Introduction to Deep Learning

Thomas Gerald

March 12, 2026

Laboratoire Interdisciplinaire des Sciences du Numérique – LISN, CNRS

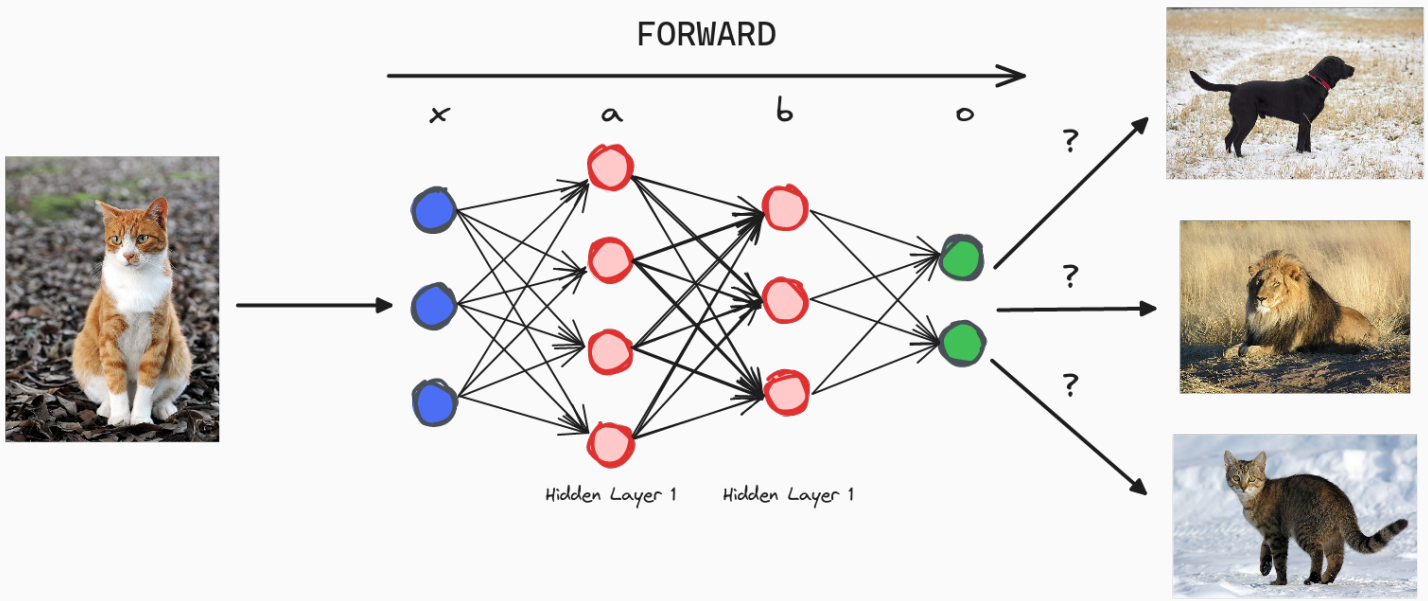
The class

The lectures

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

Grading

→ 50% exercises (reports)
→ 50% exam



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

Schedule

Thomas Gerald:

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

- → **Session 1 :** Introduction to Deep-Learning (lecture only)
- → **Session 2 :** Training Neural Networks (lecture + exercises)
- → **Session 3 :** Regularization and Optimisation and Neural Networks Libraries (lecture + lab exercises)
- → **Session 4 :** Convolutional Neural Network and Auto-Encoder (lecture +lab exercises)
- → **Session 5 :** Generative approaches (VAE, GAN and diffusion models) (lecture + lab exercises)
- → **Session 6 :** Recent architectures and challenges (Project)
- → **Session 7 :** Exam

Requirements

Background

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

Programming

- → Python, jupyter notebook
- → Numpy, matplotlib, pytorch

Ressources

→ Ressources will be made available at
`https://thomas-gerald.fr/DL/index.html`

Content

→ Machine Learning: The principle

→ Linear classification: definition and issues (convex set)

→ Representation learning and the multi-layer perceptron

Objectives

→ What is deep-learning ?

→ How does it work ?

What is Machine Learning?

Machine learning:

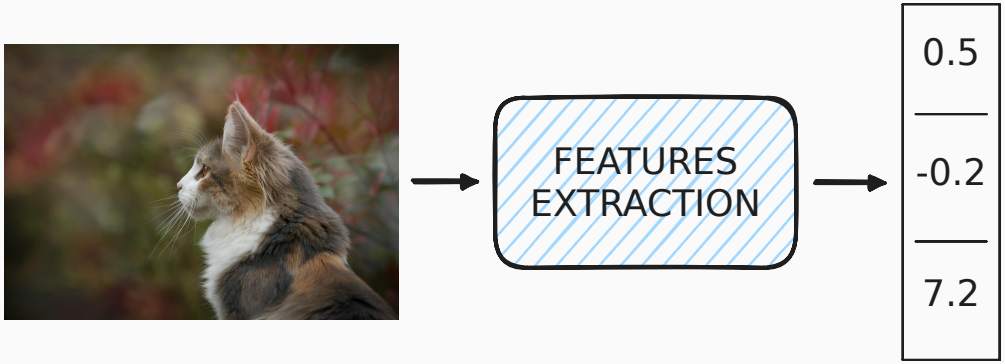
- "the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data"

`https://it.wustl.edu/ai/`

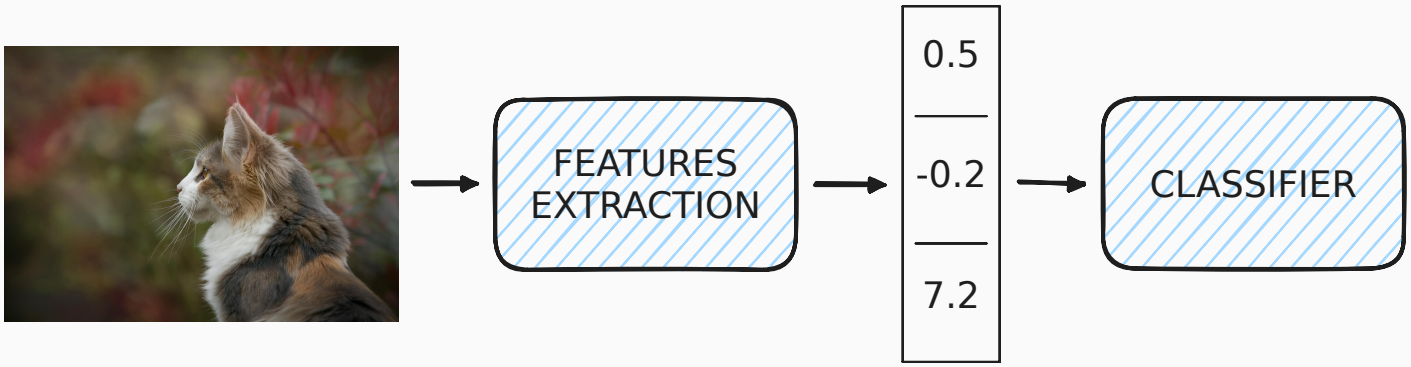
Before Deep Learning

Before Deep Learning

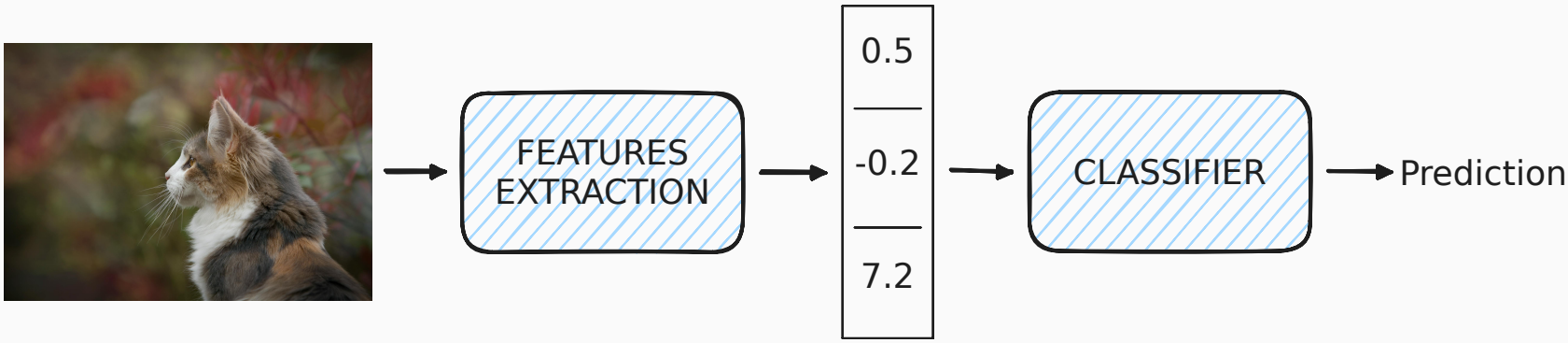
Before Deep Learning



Before Deep Learning



Before Deep Learning

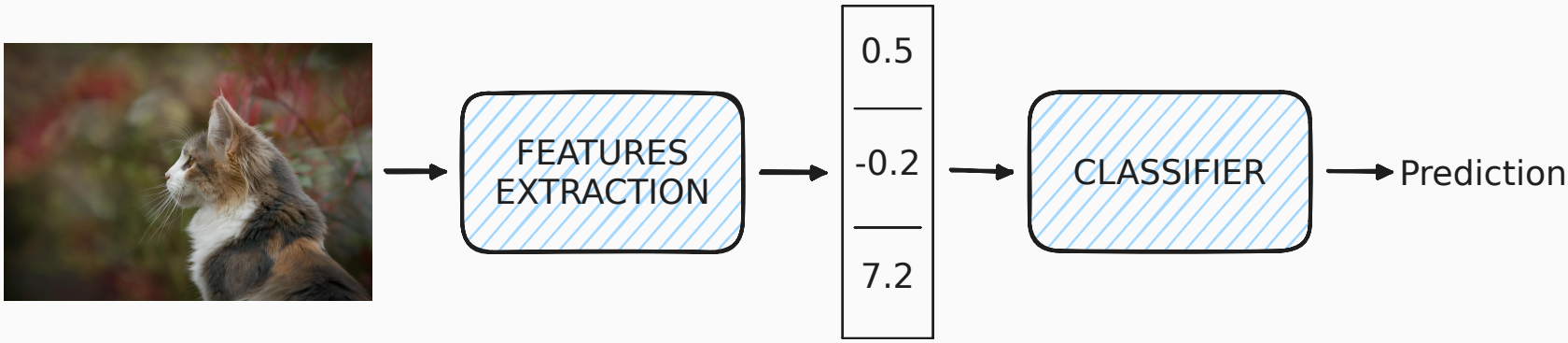


The extraction and classification pipeline

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

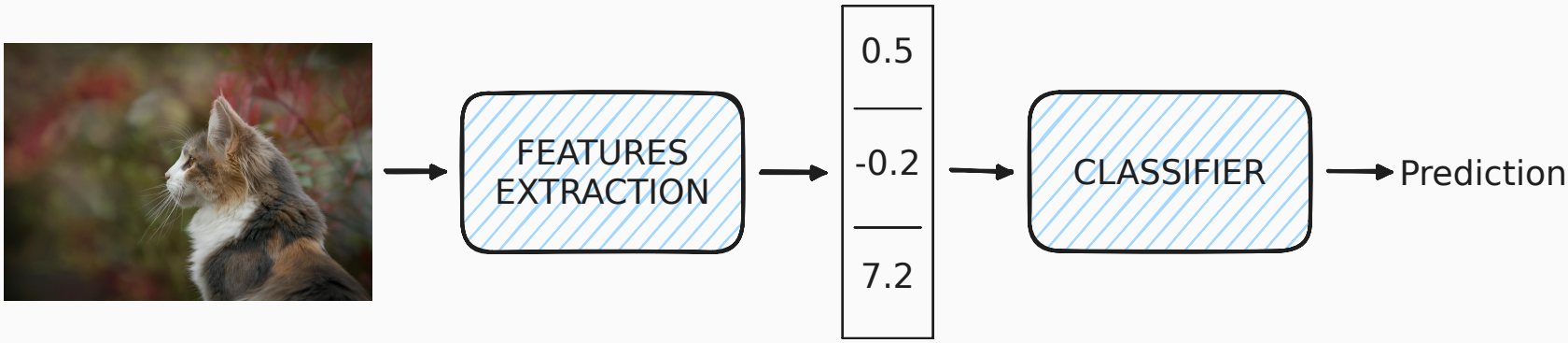
→ **Images:** Saillant point extraction, colors histograms ...

→ **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 ...

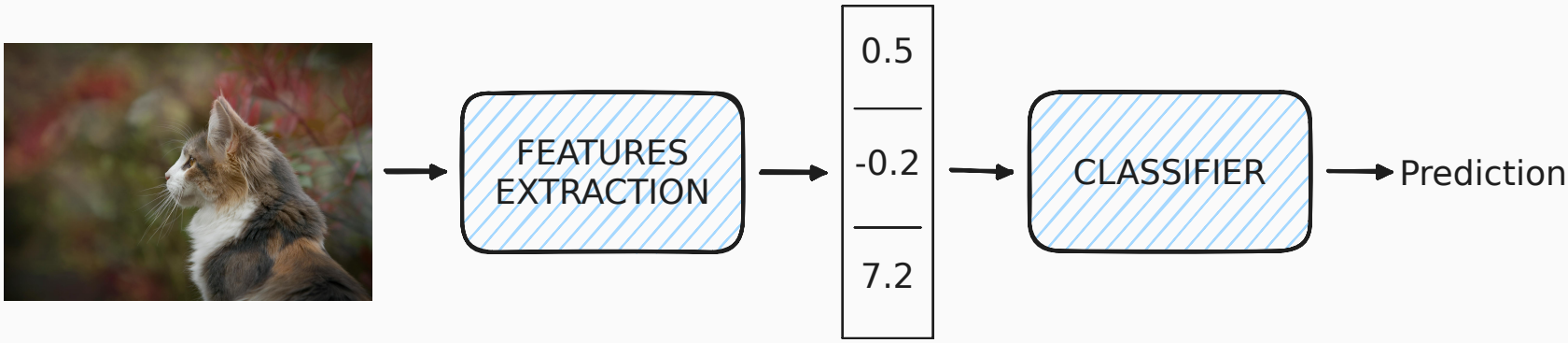
→ **Texts:** Lemmas, stems, PoS (part of speech) ...

→ 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 $f_\Theta : \mathcal{X} \to \mathcal{Y}$

Notations

Classifier

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

$\to$ Θ parameters (often written θ)

$\to$ $\mathcal{X}$ the input/features space

$\to$ $\mathcal{Y}$ the output space/labels space

Notations

Classifier

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

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

Dataset

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

Notations

Classifier

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

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

Dataset

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

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

Notations

Classifier

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

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

Different type of prediction

Dataset

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

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

Notations

Classifier

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

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

Different type of prediction

- $\to$ Regression (predict a value)

Dataset

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

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

Notations

Classifier

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

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

Dataset

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

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

Different type of prediction

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

Notations

Classifier

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

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

Dataset

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

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

Different type of prediction

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

Notations

Classifier

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

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

Dataset

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

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

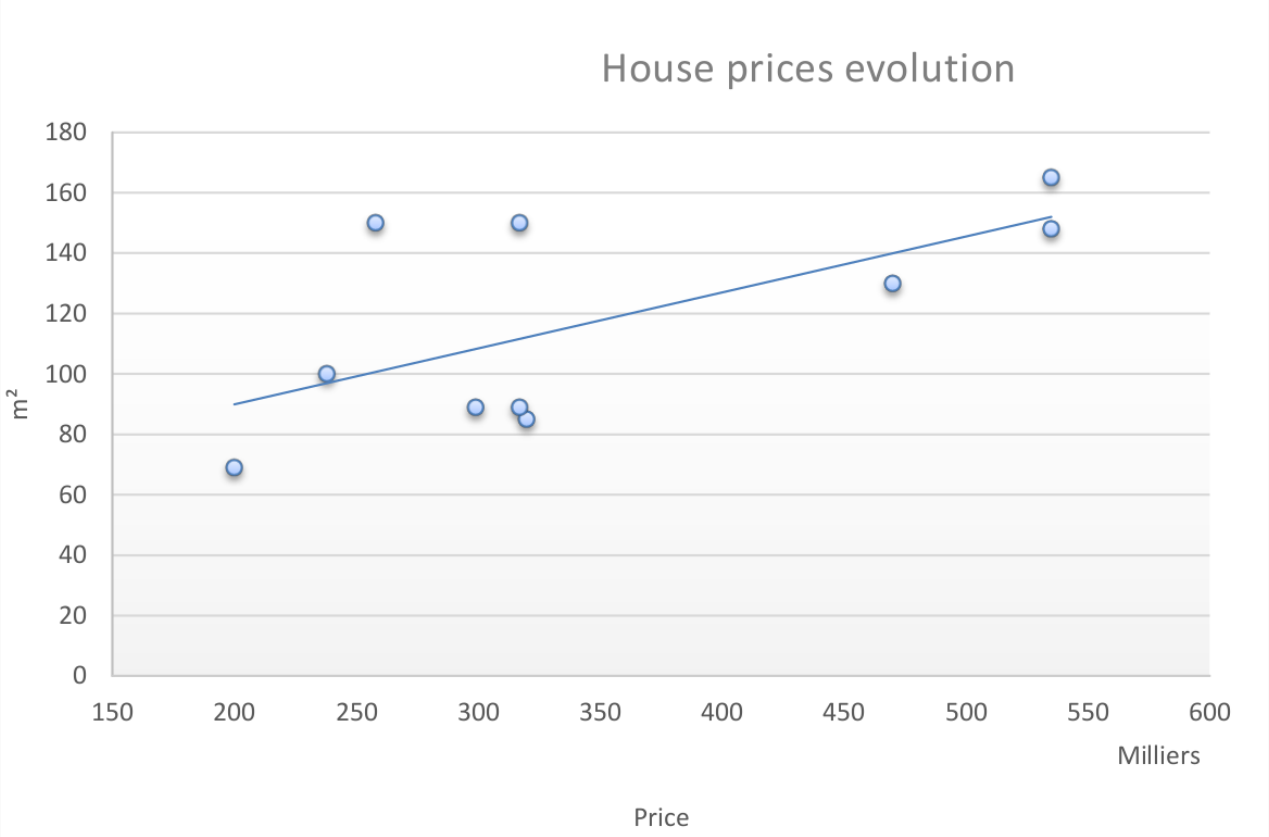
Different type of prediction

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

Regression

Predict a value considering input

→ Determine the house price considering a surface

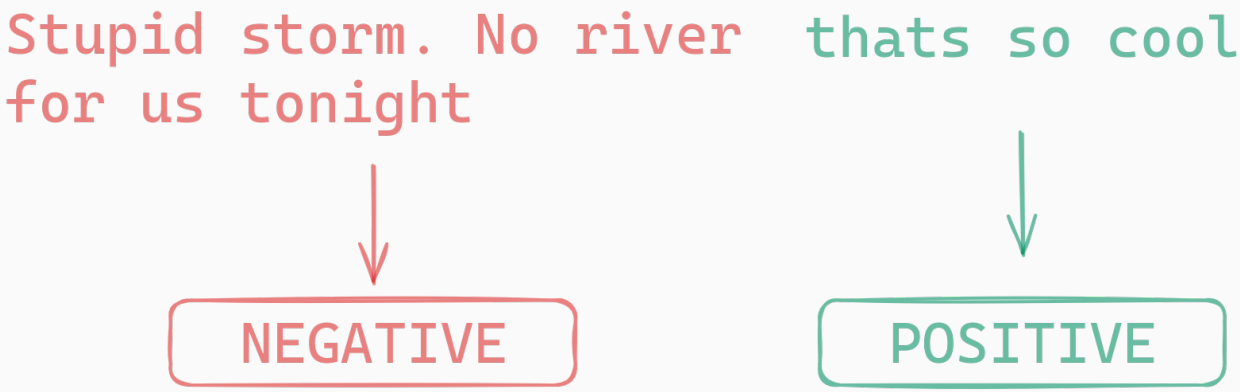


Different type of prediction: Binary classification

Binary Classification

Predict a class among two

→ Sentiment analysis (binary case)

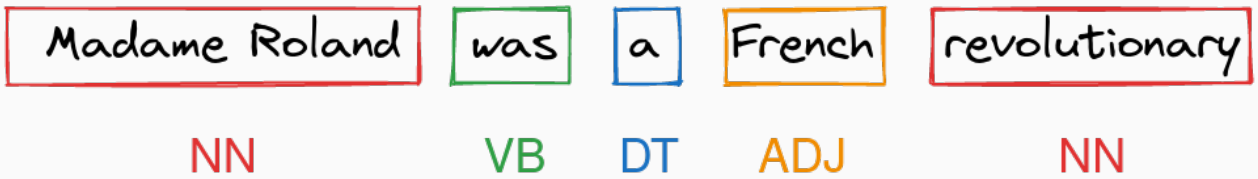


Different type of prediction: Multiclass classification

Multiclass Classification

Predict a class among many

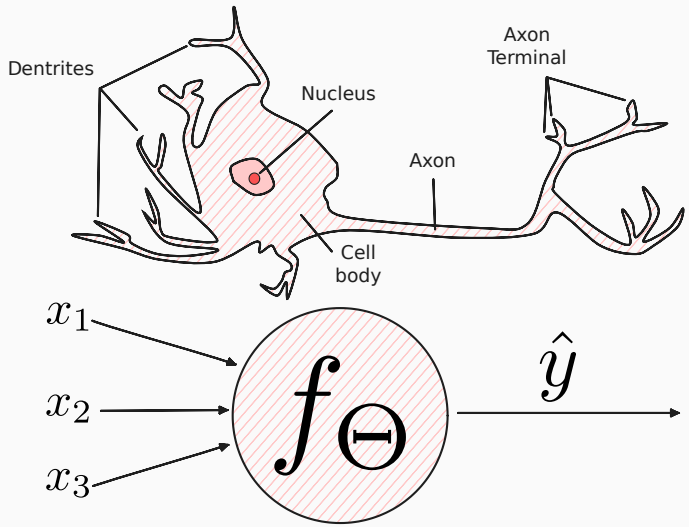
→ Determine the house price considering a surface



Inspiration or the perceptron

→ A (brain) neuron receives signals

→ Depending of the signals it trigger or not an output



Bio-inspired classifier

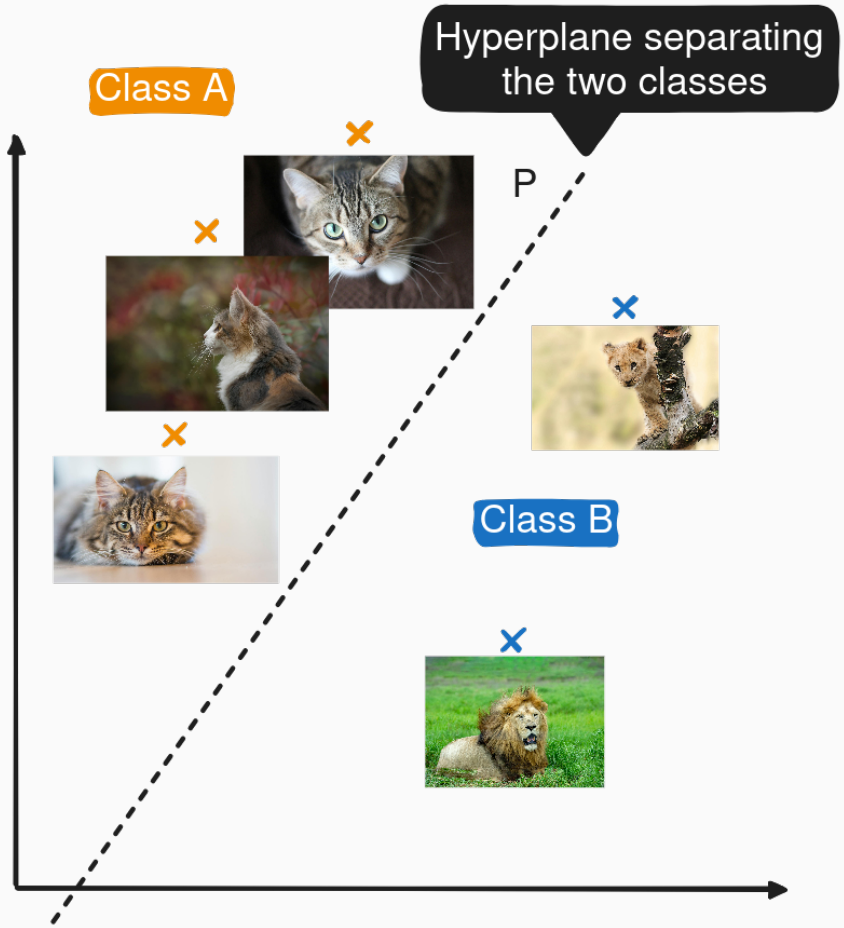
→ Perceptron/neural network neurons have most probably been inspired from biology

→ BUT they are different and work differently

→ **The biology comparison** is misleading

[0] 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:

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

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:

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

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:

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

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

$$\sum_{i=1}^{n} a_i x_i + b = 0 \iff a^\intercal x + b = 0 \iff < a, x > + b = 0$$

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:

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

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

$$\sum_{i=1}^{n} a_i x_i + b = 0 \iff a^\intercal x + b = 0 \iff < a, x > + b = 0$$

Dot product properties:

$\rightarrow a^\intercal x$ can be written as the product of magnitude of vectors and the cosinus of their angle

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

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

Linear classification: The dot product

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

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

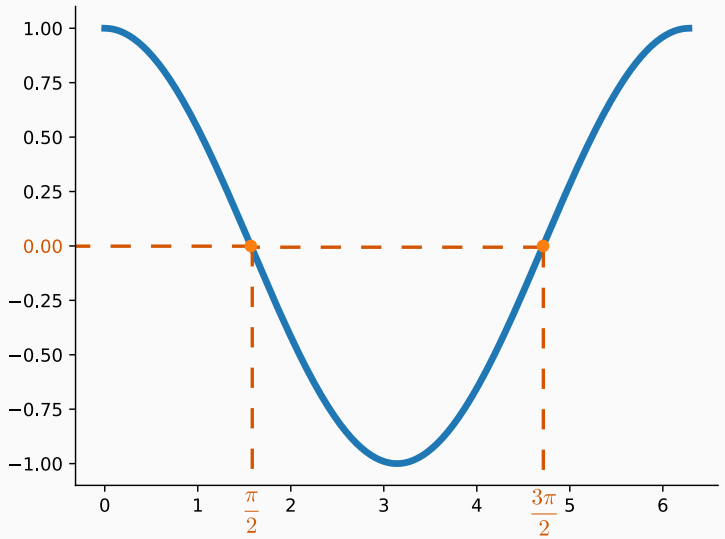
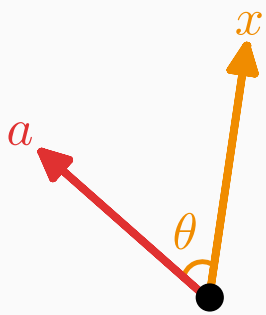


Figure 1: The cosinus function for $x \in [0, 2\pi]$



- $\theta \in]0, \frac{\pi}{2}[\cup]\frac{3\pi}{2}, 2\pi[\rightarrow$ value of $a^\intercal x$?

Linear classification: The dot product

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

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

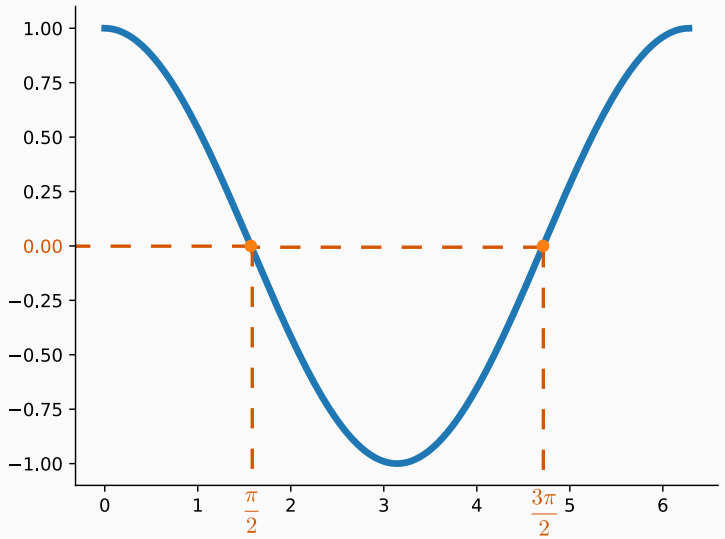
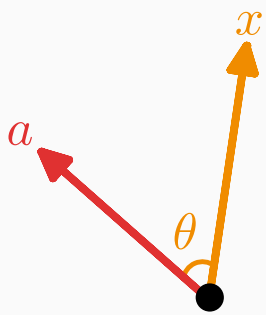


Figure 1: The cosinus function for $x \in [0, 2\pi]$



- $\theta \in]0, \frac{\pi}{2}[\cup]\frac{3\pi}{2}, 2\pi[\rightarrow$ value of $a^\intercal x$?
  $\rightarrow a^\intercal x > 0$

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

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

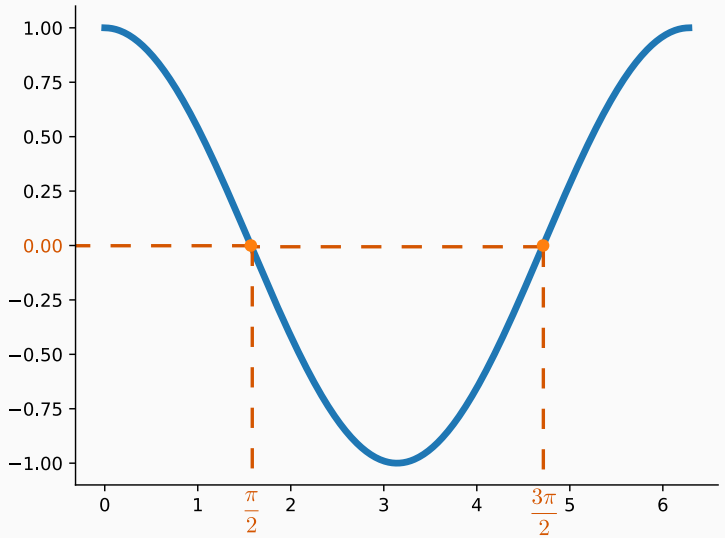
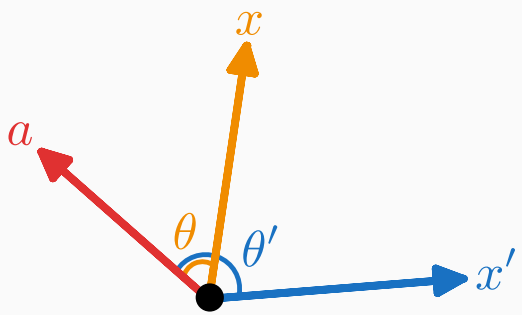


Figure 2: The cosinus function for $x \in [0, 2\pi]$



- $\theta \in]0, \frac{\pi}{2}[\cup]\frac{3\pi}{2}, 2\pi[\rightarrow$ value of $a^\intercal x$?
  $\rightarrow a^\intercal x > 0$
- $\theta \in]\frac{\pi}{2}, \frac{3\pi}{2}[\rightarrow$ value of $a^\intercal x$?

Linear classification: The dot product

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

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

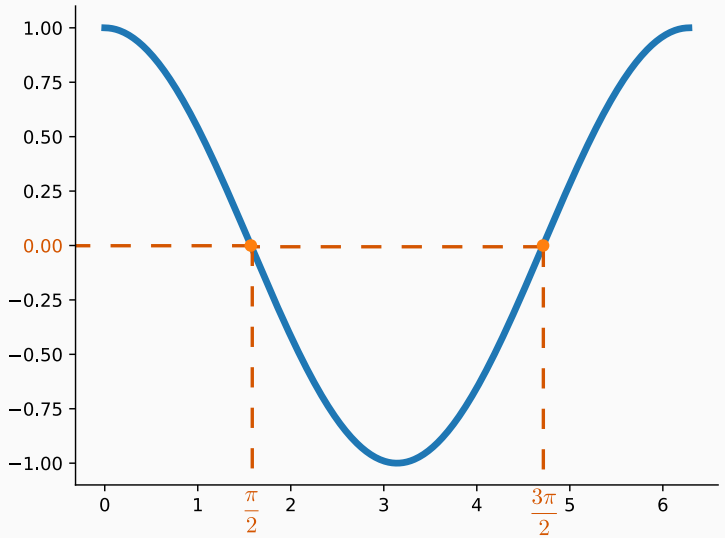
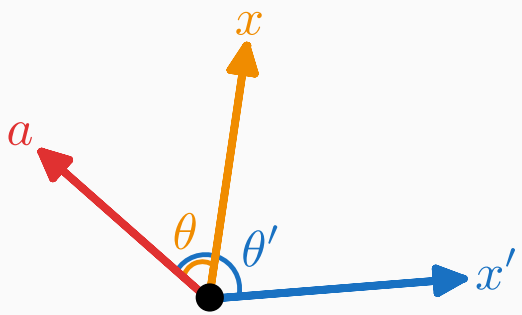


Figure 2: The cosinus function for $x \in [0, 2\pi]$



- $\theta \in]0, \frac{\pi}{2}[\cup]\frac{3\pi}{2}, 2\pi[\rightarrow$ value of $a^\intercal x$?
  $\rightarrow a^\intercal x > 0$
- $\theta \in]\frac{\pi}{2}, \frac{3\pi}{2}[\rightarrow$ value of $a^\intercal x$?
  $\rightarrow a^\intercal x < 0$

Classification function:

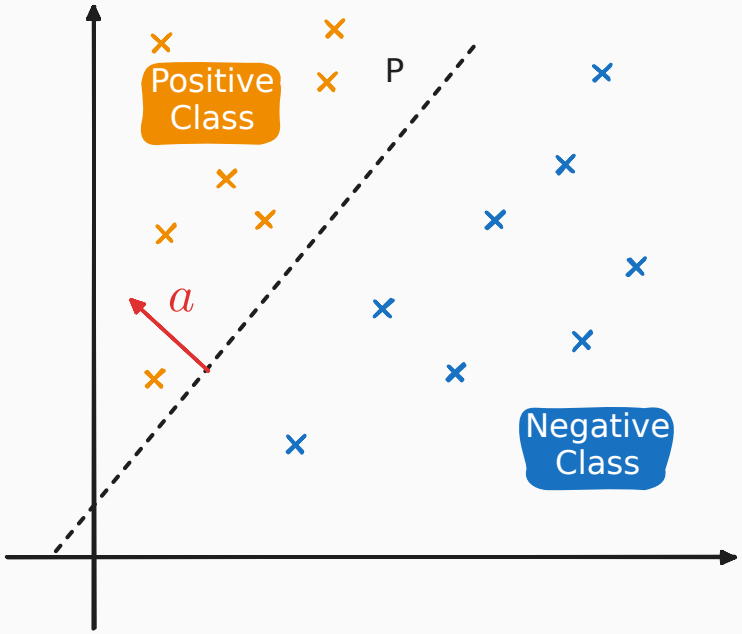
Let Θ be the parameters

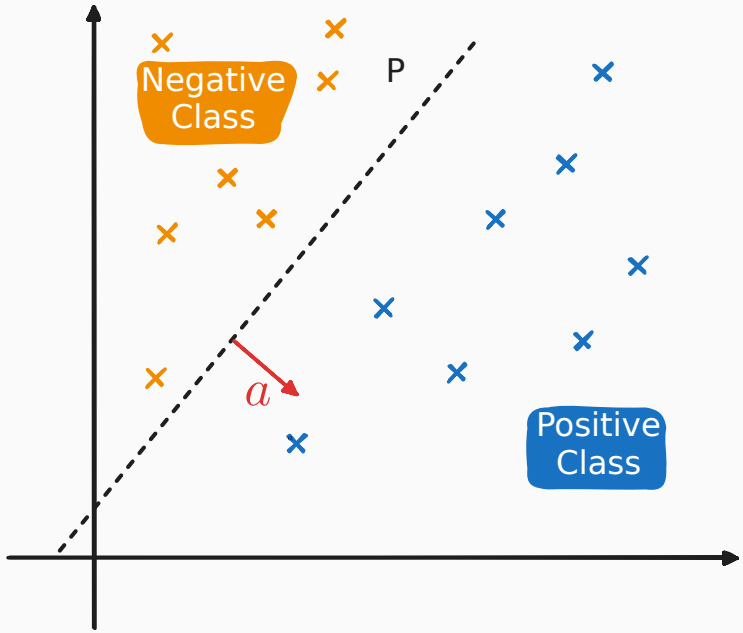
- **General Case:** $f_\Theta : \mathcal{X} \to \mathcal{Y}$ with $\mathcal{Y}$ the class set
- **Binary case:** $f_\Theta : \mathcal{X} \to \{-1, 1\}$

Perceptron

- $\Theta = \{a, b\}$ be the parameters
- Classification function:

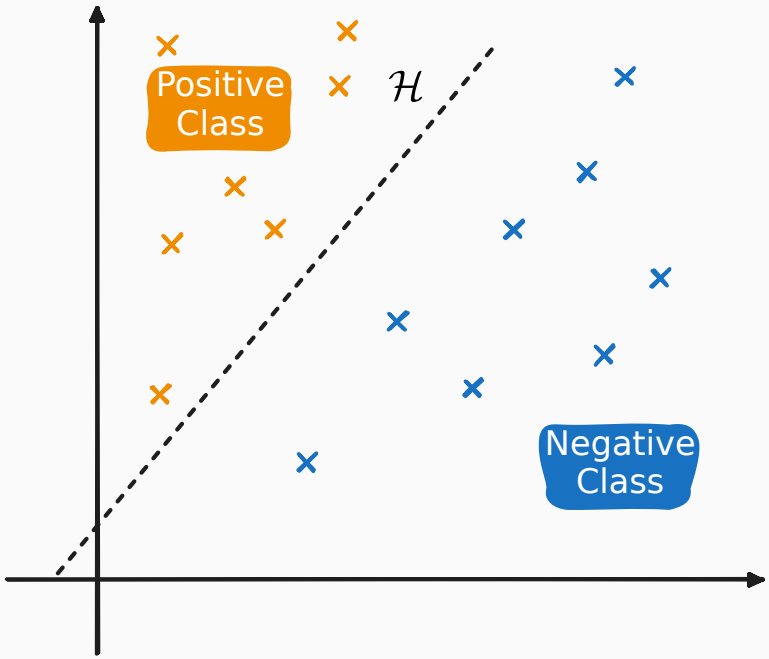
$$f_\Theta(x) = \begin{cases} 1 & \text{if } a^\intercal x + b > 0 \\ -1 & \text{if } a^\intercal x + b \leq 0 \end{cases}$$





Perceptron Summary

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



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

$$a^\intercal x + b = 0$$

Remaining questions

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

Binary Linear Classification: Find a Hyperplane

Objective

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

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

Binary Linear Classification: Find a Hyperplane

Objective

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

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

A simple loss function

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

Binary Linear Classification: Find a Hyperplane

Objective

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

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

A simple loss function

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

We can define a loss (error function):

$$\mathcal{L}(f_\Theta(x^{(i)}), y^i) = \max(0, (-a^\intercal x^{(i)})y^i)$$

Finding a hyperplane

Finding a hyperplane

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

Finding a hyperplane

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

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

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

Finding a hyperplane

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

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

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

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

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

Finding a hyperplane

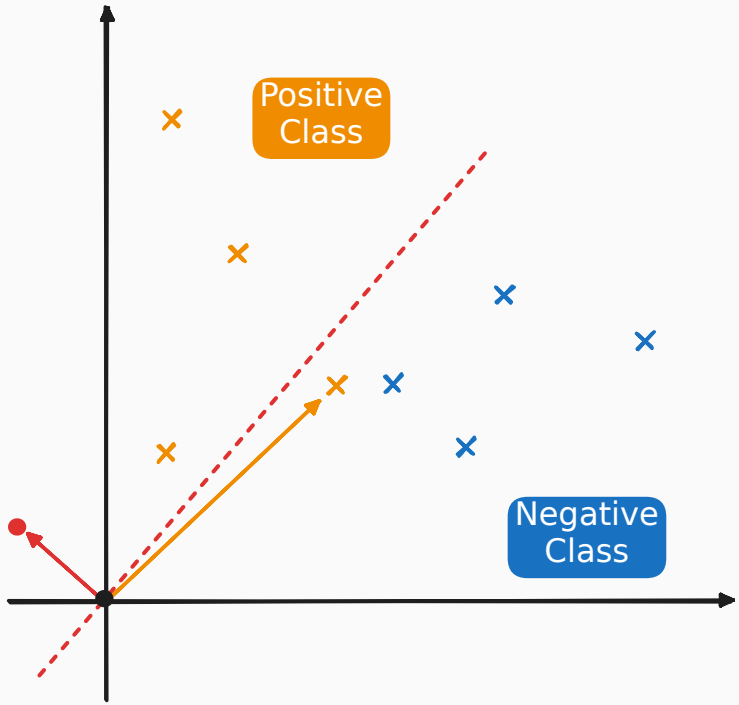
Let us consider $a^{(t)}$ parameter (at time t) and $\lambda = 1$

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

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

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

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



Repeat until convergence (or maximum iteration)

Finding a hyperplane

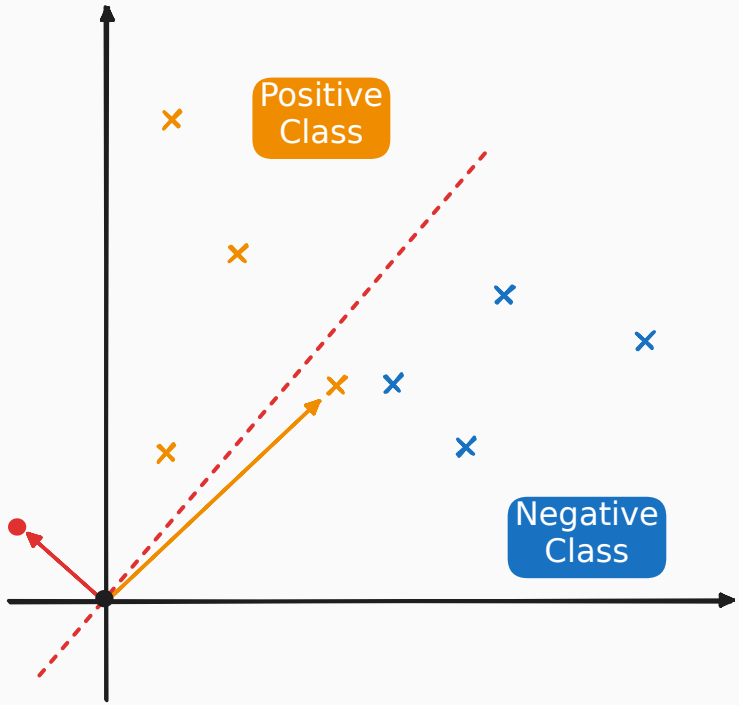
Let us consider $a^{(t)}$ parameter (at time t) and $\lambda = 1$

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

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

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

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



Repeat until convergence (or maximum iteration)

Finding a hyperplane

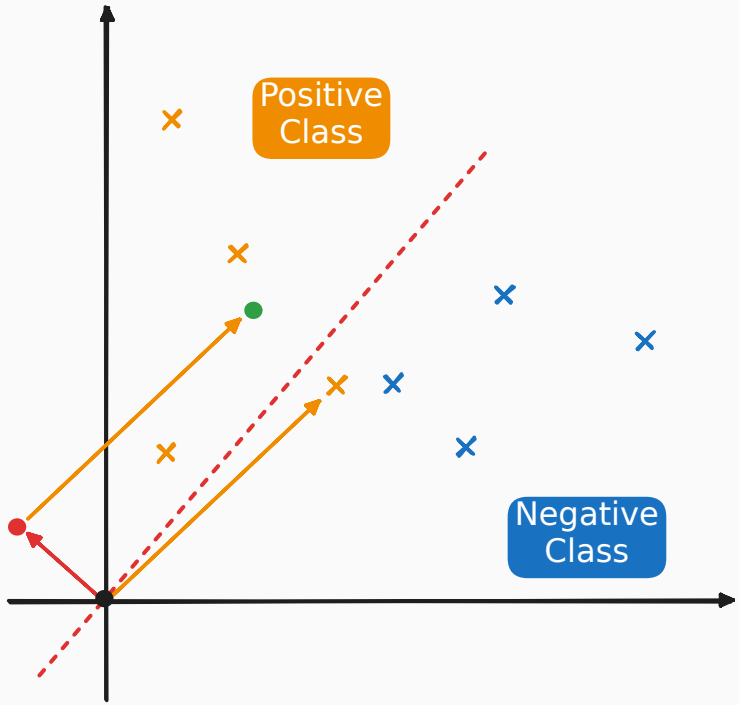
Let us consider $a^{(t)}$ parameter (at time t) and $\lambda = 1$

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

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

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

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



Repeat until convergence (or maximum iteration)

Finding a hyperplane

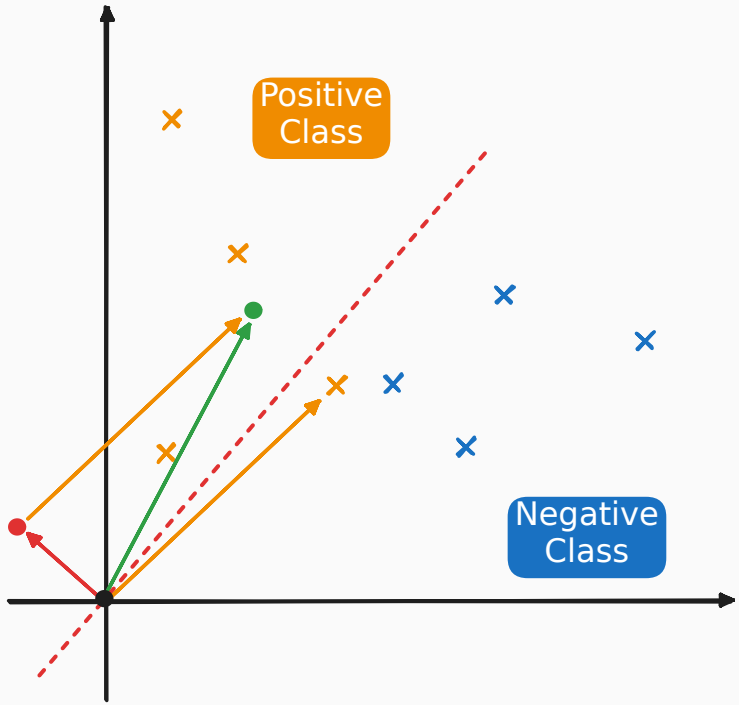
Let us consider $a^{(t)}$ parameter (at time t) and $\lambda = 1$

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

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

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

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



Repeat until convergence (or maximum iteration)

Finding a hyperplane

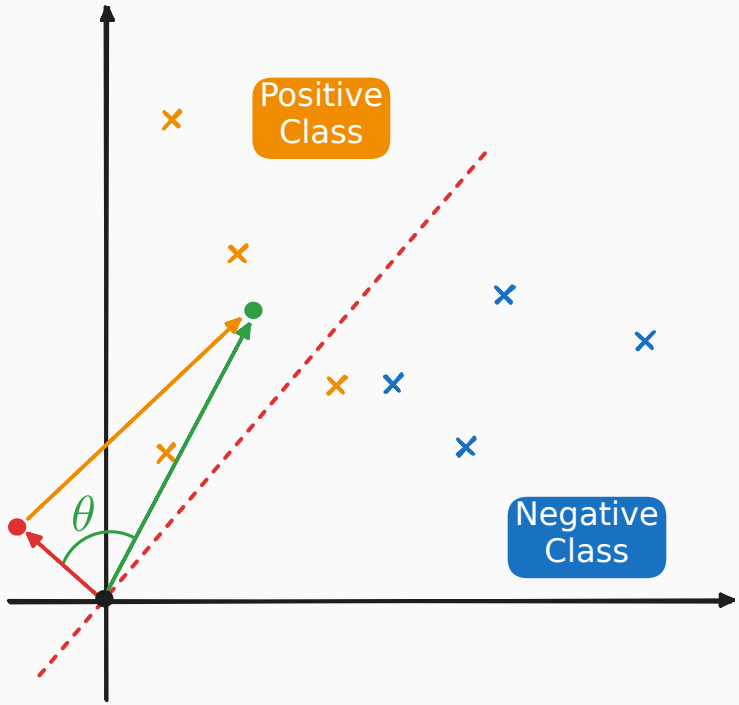
Let us consider $a^{(t)}$ parameter (at time t) and $\lambda = 1$

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

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

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

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



Repeat until convergence (or maximum iteration)

Finding a hyperplane

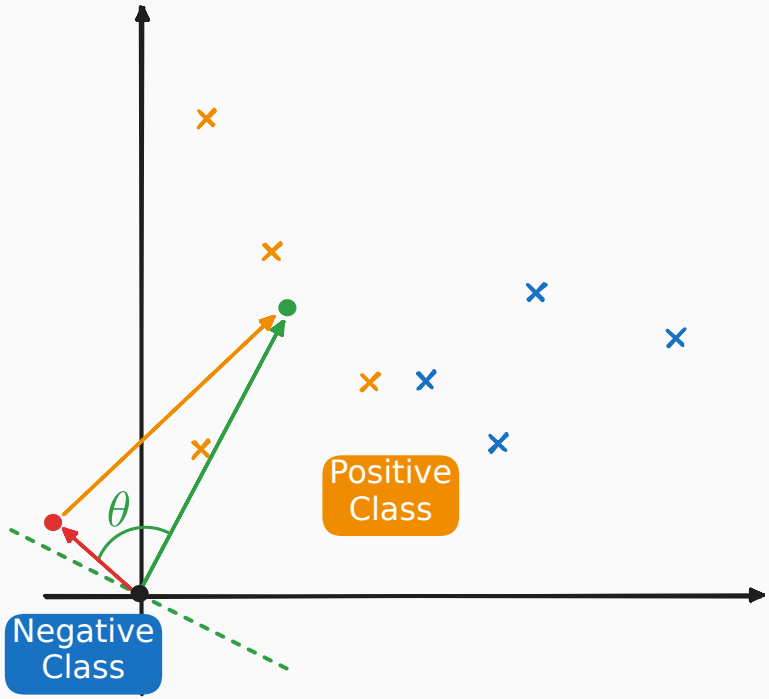
Let us consider $a^{(t)}$ parameter (at time t) and $\lambda = 1$

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

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

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

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



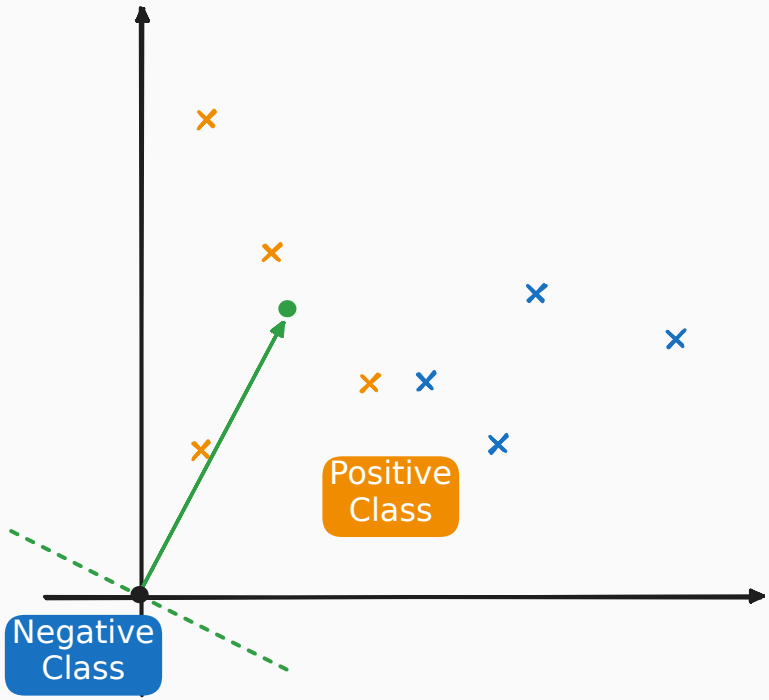
Repeat until convergence (or maximum iteration)

Finding a hyperplane

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

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

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



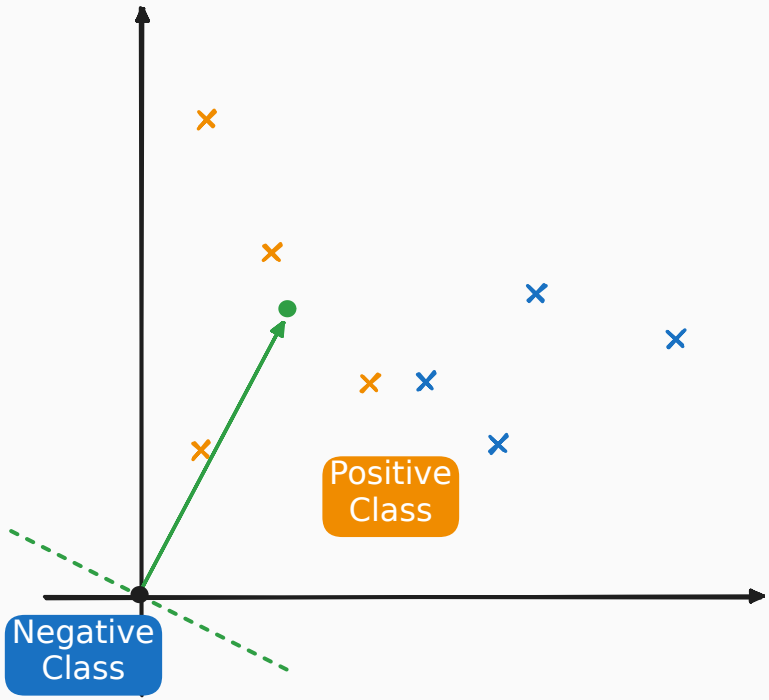
Repeat until convergence (or maximum iteration)

Finding a hyperplane

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

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

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



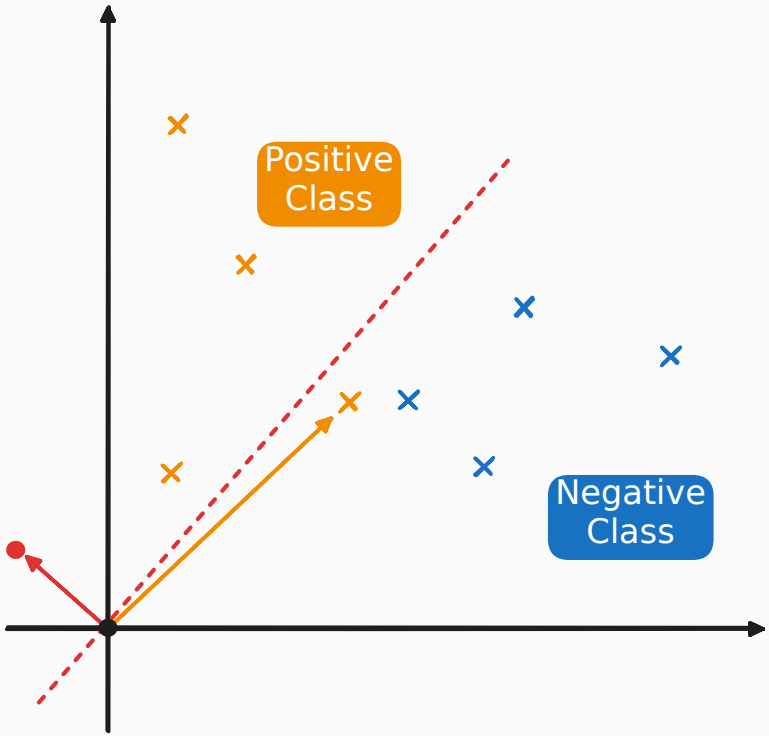
Repeat until convergence (or maximum iteration)
$\rightarrow$ The correction is too large

Finding a Hyperplane

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

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

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

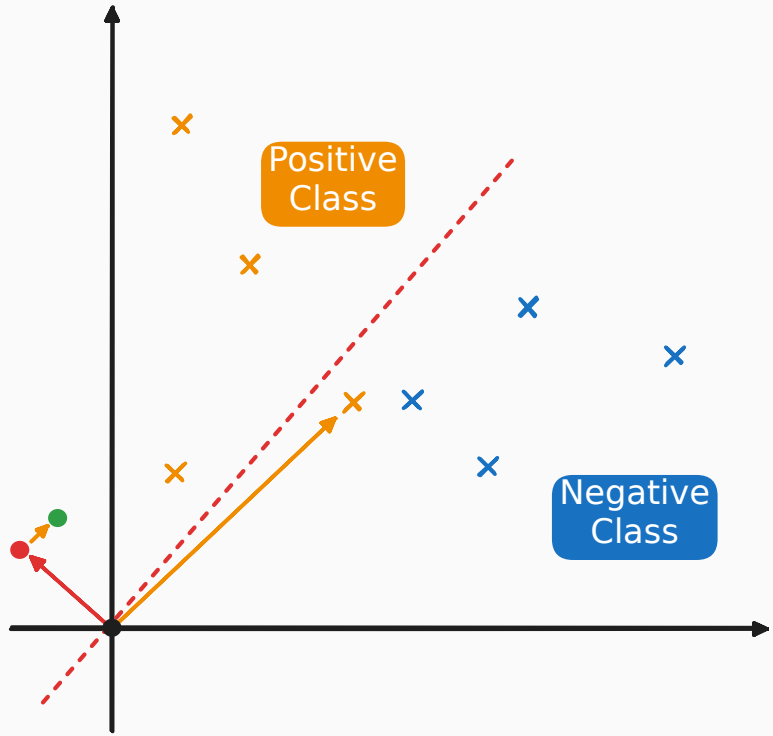


Finding a Hyperplane

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

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

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

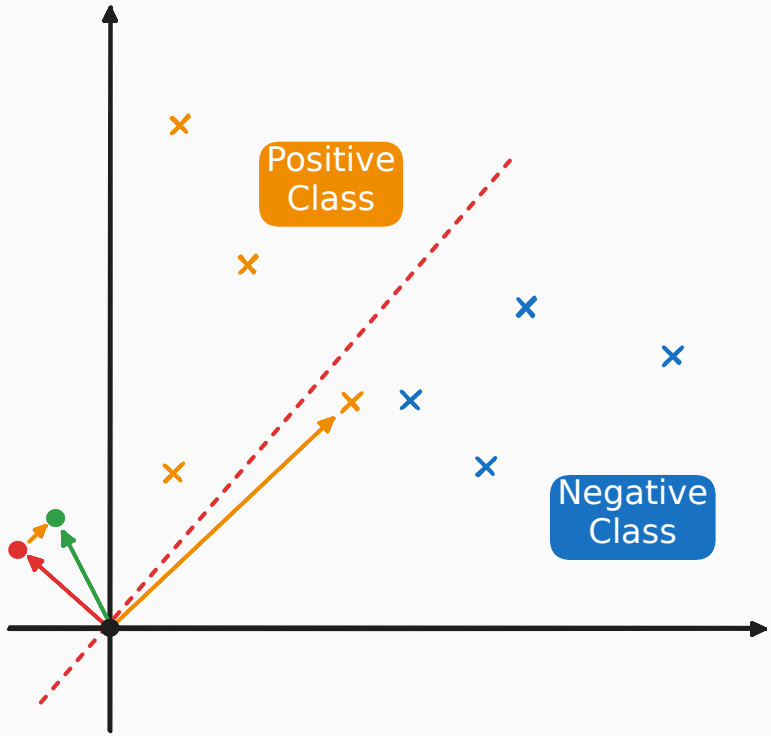


Finding a Hyperplane

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

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

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

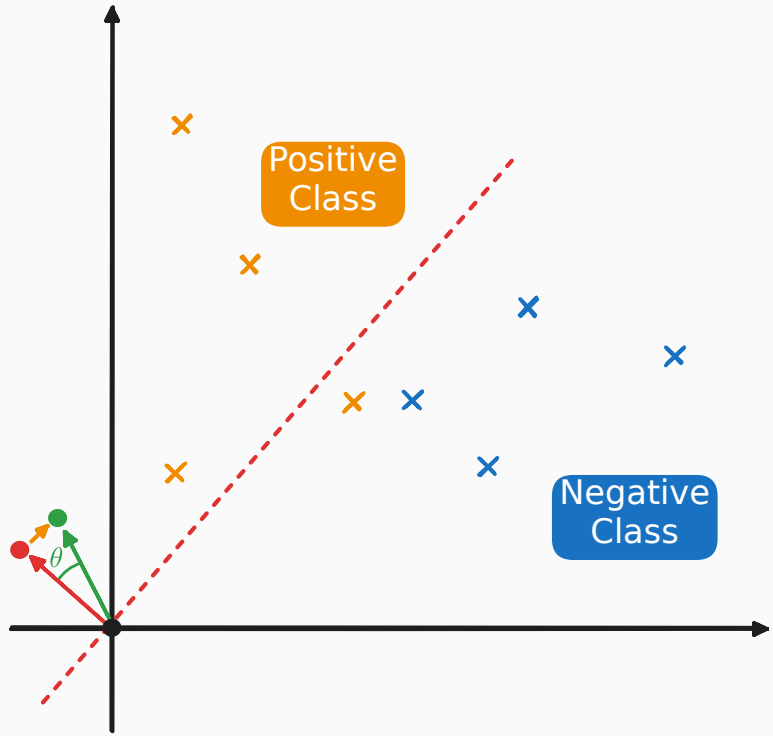


Finding a Hyperplane

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

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

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

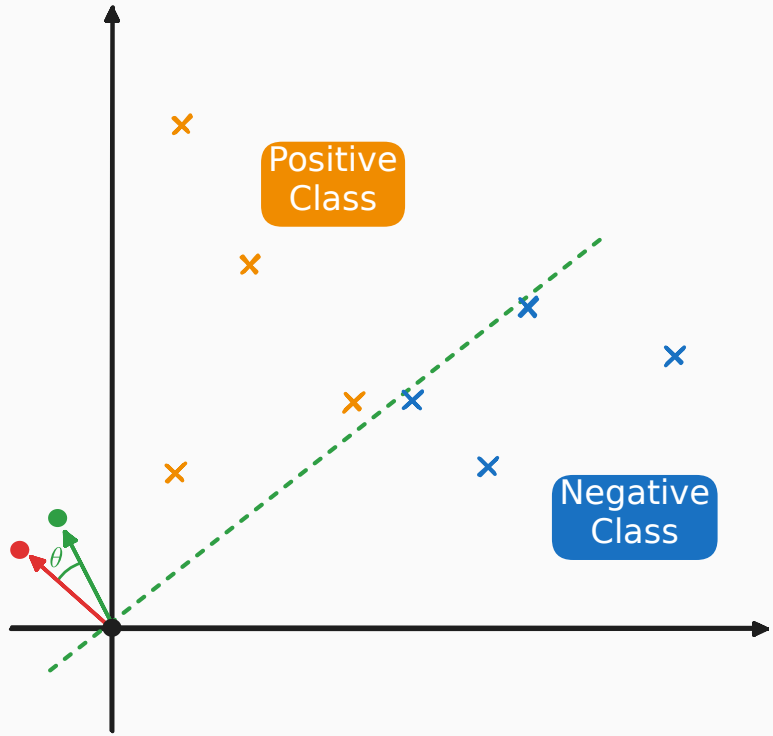


Finding a Hyperplane

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

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

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



Finding a Hyperplane

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

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

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

Exercise 1:
Let the dataset and the classifier

	$x_1^{(i)}$	$x_2^{(i)}$
$x^{(1)}$	2	2
$x^{(2)}$	2	5
$x^{(3)}$	4	3

$$\Theta = (\begin{pmatrix} -1 \\ 2 \end{pmatrix}, 3)$$

What are the positive data points ?

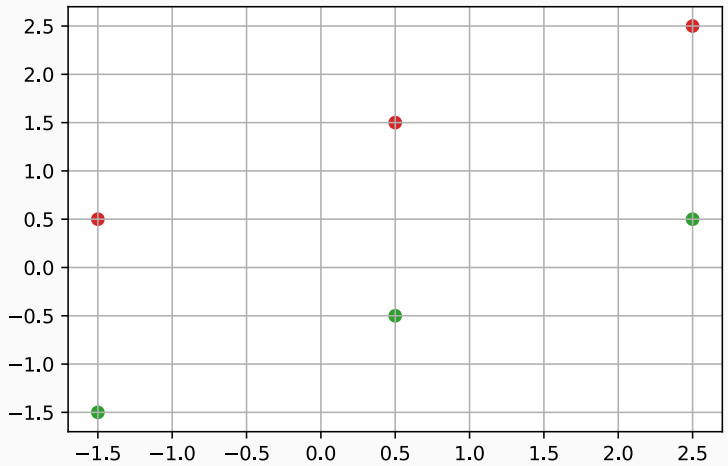
Exercise 3:
Consider the dataset and the classifier

	$x_1^{(i)}$	$x_2^{(i)}$	$y^{(i)}$
$x^{(1)}$	0	0	-1
$x^{(2)}$	-5	5	1
$x^{(3)}$	4	3	-1
$x^{(4)}$	1	3	-1

$$\Theta = (\begin{pmatrix} -1 \\ -1 \end{pmatrix}, 1)$$

Determine the number of errors

Exercise 2:
Find the parameters Θ for the following points



Binary Linear Classification: Find a Hyperplane(end)

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

Binary Linear Classification: Find a Hyperplane(end)

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

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

Binary Linear Classification: Find a Hyperplane(end)

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

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

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

Binary Linear Classification: Find a Hyperplane(end)

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

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

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

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

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

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

Binary Linear Classification: Find a Hyperplane(end)

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

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

$$= -a_1 x_1^{(i)} y^{(i)} - a_2 x_2^{(i)} y^{(i)}$$

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

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

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

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

Thus the hyperplane update is equivalent to:

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

Binary Linear Classification: Find a Hyperplane

Gradient Descent algorithm:

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

This approach are named gradient descent algorithms

$\rightarrow$ **We will study those algorithms later**

Binary Linear Classification: Find a Hyperplane

Gradient Descent algorithm:

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

This approach are named gradient descent algorithms

$\rightarrow$ **We will study those algorithms later**

Remaining questions

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

Binary Linear Classification: Find a Hyperplane

Gradient Descent algorithm:

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

This approach are named gradient descent algorithms

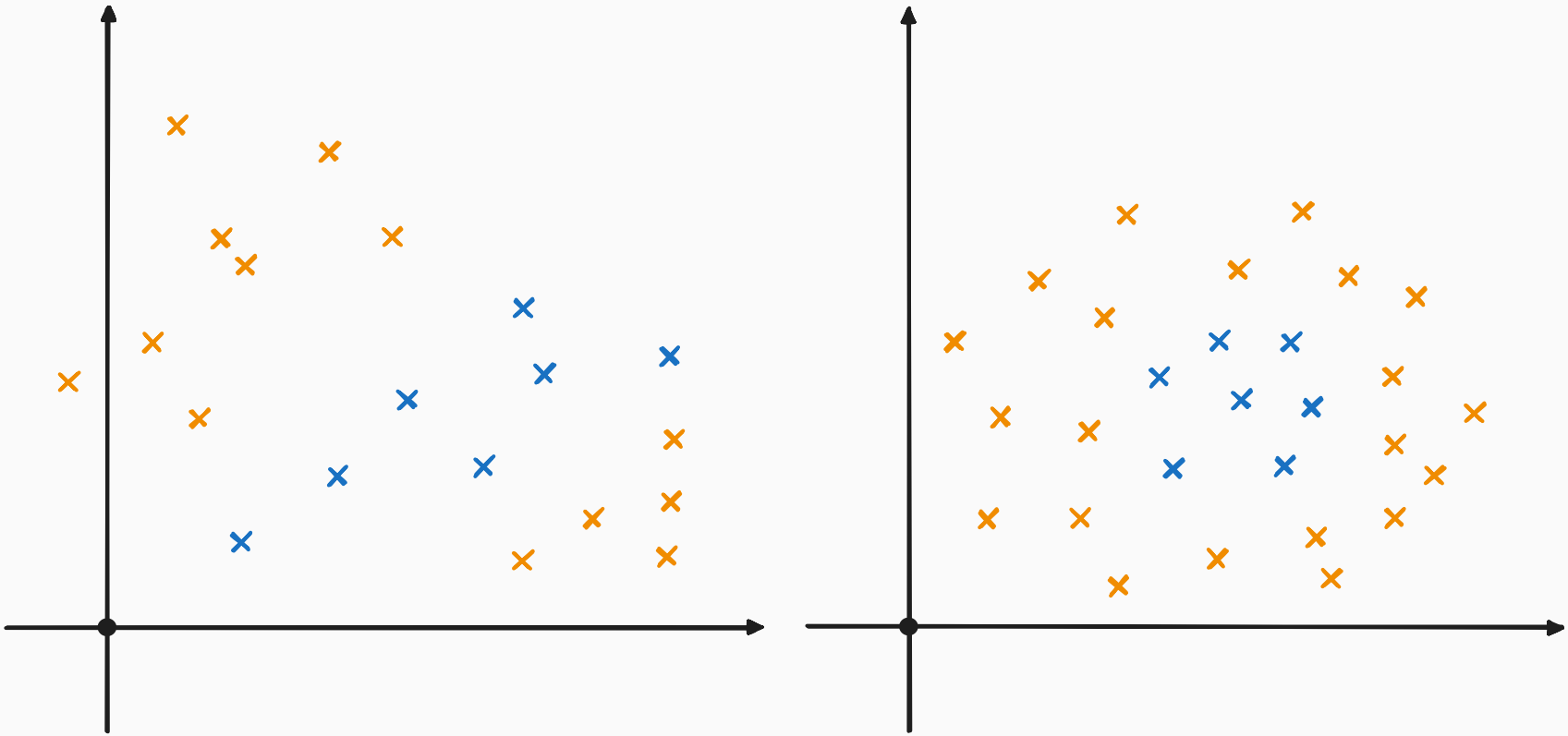
$\rightarrow$ **We will study those algorithms later**

Remaining questions

- ~~How to find the hyperplane $\mathcal{H}$, i.e. how do we retrieve parameters Θ?~~
- 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?

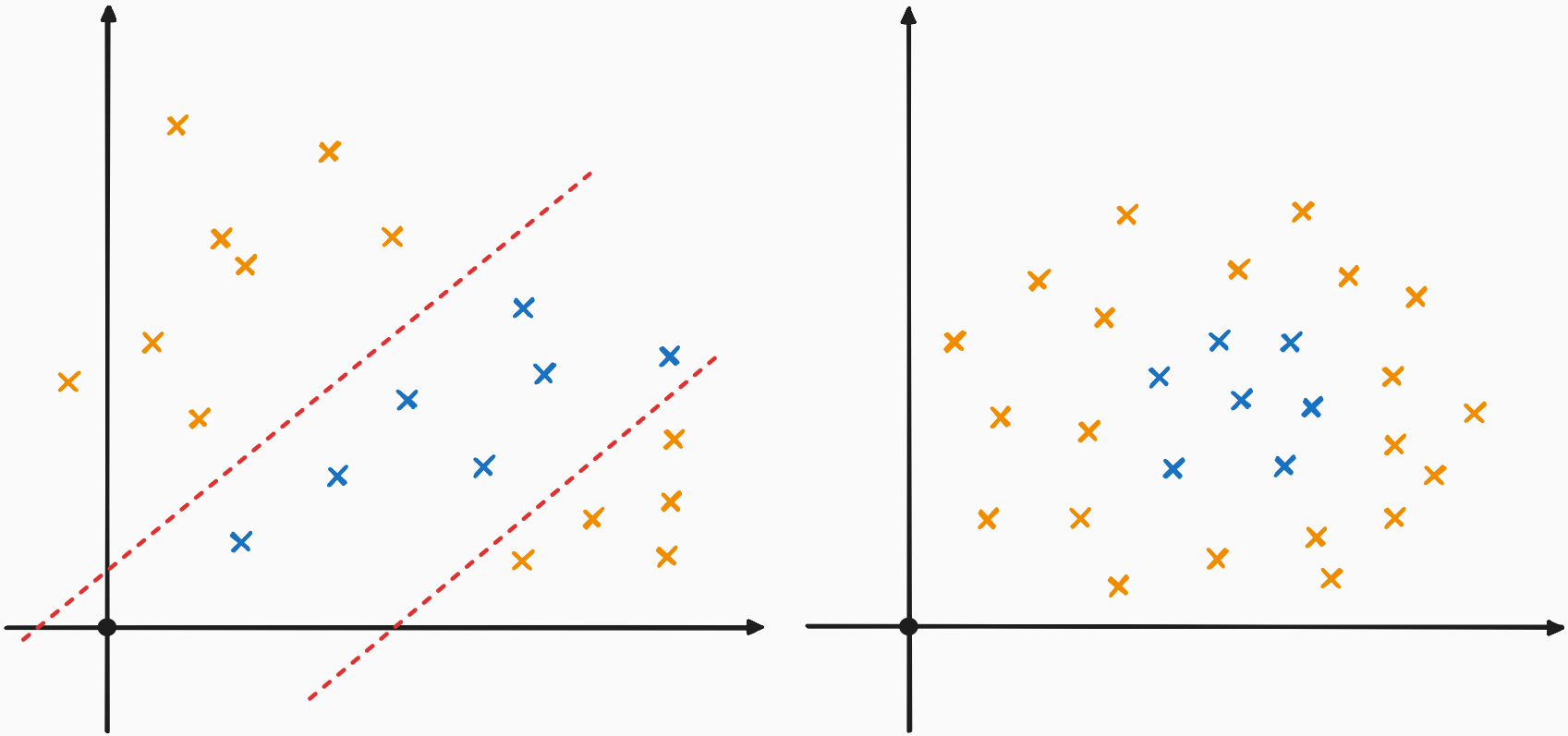
Linear classification: limits

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



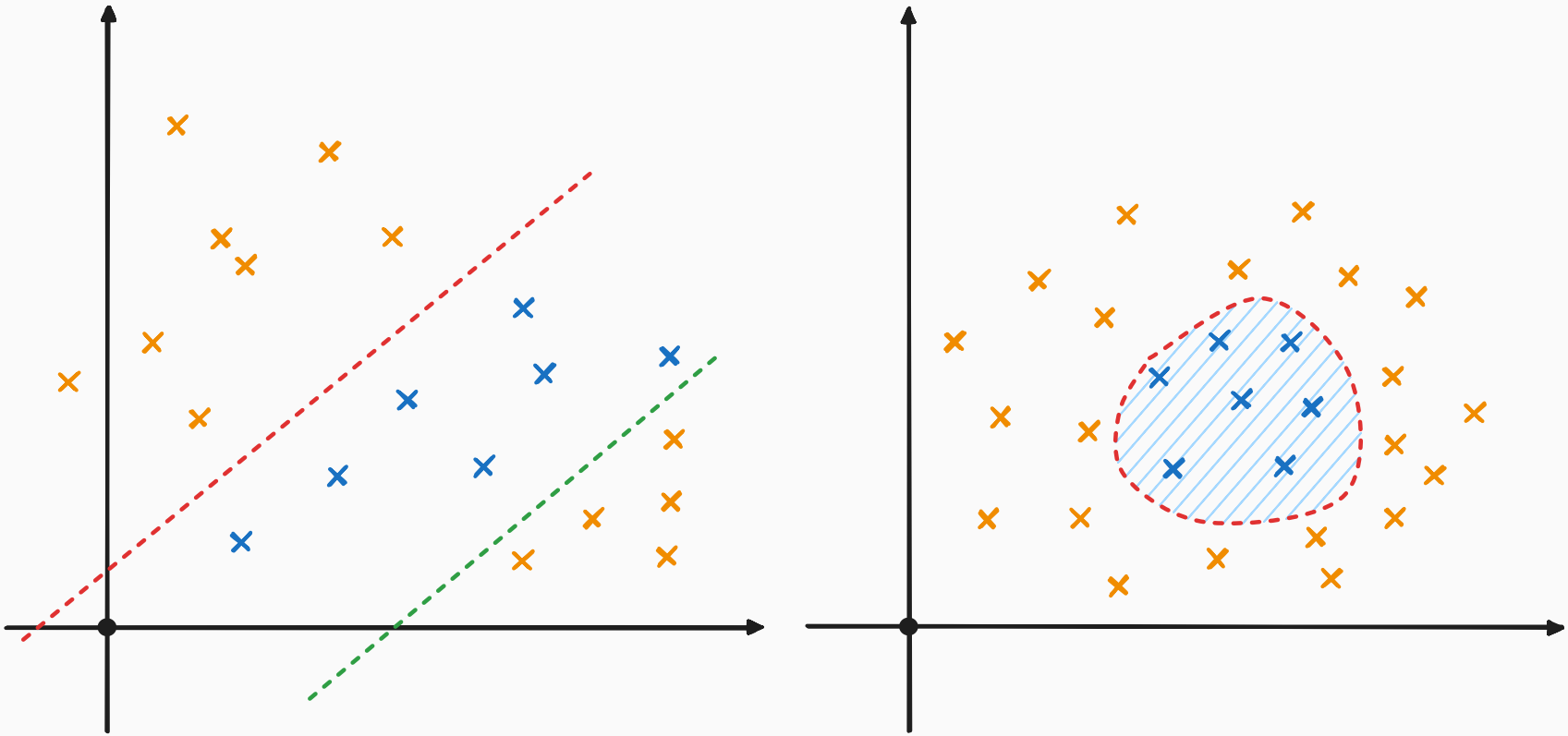
Linear classification: limits

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



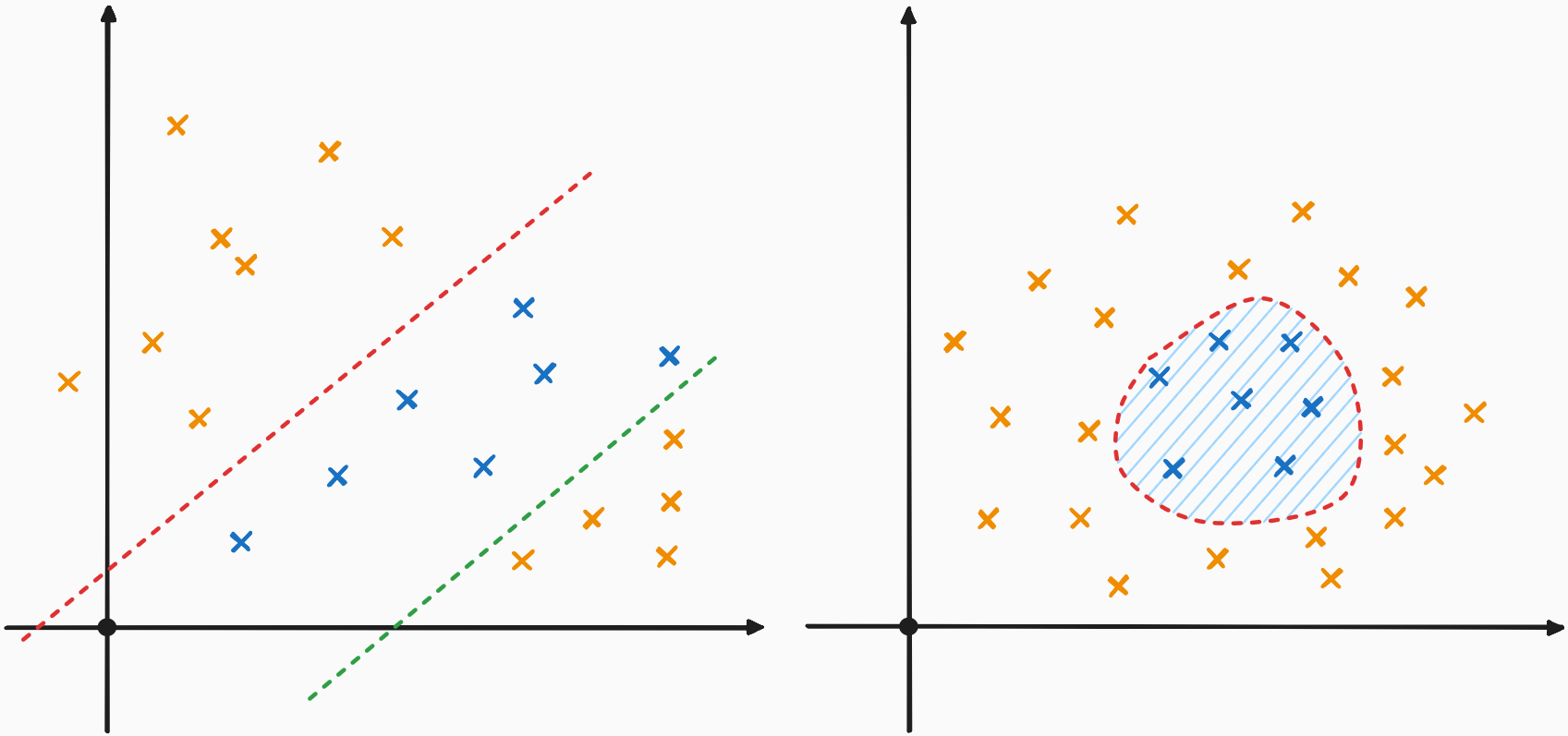
Linear classification: limits

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



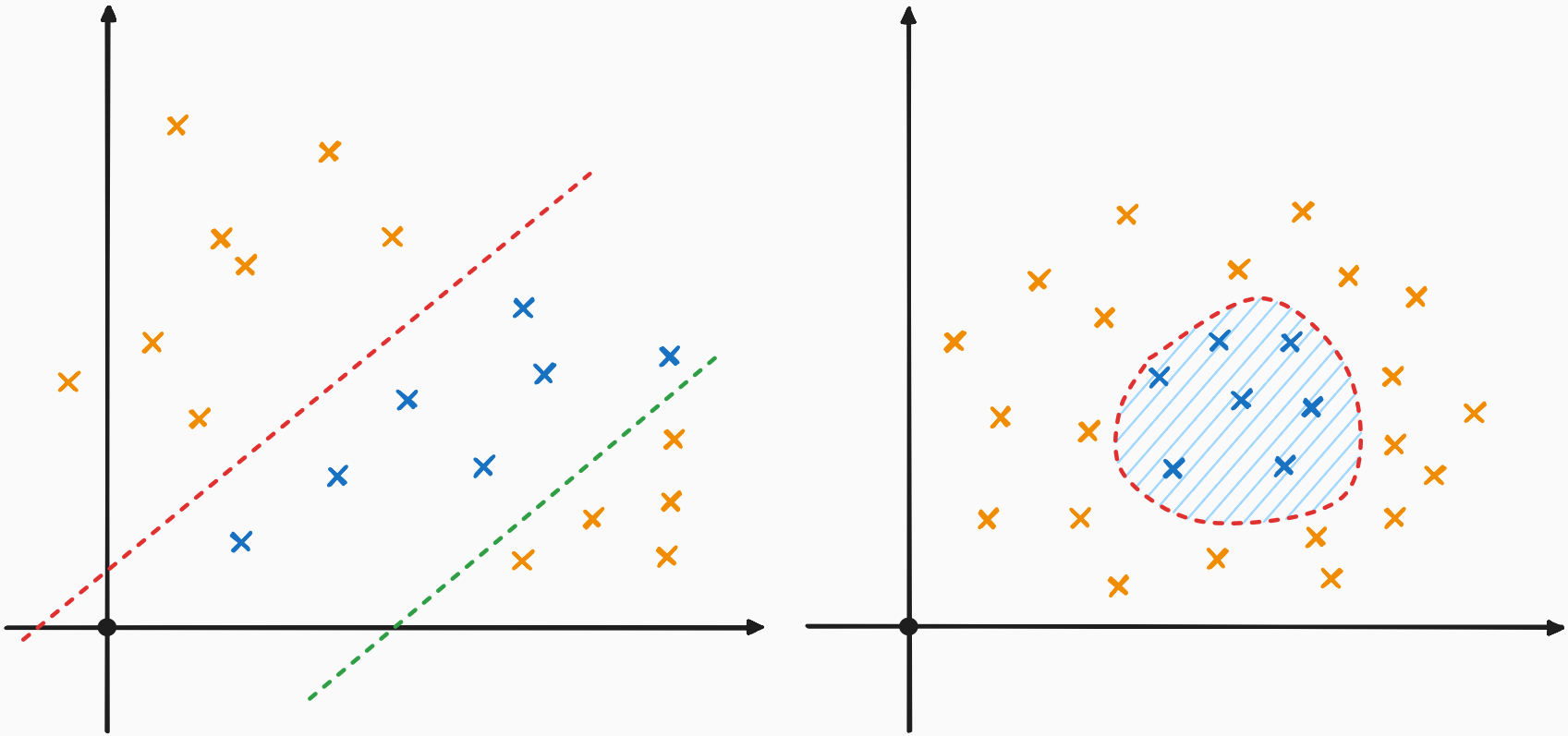
Linear classification: limits

- 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

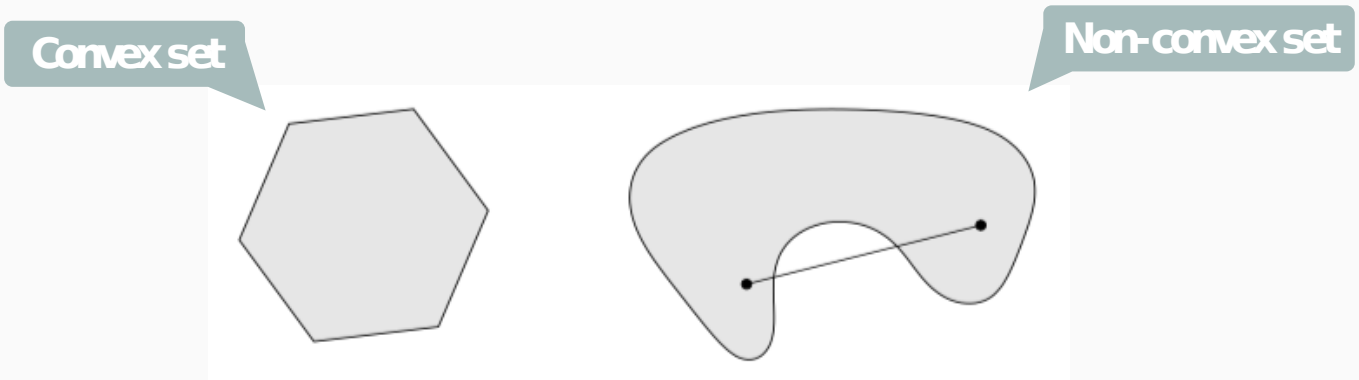


Definition: Convex Set

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



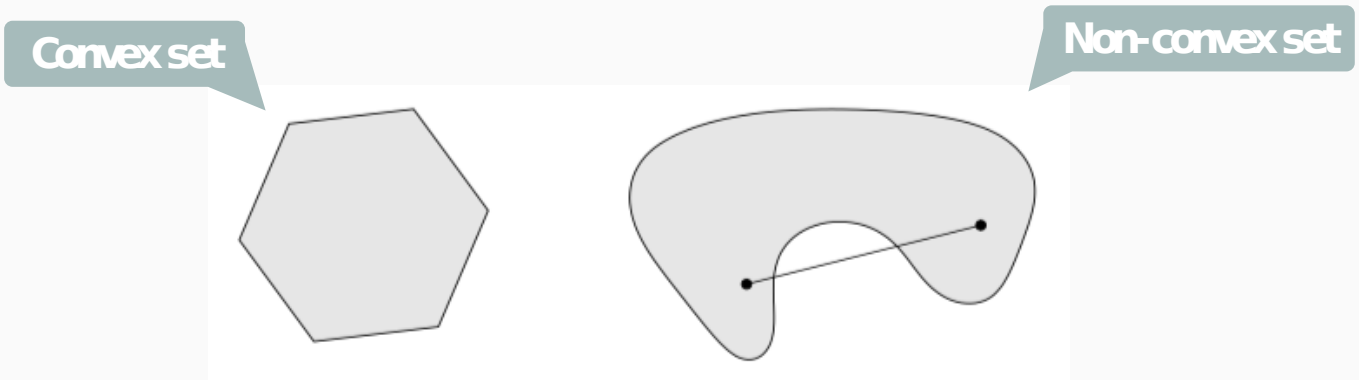
[0]Picture from Convex Optimization, Boyd and Vandenberghe

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



Can we define a convex set from the data points?

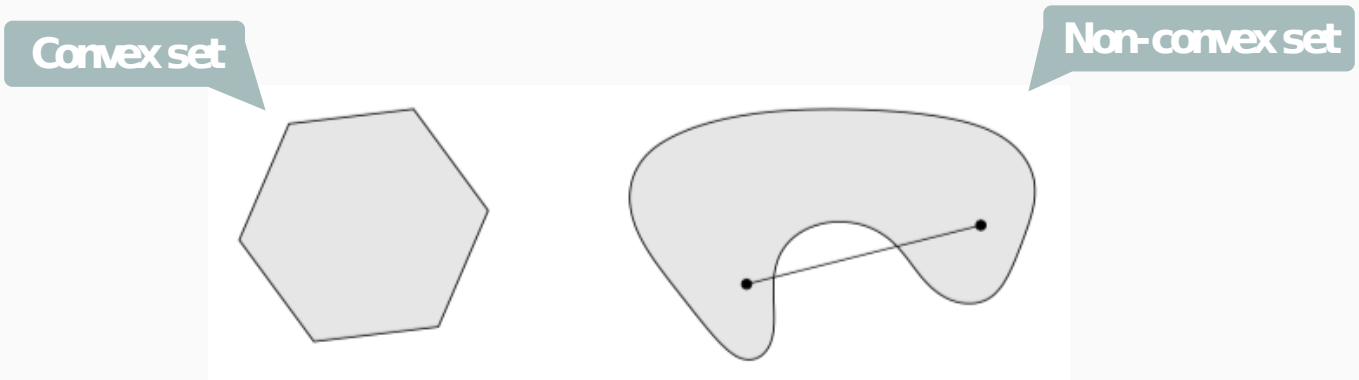
[0]Picture from Convex Optimization, Boyd and Vandenberghe

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

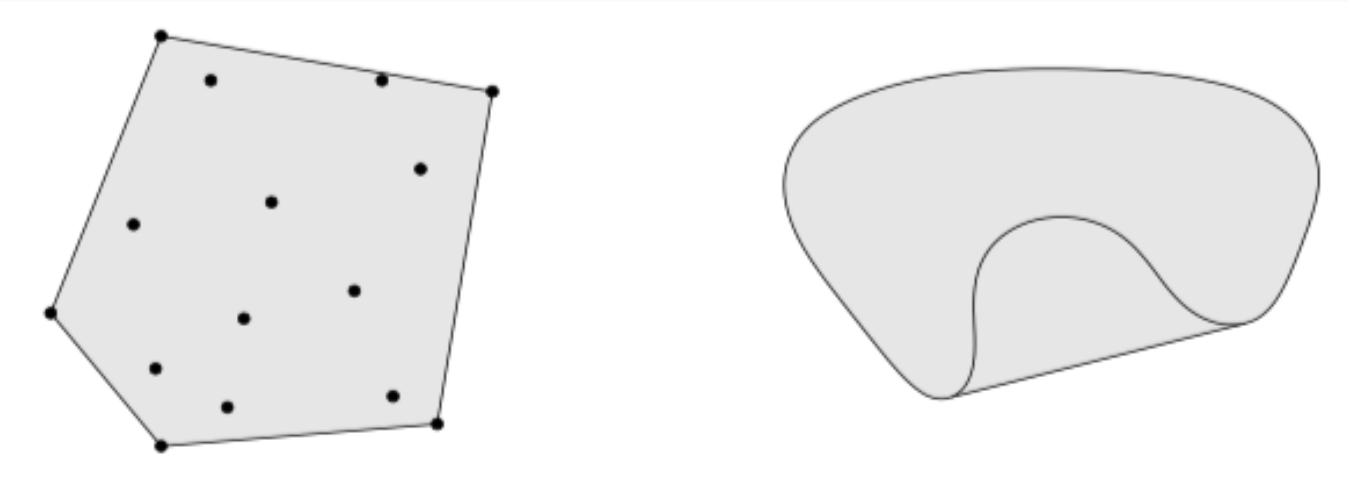


Can we define a convex set from the data points? Yes

[0] 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: $S = \{\epsilon_1 x^{(1)} + \epsilon_2 x^{(2)+\ldots+\epsilon_K x^{(K)}} | x^{(i)} \in C; \epsilon_i \geq 0; \sum_{i=1}^{K} \epsilon_i = 1\}$



[0]Picture from Convex Optimization, Boyd and Vandenberghe

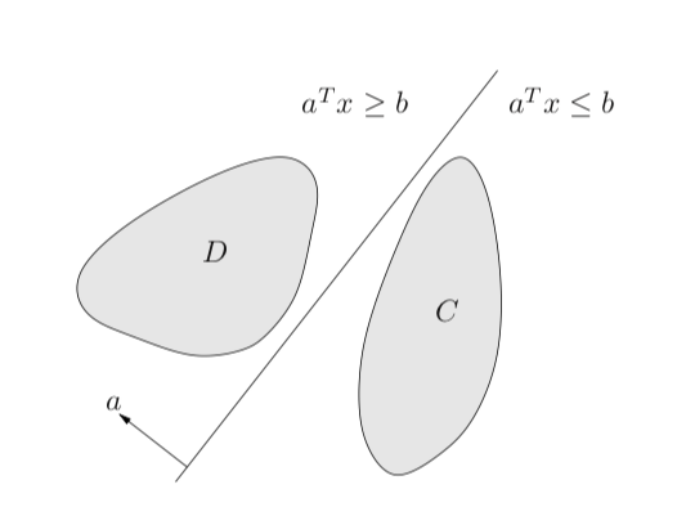
Linearly Separable

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:

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

With a and b parameters of the hyperplane



[0]Picture from Convex Optimization, Boyd and Vandenberghe

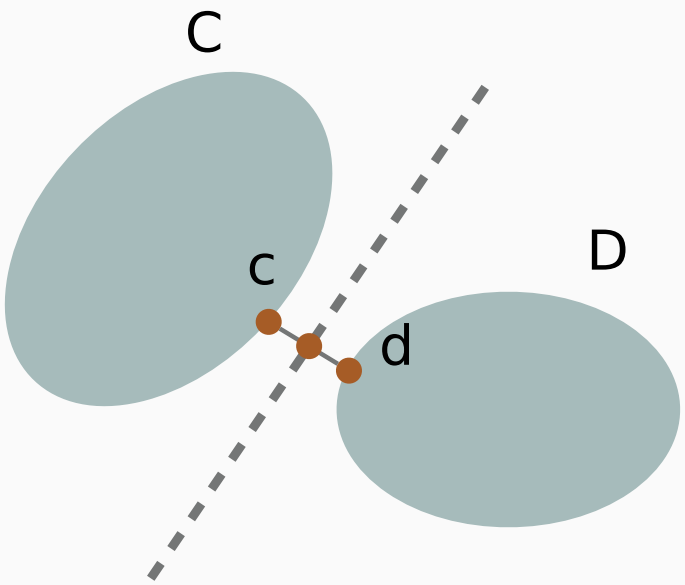
Parameter of the Hyperplane

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 $||c - d|| = dist(C, D)$ 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.



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 $||c - d|| = dist(C, D)$ 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

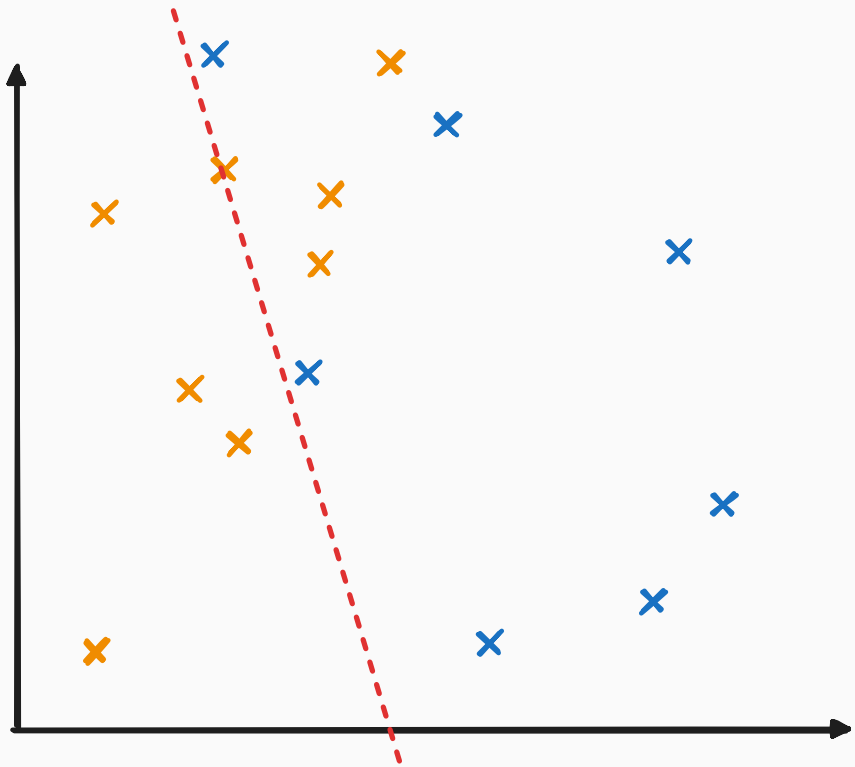
$\rightarrow$ Online algorithm prefered (such as the previous one)

Non-linearly separable problems

Non linearly separable

Handle non linearity?

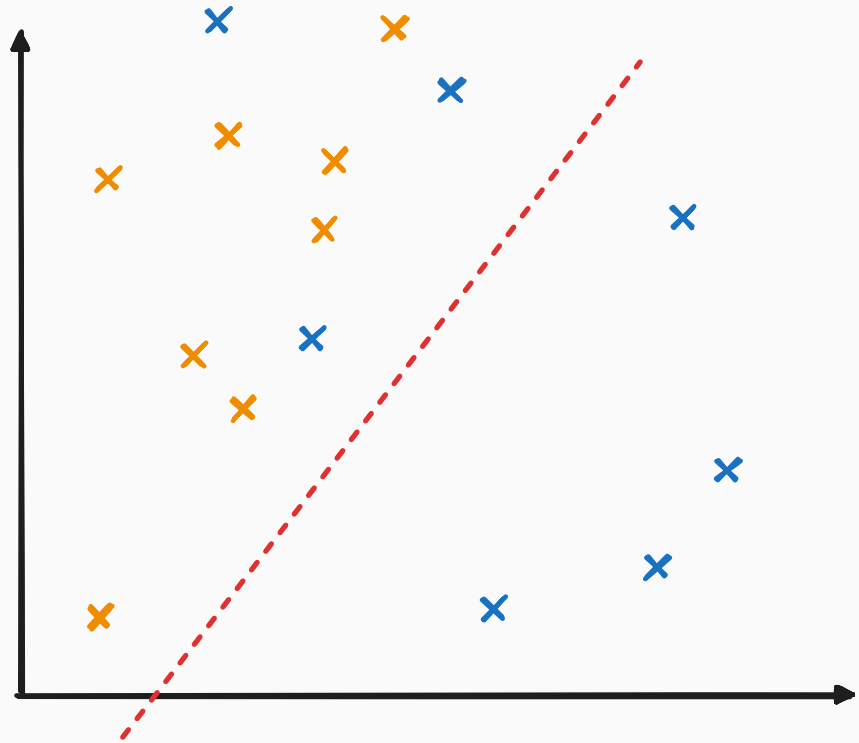
- Linear separation on raw features



Non linearly separable

Handle non linearity?

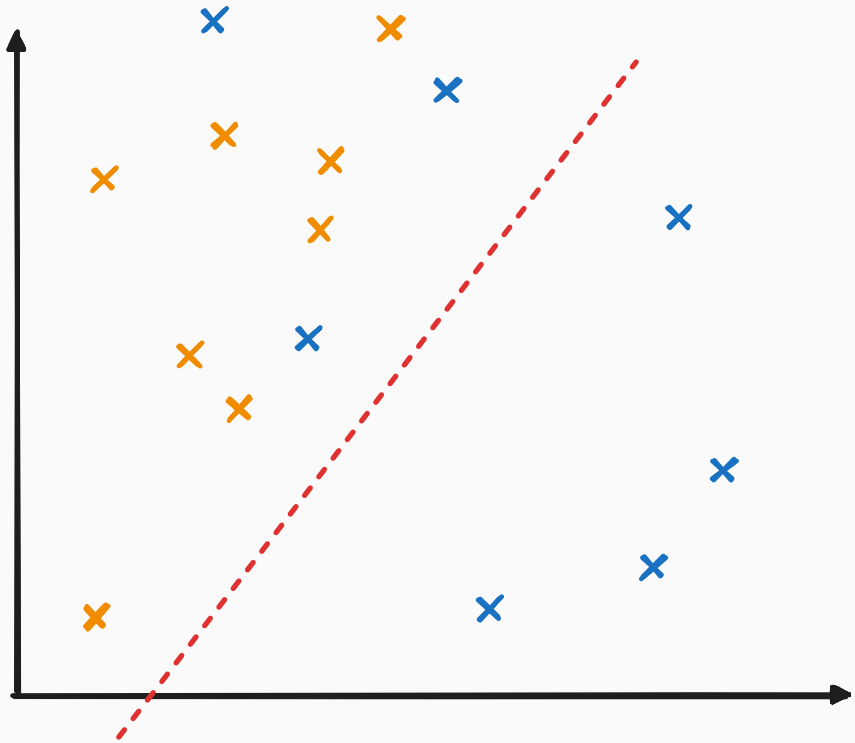
- Linear separation on raw features



Non linearly separable

Handle non linearity?

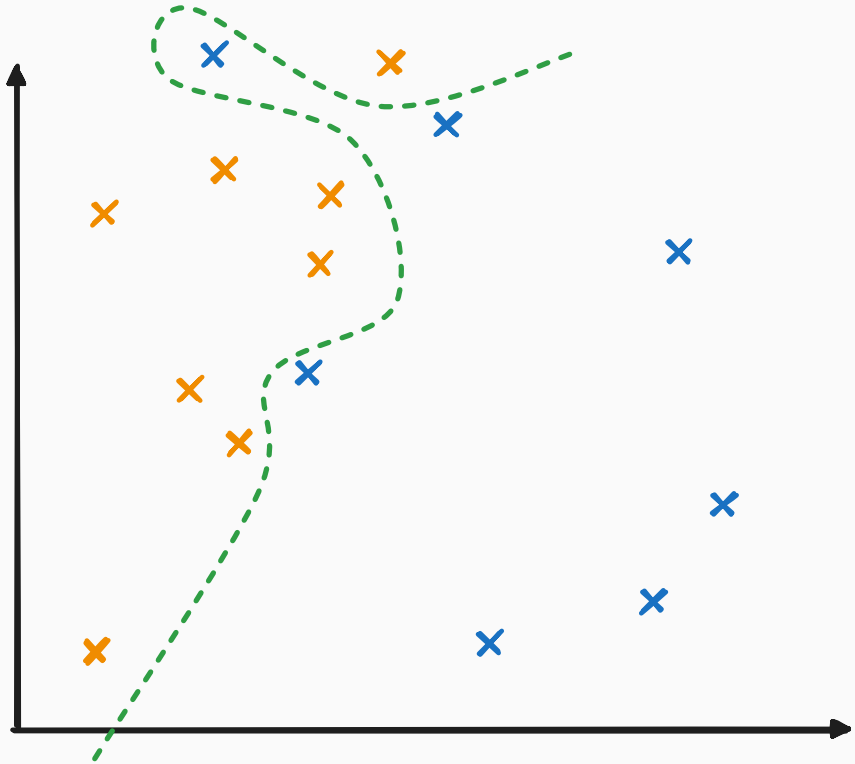
- ~~Linear separation on raw features~~



Non linearly separable

Handle non linearity?

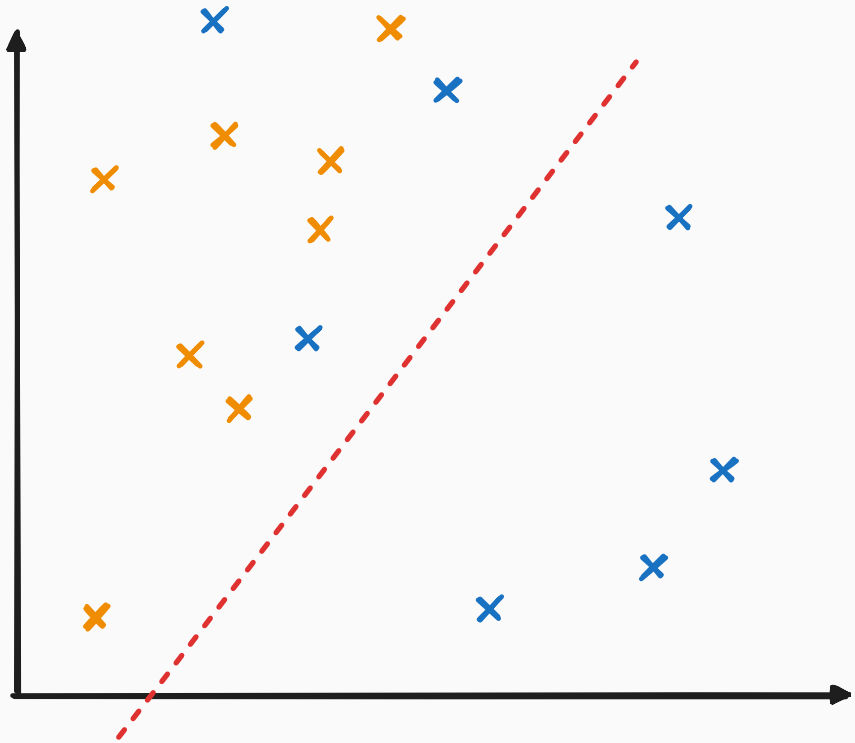
- ~~Linear separation on raw features~~
- **method 1** $\rightarrow$?



Non linearly separable

Handle non linearity?

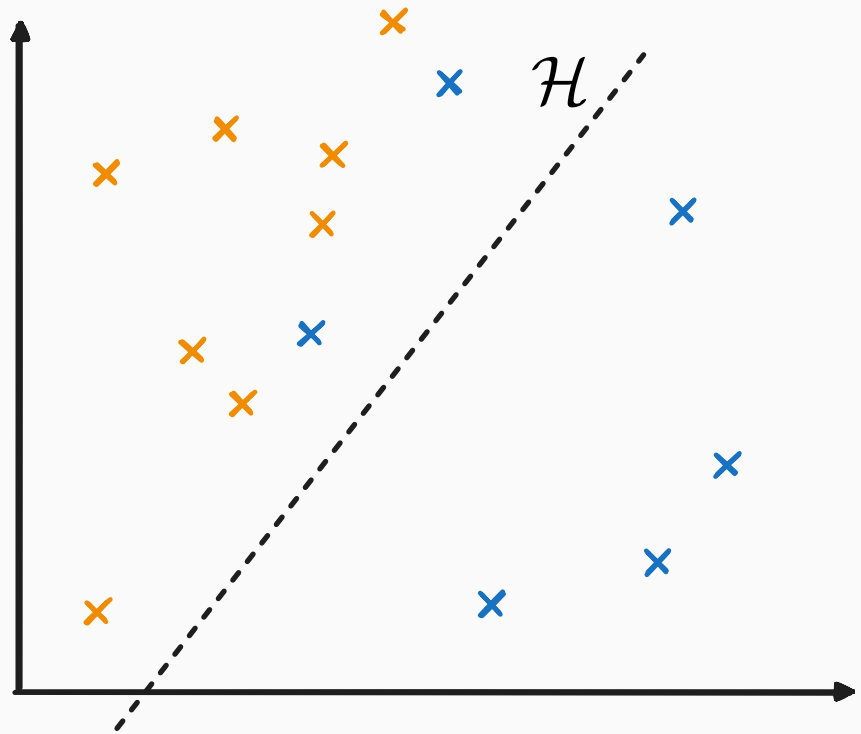
- ~~Linear separation on raw features~~
- **method 1** $\rightarrow$ Using more complex separator (non-linear)



Non linearly separable

Handle non linearity?

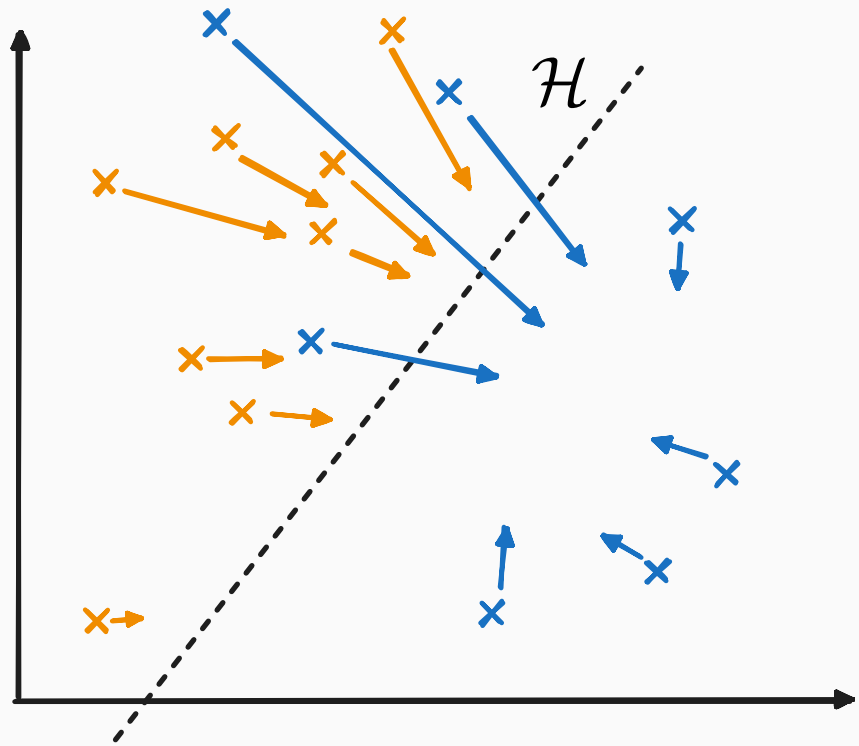
- ~~Linear separation on raw features~~
- **method 1** $\rightarrow$ Using more complex separator (non-linear)
- **method 2** $\rightarrow$?



Non linearly separable

Handle non linearity?

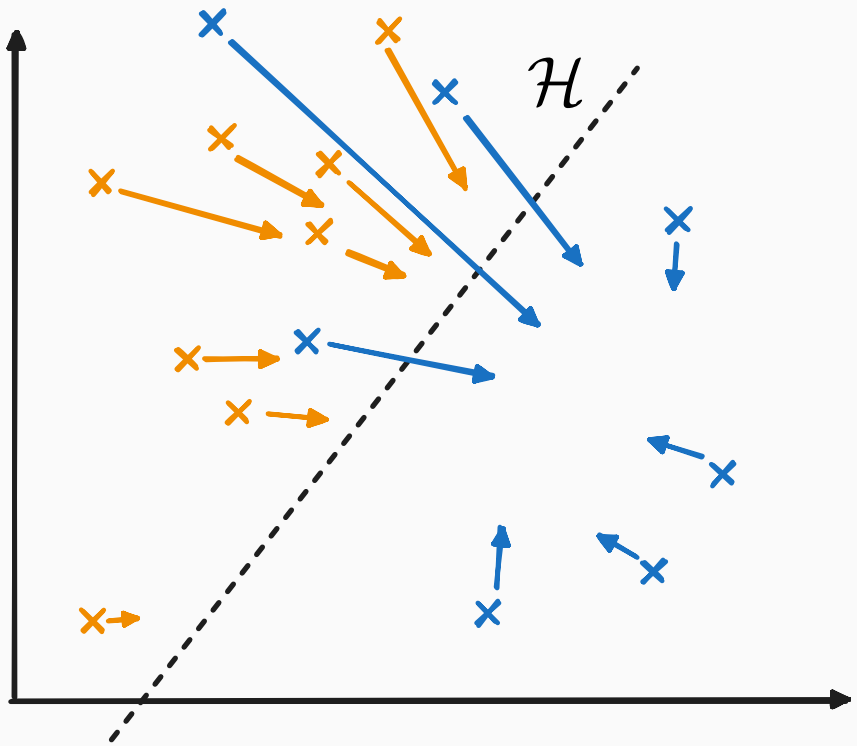
- ~~Linear separation on raw features~~
- **method 1** $\rightarrow$ Using more complex separator (non-linear)
- **method 2** $\rightarrow$?



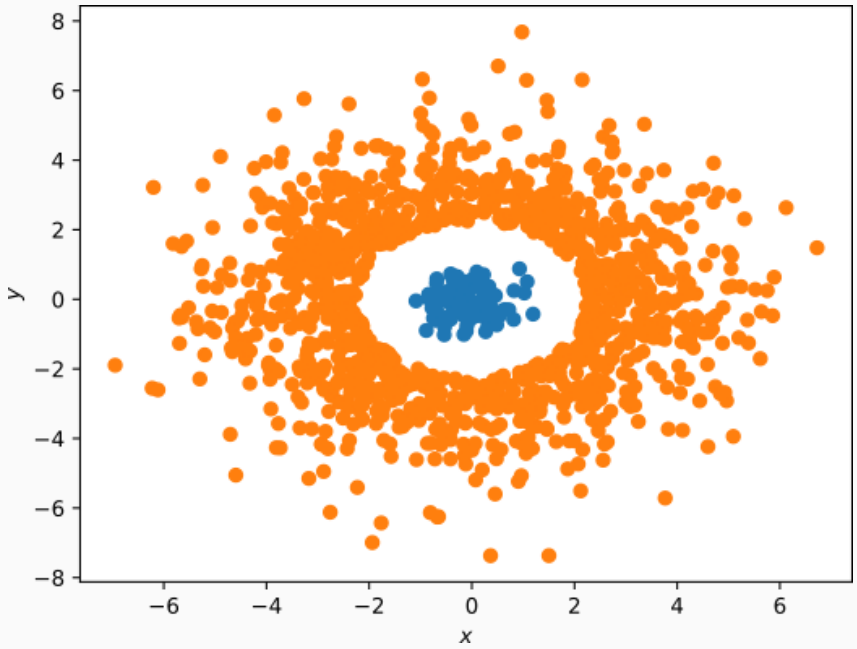
Non linearly separable

Handle non linearity?

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

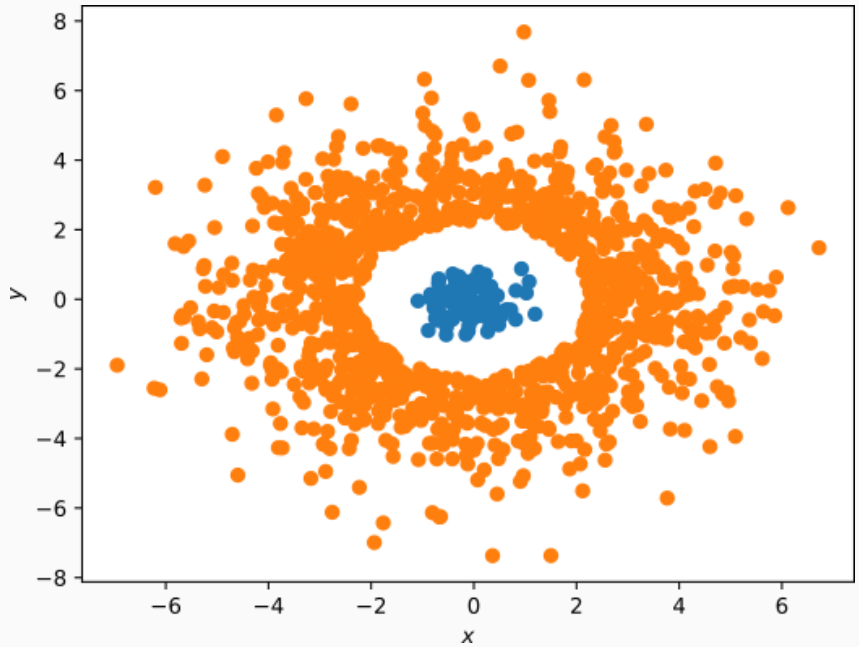


Handle non linearity: example (1)



First example
Any idea of a transformation?

Handle non linearity: example (1)



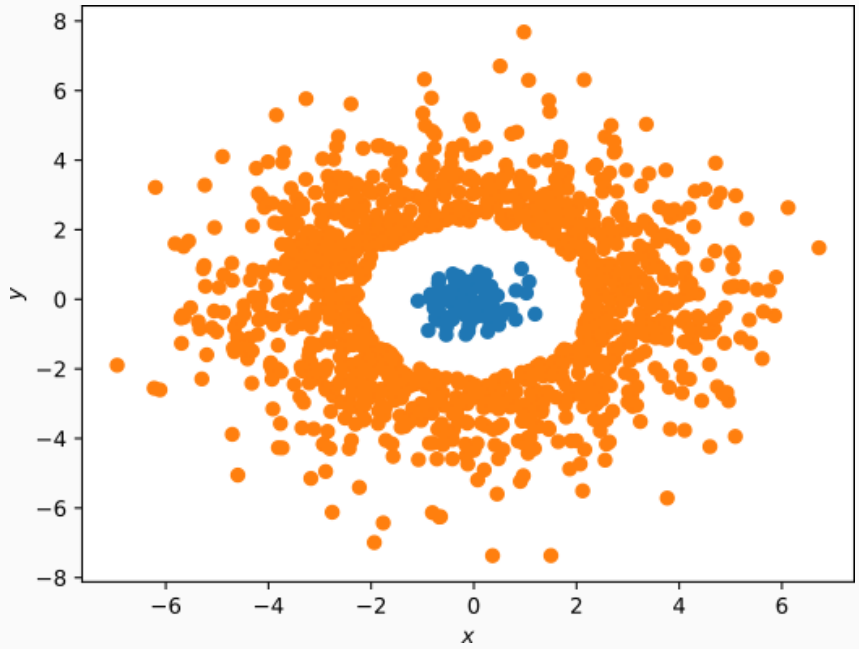
First example
Any idea of a transformation?

Solution

Transform to polar coordinate

$\rightarrow$ only the norm is important here

Handle non linearity: example (1)



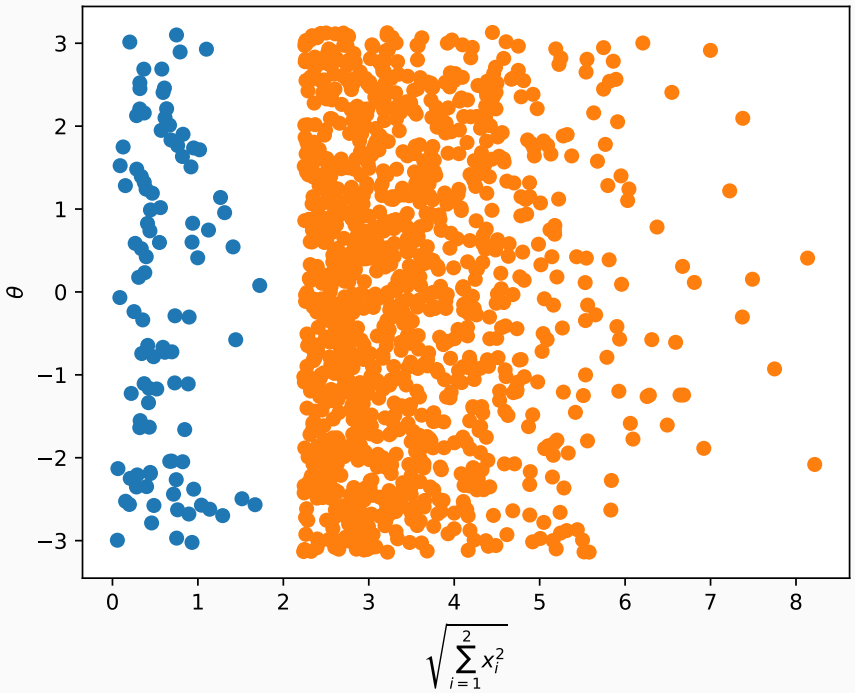
First example:
Any idea of a transformation?

Solution

Transform to polar coordinate

$\rightarrow$ only the norm is important here

Handle non linearity: example (1)



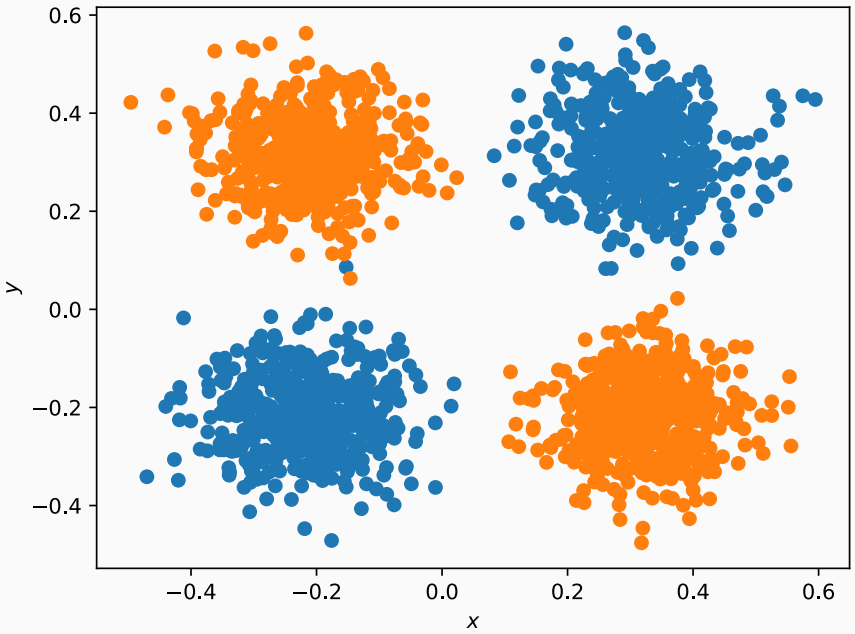
First example:
Any idea of a transformation?

Solution

Transform to polar coordinate

$\rightarrow$ only the norm is important here

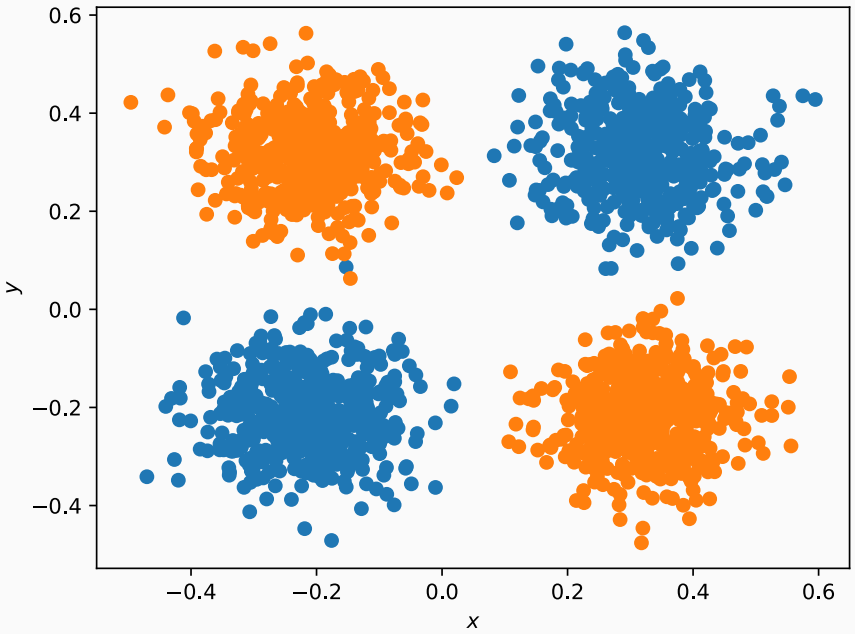
Handle non linearity: example (2)



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?

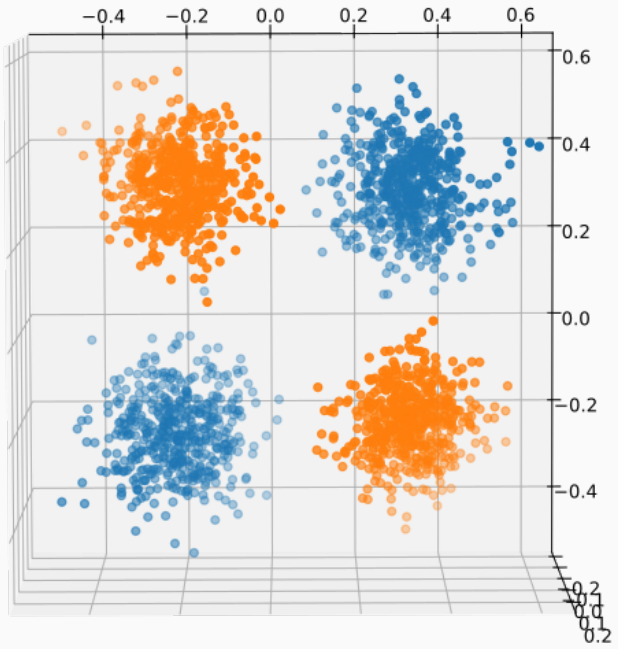
Solution

Add a new dimenssion, transforming data point using:

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

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

Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

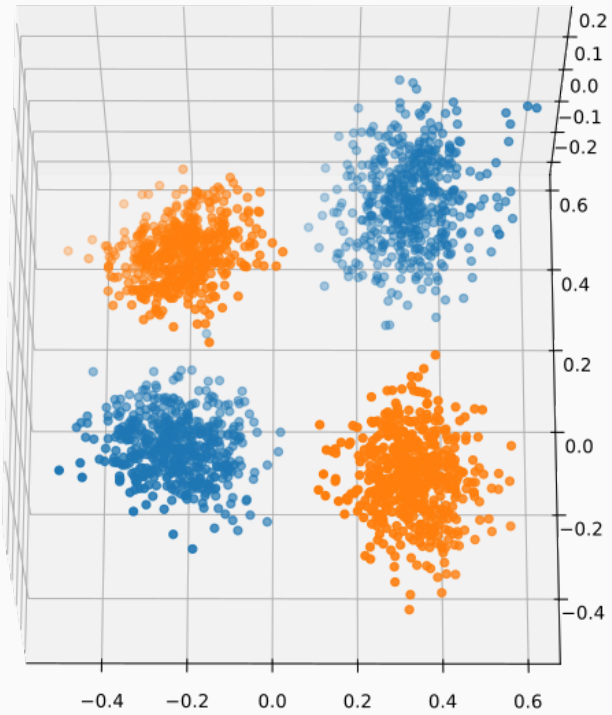
Solution

Add a new dimenssion, transforming data point using:

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

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

Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

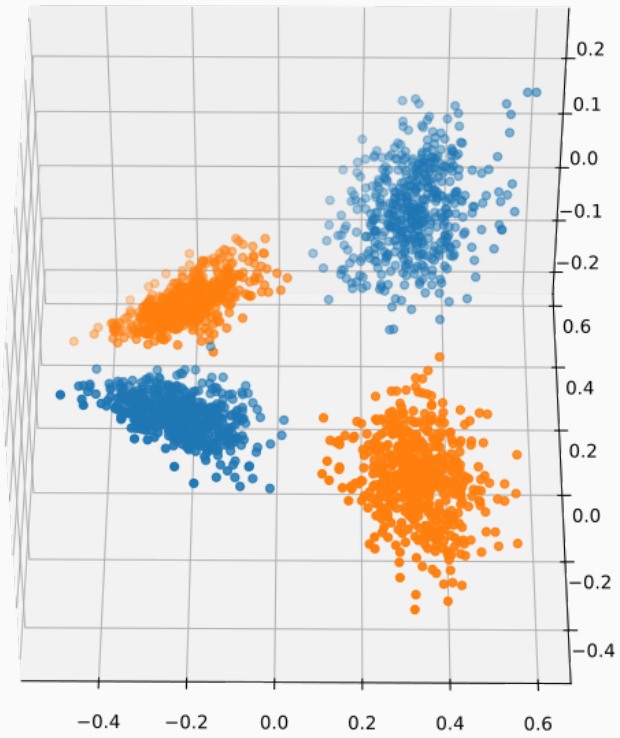
Solution

Add a new dimenssion, transforming data point using:

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

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

Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

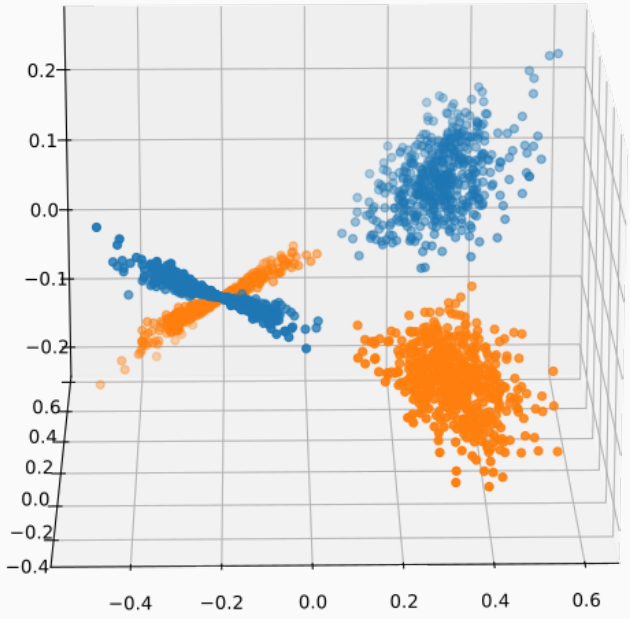
Solution

Add a new dimenssion, transforming data point using:

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

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

Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

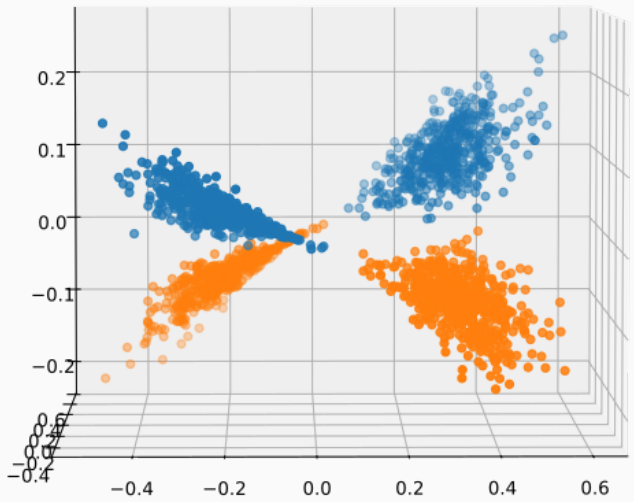
Solution

Add a new dimenssion, transforming data point using:

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

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

Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

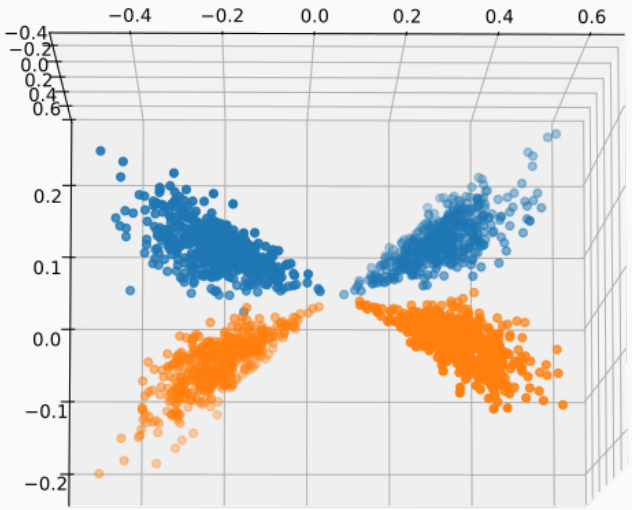
Solution

Add a new dimenssion, transforming data point using:

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

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

Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

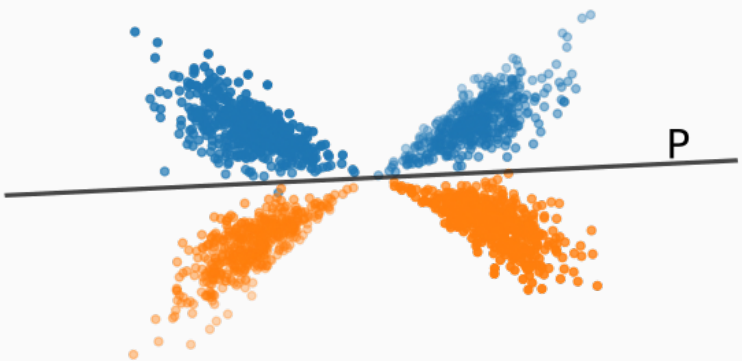
Solution

Add a new dimenssion, transforming data point using:

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

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

Handle non linearity: example (2)



The chessboard problem:

A solution to make it linearly separable?

Solution

Add a new dimenssion, transforming data point using:

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

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

Multilayer Perceptron

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:

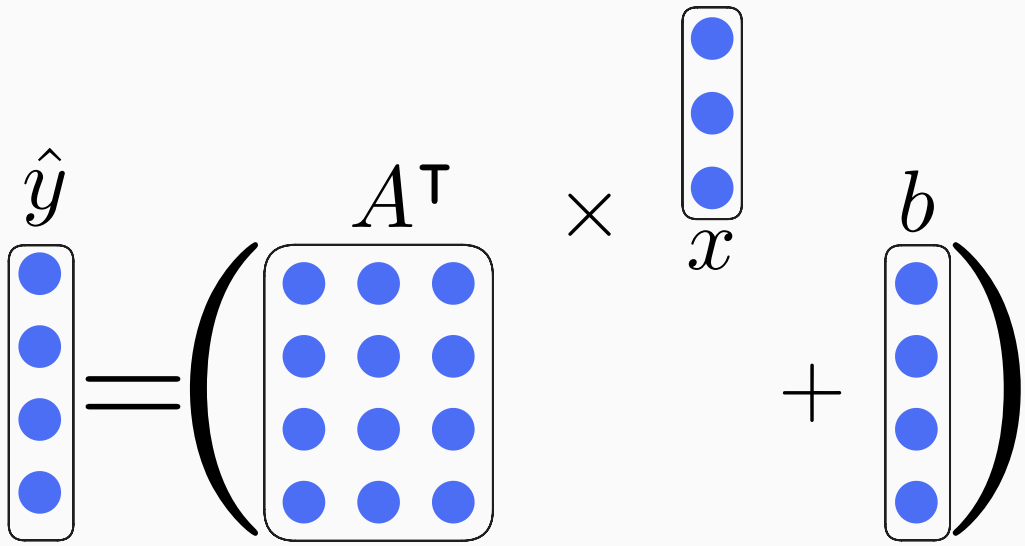
$$f_\Theta : \mathcal{X} \longrightarrow \mathcal{Y}$$
$$x \longmapsto g_{\Theta_H}(\ldots g_{\Theta_2}(g_{\Theta_1}(x)))$$

Problem

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

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

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



Multi-layer Perceptron: base block

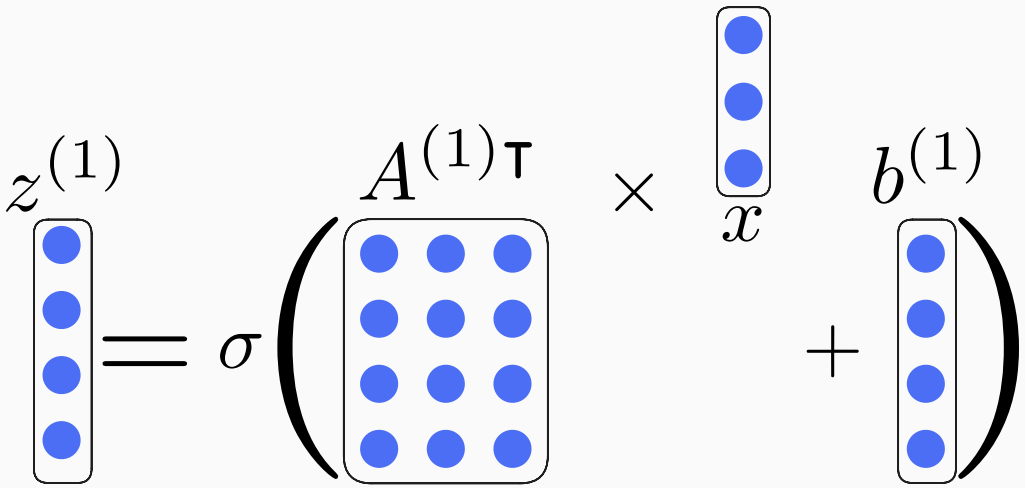
first hidden layer
$$z^{(1)} = \sigma^{(1)}\left(A^{(1)\intercal}x + b^{(1)}\right)$$

second hidden layer
$$z^{(2)} = \sigma^{(2)}\left(A^{(2)\intercal}z^{(1)} + b^{(2)}\right)$$

output projection
$$o = \sigma^{(2)}\left(A^{(3)\intercal}z^{(2)} + b^{(3)}\right)$$

Notations

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



Activation functions

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

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

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?

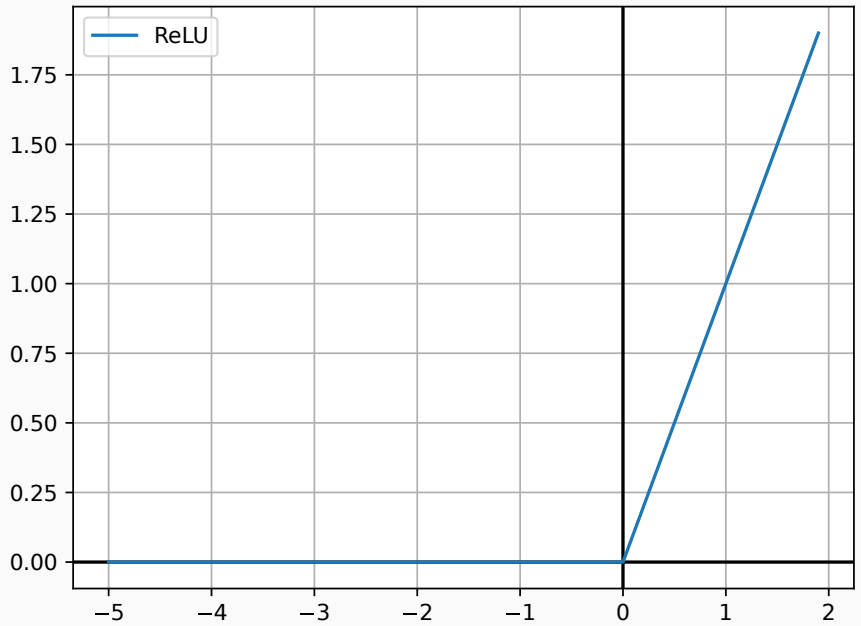
Let consider $A^{(1)} \in \mathbb{R}^{n \times m}$ and $A^{(2)} \in \mathbb{R}^{m \times k}$

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

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

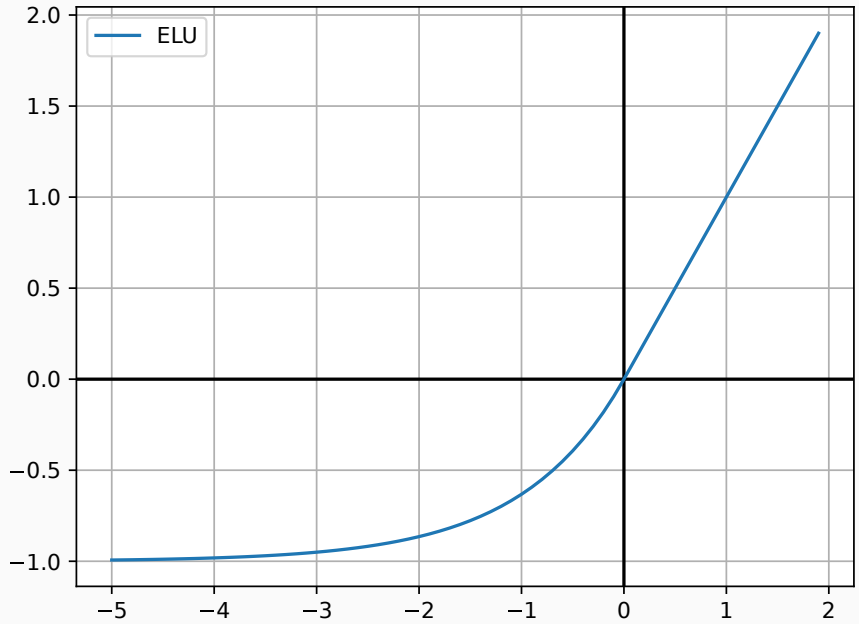
Rectified Linear Unit :

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



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 :

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

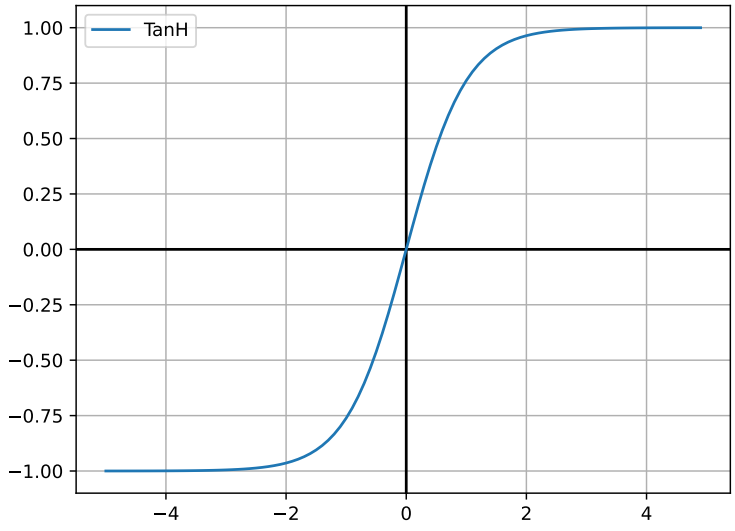


Figure 3: The Hyperbolic tangent activation function

Sigmoid:
Sigmoid activation is defined by :

$$\begin{aligned} \mathbf{f} : \mathbb{R} &\longrightarrow]0,1[\\ x &\longmapsto \frac{1}{1+e^{-x}} \end{aligned}$$

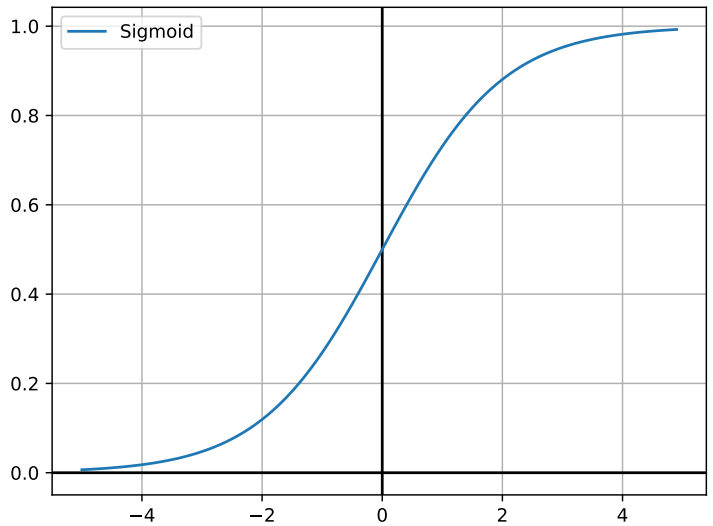


Figure 4: Sigmoid activation function

Activation functions

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

There is much more activation functions...

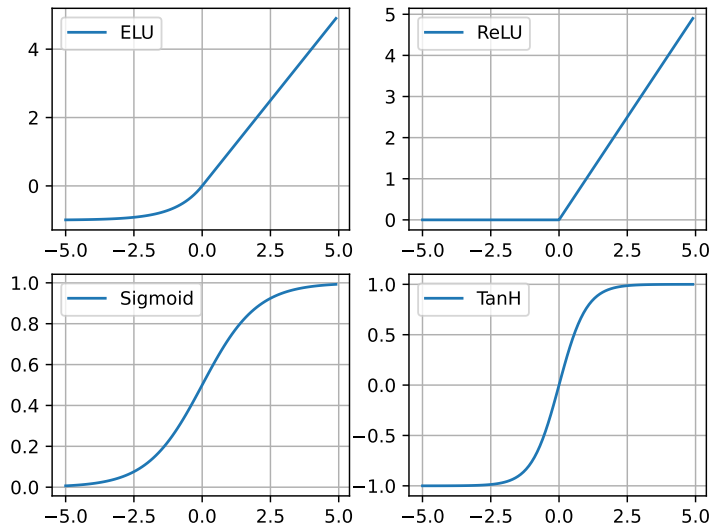
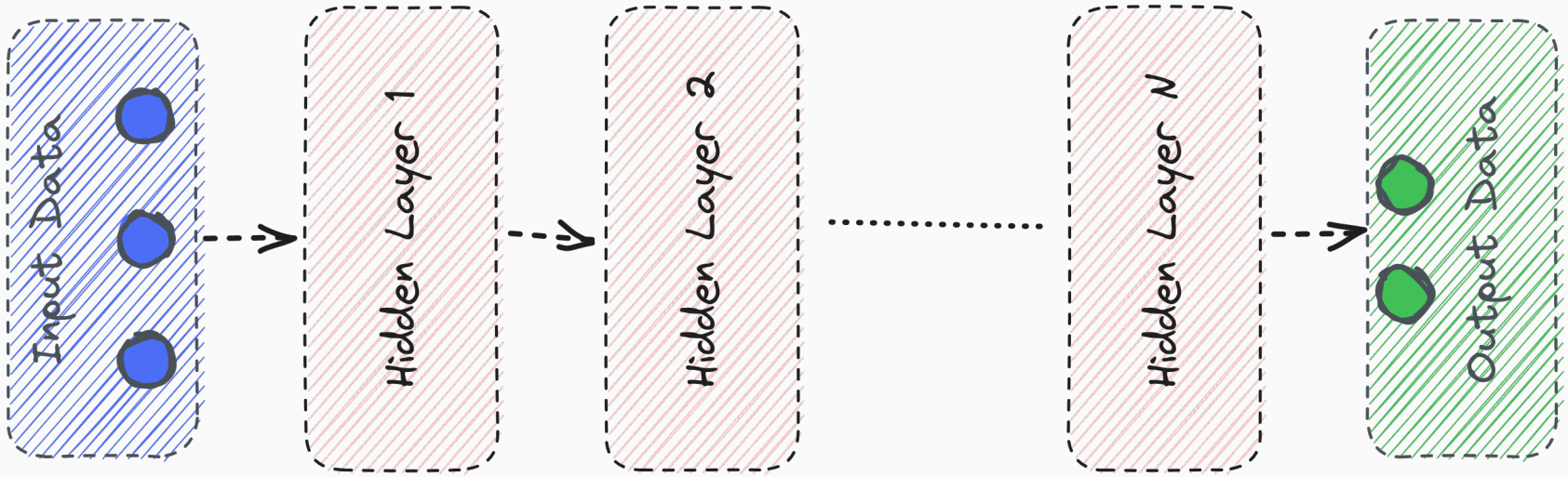


Figure 5: Different activation functions

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 differents functions possible

Summary: Hidden layer

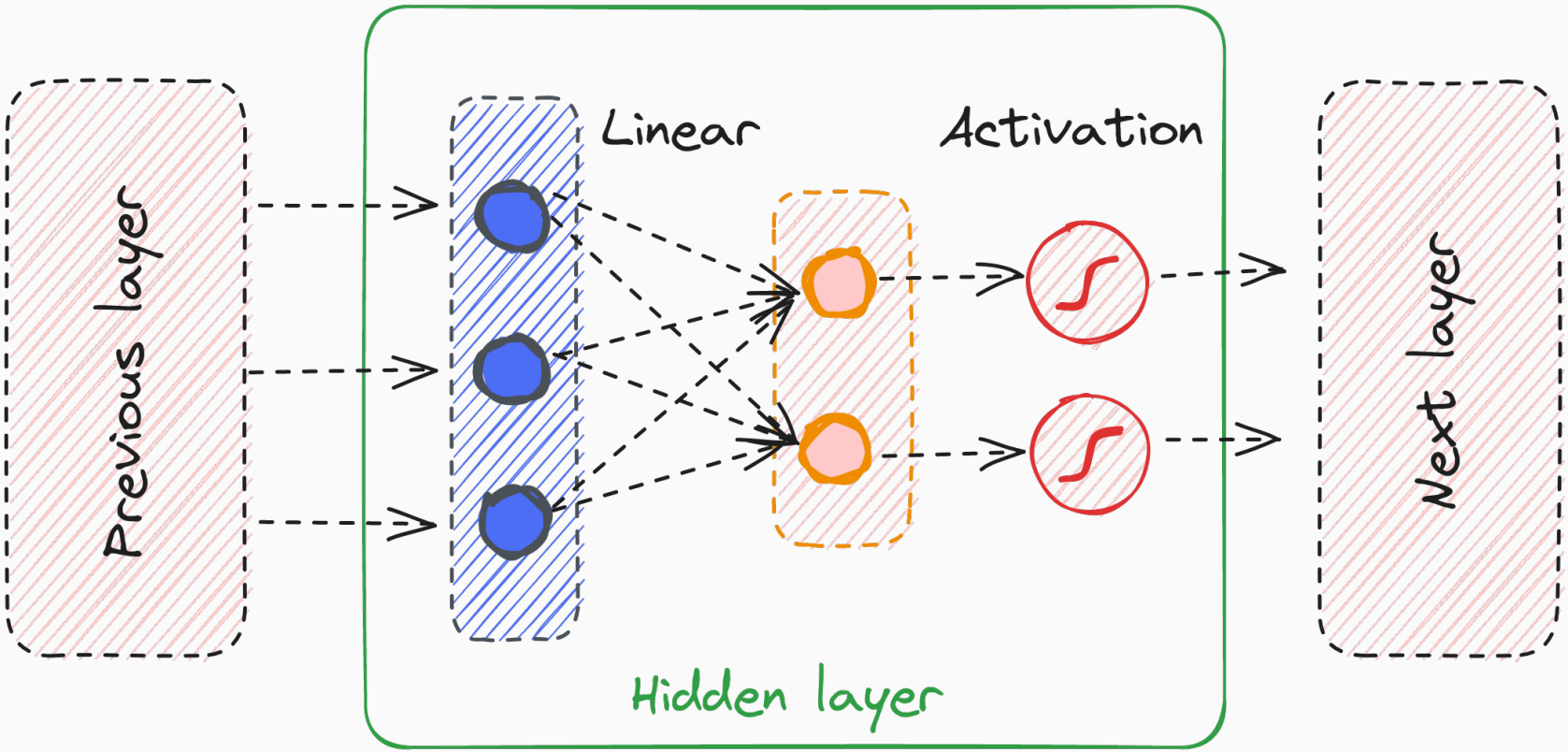


Figure 6: Classical representation of a neural network layer

A full example

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

$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{A}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$$

$$\boldsymbol{z}^{(2)} = \sigma\left(\boldsymbol{A}^{(2)}\boldsymbol{z}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

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

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

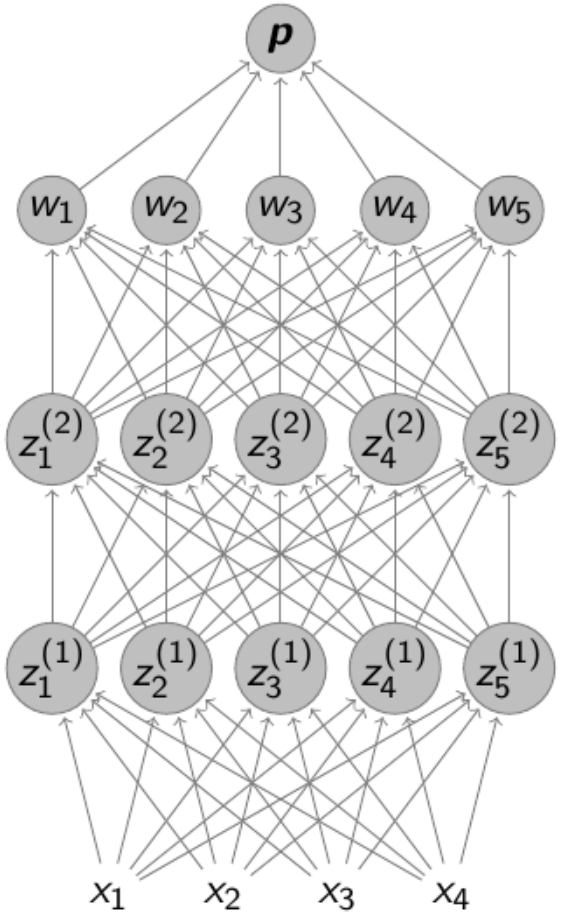


Figure 7: Mathematical and Graphical model side by side

Predictions function

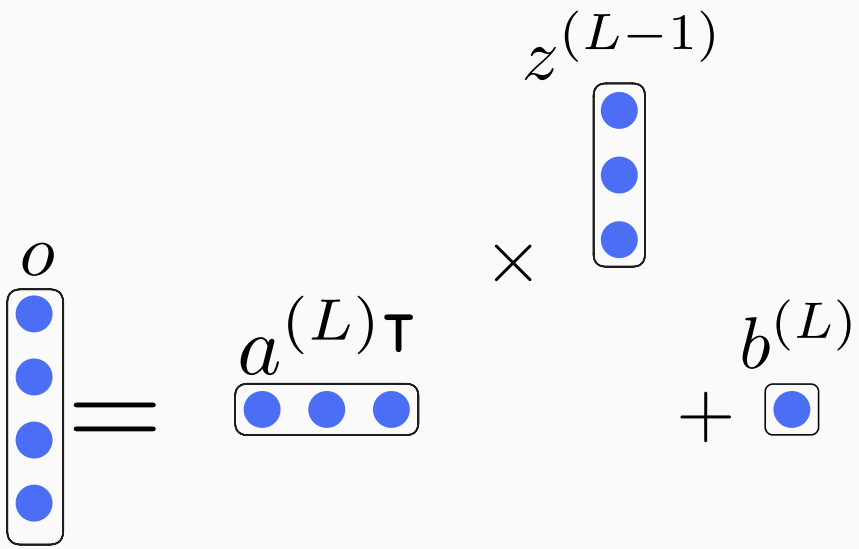
Vocabulary issue

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)

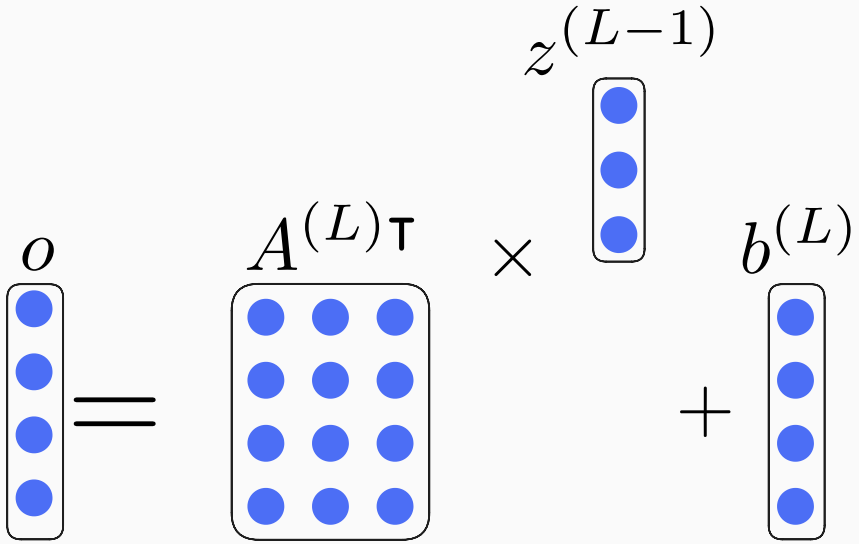
Loss functions in Binary classification



Loss functions

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

Predictions function in Multiclass classification



Prediction functions:

- **Integer output:** $\hat{y} = \underset{i \in \{1,\dots,I\}}{\arg\max}\, o_i$
- **One-hot vector output:** $\hat{y} = \underset{h \in E(k)}{\arg\max} < h, o >$

  $\rightarrow E(k) = \{h \mid h \in \mathbb{R}^k; \exists h_i = 1; \sum_{i=1}^{k} = 1\}$
- **Probabilistic output (distribution):** $\hat{y} = softmax(o)$

Precision on the sotmax

Softmax function:

For a vector o the softmax is defined by:

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

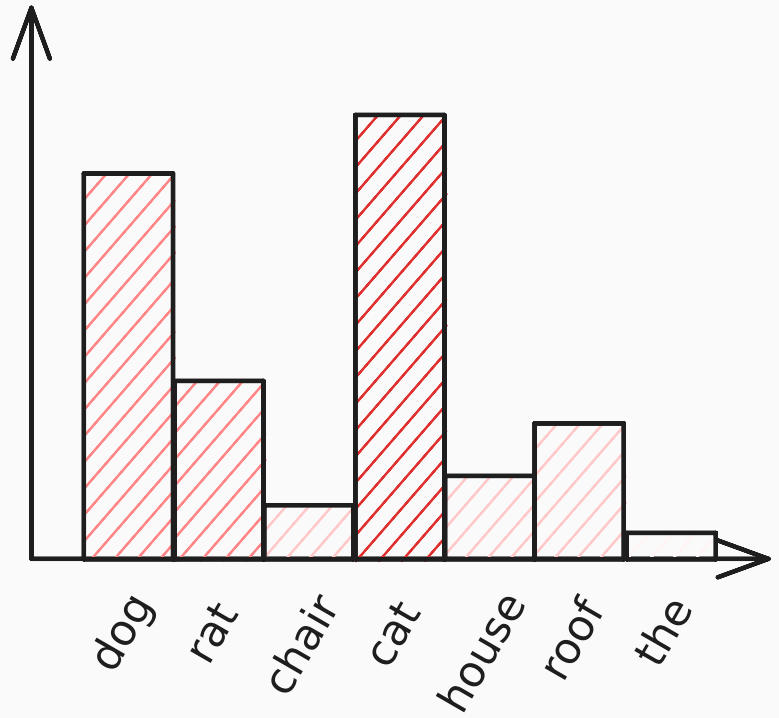


Figure 8: Model distribution to predict a word

Softmax function:

For a vector o the softmax is defined by:

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

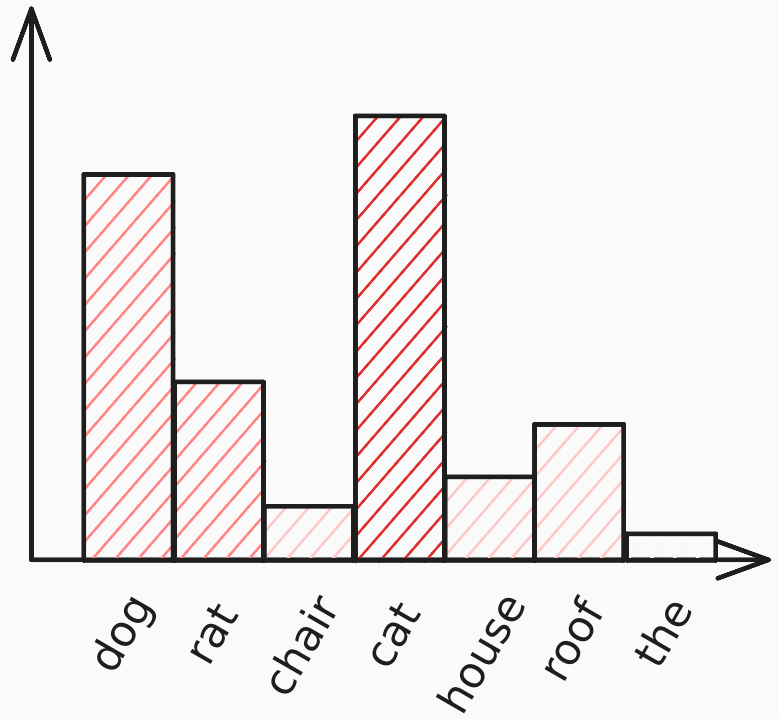
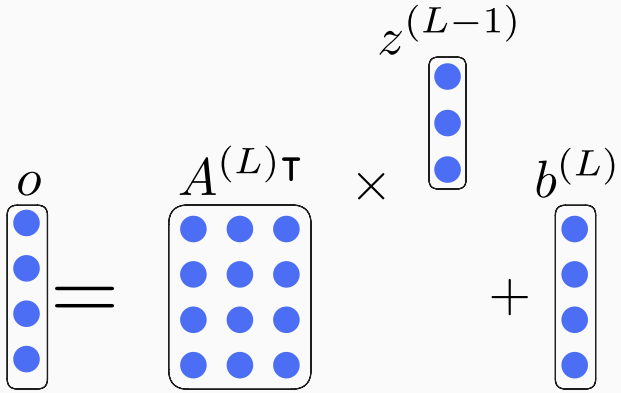


Figure 8: Model distribution to predict a word

Using Exponential?

- Having a spiked distribution
- Property in training

loss function in Multiclass classification

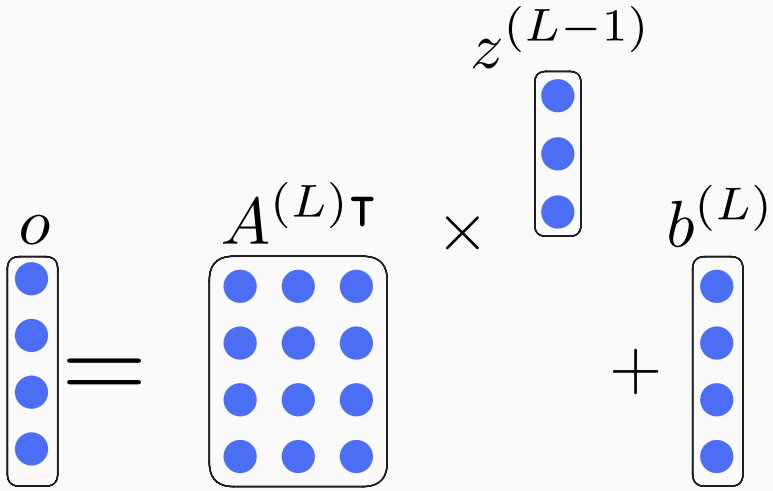


Loss functions:

- **Hinge loss:** $\mathcal{L}(o, y) = \max(0, m - < y, o > + \max_{h \in E^k \setminus \{y\}} (< h, o >))$
- **Negative Log-likelihood:(y one hot)** $\mathcal{L}(o, y) = - < y, o > + log \sum_{i=1}^{k} e^{o_i}$
- **Negative Log-likelihood:(y class index)** $\mathcal{L}(o, y) = -o_y + log \sum_{i=1}^{k} e^{o_i}$

---

[0] The log-likelihood (cross-entropy) is written $-\sum_{i=1}^{k} p_i log(q_{o_i})$ with p_i probability of the label i (notice that in multiclass only $p_y = 1$ and $p_{i \neq y} = 0$) and q_{o_i} the probability of the class i in output vector (using a softmax). Unrolling the calculus of general case will lead to the formula.



Prediction function:

- $\hat{y} = o$

Loss functions:

- **Mean Square Error (MSE):** $\mathcal{L}(o, y) = (y - o)^2$
- **Absolute Error Loss:** $\mathcal{L}(o, y) = |y - o|$

Finding Parameters in neural networks

Gradient-based Training

Neural network

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

Training

- Labeled example: feature + "gold" (class) answer
- Train set: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$
- Find parameters Θ such that $\forall (x, y) \in \mathcal{D}$:

$$f_\Theta(x) \approx y$$

Testing or evaluation

- Test if the model generalizes to unseen data

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/log likelihood 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

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

Finding θ s.t we minimize

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

$\rightarrow$ We consider E differentiable (E at least C_1)

Finding θ s.t we minimize

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

$\rightarrow$ We consider E differentiable (E at least C_1)

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

Finding θ s.t we minimize

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

$\rightarrow$ We consider E differentiable (E at least C_1)

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

$\rightarrow$ A unique minimum

Finding θ s.t we minimize

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

$\rightarrow$ We consider E differentiable (E at least C_1)

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

$\rightarrow$ A unique minimum

$\rightarrow$ Gradient of error is null

Finding θ s.t we minimize

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

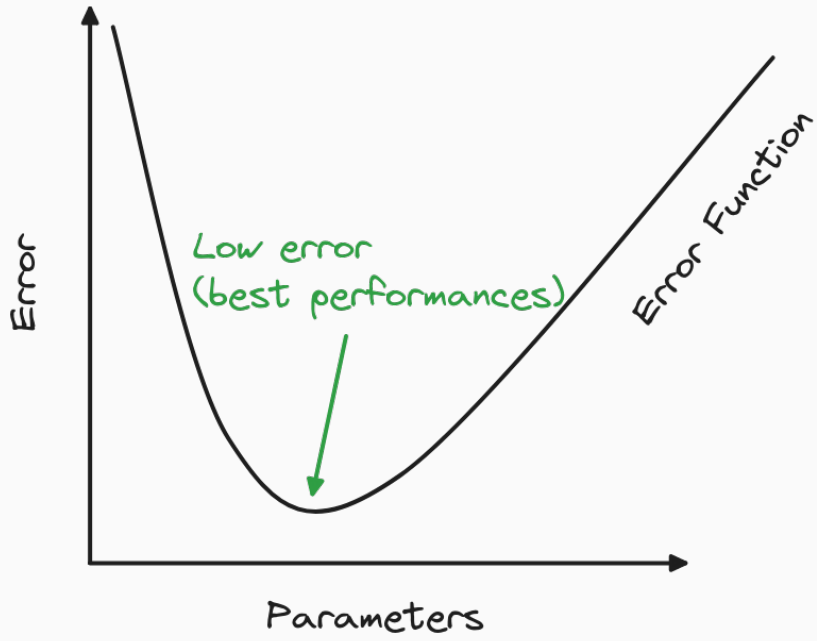
$\rightarrow$ We consider E differentiable (E at least C_1)

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

$\rightarrow$ A unique minimum

$\rightarrow$ Gradient of error is null

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



Finding θ s.t we minimize

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

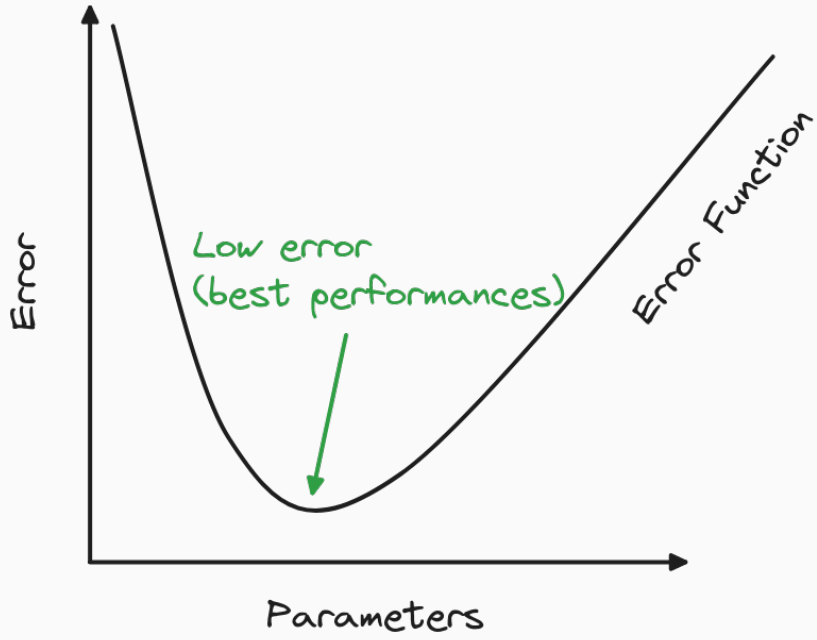
$\rightarrow$ We consider E differentiable (E at least C_1)

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

$\rightarrow$ A unique minimum

$\rightarrow$ 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: Θ_0
- Make T iterations/steps: $\Theta_{t+1} = \Theta_t - \lambda \nabla E(\mathcal{D}, f_\Theta)$
- λ is called the learning rate (or stepsize)

Gradient Descent:

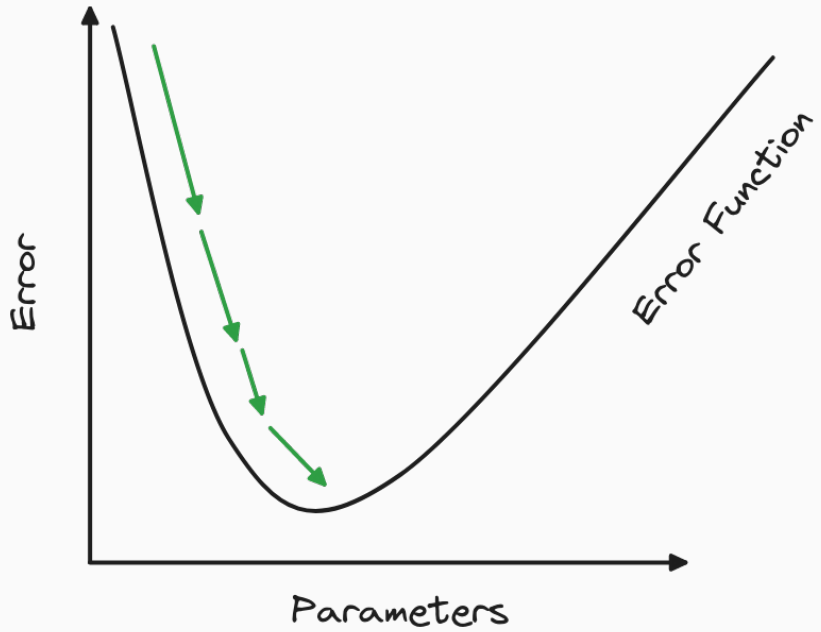
Intuition: Follow the error function "slope"

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

Gradient Descent:

Intuition: Follow the error function "slope"

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



Gradient Descent:

Intuition: Follow the error function "slope"

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

Carrefully choose λ (the gradient step)

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

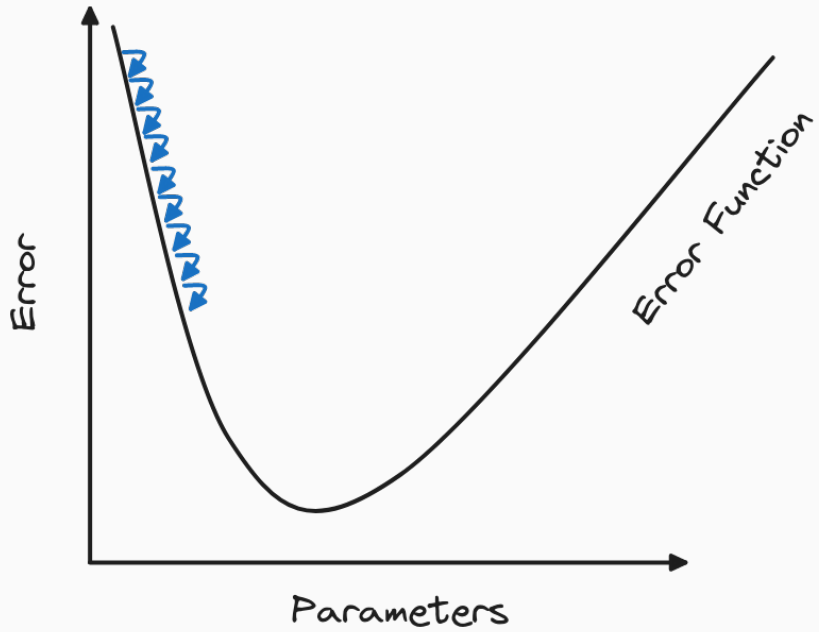
Gradient Descent:

Intuition: Follow the error function "slope"

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

Carrefully choose λ (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 λ (the gradient step)

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

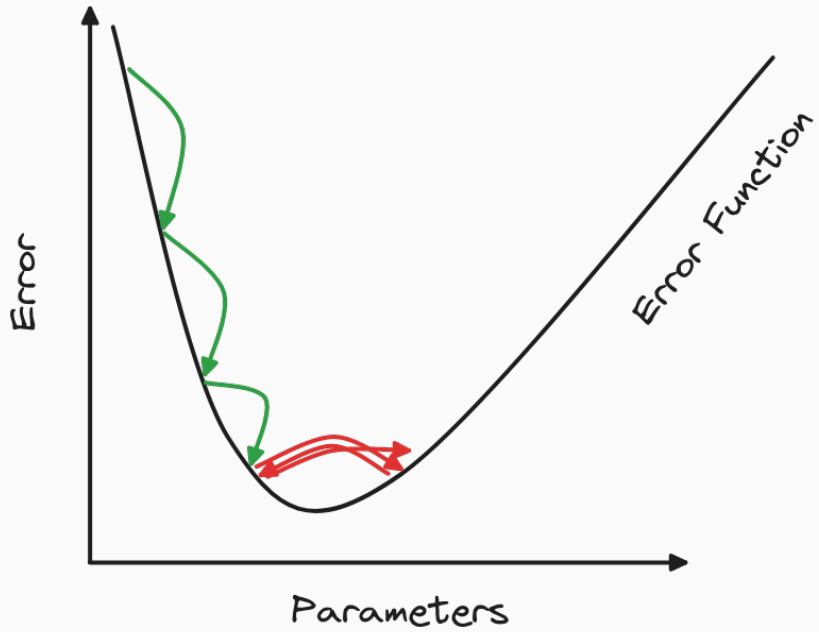
Gradient Descent:

Intuition: Follow the error function "slope"

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

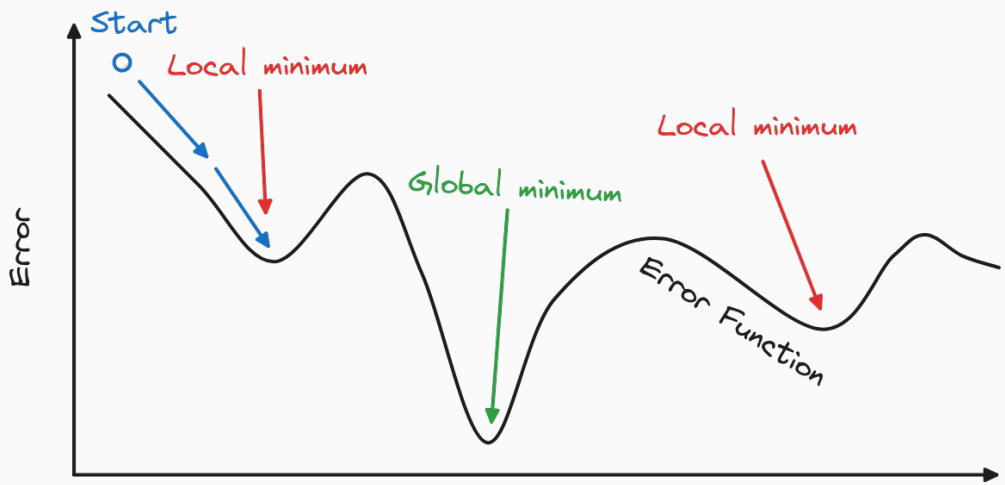
Carrefully choose λ (the gradient step)

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



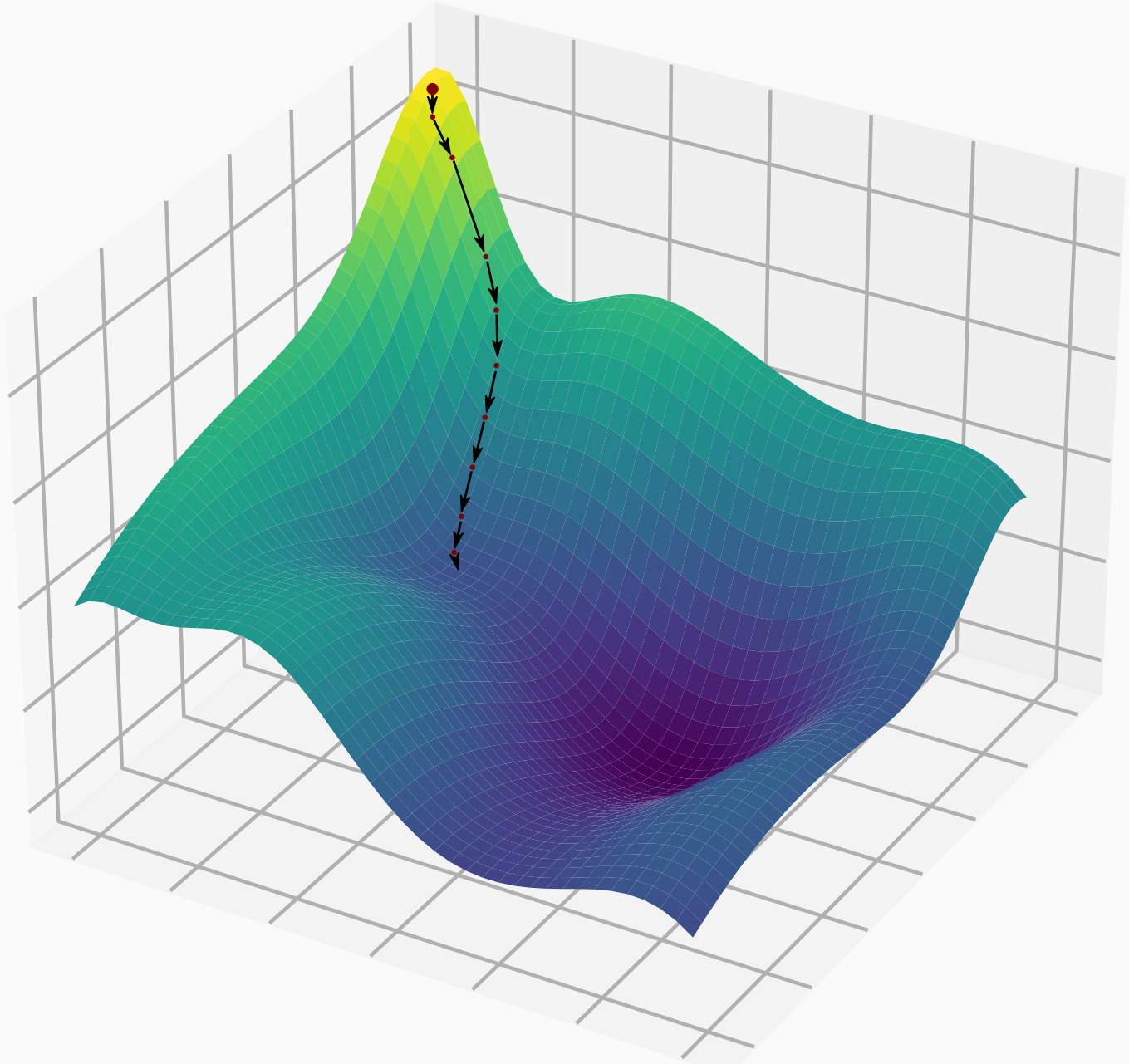
Non convex fonction

$\rightarrow$ suboptimal solution with gradient descent !

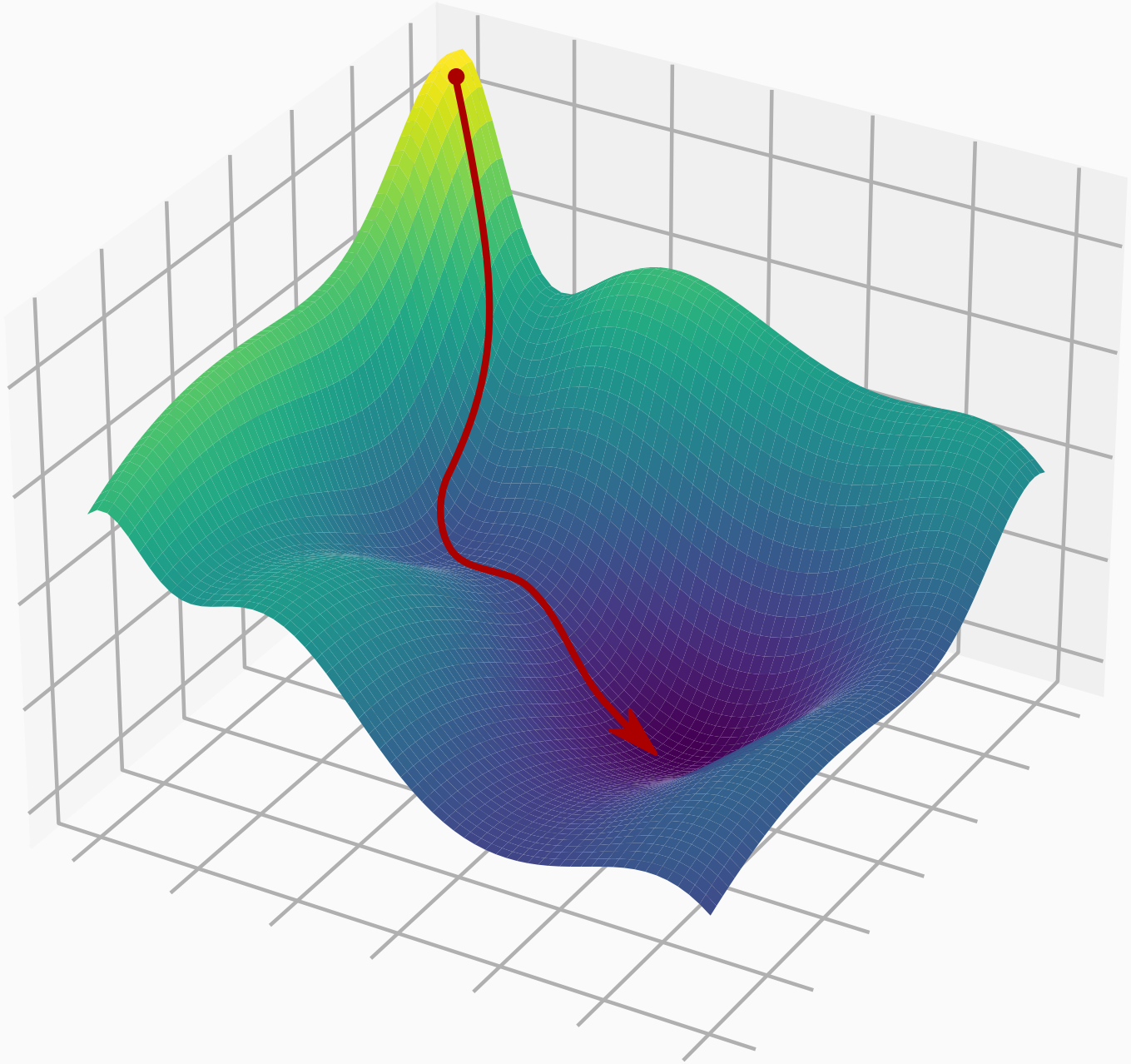


In practice we find local minima(and it is ok to not find global minimum)

Neural Networks: Cost function and Optimisation



Neural Networks: Cost function and Optimisation



Train, Dev and Test set

Parameters vs. hyper-parameters

- Parameters: the parameters of the function that are learned during training
- Hyper-parameters: the parameters of the training algorithm (learning rate), and the nueural architecture choice (number of layers, dimensions of hidden layer)

Three dataset

- **Train set:** use to compute the objective and its gradient
- **Development or validation set:** used during training to choose hyper-parameters (and to know when to stop training)
- **Test set:** used to evaluate the model (never during training)

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

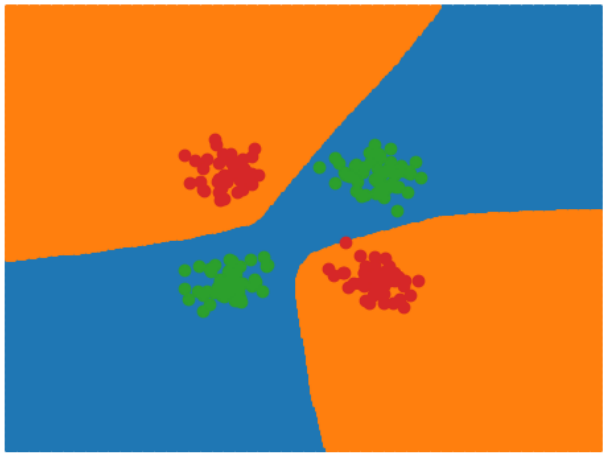
Monitoring the training process

- 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 mutlilayer perceptron)

Handle non linearity: example (3)

Animation of the same problem

Neural Architectures

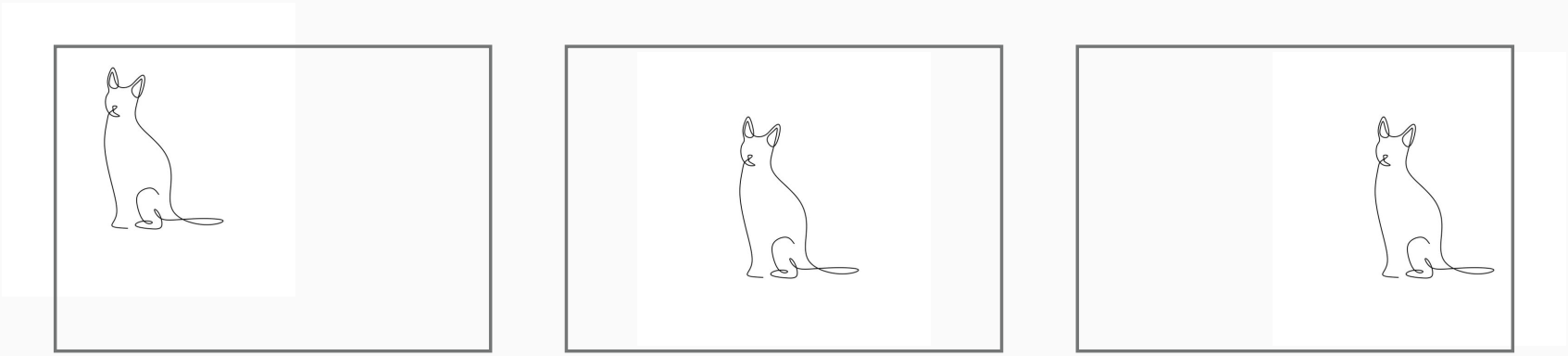
Neural Architecture design

A neural network is a complicated parameterized function

How to choose the function?

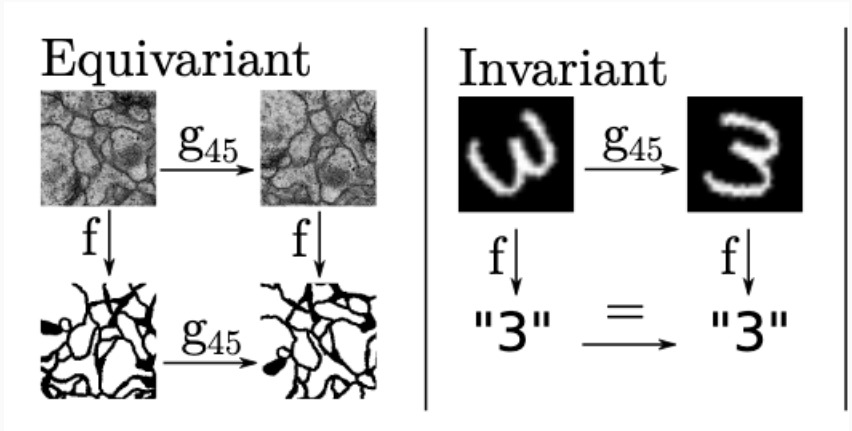
- Inductive bias: take into account the data to design the architecture
- Time complexity/speed (operation time, number of parameters)
- Mathematical properties for efficient training:
  $\rightarrow$ Differentiability, prevent vanishing/exploding gradients...

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

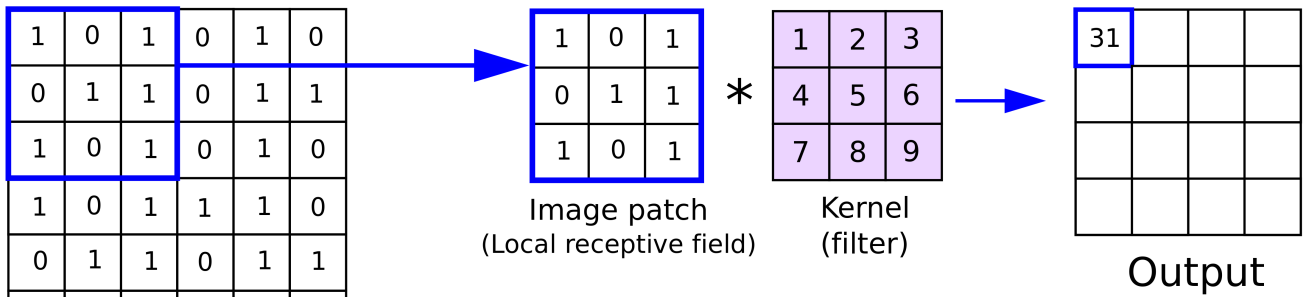


CNN block (not formal)

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 $poling(p_i)$

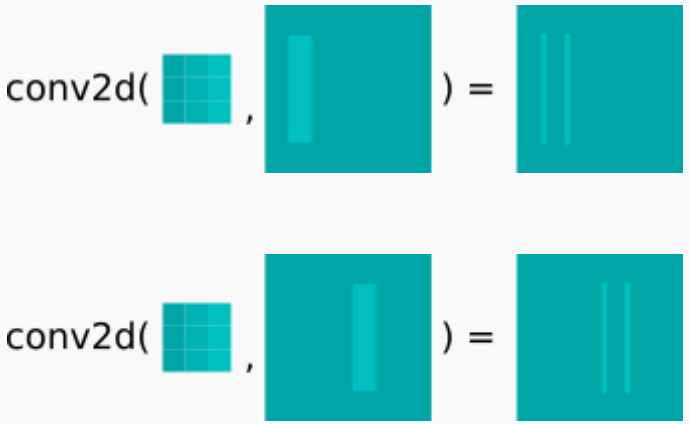
Poling method can be the mean, the max,... **Many filter, Many block**



Translation equivariant convolution

Preserves the "translation structure"

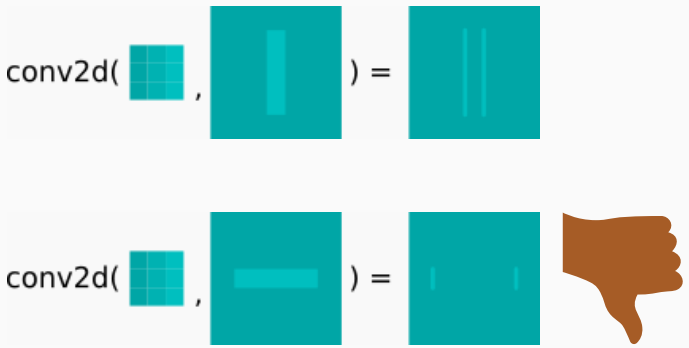
- Input transposed $\Longrightarrow$ output transposed
- Pooling make it invariant



Rotation equivariant convolution

Preserves the "rotation structure"

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



Standar convolution is not rotation equivariant

[0] Image from `https://anhreynolds.com/blogs/cnn.html`

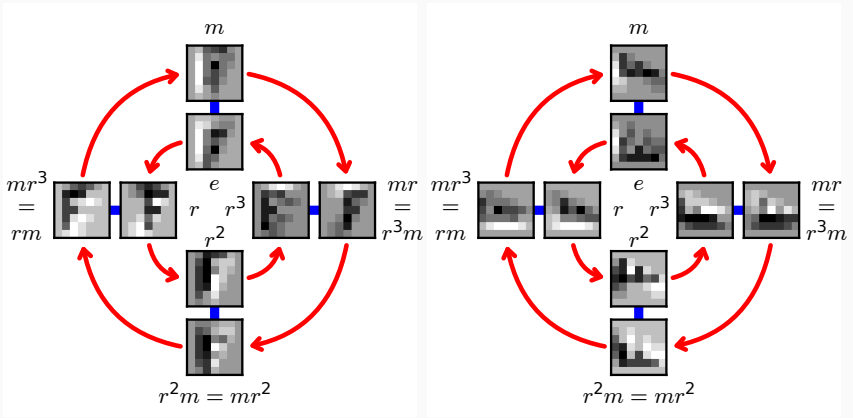
Taking into account rotations (CNN)

Invariant by rotation

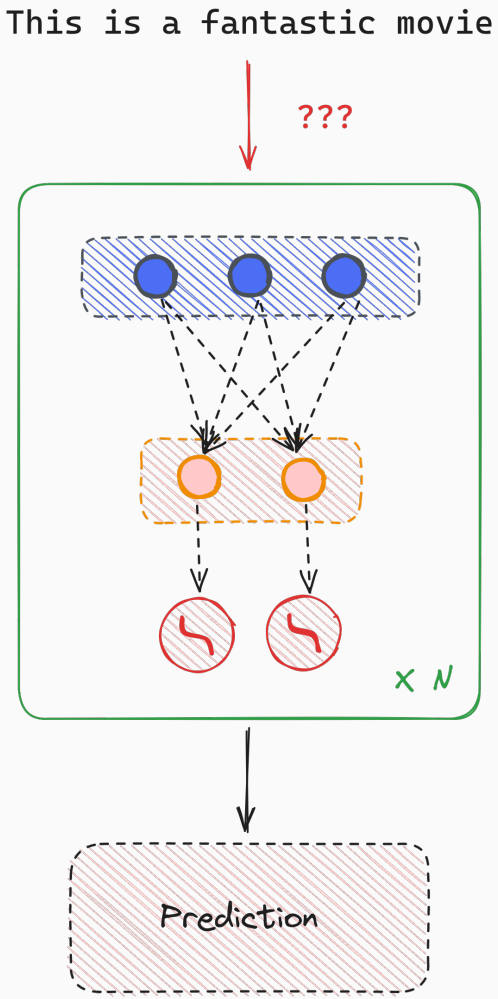
An image rotated will not be classified similarly from the original

$\rightarrow$ consider rotation of filters

Proposal of p4 map



Encoding sequence:(text example)

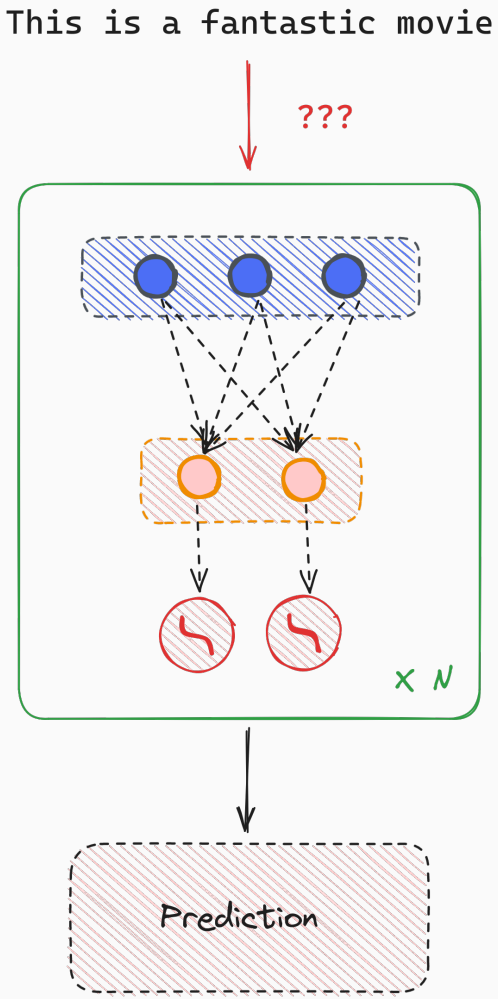


Encoding Textual Information

Neural networks or classifiers (linear for instance) :

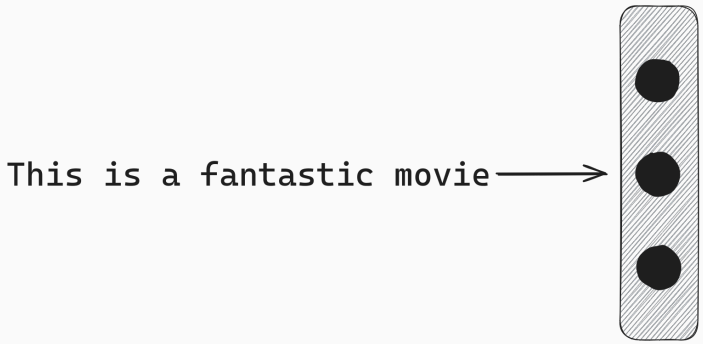
$\rightarrow$ **Need vectorial representation as input**

Encoding sequence:(text example)

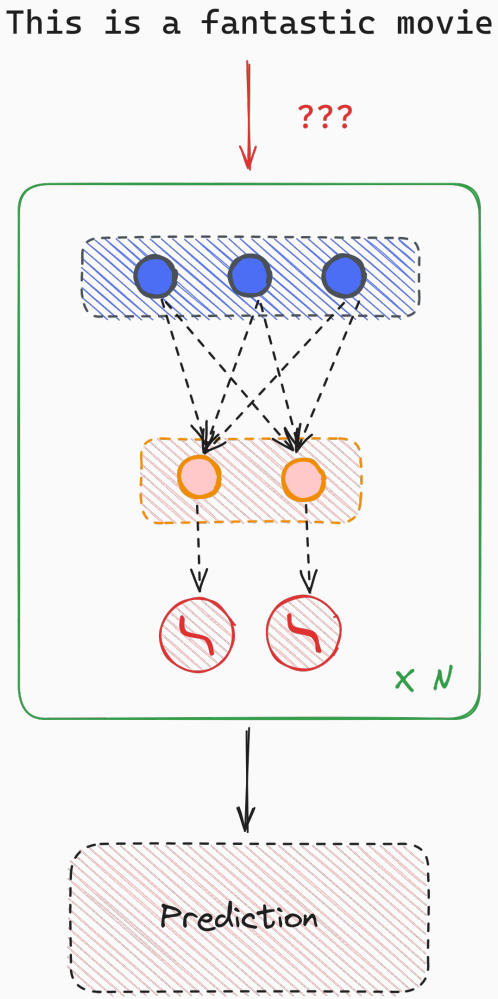


Encoding Textual Information

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

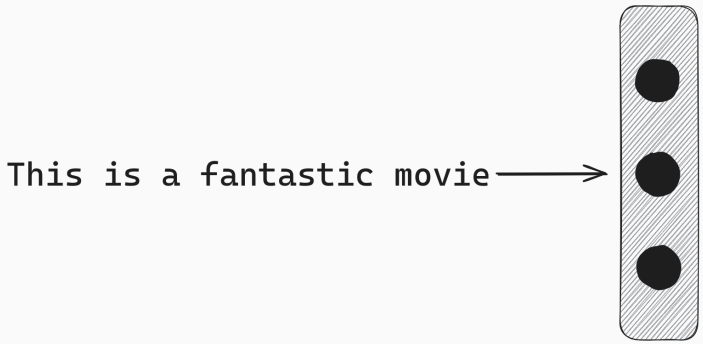


Encoding sequence:(text example)



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:

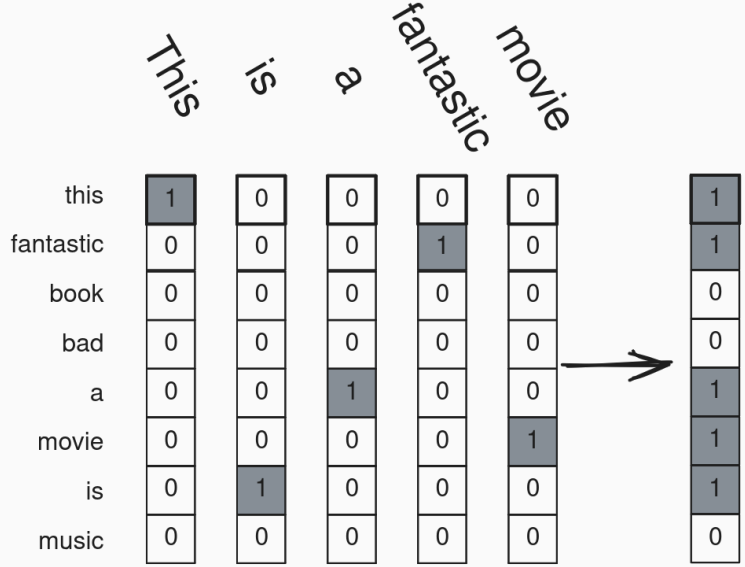
$$w_j = \begin{cases} i = j & 1 \\ i \neq j & 0 \end{cases}$$

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:

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

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

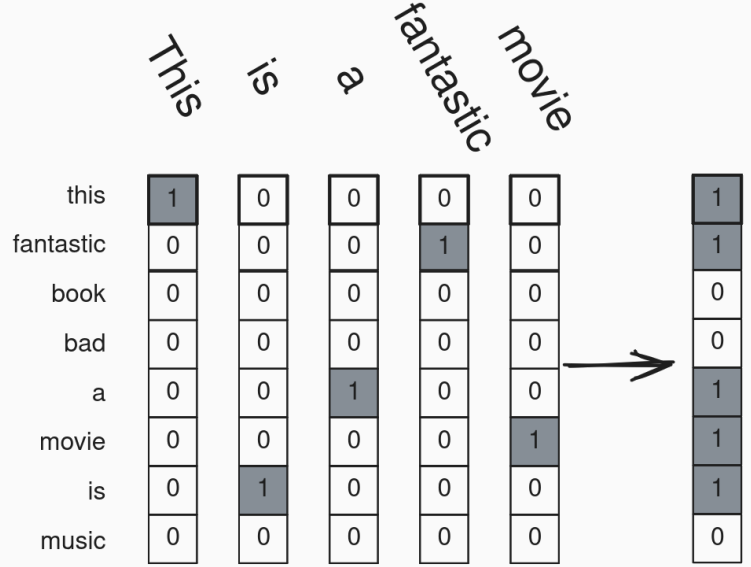


Bag of Word representation:

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:

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

A sequence of word $w^1, w^2, \ldots, w^n$ 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 :
$\rightarrow$ The representation of the word “cat” will be as close from “dog” and “car”
$\rightarrow$ There is no semantics in the representation

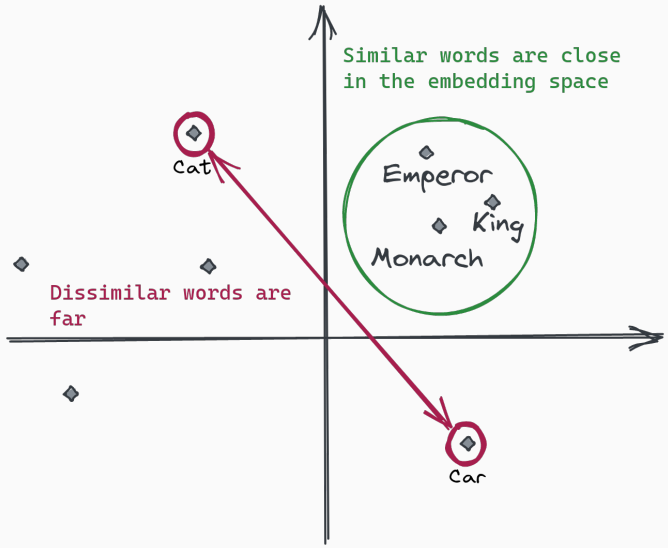
Bag of words issue:

BoW representation does not capture proximity between words :
$\rightarrow$ The representation of the word "cat" will be as close from "dog" and "car"
$\rightarrow$ There is no semantics in the representation

Taking into account semantics:

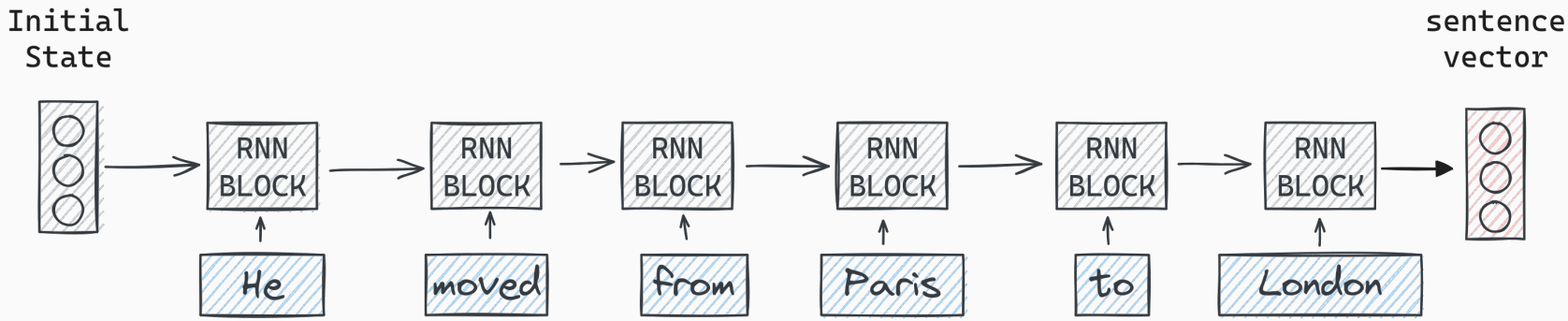
Considering a similarity metric or a distance

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



Importance of words order

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



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:

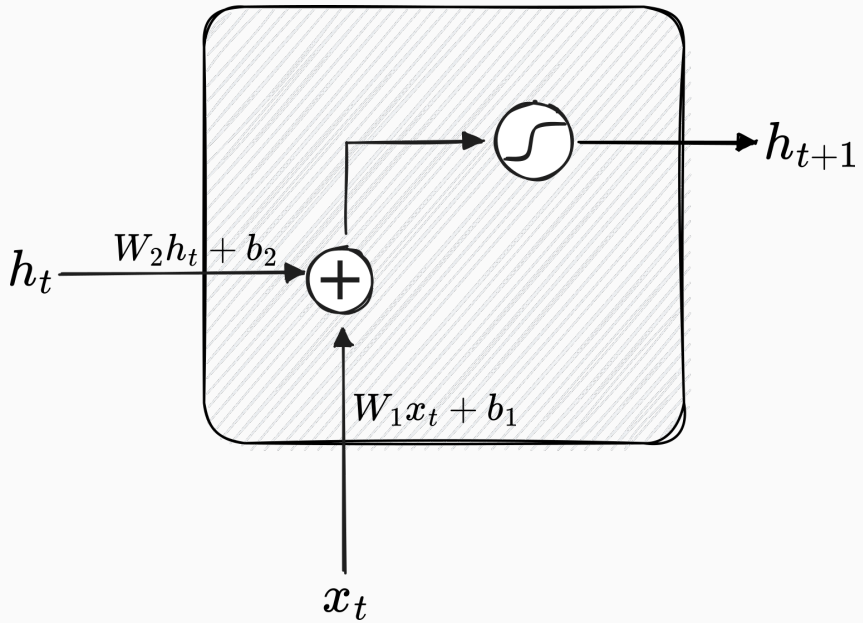
$$f_\theta : \mathbb{R}^D \times \mathbb{R}^n \to \mathbb{R}^D$$
$$(h_t, x_t) \mapsto h_{t+1}$$

Vanilla RNN

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

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

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



Vanilla RNN

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

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

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

Algorithm 1 Many to One forward

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

$h \leftarrow h_0$

for $t \leq T$ **do**

$h' \leftarrow W_h h$

$x' \leftarrow W_x x_t$

$h \leftarrow tanh(h' + x' + b\mathbf{1}^\intercal)$

end for

return $W_y h + b_y \mathbf{1}^\intercal$

Contextualize word in sentence: Vanilla RNN

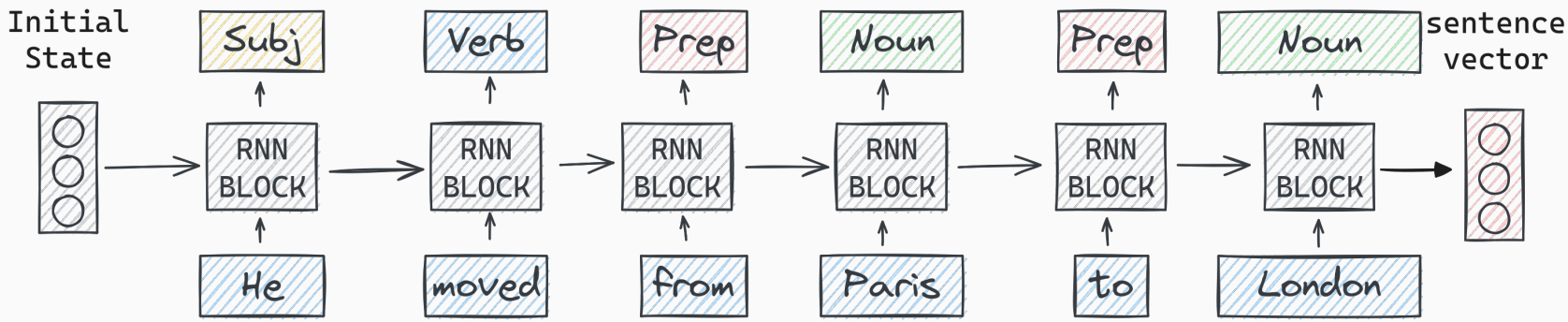


Figure 9: The many to many Variant

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

$\rightarrow$ In Part of speech tagging

Contextualize word in sentence: Vanilla RNN

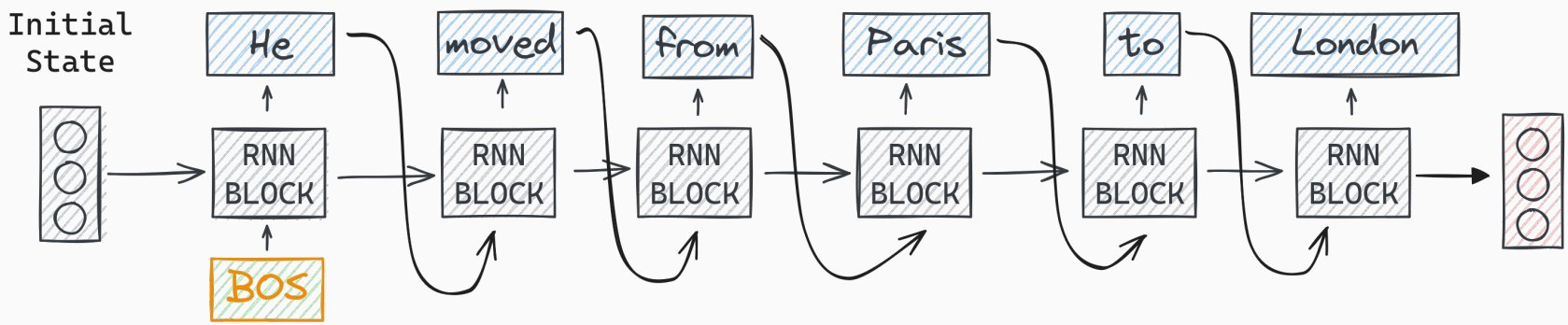


Figure 10: Geneative approache with RNN

One to Many RNN
Given an input x_1 generate an output

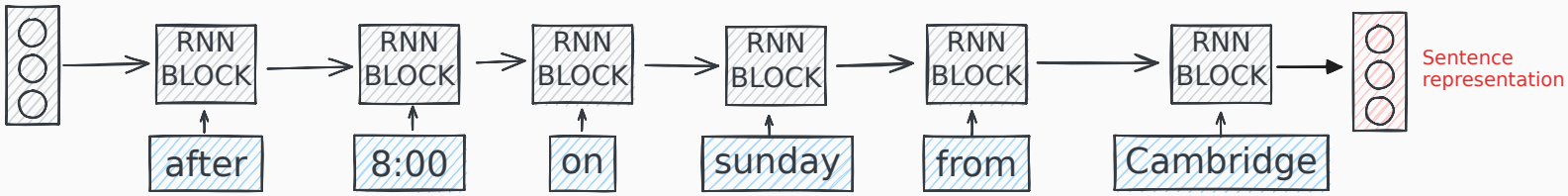
$\rightarrow$ In generative task

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

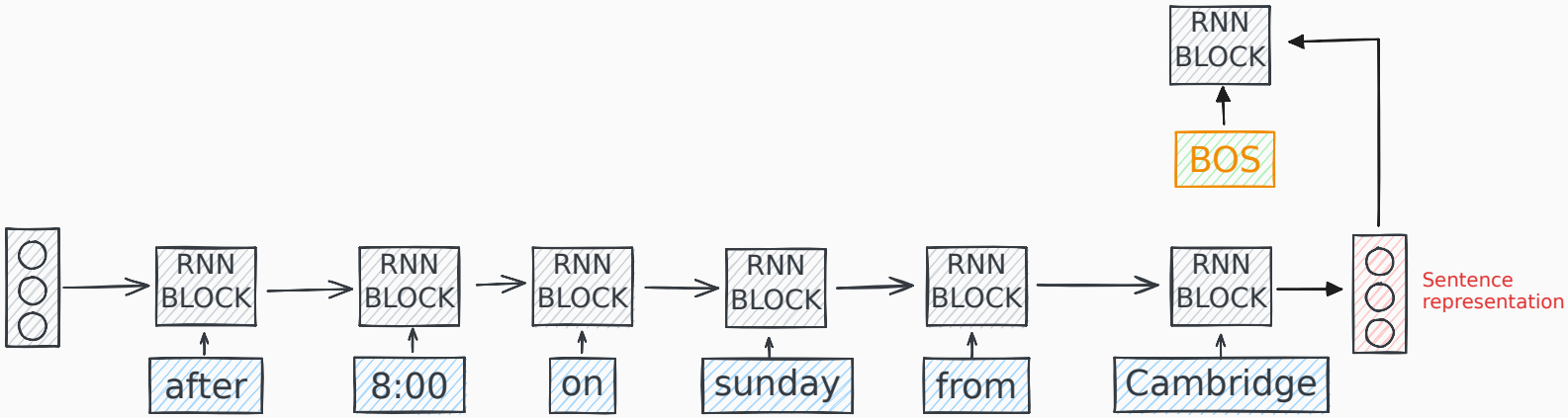


Sequence to Sequence

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

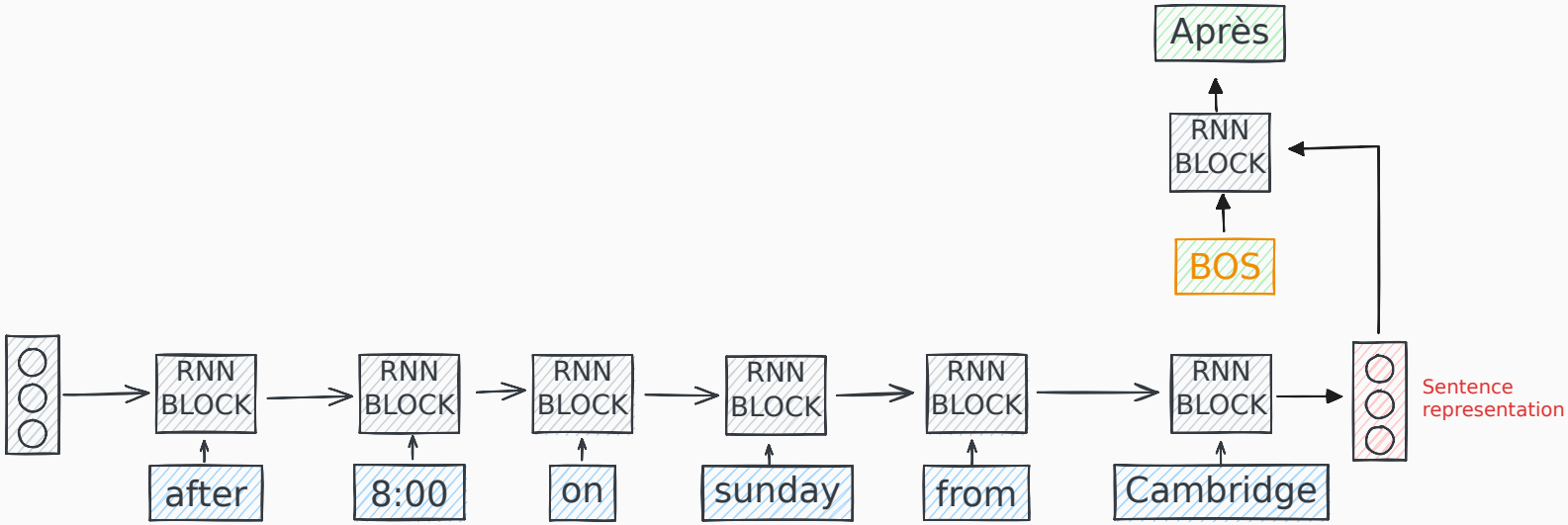


Sequence to Sequence

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

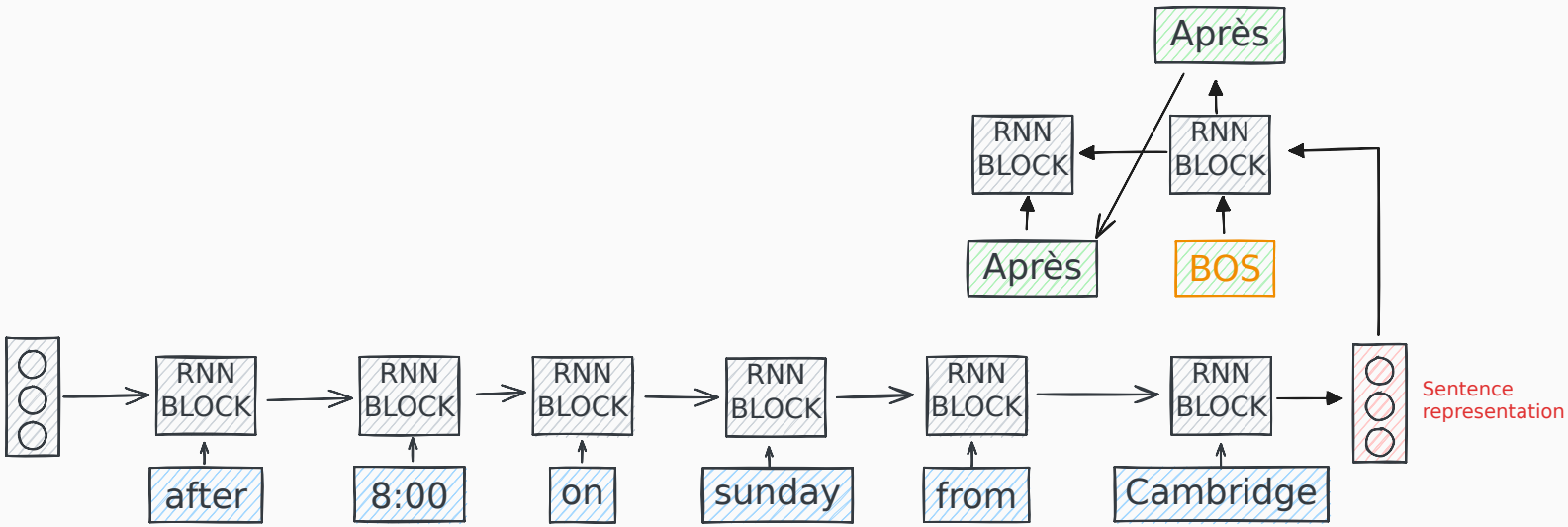


Sequence to Sequence

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

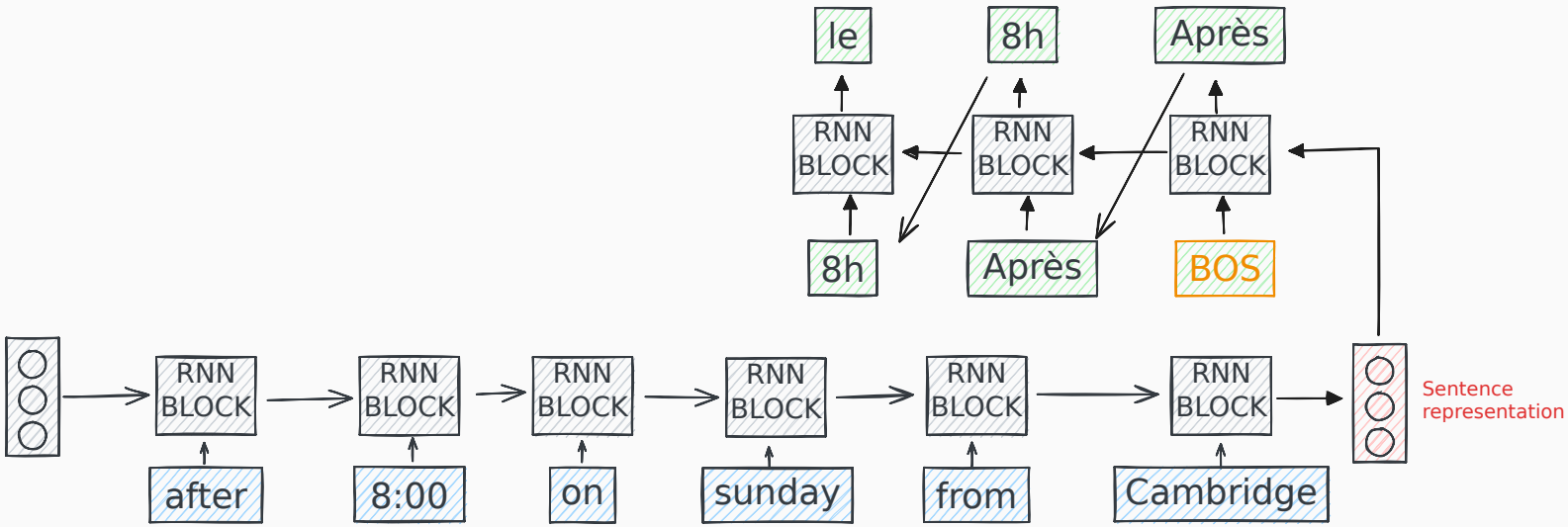


Sequence to Sequence

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

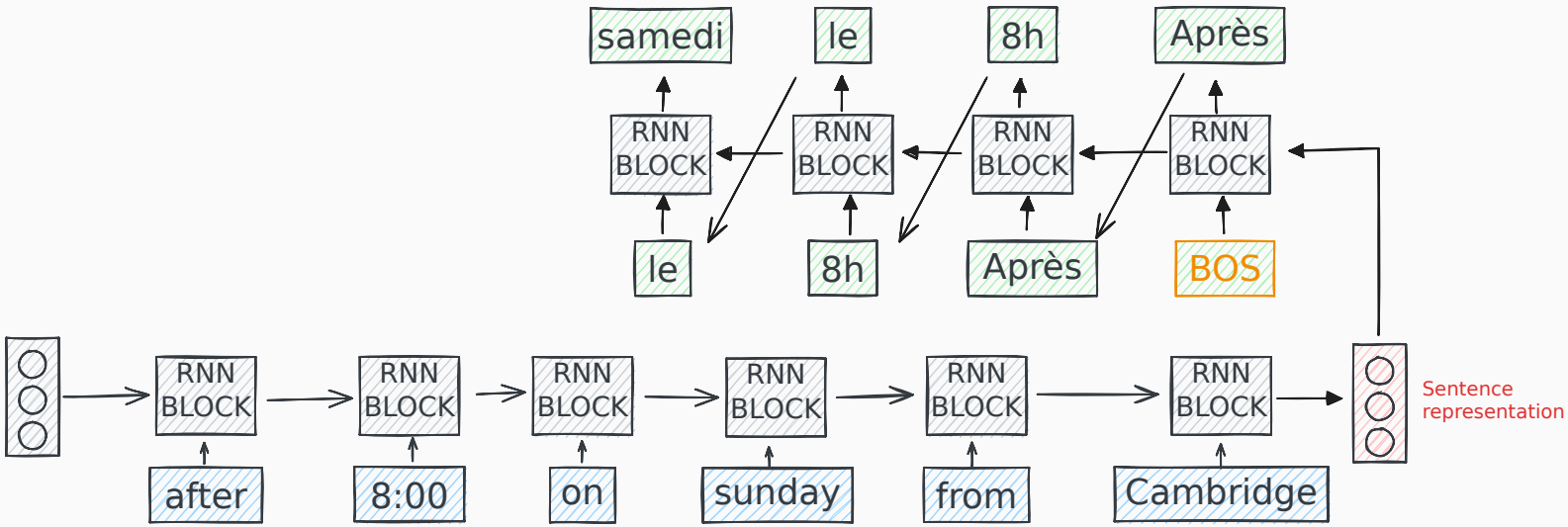


Sequence to Sequence

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

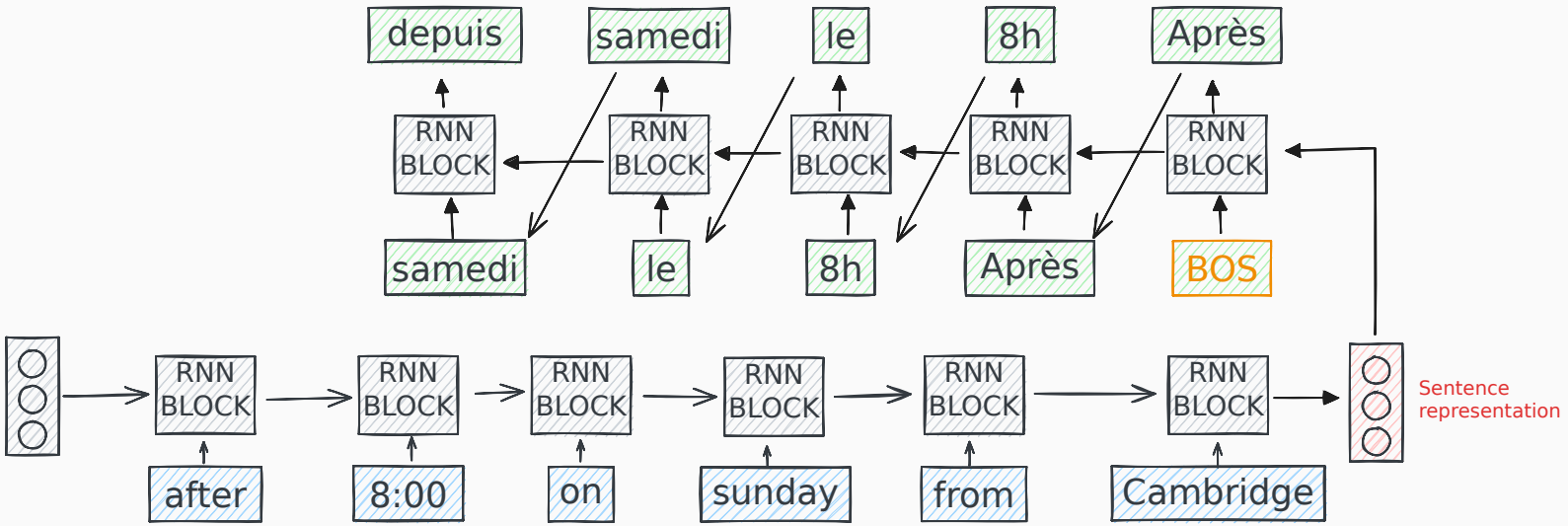


Sequence to Sequence

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

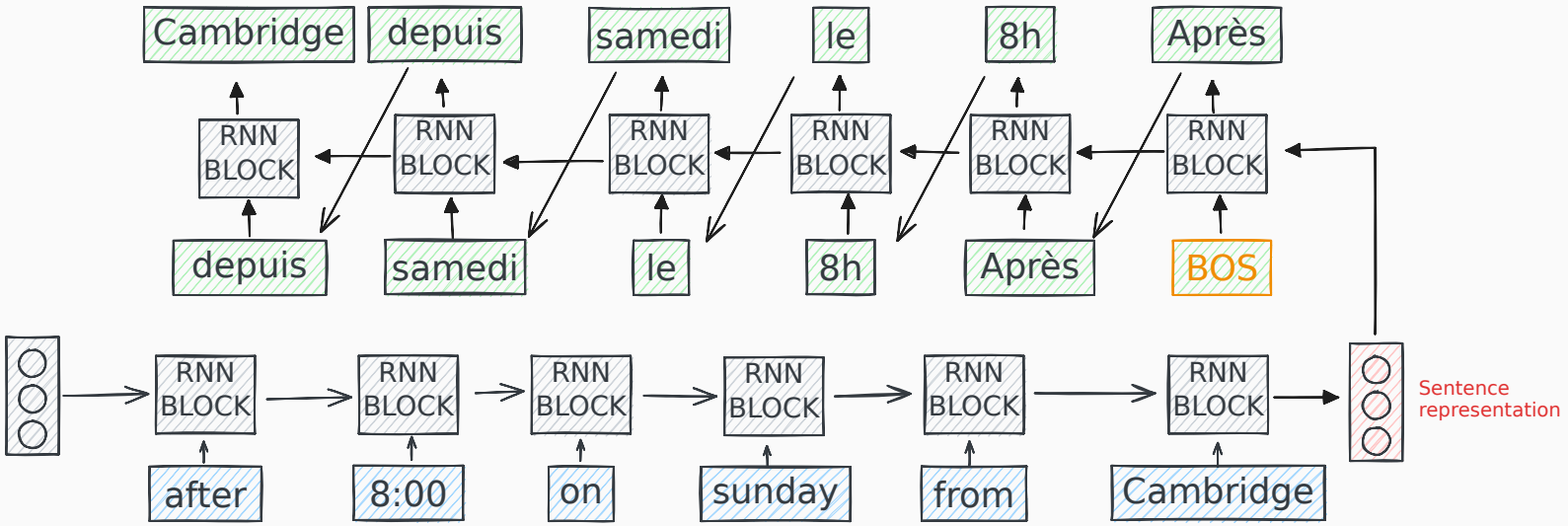


Sequence to Sequence

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

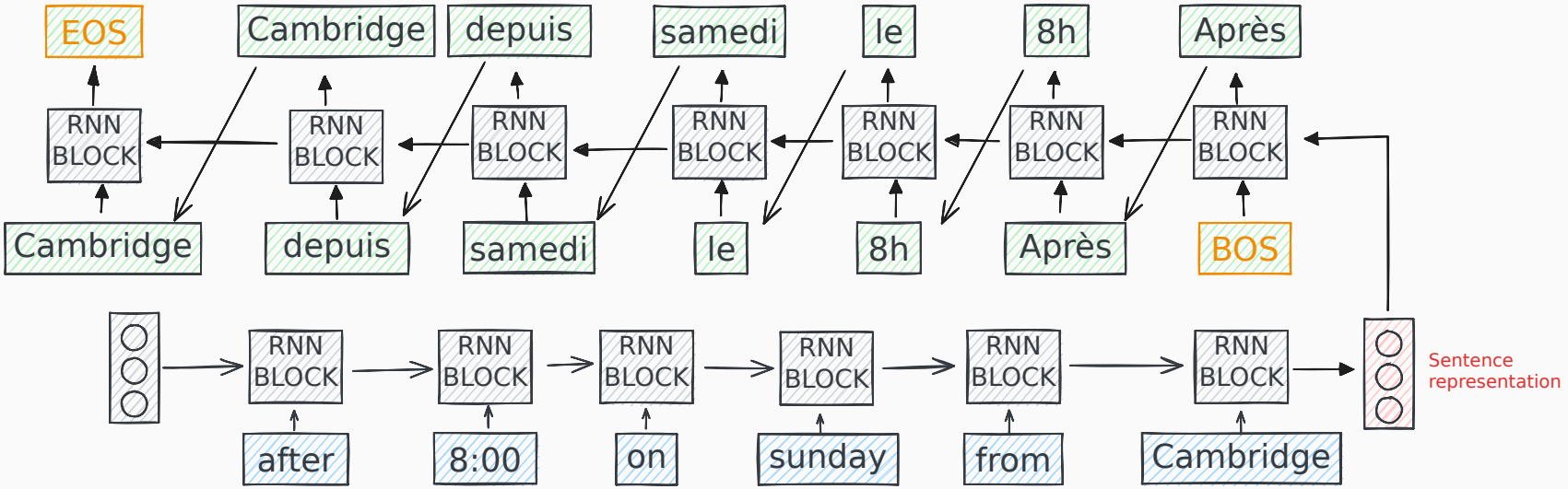


Sequence to Sequence

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

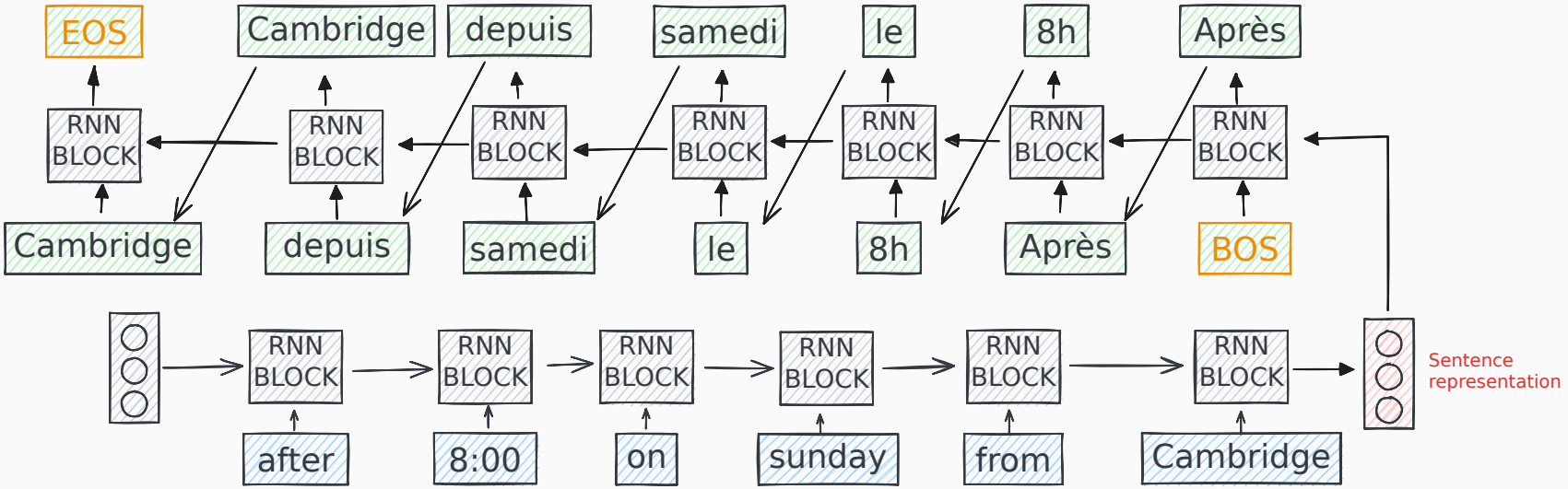


Sequence to Sequence

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

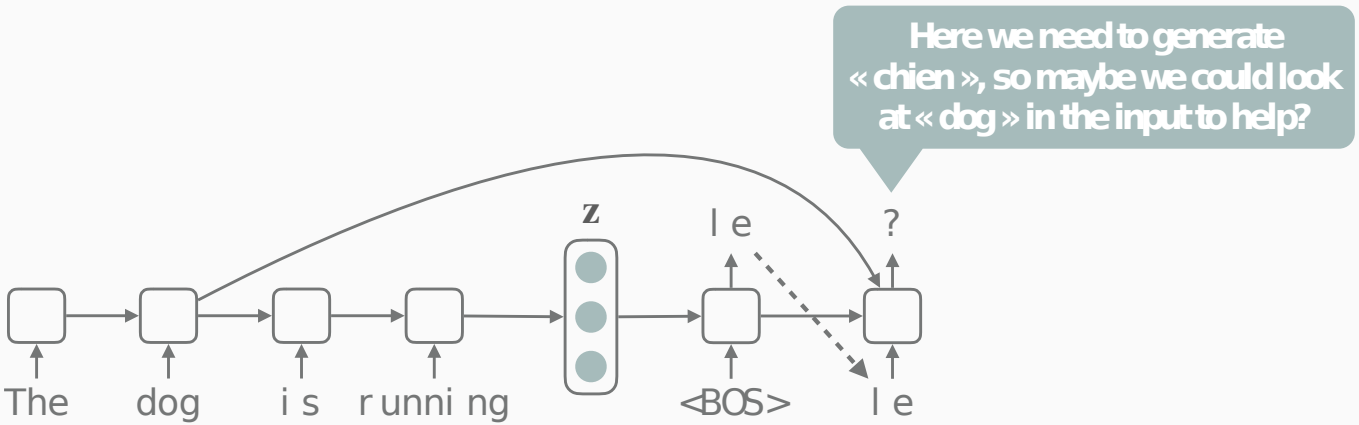
Sequence to Sequence with attention

Intuition

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

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 frrom the input

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):

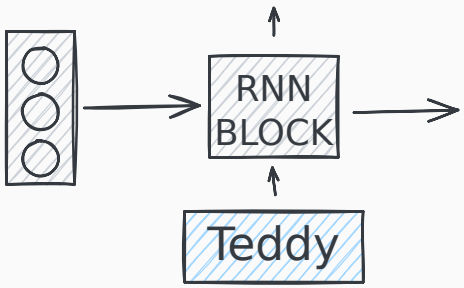
- The word correspond to a person: Pers
- The word does not correspond to a person: NPers

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

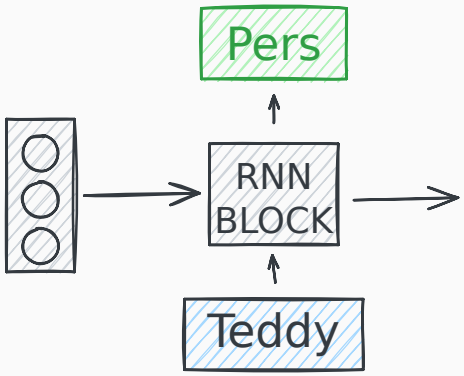


Contextualize word in sentence: RNN unidirection issue

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):

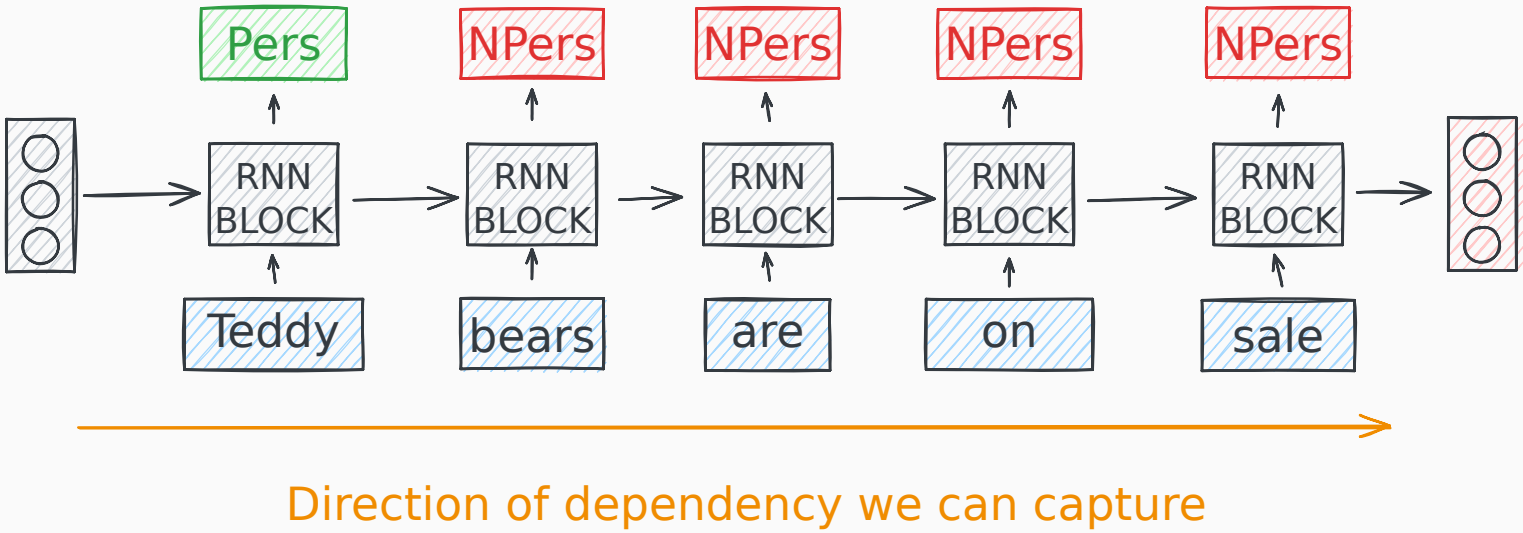
- The word correspond to a person: Pers
- The word does not correspond to a person: NPers

Contextualize word in sentence: RNN unidirection issue

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

$\rightarrow$ For some task it is necessary to know the next words before prediction

Contextualize word in sentence: RNN unidirection issue

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

Contextualize word in sentence: RNN unidirection issue

$\rightarrow$ For some task it is necessary to know the next words before prediction

Let consider two RNNs:

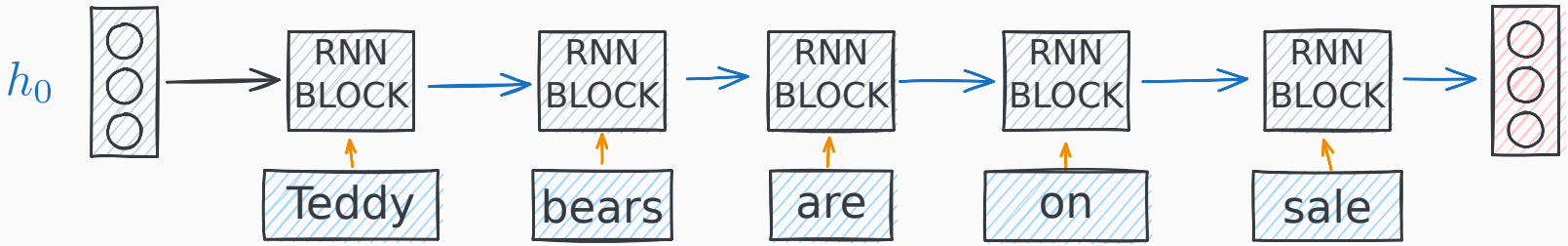
- The first RNN processing forward the sequence and produce hidden state $\{h_1, h_2, \ldots, h_T\}$
- A second RNN processing the sequence backward and produce hidden state $\{h'_T, h'_{T-1+1}, \ldots, h'_1\}$

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

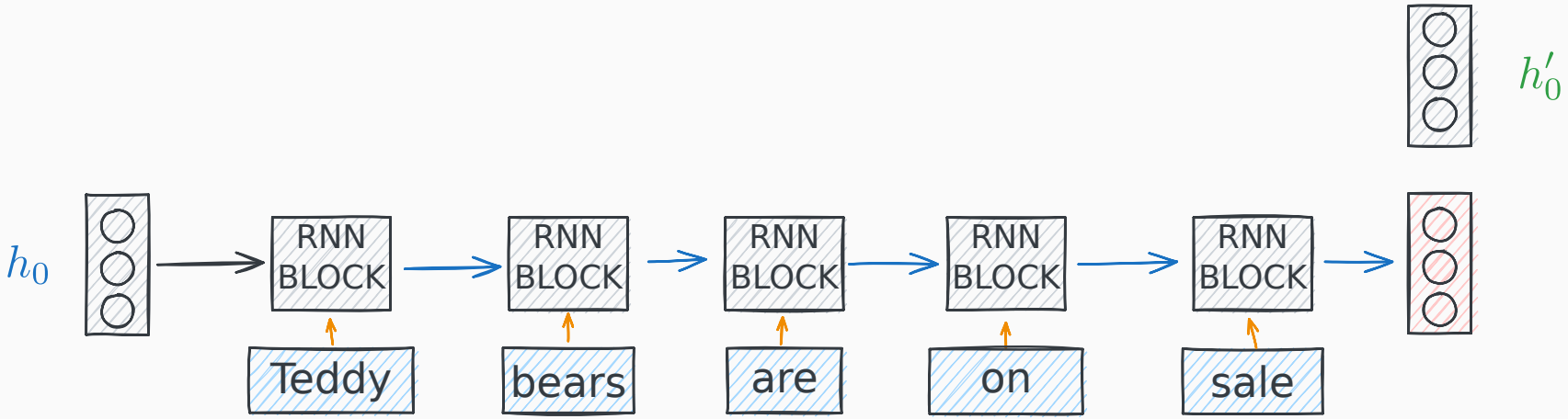
$$f_\theta : \mathbb{R}^d \times \mathbb{R}^d \rightarrow [-1, 1]$$
$$(h_t, h'_{T-t+1}) \mapsto tanh(W_{f_1} h_t + W_{f_2} h'_{T-t+1} + b_f)$$

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

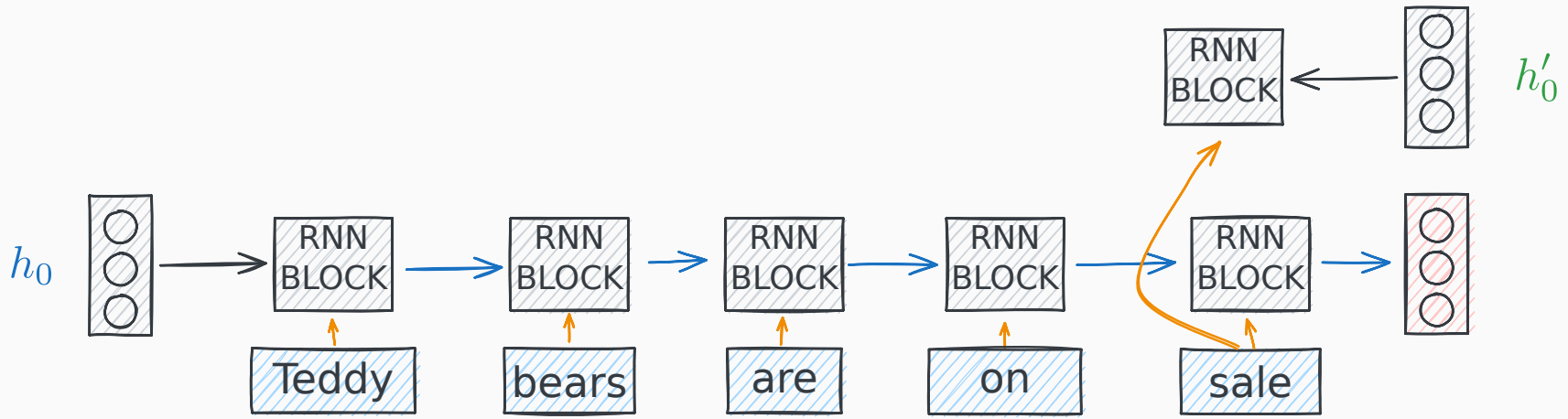
Contextualize word in sentence: RNN unidirection issue



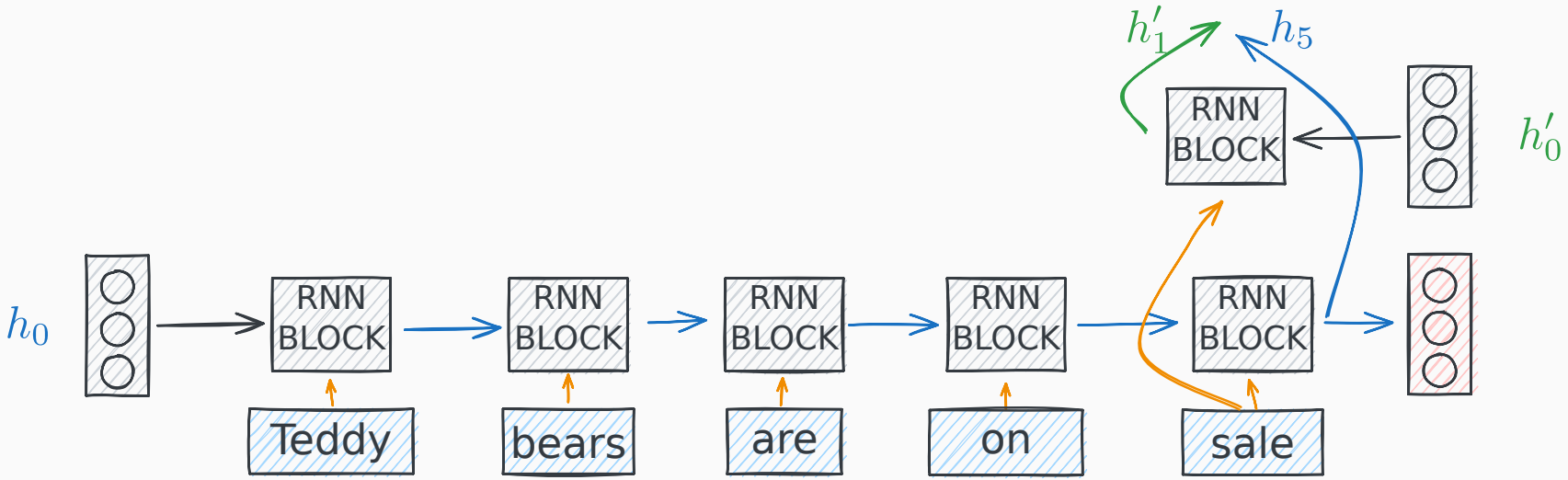
Contextualize word in sentence: RNN unidirection issue



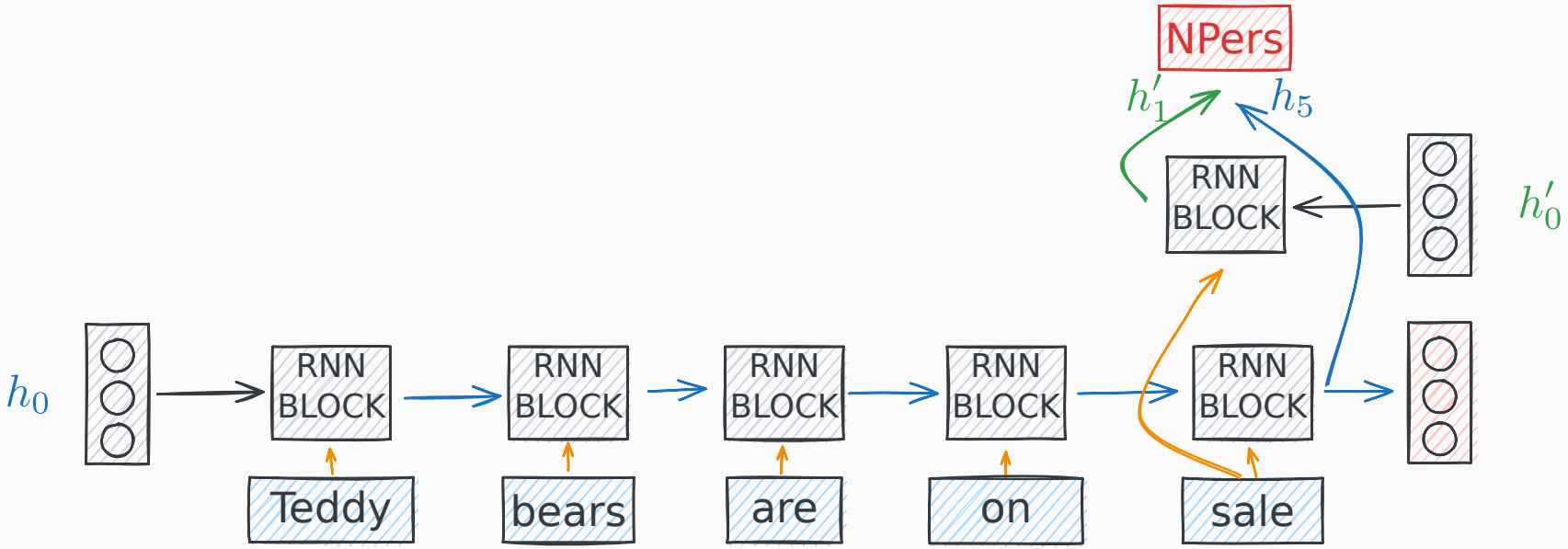
Contextualize word in sentence: RNN unidirection issue



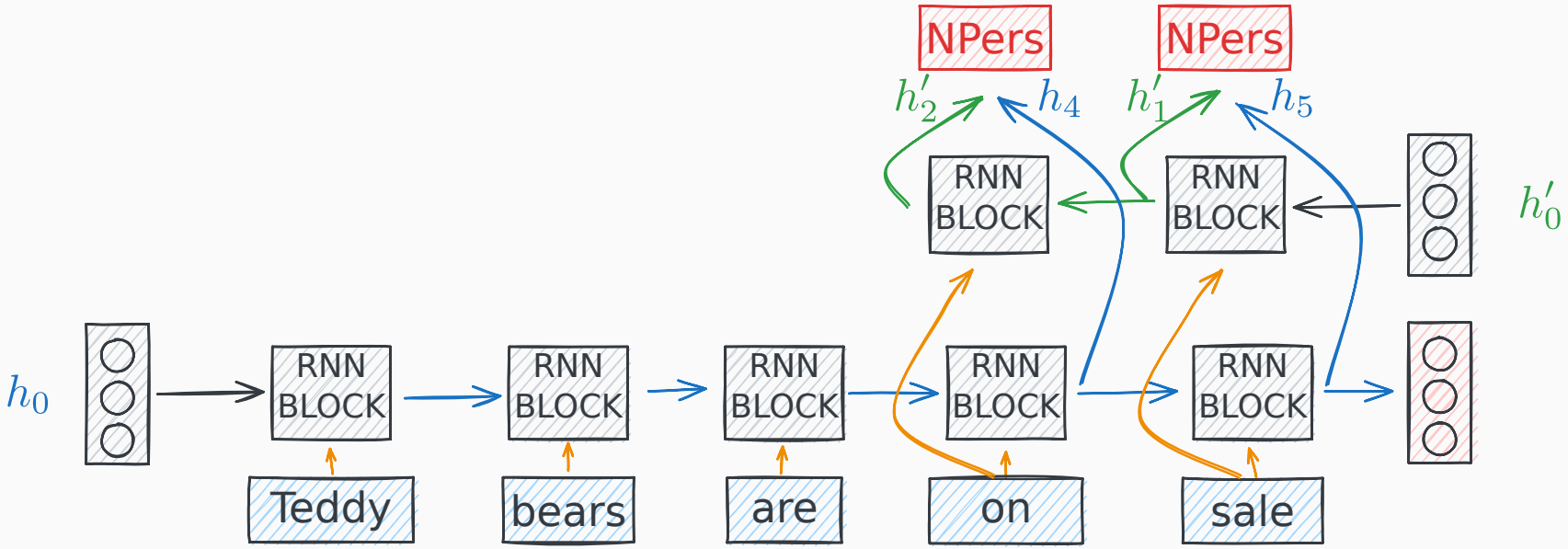
Contextualize word in sentence: RNN unidirection issue



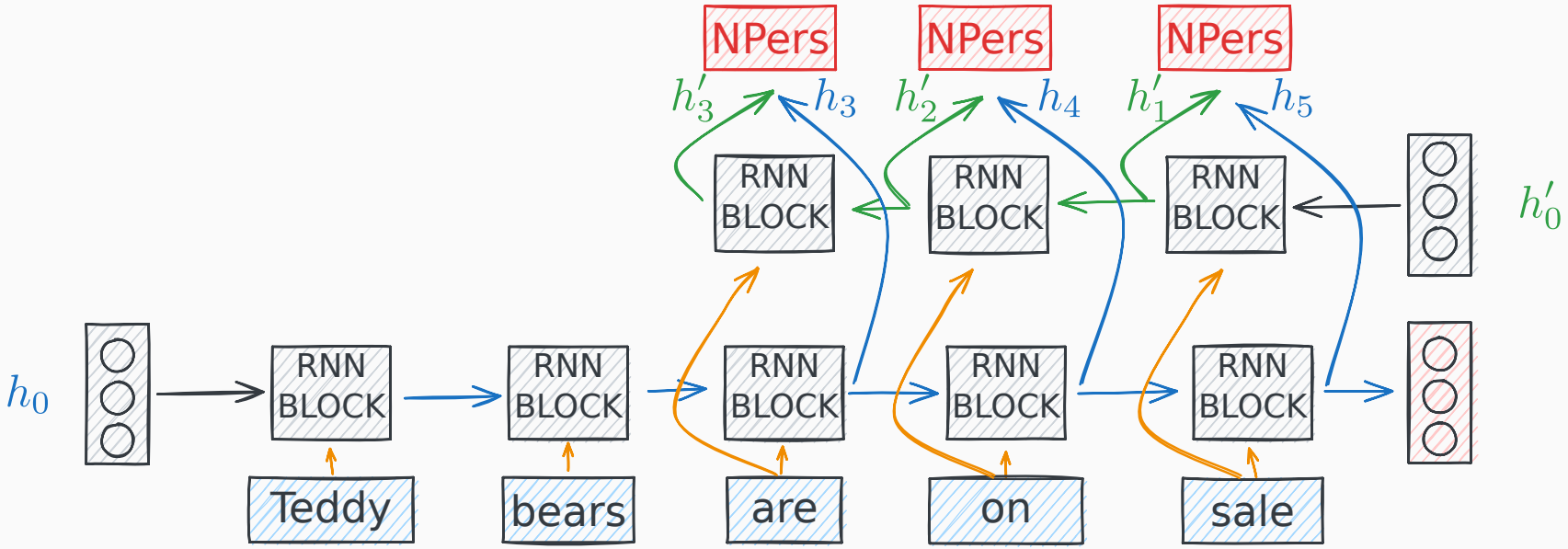
Contextualize word in sentence: RNN unidirection issue



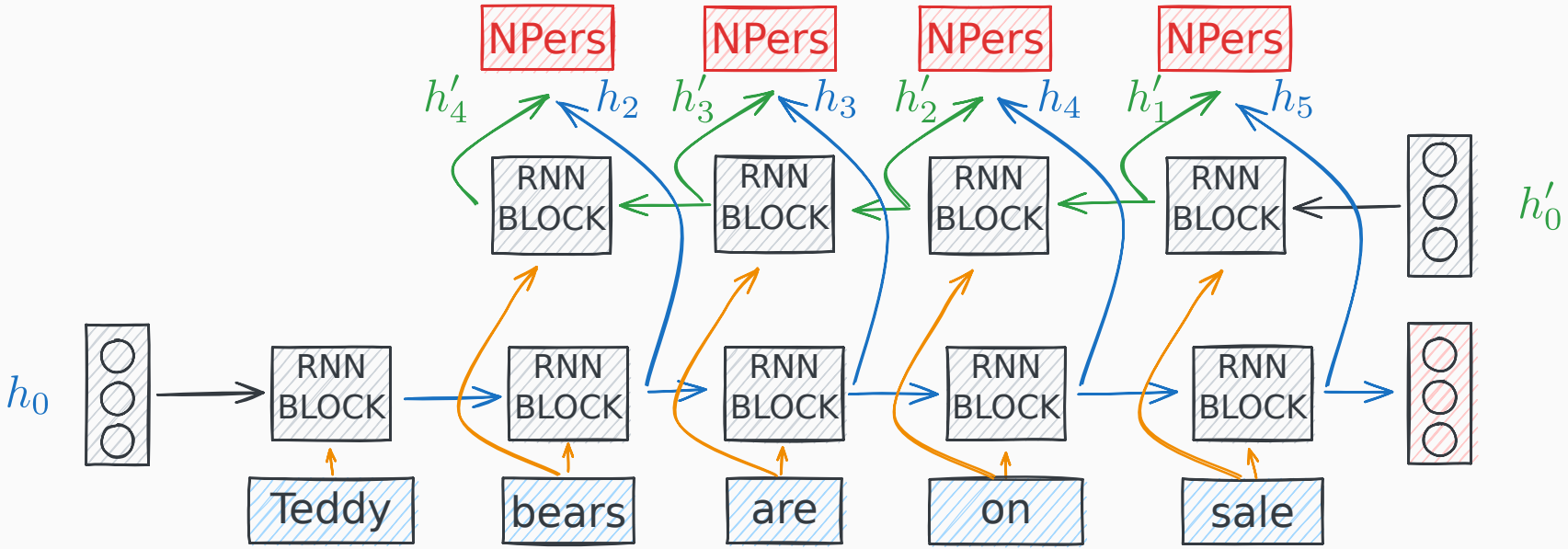
Contextualize word in sentence: RNN unidirection issue



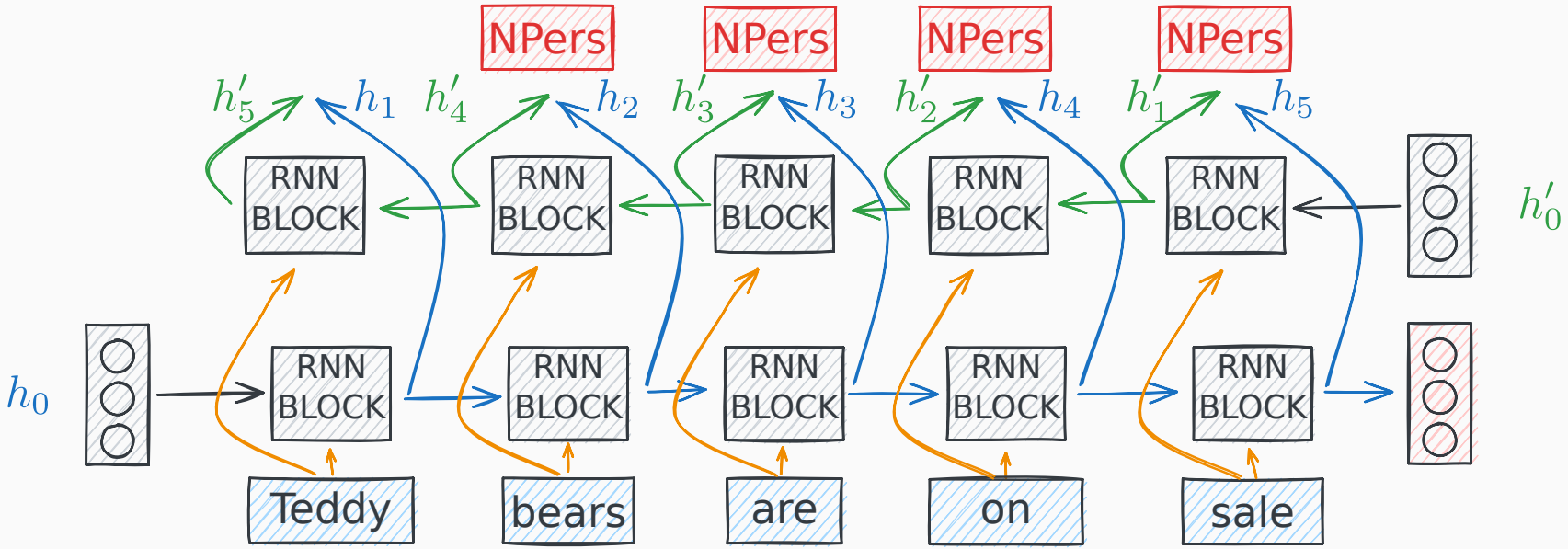
Contextualize word in sentence: RNN unidirection issue



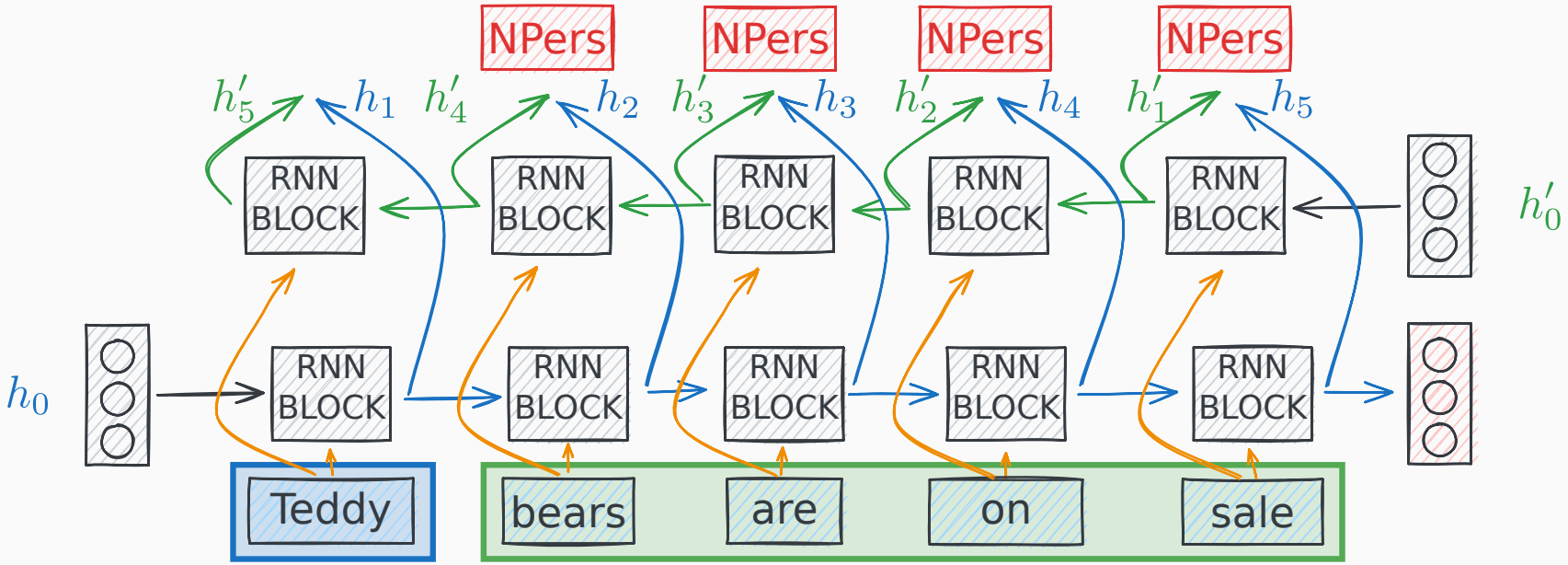
Contextualize word in sentence: RNN unidirection issue



Contextualize word in sentence: RNN unidirection issue



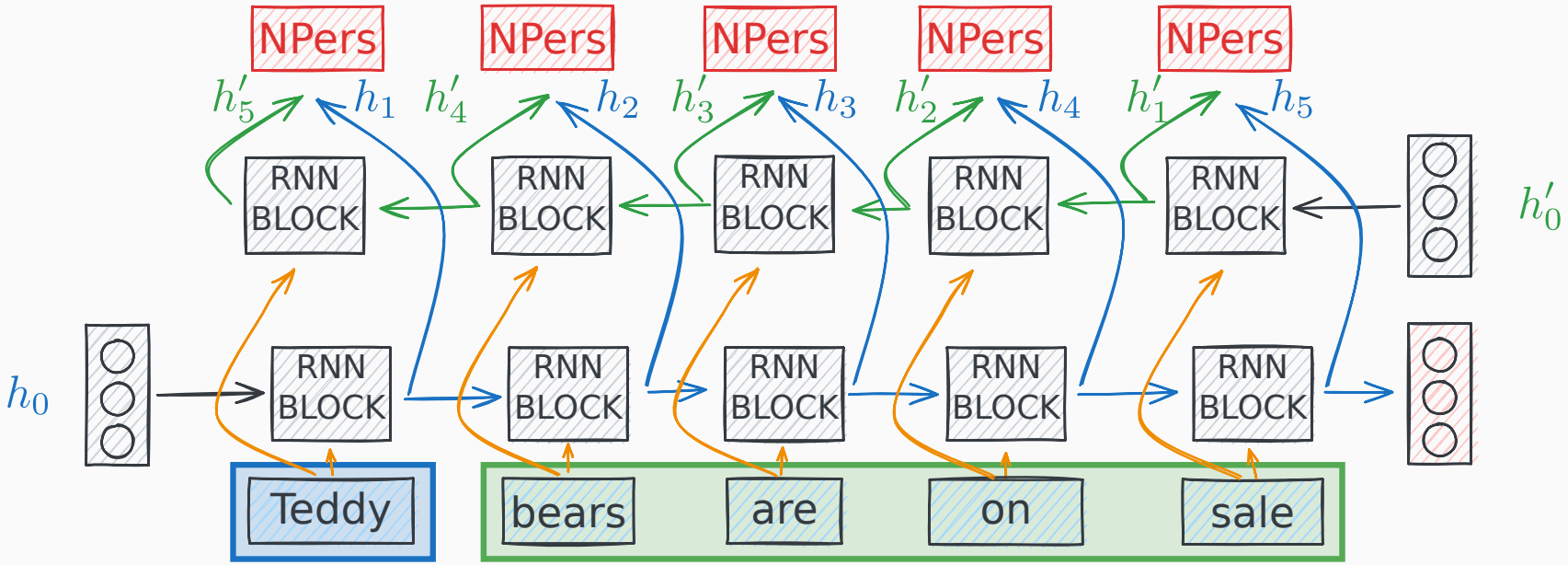
Contextualize word in sentence: RNN unidirection issue



Information of backward state and forward state

$\rightarrow$ Information at each state of the whole sequence

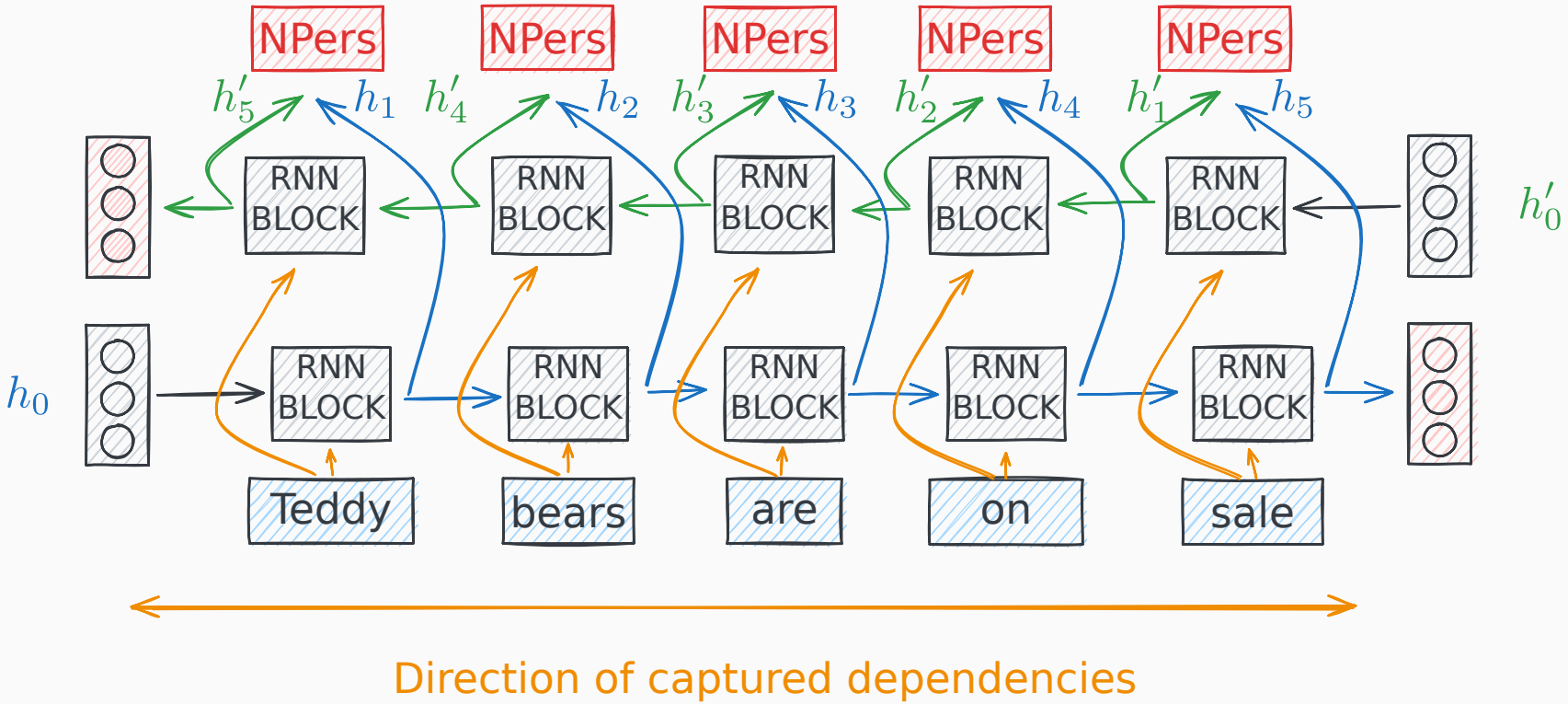
Contextualize word in sentence: RNN unidirection issue



Information of backward state and forward state

$\rightarrow$ Information at each state of the whole sequence

Contextualize word in sentence: RNN unidirection issue



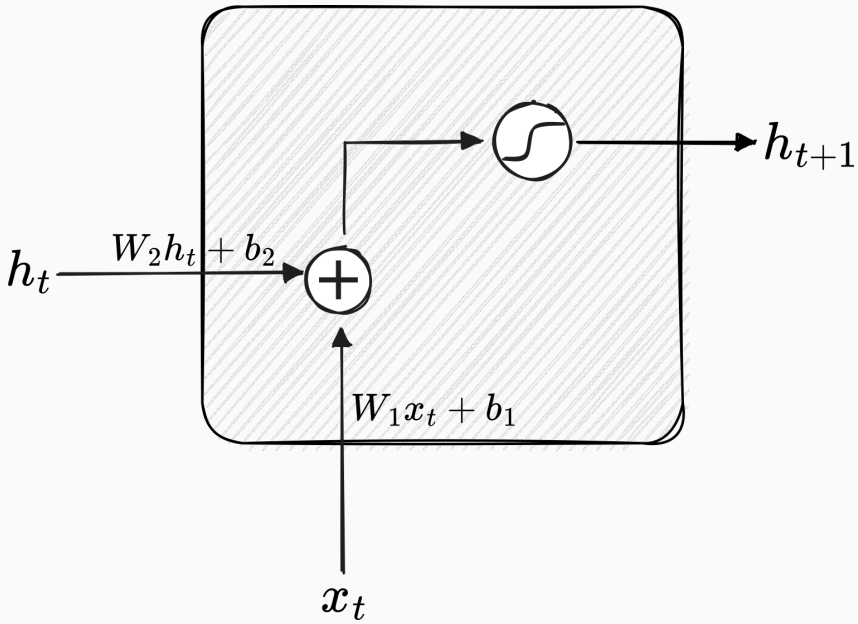
Information of backward state and forward state

$\rightarrow$ Information at each state of the whole sequence

Contextualize word in sentence: Vanilla RNN block

Issue with Vanilla RNN
Difficult to train and mitiggate 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 $\rightarrow$ 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...

Contextualize word in sentence: Reccurent Neural Networks

Vanilla RNN:

$\rightarrow$ Only for short sequence

Contextualize word in sentence: Reccurent Neural Networks

Vanilla RNN:

$\rightarrow$ Only for short sequence

$\rightarrow$ Change the architecture of block?

Contextualize word in sentence: Reccurent 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

Contextualize word in sentence: Reccurent Neural Networks

Vanilla RNN:

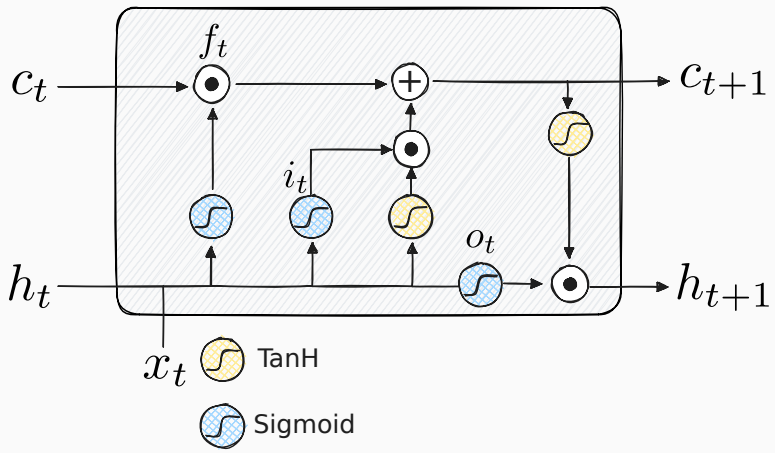
→ Only for short sequence

→ 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: Reccurent Neural Networks

Vanilla RNN:

→ Only for short sequence

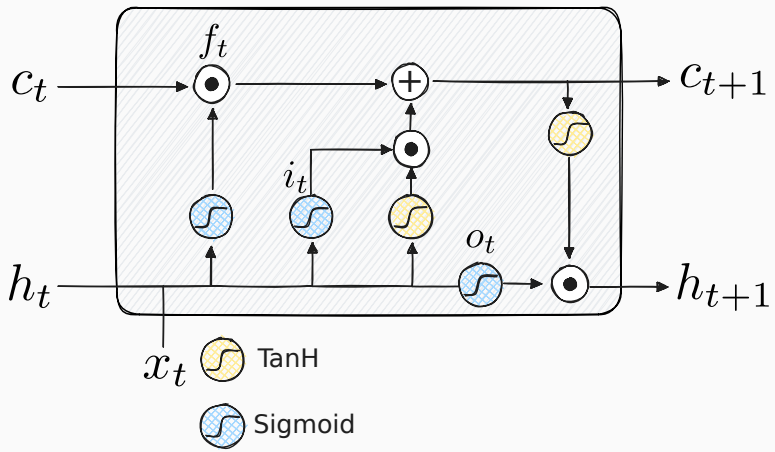
→ Change the architecture of block?

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**"



$$f_t = \sigma(W_{if} x_t + W_{hf} h + b_f)$$
$$i_t = \sigma(W_{ii} x_t + W_{hi} h + b_i)$$
$$g_t = \tanh(W_{ig} x_t + W_{hg} h + b_g)$$
$$o_t = \sigma(W_{io} x_t + W_{ho} h + b_o)$$
$$c_{t+1} = f_t \odot c_t + i_t \odot g_t$$
$$h_{t+1} = o_t \odot \tanh(c_{t+1})$$