Friedrich-Alexander-Universität
Technische Fakultät

Machine Learning Data Analytics

FAU

Machine Learning for Time Series

(MLTS or MLTS-Deluxe Lectures)

Dr. Dario Zanca

Machine Learning and Data Analytics (MaD) Lab
Friedrich-Alexander-Universität Erlangen-Nürnberg
17.01.2023

Topics overview

- Time series fundamentals and definitions (2 lectures)
- Bayesian Inference (1 lecture)
- Gussian processes (2 lectures)
- State space models (2 lectures)
- Autoregressive models (1 lecture)
- Data mining on time series (1 lecture)
- Deep learning on time series (4 lectures) ←
- Domain adaptation (1 lecture)

In this lecture…

1. **Introduction to Deep Learning (DL)**
2. **Recurrent Neural Networks (RNNs)**
3. **Backpropagation Through Time (BPTT)**

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?

Previous method needed **handcrafted features**:

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

What if we can not define generally applicable features?

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

→ With DL we can find features in a data driven way (e.g., Yolo beats prior approaches (4))

(4) Dean, T., Ruzon, M. A., Segal, M., Shlens, J., Vijayanarasimhan, S., & Yagnik, J. (2013). Fast, accurate detection of 100,000 object classes on a single machine. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1814-1821).

Deep learning for Time Series – Recurrent models

Introduction to Deep Learning

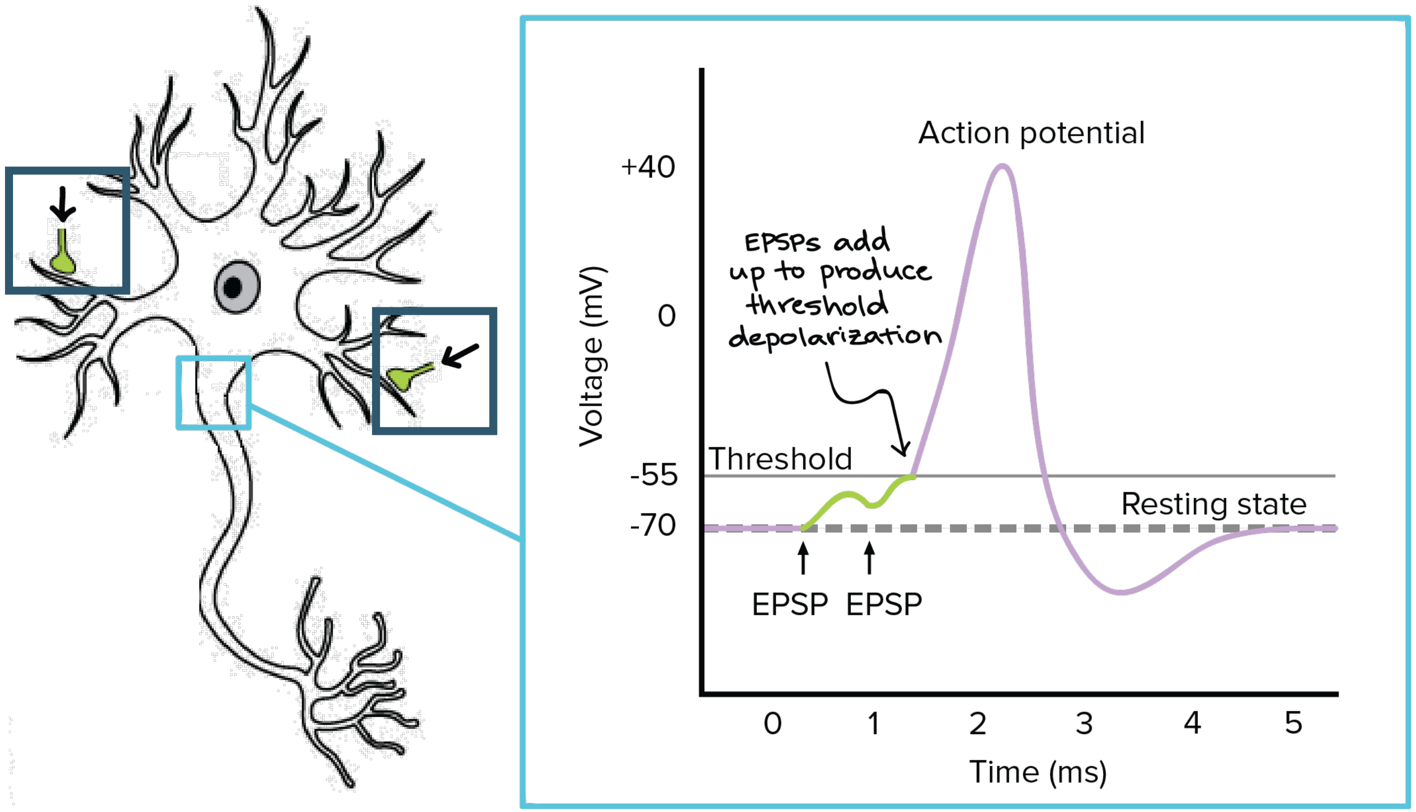
The Human Brain

The human brain is our reference for an intelligent agent, that

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

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

The Brain’s Neuron



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

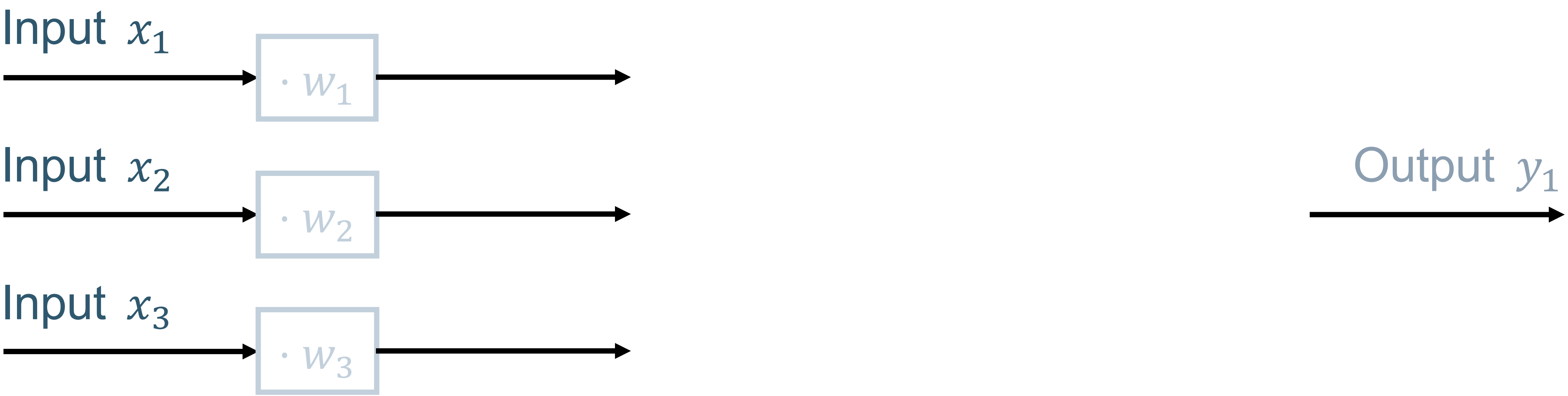
Image from https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/the-synapse

The Perceptron – Computational model of a neuron

1. Let’s start by adding some basic components (input and output), we’re subsequently going to build the computational model step by step

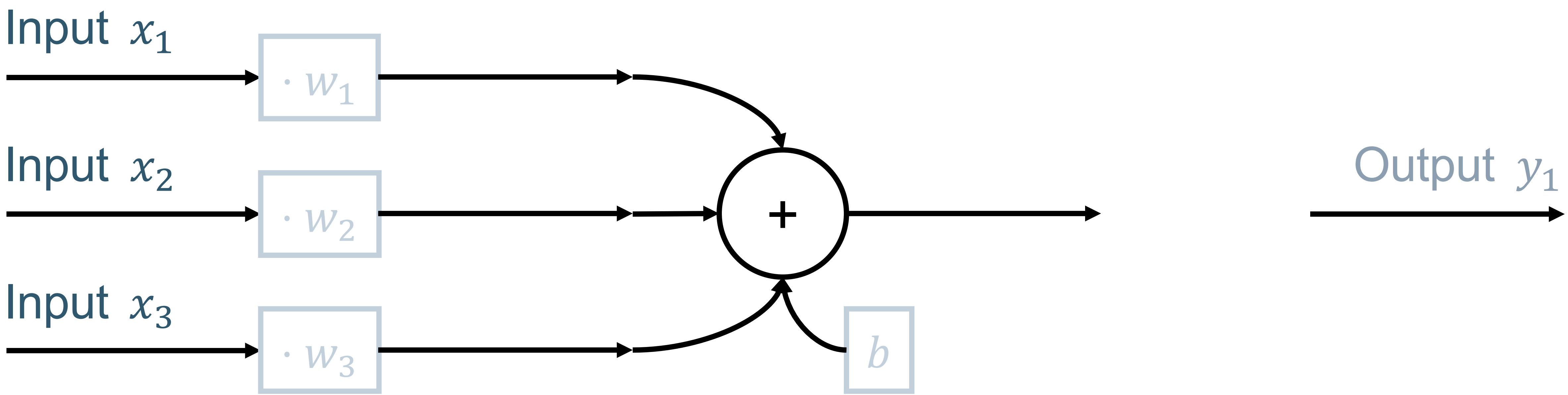
The Perceptron – Computational model of a neuron

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

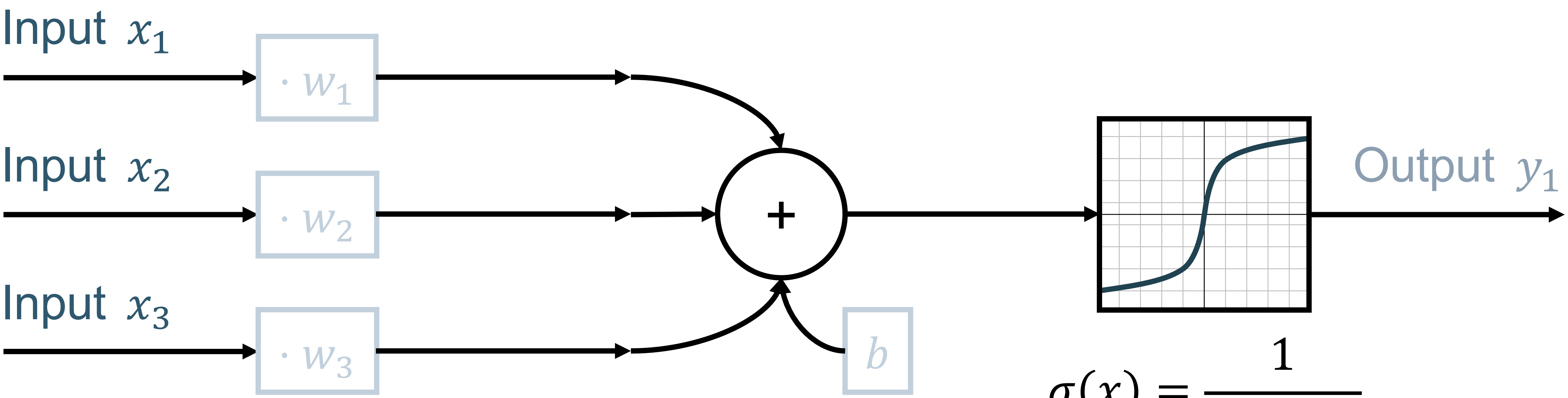


The Perceptron – Computational model of a neuron

3. We add up all the exitatory signals and the resting potential to determine the current potential.



4. A threshold function σ is applied to determine wether an action potential has to be sent in the output

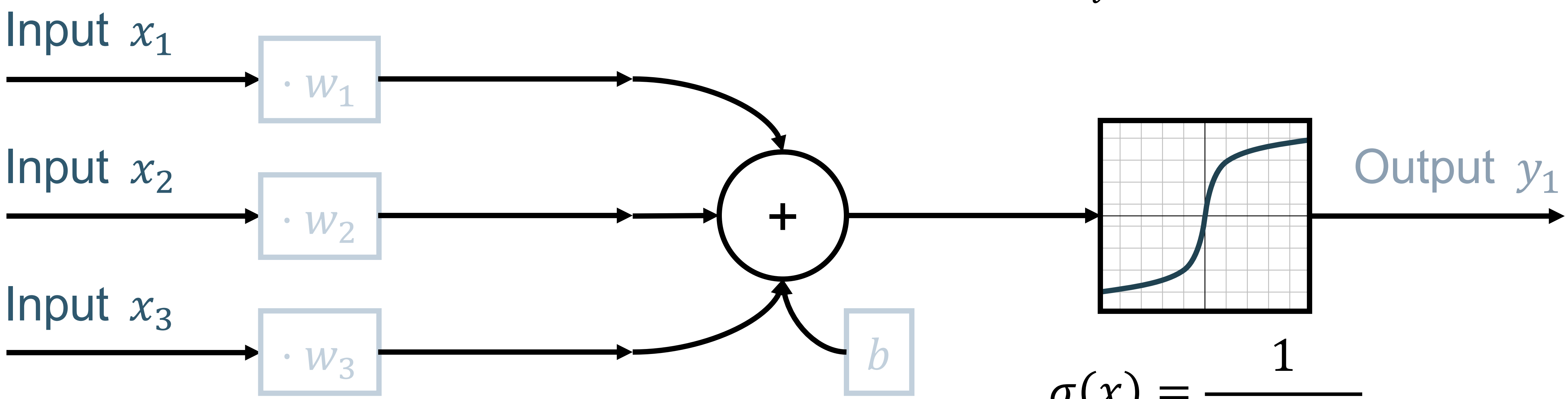


$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

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:

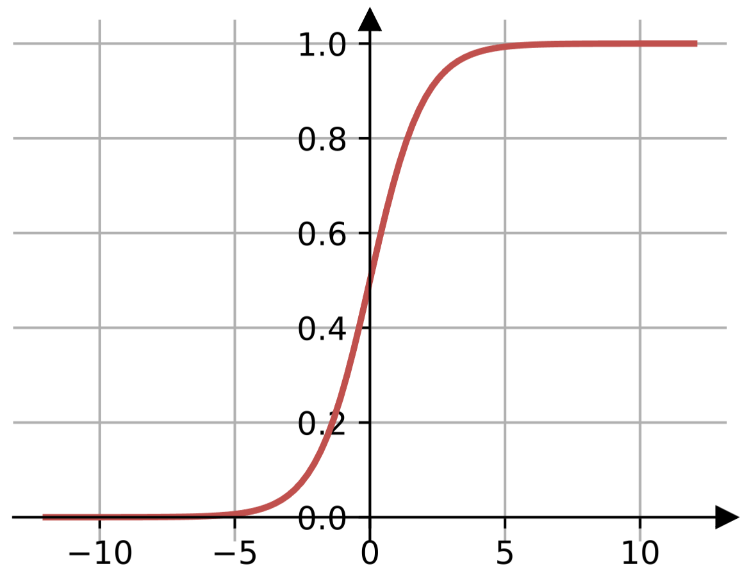
$$y_1 = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b) = \sigma(\sum_i w_i \cdot x_i + b)$$



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

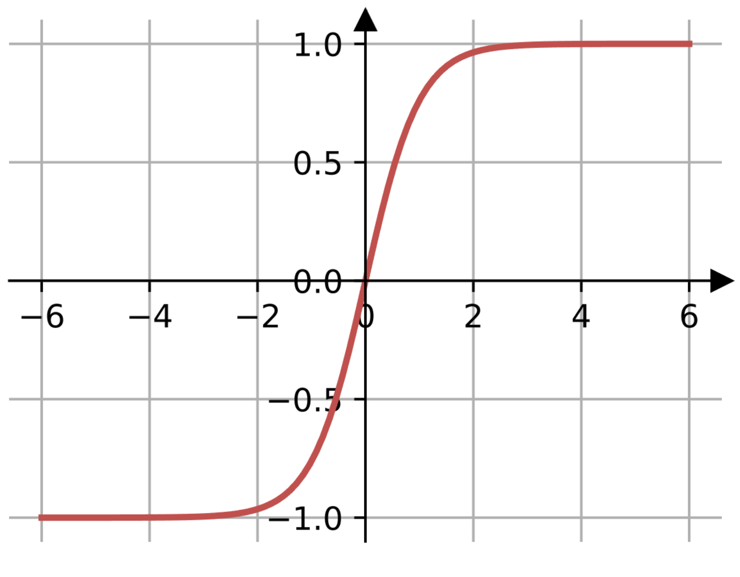
Activation funcitons

Sigmoid



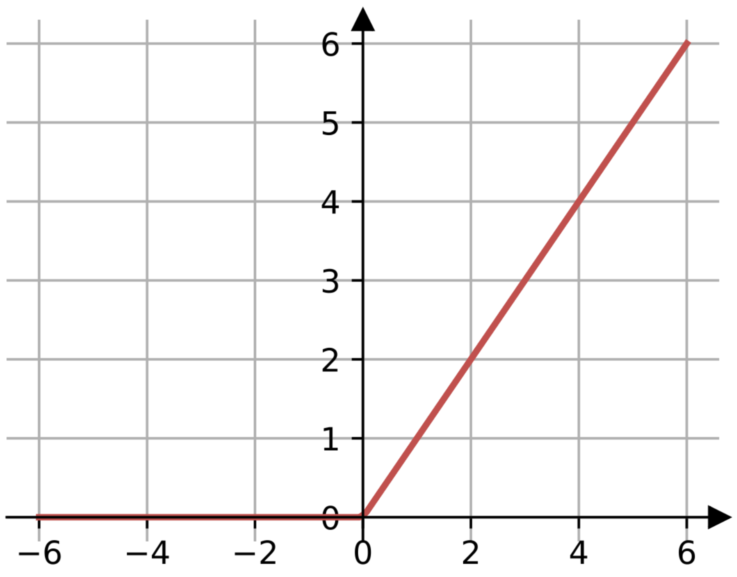
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Hyperbolic Tangent



$$\sigma(x) = \tanh(x)$$

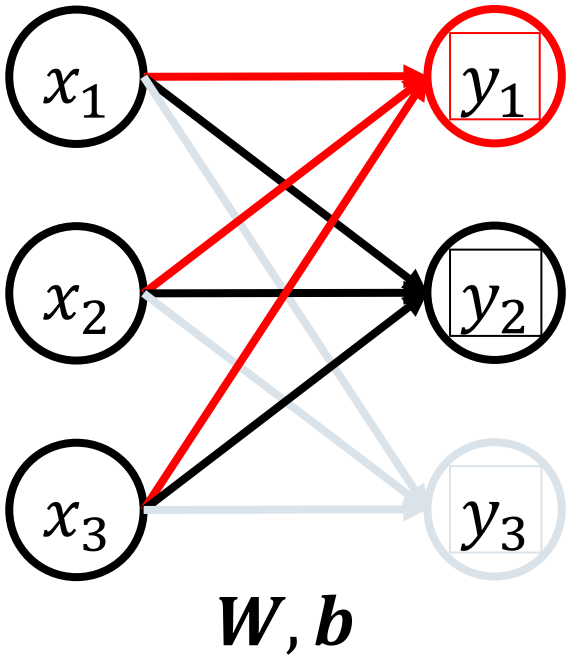
Rectified Linear Unit



$$\sigma(x) = \max(x, 0)$$

Shallow networks

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



We can thus rewrite the three computations as:

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

Or in a more simplified form:

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

Multilayer perceptron (MLP)

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

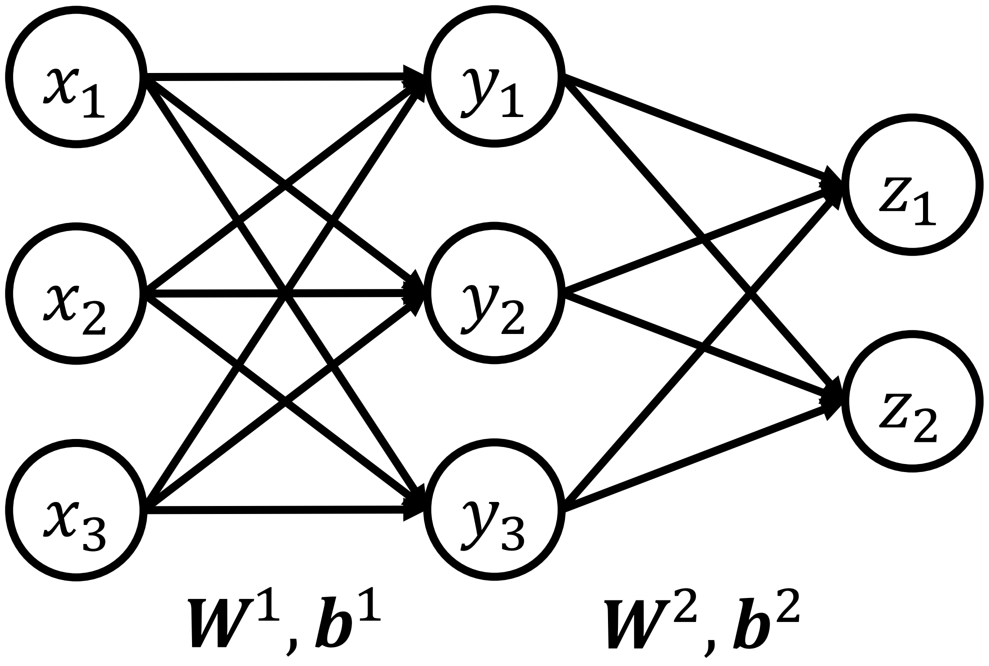
$$y = \sigma(W^1 \cdot x + b^1)$$

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

Combining these leads us to:

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

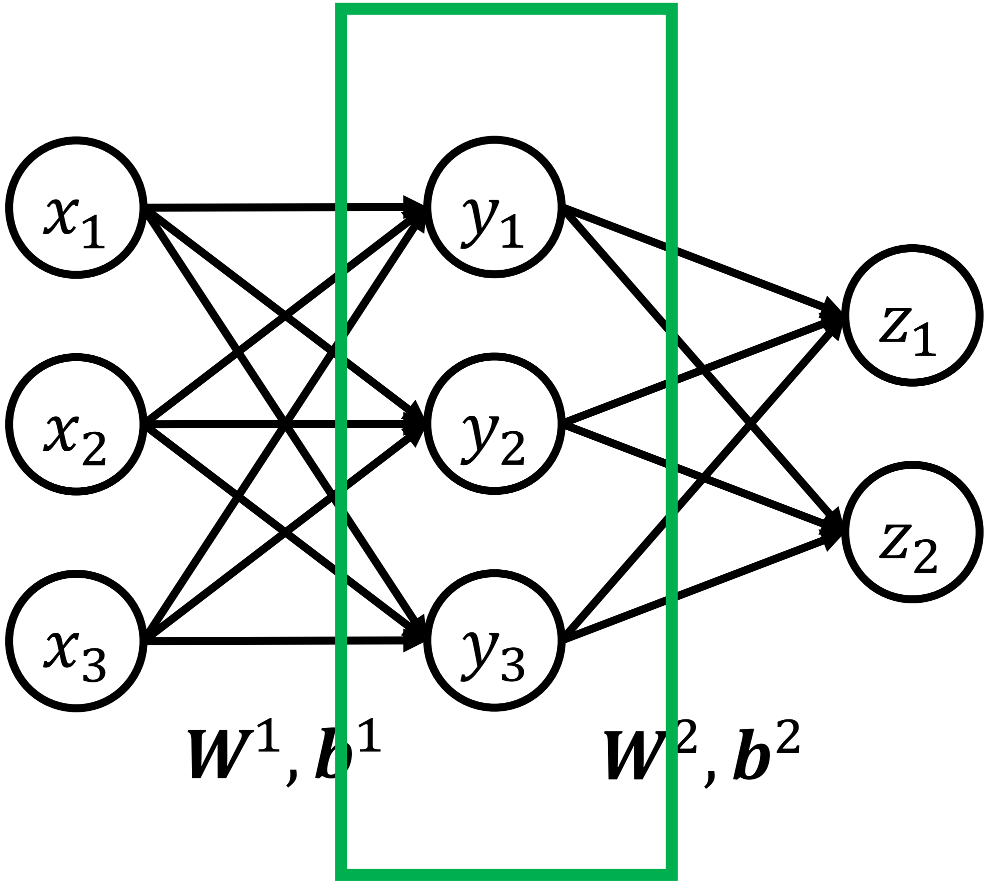
- Each layer has its own set of parameters (weights $\boldsymbol{W}^i$ and bias $\boldsymbol{b}^i$)
- The underlying computation is a matrix multiplication described by
$$\boldsymbol{y}^{i+1} = \sigma(\boldsymbol{W}^i \cdot \boldsymbol{y}^i + \boldsymbol{b}^i)$$



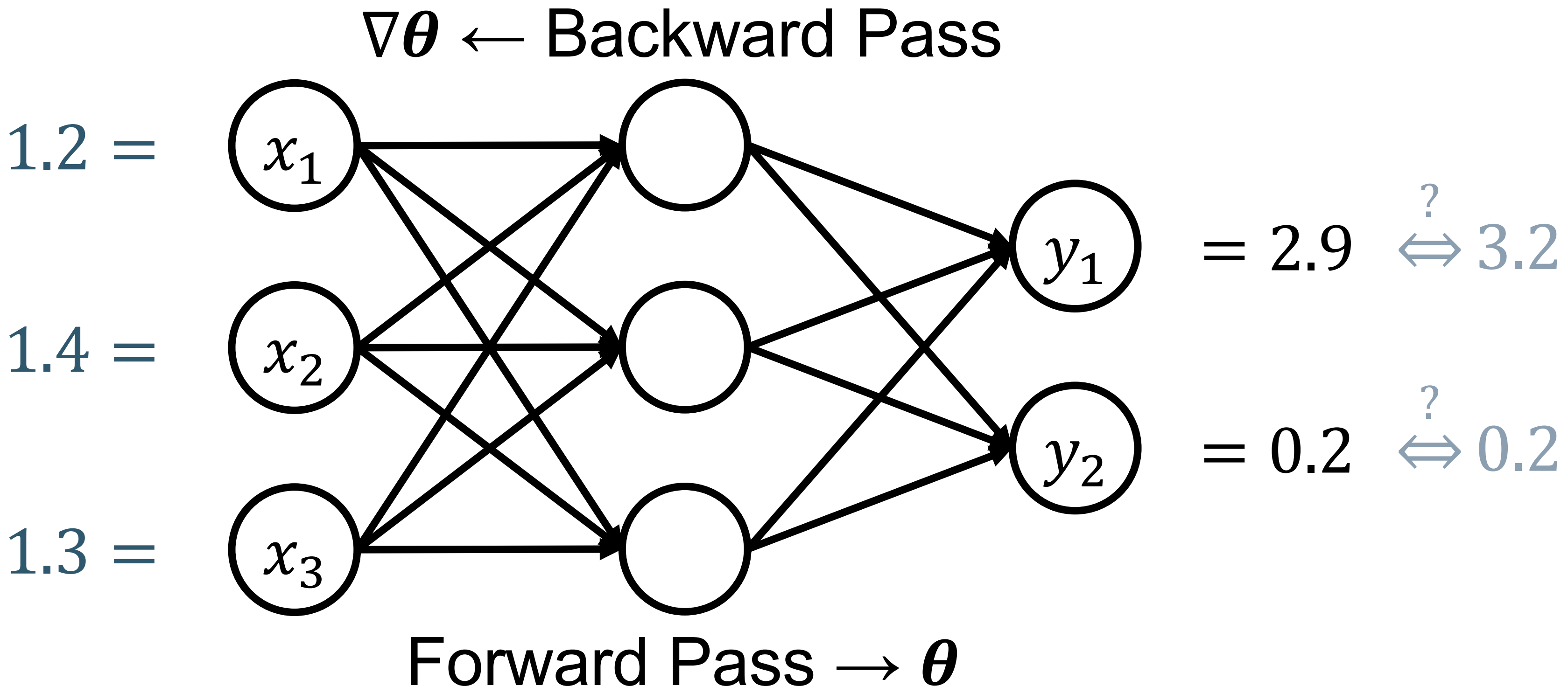
Multilayer perceptron (MLP)

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

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



How do we learn model parameters?



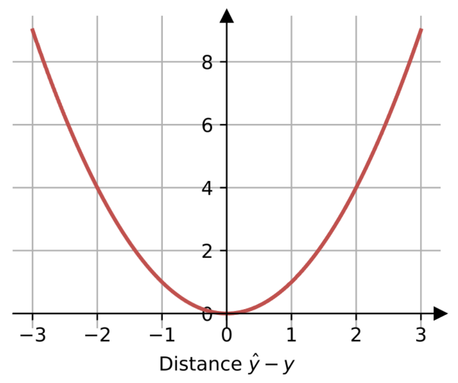
The loss function

The loss function is a comparison metric between the predicted outputs $\hat{y}_i$ and the expected outputs y_i.

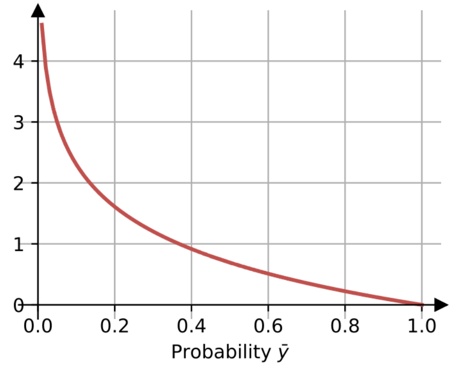
The choice of the loss function usually depends on the type of problem:

- For regression, a common metric is the mean squared error
- For classification, a common metric is the cross entropy

$$MSE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$



$$CE(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{i} y_i \log(\hat{y}_j)$$



Gradient descent

Gradient Descent can be used to incrementally adjusts the parameters θ based on the gradient $\nabla\theta$ of the parameters

- For each iteration:
  - Compute the error of the parameters θ
  - Compute the gradient of the parameters $\nabla\theta$
  - Update parameters using $\theta^{i+1} = \theta^{i} - \lambda \cdot \nabla\theta^{i}$

where we denoted $\nabla\theta^{i} = \partial\mathcal{L}/\partial\theta^{i}$.

- The learning rate λ can lead to slow convergence if not properly configured
- There is no guarantee that the algorithm converges to the global optimum (e.g., due to learning rate λ or initial parameters θ^{1})

Backpropagation (BP)

Backpropagation algorithm is widely used to train artificial neural networks.

- In *fitting* a neural network, backpropagation computes **efficiently** the gradients of the loss function with respect to all weights in the network.
- This efficiency makes it feasible to use **gradient methods** for training multilayer networks, updating weights to minimize loss, like gradient descent.
- BP algorithm makes use of the **chain rule**, computing the gradient one layer at a time, to avoid redundant calculations.

Backpropagation (BP)

Let a_i^k be the activation of the i-th neuron in the k-th layer. This is related to the previous $(k-1)$-th layer by

$$a_i^k = \sum_{j=0}^{r_{k-1}} w_{ji}^k o_j^{k-1}$$

where r_{k-1} is the number of units in the $(k-1)$-th layer, and we simplified the notation incorporating the bias in the weights vector, as w_{0i}^k element.

Backpropagation (BP)

Let the loss function be:

$$E(X, \theta) = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

where y is the desired output and $\hat{y}_i$ is the predicted output from the neural network.

Backpropagation (BP)

The derivation of the backpropagation algorithm begins by applying the chain rule to the error function partial derivative

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k}\frac{\partial a_j^k}{\partial w_{ij}^k}$$

where the first term is usually called error and denoted by $\delta_j^k = \frac{\partial E}{\partial a_j^k}$

and the second term $\frac{\partial a_j^k}{\partial w_{ij}^k} = \frac{\partial}{\partial w_{ij}^k}\left(\sum_{j=0}^{r_{k-1}} w_{ji}^k o_j^{k-1} + b_j^l\right) = \mathrm{o}_\mathrm{i}^{k-1}.$

Thus, $\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k \mathrm{o}_\mathrm{i}^{k-1}$

Backpropagation (BP)

Now, considering the final layer m, let σ be the activation function of the final layer, applying the partial derivatives and using the chain rule we obtain

$$\delta_j^m = (\hat{y} - y)\sigma'\left(a_j^m\right)$$

which we can exploit to compute the error w.r.t. a specific weight:

$$\frac{\partial E}{\partial w_{ij}^m} = \delta_j^m o_i^{m-1} = (\hat{y} - y)\sigma'\left(a_j^m\right)o_i^{m-1}$$

Backpropagation (BP)

For an hidden layer k, the error term is given by

$$\delta_j^k = \sigma'(a_j^k) \sum_{i=1}^{r_{k+1}} w_{ji}^{k+1} \delta_i^{k+1}$$

which, similarly, we can exploit to compute the error w.r.t. a specific weight:

$$\frac{\partial E}{\partial w_{ij}^k} = \delta_j^k o_i^{k-1} = \sigma'(a_j^k) \left(\sum_{l=1}^{r_{k+1}} w_{ji}^{k+1} \delta_j^{k+1} \right) o_i^{k-1}$$

Backpropagation (BP)

Finally the update for each single weight is given by

$$\Delta w_{ij}^{k} = -\lambda \frac{\partial E}{\partial w_{ij}^{k}}$$

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

Deep learning for Time Series – Recurrent models

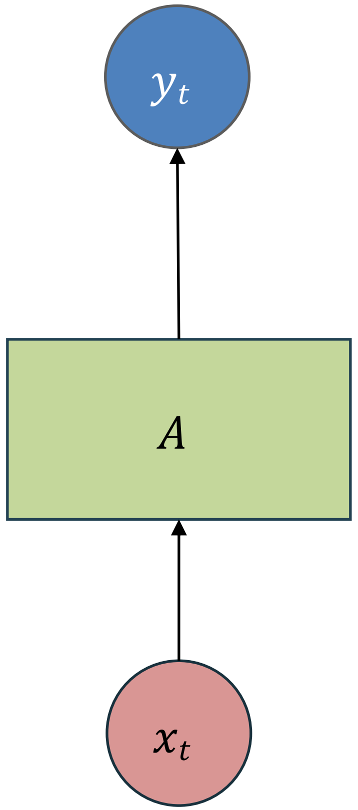
Recurrent Neural Networks (RNNs)

Limitations of NN for time series data

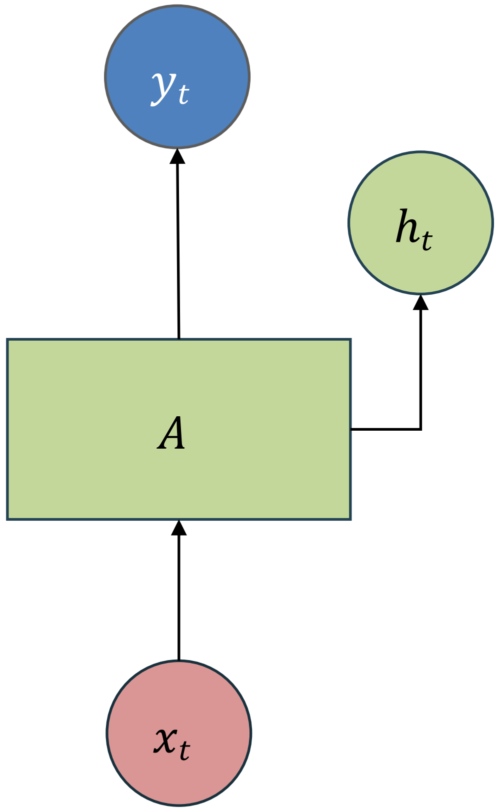
Feed-forward neural networks present some **disadvantages**, when applied to sequential data:

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

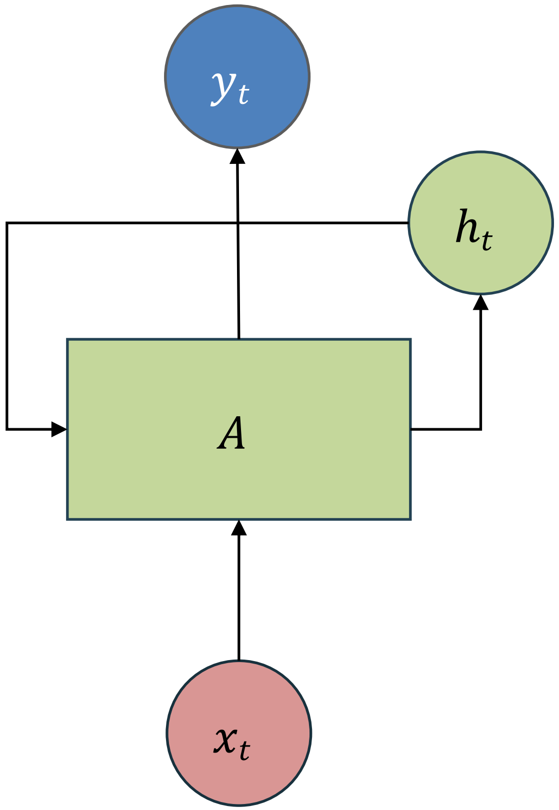
Recurrent Neural Network (RNN)



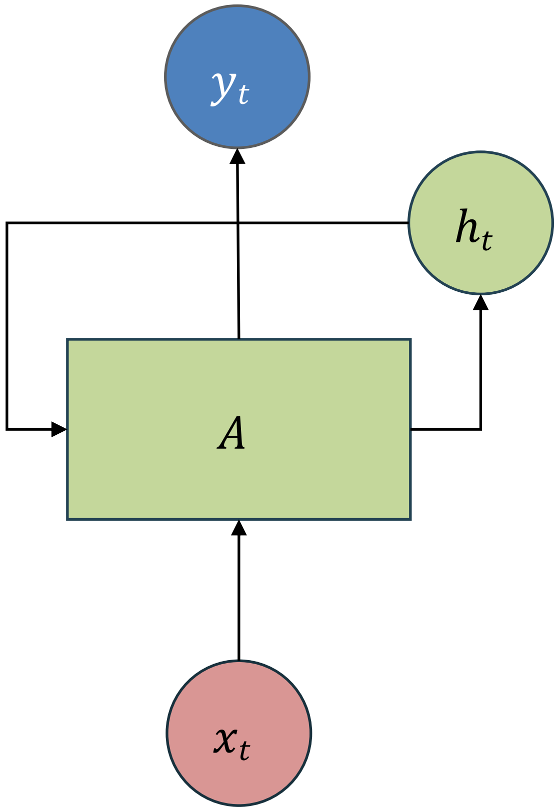
Recurrent Neural Network (RNN)



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:

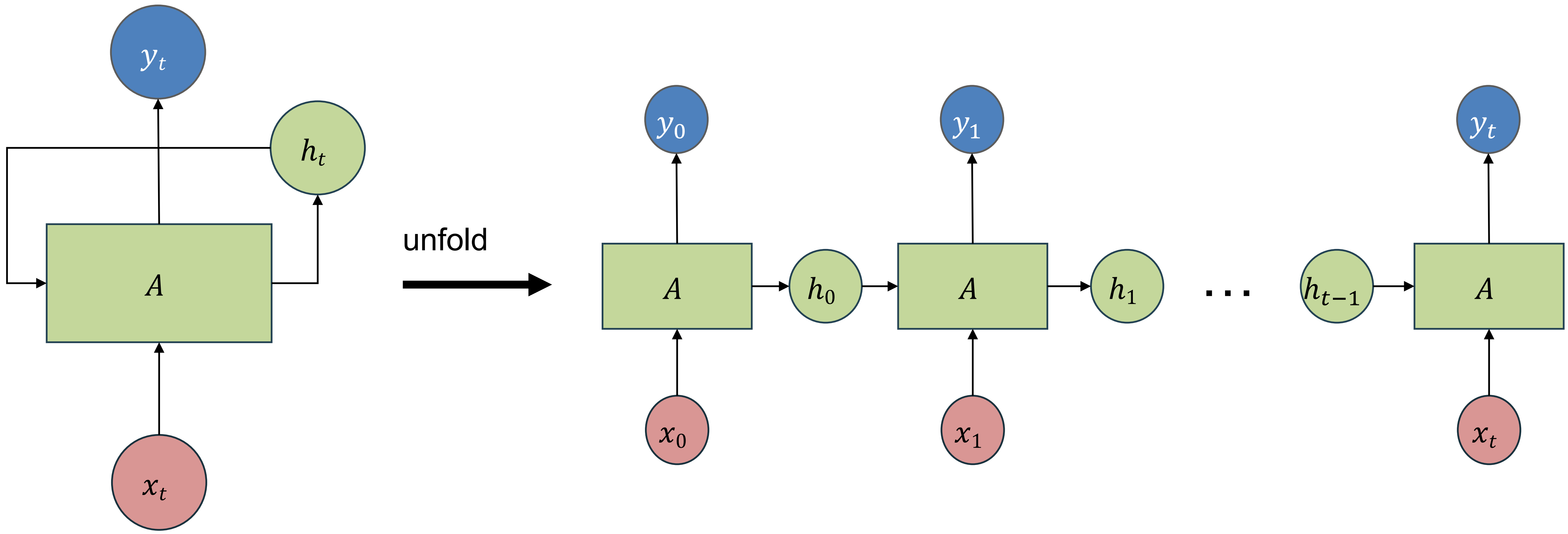
x_t: input at time t

A: neural network

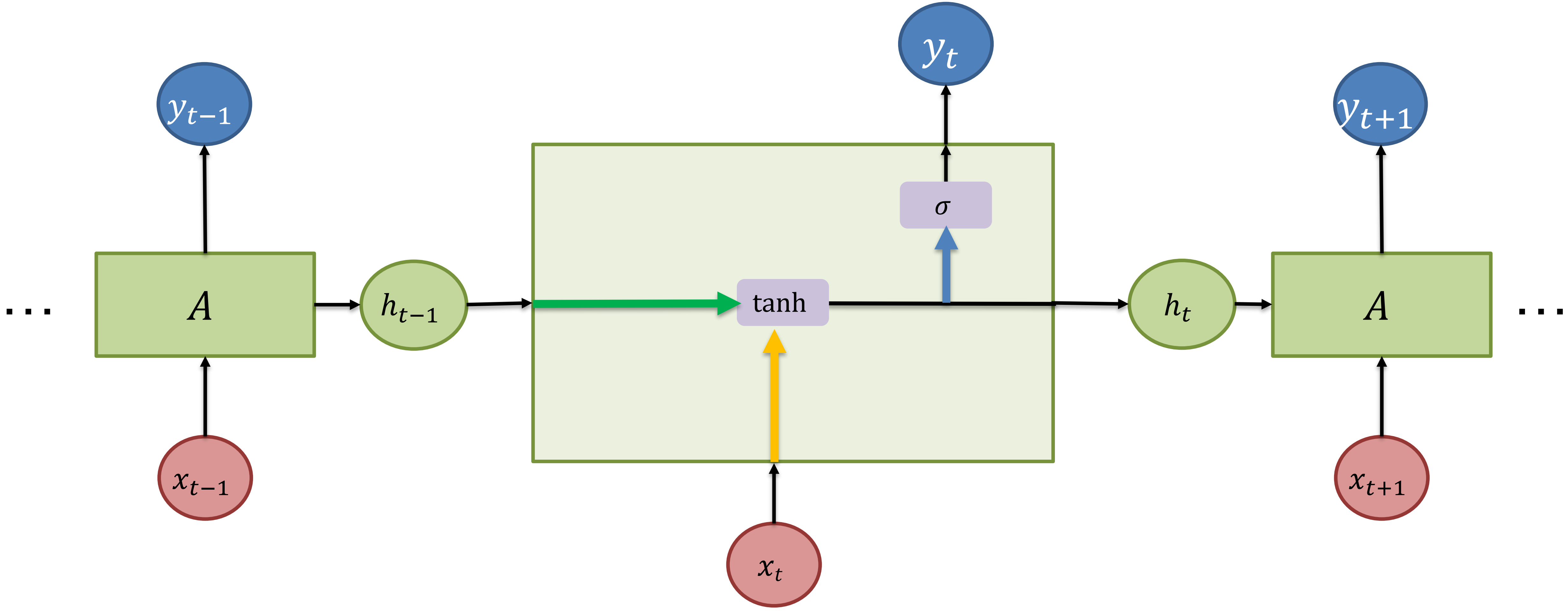
h_t: hidden state

y_t: output at time t

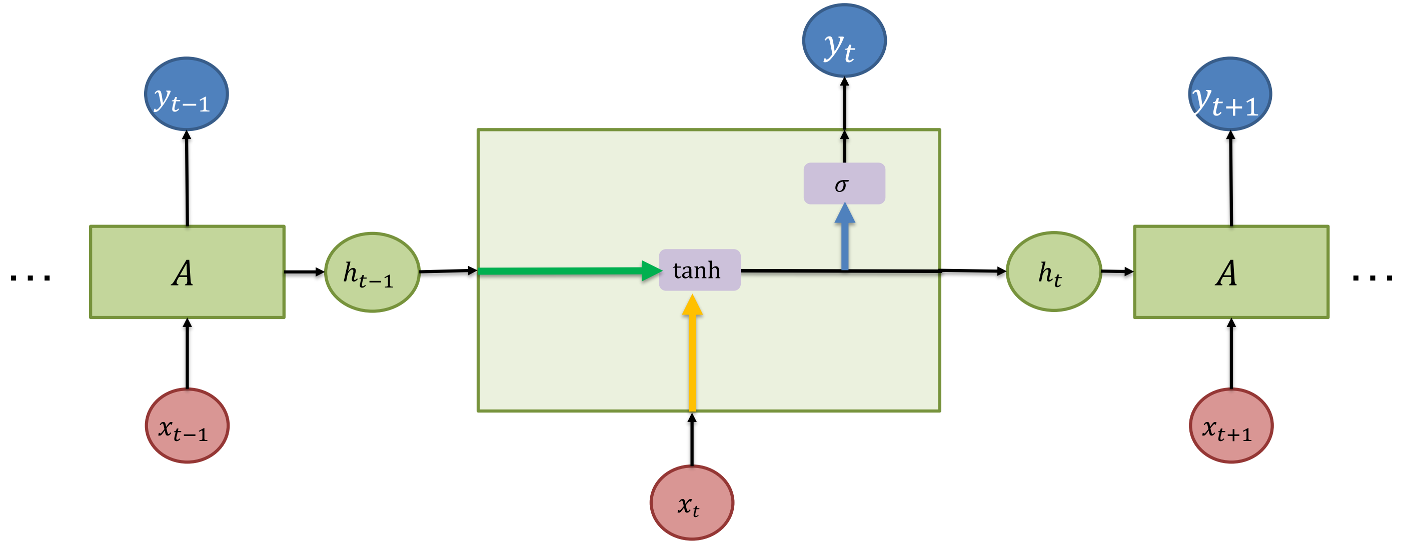
RNN Unfolding



Mathematical formulation



Mathematical formulation



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

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

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

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

Mathematical formulation

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

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

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

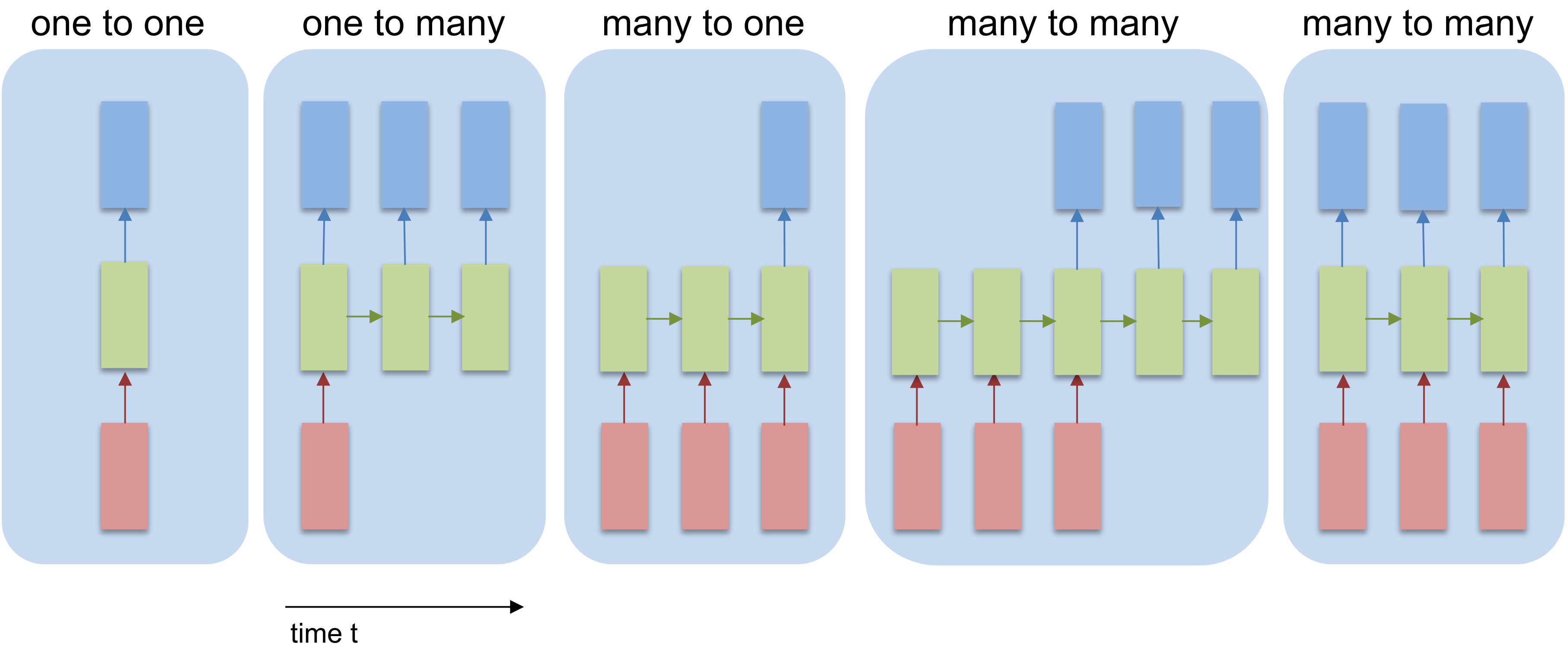
RNNs are universal approximators

The Universal Approximation Theorem states that:

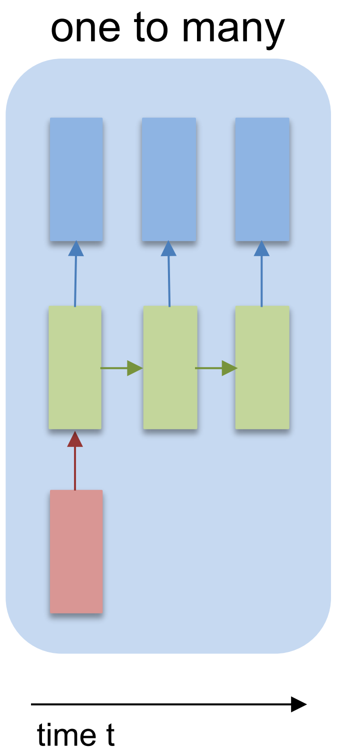
> *„Any non-linear dynamical system can be approximated to any accuracy by a recurrent neural network, with no restrictions on the compactness of the state space, provided that the network has enough sigmoidal hidden units.“*

- 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

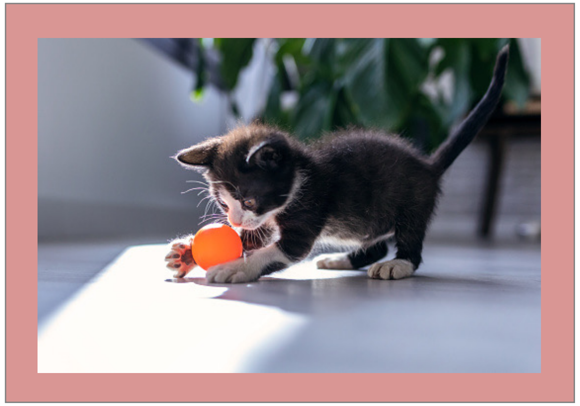


Example: one to many



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

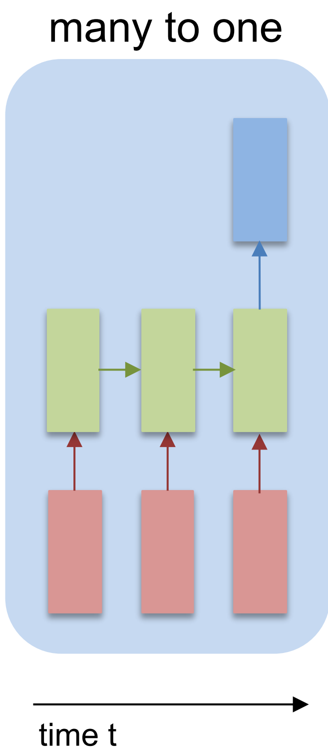
Input:



Output:

A | cat | playing | with | a | ball

Example: many to one

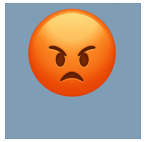


A typical example of a many to one problem is that of **sentiment analysis.**

Input:

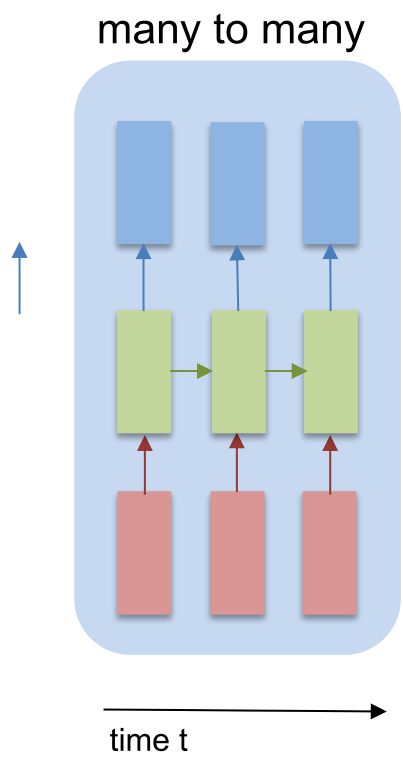
Horrible service the room was dirty

Output:



("negative")

Example: many to many



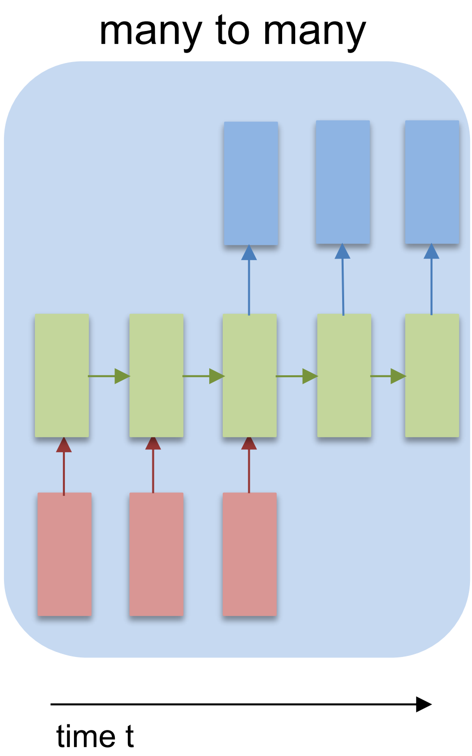
A typical example of a many to many problem is that of name entity recognition.

Input:

Output:

Harry	Potter	and	Hermione	invented	a	new	spell
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)

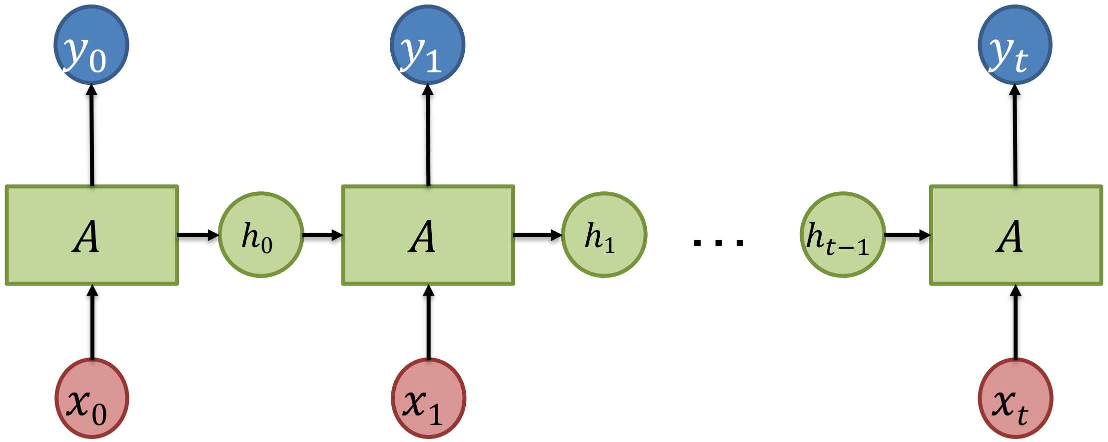
Backpropagation Through Time (BPTT) learning algorithm is a natural extension of the standard backpropagation that performs gradient descent on a complete unfolded RNN.

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

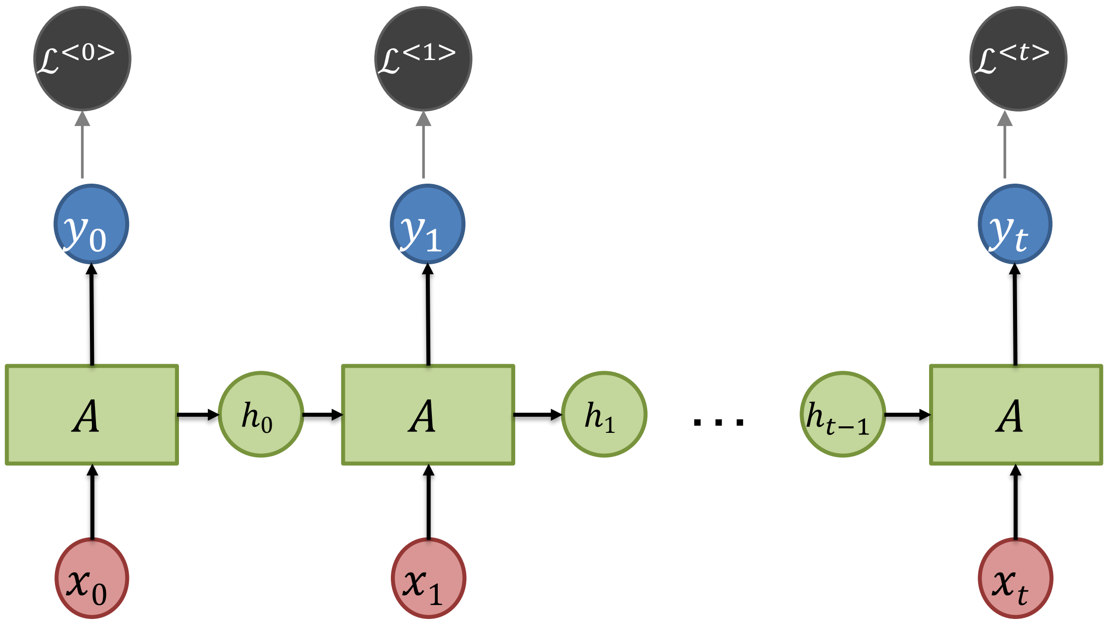
BPTT: Forward propagation

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



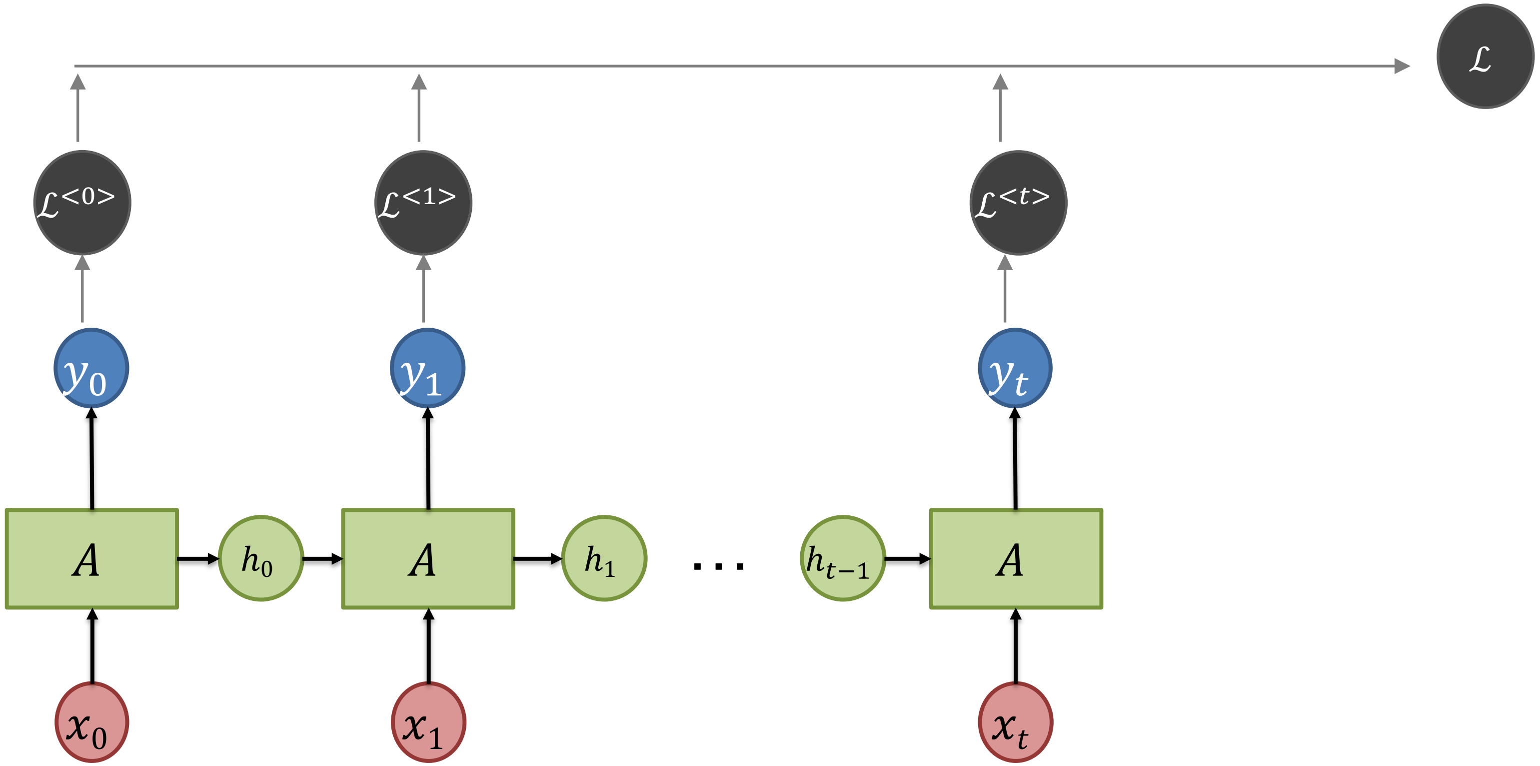
BPTT: Loss computation

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



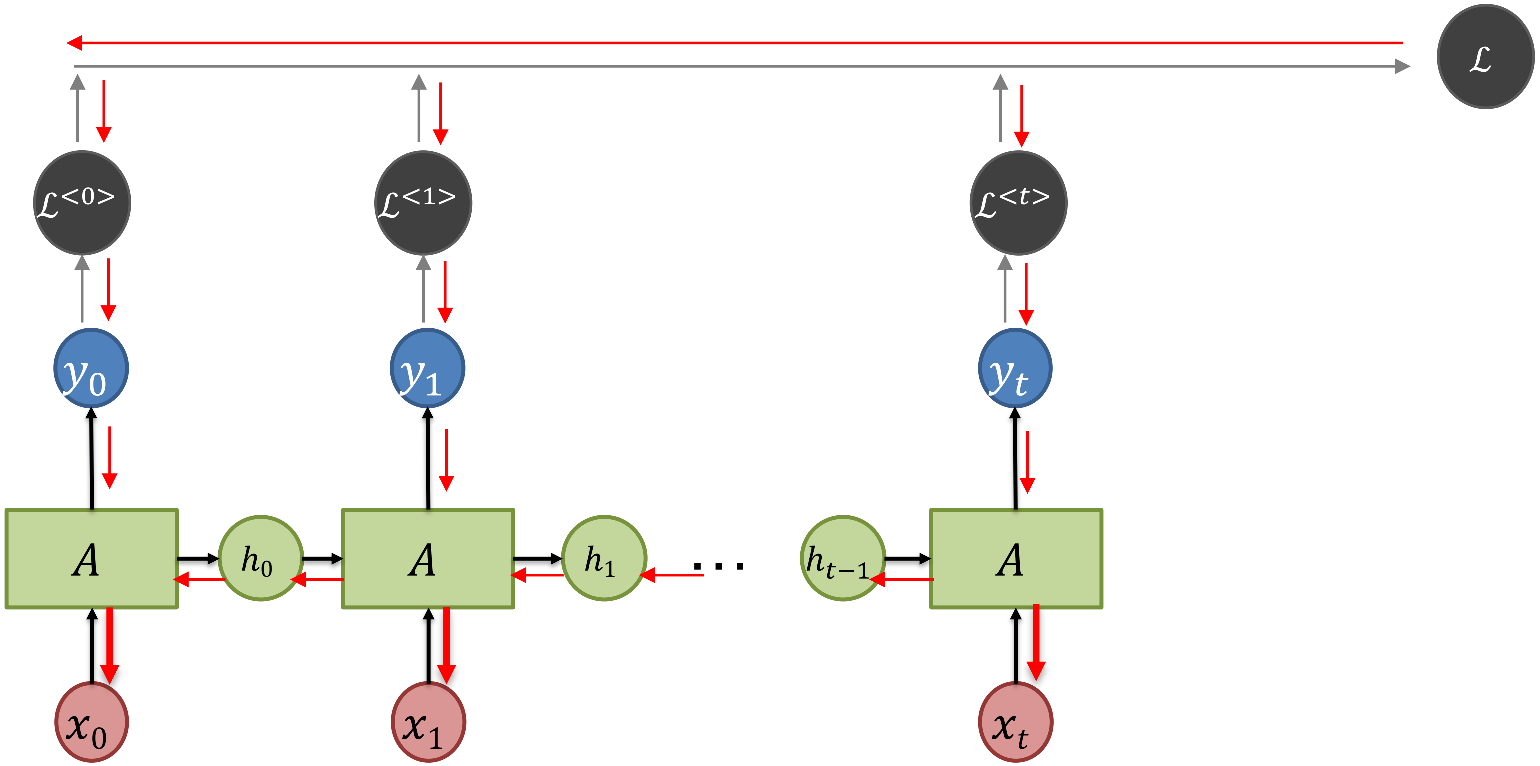
BPTT: Loss computation

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



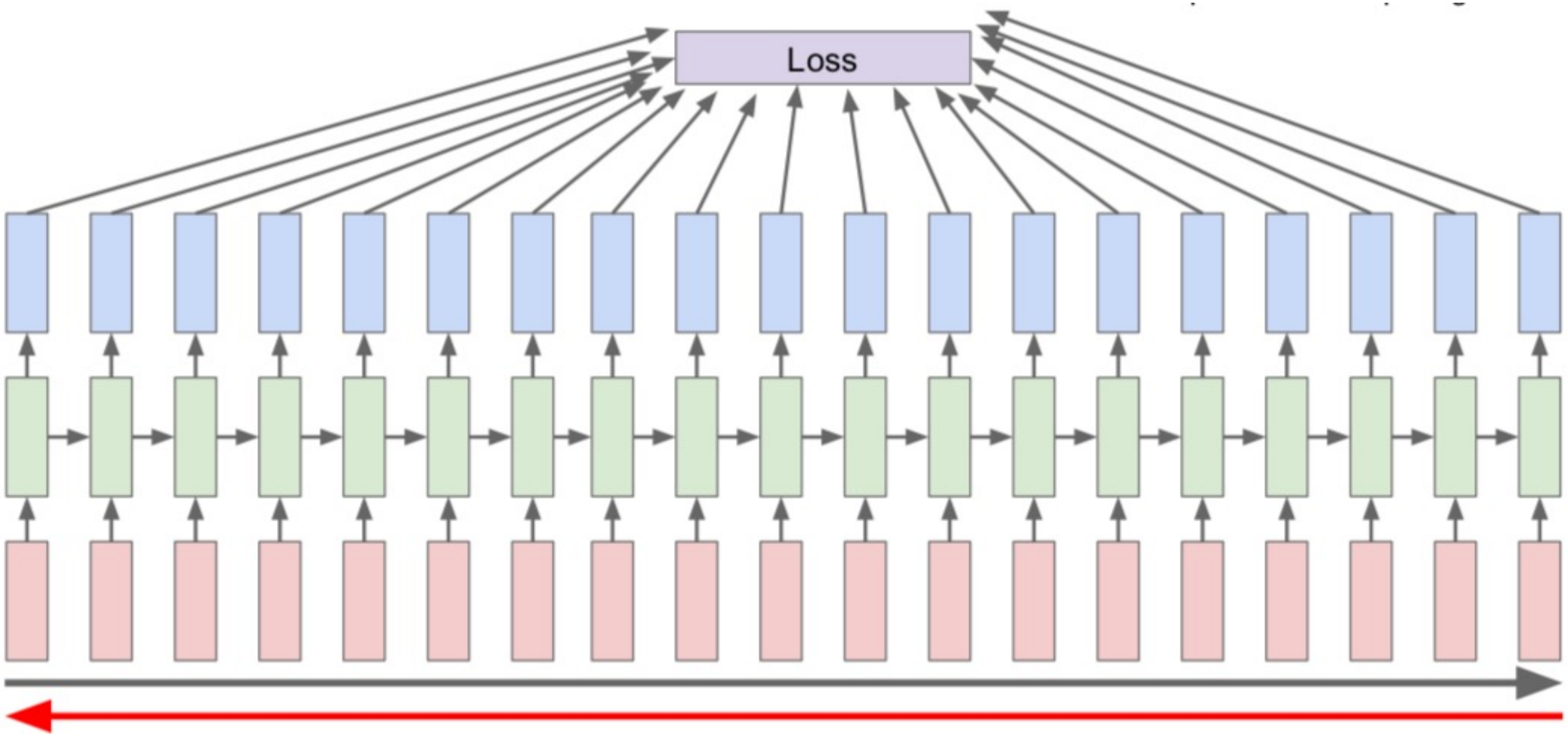
BPTT: Backpropagation

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



BPTT: Limitations

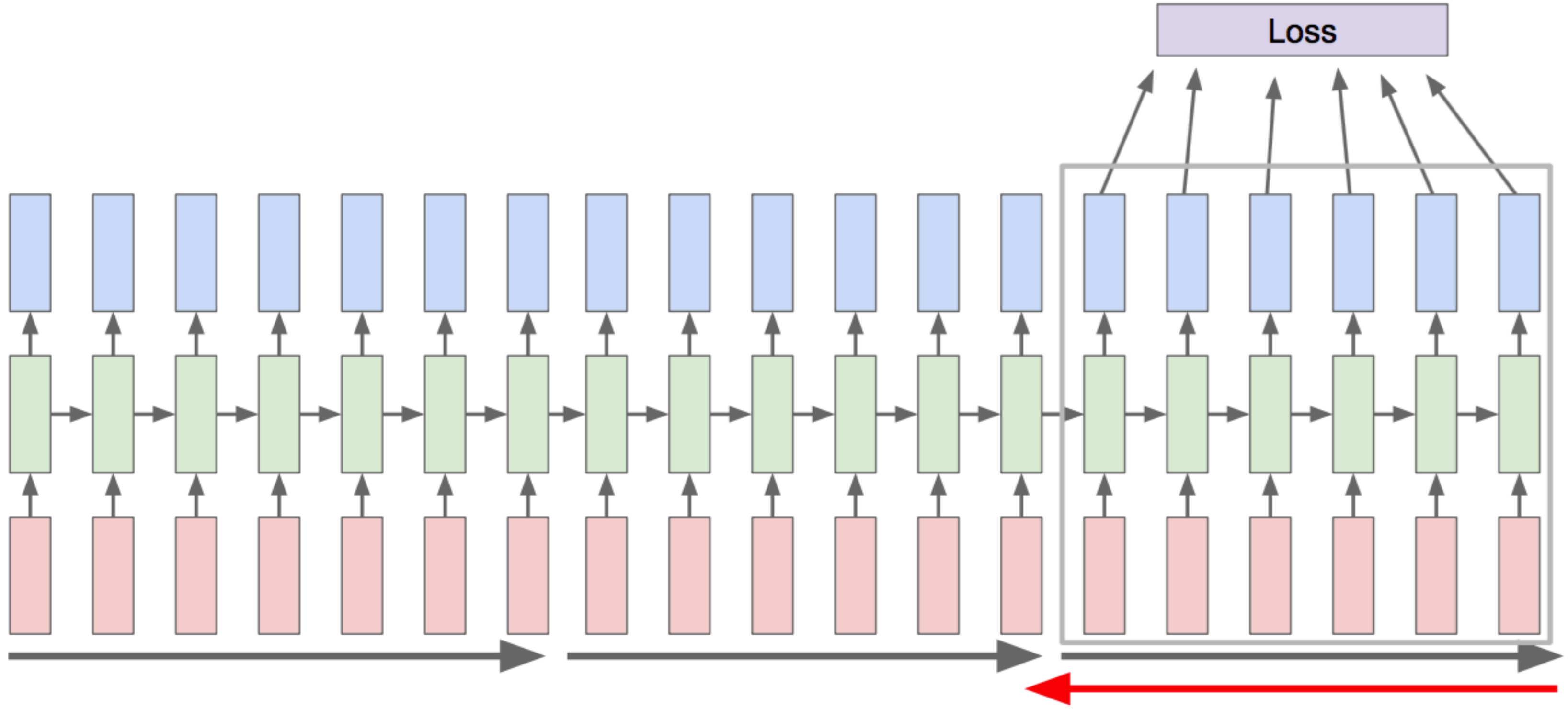
BPTT can be computationally very expensive as a lot of partial derivatives have to be computed, depending on the complexity of the network.



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

Recap

In this lecture

- **Deep learning**
    - Perceptron
    - Layers
    - MLP
    - Gradient Descent
    - Backpropagation
- **Recurrent neural network**
    - Model
    - Architectures
- **Backpropagation through time**
    - BPTT
    - Trunc-BPTT

RNNs pros and cons

Pros:

- 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:

- Vanishing/Exploding gradient