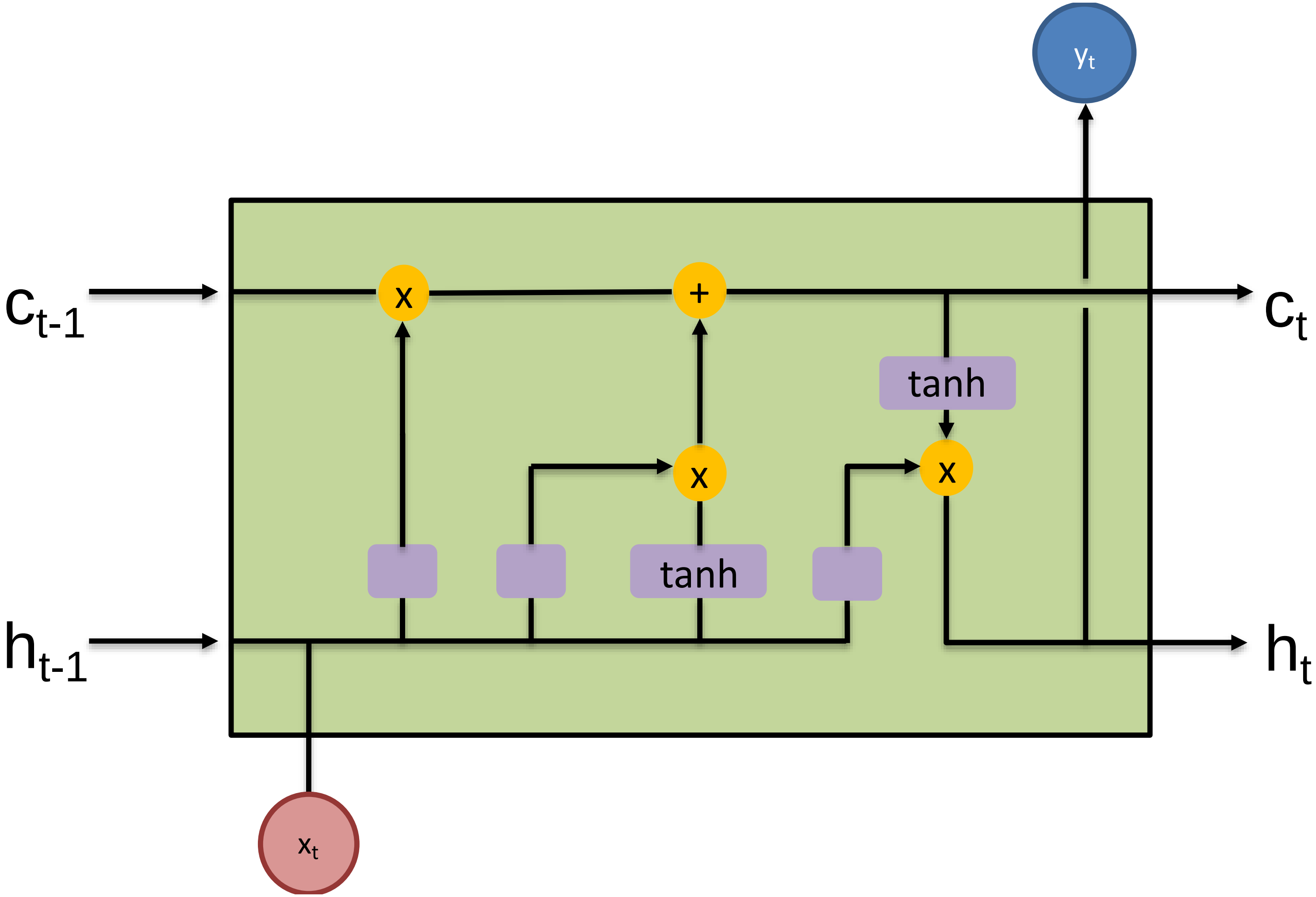
Truncated Backpropagation Through Time (Trunc-BPTT)

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

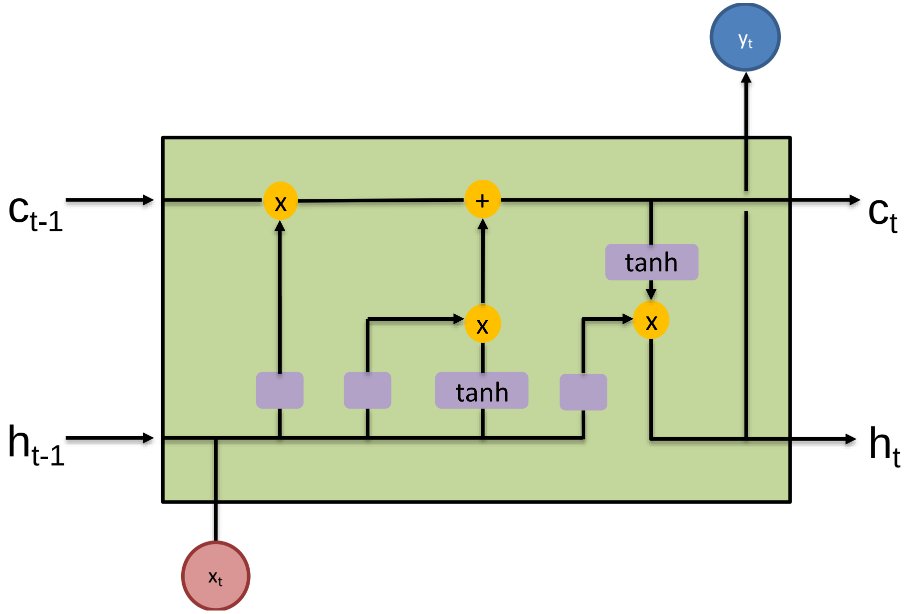


Adapted from https://tjmachinelearning.com/lectures/guest/rnnadv/rnnadv.html

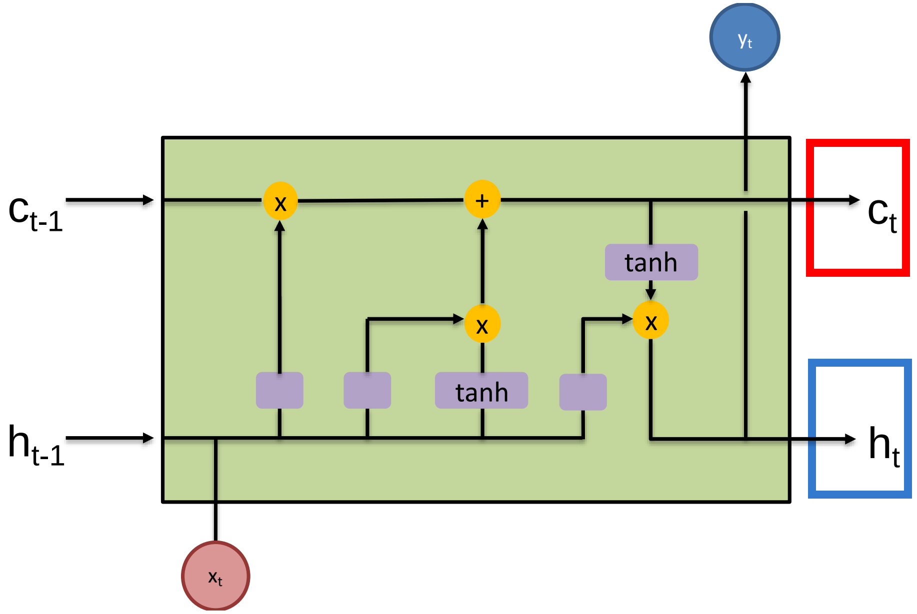
Long-short term memory networks (LSTMs)



Long-short term memory networks (LSTMs)

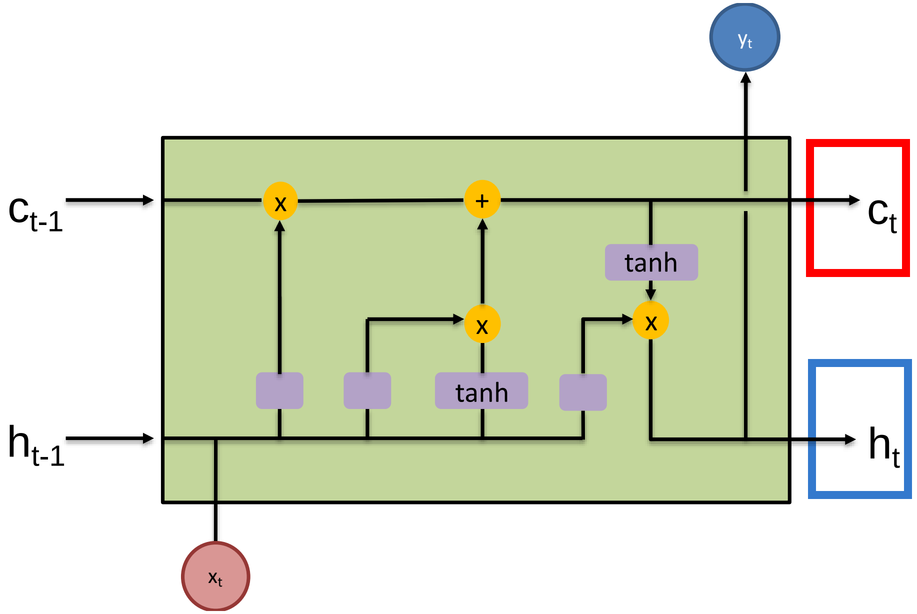


Long-short term memory networks (LSTMs)



Observations:

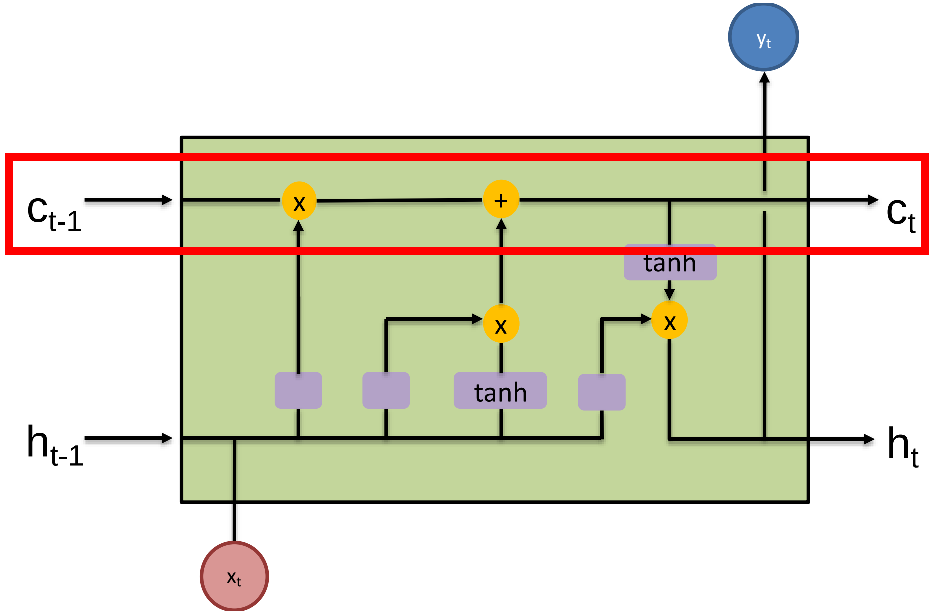
- Compared to RNNs, additionally to the **hidden state** h_t, we have another state, c_t, said the **cell state**.



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.

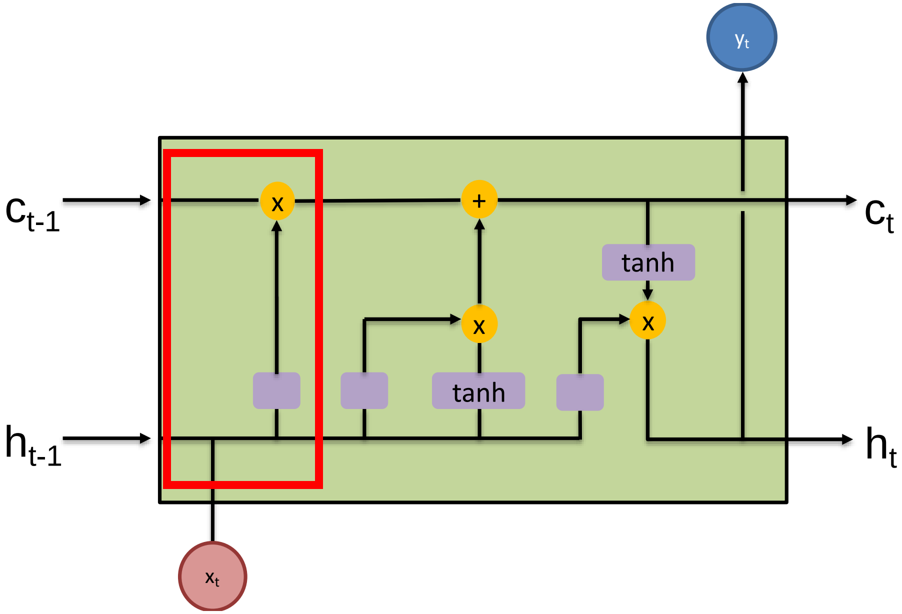
Long-short term memory networks (LSTMs)



The cell state has:

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

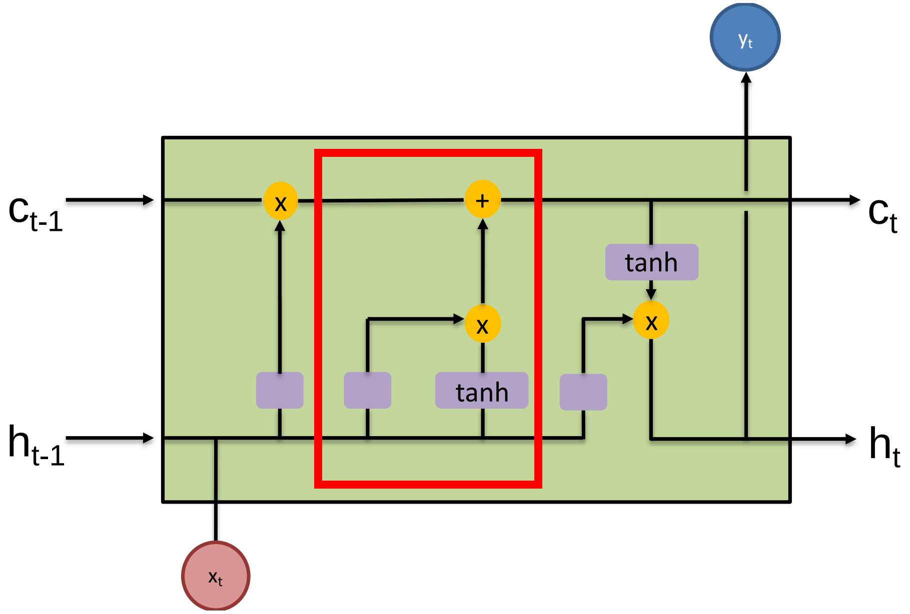
Long-short term memory networks (LSTMs)



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

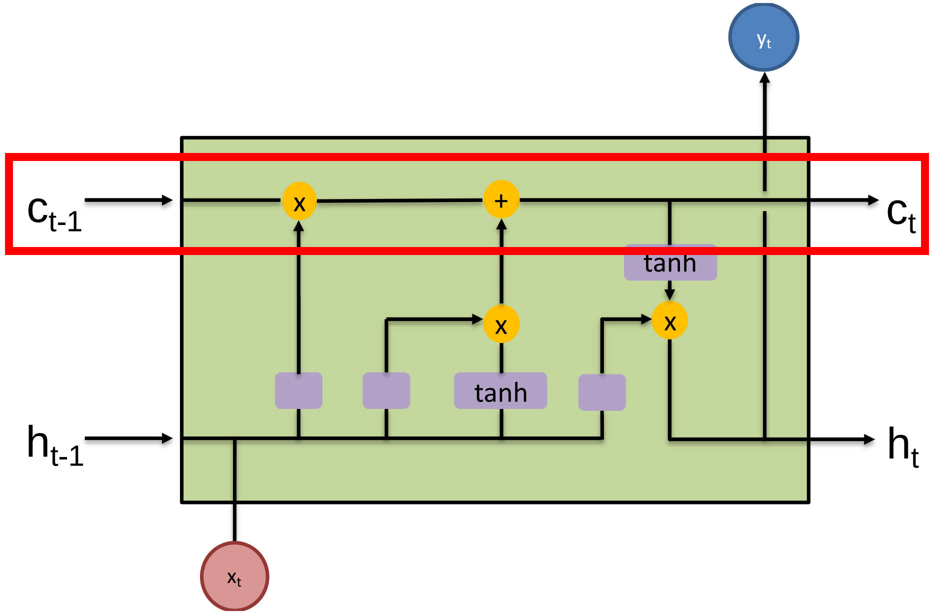
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Long-short term memory networks (LSTMs)



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

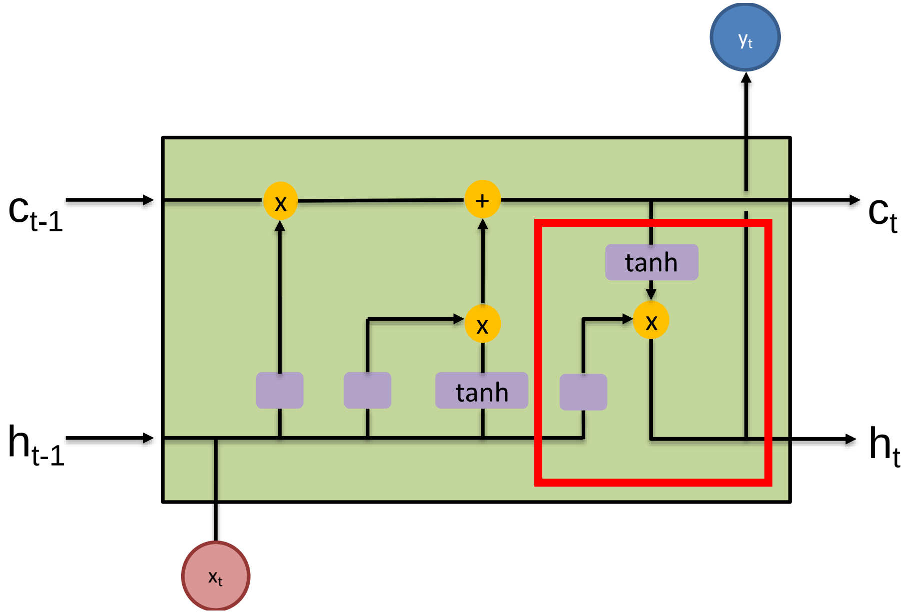
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\mathrm{g}_\mathrm{t} = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$



The values can be combined to update the cell state

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

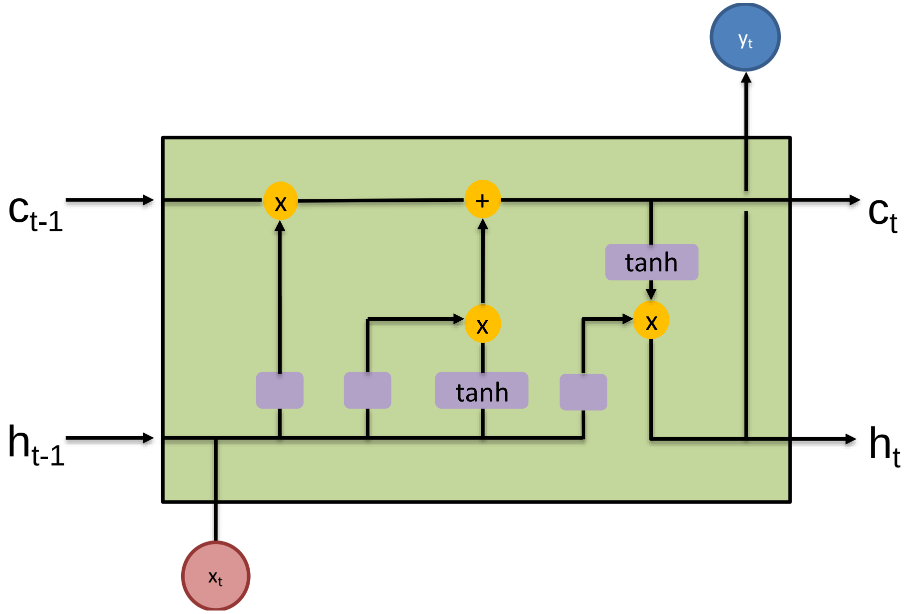
Long-short term memory networks (LSTMs)



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:

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

ADLTS \ DL for TS \
Transformers

Attention Is All You Need [1]

The **Transformer** [1] architecture was published in 2017. Models built on this architecture have become state-of-the-art in many domains, starting with **Natural Language Processing**.

- Completely built on the **self-attention** mechanism
- Does **not** use sequence any recurrent architecture:
  - More efficient: input sequences can be processed in **parallel**
  - Has no inherent understanding of sequence order
- Can be applied to various tasks, including in time series machine learning

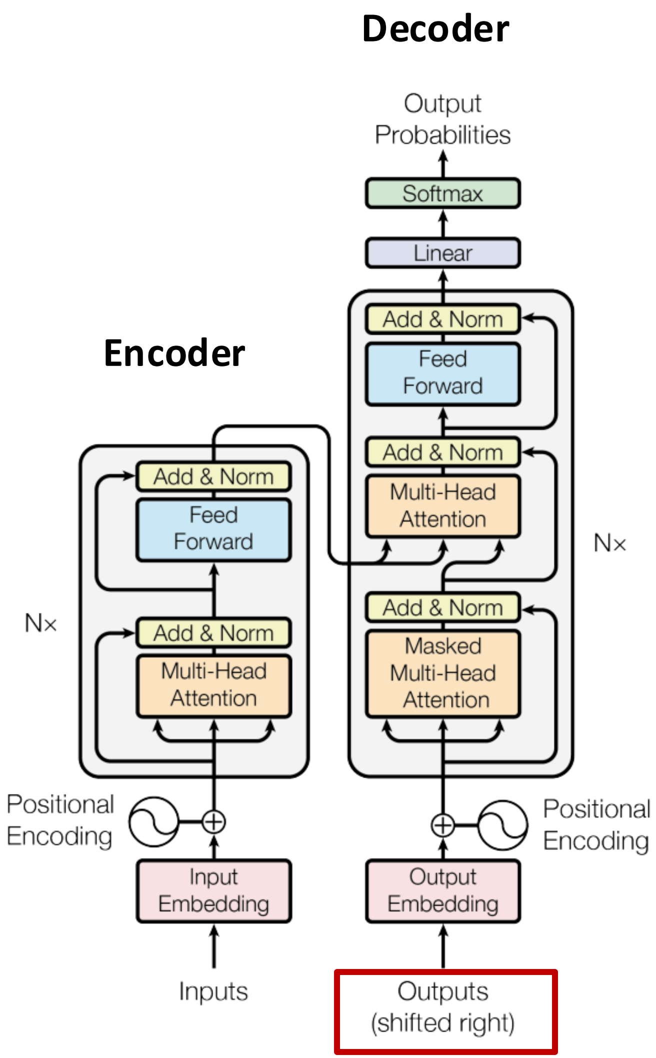
[1] “Attention is all you need”, Vaswani, et al.

The Transformer Architecture

The fundamentals components of the Transformer architecture are:

- **Positional encoding**
- **Multi-head self attention** based scaled dot product attention
- An **encoder-decoder** architecture

The transformer was first proposed as a **machine translation** model.



“Attention is all you need”, Vaswani, et al.

Transformer: Positional Encoding

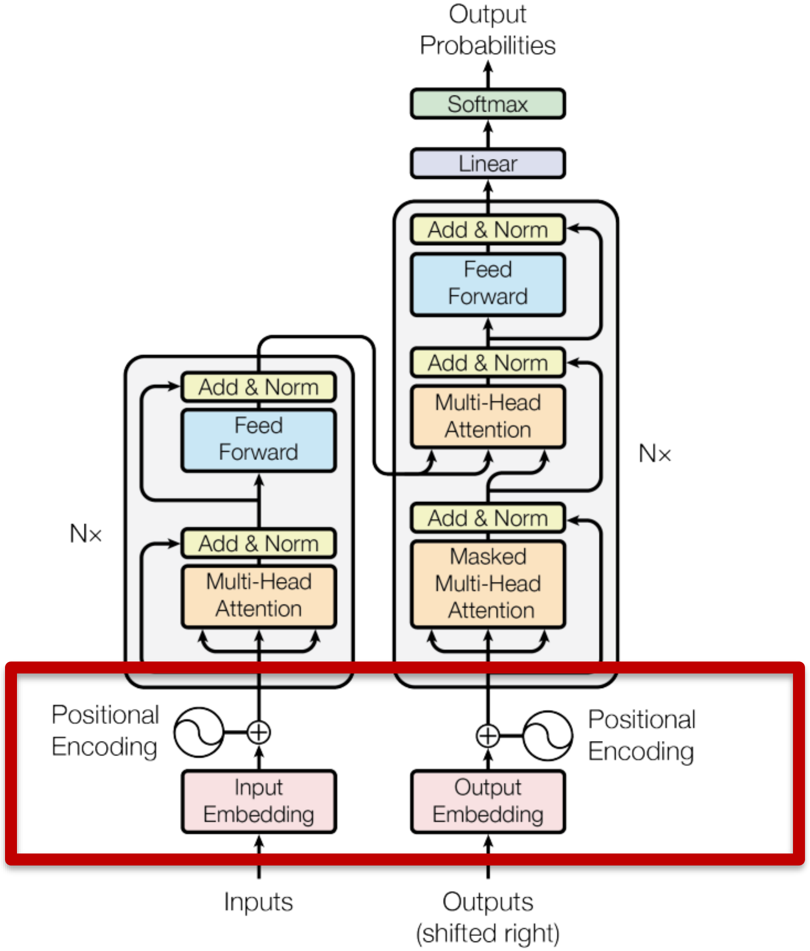
Positional encoding is added to the input and output sequence embeddings in the encoder and decoder. It is necessary to give information on **word order**.

Positional encoding is defined by using sine and cosine functions with different frequencies:

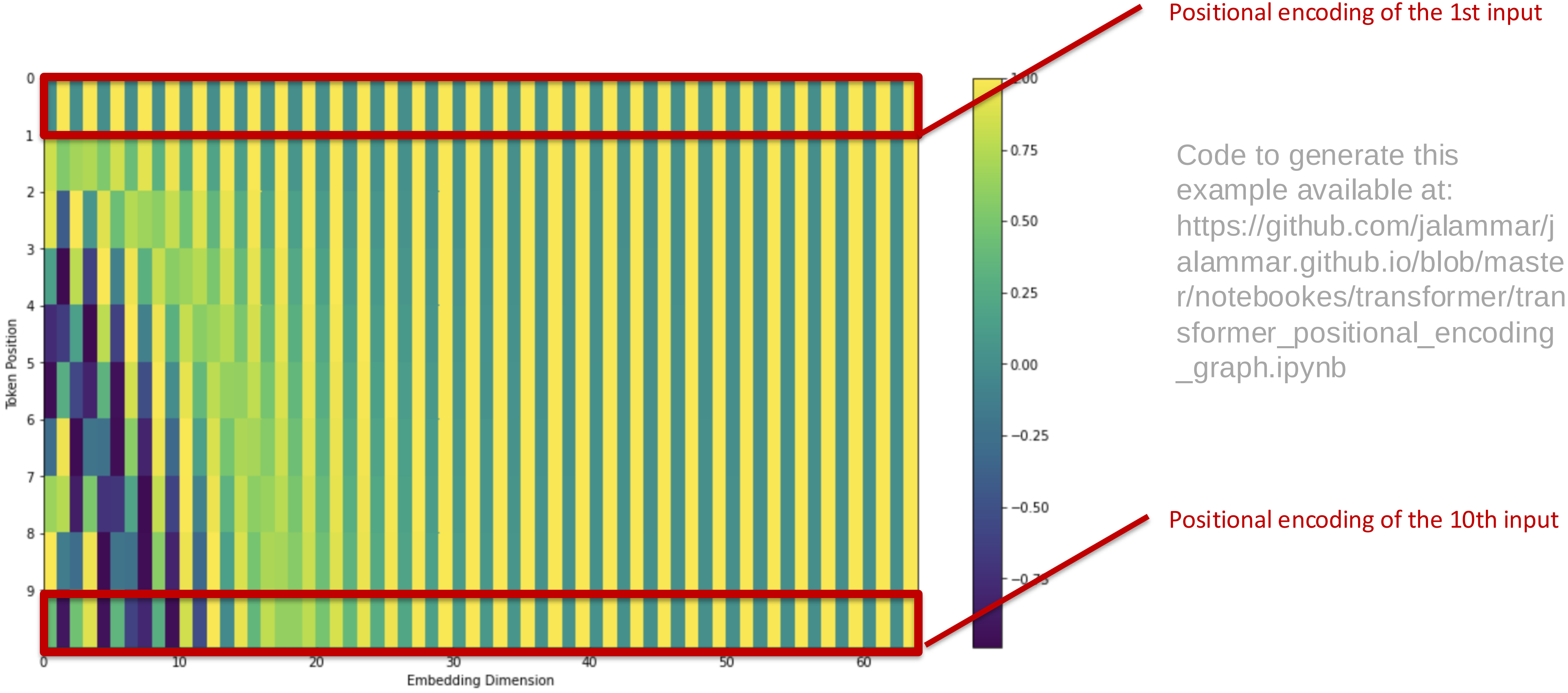
$$PE_{(pos,\,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,\,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

where pos is the position and i is the embedding dimension.

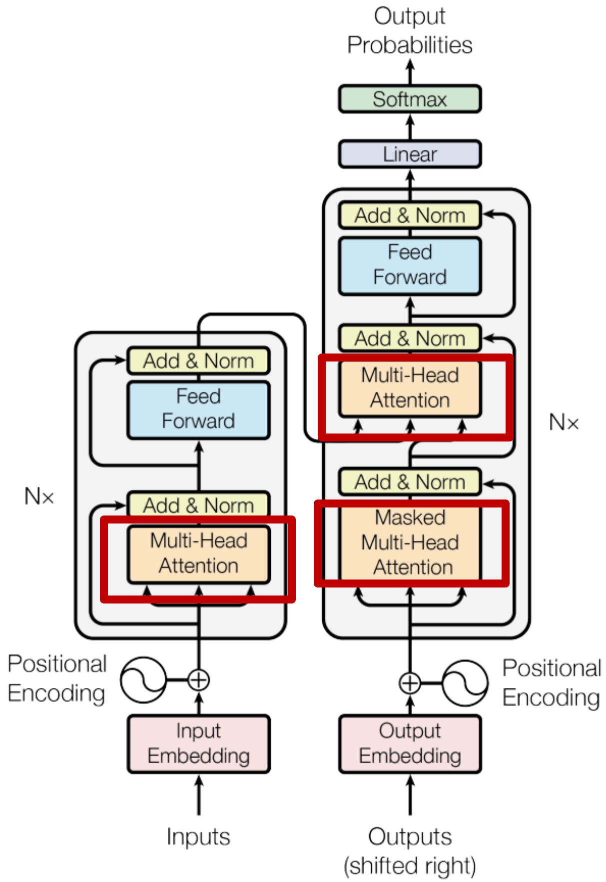


Transformer: Positional Encoding



Code to generate this example available at: https://github.com/jalammar/jalammar.github.io/blob/master/notebookes/transformer/transformer_positional_encoding_graph.ipynb

Transformer: Multi-head Attention Mechanisms



Three multi-head attention mechanisms are used in the Transformer architecture:

- **Encoder self-attention**: captures relationships between input sequence tokens
- **Decoder self-attention:** captures relationships between output sequence tokens
- **Encoder-decoder cross-attention**: captures relationships between input and output sequences

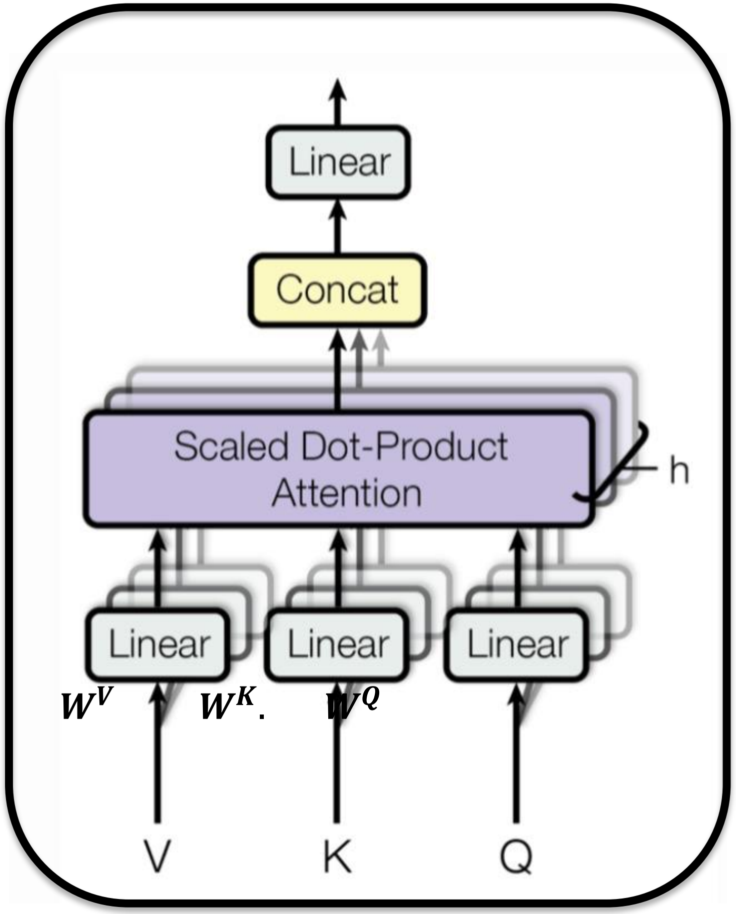
“Attention is all you need”, Vaswani, et al.

Transformer: Multi-Head Self-Attention

Multi-head attention is used to compute attention several times in **parallel**, using **independent** weight matrices $W^{V,i}$ $W^{K,i}$ and $W^{Q,i}$. These operations are called **attention heads**, and are then **concatenated**.

Basic scaled dot product attention is not sufficient to encode the **complexity** of language, as it might only focus on one aspect of relationships between tokens.

For instance, the multi-head attention mechanism can focus both on **long range and sort range relationships** through different attention heads.



Multi-head attention

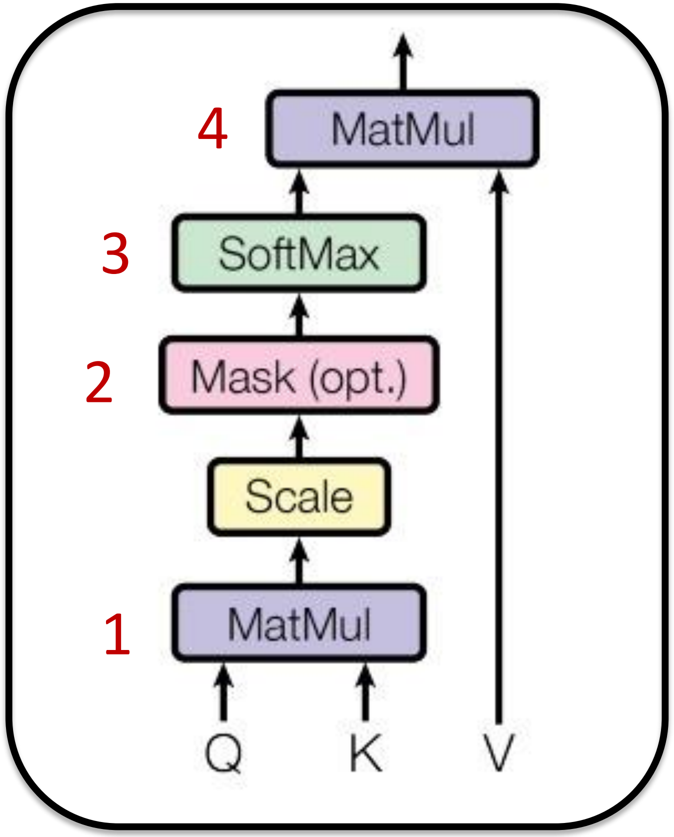
Transformer: Scaled Dot Product Attention

Then, the scaled dot-product attention is computed by:

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the dimension of $\boldsymbol{Q}$ and $\boldsymbol{K}$.

1. How each token is related to the query (i.e., similarity score)
2. Converts into probabilities (sum to 1)
3. **Causal masking** is used to prevent information leakage
4. Weighted sum of values that highlights relevant vectors



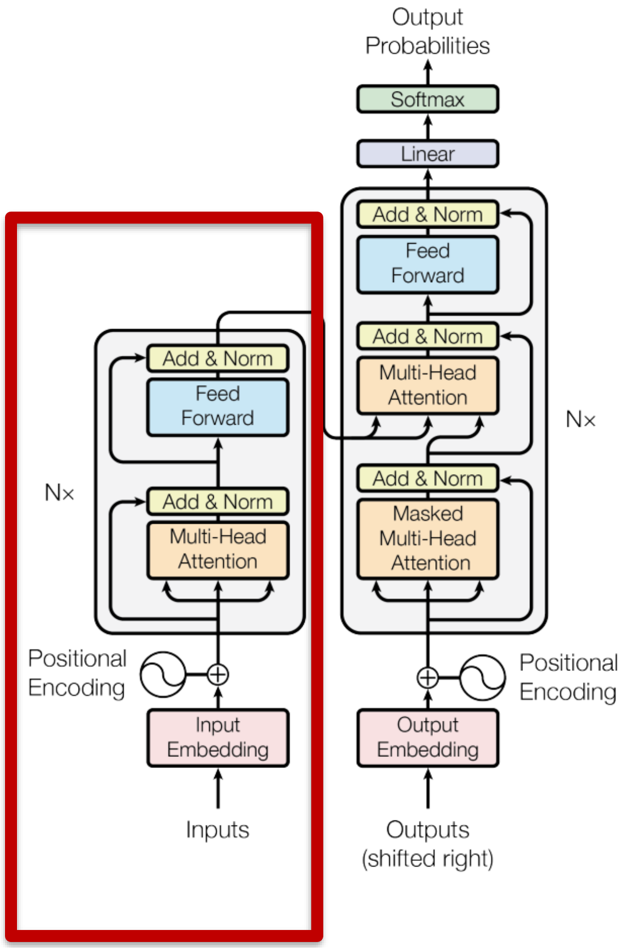
Scaled Dot-Product Attention

“Attention is all you need”, Vaswani, et al.

Transformer: Encoder

The encoder generates a **contextualized representation** of the **input sequence**. This representation has the same length as the input sequence, and is used to condition the decoder.

- It is made of N_x identical layers, each composed of
  - Multi-head **self-attention**
  - Feedforward neural network
  - Residual connection
  - Layer normalization

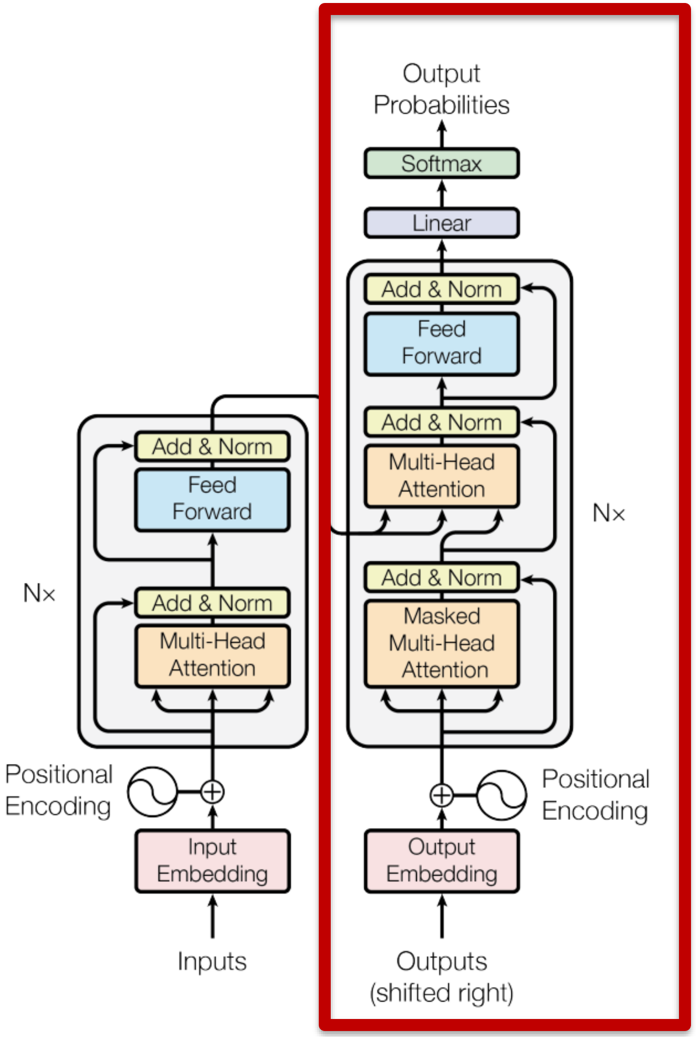


“Attention is all you need”, Vaswani, et al.

Transformer: Decoder

The **decoder** generates the output of the model (e.g., translated text). It generates tokens one at a time, using network information from the **encoded representation**.

- It is also made of N_x identical layers:
  - **Masked** multi-head self-attention
  - Multi-head **cross-attention**
  - Feedforward neural network
  - Residual connection
  - Layer normalization



[2] “Attention is all you need”, Vaswani, et al.

Transformer Model: Pros and Cons

The main advantages of Transformer models are:

- **Parallelization:** Transformers can process all tokens in a sequence simultaneously.
- **Long-Range Dependencies:** Transformers can capturing dependencies between distant tokens, as long as they are all present in the same sequence

However:

- **Quadratic time complexity** of attention mechanism

ADLTS \ DL for TS \
Recap

Lecture outline

1. Introduction to Deep Learning
2. Convolutional Neural Networks (CNNs)
3. Recurrent models (RNNs and LSTMs)
4. Transformers