

**INF721**

2024/2

**UFV**

# Deep Learning

## L2: Machine Learning

# Logistics

## Announcements

- ▶ We have a **google spaces** for the course
- ▶ Lecture 1 is already available on the course webpage

## Last Lecture

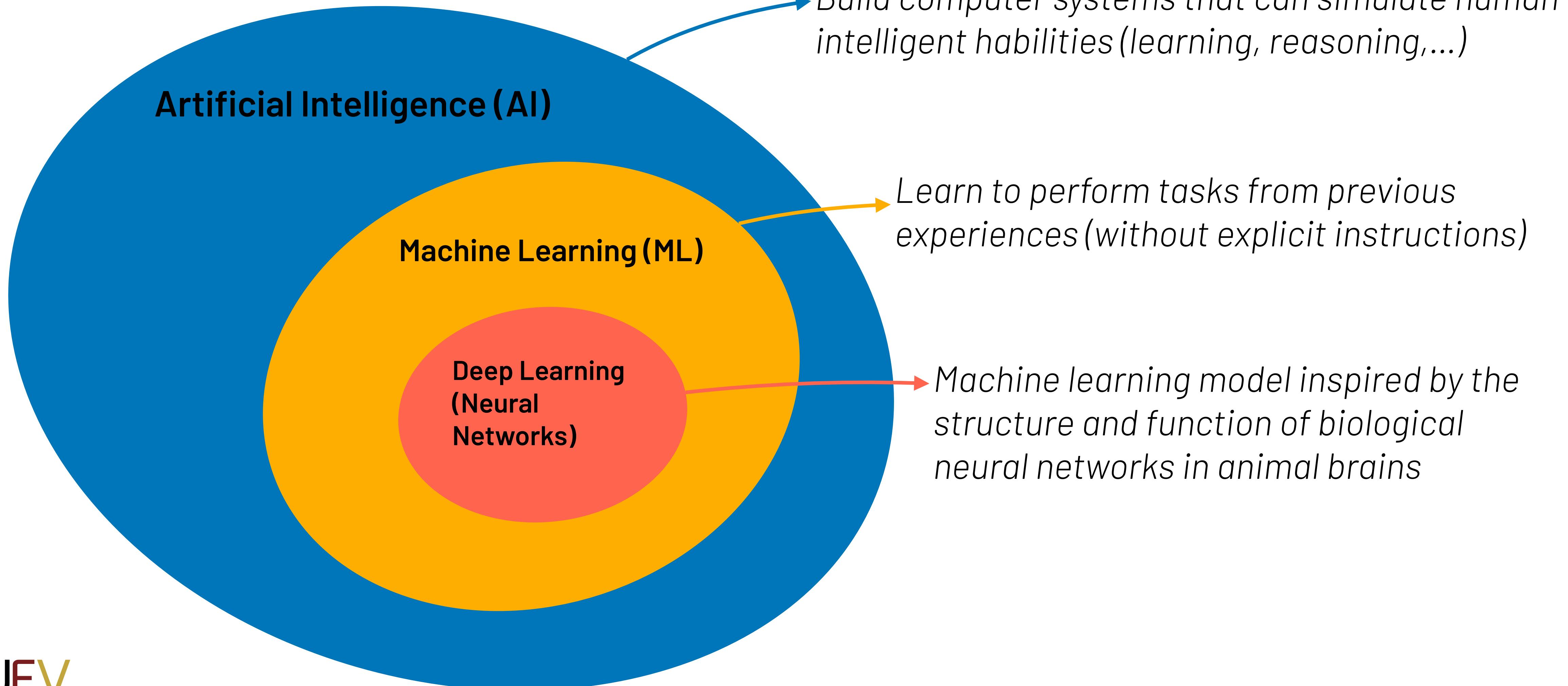
- ▶ Motivation
- ▶ Course syllabus

# Lecture outline

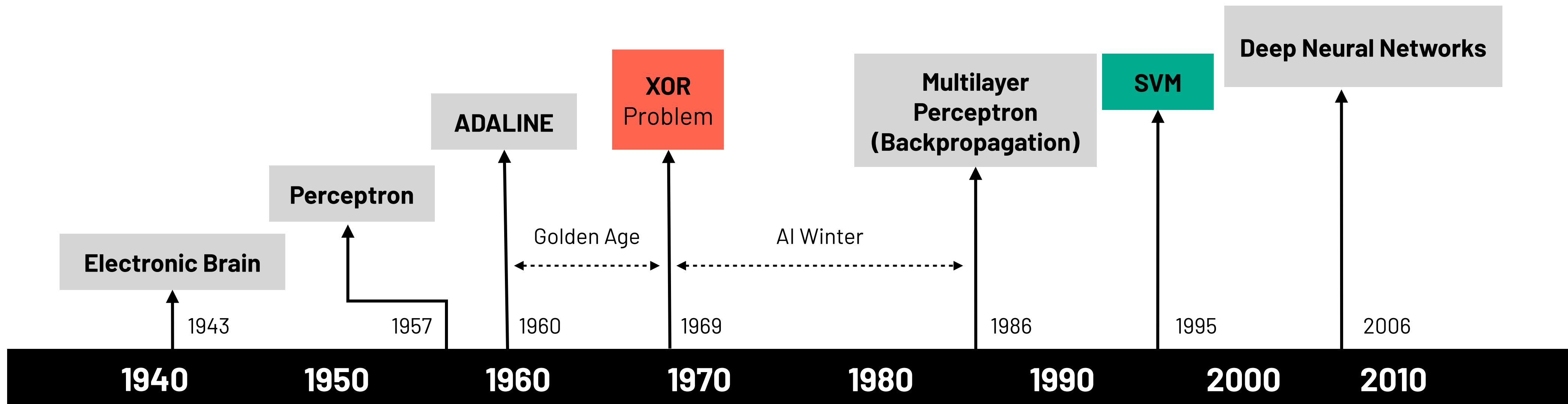
- ▶ Machine Learning
  - ▶ Brief History
  - ▶ Fomulation
  - ▶ Types of problems
- ▶ Supervised Learning Algorithms
  - ▶ Hypothesis space
  - ▶ Loss function
  - ▶ Evaluating Performance

# Machine Learning

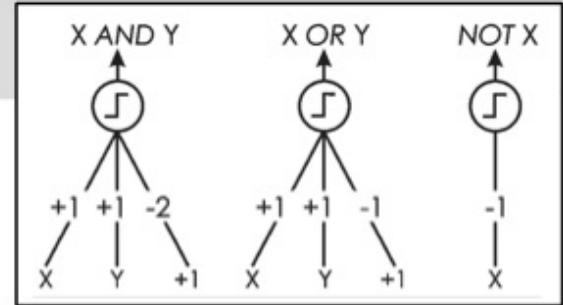
# Machine Learning



# Machine Learning



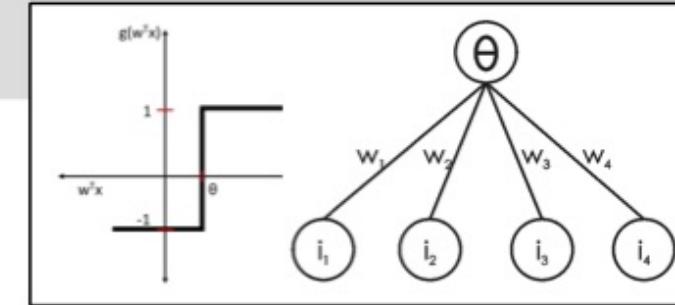
S. McCulloch - W. Pitts



- Adjustable Weights
- Weights are not Learned



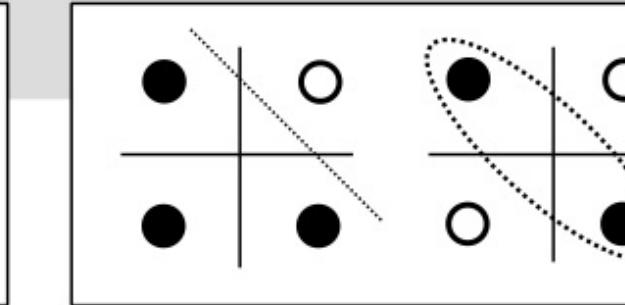
F. Rosenblatt



- Learnable Weights and Threshold



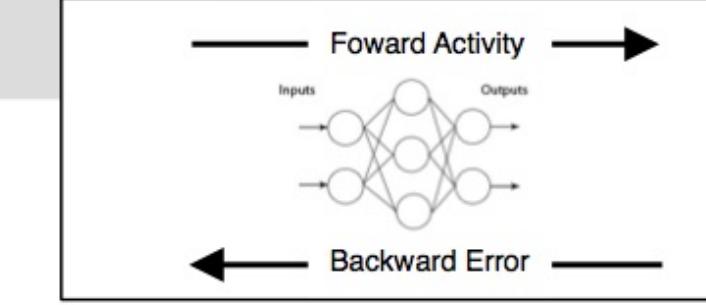
B. Widrow - M. Hoff



- XOR Problem



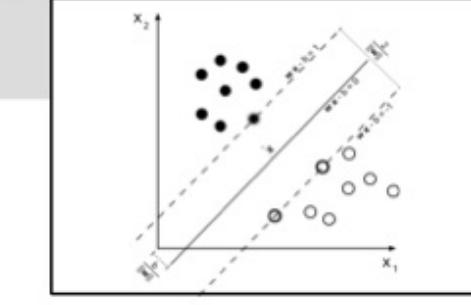
M. Minsky - S. Papert



- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting



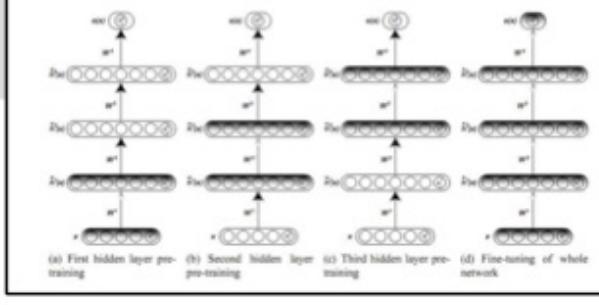
D. Rumelhart - G. Hinton - R. Williams



- Limitations of learning prior knowledge
- Kernel function: Human Intervention

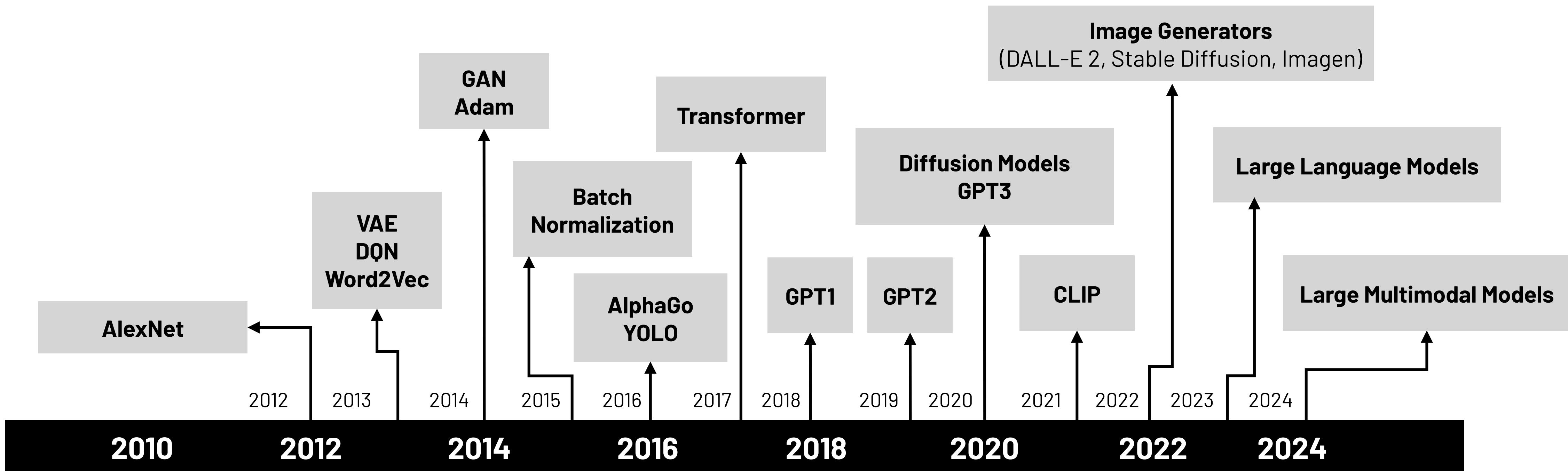


V. Vapnik - C. Cortes



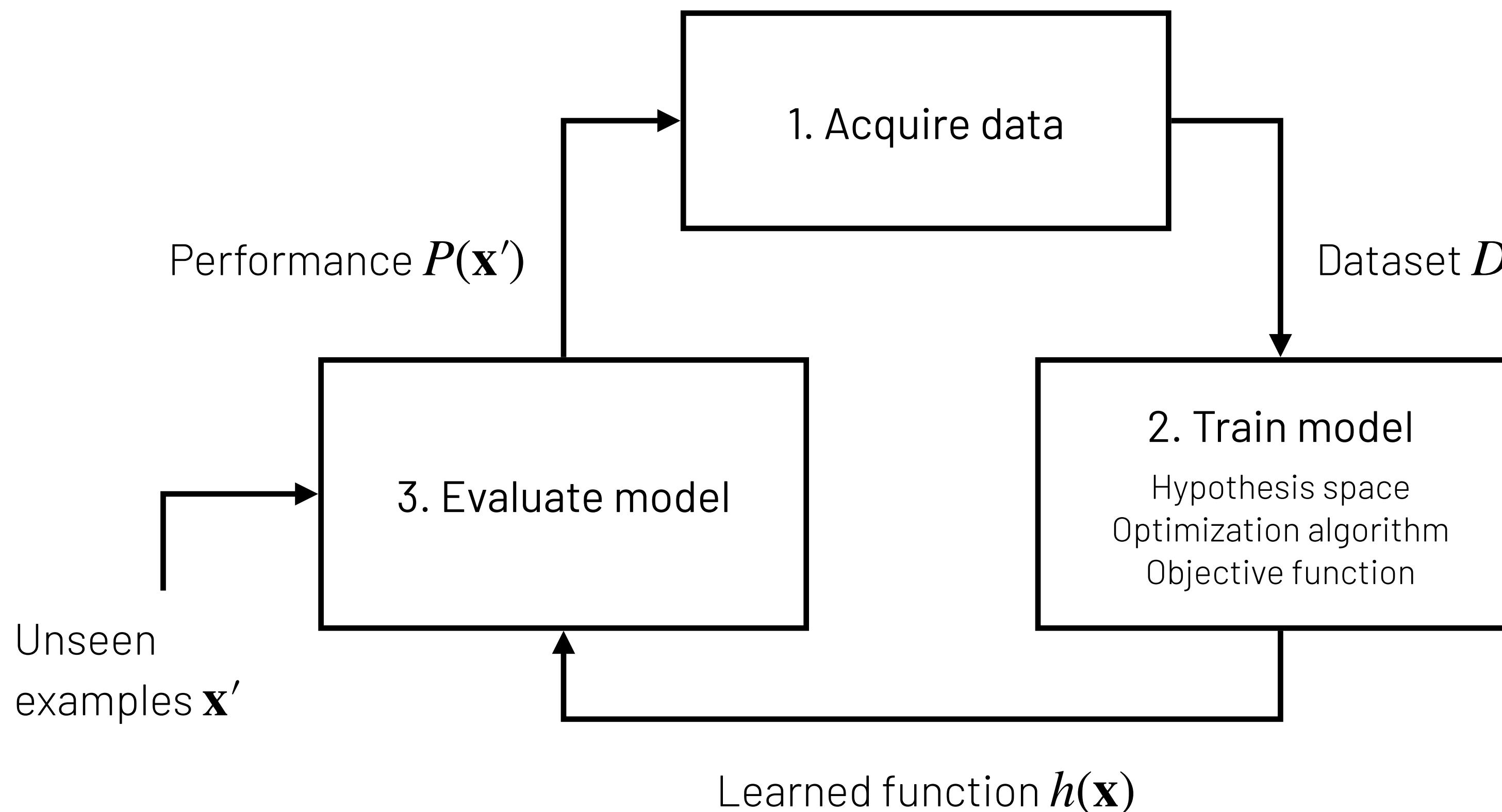
- Hierarchical feature Learning

# Machine Learning



# Machine Learning (ML)

Given a dataset  $D$  and a performance metric  $P$ , we want to learn a function  $h(\mathbf{x})$  (i.e., model) that maximizes  $P(\mathbf{x}')$  on unseen examples  $\mathbf{x}'$



The type of learning is defined by the type of experience given to the model:

Labelled Data

- ▶ **Supervised Learning**

Unlabelled Data

- ▶ **Unsupervised Learning**

Rewards from the environment

- ▶ **Reinforcement Learning**

# Supervised Learning

In **supervised learning** problems, we have a dataset  $D$  of tuples  $(\mathbf{x}^{(i)}, y^{(i)})$  and the goal is to predict the labels  $y^{(i)}$  from the feature vectors  $\mathbf{x}^{(i)}$ , minimizing prediction error on unseen examples  $\mathbf{x}'$ .

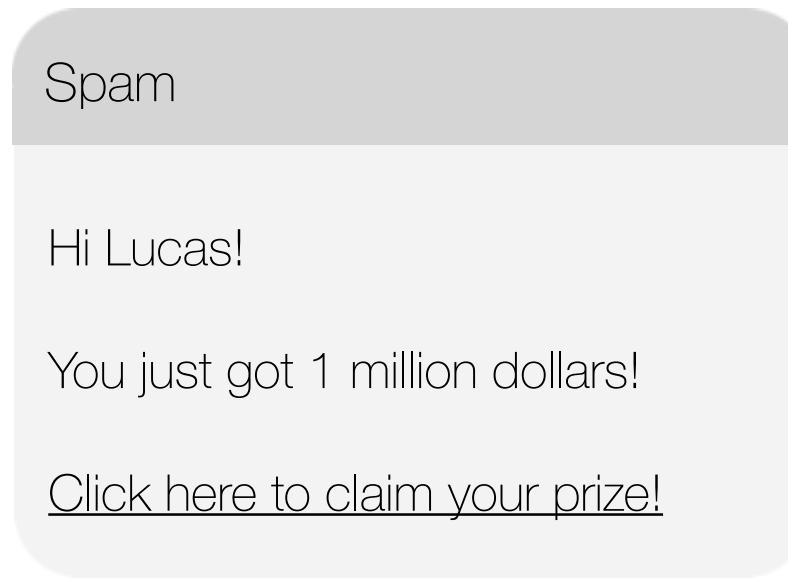
Formally:

$$D = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\} \subseteq \mathbb{R}^d \times C, \text{ where:}$$

- ▶  $m$  is the number of examples in the dataset
- ▶  $\mathbf{x}^{(i)}$  is the feature vector of the  $i^{th}$  example
- ▶  $y^{(i)}$  is the label (or class) of the  $i^{th}$  example
- ▶  $\mathbb{R}^d$  is  $d$ -dimensional feature space
- ▶  $C$  is the label space

# Examples of supervised learning problems

$$D = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\} \subseteq \mathbb{R}^d \times C$$



## Spam Detection (Binary classification)

$\mathbf{x}^{(i)}$ : frequency of the  $i^{th}$  word in a dictionary

$$y^{(i)} \in C = \{0,1\}$$



## Handwritten Digits Classification (Multiclass classification)

$\mathbf{x}^{(i)}$ : color value of the  $i^{th}$  pixel of the flattened image

$$y^{(i)} \in C = \{0,1,2,3,4,5,6,7,8,9\}$$



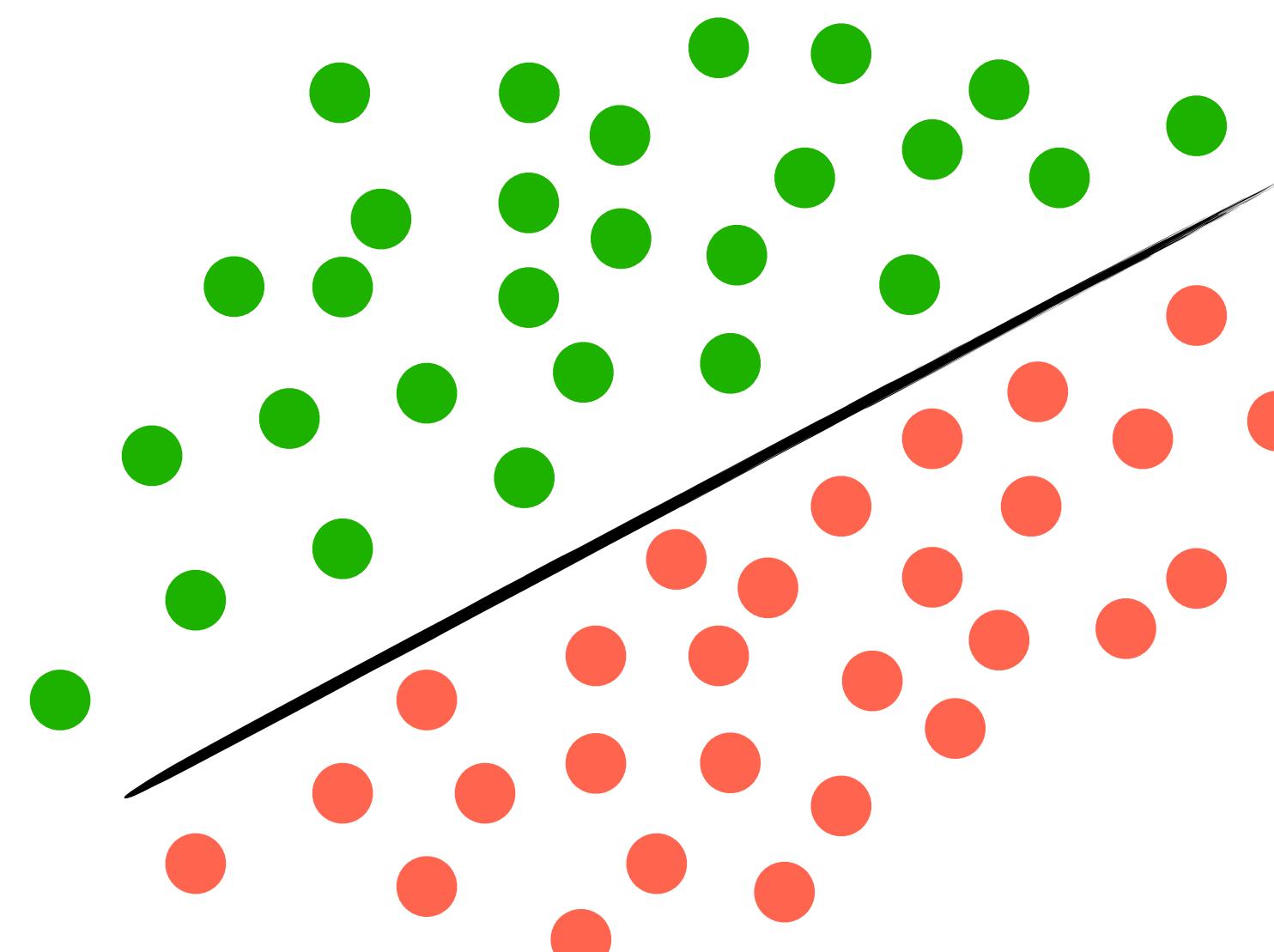
## Houce price prediction (Regression)

$\mathbf{x}_1^{(i)}$ : size ,  $\mathbf{x}_2^{(i)}$ : location, ...,  $\mathbf{x}_n^{(i)}$ : number of bedrooms

$$y^{(i)} \in C = \mathbb{R}$$

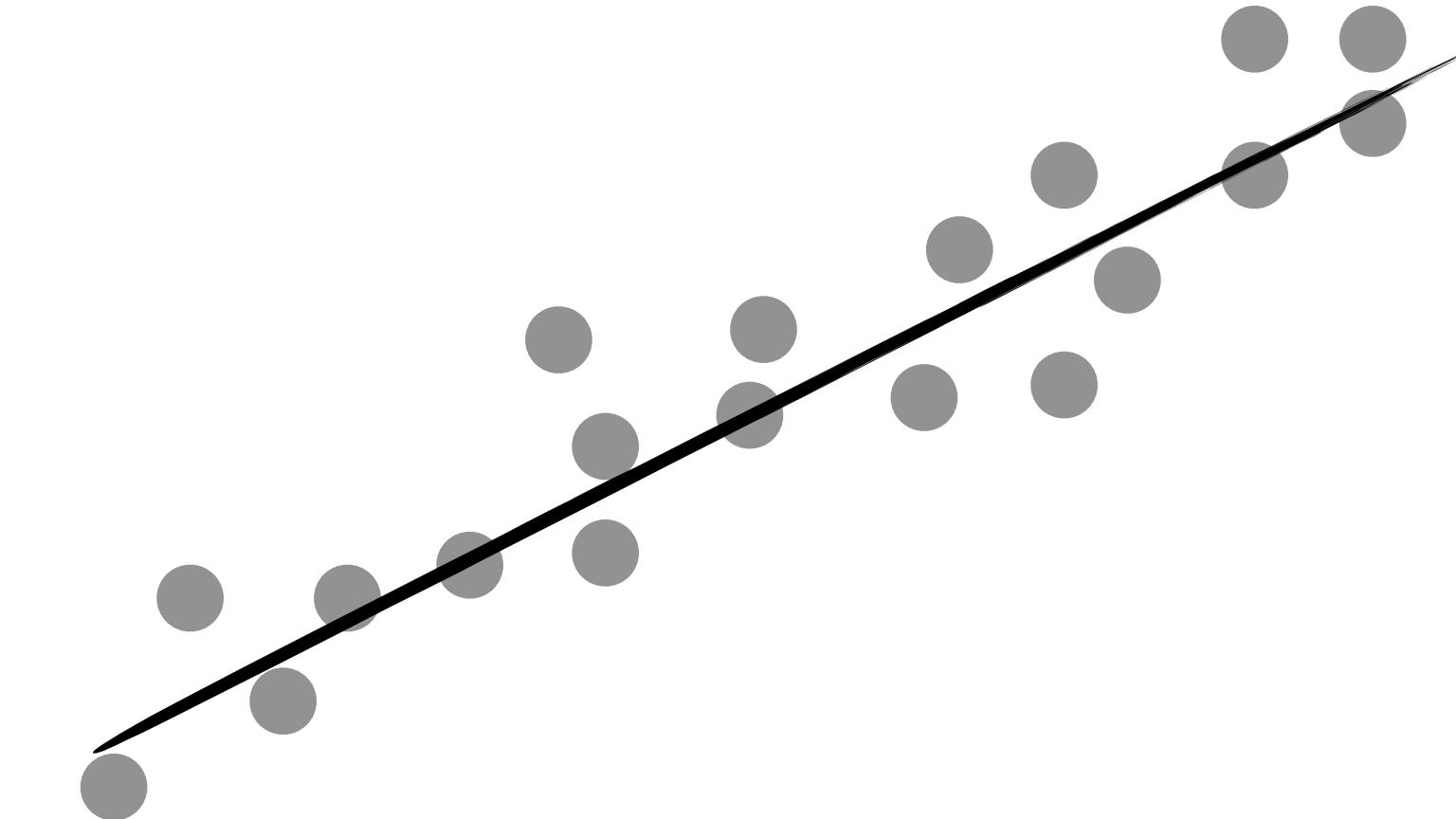
# Visualization of Supervised Learning Problems

**Classification**



Find a function (e.g., linear) that splits the data points by their classes

**Regression**



Find a function (e.g., linear) that fits the data points

# Unsupervised Learning

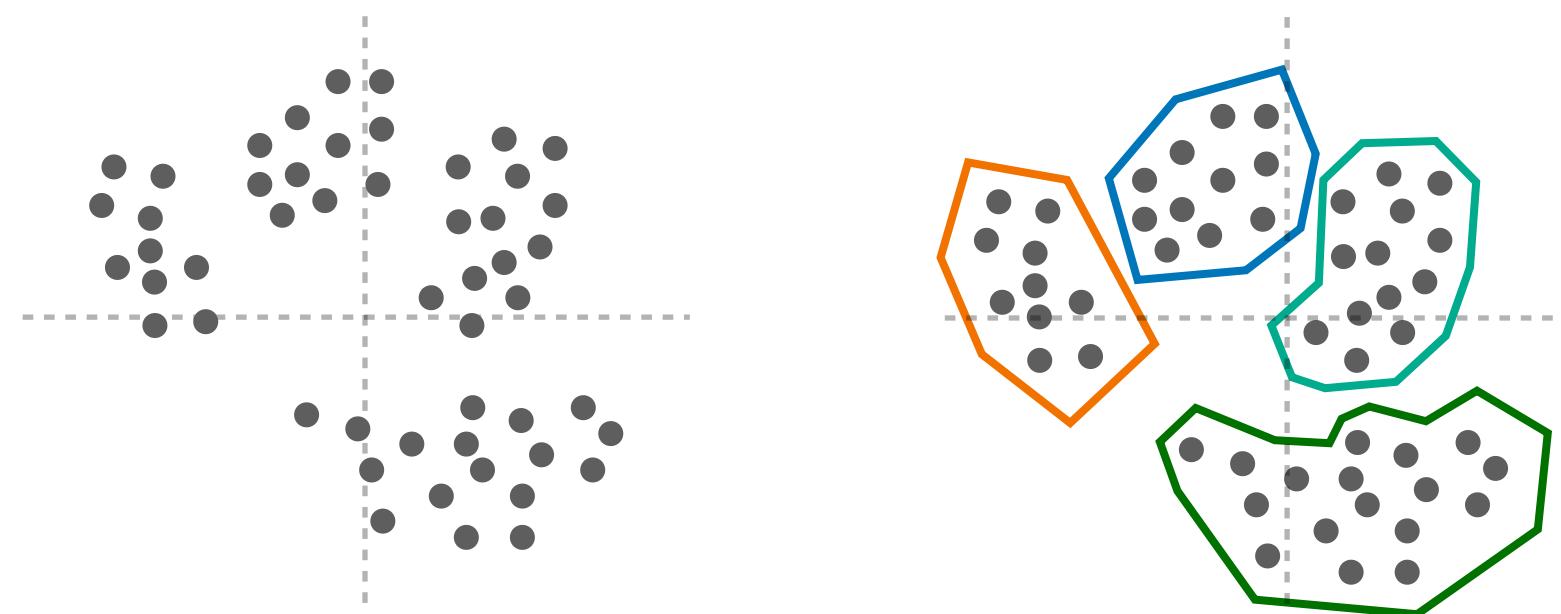
In **unsupervised learning**, the examples in the dataset do not have labels  $y^{(i)}$  and the goal is to discover patterns and relationships in the data.

Formally:

$$D = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}\} \subseteq \mathbb{R}^d, \text{ where:}$$

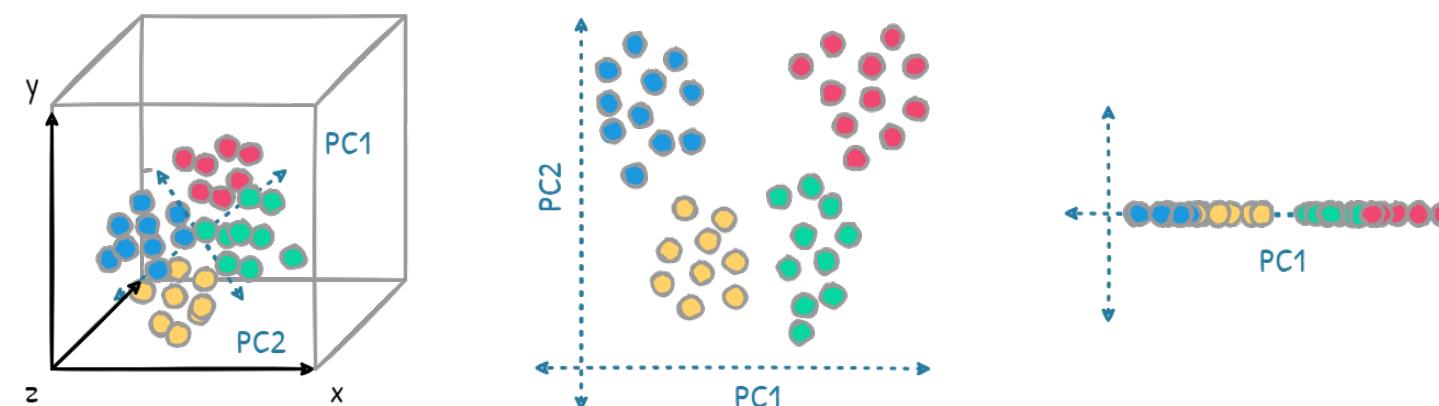
- ▶  $\mathbf{x}^{(i)}$  is the feature vector of the  $i^{th}$  example
- ▶  $\mathbb{R}^d$  is  $d$ -dimensional feature space

# Examples of unsupervised learning problems



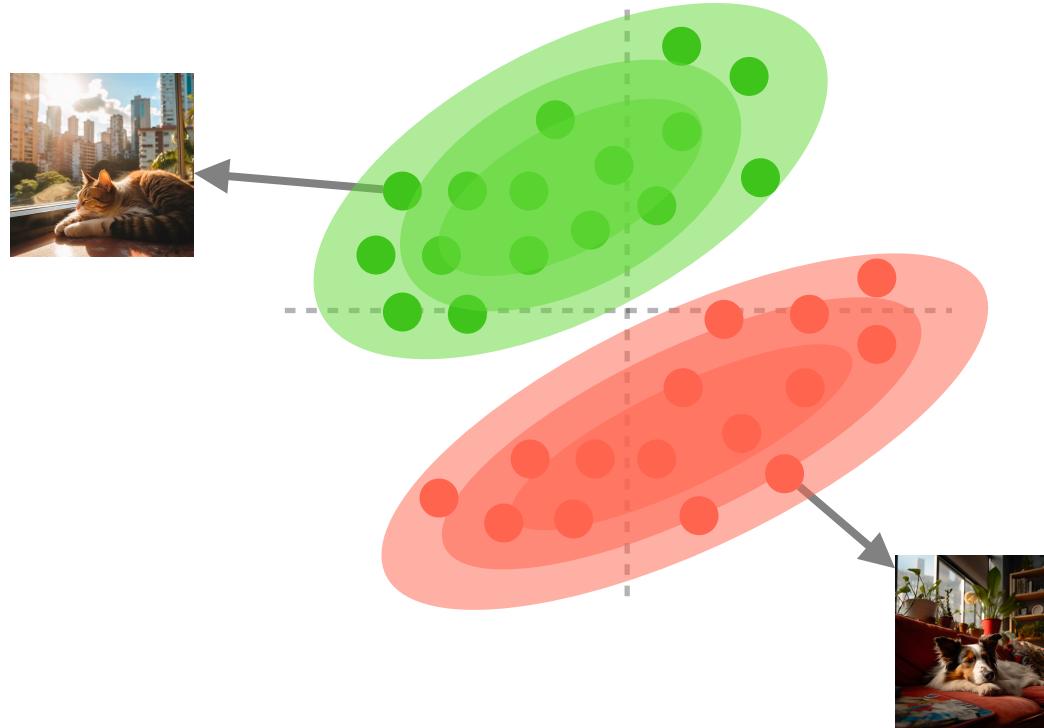
## Clustering

Discover the inherent groupings in the data



## Dimensionality Reduction

Representing a given dataset using a lower number of features



## Generative AI

Generate new examples similar to the dataset

# Reinforcement Learning

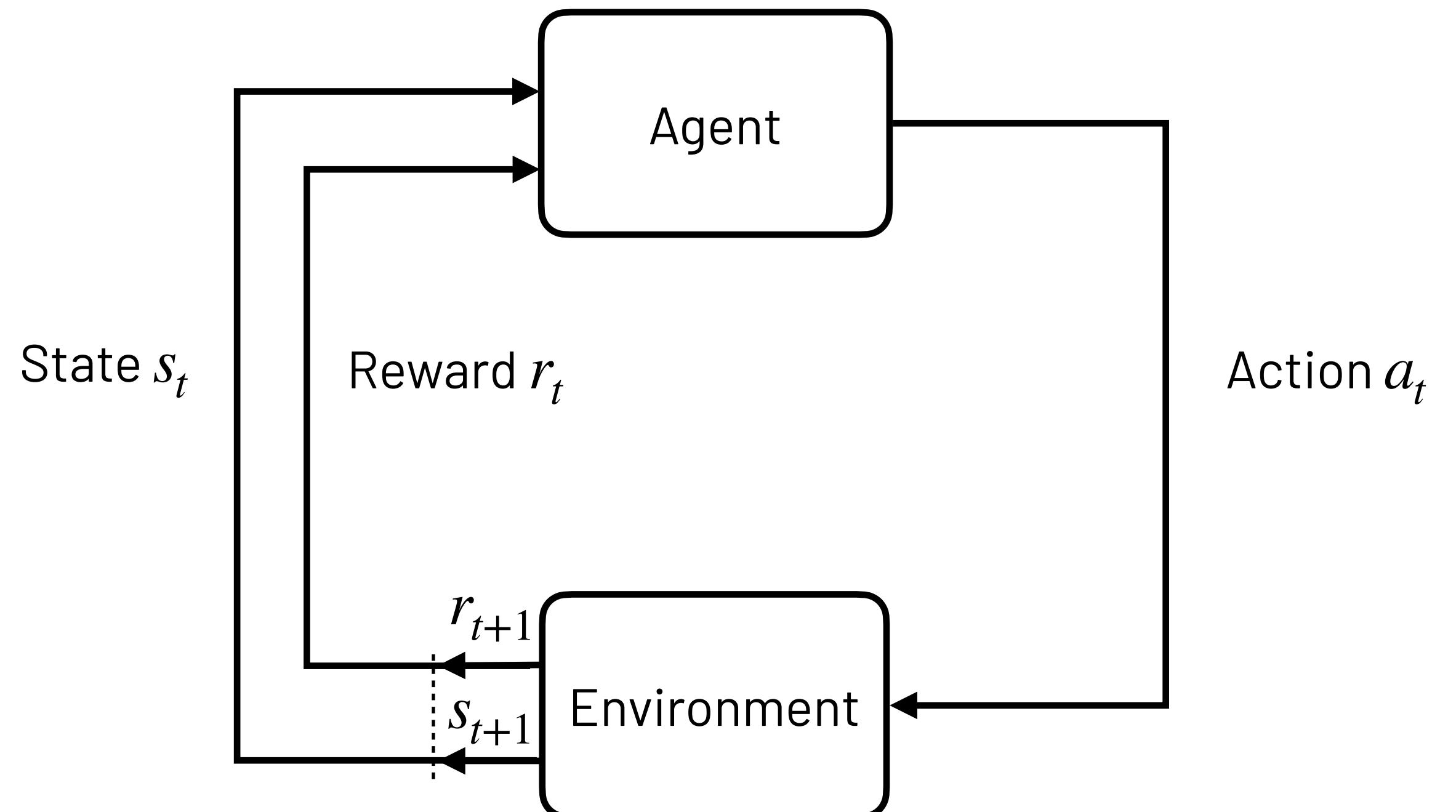
In **reinforcement learning**, the goal is to learn a function  $\pi(s) = a$  to predict an action  $a$  from a state  $s$ , maximizing the expected sum of rewards received by the environment.

## Agent

- ▶ Receives a state  $s_t$  at time  $t$
- ▶ Performs an action  $a_t$  at time  $t$

## Environment

- ▶ Returns a reward value  $r_{t+1}$  and;
- ▶ The next state  $s_{t+1}$



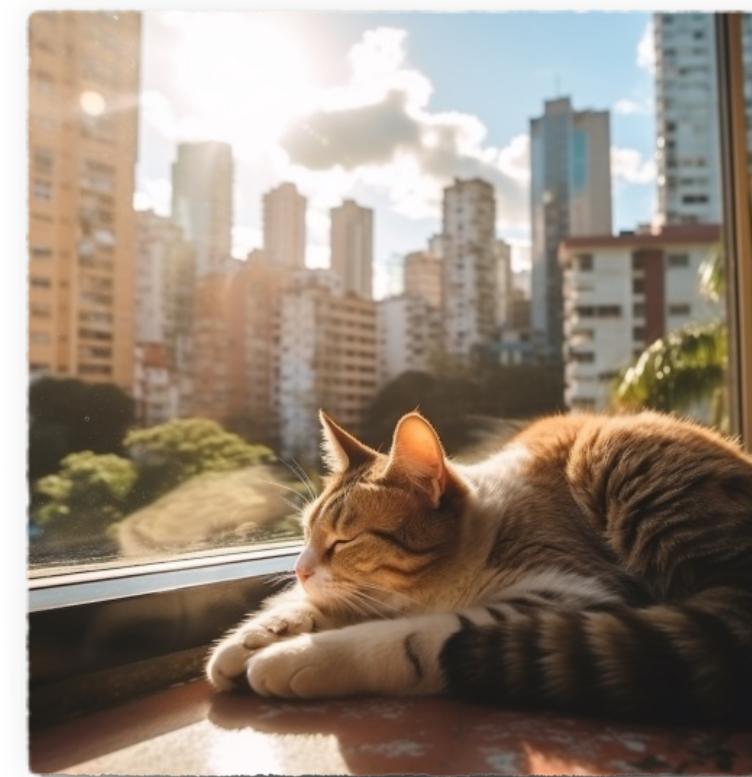
# Data Types

## Structured Data (tabular)

Size (m2)	Location	N. of bedrooms	...	Price
72	Centro	2		
54	Centro	1		
...	...	...	...	...
72	Clélia	3		

Age	State	Ad Id	...	Click
72	MG	93242		1
54	SP	93287		0
...	...	...	...	...
72	RJ	71244		1

## Unstructured Data



Spam

Hi Lucas!

You just got 1 million dollars!

[Click here to claim your prize!](#)

Text

Images

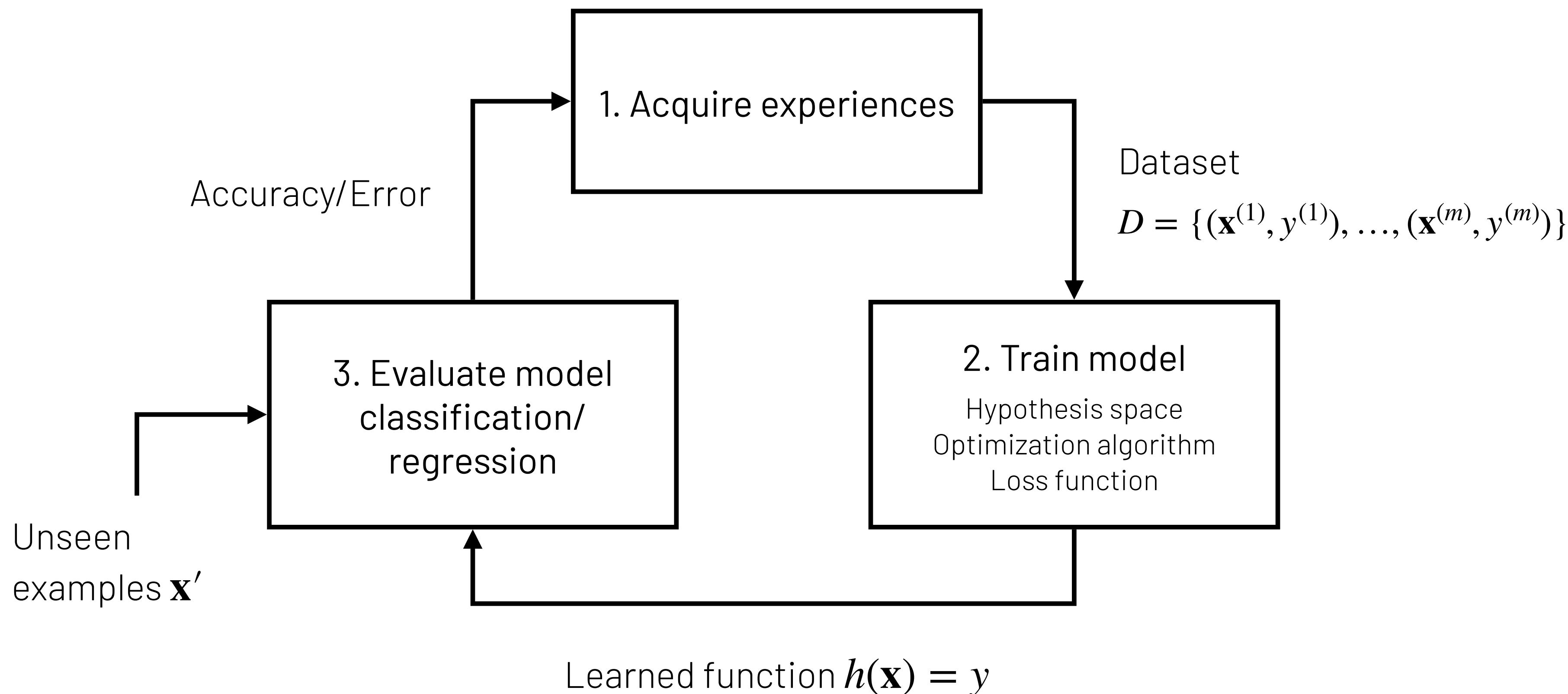


Audio

# Supervised Learning Algorithms

# Supervised Learning

Learn a function  $h(\mathbf{x}) = y$  from a dataset  $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}$  to predict the labels  $y^{(i)}$  from the feature vectors  $\mathbf{x}^{(i)}$ , minimizing error on unseen examples  $\mathbf{x}'$



# Training a model

**Training a model** means finding a function  $h \in H$  in a space of functions  $H$

To do that, a supervised learning algorithm needs to:

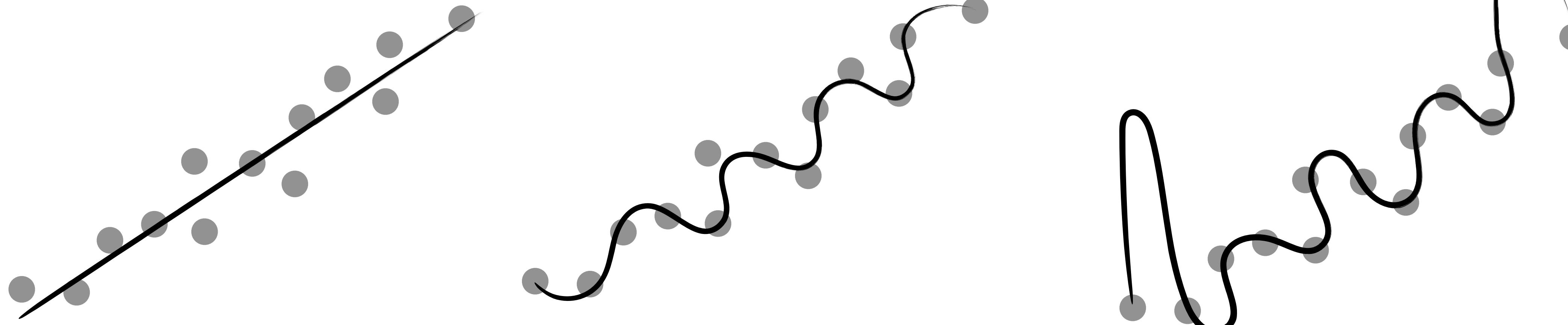
1. Define an specific espace of functions, called **hypothesis space  $H$** ;
2. Find the best function  $h \in H$ , i.e., the function that minimizes the prediction error according to some **loss function  $L$** .

**In neural networks (and many other ML algorithms), this step is formalized as an optimization algorithm!**

# Hypothesis Space

The **hypothesis space**  $H$  defines the set of functions an ML algorithm can find.

Examples:



Linear

$$h(x) = w_1x + w_0$$

Sinusoidal

$$h(x) = w_1x + \sin(w_0x)$$

Degree-12 Polynomial

$$h(x) = \sum_{i=0}^{12} w_i x_i$$

# Loss function

The **loss function**  $L$  evaluates a function  $h \in H$  with the dataset  $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}$ :

- ▶ Measures how far the predictions  $h(\mathbf{x}^{(i)})$  are from labels  $y_i$  of examples  $(\mathbf{x}^{(i)}, y^{(i)}) \in D$ ;
- ▶ Loss values  $L(h)$  are always positive;
- ▶ The lower the  $L(h)$ , the better the function  $h$  – a function with loss  $L(h) = 0$  (zero) correctly predicts the labels of all examples in  $D$ ;
- ▶ Typically, loss functions are normalized to be independent from the size  $m$  of the dataset  $D$ .

Examples:

## Zero-One Loss

$$L(h) = \frac{1}{m} \sum_{i=1}^n \delta_{h(\mathbf{x}^{(i)}) \neq y_i} \text{ where } \delta_{h(\mathbf{x}^{(i)}) \neq y_i} = \begin{cases} 1, & \text{if } h(x_i) \neq y_i \\ 0, & \text{otherwise} \end{cases}$$

## Mean Squared Error (MSE)

$$L(h) = \frac{1}{m} \sum_{i=1}^n (h(x_i) - y_i)^2$$

## Mean Absolute Error (MAE)

$$L(h) = \frac{1}{m} \sum_{i=1}^n |h(x_i) - y_i|$$

# Evaluating Model's Performance

Given a dataset  $D$ , a hypothesis space  $H$  and a loss function  $L$ , we want to find the function  $h \in H$  that:

$$h = \operatorname{argmin}_{h \in H} L(h)$$

If we find a function  $h \in H$  with low loss in  $D$ , how do we know that it also has low loss in new examples  $(x', y') \notin D$ ?

Consider the following “memorizer” function:

$$h(x) = \begin{cases} y^{(i)}, & \text{if } \exists (x^{(i)}, y^{(i)}) \in D, \text{ such that, } x = x^{(i)} \\ 0, & \text{otherwise} \end{cases}$$

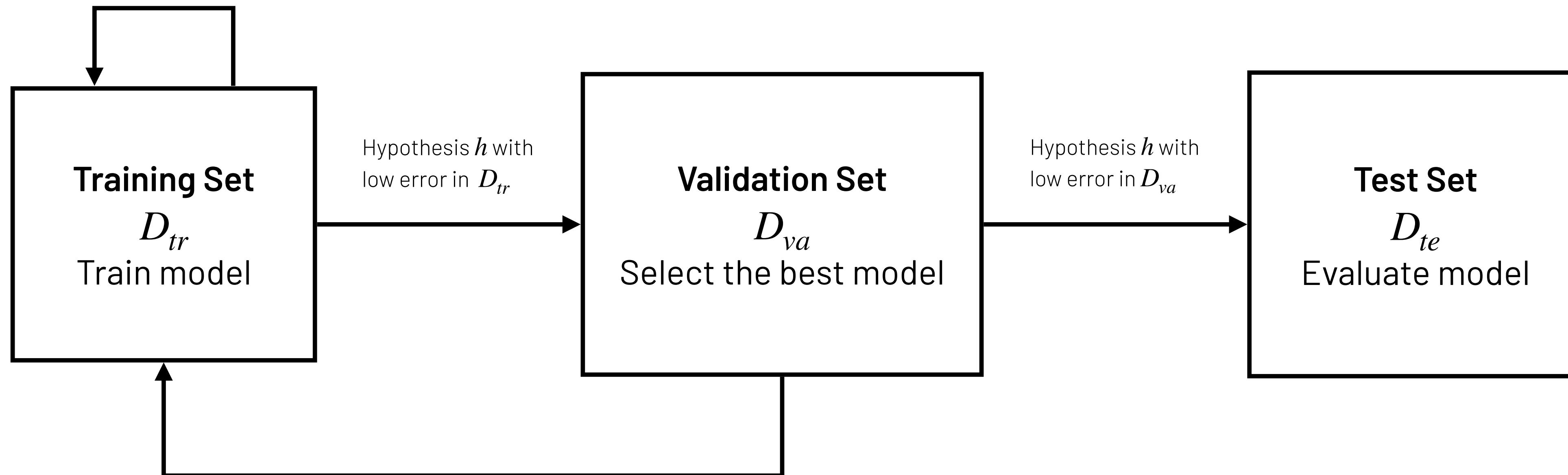
- ▶ Loss 0 for examples in  $D$ ;
- ▶ Very high loss for unseen examples!

**This problem is called overfit!**

# Evaluating Model's Performance

To evaluate a model on unseen examples, we typically divide the dataset  $D$  in 3 disjoint subsets:  
 $D_{tr}$ ,  $D_{va}$  e  $D_{te}$

Hypothesis  $h$  with high error in  $D_{tr}$   $\longrightarrow$  **Underfit!**



Hypothesis  $h$  with high error in  $D_{va}$   $\longrightarrow$  **Overfit!**

# Supervised Learning Algorithms

There are many supervised learning algorithms and each one assumes a different hypothesis space  $H$ :

- ▶ Linear Regression
- ▶ Logistic Regression
- ▶ Decision Trees
- ▶ K-Nearest Neighbors (KNN)
- ▶ Naive Bayes
- ▶ Support Vector Machines (SVMs)
- ▶ Neural Networks

# Next Lecture

## L3: Linear Models

Discuss simple models that are the basis to how neural networks solve supervised learning problems:

- ▶ Linear Regression
- ▶ Perceptron
- ▶ Logistic Regression
- ▶ Gradient Descent