

# Introducing Deep Learning

---

Thomas Gerald

March 7, 2024

Laboratoire Interdisciplinaire des Sciences du Numérique – LISN, CNRS

# The course

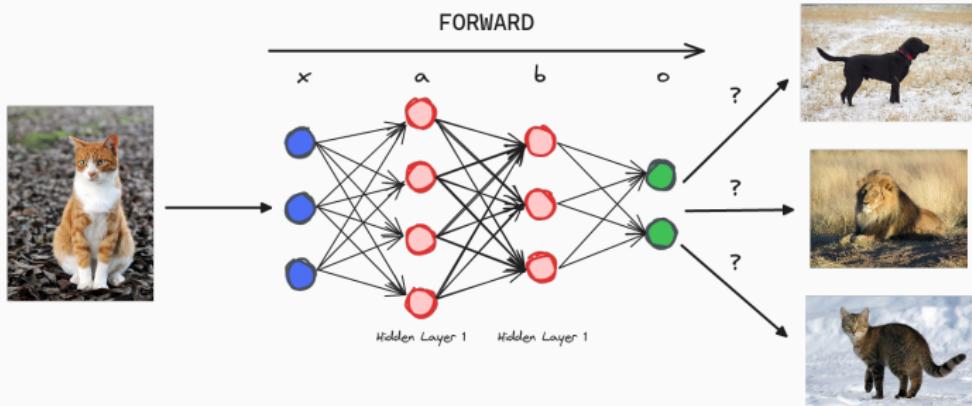
## The lectures

- Introduction to neural networks
- Neural networks architectures overview
- Training neural networks
- Convolutional neural networks
- Generative models (GAN, VAE)

## Information

Course material:  
→ E-Campus<sup>1</sup>

Grading :  
→ 50% exercises (on machine)  
→ 50% exam



<sup>1</sup><https://ecampus.paris-saclay.fr/course/view.php?id=115777#section-15>

# Schedule

## Thomas Gerald:

Contact : thomas.gerald@universite-paris-saclay.fr

- **03/07/2024:** Introduction to Deep-Learning approaches (lecture)
- **03/14/2024:** Training Neural Networks (lecture + exercises)
- **03/21/2024:** Neural Networks Architecture (lab exercises)
- **03/28/2024:** Introduction to numpy and the Back-propagation Algorithm (lab exercises)
- **04/11/2024:** Pytorch (lab exercises)

## Michèle Sébag:

- **04/04/2024:** Convolutional Neural Networks (lecture)

# Requirements

## Background

- Machine learning basics
- Gradient computation/ derivative  
(for the second lecture)

## Programming

- Python, jupyter notebook
- Numpy, matplotlib, pytorch

# Introduction to Deep-Learning approaches

## Content

- Machine Learning: The principle
- Linear classification: definition and issues (convex set)
- Representation learning and the multi-layer perceptron

## Objectives

- Explain what is deep-learning
- Develop a classifier/regression with different deep architecture
- Aware of the

# What is Machine Learning?

## Machine learning:

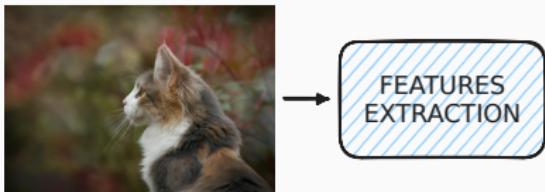
- “the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data”

<https://it.wustl.edu/ai/>

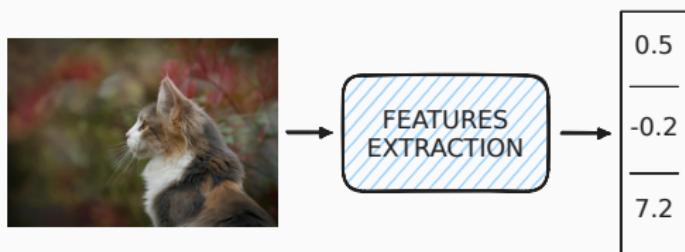
# Before Deep Learning



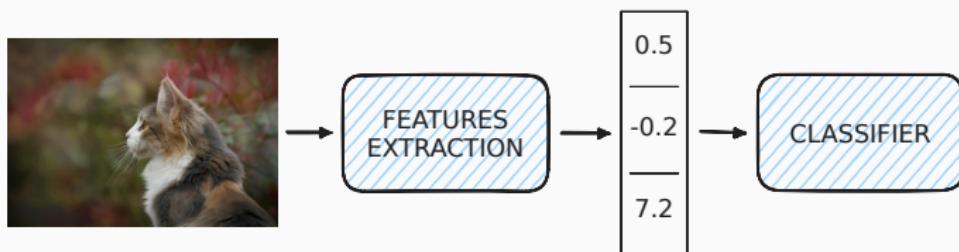
# Before Deep Learning



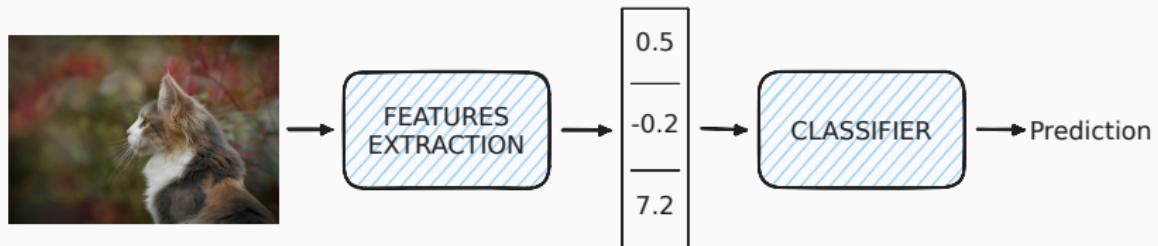
# Before Deep Learning



# Before Deep Learning



# Before Deep Learning

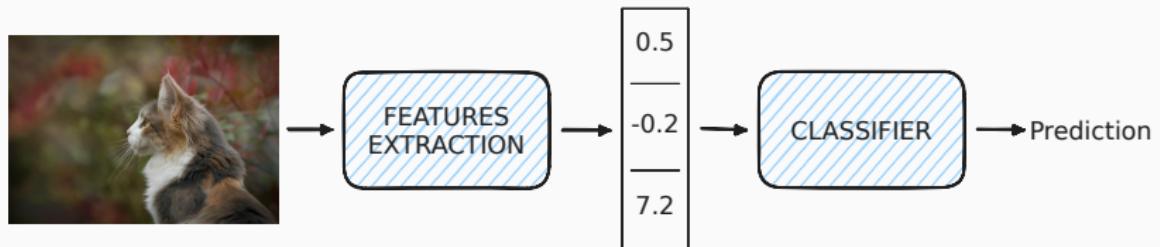


## The extraction and classification pipeline

1. Extract information from data in quantitative format
2. Classify the extracted information

**What kind of extractor? What kind of classifier?**

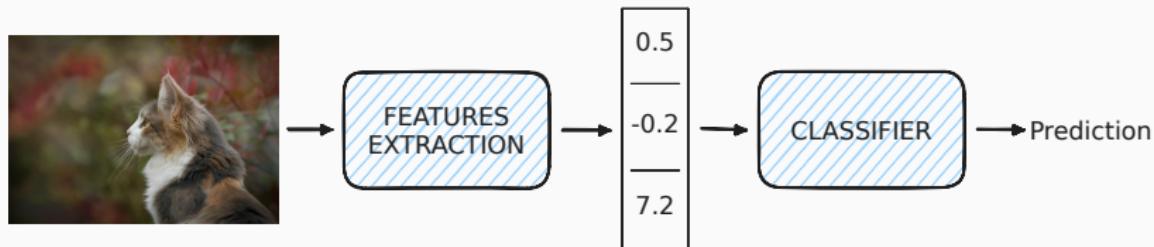
# Before Deep Learning



## Feature extraction:

- Depending on the task/problem
  - **Images:** Saillant point extraction, colors histograms ...
  - **Texts:** Lemmas, stems, PoS (part of speech) ...
- Extract manually important features, automatic design of a function
- Raw data in some cases

# Before Deep Learning

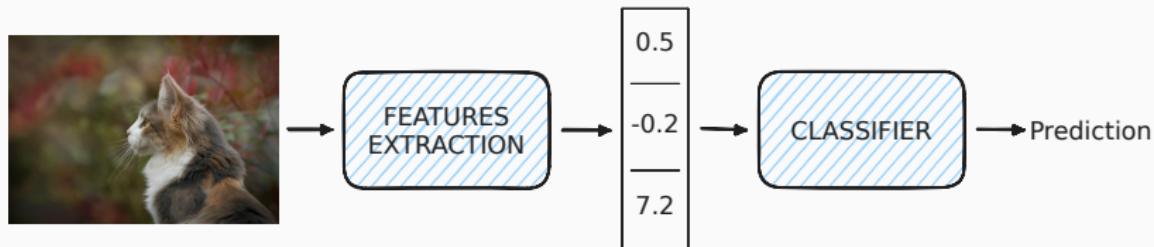


## Feature extraction:

- Depending on the task/problem
  - **Images:** Saillant point extraction, colors histograms ...
  - **Texts:** Lemmas, stems, PoS (part of speech) ...
- Extract manually important features, automatic design of a function
- Raw data in some cases

**Difficulties in retrieving the most adapted features**

# Before Deep Learning



## Classification:

- Decision Tree:
  - Decision considering a limited number of features
  - Use conjunction (path in the tree) to make a prediction
- K-nearest neighbors:
  - Affect the class of closest feature vectors
- Linear classifiers (Linear SVM, perceptron):
  - Weight features so they are more or less important to make a decision

# Notations

## Classifier

Parameterized function  $f_\Theta : \mathcal{X} \rightarrow \mathcal{Y}$

# Notations

## Classifier

Parameterized function  $f_{\Theta} : \mathcal{X} \rightarrow \mathcal{Y}$

- $\Theta$  parameters (often written  $\theta$ )
- $\mathcal{X}$  the input/features space
- $\mathcal{Y}$  the output space/labels space

# Notations

## Classifier

Parameterized function  $f_{\Theta} : \mathcal{X} \rightarrow \mathcal{Y}$

- $\Theta$  parameters (often written  $\theta$ )
- $\mathcal{X}$  the input/features space
- $\mathcal{Y}$  the output space/labels space

## Dataset

$\mathcal{D} \subset \mathcal{X} \rightarrow \mathcal{Y}$  the set containing data

# Notations

## Classifier

Parameterized function  $f_{\Theta} : \mathcal{X} \rightarrow \mathcal{Y}$

- $\Theta$  parameters (often written  $\theta$ )
- $\mathcal{X}$  the input/features space
- $\mathcal{Y}$  the output space/labels space

## Dataset

$\mathcal{D} \subset \mathcal{X} \rightarrow \mathcal{Y}$  the set containing data

- $x^{(i)}$  the  $i^{th}$  input
- $y^{(i)}$  the output/label associate to the  $i^{(th)}$  sample

# Notations

## Classifier

Parameterized function  $f_{\Theta} : \mathcal{X} \rightarrow \mathcal{Y}$

- $\Theta$  parameters (often written  $\theta$ )
- $\mathcal{X}$  the input/features space
- $\mathcal{Y}$  the output space/labels space

## Different type of prediction

## Dataset

$\mathcal{D} \subset \mathcal{X} \rightarrow \mathcal{Y}$  the set containing data

- $x^{(i)}$  the  $i^{th}$  input
- $y^{(i)}$  the output/label associate to the  $i^{(th)}$  sample

# Notations

## Classifier

Parameterized function  $f_{\Theta} : \mathcal{X} \rightarrow \mathcal{Y}$

- $\Theta$  parameters (often written  $\theta$ )
- $\mathcal{X}$  the input/features space
- $\mathcal{Y}$  the output space/labels space

## Dataset

$\mathcal{D} \subset \mathcal{X} \rightarrow \mathcal{Y}$  the set containing data

- $x^{(i)}$  the  $i^{th}$  input
- $y^{(i)}$  the output/label associate to the  $i^{(th)}$  sample

## Different type of prediction

- Regression (predict a value)

# Notations

## Classifier

Parameterized function  $f_{\Theta} : \mathcal{X} \rightarrow \mathcal{Y}$

- $\Theta$  parameters (often written  $\theta$ )
- $\mathcal{X}$  the input/features space
- $\mathcal{Y}$  the output space/labels space

## Dataset

$\mathcal{D} \subset \mathcal{X} \rightarrow \mathcal{Y}$  the set containing data

- $x^{(i)}$  the  $i^{th}$  input
- $y^{(i)}$  the output/label associate to the  $i^{(th)}$  sample

## Different type of prediction

- Regression (predict a value)
- Binary classification (predict class among two)

# Notations

## Classifier

Parameterized function  $f_{\Theta} : \mathcal{X} \rightarrow \mathcal{Y}$

- $\Theta$  parameters (often written  $\theta$ )
- $\mathcal{X}$  the input/features space
- $\mathcal{Y}$  the output space/labels space

## Dataset

$\mathcal{D} \subset \mathcal{X} \rightarrow \mathcal{Y}$  the set containing data

- $x^{(i)}$  the  $i^{th}$  input
- $y^{(i)}$  the output/label associate to the  $i^{(th)}$  sample

## Different type of prediction

- Regression (predict a value)
- Binary classification (predict class among two)
- Multiclass classification (predict a class among many)

# Notations

## Classifier

Parameterized function  $f_{\Theta} : \mathcal{X} \rightarrow \mathcal{Y}$

- $\Theta$  parameters (often written  $\theta$ )
- $\mathcal{X}$  the input/features space
- $\mathcal{Y}$  the output space/labels space

## Dataset

$\mathcal{D} \subset \mathcal{X} \rightarrow \mathcal{Y}$  the set containing data

- $x^{(i)}$  the  $i^{th}$  input
- $y^{(i)}$  the output/label associate to the  $i^{(th)}$  sample

## Different type of prediction

- Regression (predict a value)
- Binary classification (predict class among two)
- Multiclass classification (predict a class among many)
- Multiclass Multi-Label (predict many classes among many)

# Different type of prediction: Regression

## Regression

Predict a value considering input

→ Determine the house price considering a surface



# Different type of prediction: Binary classification

## Binary Classification

Predict a class among two

→ Sentiment analysis (binary case)

Stupid storm. No river thats so cool  
for us tonight



NEGATIVE



POSITIVE

# Different type of prediction: Multiclass classification

## Multiclass Classification

Predict a class among many

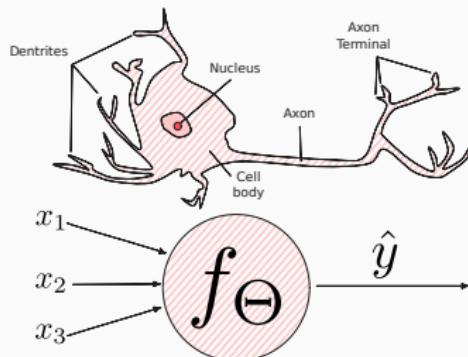
→ Determine the house price considering a surface

Madame Roland was a French revolutionary

NN            VB      DT      ADJ            NN

## Inspiration or the perceptron

- A (brain) neuron receives signals
- Depending of the signals it trigger or not an output



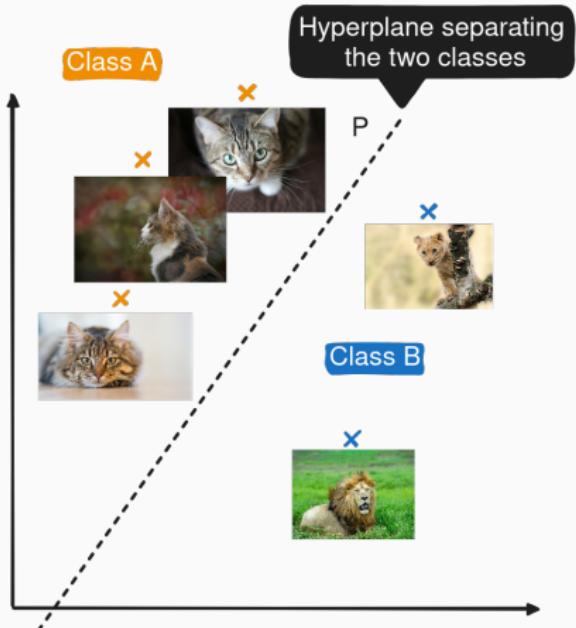
## Bio-inspired classifier

- Perceptron/neural network neurons have most probably been inspired from biology
- BUT they are different and work differently
- **The biology comparison is misleading**

---

<sup>0</sup>picture: <https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network>

# Linear classification: Introduction (or reminder)



**Binary linear classification:**

**Objective:**

Finding a hyperplane separating classes

- An hyperplane  $\mathcal{H}$
- One hyperplane side  $\rightarrow$  one class

## Linear classification: Introduction (or reminder)

Let  $a \in \mathbb{R}^n$  and  $x \in \mathbb{R}^n$  two vectors and  $b \in \mathbb{R}$  a scalar:

## Linear classification: Introduction (or reminder)

Let  $a \in \mathbb{R}^n$  and  $x \in \mathbb{R}^n$  two vectors and  $b \in \mathbb{R}$  a scalar:

$$a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

## Linear classification: Introduction (or reminder)

Let  $a \in \mathbb{R}^n$  and  $x \in \mathbb{R}^n$  two vectors and  $b \in \mathbb{R}$  a scalar:

$$a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \quad a^T = \begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix}$$

## Linear classification: Introduction (or reminder)

Let  $a \in \mathbb{R}^n$  and  $x \in \mathbb{R}^n$  two vectors and  $b \in \mathbb{R}$  a scalar:

$$a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \quad a^T = \begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

## Linear classification: Introduction (or reminder)

Let  $a \in \mathbb{R}^n$  and  $x \in \mathbb{R}^n$  two vectors and  $b \in \mathbb{R}$  a scalar:

$$a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \quad a^T = \begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

$a$  and  $b$  define the hyperplane such that  $x$  belong to  $\mathcal{H}$  iff :

$$\sum_{i=1}^n a_i x_i + b = 0 \iff a^T x + b = 0 \iff \langle a, x \rangle + b = 0$$

## Linear classification: Introduction (or reminder)

Let  $a \in \mathbb{R}^n$  and  $x \in \mathbb{R}^n$  two vectors and  $b \in \mathbb{R}$  a scalar:

$$a = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \quad a^T = \begin{pmatrix} a_1 & a_2 & \dots & a_n \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

$a$  and  $b$  define the hyperplane such that  $x$  belong to  $\mathcal{H}$  iff :

$$\sum_{i=1}^n a_i x_i + b = 0 \iff a^T x + b = 0 \iff \langle a, x \rangle + b = 0$$

### Dot product properties:

→  $a^T x$  can be written as the product of magnitude of vectors and the cosinus of their angle

$$a^T x = \|a\| \|x\| \cos(\theta)$$

With  $\|a\|_2 = \sqrt{\sum_{i=1}^n a_i^2}$

## Linear classification: The dot product

$$\rightarrow \mathbf{a}^T \mathbf{x} = \|\mathbf{a}\| \|\mathbf{x}\| \cos(\theta)$$

$$\rightarrow \|\mathbf{a}\|_2 \geq 0$$

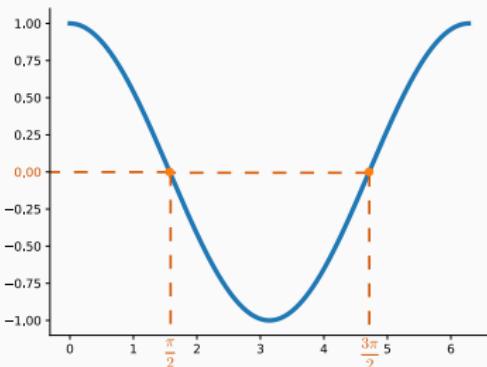
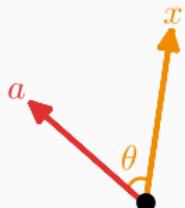


Figure 1: The cosinus function for  $x \in [0, 2\pi]$



- $\theta \in ]0, \frac{\pi}{2}[$ . What is the value of  $\mathbf{a}^T \mathbf{x}$  ?

## Linear classification: The dot product

$$\rightarrow \mathbf{a}^T \mathbf{x} = \|\mathbf{a}\| \|\mathbf{x}\| \cos(\theta)$$

$$\rightarrow \|\mathbf{a}\|_2 \geq 0$$

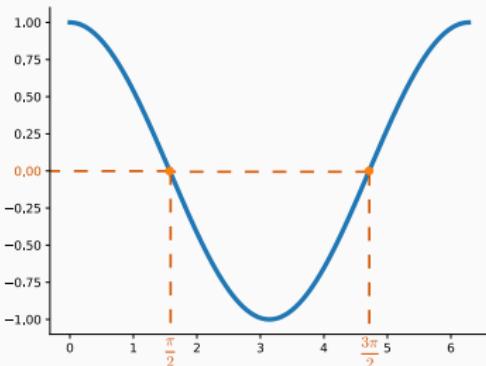
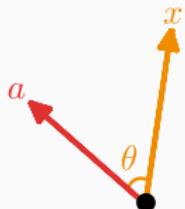


Figure 1: The cosinus function for  $x \in [0, 2\pi]$



- $\theta \in ]0, \frac{\pi}{2}[$ . What is the value of  $\mathbf{a}^T \mathbf{x}$  ?  
 $\rightarrow \mathbf{a}^T \mathbf{x} > 0$

## Linear classification: The dot product

$$\rightarrow a^T x = \|a\| \|x\| \cos(\theta)$$

$$\rightarrow \|a\|_2 \geq 0$$

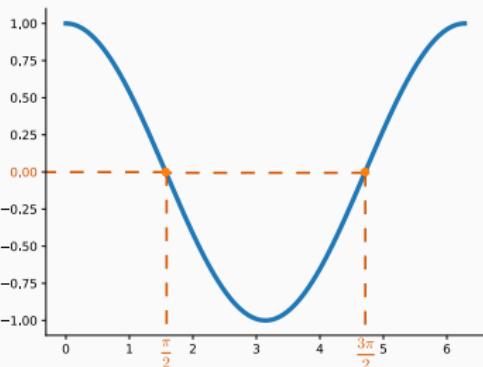
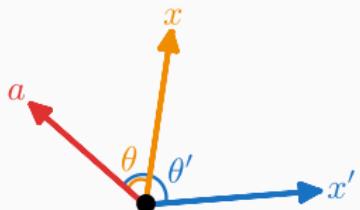


Figure 2: The cosinus function for  $x \in [0, 2\pi]$



- $\theta \in ]0, \frac{\pi}{2}[$ . What is the value of  $a^T x$ ?  
 $\rightarrow a^T x > 0$
- $\theta \in ]\frac{\pi}{2}, \frac{3\pi}{2}[$ . What is the value of  $a^T x$ ?

## Linear classification: The dot product

$$\rightarrow a^T x = \|a\| \|x\| \cos(\theta)$$

$$\rightarrow \|a\|_2 \geq 0$$

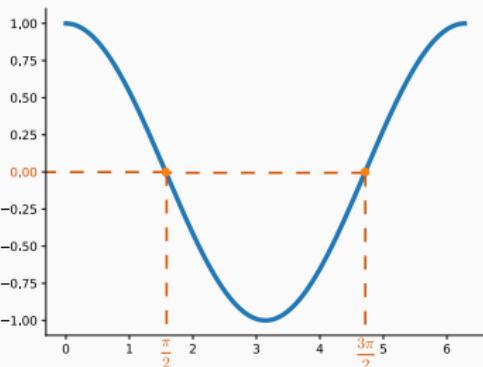
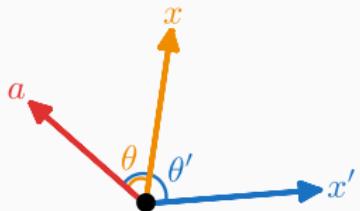


Figure 2: The cosinus function for  $x \in [0, 2\pi]$



- $\theta \in ]0, \frac{\pi}{2}[$ . What is the value of  $a^T x$ ?  
→  $a^T x > 0$
- $\theta \in ]\frac{\pi}{2}, \frac{3\pi}{2}[$ . What is the value of  $a^T x$ ?  
→  $a^T x < 0$

# Binary Linear Classification

## Classification function:

Let  $\theta$  the parameters

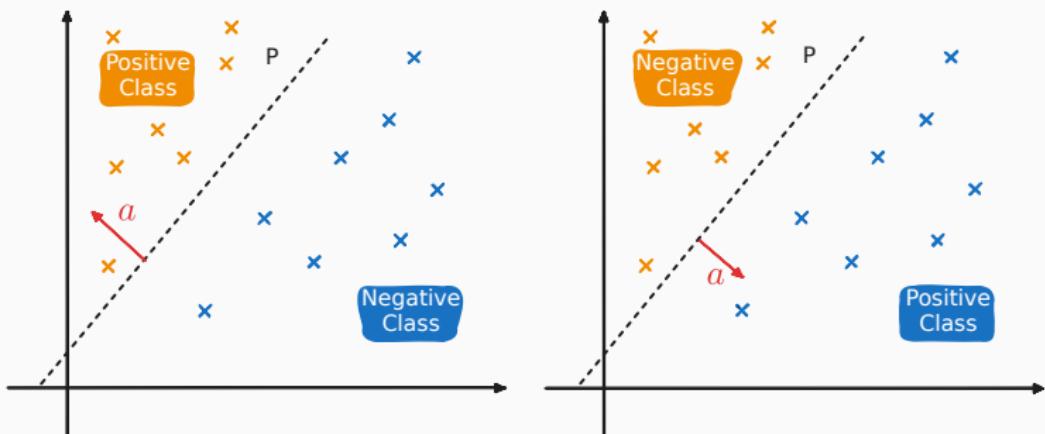
- **General Case:**  $f_{\theta} : \mathcal{X} \rightarrow \mathcal{Y}$  with  $\mathcal{Y}$  the class set
- **Binary case:**  $f_{\theta} : \mathcal{X} \rightarrow \{-1, 1\}$

## Perceptron

- $\theta = \{a, b\}$  the parameters

- Classification function:

$$f_{\theta}(x) = \begin{cases} 1 & \text{if } a^T x + b > 0 \\ -1 & \text{if } a^T x + b \leq 0 \end{cases}$$



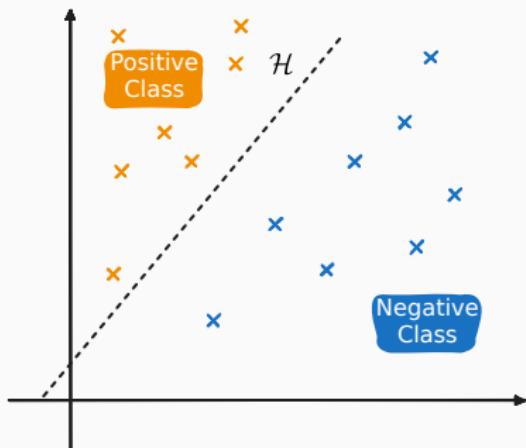
# Binary Linear Classification

## Perceptron Summary

$$f_{\Theta}(x) = \begin{cases} 1 & \text{if } a^T x + b > 0 \\ -1 & \text{if } a^T x + b \leq 0 \end{cases}$$

- $\Theta = \{a, b\}$  the parameters
- Decision boundary is defined by an hyperplane  $\mathcal{H}$  such that:

$$a^T + b = 0$$



## Remaining questions

- How to find the hyperplane  $\mathcal{H}$ , i.e. how do we retrieve parameters  $\theta$ ?
- What kind of problems can be solved finding a hyperplane?

# Binary Linear Classification: Retrieving Hyperplane

## Objective

Finding  $\mathcal{H}$  parametrized by  $\theta = \{a\}$  (let not considering bias) such that:

$$\begin{cases} a^T x^{(i)} > 0 & \text{if } y^{(i)} = 1 \\ a^T x^{(i)} \leq 0 & \text{if } y^{(i)} = -1 \end{cases}$$

# Binary Linear Classification: Retrieving Hyperplane

## Objective

Finding  $\mathcal{H}$  parametrized by  $\theta = \{a\}$  (let not considering bias) such that:

$$\begin{cases} a^T x^{(i)} > 0 & \text{if } y^{(i)} = 1 \\ a^T x^{(i)} \leq 0 & \text{if } y^{(i)} = -1 \end{cases}$$

## A simple loss function

- if  $a^T x^{(i)}$  and  $y^i$  have **same** sign then  $(a^T x^{(i)})y^i \Rightarrow 0$  (correct)
- if  $a^T x^{(i)}$  and  $y^i$  have **different** sign then  $(a^T x^{(i)})y^i \leq 0$  (incorrect)

# Binary Linear Classification: Retrieving Hyperplane

## Objective

Finding  $\mathcal{H}$  parametrized by  $\theta = \{a\}$  (let not considering bias) such that:

$$\begin{cases} a^T x^{(i)} > 0 & \text{if } y^{(i)} = 1 \\ a^T x^{(i)} \leq 0 & \text{if } y^{(i)} = -1 \end{cases}$$

## A simple loss function

- if  $a^T x^{(i)}$  and  $y^i$  have **same** sign then  $(a^T x^{(i)})y^i \Rightarrow 0$  (correct)
- if  $a^T x^{(i)}$  and  $y^i$  have **different** sign then  $(a^T x^{(i)})y^i \leq 0$  (incorrect)

We can define a loss (error function):

$$l(f_\Theta(x^{(i)}), y^i) = \max(0, (-a^T x^{(i)})y^i)$$

# Binary Linear Classification: Retrieving Hyperplane

Finding a hyperplane

## Binary Linear Classification: Retrieving Hyperplane

### Finding a hyperplane

Let consider  $a^{(t)}$  parameter (at time t)

and  $\lambda = 1$

# Binary Linear Classification: Retrieving Hyperplane

## Finding a hyperplane

Let consider  $a^{(t)}$  parameter (at time t)

and  $\lambda = 1$

- Find an example  $x^{(i)}$  with:

$$l(f_\Theta(x^{(i)}), y^i) > 0$$

# Binary Linear Classification: Retrieving Hyperplane

## Finding a hyperplane

Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = 1$

- Find an example  $x^{(i)}$  with:

$$l(f_\Theta(x^{(i)}), y^i) > 0$$

- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$

# Binary Linear Classification: Retrieving Hyperplane

## Finding a hyperplane

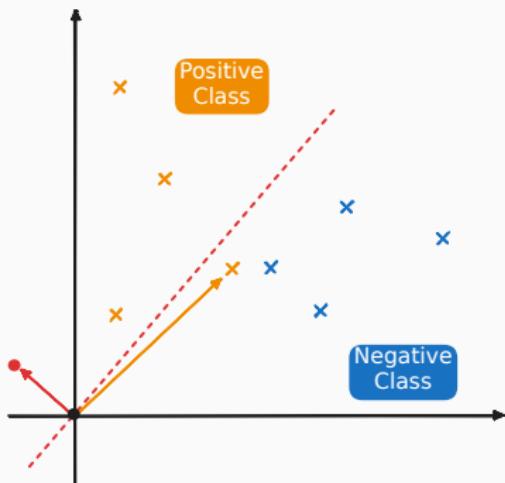
Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = 1$

- Find an example  $x^{(i)}$  with:

$$I(f_\Theta(x^{(i)}), y^{(i)}) > 0$$

- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



Repeat until convergence (or maximum iteration)

# Binary Linear Classification: Retrieving Hyperplane

## Finding a hyperplane

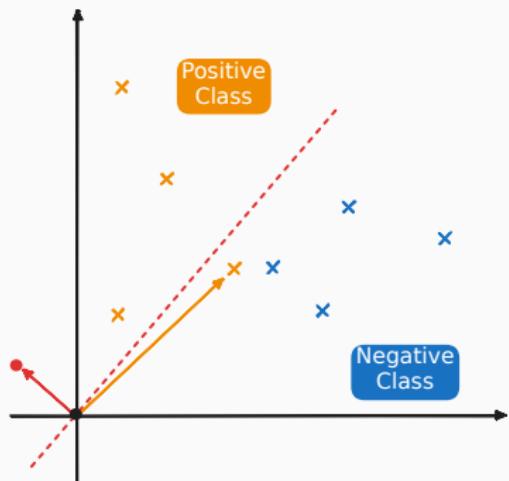
Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = 1$

- Find an example  $x^{(i)}$  with:

$$I(f_{\Theta}(x^{(i)}), y^{(i)}) > 0$$

- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



Repeat until convergence (or maximum iteration)

# Binary Linear Classification: Retrieving Hyperplane

## Finding a hyperplane

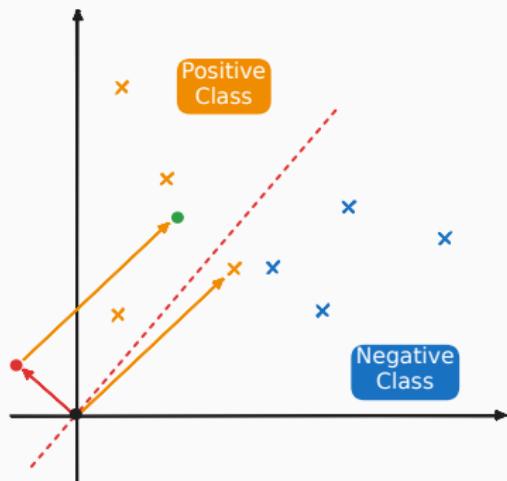
Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = 1$

- Find an example  $x^{(i)}$  with:

$$I(f_{\Theta}(x^{(i)}), y^{(i)}) > 0$$

- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



Repeat until convergence (or maximum iteration)

# Binary Linear Classification: Retrieving Hyperplane

## Finding a hyperplane

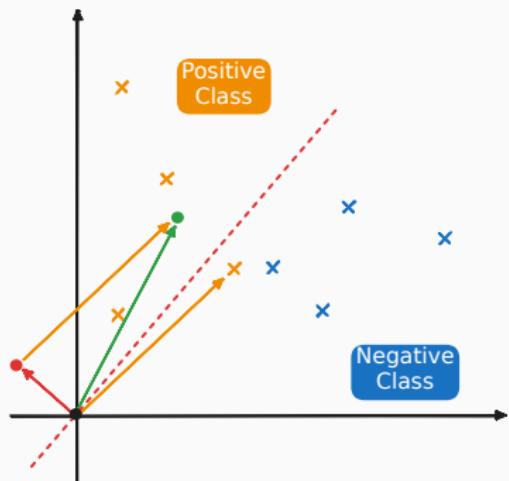
Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = 1$

- Find an example  $x^{(i)}$  with:

$$I(f_{\Theta}(x^{(i)}), y^{(i)}) > 0$$

- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



Repeat until convergence (or maximum iteration)

# Binary Linear Classification: Retrieving Hyperplane

## Finding a hyperplane

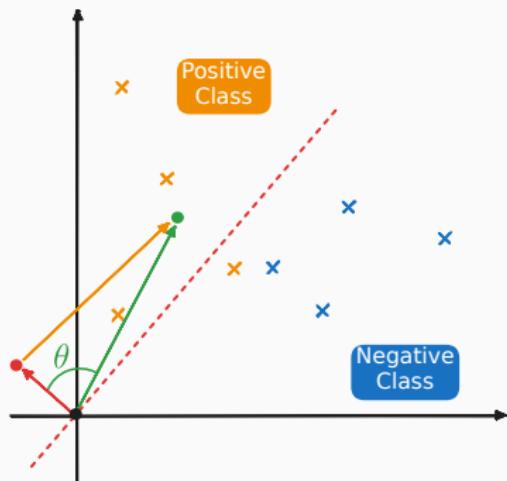
Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = 1$

- Find an example  $x^{(i)}$  with:

$$I(f_{\Theta}(x^{(i)}), y^{(i)}) > 0$$

- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



Repeat until convergence (or maximum iteration)

# Binary Linear Classification: Retrieving Hyperplane

## Finding a hyperplane

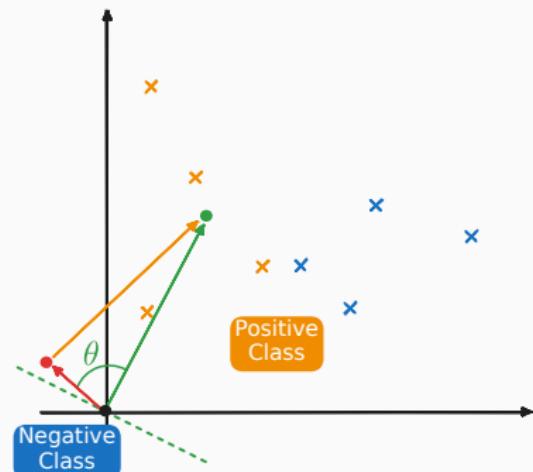
Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = 1$

- Find an example  $x^{(i)}$  with:

$$I(f_{\Theta}(x^{(i)}), y^{(i)}) > 0$$

- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



Repeat until convergence (or maximum iteration)

# Binary Linear Classification: Retrieving Hyperplane

## Finding a Hyperplane

Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = 1$

- Find an example  $x^{(i)}$  with  
 $I(f_\Theta(x^{(i)}), y^{(i)}) > 0$
- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



Repeat until convergence (or maximum iteration)

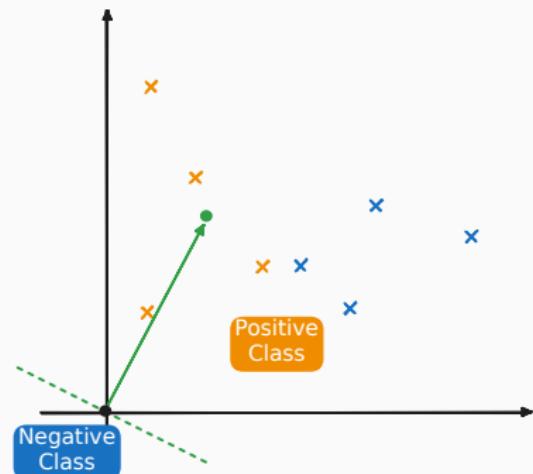
# Binary Linear Classification: Retrieving Hyperplane

## Finding a Hyperplane

Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = 1$

- Find an example  $x^{(i)}$  with  
 $I(f_\Theta(x^{(i)}), y^{(i)}) > 0$
- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



Repeat until convergence (or maximum iteration)

→ The correction is too large

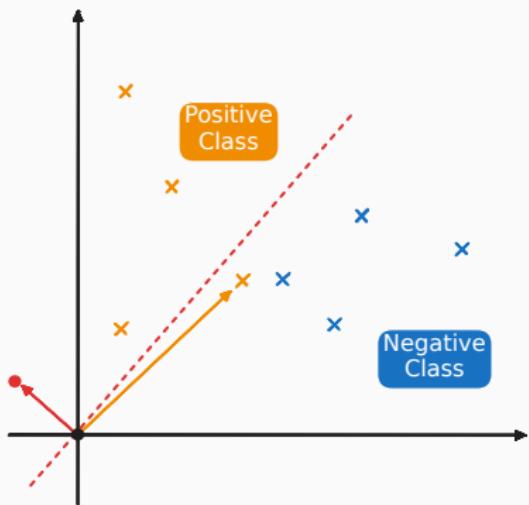
## Binary Linear Classification: Retrieving Hyperplane

### Finding a Hyperplane

Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = \epsilon$  (small)

- Find an example  $x^{(i)}$  with  
 $I(f_\Theta(x^{(i)}), y^i) > 0$
- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



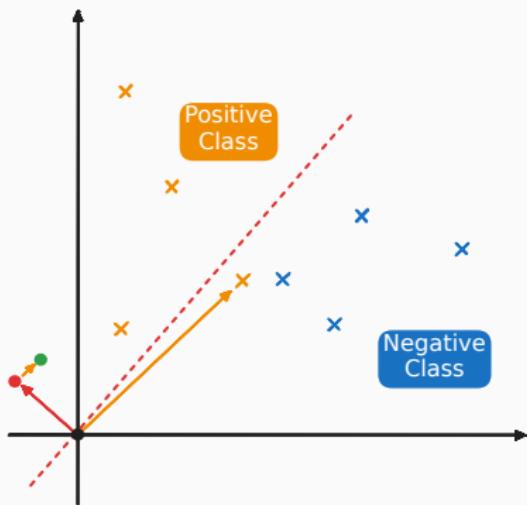
# Binary Linear Classification: Retrieving Hyperplane

## Finding a Hyperplane

Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = \epsilon$  (small)

- Find an example  $x^{(i)}$  with  
 $I(f_\Theta(x^{(i)}), y^i) > 0$
- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



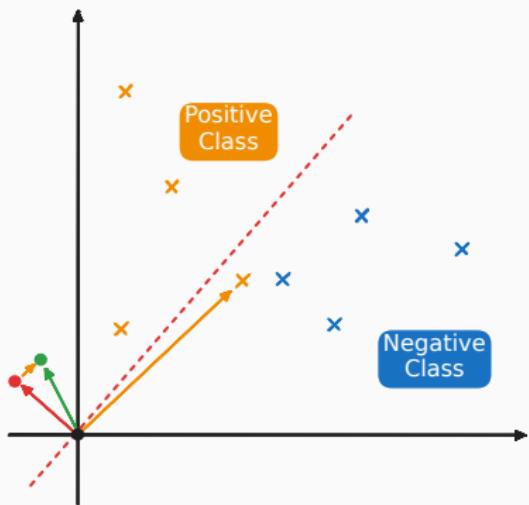
# Binary Linear Classification: Retrieving Hyperplane

## Finding a Hyperplane

Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = \epsilon$  (small)

- Find an example  $x^{(i)}$  with  
 $I(f_\Theta(x^{(i)}), y^i) > 0$
- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



# Binary Linear Classification: Retrieving Hyperplane

## Finding a Hyperplane

Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = \epsilon$  (small)

- Find an example  $x^{(i)}$  with  
 $I(f_\Theta(x^{(i)}), y^i) > 0$
- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



## Binary Linear Classification: Retrieving Hyperplane

### Finding a Hyperplane

Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = \epsilon$  (small)

- Find an example  $x^{(i)}$  with  
 $I(f_\Theta(x^{(i)}), y^i) > 0$
- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



## Binary Linear Classification: Retrieving Hyperplane

### Finding a Hyperplane

Let consider  $a^{(t)}$  parameter (at time t)  
and  $\lambda = \epsilon$  (small)

- Find an example  $x^{(i)}$  with  
 $I(f_\Theta(x^{(i)}), y^i) > 0$
- Update  $a^{(t)}$  with the following rule:

$$a^{(t+1)} = a^{(t)} + \lambda x^{(i)} y^{(i)}$$



# Binary Linear Classification: Retrieving Hyperplane

## Exercise 1:

Let the dataset and the classifier

$$\begin{array}{c} & x_1^{(i)} & x_2^{(i)} \\ \hline x^{(1)} & 2 & 2 \\ x^{(2)} & 2 & 5 \\ x^{(3)} & 4 & 3 \end{array} \quad \Theta = \left( \begin{pmatrix} -1 \\ 2 \end{pmatrix}, 3 \right)$$

What is the number of error of the function ?

## Exercise 3:

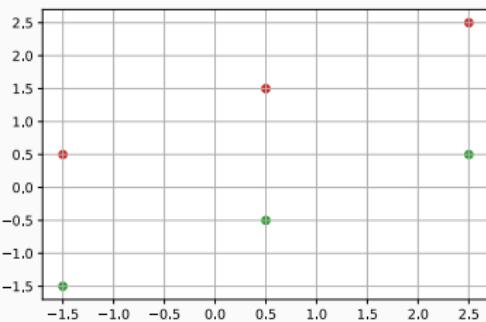
Let the dataset and the classifier

$$\begin{array}{c} & x_1^{(i)} & x_2^{(i)} & y^{(i)} \\ \hline x^{(1)} & 0 & 0 & -1 \\ x^{(2)} & -5 & 5 & 1 \\ x^{(3)} & 4 & 3 & -1 \\ x^{(4)} & 1 & 3 & -1 \end{array} \quad \Theta = \left( \begin{pmatrix} -1 \\ -1 \end{pmatrix}, 1 \right)$$

Determine the number of errors

## Exercise 2:

Find the parameters  $\Theta$  for the following points



## Binary Linear Classification: Retrieving Hyperplane(end)

Let compute the gradient  $\nabla_{a^{(t)}} \left( (-a^T x^{(i)}) y^{(i)} \right)$  where  $a, x^{(i)} \in \mathbb{R}^2$  and  $y^{(i)} \in \mathbb{R}$

## Binary Linear Classification: Retrieving Hyperplane(end)

Let compute the gradient  $\nabla_{a^{(t)}} \left( (-a^T x^{(i)}) y^{(i)} \right)$  where  $a, x^{(i)} \in \mathbb{R}^2$  and  $y^{(i)} \in \mathbb{R}$

$$\begin{aligned} (-a^T x^{(i)}) y^{(i)} &= \begin{pmatrix} -a_1 & -a_2 \end{pmatrix} \begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \end{pmatrix} y^{(i)} \\ &= -a_1 x_1^{(i)} y^{(i)} - a_2 x_2^{(i)} y^{(i)} \end{aligned}$$

## Binary Linear Classification: Retrieving Hyperplane(end)

Let compute the gradient  $\nabla_{a^{(t)}} \left( (-a^T x^{(i)}) y^{(i)} \right)$  where  $a, x^{(i)} \in \mathbb{R}^2$  and  $y^{(i)} \in \mathbb{R}$

$$\begin{aligned} (-a^T x^{(i)}) y^{(i)} &= \begin{pmatrix} -a_1 & -a_2 \end{pmatrix} \begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \end{pmatrix} y^{(i)} \\ &= -a_1 x_1^{(i)} y^{(i)} - a_2 x_2^{(i)} y^{(i)} \end{aligned}$$

$$\frac{\partial (-a^T x^{(i)}) y^{(i)}}{\partial a_1} = \frac{-a_1 x_1^{(i)} y^{(i)}}{\partial a_1} \quad (1)$$

$$= -x_1^{(i)} y^{(i)} \quad (2)$$

## Binary Linear Classification: Retrieving Hyperplane(end)

Let compute the gradient  $\nabla_{a^{(t)}} \left( (-a^T x^{(i)}) y^{(i)} \right)$  where  $a, x^{(i)} \in \mathbb{R}^2$  and  $y^{(i)} \in \mathbb{R}$

$$\begin{aligned} (-a^T x^{(i)}) y^{(i)} &= \begin{pmatrix} -a_1 & -a_2 \end{pmatrix} \begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \end{pmatrix} y^{(i)} \\ &= -a_1 x_1^{(i)} y^{(i)} - a_2 x_2^{(i)} y^{(i)} \end{aligned}$$

$$\frac{\partial(-a^T x^{(i)}) y^{(i)}}{\partial a_1} = \frac{-a_1 x_1^{(i)} y^{(i)}}{\partial a_1} \quad (1)$$

$$= -x_1^{(i)} y^{(i)} \quad (2)$$

$$\frac{\partial(-a^T x^{(i)}) y^{(i)}}{\partial a_2} = \frac{-a_2 x_2^{(i)} y^{(i)}}{\partial a_2} \quad (3)$$

$$= -x_2^{(i)} y^{(i)} \quad (4)$$

## Binary Linear Classification: Retrieving Hyperplane(end)

Let compute the gradient  $\nabla_{a^{(t)}} \left( (-a^\top x^{(i)}) y^{(i)} \right)$  where  $a, x^{(i)} \in \mathbb{R}^2$  and  $y^{(i)} \in \mathbb{R}$

$$\begin{aligned} (-a^\top x^{(i)}) y^{(i)} &= \begin{pmatrix} -a_1 & -a_2 \end{pmatrix} \begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \end{pmatrix} y^{(i)} \\ &= -a_1 x_1^{(i)} y^{(i)} - a_2 x_2^{(i)} y^{(i)} \end{aligned}$$

$$\begin{aligned} \frac{\partial(-a^\top x^{(i)}) y^{(i)}}{\partial a_1} &= \frac{-a_1 x_1^{(i)} y^{(i)}}{\partial a_1} & (1) \\ &= -x_1^{(i)} y^{(i)} & (2) \end{aligned}$$

$$\begin{aligned} \frac{\partial(-a^\top x^{(i)}) y^{(i)}}{\partial a_2} &= \frac{-a_2 x_2^{(i)} y^{(i)}}{\partial a_2} & (3) \\ &= -x_2^{(i)} y^{(i)} & (4) \end{aligned}$$

Thus the hyperplane update is equivalent to:

$$a^{(t+1)} = a^{(t)} - \nabla_{a^{(t)}} \left( (-a^\top x^{(i)}) y^{(i)} \right)$$

## Binary Linear Classification: Retrieving Hyperplane

**Gradient Descent algorithm:**

$$\Theta^{(t+1)} = \Theta^{(t)} - \nabla_{\Theta} (J(x, y))$$

This approach are named gradient descent algorithms

→ We will study them later

# Binary Linear Classification: Retrieving Hyperplane

**Gradient Descent algorithm:**

$$\Theta^{(t+1)} = \Theta^{(t)} - \nabla_{\Theta} (I(x, y))$$

This approach are named gradient descent algorithms

→ We will study them later

**Remaining questions**

- How to find the hyperplane  $\mathcal{H}$ , i.e. how do we retrieve parameters  $\Theta$ ?
- What kind of problems can be solved finding a hyperplane?

## Linear classification: limits

**Gradient Descent algorithm:**

$$\Theta^{(t+1)} = \Theta^{(t)} - \nabla_{\Theta} (J(x, y))$$

This approach are named gradient descent algorithms

→ We will study them later

## Gradient Descent algorithm:

$$\Theta^{(t+1)} = \Theta^{(t)} - \nabla_{\Theta} (J(x, y))$$

This approach are named gradient descent algorithms

→ We will study them later

## Remaining questions

- How to find the hyperplane  $\mathcal{H}$ , i.e. how do we retrieve parameters  $\Theta$ ?
- What kind of problems can be solved finding a hyperplane?

# Linear classification: limits

**Gradient Descent algorithm:**

$$\Theta^{(t+1)} = \Theta^{(t)} - \nabla_{\Theta} (J(x, y))$$

This approach are named gradient descent algorithms

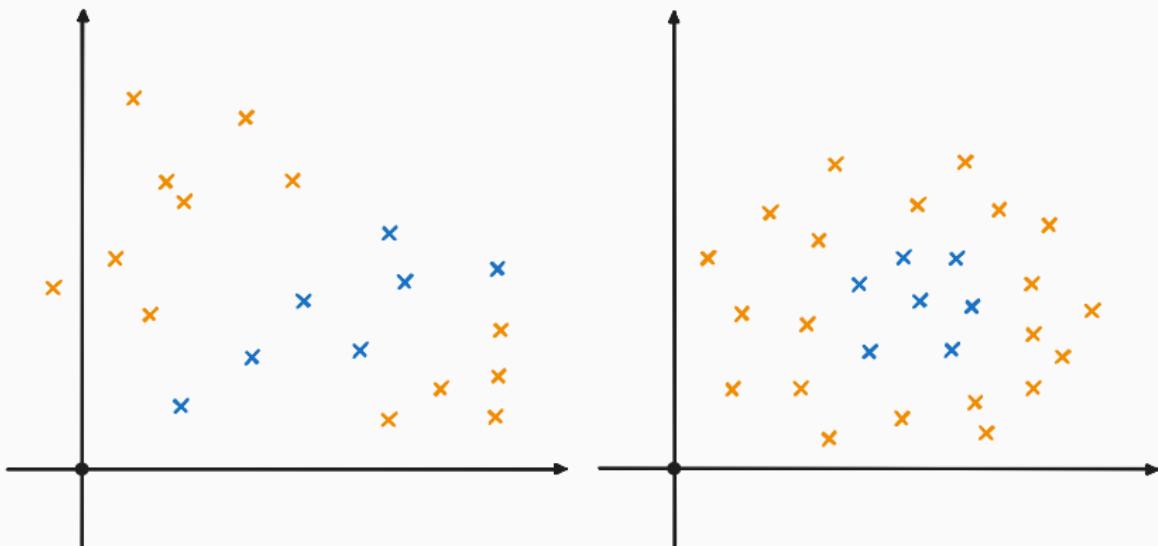
→ We will study them later

**Remaining questions**

- How to find the hyperplane  $\mathcal{H}$ , i.e. how do we retrieve parameters  $\Theta$ ?
- What kind of problems can be solved finding a hyperplane?
  - Can we always find an hyperplane separating classes?
  - Can we formally define **linearly separable** problems?

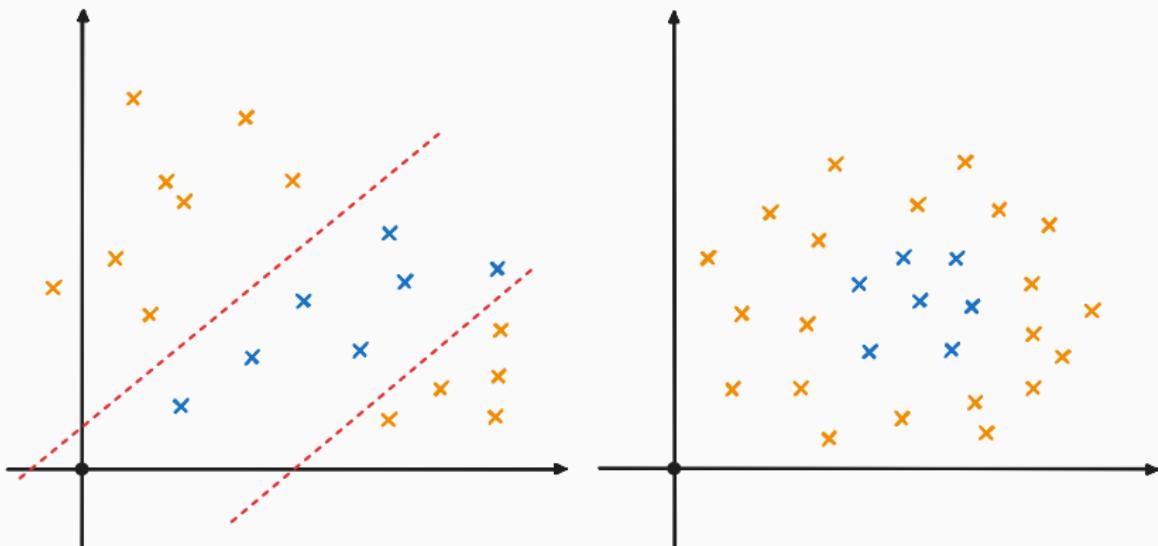
## Linear classification: limits

- Can we always find an hyperplane separating classes?
- Can we formally define **linearly separable** problems?



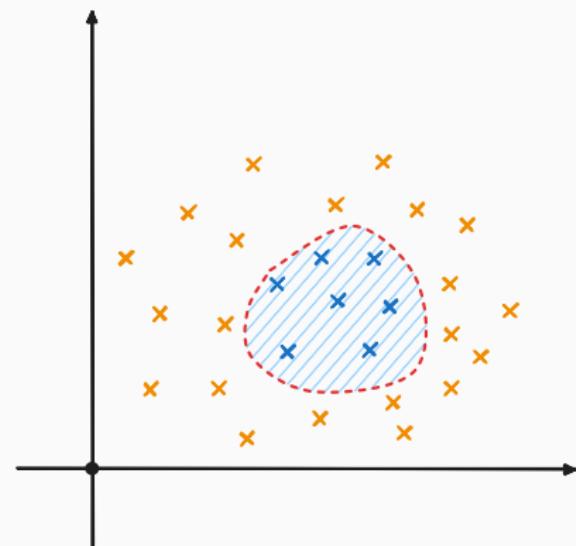
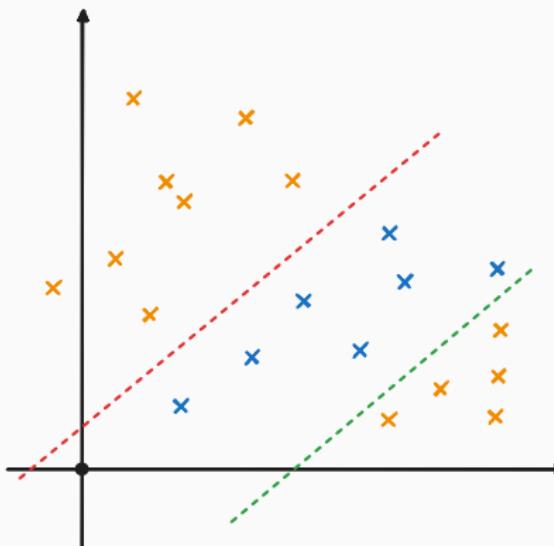
## Linear classification: limits

- Can we always find an hyperplane separating classes?
- Can we formally define **linearly separable** problems?



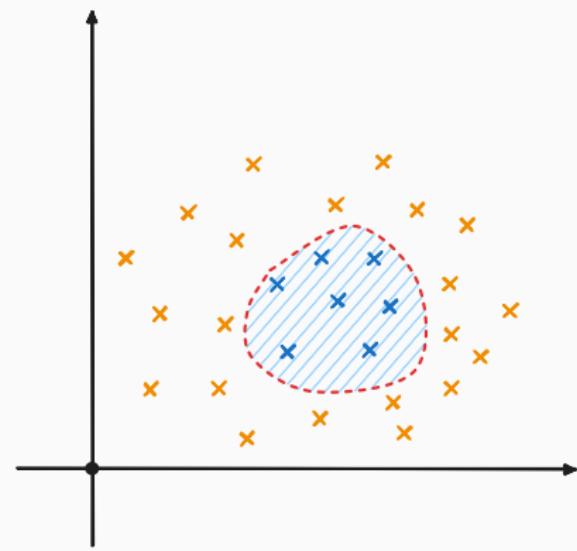
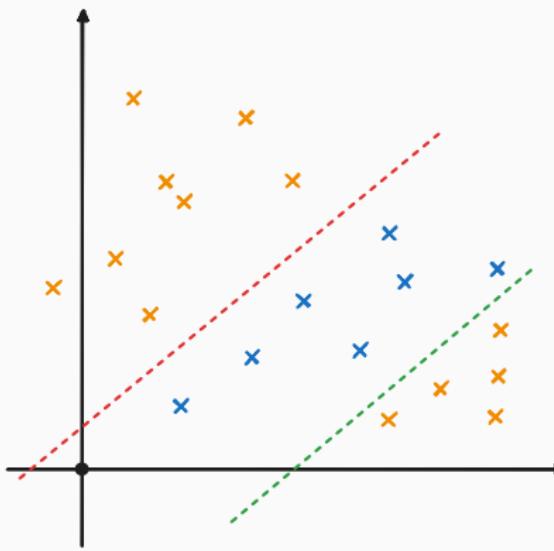
## Linear classification: limits

- Can we always find an hyperplane separating classes?
- Can we formally define **linearly separable** problems?



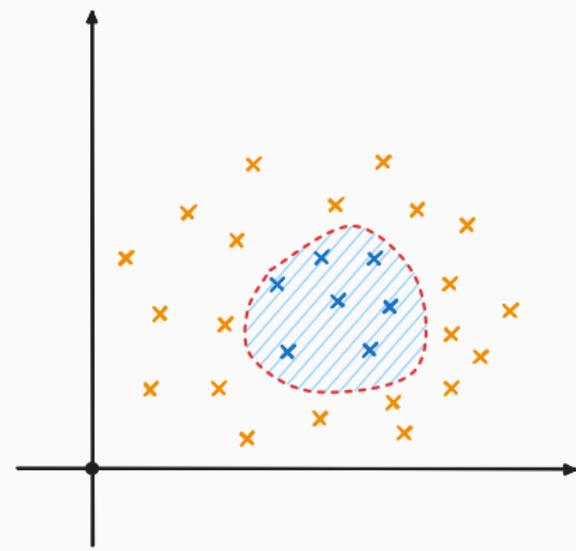
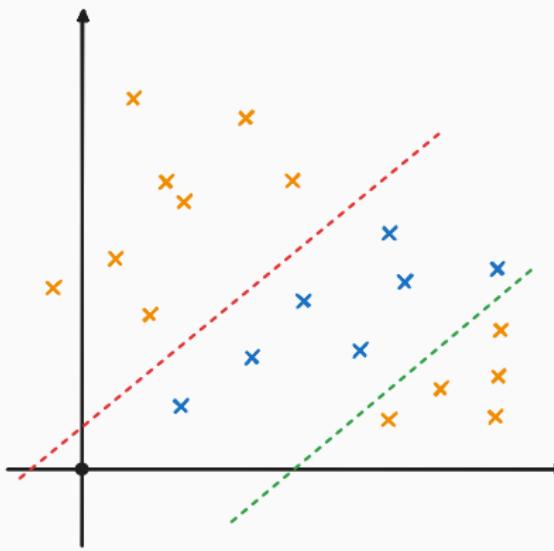
## Linear classification: limits

- Can we always find an hyperplane separating classes? **NO**
- Can we formally define **linearly separable** problems?



## Linear classification: limits

- Can we always find an hyperplane separating classes? **NO**
- Can we formally define **linearly separable** problems? **YES**



# Convex Set

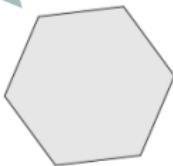
## Definition: Convex Set

Let  $C \in \mathbb{R}^n$ .  $C$  is convex iff :

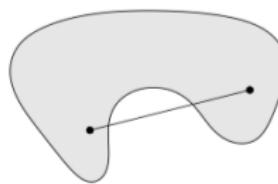
$$\forall x, y \in C, \forall \epsilon \in [0, 1] : \epsilon x + (1 - \epsilon)y \in C$$

Or  $C$  is convex if every point on the line segment connecting  $x$  and  $y$  other than the endpoints is in  $C$

Convex set



Non-convex set



<sup>0</sup>Picture from Convex Optimization, Boyd and Vandenberghe

# Convex Set

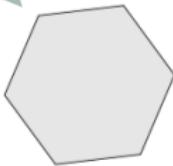
## Definition: Convex Set

Let  $C \in \mathbb{R}^n$ .  $C$  is convex iff :

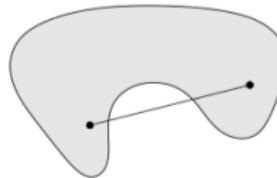
$$\forall x, y \in C, \forall \epsilon \in [0, 1] : \epsilon x + (1 - \epsilon)y \in C$$

Or  $C$  is convex if every point on the line segment connecting  $x$  and  $y$  other than the endpoints is in  $C$

Convex set



Non-convex set



Can we define a convex set from the data points?

<sup>0</sup>Picture from Convex Optimization, Boyd and Vandenberghe

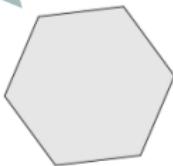
## Definition: Convex Set

Let  $C \in \mathbb{R}^n$ .  $C$  is convex iff :

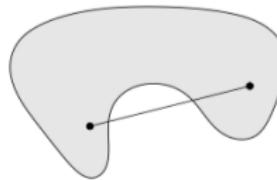
$$\forall x, y \in C, \forall \epsilon \in [0, 1] : \epsilon x + (1 - \epsilon)y \in C$$

Or  $C$  is convex if every point on the line segment connecting  $x$  and  $y$  other than the endpoints is in  $C$

Convex set



Non-convex set

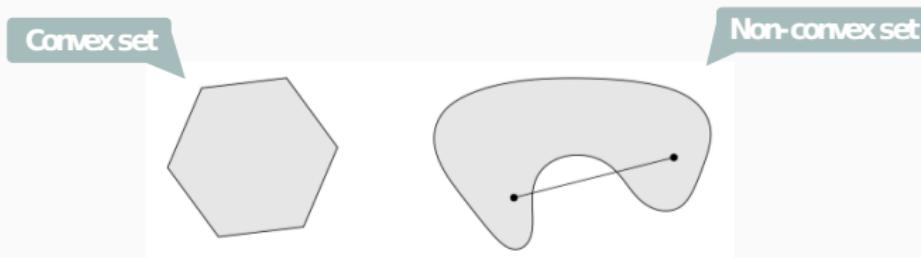


Can we define a convex set from the data points? Yes

<sup>0</sup>Picture from Convex Optimization, Boyd and Vandenberghe

## Definition: Convex Hull

The **convex hull**  $S$  of a set  $C \in \mathbb{R}^n$  is the set of all convex combination of points in  $C$ :  $S = \{\epsilon_1 x^{(1)} + \epsilon_2 x^{(2)} + \dots + \epsilon_K x^{(K)} \mid x^{(i)} \in C; \epsilon_i \geq 0; \sum_{i=1}^K \epsilon_i = 1\}$



<sup>0</sup>Picture from Convex Optimization, Boyd and Vandenberghe

# Linearly Separable

## Theorem

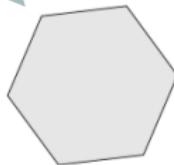
Let  $C \in \mathbb{R}^n$  and  $D \in \mathbb{R}^n$  be two convex sets. If  $C \cap D = \emptyset$  (empty intersection) then it exists an hyperplane such that:

$$\forall x \in C; a^T x + b \geq 0$$

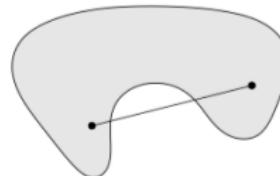
$$\forall x \in D; a^T x + b \leq 0$$

With  $a$  and  $b$  parameters of the hyperplane

Convex set



Non-convex set



<sup>0</sup>Picture from Convex Optimization, Boyd and Vandenberghe

# Parameter of the Hyperplane

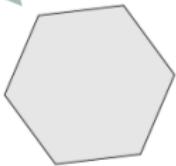
## Proposition

Let  $C \in \mathbb{R}^n$  and  $D \in \mathbb{R}^n$  be two convex sets and  $C \cap D = \emptyset$ . If  $c \in C$  and  $d \in D$  such that  $\|c - d\| = \text{dist}(C, D)$  the hyperplane defined by :

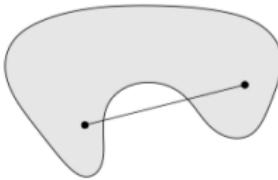
$$a = d - c \qquad b = -\frac{\|d\|^2 + \|c\|^2}{2}$$

for proof see "Convex Optimization" (Boyd and Vandenberghe) section 2.5.1.

Convex set



Non-convex set



# Parameter of the Hyperplane

## Proposition

Let  $C \in \mathbb{R}^n$  and  $D \in \mathbb{R}^n$  be two convex sets and  $C \cap D = \emptyset$ . If  $c \in C$  and  $d \in D$  such that  $\|c - d\| = \text{dist}(C, D)$  the hyperplane defined by :

$$a = d - c \qquad b = -\frac{\|d\|^2 + \|c\|^2}{2}$$

for proof see "Convex Optimization" (Boyd and Vandenberghe) section 2.5.1.

## In practice:

- Data is not linearly separable (i.e no such hyperplane)
- Computing solution (convex hull) can be very expensive

→ Online algorithm preferred (such as the previous one)

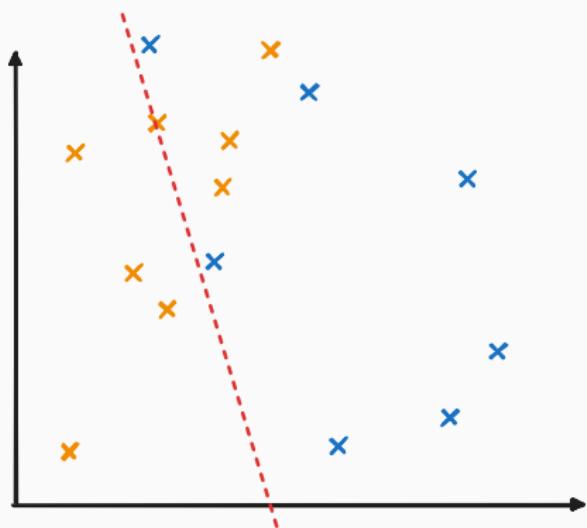
## Non-linearly separable problems

---

# Non linearly separable

Handle non linearity?

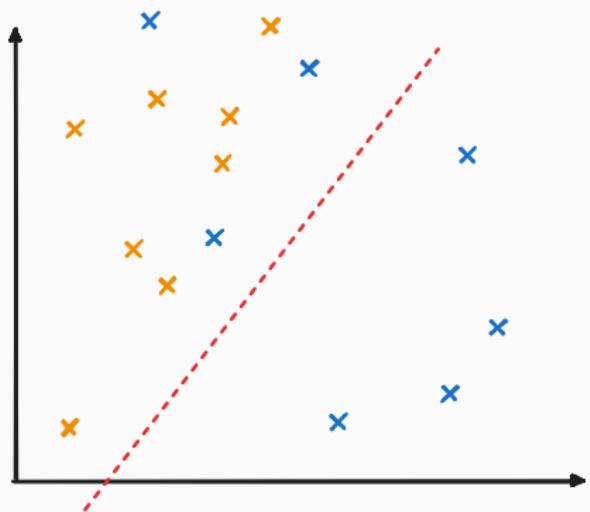
- Linear separation on raw features



# Non linearly separable

Handle non linearity?

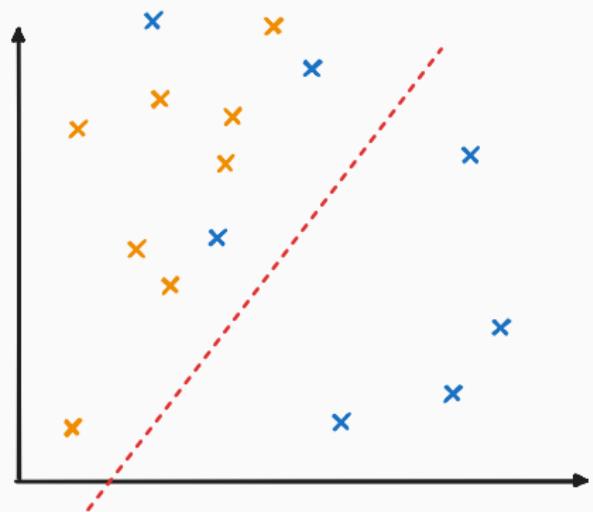
- Linear separation on raw features



# Non linearly separable

Handle non linearity?

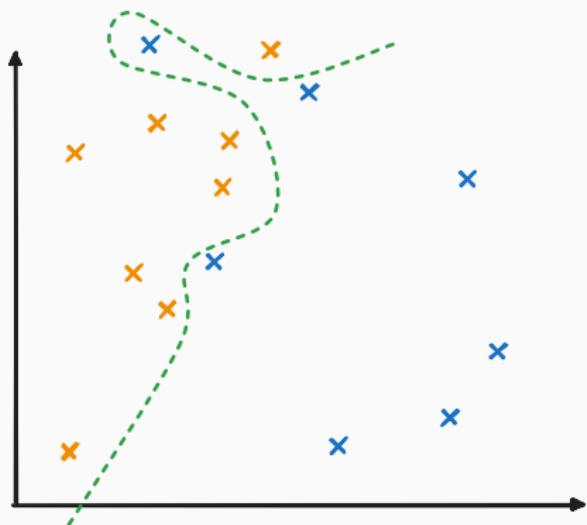
- Linear separation on raw features



# Non linearly separable

Handle non linearity?

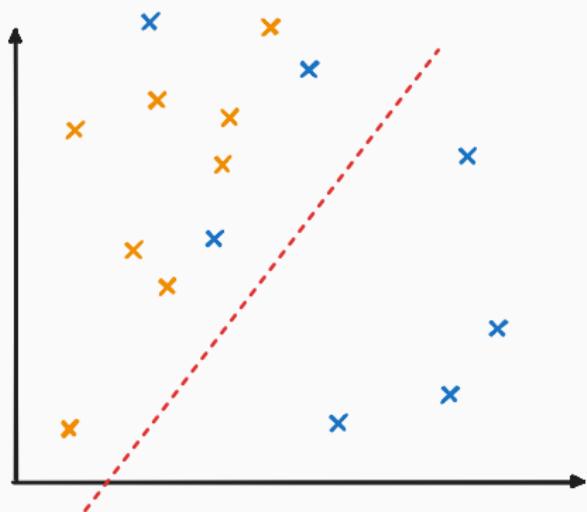
- Linear separation on raw features
- method 1 → ?



# Non linearly separable

Handle non linearity?

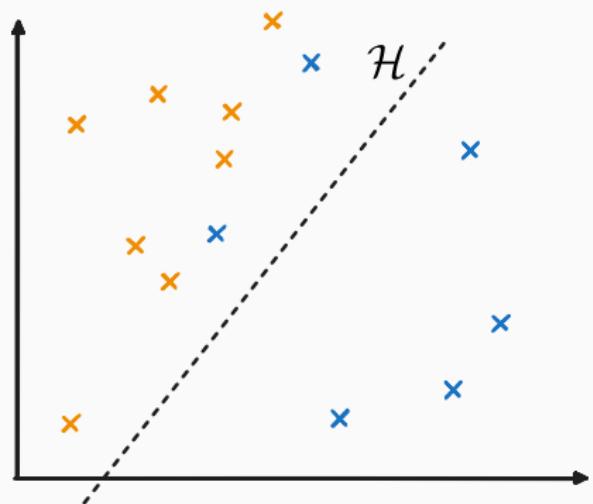
- Linear separation on raw features
- **method 1** → Using more complex separator (non-linear)



# Non linearly separable

Handle non linearity?

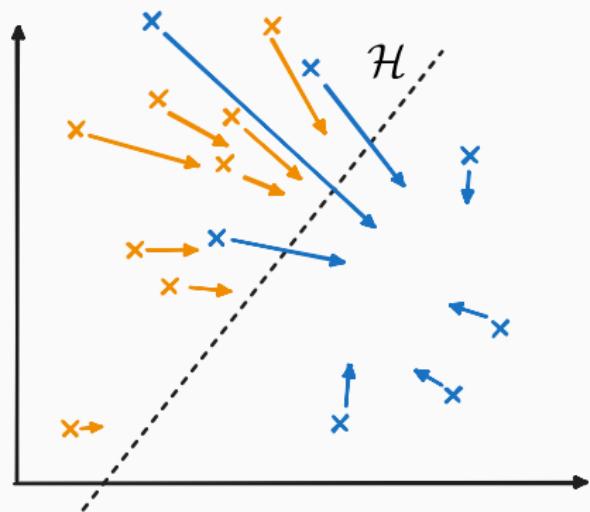
- Linear separation on raw features
- **method 1** → Using more complex separator (non-linear)
- **method 2** → ?



# Non linearly separable

Handle non linearity?

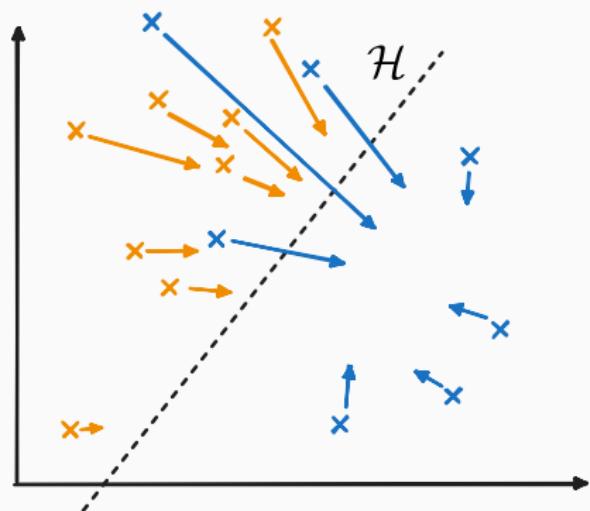
- Linear separation on raw features
- **method 1** → Using more complex separator (non-linear)
- **method 2** → ?



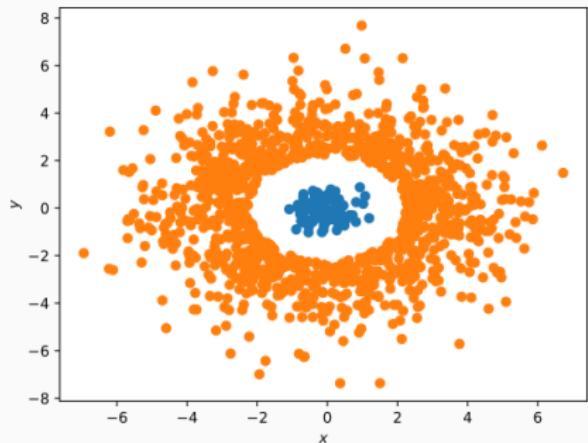
# Non linearly separable

Handle non linearity?

- Linear separation on raw features
- **method 1** → Using more complex separator (non-linear)
- **method 2** → Transform features space to make it linearly separable



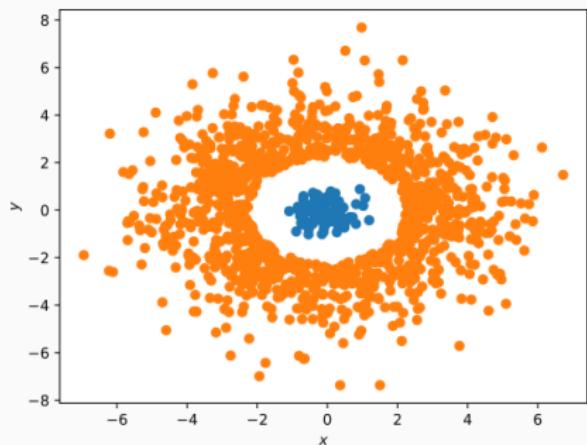
## Handle non linearity: example (1)



### First example

Any idea of a transformation?

## Handle non linearity: example (1)



### First example

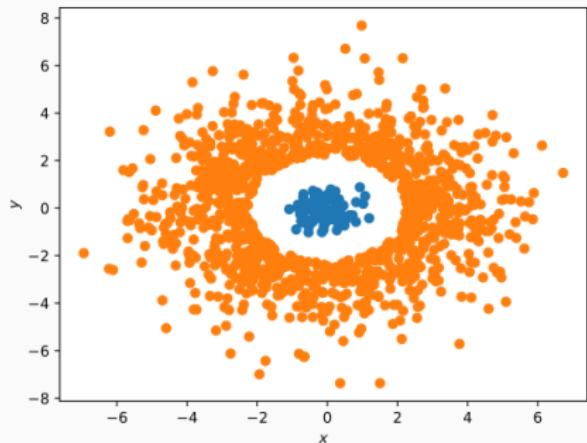
Any idea of a transformation?

### Solution

Transform to polar coordinate

→ only the norm is important here

## Handle non linearity: example (1)



**First example:**

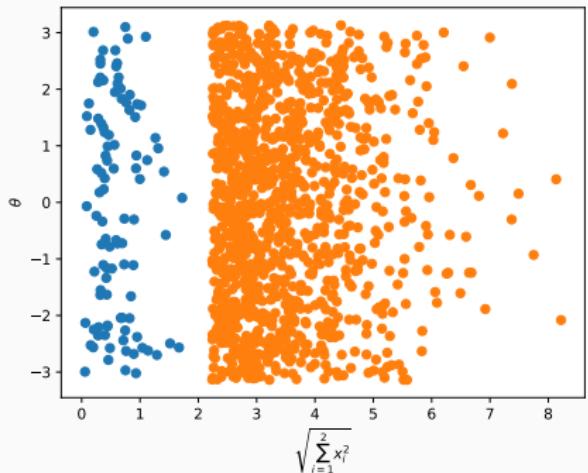
Any idea of a transformation?

### Solution

Transform to polar coordinate

→ only the norm is important here

## Handle non linearity: example (1)



**First example:**

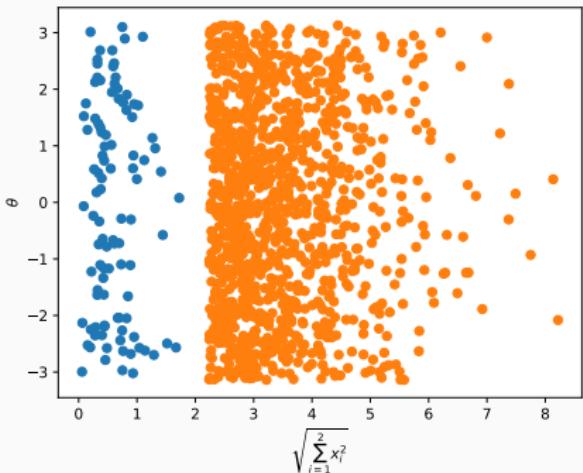
Any idea of a transformation?

### Solution

Transform to polar coordinate

→ only the norm is important here

## Handle non linearity: example (1)



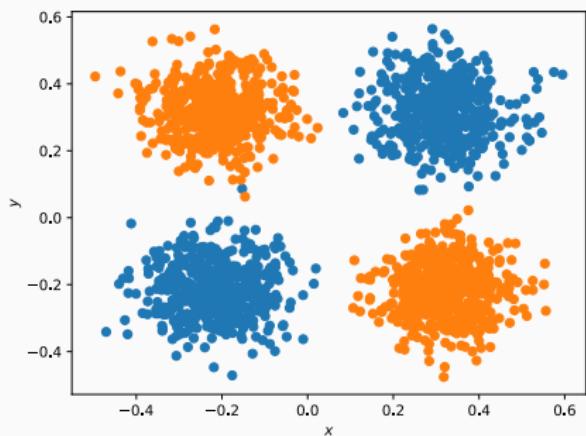
**First example:**

Any idea of a transformation?

**Solution**

Transform to polar coordinate → only  
the norm is important here

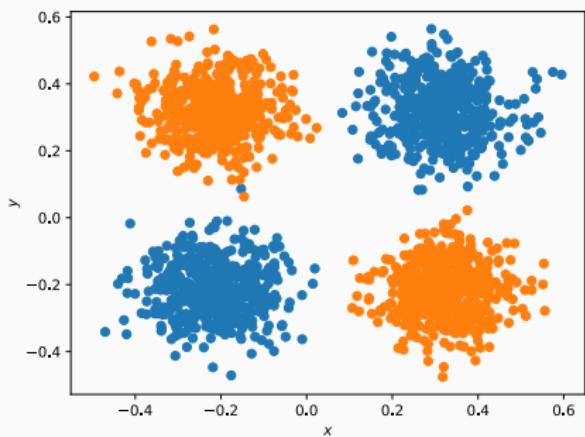
## Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

## Handle non linearity: example (2)



### The chessboard problem:

A solution to make it linearly separable?

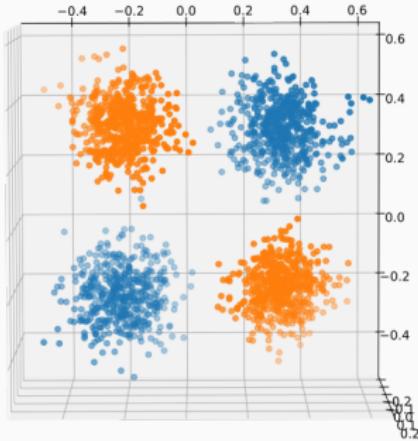
#### Solution

Add a new dimension, transforming data point using:

$$g: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longmapsto \begin{pmatrix} x_1 \\ x_2 \\ x_1x_2 \end{pmatrix}$$

## Handle non linearity: example (2)



### The chessboard problem:

A solution to make it linearly separable?

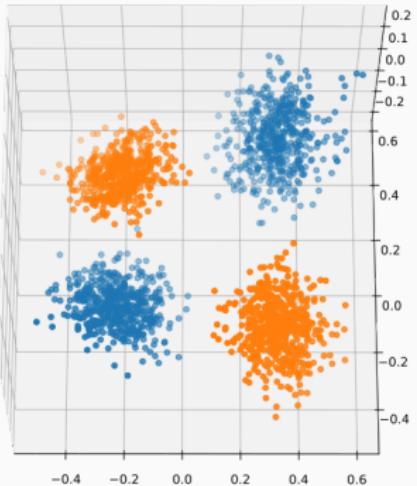
#### Solution

Add a new dimension, transforming data point using:

$$g: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longmapsto \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

## Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

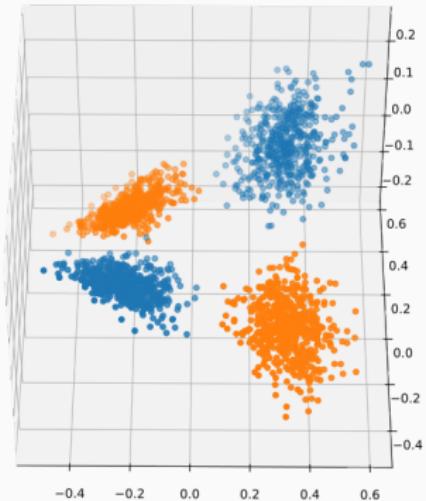
### Solution

Add a new dimension, transforming data point using:

$$g: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longmapsto \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

## Handle non linearity: example (2)



**The chessboard problem:**

A solution to make it linearly separable?

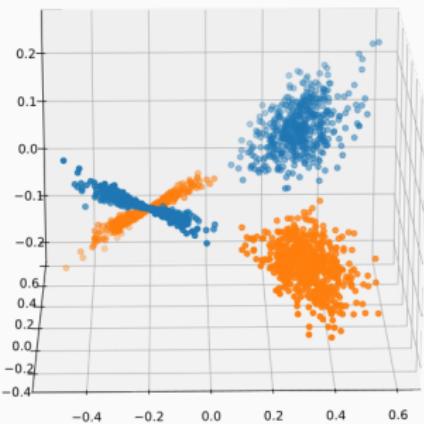
### Solution

Add a new dimension, transforming data point using:

$$g: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longmapsto \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

## Handle non linearity: example (2)



### The chessboard problem:

A solution to make it linearly separable?

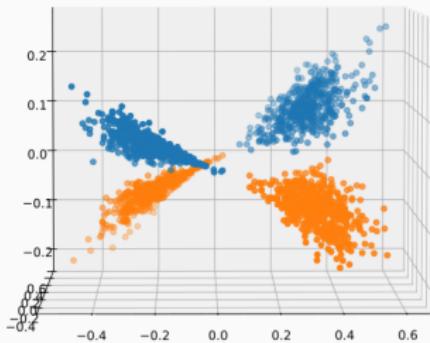
#### Solution

Add a new dimension, transforming data point using:

$$g: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longmapsto \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

## Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

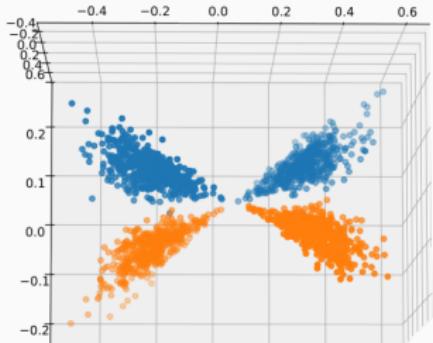
### Solution

Add a new dimension, transforming data point using:

$$g: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longmapsto \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

## Handle non linearity: example (2)



### The chessboard problem:

A solution to make it linearly separable?

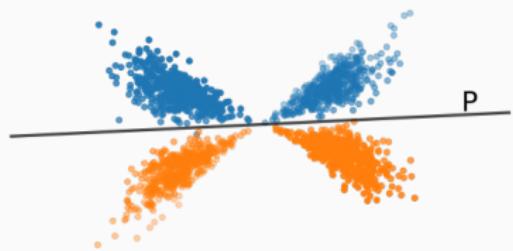
#### Solution

Add a new dimension, transforming data point using:

$$g: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longmapsto \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

## Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

### Solution

Add a new dimension, transforming data point using:

$$g: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longmapsto \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

## Multilayer Perceptron

---

# Main Idea

Dealing with non-linearly separable inputs?

- Manually transform the inputs
- Learn automatically a transformation

The multi-layer perceptron

- Compute “latent” hidden representations so that classes are linearly separable  
(intermediate representations of the inputs)
- Use non-linear function(s) so the transformation is not linear

**Classifier:**

Parameterized function:

$$f_{\Theta}: \mathcal{X} \longrightarrow \mathcal{Y}$$
$$x \longmapsto g_{\Theta_H}(\dots g_{\Theta_2}(g_{\Theta_1}(x)))$$

# Linear Classifier for Multi-Class Classification

## Problem

- **Input:** Features in  $\mathcal{X} \subset \mathbb{R}^n$
- **Output:** a vector in  $\mathcal{Y} \subset \mathbb{R}^k$  with  $k$  the number of classes

### Linear classifier

$$\hat{y} = A^T x + b \rightarrow \text{dimension?}$$

- $A \in \mathbb{R}^{n \times k}$
- $b \in \mathbb{R}^k$

$$\hat{y} = \left( \begin{array}{c} \text{blue dots} \\ \text{blue dots} \\ \text{blue dots} \\ \text{blue dots} \end{array} \right) = \left( \begin{array}{ccc} & A^T & \\ \text{blue dots} & \times & \left( \begin{array}{c} \text{blue dots} \\ x \\ \text{blue dots} \end{array} \right) + \left( \begin{array}{c} \text{blue dots} \\ \text{blue dots} \\ \text{blue dots} \\ \text{blue dots} \end{array} \right) \end{array} \right)$$

# Multi-layer Perceptron: base block

first hidden layer                    second hidden layer                    output projection

$$z^{(1)} = \sigma^{(1)} \left( A^{(1)\top} x + b^{(1)} \right) \quad z^{(2)} = \sigma^{(2)} \left( A^{(2)\top} z^{(1)} + b^{(2)} \right) \quad o = \sigma^{(2)} \left( A^{(3)\top} z^{(2)} + b^{(3)} \right)$$

## Notations

- $x$ : input features
- $z^{(i)}$ : hidden representation
- $o$ : output logits
- $\Theta = \{A^{(1)}, b^{(1)}\}$ : parameters
- $\sigma^{(i)}$ : non-linear activation

$$z^{(1)} = \sigma \left( \begin{array}{c|ccccc} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\ \hline & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & \text{---} & \text{---} & \text{---} & \text{---} & \text{---} \end{array} \right) \times \begin{matrix} \text{---} \\ \text{---} \\ x \\ \text{---} \end{matrix} + \begin{matrix} \text{---} \\ \text{---} \\ b^{(1)} \\ \text{---} \end{matrix}$$

# Multi-layer Perceptron: Non-Linear Activation Functions

## Activation functions

- Apply a non-linear transformation
- Applied piecewise
- Many choices possible

$$\sigma(x) = \begin{pmatrix} \sigma(x_1) \\ \sigma(x_2) \\ \vdots \\ \sigma(x_n) \end{pmatrix}$$

# Multi-layer Perceptron: Non-Linear Activation Functions

## Activation functions

- Apply a non-linear transformation
- Applied piecewise
- Many choices possible

$$\sigma(x) = \begin{pmatrix} \sigma(x_1) \\ \sigma(x_2) \\ \vdots \\ \sigma(x_n) \end{pmatrix}$$

## Why activations?

Let consider  $A^{(1)} \in \mathbb{R}^{n \times m}$  and  $A^{(2)} \in \mathbb{R}^{m \times k}$

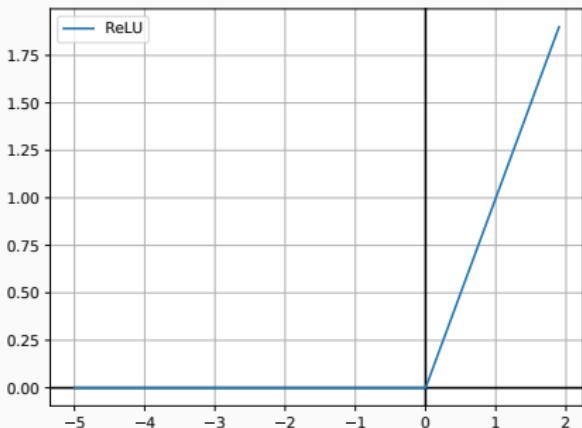
$$A^{(2)\top}(A^{(1)\top}x) \iff (A^{(2)\top}A^{(1)\top})x$$

And  $A^{(2)\top}A^{(1)\top}$  is a linear function ( $A^{1,2} \in \mathbb{R}^{n \times k}$ )

# Neural Network architectures : Activation function review

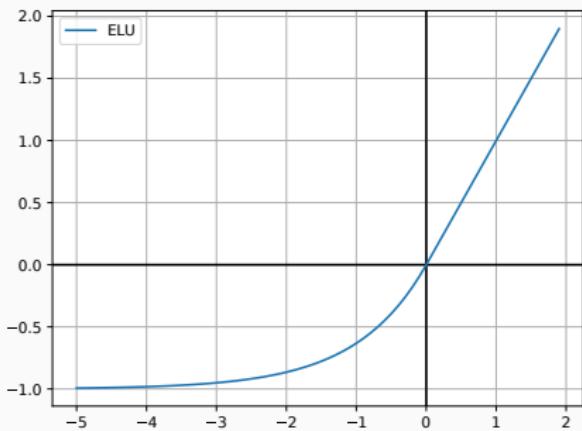
Rectified Linear Unit :

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$



Exponential Linear Unit :

$$ELU(x) = \begin{cases} x & \text{if } x \geq 0 \\ e^x - 1 & \text{if } x \leq 0 \end{cases}$$



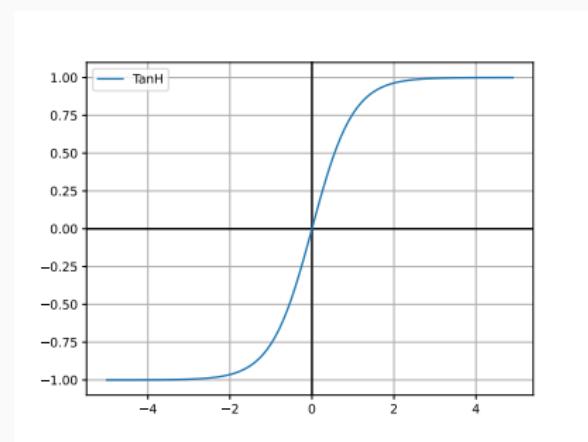
# Neural Networks : TanH activation function

**Hyperbolic tangent:**

Hyperbolic tangent (**TanH**) is defined by :

$$f: \mathbb{R} \longrightarrow [-1, 1]$$

$$x \longmapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



**Figure 3:** The Hyperbolic tangent activation function

# Neural Networks : Sigmoid activation function

## Sigmoid:

Sigmoid activation is defined by :

$$\begin{aligned} f: \mathbb{R} &\longrightarrow [-1, 1] \\ x &\longmapsto \frac{1}{1 + e^{-x}} \end{aligned}$$

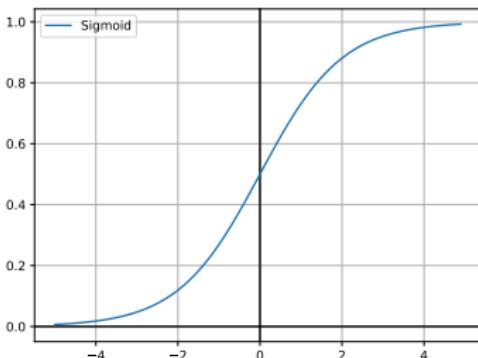


Figure 4: Sigmoid activation function

# Neural Networks : Sigmoid activation function

## Activation functions

- Introducing non-linearity
- Depends on the layers (LSTM use sigmoid to filter representation)

There is much more activation functions...

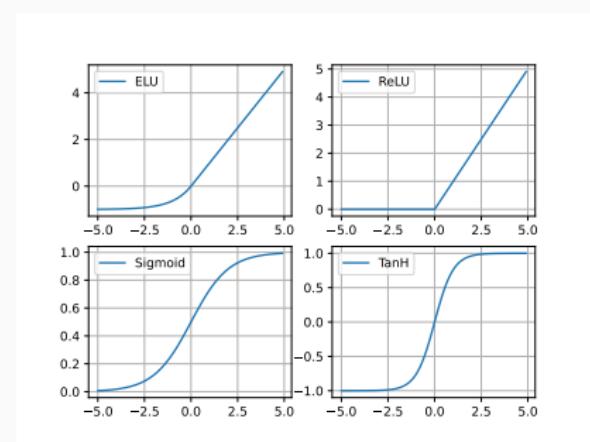
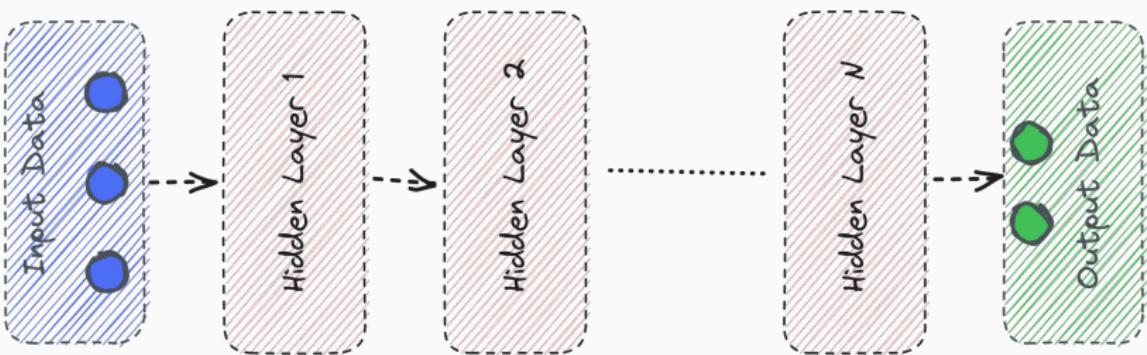


Figure 5: Different activation functions

## Summary: MLP



### Neural Networks

A stack of hidden-layers

- Often same kind of layers (true for the multi-layer perceptron)
- More hidden layers  $\rightarrow$  more different functions possible

## Summary: Hidden layer

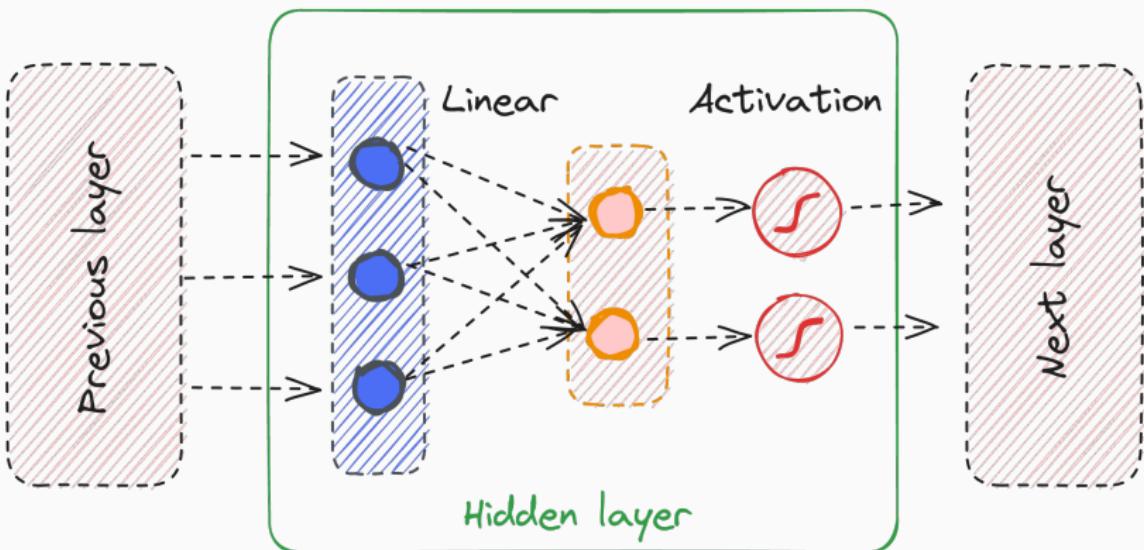


Figure 6: Classical representation of a neural network layer

# A full example

- ▶  $x$ : input features
- ▶  $z^{(1)}, z^{(2)}$ : hidden representation
- ▶  $w$ : output logits or class weights
- ▶  $p$ : probability distribution over classes
- ▶  $\theta = \{\mathbf{A}^{(1)}, \mathbf{b}^{(1)}, \dots\}$ : parameters
- ▶  $\sigma$ : non-linear activation function

$$z^{(1)} = \sigma(\mathbf{A}^{(1)}x + \mathbf{b}^{(1)})$$

$$z^{(2)} = \sigma(\mathbf{A}^{(2)}z^{(1)} + \mathbf{b}^{(2)})$$

$$w = \sigma(\mathbf{A}^{(3)}z^{(2)} + \mathbf{b}^{(3)})$$

$$p = \text{Softmax}(w) \quad \text{i.e.} \quad p_i = \frac{\exp(w_i)}{\sum_j \exp(w_j)}$$

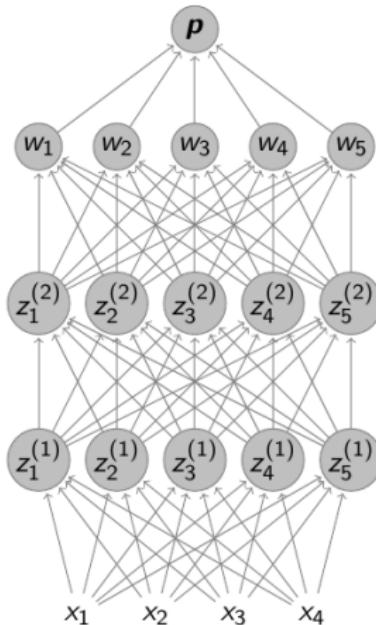


Figure 7: Mathematical and Graphical model side by side

## Vocabulary issue

The term "prediction function" can refer to both the "full model" or only the function that transforms the class weights/logits/scores to an actual output.

## DO NOT CONFUSE

- The (non-linear) activation function (inside the neural network)
- The function that transforms weights/logits/scores into an output (at the output of the neural network)

## Loss functions in Binary classification

$$o = a^{(L)\top} z^{(L-1)} + b^{(L)}$$

The diagram illustrates the computation of the output  $o$ . On the left, a vertical vector  $o$  is shown with four blue dots. To its right is an equals sign. To the right of the equals sign is a multiplication symbol ( $\times$ ). To the right of the multiplication symbol is another vertical vector  $z^{(L-1)}$  with three blue dots. To the right of  $z^{(L-1)}$  is a plus sign (+). To the right of the plus sign is a square bracket containing a blue dot, representing the bias term  $b^{(L)}$ . To the left of the equals sign is a horizontal vector  $a^{(L)\top}$  enclosed in a rectangular frame, with three blue dots. This represents the transpose of the weight vector  $a^{(L)}$ .

### Prediction function

- **Hinge loss (gold is 0/1):**  $l(o, y) = \max(0, 1 - (2y - 1)o)$
- **Hinge loss (gold is -1/1):**  $l(o, y) = \max(0, 1 - yo)$
- **Negative log-likelihood (gold is 0/1):**  $l(o, y) = -yo + \log(1 + e^w)$
- **Negative log-likelihood (gold is -1/1):**  $l(o, y) = \log(1 + \exp(-yw))$

# Predictions function in Multiclass classification

## Prediction functions

$$o = A^{(L)\top} z^{(L-1)} + b^{(L)}$$

- **Integer output:**  $\hat{y} = \arg \max_{i \in \{1, \dots, l\}} o_i$

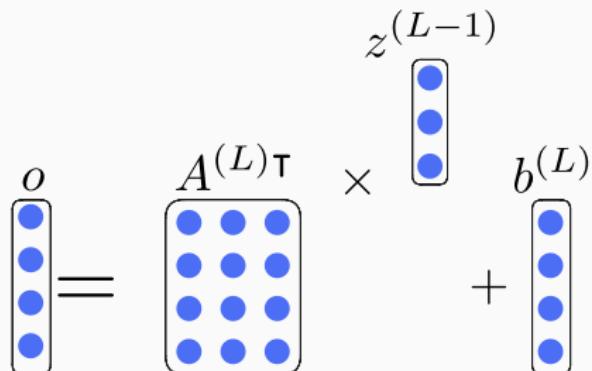
- **One-hot vector output:**

$$\hat{y} = \arg \max_{h \in E(k)} \langle h, o \rangle$$
$$\rightarrow E(k) = \{h \mid h \in \mathbb{R}^k; \exists h_i = 1; \sum_{i=1}^k = 1\}$$

- **Probabilistic output:**

$$\hat{y} = \text{softmax}(o) = \begin{pmatrix} \frac{e^{o_1}}{\sum_{i=1}^k e^{o_i}} \\ \frac{e^{o_2}}{\sum_{i=1}^k e^{o_i}} \\ \vdots \\ \frac{e^{o_k}}{\sum_{i=1}^k e^{o_i}} \end{pmatrix}$$

## Predictions function in Multiclass classification



**Prediction functions:**

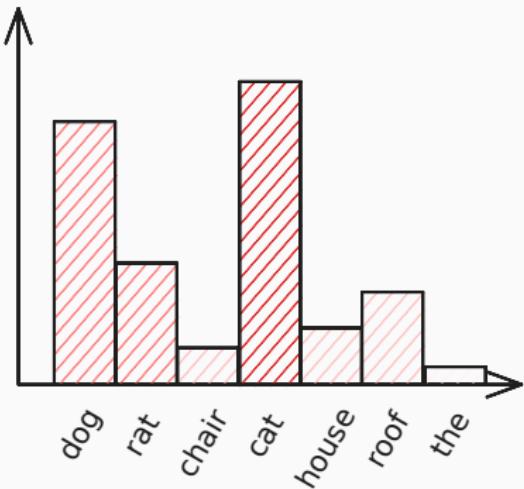
- **Integer output:**  $\hat{y} = \arg \max_{i \in \{1, \dots, l\}} o_i$
- **One-hot vector output:**  $\hat{y} = \arg \max_{h \in E(k)} \langle h, o \rangle$   
 $\rightarrow E(k) = \{h \mid h \in \mathbb{R}^k; \exists h_i = 1; \sum_{i=1}^k h_i = 1\}$
- **Probabilistic output (distribution):**  $\hat{y} = \text{softmax}(o)$

# Precision on the softmax

**Softmax function:**

For a vector  $\mathbf{o}$  the softmax is defined by:

$$\text{softmax}(\mathbf{o}) = \begin{pmatrix} \frac{e^{o_1}}{\sum\limits_{i=1}^k e^{o_i}} \\ \frac{e^{o_2}}{\sum\limits_{i=1}^k e^{o_i}} \\ \vdots \\ \frac{e^{o_k}}{\sum\limits_{i=1}^k e^{o_i}} \end{pmatrix}$$



**Figure 8:** Model distribution to predict a word

# Precision on the softmax

Softmax function:

For a vector  $\mathbf{o}$  the softmax is defined by:

$$\text{softmax}(\mathbf{o}) = \begin{pmatrix} \frac{e^{o_1}}{\sum\limits_{i=1}^k e^{o_i}} \\ \frac{e^{o_2}}{\sum\limits_{i=1}^k e^{o_i}} \\ \vdots \\ \frac{e^{o_k}}{\sum\limits_{i=1}^k e^{o_i}} \end{pmatrix}$$

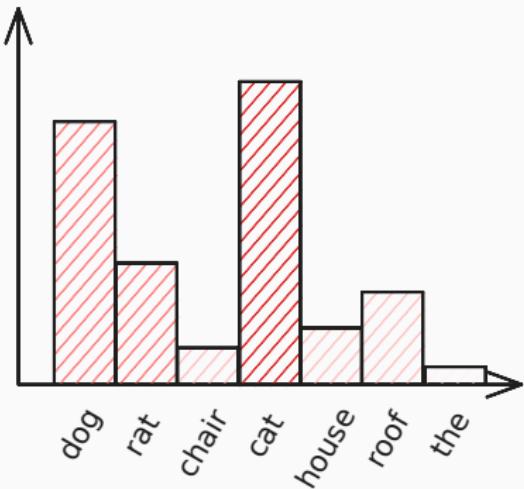
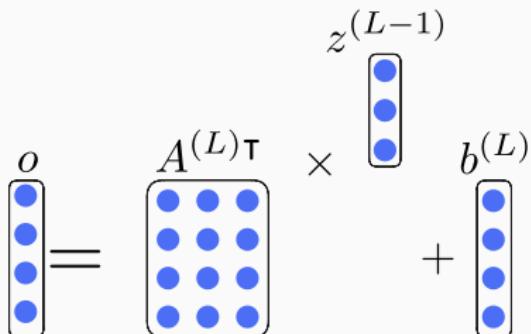


Figure 8: Model distribution to predict a word

## Using Exponential?

- Having a spiked distribution
- Property in training

# loss function in Multiclass classification



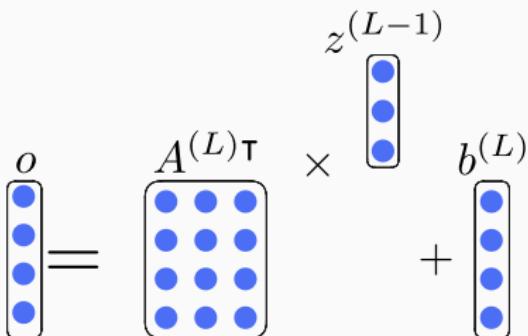
## Loss functions:

- **Hinge loss:**  $l(o, y) = \max(0, m - \langle y, w \rangle + \max_{h \in E^k \setminus \{y\}} (\langle h, o \rangle))$
- **Negative Log-likelihood:(y one hot)**  $l(o, y) = -\langle y, o \rangle + \log \sum_{i=1}^k e^{w_i}$
- **Negative Log-likelihood:(y class index)**  $l(o, y) = -o_y + \log \sum_{i=1}^k e^{w_i}$

---

<sup>0</sup>The log-likelihood is in general case  $\sum_{i=1}^k p_i \log(q_{o_i})$  with  $p_i$  probability of the label  $i$  (here 1 for the truth class else 0), and  $q_{o_i}$  the probability of the class  $i$  in output vector

# Regression

$$o = A^{(L)\top} z^{(L-1)} + b^{(L)}$$


**Prediction functions:**

- **Hinge loss:**  $\hat{y} = o$

**Loss functions:**

- **Mean Square Error (MSE):**  $l(o, y) = (y - o)^2$
- **Absolute Error Loss:**  $l(o, y) = |y - o|$

## Finding Parameters in neural networks

---

# Objective function

## Intuition

- Compare the output with the gold output
- Minimization of the loss (bounded by 0)
- Must be related to the evaluation function, but often different  
Minimize cross entropy but evaluate counting correct classification

## Learning Objective

$$\Theta^* = \arg \min_{\Theta} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y^{(i)}, f_{\Theta}(x^{(i)}))$$

We call in the following

$$E(\mathcal{D}, f_{\Theta}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y^{(i)}, f_{\Theta}(x^{(i)}))$$

# Neural Networks: Cost function and Optimisation

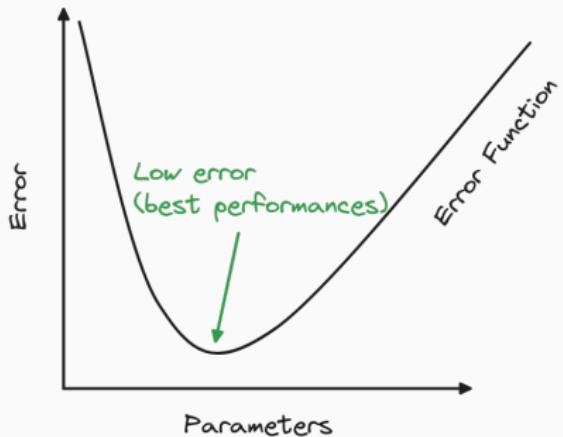
Finding  $\theta$  s.t we minimize

$$E(\mathcal{D}, f_{\Theta}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y^{(i)}, f_{\Theta}(x^{(i)}))$$

$E(\mathcal{D}, f_{\Theta})$  convex ?

- A unique minimum
- Gradient of error is null

$$\nabla E(\mathcal{D}, f_{\Theta}) = 0_E$$



# Neural Networks: Cost function and Optimisation

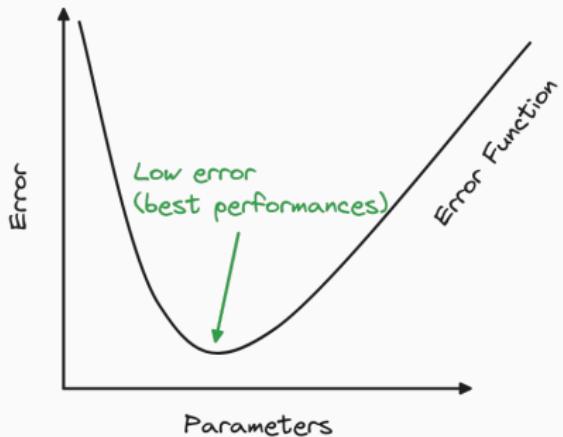
Finding  $\theta$  s.t we minimize

$$E(\mathcal{D}, f_\Theta) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y^{(i)}, f_\Theta(x^{(i)}))$$

$E(\mathcal{D}, f_\Theta)$  convex ?

- A unique minimum
- Gradient of error is null

$$\nabla E(\mathcal{D}, f_\Theta) = 0_E$$



Not always a closed form solution, error function not convex

# Gradient Descent

## Problem

Solve the following problem  $\arg \min_{\Theta} E(\mathcal{D}, f_{\Theta})$

## Intuition

- All you can compute: evaluate the function and its gradient at a given point
- You can use gradient information to see in which direction the function is decreasing
- Therefore: just make a small step in this direction!

## Formally

- Choose an initial point randomly:  $\Theta_0$
- Make T iterations/steps:  $\Theta_{t+1} = \Theta_t - \lambda \nabla E(\mathcal{D}, f_{\Theta})$

## Gradient Descent:

**Intuition:** Follow the error function  
“slope”

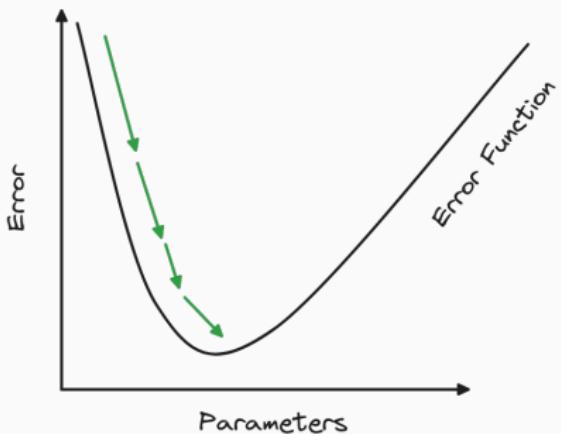
$$\Theta_{t+1} = \Theta_t - \lambda \nabla E(\mathcal{D}, f_\Theta)$$

# Neural Networks: Cost function and Optimisation

**Gradient Descent:**

**Intuition:** Follow the error function  
“slope”

$$\Theta_{t+1} = \Theta_t - \lambda \nabla E(\mathcal{D}, f_\Theta)$$



## Gradient Descent:

**Intuition:** Follow the error function  
“slope”

$$\Theta_{t+1} = \Theta_t - \lambda \nabla E(\mathcal{D}, f_\Theta)$$

Carrefully choose  $\lambda$  (the gradient step)

- To small : Many step (and bad local minima if not convex)

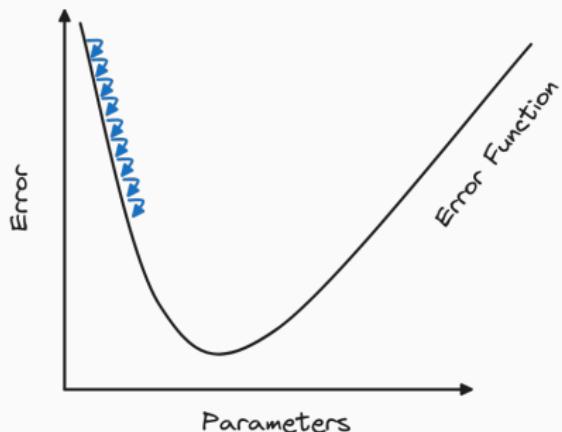
## Gradient Descent:

**Intuition:** Follow the error function  
"slope"

$$\Theta_{t+1} = \Theta_t - \lambda \nabla E(\mathcal{D}, f_\Theta)$$

Carrefully choose  $\lambda$  (the gradient step)

- To small : Many step (and bad local minima if not convex)



## Gradient Descent:

**Intuition:** Follow the error function  
“slope”

$$\Theta_{t+1} = \Theta_t - \lambda \nabla E(\mathcal{D}, f_\Theta)$$

Carrefully choose  $\lambda$  (the gradient step)

- To small : Many step
- To large : Can not reach minima

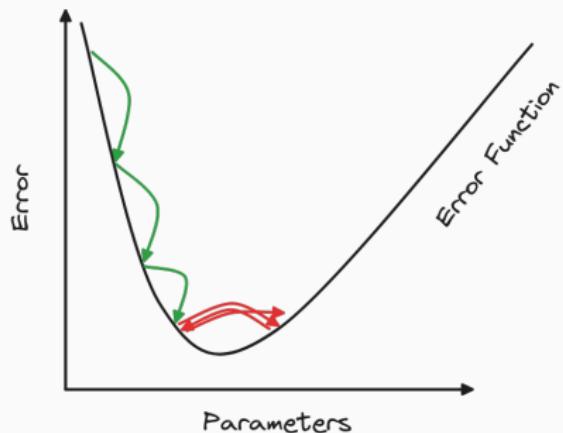
## Gradient Descent:

**Intuition:** Follow the error function  
“slope”

$$\Theta_{t+1} = \Theta_t - \lambda \nabla E(\mathcal{D}, f_\Theta)$$

Carrefully choose  $\lambda$  (the gradient step)

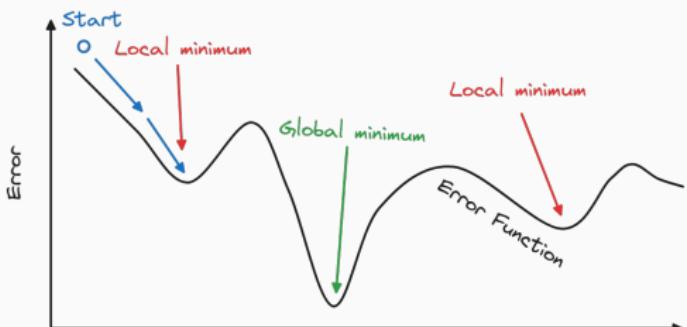
- To small : Many step
- To large : Can not reach minima



# Neural Networks: Cost function and Optimisation

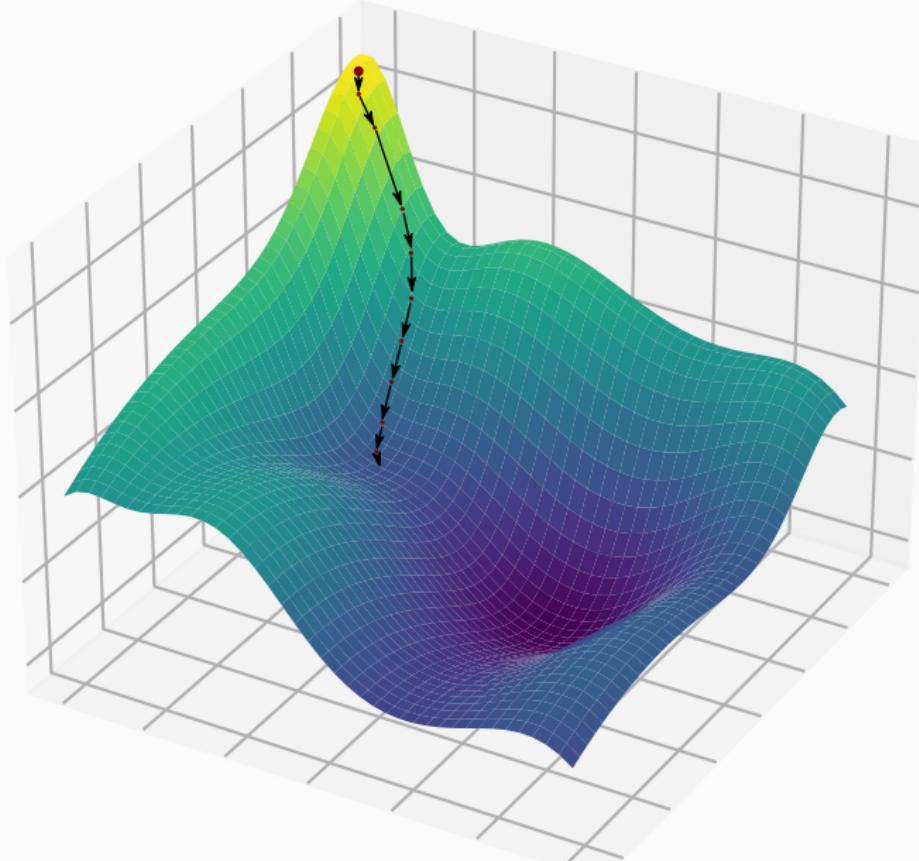
**Non convex fonction**

→ suboptimal solution with gradient descent !

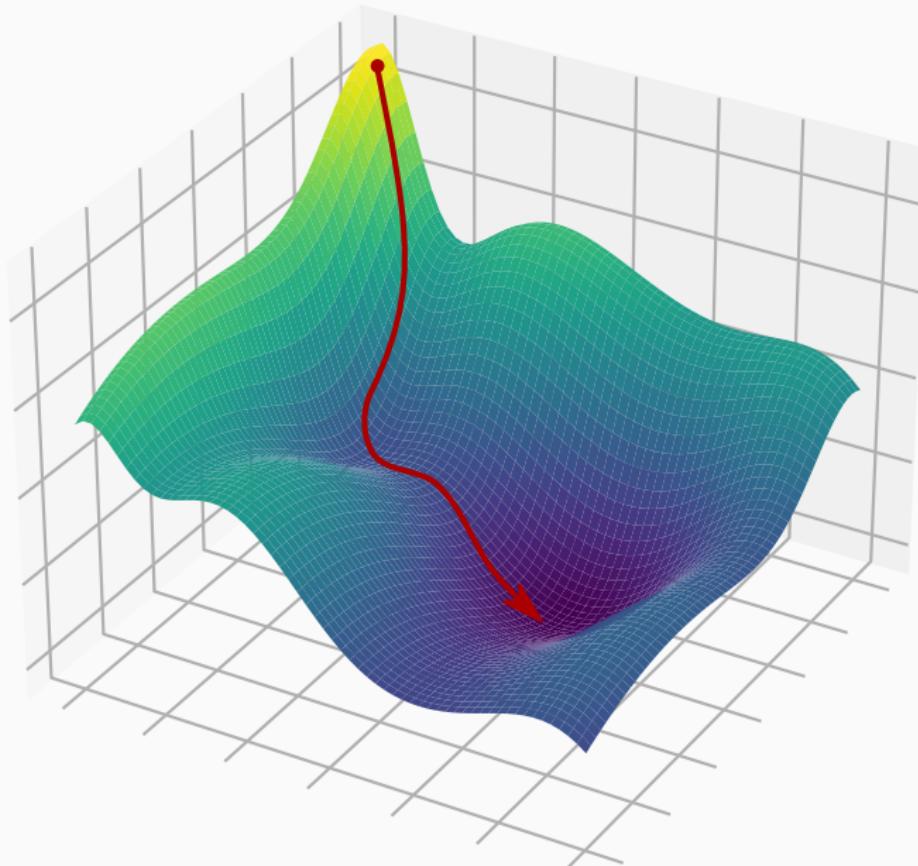


In practice we find local minima (and it is ok to not find global minimum)

# Neural Networks: Cost function and Optimisation



# Neural Networks: Cost function and Optimisation



# Generalisation

## Overparameterized neural Networks

- Networks where the number of parameters exceed the training set size
- Can learn by heart the dataset (Overfitting)
- Are easier to optimize in practice

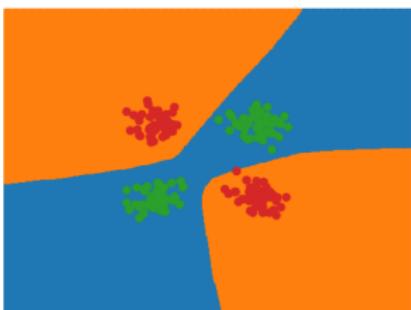
## Monitoring the training process

- Loss should go down
- Training accuracy should go up
- Dev accuracy should go up

## Regularization

Techniques to control parameters to prevent overfitting (during training step)

## Handle non linearity: example (3)



### The chessboard problem:

This time using non-linear separation (a multilayer perceptron)

## Handle non linearity: example (3)

Animation of the same problem

## Neural Architectures

---

A neural network is a complicated parameterized function

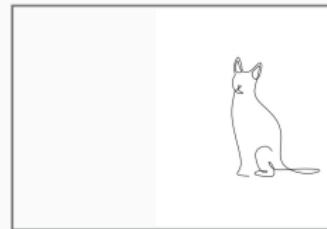
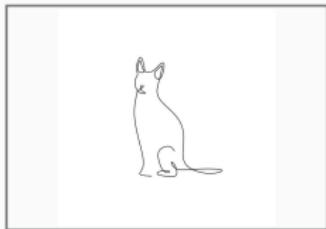
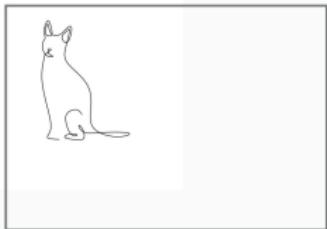
## How to choose the function()

- Inductive bias: take into account the data to design the architecture
- Time complexity/speed (operation time, number of parameters)
- Mathematical properties for efficient training:  
→ Differentiability, prevent vanishing/exploding gradients...

# Convolutional Neural Networks (CNN)

## Intuition

No matter where the cat is in the picture → class is cat  
→ We want to encode this fact in the neural architecture!

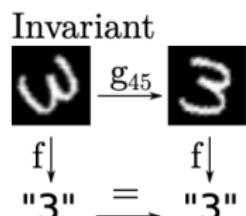
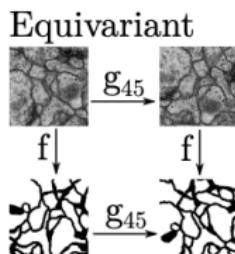


## Equivariant function

If we apply a transformation on the input, the output will be transformed in the "same" way

## Invariant function

If we apply a transformation on the input, the output will remain the same



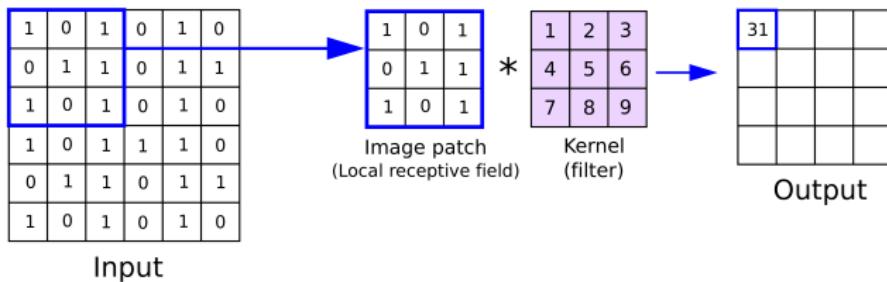
# Convolutional Neural Networks (CNN)

## CNN block (not formal)

Let be  $x \in \mathbb{R}^{n \times m}$ ,  $f \in \mathbb{R}^{k \times k}$

- We apply for different part of the image  $p_i$ ,  $g_i = p_i * f$  (Hadamard product)
- For each part we apply a pooling  $pooling(p_i)$

Pooling method can be the mean, the max,... **Many filter, Many block**



# Convolutional Neural Networks (CNN)

## Translation equivariant convolution

Preserves the “translation structure”

- Input transposed  $\implies$  output transposed
- Pooling make it invariant



## Translation equivariant convolution

Preserves the “rotation structure”

- Input rotated  $\implies$  output rotated
- The output is rotated



Standard convolution is not rotation equivariant



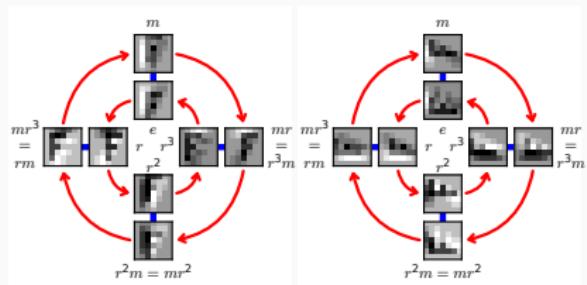
<sup>0</sup>Image from <https://anhreynolds.com/blogs/cnn.html>

# Taking into account rotations (CNN)

## Invariant by rotation

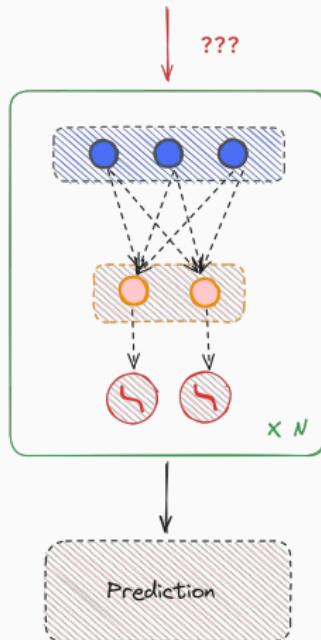
An image rotated will not be classified similarly from the original  
→ consider rotation of filters

Proposal of p4 map



# Encoding sequence:(text example)

This is a fantastic movie

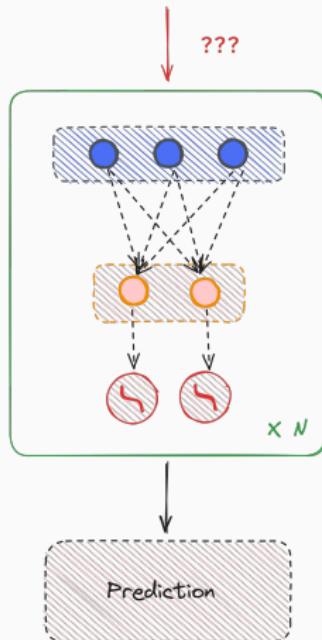


## Encoding Textual Information

Neural networks or classifiers (linear for instance) :  
→ Need vectorial representation as input

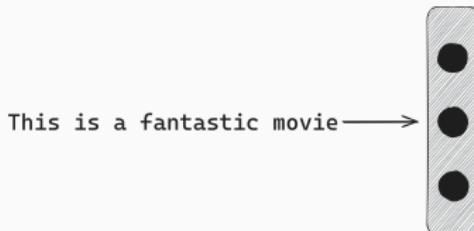
# Encoding sequence:(text example)

This is a fantastic movie



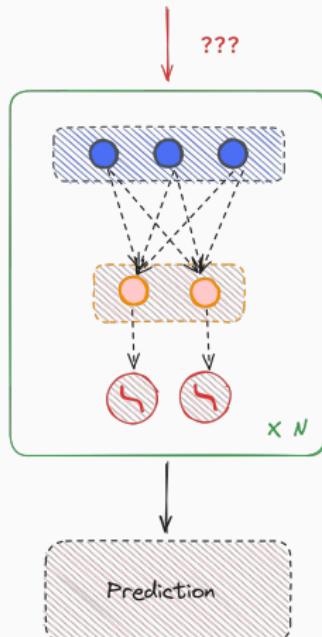
## Encoding Textual Information

Neural networks or classifiers (linear for instance) :  
→ Need vectorial representation as input



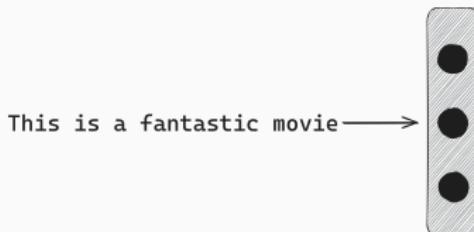
# Encoding sequence:(text example)

This is a fantastic movie



## Encoding Textual Information

Neural networks or classifiers (linear for instance) :  
→ Need **vectorial representation as input**



Associate each word of the vocabulary to a vector and aggregate the vocabulary of the input text :  
→ **Encoding as Bag of Word**

## Bag of Word representation:

Each word is associated to an integer,  
representation of a word of index  $i$  is  
 $w \in \mathbb{R}^{|V|}$  with  $|V|$  the vocabulary size with:

$$w_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

# Natural Language Processing: Bag of Word representation

## Bag of Word representation:

Each word is associated to an integer, representation of a word of index  $i$  is  $w \in \mathbb{R}^{|V|}$  with  $|V|$  the vocabulary size with:

$$w_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

A sequence of word  $w^1, w^2, \dots, w^n$  can be represented by  $\sum_{k=0}^n w^k$

The diagram illustrates the construction of a Bag of Words vector. On the left, the words "this", "is", "a", "fantastic", and "movie" are listed above their corresponding binary vectors. An arrow points from the word "fantastic" to its vector, which is then combined with the other vectors via addition to produce the final vector on the right.

this	1	0	0	0	0	1
fantastic	0	0	0	1	0	1
book	0	0	0	0	0	0
bad	0	0	0	0	0	0
a	0	0	1	0	0	1
movie	0	0	0	0	1	1
is	0	1	0	0	0	1
music	0	0	0	0	0	0

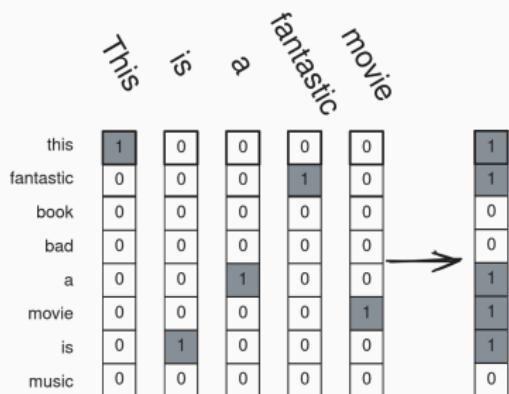
# Natural Language Processing: Bag of Word representation

## Bag of Word representation:

Each word is associated to an integer, representation of a word of index  $i$  is  $w \in \mathbb{R}^{|V|}$  with  $|V|$  the vocabulary size with:

$$w_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

A sequence of word  $w^1, w^2, \dots, w^n$  can be represented by  $\sum_{k=0}^n w^k$



NB: related encoding exists such as using the word (term) frequency or using other statistical value on words (inverse document frequency)

## Bag of words issue:

BoW representation does not capture proximity between words :

- The representation of the word “cat” will be as close from “dog” and “car”
- There is no semantics in the representation

# Natural Language Processing: Word embedding

## Bag of words issue:

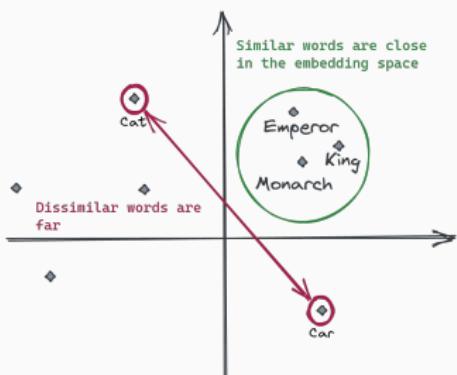
BoW representation does not capture proximity between words :

- The representation of the word “cat” will be as close from “dog” and “car”
- There is no semantics in the representation

## Taking into account semantics:

Considering a similarity metric or a distance

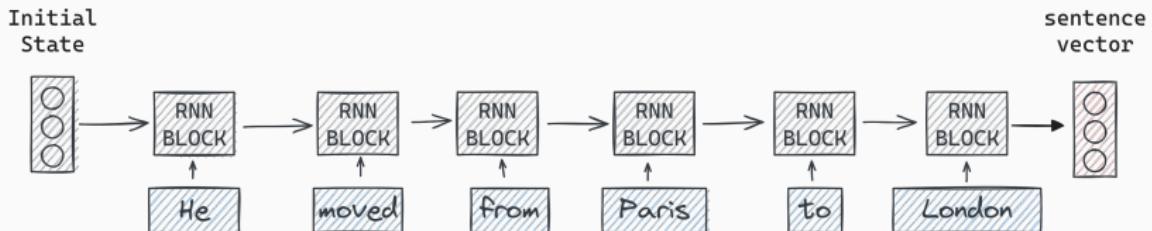
- Ensure word representation close for semantically close words
- Semantically unrelated words should be far from each others



## Importance of words order

- "Brian moved to the bathroom then Brian went to the kitchen"  
→ Sequence of action
- "Alexander III was a king of Macedon. **He** succeeded **his** father Philip II"  
→ Personal pronouns (co-reference)
- In pos-tagging approaches
- In generative approaches (translation/question answering)
- And much more

# Natural Language Processing: Modeling sequence by architecture



## The Recurrent Neural Network

- Process sequence of various size
- Each block “merge” information from previous state and current input

Let be  $h_t$  the  $t^{th}$  state vector and  $x_t$  the  $t^{th}$  input, RNN block is defined by:

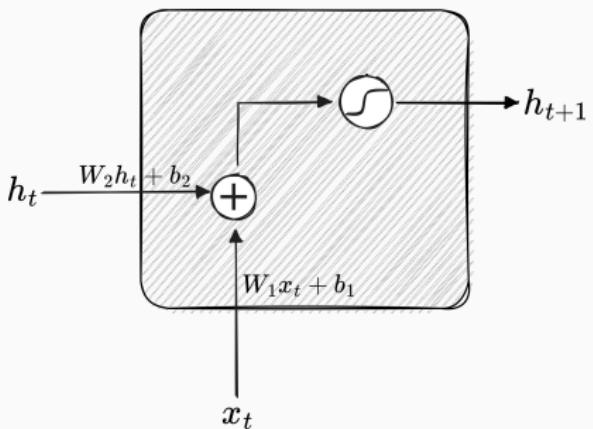
$$f_{\theta} : \mathbb{R}^D \times \mathbb{R}^n \rightarrow \mathbb{R}^D$$
$$(h_t, x_t) \mapsto h_{t+1}$$

## Vanilla RNN

For the Vanilla RNN the computation of  $h_{t+1}$  is given by

$$\tanh(W_h h_t + b_h \mathbf{1}^\top + W_x x_t + b_x \mathbf{1}^\top)$$

$\rightarrow W_h \in \mathbb{R}^{d \times d}$ ,  $W_x \in \mathbb{R}^{d \times n}$  and  
 $b_h, b_x \in \mathbb{R}^d$  the trainable parameters



# Contextualize word in sentence: Vanilla RNN

## Vanilla RNN

For the Vanilla RNN the computation of  $h_{t+1}$  is given by

$$\tanh(W_h h_t + b_h \mathbf{1}^\top + W_x x_t + b_x \mathbf{1}^\top)$$

$\rightarrow W_h \in \mathbb{R}^{d \times d}$ ,  $W_x \in \mathbb{R}^{d \times n}$  and  $b_h, b_x \in \mathbb{R}^d$   
the trainable parameters

---

### Algorithm 1 Many to One forward

---

**Require:**  $x_1, x_2, \dots, x_T \in \mathbb{R}^n$ ,  $h_0 \in \mathbb{R}^d$

```
     $h \leftarrow h_0$ 
    for  $t \leq T$  do
         $h' \leftarrow W_h h$ 
         $x' \leftarrow W_x x_t$ 
         $h \leftarrow \tanh(h' + x' + b \mathbf{1}^\top)$ 
    end for
    return  $W_y h + b_y \mathbf{1}^\top$ 
```

---

## Contextualize word in sentence: Vanilla RNN

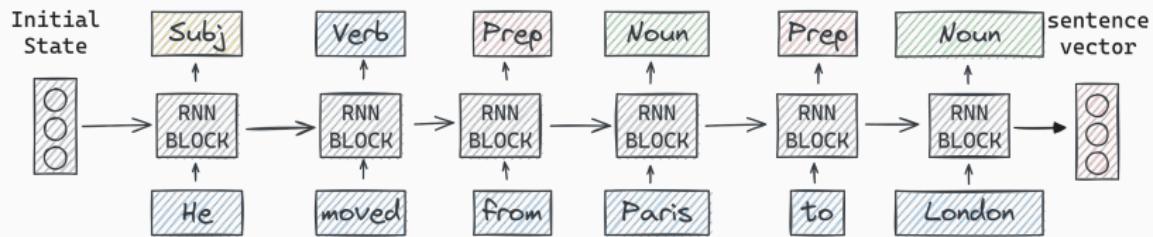


Figure 9: The many to many Variant

### Many to Many RNN

Branch a classifier (or a neural network) on top of every  $h_t \forall 1 \leq t \leq T$ .

→ In Part of speech tagging

## Contextualize word in sentence: Vanilla RNN

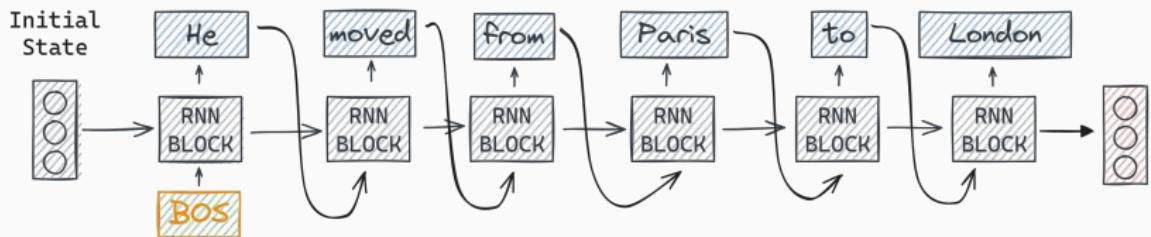


Figure 10: The many to many Variant

### One to Many RNN

Given an input  $x_1$  generate an output

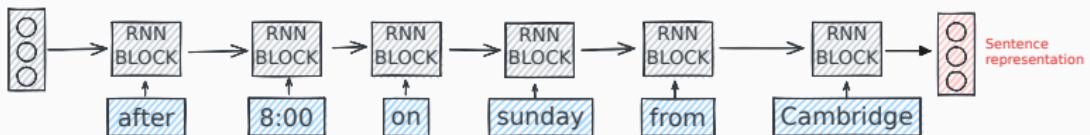
→ In generative task

# Contextualize word in sentence: Seq2Seq with RNN

## Sequence to Sequence

Model a sequence to sequence task with RNN :

- Use “*many to one*” variant to encode the input sequence
- Use “*one to many*” variant to generate a new sequence

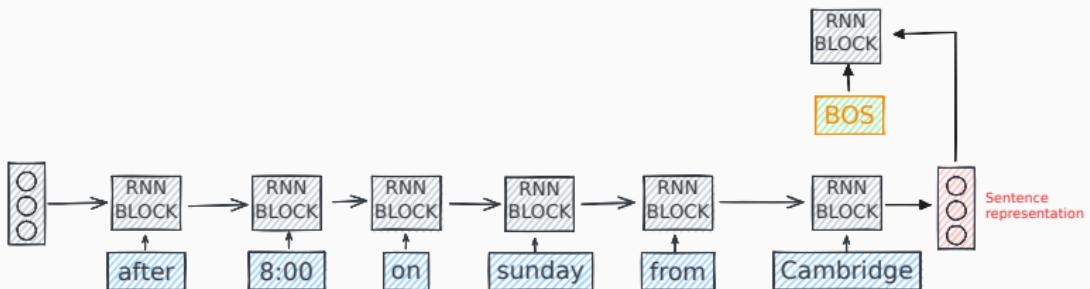


# Contextualize word in sentence: Seq2Seq with RNN

## Sequence to Sequence

Model a sequence to sequence task with RNN :

- Use “*many to one*” variant to encode the input sequence
- Use “*one to many*” variant to generate a new sequence

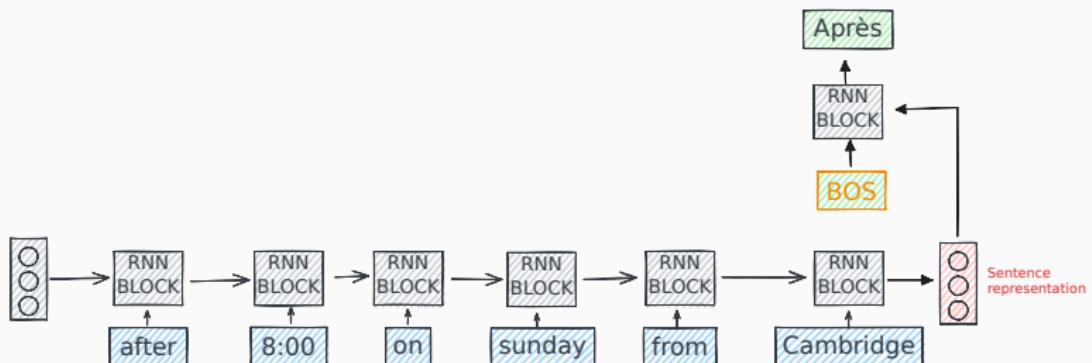


# Contextualize word in sentence: Seq2Seq with RNN

## Sequence to Sequence

Model a sequence to sequence task with RNN :

- Use “*many to one*” variant to encode the input sequence
- Use “*one to many*” variant to generate a new sequence

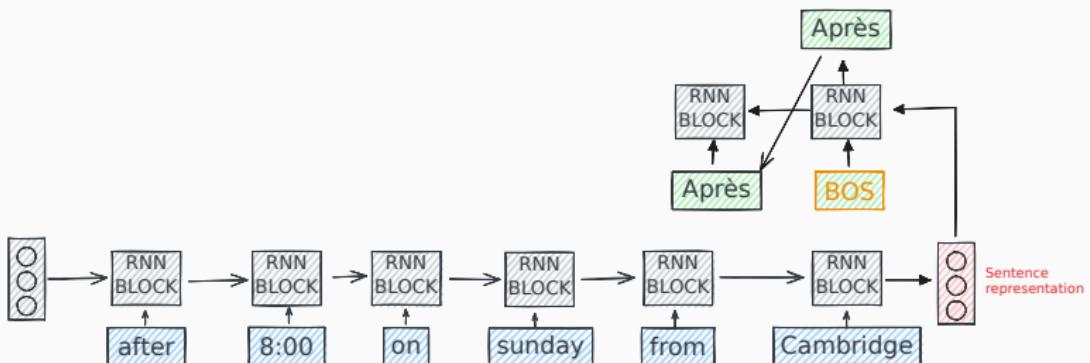


# Contextualize word in sentence: Seq2Seq with RNN

## Sequence to Sequence

Model a sequence to sequence task with RNN :

- Use “many to one” variant to encode the input sequence
- Use “one to many” variant to generate a new sequence

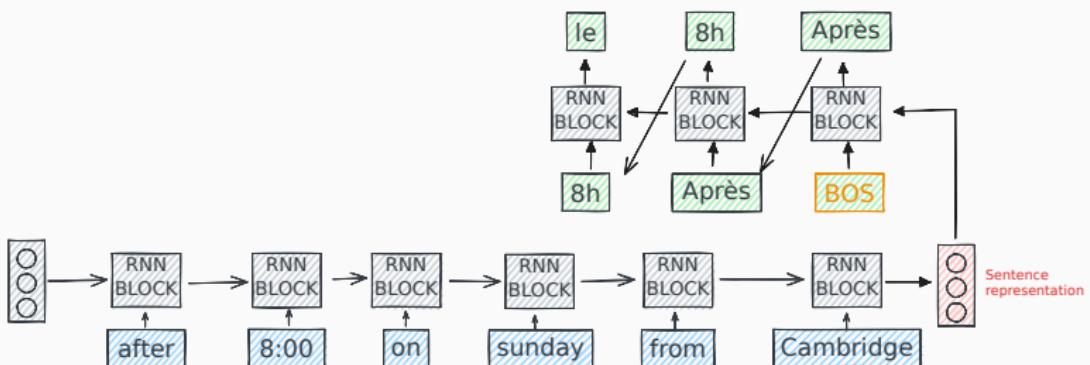


# Contextualize word in sentence: Seq2Seq with RNN

## Sequence to Sequence

Model a sequence to sequence task with RNN :

- Use “many to one” variant to encode the input sequence
- Use “one to many” variant to generate a new sequence

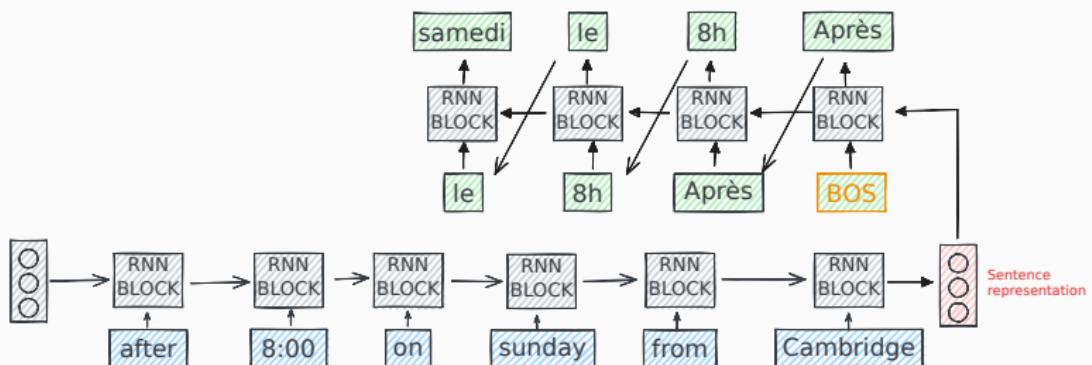


# Contextualize word in sentence: Seq2Seq with RNN

## Sequence to Sequence

Model a sequence to sequence task with RNN :

- Use “many to one” variant to encode the input sequence
- Use “one to many” variant to generate a new sequence

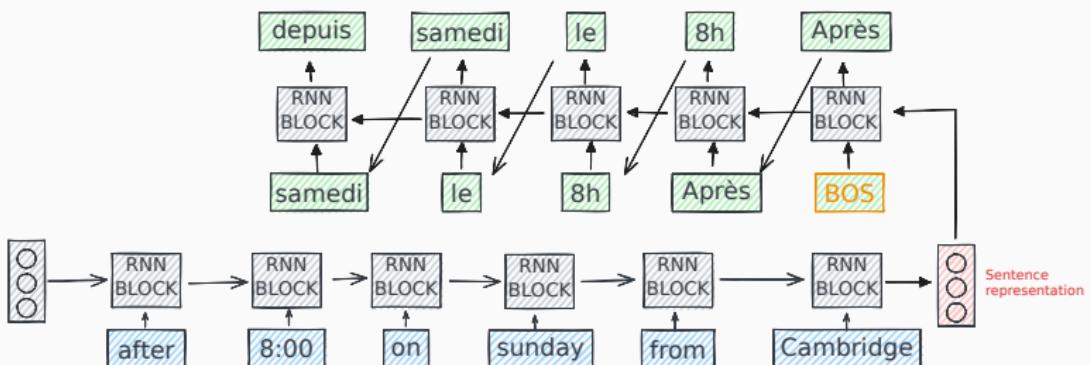


# Contextualize word in sentence: Seq2Seq with RNN

## Sequence to Sequence

Model a sequence to sequence task with RNN :

- Use “many to one” variant to encode the input sequence
- Use “one to many” variant to generate a new sequence

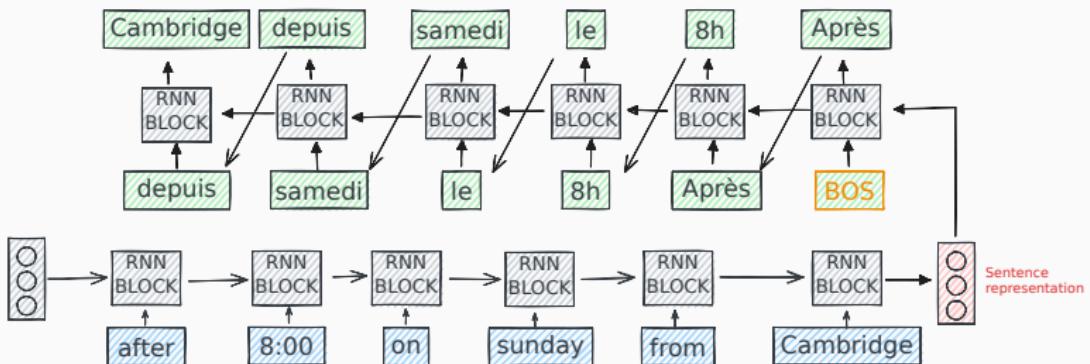


# Contextualize word in sentence: Seq2Seq with RNN

## Sequence to Sequence

Model a sequence to sequence task with RNN :

- Use “many to one” variant to encode the input sequence
- Use “one to many” variant to generate a new sequence

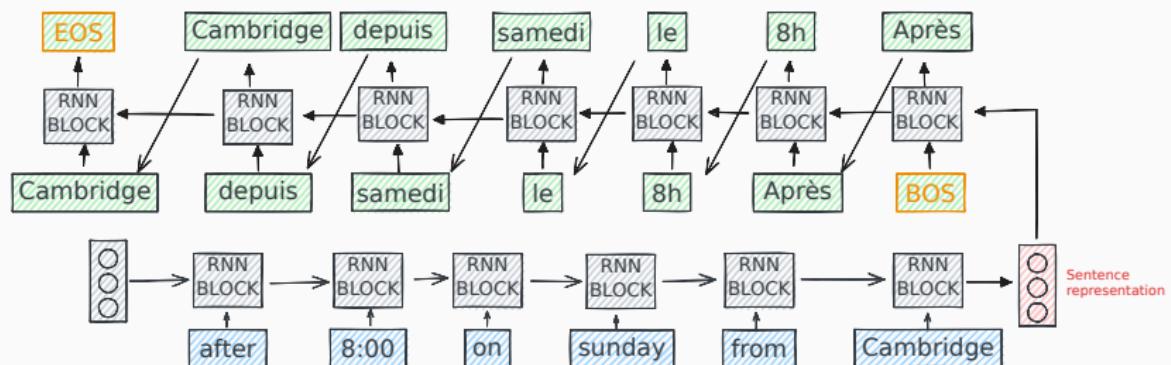


# Contextualize word in sentence: Seq2Seq with RNN

## Sequence to Sequence

Model a sequence to sequence task with RNN :

- Use “many to one” variant to encode the input sequence
- Use “one to many” variant to generate a new sequence

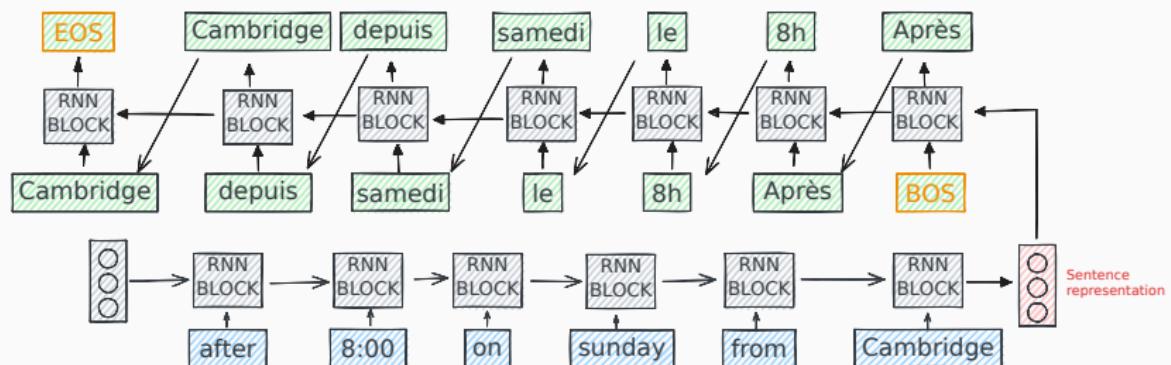


# Contextualize word in sentence: Seq2Seq with RNN

## Sequence to Sequence

Model a sequence to sequence task with RNN :

- Use “many to one” variant to encode the input sequence
- Use “one to many” variant to generate a new sequence



**NB:** The EOS word is a special word indicating the End Of the Sentence

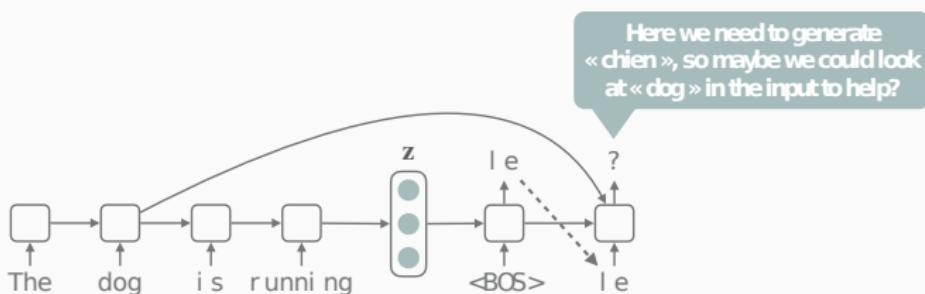
## Intuition

- During decoding, we want to look at the input sentence
- Particularly, we want to focus on specific words

# Sequence to Sequence with attention

## Intuition

- During decoding, we want to look at the input sentence
- Particularly, we want to focus on specific words



## Attention Mechanism

Adding a “module” learning to look at a word from the input

## Unidirectional dependency

Let consider the classification problem where given a sentence we predict for each word (using many-to-many variant):

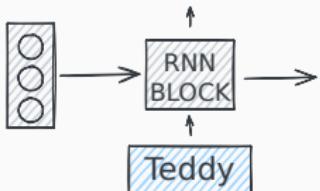
- The word correspond to a person: Pers
- The word does not correspond to a person: NPers

# Contextualize word in sentence: RNN unidirection issue

## Unidirectionnal dependency

Let consider the classification problem where given a sentence we predict for each word (using many-to-many variant):

- The word correspond to a person: Pers
- The word does not correspond to a person: NPers

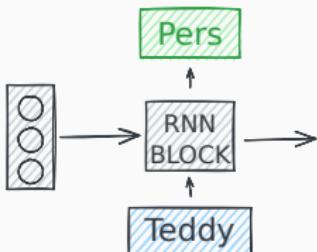


# Contextualize word in sentence: RNN unidirection issue

## Unidirectionnal dependency

Let consider the classification problem where given a sentence we predict for each word (using many-to-many variant):

- The word correspond to a person: Pers
- The word does not correspond to a person: NPers

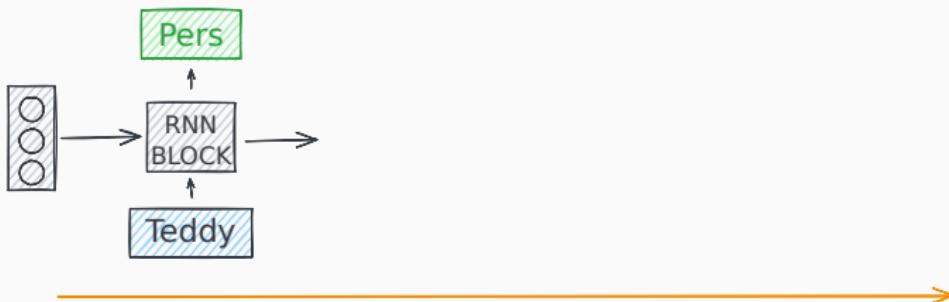


# Contextualize word in sentence: RNN unidirection issue

## Unidirectional dependency

Let consider the classification problem where given a sentence we predict for each word (using many-to-many variant):

- The word correspond to a person: Pers
- The word does not correspond to a person: NPers



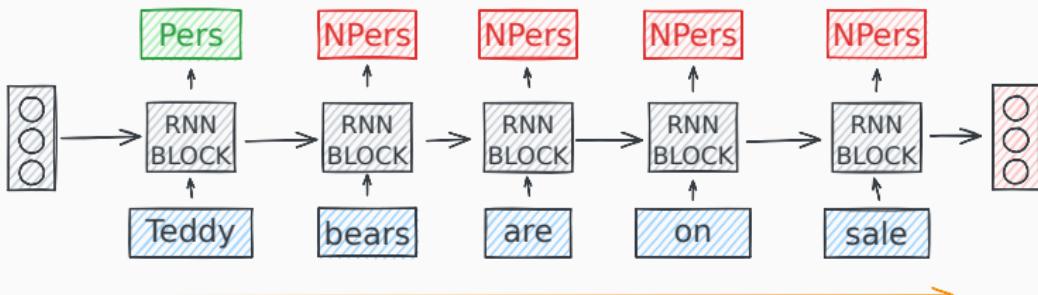
Direction of dependency we can capture

# Contextualize word in sentence: RNN unidirection issue

## Unidirectional dependency

Let consider the classification problem where given a sentence we predict for each word (using many-to-many variant):

- The word correspond to a person: Pers
- The word does not correspond to a person: NPers



Direction of dependency we can capture

## Contextualize word in sentence: RNN unidirection issue

→ For some task it is necessary to know the next words before prediction

## Contextualize word in sentence: RNN unidirection issue

→ For some task it is necessary to know the next words before prediction

**Let consider two RNNs:**

- The first RNN processing forward the sequence and produce hidden state  $\{h_1, h_2, \dots, h_T\}$
- A second RNN processing the sequence backward and produce hidden state  $\{h'_T, h'_{T-1+1}, \dots, h'_1\}$

## Contextualize word in sentence: RNN unidirection issue

→ For some task it is necessary to know the next words before prediction

Let consider two RNNs:

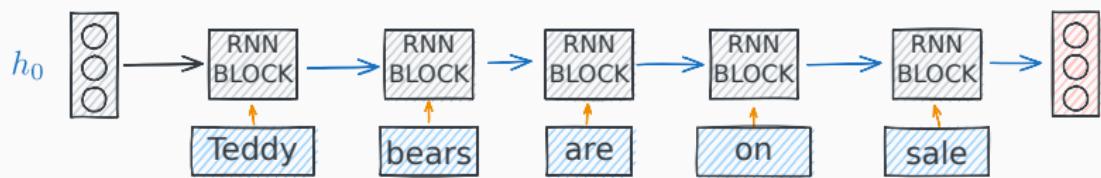
- The first RNN processing forward the sequence and produce hidden state  $\{h_1, h_2, \dots, h_T\}$
- A second RNN processing the sequence backward and produce hidden state  $\{h'_T, h'_{T-1+1}, \dots, h'_1\}$

The prediction of the  $t^{th}$  word is performed by a classifier :

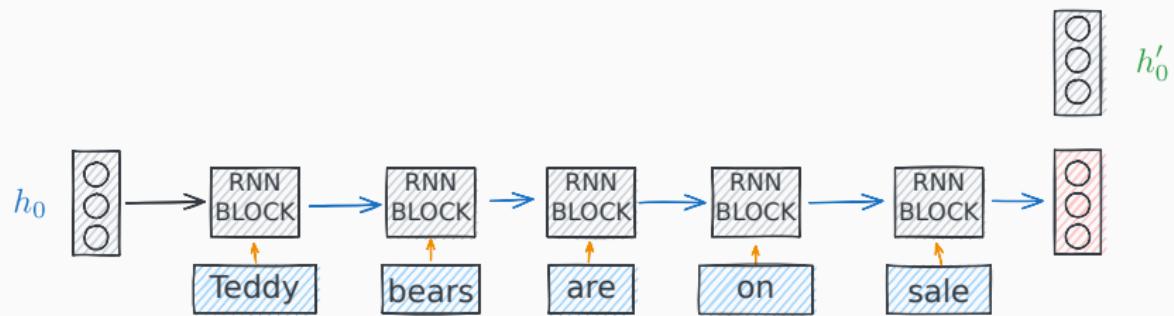
$$f_\theta : \mathbb{R}^d \times \mathbb{R}^d \rightarrow [-1, 1]$$
$$(h_t, h'_{T-t+1}) \mapsto \tanh(W_{f_1} h_t + W_{f_2} h'_{T-t+1} + b_f)$$

And  $f(h_t, h'_{T-t+1}) \geq 0$  means that the  $t^{th}$  word refer to a person

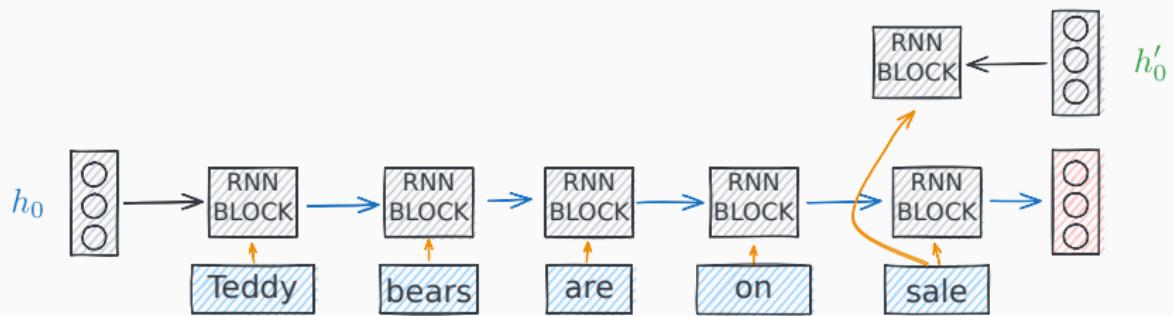
## Contextualize word in sentence: RNN unidirection issue



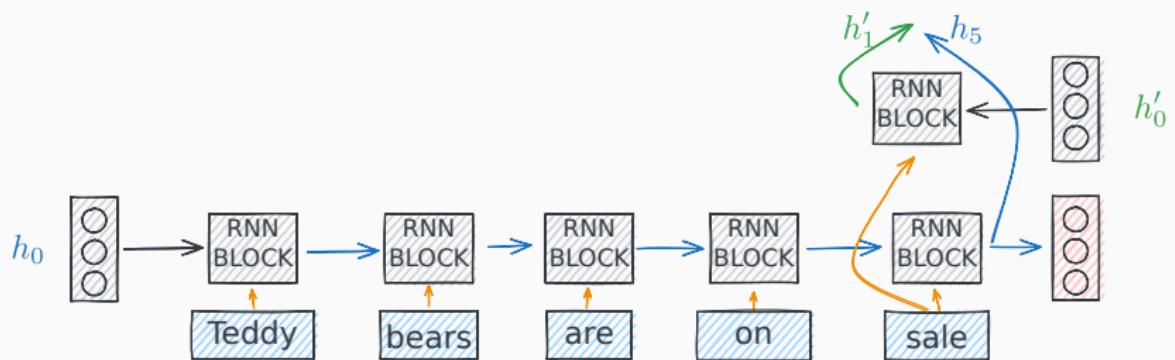
## Contextualize word in sentence: RNN unidirection issue



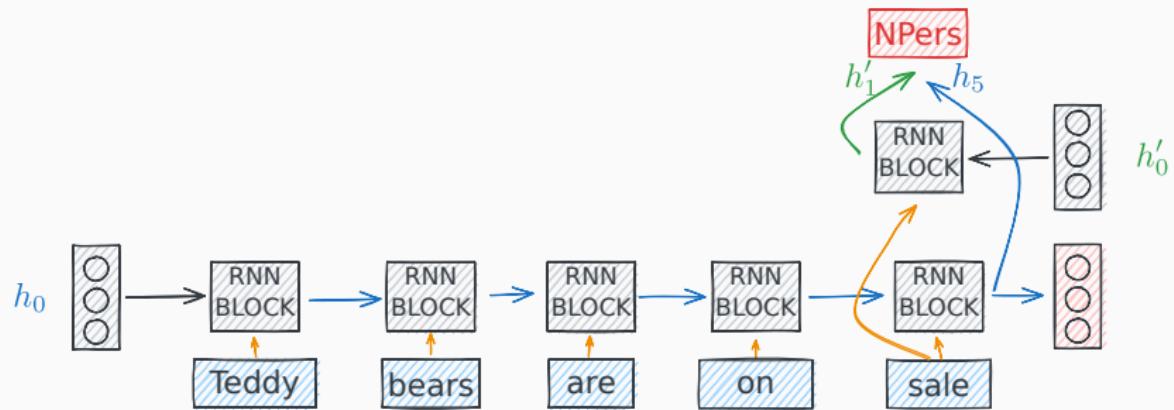
## Contextualize word in sentence: RNN unidirection issue



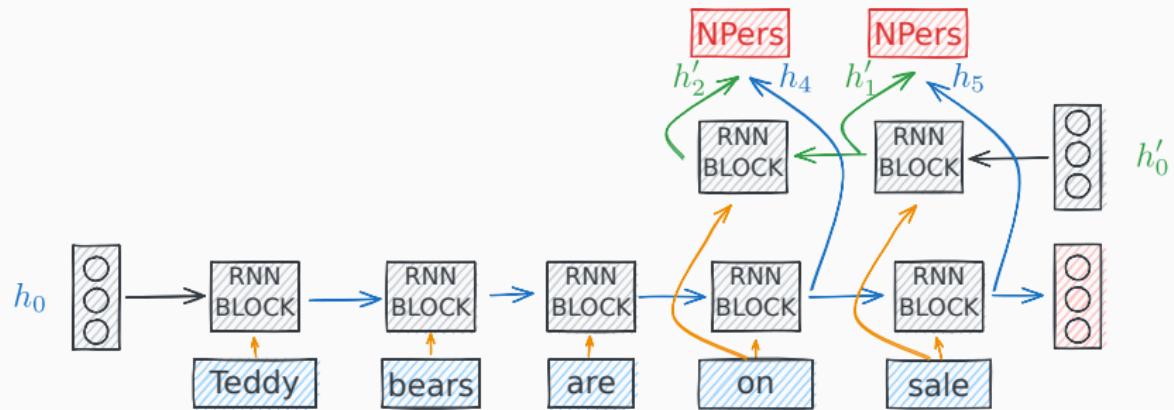
## Contextualize word in sentence: RNN unidirection issue



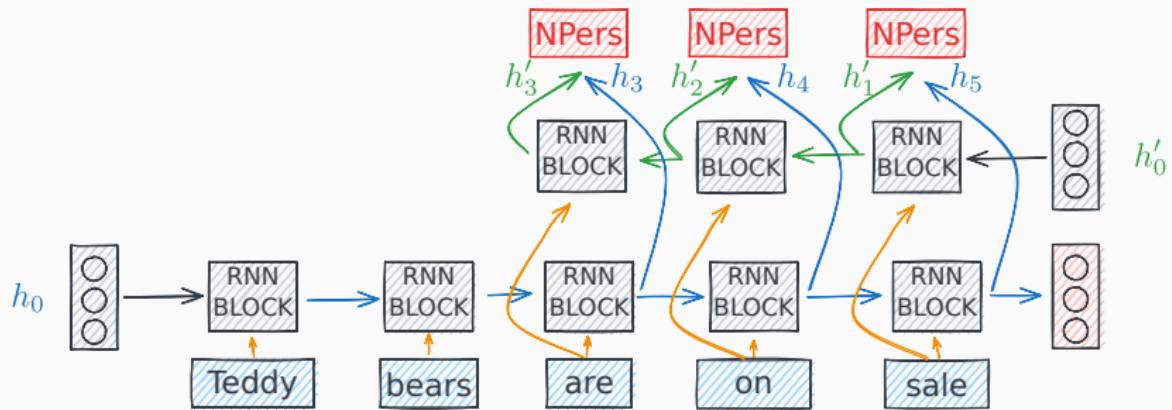
## Contextualize word in sentence: RNN unidirection issue



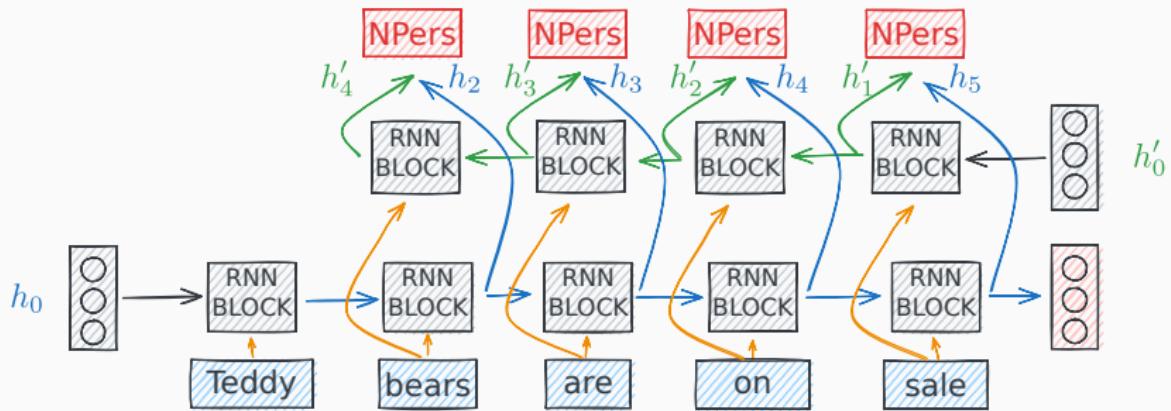
## Contextualize word in sentence: RNN unidirection issue



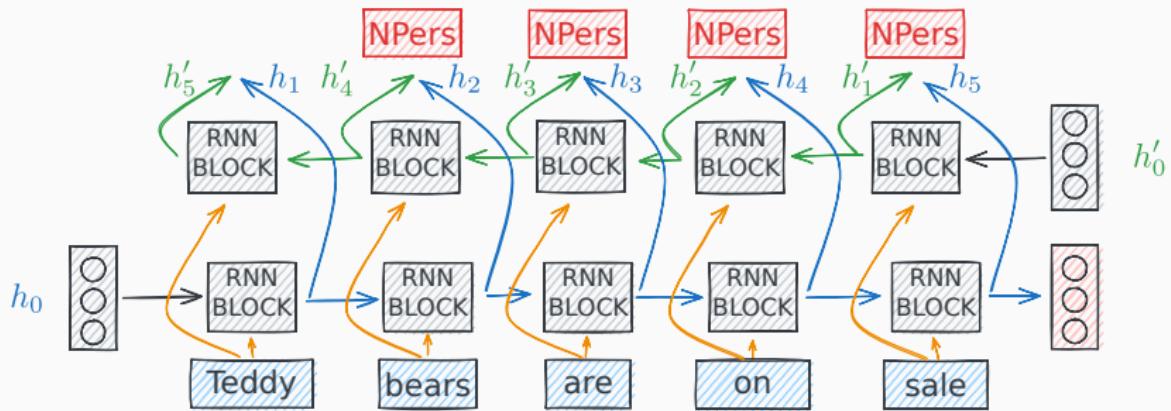
## Contextualize word in sentence: RNN unidirection issue



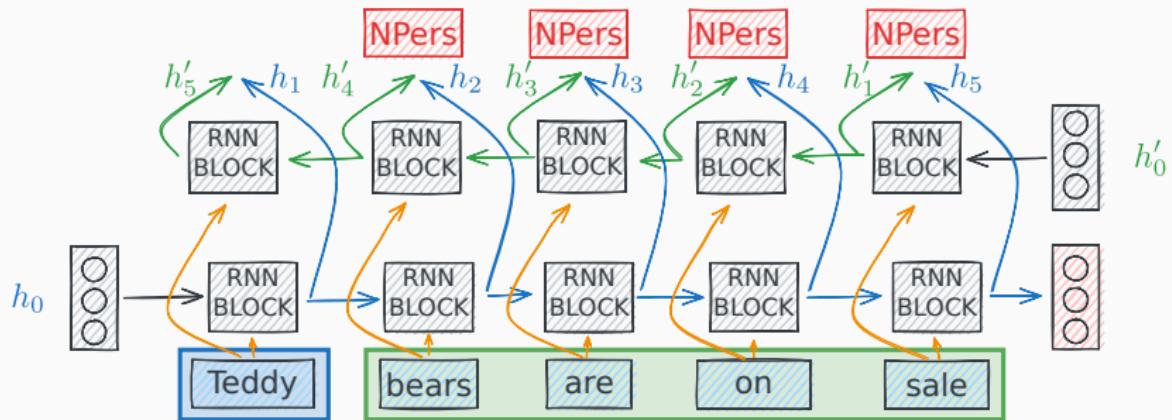
## Contextualize word in sentence: RNN unidirection issue



## Contextualize word in sentence: RNN unidirection issue



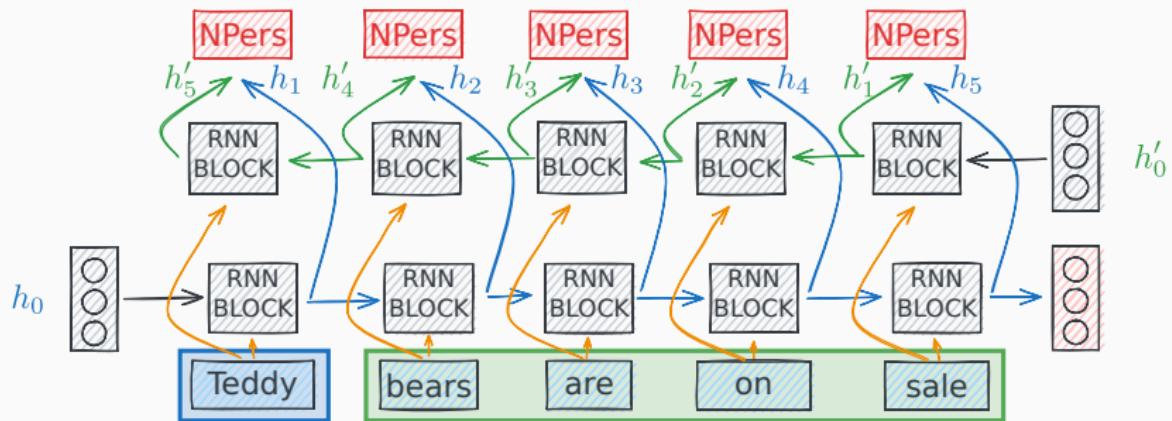
## Contextualize word in sentence: RNN unidirection issue



Information of backward state and forward state

→ Information at each state of the whole sequence

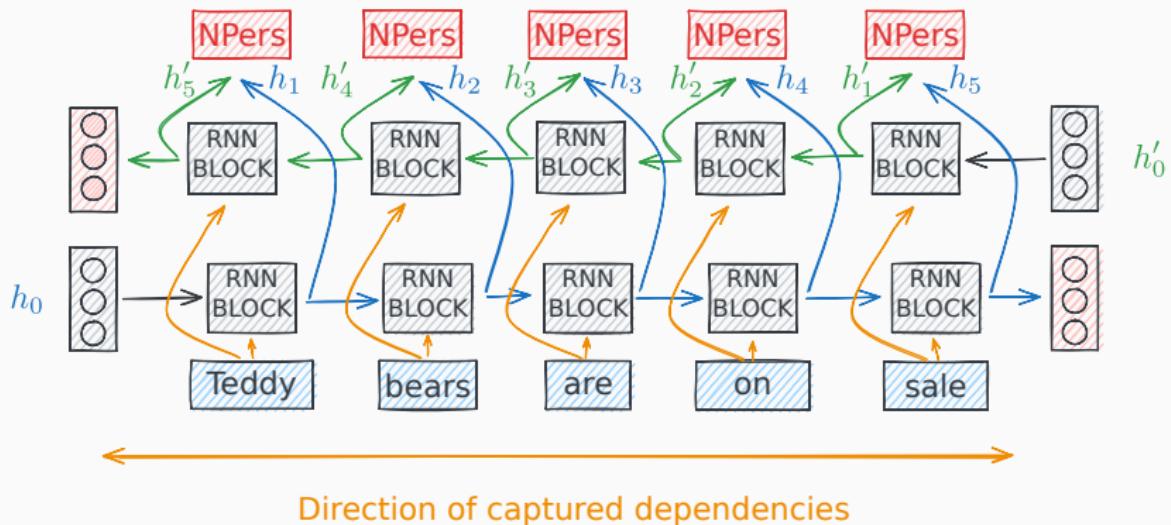
## Contextualize word in sentence: RNN unidirection issue



Information of backward state and forward state

→ Information at each state of the whole sequence

## Contextualize word in sentence: RNN unidirection issue



Information of backward state and forward state

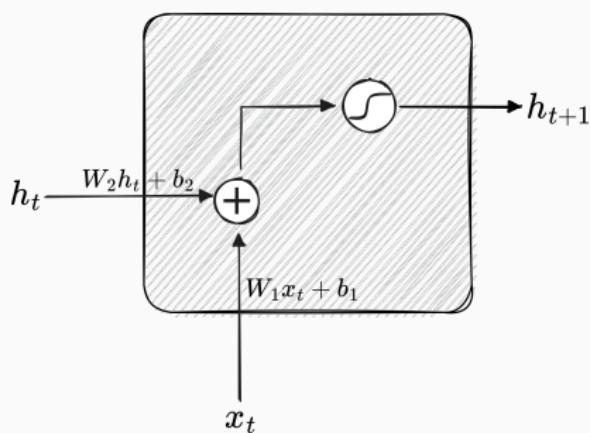
→ Information at each state of the whole sequence

## Contextualize word in sentence: Vanilla RNN block

### Issue with Vanilla RNN

Difficult to train and mitigate results with  
Vanilla RNN

- **Short term memory:** Works on slightly short sequences
- **The Vanishing Gradient problem:** The gradient become too small for first element of the sequence → Only last element of the sequence influence the prediction
- **The Exploding Gradient problem:** The gradient become too large for first element of the sequence



It depends on the matrix  $W_h$ ...

## Vanilla RNN:

→ Only for short sequence

## Vanilla RNN:

- Only for short sequence
- Change the architecture of block?

## Vanilla RNN:

- Only for short sequence
- Change the architecture of block?

## Long Short Term Memory Cell:

Two “memory”, a long term (cell state)  
and a short term (hidden state)

→ “Long Short-Term Memory” Hochreiter et al. 1997

# Contextualize word in sentence: Recurrent Neural Networks

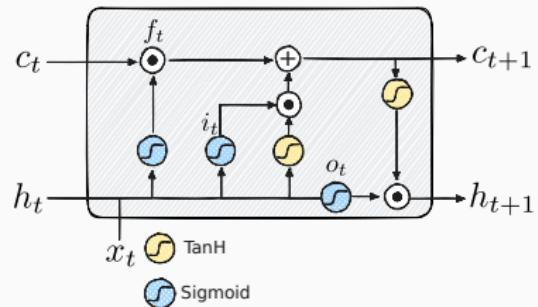
## Vanilla RNN:

- Only for short sequence
- Change the architecture of block?

## Long Short Term Memory Cell:

Two “memory”, a long term (cell state) and a short term (hidden state)

→ “Long Short-Term Memory” Hochreiter et al. 1997



- Adding a new state  $c_t$  (cell state)
- Adding a “forget gate” to decide “what to forget”

# Contextualize word in sentence: Recurrent Neural Networks

## Vanilla RNN:

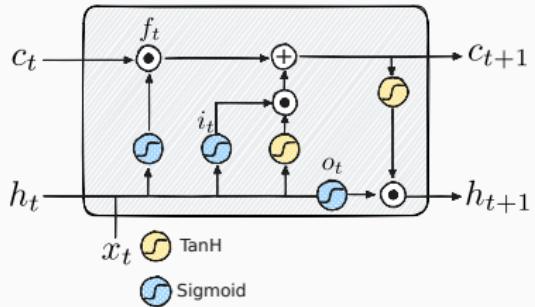
- Only for short sequence
- Change the architecture of block?

## Long Short Term Memory Cell:

Two “memory”, a long term (cell state) and a short term (hidden state)

→ “Long Short-Term Memory” Hochreiter et al. 1997

- Adding a new state  $c_t$  (cell state)
- Adding a “forget gate” to decide “what to forget”



$$f_t = \sigma(W_{if}x_t + W_{hf}h_t + b_f)$$

$$i_t = \sigma(W_{ii}x_t + W_{hi}h_t + b_i)$$

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_t + b_g)$$

$$o_t = \sigma(W_{io}x_t + W_{ho}h_t + b_o)$$

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

$$h_{t+1} = o_t \odot \tanh(c_{t+1})$$

## Vanilla RNN:

→ Only for short sequence

## Vanilla RNN:

- Only for short sequence
- Change the architecture of block?

## Vanilla RNN:

- Only for short sequence
- Change the architecture of block?

## Long Short Term Memory Cell:

Two “memory”, a long term (cell state)  
and a short term (hidden state)

→ “Long Short-Term Memory” Hochreiter et al. 1997

# Contextualize word in sentence: Recurrent Neural Networks

## Vanilla RNN:

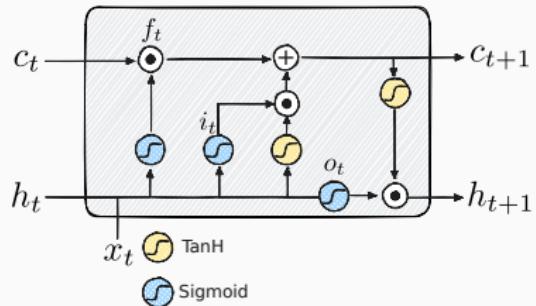
- Only for short sequence
- Change the architecture of block?

## Long Short Term Memory Cell:

Two “memory”, a long term (cell state) and a short term (hidden state)

→ “Long Short-Term Memory” Hochreiter et al. 1997

- Adding a new state  $c_t$  (cell state)
- Adding a “forget gate” to decide “what to forget”



# Contextualize word in sentence: Recurrent Neural Networks

## Vanilla RNN:

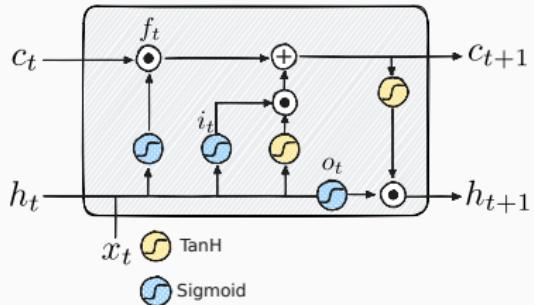
- Only for short sequence
- Change the architecture of block?

## Long Short Term Memory Cell:

Two “memory”, a long term (cell state) and a short term (hidden state)

→ “Long Short-Term Memory” Hochreiter et al. 1997

- Adding a new state  $c_t$  (cell state)
- Adding a “forget gate” to decide “what to forget”



$$f_t = \sigma(W_{if}x_t + W_{hf}h_t + b_f)$$

$$i_t = \sigma(W_{ii}x_t + W_{hi}h_t + b_i)$$

$$g_t = \tanh(W_{ig}x_t + W_{hg}h_t + b_g)$$

$$o_t = \sigma(W_{io}x_t + W_{ho}h_t + b_o)$$

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

$$h_{t+1} = o_t \odot \tanh(c_{t+1})$$