



# **01 - Machine Learning**

Daniele Gamba

2022/2023

# What is Machine Learning

Let's start with some definitions

[Machine learning is the] field of study that gives computers the **ability to learn** without being explicitly programmed.

Arthur Samuel, 1959

A computer program is said to learn from **experience** E with respect to some **task** T and some performance **measure** P, if its performance on T, as measured by P, improves with experience E.

Tom Mitchell, 1997

## Experience - E

In order to learn an algorithm it must somehow accumulate experience.

In most cases the experience is represented by a set of data called **a dataset**.

Normally, we represent the inputs of our algorithm as X, and the outputs as Y.

The dataset is the set of both, each input associates an output (if present).

# Datasets - Examples

	A	B	C	D	E	F
1	sepal_length	sepal_width	petal_length	petal_width	species	
2	5.1	3.5	1.4	0.2	setosa	
3	4.9	3	1.4	0.2	setosa	
4	4.7	3.2	1.3	0.2	setosa	
5	4.6	3.1	1.5	0.2	setosa	
6	5	3.6	1.4	0.2	setosa	
7	5.4	3.9	1.7	0.4	setosa	
8	4.6	3.4	1.4	0.3	setosa	
9	5	3.4	1.5	0.2	setosa	
10	4.4	2.9	1.4	0.2	setosa	
11	4.9	3.1	1.5	0.1	setosa	
12	5.4	3.7	1.5	0.2	setosa	
13	4.8	3.4	1.6	0.2	setosa	
14	4.8	3	1.4	0.1	setosa	
15	4.3	3	1.1	0.1	setosa	
16	5.8	4	1.2	0.2	setosa	
17	5.7	4.4	1.5	0.4	setosa	
18	5.4	3.9	1.3	0.4	setosa	
19	5.1	3.5	1.4	0.3	setosa	
20	5.7	3.8	1.7	0.3	setosa	
21	5.1	3.8	1.5	0.3	setosa	

Iris Dataset

X: length and width of stems and petals)  
Y: Iris species



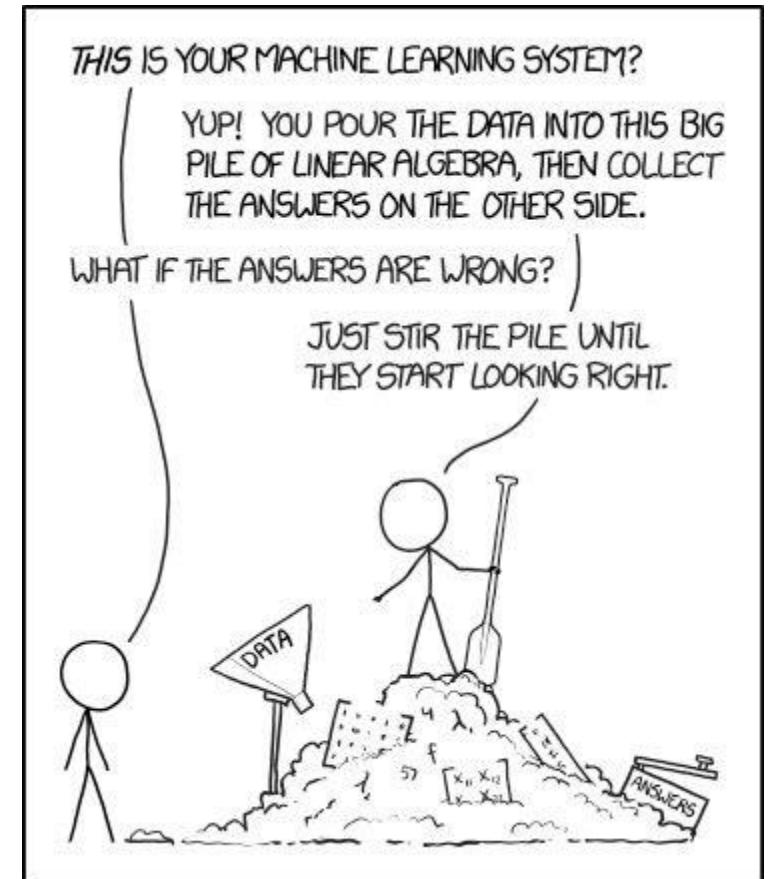
COCO Dataset

X: 200,000 images  
Y: segmentations of 80 different classes

# Datasets

The quality of the dataset is crucial.

It is equally important to have enough data and that this is expressed in a "convenient" way for the algorithms to learn.



## Tasks - T

We can ask algorithms to do many different things

driving a car, checking the quality of a part, optimizing consumption, changing the background of a video, etc.

These objectives are the composition of one or more Tasks that we give to a machine learning algorithm.

There are many tasks and there are always new ones

## Tasks - T

- **Regression** : finding parameters that describe data
- **Classification** : discriminating between different groups
- **Forecasting** : predicting future data
- **Clustering** : Finding different patterns within your data
- **Object detection** : find where an object is located within an image
- **Object segmentation** : finding which pixels belong to the object
- **Anomaly detection** : finding what deviates from normality
- Etc.

## Measure - P

To allow an algorithm to learn we must give it a metric, a measure, which allows it to understand in which direction the optimum we are looking for is.

For this reason, performance measures are defined that allow the algorithm to learn.

Here too there are different types

- Mean Absolute Error
- Mean Squared Error
- Binary Crossentropy
- Mean Average Precision
- ...

## Example – Spam Filter

A spam filter is a good example of a machine learning algorithm.

We have a collection of emails classified as good and others classified as spam.

Our **dataset** will be composed of all the email texts (X) and their good/spam classification (Y).

The algorithm will have to **classify** a new example and put it into one of two groups.

The performance measure is the **accuracy** of the model, i.e. how many predictions it correctly performs out of the total.

## Example – Spam Filter

Now that we have all the requirements of the problem (dataset, task and measurement) all we have to do is solve it.

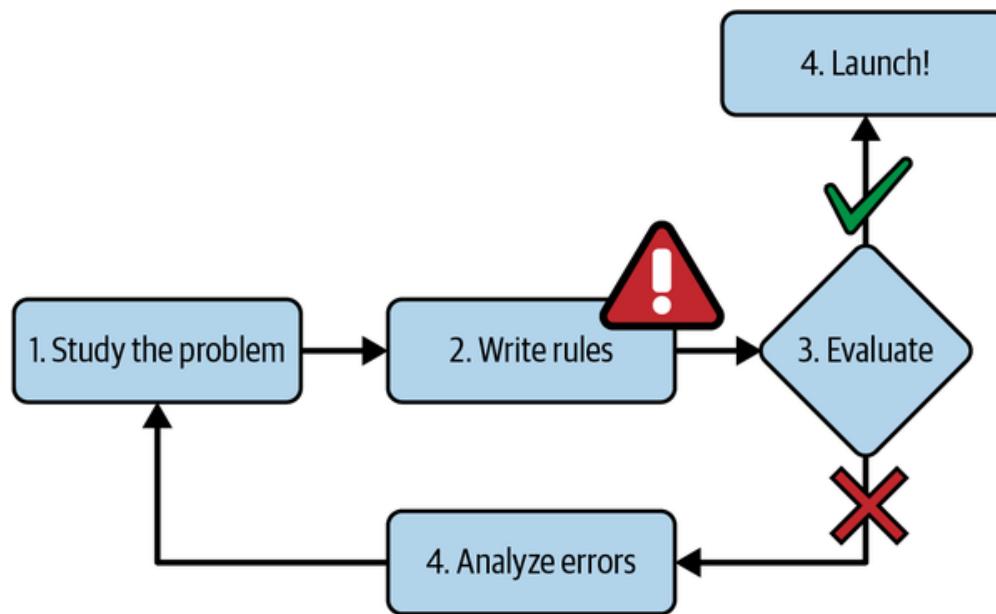
If we approached the problem in a '**traditional**' way we would have to look for some rule that we can codify to discriminate between emails.

An example could be that if there are keywords like "you won", "prince's legacy", "extend appendages", then we put the email in the spam folder, otherwise it is good.

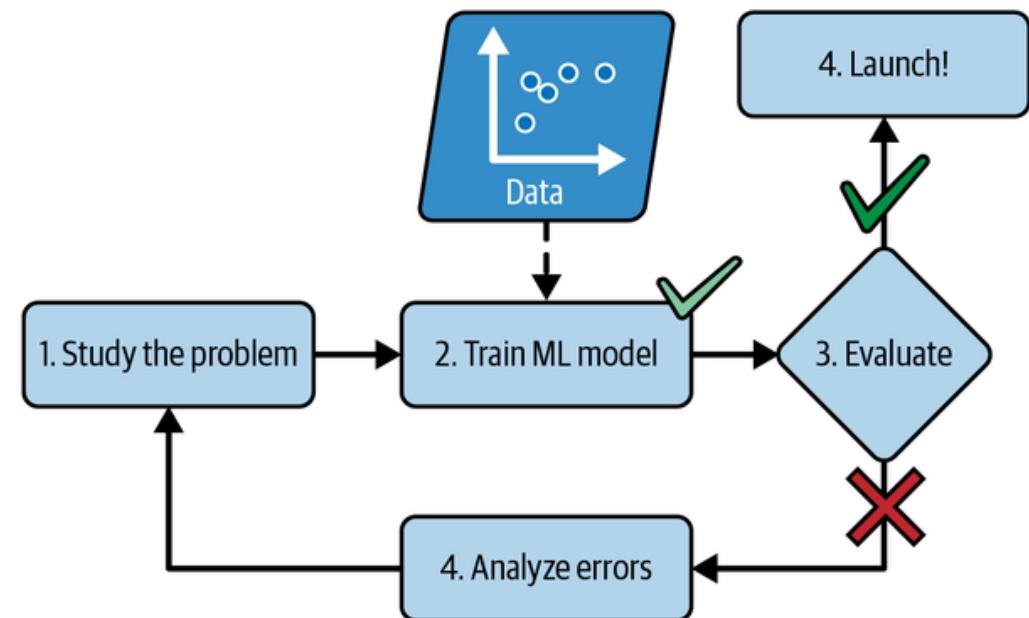
This approach, as you can easily imagine, is extremely onerous and inflexible, it would be very easy for spammers to circumvent the statically defined rules with synonyms.

# Programming vs Machine Learning

## Traditional approach



## ML approach



# Model

As you have seen, Machine Learning allows us to exploit the data available so that a **model** learns to solve the task for us.

The model is in general a simplified representation of the world.

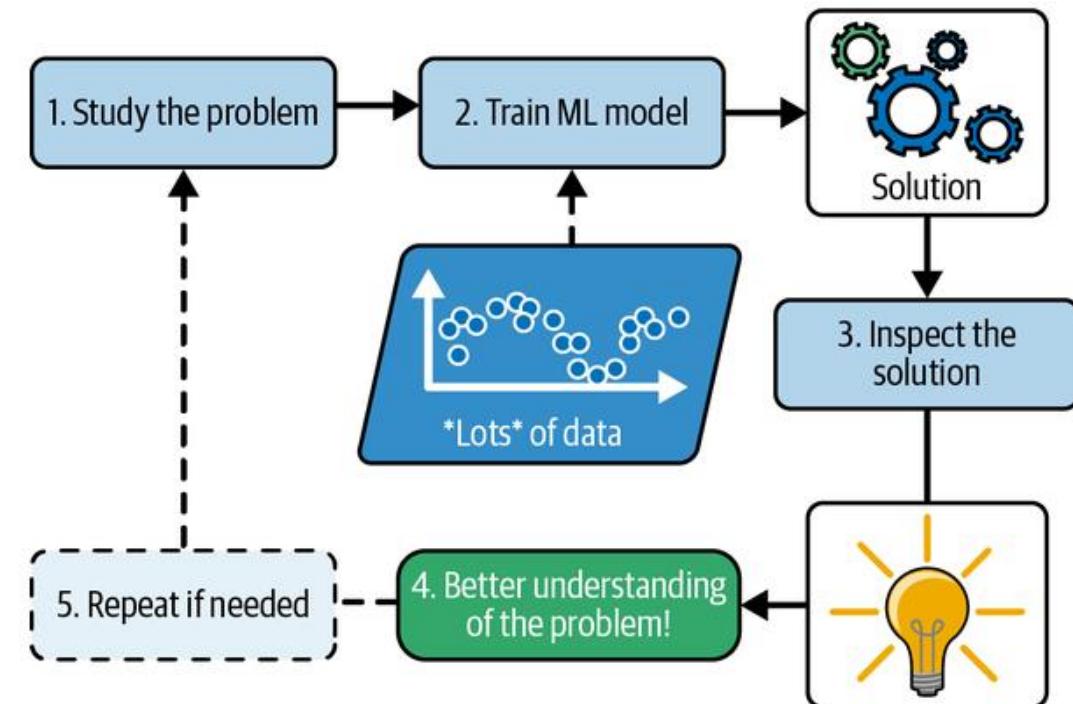
Once a task model has been created we can exploit it to solve the problem on which it was trained.

In the course we will see different models, from simple regressions to neural networks. All models that learn or make use of data are Machine Learning models.

## Example – Spam Filter

Let's say our model is nothing more than a Naive Bayes ( I will explain in detail later).

Let's see the probabilities of the frequency of each word, emails where wealth, money, winnings, etc. are repeated many times. they will automatically be classified as spam without me having to write specific rules for that to happen.



# Types of Machine Learning

Depending on the problem we need to solve, we have different ML algorithms available

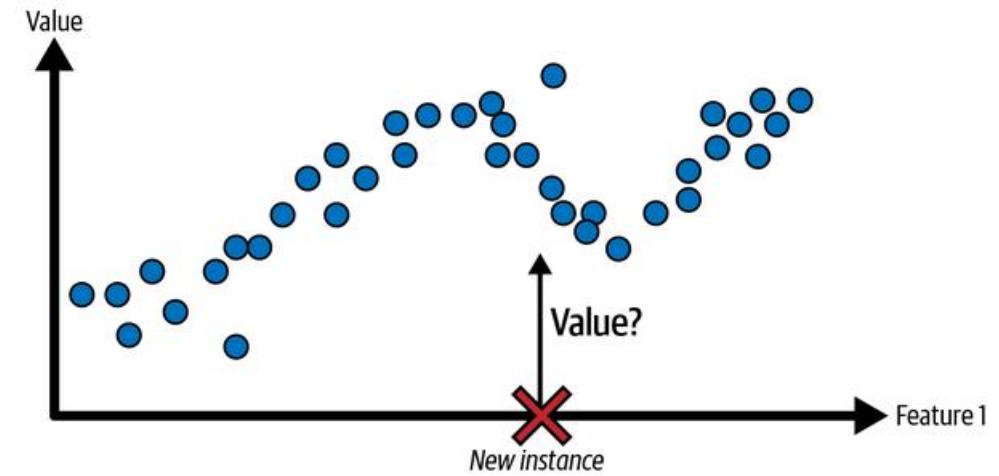
- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

## Supervised Learning

It is the most classic of applications, in which our dataset consists of both X (inputs) and Y (outputs).

The spam filter is an example of this, in the same way if we were to predict the price of a house we will have a history of square footage and areas (X) and prices (Y).

Most algorithms of this type are based on minimizing the distance between the true  $y$  and the  $\hat{y}$  predicted by the model to make **predictions** or **classifications**.

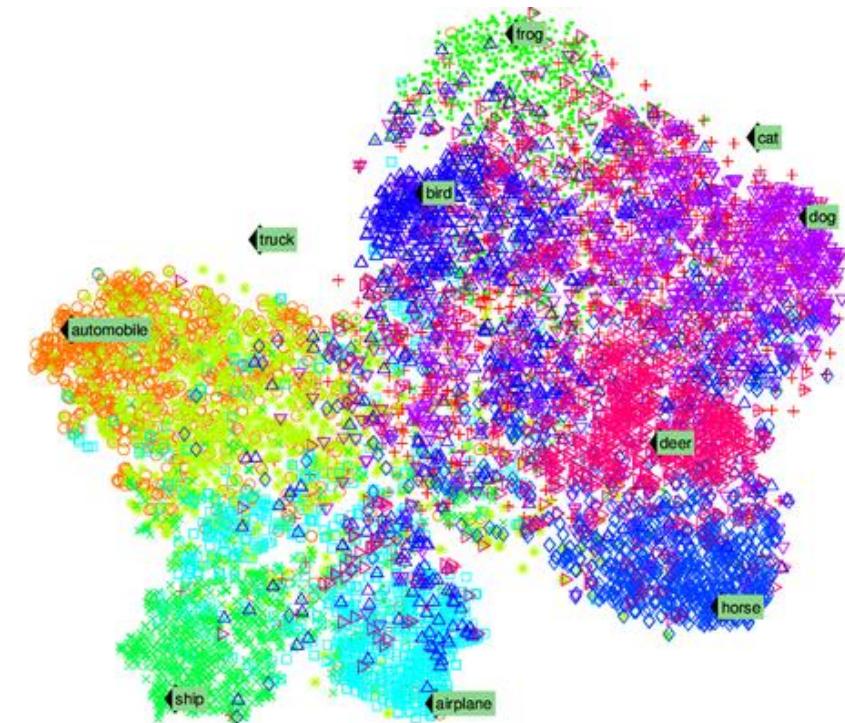


# Unsupervised Learning

In these cases we only have the Xs available but not the Ys.

We want to look for patterns in the data that allow us to understand backwards what they contain.

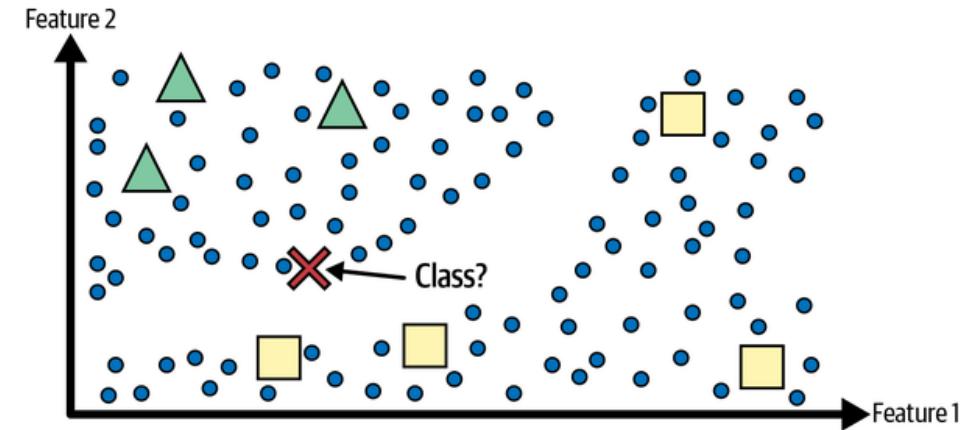
An example is **clustering**, in which we look for similar groups of data, or anomaly **detection**, in which we observe those that deviate from normality.



## Semi- supervised Learning

In this case we have a lot of unlabeled data (label, Y, outputs) and some labeled examples instead.

In the last year, it has become very popular to exploit large databases through self- supervised learning techniques, i.e. by providing tasks other than the one to be solved but which allow the model to learn the distribution of the data and then specialize it on the few labeled examples.

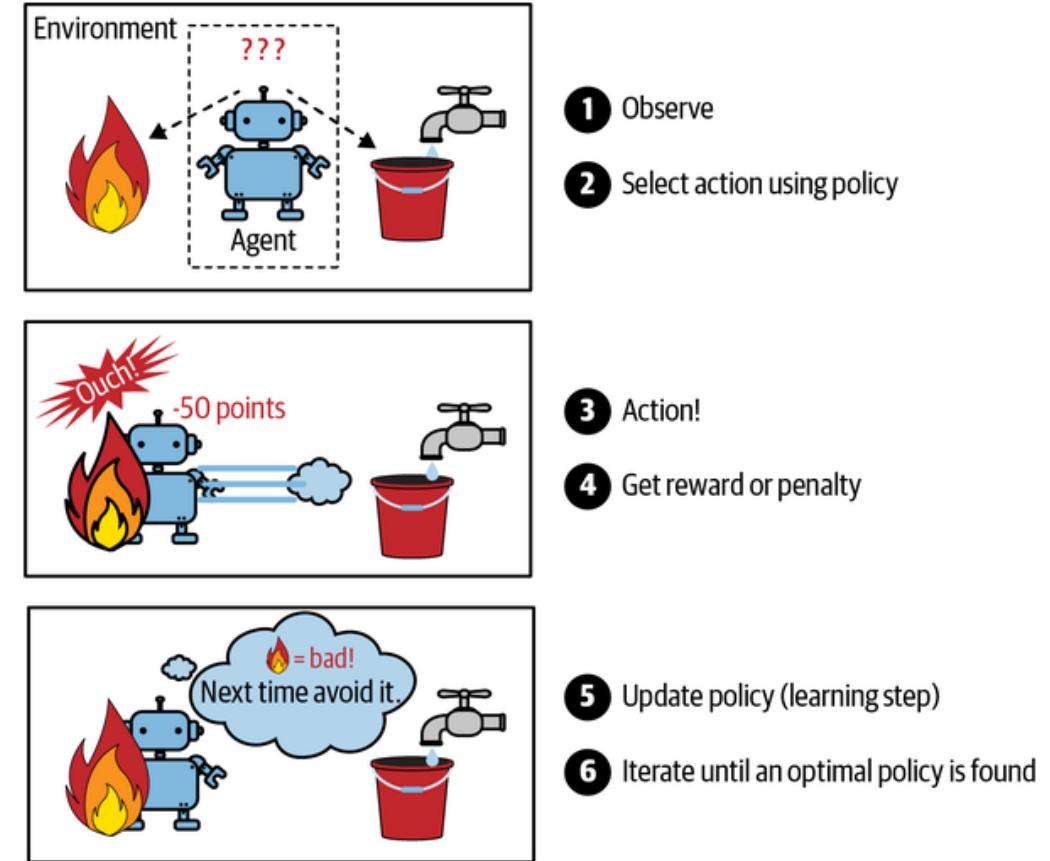


# Reinforcement Learning

It's the type of ML we can leverage when we have an environment to interact with.

The algorithm experiments and is trained for 'reward', examples of which are many AIs in chess, go and other online games.

The algorithm explores different strategies and optimizes with respect to the reward provided to it



# Model based vs Instance based

There is a further distinction

## **Model Based or parametric**

That is, algorithms where a model that describes them is learned from the data, a simplified representation of the problem to be solved through the learning of some parameters

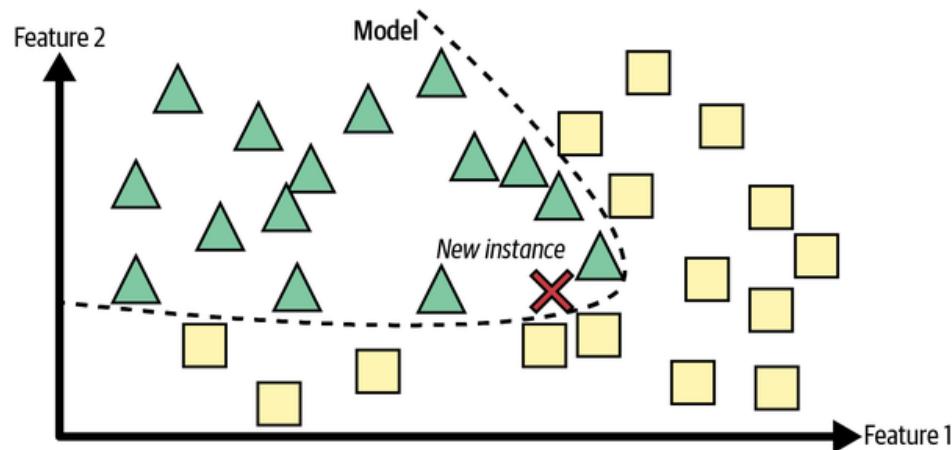
## **Instance Based or Non-parametric**

Algorithms that do not learn parameters but exploit the data itself to compare a new example using similarity metrics.

# Model based vs Instance based

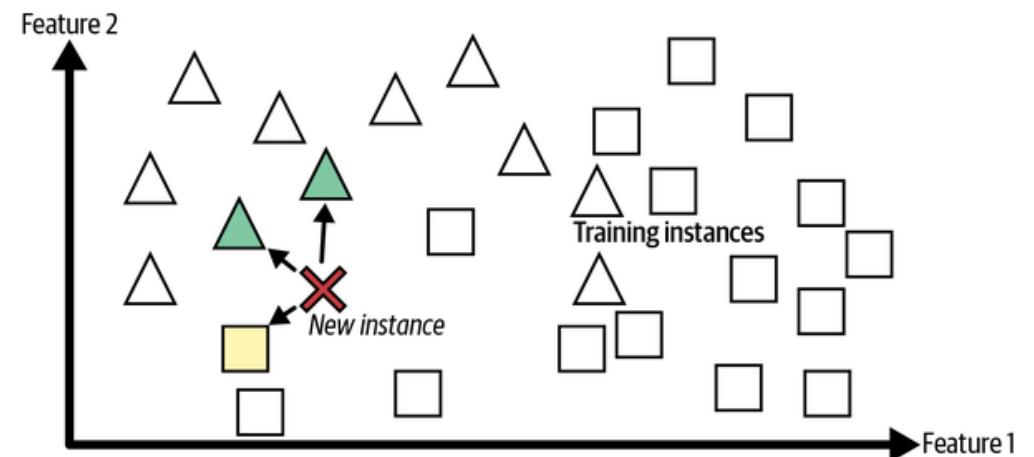
## Model based - parametric

The curve separating the two classes will be described by a model  $\alpha x_1^2 + \beta x_2^2 + \gamma$  with  $\alpha, \beta, \gamma$  parameters

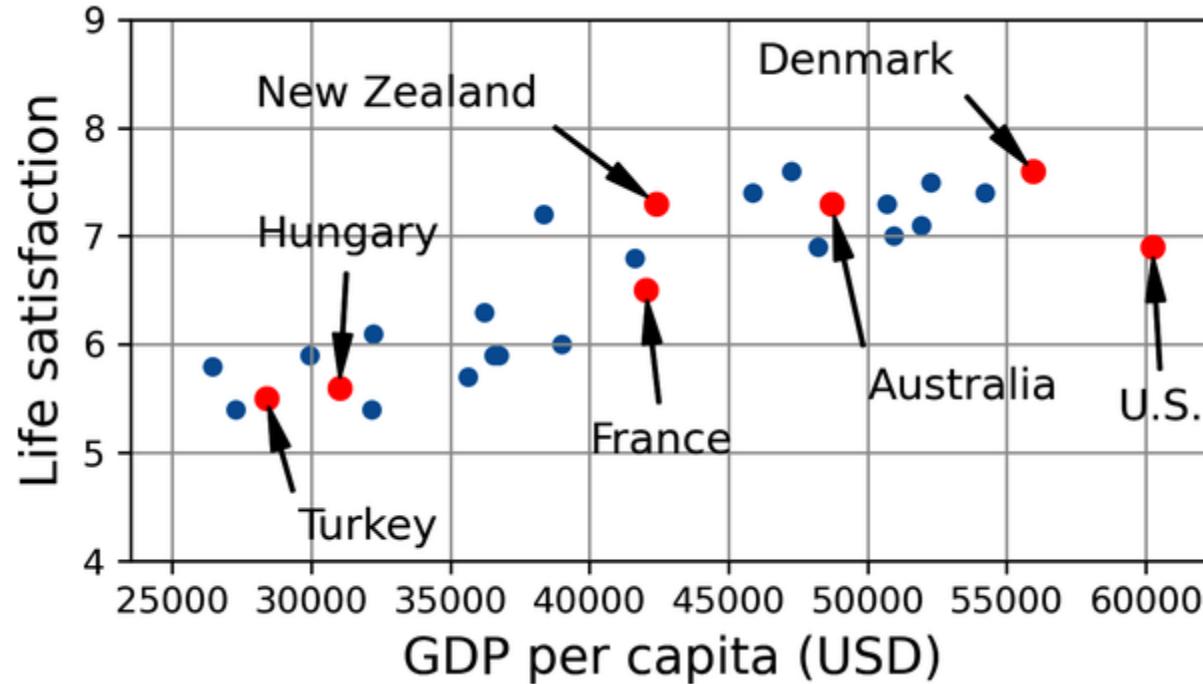


## Instance based - non-parametric

The new example will be compared with neighboring examples and classified according to the similarity policy



## E0101 – Does money make you happier?



[https://colab.research.google.com/github/ageron/handson-ml3/blob/main/01\\_the\\_machine\\_learning\\_landscape.ipynb](https://colab.research.google.com/github/ageron/handson-ml3/blob/main/01_the_machine_learning_landscape.ipynb)



# **02 – Regression Models**

Daniele Gamba

2022/2023

## Premise

Each author, framework and tutorial uses different conventions to indicate features, parameters and matrices.

To orient yourself in the slides, remember that usually

- $x, u, \varphi$  they represent inputs, inputs, features
- $\beta, \theta, \omega$  represent the parameters
- $\hat{y}, y(\omega), \hat{t}$  represent our estimates
- $x_t$  it is the single data,  $x_i$  it is the vector of the variables of an example,  $X$  the complete matrix

In case you don't understand what an equation refers to, ask me without problems

I won't ask you for one notation rather than the other as long as it is understood

# Regression

The purpose of regression is to identify the parameters that describe the **model generating the data**.

Generally we talk about regression for **continuous** supervised **learning problems**.

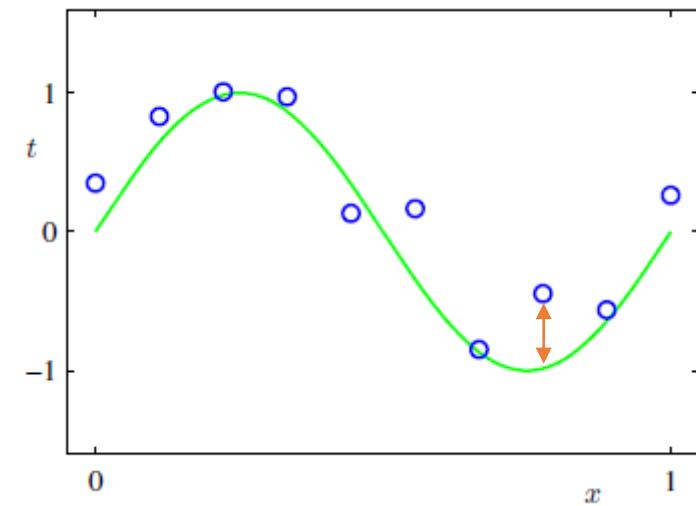
This topic should have already been covered extensively in other courses (below) and will therefore be covered briefly

- IMAD
- Stochastic Models
- Statistical Learning

# Errors

All data is plagued by different types of errors

- Measurement errors
- Quantization errors
- Representation errors
- Human errors
- ...



The objective is to leave only these contributions in the error term, correctly identifying the other parameters. As you will have seen in Stochastic Models, one of the ways to evaluate whether errors contain other information is to do a "whiteness test", or make sure that the correlation between the error and our  $y$  is zero.

# Model

The classic regression model is described by

$$y = \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_0 + e$$

Where

- $y$  is our exit
- $x_1, x_2$  are the input variables
- $\theta_1, \theta_2$  are the parameters we are looking for,  $\theta_0$  it is a constant parameter also called intercept
- $e$  is the irreducible error of the data

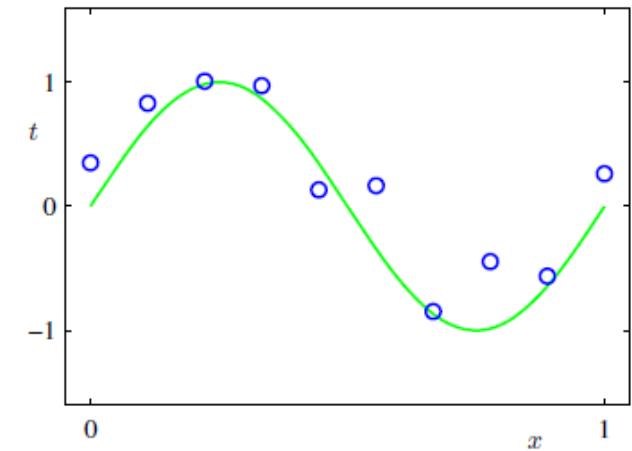
Let's imagine having to predict the satisfaction index of a state given its Gross Domestic Product per capita.

## Polynomial regression

Let's take the previous graph and assume that we need to regress the model only between an  $x$  and a  $y$  (Single Input, Single Output).

Since we only have one variable we can apply a series of **basis functions** to it that describe different behaviors.

As **features** we can use the polynomial expansion of our input variable.



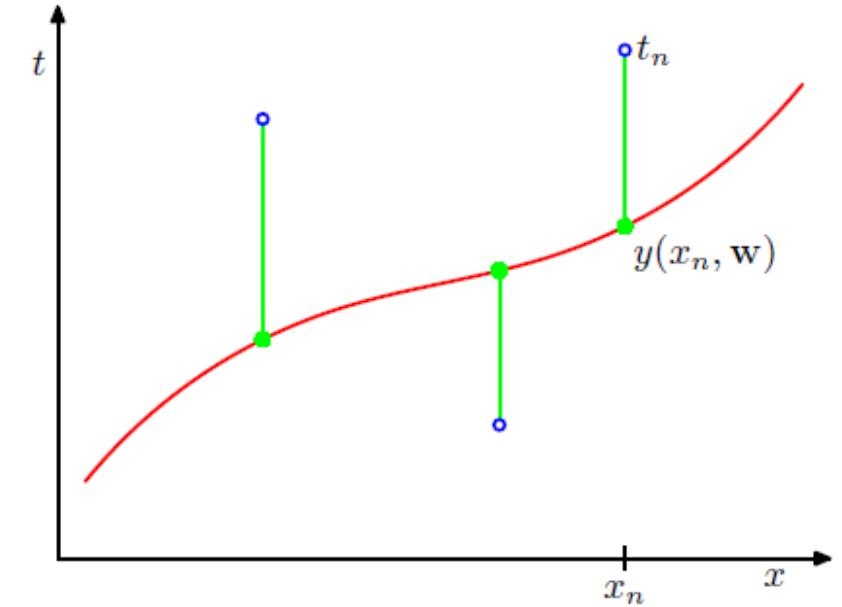
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{j=0}^M w_j x^j$$

## Polynomial regression

Once we have defined our function, let's define our metric.

In many, but not all, cases, it is very convenient for regression problems to use the Mean Squared Error , the mean square error.

So let's define how



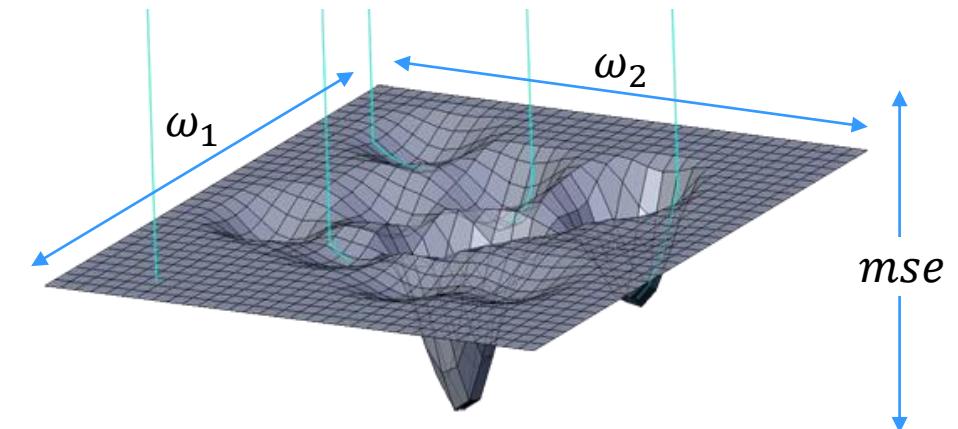
$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2$$

## Solution space / parameters

Depending on the values  $\omega_1, \omega_2$ , etc. we will have a different performance, that is, the error between our predictions and the real data will be different.

There are different combinations of parameters that give the same performance, there is usually only one optimal solution, i.e. with the lowest absolute error value.

Normally, optimal parameters are searched for by numerical optimization via gradient descent.



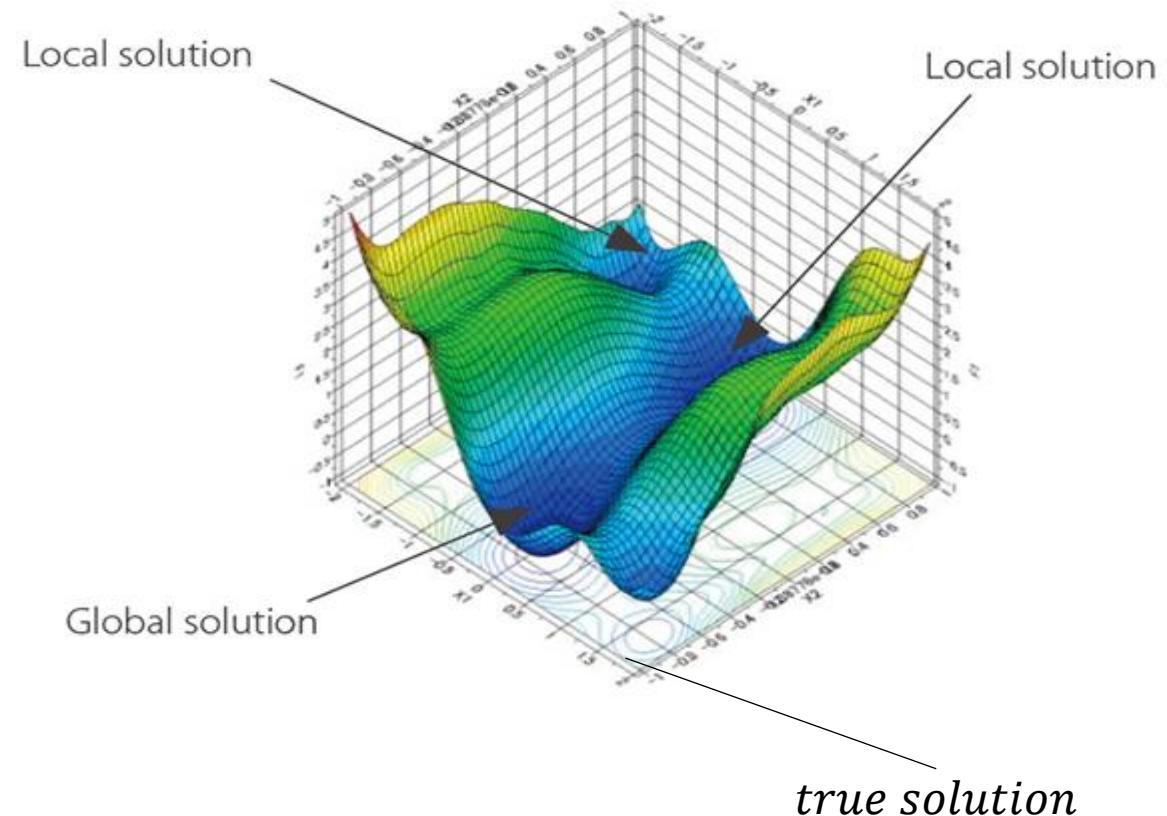
## Solution space / parameters

Depending on the parameters we will have local or global optima.

The achievable optimum does not necessarily coincide with the actual optimum.

It is not certain that we will ever be able to solve a problem exactly because

- We don't have enough information
- There is no numerical solution to the problem
- There is too much noise in the data



## Ordinary Least Squares

Only in the case of **linear regressions with mse** ( mean squared error ) we can find the optimal solution in one fell swoop through OLS.

If we rewrite our problem as

$$\mathbf{X} = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1p} \\ X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{np} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.$$

$$S(\boldsymbol{\beta}) = \sum_{i=1}^n \left| y_i - \sum_{j=1}^p X_{ij} \beta_j \right|^2 = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2.$$

then we can get the **optimal parameters** in one go

$$(\mathbf{X}^\top \mathbf{X}) \hat{\boldsymbol{\beta}} = \mathbf{X}^\top \mathbf{y}.$$

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

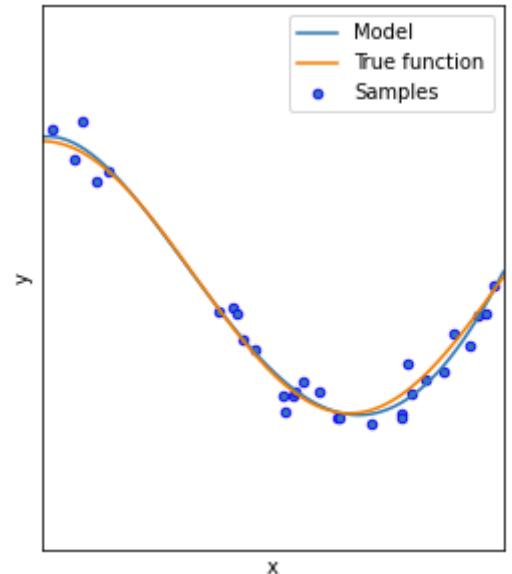
## Polynomial regression

We conclude the example of polynomial regression with a quick reference to the selection of features that we will see better in the next lesson.

If we have

$$y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j$$

what degree of  $x$  do we get to describe  $y$  ?

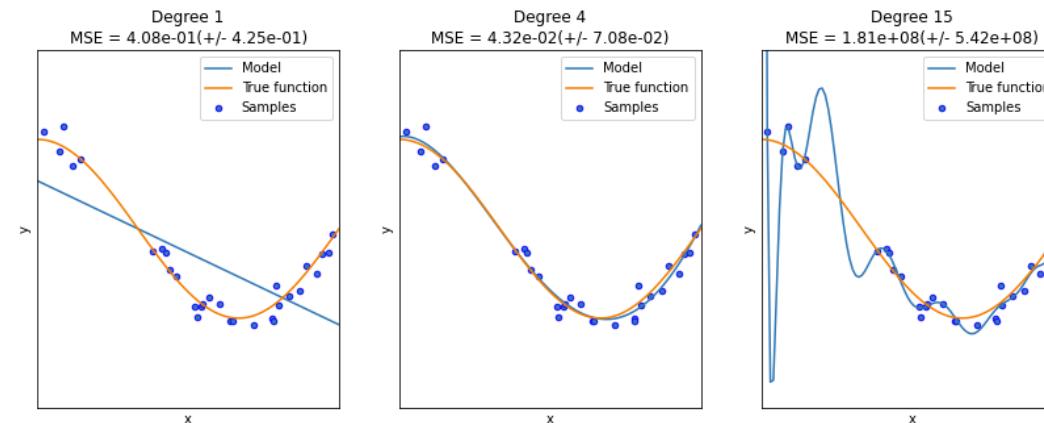


# Polynomial regression – Overfitting and Underfitting

We find ourselves faced with the problem of overfitting or underfitting .

Choosing the right features, between variables and basic functions to apply to the variables, is difficult.

Putting too many will mean over-fitting the model and making mistakes in learning the data model, putting too few will give us very low performance. Let's see it with a practical example



[https://colab.research.google.com/drive/1kRmqPUYDtRHpq1oaFekm2vLrj-NqVndw#scrollTo=AqQ\\_mcg-m0yi](https://colab.research.google.com/drive/1kRmqPUYDtRHpq1oaFekm2vLrj-NqVndw#scrollTo=AqQ_mcg-m0yi)



# **03 – Regularization & Model selection**

Daniele Gamba

2022/2023

## Extra

What if we knew the basic function that generates our data?

<https://colab.research.google.com/drive/1kRmqPUYDtRHpq1oaFekm2vLrj-NqVndw#scrollTo=duRMAKBDGsxo>

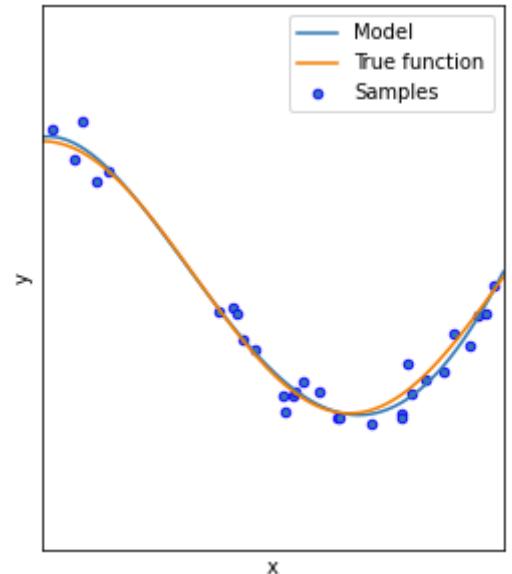
## Polynomial regression

Let's take the example of polynomial regression again with a quick reference to the selection of features which we will see better in the next lesson.

If we have

$$y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j$$

what degree of  $x$  do we get to describe  $y$  ?

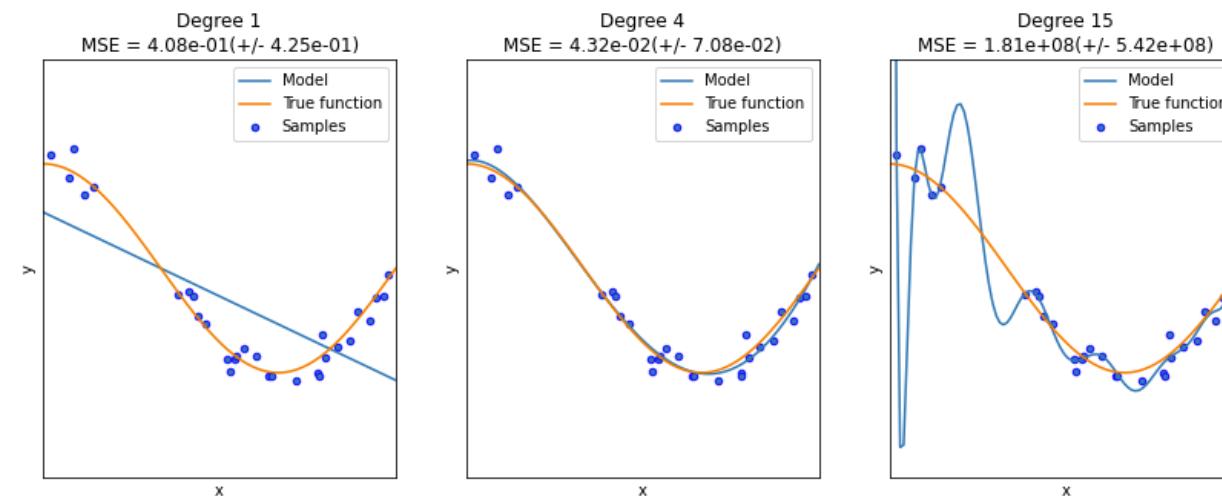


# Polynomial regression – Overfitting and Underfitting

We find ourselves faced with the problem of overfitting or underfitting .

Choosing the right features, between variables and basic functions to apply to the variables, is difficult.

Putting too many will mean over-fitting the model and making mistakes in learning the data model, putting too few will give us very low performance.



# Overfitting

Let's start with the concept of overfitting .

Our model is trying to minimize the error even at the expense of learning a performant model. Not knowing a priori how complex the model that generates the data is, he tries to optimize as much as he can.

The more features we have or the less data per feature we have, the higher the risk of overfitting .

So we can

- Try to collect more data
- Select our features better
- Limit model learning

## More data?

If we have a problem from which we can easily sample more data it is better to collect as much as possible, as long as they are IID, i.e. **independent and identically distributed** .

If we sample non-independent data several times we would not add any information to the model.

### Requests

- Is sampling different frames from the same video useful?
- Perform the same scientific experiment multiple times?

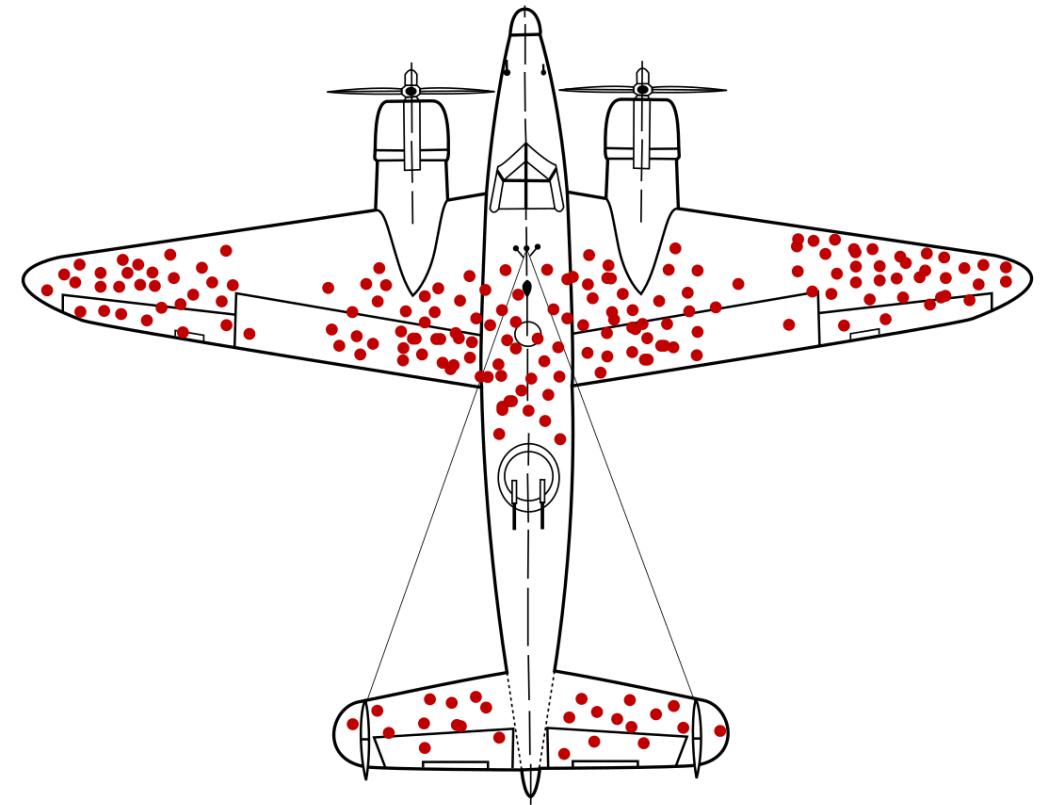
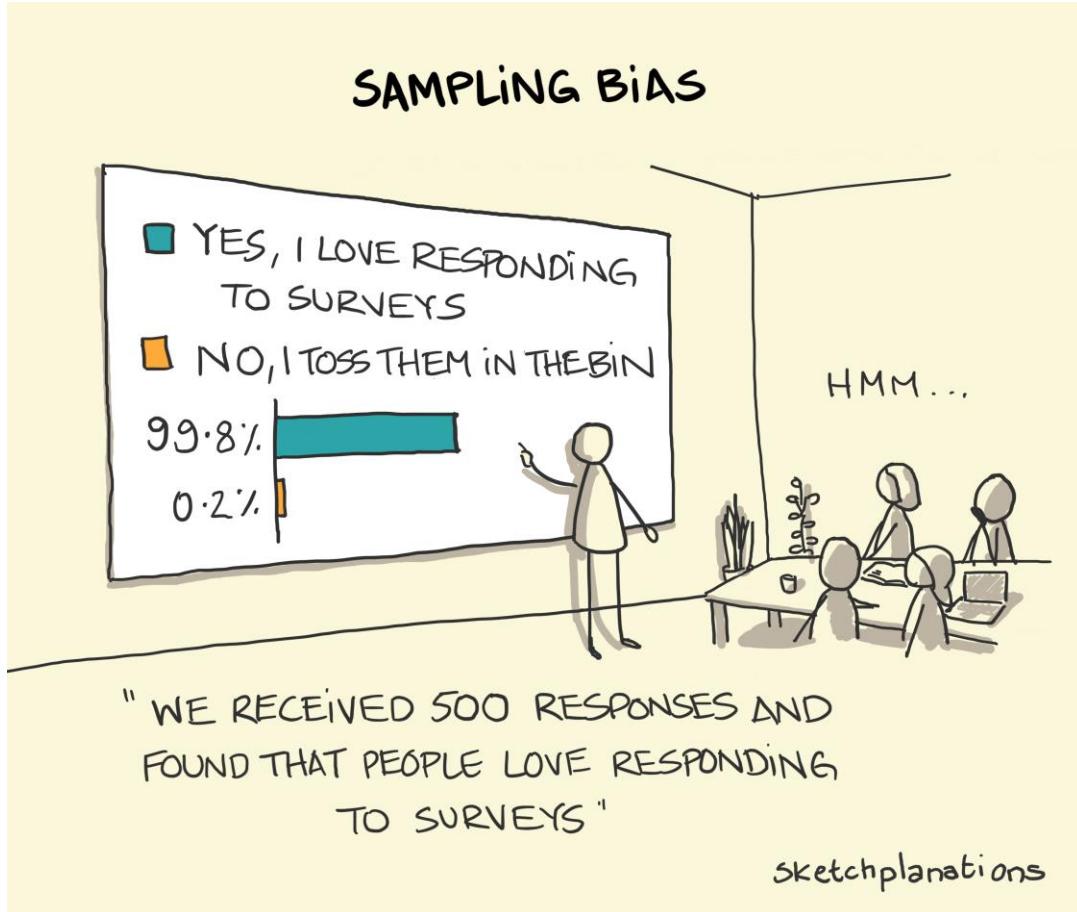
Most of the time it is not possible to have access to more data, perhaps because it is very expensive to collect and label it or because it is not possible at all.

## Data problems

We have seen that data contains **errors** of different nature, this poses a series of challenges to the algorithms

- **Insufficient data** (e.g. train an LLM on 10 sentences)
- **Poor quality data** (e.g. full of annotation errors)
- **Non-representative data** (e.g. acquired in conditions other than production)
- **Sampling ( selection ) bias** ( next slide)
- **Irrelevant features**
- **Confounding features** (we will see this at the end of the course)

# Sampling Bias



Distribution of bullet holes on aircraft returning from missions

## Thumb rule

The rule of thumb is to have 20 examples per parameter.

In our case of polynomial regression, if we had 20 examples we could use at most 1 degree of freedom (i.e. the mean), with 40 we could add linear dependence. For the best performance of the fourth-order model we should have had 100 examples.

It is not always possible to have 20 examples per parameter, so we must study techniques that allow us to be robust even if we do not reach this threshold.

## Features selection

A first way is to select only the features that are most *significant* for the problem we are trying to solve.

Fewer features = less expressive capacity of the model but more examples per parameter.

How can we select the different features?

One way is to only take the features with the correlation with the highest output (e.g. Pearson's Correlation Coefficient PCC).

Another way is to use **regularization techniques** that allow us to send the least "useful" terms to zero.

# Regularization

Returning to our example of polynomial regression, if we look at the coefficients of each feature we observe that the parameters explode in size as the number of features increases.

```
Grade 1  
↳ [array([-1.60931179]),  
 array([-7.31956683,  5.55955392]),  
 array([ 0.46754142, -17.78954475,  23.5926603 , -7.26289872]),  
 array([-2.98291188e+03,  1.03898766e+05, -1.87415056e+06,  2.03715125e+07,  
       -1.44872551e+08,  7.09311979e+08, -2.47064676e+09,  6.24558367e+09,  
       -1.15676035e+10,  1.56894317e+10, -1.54005437e+10,  1.06456871e+10,  
       -4.91375763e+09,  1.35919168e+09, -1.70380199e+08])]  
  
Grade 15
```

## Ridge Regression

We can therefore hypothesize to put a term that negatively weighs the explosion of the parameters, adding a **penalty term** to our cost function.

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

This regression is called **Ridge Regression** or **L2** and takes into account the square of the norm of the parameters.

It's the only other case where we can use OLS to get the optimal  $\mathbf{w}^*$ 's for our problem.

# Ridge Regression

Thanks to Ridge we can control the complexity of the model only from one parameter  $\lambda$ .

Below is an example of how different values impact  $\lambda$  the parameters learned from a fixed order polynomial regression.

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^*$	0.35	0.35	0.13
$w_1^*$	232.37	4.74	-0.05
$w_2^*$	-5321.83	-0.77	-0.06
$w_3^*$	48568.31	-31.97	-0.05
$w_4^*$	-231639.30	-3.89	-0.03
$w_5^*$	640042.26	55.28	-0.02
$w_6^*$	-1061800.52	41.32	-0.01
$w_7^*$	1042400.18	-45.95	-0.00
$w_8^*$	-557682.99	-91.53	0.00
$w_9^*$	125201.43	72.68	0.01

## Lasso Regression

We can extend the case of norm penalization to any order  $q$ .

In the general case

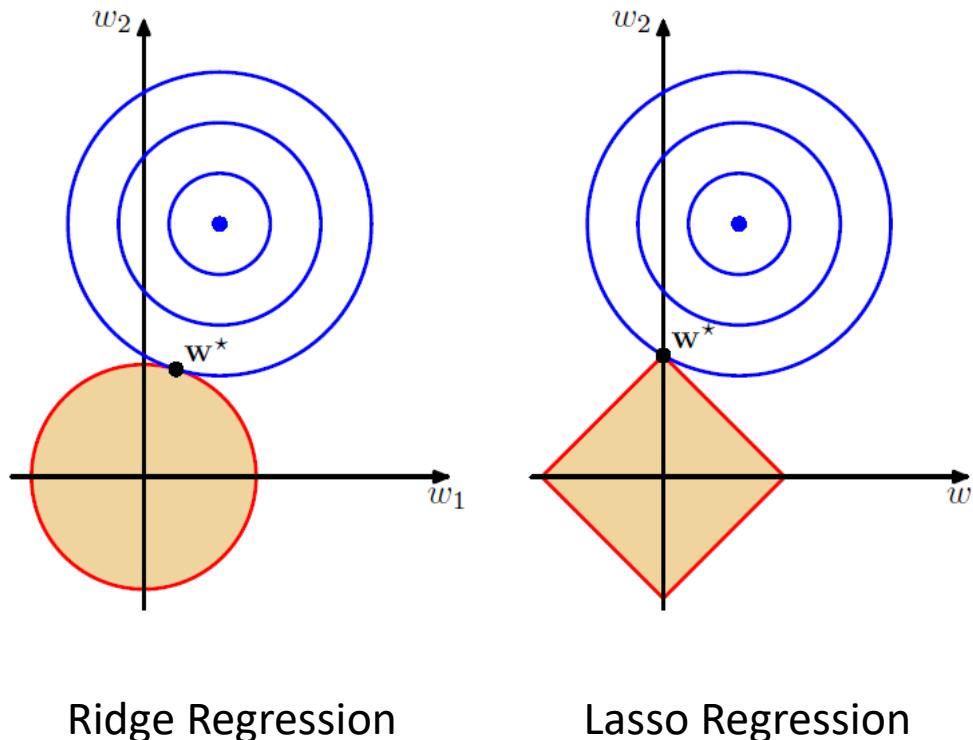
$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$

The special case  $q=1$  is called **Lasso Regression** or **L1**.

The Lasso is obtained by numerical optimization through iterations (we will see how later).

## Lasso Regression

Lasso has the disadvantage of not being able to use OLS to find the optimal solution but has the big advantage that it tends to send the less significant features to zero.



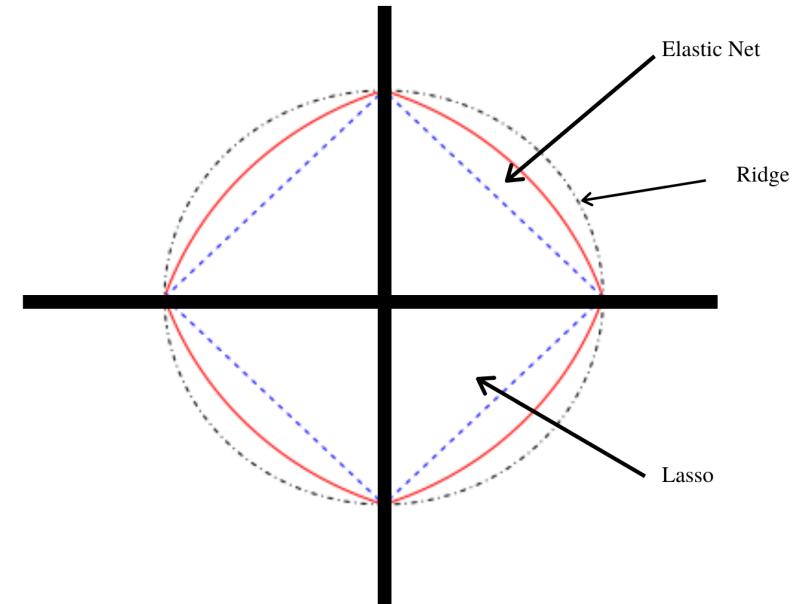
Ridge Regression

Lasso Regression

## Elastic Net

Elastic Net regularization linearly combines Ridge and Lasso into a single equation.

$$\hat{\beta} \equiv \underset{\beta}{\operatorname{argmin}} (\|y - X\beta\|^2 + \lambda_2 \|\beta\|^2 + \lambda_1 \|\beta\|_1).$$



It is used in a nonparametric technique that we will see later called Support Vector Machine.

## Penalized regression

Let's now see in practice how Ridge and Lasso behave on the parameters of our polynomial regression from the previous lesson.

<https://colab.research.google.com/drive/1kRmqPUYDtRHpq1oaFekm2vLrj-NqVndw#scrollTo=duRMAKBDGsxo>

## Akaike Information Criterion

Another method to penalize models is to evaluate the gain of adding an extra feature to the model. Assuming we have  $d$  parameters and  $N$  data, the dependence on the number of features is added to the cost function.

$$\text{AIC}(d) = 2 \cdot \frac{d}{N} + \ln [ E(\hat{\theta}_N; d) ]$$

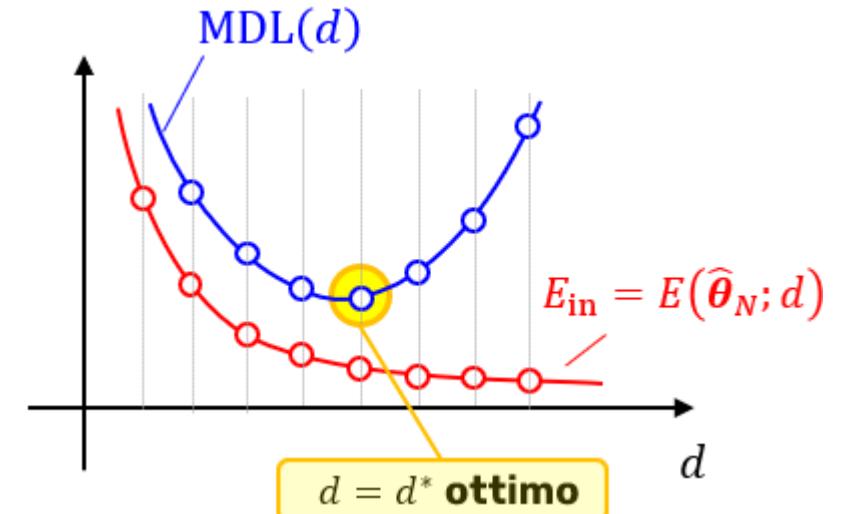
Where  $E(\hat{\theta}_N; d)$  is our performance measure. The more parameters we have, the more the  $d/N$  value grows and penalizes our performance.

## Information Criteria

In addition to Akaike , there are other criteria based on "information" that we add to the model, including Bayesian Information Criterion (BIC) and Minimum Description Length (MDL).

In all cases, models are built that penalize the addition of parameters depending on the data and their capacity for expression.

$$\text{MDL}(d) = \ln[N] \cdot \frac{d}{N} + \ln[E(\hat{\theta}_N; d)]$$



## Recap

So far we have seen simple linear regression models.

Even when doing a simple regression we must be careful not to overfit our data.

There are a number of techniques that allow us to control the complexity of our model.

So far we have only seen techniques that intervene on the model, what if we intervened on the data?

## Train, validation, test

Validation is the operation of measuring the performance of our model on a set of data on which it has not optimized.

In case we have enough data it is very convenient to divide our data into

- **Train** : the dataset on which our model learns
- **Validation** : the dataset we use to make model selection
- **Test** : the dataset on which we only measure our final performance

The three datasets must be sampled by the same process, otherwise we will make wrong decisions (e.g. due to a sampling bias all the difficult cases end up in the test set).

## Train, validation, test

If we have a lot of data we can divide our dataset randomly **between** the 3 datasets, normally percentages are used

- 60, 20, 20
- 70, 15, 15
- 70, 20, 10

The training dataset will have the largest quantity as we have to base our training only on that.

In some cases it may make sense to look back at the distributions of the data to be sure that they are distributed consistently.

## Model selection

When we train multiple regression models, we will choose the best performing one on the validation dataset.

### Example

- We create polynomial regression models from degree 2 to degree 15
- We measure the performance of each model on the validation dataset
- Let's take the best performing model on validation
- We measure its performance on the test, that result *should* be the performance of the production model

Using the test dataset to choose the best model is cheating until you put the model into production.

Once we have chosen the best model we can retrain it on training and validation.

## Validation

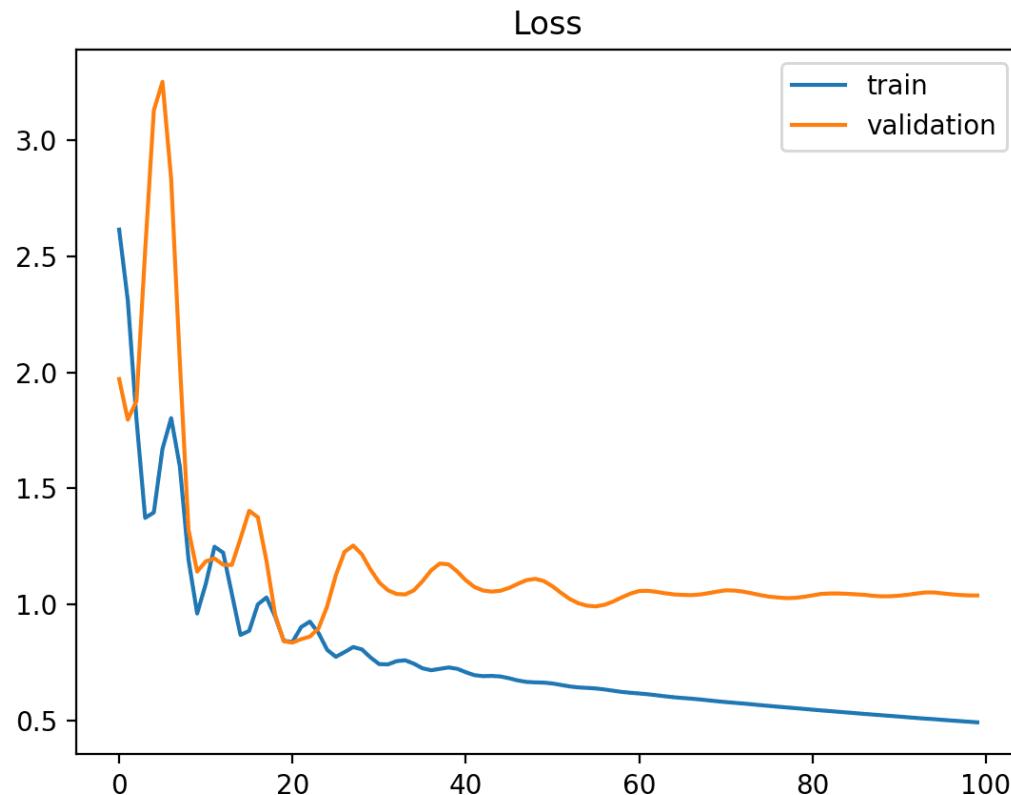
The validation dataset is also sometimes called out-of-sample and its performance is referred to as

$$MSE_{val} \text{ or } MSE_{out}$$

in the case of the MSE.

In the case of iterative learning, by gradient descent, we can measure the performance on the validation dataset even during learning. In this way we will evaluate if and when the model starts to perform.

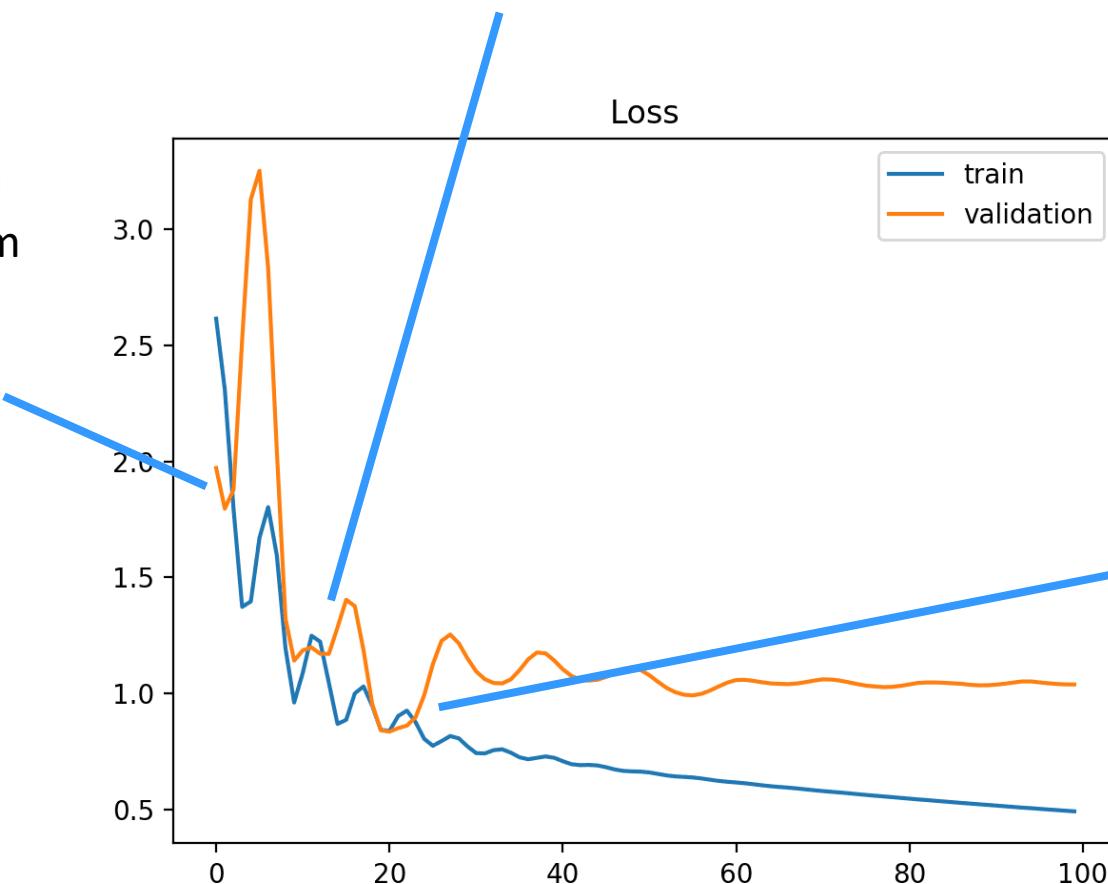
# Validation



# Validation

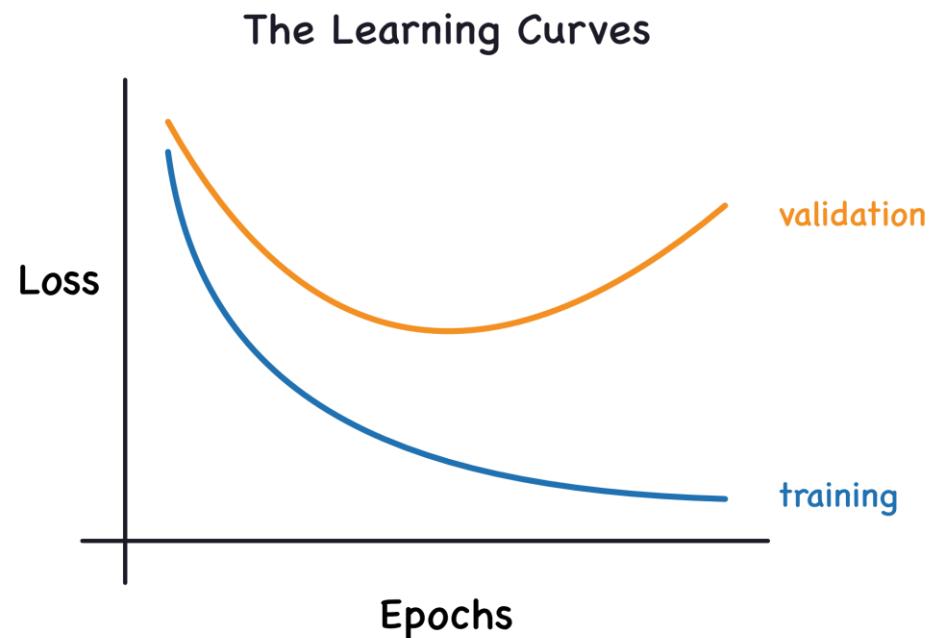
Beginning of learning, model random weights may perform better on validation than on training

optimization

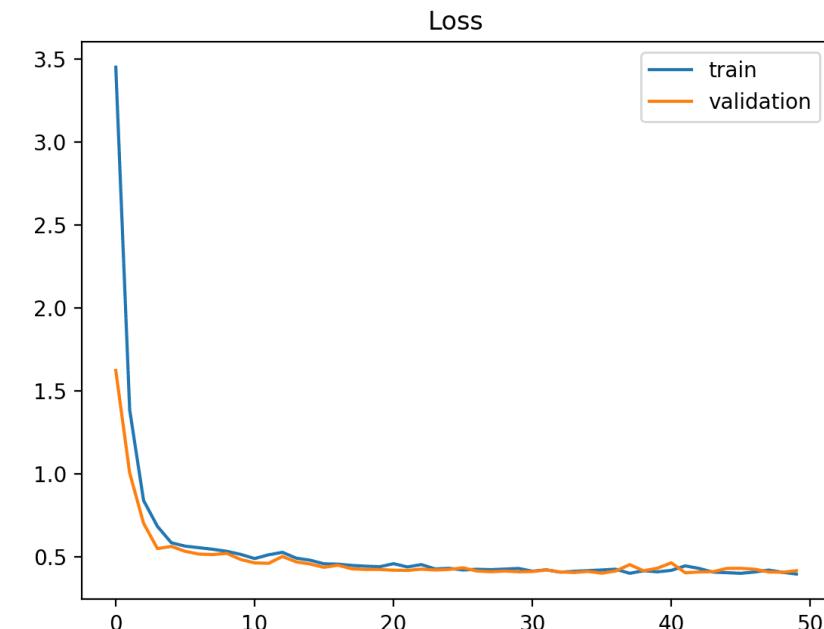


Training and validation begin to diverge, the model begins to overtrain and loses generalization on unseen data

## Cases



Overfitting -> validation grows  
Early Stop

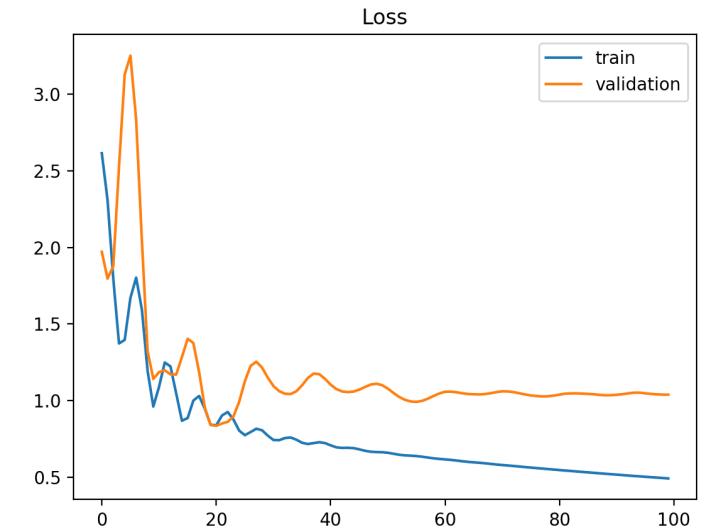


Underfitting or ideal model -> validation and training have no differences

## Validation during training

At each iteration we are comparing a new model (in parameters) to the validation data.

In the case of iterative learning we will have to compare not only the models during their training, but also between all possible other models.



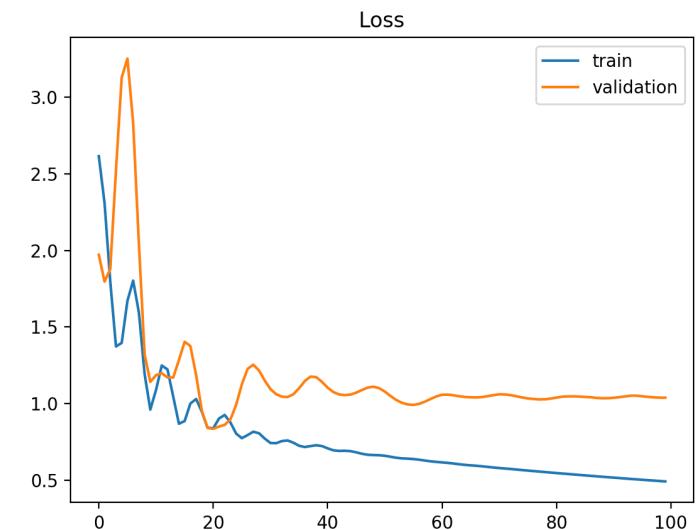
## Early Stop – Best in validation

There are several techniques for taking a good model during training

- **Early stop**, i.e. we stop training when we observe train and validation diverging
- **Best validation**, i.e. we save the model that performs best in validation

Both techniques are very prone to noise in the data and to "lucky" moments in which our parameters better express the variance of the data in validation compared to training.

Especially in these cases, observe the training-validation graph **it must guide us** in choosing the model and sizing it appropriately (which features, how many neurons - we will see later, etc.)



## Cross-validation

In case we do not have enough data to divide a priori between train , validation and test, we must use other techniques that allow us to validate our performances.

Cross-validation consists of reusing a piece of the training set itself to validate performance by re-training not one but N models.

Carrying out cross-validation therefore requires carrying out N training sessions and can be more expensive in terms of computation.

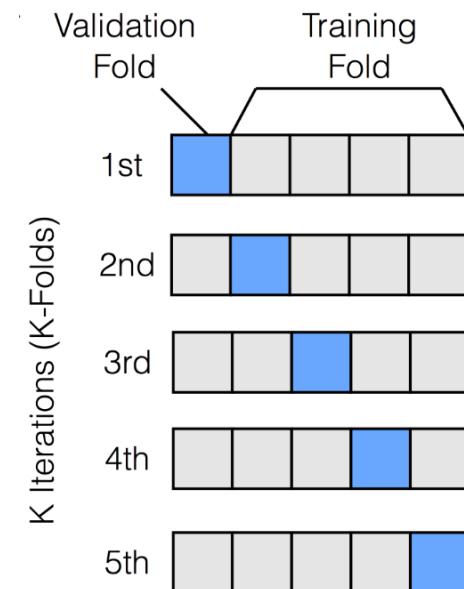
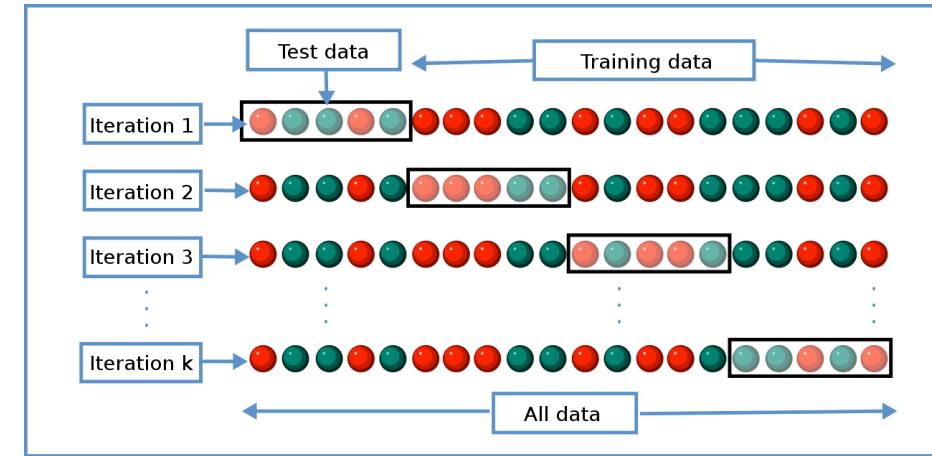
## k- fold Cross- Validation

In the case of **k- fold cross- validation** let's split our dataset into k groups, and execute the training on data

$\frac{N}{k}(k - 1)k$  times and measure the performance on the

latest  $\frac{N}{k}$  data.

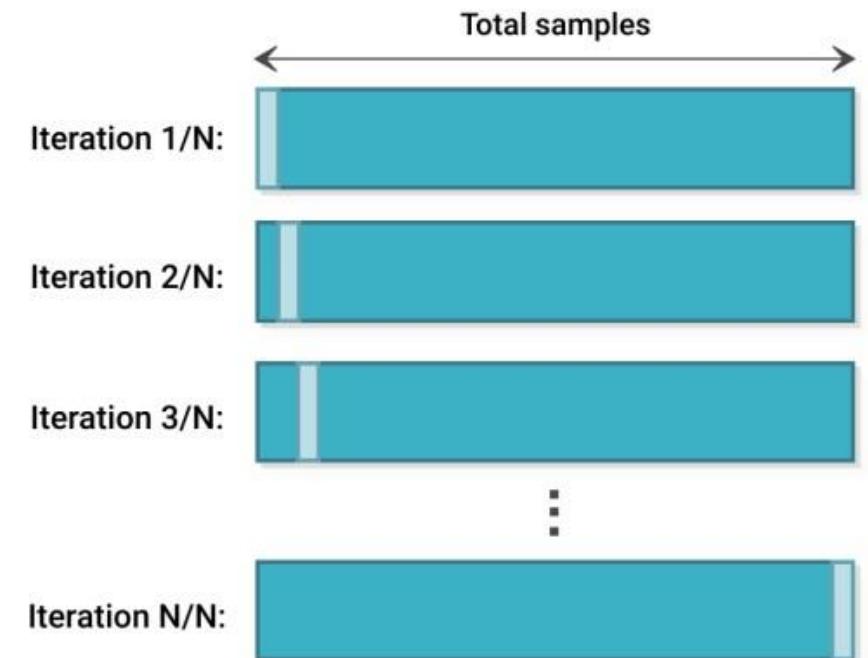
Our cross-validation performance will be the average of the k test performances.



## Leave -one-out

In the case in which we have **very little data**, but very little for which even dividing by  $k$  would result in a training set that is too small, we can take  $k = N$  and perform  $N$  training sessions leaving only one piece of data out each time.

Our cross-validation performance will be the average of all prediction errors on the individual examples excluded from training.



## Validation and cross-validation

Now that we know how to measure the model's ability to generalize to unobserved data, let's take the example of polynomial regression and observe how the model that visually performs best is also the best in cross-validation.

<https://colab.research.google.com/drive/1kRmqPUYDtRHpq1oaFekm2vLrj-NqVndw#scrollTo=duRMAKBDGsxo>

## Recap

We have seen

- How to intervene on the data to choose the best model
- How to read the differences between training and validation
- How to carry out analysis with little data

Let's see everything together in an example

[https://colab.research.google.com/github/ageron/handson-ml3/blob/main/04\\_training\\_linear\\_models.ipynb#scrollTo=vk57NkQKoS0j](https://colab.research.google.com/github/ageron/handson-ml3/blob/main/04_training_linear_models.ipynb#scrollTo=vk57NkQKoS0j)



# **04 – Classification**

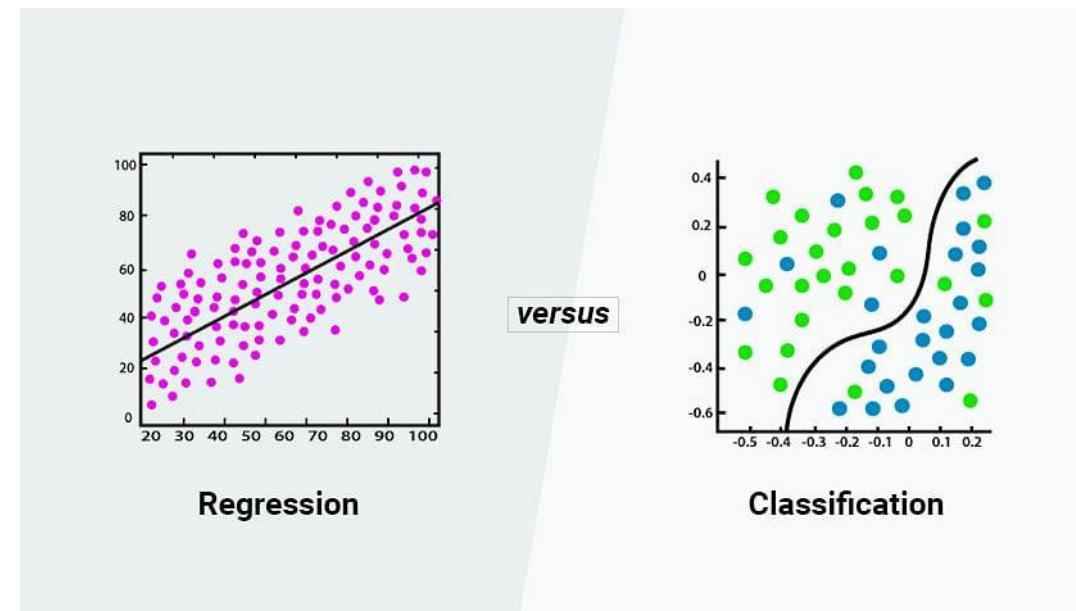
Daniele Gamba

2022/2023

# Classification

**Classification** problems are problems in which we need to determine what class an example belongs to.

Determining a cat's breed or whether a patient is healthy or sick are classification problems.



# Classification

A classification model will therefore have a set of features and will have one or more classes at the output.

The classification can be

- **Binary**, in case we have two classes (0/1) or one-vs- rest (e.g. black hair vs other)
- **Multi-class**, in case we have N classes to choose from (e.g. Siamese, Persian, Burmese, ...)

Furthermore, a problem can also be **multi-label** in case our example can belong to multiple classes at the same time

# Classification

**Binary Classification**



**Multiclass Classification**



**Multilabel Classification**



# Performance

In the regression we saw that there are different performance metrics, taking them back

## **Mean Squared Error**

The main one, even in the Root MSE version, is the most robust and allows us to use OLS

## **Mean Absolute Error**

performs better where we may have dirtier data and for a "higher level of detail" in generative models

## **Mean Absolute Percentage Error**

More difficult to converge but great for problems where percentage error is important.

# Confusion Matrix

Similarly for classification problems we have performance measures.

		True condition			
Total population	Condition positive	Condition negative	Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$	
Predicted condition	Predicted condition positive	True positive, Power	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR}^+}{\text{LR}^-}$
		False negative rate (FNR), Miss rate = $\frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$	

## Accuracy , F1-score

Accuracy represents the total of how many we correctly predict out of the total.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

It is the most important metric but it errs the more the classes are unbalanced.

*A model that tells you that it is never Christmas Day given the day will have an accuracy of 99.7%.*

For this we need other metrics, including the  $F_1$  score harmonic mean of precision and recall.

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2\text{tp}}{2\text{tp} + \text{fp} + \text{fn}}.$$

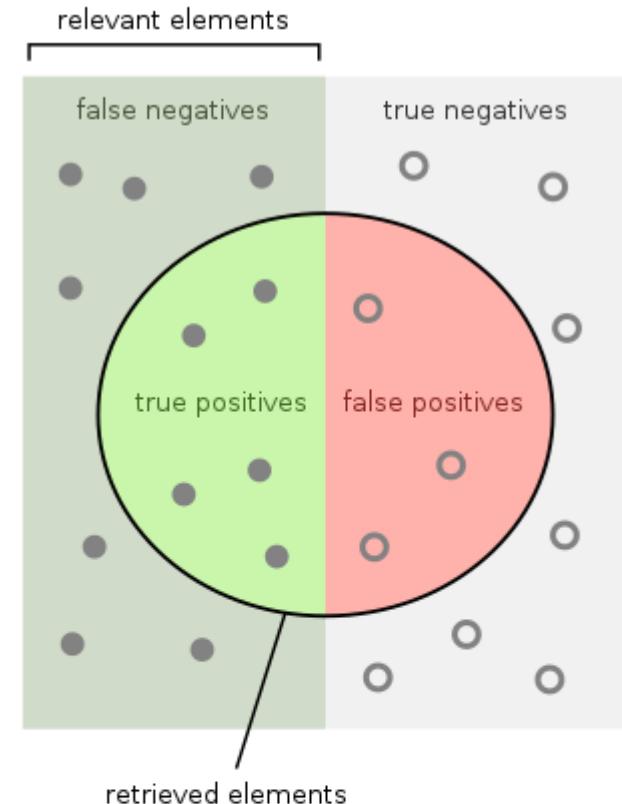
# Precision, Recall

In addition to the total accuracy ( Accuracy ) of the model, we are often interested in evaluating it

## Precision

How many correct predictions out of the total of correct and false positives

«How many images in which I say there is a dog are there really a dog?»



## Recall

How many correct predictions out of the total of correct and false negatives.

«How many dog images can I identify out of the total number of dog images?»

How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

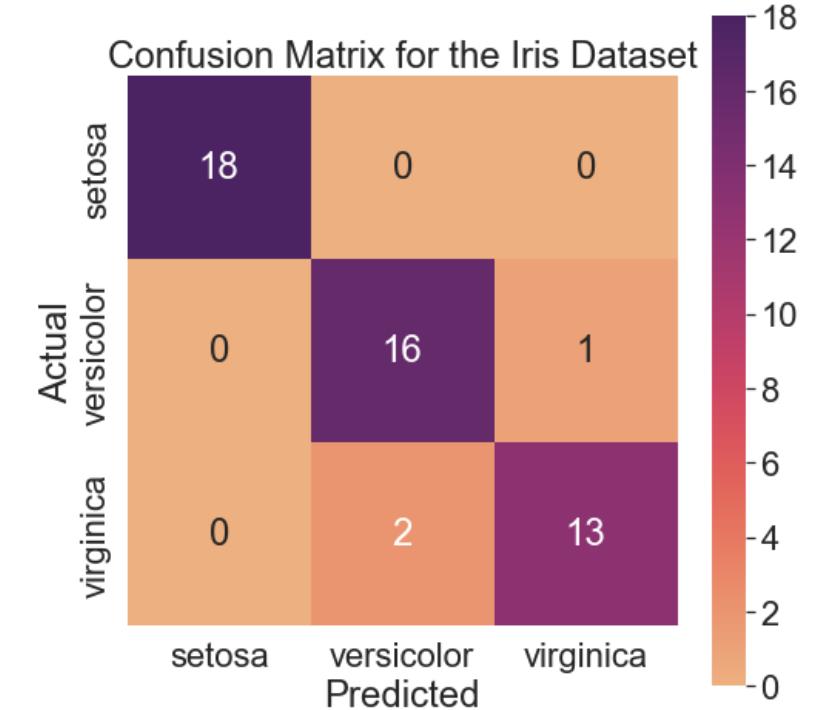
How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

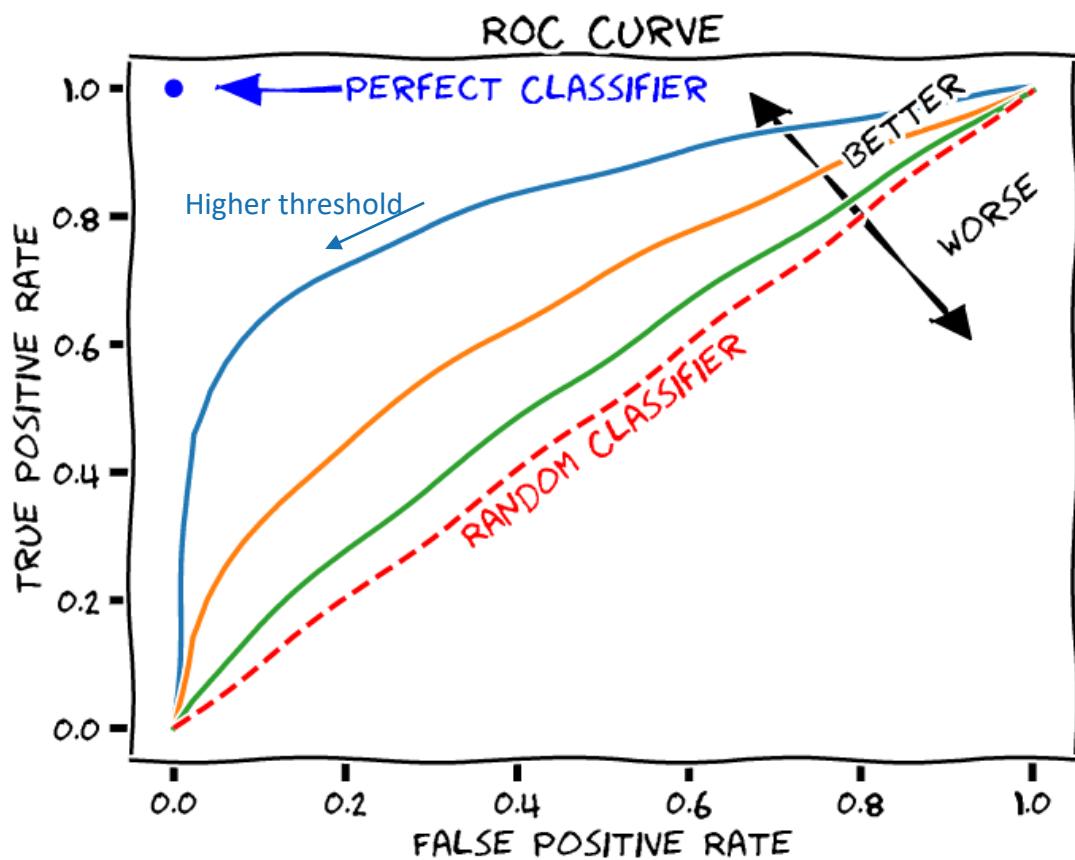
# Confusion Matrix

The confusion matrix becomes more useful the more the number of classes grows.

This allows us to immediately see the classes where we make the most mistakes in predicting and which are most *confused*.



# ROC - Receiver Operating Curves



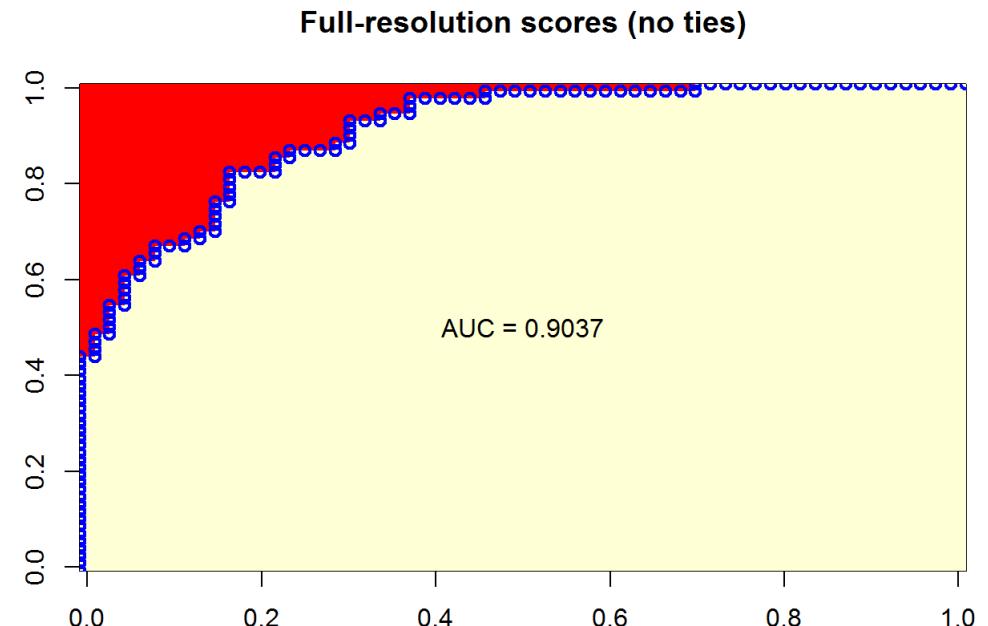
The ROC curve allows us to

- Compare different classifiers
- Estimate the false positive/negative rate as a function of the thresholds we set to classify

## AUC – Area Under Curve

As the name suggests, AUC is a metric for comparing different models by summarizing the ROC in a single number.

The greater the area under the curve, the greater the performance of the model.



# Classification Metrics

Let's look at some of these metrics in an example

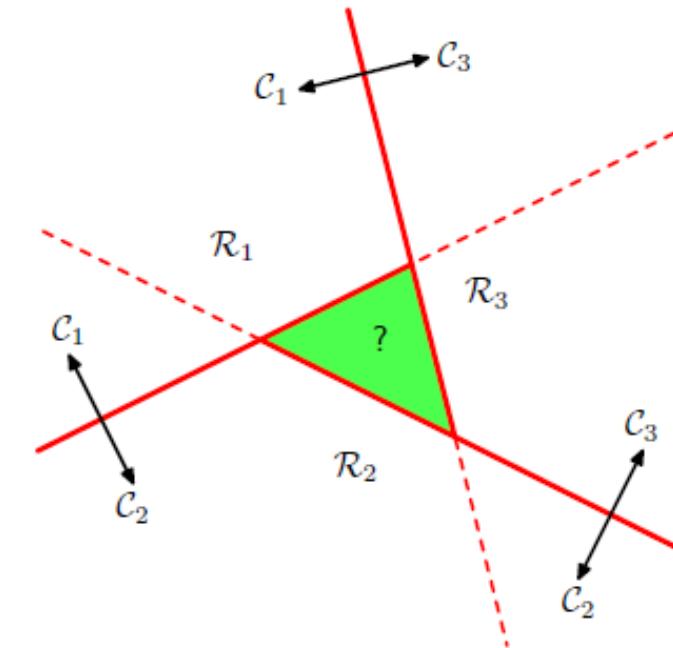
0401\_classification

[https://colab.research.google.com/drive/1\\_Rt6aEGC63LyZLGNE8NyaCph939nni1s#scrollTo=43XuLeYqU5ep](https://colab.research.google.com/drive/1_Rt6aEGC63LyZLGNE8NyaCph939nni1s#scrollTo=43XuLeYqU5ep)

## Binary classification

Almost all classification problems can be interpreted as one-vs-rest and therefore binary classification problems. If I only have two classes, one model is enough, if I have N classes it will have to train N classifiers.

A parametric model searches for the hyperplane that allows classes to be divided between each other so that the distances of the points from the plane (or decision surface) are maximum.



## Binary classification

Just like in regression problems, for linear classification models, we will look for a formula

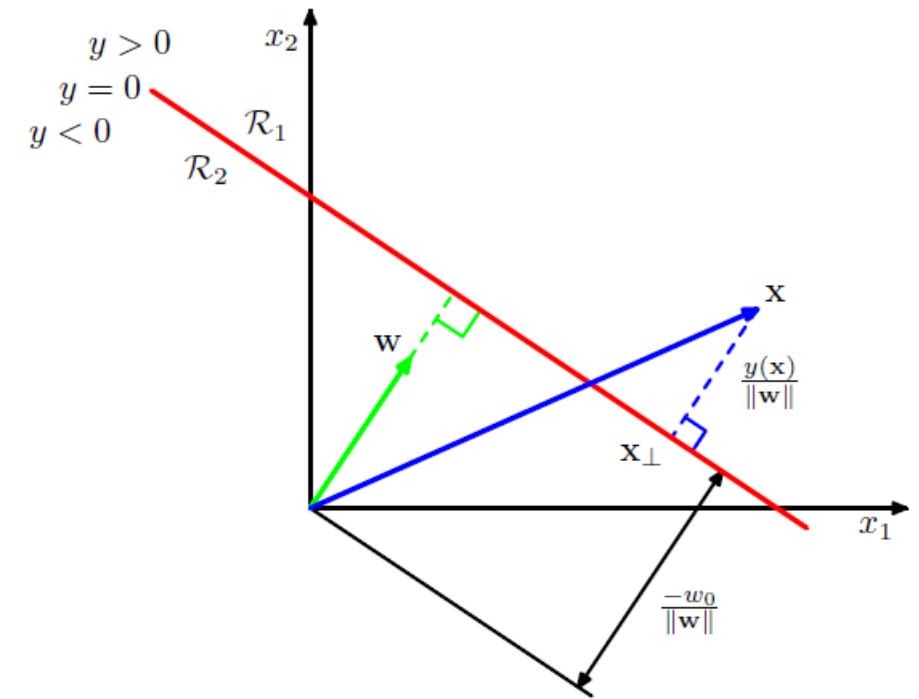
$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

that describes the decision surface.

We will assign class A if  $y(x) \geq 0$ ,

to B otherwise.

The decision plan will then be described by  $y(x) = 0$



# Perceptron

In 1962 Rosenblatt defined the **perceptron**.

The perceptron is a binary classification model in which the  $x$ 's are first passed through a nonlinear transformation function (basis function) and then build a linear model.

$$y(x) = f(w^T \phi(x))$$

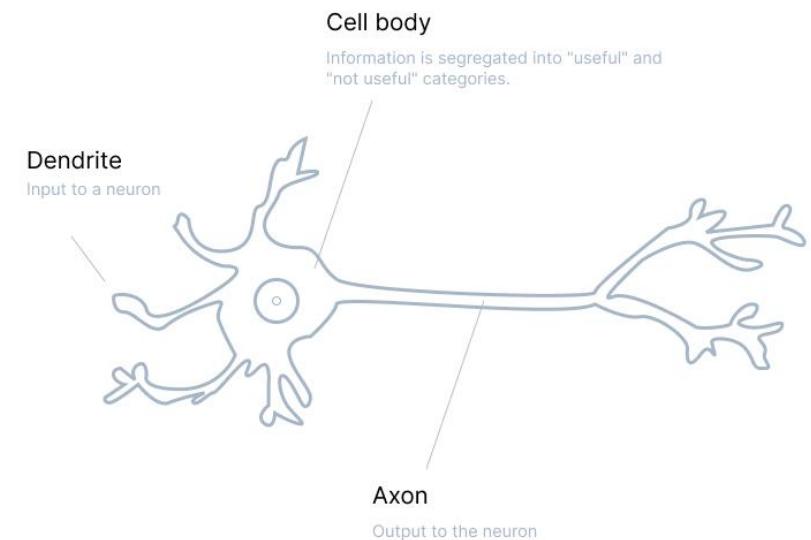
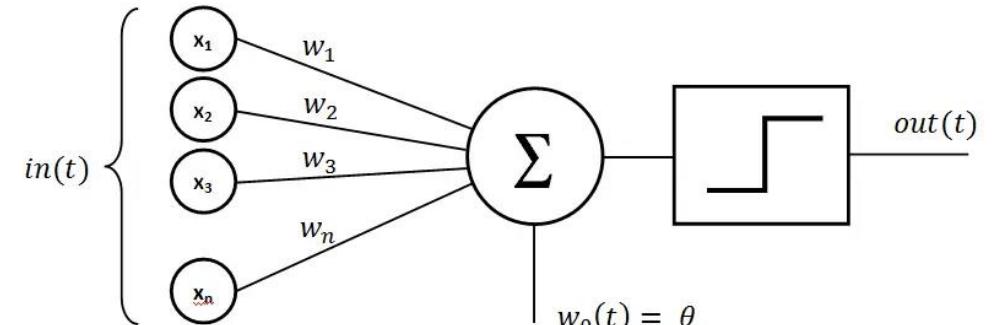
The **activation function  $f$**  is

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0. \end{cases}$$

# Neuron

Rosenblatt's perceptron is considered the first artificial neuron.

It resembles the structure of biological neurons in which inputs are taken from a series of inputs, summed across the cell, and passed on if a threshold level is reached.



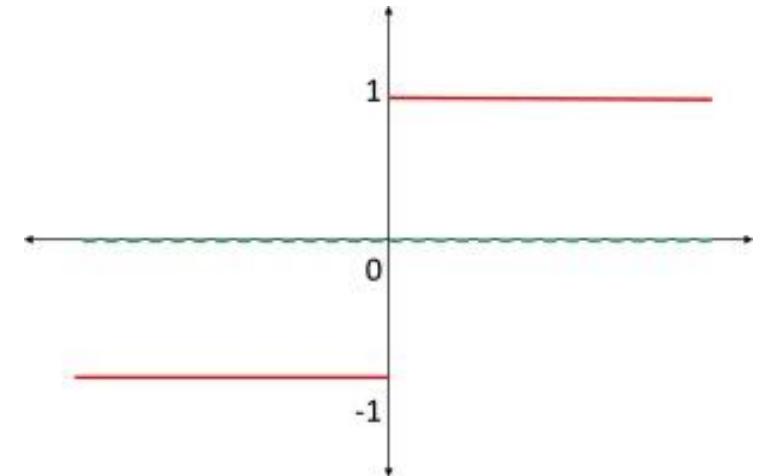
## Activation function

The activation function is the function that weights the input values to generate an output, often non-linear, result.

Perceptron activation is nothing more than sign verification.

Given a number, it returns 1 if positive, -1 if negative

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0. \end{cases}$$



## Perceptron Criterion

$$y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x}))$$

Ideally we would like to minimize the number of misclassified examples.

The problem with the perceptron is that its gradient is often zero and difficult to optimize.

You can rewrite the error quantity like this

$$E_P(\mathbf{w}) = - \sum_{n \in M} \mathbf{w}^T \phi_n t_n$$

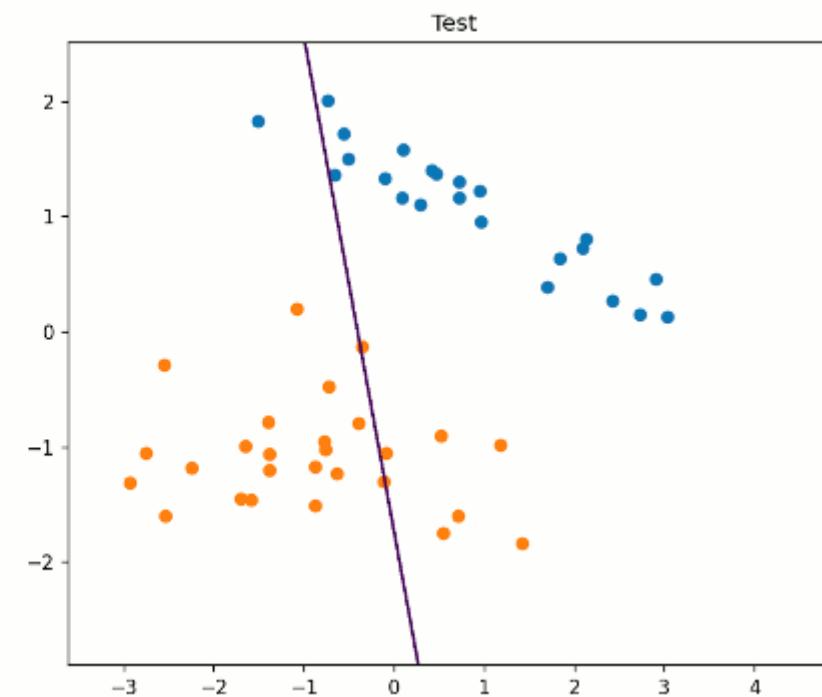
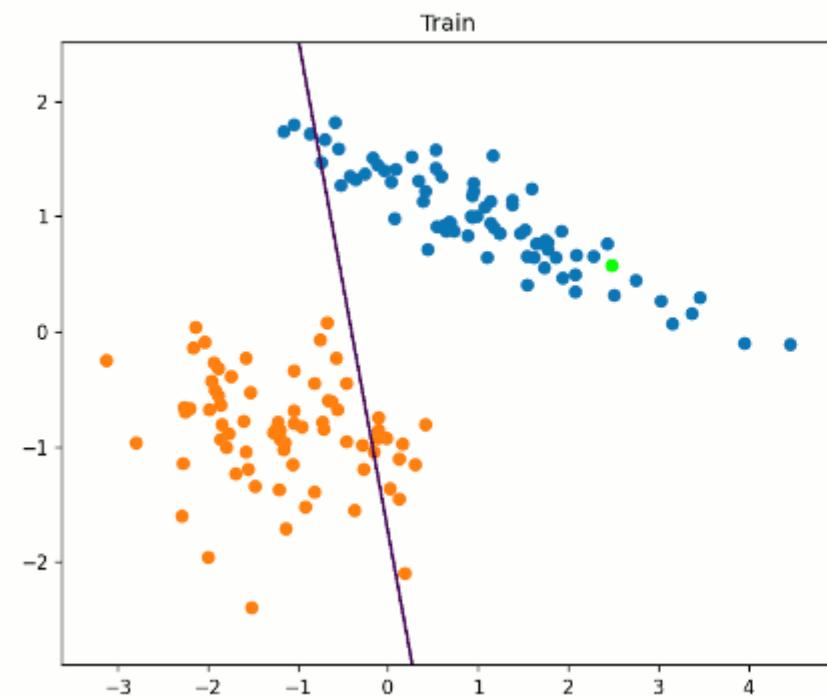
so as to search for a vector  $\mathbf{w}$  such that the examples  $\mathbf{x}$  belonging to A are  $\mathbf{w}^T \phi(\mathbf{x})$  greater than zero and the examples  $\mathbf{x}$  belonging to B are less than zero. Multiplying them by your label always results in a positive quantity in the summation for the correct examples, and a negative one for the incorrect examples.

By selecting only the incorrect ones ( $n \in M$ ) we have a continuous value to optimize.

# Perceptron

$$E_P(\mathbf{w}) = - \sum_{n \in \mathcal{M}} \mathbf{w}^T \phi_n t_n$$

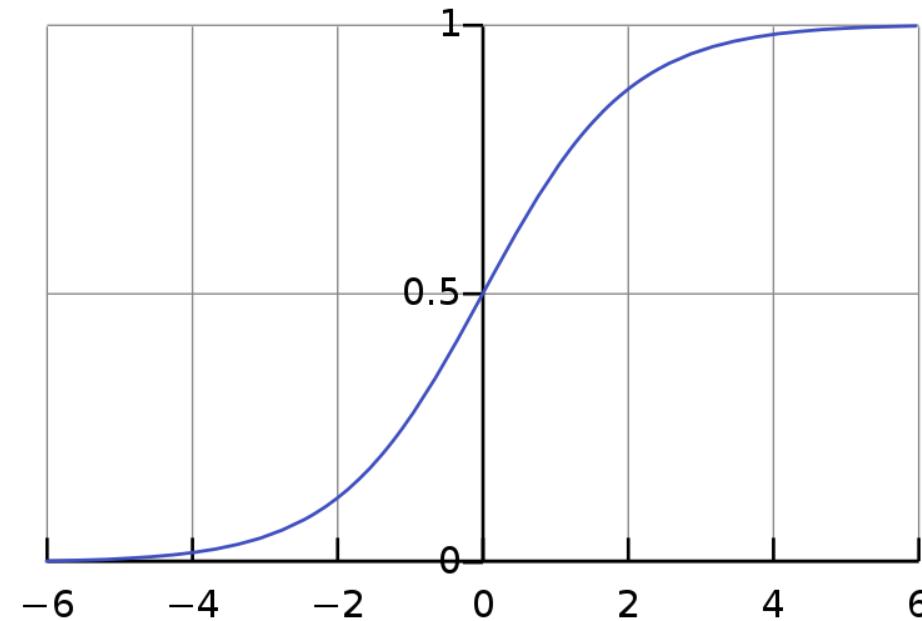
Iteration: 1/2; Point: 1/150



## Logistics Regression

**Logistic regression** ( also called logit or logreg ) is an evolution of the perceptron by changing the activation function from a heavily discontinuous function (the sign) to a continuous one called **sigmoid** and often represented with  $\sigma(\cdot)$

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



## Logistics Regression

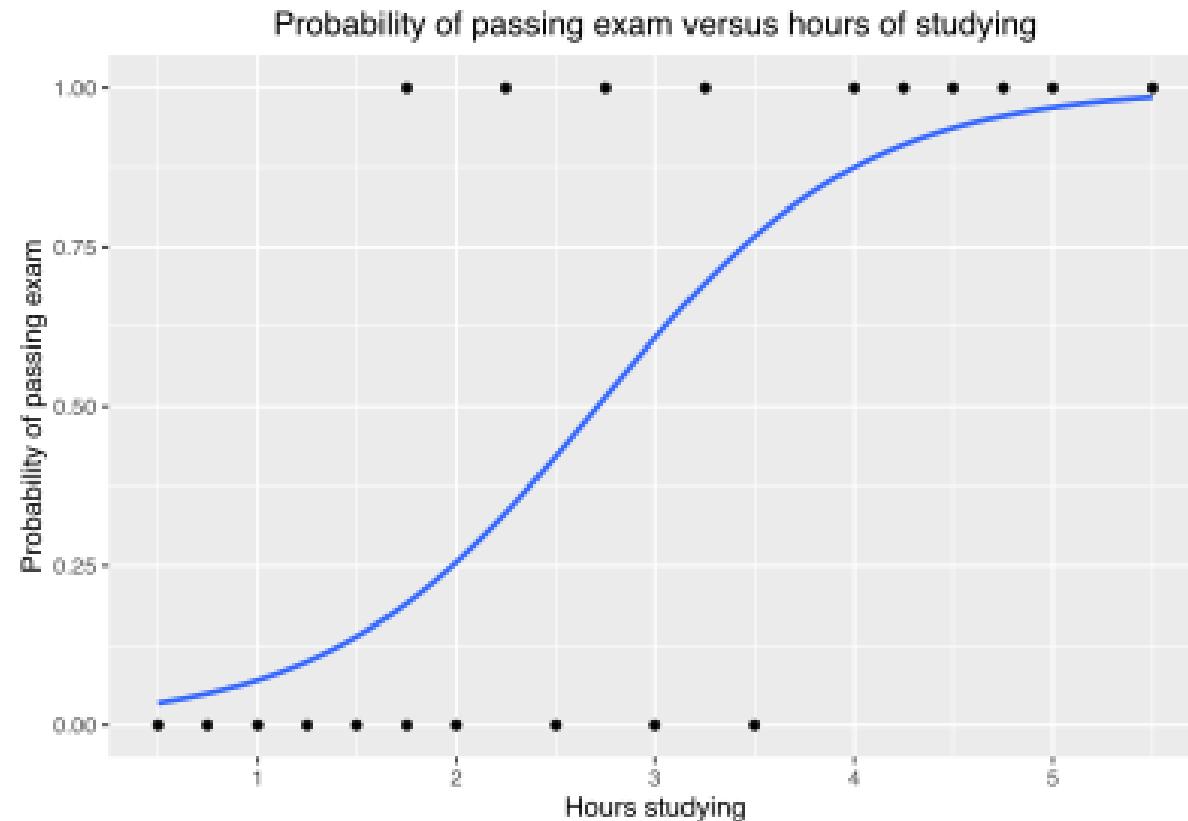
Logistic regression has many qualities, the main one being that it varies continuously between zero and one.

For this reason it is very convenient to use to represent the probability of belonging to a class.

- Probability of a failed engine
- Probability that the patient is ill
- Probability that an email is spam

Once the probability of belonging to class A has been estimated, the actual prediction occurs simply by setting a threshold value (e.g. 0.5).

## Monovariate



$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

$$p(x) = \frac{1}{1 + e^{-(x - \mu)/s}}$$

## Log loss

The logistics regression introduces a new cost function, called **Log Loss** or **Cross- Entropy** .

We can define that for the  $k$ -th point, with  $p_k$  the predicted probability, the log loss will be

$$\begin{cases} -\ln p_k & \text{if } y_k = 1, \\ -\ln(1 - p_k) & \text{if } y_k = 0. \end{cases}$$

Which is equivalent to the likelihoods of the prediction knowing the label.

If  $p_k$  it tended to zero with  $y_k = 1$  the logarithm it would grow a lot. The same applies to the opposite case tending  $p_k$  to 1. The cost will instead be almost zero if the probability follows the label.

## Log loss

Combining into a single equation we get

$$-y_k \ln p_k - (1 - y_k) \ln(1 - p_k).$$

which, summing over all the  $k$  examples of the dataset, represents our cost function to minimize.

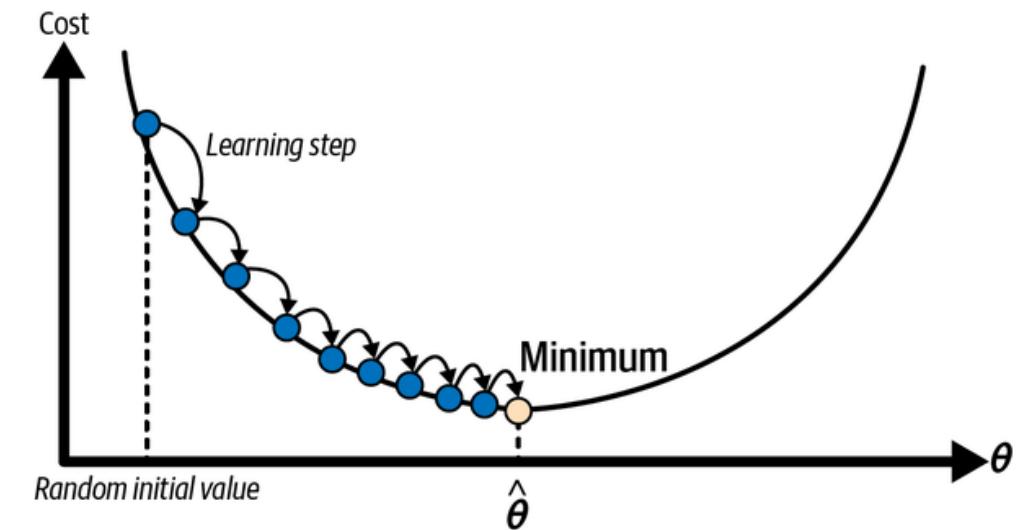
The log loss does not have a closed-form solution like the MSE in linear regression problems, but applied to logistic regression it is convex and therefore a single global optimum is always reached.

## Gradient descent

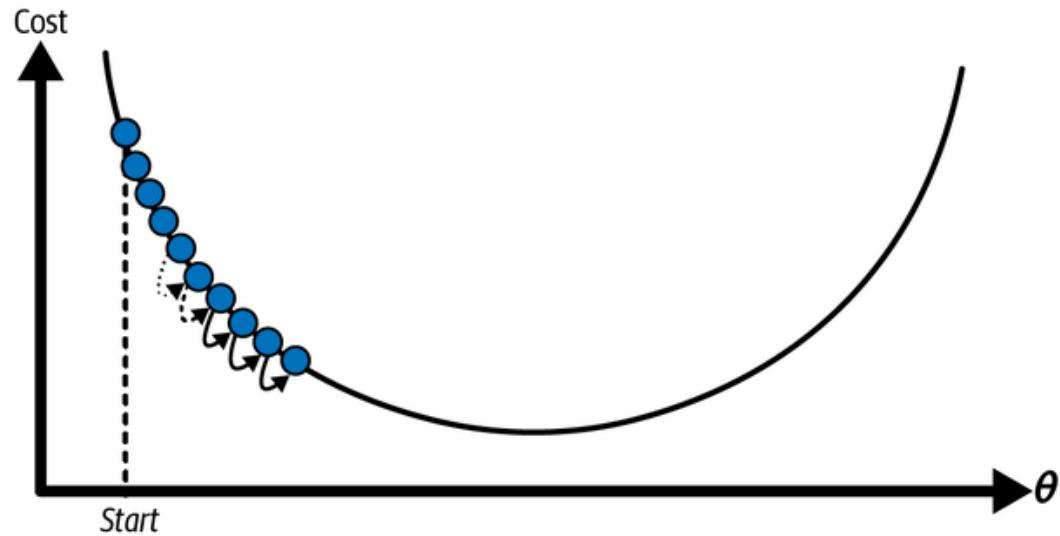
Knowing that our cost function is convex we need to move down the gradient to find the optimal parameters.

It is good practice to randomize the  $\theta$  initial parameters.

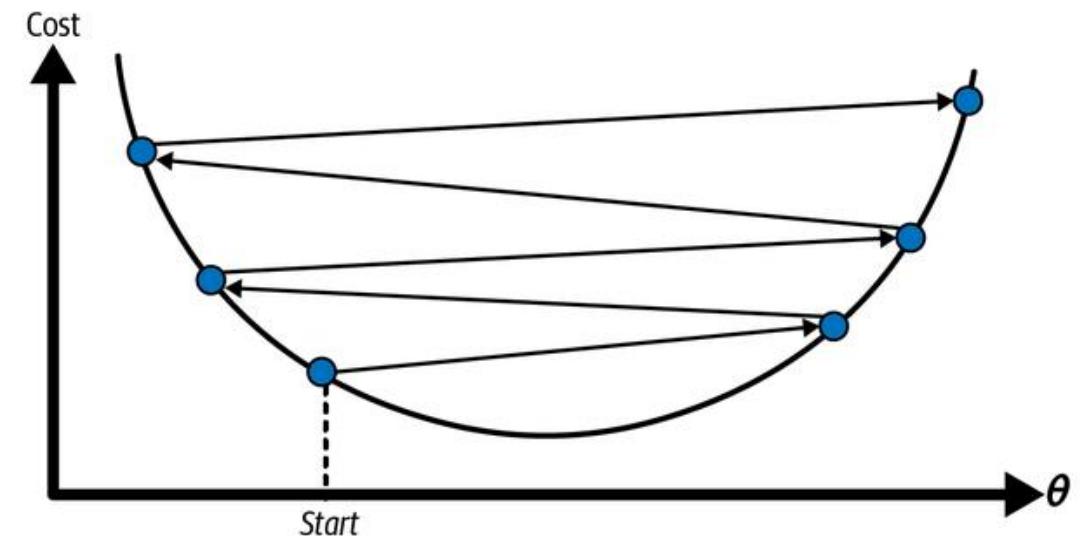
It is equally important to look for a **learning rate** suited to the problem



## Gradient descent

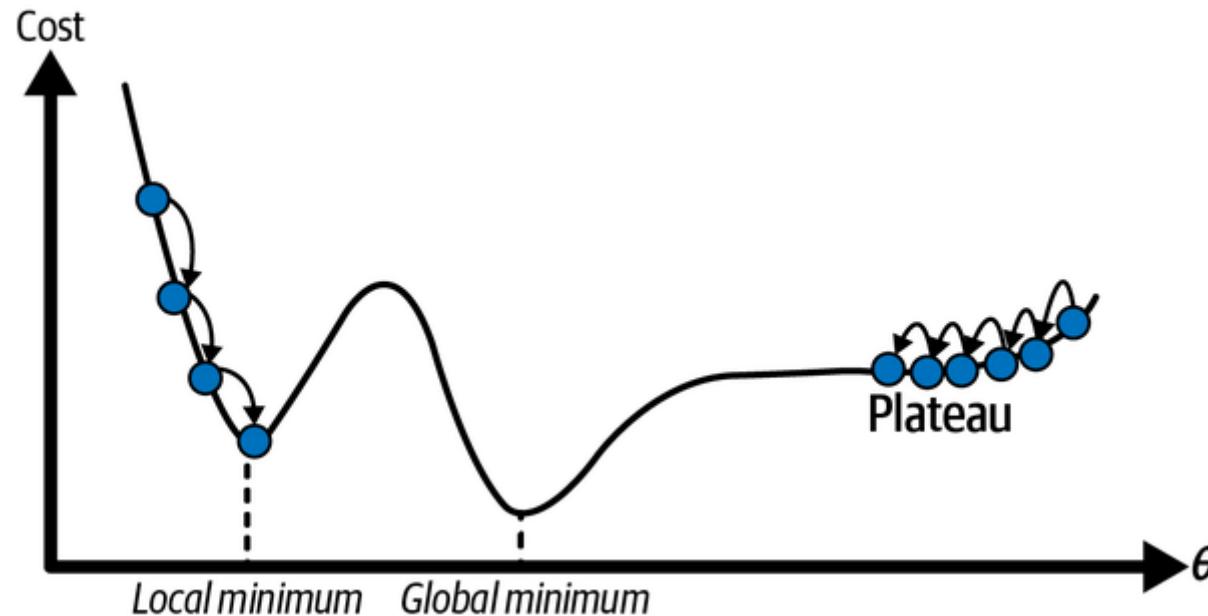


Learning rate too low



Learning rate too high

## Gradient descent



## Gradient descent

At each optimization cycle, for each example, we calculate the partial derivatives with respect to the parameters and re -update the parameters by multiplying them by the learning rate.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n$$

An entire optimization cycle on all data is called **an epoch** .

We can speed up learning by applying parameter updates not to every example but to a **batch** of data together.

The *stochastic gradient descent* updates the parameters for each point, or for a set of sampled points, while the classic one occurs by averaging all the partial derivatives. The first has several advantages, including the ability to escape local minima by always sampling the dataset for optimization in a different way.

## Gradient descent

Once the basic equations of the Logistic regression and its relative defined cost function ( for simplicity of notation , yes consider the case with a single observation , m=1)

$$\begin{aligned} z &= x * w + b \\ h_{\theta} &= \frac{1}{1 + e^{-z}} \\ cost_{(\theta,y)} &= \frac{1}{m} \sum_{j=1}^m cost_{j(\theta,y=0)} + cost_{j(\theta,y=1)} = \frac{1}{m} \sum_{j=1}^m -y_j * log(h_{\theta_j}) - (1 - y_j) * log(1 - h_{\theta_j})) \end{aligned}$$

## Gradient descent

the usual objective is to minimize this function through identification of the values optimal of  $w$  and  $b$  and to do this join the game Gradient **Descent** again .

So let's calculate the values which allow us to update ours system variables : e

$$\frac{\partial \text{cost}}{\partial w} \quad \frac{\partial \text{cost}}{\partial b}$$

$$\frac{\partial \text{cost}_{(\theta,y=1)}}{\partial w} = \frac{-y}{h_\theta} * \frac{\partial h_\theta}{\partial z} * \frac{\partial z}{\partial w} = \frac{-y}{h_\theta} * h_\theta * (1 - h_\theta) * x = -y * (1 - h_\theta) * x$$

$$\frac{\partial \text{cost}_{(\theta,y=0)}}{\partial w} = \frac{1-y}{1-h_\theta} * \frac{\partial h_\theta}{\partial z} * \frac{\partial z}{\partial w} = \frac{1-y}{1-h_\theta} * h_\theta * (1 - h_\theta) * x = (1-y) * h_\theta * x$$

$$\begin{aligned}\frac{\partial \text{cost}_{(\theta,y)}}{\partial w} &= \frac{\partial \text{cost}_{(\theta,y=1)}}{\partial w} + \frac{\partial \text{cost}_{(\theta,y=0)}}{\partial w} = -y * (1 - h_\theta) * x + (1-y) * h_\theta * x \\ &= x * (-y + y * h_\theta + h_\theta - y * h_\theta) = x * (h_\theta - y)\end{aligned}$$

## Gradient descent

$$\frac{\partial \text{cost}_{(\theta,y=1)}}{\partial b} = \frac{-y}{h_\theta} * \frac{\partial h_\theta}{\partial z} * \frac{\partial z}{\partial b} = \frac{-y}{h_\theta} * h_\theta * (1 - h_\theta) * 1 = -y * (1 - h_\theta)$$

$$\frac{\partial \text{cost}_{(\theta,y=0)}}{\partial b} = \frac{1-y}{1-h_\theta} * \frac{\partial h_\theta}{\partial z} * \frac{\partial z}{\partial b} = \frac{1-y}{1-h_\theta} * h_\theta * (1 - h_\theta) * 1 = (1-y) * h_\theta$$

$$\begin{aligned}\frac{\partial \text{cost}_{(\theta,y)}}{\partial b} &= \frac{\partial \text{cost}_{(\theta,y=1)}}{\partial b} + \frac{\partial \text{cost}_{(\theta,y=0)}}{\partial b} = -y * (1 - h_\theta) + (1-y) * h_\theta \\ &= -y + y * h_\theta + h_\theta - y * h_\theta = h_\theta - y\end{aligned}$$

Calculated the updates to apply in the iteration current of Gradient Descent, we apply the update and then move on to the iteration next :

$$w_{i+1} = w_i - \alpha * \frac{\partial \text{cost}_{(\theta,y)_i}}{\partial w} = w_i - \alpha * (h_{\theta_i} - y) * x$$

$$b_{i+1} = b_i - \alpha * \frac{\partial \text{cost}_{(\theta,y)_i}}{\partial b} = b_i - \alpha * (h_{\theta_i} - y)$$

## Features scaling

In almost all problems it is very important to ensure that the input values have the same or similar scales.

If we had a parameter that varies by thousands of units, and a second parameter that varies by hundredths, the gradient will have much more difficulty comparing the contribution of each factor, compared to two variables that are both in the 0-1 range.

	PassengerId	Survived	Pclass	Age
count	183.000000	183.000000	183.000000	183.000000
mean	455.366120	0.672131	1.191257	35.674426
std	247.052476	0.470725	0.515187	15.643866
min	2.000000	0.000000	1.000000	0.920000
25%	263.500000	0.000000	1.000000	24.000000
50%	457.000000	1.000000	1.000000	36.000000
75%	676.000000	1.000000	1.000000	47.500000
max	890.000000	1.000000	3.000000	80.000000

## Features scaling

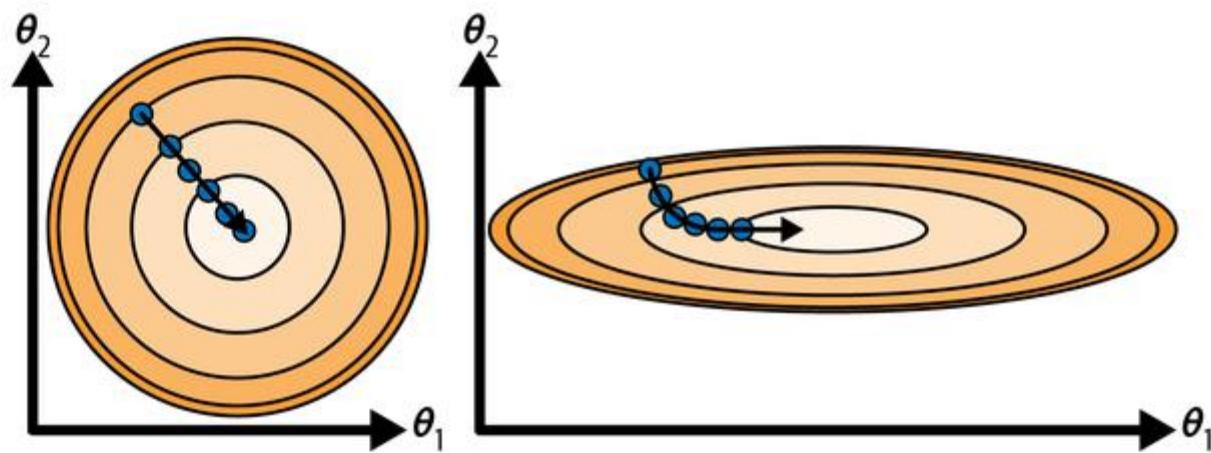


Figure 4-7. Gradient descent with (left) and without (right) feature scaling

## Features scaling

Another big advantage of features scaling is that when faced with features with the same range, the  $w_i$  multiplying coefficient  $x_{i \text{ scaled}}$  tells us its "importance".

This is only true if all the features are **correlated** with each other.

Looking at the model coefficients always gives us a good indication of what is proving to be most useful for learning purposes.

## Multiclass

multiclass case , i.e. with more than two classes to discriminate between, the single case can be reviewed as the likelihood of the class given the vector

$$\begin{aligned} p(\mathcal{C}_k | \mathbf{x}) &= \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)} \\ &= \frac{\exp(a_k)}{\sum_j \exp(a_j)} \end{aligned}$$

Also called « normalized exponential » or « **softmax** » as it reweights the outputs as a probability in relation to the other output values.

# Classification 101

Let's now take an example and apply some of the concepts seen

[https://colab.research.google.com/drive/1\\_Rt6aEGC63LyZLGNE8NyaCph939nni1s#scrollTo=BD8nqalsRmqi](https://colab.research.google.com/drive/1_Rt6aEGC63LyZLGNE8NyaCph939nni1s#scrollTo=BD8nqalsRmqi)



# **05 – Regression & Classification methods**

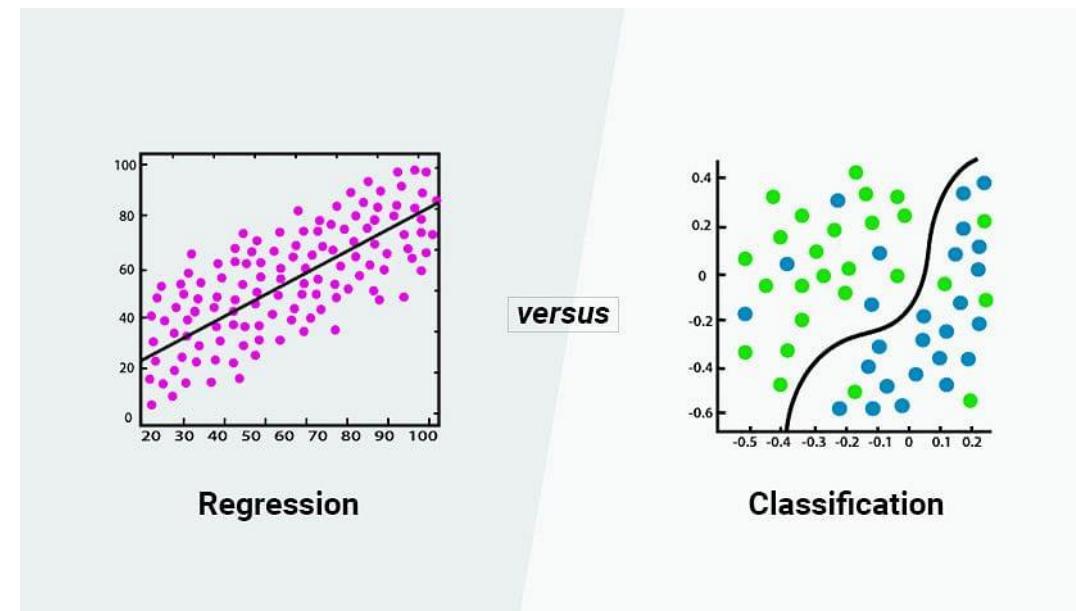
Daniele Gamba

2022/2023

# Classification

**Classification** problems are problems in which we need to determine what class an example belongs to.

Determining a cat's breed or whether a patient is healthy or sick are classification problems.



# Classification

**Binary Classification**



**Multiclass Classification**



**Multilabel Classification**



# Confusion Matrix

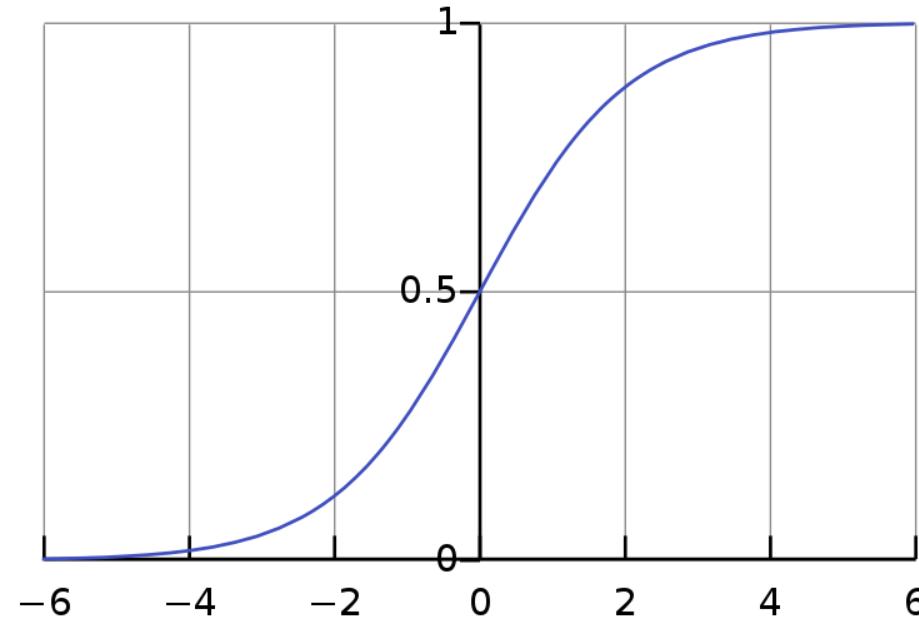
Similarly for classification problems we have performance measures.

		True condition			
Total population	Condition positive	Condition negative	Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$	
Predicted condition	Predicted condition positive	True positive, Power	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR}^+}{\text{LR}^-}$
		False negative rate (FNR), Miss rate = $\frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$	

## Logistics Regression

**Logistic regression** ( also called logit or logreg ) is an evolution of the perceptron by changing the activation function from a heavily discontinuous function (the sign) to a continuous one called **sigmoid** and often represented with  $\sigma(\cdot)$

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



## Logistics Regression

Logistic regression has many qualities, the main one being that it varies continuously between zero and one.

For this reason it is very convenient to use to represent the probability of belonging to a class.

- Probability of a failed engine
- Probability that the patient is ill
- Probability that an email is spam

Once the probability of belonging to class A has been estimated, the actual prediction occurs simply by setting a threshold value (e.g. 0.5).

## 1-vs-All

We mentioned the fact that faced with a multiclass problem classification we can solve the problem by training N 1-vs-all classifiers.

Let's see it together (and how sklearn does it automatically for us)

[https://colab.research.google.com/drive/1t0VlmhJ-cKSfNY\\_xR36oYsFOLp7cpsQf](https://colab.research.google.com/drive/1t0VlmhJ-cKSfNY_xR36oYsFOLp7cpsQf)

# Threshold values

As we have in the previous lesson, by setting different threshold values we obtain classifiers that tend to have more or fewer false positives than false negatives.

Depending on the problem we want to solve we will use threshold values that move the incorrect predictions in the direction in which we have more tolerance.

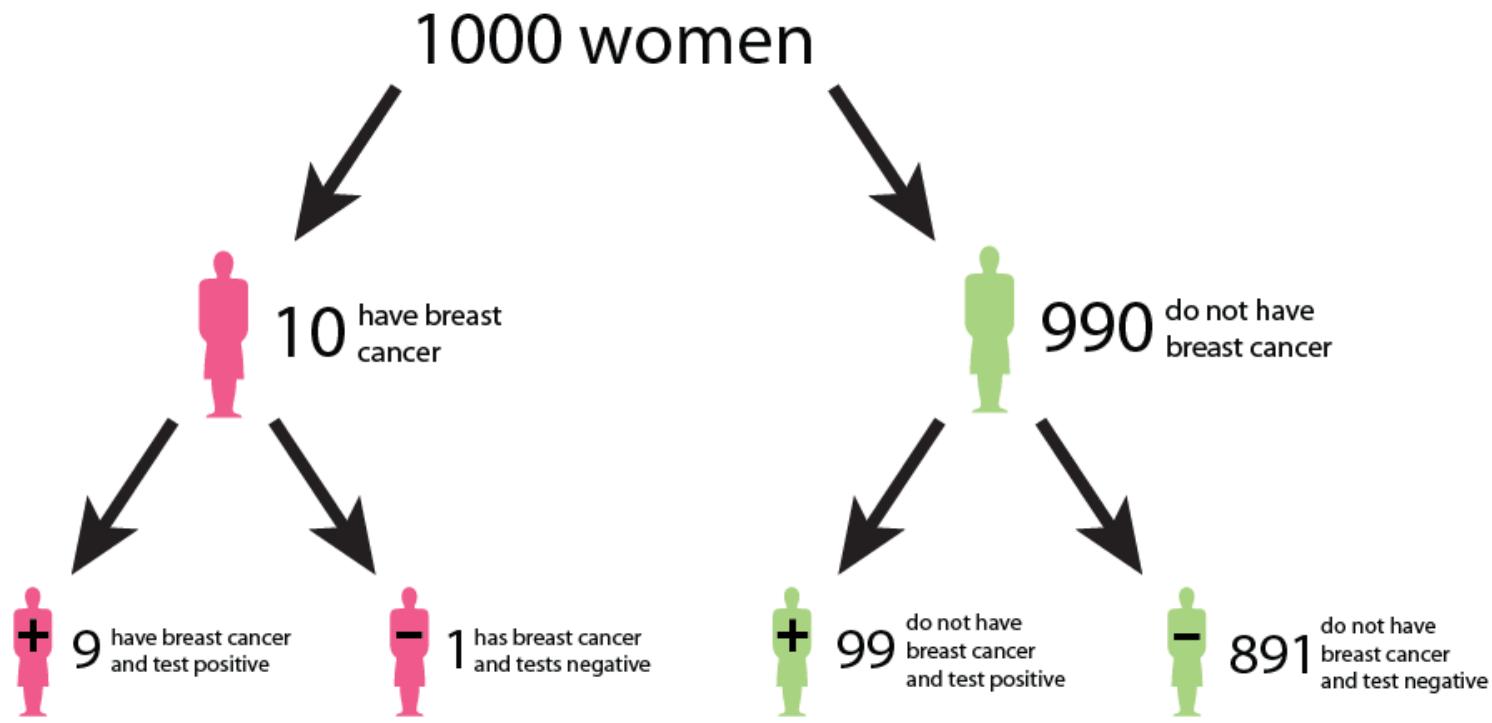
## **Diagnostic test**

We want false negatives (sick patients reported as healthy) to be virtually zero, so we will set very low threshold values (e.g. 0.1 with 1 definitely sick patient). This will have the contraindication that many patients will have false positives and will have to undergo other tests.

## **Buying shares on the stock exchange**

We want a model that tells us if it's the right time to buy a stock. In this case, associating the optimal time to buy on the stock market with 1, we will probably only want to buy the securities that the model is very sure are good by setting a high threshold value (e.g. 0.9)

## Threshold values



Breast cancer for women over 50

If diagnosed as positive (108) there is only a 1 in 10 chance of having it.

The model has 90% accuracy

## Unbalanced dataset

The example of cancer is a real condition in which the performance of the model is measured against an unbalanced dataset (ill patients are 1%).

Recognizing problems early on where the dataset is very unbalanced allows us to take them into account.

It is important to keep focused on the problem we are trying to solve and the direction in which the false positive or false negative can impact the problem.

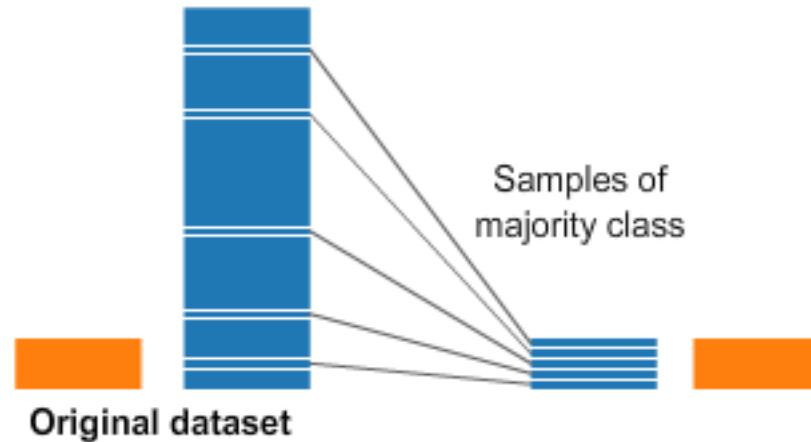
## Unbalanced dataset

To handle imbalanced datasets we can use different techniques

- **Resample the dataset**
  - By subsampling the largest class
  - Oversampling the poorest class
- Use **correct metrics in validation** to choose more balanced models (f1 or AUC instead of accuracy)
- **Weighing** the classes differently during the training phase

# Resampling

**Undersampling**



**Oversampling**



## Class weight

When reapplying the gradient to our weights to correct their value

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n$$

we can weight the  $\nabla E_n$  so that it weighs more in the less frequent classes of the dataset by multiplying it by a factor.

In this way we compensate for the fact that the model optimizes the parameters  $N$  more times on the most represented class than on the undersampled one .

## Unbalanced dataset

Let's now see an example of how to manage a very unbalanced dataset

[https://colab.research.google.com/drive/1t0VlmhJ-cKSfNY\\_xR36oYsFOLp7cpsQf](https://colab.research.google.com/drive/1t0VlmhJ-cKSfNY_xR36oYsFOLp7cpsQf)

<https://replicate.com/lambdal/text-to-pokemon>



## Classification algorithms

We have seen logistic regression as an algorithm for classifying between two (or more) classes.

There are various algorithms, let's now start to see an overview of the most used methods and their advantages.

## Naïve Bayes

The Naïve Bayes is a very simple binary classification algorithm.

It assumes that each variable independently of the others carries a probability component that the example belongs to the positive class based on Bayes' rule of conditional probability.

$$P(y | x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n | y)}{P(x_1, \dots, x_n)}$$
$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$

With the assumption of conditional independence of the second equation we can calculate

$$P(y | x_1, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i | y)$$

↓

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i | y),$$

[https://scikit-learn.org/stable/modules/naive\\_bayes.html](https://scikit-learn.org/stable/modules/naive_bayes.html)

## Naïve Bayes

There is also a version with Gaussian distributions

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The assumption of independence between the variables is what makes this algorithm naive because

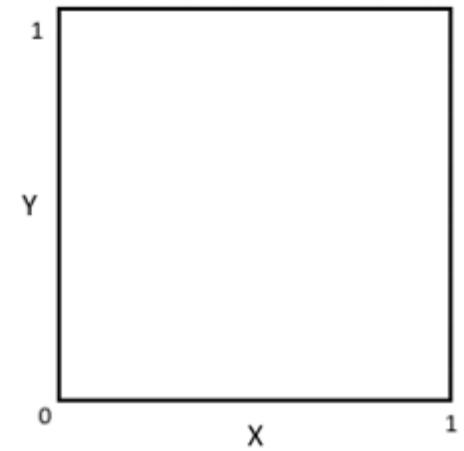
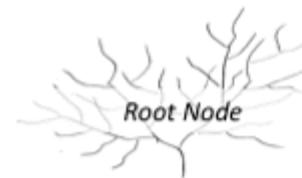
- The variables are often not independent of each other
- The variables do not all have the same impact on the outcome of the predictor

Although it is a very simple classifier and is no longer used, it has proven very useful in various cases, including spam filters.

## Decision tree

The decision tree is one of the simplest and most interpretable classification and regression algorithms.

Iteratively, a problem dimension is taken and a threshold value capable of discriminating between two groups is chosen. In this way, from the root we create a decision tree which discriminates better between the points at each step.

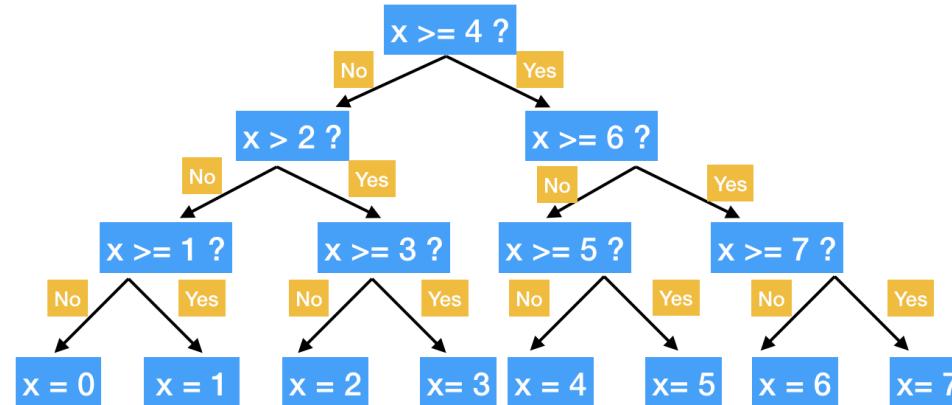
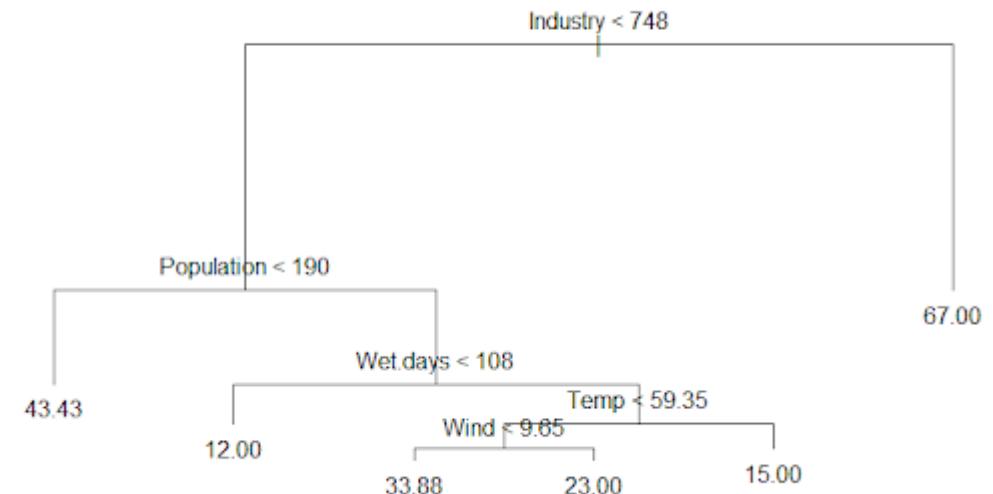


For more tutorials: annalyzin.wordpress.com

## Regression tree

In regression problems we use the same technique to discriminate between groups of data with similar values.

The final predicted value will be the average of the examples that remain in the "leaf" group of our tree.



## Decision tree

The first node on which the dataset is partitioned is the root node.

The last ones we arrive at are the leaf or terminal nodes, in the middle there are the branches.

As in the case of logistic regression, the weights indicated the importance of the ( rescaled ) features in making the decision, in the trees the first node is the one with the greatest impact, and so on for the subsequent ones.

It is much easier to visualize the decision making process of a tree than other techniques, but it is also limited in its expressiveness

- For single-value regression problems
- In general it does not reach very high performance levels

## Decision tree

regressor space (the space of possible values for the features) into  $M$  distinct regions.

For each region the prediction will be the average of the values present within.

For each region we calculate the best threshold value and a degree of "node impurity" which allows us to choose the nodes most decisive in discriminating the groups.

## Decision tree

In addition to the learning phase, we need to understand when to stop and further divide the space.

This is done with two approaches

- Put a tolerated error/impurity threshold value inside the leaf node
- Build the entire tree to the end (i.e. 1 node for each value if it cannot be done otherwise) and then intervene with "**pruning**" techniques to cut the leaf nodes with too high complexity, balancing via a regularization value

$$y_\tau = \frac{1}{N_\tau} \sum_{\mathbf{x}_n \in \mathcal{R}_\tau} t_n$$

Excellent prediction for  
the tau region

$$Q_\tau(T) = \sum_{\mathbf{x}_n \in \mathcal{R}_\tau} \{t_n - y_\tau\}^2.$$

Contribution to the  
quadratic error of the  
residuals  
(regression)

$$C(T) = \sum_{\tau=1}^{|T|} Q_\tau(T) + \lambda|T|$$

Pruning

## Gini index

The Gini index is an index which, in classification problems, indicates an impurity of the node given the region k.

We can use it both to set the threshold value at which to stop and as a criterion for pruning in classification problems

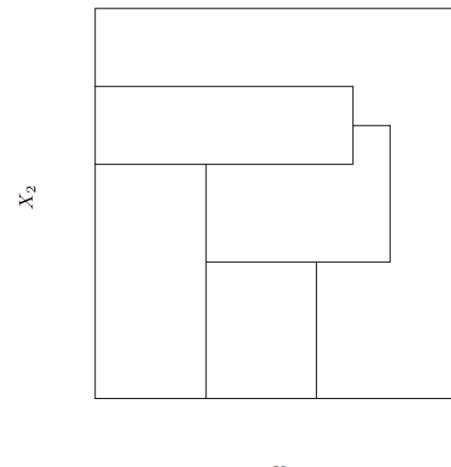
$$Q_\tau(T) = \sum_{k=1}^K p_{\tau k} (1 - p_{\tau k}).$$

with  $p$  proportion of data belonging to class  $k$  of the node.

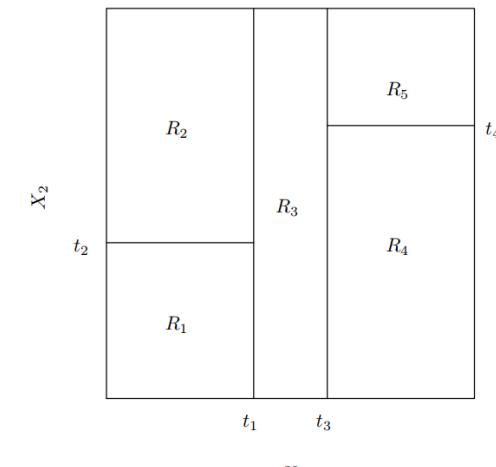
## Limits

One of the limitations of using a decision tree is that we cannot easily represent complex separation areas.

The binary decision tree can only divide areas by iterations and not by groups.



Non-divisible area



Divisible area

## Overcoming limits

Trees are limited in their expressive power but have proven extremely effective in solving simpler problems.

One of the ways to improve our performance is to not use just one model but to use N models and average their predictions. In this way we will average the output value expecting to improve performance in more complex cases.

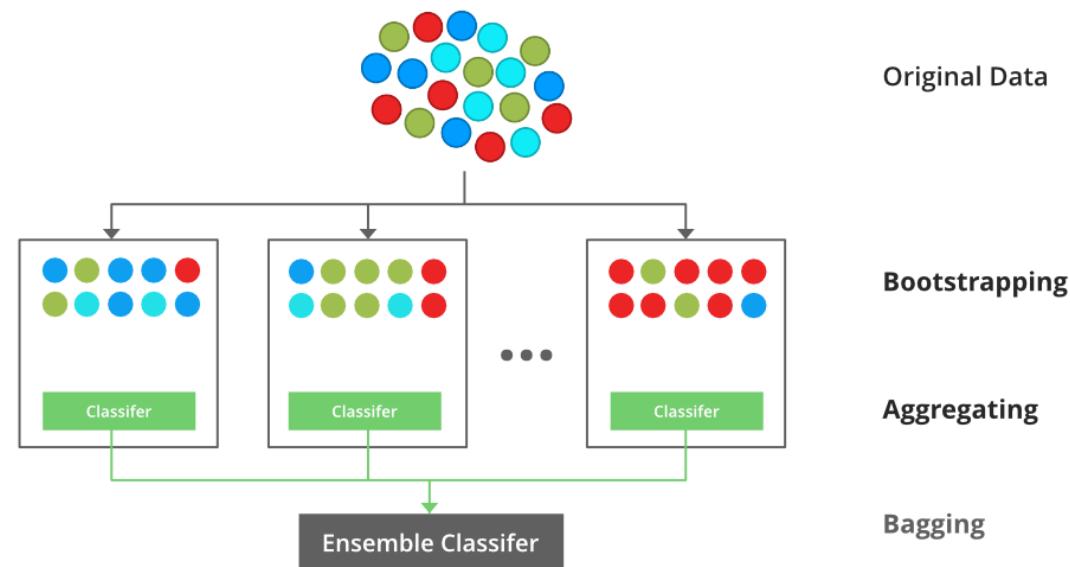
Using N predictors , the output will be:

- **regression problems** : the **average** of the predicted values
- In **classification problems** : the **majority voting** of the class

# Bagging

Bagging , or bootstrapping , is a technique whereby we train N models on N different samples of our dataset .

In this way all the models will have different predictions, we will average them in the end to obtain a single value.



## Random forest

The random forest is a technique in which we train N decisions tree and bagging it .  
N trees make a forest.

The random forest also tries to decorrelate the trees to each other.

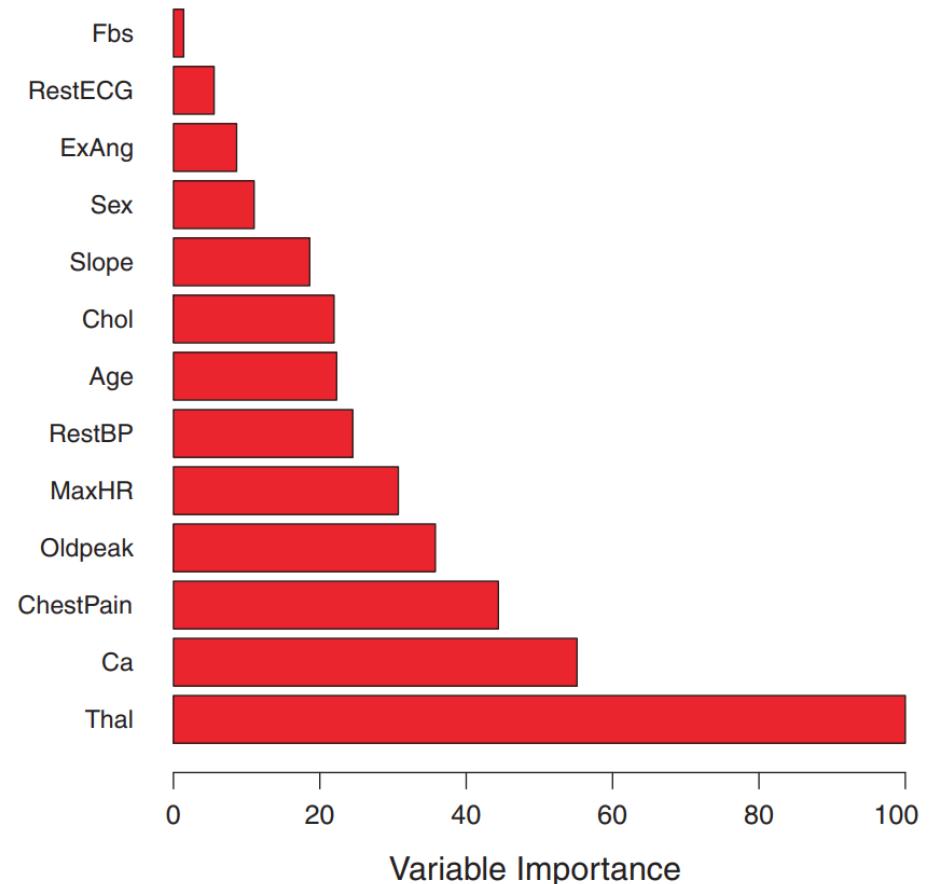
At each split it randomly selects only q regressors among the total ids and limits the split to those q regressors .

In this way the single tree is forced to take learning paths different from the others.

## Random forest

As in the previous case we can see the importance of each feature by measuring how much variance (or RSS - Sum of Squared Residuals -  $Q(t)$  on slide 20) explains each node that divides that regressor .

By summing the amount of error explained by each regressor we can make an importance ranking.



## Ensemble methods

The concept of combining different models to obtain more accurate predictions falls under the umbrella of "ensemble methods " or ensemble learning.

There are different types, including

- **Bagging** , which we saw previously
- **Boosting** , where trained models weight differently to their own performance. The more the previous classifier makes mistakes on a data set, the more they are weighted for the next one
- **Stacking** , where predictions from N models enter other models for refinement

Many times it is an ensemble with many « weak learner », compared to having a few complex models.

## Random forest

Let's take the example of the iris dataset and try to do classification with the random forest  
We will see how the performance compared to the single tree is improved incredibly.

[https://colab.research.google.com/drive/1t0VlmhJ-cKSfNY\\_xR36oYsFOLp7cpsQf](https://colab.research.google.com/drive/1t0VlmhJ-cKSfNY_xR36oYsFOLp7cpsQf)

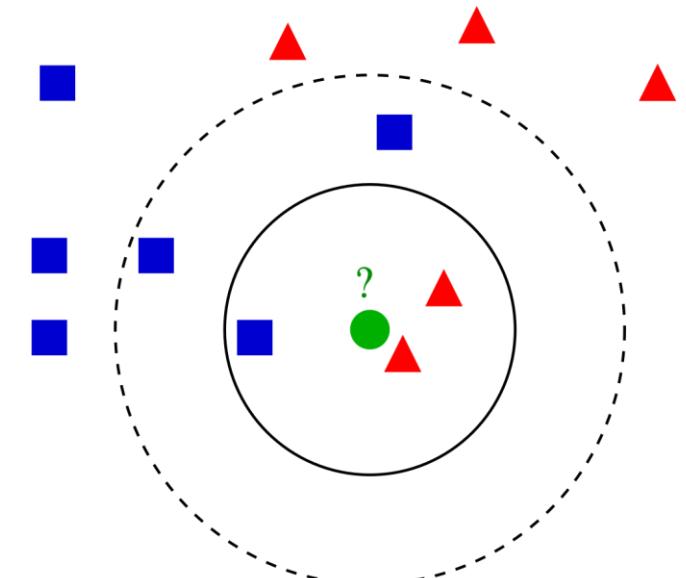
## k- Nearest Neighbor

**non-parametric** classification and regression algorithm .

When having to decide which class an example belongs to, we can compare our vector  $x$  with the vectors of other  $k$  closest examples according to a *metric* .

The decision is made by majority voting in cases of classification, as an average in cases of regression.

*K odd numbers* are chosen ,  $k=1$  indicates to take the closest.



## k- Nearest Neighbor

All examples in the dataset must be comparable to each other according to a metric called **the kernel** .

$$K(x_1, x_2) = \langle f(x_1), f(x_2) \rangle$$

The kernel function independently applies to  $x_1$  and  $x_2$  a transformation in an  $m$  -dimensional space, normally larger than the dimension of the vector.

At the end the scalar product of the two components is performed resulting in a scalar. In this way we are able to compare different examples of the dataset and sort them.

## k- Nearest Neighbor

There are numerous kernel functions depending on the comparison you want to perform between the data

Some that are often used

- Linear
- Polynomial
- Gaussian or RBF ( Radial Basis Function )
- Sigmoidal
- Cosine distance
- ...

## k- Nearest Neighbor

The main disadvantage of k-NN is its poor scalability.

Not having a parametric model of our problem, for each new example we have to re-perform as many comparisons as our labeled data.

This is all the more burdensome the larger the dataset grows

There is no training phase, it costs 0

Each inference is linear or slightly more with the length of the dataset

## Recap

We have seen

- 1-vs-all for logistic regression and how it extends with a single output model
- How does it impact our threshold value between false positives and false negatives
- How to deal with imbalanced datasets
- Started seeing a rundown of more complex algorithms for regression and classification
  - Naïve Bayes
  - Decision tree
  - Random forest & ensemble methods
  - k-NN

The next lesson we continue with parametric and non-parametric methods to conclude an overview of techniques.



# **06 – Regression & Classification methods 2**

Daniele Gamba

2022/2023

## Recap

We have seen

- 1-vs-all for logistic regression and how it extends with a single output model
- How does it impact our threshold value between false positives and false negatives
- How to deal with imbalanced datasets
- Started seeing a rundown of more complex algorithms for regression and classification
  - Naïve Bayes
  - Decision tree
  - Random forest & **ensemble methods**
  - **k-NN**

## Ensemble methods

The concept of combining different models to obtain more accurate predictions falls under the umbrella of "ensemble methods " or ensemble learning.

There are different types, including

- **Bagging** , which we saw previously
- **Boosting** , where trained models weight differently to their own performance. The more the previous classifier makes mistakes on a data set, the more they are weighted for the next one
- **Stacking** , where predictions from N models enter other models for refinement

Many times it is an ensemble with many « weak learner », compared to having a few complex models.

## AdaBoost

AdaBoosting is the implementation of the classic Boosting algorithm directly in sklearn .

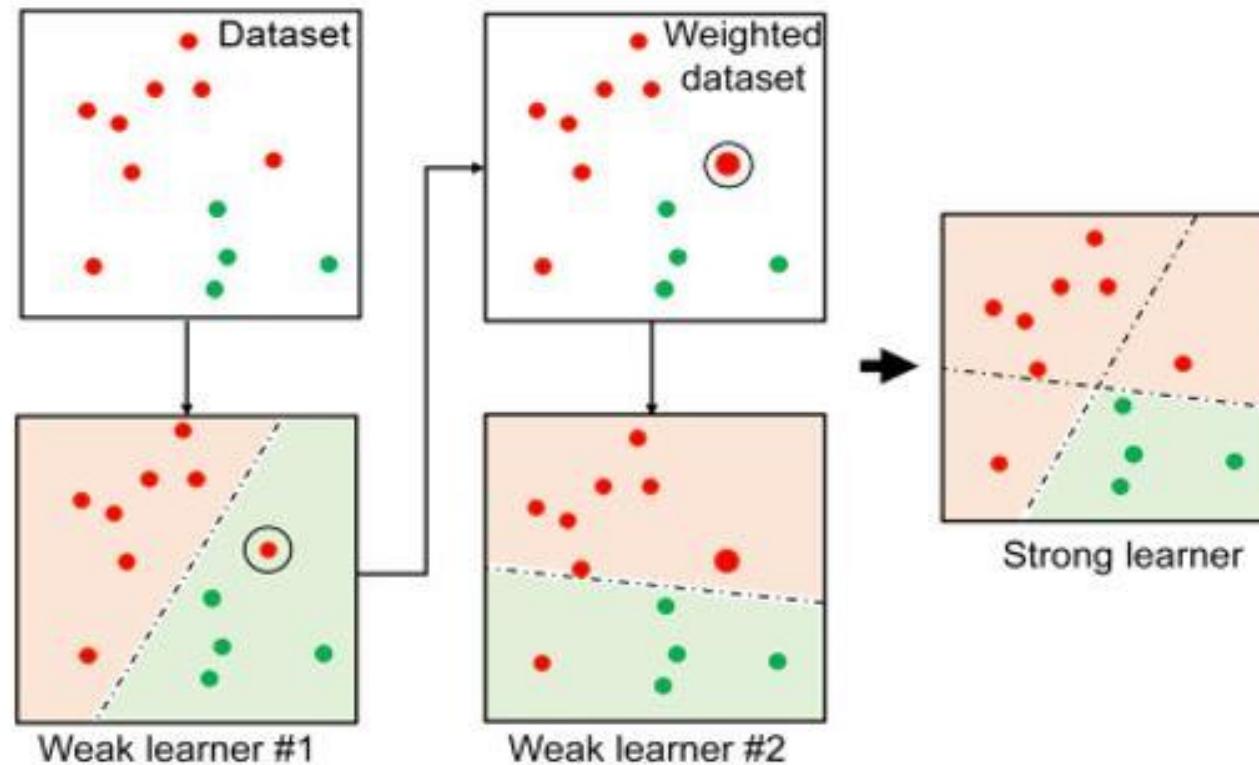
It receives as input the chosen regressor (estimator), by default a DecisionTree , and applies boosting by fitting N estimators, reweighting the datasets each time based on the examples on which it was most wrong.

`sklearn.ensemble.AdaBoostClassifier`

```
class sklearn.ensemble.AdaBoostClassifier(estimator=None, *, n_estimators=50,  
learning_rate=1.0, algorithm='SAMME.R', random_state=None, base_estimator='deprecated')
```

[\[source\]](#)

# AdaBoost



# Gradient Boosting

The Gradient Boosting predicts that the weak subsequent learners learn only the residuals of the previous estimators.

The S1 estimator will make a first estimate by predicting a value.

Instead of reweighting the dataset like AdaBoost we calculate the residuals and train a second estimator S2 to predict the residuals and so on to predict the residuals of the residuals.

In this way we build an increasingly refined estimator.

Normally estimators are decisions tree .

# Gradient Boosting

Example, let's predict *salary* with a regressor

Age	Work Exp	Degree	Salary
27	4	B.E.	60k
30	7	B.Com	50k
32	9	B.Sc	65k

We calculate the residuals from our estimate

Age	Work Exp	Degree	Salary	Predicted Salary	Residual
27	4	B.E.	60k	58k	12
30	7	B.Com	50k	58k	-8
32	9	B.Sc	65k	58k	7

# Gradient Boosting

We then train a second regressor that predicts the residuals.

Age	Work Exp	Degree	Salary	Predicted Salary	R1(Actual Residual)	R2(Predicted Residual)
27	4	B.E.	60k	58k	12	8
30	7	B.Com	50k	58k	-8	-4
32	9	B.Sc	65k	58k	7	5

The final estimate will be given by

$$\hat{y} = \hat{y}_{R1} + \hat{y}_{R2} + \hat{y}_{R3} + \dots$$

## XGBoost

and **X**treme **G**radient **B**oosting is an algorithm that extends Gradient Boosting including regularization (L1 and L2).

A famous Machine Learning package is called XGBoost and implements Gradient in a very efficient way and achieving very high performance. Boosting .

The library is known for easily scaling across distributed environments ( Hadoop , Spark, etc.), so it is particularly convenient for building models for huge datasets.



## Kernel Methods

**non-parametric** methods also called Kernel Methods as they are based on the "kernel" distance metric.

Let's start again from the K-NN slides

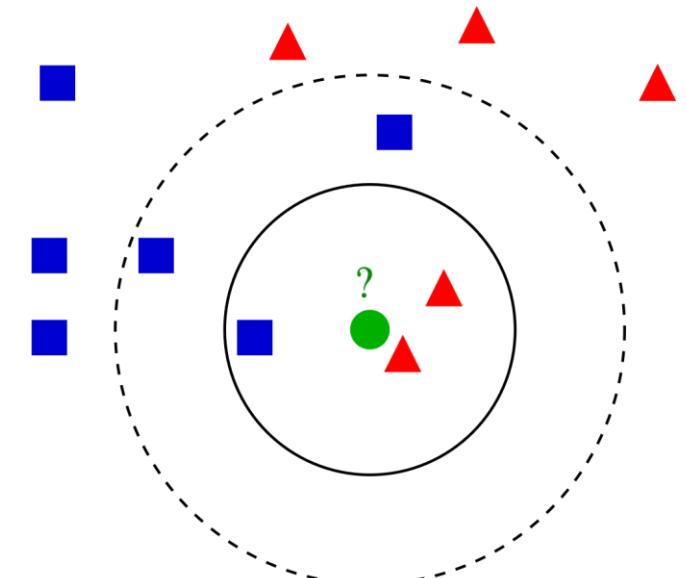
## k- Nearest Neighbor

**non-parametric** classification and regression algorithm .

When having to decide which class an example belongs to, we can compare our vector  $x$  with the vectors of other  $k$  closest examples according to a *metric* .

The decision is made by majority voting in cases of classification, as an average in cases of regression.

*K odd numbers* are chosen ,  $k=1$  indicates to take the closest.



## k- Nearest Neighbor

All examples in the dataset must be comparable to each other according to a metric called **the kernel** .

$$K(x_1, x_2) = \langle f(x_1), f(x_2) \rangle$$

The kernel function independently applies to  $x_1$  and  $x_2$  a transformation in an  $m$  -dimensional space, normally larger than the dimension of the vector.

At the end the scalar product of the two components is performed resulting in a scalar. In this way we are able to compare different examples of the dataset and sort them.

## k- Nearest Neighbor

The main disadvantage of k-NN is its poor scalability.

Not having a parametric model of our problem, for each new example we have to re-perform as many comparisons as our labeled data.

This is all the more burdensome the larger the dataset grows

There is no training phase, it costs 0

Each inference is linear or slightly more with the length of the dataset

## k-NN

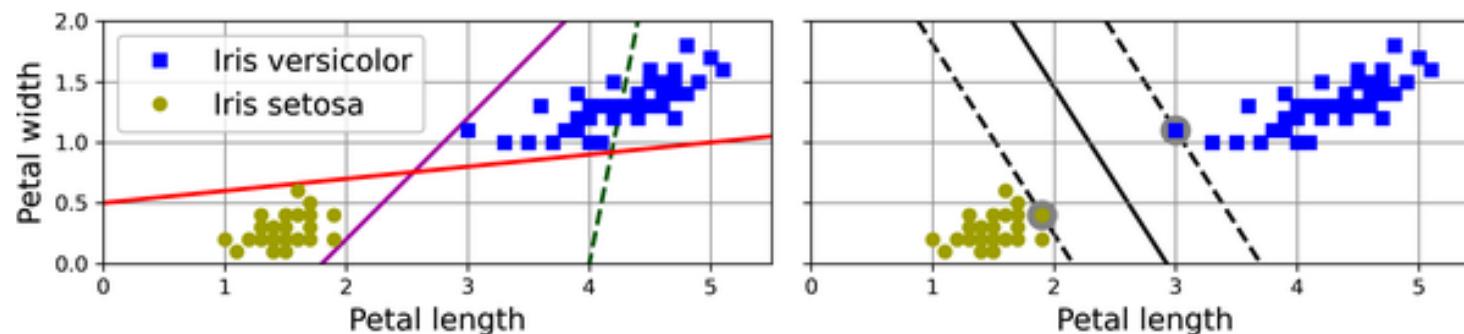
Before proceeding further, let's see concretely how it differs from a traditional model.

<https://colab.research.google.com/drive/1HzaZ7ajAkpH8Izlw9MB250VjVfwbQP2R>

# Support Vector Machines

As we have seen, one of the limits of the kNN is that in inference (in theory, unless optimized by sklearn ) it must make N comparisons.

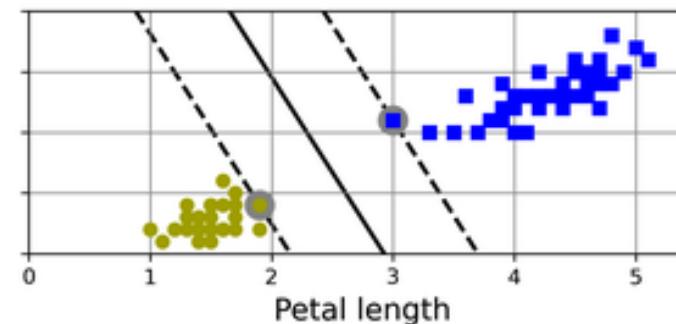
The idea behind another non-parametric technique, Support Vector Machine, is summarized in the image below. With linear classifiers we are able to distinguish the two groups, but the ideal is to find the classifier with the largest margin, also called "*large margin*". *classification*".



# Support Vector Machines

SVM is a non-parametric technique as it selects within the dataset itself those points that allow it to have the maximum margin.

Adding points in the respective groups does not change the decision boundary because it is *supported* by points on the edge, i.e. **support vectors**.

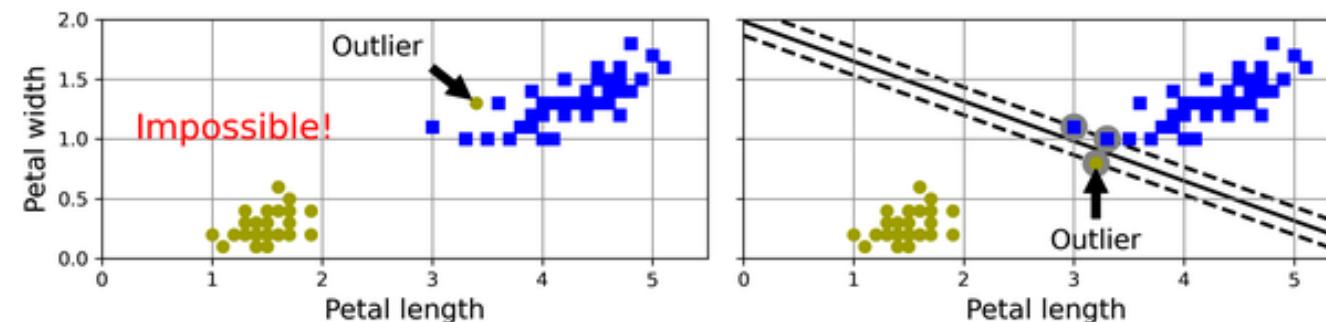


## Hard Margin vs Soft Margin

If we imposed that all training points must be correctly classified we would be imposing a **hard- margin**

What we ask is to nominate all the points in the closest areas as support vectors and therefore impose our decision boundary follows a complex path.

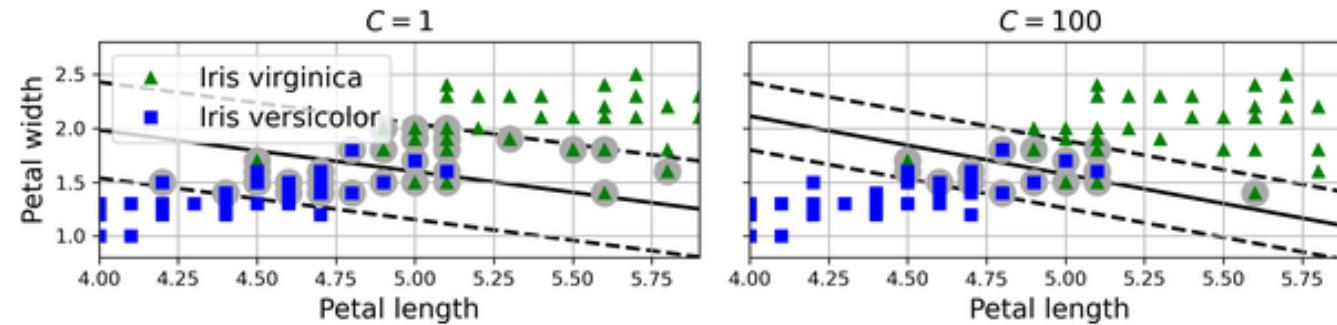
This approach is only possible if there are no outliers , otherwise points that are incorrectly classified or anomalous conditions result in decisions impossible boundaries .



## Hard Margin vs Soft Margin

In the event that we tolerated violations of the decision , even in training boundary then we treat a *soft-margin* . In the SVM this is handled via a **regularization term C**.

Lower C values allow the support area to be enlarged but lead to more violations, high values reduce the support area and the number of violations. However, by lowering it too much we risk having underfitting as we are too "tolerant".



## Support Vector Machine

Unlike the other models seen, SVM does not have `predict_proba`, as it only observes which side of the decision boundary is the point, it does not have the possibility of estimating an associated probability.

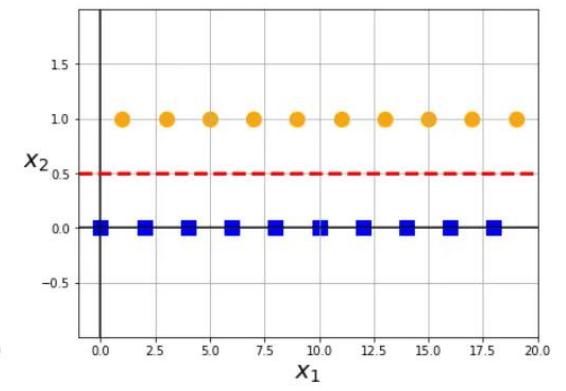
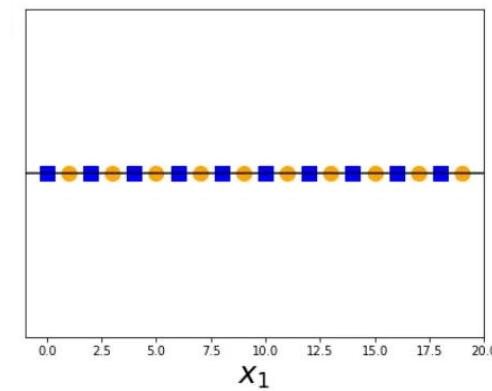
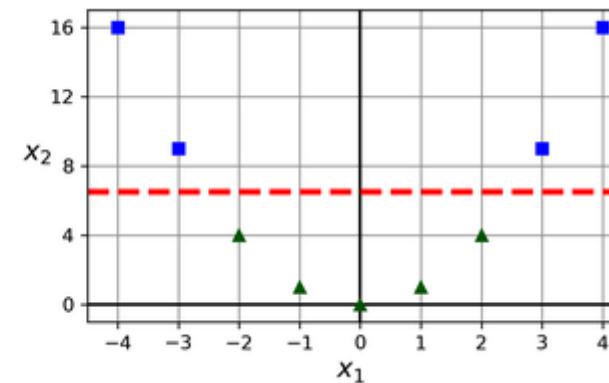
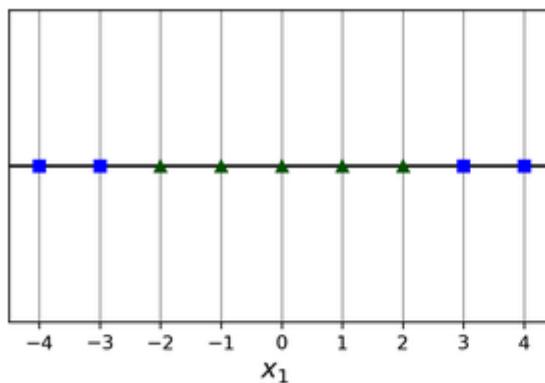
As we have seen so far, SVM tries to make a decision linear boundary to separate classes.

But how can we separate classes that are not easily separable (like iris) with SVM?

## Linearly Separable dataset

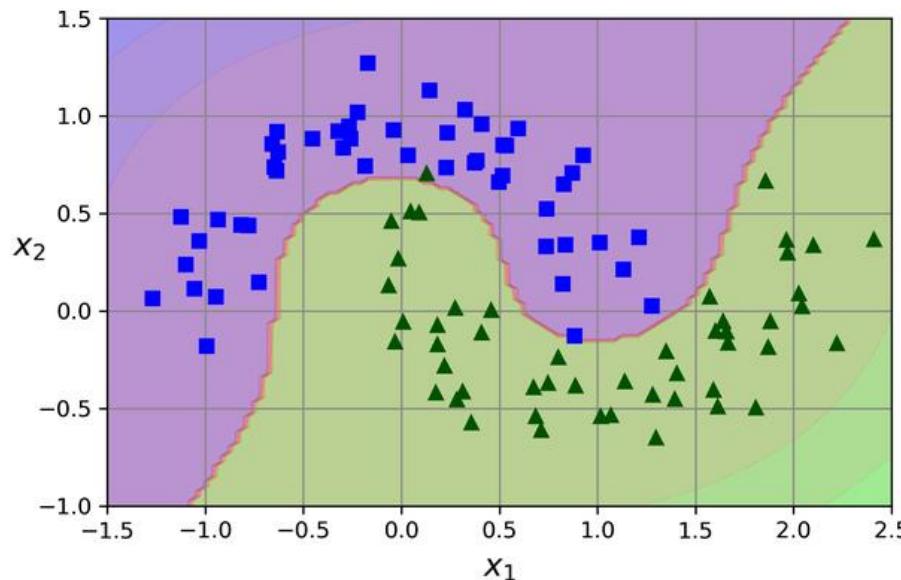
We can transform our features so that they are linearly separable.

In this case ours  $x_1$  would never be linearly separable, but if we calculated  $x_1^2$  we could easily separate the two classes. By applying basis functions we can project our feature space into a larger one in which features can be separated linearly.



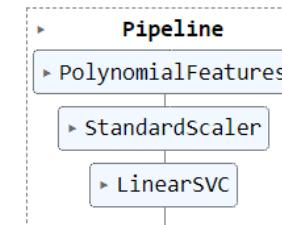
## Linear SVC

We can use SVMs to do classification with a Linear Support Vector Classifier in Sklearn . By appropriately preparing the features with a polynomial expansion we can separate even very complex shapes without worrying about the fact that there is a decision underneath linear boundary .



```
[121] X,y=make_moons(n_samples=100,noisy=0.15,random_state=42)

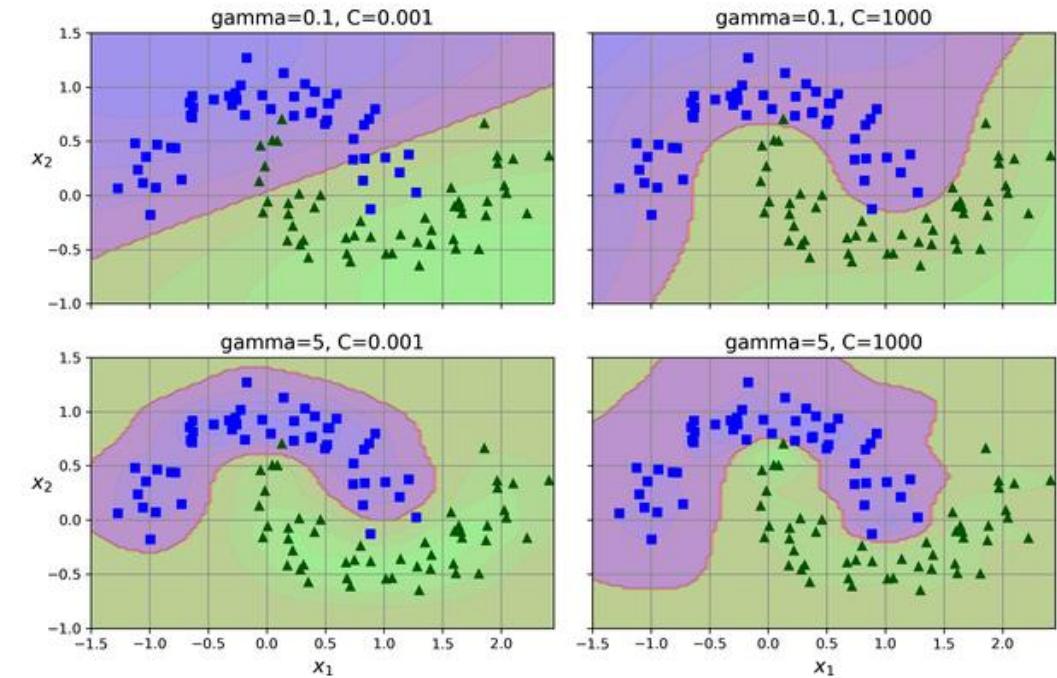
polynomial_svm_clf = make_pipeline(
    PolynomialFeatures(degree=3),
    StandardScaler(),
    LinearSVC(C=10,max_iter=10_000,random_state=42)
)
polynomial_svm_clf.fit(x,y)
```



## Non-linear SVM

We can also choose not to use a decision linear boundary .

Based on the kernels we can choose the one that is most convenient for us to solve our classification problem including ' poly ' for polynomial expansions or ' rbf ', for Gaussian kernels, each with its parameters (e.g. degree for the poly kernel )



## Kernel Trick

A key step in SVM is the kernel trick .

We have already seen that in kernel methods the distance between points is calculated by a kernel function

$$K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle$$

which applies the same transformation independently to the two examples and then performs the dot product.

If our basis functions allow us to project the points into a space where they are linearly separable we have the optimum for our problem.

## Kernel trick

Let's start with an example, let's hypothesize

$$\mathbf{x} = (x_1, x_2, x_3)^T$$
$$\mathbf{y} = (y_1, y_2, y_3)^T$$

and as a basis function the third-order polynomial expansion

$$\phi(\mathbf{x}) = (x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2)^T$$
$$\phi(\mathbf{y}) = (y_1^2, y_1y_2, y_1y_3, y_2y_1, y_2^2, y_2y_3, y_3y_1, y_3y_2, y_3^2)^T$$

multiplying them scalarly we obtain

$$\phi(\mathbf{x})^T \phi(\mathbf{y}) = \sum_{i,j=1}^3 x_i x_j y_i y_j$$

## Kernel trick

We could obtain the same result by simply performing the scalar product between  $x^T$  and  $y$

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= (\mathbf{x}^T \mathbf{y})^2 \\ &= (x_1 y_1 + x_2 y_2 + x_3 y_3)^2 \\ &= \sum_{i,j=1}^3 x_i x_j y_i y_j \end{aligned}$$

The basic functions we chose are nowhere to be seen and we got the same result, this is the kernel trick

## Other kernels

In general the kernel trick for a polynomial expansion of degree  $d$  is equivalent to

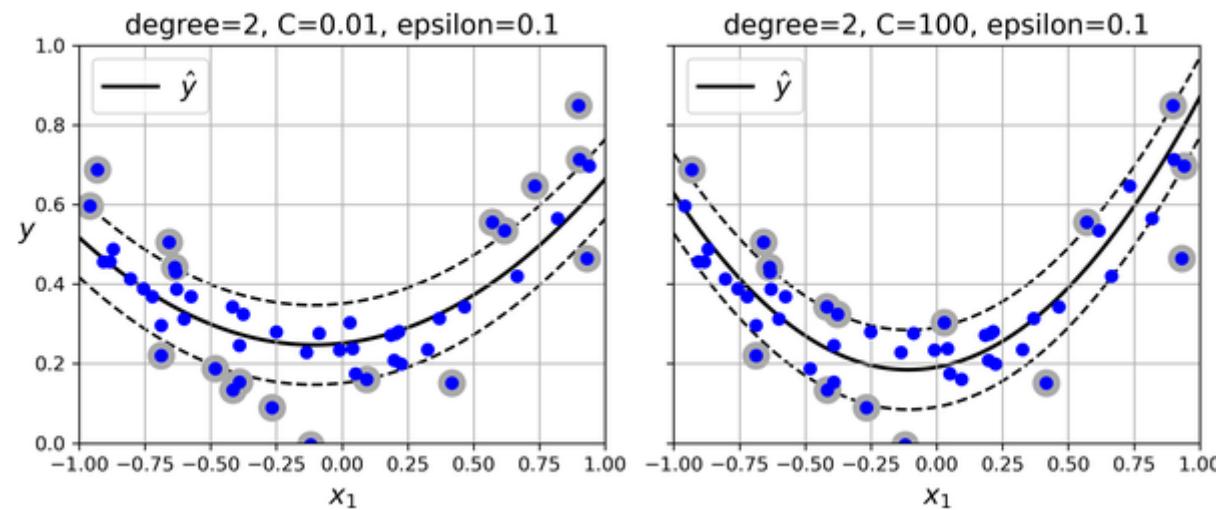
$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$$

while for RBF kernel we have

$$k(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x} - \mathbf{y}\|^2}, \gamma > 0$$

## SVM for regressions

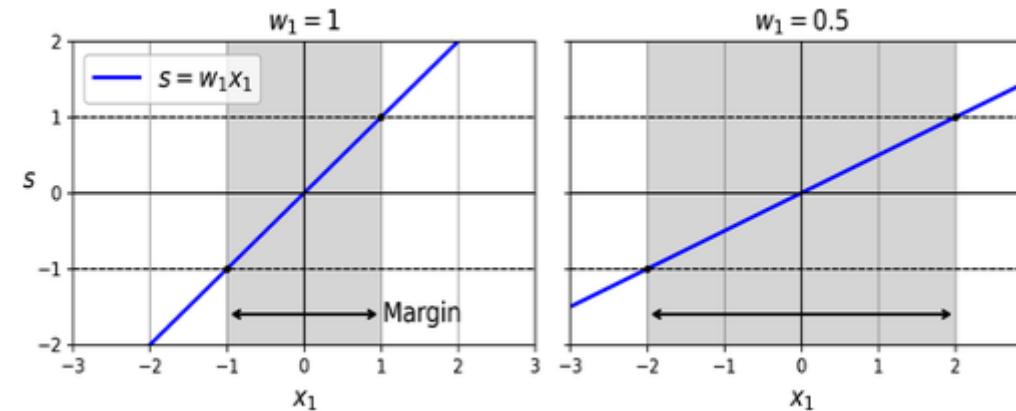
SVMs for regressions rewrite the problem slightly, instead of looking for the decision boundary that maximizes the distance, searches for the points that allow the greatest number of points to fit within the margin.



# Training

Training an SVM consists of finding the support vectors and weights that make the margin as large as possible while limiting the number of violations.

Smaller weights result in larger margins.



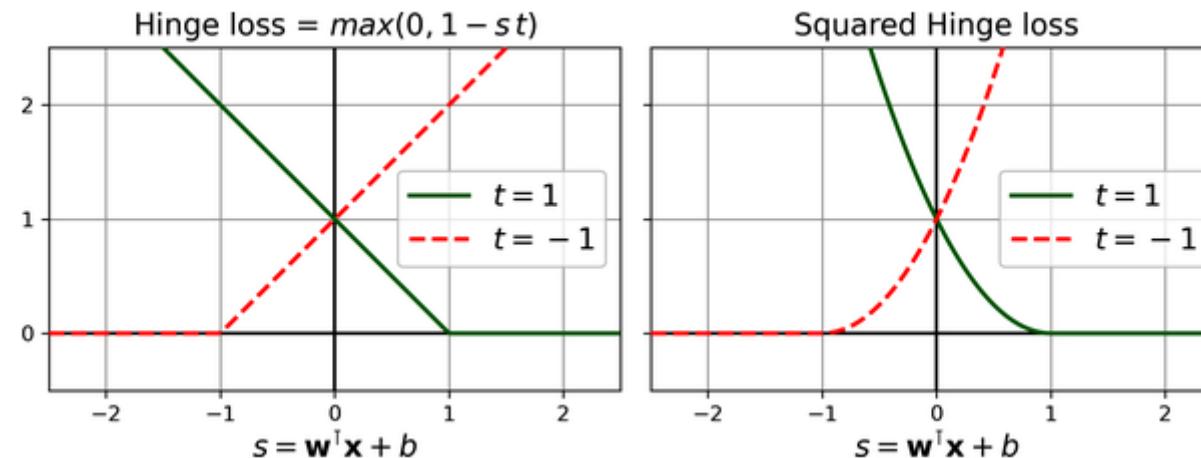
Finding the points that maximize the margin is an optimization problem that can be solved with constrained optimization algorithms or quadratic programming.

# Training

A cost function that is often used for training SVMs is Hinge Loss.

The correctly classified point does not carry any information ( loss = 0), while the incorrect ones contribute linearly to the error (or quadratically in the Squared version ) and are optimized via SGD.

$$\ell(y) = \max(0, 1 - t \cdot y)$$



## Computational complexity

The complexity of SVM, which is why it is often much slower than many other parametric techniques in training, is that learning requires numerical optimization to find the best candidates.

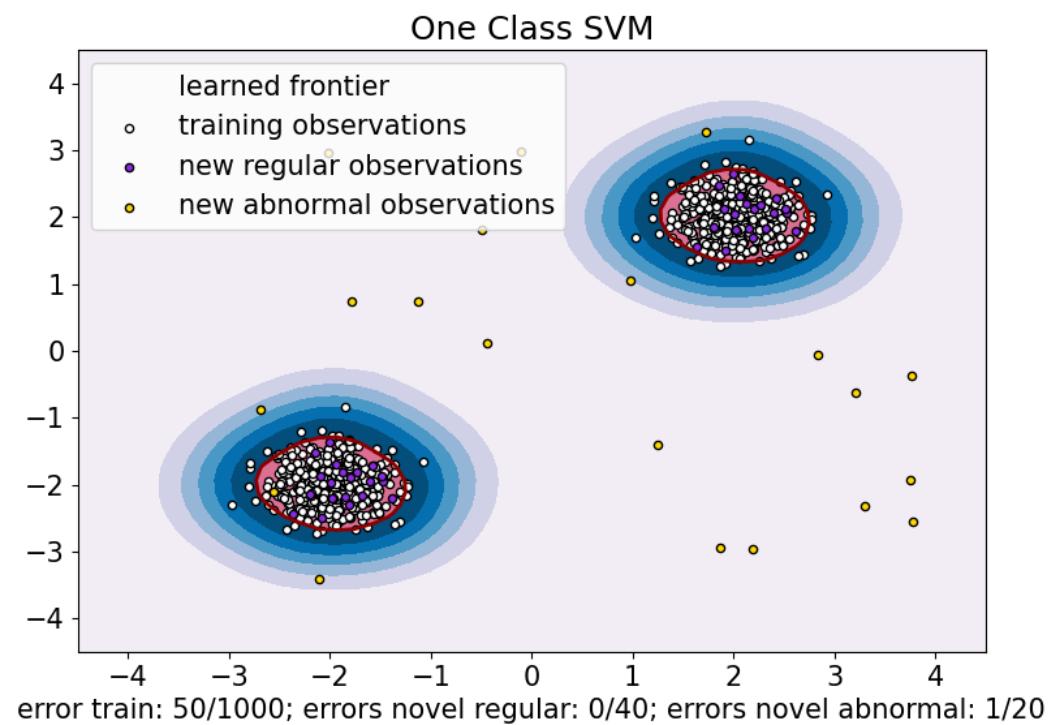
Class	Time complexity	Out-of-core support	Scaling required	Kernel trick
LinearSVC	$O(m \times n)$	No	Yes	No
SVC	$O(m^2 \times n)$ to $O(m^3 \times n)$	No	Yes	Yes
SGDClassifier	$O(m \times n)$	Yes	Yes	No

## One-Class SVM

Let's preview a concept of unsupervised methods .

SVMs can also be used to make novelties detection , in which we do not look for the maximum possible image but the minimum possible area.

The same rules apply as for SVMs so we can tolerate a number of violations.

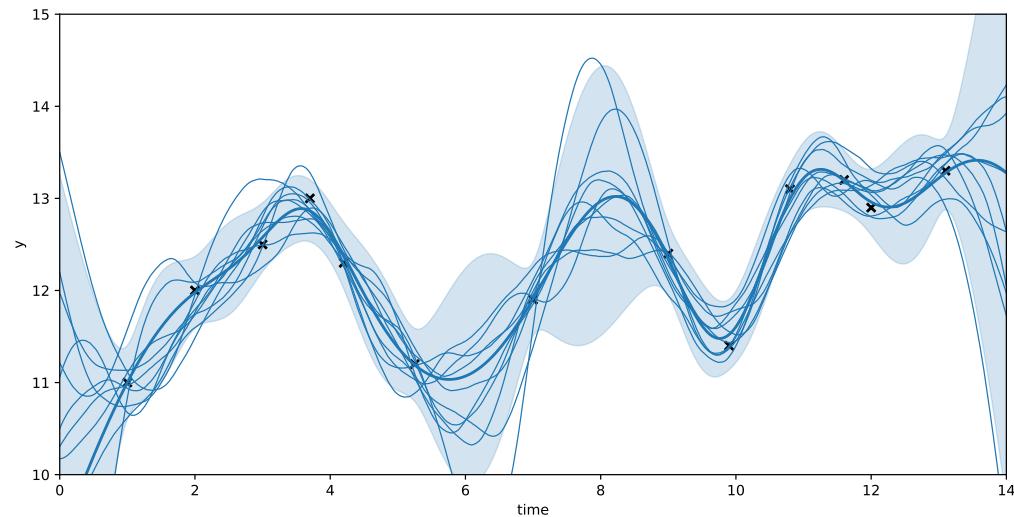


# Gaussian Processes

Gaussian Process is a non-parametric technique that interprets data as generated by a multivariate distribution described by matrices of means and covariances between all points.

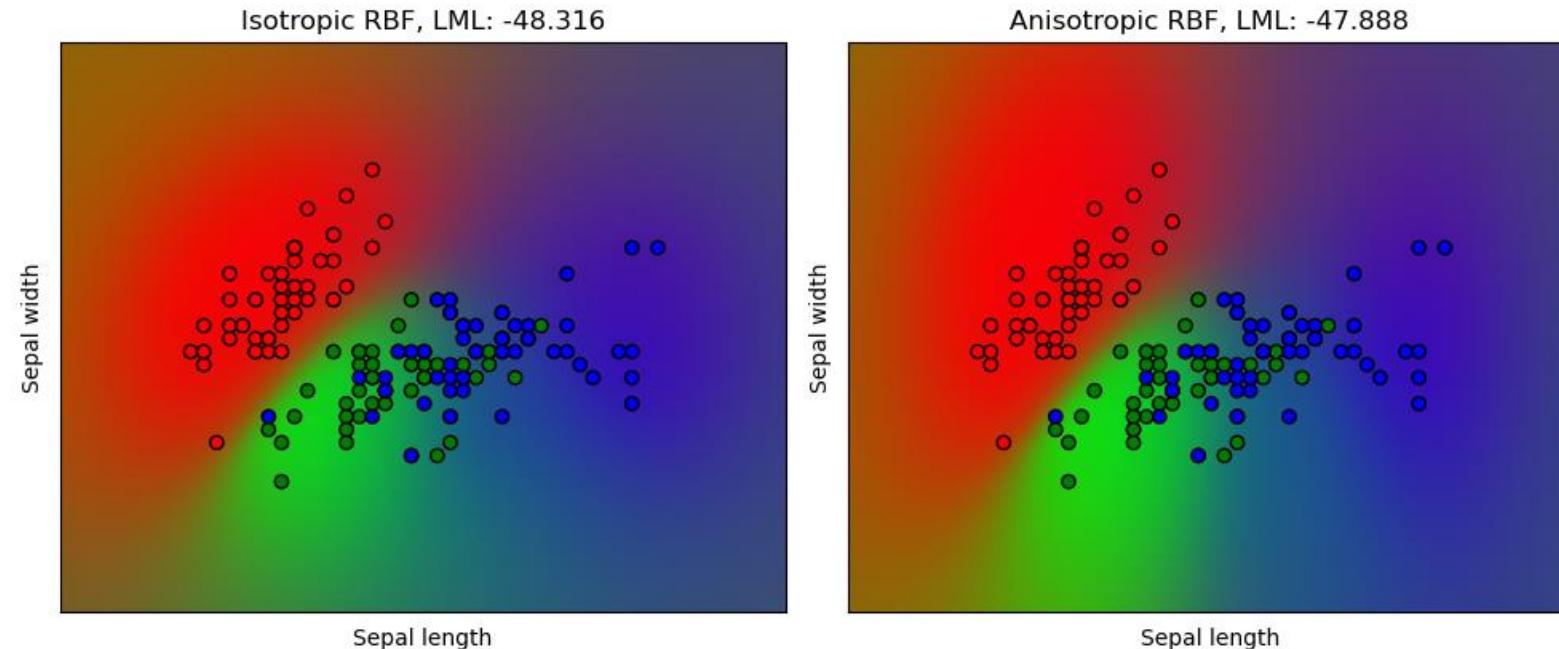
The kernel function is higher the further apart the points are. A priori knowledge about the process combined with observations allows us to estimate not only the regression but also the associated uncertainty.

It is a very expensive technique from a computational point of view which loses expressiveness in N-dimensional problems with N greater than ten.



## Gaussian Process

They are normally used in regression problems, but Gaussians Process can also be used for classification problems.



## Hyperparameter tuning

In all the models we have seen there are parameters (the weights of the features, the selected examples, ...) and hyperparameters that represent all the configurations of our regressors .

These are examples of hyperparameters

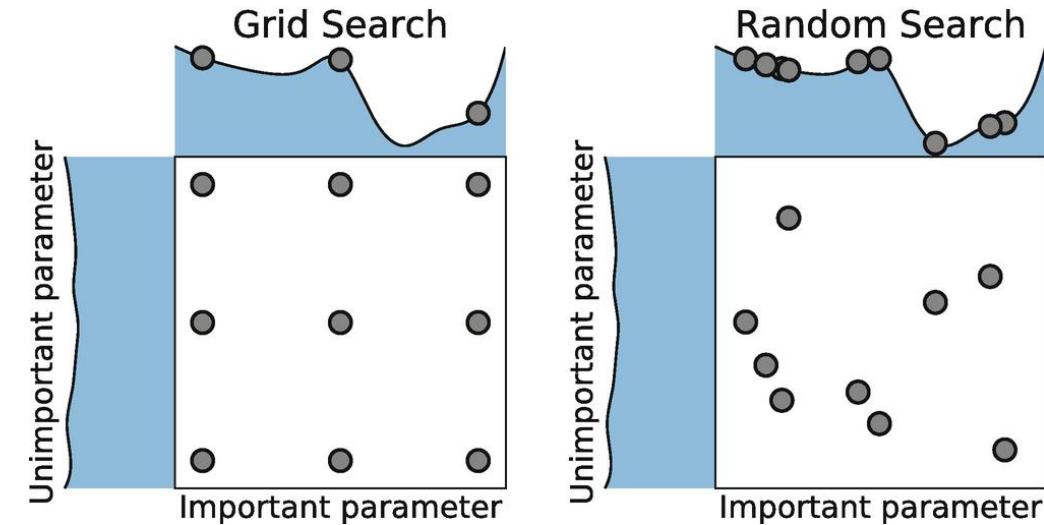
- The penalty to use and its weight in the regressions
- The depth of the tree and the impurity score in the trees
- The kernel type and degree of tolerance to violations in SVMs

All hyperparameters are largely hand -tuned by the data scientist / ML engineer based on his experience, understanding of the problem and knowledge of the technique being used. Tunando appropriately , even a less performing method can achieve the desired performance.

## Hyperparameter tuning

However, there are techniques that allow us to test different hyperparameters automatically in order to have the greatest performance in validation. These can be a basis with which to identify starting points and then proceed iteratively by hand in the search for the optimum.

An example of a technique is the **Grid Search**, creates a grid of values and performs an exhaustive search trying all combinations of set hyperparameters .



## Hyperparameter tuning

Clearly GridSearch , and all methods that allow you to test N models with M hyperparameters and K values per hyperparameter in cross validation, are computationally very expensive.

In the case of large datasets with long learning curves, reaching a good level of performance becomes an intractable problem.

In some cases we tend to sample a part of the dataset, perhaps limiting the problem only to a set of classes smaller than the total, to search in a more or less exhaustive way for the technique that potentially performs best and then apply it on the entire dataset.

Subsequently we will see in neural networks the number of possible combinations of layers , activations, kernels, neurons, etc. explodes quickly. The data scientist's sensitivity and knowledge of the problem remain the main path to achieving a good result.

## Recap

We have seen

- Ensemble methods
- Non-parametric methods
- Hyperparameter Tuning

Let's now see everything together in a regression problem and a classification problem

<https://colab.research.google.com/drive/1HzaZ7ajAkpH8Izlw9MB250VjVfwbQP2R>



# **07 – Unsupervised Learning**

Daniele Gamba

2022/2023

## Previous

We have seen

- Ensemble methods
- Non-parametric methods
- Hyperparameter Tuning

Let's now see everything together in a regression problem

<https://colab.research.google.com/drive/1HzaZ7ajAkpH8Izlw9MB250VjVfwbQP2R>

## In this lesson

### Unsupervised Learning

- Dimensionality Reduction
- Clustering (part 1)

When we deal with neural networks, always unsupervised

- Autoencoders and embeddings
- Self- supervised
- Anomaly detection

## Dimensionality reduction

We have seen that one of the problems is having enough data per parameter.

Reducing the input features allows us to have

- a greater data-parameter ratio
- greater interpretability of the results
- greater robustness of the model

On the other hand, we always certainly lose the expressiveness of the model if the features we select contain information useful for our estimate.

## Dimensionality reduction

**regularization** techniques (e.g. Lasso) allow us to **select** some more significant features automatically, there are techniques that allow us to **reduce** the dimensionality of our problem in a more or less automatic way.

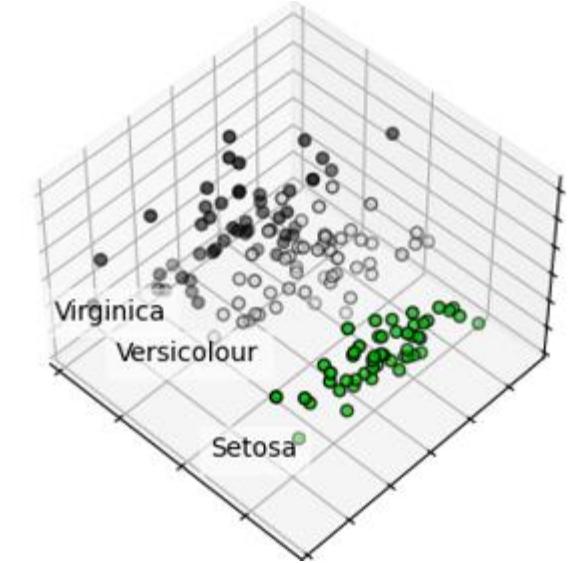
dimensionality of the problem is reduced when we want to work with fewer features or we want to bring the problem back into a viewable space (2D-3D) to understand how our regressor is behaving .

## Data visualization

In the case of our IRIS dataset problem we have 4 features, which are difficult to visualize in a single graph.

We can therefore choose to display N selections of 2 features or to project our points into a 2D space starting from all the features.

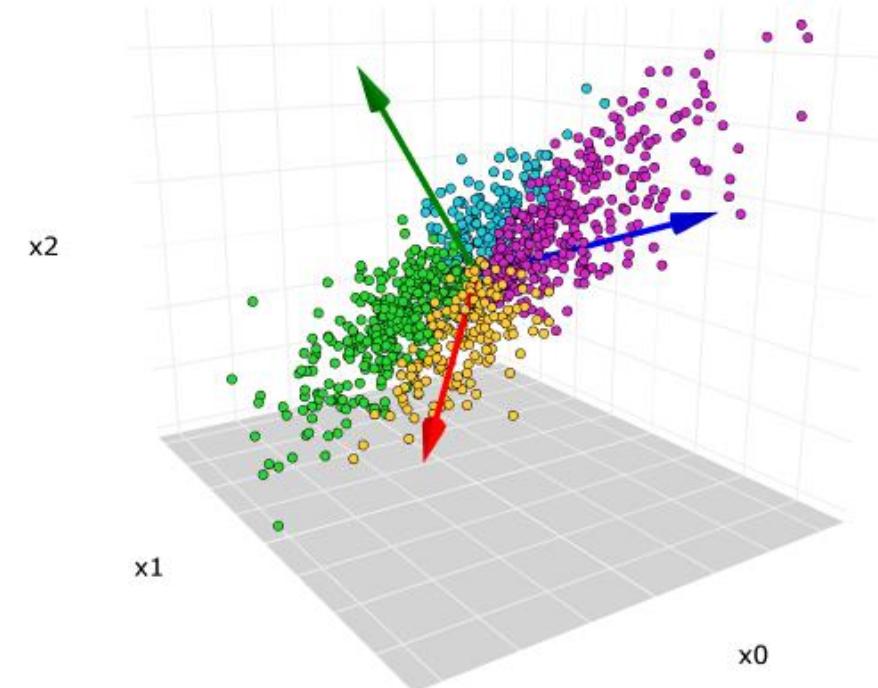
The more the problem has many features, the more necessary it becomes to find techniques that do this projection for us.



# Principal Component Analysis

PCA is a technique that **is** used to

- **Data visualization** , i.e. reporting a problem with N dimensions in 2D / 3D
- **Data compression** , i.e. projecting our points into a space with fewer dimensions losing some information (lossy compression )

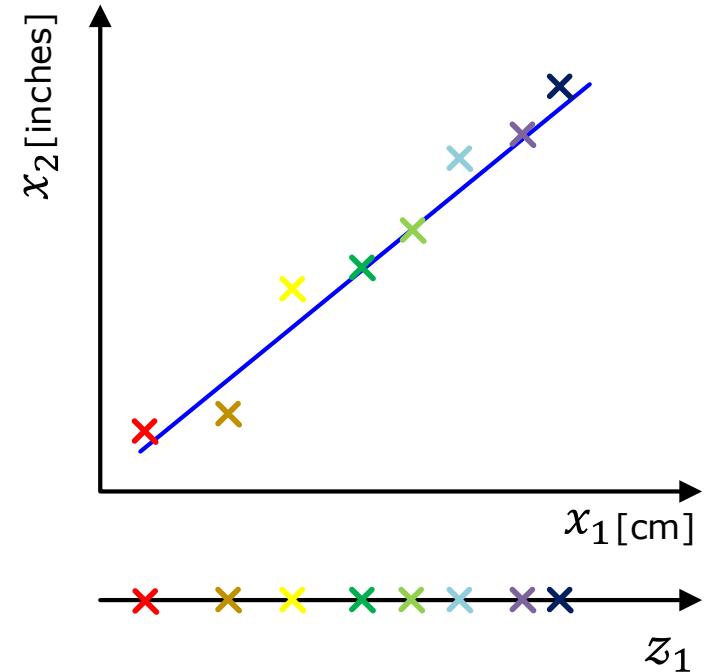


# PCA

PCA is constructed for **linear combinations of the** input features and obtains **uncorrelated variables as output** between them.

The idea is to find features that are highly correlated with each other and summarize them in a smaller space with a reduced number of variables.

Values that deviate from perfect correlation are approximated and their share of information is lost.

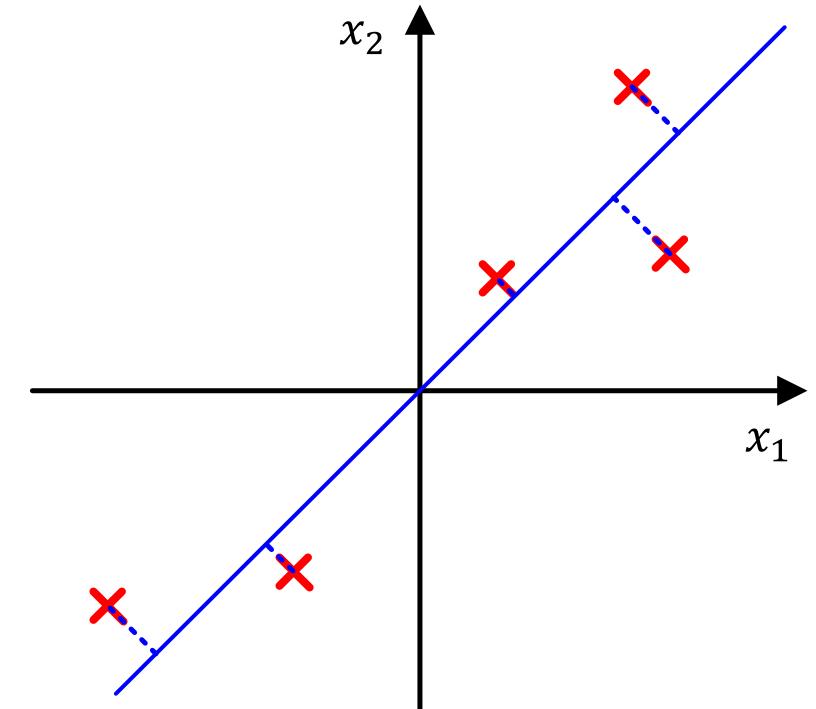


## Construction

PCA is obtained intuitively by following the following process

- The direction in which it is best to project the data is identified, i.e. the one in which the data varies the most
- The form that minimizes the projection error is found
- The data is projected and the procedure is re-executed

The new direction will be orthogonal to the one previously identified. Fortunately we can calculate PCA with just one matrix calculation.



## PCA - Implementation

- Let's take our features in column  $X$
- We remove the mean from each feature and standardize with respect to its standard deviation, so as to obtain each feature with a mean of zero and a standard deviation of 1.
- Let's calculate the **Singular Value Decomposition** SVD of the normalized matrix  $\tilde{X} \in \mathbb{R}^{N \times d}$

$$\tilde{X} = USV^T$$

$$U = \begin{bmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{bmatrix}_{N \times N}$$

$$S = \begin{bmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{bmatrix}_{N \times d}$$

$$V^T = \begin{bmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{bmatrix}_{d \times d}$$

## PCA - Implementation

The columns of the matrix U are an orthonormal basis of  $\mathbb{R}^N$

The elements on the diagonal of S are the **singular values** of  $\tilde{X}$

The columns of the matrix V are the orthonormal basis of  $\mathbb{R}^d$  and are the eigenvalues of the covariance

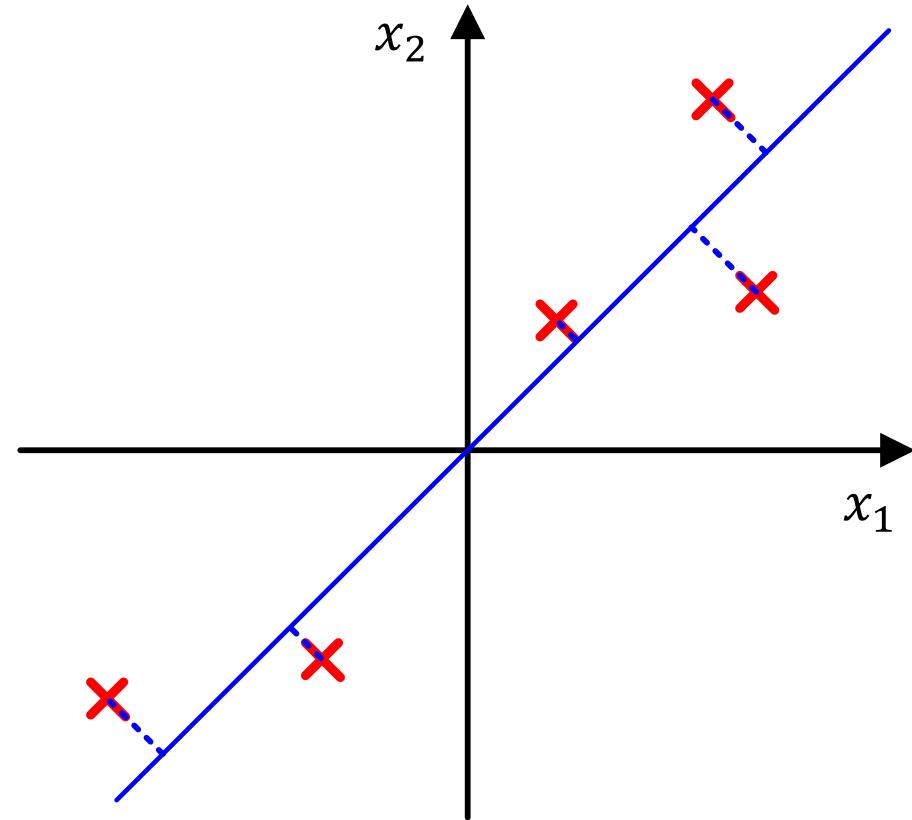
$$\text{matrix } \Sigma = \frac{1}{N} \tilde{X}^\top \tilde{X}$$

Taking the first ones  $q \leq d$  columns from V, we get our reduced matrix  $V_q$

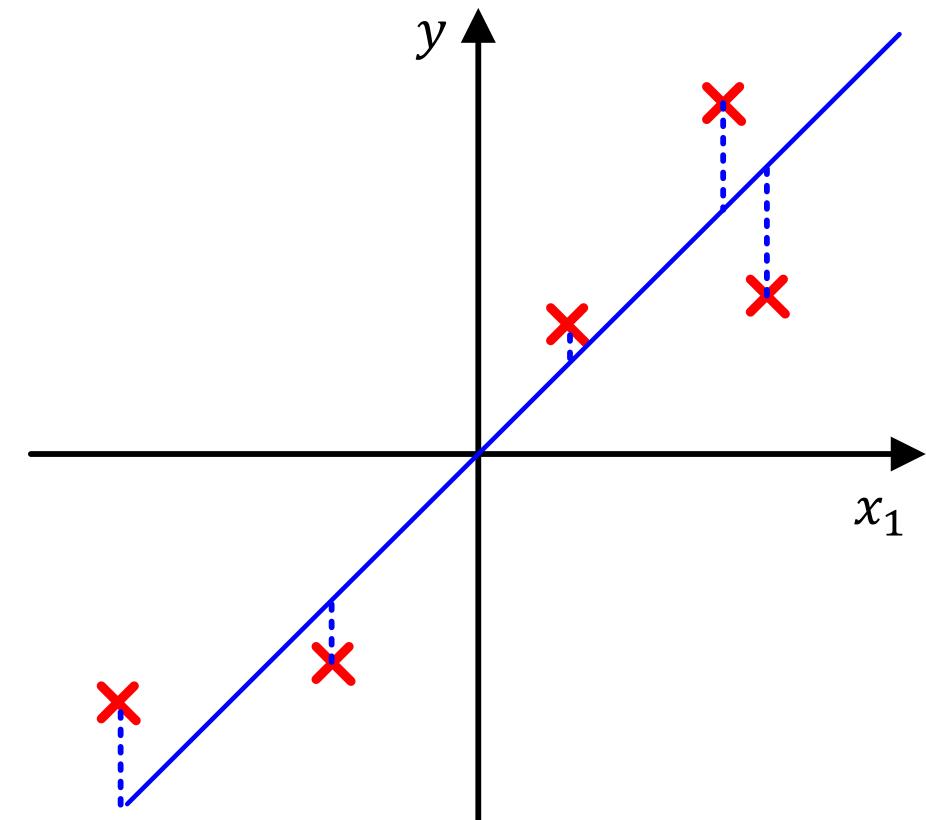
We can calculate us our new ones values projected as  $Z = \tilde{X}V_q \in \mathbb{R}^{N \times q}$

## PCA vs Linear Regression

PCA



Linear regression



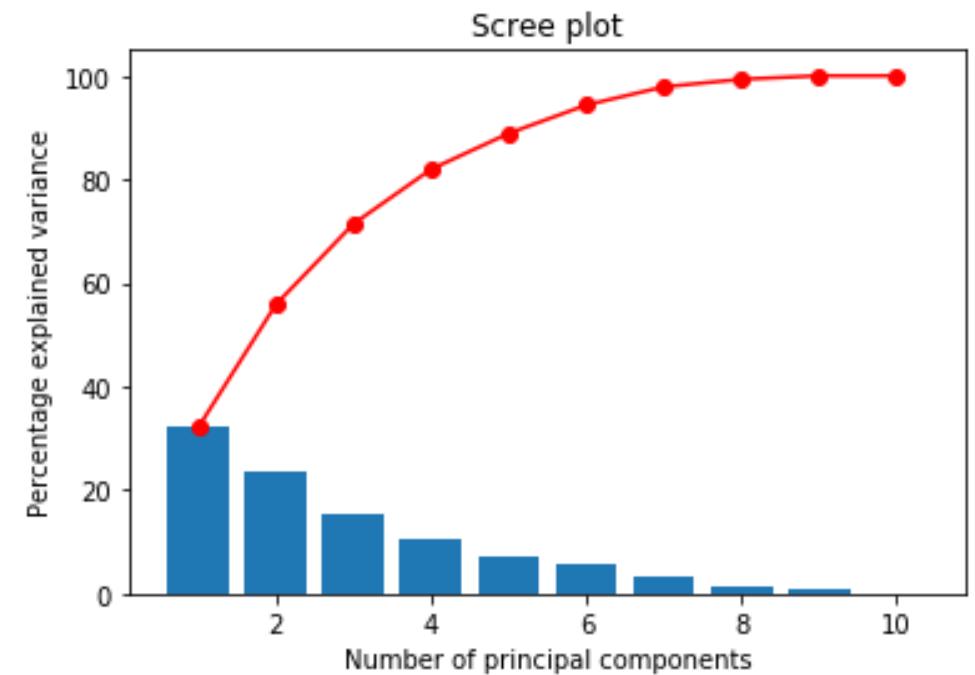
## Variance explained

In the case of PCA we are trying to find the projections that best explain the variance of the dataset.

Consequently, each calculated variable "explains" a portion of the variance.

We can use this information to **choose the number  $q$  of components** we want to obtain as a function of the variance we tolerate losing.

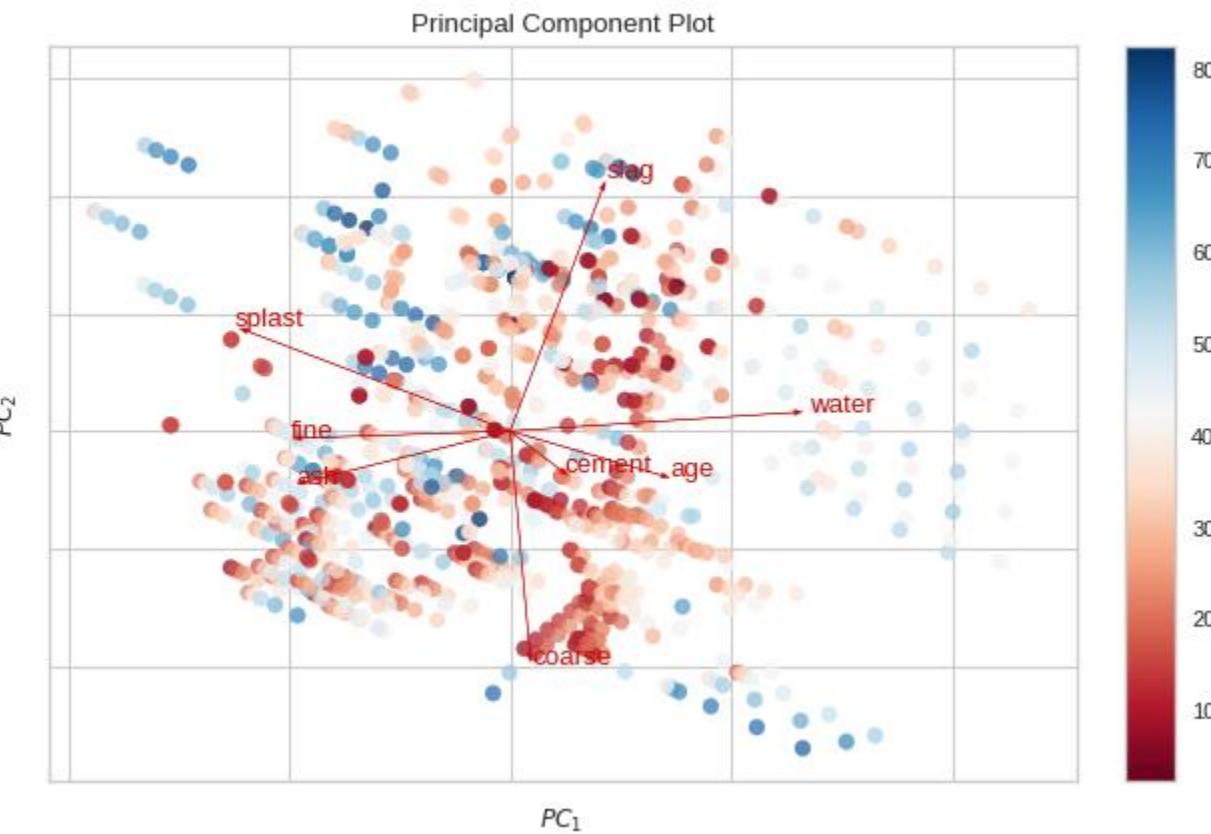
In many cases we will find that adding components will add little or nothing to the variance of the projected dataset.



## Projected axes

Going backwards we can go back to interpreting the main components to understand how they are composed.

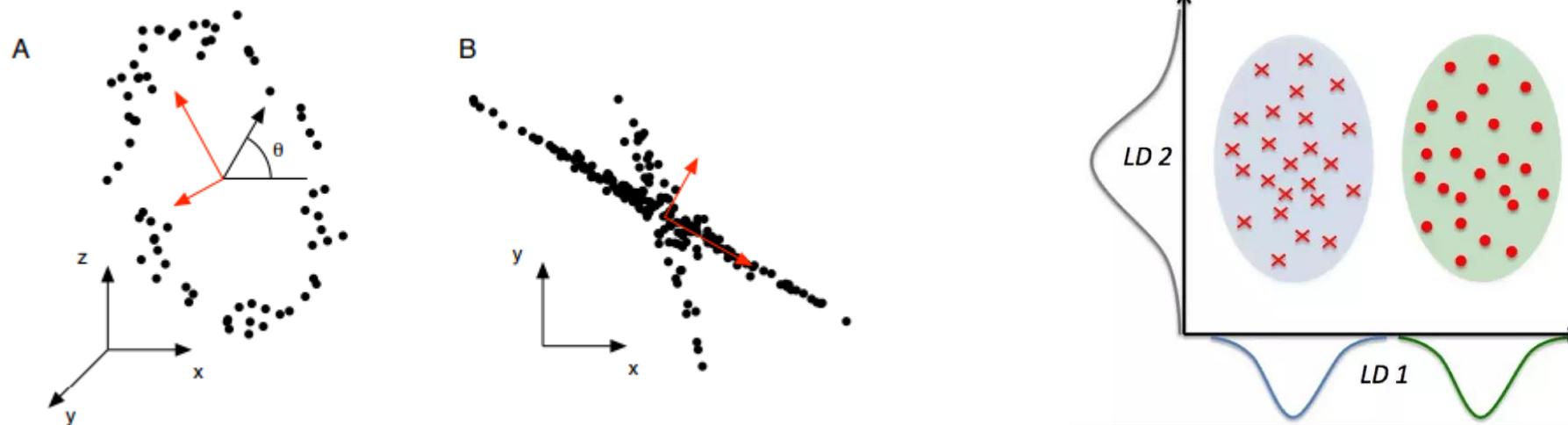
This allows us to better interpret the variables created and try to provide an explanation for them.



# Disadvantages of PCA

PCA has some disadvantages

- It assumes that the variables are linearly correlated, in many cases we already know that they are not
- Only the axes with the greatest variance are considered, there are some cases in which this leads to enormous errors

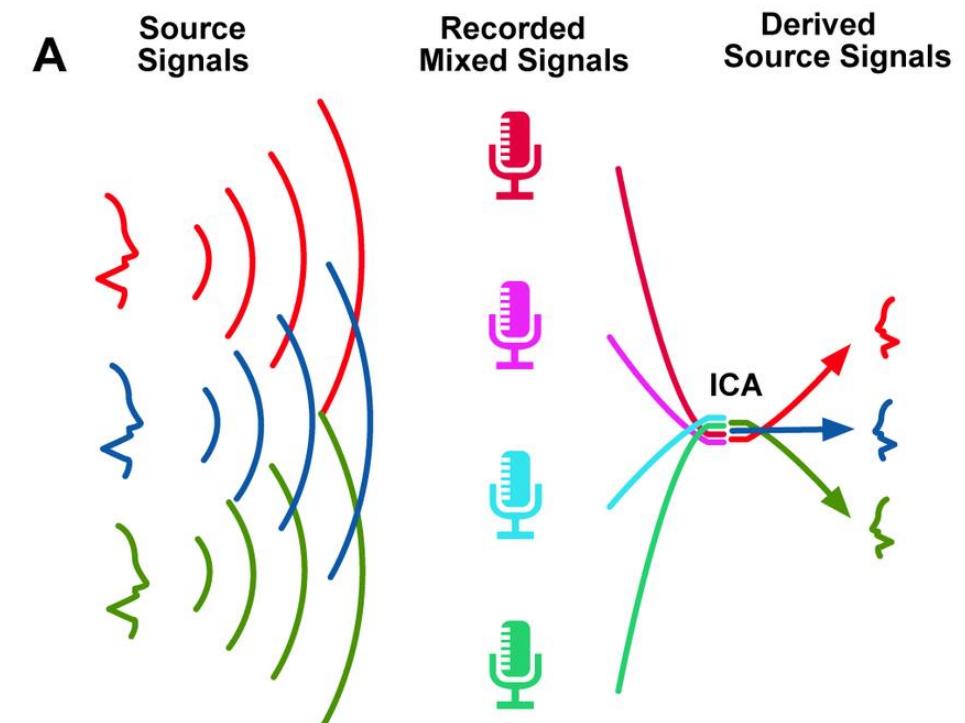


# Independent Component Analysis

ICA is an unsupervised learning technique that does not serve to reduce the number of features, but to make the components independent.

It is often used when the dataset needs to be transformed to separate different sources of information, for example to recognize 3 different people speaking by having access to recordings from 3 microphones in the same room.

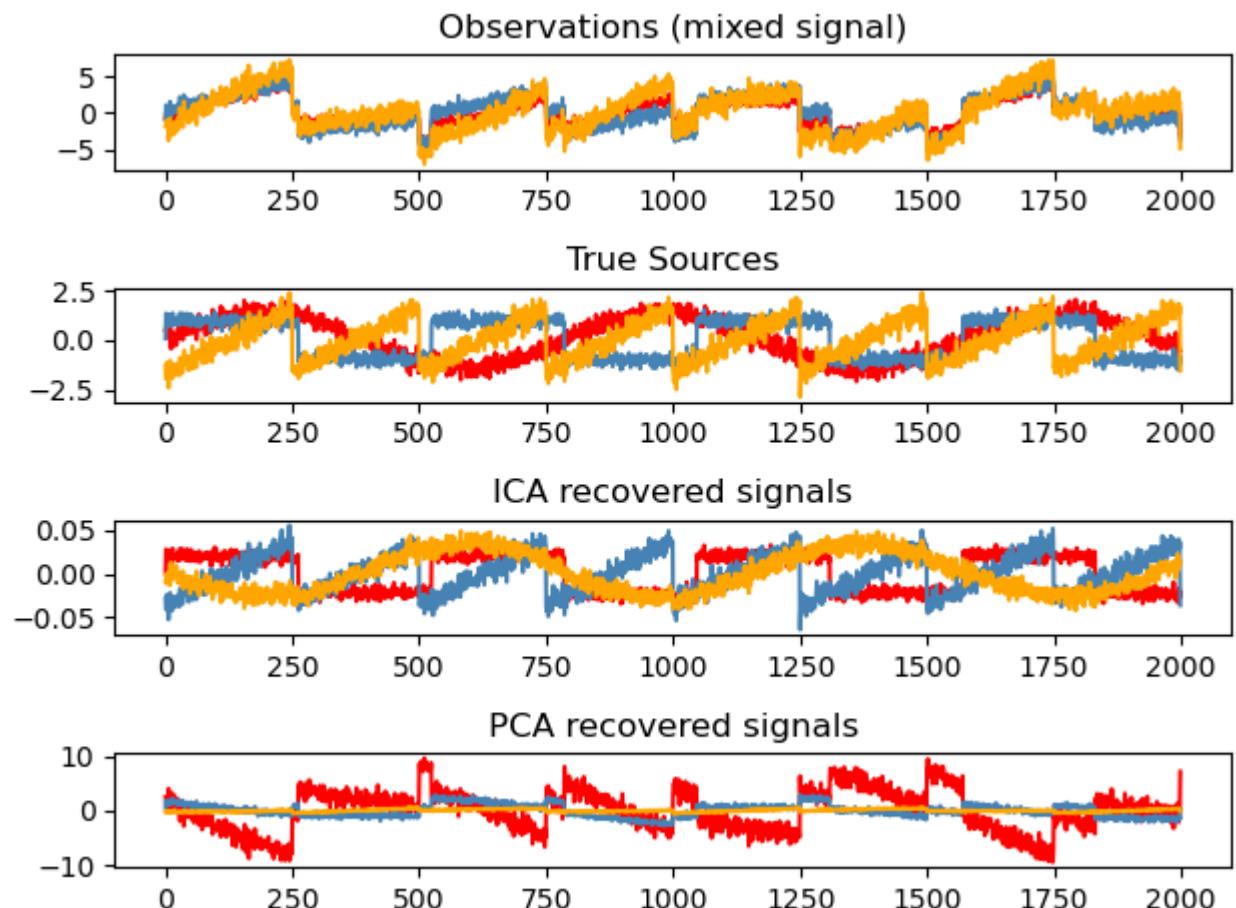
The ICA allows us to have more robust and potentially interpretable input features to the model, reducing the correlation terms.



## PCA vs ICA

PCA and ICA have different objectives and both must be used correctly.

Let's see the difference with this example of source reconstruction.



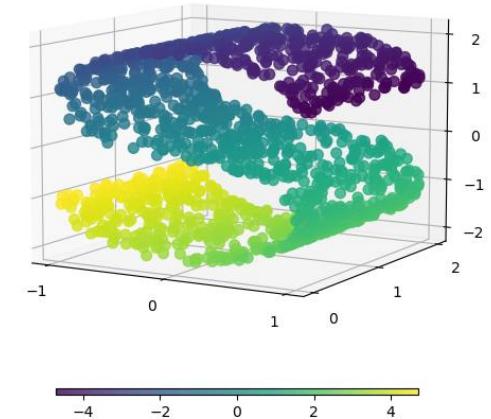
## Non-linearity

The linearity assumption of PCA is very convenient because we can calculate all the components and their importance (the explained variance) very quickly.

On the other hand, we know well that not all features have linear relationships, so we need to delve into dimensionality techniques reduction that can handle these non-linearities well.

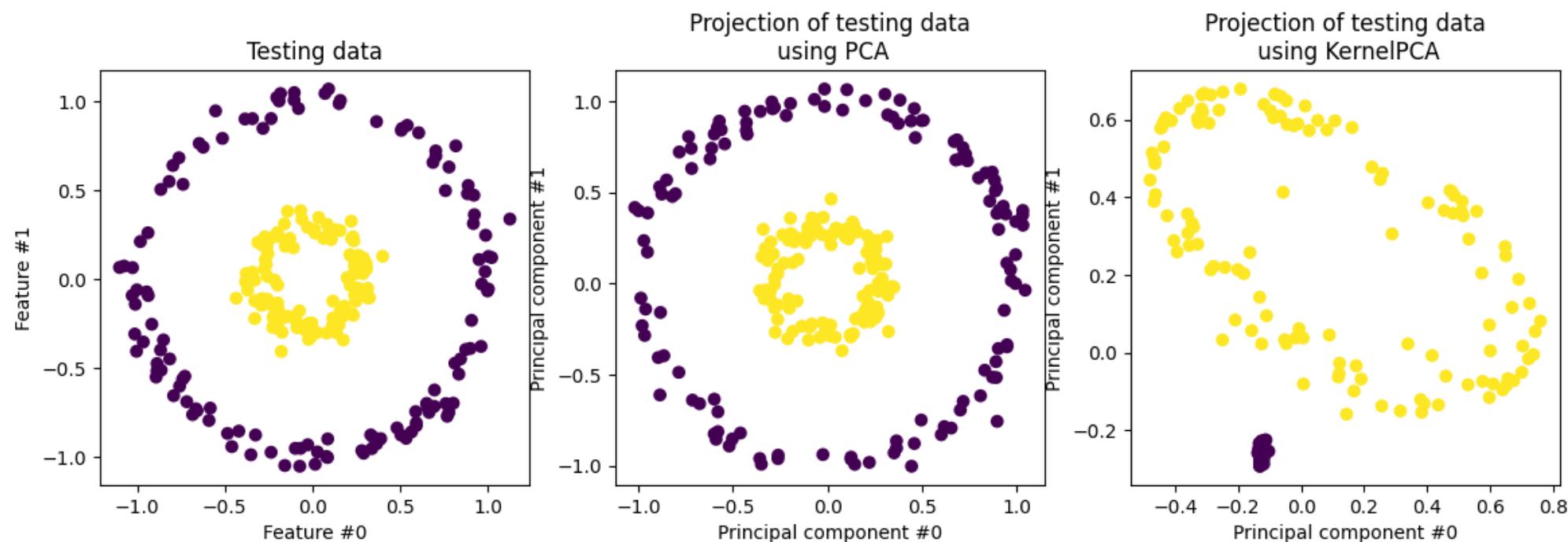
We will see a quick overview of these techniques, also called *manifold learning*.

Original S-curve samples



## Kernel PCA

Just like for non-parametric classification methods, also in dimensionality reduction we can use a **Kernel** to find a linearly separable feature space and then apply PCA to it.



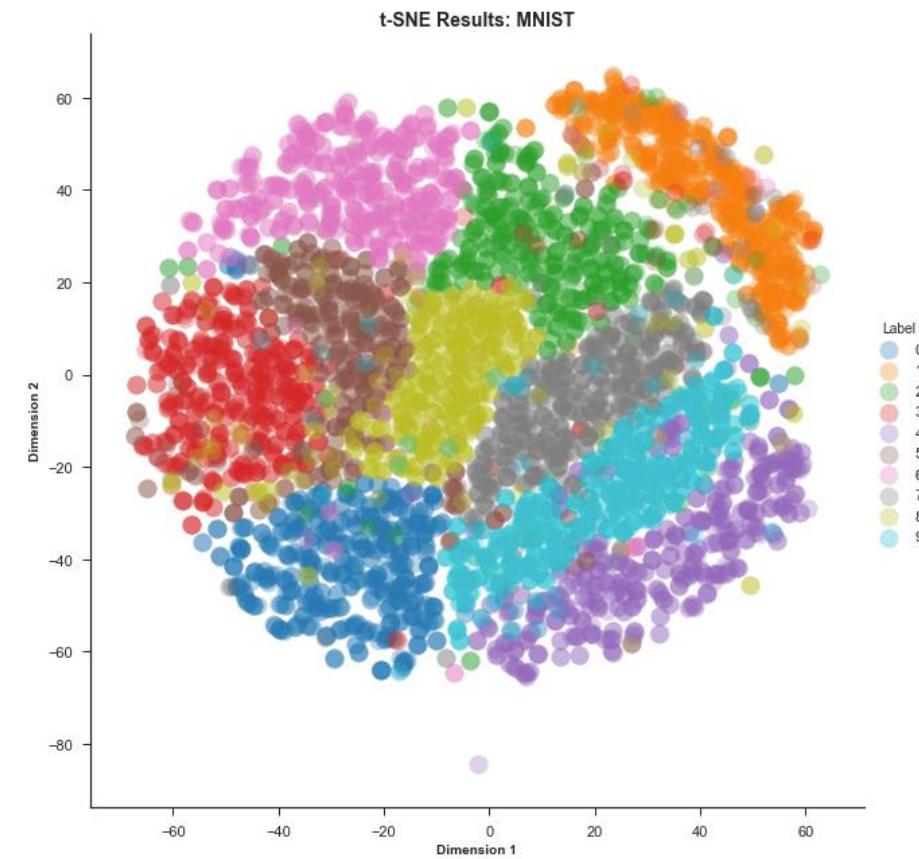
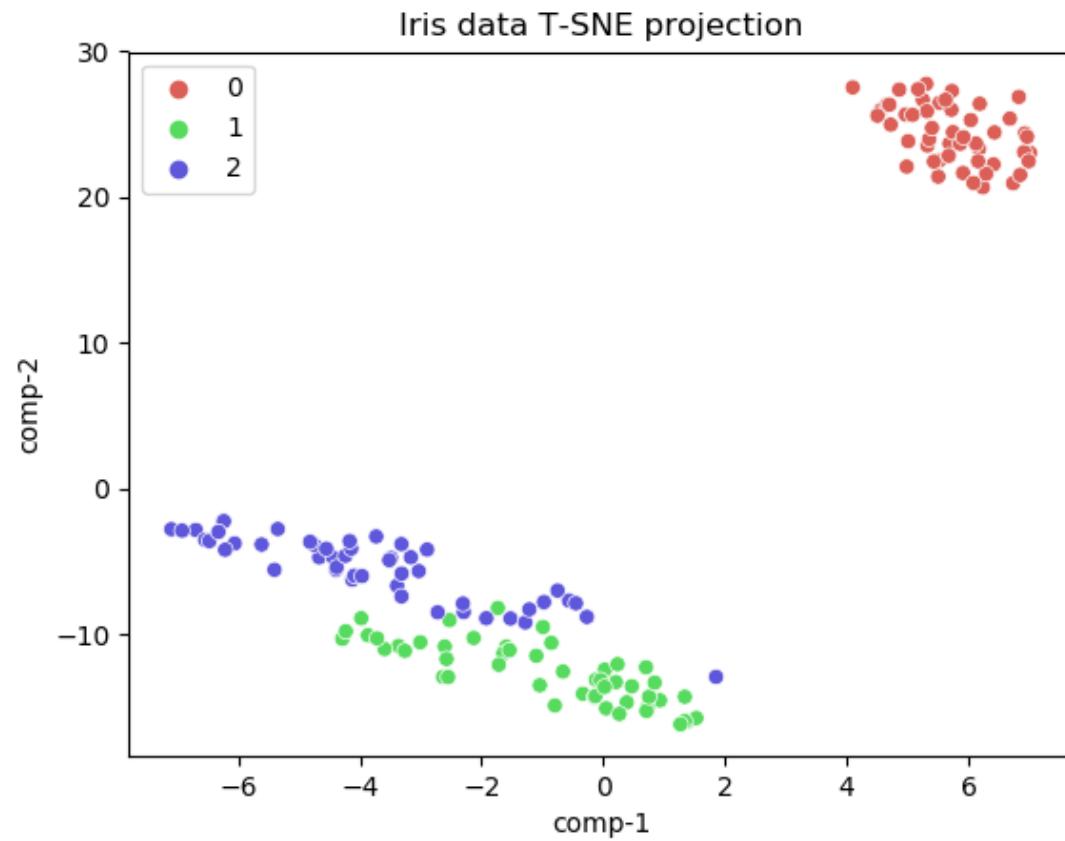
## t-SNE

The **t-distributed stochastic neighbor embedding**, or t-SNE, is a widely used technique for projecting problems with a very large number of features into a 2D or 3D space for visualization.

It is based on comparing the distance of points so that points that are close in N-dimensional space are also close in 2D space.

Two neighborhood probability distributions are constructed, one in N-dimensional space, and one in reduced space. The algorithm iteratively tries to move the points of the second one so that the distance (KL Distance) between the two distributions is minimized.

## t-SNE

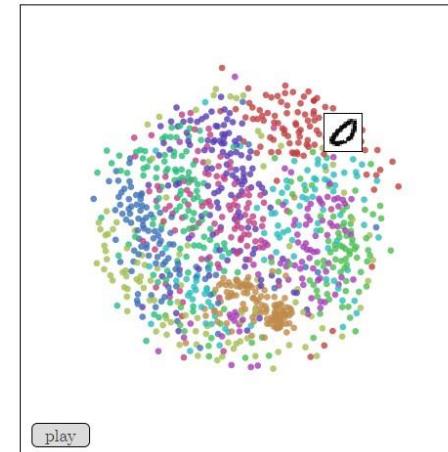


## MDS - Multi Dimensional Scaling

MDS is another technique used to do dimensionality reduction .

The concept is similar to that of t-SNE, distances between elements are calculated in a « dissimilarity matrix » in this case. However, instead of estimating two distributions and minimizing the difference, in this case MDS tries to keep the distances of the points unchanged by optimizing a function called *Strain* with the B elements calculated starting from the dissimilarity matrix D.

$$\text{Strain}_D(x_1, x_2, \dots, x_N) = \left( \frac{\sum_{i,j} (b_{ij} - x_i^T x_j)^2}{\sum_{i,j} b_{ij}^2} \right)^{1/2},$$



Visualizing MNIST with MDS

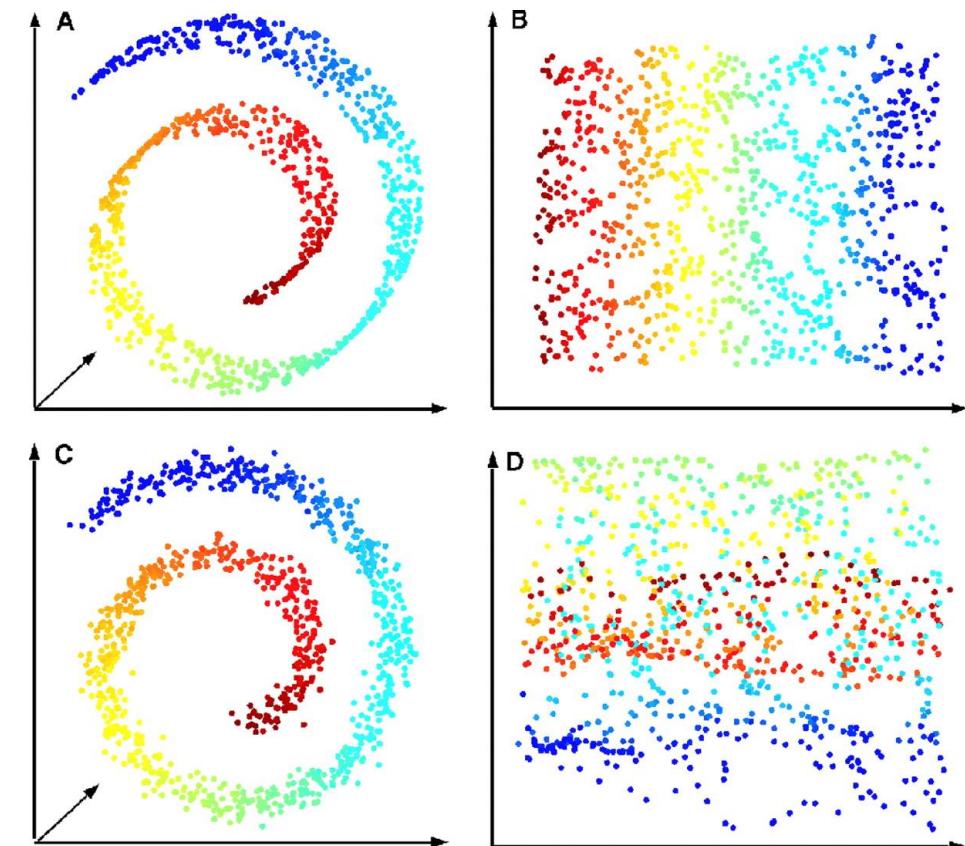
# Isomap

Isomap is another algorithm based on distances between points but in this case it works at a local level.

Iteratively

- determines the closest points to each candidate (e.g. within a radius or its k-NNs)
- constructs a graph of neighbors with edges weighted as a function of distance
- calculate the minimum distance between two nodes using Dijkstra or other techniques and then calculate the new projection space

This way is very effective for complicated cases like the spiral, where the data have much more significant local distances than the variance axes.



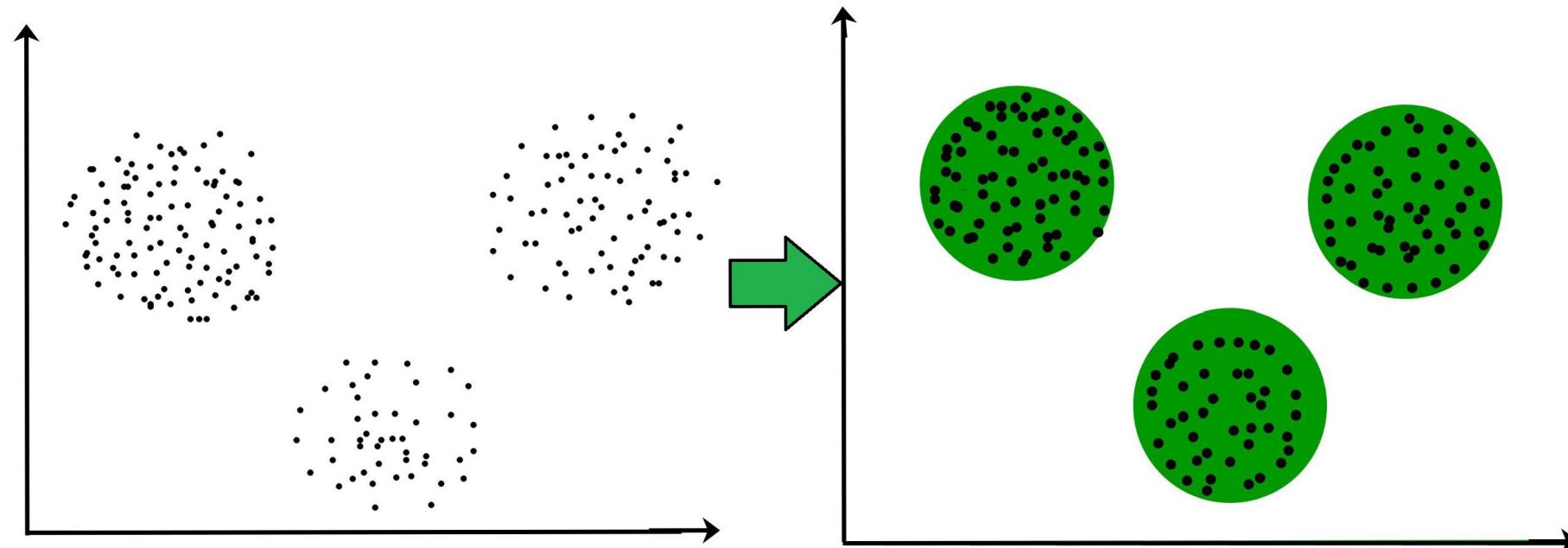
## Exercise

Let's try to perform these dimensionality techniques reduction on some known issues

[https://colab.research.google.com/drive/12X9Tc0FJFZduYrmz\\_yT9\\_quAuBqd0Fm0?usp=sharing](https://colab.research.google.com/drive/12X9Tc0FJFZduYrmz_yT9_quAuBqd0Fm0?usp=sharing)

# Clustering

Clustering is an area of unsupervised learning in which we find "clusters" or groups of points similar to each other within the data without having a label.

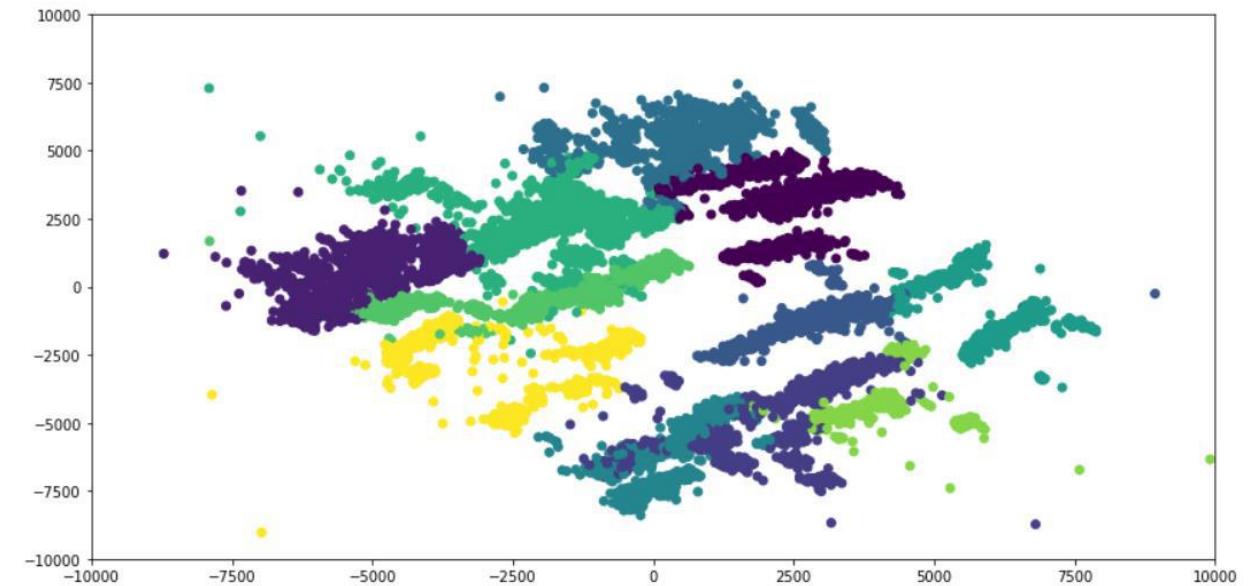


# Clustering

Clustering allows us to interpret patterns present in the data without making them explicit.

This is very useful in cases where we have

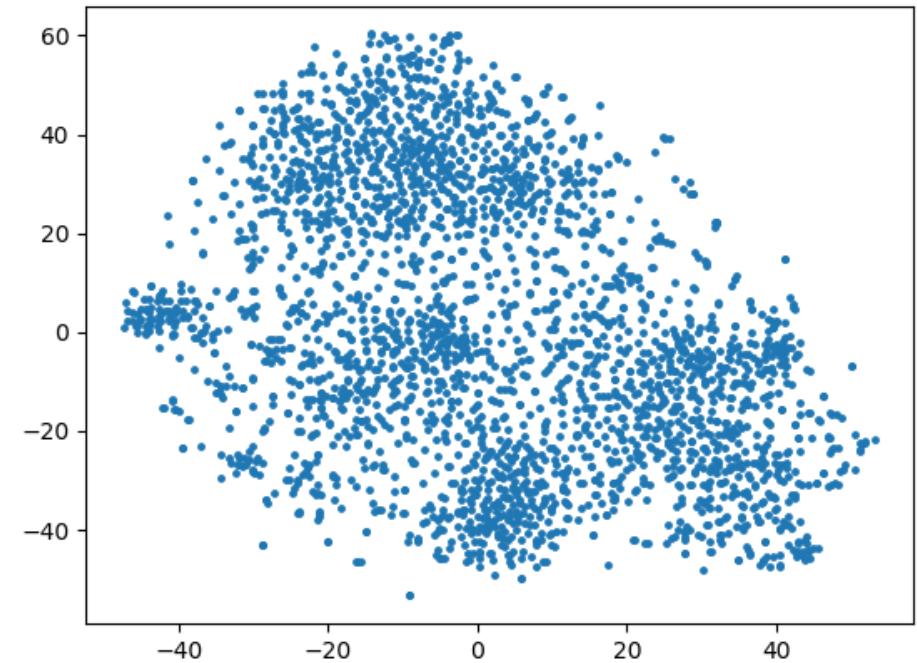
- Groups of users who behave similarly
- Positioning of products in the market
- Similar films
- ...



## Problem

A customer, operating in the world of customer service, provides us with a large amount of assistance requests via chat for a coffee pod manufacturer.

The objective is to identify the most frequent requests, without knowing which group each historic request belongs to, in order to report to the customer the areas in which to intervene most to improve the quality of the service.



## Problem

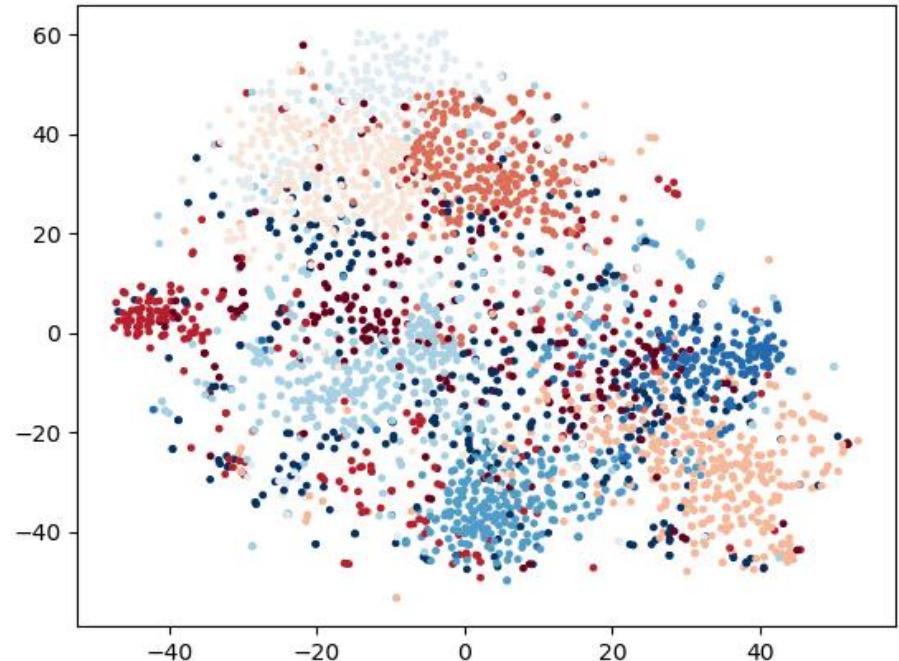
Given the texts we need it

- A way of codifying our requests for assistance
- A way of projecting them into a space

We will see the techniques for the text later, let's assume we have a method that, given two sentences, calculates a semantic distance.

We can choose to calculate all the distances between all the sentences or sample a certain random number and use them as the basis of our space. At that point we have our features, distances from specific examples, and we can go on to structure our clustering problem.

Clustering is best done in a larger N-dimensional space and then for convenience view it in 2D or 3D.

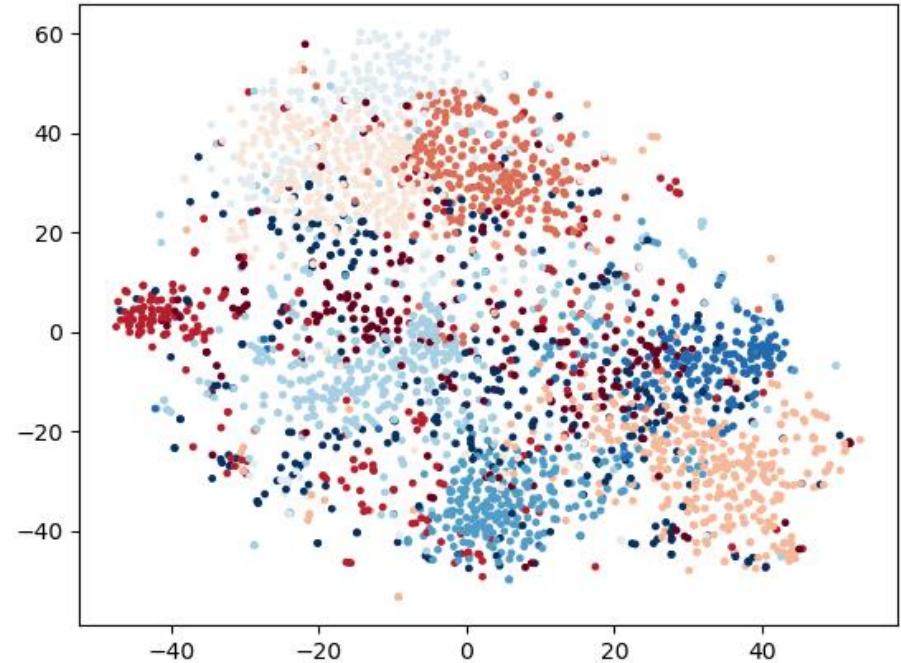


## Problem

Having done these steps, all we have to do is go back to interpreting the clusters, looking at the sentences that compose them and what they refer to.

The red cluster on the left are all requests for not having received a promotion confirmation email.

The customer did not realize that the promotional email server had shut down and reported the problem and resolved around thirty requests per month.



## Clustering

Normally it is much easier to obtain unlabeled data than labeled data, so in some cases we could do clustering first, take examples from a cluster to evaluate that they all belong to the same category and possibly label the entire cluster.

It is also interesting to try to understand why, or rather what are the characteristics that the examples of the identified groups have in common.

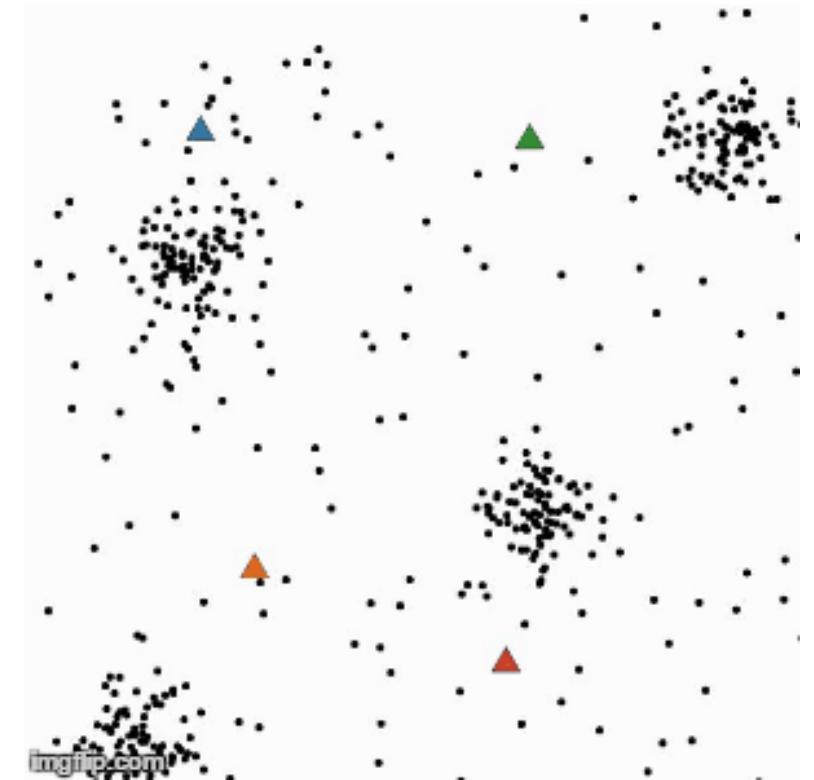
Finally we can use the information about the cluster to which they belong to define policies, for example we will have groups of items that sell constantly over time, so it makes sense to stock, and items that instead have a high volatility, so we will order stocks only every customer's order.

## K- means

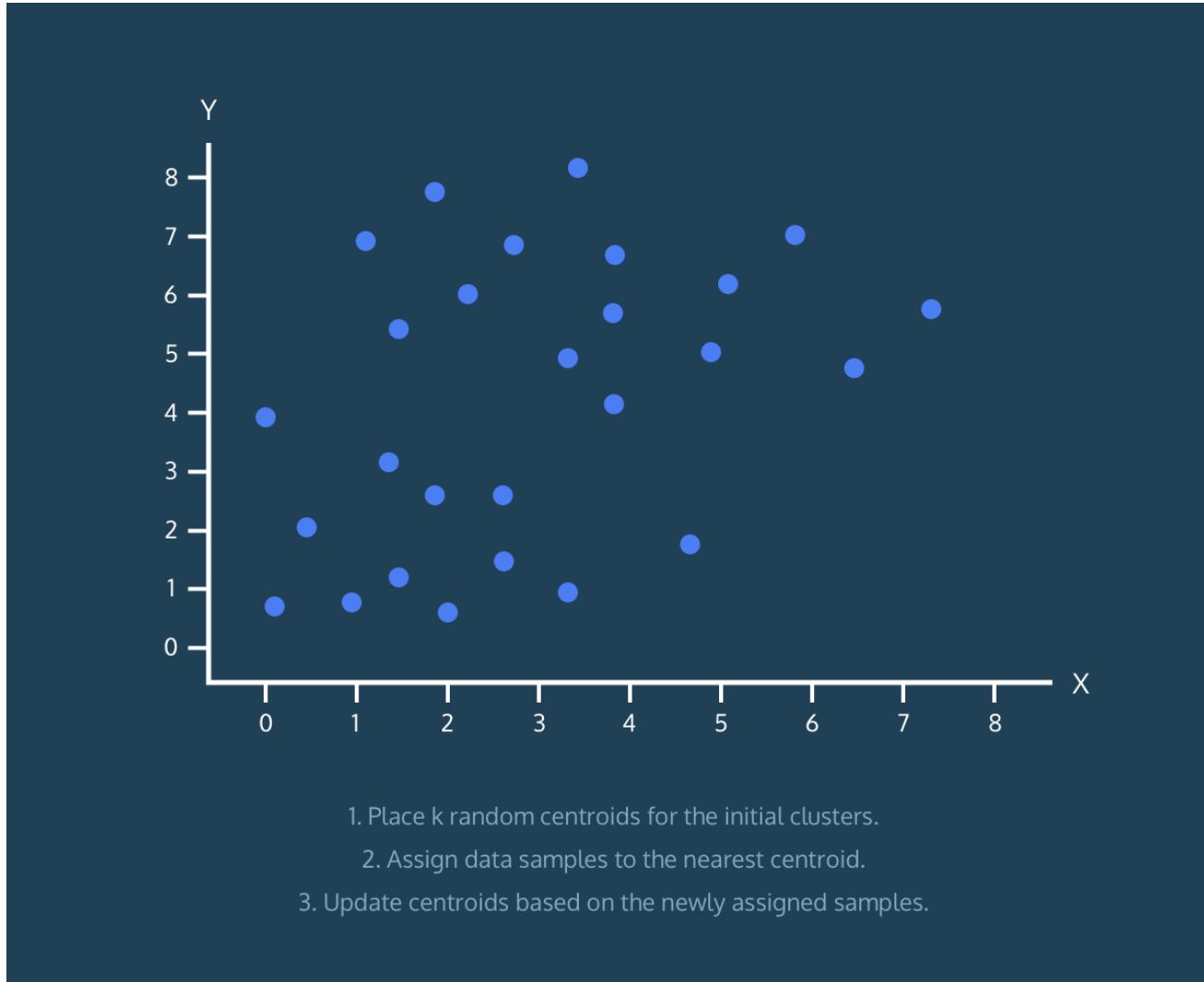
K- means is the simplest clustering technique.

We need to know no matter how many clusters we are looking for in the data ( $k$ ).

The objective of the technique is to minimize the total sum of the squares of the distances of each of its points from its centroid .



## K- means



## K- means

As we saw in the images

- We define a number of clusters
- Let's run random K centroids
- Iteratively
  - We associate each point with the closest centroid
  - We recalculate the position of the centroid as the average of the distances from its points

When we no longer move any points we have finished the search.

K- means is a greedy algorithm that converges to a local minimum.

We can search for clusters with random centroids multiple times to improve performance.

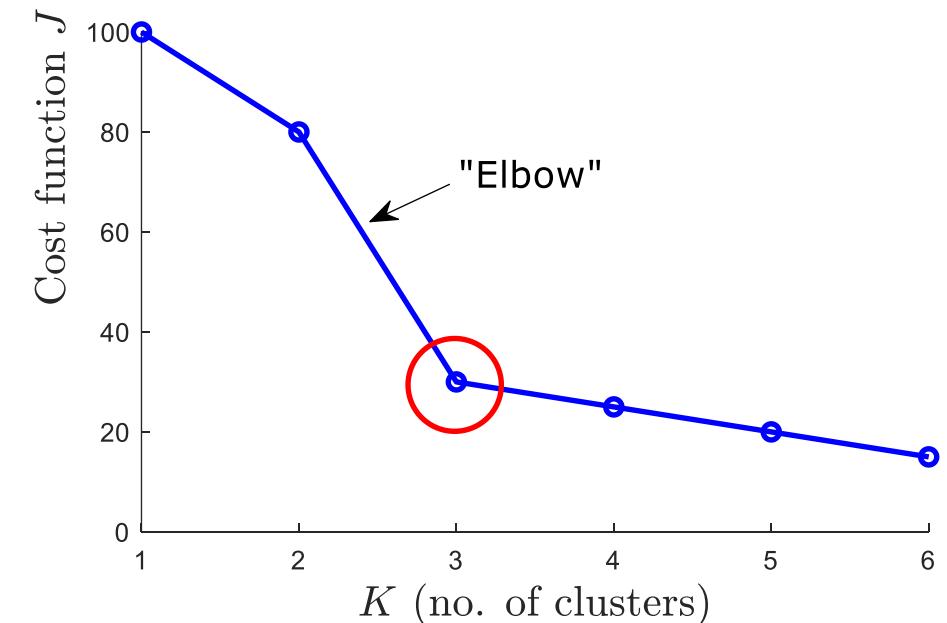
## Number of clusters

One of the fundamental choices is that of the number of clusters.

unsupervised problems we do not know a priori how many groups there might be, which is why we can rely on

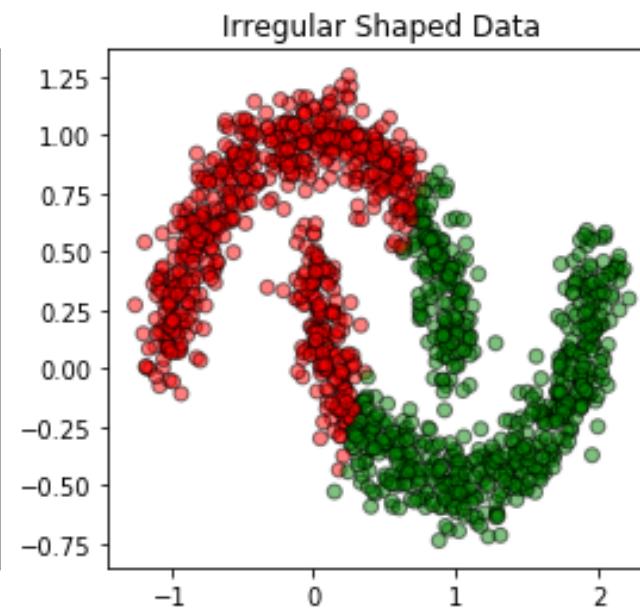
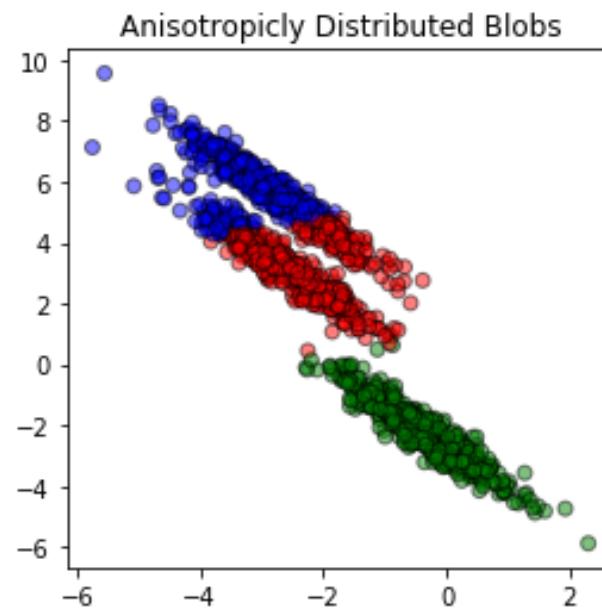
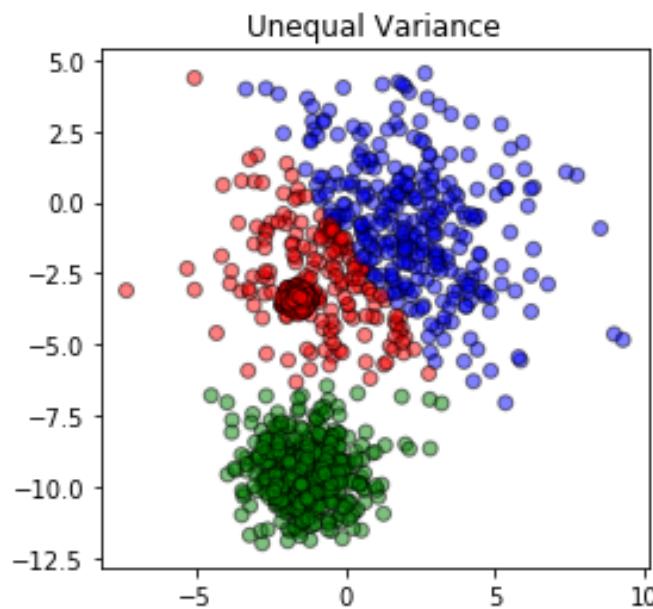
- Knowledge about the data and the problem
- Data visualization (especially in low cardinality cases)
- The " Elbow " rule, i.e. observing the variation that adding other clusters causes on the division of the problem.

An example is how many sizes to divide the dimensions of the t-shirts into. We can cluster our customers' sizes and see where there is variation where adding a size makes less sense.



## K- means errors

The k- means is subject to many types of error and tends to give each cluster the same number of points. Furthermore, randomizing the initial state is very subject to different results, often ending up in very inexpressive local minima.



## Exercise

Let's try running our first clustering technique on a dataset and see if it extracts any useful patterns

[https://colab.research.google.com/drive/12X9Tc0FJFZduYrmz\\_yT9\\_quAuBqd0Fm0?usp=sharing](https://colab.research.google.com/drive/12X9Tc0FJFZduYrmz_yT9_quAuBqd0Fm0?usp=sharing)



# **08 – Unsupervised Learning**

Daniele Gamba

2022/2023

# Top TV Series – ML Course 2023

Series taken from the IMDB Top 250 by number of reviews.

Reply that we try to watch the series most similar to your tastes

<https://forms.gle/Mz4cqNSQ2kTge1sw9>

Rank & Title	IMDb Rating	Your Rating
 13. Il trono di spade (2011)	9,1	
 2. Breaking Bad - Reazioni collaterali (2008)	9,4	
 99. Stranger Things (2016)	8,6	
 50. Friends (1994)	8,8	
 20. Sherlock (2010)	9,0	
 5. Chernobyl (2019)	9,3	
 126. Dexter (2006)	8,6	
 28. The Office (2005)	8,9	



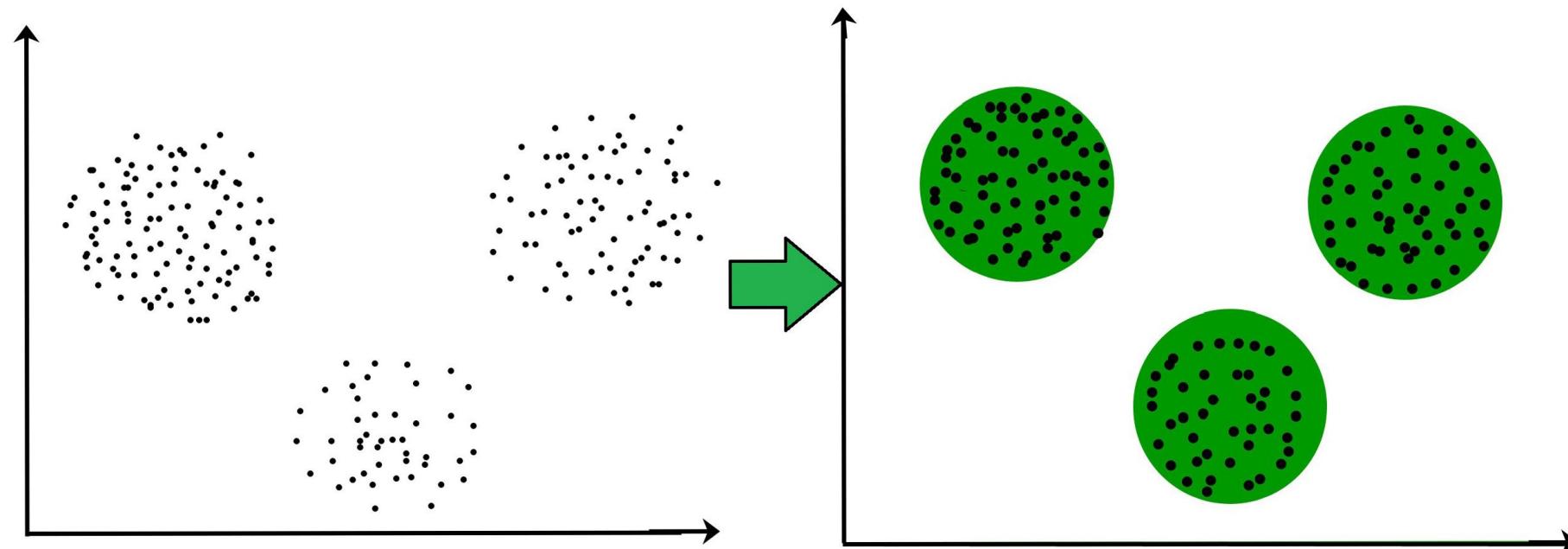
## Previous

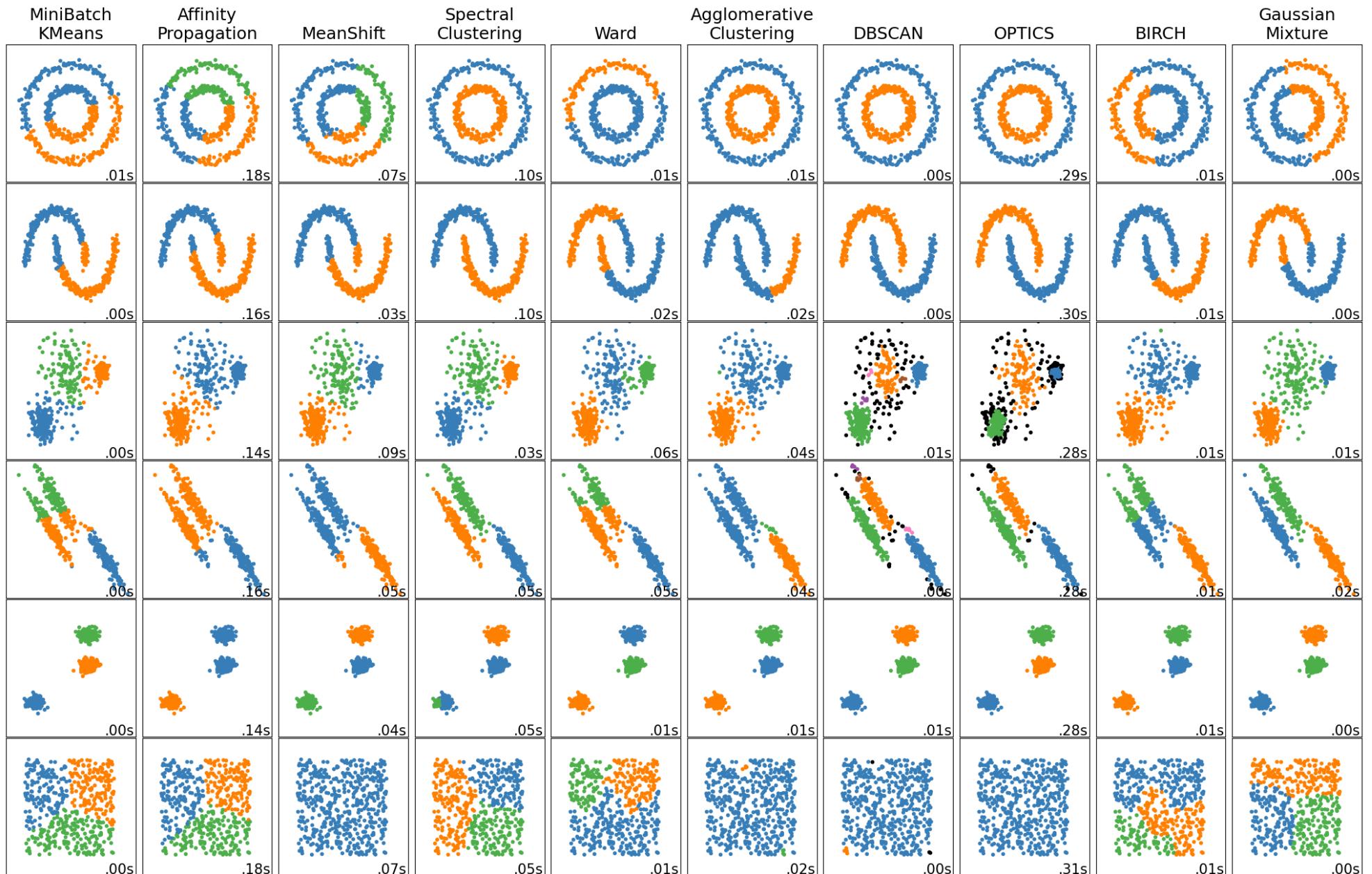
We have seen

- Dimensionality reduction
- Introduction to clustering
- k- means

# Clustering

Clustering is an area of unsupervised learning in which we find "clusters" or groups of points similar to each other within the data without having a label.

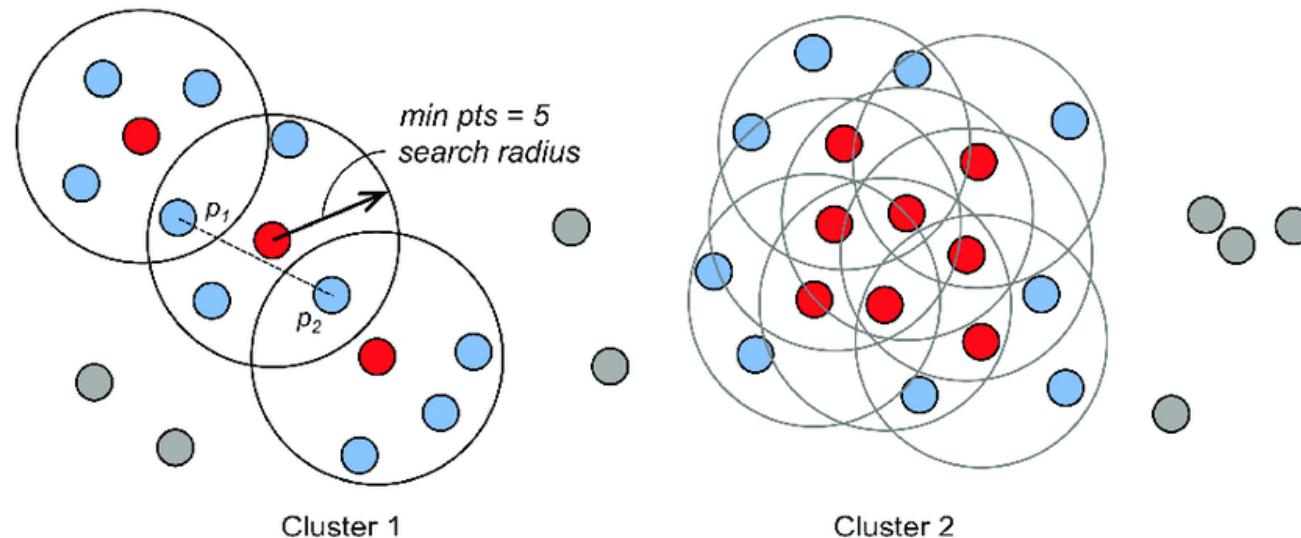




## DBSCAN

DBSCAN is a clustering technique that is based on the concept of density.

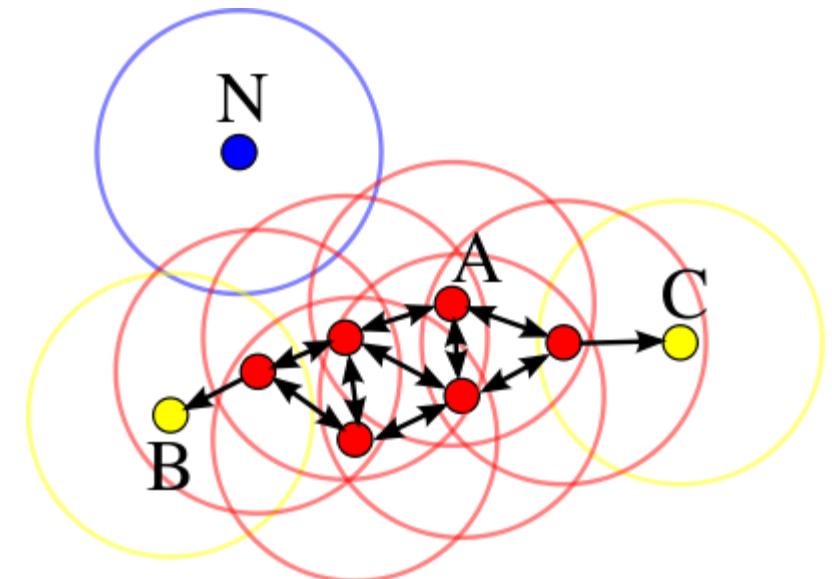
Taking a random point, we expand the cluster to all points that have a smaller distance of a  $\epsilon$ fixed person.



## DBSCAN

At the end of the first cluster, we will find ourselves with N points belonging to one cluster and others outside.

For each point we evaluate the number of other points within its radius. If they are above a certain threshold we will consider it part of a new cluster and we will start to expand it, if the point is sparse with nothing or few points in its surroundings it is classified as noise .



## DBSCAN - Limits

The biggest limitation of DBSCAN is that the fixed radius severely limits its capabilities when there are very close points.

It is therefore unable to interpret the density *variation as long as it manages to expand the cluster.*

If the clusters were even weakly connected to each other DBSCAN would tend to make them all belong to the same cluster or would require a lot of tuning.

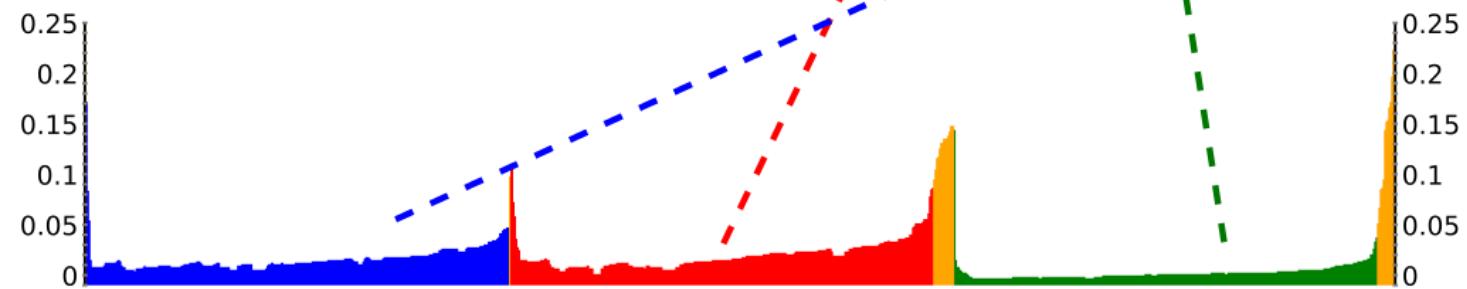
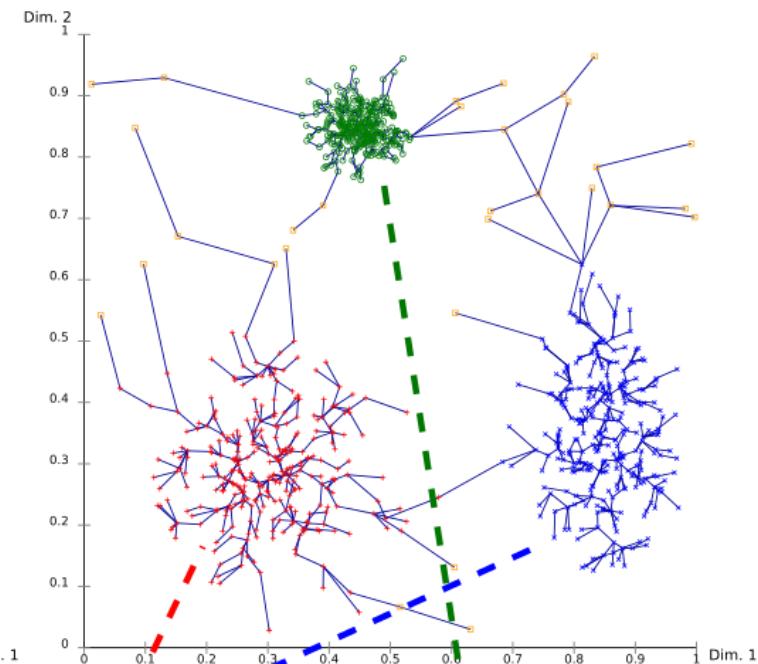
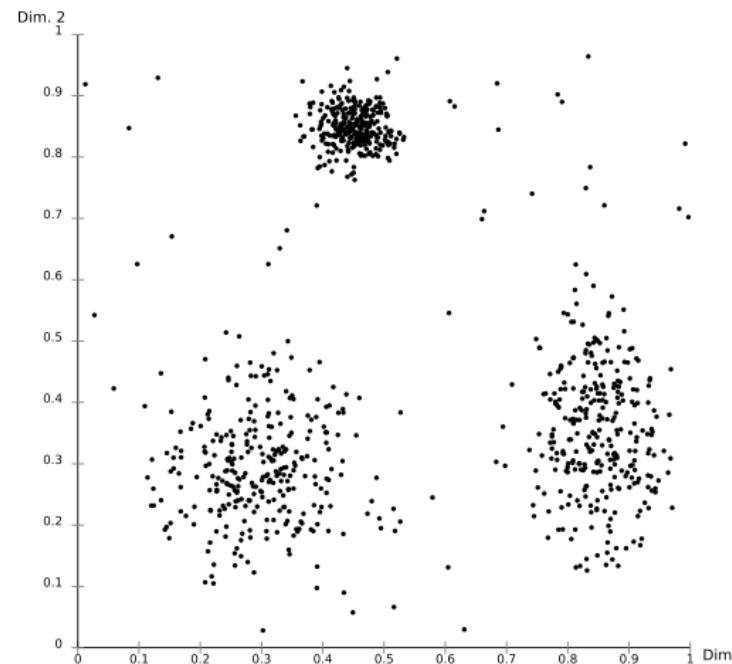
# OPTICS

**Ordering points to identify the clustering structure – OPTICS** is an algorithm That wants overcome the DBSCAN limits by weighing the density variations .

The points they come sorted so that the points more neighbors result neighbors in the ordering . Even in this case there is a  $\varepsilon$  which describes the maximum distance to consider and the minimum number of points to form a cluster.

Each point is also assigned a "reachability" metric from the nearest cluster center. By measuring the reachability of each point we can identify the clusters according to their density. The reachabilities will be distributed as «valleys» in the dendrogram .

# OPTICS



## Hierarchical clustering

In many problems, such as that of requests for assistance, it is common to find data belonging to different areas (mechanical, quality, commercial, ..) with greater detail for each of these (pump problems, resistance problems, mechanical problems, ... ).

This is because the data is structured into individual thematic and problem areas with a **hierarchy**.

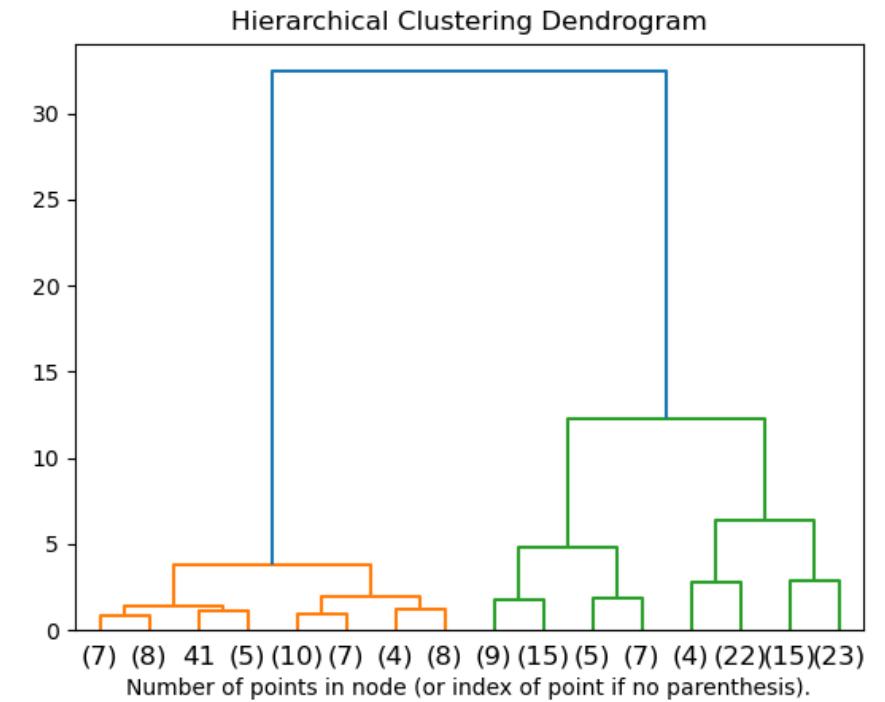
Hierarchical clustering is a technique that allows us to build a tree of relationships following one approach

- Bottom-up, iteratively aggregating similar clusters
- Top-down, dividing the clusters into denser subclusters

## Hierarchical clustering

Hierarchical has several metrics with which to proceed to divide or aggregate clusters.

One of these, called **ward**, minimizes the sum of the squared distances within all clusters, trying to minimize the variance within the cluster.



## Exercise - Telco

We are a telephone company and we want to understand more why our customers abandon us.

We have collected information on our customers who have abandoned us in the last month including the services they had active, demographic information, how much they spent, etc.

Let's try to understand how our lost customers are distributed and group them in order to target them with ad-hoc campaigns.

[https://colab.research.google.com/drive/12X9Tc0FJFZduYrmz\\_yT9\\_quAuBqd0Fm0?usp=sharing](https://colab.research.google.com/drive/12X9Tc0FJFZduYrmz_yT9_quAuBqd0Fm0?usp=sharing)

# Recommender systems

Recommender systems are algorithms that try to suggest a new item to a user in the hope that he or she will like it .

Recommendation algorithms are everywhere, Netflix, Amazon, YouTube, Spotify, etc.

They represent a critical part of the turnover, both to suggest other purchases (Amazon, ..) and to provide a better quality of service (YouTube, Spotify, Netflix).

Some recommendation algorithms can be seen both as **unsupervised problems** (I find users more similar to me) and as **regression problems** (what could be the rating I will give to that film).

How likely are you to recommend Windows 10 to a friend or colleague?

1       2       3       4       5

Not at all likely

Extremely likely

Please explain why you gave this score.

I need you to understand that people don't have conversations where they randomly recommend operating systems to one another

## Recommender systems



Movies



Online Videos



Music



Books & products



Softwares



PageRank



tripadvisor®

Restaurants



Accommodation



Dating



Friends

# Netflix Challenge

From 2006 to 2009, Netflix launched a competition for a ratings prediction algorithm.

It offered a \$1 million prize to anyone who improved the state of the art ( Cinematch ) by 10%.  
20,000 teams from 150 countries participated.

On June 26, 2009 the BellKor's team Pragmatic Chaos wins with an improvement of 10.05%.

The screenshot shows the Netflix Prize Leaderboard. At the top, it displays "Leaderboard 10.05%" and a link to "Display top 20 leaders". A yellow arrow points to the "% Improvement" column in the table below. The table has columns for Rank, Team Name, Best Score, % Improvement, and Last Submit Time. The top entry is for "BellKor's Pragmatic Chaos" with a score of 0.8558 and an improvement of 10.05%. The "Grand Prize - RMSE <= 0.8563" is highlighted in red at the bottom of the table.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	BellKor's Pragmatic Chaos	0.8558	10.05	2009-06-26 18:42:37
Grand Prize - RMSE <= 0.8563				
2	PragmaticTheory	0.8582	9.80	2009-06-25 22:15:51
3	BellKor in BigChaos	0.8590	9.71	2009-05-13 08:14:09

# Content Based vs Collaborative Filtering

Two of the classic recommender systems techniques are based on two different approaches

## **Content Based**

Recommend items that are similar to ones the user has liked in the past

## **Collaborative Filtering**

Recommend items that users similar to the target have appreciated

The second approach has become more popular since

- Items that I have already liked I may no longer need (e.g. I have already purchased the phone I liked on Amazon, I will not immediately buy another one)
- Similar users may have explored new items that I didn't know existed

## Collaborative filtering

Behind collaborative filtering there is the idea that information about users' tastes is widespread.

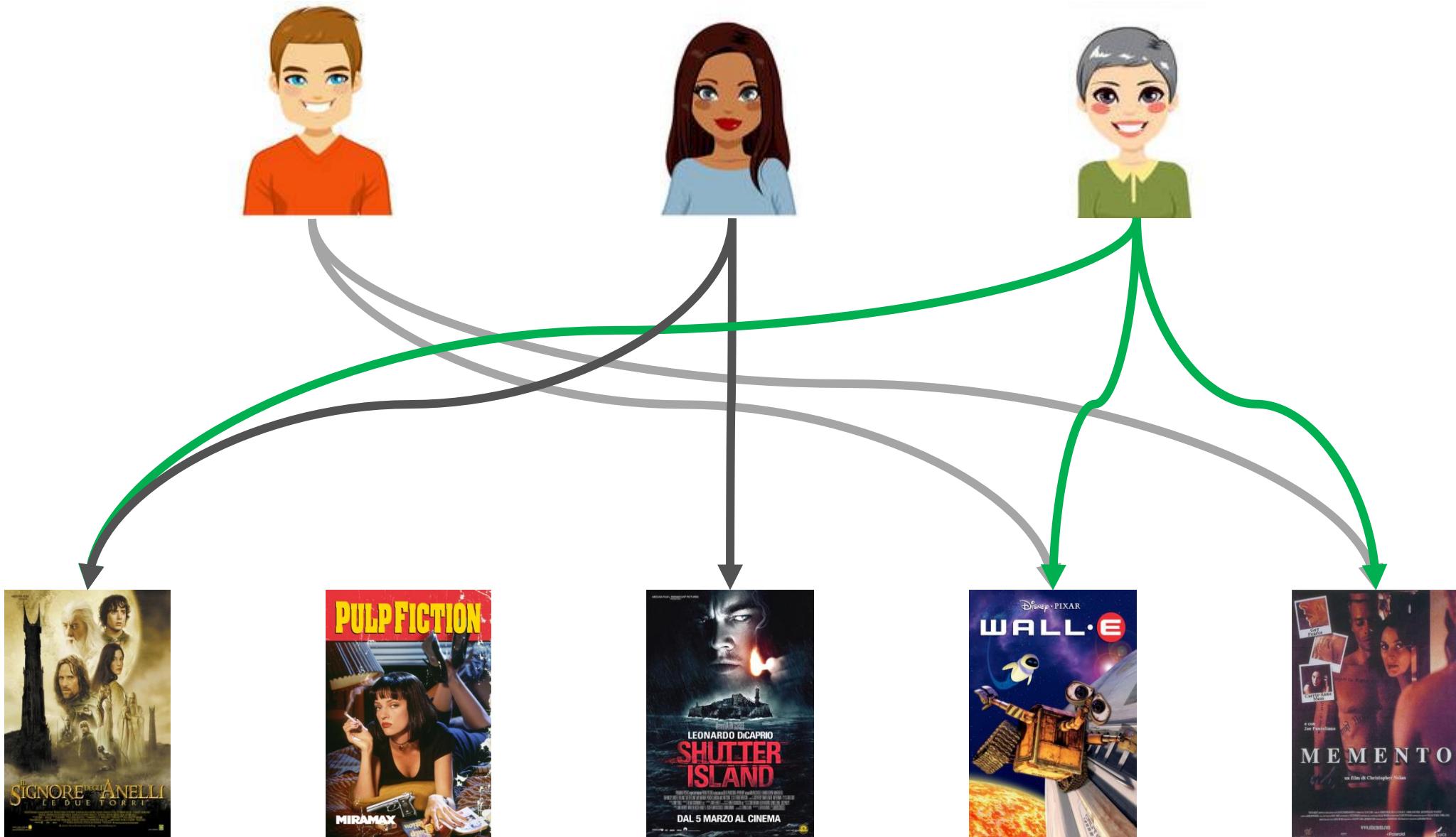
Different users, who explore more, will tend to explore and evaluate the different items differently, allowing even less daring new users to explore "the space of tastes".

Similar users, who share the same tastes, will allow you to propose new items to others.

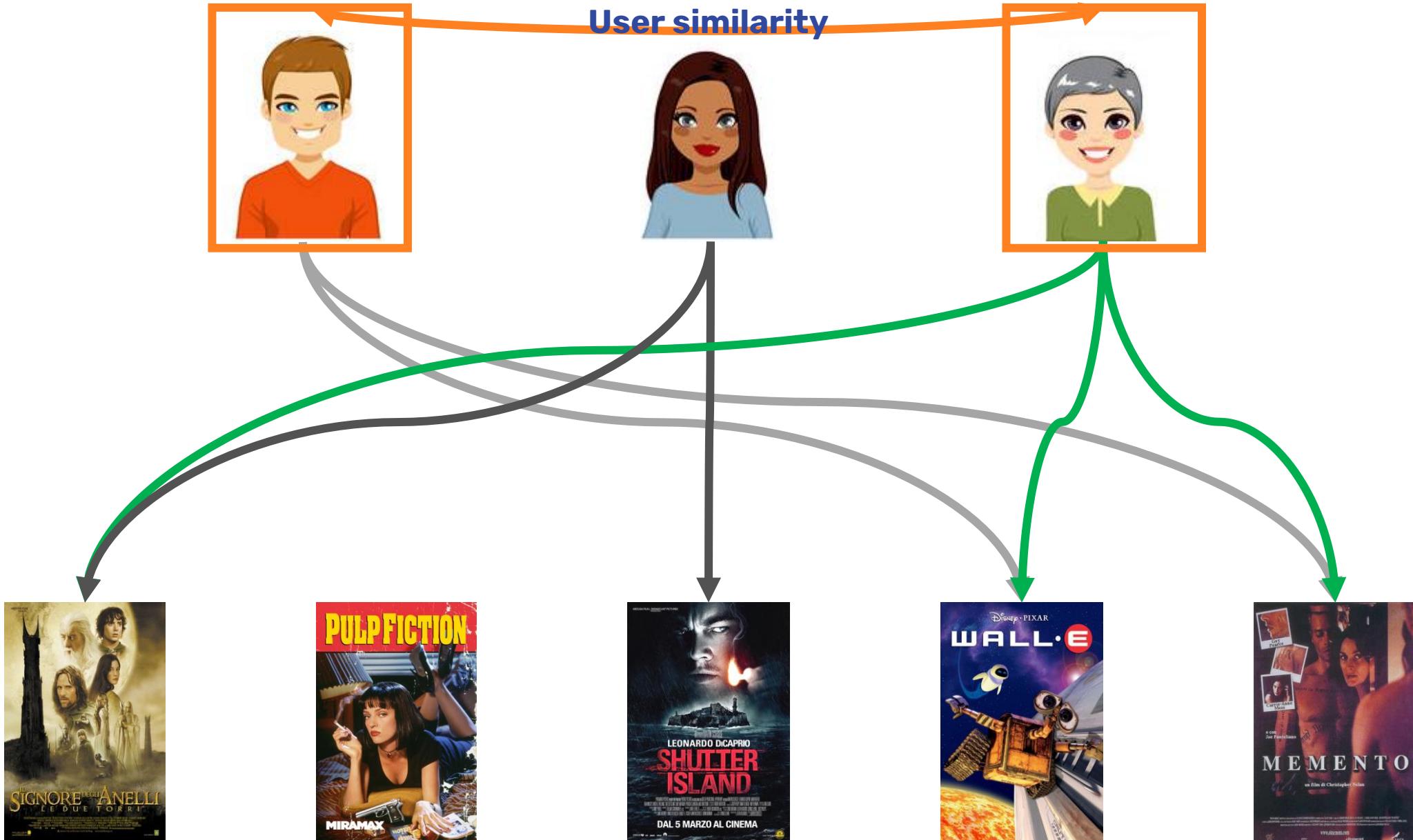
There are two identifiable approaches

- Item- based
- User- based

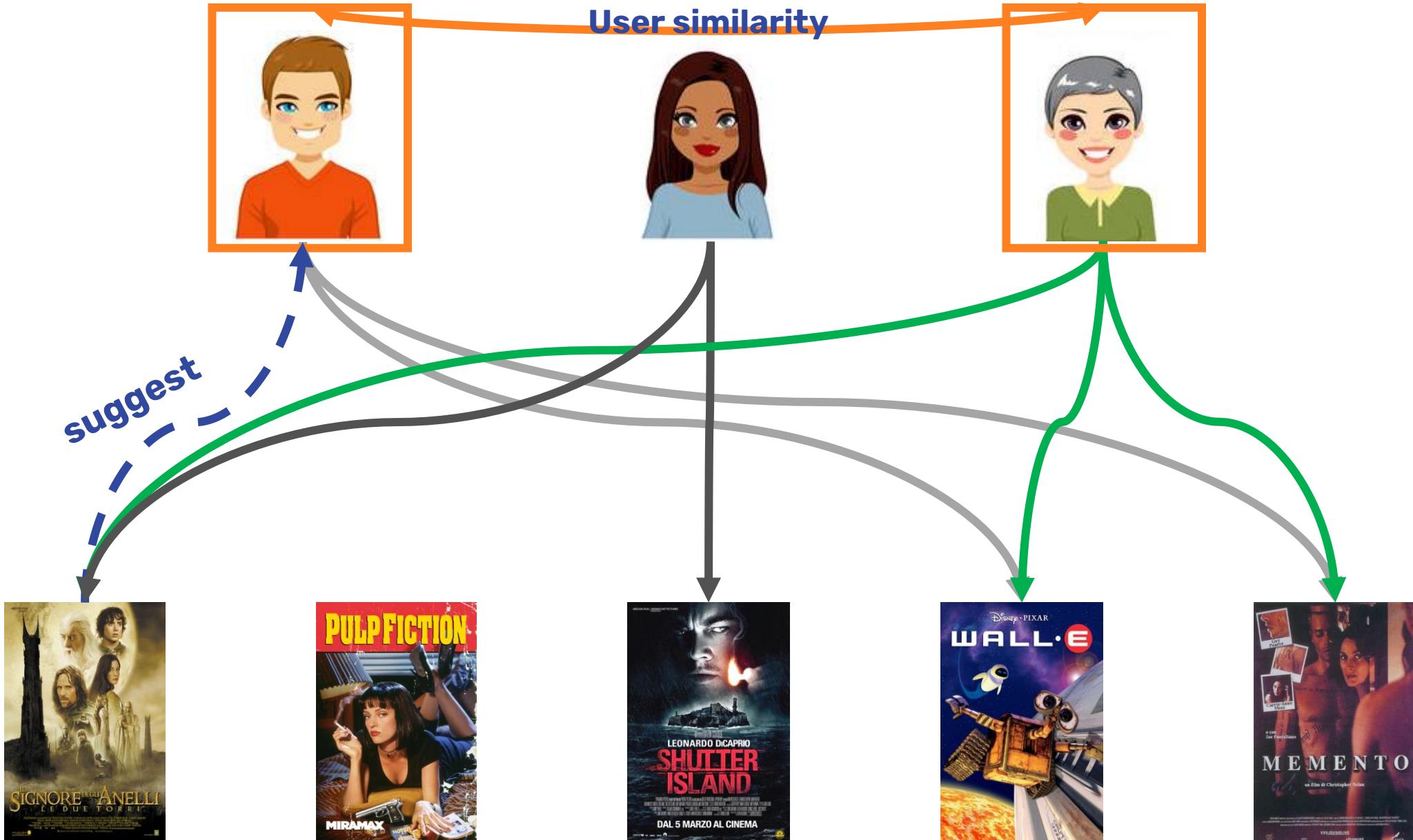
# Collaborative Filtering – User based



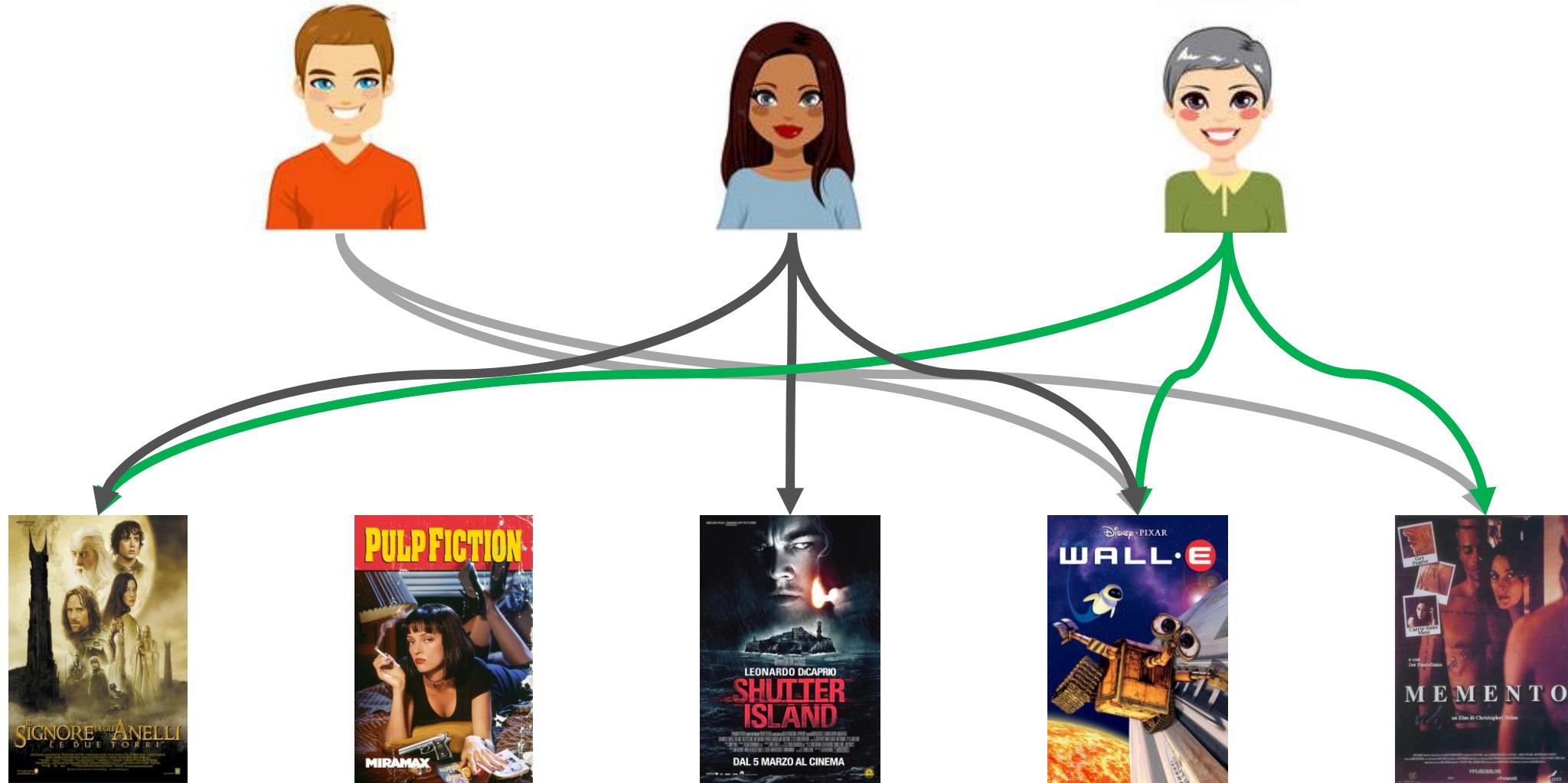
# Collaborative Filtering – User based



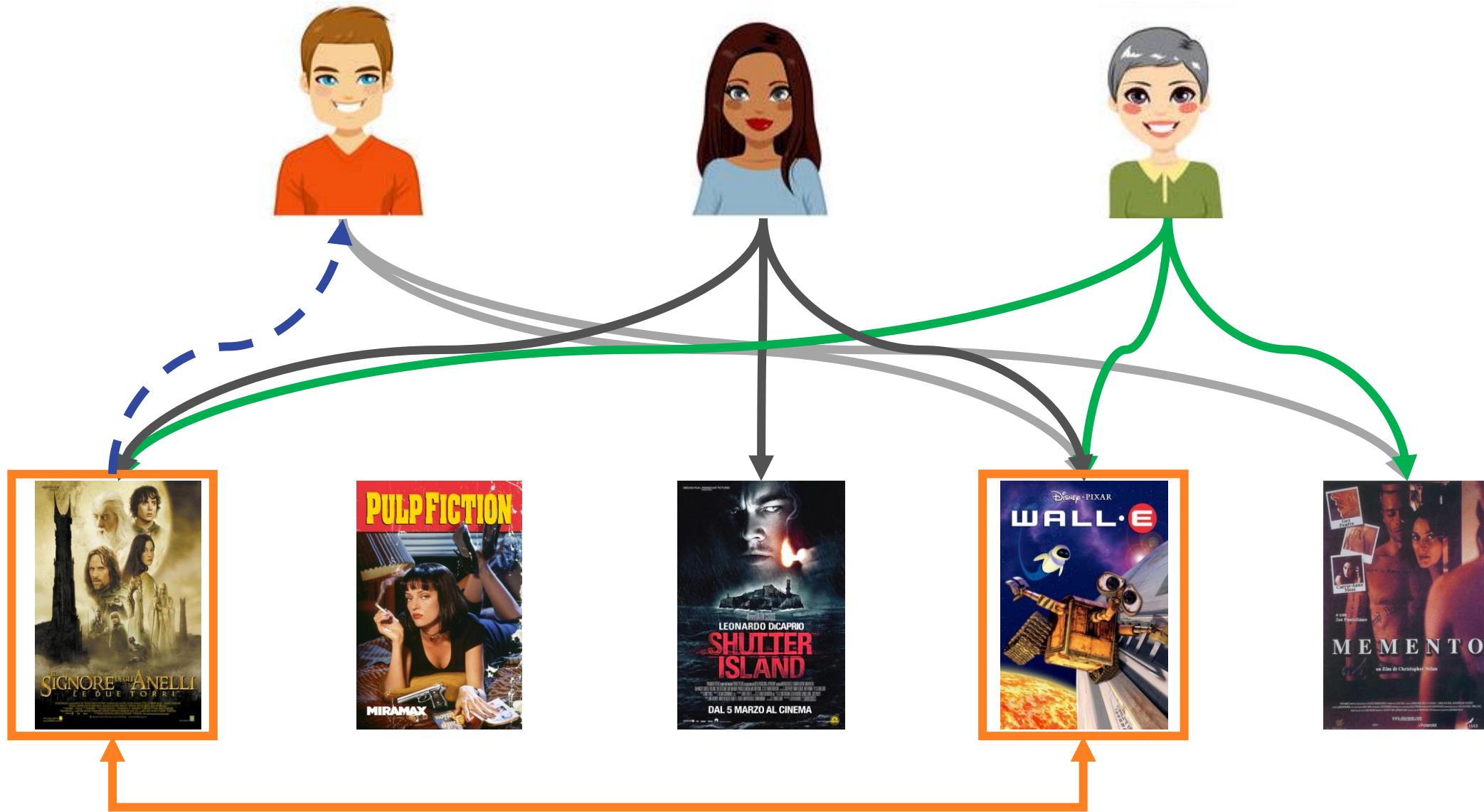
# Collaborative Filtering – User based



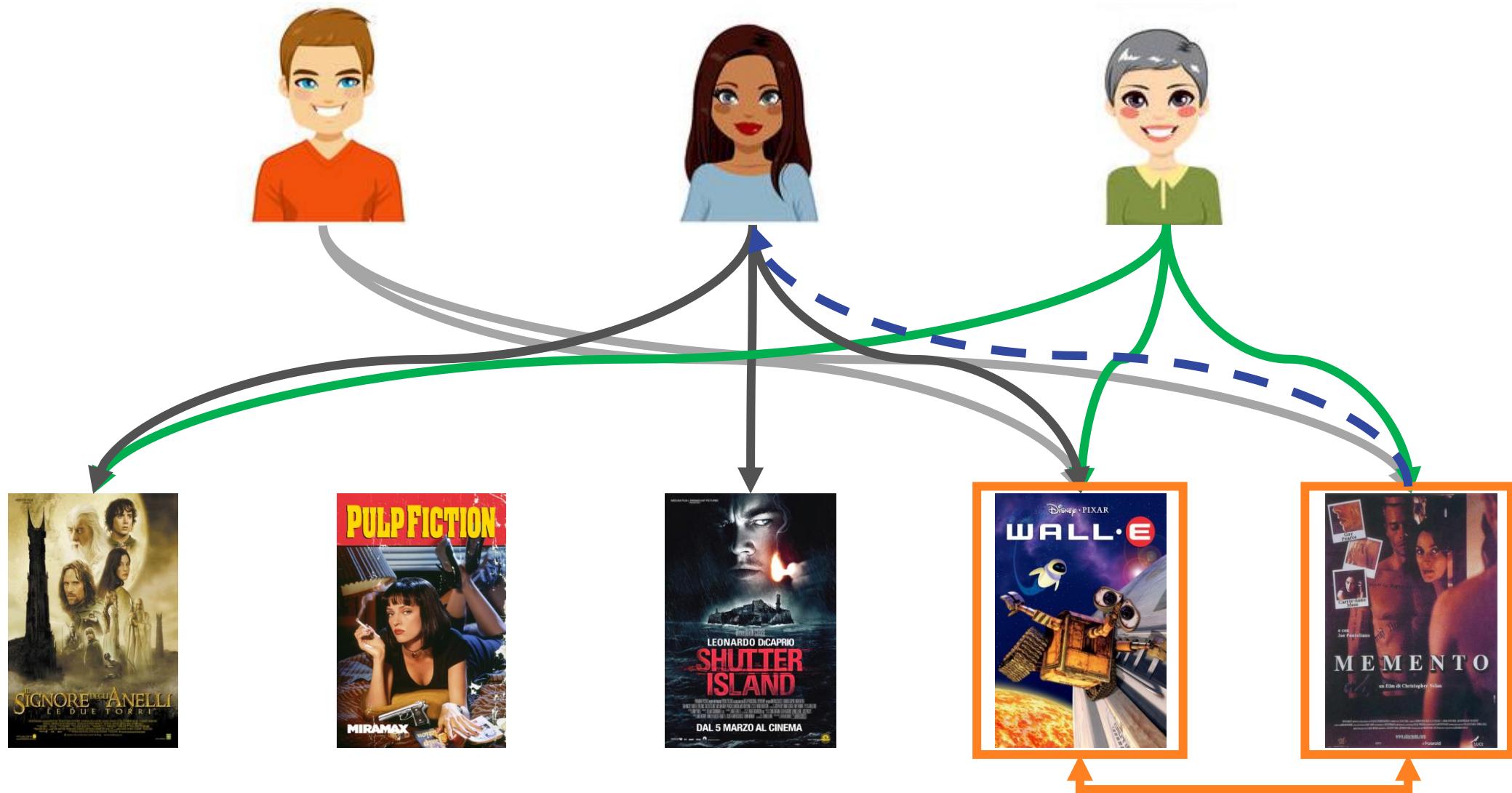
# Collaborative Filtering – User based



# Collaborative Filtering – User based



# Collaborative Filtering – User based



## User based

As seen in the example, in this case we will propose items to users again by selecting them from those appreciated by a sampling of other users with similar items.

It's a technique we can see how

- Parametric, if we try to define the tastes of a type of people
- Non-parametric, if we directly compare the tastes of other users

As distance metrics we can use

- Euclidean Distance
- Pearson's Correlation
- Cosine Similarity
- Jaccard Distance

## Item Based

Unlike user-based, item based involves comparison between items.

We will calculate a distance metric between all the items in order to find films that are similar to each other.

We will suggest to the user those similar to the films he has given high ratings to.

In the dataset, let's also try to calculate the similarity between the series based on the ratings.

# Top TV Series – ML Course 2023

Series taken from the IMDB Top 250 by number of reviews.

Reply that we try to watch the series most similar to your tastes

<https://forms.gle/Mz4cqNSQ2kTge1sw9>

Rank & Title	IMDb Rating	Your Rating
 13. Il trono di spade (2011)	9,1	
 2. Breaking Bad - Reazioni collaterali (2008)	9,4	
 99. Stranger Things (2016)	8,6	
 50. Friends (1994)	8,8	
 20. Sherlock (2010)	9,0	
 5. Chernobyl (2019)	9,3	
 126. Dexter (2006)	8,6	
 28. The Office (2005)	8,9	



## Exercise – Collaborative filtering

Who will have the most similar tastes to the professor?

<https://colab.research.google.com/drive/1mfTGtubZSfwIk4jhQcAGXnffnFUK1HZY?usp=sharing>

## Content- based filtering

Content-based filtering criteria are based on characteristics of the item itself.

We can exploit features or characteristics such as the category, the director, the actor, etc. and then provide suggestions based on these.

- Directed by **Christopher Nolan**  **Suggest The Prestige**
- Inception is a **psycho thriller**  **Suggest Shutter Island**
- The main actor is **DiCaprio**  **Suggest Titanic**

## Recap

We have finished seeing a traditional part of Machine Learning.

We will proceed in the second part of the course to revisit the problems of regression, classification and dimensionality reduction , anomaly detection , etc. via deep learning.



# **09 – Neural Networks**

Daniele Gamba

2022/2023

## Previous

We have finished seeing a roundup of classic Machine Learning techniques for problems

- Supervised
  - Regression
  - Classification
- Unsupervised
  - Dimensionality reduction
  - Recommendation

## Neural networks

In 1943, the possibility of mathematically describing the functioning of a neuron to solve logic problems was hypothesized. The fact of being able to connect multiple simple units, neurons, which in some way emulate the functioning of the brain, to create neural networks is not new.

However, the idea quickly went out of fashion and was revived only in the 1980s, under the new formulation of "*connectionism*", or the study of connections.

However, in the 1990s other Machine Learning techniques were invented, including SVM, so neural networks faded into the background.

Finally, in recent years, neural networks have come back into fashion and seem to be the way to solve a good part of the typical problems of machine learning.

# Neural networks

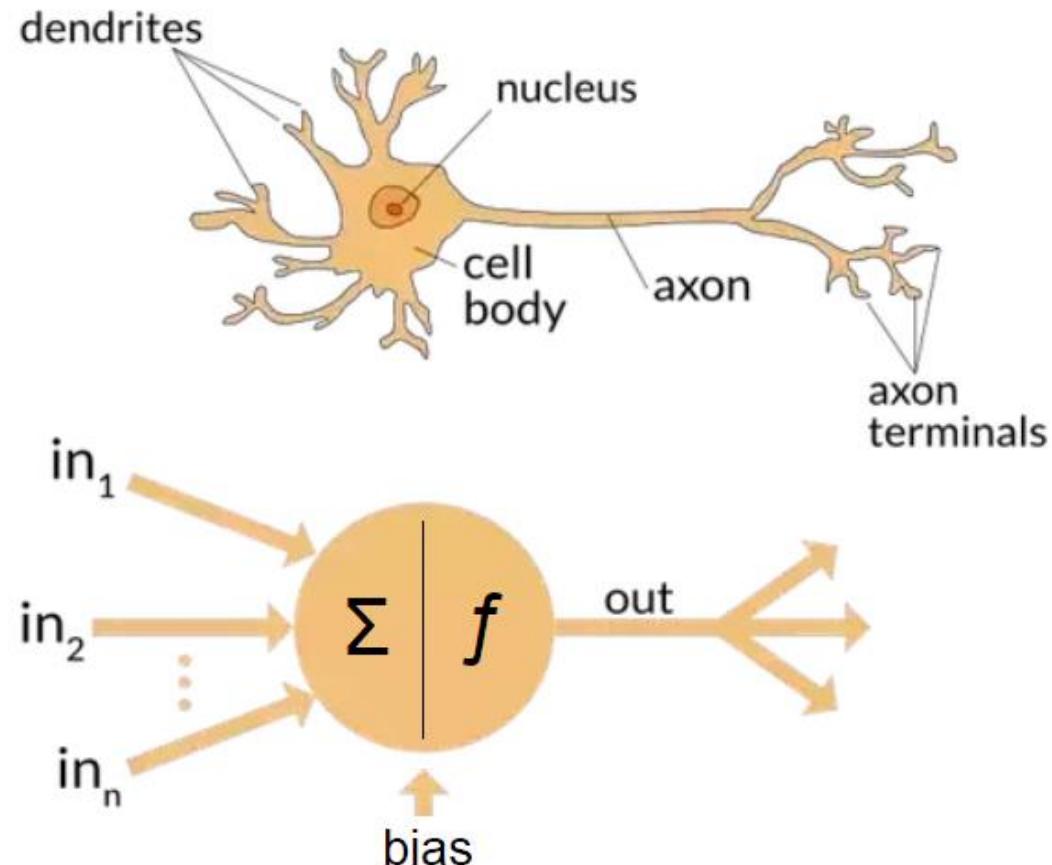
Neural networks are **universal approximators**.

As long as we connect enough neurons together and have enough data available, we can try to solve any problem.

In recent years they have established themselves due to

- Increasingly larger amounts of data, from different sources, and cheaper to collect
- Increasing computing power (especially for GPUs)
- Increasingly cheaper computing power (thanks to the cloud)

# Neuron



## Neural networks

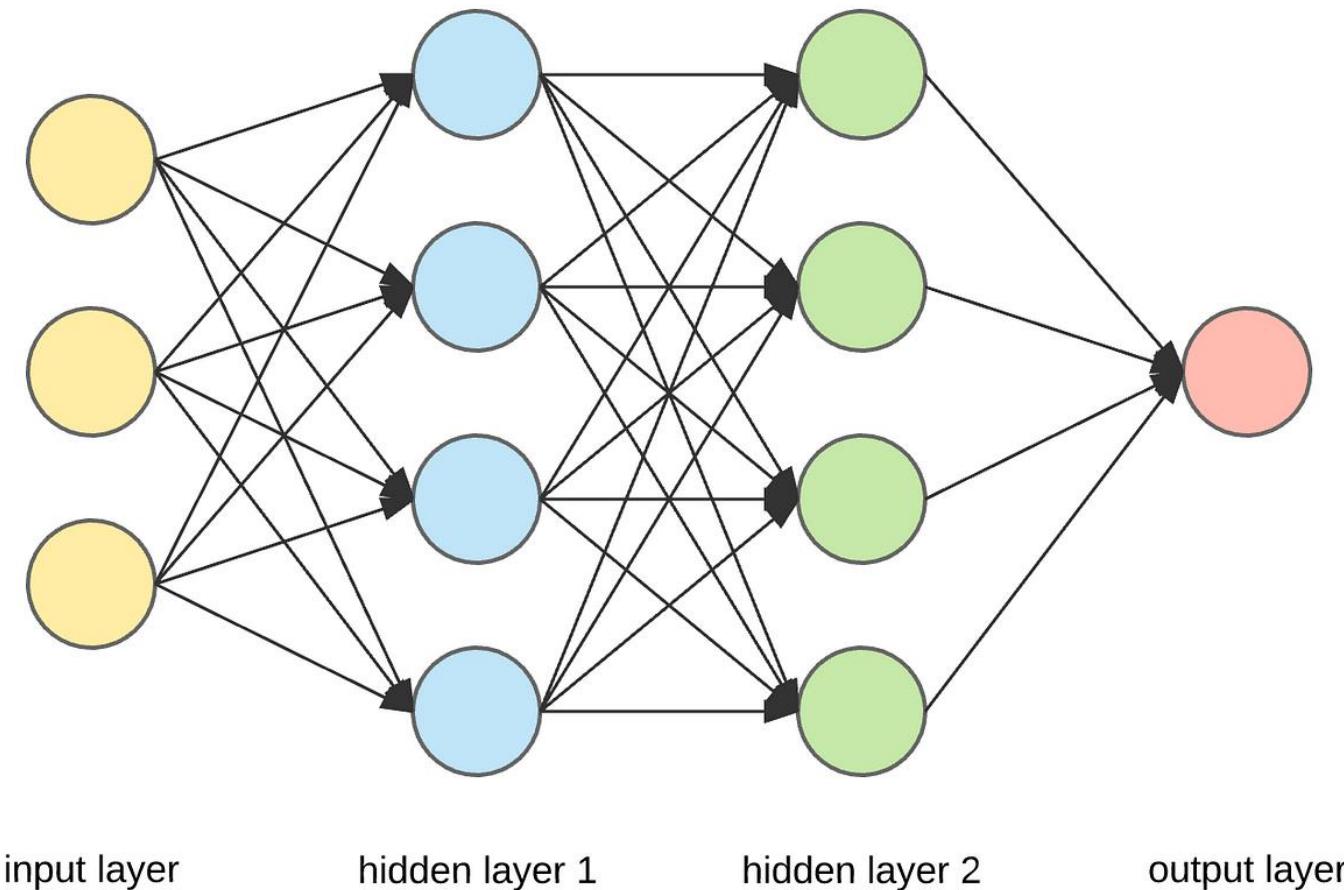
By connecting the output of a group of neurons with the input of other neurons we obtain a **neural network**.

Each group of neurons on the same level is called **a layer**.

The first input layer is called the input layer and is connected to our data,  $x$ .

The outputs of the last layer are the model outputs and correspond to the  $\hat{y}$ .

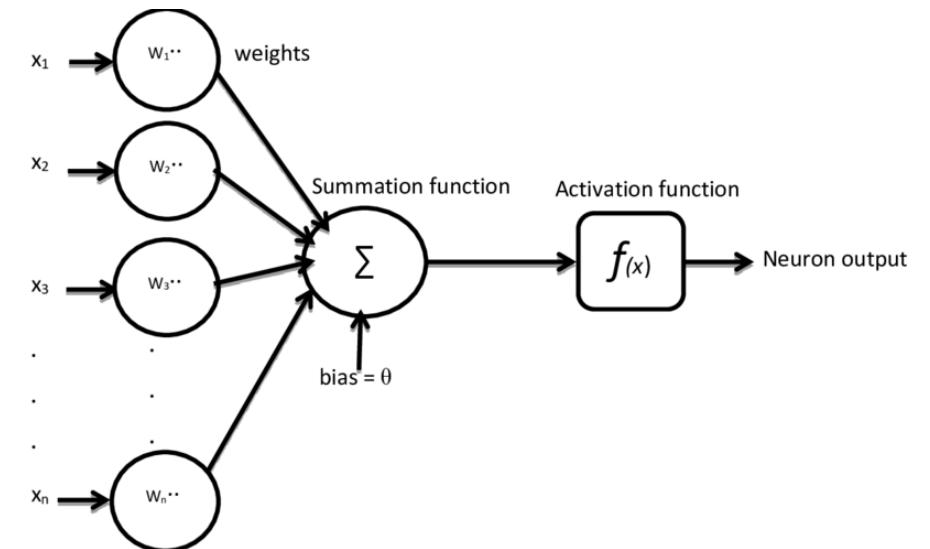
# Neural networks



# Activation function

Every neuron

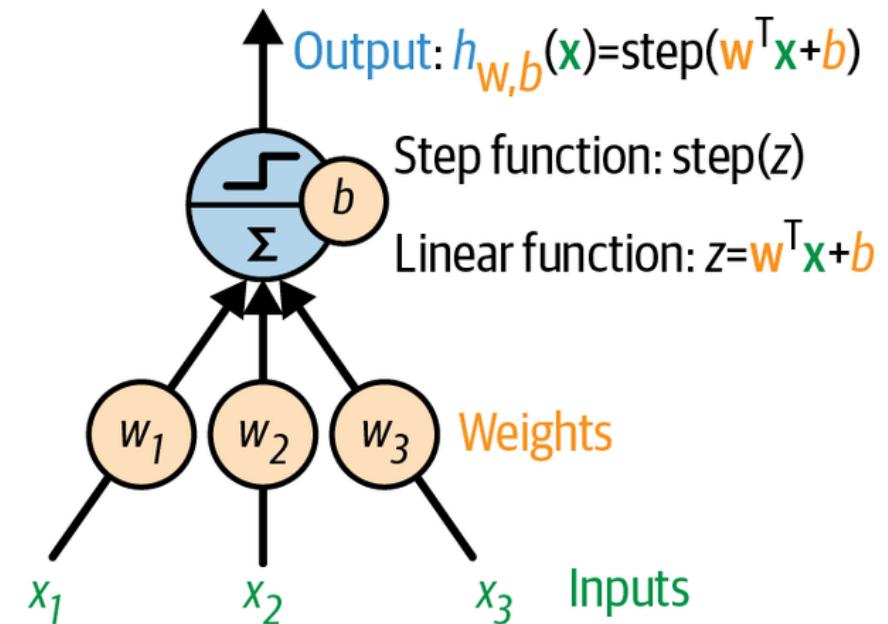
- It will accept inputs
- It will weigh them for a weight  $w$
- It will add all the results
- It will apply a **toggle function**
- Will share his output



## Activation functions

In the parts of ML applied to regression and classification we have already seen some activation functions. The most famous in the field of neural networks is the Rosenblatt perceptron .

The activation function corresponded to the sign function. It is produced as output +1 or -1 simply if the result of the internal summation is greater or less than zero.



## Activation functions

Other activation functions that we have seen and that are used a lot are

- The linear function
- The sigmoid
- The hyperbolic tangent
- ReLU
- ...



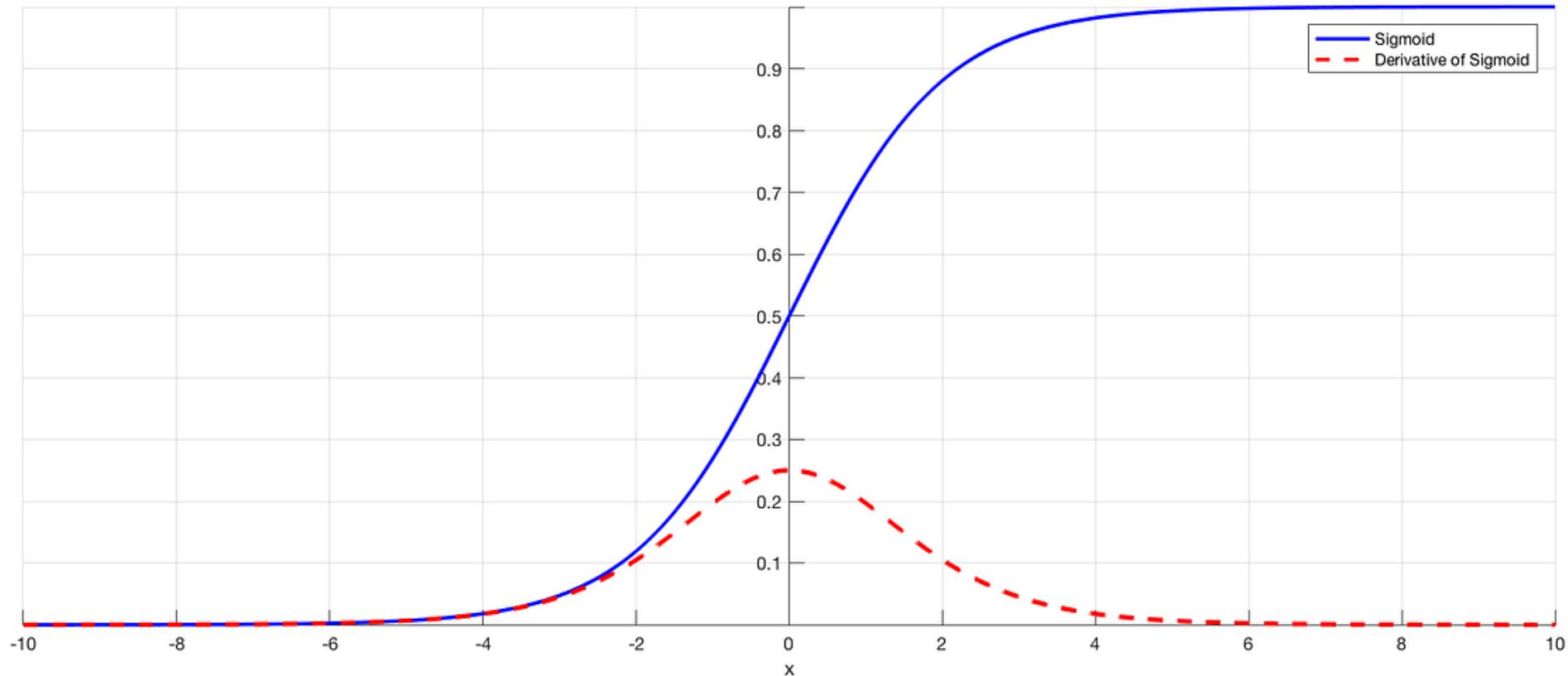
## Features of the activation function

The activation function transforms data into weighted inputs to generate an output.

In neural networks we want our algorithm to be able to deal with the deep non-linearities present in the data, so we will look for non-linear activation functions which, if applied, one layer after another, allow us to approximate even very complex functions.

One aspect to keep in mind is that our activation function must be differentiable and that its derivative somehow favors the learning of the network, i.e. that its derivative is never zero.

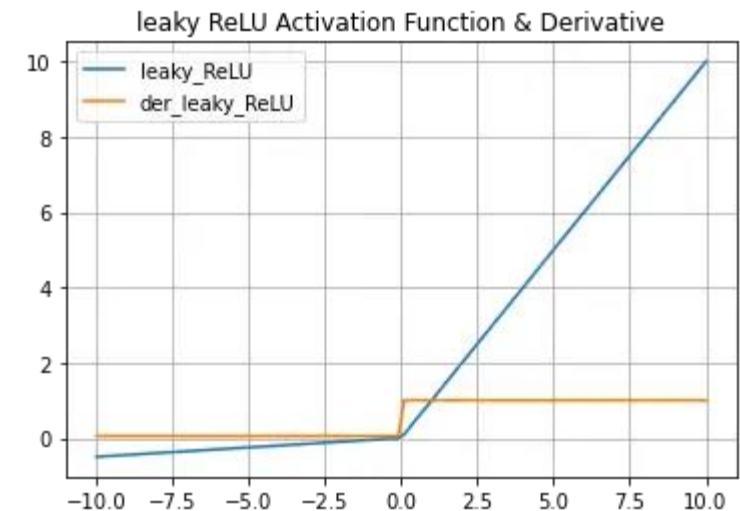
## Derivative of the sigmoid



## Derivative

Sigmoid , ReLU , and many other activation functions suffer from the fact that if we are very far from the non-linearity area (i.e. very large or very negative values in the second case), the derivative does not help us to correct the value of the weights .

Functions like Leaky-ReLU have a small but non-zero derivative in their entire space.



## Linear activation

The linear activation function is nothing more than the linear combination of the inputs.  
Using multiple consecutive layers of linear activations does not add any information to the model, it can be immediately approximated with a single layer .

layer neural network with linear activations.

In some cases it is still useful to use linear activation.

Example: the last layer of a neural network for a regression problem where we are not sure of the upper or lower bounds and by normalizing the output we would not be able to cover all possible values in production.

## Neural networks

Let's now try to play with a neural network already implemented for regression and classification problems, let's see how the different number of neurons, layers and activation functions impact

<https://playground.tensorflow.org/>

## Multi-Layer Perceptron - MLP

The Multi-Layer Perceptron , or MLP, is the type of neural network we have seen so far.

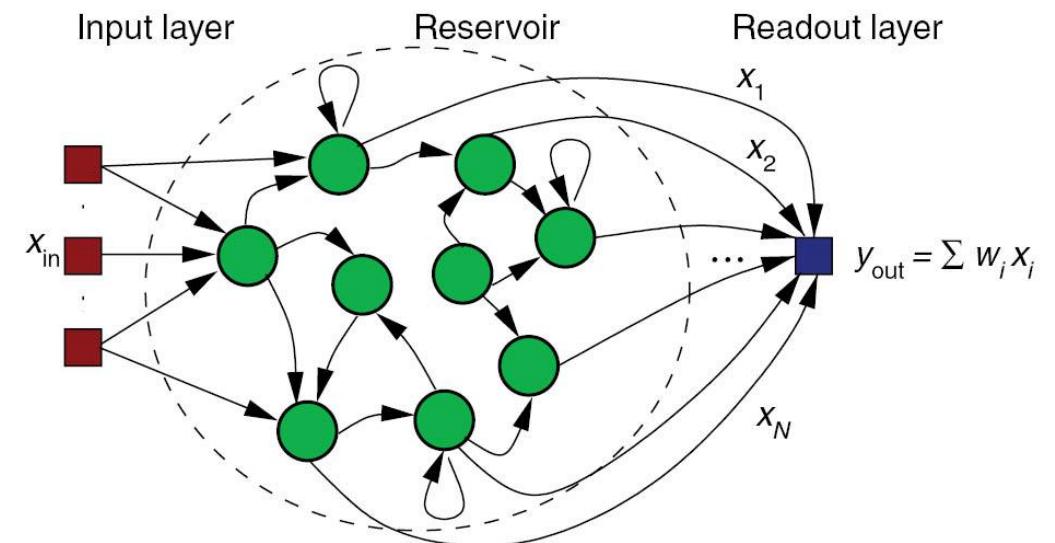
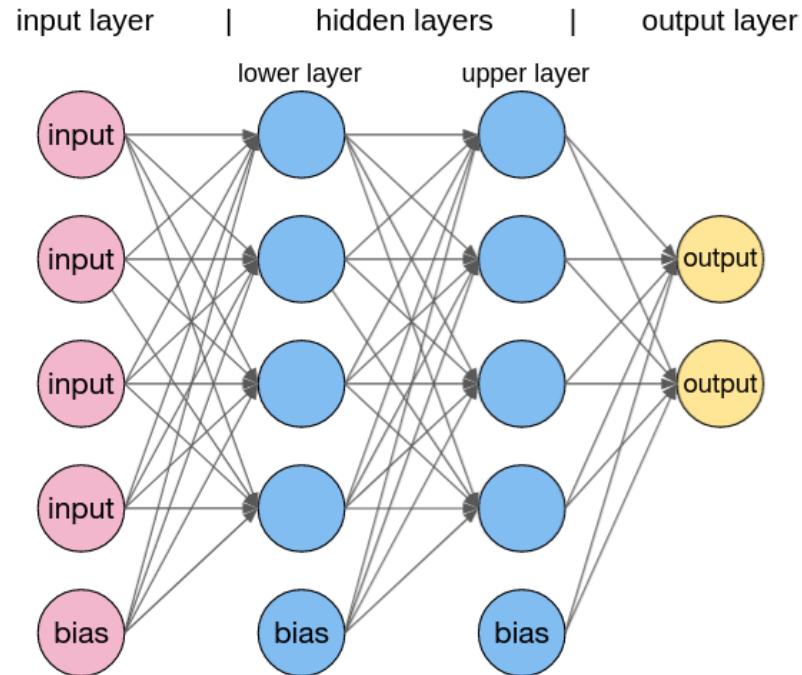
All neurons from the previous layer are connected to the next layer .

This formulation is very convenient because it allows us

- to easily define our layers
- to address very nonlinear problems by simply adding more layers
- to parallelize the accounts of each layer as each node is independent

In contrast, the number of parameters wwithin the model grows very rapidly.

## MLP vs rest



In reservoir networks there is no distinction between one layer and another, so to process a neuron we must process the entire sequence.

# Learning

So far we have seen how to define a neural network, but how does it learn?

The learning process of a neural network is called back- propagation and is made up of these steps

- A batch of data is taken (e.g. 16 data)
- It is inserted into the input layer and passed through the network until the outputs are produced (forward pass)
- Intermediate results are saved
- The prediction error is measured using the **loss functions**
- layer contributes to the error is calculated
- The average adjustment for each neuron is calculated and reapplied via gradient descent
- Recalculating the values that should have obtained the activations of the previous layer , we repeat for each layer up to the input

## Learning

Going into a little more detail, starting with our predictions and our prediction error

$$y_k = \sum_i w_{ki} x_i \quad E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2$$

The gradient with respect to weight is

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}$$

But each neuron is an activation (  $wx$  ) passed through an activation function (  $h$  ), so each output is

$$a_j = \sum_i w_{ji} z_i \quad z_j = h(a_j).$$

## Learning

Calculating that we will have to have the derivative of the composite function to modify our weights, we exploit the chain rule and define  $\delta_j$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}. \quad \delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

we'll have

$$\frac{\partial a_j}{\partial w_{ji}} = z_i. \quad \frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i.$$

and after a couple of manual steps

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

with  $\delta_j$  equal to the error to apply to our weight

## Attention

There are a few points to pay attention to when training a neural network

- All weights must be **initialized randomly**, otherwise it would not learn anything. If two weights were exactly identical within the same layer, their contribution would be identical and halved.
- It trains on **batches** of data, randomly sampled, according to **stochastic gradient descent** otherwise each training example would vary each neuron. Once all the batches of the training set are finished, one epoch is finished and the next one starts again.

All the considerations made for other machine learning techniques remain valid

- Scaling or normalizing the input features helps enormously
- We easily suffer from overfitting due to the high number of weights, so we can use regularization techniques
- The dataset better be balanced appropriately
- The learning rate has a great impact and often needs to be varied during learning.

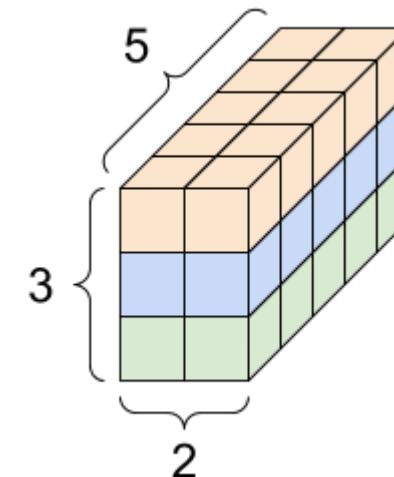
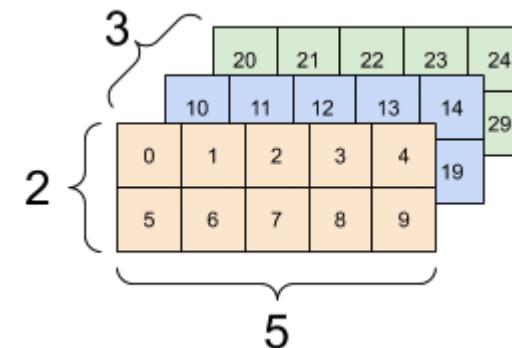
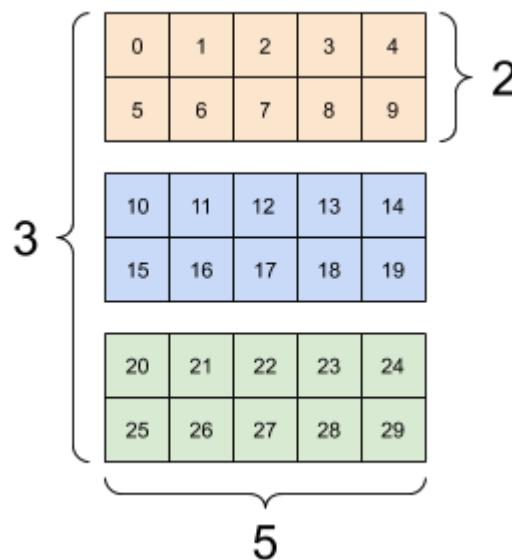
## Introduction to Tensorflow

More flexible and low-level frameworks are used to build and train neural networks than Scikit-Learn . The two main frameworks are **Tensorflow** (by Google) and **PyTorch** (ex. Facebook, now Linux Foundation). In the course we will mainly see Tensorflow 2, possibly with some references to PyTorch , the basic concepts are identical, the implementation changes slightly.

With these frameworks we build the Machine Learning algorithm ourselves instead of finding it already implemented.

# Tensors

Let's start with the concept of a tensor, which is an in-memory representation of an N-dimensional, immutable object



<https://www.tensorflow.org/guide/tensor?hl=it>

## Definition of a model

Our algorithm will consist of three parts

- A network structure
- A loss function to optimize
- An optimizer that applies loss to vary weights

## Network structure

In our MLP the network structure is defined by

- The number of layers
- How many neurons will each layer have
- Which activation function do we choose for each layer
- Any function layers ( Flatten , DropOut , etc. which we will see later)
- Any regularizations
- What each layer is connected to

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 3072)	0
dense_6 (Dense)	(None, 200)	614600
dense_7 (Dense)	(None, 150)	30150
dense_8 (Dense)	(None, 10)	1510

Total params: 646,260
Trainable params: 646,260
Non-trainable params: 0

## Loss function

With the Loss function we are going to define the cost function that we will have to optimize, the larger it is the further we will be from optimal.

We can exploit a set of standard cost functions such as

- MeanSquaredError , MeanAbsoluteError , MAPE, etc. for regression problems
- BinaryCrossentropy , CategoricalCrossentropy , CosineSimilarity , etc. for classification problems
- and numerous others

Or we can build a custom loss function , such as a combination of these for different parts of our algorithm.

## Metrics

loss function , we can have the model automatically calculate metrics that can give us information about our model.

The difference is that we will not optimize against these metrics, they are simple measures.

For example, in classification problems

- AUC
- Accuracy
- Precision, recall

# Optimizers

When we go to do back-propagation we have to decide how to apply the gradient.

Over the years, several optimizers have been developed that update the weights slightly differently

## **Stochastic Gradient Descent**

It is the traditional one that we have already seen in traditional machine learning too. A "momentum" is added to the learning rate which takes into account the speed with which the gradient is descended. It will lower the learning rate as optimization slows down.

## **RMSProp**

Similar to SGD, it calculates a moving average of the squares of the gradients which it uses to divide, under the root, the learning rate. It's another way of looking at how quickly you're moving down the gradient.

## **Adagrad**

This is the same as RMSProp in setting but a moving average is saved for each direction. In this way different parameters are updated with differently weighted learning rates, making it more robust to variations in "scale" between features.

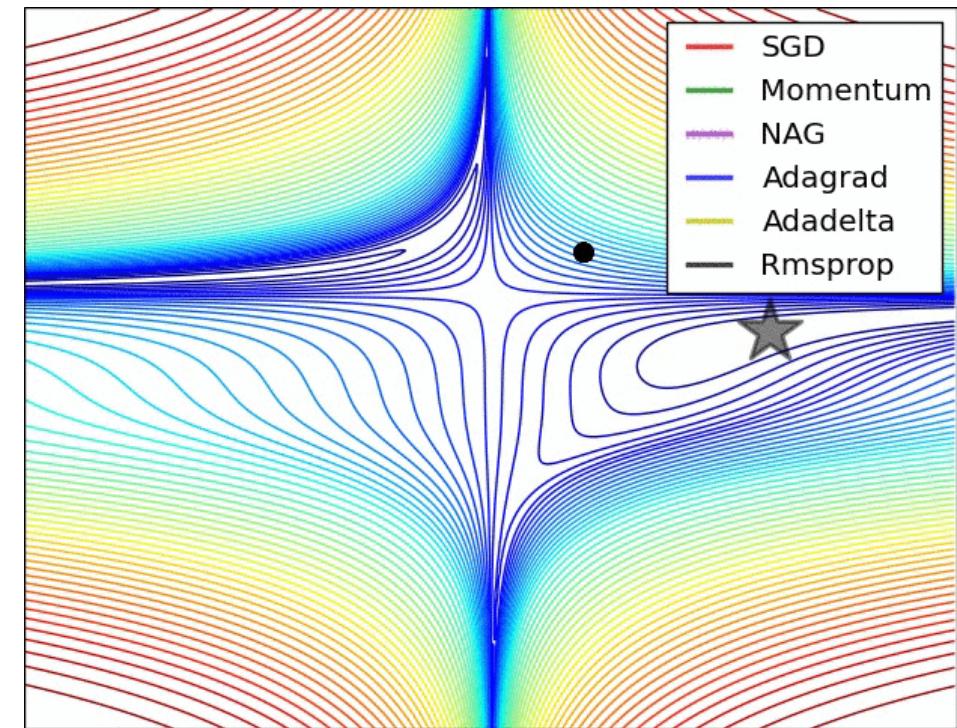
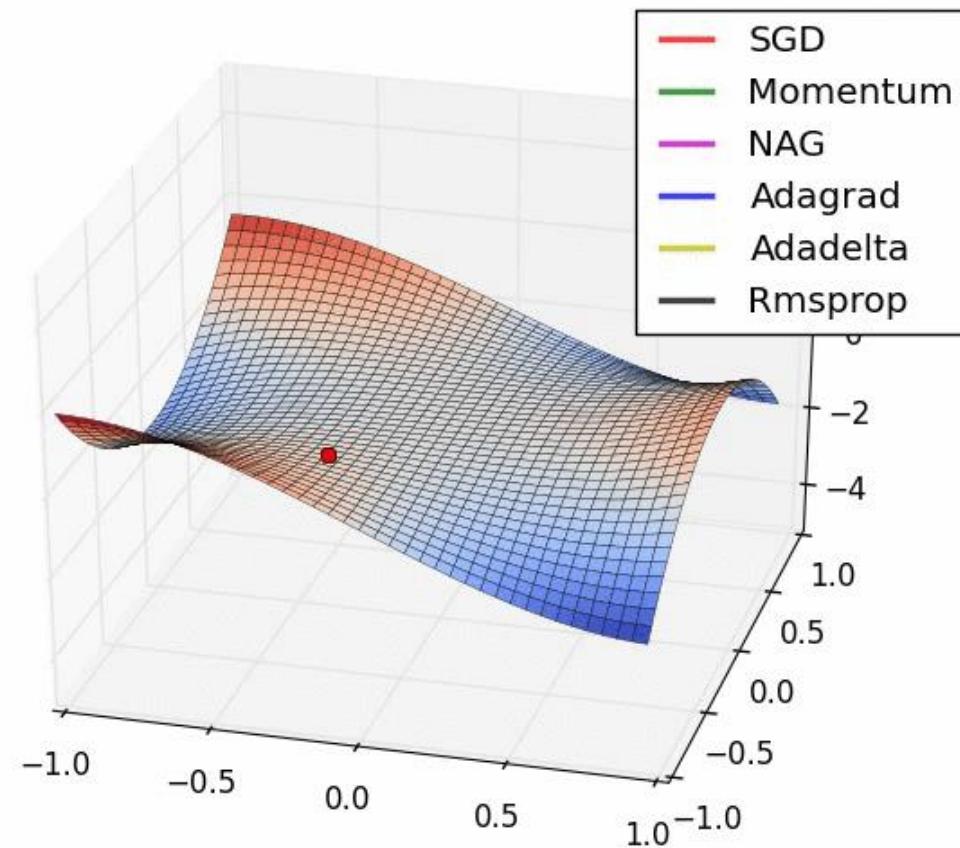
# Optimizers

## Adam

It is one of the most used and adds to the RMSProp and Adagrad setting a cumulative gradient history to be used as the SGD momentum . In general we start by using this and then vary the different parameters.

All these optimizers in turn have parameters to look for, first of all the learning rate and then all the others. The choice of the optimizer and its hyperparameters is critical because it determines the speed with which the network learns. The larger we have structures, the more decisive it is.

# Optimizers



## Exercise

Let's apply an optimizer to a variable to optimize it and try to build an MLP both with Sklearn MLPRegressor than with Tensorflow to see the differences.

<https://colab.research.google.com/drive/1IEK-KNBY5qx-CthQag5yIMliron-4Yie?usp=sharing>



# **10 – Neural Networks**

Daniele Gamba

2022/2023

## Previous

We started to see

- What is a neural network
- Concept of neuron and layer of neurons
- Activation functions
- Optimizers
- Loss and metrics

## Exe

Let's build an MLP, compile it, and train it to predict house prices

[https://colab.research.google.com/drive/1BemOYYUABKNIBKKoy7y\\_Hd6VAJ9ttpjj?usp=sharing](https://colab.research.google.com/drive/1BemOYYUABKNIBKKoy7y_Hd6VAJ9ttpjj?usp=sharing)

## MLP

We saw in the last lesson that the MLP is a series of layers , all connected to each other, with one or more inputs and one or more outputs.

**dense** neurons are connected to each other but we can freely apply the concept to recreate different useful blocks.

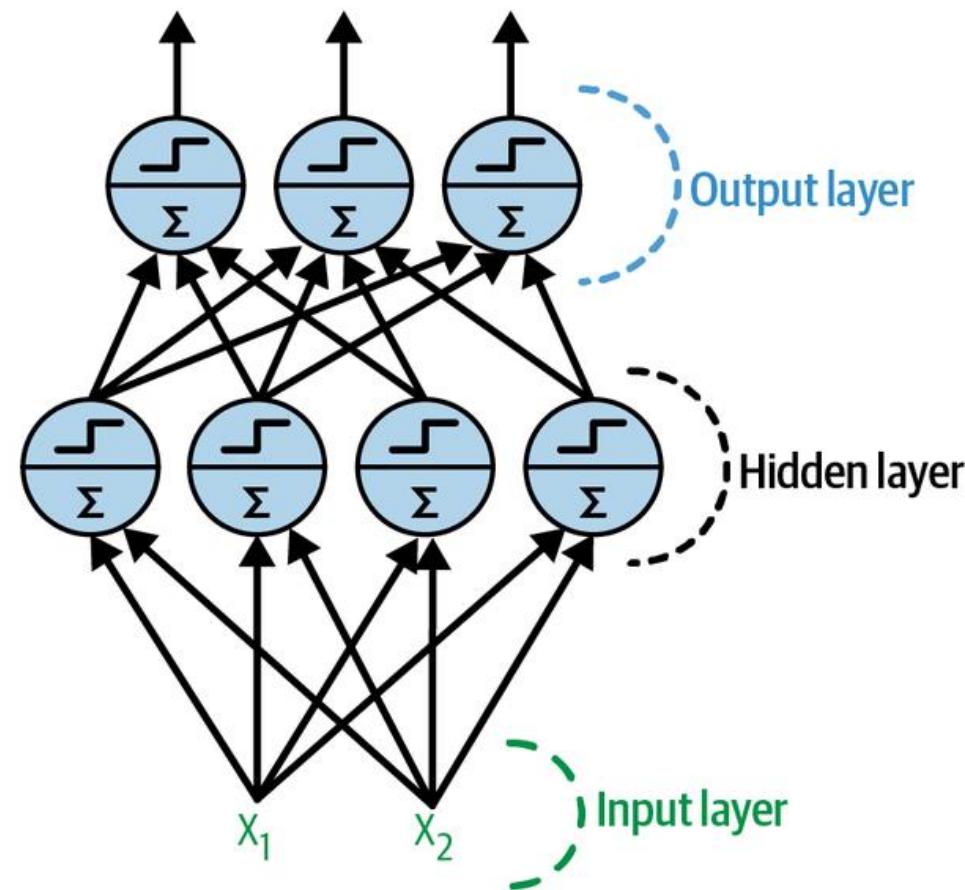
# Sequential model

The basis is a **sequential model**, i.e. in which all the layers are connected one after the other.

To define this type of models we can rely on the Keras Sequential model in which we simply have to specify which layers we want and the class will implement it.

```
norm_layer = tf.keras.layers.Normalization(input_shape=X_train.shape[1:])
model = tf.keras.Sequential([
    norm_layer,
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(50, activation="relu"),
    tf.keras.layers.Dense(1)
])
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)
model.compile(loss="mse", optimizer=optimizer, metrics=["RootMeanSquaredError"])
norm_layer.adapt(X_train)
history = model.fit(X_train, y_train, epochs=20,
                     validation_data=(X_valid, y_valid))
mse_test, rmse_test = model.evaluate(X_test, y_test)
X_new = X_test[:3]
y_pred = model.predict(X_new)
```

## Sequential model

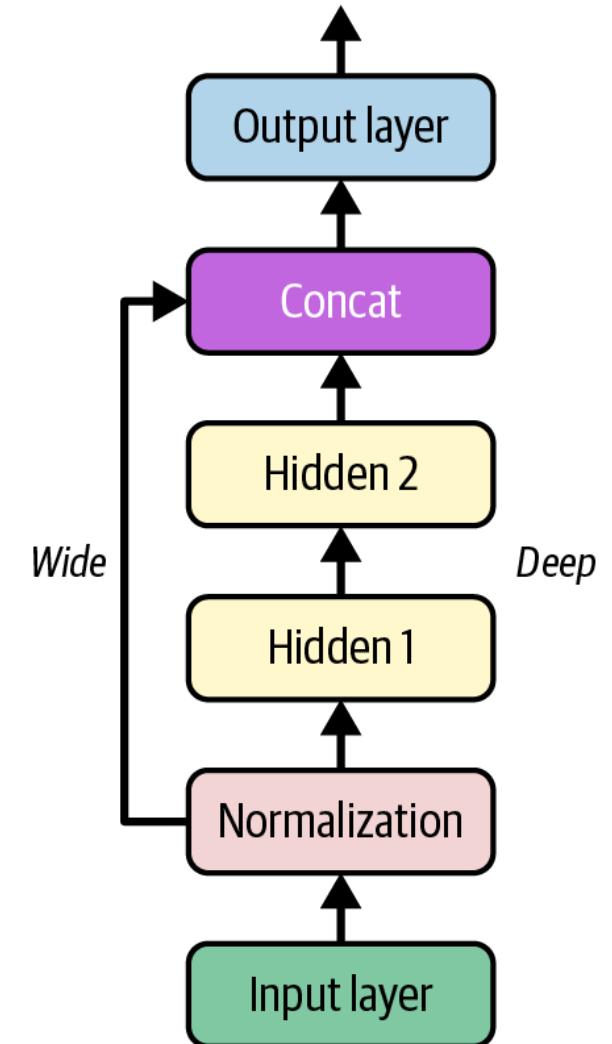


## Wide and deep

Another structure that we can create is the **Wide and Deep**.

In this case the model passes the input both through a deep neural network and directly to the output.

In this way our output layer will be able to exploit information that arrives directly from the output and "deeper" information that arrives through the structure of the network.



## Wide and deep

In this case the network is no longer entirely sequential because the input has also passed directly to the output.

layers

For this we must specify the layers and then specify how they connect to each other.

connections

```
normalization_layer = tf.keras.layers.Normalization()  
hidden_layer1 = tf.keras.layers.Dense(30, activation="relu")  
hidden_layer2 = tf.keras.layers.Dense(30, activation="relu")  
concat_layer = tf.keras.layers.concatenate()  
output_layer = tf.keras.layers.Dense(1)
```

```
input_ = tf.keras.layers.Input(shape=X_train.shape[1:])  
normalized = normalization_layer(input_)  
hidden1 = hidden_layer1(normalized)  
hidden2 = hidden_layer2(hidden1)  
concat = concat_layer([normalized, hidden2])  
output = output_layer(concat)
```

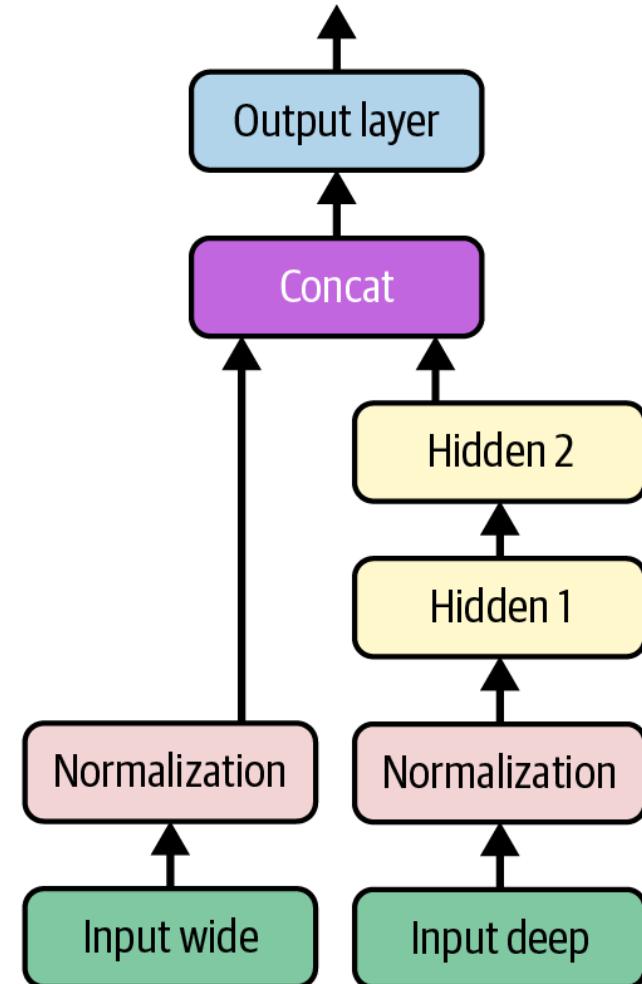
model

```
model = tf.keras.Model(inputs=[input_], outputs=[output])
```

## Multiple inputs

Let's say that we have a price of a house that comes from a general model and that we want to refine with our rougher features. In this case we would have two different inputs that we can process with two different flows in the network.

We can easily define two different inputs and thus build our model.



## Multiple inputs

In the case of multiple inputs we will have to pay attention when we create the model to indicate both layers as input and to concatenate them.

It is also important during the inference phase to pass the data in the right order as we imagined their path.

inference

```
input_wide = tf.keras.layers.Input(shape=[5]) # features 0 to 4
input_deep = tf.keras.layers.Input(shape=[6]) # features 2 to 7
norm_layer_wide = tf.keras.layers.Normalization()
norm_layer_deep = tf.keras.layers.Normalization()
norm_wide = norm_layer_wide(input_wide)
norm_deep = norm_layer_deep(input_deep)
hidden1 = tf.keras.layers.Dense(30, activation="relu")(norm_deep)
hidden2 = tf.keras.layers.Dense(30, activation="relu")(hidden1)
concat = tf.keras.layers.concatenate([norm_wide, hidden2])
output = tf.keras.layers.Dense(1)(concat)
model = tf.keras.Model(inputs=[input_wide, input_deep], outputs=[output])

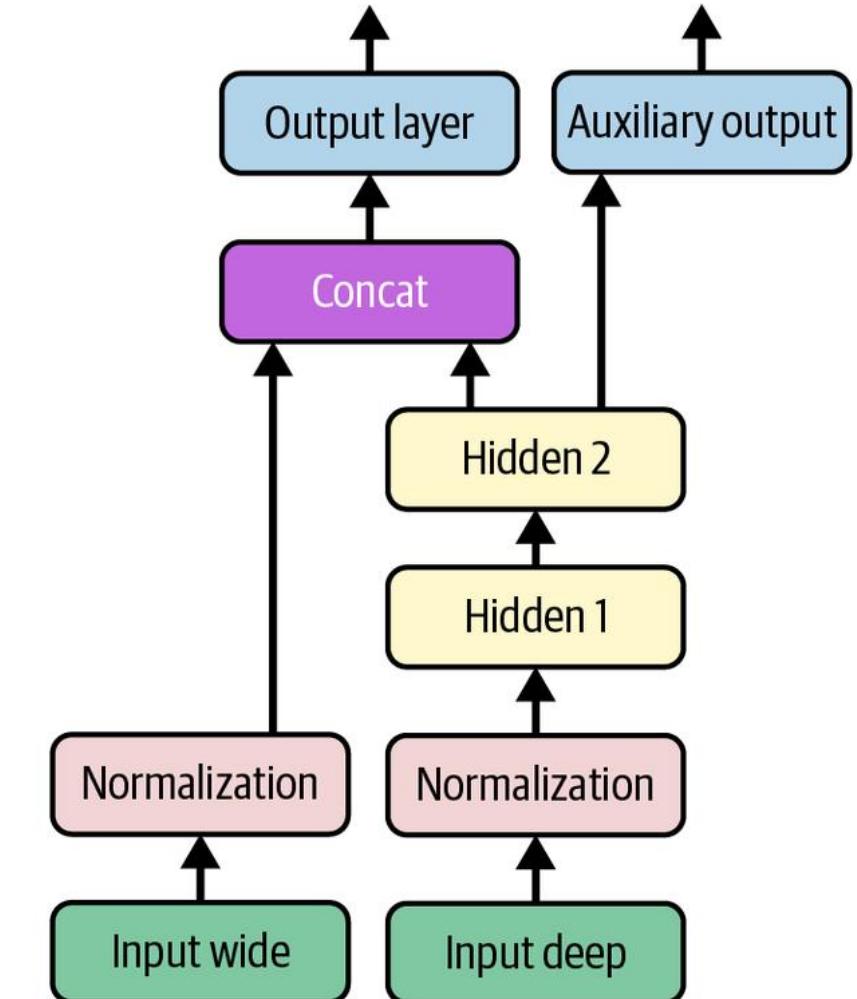
X_train_wide, X_train_deep = X_train[:, :5], X_train[:, 2:]
X_valid_wide, X_valid_deep = X_valid[:, :5], X_valid[:, 2:]
X_test_wide, X_test_deep = X_test[:, :5], X_test[:, 2:]
X_new_wide, X_new_deep = X_test_wide[:3], X_test_deep[:3]

norm_layer_wide.adapt(X_train_wide)
norm_layer_deep.adapt(X_train_deep)
history = model.fit((X_train_wide, X_train_deep), y_train, epochs=20,
                     validation_data=((X_valid_wide, X_valid_deep), y_valid))
mse_test = model.evaluate((X_test_wide, X_test_deep), y_test)
y_pred = model.predict((X_new_wide, X_new_deep))
```

## Multiple outputs

Clearly just as we can have multiple inputs we can specify different layers as model outputs.

Let's say we want not only the prediction but also the corrections that our model predicts. We can specify a second output layer and add it to the model.



## Multiple outputs

In the case of multiple outputs we will need to specify **a loss function** for each output and consequently a target against which to optimize.

Just as with the inputs we will have to remember in train and inference to pass all the data in the right order.

model |

```
[...] # Same as above, up to the main output layer  
output = tf.keras.layers.Dense(1)(concat)  
aux_output = tf.keras.layers.Dense(1)(hidden2)  
model = tf.keras.Model(inputs=[input_wide, input_deep],  
outputs=[output, aux_output])
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)  
model.compile(loss=["mse", "mse"], loss_weights=(0.9, 0.1), optimizer=optimizer,  
metrics=["RootMeanSquaredError"])
```

## Output layers

We have seen how the output layer represents our prediction  $\hat{y}$

In the case of **regression** / forecasting problems of a single variable we can use a **layer linear** output to directly predict the value or use other activation functions possibly rescaling the data.

In the case of **single class classifications** we can use a **single neuron with a sigmoidal** activation function to represent its probability between 0 and 1.

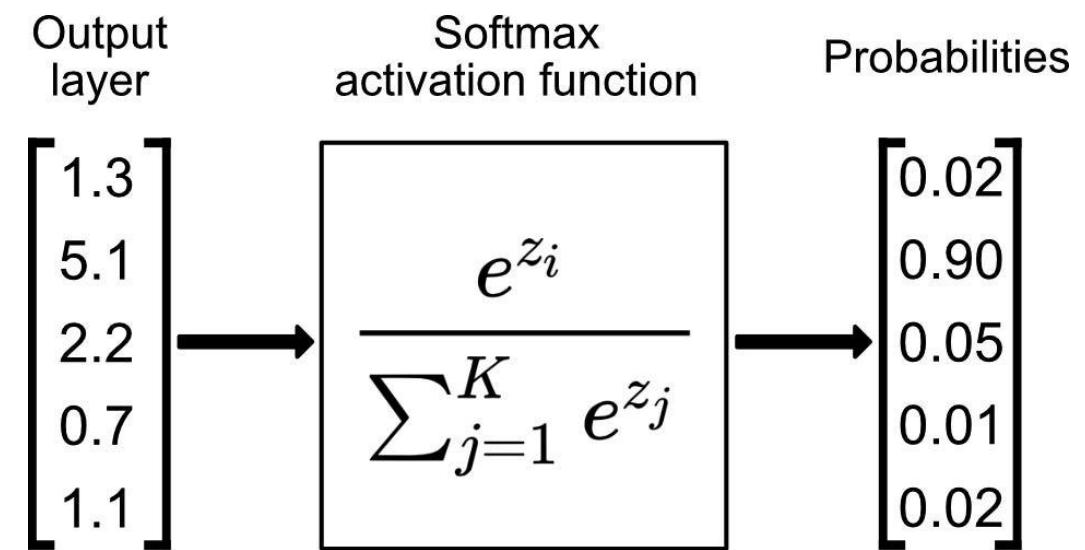
In the case of **multiclass problems classification** we must distinguish

- Single label, i.e. our example can only belong to one class (e.g. dog, cat, rabbit, ..)
- Multi label, i.e. our example can belong to multiple classes (healthy, functioning, correct consumption, ..)

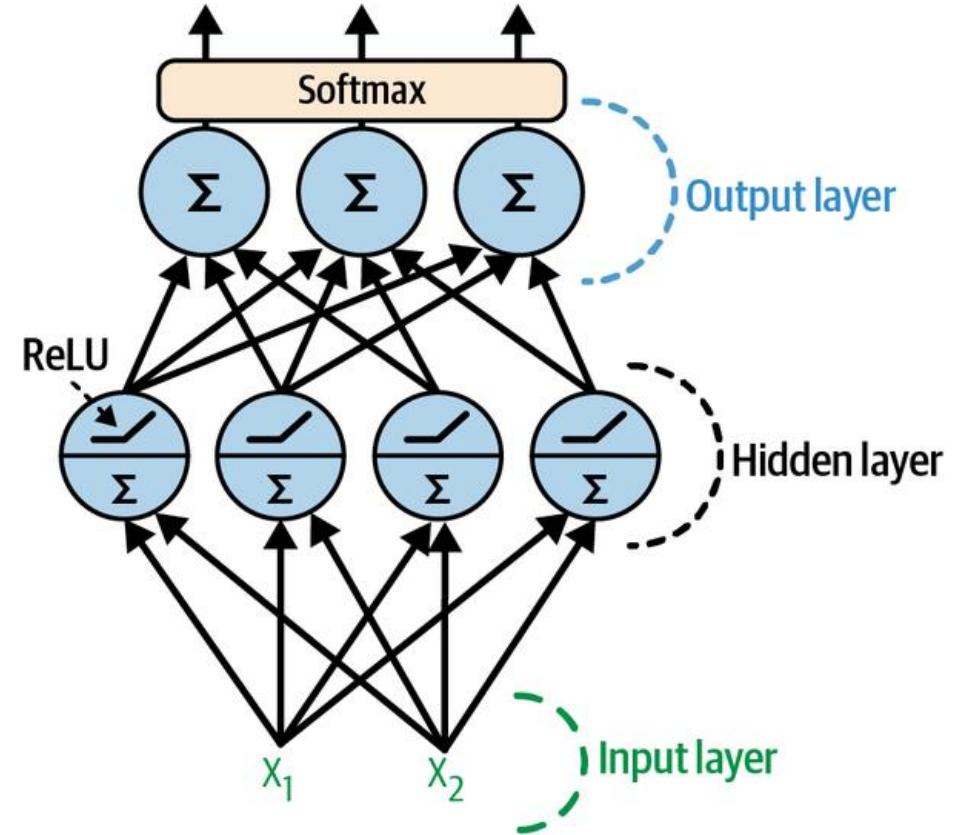
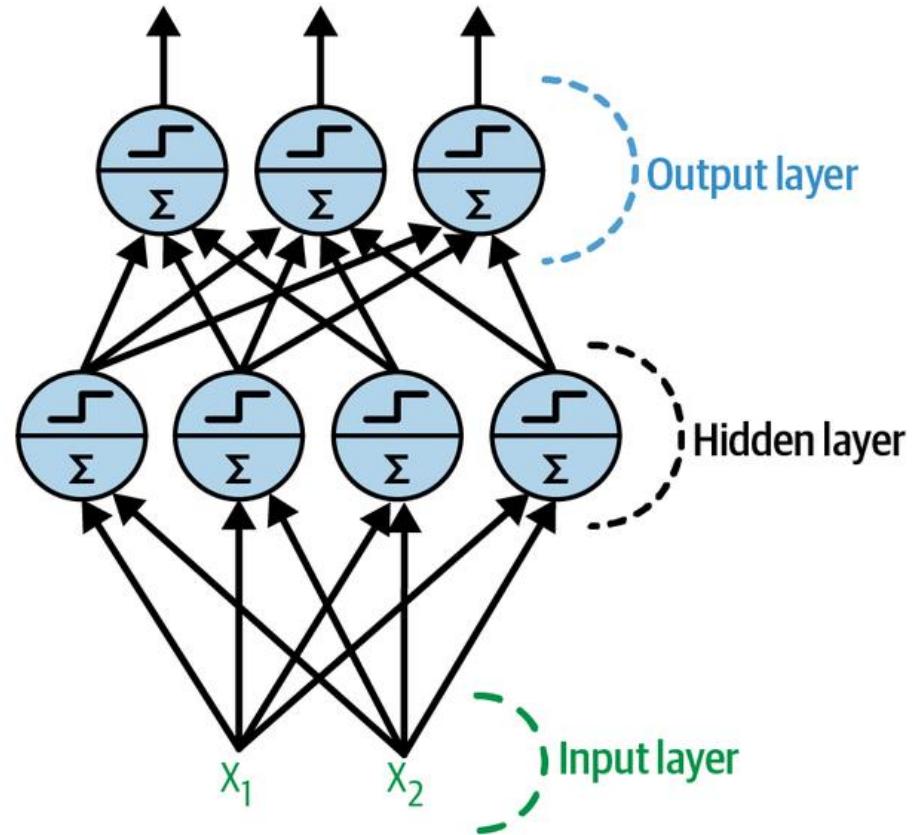
## Output layers

In the single-label case with N classes we can use the softmax which allows us to choose the one with the highest value as the output class, rescaling the probabilities according to the relative values.

In the multi-label case, each neuron will instead have its own independent activation function.



## Softmax vs Independent



## Vanishing gradient

Another of the problems encountered in training networks is called "disappearing gradient".

When we use activation functions like sigmoid or hyperbolic tangent the derivative tends to be very small as we move away from the center. During the learning phase, the chain rule will multiply the partial derivatives of each parameter together, thus resulting in the product between them of very small values.

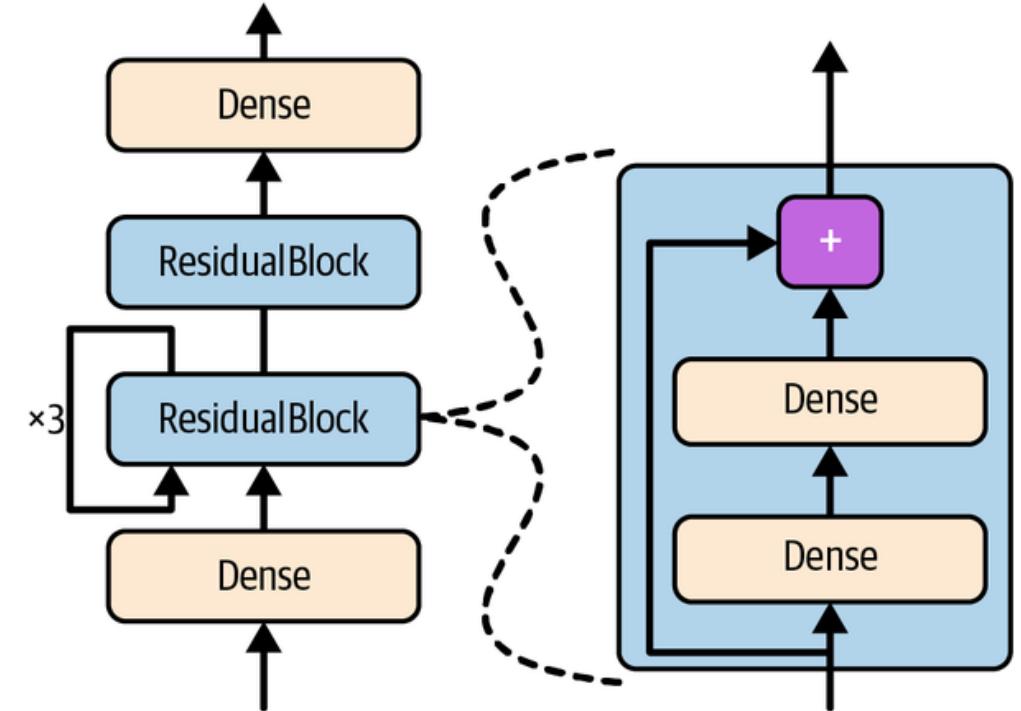
Consequently, in deep networks the gradient disappears, so the first layers are unable to learn. For this reason, activation functions can be changed or "skip-connections" can be exploited, as in the wide and deep network, to bring the gradient directly to the deeper layers of the network.

## Residual block

The **residual block** is a module of the network that we can exploit in cases of vanishing gradient .

It is based on the concept of **skip-connection** , that is, directly connecting the input with the output of the block, keeping the deeper features in the middle. The skip connection takes the residue deeper into the network, which is why they are also called **ResNet** .

In many cases it is used to concatenate multiple residuals blocks in a row.



# Custom Layers

In Tensorflow we can define a custom layer such as the residual block , extending the Layer class. In this way we can avoid writing all the layers by hand , for example by applying N dense layers automatically.

We just need to initialize our ResidualBlock like any of the other layers .

```
class ResidualBlock(tf.keras.layers.Layer):
    def __init__(self, n_layers, n_neurons, **kwargs):
        super().__init__(**kwargs)
        self.hidden = [tf.keras.layers.Dense(n_neurons, activation="relu",
                                            kernel_initializer="he_normal")
                      for _ in range(n_layers)]

    def call(self, inputs):
        Z = inputs
        for layer in self.hidden:
            Z = layer(Z)
        return inputs + Z
```

## Recap

We have seen how an MLP is structured

- Sequential and non-sequential
- With multiple inputs and outputs
- With different output layers
- With Residual Blocks

## Training a network

We have seen how the training involves calling the `.fit` our model and our optimizer will take care of applying the gradient to our parameters to improve them.

If we start to define custom structures and custom loss functions , we may also need to customize the training of our network.

To do this, Tensorflow provides us with Autodiff to automatically calculate the gradients to apply depending on how our variables are used.

## Gradient Tape

Inside the «with tf.GradientTape () ..» the «adhesive tape» pastes all the operations that concern the variables.

Finally calling . gradient () all partial derivatives are calculated.

```
w1, w2 = tf.Variable(5.), tf.Variable(3.)
```

```
with tf.GradientTape() as tape:
```

```
    z = f(w1, w2)
```

```
gradients = tape.gradient(z, [w1, w2])
```

# Custom training loops

Now that we know how to calculate gradients we can build a complete custom training loop.

After calculating the gradients we pass them to our optimizer which will apply them, with its logic, to our weights.

All this is normally abstracted from the `.fit` of the model but we will see that for more complex algorithms we will need to write it by hand.

```
for epoch in range(1, n_epochs + 1):
    print("Epoch {}/{}".format(epoch, n_epochs))
    for step in range(1, n_steps + 1):
        X_batch, y_batch = random_batch(X_train_scaled, y_train)
        with tf.GradientTape() as tape:
            y_pred = model(X_batch, training=True)
            main_loss = tf.reduce_mean(loss_fn(y_batch, y_pred))
            loss = tf.add_n([main_loss] + model.losses)

        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
        mean_loss(loss)
        for metric in metrics:
            metric(y_batch, y_pred)

        print_status_bar(step, n_steps, mean_loss, metrics)
```

## Pre -training

Once the structure of our model has been defined we can generally carry out an initial train and then a fine tuning.

The first train has the objective of verifying the model's ability to converge. In this phase, the number of layers , the activation functions, the structure of the model and the choice of loss are evaluated .

Vaulted pre-training is also performed on a larger and potentially dirty dataset with a higher learning rate.

In the pre-train we can also use optimization methods such as **GridSearch** to tune some hyperparameters such as the number of layers . However, doing **Hyperparameter tuning** in the case of neural networks is extremely more onerous because we no longer have *weak-learners* but actual complex objects.

## Fine tuning

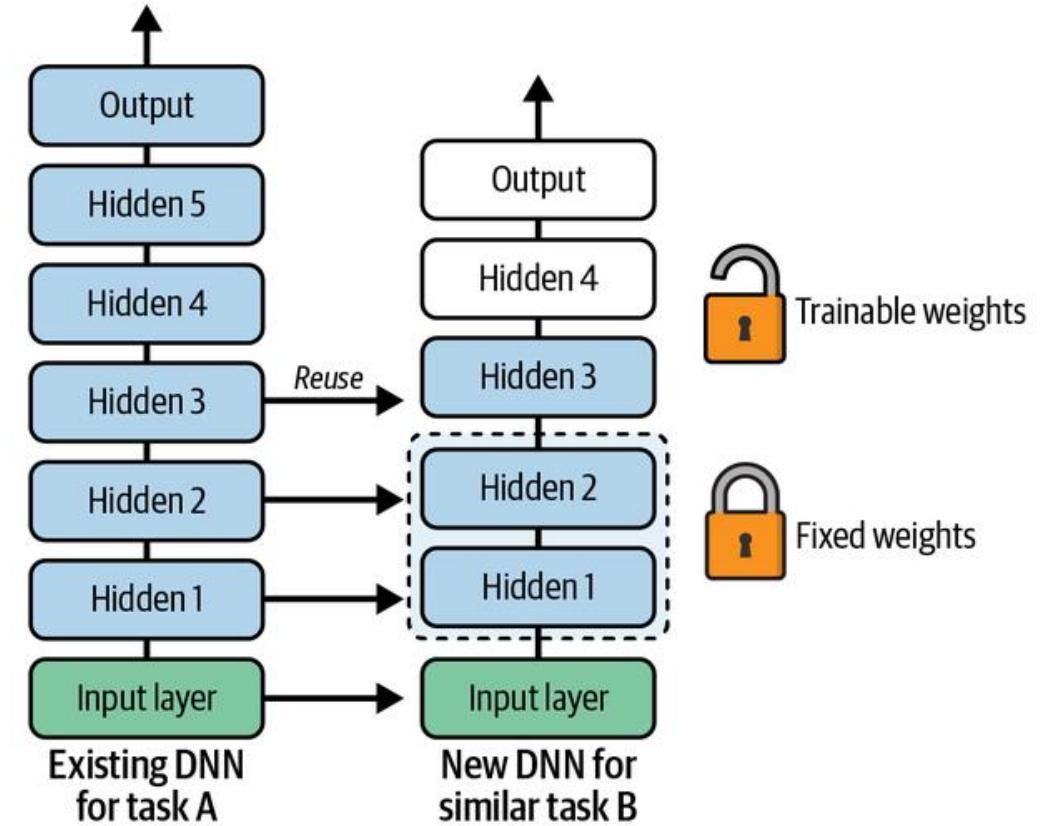
Fine tuning instead consists of perfecting the model so that it works best on our task.

To do this we can

- Work on a higher quality data set
- Go and change the learning rate
- Add or edit regularization parameters
- Change batch size
- Fix the value of some layers
- ...

## Fixed weight

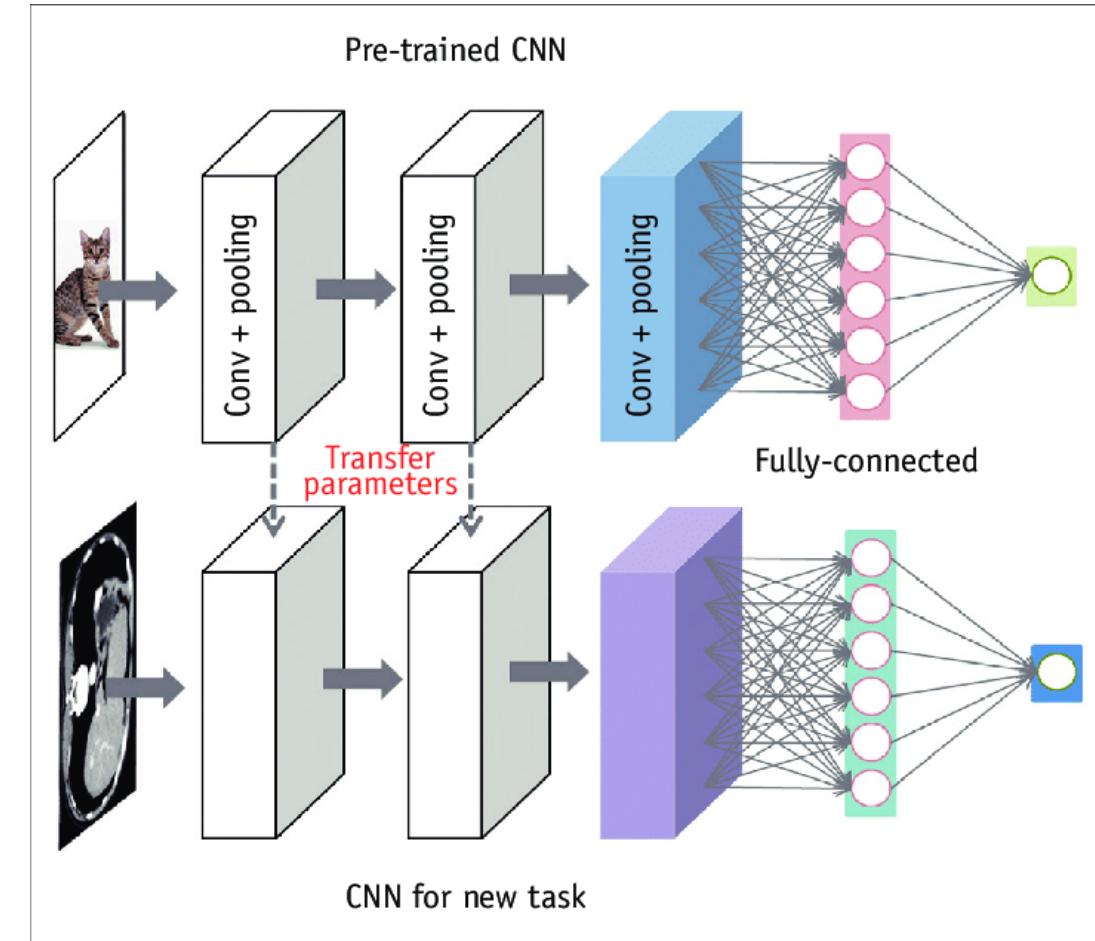
If we have already trained a model on a similar task, or we have done a satisfactory pre-train and want to keep the "low level" features that this extracts unchanged, we can set the weights of some layers .



## Transfer learning – supervised pre-train

pre -trained network, or piece of network, to solve a slightly different task is called Transfer Learning.

Thanks to transfer learning we can pre -train a network for example on the COCO dataset to classify images between 80 classes and re -exploit that knowledge, the features it extracts for its classification, to train our classifier.



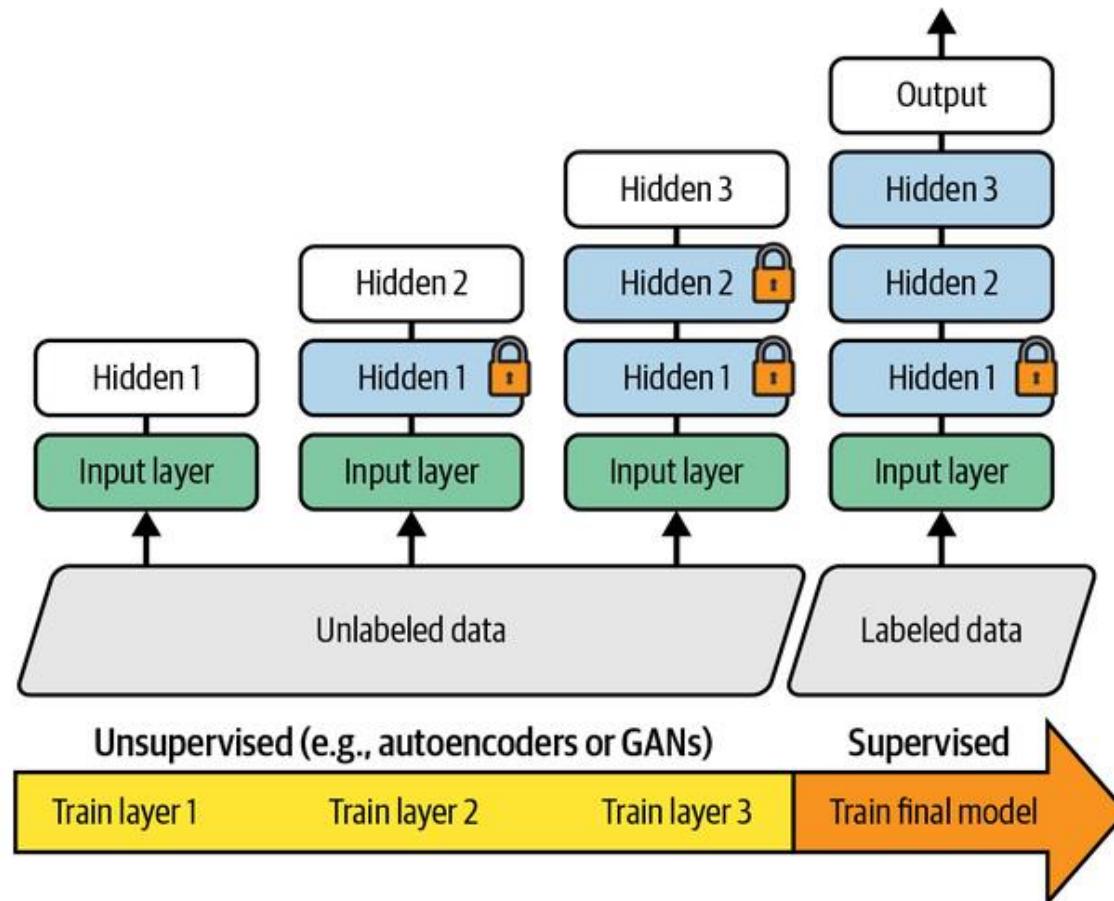
## Unsupervised pre-train

In the case of unsupervised pre-train we will use unsupervised learning techniques to train our layers to extract significant features on the same dataset as ours but with a different task, for example

- Compress information and decompress it ( Autoencoder )
- Generate credible data (Generative Adversarial Networks)
- Reconstruct parts of data (self- supervised )
- Removing noise from data (de- noising )

These different tasks lay the foundations for extracting significant features in the first layers of the network.

## Unsupervised pre-train



## Overfitting

In the case of neural networks we often have a very large number of parameters, so the classic ratio of 10-20 examples per parameter is often unattainable.

Pre -training and tuning the network by blocking some layers , or with a much lower learning rate, are just two of the different techniques we can apply to reduce overfitting .

As we have already seen we can add regularization (which will enter the model loss ) to the layers to keep the weights with low absolute values.

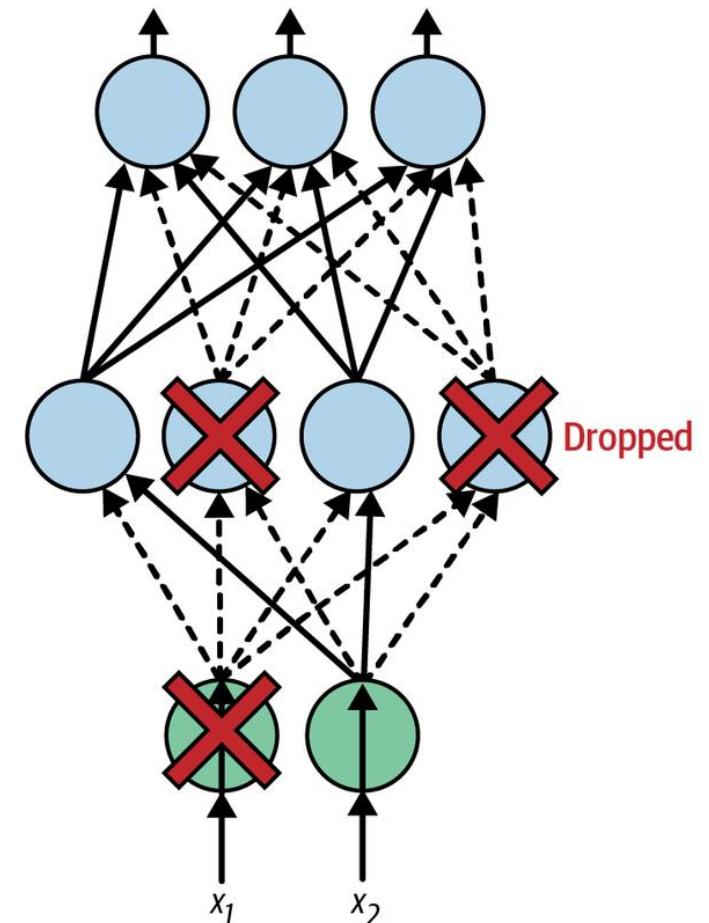
We can also add a layer specifically to reduce the risk of overfitting .

## Dropouts

The dropout layer is a layer **that** "deletes" some random neurons during the training phase, setting their output to zero.

In this way the model is forced to distribute knowledge among multiple neurons, not having the possibility of always exploiting the same features to make the decision. For each layer we will specify a percentage *drop-rate* of number of neurons at zero.

*"Would a company perform better if its employees were told to toss a coin every morning to decide whether or not to go to work? The company would be forced to adapt its organization; it could not rely on any single person to work the coffee machine or perform any other critical tasks, so this expertise would have to be spread across several people."*

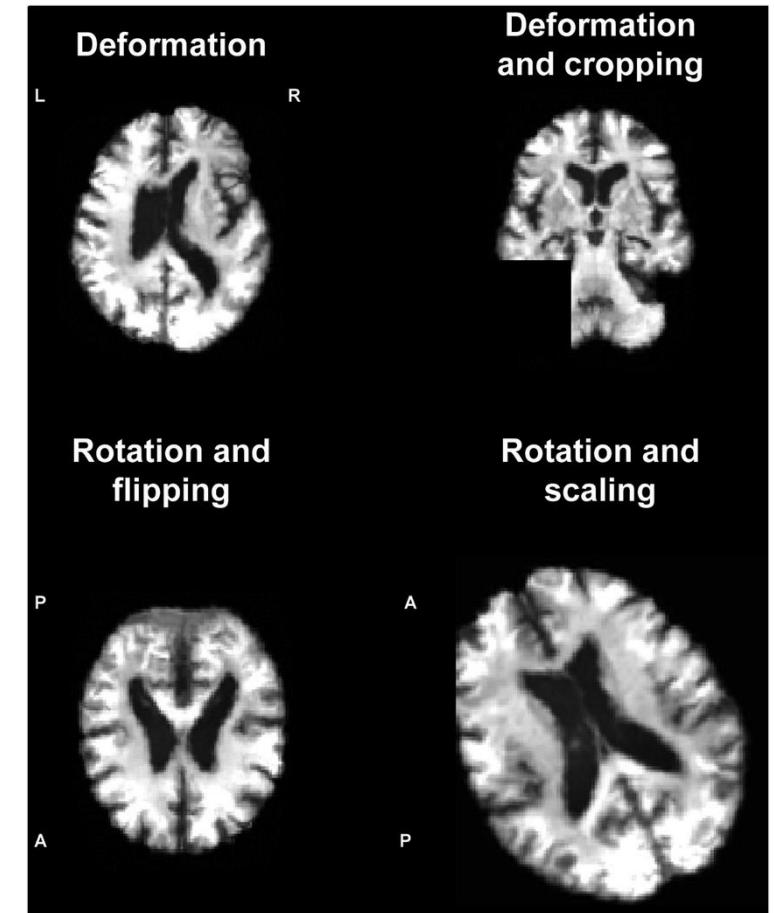


# Data augmentation

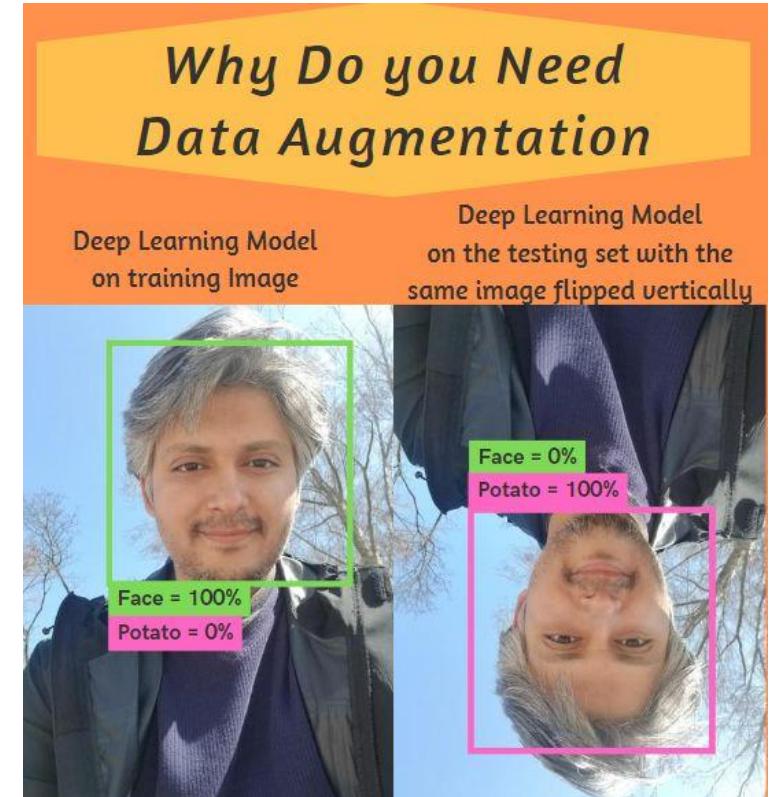
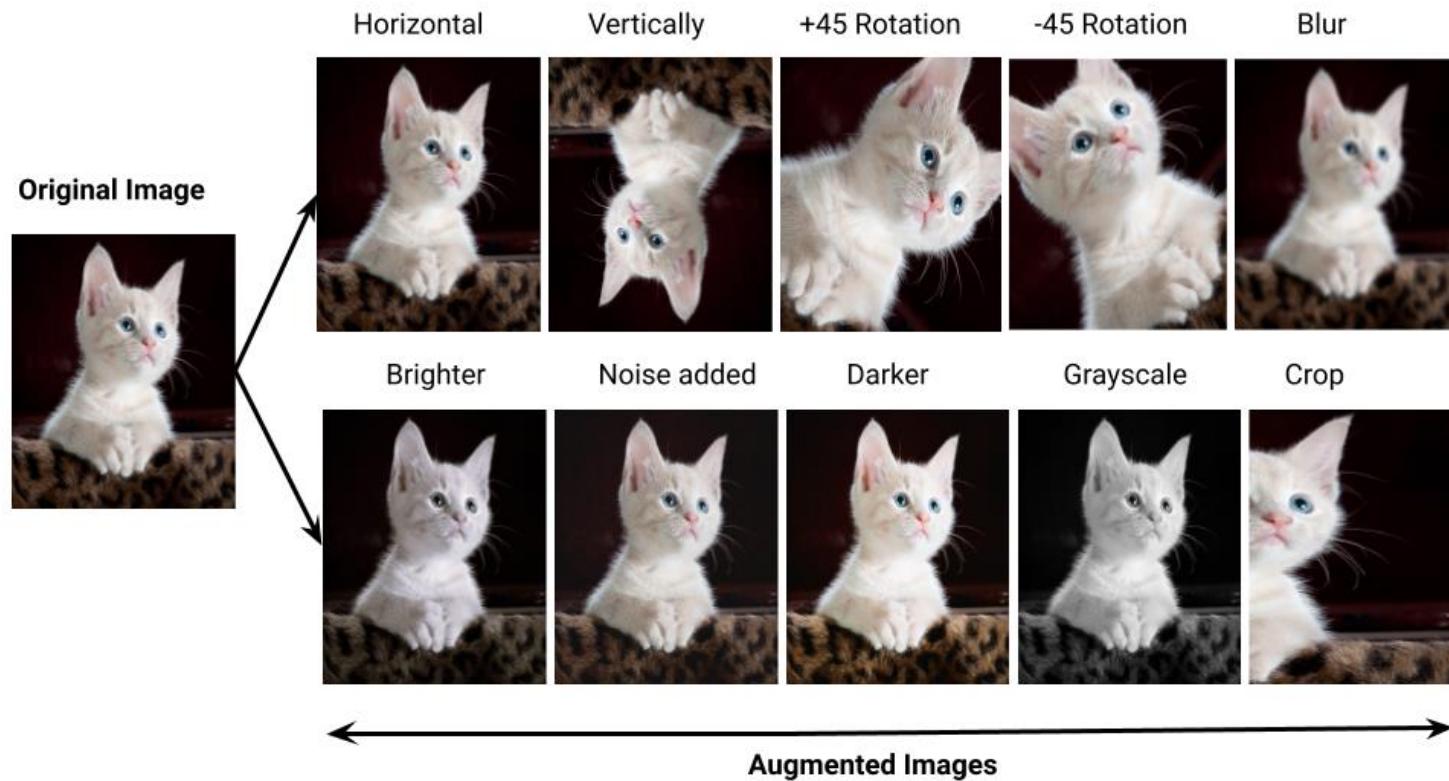
Data augmentation is the process by which we create new "synthetic" examples starting from real ones, distorting them appropriately so that the information contained is still detectable.

Noises are often added which have a lower variance than the difference between the examples. In the case of images we can apply many other transformations (perspectives, rotations, noise, blur, exposure, etc.)

This way we can multiply our input data making the model more robust to new data.



# Data augmentation

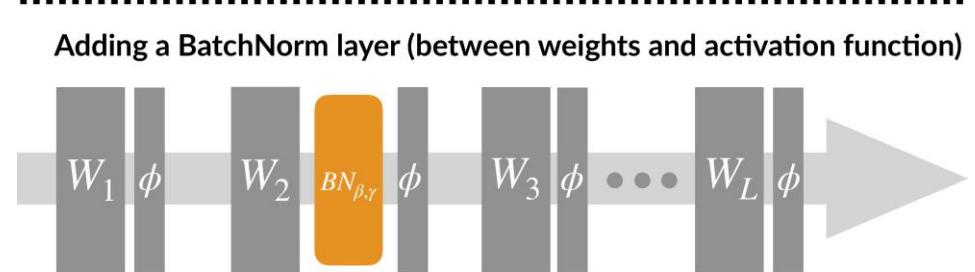
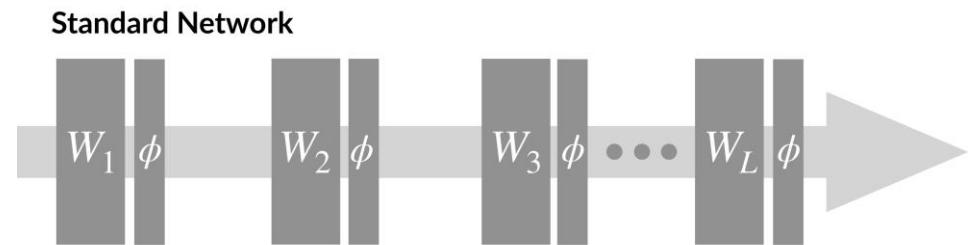


## Batch normalization

Another layer useful for learning, even if it does not prevent overfitting , is Batch Normalization .

Where inserted, it takes all the features predicted by the neurons of the previous layer and normalizes the outputs with respect to the single training batch, rescaling the values so that they are distributed like a normal.

It has been shown over time that this greatly favors the learning time and the ability of the model to converge.



## Function layer

There are other layers that have no impact on training or model performance but are useful for building more complex models.

Some of these layers are

- **Flatten** , i.e. a layer that takes an n-dimensional input array and reduces it to a vector
- **Concatenate** , which takes two or more n-dimensional arrays and concatenates them along an axis
- **Maximum, Minimum, Average** , i.e. layer that extracts a single value from all the features
- **Reshape** , to change the structure of an n-dimensional array

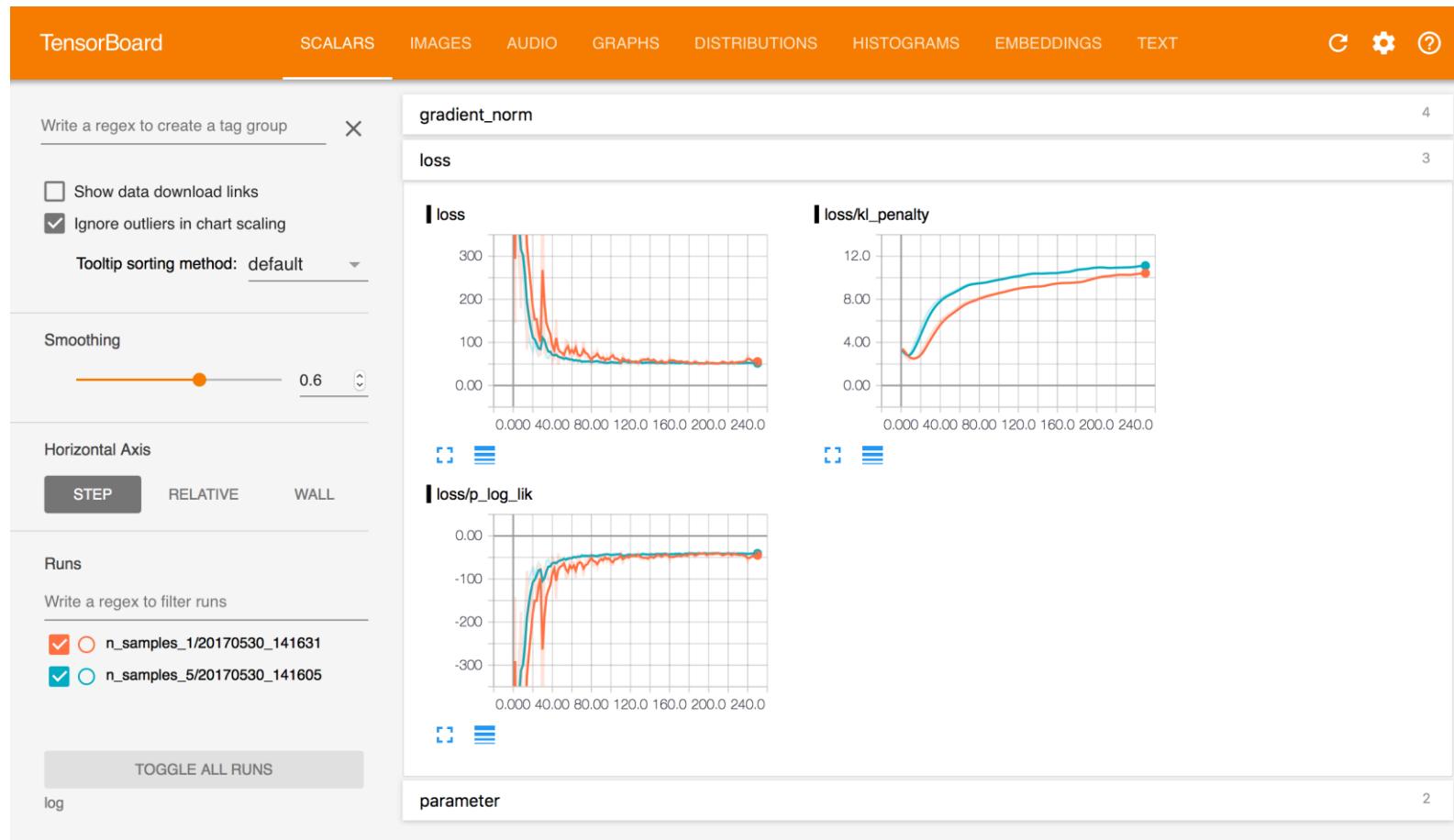
## Callbacks

Finally, once the training of a model has been launched we can execute, at the end of each epoch, callbacks , i.e. functions that can help us.

Between these

- **EarlyStopping** , a callback that stops training when it observes that the loss between training and validation deviates too much and therefore may have started to overfit the model
- **ModelCheckpointer** , a callback that saves the model, for example every time we make a better score in validation
- **Tensorboard** , a convenient callback to log all the training data and model characteristics to be able to observe them in the tensorboard

# Tensorboard



## Recap

We have seen

- The custom training cycle
- Some function layers that we can use
- How to prevent overfitting
- The callbacks

Exe w/ Tensorboard



# **11 – Convolutional Neural Networks**

Daniele Gamba

2022/2023

## Previous

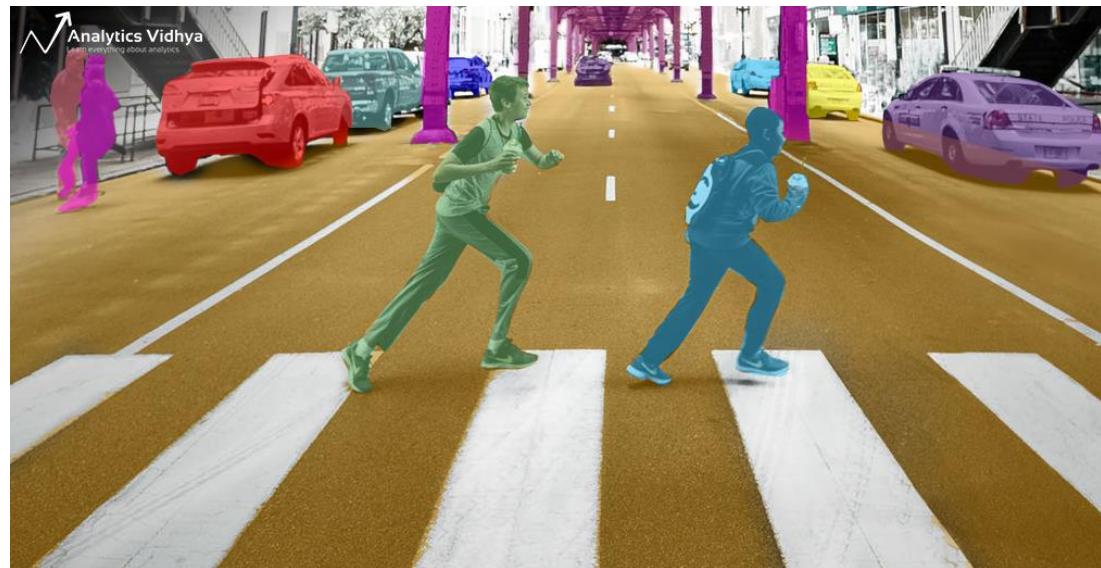
We have seen how an MLP is structured

- Sequential and non-sequential
- With multiple inputs and outputs
- With different output layers
- With Residual Blocks
- The custom training cycle
- Some function layers that we can use
- How to prevent overfitting
- The callbacks

## Other issues

In the problems we have seen so far we have always had a fairly traditional dataset, that is, an input table  $X$  (  $N_{\text{examples}}$  ,  $N_{\text{features}}$  ) and one or more outputs  $y$ .

But what if our data were images? Or signals in time without a precise beginning and end?



# MNIST

The “Modified National Institute of Standards and Technology database,” or MNIST, is a dataset of 70,000 handwritten numbers .

The goal was to automatically recognize characters on self-sorting letters and checks.

Yann LeCun and others successfully achieved performance above 99% in 1999.



## Exe

Let's try to tackle the MNIST problem with the machine learning techniques we know

[https://knowyourdata-tfds.withgoogle.com/#dataset=mnist&tab=STATS&select=default\\_segment.mnist.label.value](https://knowyourdata-tfds.withgoogle.com/#dataset=mnist&tab=STATS&select=default_segment.mnist.label.value)

<https://colab.research.google.com/drive/1ynkAvgUh2KfHrFBGTvEUifrwkEfMo6Wu#scrollTo=q6oosyU-8e6S>

## Local information

As we have seen, it is very difficult to build an algorithm that takes images as input compared to the techniques we know.

The features of an image, the value of each pixel, have a meaning only if contextualized with respect to its neighboring pixels.

The information is therefore local and limited, only by looking at the image as a whole can we understand its meaning.

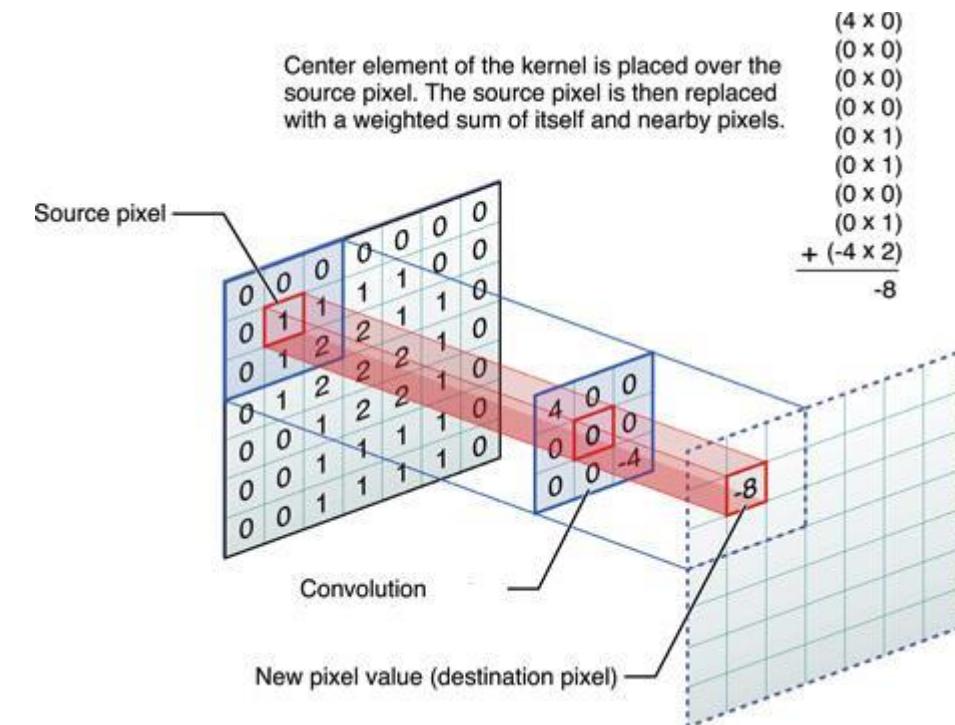
# Convolution

Convolution is the mathematical operation of sliding two vectors relative to each other.

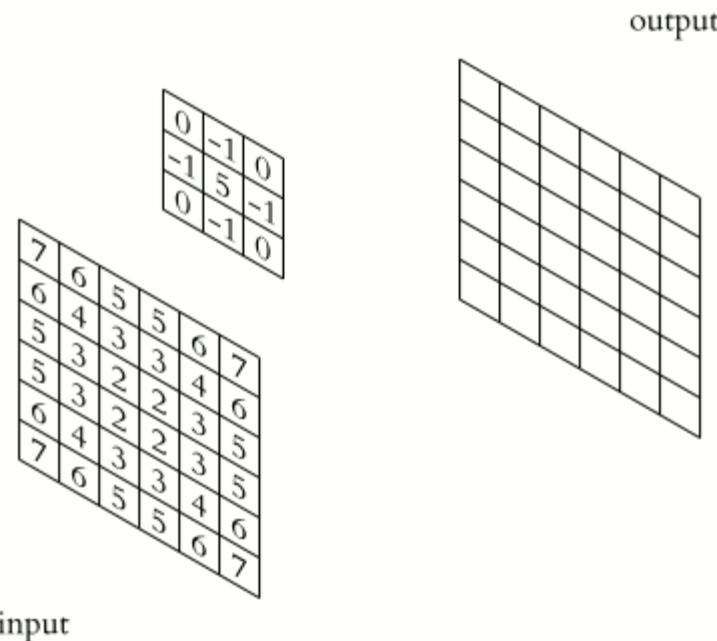
In the case of images, the idea is to slide a smaller matrix, called **a kernel**, across our entire image.

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

To **convolve a kernel** with an **input signal**:  
flip the signal, move to the desired time,  
and accumulate every interaction with the kernel



# Convolution



$$\begin{matrix} 7 & 2 & 3 & 3 & 8 \\ 4 & 5 & 3 & 8 & 4 \\ 3 & 3 & 2 & 8 & 4 \\ 2 & 8 & 7 & 2 & 7 \\ 5 & 4 & 4 & 5 & 4 \end{matrix} * \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} = \begin{matrix} 6 & & \\ & & \\ & & \end{matrix}$$

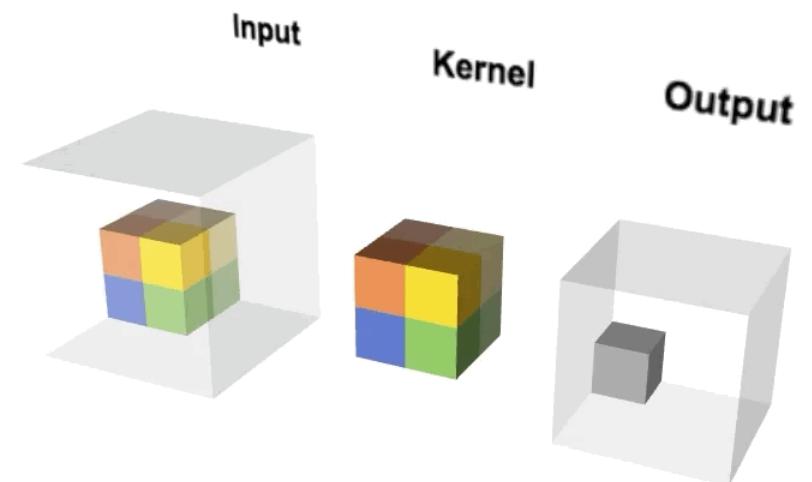
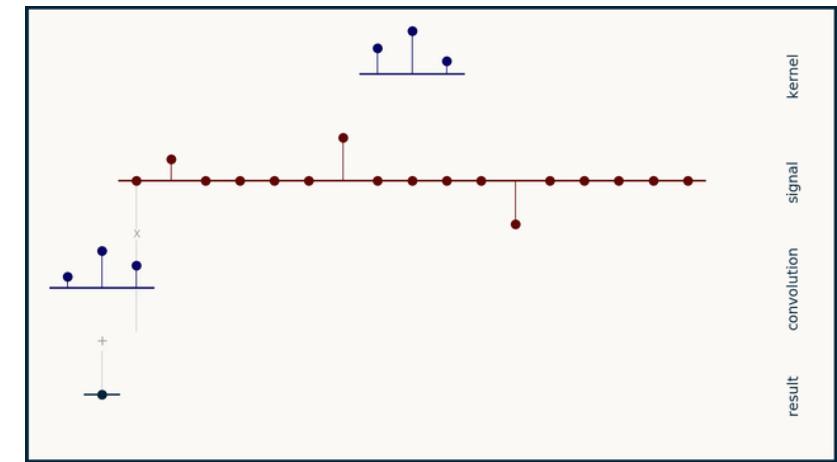
$7 \times 1 + 4 \times 1 + 3 \times 1 + 2 \times 0 + 5 \times 0 + 3 \times 0 + 3 \times -1 + 3 \times -1 + 2 \times -1 = 6$

# Convolution

The convolution can be 1D, 2D or 3D.

The values that the kernel that scans the input takes are the parameters of our function.

The hyperparameters are different, the first is certainly the size of the kernel **matrix**.

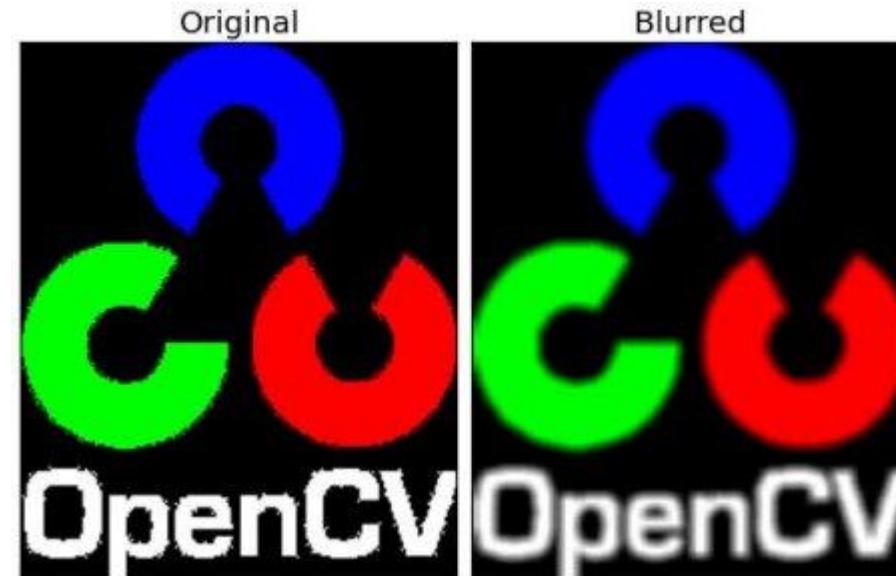


## Traditional convolution

Convolution kernels are often also called *filters* because in traditional computer vision convolutions have existed and been applied long before deep learning to process images.

Depending on the values and dimensions that are assigned to the kernel we can obtain different results.

$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

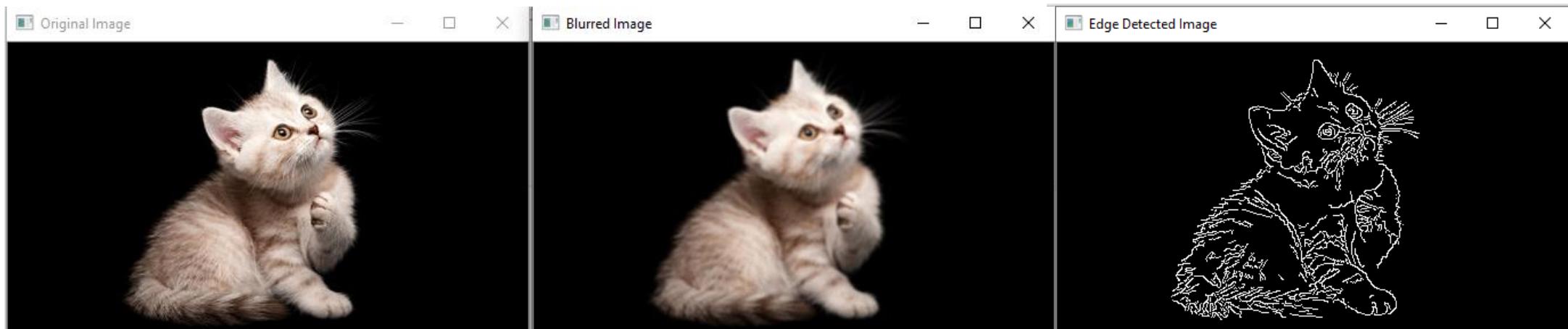


## Traditional convolution

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

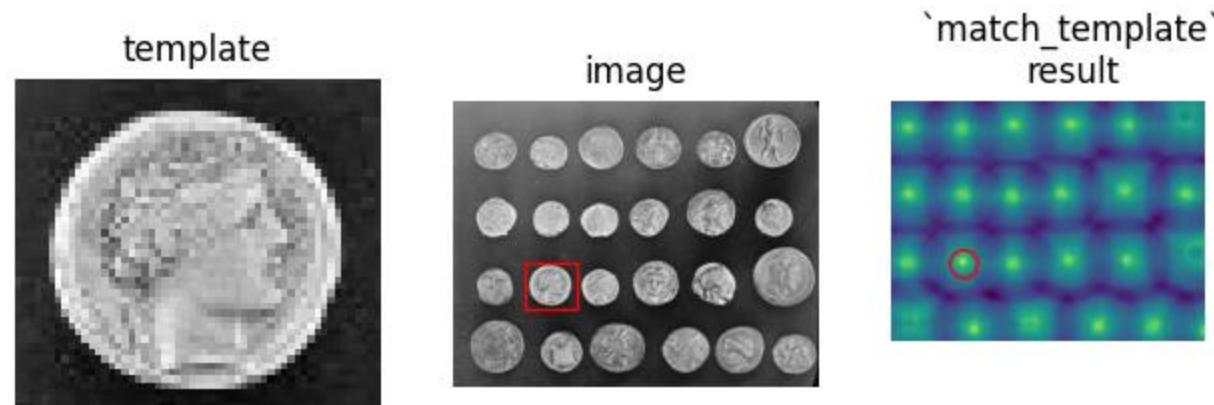
$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



## Template matching

Template matching is a traditional computer vision problem in which you need to find a template within a larger image.

It's a notoriously difficult problem, and the traditional approach involves convolving our template over the entire image. In each position a correlation index will be calculated and at the end the maximum obtained will be evaluated.



## Template matching

As with MNIST, traditional template matching suffers terribly from variations

- Scales
- Shape (and calligraphy)
- Prospect
- Rotation

For this reason, deep learning techniques that are able to better interpret local features and still reconstruct the presence of our object of interest have become increasingly popular.

## Conv2D

Going back to our convolution kernel in neural networks, we can define a Conv2D layer that defines

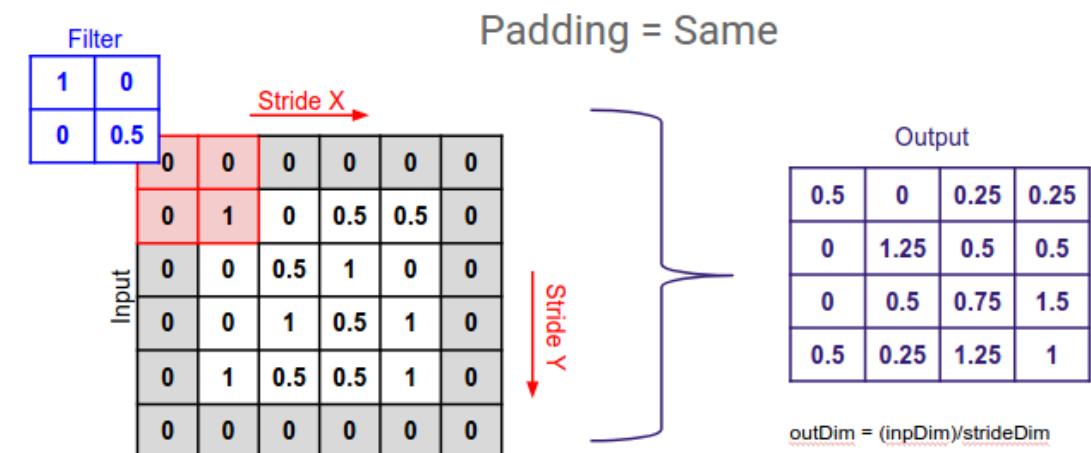
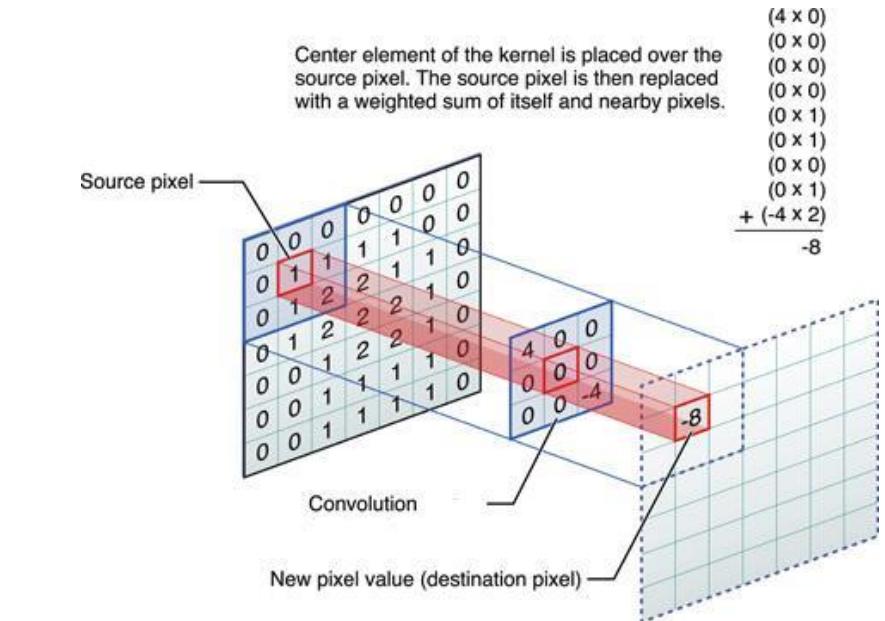
- The size of our kernels
- The number of kernels we want to apply
- The activation function to be applied possibly after the scalar product
- A set of other hyperparameters that we will see

# Padding

Since the convolution starts "inside" our image, the output matrix will have each dimension that is  $2 * (\text{kernel side} // 2)$  less.

This often results in our network layer sizes being very strange and difficult to split. For this reason it is used to add **padding** around the matrix , i.e. edges of zeros which allow the convolution to obtain more manageable dimensions.

Using padding = ' same ' we will get an output that has the same dimensions as our input matrix.



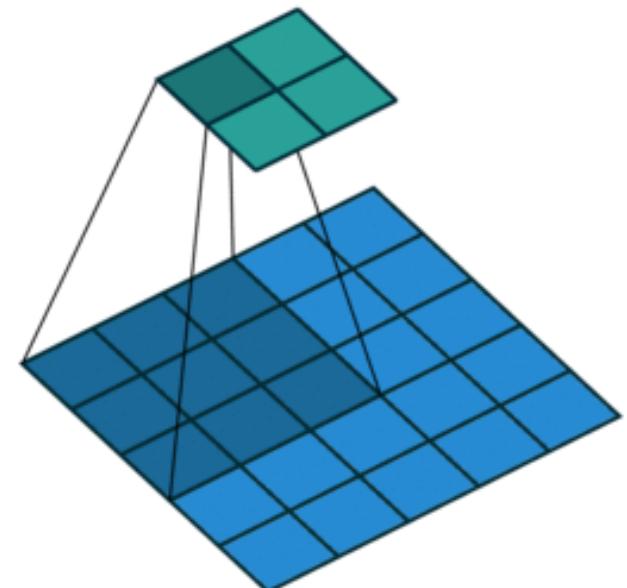
## Stride

The **stride** is the number of cells we move with each convolution.

If we wanted to perfectly scan the entire image, with stride=1 our kernel will move to all the pixels of the image.

With stride equal to 2 instead of moving just one square we will move 2, and so on.

By moving more than one box at a time our output will have smaller output dimensions than the input even in the presence of padding .

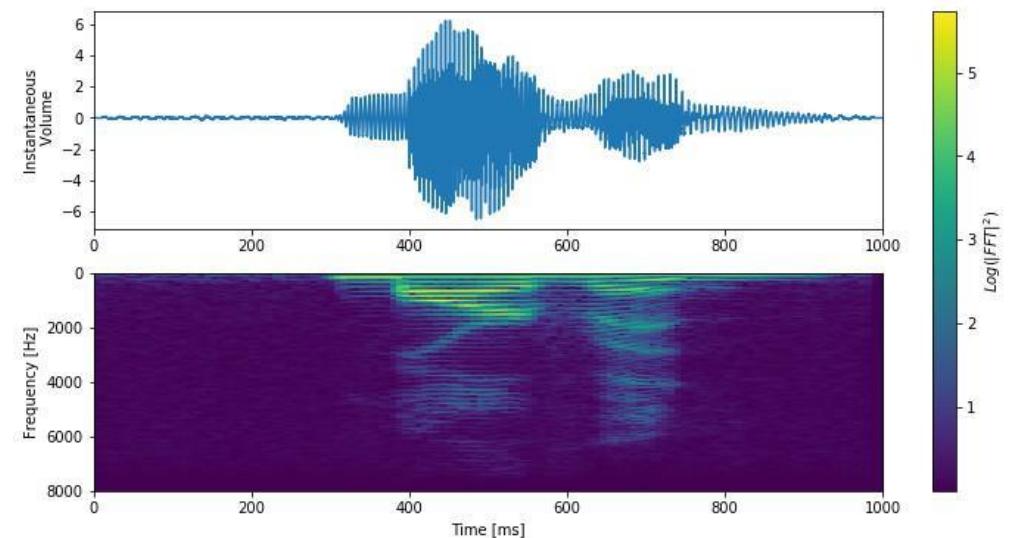
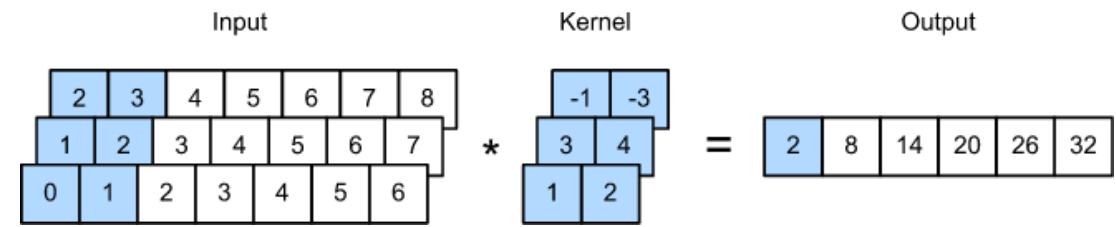


# Conv1D

1D convolution is commonly used for time series.

Convolution allows us to intercept typical local features of signals over time.

Furthermore, we can also apply a 1D convolution on spectrograms by applying kernels as high as our features and which moves in the direction of time.

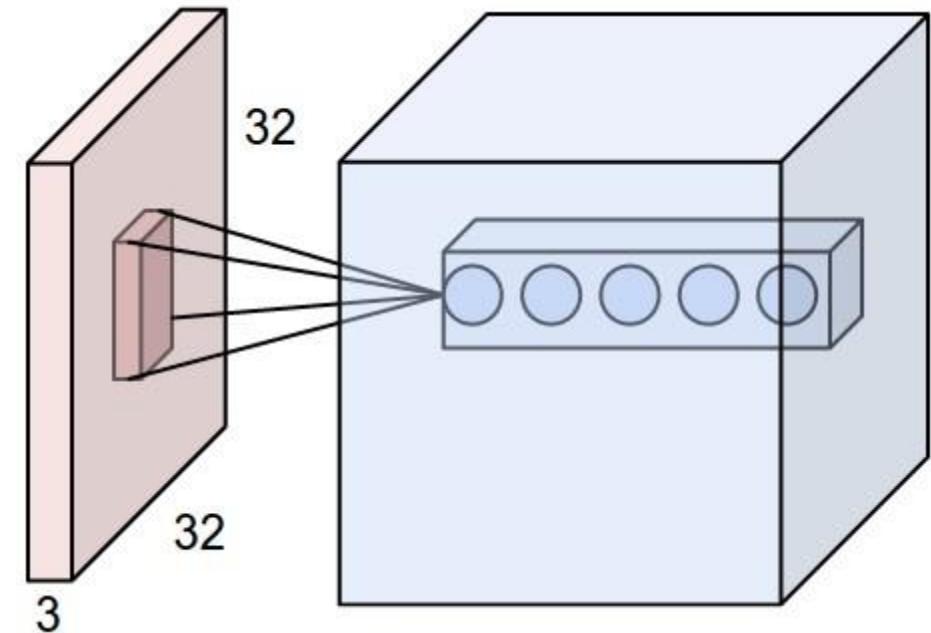


## Conv2D

Once we have defined our layer , hyperparameters and activation function, the number of kernels we have decided on will project N different computations into a hypercube .

**Our kernel will be only as deep as the input layer.**

They will all be representations with the same level of detail as our input matrix (possibly subsampled with the stride), but we need another tool to reduce the dimensionality and be able to better interpret the features.



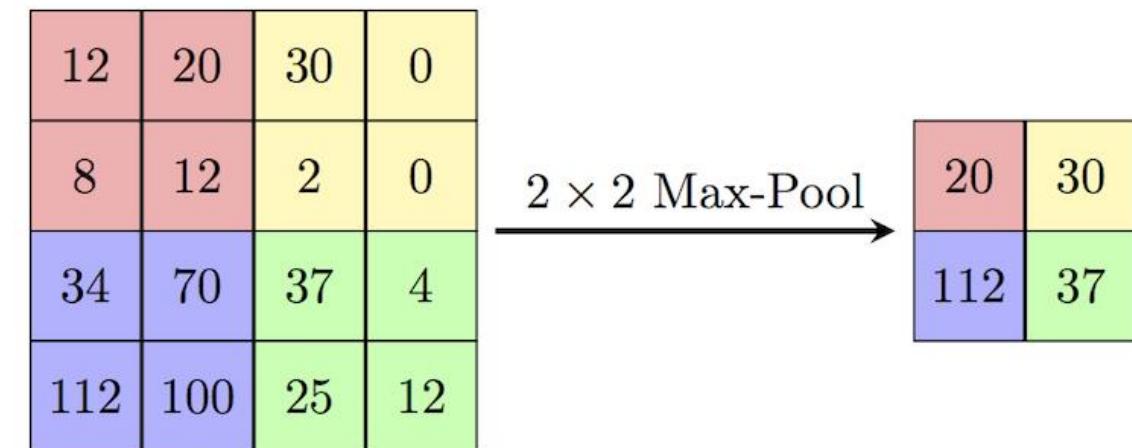
# Pooling

Once we have processed our features we need a way to reduce the dimensionality of our layer to be able to interpret our features at a higher level.

This is why there is a **pooling** layer , where we get the maximum, minimum, average, etc. every x cells to construct a matrix that has size divided by x.

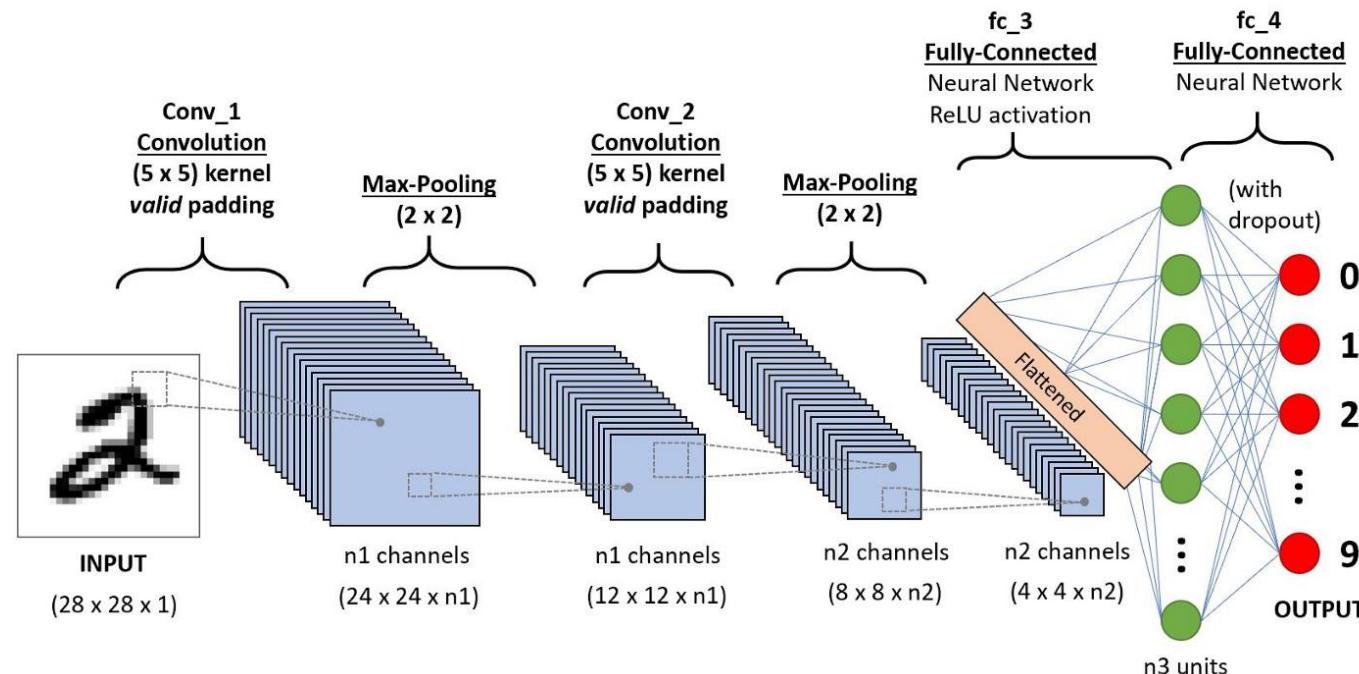
In this case with a 2x2 Max-Pool we take the maximum every 2x2 cells and build our output. From a 4x4 matrix we get a new 2x2 matrix.

Precisely for the applicability of pooling, we try to have matrices divisible by multiples of 2, possibly using padding in the convolution.



# CNN

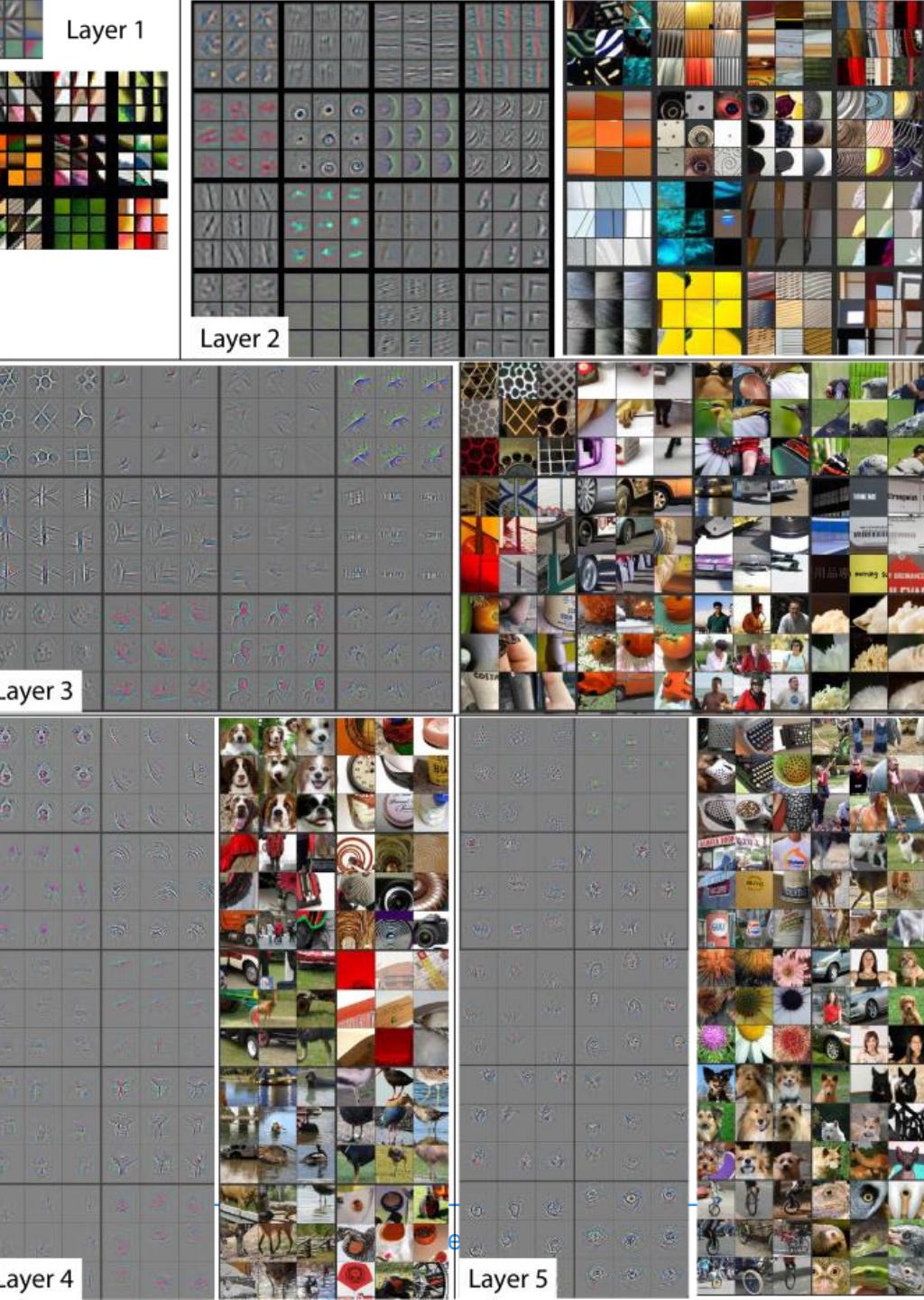
A Convolutional Neural Network is therefore a structure that uses convolution layers and possible pooling layers to perform various tasks. Normally we tend to **reduce the dimensionality on x and y** through pooling and **increase it on z** via the number of kernels. At the bottom of the network we will then flatten the result and apply dense layers to classify the result of the convolutional features .



## Features visualization

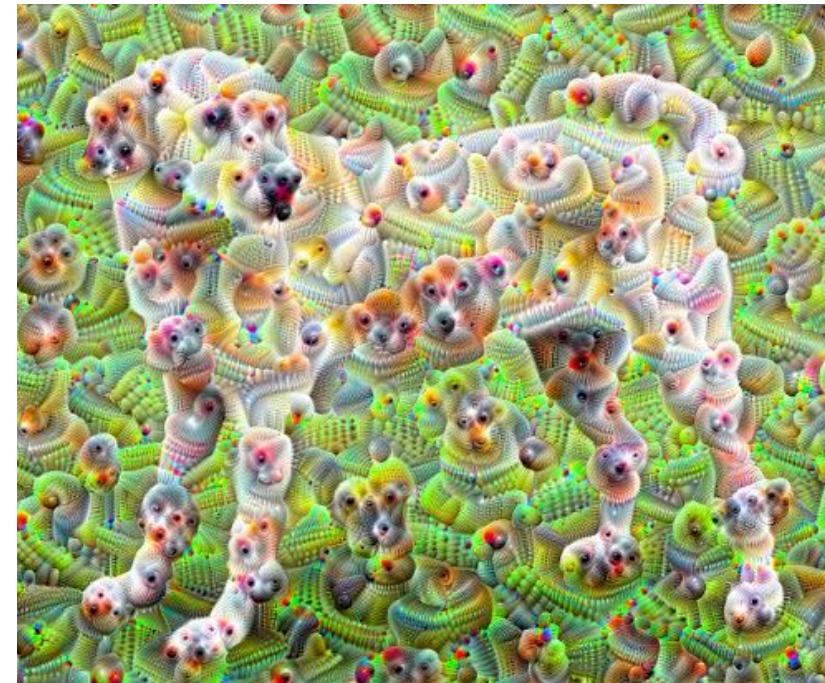
As we have already seen, intuitively in the first layers we have the simplest features to show lines, circles, ..., and with pooling and subsequent convolutions we interpret combinations of these features which gradually become more complex and numerous.

E.g. View intermediate features



## Deepdream

Deepdream is a paper in which we wanted to show which patterns are decisive for a network in deciding what class an object is and they do so by reprojecting the patterns into the image itself. In fact it is like observing what the neural network is dreaming.



# Deepdream



## Recap

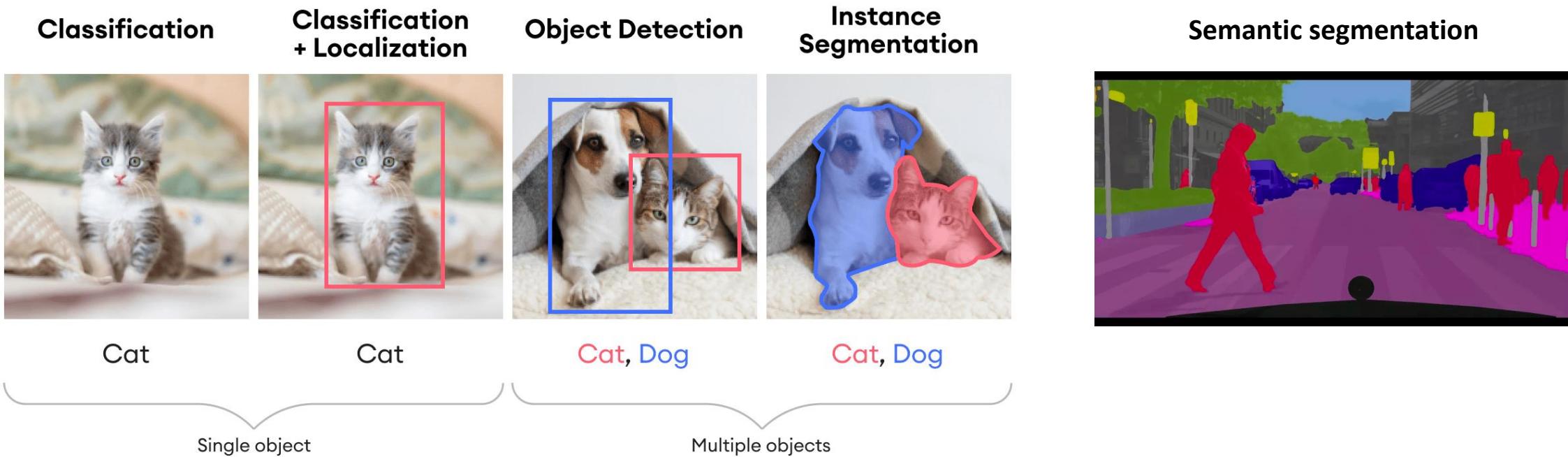
We have seen how to structure a CNN via

- Convolutions
- Padding
- Pooling
- A standard architecture for classification

Now we can try again to classify our MNIST using a CNN

# Tasks

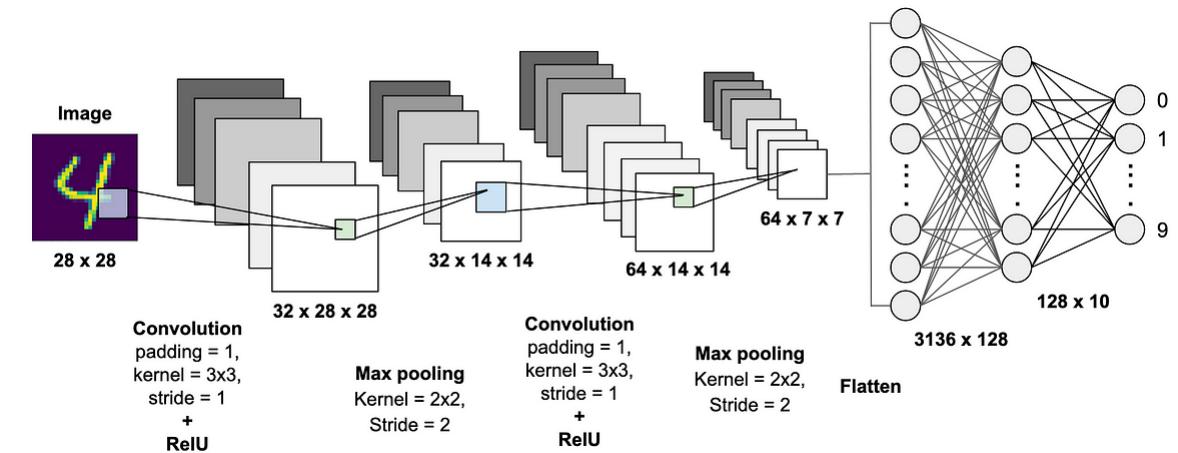
CNNs are particularly useful for working on 1D and 2D data to solve certain types of tasks



# Classification

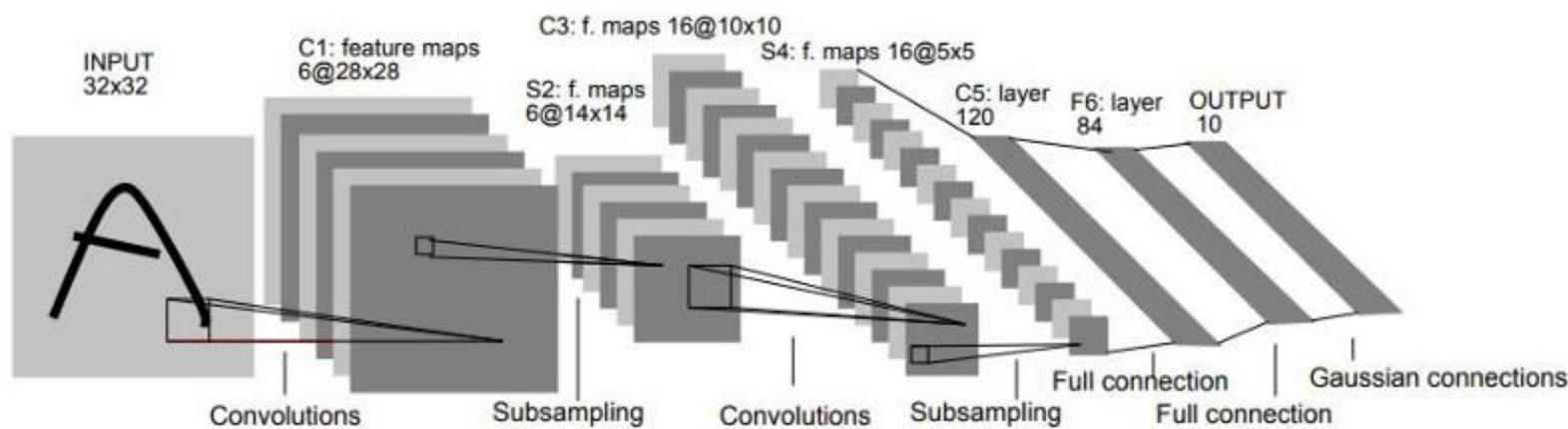
As we have seen, the first task is classification.

Given an image we need to tell which class it belongs to. In this case the head of the network will be an MLP with multiclass and possibly multilabel output .



## LeNet5

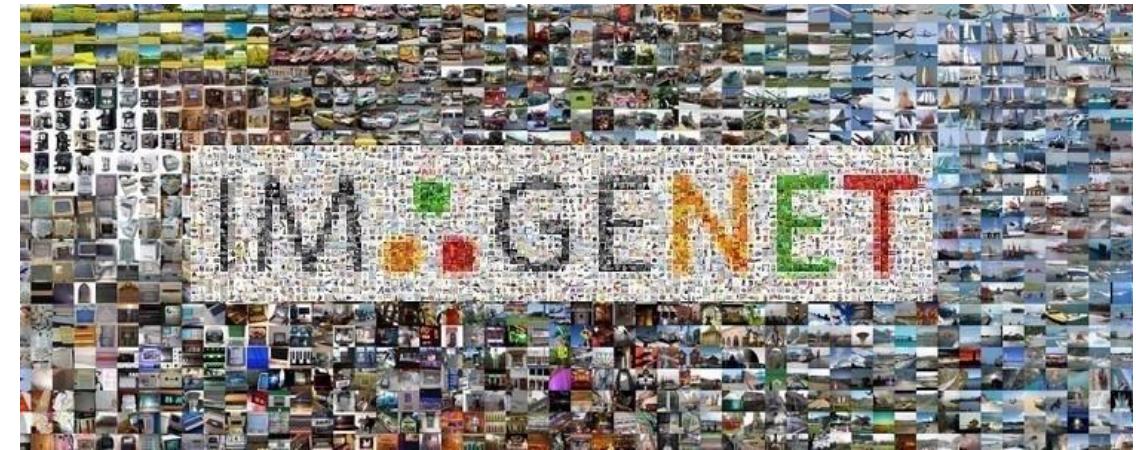
Yann LeCun's LeNet5, which he achieved 0.8% error on MNIST in 1998. It is a simple CNN that performed well on the character recognition task but failed to scale to more complex tasks.



# ImageNet

ImageNet is one of the most important and difficult datasets to tackle.

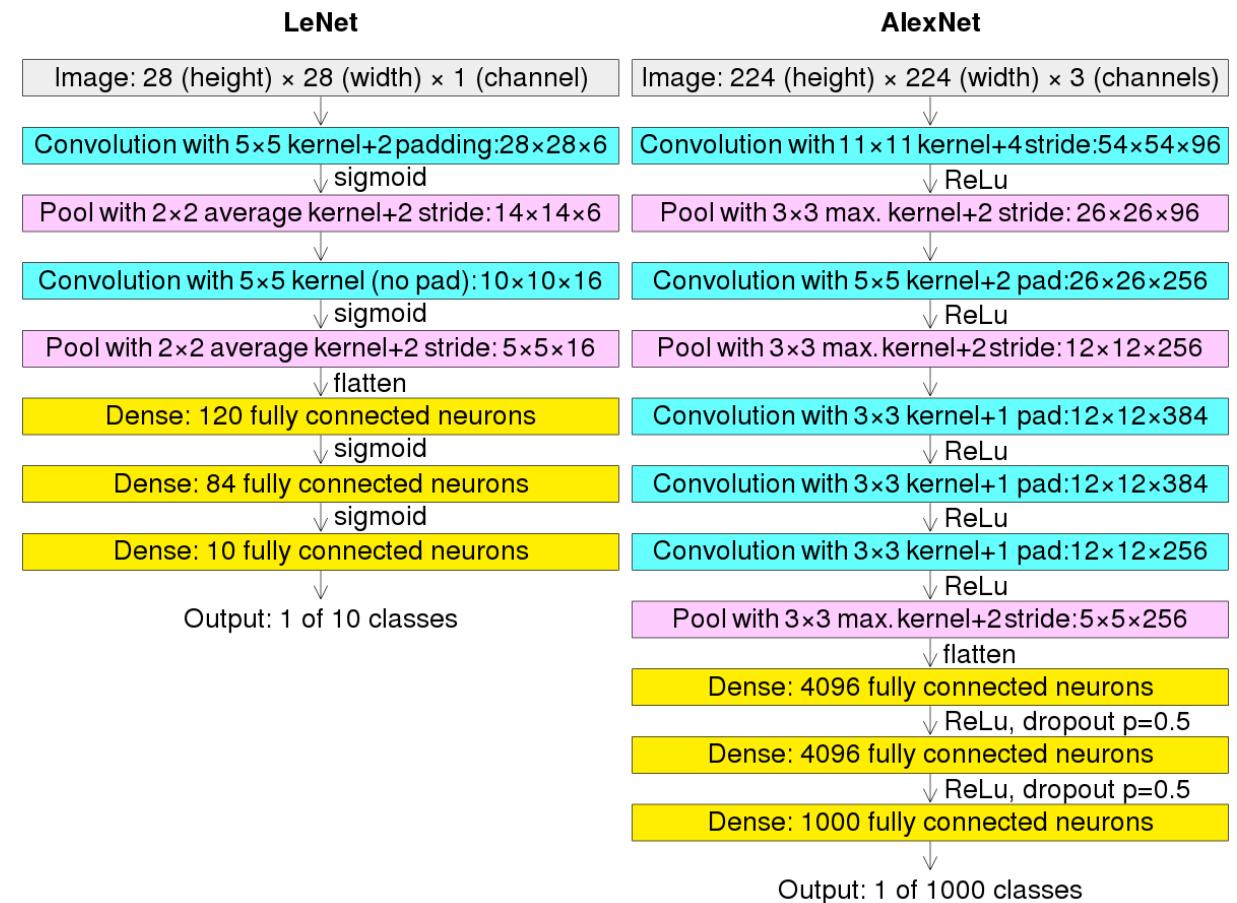
Until 2012 the competition consisted of 10,000,000 images from more than 10,000 categories.



# AlexNet

In 2012, AlexNet won the competition by a notable margin (17% Top-5 error versus 26% for the runner-up). The net was simply wider and deeper plus some other improvements.

The network had approximately 60 million parameters.

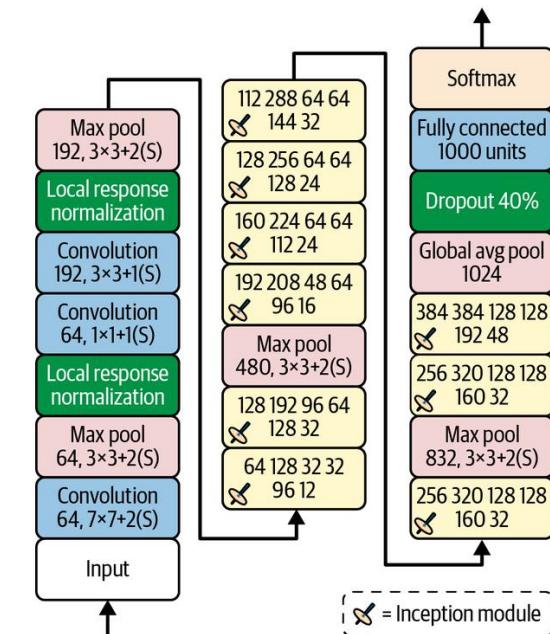
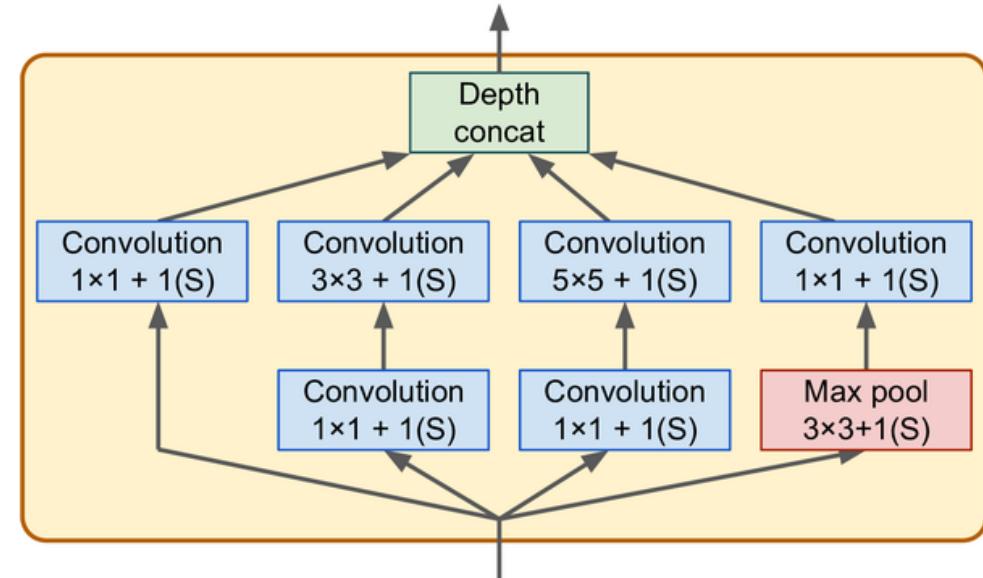


## Google LeNet

In 2014, Google won the same competition, bringing the Top-5 error to 7% with a network 10 times lighter than AlexNet (around 6 million parameters).

The performance is given by the fact that the network is much deeper and **implements Inception module**.

Thanks to this module you can learn much more efficient features.



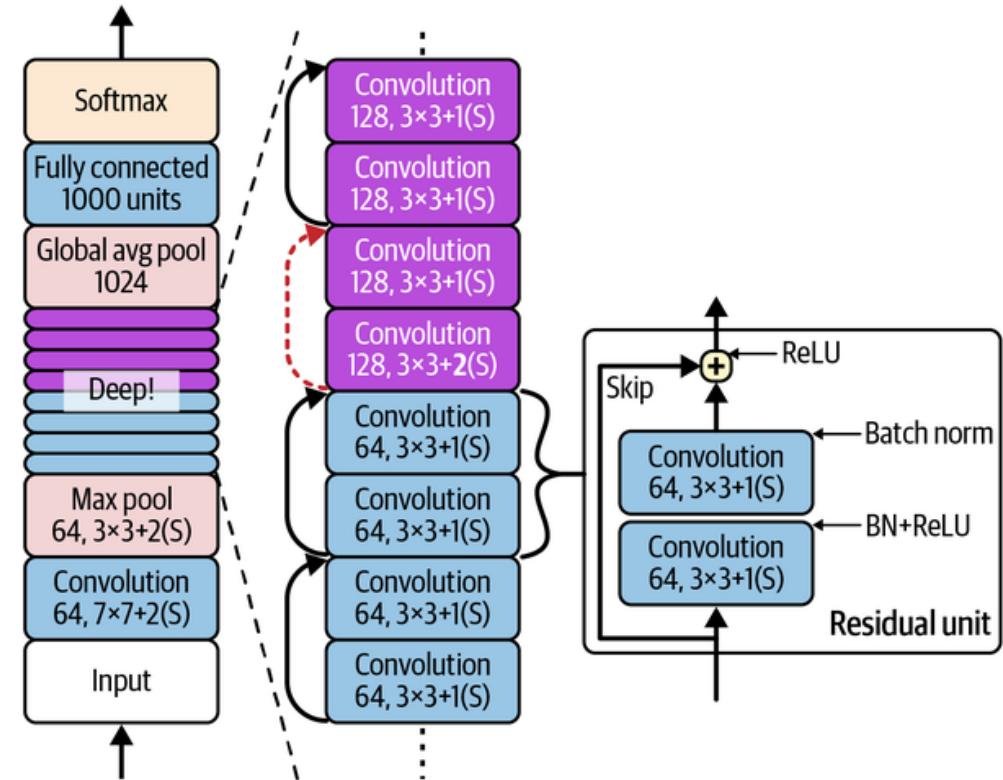
# ResNet

In 2015, Kaiming He et al. they won by proposing

**ResNet** with a Top-5 error of only 3.6%.

The ResNet had fewer parameters but **152 layers**, confirming the intuition that by diluting the information across multiple layers it was possible to better interpret what was being observed.

To train this deep network it was necessary to implement the **skip-connections** that we saw in the last lesson.



## Other models

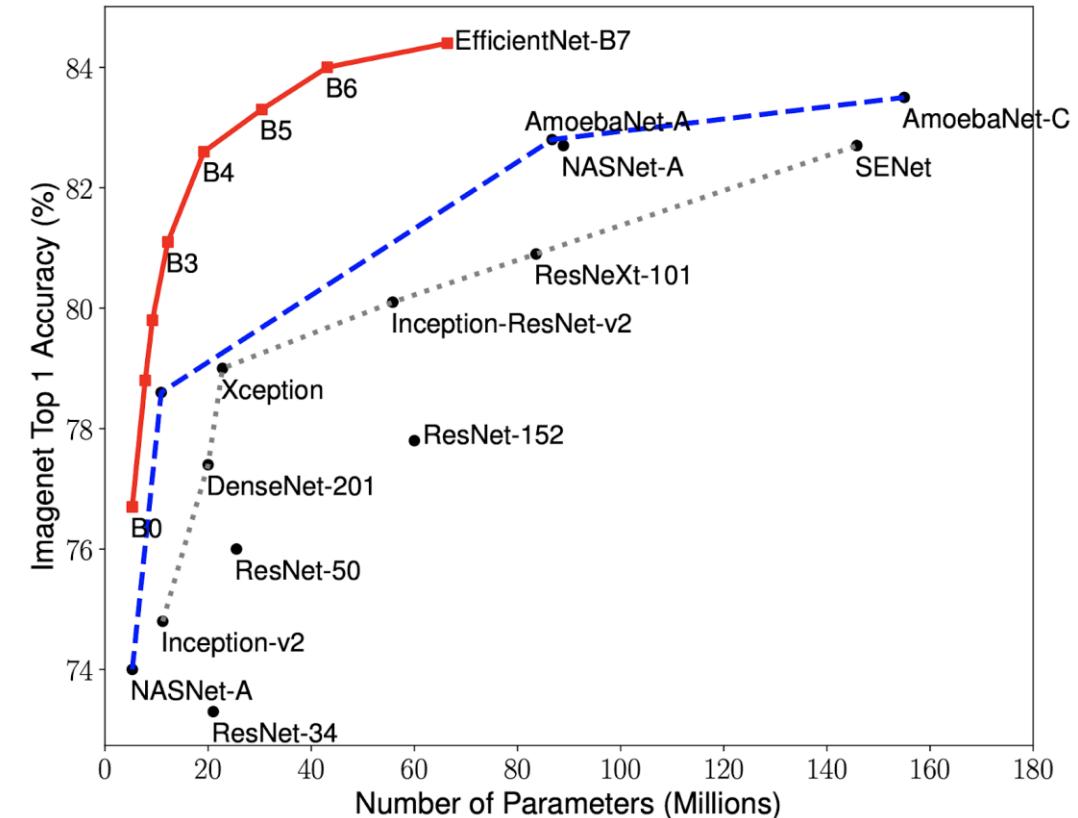
Historically there are other networks that have become extremely popular

### MobileNet

They are networks designed to run on smartphones, therefore small and fast, there are different types but they are widely used.

### EfficientNet

It is one of the most efficient networks, with very high performance on ImageNet gradually trained on increasingly higher resolution images.



## Pre-trained

pre -trained models on a task, take part of the weights and reuse them for our application.

Keras provides us with a series of pre -trained models for **transfer learning** .

These models already know how to interpret images very well and extract very significant features . All we need to do is cut the last layers and add our head to classify the objects of our interest.

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9
InceptionResNetV2	215	80.3%	95.3%	55.9M	449	130.2	10.0
MobileNet	16	70.4%	89.5%	4.3M	55	22.6	3.4
MobileNetV2	14	71.3%	90.1%	3.5M	105	25.9	3.8
DenseNet121	33	75.0%	92.3%	8.1M	242	77.1	5.4
DenseNet169	57	76.2%	93.2%	14.3M	338	96.4	6.3
DenseNet201	80	77.3%	93.6%	20.2M	402	127.2	6.7
NASNetMobile	23	74.4%	91.9%	5.3M	389	27.0	6.7
NASNetLarge	343	82.5%	96.0%	88.9M	533	344.5	20.0
EfficientNetB0	29	77.1%	93.3%	5.3M	132	46.0	4.9
EfficientNetB1	31	79.1%	94.4%	7.9M	186	60.2	5.6
EfficientNetB2	36	80.1%	94.9%	9.2M	186	80.8	6.5
EfficientNetB3	48	81.6%	95.7%	12.3M	210	140.0	8.8
EfficientNetB4	75	82.9%	96.4%	19.5M	258	308.3	15.1

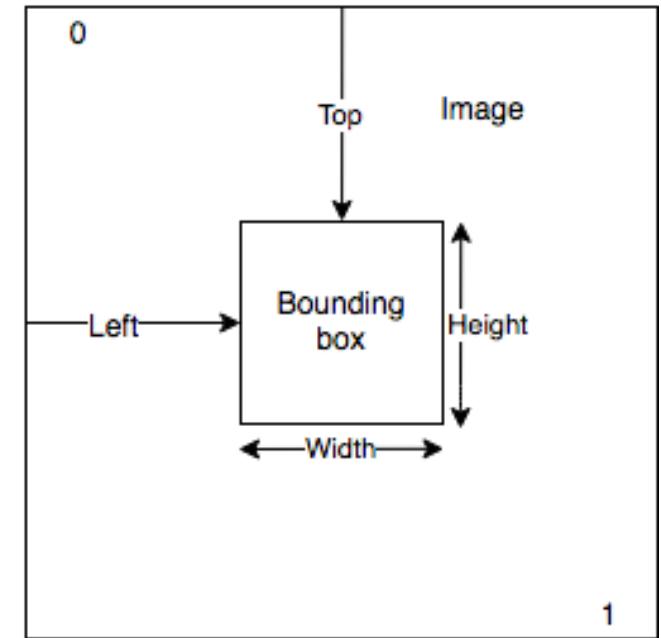
## Exe

Before proceeding, let's try to take a pre-trained CNN, cut off its head to see the output embeddings and perform classification on MNIST by classifying the embeddings or by doing transfer learning with our own head.

## Object localization

If we wanted not only to classify which object is in an image but also to have an indication of where it is, we can ask the network to extract a position and scale of the object. In this way our output will have, in addition to the probability that the object is there, also the coordinates  $x, y, w, h$  ( width , height ).

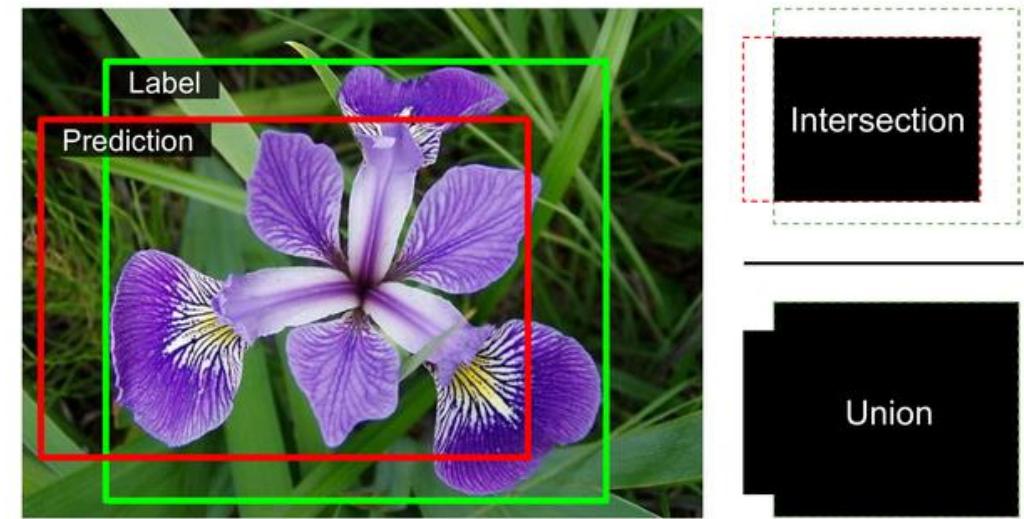
In these cases we will therefore have labels that will also include the object's **bounding box** , i.e. the box that encloses it.



# IoU

When we have bounding boxes as labels we can no longer use MSE or cross-entropy as losses , but we need something that indicates a degree of accuracy of our bounding box.

Intersection -over-Union was introduced which indicates the intersection area divided by the union area. Perfection occurs when the two BBs are perfectly overlapped.



## Object detection

When there are multiple objects in our image we want the bounding box and its confidence to be generated in output for each class. By no longer dealing with just one object but possibly N in the image, we need to find a way to encode a variable number of output objects.

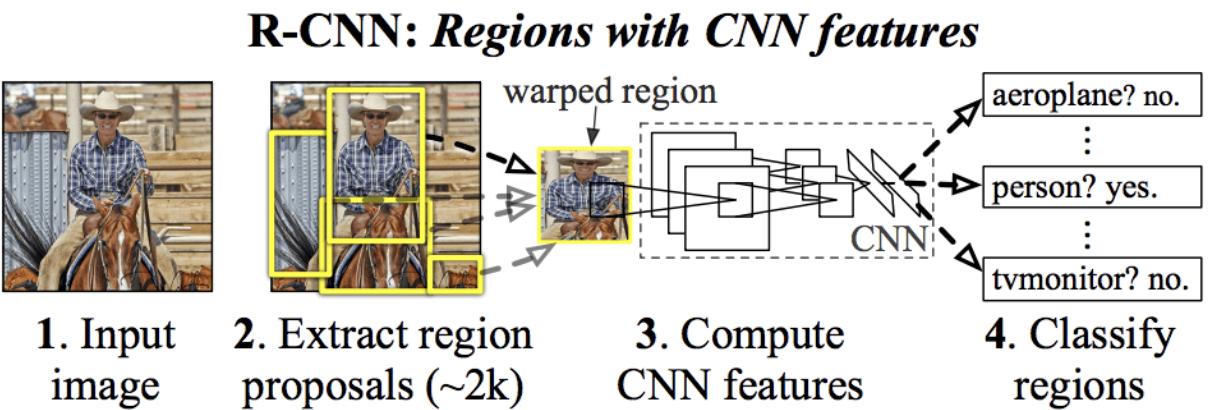
This problem has been solved in different ways by different network architectures.

# R-CNN

Region -based CNNs, or R-CNNs, are very simple 3-step networks

- A first very simple algorithm proposes possible cropping of the image, i.e. it makes re **gion-proposal** (about 2,000)
- A CNN extracts deep features, in the first implementation AlexNet was used
- A classifier, such as SVM, classifies the vector with deep features to say whether the object is present in the region or not.

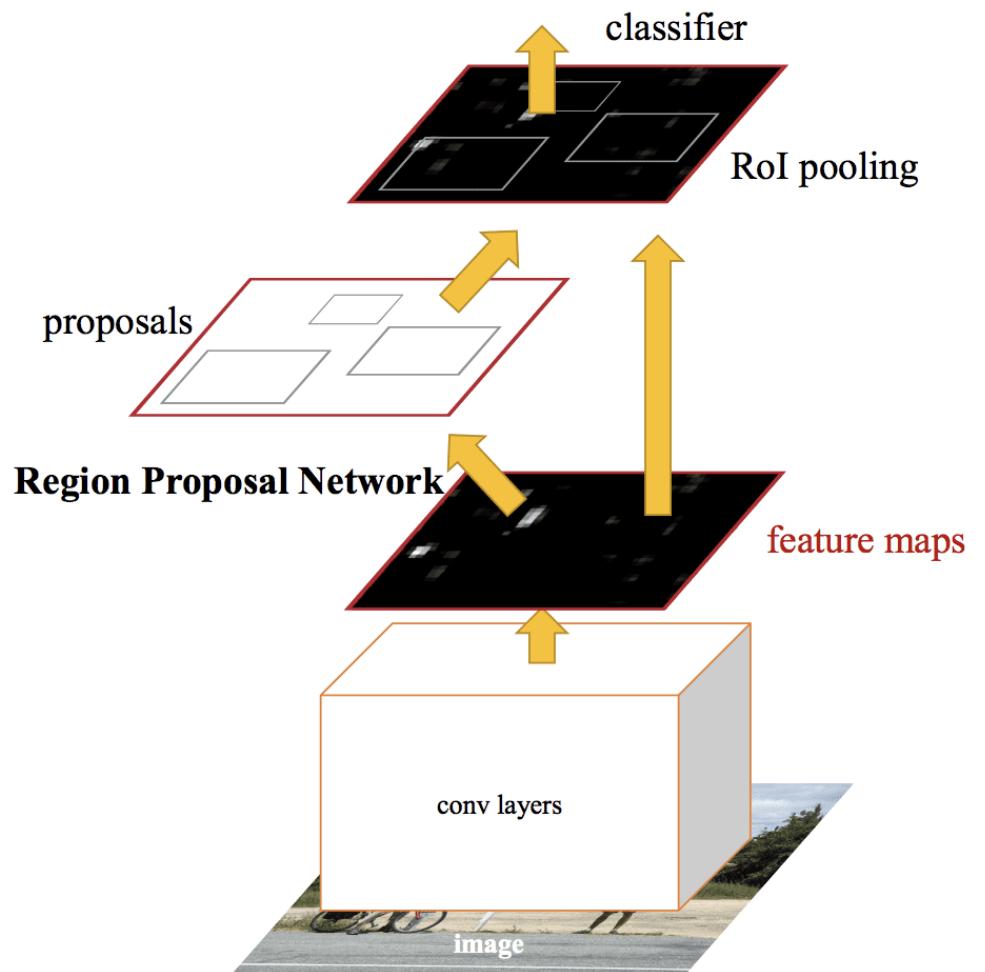
region , or union of regions , with the highest confidence will be taken . The same technique can be used to make objects segmentation .



## Faster R-CNN

An evolution of the R-CNN is the Faster R-CNN which involves the use of a single model to carry out the three steps of region-proposal , features extraction and classification.

Through RoI Pooling, only the most promising Regions of Interest are selected . Using only one model, this technique achieved first place in 2016 for accuracy and is also much faster and more accurate at proposing regions of interest (around 300 per image).



# Yolo

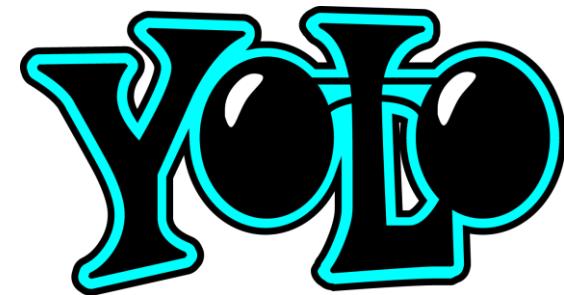
You Only Look One – Yolo , is one of the object techniques most famous and used detection .

Created by **Pjreddie** who also gave life to his framework ( darknet ) and several other algorithms.

<https://pjreddie.com/>

His CV is used as an example in human resources courses

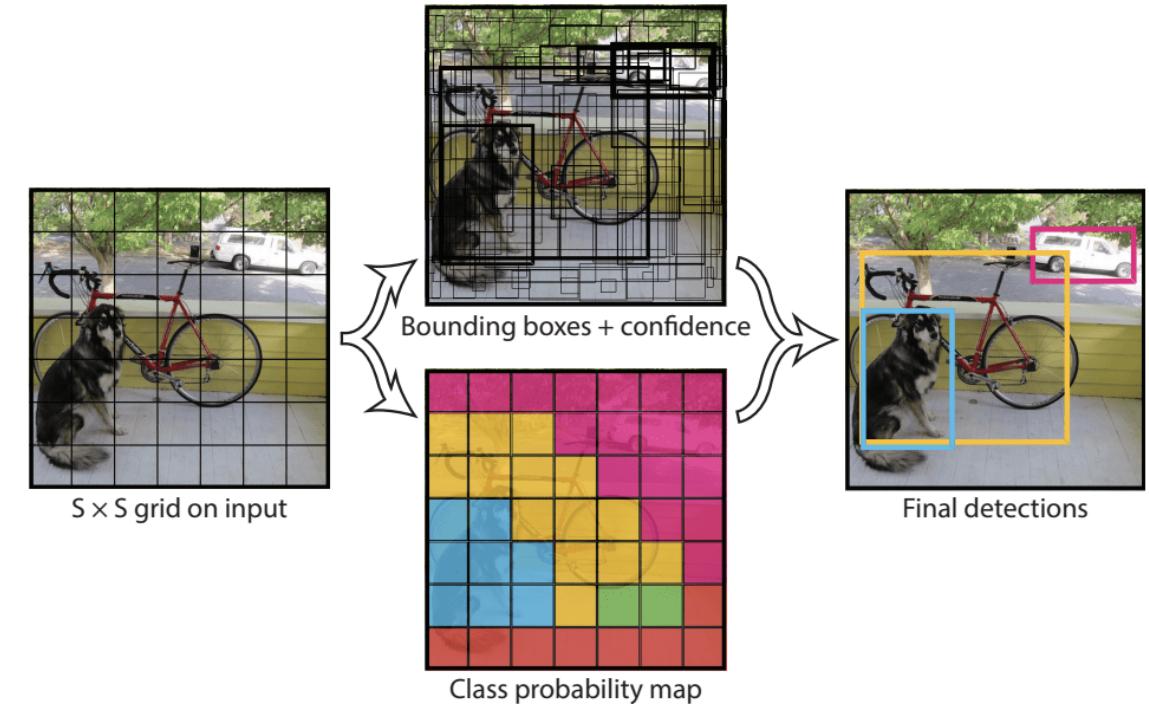
<https://pjreddie.com/static/Redmon%20Resume.pdf>



# Yolo

Yolo networks are usually faster but less accurate than R-CNNs or other types of models.

It is based on dividing the image into cells, each cell is responsible for classifying an object if the center of its bounding box falls inside it. In addition to the classification it must also tell the center offset, height, width and confidence.



## Other models

In addition to Yolo and Faster R-CNN there are many other more recent models that are used to make objects detection

### **Yolo-based : Yolo-v3, Yolo-v5, Yolo-v7, YoloR , YoloX , ...**

Normally faster but a little less accurate

### **EfficientDet , ConvNext ,**

Again based on convolutions, generally more accurate depending on the year of release

### **Transformer- based : ViT , DETR, DINO**

Based on Transformer which we will see in the lesson related to text processing

## Recap

They remain to be seen

- NMS
- Object segmentation ( unet + yolo head)
- Visual Embeddings
- Gradcam
- Generation ( diffusion + vae )
- Training pipeline



# **12 – Convolutional Neural Networks**

Daniele Gamba

2022/2023

## Previous

We have seen what a CNN is

- Convolutions
- Padding
- Pooling
- Object classification
- Object detection
- Tuning a Yolo to identify road signs

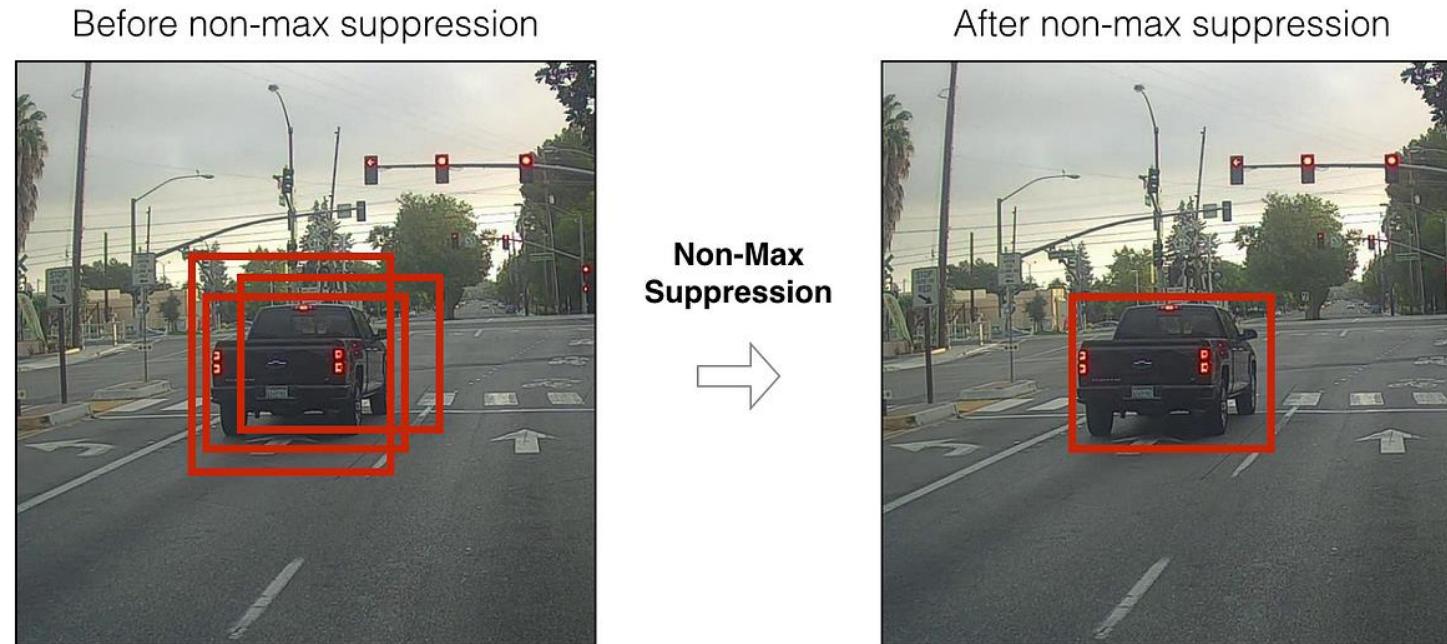
## Recap

They remain to be seen

- NMS
- Object segmentation ( unet + yolo head)
- Visual Embeddings
- Gradcam
- Generation ( diffusion + vae )
- Training pipeline

## Non-maximum suppression

Anchor-based models (e.g. Yolo ) make N predictions of the object of interest because each anchor believes it owns the center of the object. Consequently we find ourselves having to discriminate between many predictions as to which one best represents the object.



## Non-maximum suppression

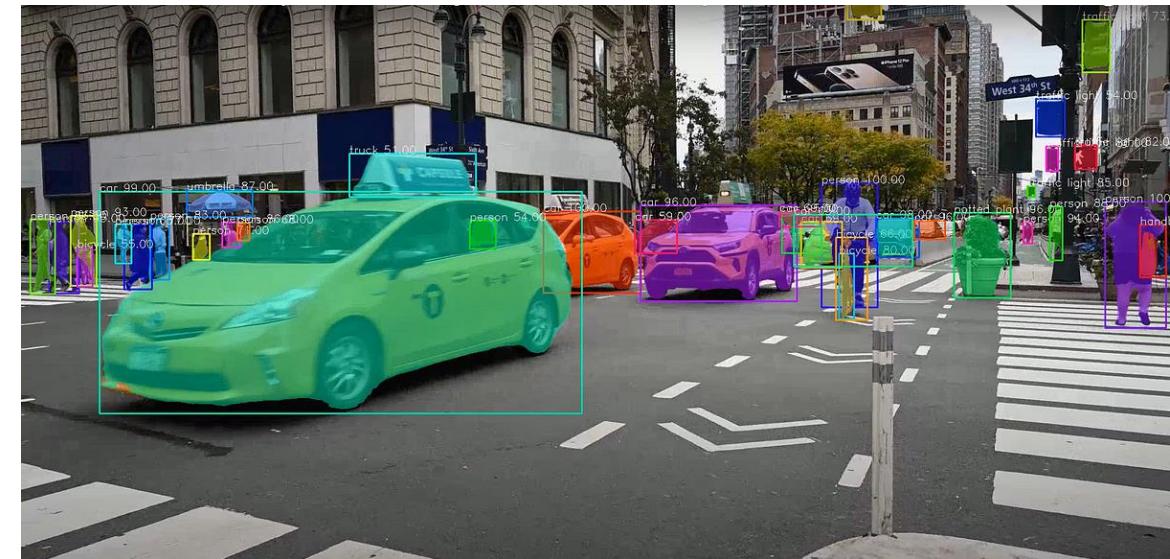
To filter bounding boxes

- We take the bb with the highest confidence and keep it as good
- We take all the other predictions of the same class, if the IoU exceeds a certain threshold we remove them from the set of proposals
- Let's start from the first point for each class and each BB with IoU lower than the threshold

# Object segmentation

In case we wanted to train an algorithm not only to frame our object in a bouding box but also to indicate the individual pixels.

The semantic segmentation task instead deals with identifying only the class of each pixel without identifying whether they are instances of different objects

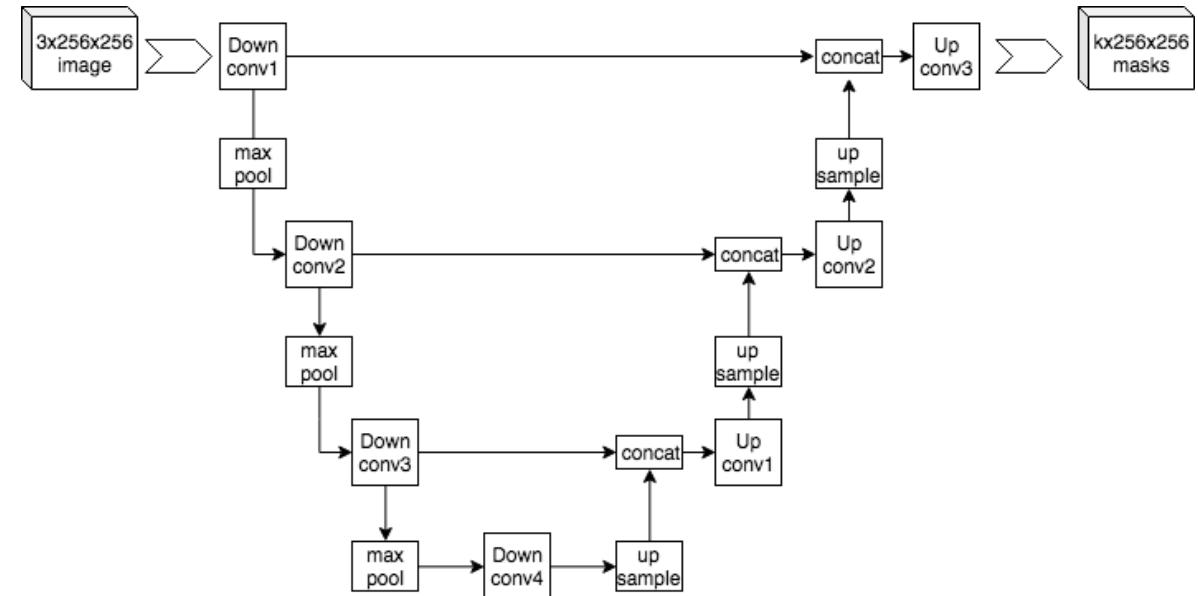


## U-Net

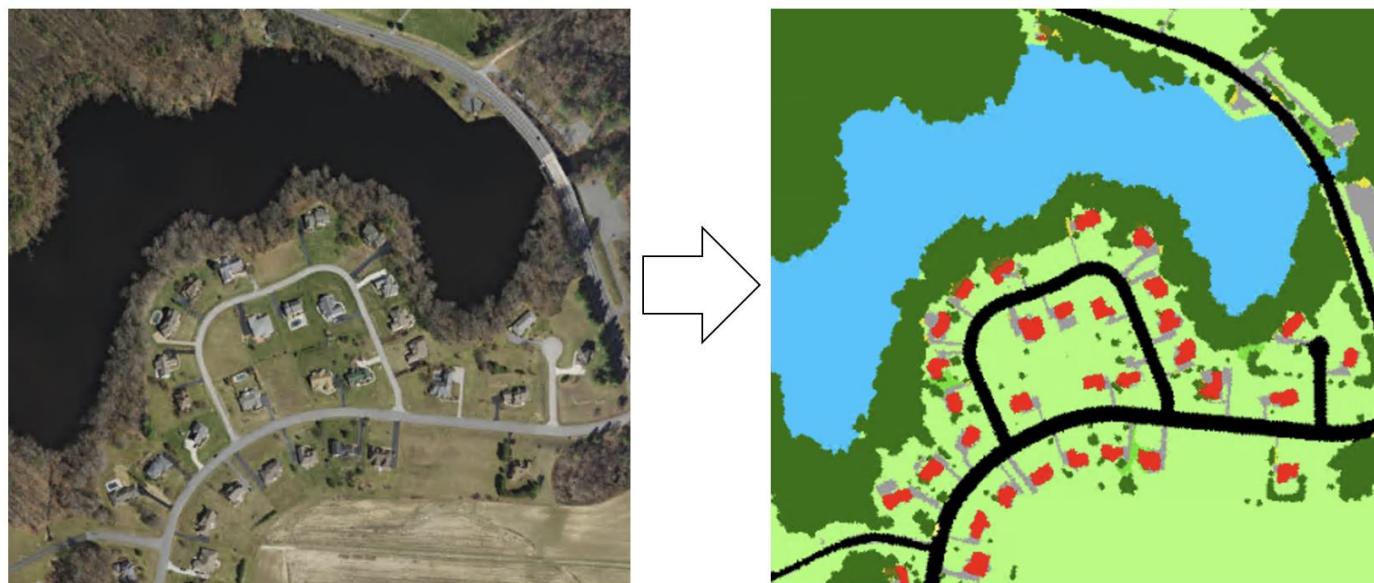
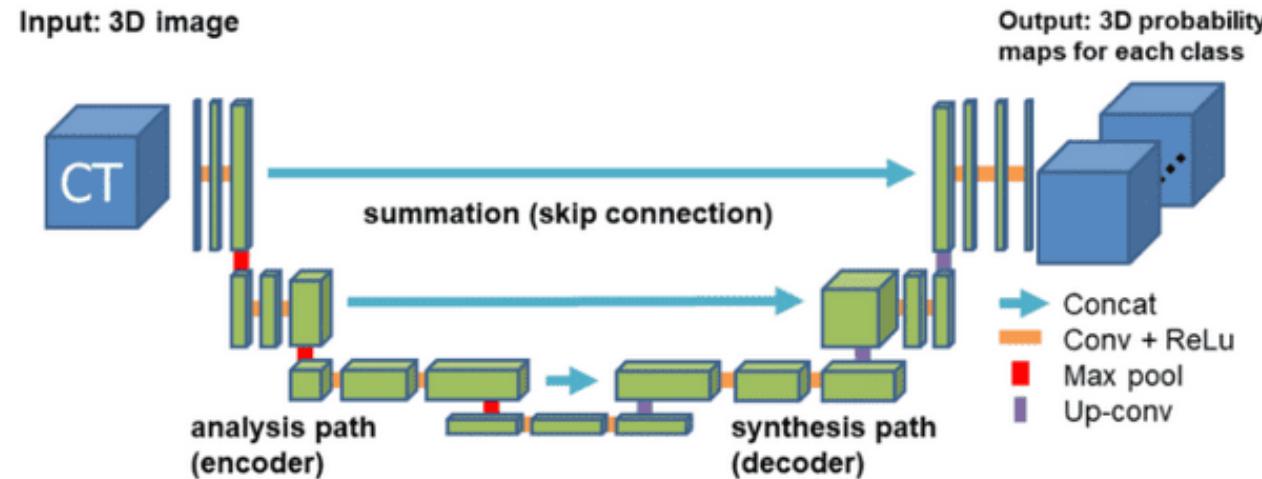
One of the main techniques for segmentation is the U-Net.

The way it is built, it tries to create a U in which we have convolutions that gradually extract deeper and deeper features, and direct connections that transport the data to different levels of detail.

The deep features interpret the groups of pixels, the layers then try to remap the information onto the larger image.



# U-Net



## U-Net

To train U-Nets we need labels that are as expressive as they can be

- Polygons that enclose the object of interest
- Real images / maps that we want to get out of the network

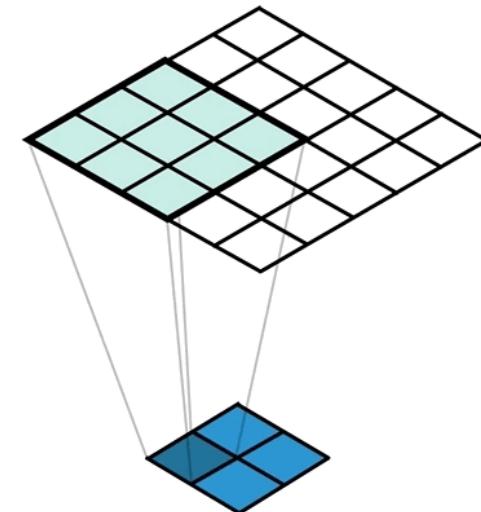
The cost function must try to maintain a good level of detail, so we tend to use cost functions that do not quadratically scale the most distant errors but functions like MAE.

We also need to know how to move from a low resolution layer to a larger one.

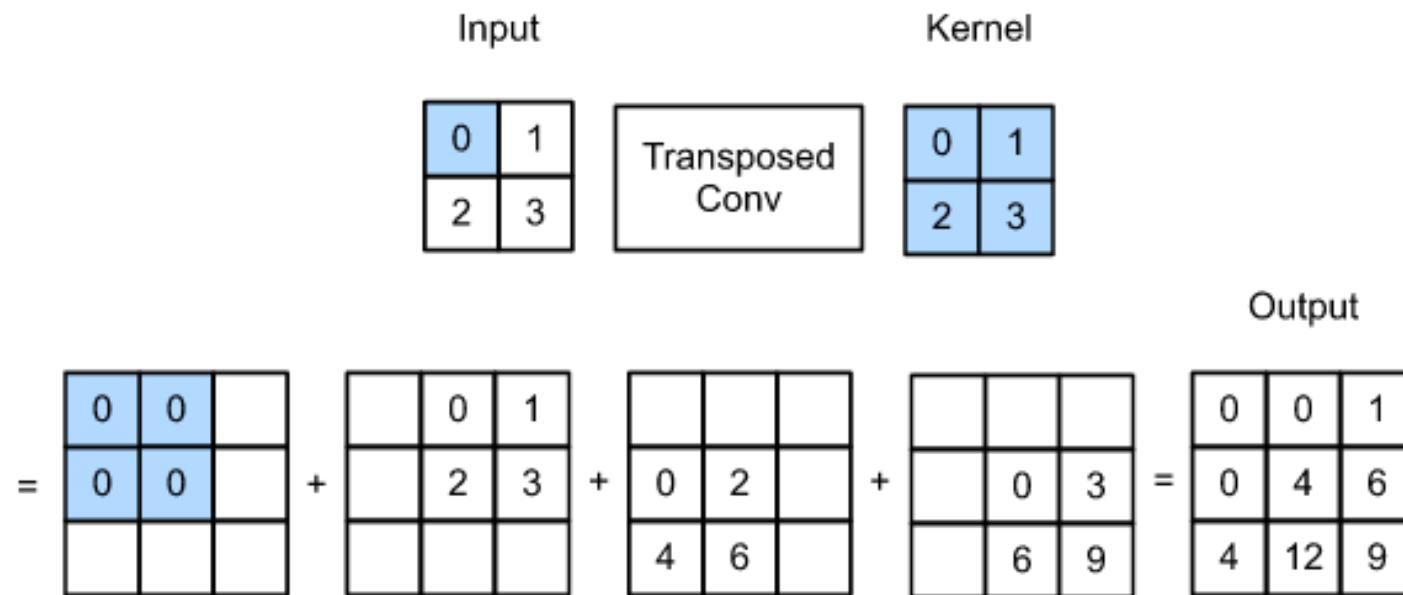
## Conv2DTranspose

In the U-Net we need to scale up once we reach the lowest level of our CNN. To do this we can use Conv2DTranspose to do upsampling .

Our layer will learn a kernel (as in convolution) which it will then use to upsample the previous layer .



## Conv2DTranspose



## Yolo & Mask R-CNN

Both R-CNN and Yolo can be repurposed to make objects segmentation .

Both do this by predicting either a list of points that contain the object's polygon or by predicting masks which are then positioned inside each Bounding Box.



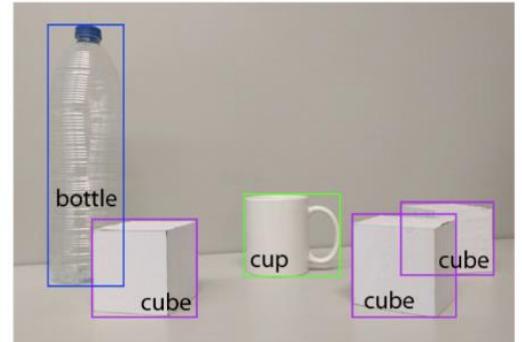
# Instance & Semantic

For implementing a segmentation model it is important to distinguish between Semantic and Instance Segmentation.

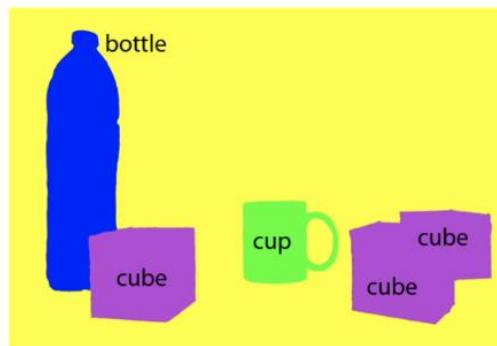
- For **Semantic segmentation** we want just to know at what class belong the pixel behind. For this kind of problems Unet are often used and the output is a matrix with the same dimension as the input image
- In **Instance segmentation** we want to distinguish each object of interest but not the background for example, in this case the output of the model is the coordinates of the polygon inscribing the object.



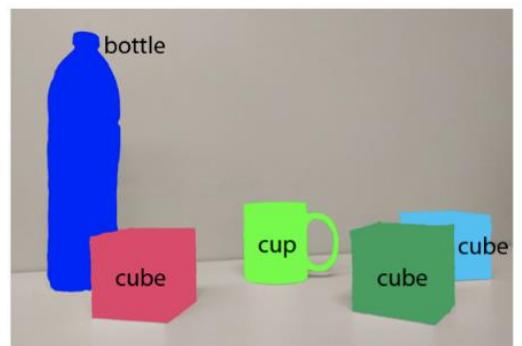
(a) Image classification



(b) Object localization



(c) Semantic segmentation



(d) Instance segmentation

## SAM

Segment Anything Model (SAM) is the latest model released in the object world segmentation from Meta.

It is a huge model compared to models like Yolo or UNet and in fact it was largely trained in self-supervising learning on an infinite dataset (11 million images and more than 1 billion masks).

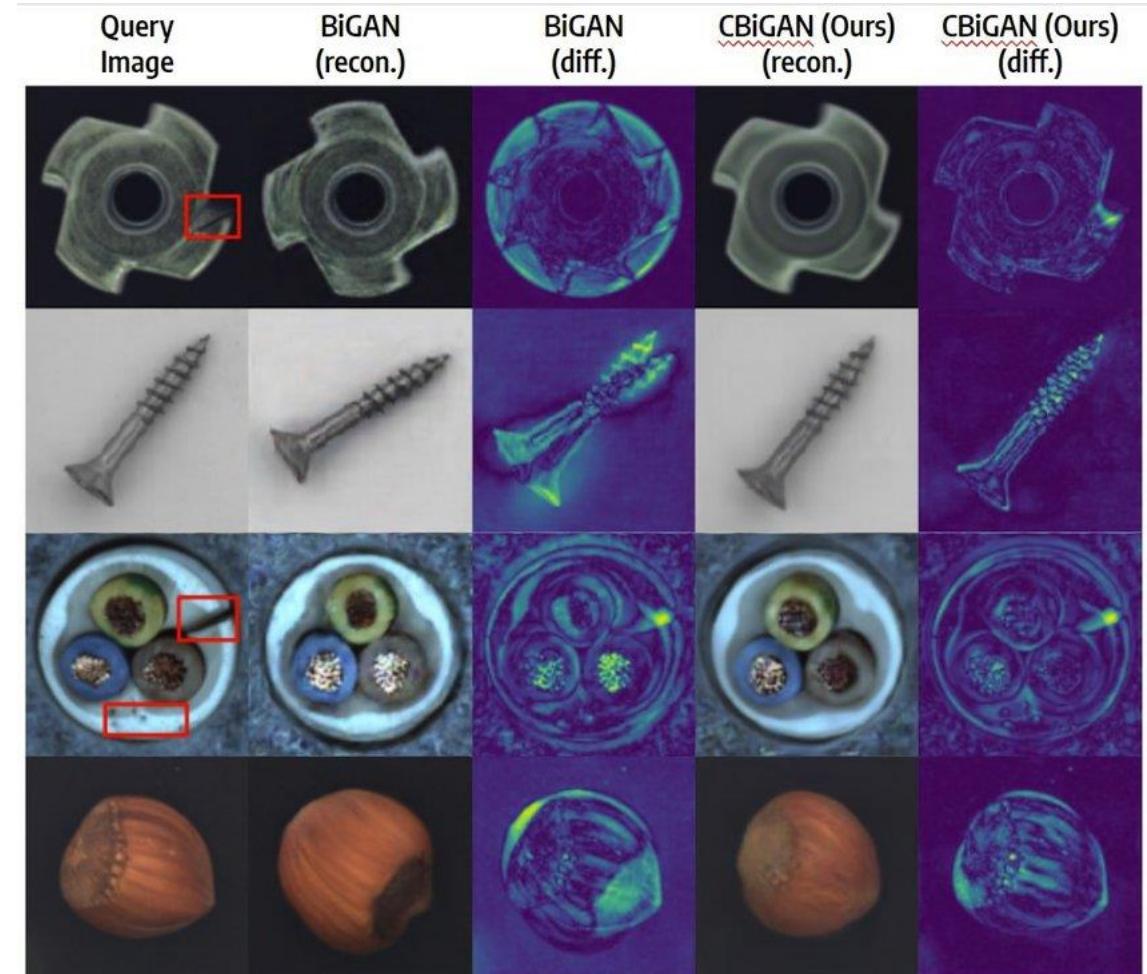


Generate multiple valid masks for ambiguous prompts

## Visual Anomaly

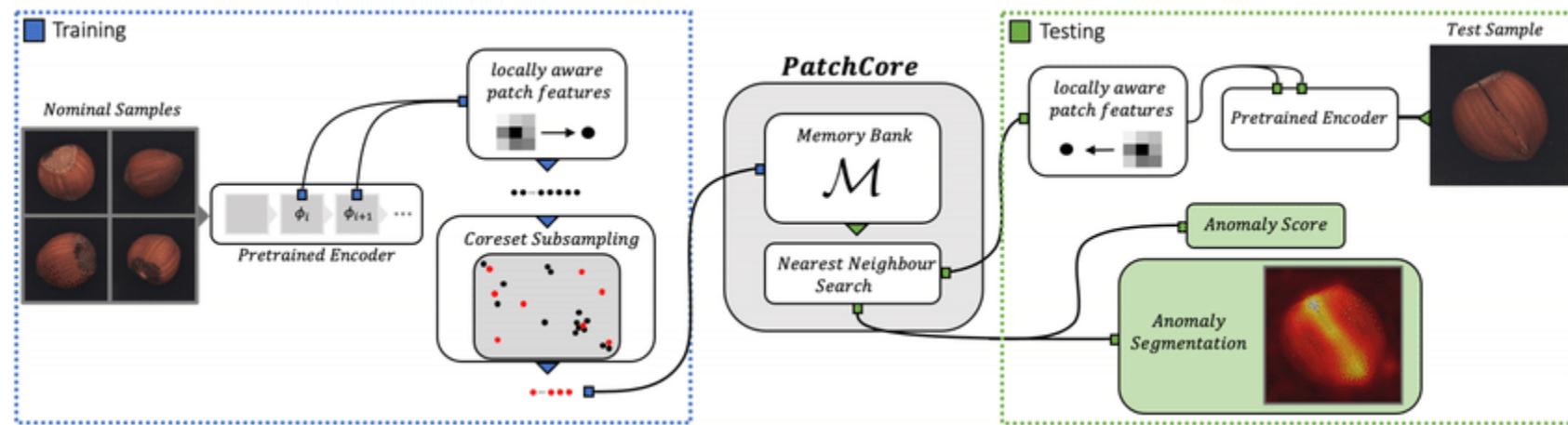
What if we don't know exactly what we're looking for but we just want to identify anomalies?

This is another task that we'll see later with Autoencoders but could be approached also with convolutional features.

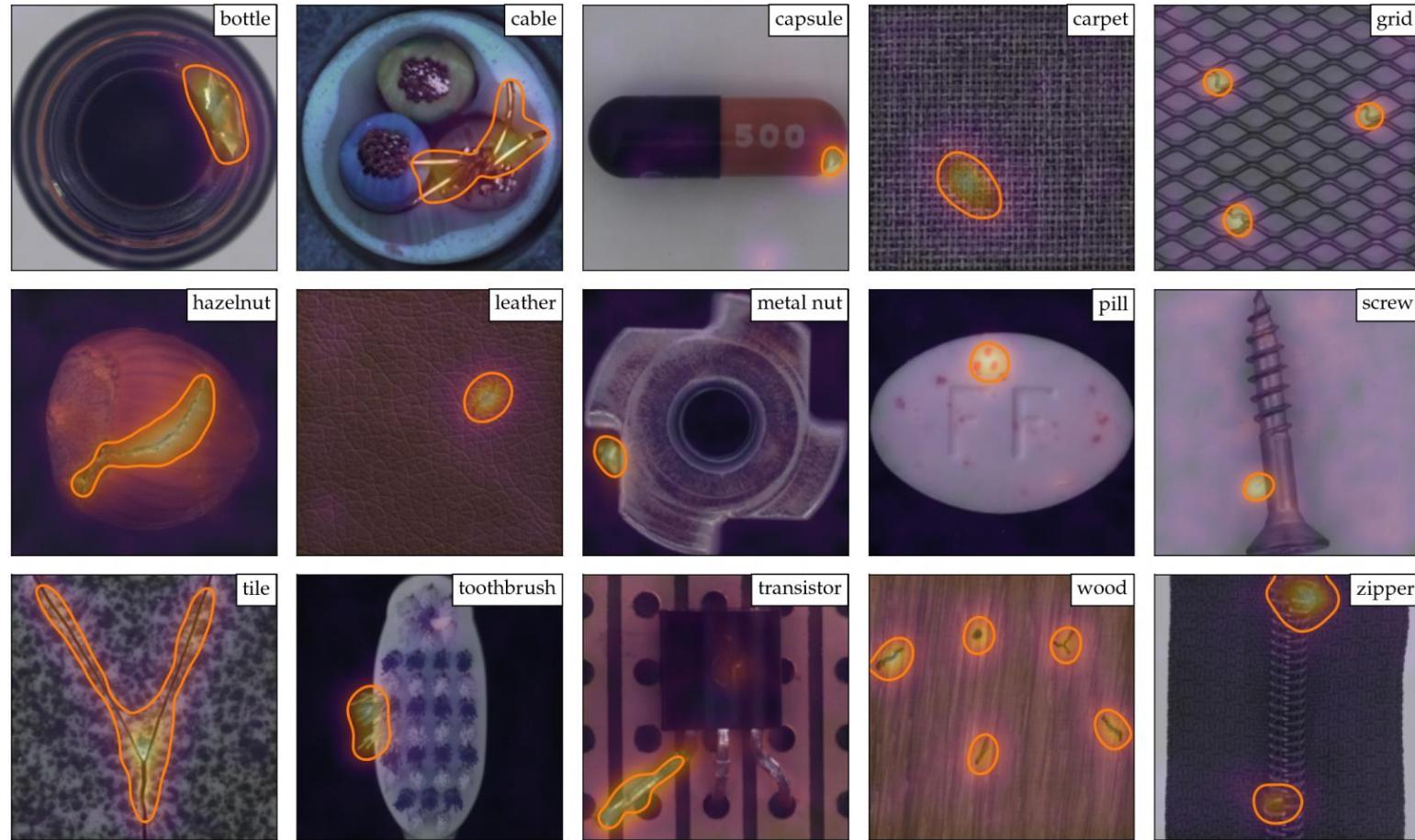


## Patchcore

One of the best algorithms for Anomaly Detection is the Patchcore. The algorithm is very simple, samples features extracted by a backbone, during the training phase characterize the «good» samples selecting a set of features points, during inference will compare the new example with the memorized features of training samples.



# Patchcore



## Recap

We have seen

- non-maximal suppression
- Object segmentation with
  - Unet
  - Standard models ( Yolo , MaskRCNN )
  - SAM
- Intro to anomalies

# **13 – RNN**

Daniele Gamba

2022/2023

## Input sequences

So far we have seen networks that have inputs of known dimensions, for example our feature vector or the image size.

Except Fully -Convolutional NNs allow us to scale to larger dimensions than they were trained on, how can we deal with data with variable lengths?

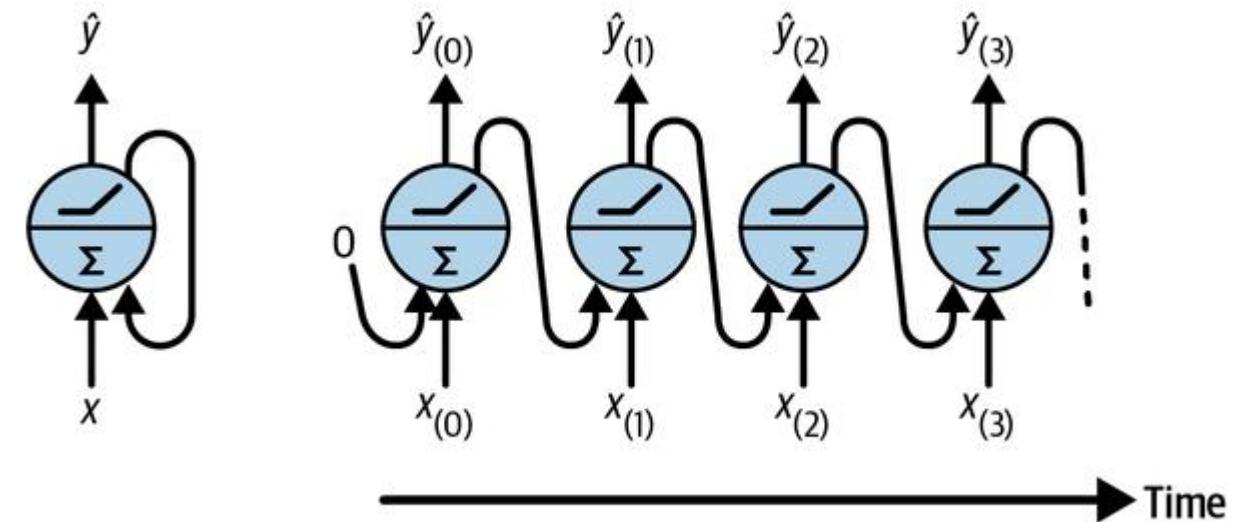
How do we handle input sequences to predict the future of a time series?

# Recurrent Neural Networks

Recurrent neural networks, or **RNNs**, are structures that allow us to process arbitrarily long sequences.

They are based on the recurrent neuron, a structure capable of receiving an input and passing a value as a second input to itself.

This way, regardless of the length of the series, they can continue to process input.

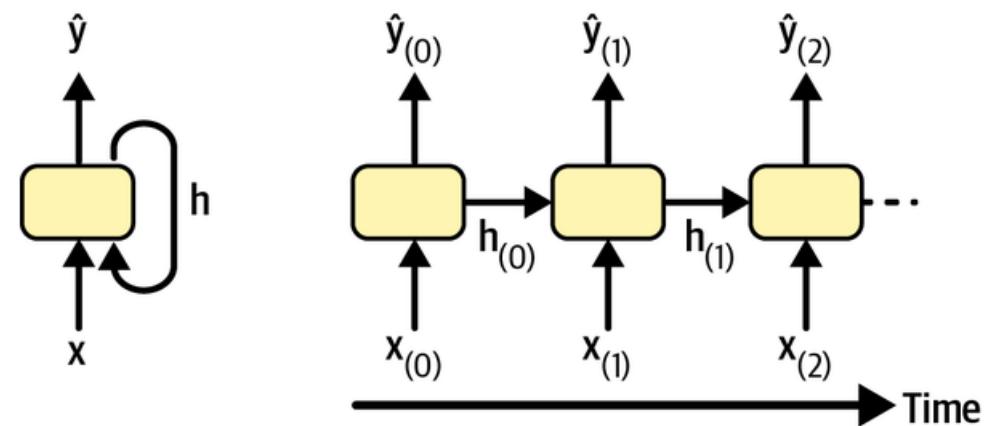
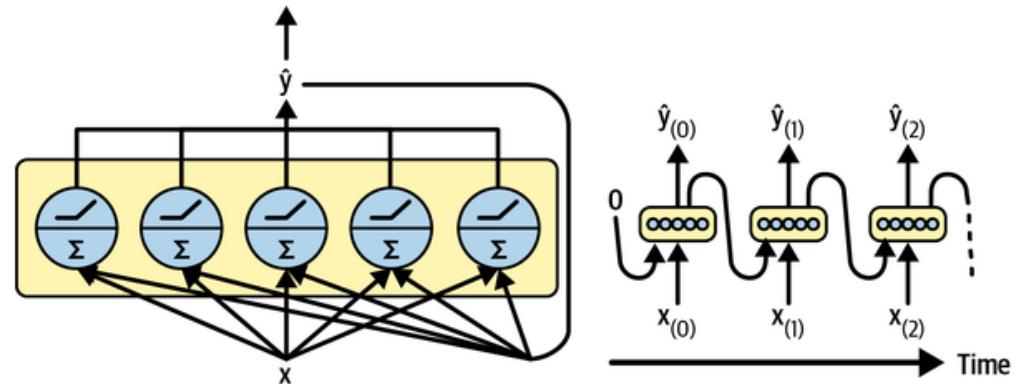


## Recurrent Neuron

Each neuron will have two sets of weights, those related to the external input  $x_t$  and those related to its own  $y_{t-1}$

In this way, neurons accumulate a memory of what has been observed.

The passed value does not necessarily equal the output, we can also choose to pass a hidden state  $h$  as we carry out ( unroll ) the calculation.



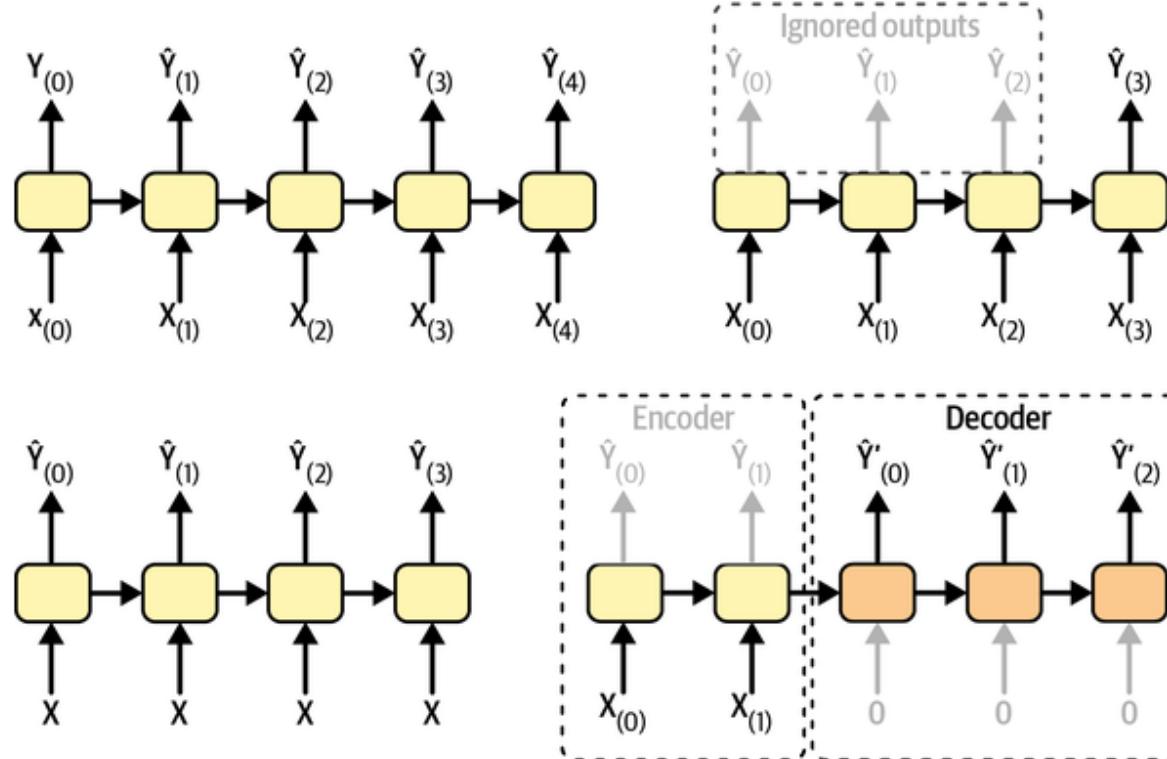
## RNN structures

The RNN is very flexible and we can create very different structures

- Sequence to Sequence , where both input and output are calculated together
- Sequence to Vector , if we want to represent an arbitrary sequence in dense form
- Vector to Sequence , to unfold the contents of a vector
- Encoder Decoder, where the encoder network encodes a state internally which then passes to the decoder

The first and last are used a lot for, for example, making translations

## RNN structures



*Figure 15-4. Sequence-to-sequence (top left), sequence-to-vector (top right), vector-to-sequence (bottom left), and encoder-decoder (bottom right) networks*

## AR, ARMA, ARIMA, etc.

When modeling time series, let's not forget the importance of traditional models.

# RNN

An RNN is nothing more than a neural network that makes use of recurrent neurons.

We can build more or less deep structures, also integrating dense parts.

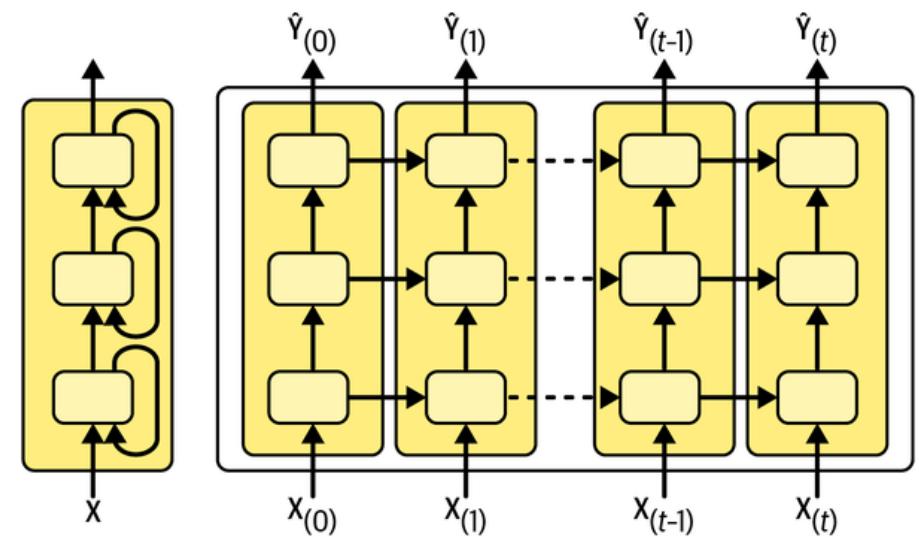


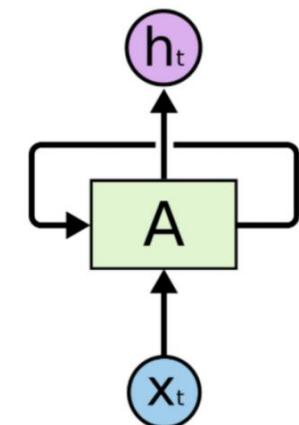
Figure 15-10. A deep RNN (left) unrolled through time (right)

# SimpleRNN

The first recurring Layer we see is called SimpleRNN .

To be able to be used it requires some additional parameters

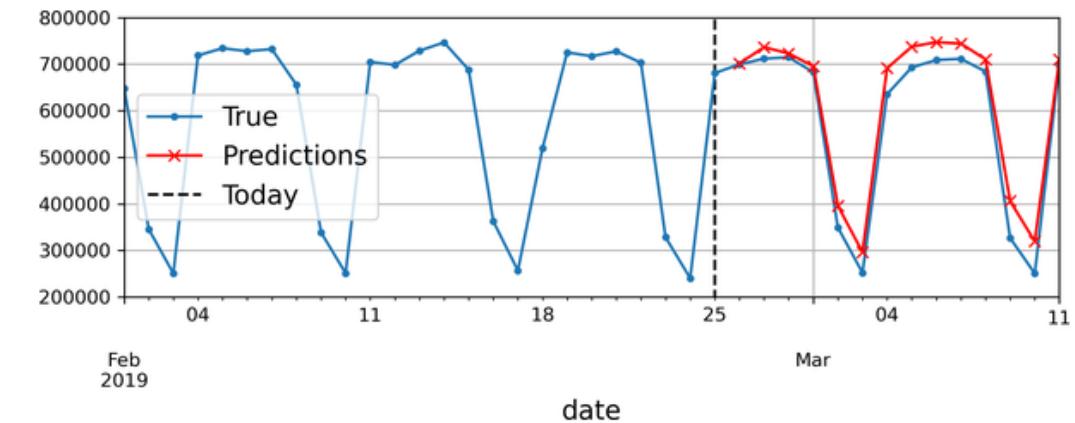
- **units** : i.e. the number of recurring neurons
- **return\_sequence** : T/F if we want the entire sequence to be returned or just the last output
- **return\_state** : if we want the internal state to be returned in addition to the output
- **stateful** : if we want the state to be maintained, it is passed to the next batch
- **unroll** : to unroll the entire neuron in memory instead of leaving it recurring, only for defined and relatively short sequences
- **input\_shape**
  - **batch\_size**
  - **time\_step** : we can indicate the maximum length or None for arbitrarily long sequences
  - **dimensionality** : the number of input features



## Forecasting

To train a network to perform forecasting we will have to give it the sequence as input and the time instant  $t+1$  as output.

Once trained to predict subsequent time instants, we will append our forecast of time  $t+1$  to the input sequence to predict time  $t+2$ , etc.



## RNN problems

RNNs tend to have several problems

First of all, they are slower in both learning and inference as they have to carry out the entire sequence each time; the next step must wait for the status of the previous step.

They tend to have problems carrying forward the memory, a hidden state is multiplied every time inside the neuron and is unlikely to be carried very long in the sequence. Batch normalization cannot be used effectively, especially if the sequence continues over multiple batches.

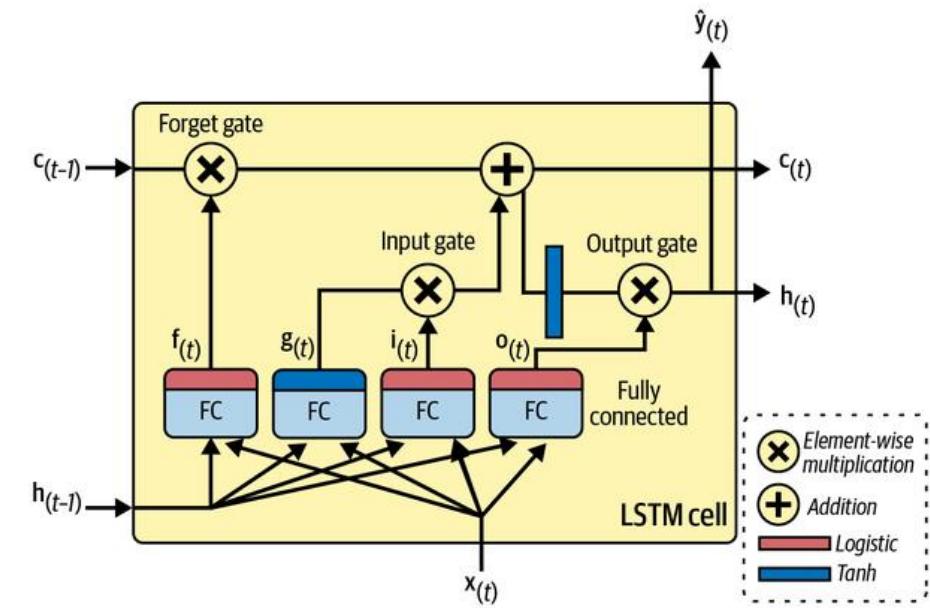
This is why other types of recurrent neurons were invented

## Long-short term memory

LSTM cells were invented to be able to effectively carry forward memory even in long sequences.

These cells have two states that are carried forward, a "long- term "  $c$  and a "short- term "  $h$  .

The cell decides when to maintain long-term memory, when to erase the memory, and when to read it from the input through a series of *gates* .



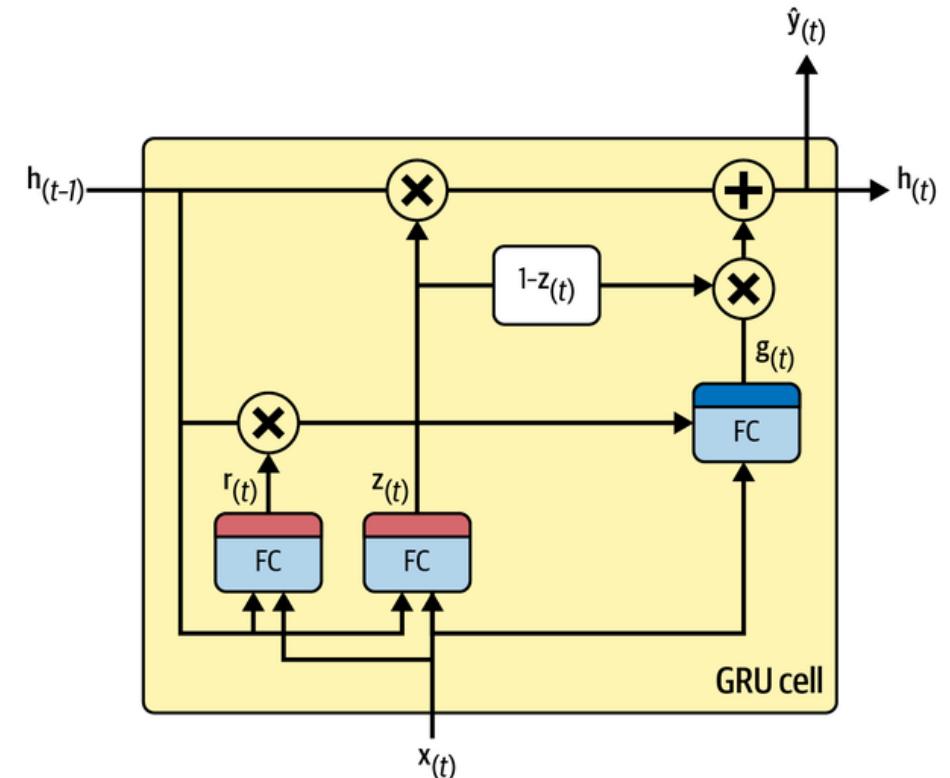
## Gated-recurrent Unit

The Gated Recurrent Unit, GRU, is a neuron that simplifies the LSTM.

There is no distinction between state and output.

The state is unique (there is no distinction between long- term and short- term ) and controlled by a single gate.

In many applications, despite being a simplification, it has proven to perform as well as the LSTM and for this reason it is often more used.

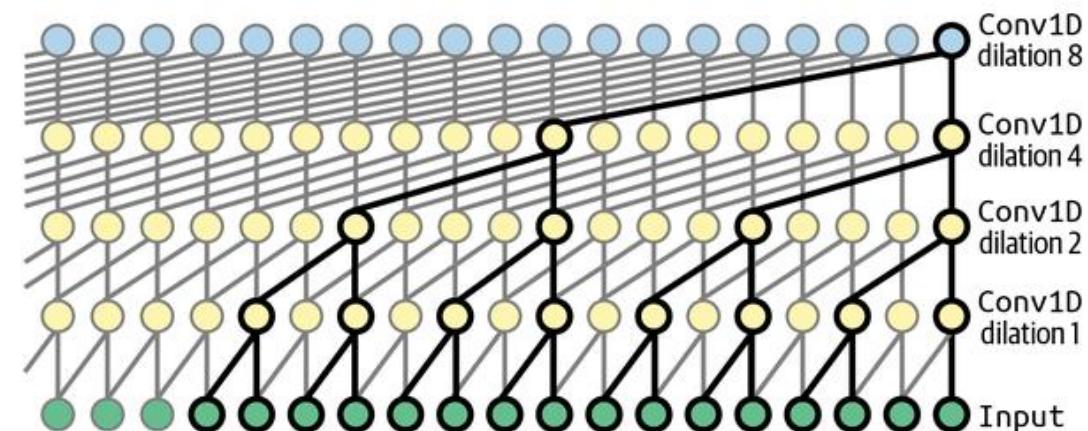


## 1D Convolution - WaveNet

As we saw in the previous lesson, an alternative to RNNs for processing time series of unknown length is to use 1D Convolutions.

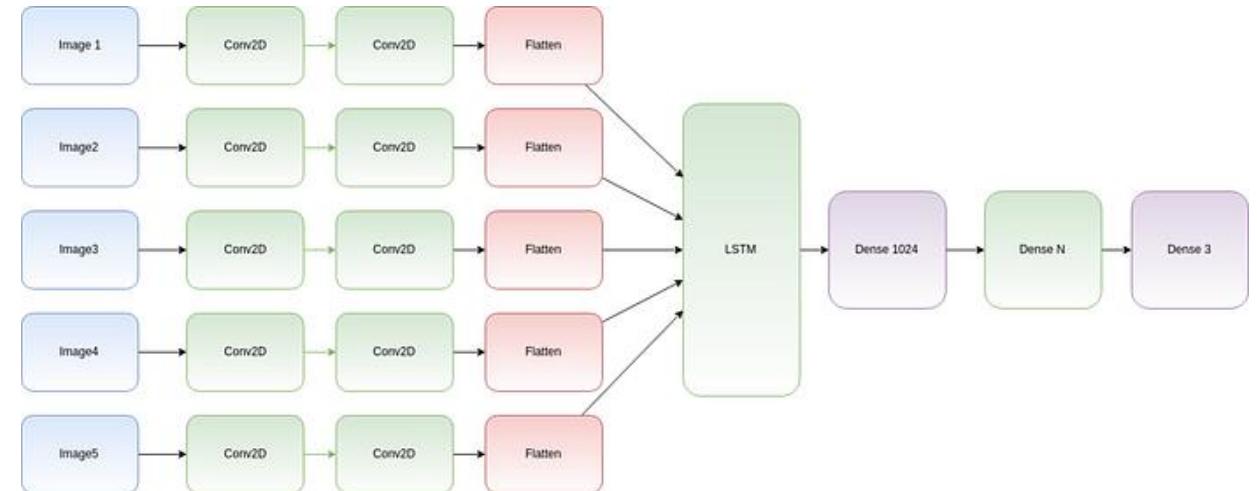
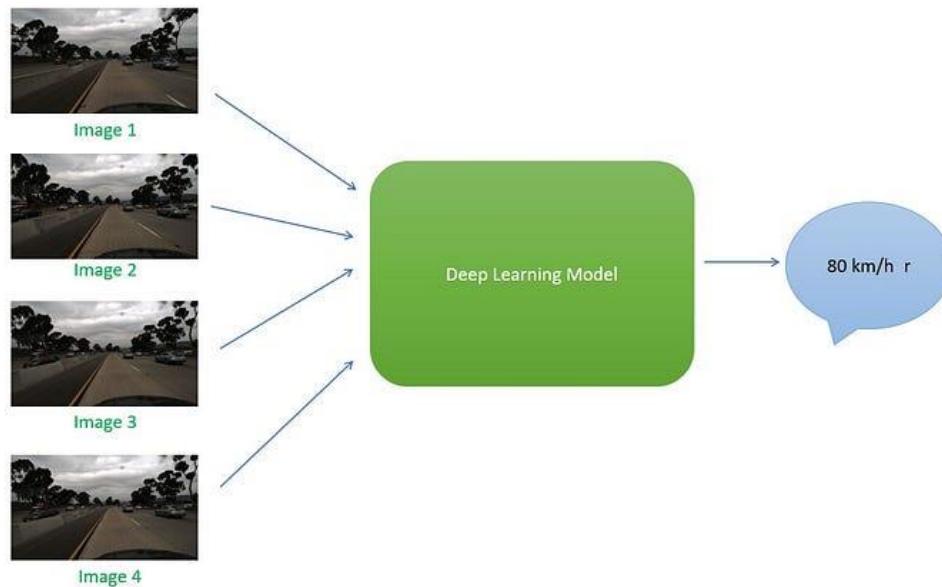
In this sense, the most famous network is WaveNet , in which a series of 1D convolutions are concatenated in which each layer doubles the " dilation rate ", i.e. the distance between the inputs considered.

In this way WaveNet intercepts local "short- term " patterns in the lower layers while in the higher layers it interprets longer sequences despite having no memory.



## TimeDistributed

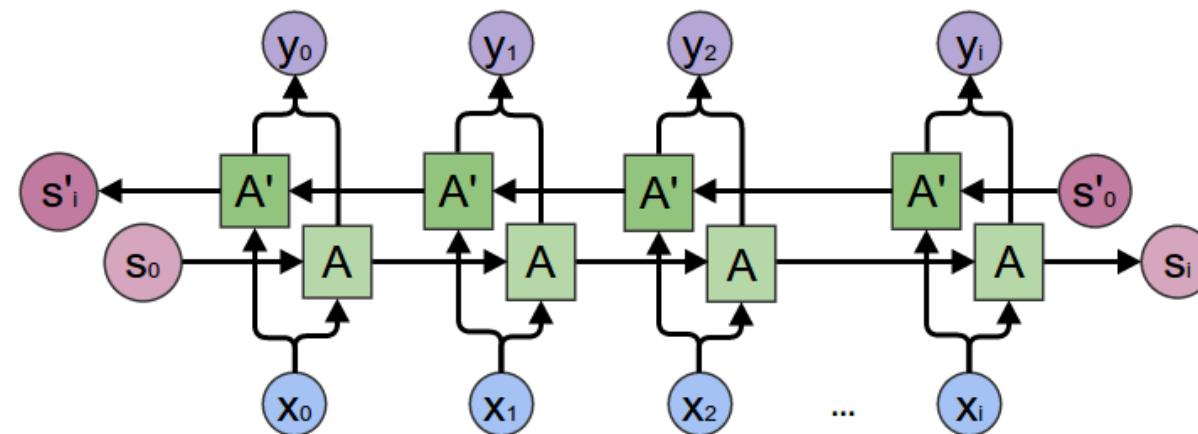
In case we want to apply some pattern pieces independently of others, we can use the `TimeDistributed` decorator . In this way Tensorflow will automatically apply that piece of model along the time axis keeping its weights unchanged and then passing the output to an RNN or Densa.



## Bi- directional RNN

If our inputs are in sequence and we do not have to analyze them as they arrive but we have the entire sequence available, we can use bi-directional RNNs.

In this way the sequence is scanned in both directions and the model is knowledge not only of the previous history but also of the future one. This type of networks is very useful in the case, for example, of textual analysis.



## Recap

We have seen that RNNs are recurrent structures that allow us to have memory.

Having to pass the state of the previous neuron to each neuron, they are more difficult to parallelize and therefore slower.

They are widely used for forecasting and text processing.



# **14 – NLP**

Daniele Gamba

2022/2023

# Natural Language Processing

Natural Language Processing concerns text processing applications.

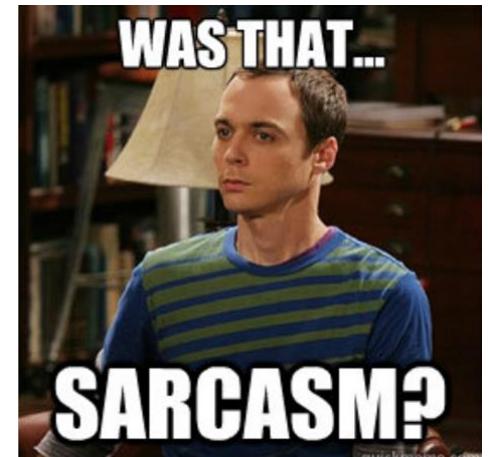
From the text we may want to extract various information such as

- Identify the key parts of a sentence
- Search within a document
- Compare different sentences
- Generate questions and answers

# Difficulty

The text is historically difficult to deal with because

- We have many letters in the alphabet
- Similar letter combinations do not necessarily have the same meaning (house, case, ..)
- The same word takes on different meanings depending on the context
- The meaning of the phrase is not universally defined
- The same concept can be expressed in a multitude of different written forms
- There are different languages and sayings
- On the internet people tend to write very badly (abbreviations, meaningless sentences, CoSECOsII !!11!!)
- Sarcasm



## Representation

First we need to define a numerical representation for the text.

The one historically most used is "one-hot encoding ", that is, for each word we build a vector of as many zeros as our vocabulary is large and with 1 in the position associated with the word.

Clearly this approach has problems

- We need a finite dictionary of words, potentially very large
- The representation is extremely sparse
- We need to find a way to represent words that are similar or have the same meaning

## Pre -processing

A first operation to do, in case we want to represent the meaning of a sentence, is to clean our input from a whole series of characters that can bother us.

Preprocessing will be voted on

- Put everything in lowercase ( CaSa -> casa)
- Remove articles (the, and, of, ..)
- Remove plurals (in the English case \_s)
- Remove commas, spaces, etc.

## Stemming

One way to get closer to a denser representation of our vocabulary is to keep only the roots of words through **stemming**.

*announces, announced, announcing* are removed to get *announc*.

You can use fixed rules or exploit a larger database of rules.

It is clear that in languages like Italian it is more complicated to do stemming to preserve a general meaning (house, case, bank, counter, ...)

## Term Frequency

Once stemming has been done we can move on to representing our sentence vectorically by adding the one hot encodings of all the words.

In this way we obtain what is called **Term Frequency** or **TF**.

d1 – jazz music has a swing rythm.

d2 – swing is hard to explain

d3 – swing rythm is a natural rythm

	a	explain	hard	has	is	jazz	music	natural	rythm	swing	to
d1	1	0	0	1	0	1	1	0	1	1	0
d2	0	1	1	0	1	0	0	0	0	1	1
d3	1	0	0	0	1	0	0	1	2	1	0

## Inverse Document Frequency

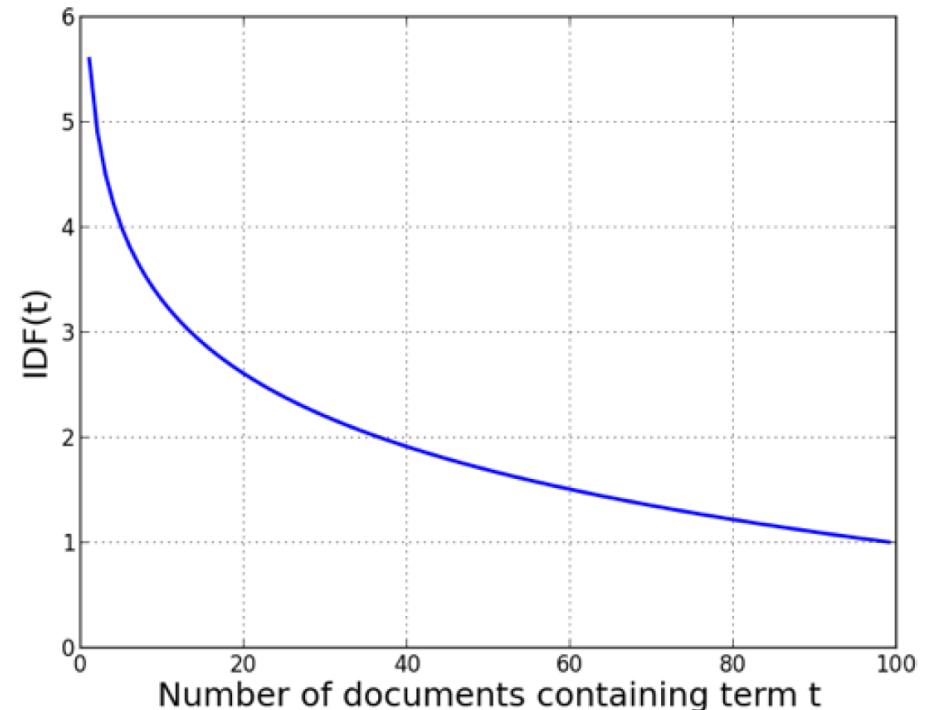
TF has the disadvantage that it tends to favor documents, for example in research, that simply repeat a term many times without adding any meaning. For this reason it is complementary to consider the documents in which a term appears less frequently.

In this case the **Inverse is calculated Document Frequency** or **IDF** and in fact calculates the level of sparsity of a term on N documents.

$$IDF(t) = 1 + \log\left(\frac{\text{total number of docs}}{\text{Number of docs containing } t}\right)$$

## Inverse Document Frequency

Stopwords or other particularly frequent terms will therefore have an IDF almost equal to 1 because it is shared in all documents, while very rare terms (and therefore significant with respect to the definition of information) will have much higher values.



## TF-IDF

The combination of frequency and sparsification allows us to obtain the TF-IDF representation.

$$TFIDF(t, d) = TF(t, d) \cdot IDF(t)$$

In this way, each document is weighted by the frequency of its words and the significance that these have on the total.

Let's say you search for a specific document that deals with a topic, specific words that are only discussed in that context will take on great weight in the document.

# Example – Jazz Musicians

Consider the biography taken from Wikipedia, of 15 Jazz musicians:

## ***Charlie Parker***

Charles “Charlie” Parker, Jr., was an American jazz saxophonist and composer. Miles Davis once said, “You can tell the history of jazz in four words: Louis Armstrong. Charlie Parker.” Parker acquired the nickname “Yardbird” early in his career and the shortened form, “Bird,” which continued to be used for the rest of his life, inspired the titles of a number of Parker compositions, [...]

## ***Duke Ellington***

Edward Kennedy “Duke” Ellington was an American composer, pianist, and bigband leader. Ellington wrote over 1,000 compositions. In the opinion of Bob Blumenthal of The Boston Globe, “in the century since his birth, there has been no greater composer, American or otherwise, than Edward Kennedy Ellington.” A major figure in the history of jazz, Ellington’s music stretched into various other genres, including blues, gospel, film scores, popular, and classical.[...]

## ***Miles Davis***

Miles Dewey Davis III was an American jazz musician, trumpeter, bandleader, and composer. Widely considered one of the most influential musicians of the 20th century, Miles Davis was, with his musical groups, at the forefront of several major developments in jazz music, including bebop, cool jazz, hard bop, modal jazz, and jazz fusion.[ ...]

Even with this small corpus of fifteen documents, the corpus and its vocabulary are about 2,000 features after stemming and stopword removal

# Example – Jazz Musicians

## Input phrase

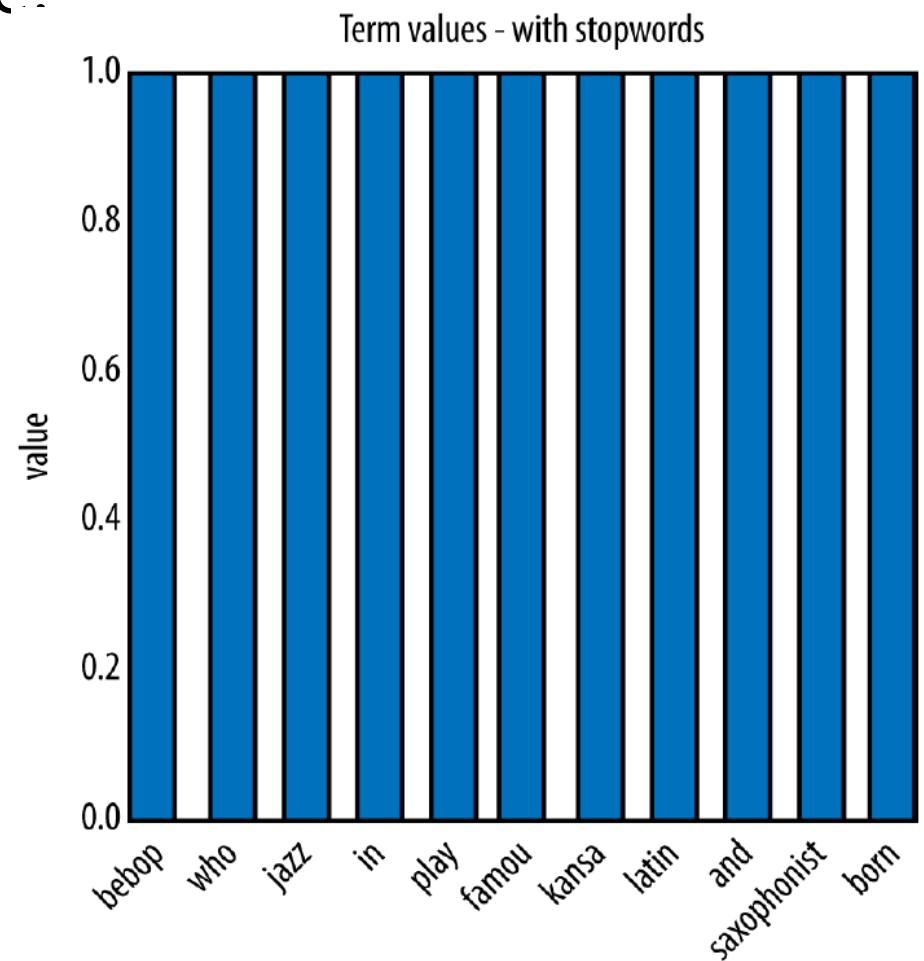
*“Famous jazz saxophonist born in Kansas who played bebop and latin ”*

Consider it as an input into a search engine.

## How to represent it

Treat it as a document.

1. **Stemming** : *kansa* as Kansas, *famou* as famous. It is not important if it is not precise, the importance is the repeatability



# Example – Jazz Musicians

## Input phrase

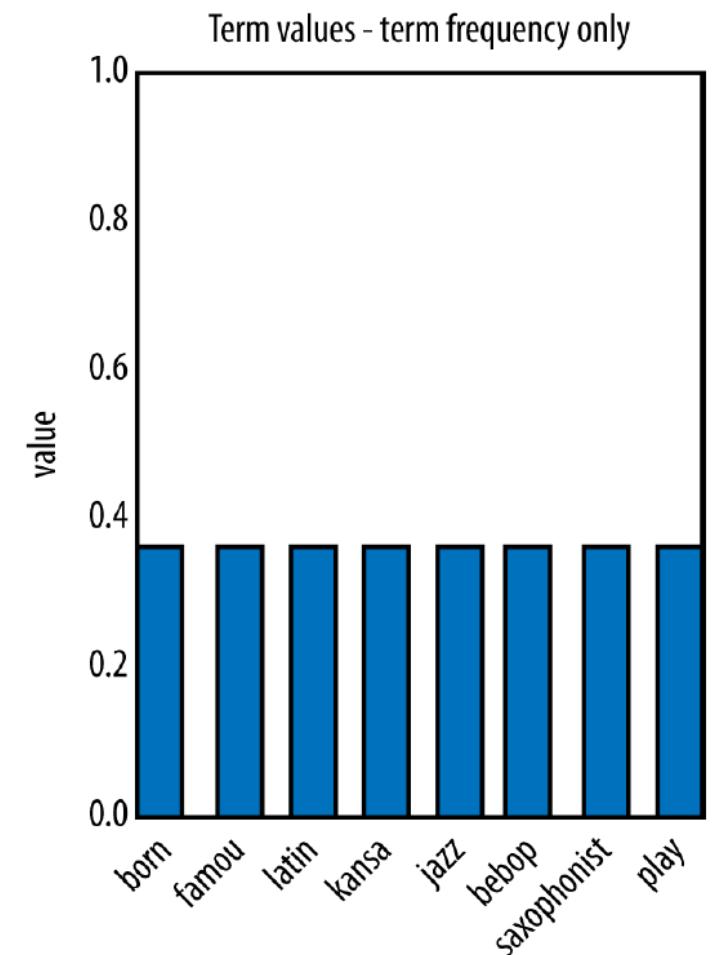
*“Famous jazz saxophonist born in Kansas who played bebop and latin ”*

Consider it as an input into a search engine.

## How to represent it - TF

Treat it as a document.

1. **Stemming** : *kansa* as Kansas, *famou* as famous. It is not important if it is not precise, the importance is the repeatability
2. **Stop word removal** : the terms *in* and *and* are removed and the value normalized



# Example – Jazz Musicians

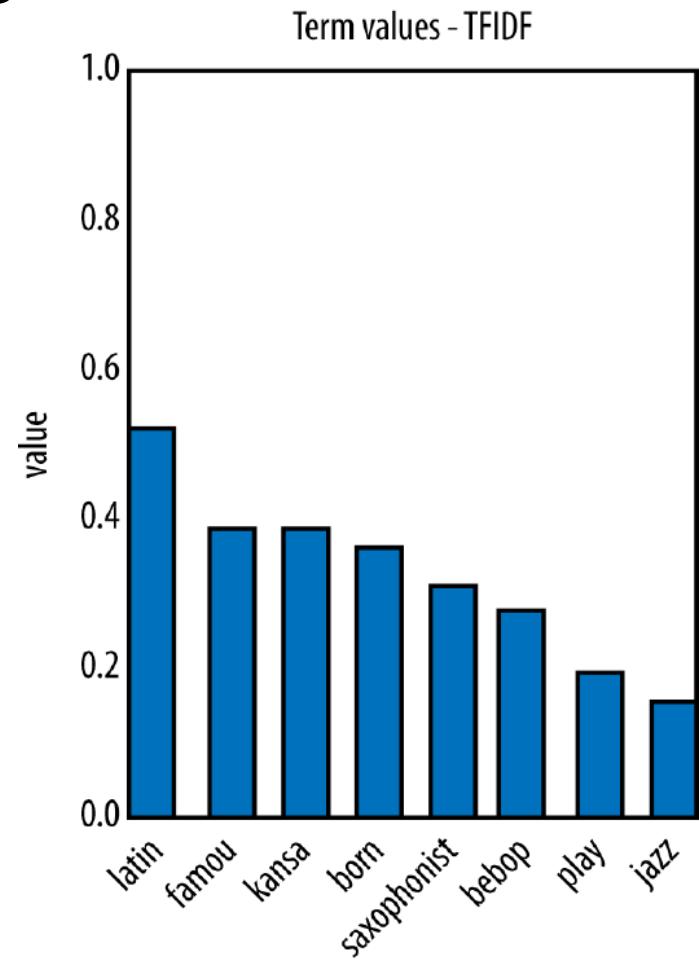
## TFIDF

The next step, requires to compute TFIDF for all the 15 documents we collected, doing for each the preprocessing shown before.

## Input phrase - TFIDF

Once we have the IDF for all the documents, we can compute the TFIDF of our input phrase.

Thus our input phrase is now a vector



# Example – Jazz Musicians

## Similarities

We need to find the jazz musician who has the TFIDF closest to the input phrase.

This means that we need to compute a distance between vectors

## Cosine similarity

A typical approach is to use the cosine similarity:

$$\text{sim}(A, B) = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Musician	Similarity	Musician	Similarity
Charlie Parker	0.135	Count Basie	0.119
Dizzie Gillespie	0.086	John Coltrane	0.079
Art Tatum	0.050	Miles Davis	0.050
Clark Terry	0.047	Sun Ra	0.030
Dave Brubeck	0.027	Nina Simone	0.026
Thelonius Monk	0.025	Fats Waller	0.020
Charles Mingus	0.019	Duke Ellington	0.017
Benny Goodman	0.016	Louis Armstrong	0.012

## Other representations

TF-IDF comes close to a more sensible representation but is still very limited in its representativeness. For this reason, various other representations of the text have been studied.

One of the limitations is that TF-IDF treats each word as independent, while we know well that words have different probabilities when viewed sequentially.

N-Gram is a representation that associates words in 2-by-2 representations in the case N=2.

## N-Gram

«*The quick brown fox jumps* »

**BOW:** { 'quick' - 'brown' - 'fox' - 'jumps' }

**2-gram:** { 'quick' - 'brown' - 'fox' - 'jumps' - 'quick\_brown' - 'brown\_fox' - 'fox\_jumps' }

**3-gram:** { 'quick' - 'brown' - 'fox' - 'jumps' - 'quick\_brown' - 'brown\_fox' - 'fox\_jumps' - 'quick\_brown\_fox' - 'brown\_fox\_jumps' }

*exceed\_analyst\_expectation* triplet is much more significant than considering the three words *analyst*, *expectation* and *exceed* separately .

The big limitation of this representation but it's that increases exponentially the cardinality of the dataset.

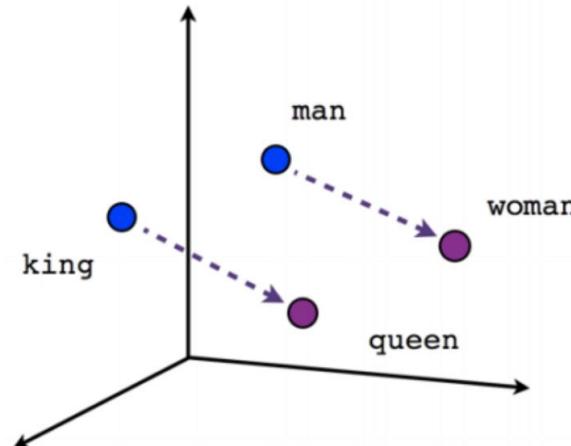
## Word2Vect

With the advent of deep-learning, we have tried to represent words with denser representations, where the vector that characterizes the word is a vector of limited cardinality (e.g. 300).

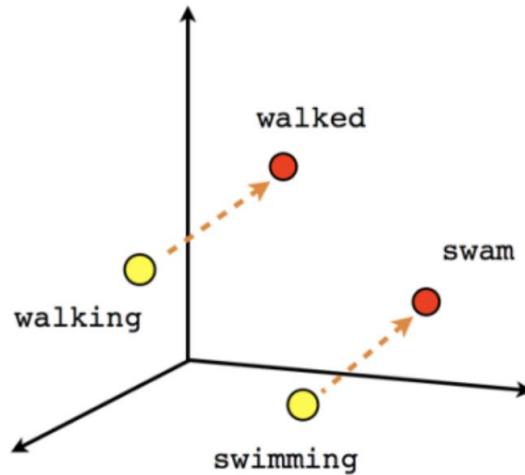
This vector, called **embedding**, contains the meaning of the word placed in an N-dimensional space and in which numerical relationships between words are attempted to be asserted.

This dense representation allows us to find the most similar words in meaning, find opposites until we can carry out operations such as subtracting the "meaning" of *man* from *king* and adding *woman* to obtain *queen*.

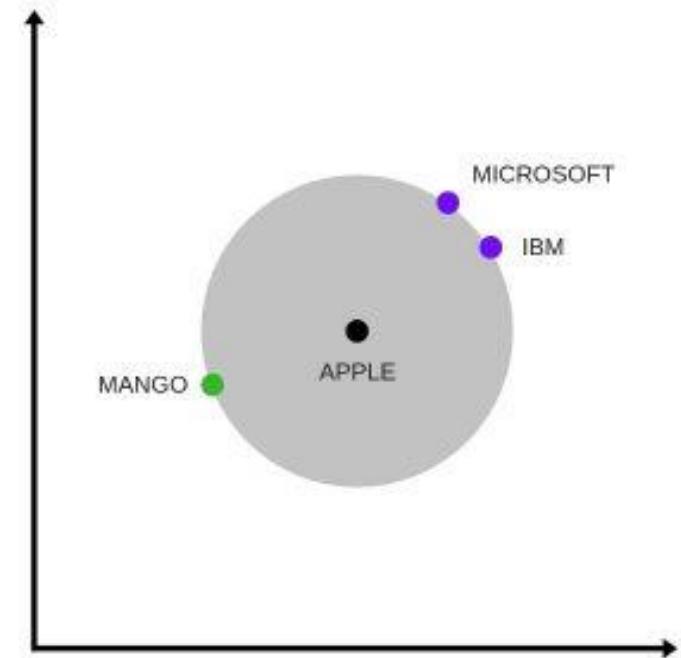
# Word2Vect



Male-Female



Verb tense



## Doc2Vect

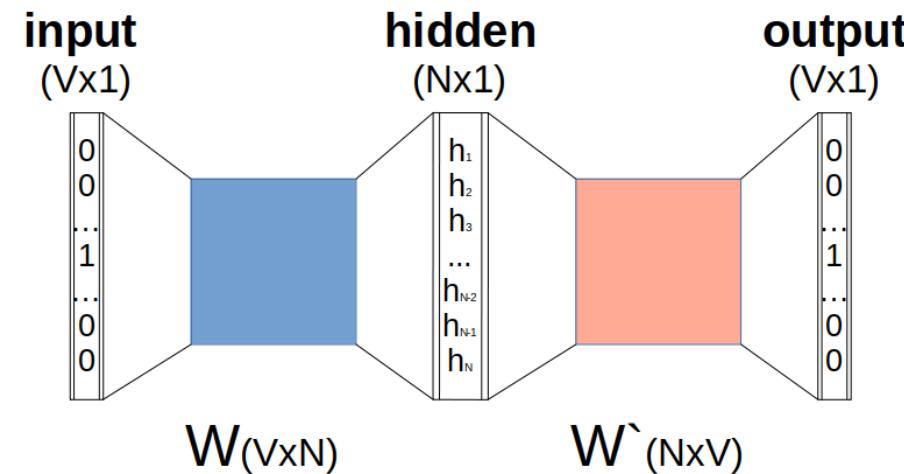
By expanding the representation no longer just to the single word but by encoding the entire document, we can encode the meaning of entire parts of text always in dense vectors.

But how can we train this Dense representation?

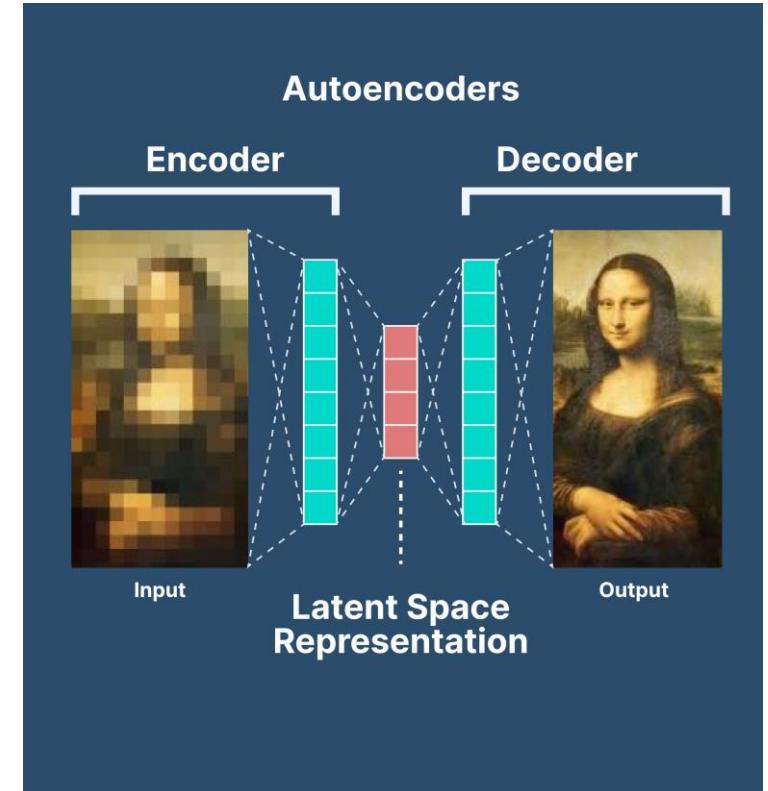
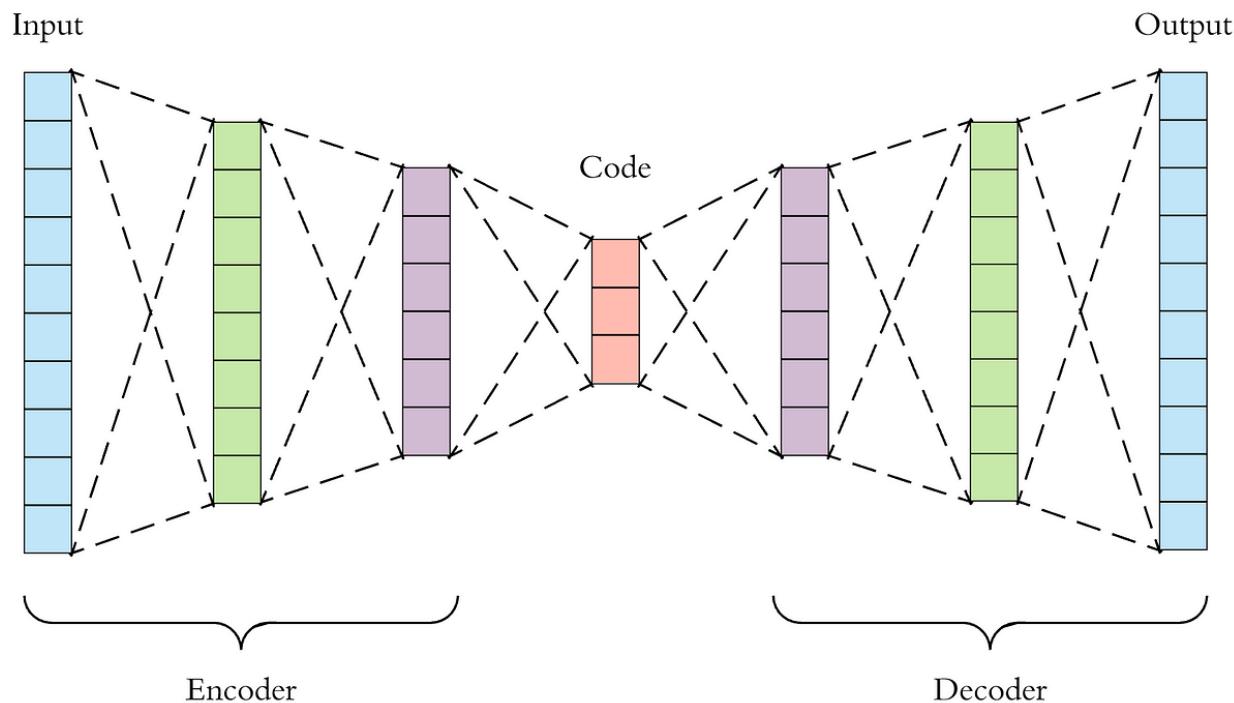
## Autoencoder

In general, dense representations of sparse inputs are trained using autoencoders .

The autoencoder is a neural network structure that takes  $N$  inputs, compresses them into a much smaller dimensionally intermediate representation which then decompresses to reconstruct the original vector. Of size  $N$ . The loss is calculated starting from the difference between input and output.



# Autoencoder

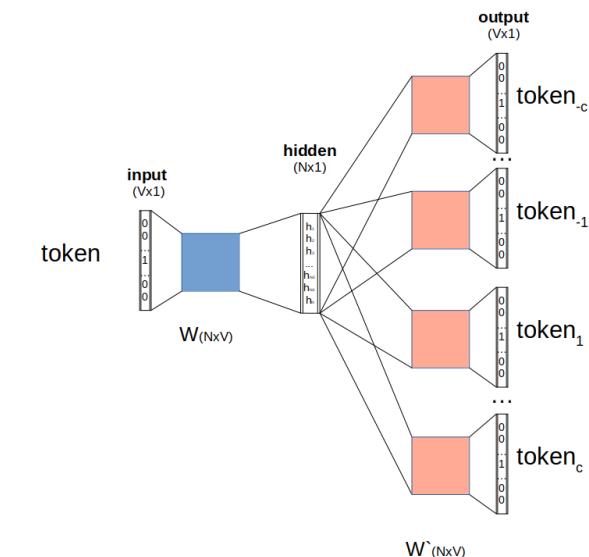
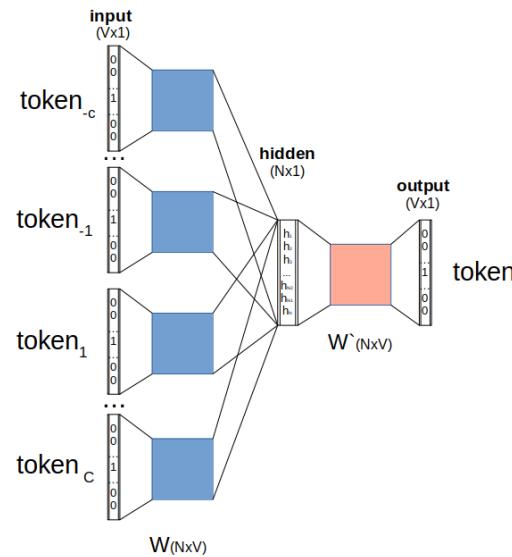


# CBOW and Skip-Gram

There are two similar approaches to training embedding vectors .

CBOW (left) aims to reconstruct a missing token by passing the context as input. The input matrices are held constant.

Skip-Gram (right) instead tries to reconstruct the most probable words given a term. Output matrices may vary.



## Char-RNN

Another approach is to encode all knowledge within a model.

In this way we do not have a dense vector representation but it is the model itself that contains the information on how to interpret and type the characters.

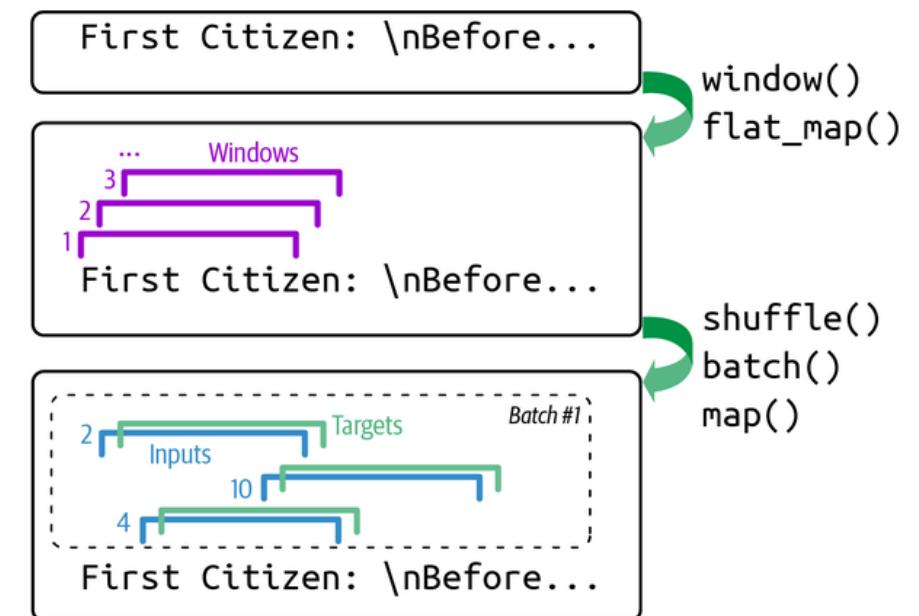
Since the characters are a sequence we could use an RNN that is able to predict the next single character in a so-called CharRNN .

# Char-RNN

To build our CharRNN we first need to encode our dataset so that, given a sequence, it predicts the next character as it unfolds.

We can then code a simple RNN to predict the next OneHot encoded character .

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=n_tokens, output_dim=16,
                              batch_input_shape=[1, None]),
    tf.keras.layers.GRU(128, return_sequences=True, stateful=True),
    tf.keras.layers.Dense(n_tokens, activation="softmax")
])
```



## Temperatures

In the case of text generation, there is a risk of ending up in some loops whereby the same word is always generated repeatedly.

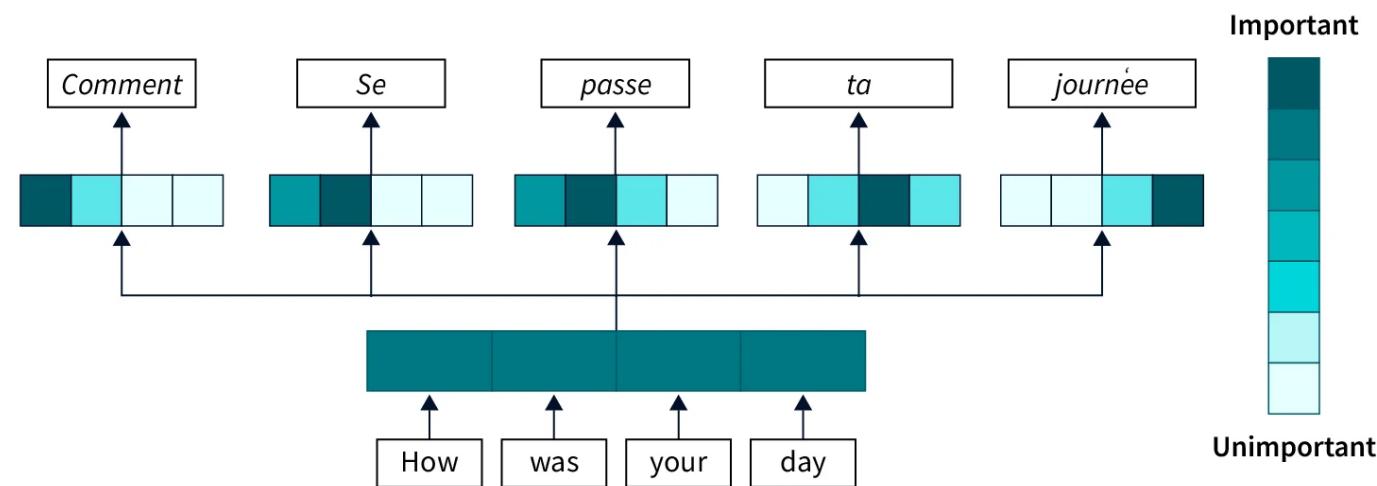
In these cases the concept of **Temperature is used**, i.e. the obtained probability of each output is divided by a temperature value and the most probable are randomly chosen.

Low temperatures (tending towards 1) will always and only prefer the most probable output and are used for example in the generation of equations, while high temperatures (for example 2-3) tend to randomly choose between different outputs and therefore generate more imaginative.

# Attention

A fundamental concept that has revolutionized machine learning in recent years is attention mechanism.

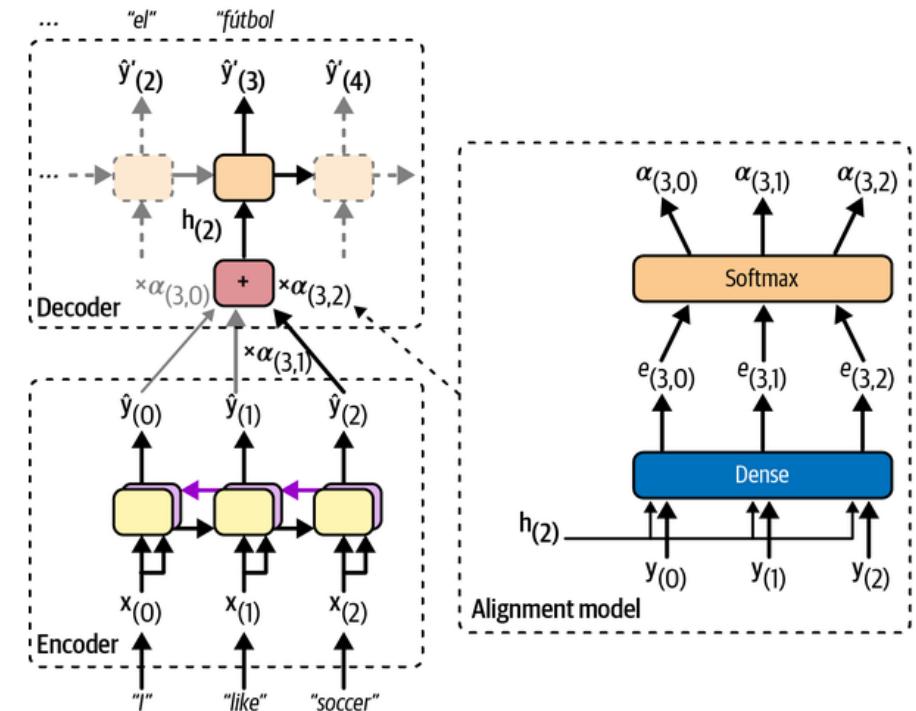
If our network can selectively focus only on different parts of the sequence, we are able to intercept more complex meanings because we link words that are perhaps distant from each other.



# Neural Translation with Attention

Let's take the example of Neural Translation, i.e. translation via Encoder/Decoder and RNN.

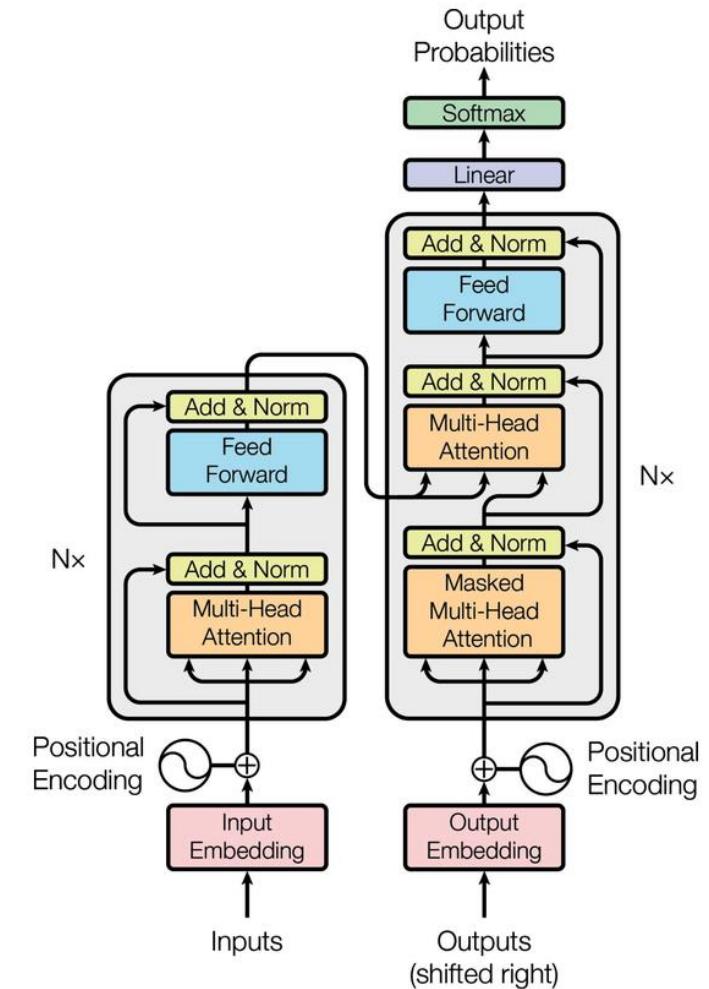
We can dynamically calculate where our decoder needs to pay attention at each instant of time via an alignment model that weights the outputs at each instant of time.



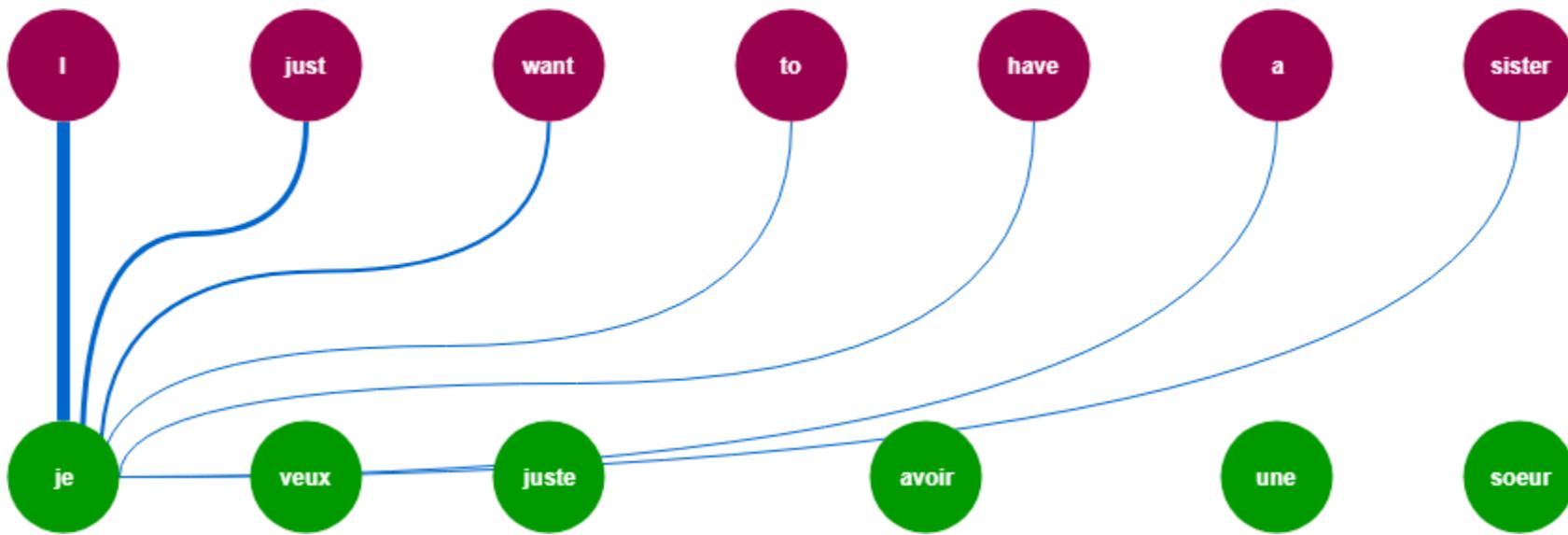
# Attention is all you need

In a famous paper called « attention is all you need », a Google team decided to remove the RNN part from the Neural model Translation to give life to an architecture entirely based on **Attention** called **Transformer**.

In this way there is no scrolling of the text but only an encoding of the input position (*positional encoding*) which is fed to the head that calculates attention.



## Attention Mechanism



# Transformers

best transformers of all time

Microphone icon and magnifying glass icon.

All Images Videos Shopping News More Settings Tools

## Best Transformers

 Bumblebee Mark Ryan	 Optimus Prime Peter Cullen	 Megatron Hugo Weavi...	 BERT Devlin et al.	 Ironhide Jess Harnell	 Starscream Charlie Adler
--	---	--	---	--	---

@debo

# Transformers

From 2018 onwards, the adoption of transformers has exploded, effectively making them the preferred architecture for NLP tasks.

At the moment, the most popular architecture is OpenAI's **Generative Pre-Trained Transformer**, or GPT .



## GPT vs Human

Generative models are extremely powerful but can also be easy to fool.

Challenge Gandalf to make you tell him the password

<https://gandalf.lakera.ai/>





# **15 – Autoencoders , GANs & Diffusion Models**

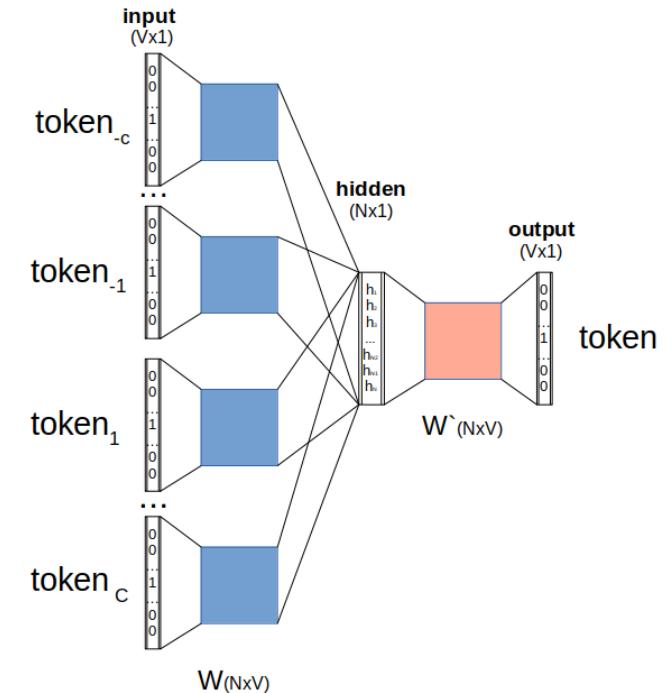
Daniele Gamba

2022/2023

# Autoencoders

We have already seen autoencoders as an excellent structure for compressing very sparse information into smaller, denser vector spaces.

One of the applications, in the text, allows us to reduce a very sparse vector of more than 50,000 one-hot encoded words, into a much more tractable vector of just 300 variables.



## Autoencoder

Autoencoders learn a latent representation in an unsupervised way .

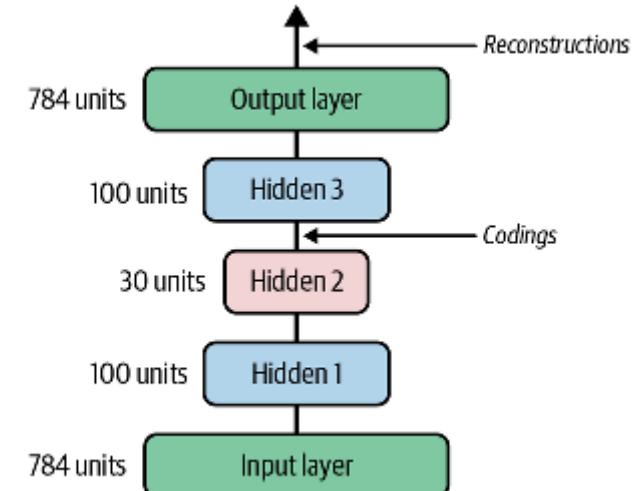
This method of compressing and decompressing information is very useful not only for reducing dimensionality but also for

- Identify anomalous data
- Rebuild missing patterns
- Generate new data

# Autoencoder

We have seen that an autoencoder is defined by an **encoder** and a **decoder**.

The encoder compresses preserving the maximum amount of information, the decoder decompresses it trying to reconstruct it in its original form. The cost function is simply the difference between input and output.



```
stacked_encoder = tf.keras.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(30, activation="relu"),
])

stacked_decoder = tf.keras.Sequential([
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(28 * 28),
    tf.keras.layers.Reshape([28, 28])
])

stacked_ae = tf.keras.Sequential([stacked_encoder, stacked_decoder])

stacked_ae.compile(loss="mse", optimizer="adam")
```

## Autoencoder

The previous example sets up an autoencoder to compress images from the Fashion MNIST, a dataset of clothing images, from their original size of  $28 \times 28$  to a single vector of 30 values and then decompress them.

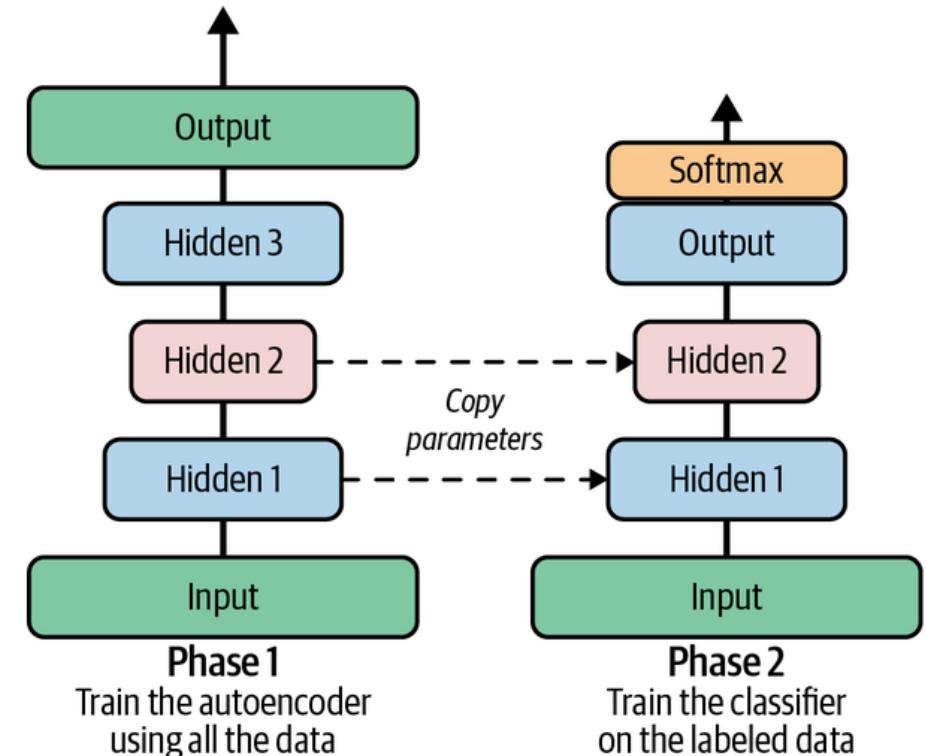
Visualizing the results it is clear that going from 784 variables to 30 we lose information, but the network still manages to preserve the main information.



## Pre-train with autoencoder

One of the main advantages of autoencoders is to extract highly significant features in a totally unsupervised way.

Thanks to this feature we can build models even with little labeled data and much unlabeled data available. In fact, we can train our encoder and decoder, then remove the second one and attach a classification head with much fewer parameters.

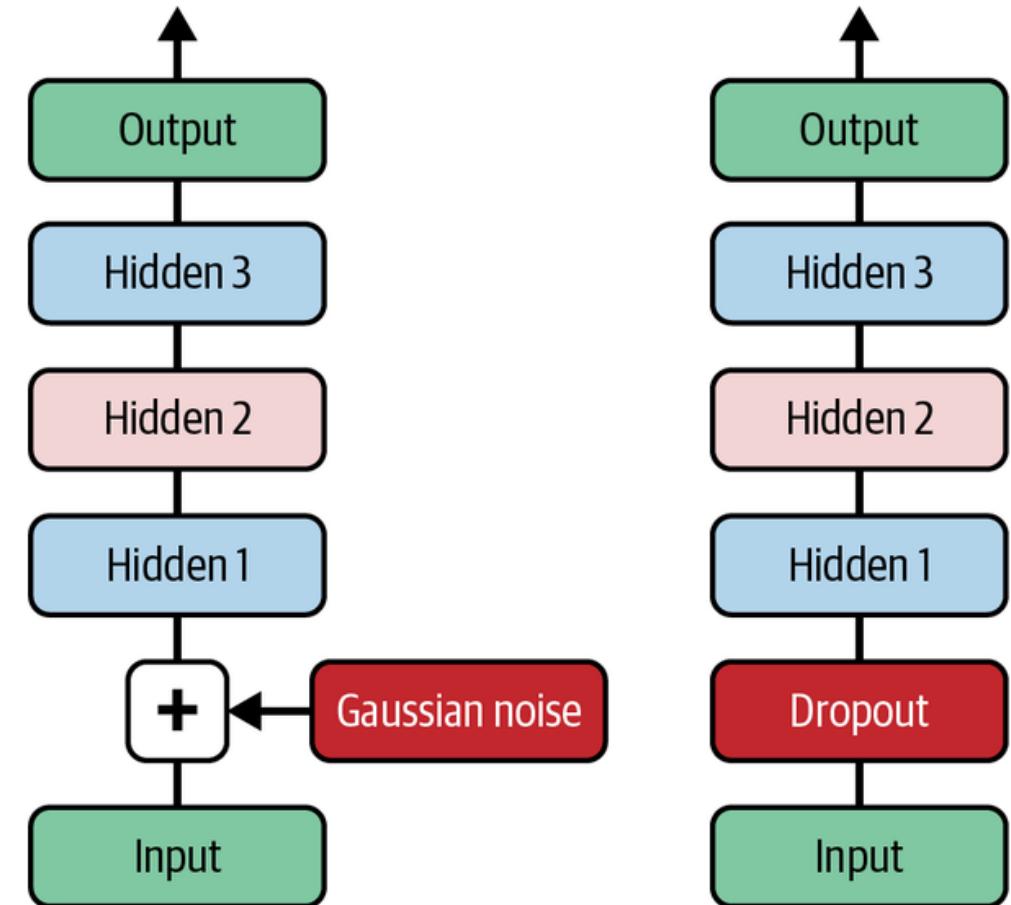


## Denoising autoencoder

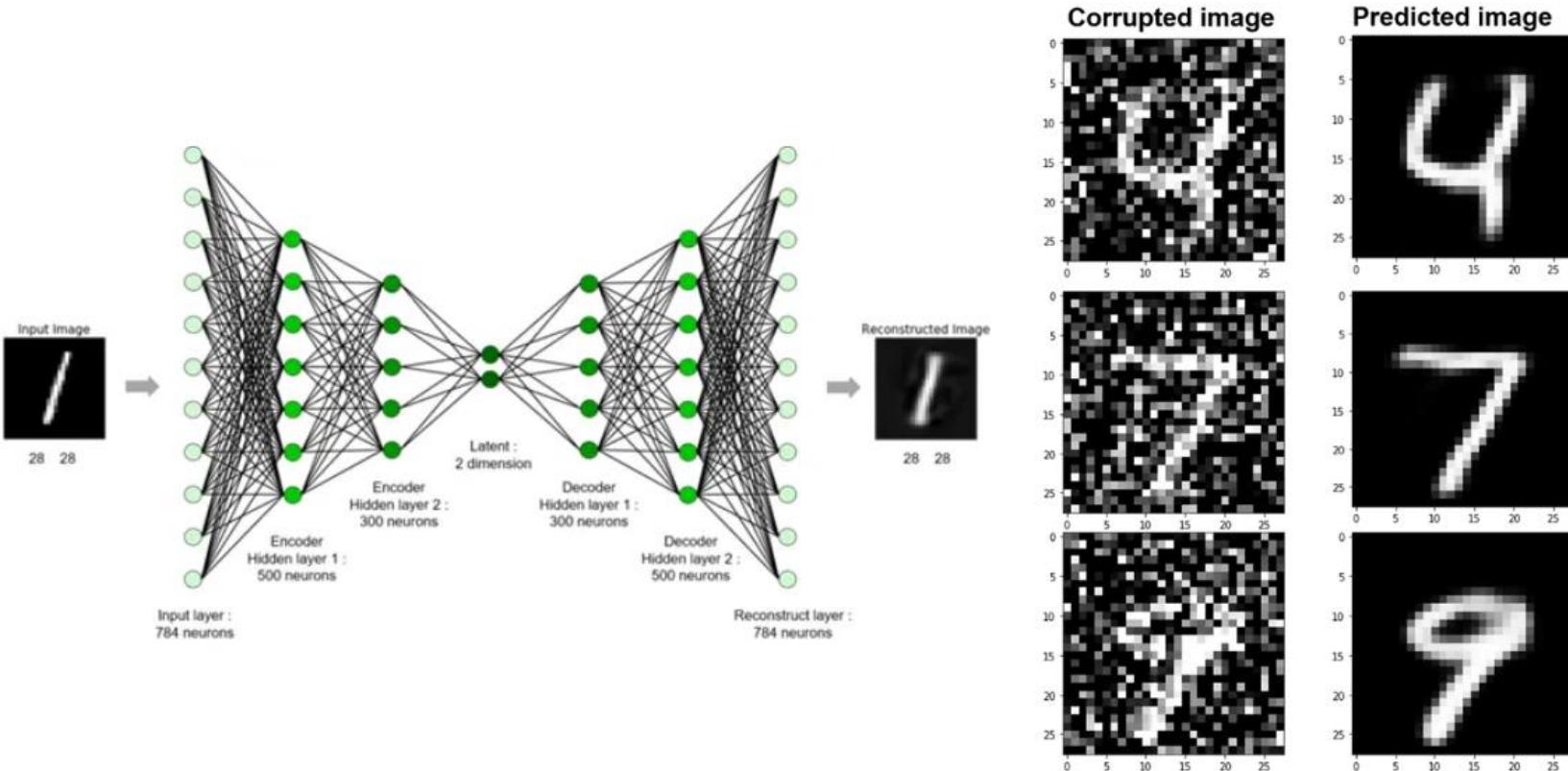
Another approach to make the autoencoder learn useful features is to add noise or add dropout to the input.

In this way our input is ruined and must be reconstructed from the other features by combining them.

In addition to being very useful in general as pre-training, we can also use this technique to clean potentially dirty data within the dataset or as a real task.



# Denoising autoencoder

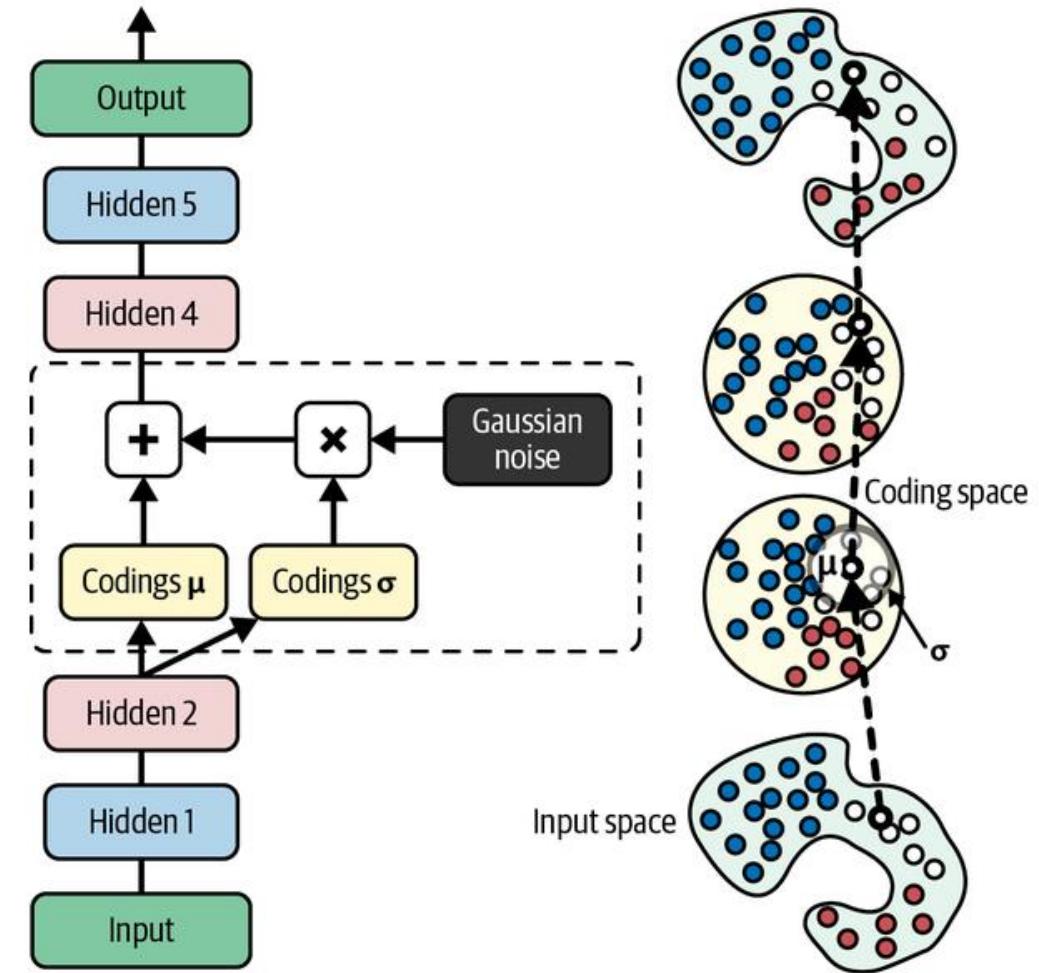


## Variations autoencoder

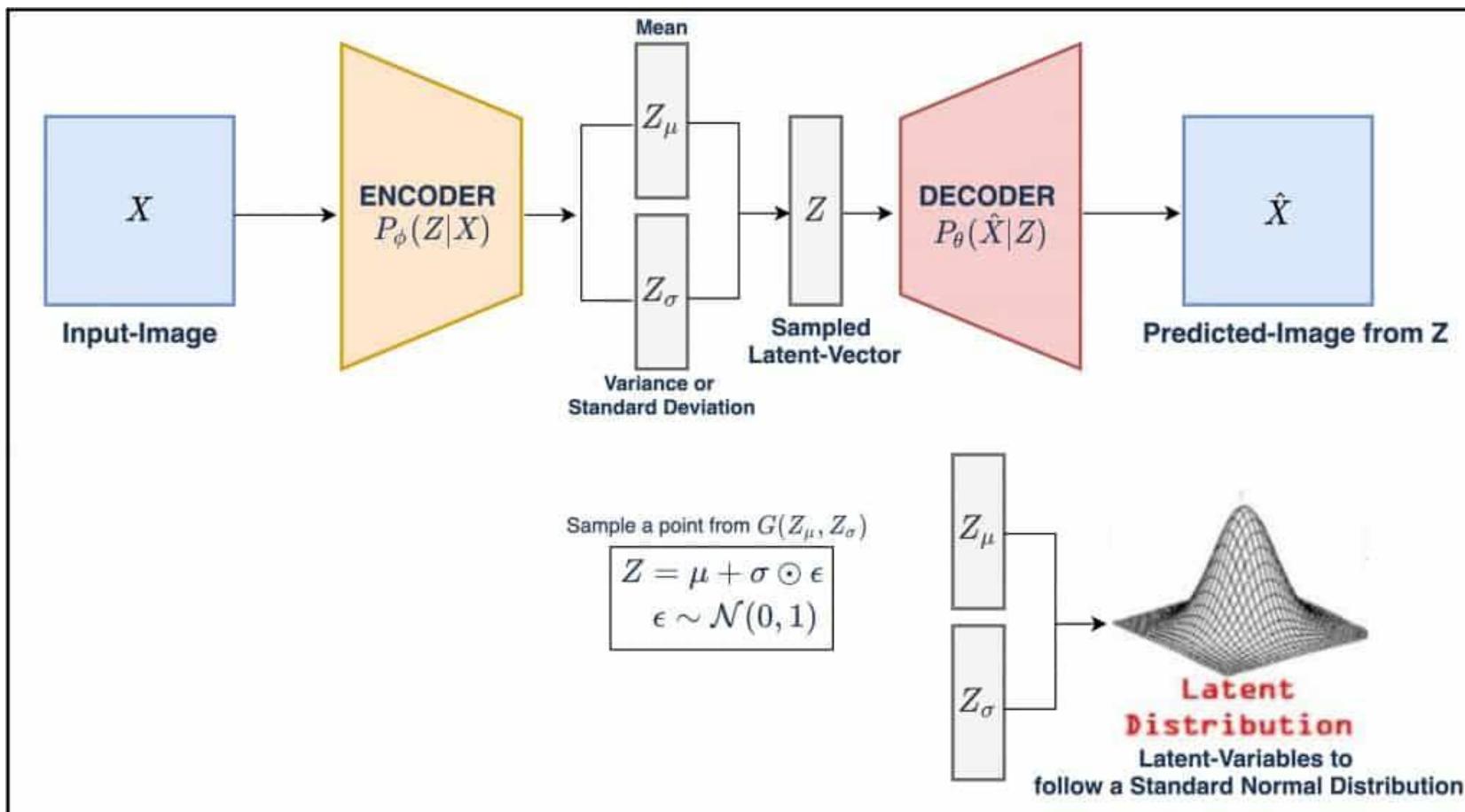
autoencoders called variational was invented .

They are first and foremost probabilistic objects, both in training and inference, and are generative, in the sense that they can produce new outputs similar to those on which they were trained.

Our encoder estimates a mean and variance for each position of the encoded vector. These are used to sample a new example (the result of the sum of Gaussian noise ) which will then be decoded by the decoder.



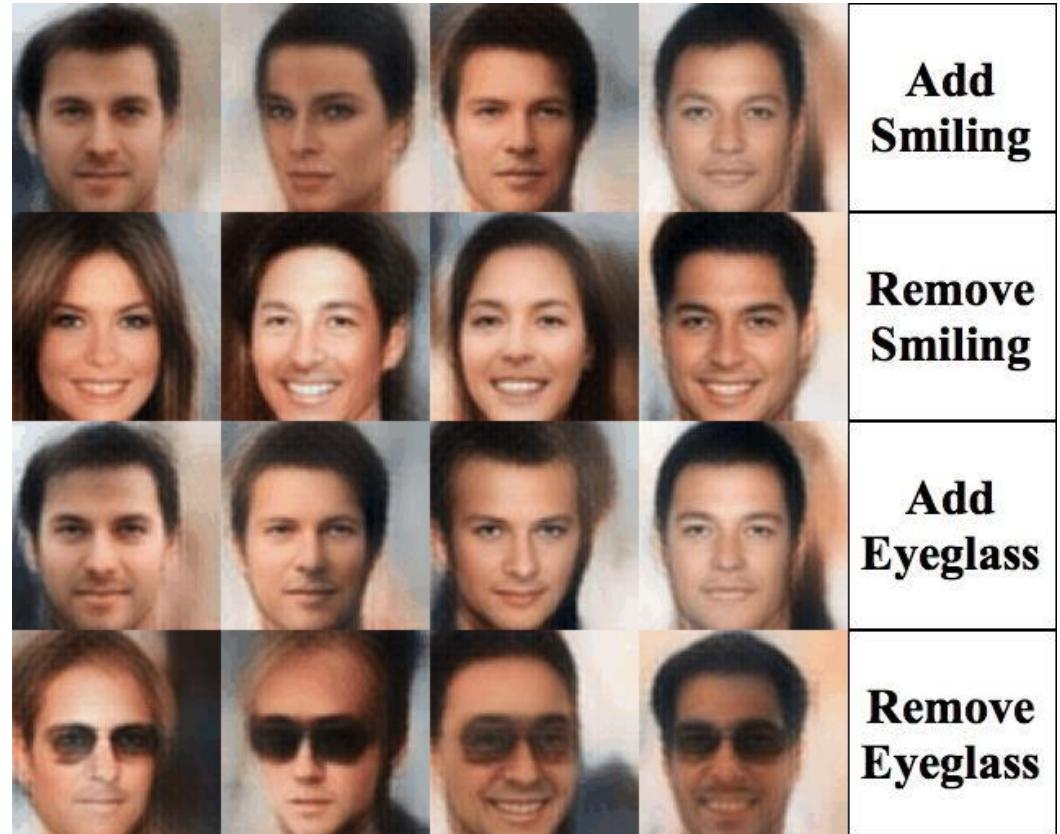
## Variations autoencoder



## Variations autoencoder

Once the VAE is trained, our latent space turns out to be entirely sampleable, so we can go and code an example to take a starting point and then start modifying the vector to see how it changes.

By appropriately setting certain values in training we can also associate some characteristics more or less explicitly with parts of our embedding .



## Variations autoencoder

Although it allows us to effectively generate new examples, the autoencoder still uses compression and therefore a significant loss of detail.

In image generation tasks, another approach took over due to the quality of details it was able to generate.





# **Generative beyond VAE**

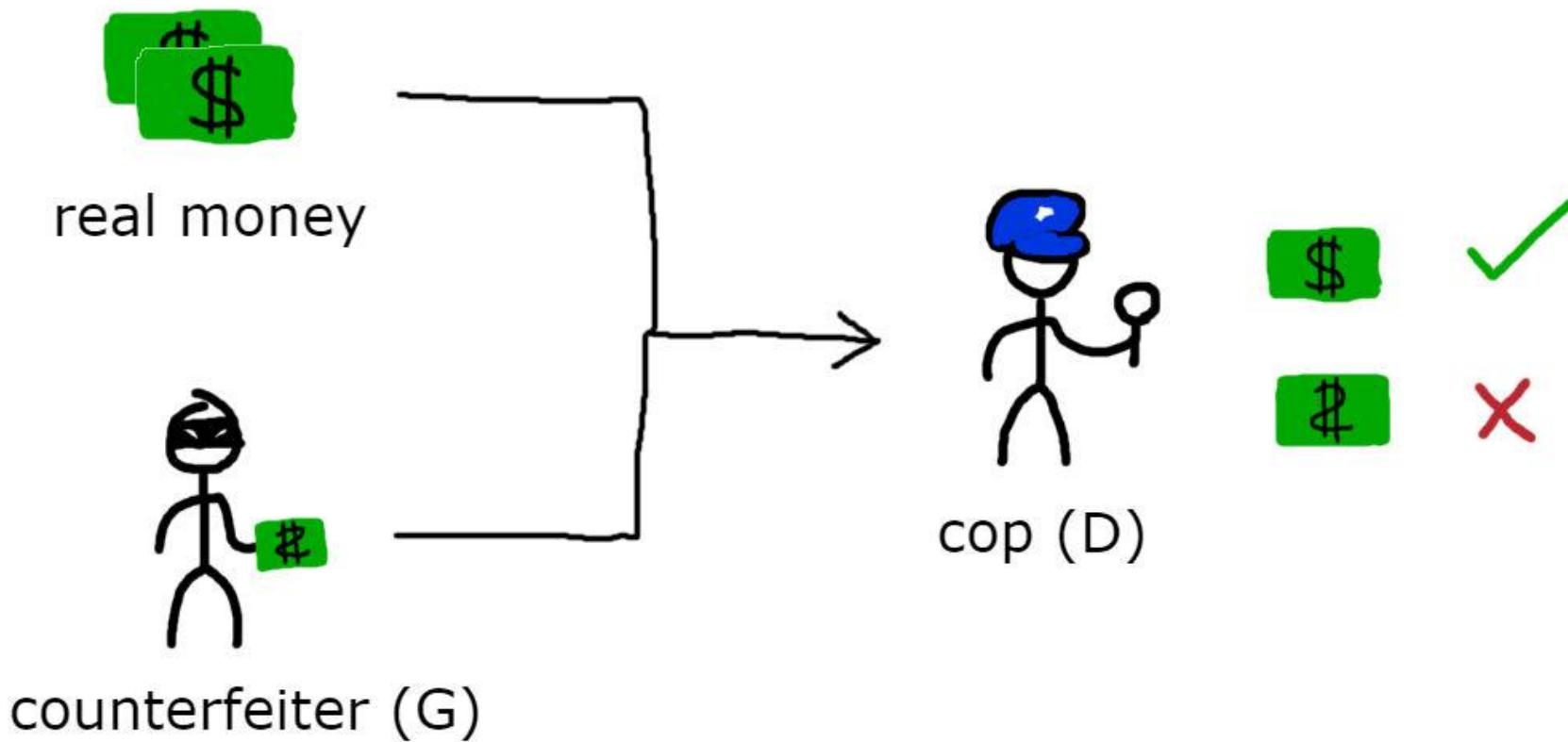
Generative **Adversarial Networks** are *generative networks* for creating new plausible data.

They consist of two opposing goals

- A **generator**, whose task is to forge new plausible examples
- A **discriminator**, which must discriminate whether the given example is true or generated

The generator tries to cheat the discriminator while the second always tries to discover whether the data provided to it is true or not.

## GAN

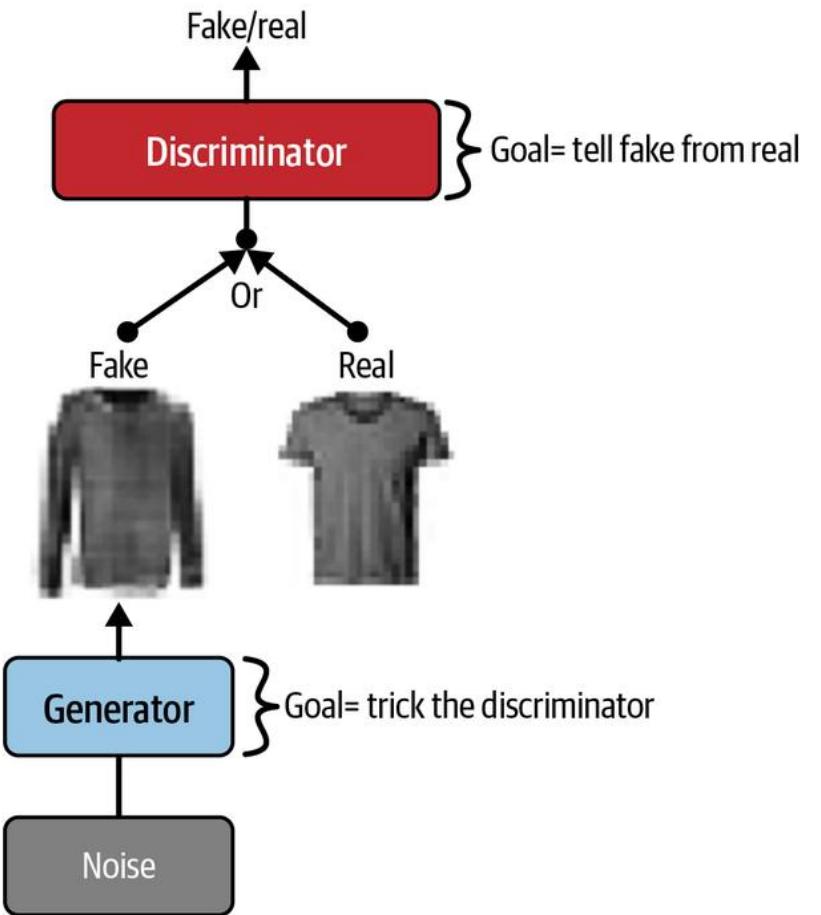


# GAN

The generator starts with its generation from a random noise vector.

This same vector is equivalent, once trained, to the VAE embedding that we can use to generate new data.

GANs are more difficult to train than other models because we will have two networks to train together and whose learning we are trying to balance.



# GAN

GANs don't have a linear training cycle like other models, so we have to write the optimization cycle ourselves.

Specifically, we will start training the discriminator by generating a new example of the noise generator and adding it to the training batch.

Next we will train the generator using the entire sequence to simplify the code.

```
codings_size = 30

Dense = tf.keras.layers.Dense
generator = tf.keras.Sequential([
    Dense(100, activation="relu", kernel_initializer="he_normal"),
    Dense(150, activation="relu", kernel_initializer="he_normal"),
    Dense(28 * 28, activation="sigmoid"),
    tf.keras.layers.Reshape([28, 28])
])
discriminator = tf.keras.Sequential([
    tf.keras.layers.Flatten(),
    Dense(150, activation="relu", kernel_initializer="he_normal"),
    Dense(100, activation="relu", kernel_initializer="he_normal"),
    Dense(1, activation="sigmoid")
])
gan = tf.keras.Sequential([generator, discriminator])

def train_gan(gan, dataset, batch_size, codings_size, n_epochs):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        for X_batch in dataset:
            # phase 1 - training the discriminator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            generated_images = generator(noise)
            X_fake_and_real = tf.concat([generated_images, X_batch], axis=0)
            y1 = tf.constant([[0.]] * batch_size + [[1.]] * batch_size)
            discriminator.train_on_batch(X_fake_and_real, y1)
            # phase 2 - training the generator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            y2 = tf.constant([[1.]] * batch_size)
            gan.train_on_batch(noise, y2)

train_gan(gan, dataset, batch_size, codings_size, n_epochs=50)
```

## GAN

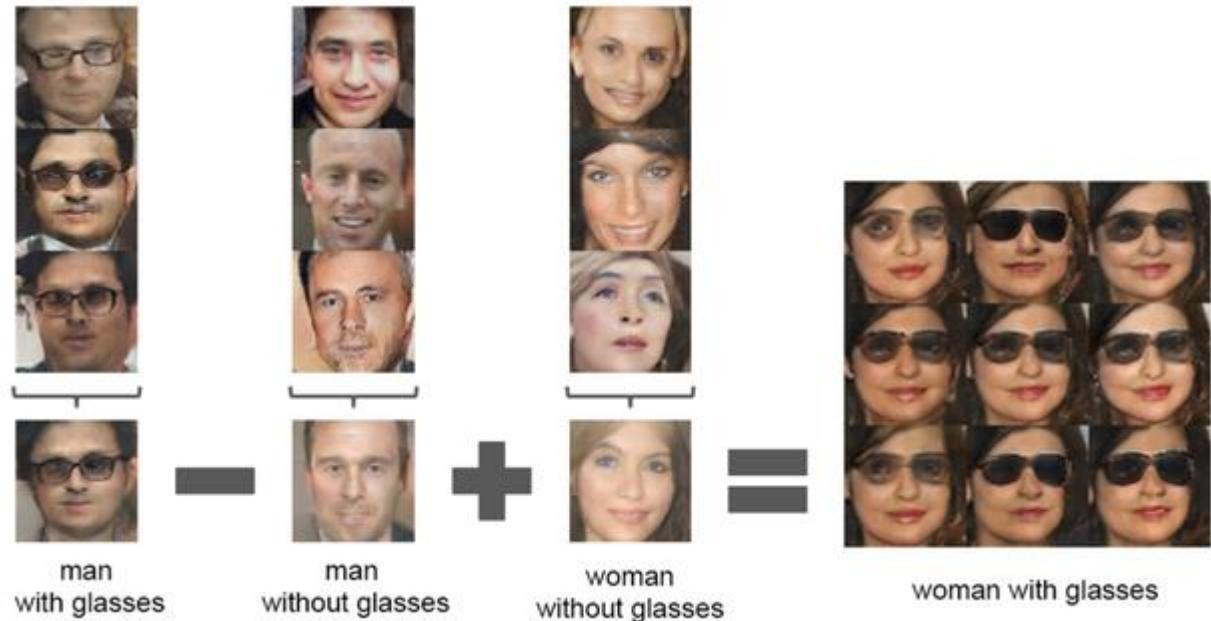
GANs are more difficult to train also because, for example, the generator could discover flaws in the discriminator and start generating only objects of a class in which the discriminator has more difficulty.

Furthermore, it is easy for training to be unstable by having two networks pushing against each other, resulting in a training collapse that causes all discriminator or generator weights to be zero.

# DCGAN

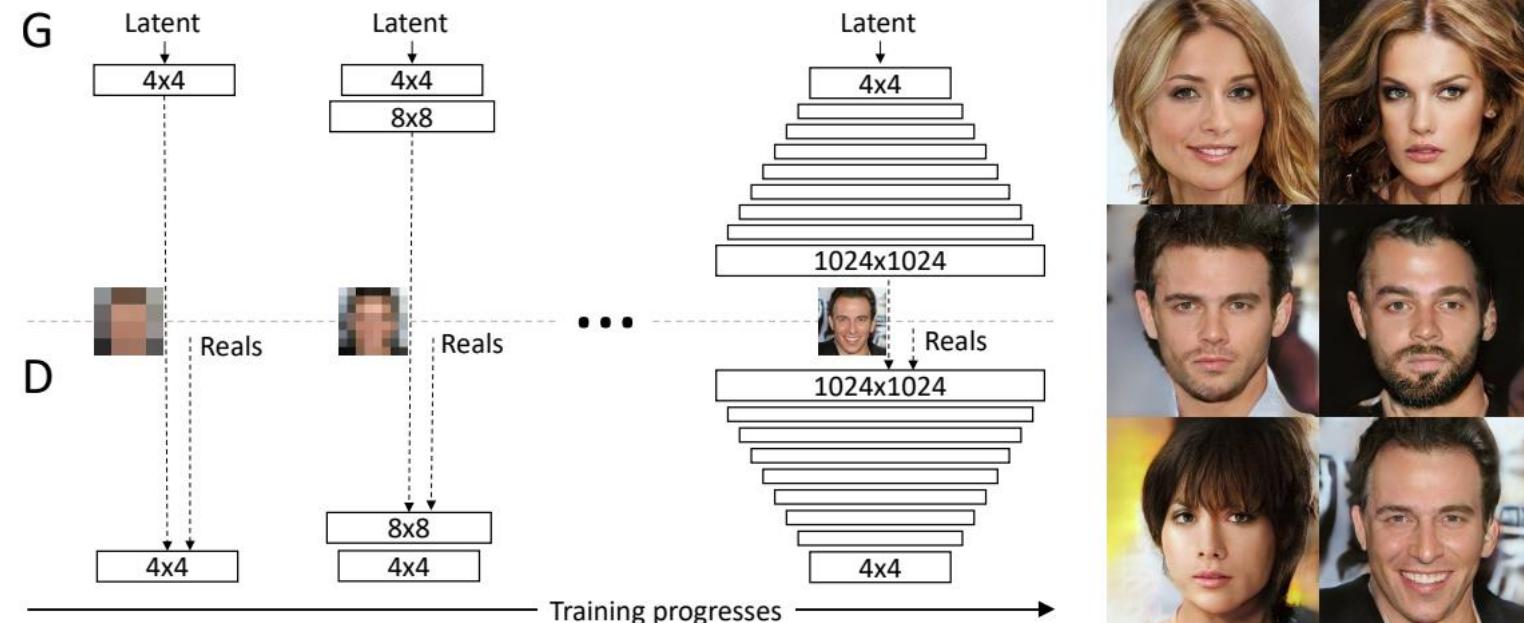
Since 2015, GANs built from deep convolutional networks , mainly used for image generation, are called Deep Convolutional GANs or DCGANs.

DCGANs are fully convolutional , do not use pooling but stride, and use a lot of batch normalization . They have paved the way for generating increasingly complex images and share with Word2Vec the projection of data into a space in which it is possible to perform mathematical operations between images (exploiting their embedding ).



# Progressive Growing GAN

The results are surprising but there was still ample room for improvement, in 2018 a team from Nvidia proposed to train GANs at small resolutions and grow with resolution by adding layers convolutional both discriminator and generator. This makes it extremely easier to train GANs with a greater level of detail. All weights are left trainable however.

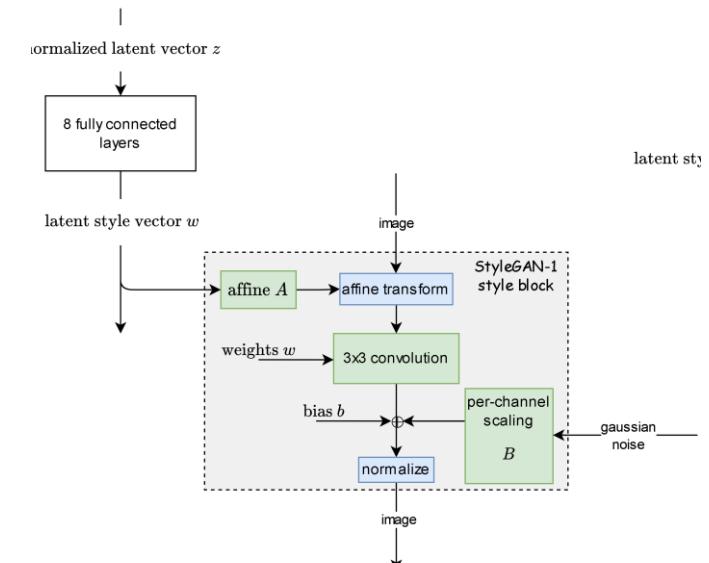
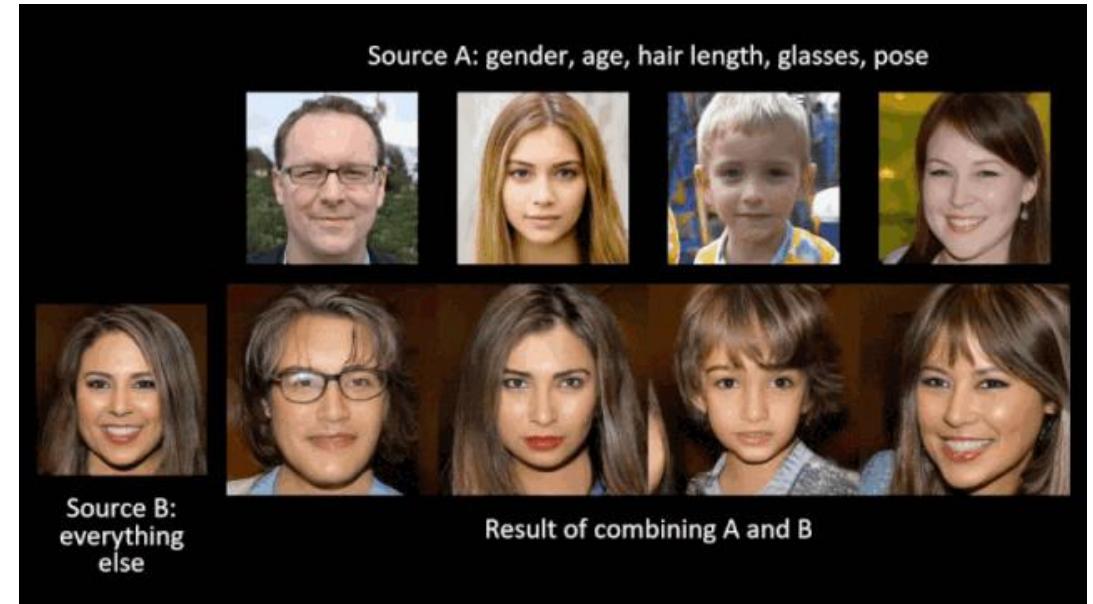


# StyleGAN

Also in 2018 Nvidia further advances the possibilities of GANs by presenting the StyleGANs .

In StyleGAN the generator is divided between style, which is coded as " noise " vectors to be added to the image, and structure which is continuously disturbed by noise .

In this way the style is encoded at different levels and added to the image while the structure remains unchanged.

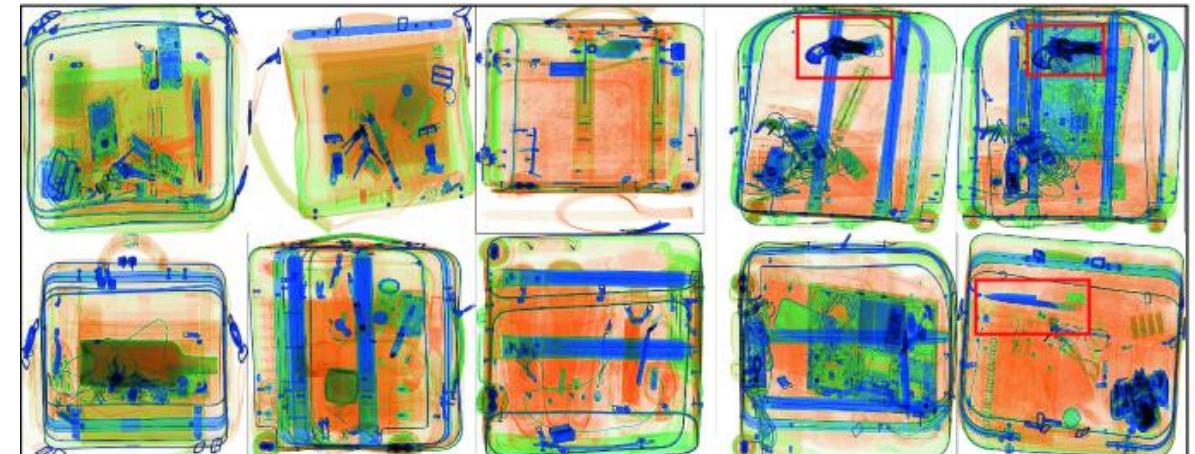
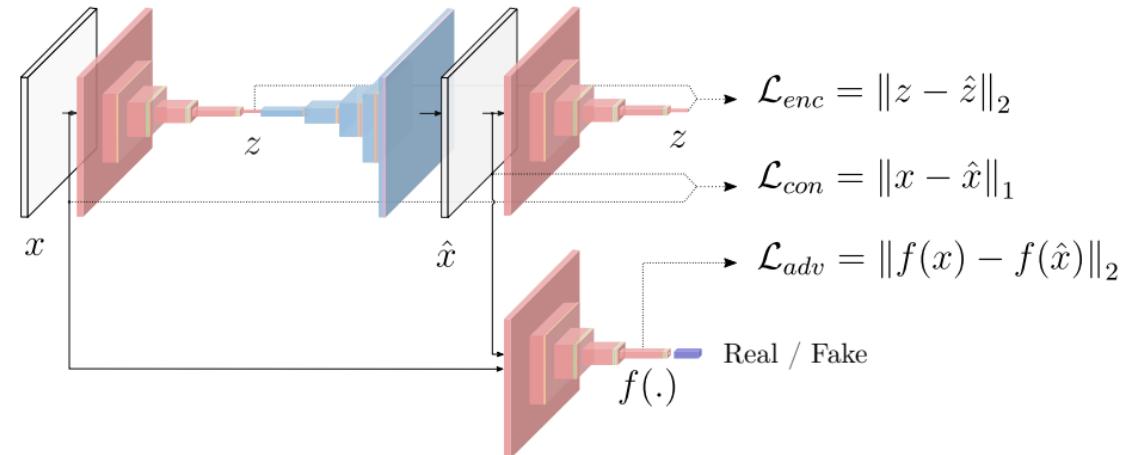


## GANomaly

As we have seen, VAE and GAN learn the structure of the dataset and to generate new plausible examples. We can also use them for the reverse process, i.e. understanding whether an example is plausible or anomalous.

One of the most interesting models is GANomaly , in which we try to combine the features of VAE and GAN to identify anomalous parts of an image.

We can exploit this approach when we have many good examples and few or almost no anomalous ones.



(a) Normal Data (X-ray Scans)

(b) Normal + Abnormal Data (X-ray Scans)

## In-depth analysis - Anomaly Detection

As we have already mentioned several times during the course, one of the problems that fall under Machine Learning is Anomaly Detection .

There are two broad categories

- **Outliers** detection , i.e. finding the dirty data within our dataset, i.e. the outliers
- **Novelty** detection , that is, once new data has been obtained, understand whether it is an outlier or not

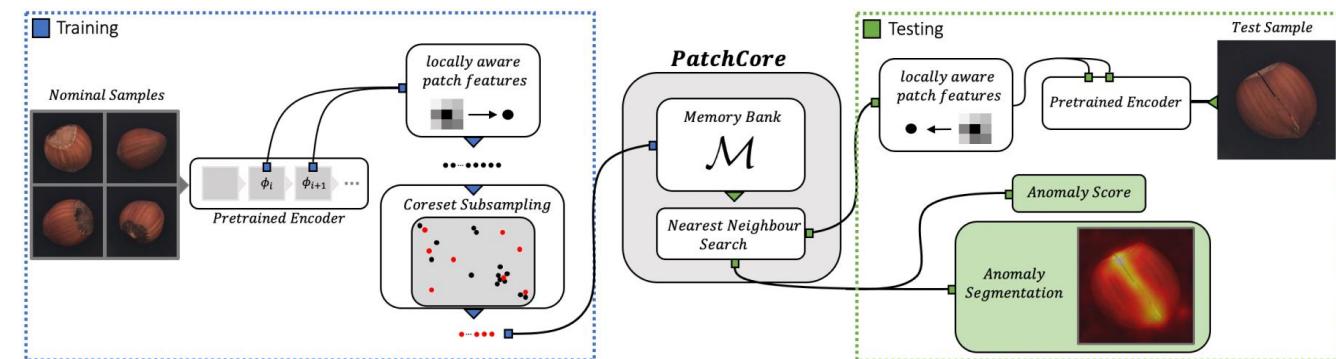
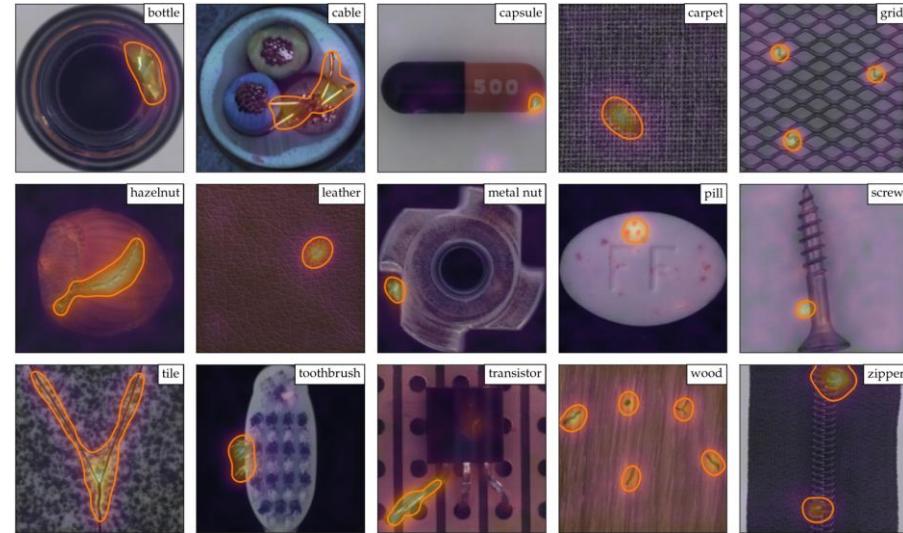
The former are often found by iteratively reviewing the dataset to evaluate the cases in which our models make mistakes, by visualizing the data or by applying different techniques.

# In-depth analysis - Anomaly Detection

We have seen, addressing SVMs, that one of the techniques is the OneClass SVM. Otherwise we can also use the encoder of our VAE to estimate whether or not our example is similar to the others available.

In image processing, however, from 2021 one of the most promising techniques is called **Patchcore**.

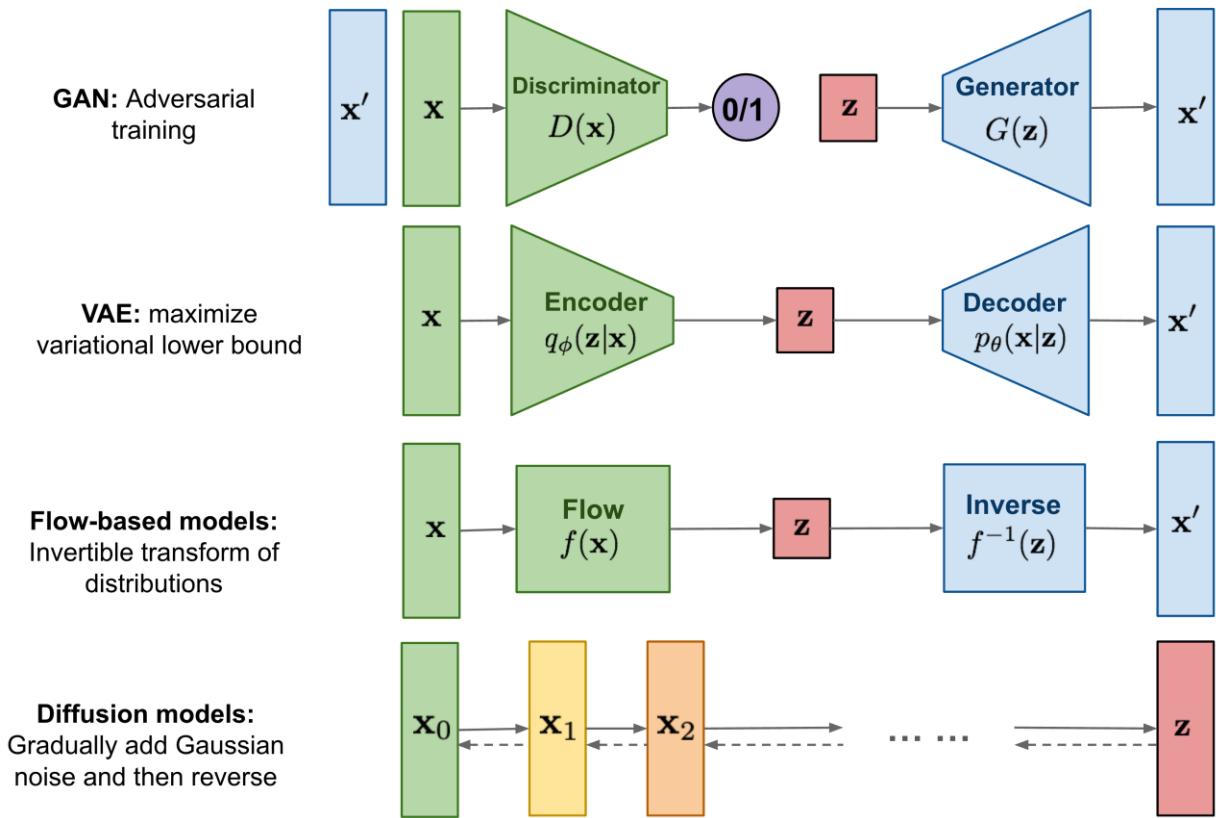
It is based on extracting a set of features from a backbone , placed in a **Memory Bank** , which are selected and compared using Nearest Neighbor with the new example to evaluate.



# Image Generation

There are several approaches to image generation.

We've already seen VAE and GANs, but in the last two years the best technique is the Diffusion Model.

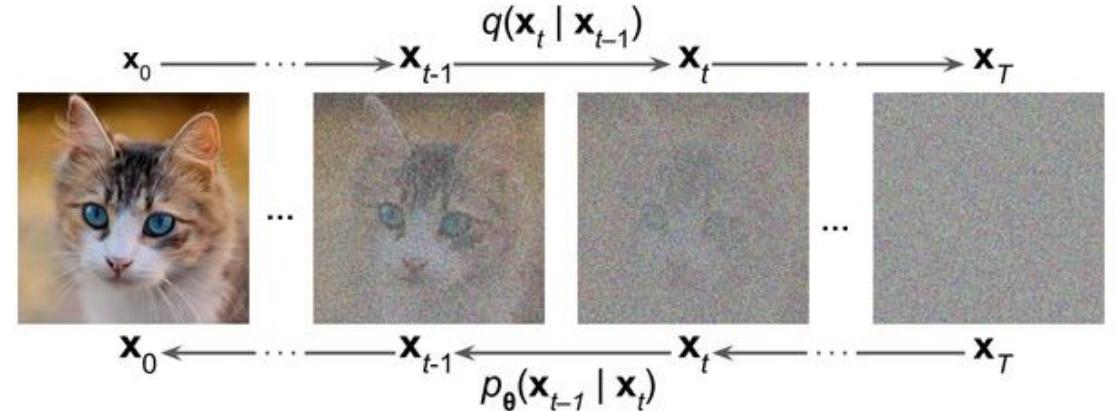


# Diffusion Models

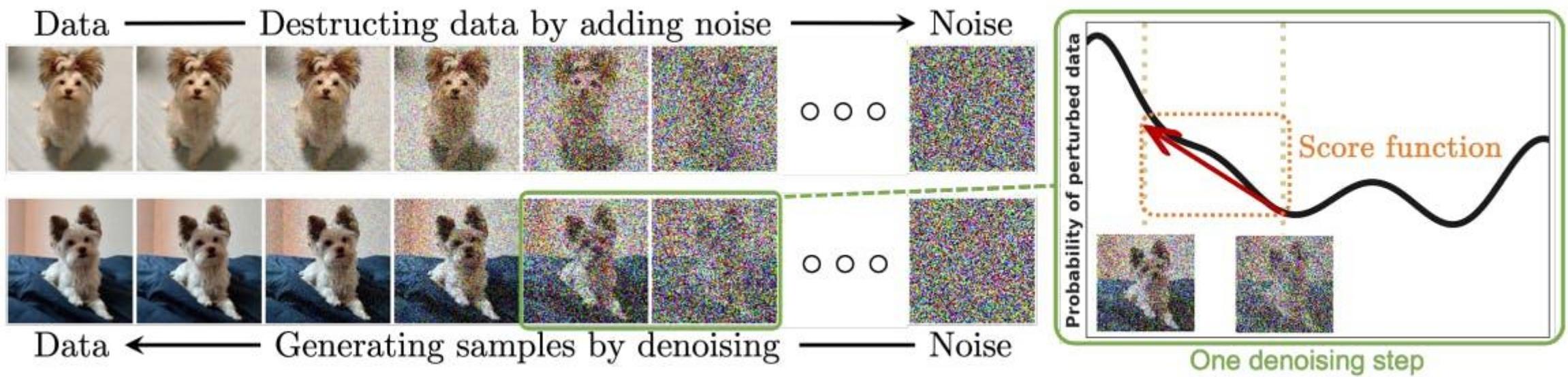
Diffusion Models are generative models that iteratively try to reconstruct an output by removing the noise present.

From 2015 onwards, and in particular from 2020 and 2021 they have gained great importance because they have beaten GANs in the quality of the results they are able to produce, with StableDiffusion as the main public model.

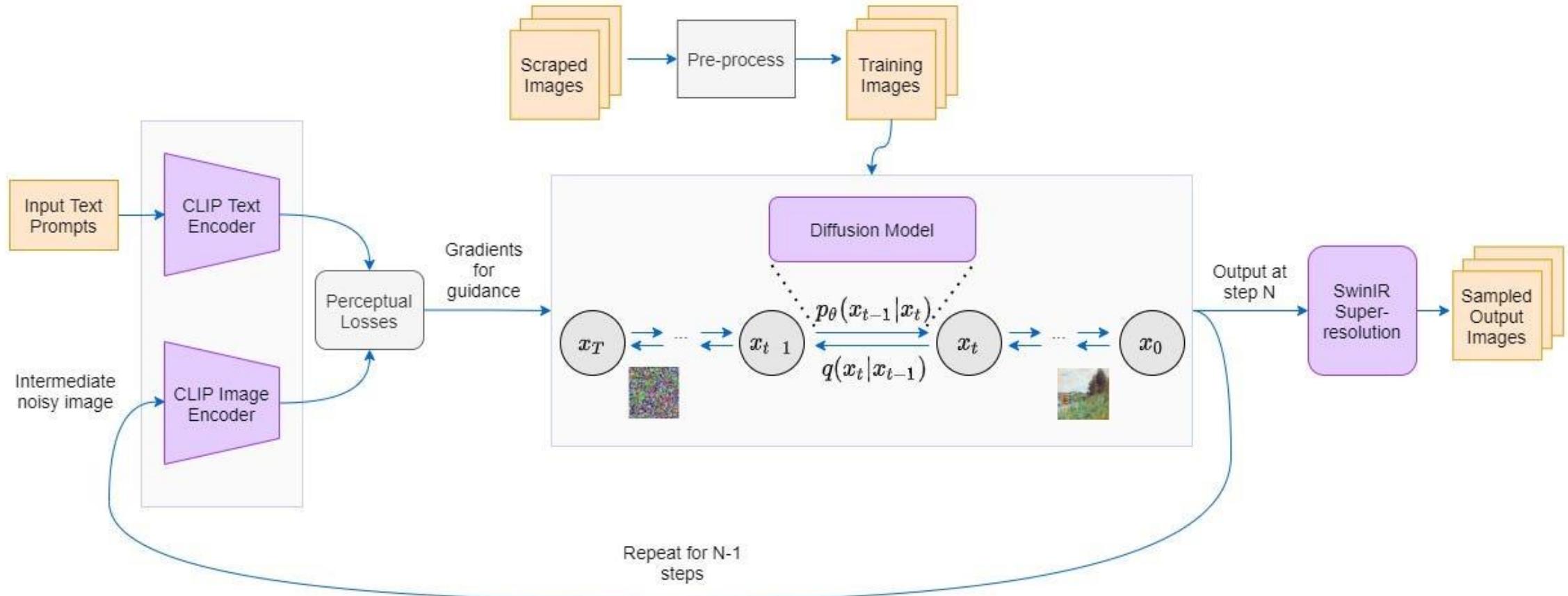
The training is done by providing an image with more and more noise at each instant of time , while the model must continuously try to subtract this noise to preserve the information within it.



# Diffusion Models



# Alignment





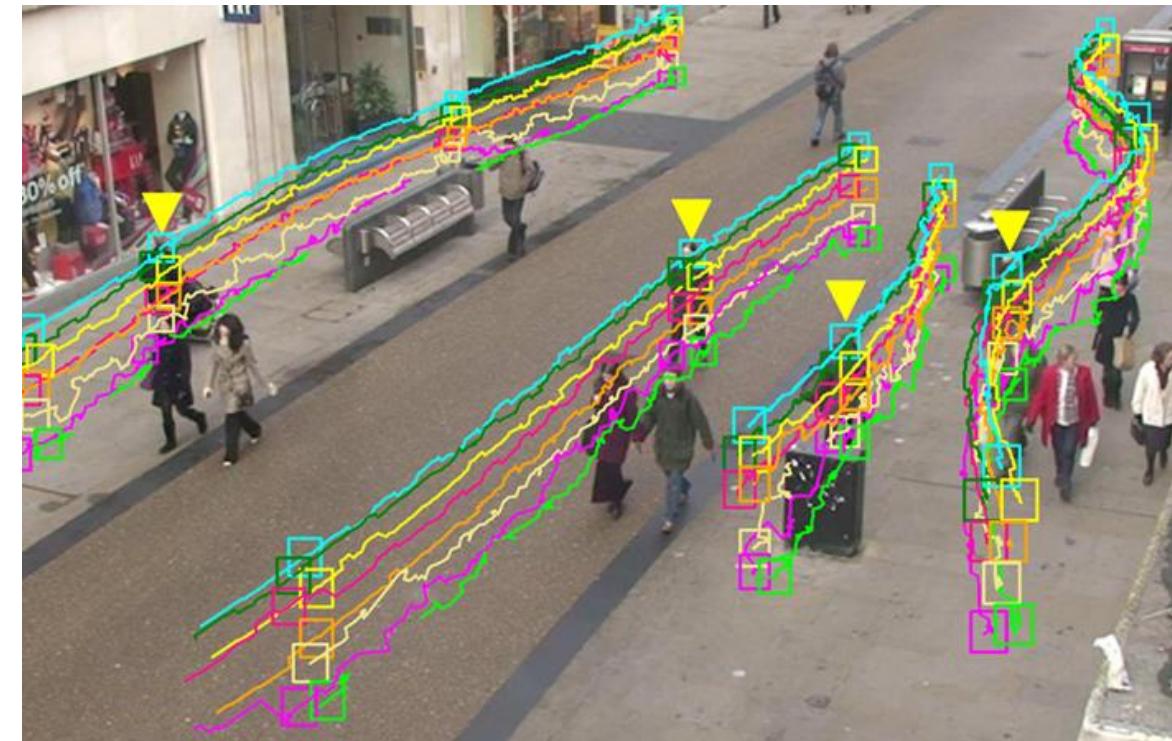
# **Other problems**

Daniele Gamba

2022/2023

# Tracking

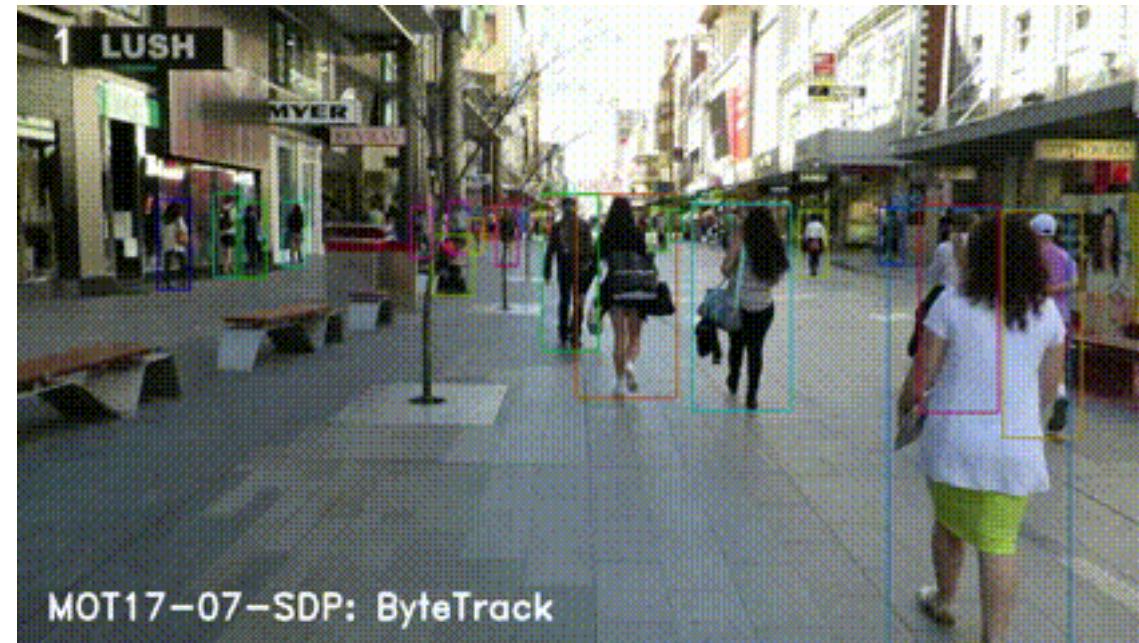
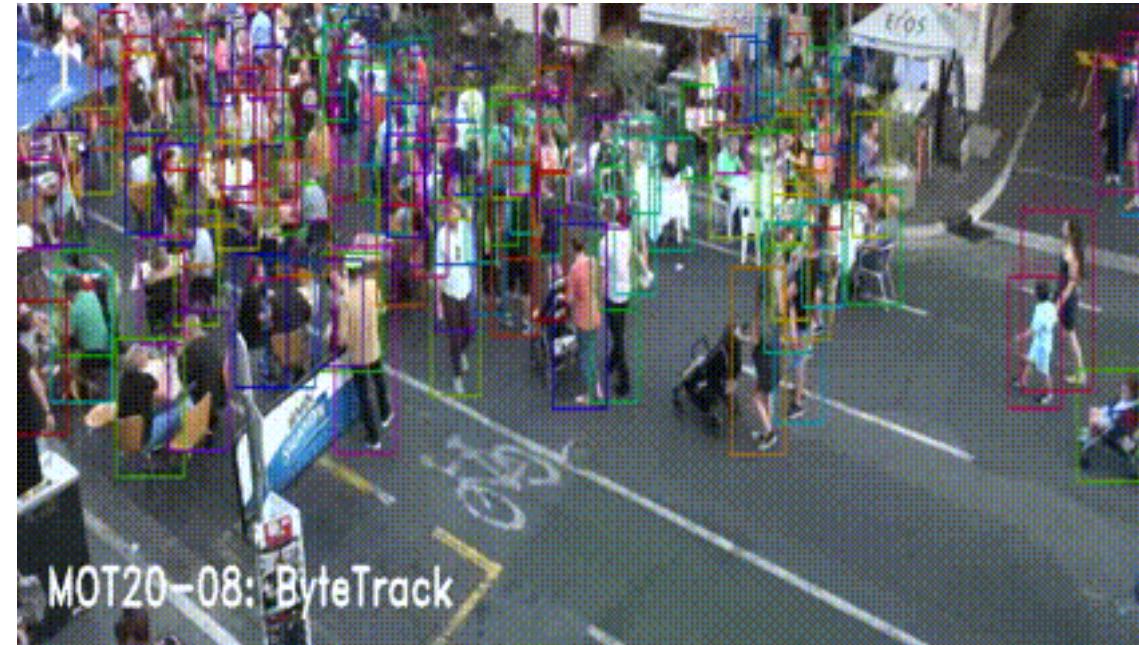
What if we don't have a single frame but a sequence and we want to **track** out Object of Interest?



## Bounding boxes Tracking

Tracing the bounding boxes means making a detection every frame, finding the bbs of the same class and re-assigning them to the closest object.

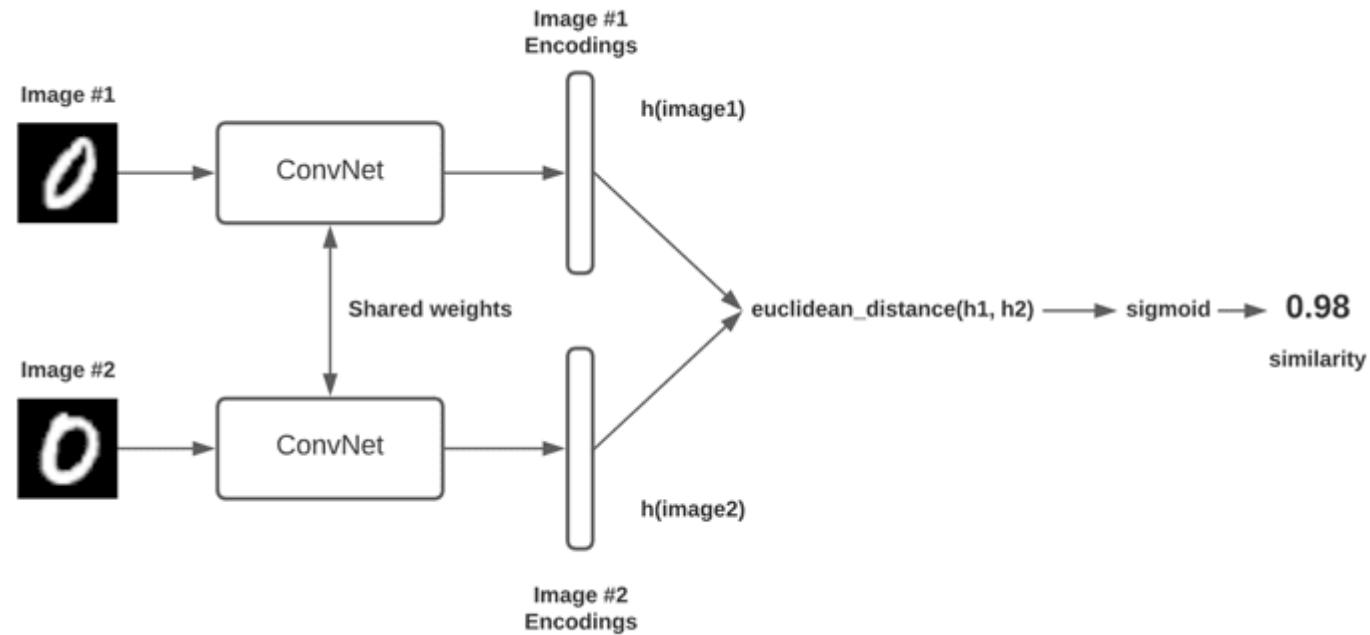
One of these techniques is **ByteTrack**, an algorithm that keeps all the detections made by a detection network and takes care of weighing them to perform Multi-Object Tracking (MOT).



# Optical Tracking

Trackers that are based on optical features, however, do not look at bounding boxes but only at a measure of similarity between subsequent frames of the same area.

The objects to be tracked must be selected and then they can continue to be tracked in the frames.



# Optical Tracking - SiamDW



— SiamFC

— Ours

## Track Anything

Track Anything, on the other hand, is a tracking algorithm built on Segment Anything.

By being able to segment objects you can then do fun in-paintings such as completely erasing objects in a video.



## 3D Reconstruction

Another computer vision task is 3D reconstruction given 1 or more images. For some years there has been a technique called NERF (Neural Radiance Field) and its evolutions (Instant-NGP - Neurangelo) which have reached very high levels of reliability with just a few shots.



## 3D Reconstruction



Grazie



# **16 – Reinforcement Learning**

Daniele Gamba

2022/2023

Reinforcement learning is another way that models can learn

It is based on the concept of an *agent* who, through *observations*, decides to perform *actions* in an *environment* to maximize his *reward*.

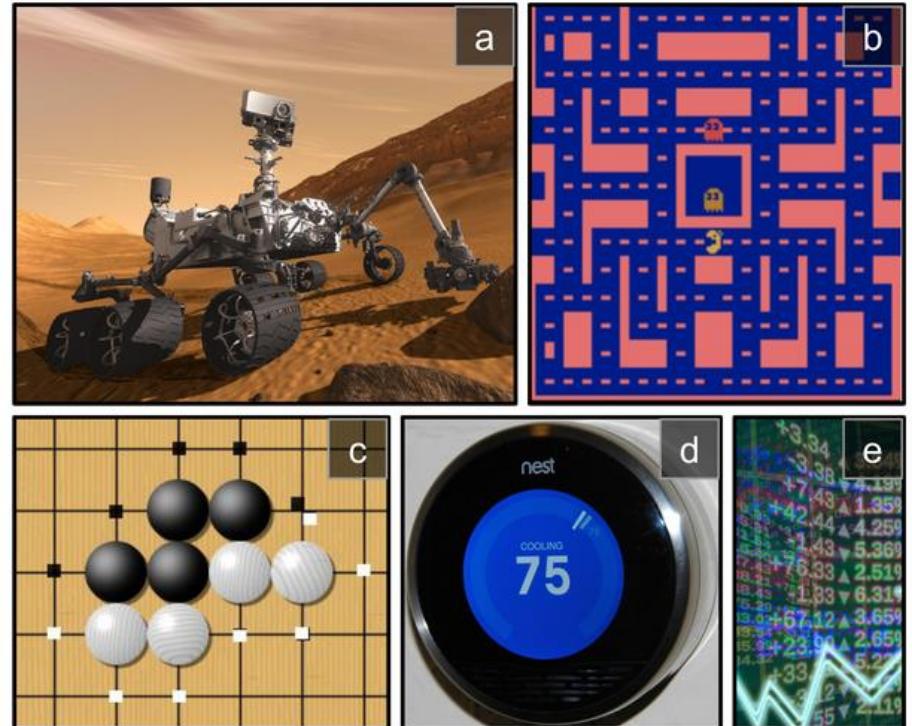
Having to interact with the environment, the RL has had great success in gaming, achieving one success after another and managing to beat champions from Atari, Go, Dota, Starcraft, etc.



# RL examples

Some examples of RL

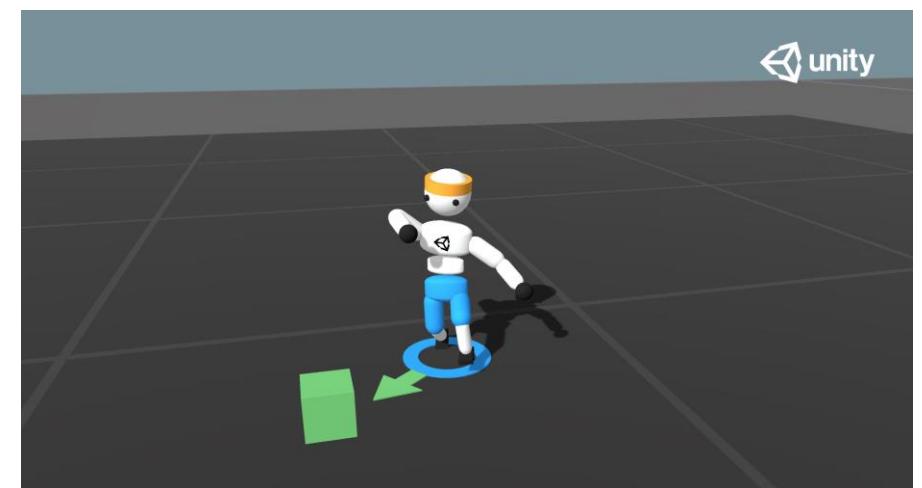
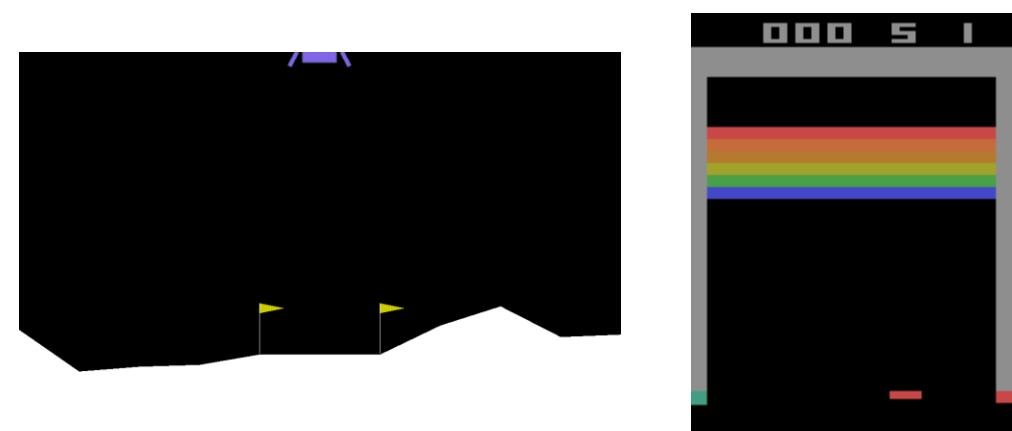
- A robot controlled by an agent, which observes the world through cameras and sensors, performs actions by controlling the motors and receives a reward by approaching its goal, while being penalized if it moves away
- An agent plays PacMan , the environment is the Atari simulation and the actions are the possible keys to press, the observations are screenshots of the game and the reward is the score achieved
- A smart thermostat receives a positive reward when it approaches the target temperature, saving energy, while it is penalized every time the human has to manually change the temperature
- A trader, ..



## OpenAI / Farama Gym

Gymnasium (formerly OpenAI Gym) is a collection of environments in which you can train different agents.

It has a Python API to be easily integrated with agent code and repeatedly execute actions and training in different environments.

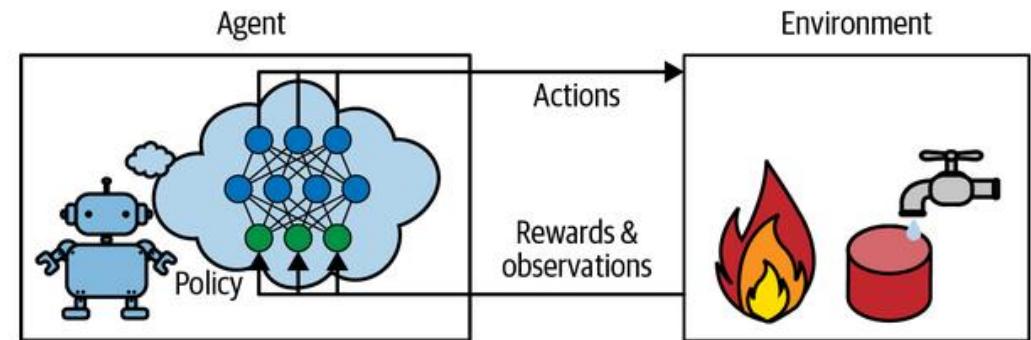


## Policy search

The algorithm that the agent uses to determine its actions is its *policy*.

The policy could be an NN or another algorithm, each policy can have one or more *parameters* that are gradually adjusted to find the optimal policy in a process called *policy search*.

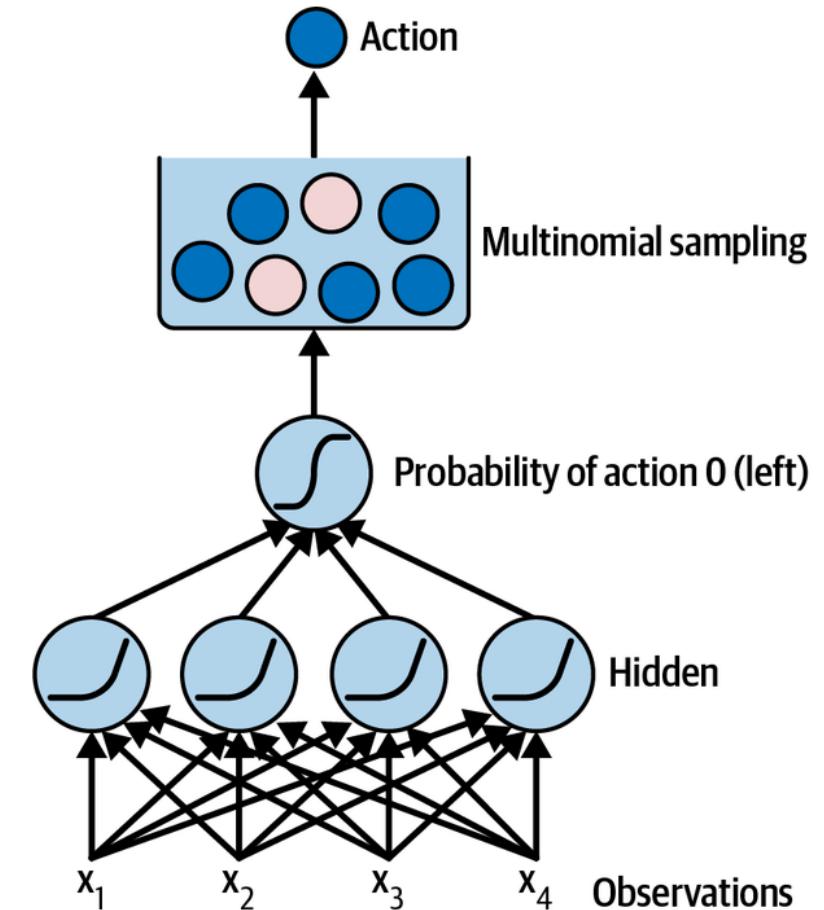
We could choose to randomize the next action, or use genetic algorithms to randomize only part of the policy parameters.



## Exploration / Exploitation

If we decided to face an environment, if we directly optimized our model, we would tend to exploit the current policy by always choosing the most probable action without ever asking ourselves the problem that other more efficient ones might exist.

, a search for better solutions, exploration, must always be balanced with optimization of those that we consider to be more performing, **exploitation** .



## Long term scores

Once our network has been defined, we must choose at every moment what is the best move to maximize our rewards .

If we knew at every moment the best move to make, everything would be a simple supervised learning problem, all we would have to do was train the network to take certain outputs based on certain inputs.

Unfortunately we don't know at any moment what the best move is but we necessarily have to get to the end of the game to evaluate our score. Hence the need to choose policies that allow us to make decisions without having visibility of the immediate result.

We must therefore assign to each move a probability that will lead us to maximize the reward in the future .

## Long term scores

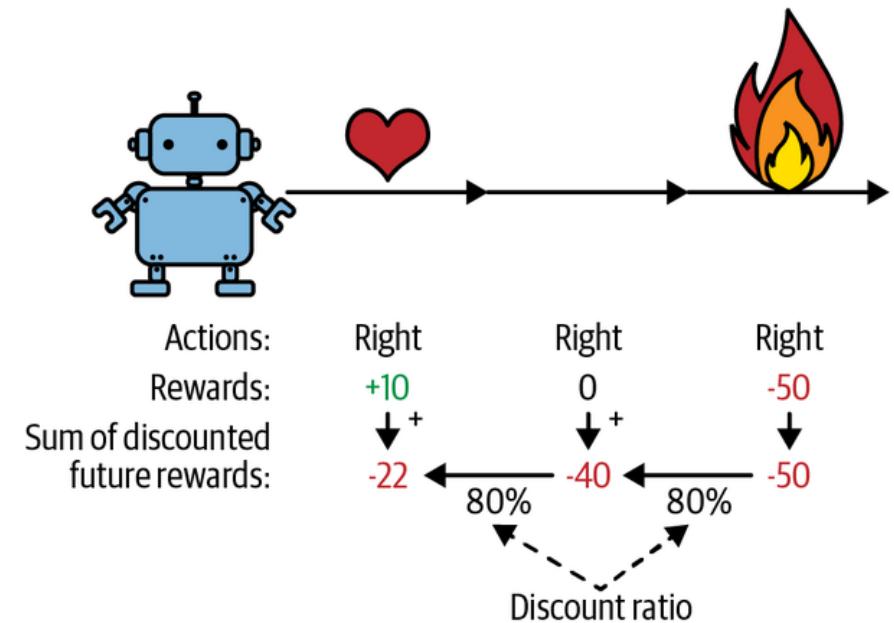
One problem is assigning a score to each move, in fact once we have reached our reward we are not necessarily able to assign to each of them whether it had a positive or negative impact.

This problem is known as *credit assignment problem* .

## Discount factor

One way of assigning the reward to the different actions is to weigh the future reward backwards using a *discount factor* range that discounts the impact of the different actions over time.

The closer the discount factor is to zero, the more only immediate decisions will have an impact; the closer it is to 1, the more even more distant decisions will contribute to achieving the objective.



## Policy Gradients

A more interesting idea is to evaluate the gradient of the policy with respect to the reward to maximize our gain.

One way is described by the REINFORCE algorithm

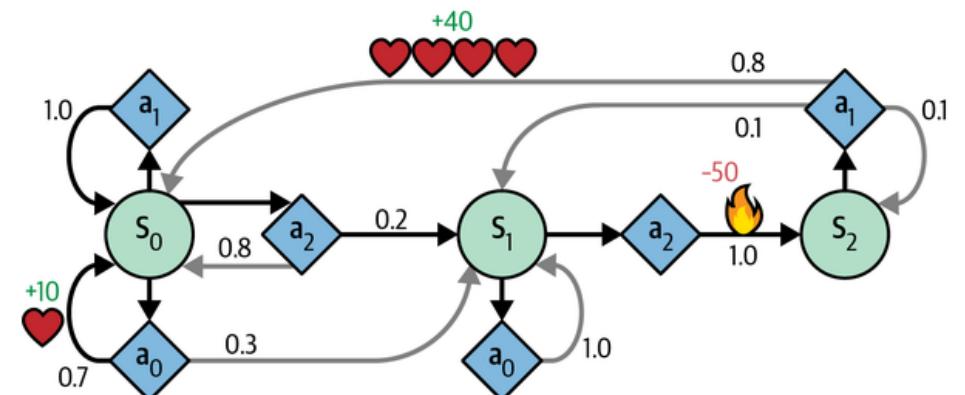
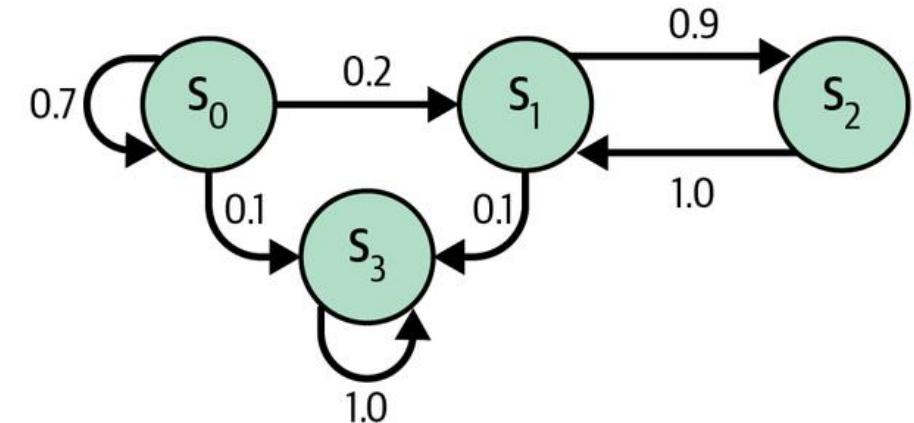
- First we play against the environment many times and calculate the gradients that make the choices made more likely without applying them
- We then evaluate the impact of the actions backwards by discounting the scores obtained
- If an action was positive then we will keep the gradient positive, otherwise we will apply the opposite of the gradient
- We calculate the average of all the gradients and do an optimization step

## Markov Decision Process

When dealing with RL we cannot fail to mention Markov Decisions Process , or MDP.

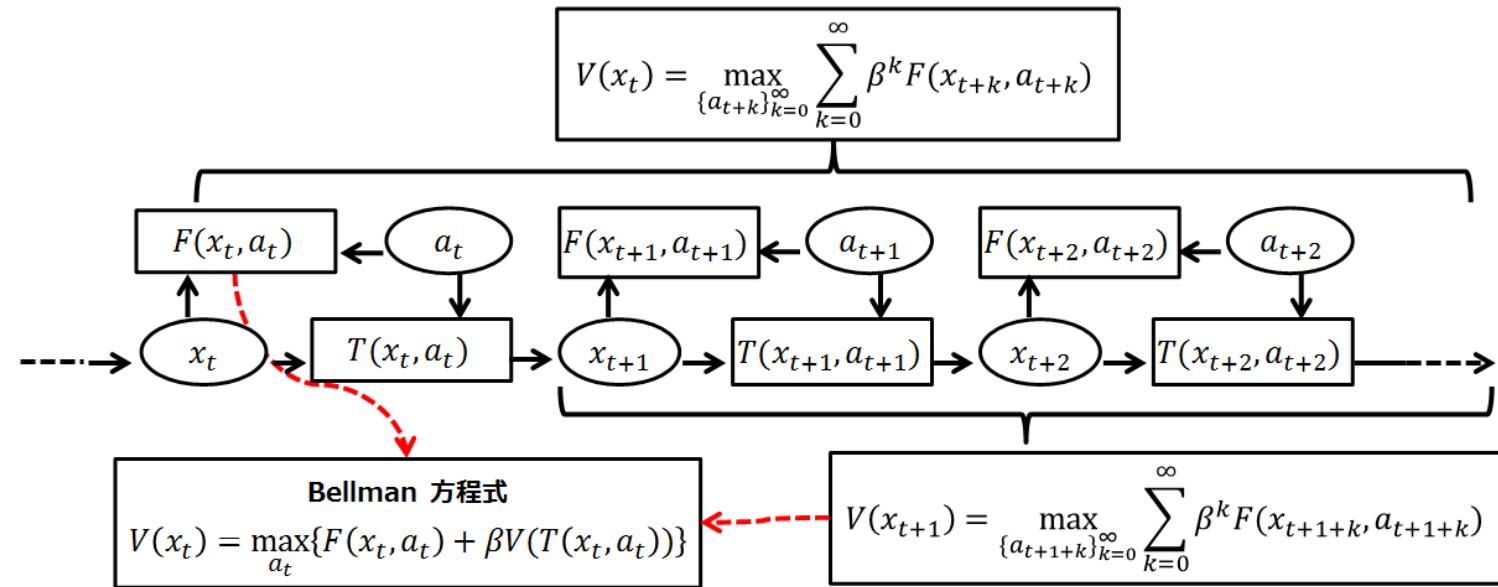
A Markov Chain is a stable, memoryless stochastic process in which it transitions from one state to another with a certain probability.

An MDP, unlike MC, is a process in which in each state I can choose to carry out actions, the outcome of which is defined by probabilities.



## Bellman optimality

Bellman demonstrated the value of each state in case the agent acts optimally through *Bellman optimality equation* and an iterative algorithm that estimates the optimal state, *Value iteration algorithm*, which you can find below for information.



## Q- value

However, the two previous formulas do not give us an indication of what the optimal policy is for the agent.

For this reason, Bellman introduced Quality- values as the output of an algorithm to estimate the optimal state-action choice.

The Q- value is calculated as the discounted sum of all future reward values associated with reaching state  $s$  and executing action  $a$ .

Q-value iteration would use the previous iteration of the Q-value on the right hand side of the equation

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \left( R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right)$$

to update the Q value estimate of the current step. Hence, the Q value update for  $k^{\text{th}}$  step would look like:

$$Q_{k+1}^*(s, a) = \sum_{s'} T(s, a, s') (R(s, a, s') + \gamma \max_{a'} Q_k^*(s', a')) .$$

## Temporal Differences

Ballman's traditional approach is knowledge of the MDP and transition probabilities. This is clearly not true in most cases where the agent has only partial knowledge of its environment's MDP.

The algorithm called Temporal Differences , or TD, takes into account the fact that he does not know the environment he is in and consequently at the beginning of learning he will tend to do *exploration* to estimate states and transition probabilities before estimating reward values .

## Q-learning

Similarly to TD, Q-learning is an algorithm that will watch an agent play almost randomly against the environment to try to estimate Q-values.

When the estimate gets close to the ones actually received, at that point it will start to exploit the policy that maximizes the return in a greedy way.

Throughout the training phase, a greedy policy is often used  $\varepsilon$ , therefore with a small probability  $\varepsilon$  of not choosing the best action but of randomizing it to continue exploring.

## Deep Q-learning

So far we have hypothesized environments with a finite number of states, but if we even just think of a 2D video game, with randomly generated prizes and enemies, the number of "states" to explore would be enormous and the algorithm would take too long to estimate them all. Probabilities.

DeepMind in 2013 suggested instead using a deep neural network to estimate Q- values , i.e. a DQN.

We will optimize it by iteratively interacting with the environment, observing the rewards , discounting them for the actions and giving the value as a learning objective to the DQN.

## Other learning algorithms

Numerous other learning algorithms have been created in the others, including

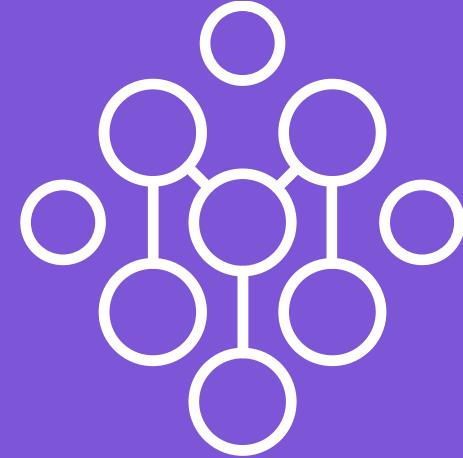
- Double DQN
- Prioritized Experience Replay
- Dueling DQN
- Monte Carlo Tree Search
- Actor Critic (also including A3C, A2C, Sof actor-critic )
- Proximal Policy Optimization
- Curiosity Based-exploration

## Hide & Seed



## RL in Colab

<https://colab.research.google.com/drive/1g7doH2zk8vKXACoe4R0HPFX7N48ehe01?usp=sharing>



# Metis

04/04/2024 - Causal  
Inference Training

POWERED BY AISENT

# Index

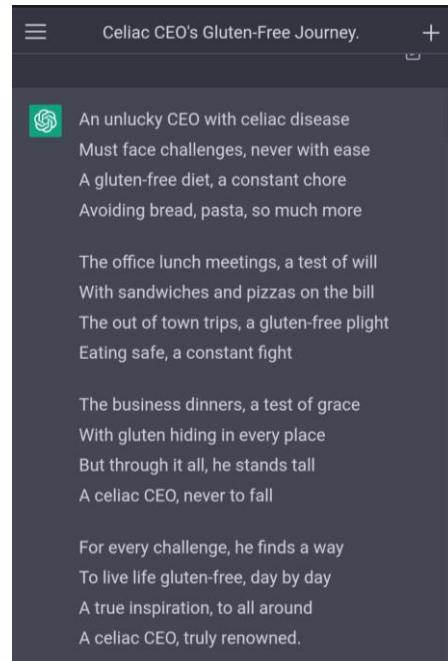
- Introduction
  - "*Correlation is not causation*"
  - *Causal Inference* : what is it and why is it needed?
- Models Reasons
  - What a model does causal
  - Types of models
  - Identification and estimation of the effects
- Methods and techniques
  - Randomized Controlled Trials
  - Observational Studies
  - Difference-in-differences
  - Instrumental variables

## Where are we now

AI algorithms are increasingly complex and obtain increasingly better and surprising results.

Yet **none** of the current algorithms can truly be said to be **intelligent**. They learn correlations between variables, even very distant ones, but are unable to understand the relationship between cause and effect.

Why isn't correlation enough?

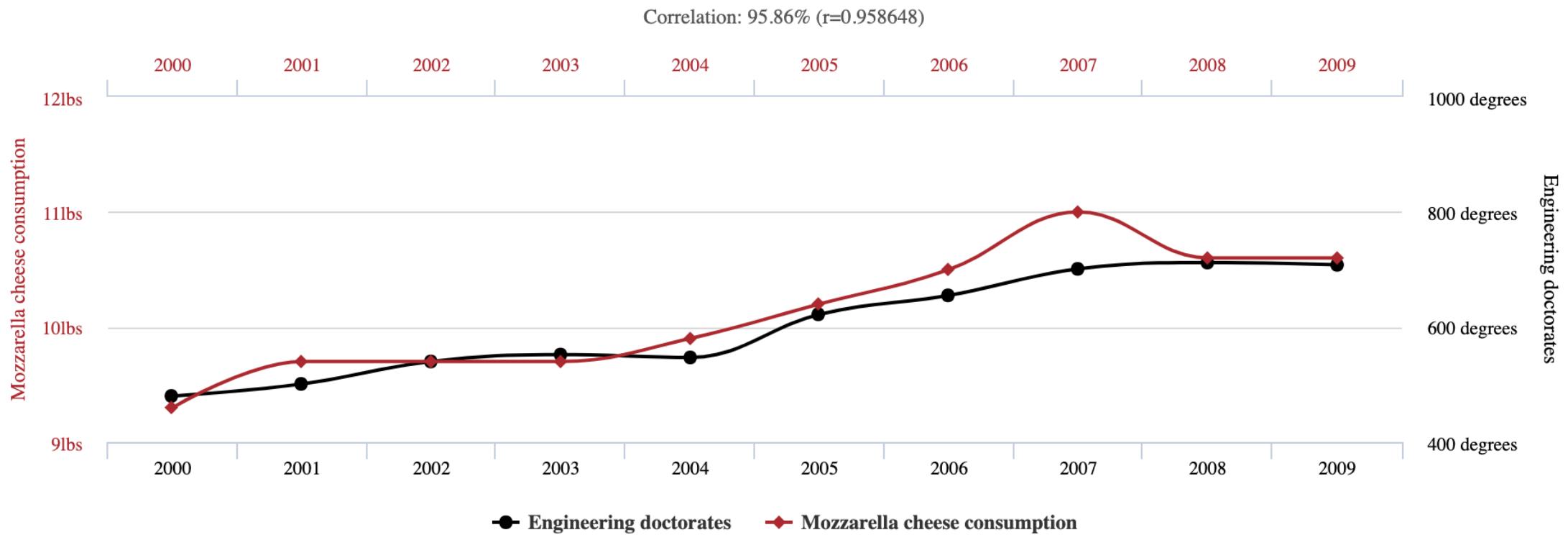


ChatGPT / Midjourney

# Correlation: does it imply causation? Not always!



## Per capita consumption of mozzarella cheese correlates with Civil engineering doctorates awarded

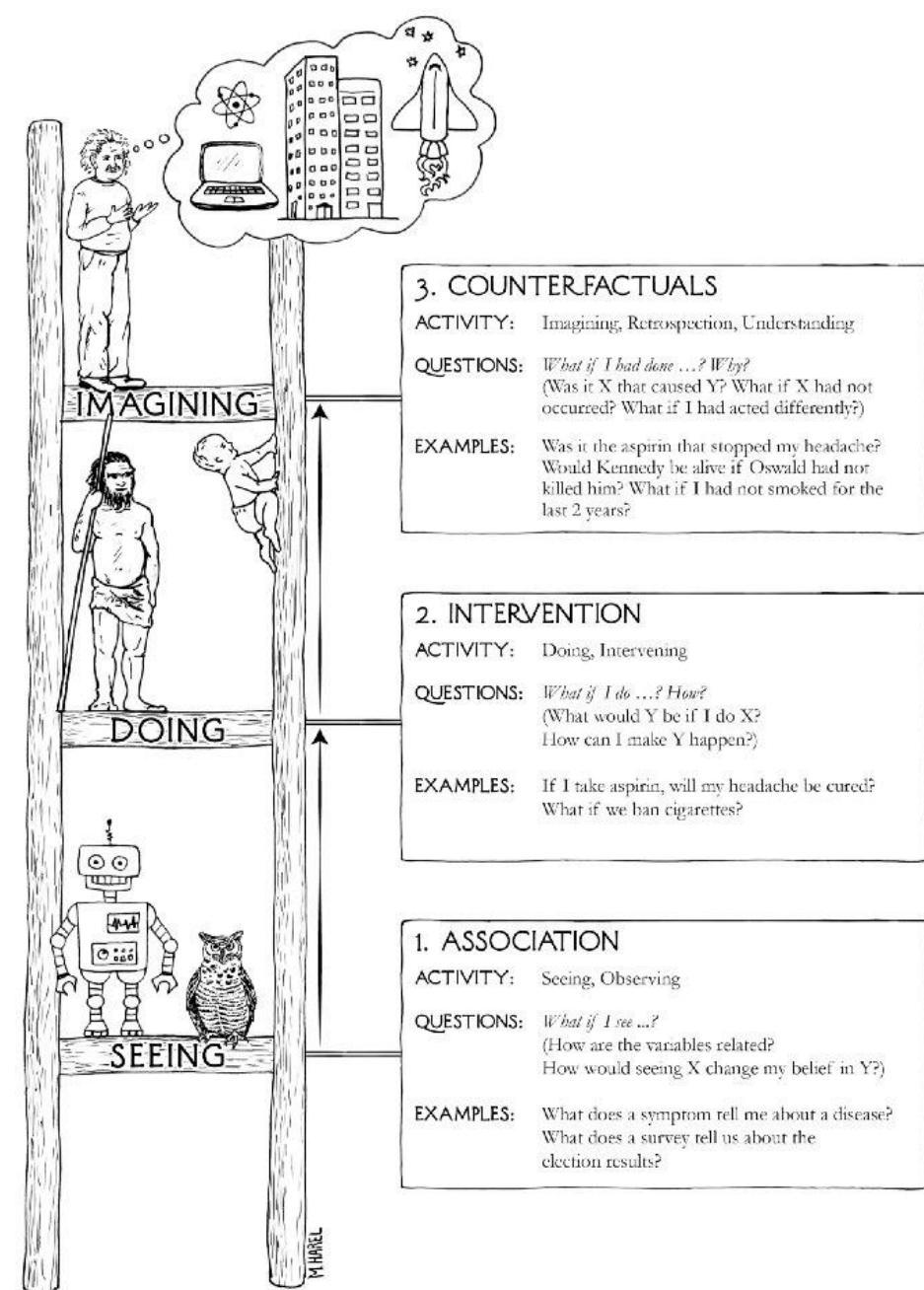
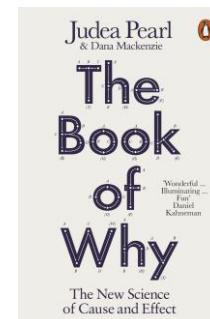


# Returning to causality

Causal models start from the concept that humans reason on cause-effect relationships.

The only way to truly understand what happens is to experience what happens through "intervention".

Judea Pearl (UCLA), J. Robins (Harvard), Y. Bengio (Turing Award) and others have discovered a series of techniques that ensure that questions can be answered in a cause-effect context.



# What are we talking about when we talk about “Causal Inference”?

More specifically, we can define Causal Inference as

- A set of techniques that allow us to understand the cause-effect relationships between different elements of the system under study.
- This includes both the determination of causality and its quantification.

We can think of Causal inference as the process that allows us to transform questions or intuition about the system under study into mathematical expressions in terms of random variables.

Today we will look at several ways in which Machine Learning can help us with this task.

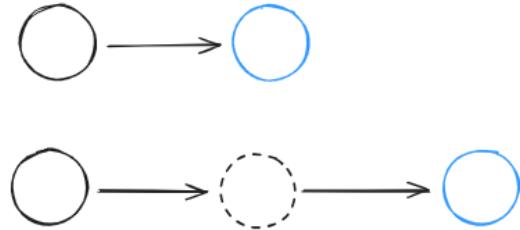
## When is it needed?

- Let's imagine the occupancy data of a hotel: the price is positively correlated with the number of rooms booked. A simple model might conclude that raising room prices is enough to increase occupancy.
- Instead, we humans who reason causally know that in the high seasons room prices rise and therefore it is not the price that causes greater occupancy.
- Unfortunately, predictive algorithms by construction cannot capture these relationships and therefore must be set manually as in the example above.
- But what if the example is less banal than hotels during vacation time? If we do not know from first principles the causal connections between the variables in our system, we risk basing our predictions on spurious cause-and-effect relationships.

# When is it needed?

There are many types of ( cor )relationships between data

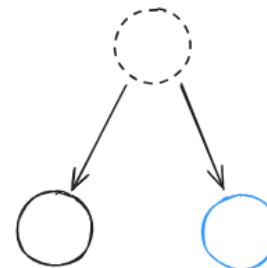
Causation



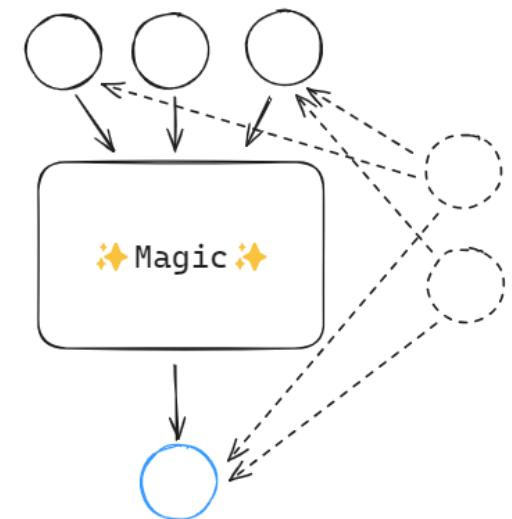
Spurious correlations



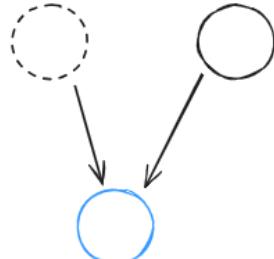
Confounders



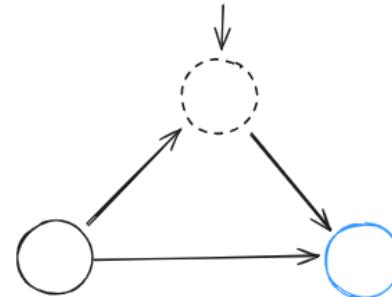
ML Approach



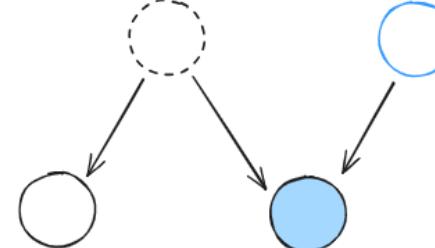
Partial observation



Intermediate effects



Observed variables



# Causal Models

## Causal Models: Definition

As we said before, the concept of **intervention** is crucial to making sense of causality. **If everything were predetermined it wouldn't make much sense to talk about causality!**

This connection is evident in the sentence

*“If I do X, then Y will happen”*

(Not to be confused with “*If X, then Y*”).

The most succinct definition is that a Causal model allows us to quantify the effect of an intervention. There are different types of models, and in some of these the concept of intervention will be more or less explicit. Nonetheless, it helps to fix ideas to think in a causal model as a tool for moving from a question about our system to an answer in terms of observational data or other information available to us.

## Causal Models: Definition

$$P(Y|X = x_0)$$

### Observational Distribution

Probability of Y given variable X is observed to be value  $x_0$

$$P(Y|do(X = x_0))$$

### Interventional Distribution

Probability of Y given variable X is artificially set to  $x_0$

## Causal Models: Types

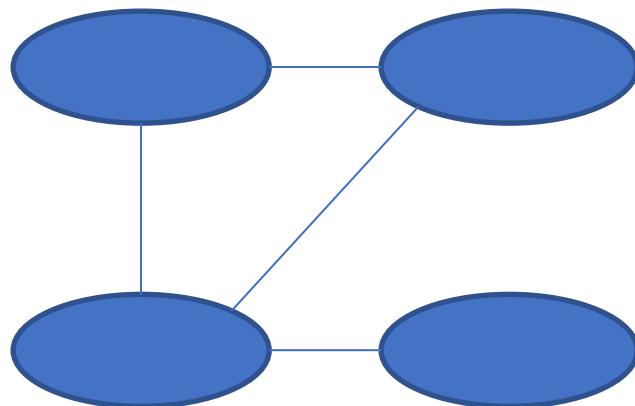
In talking about the different types of causal models we will introduce some useful elements to describe causal relationships and the effects that condition them.

In these slides we will explain some of the most used approaches in the world of Causal Inference.

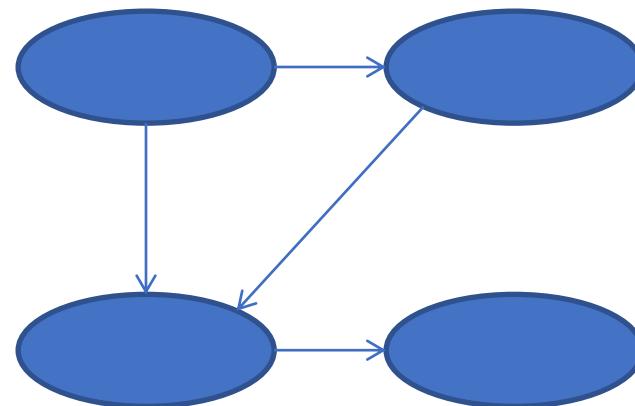
# Probabilistic Graphical Models

The first of these tools are [graphs](#). We will try to stay away from mathematical rigor, but we must at least mention a few terms:

- A [graph](#) is a set of [nodes \(vertices\)](#) and [edges](#) that connect them.
- Today we will deal mainly with [directed graphs](#), i.e. those where the edges have a direction.



Non-directed graph

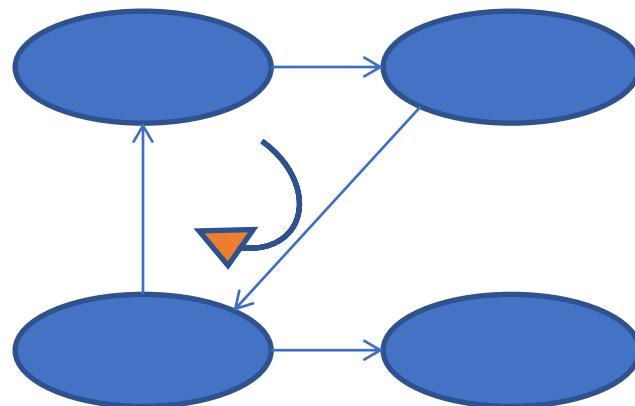


Direct graph

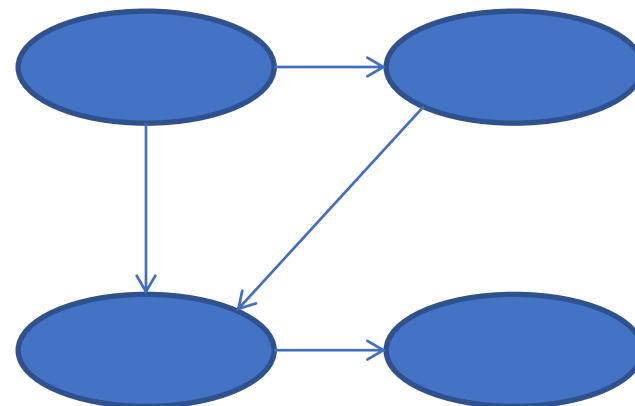
# Probabilistic Graphical Models

We define a [path](#) in this graph as an ordered set of nodes where each node is connected by an edge to the previous one. In a [directed graph](#) the edges must all have the same direction.

A closed path (where the last node is also the first) is known as a loop. For reasons that will soon become clear we will focus on directed acyclic graphs (DAGs).



Directed graph with a cycle

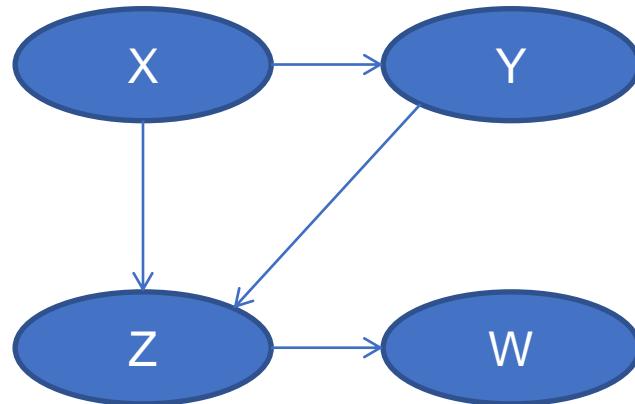


Acyclic directed graph

# Probabilistic Graphical Models

We think that for each node we have a random variable. Direct arcs encode the dependency relationships between variables. In concrete terms we have that a variable Y depends on a variable X only if there is an arc  $X \rightarrow Y$ . This together with the chain rule for probabilities simplifies the calculation of probabilities greatly.

An example:



$$P(w|x, y, z) = P(w|z)$$

$$P(z|x, y, w) = P(z|x, y)$$

## Conditional Probability Tables (CPT)

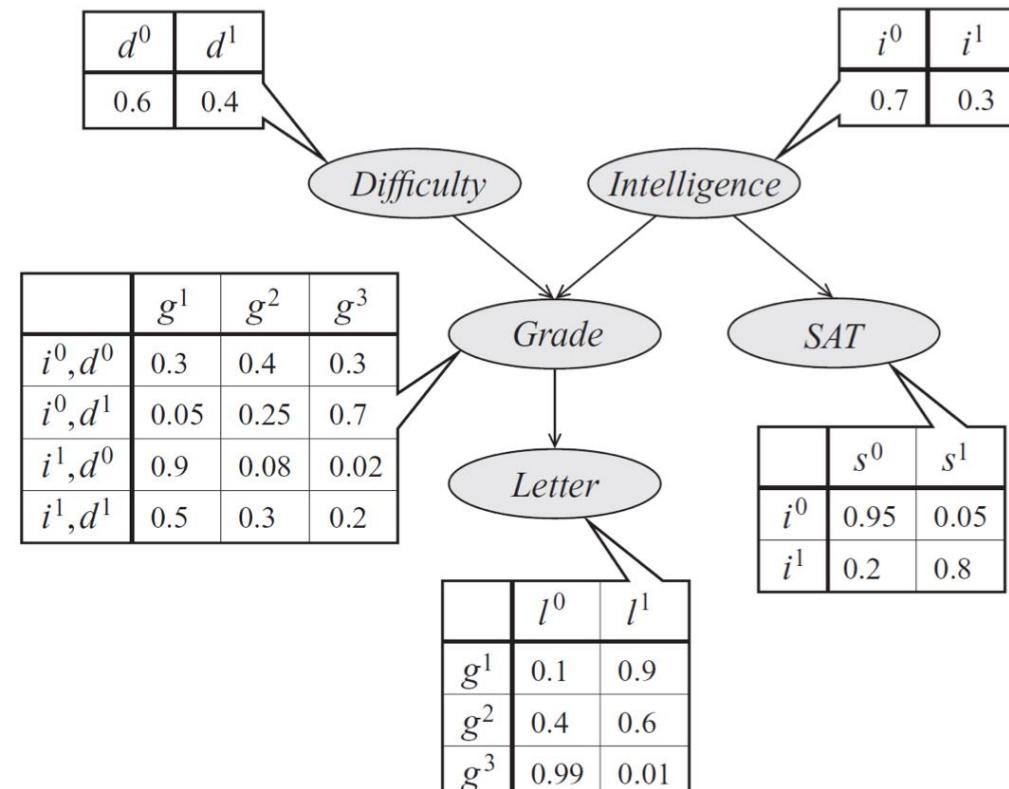
A very practical way of representing conditional probabilities is through tables that show the possible combinations of parent variables (in the case of discrete distributions). So, if in the previous case we have all binary variables (for simplicity), our Conditional Probability Table (CPT)

$$P(z|x,y,w) = P(z|x,y)$$

x, y	P(z = 0   x, y)	P(z = 1   x, y)
0.0	0.3	0.7
0.1	0.2	0.8
1.0	0.4	0.6
1.1	0.9	0.1

# Bayesian Networks

A DAG matched to the set of CPTs describing its nodes is called a Bayesian Network. These objects will be the focus of the afternoon session, so now we won't stop too much.



<https://ermongroup.github.io/cs228-notes/representation/directed/>

## Alternative routes

It is difficult to underestimate how powerful Bayesian Networks are, as they manage to summarize complex cause-and-effect relationships in a simple to analyze graphical structure. Unfortunately they presuppose a profound knowledge of the phenomenon being studied and if we do not have the CPTs (at least indicatively) their potential is very limited. The rest of this course we will study techniques that allow us to quantify the effect of one variable on another.

There are two (more or less) equivalent approaches for this purpose

- Potential Outcomes Framework
- Structural Causal Models

## How to speak from uncertainty?

Let's imagine we have a population whose individuals we will denote with values of the variable  $i$ . Let's think about a possible result (*Outcome*) that could happen for this population. To fix our ideas we could think about the possibility of falling ill with a certain disease. We will denote by  $Y_i$  the individual's health status i.

More concretely,  $Y_i = 1$  it means that the individual i has become ill, while that  $Y_i = 0$  denotes that i has remained healthy.

## Treatment

Borrowing a term from the epidemiological literature we can call *Treatment* the random variable that codes whether an individual has received treatment (or more generally has been "affected" by an intervention).

Returning to our example, there may be a vaccine that is administered to some individuals ( $T = 1$ ) or not ( $T = 0$ ).

# Potential Outcomes

In practice the question one can ask is:

*“What effect does the drug have on the chance of getting sick?”*

Now we need to introduce the notation for *potential outcomes*.

If we could give the drug to an individual, see if he gets sick (which we will denote  $Y_i(1)$ ) and then go back in time and decide not to give it to him and see whether he gets sick the same or not (which we will denote  $Y_i(0)$ ) we could calculate this effect.

For a single individual this is known as *the unit treatment effect*

$$Y_i(1) - Y_i(0)$$

*Do not confuse  $Y(0,1)$  with  $Y$ , the observed value.*

## Average Treatment Effect (ATE)

If instead we consider the entire population we arrive at one of the key quantities of this approach:

$$\text{ATE} = E[Y_i(1) - Y_i(0)] := \frac{1}{N} \sum_{i=0}^N Y_i(1) - Y_i(0)$$

## Average Treatment Effect on the Treated(ATT)

Focusing on the treated population we have this other quantity.

$$\text{ATT} = E[Y_i(1) - Y_i(0) \mid T = 1]$$

# We only have one problem

The linearity of the expected value tells us that

$$\text{ATT} = E[Y_i(1) - Y_i(0) | T = 1] = E[Y_i(1) | T = 1] - E[Y_i(0) | T = 1]$$



Factual



Contra-Factual

If we were omniscient knowing the second term would not be a problem, but in most applications this is not the case.

# How do we get closer to an observable quantity?

The only information available to us is the rest

$$E[Y_i | T = 1] - E[Y_i | T = 0]$$

We add a zero of the form  $E[Y_i(0) | T = 1] - E[Y_i(0) | T = 1]$ :

$$E[Y_i | T = 1] - E[Y_i | T = 0] = E[Y_i(1) - Y_i(0) | T = 1] + E[Y_i(0) | T = 1] - E[Y_i(0) | T = 0]$$



ATT



Bias

## Bias: what is it?

$$E[Y_i(0) | T = 1] - E[Y_i(0) | T = 0]$$

This quantity represents a possible asymmetry between the population that receives the treatment and that which does not.

Eye: We are talking about contra-factual quantities. We are comparing whether **in the absence of treatment** the group  $T = 1$  and the control ( $T = 0$ ) would have had the same result  $Y(0)$ .

We can guarantee the absence of Bias by asking for the independence of the *potential outcomes* with respect to the value of  $T$ . This is written as  $(Y(0), Y(1)) \perp\!\!\!\perp T$ . We will see later in which cases this condition is respected or not

## Bias: where does it come from?

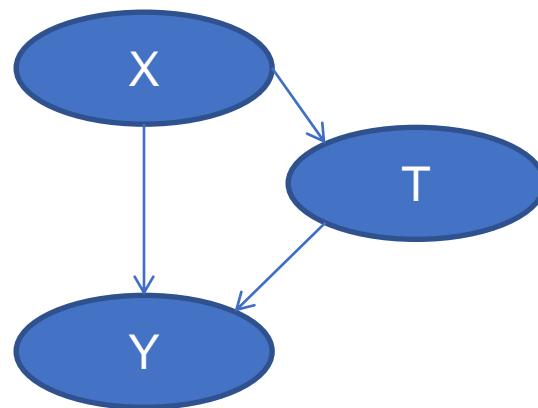
To explain the possible sources of bias the graph language will be very useful. Bias in the ATT measurement arises when other things (which we will refer to generically as the variable X) change along with our treatment T.

The clearest way this can happen is if the probability of receiving treatment is influenced by a variable which in turn can have an effect on Y. Think of a drug for a serious illness that is given more often to seriously ill people and so **it would seem that receiving the drug increases the probability of getting sick !**

This is a very common error and is called “Confounding bias”.

## Bias: where does it come from?

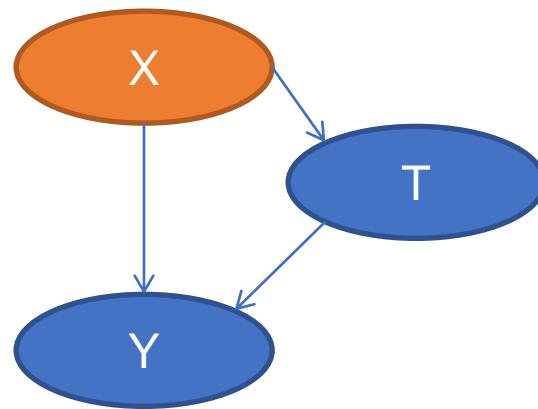
Graphically the situation is this:



The idea is that the probability “flows” from T following other paths than the direct one and that violates the condition  $(Y(0), Y(1)) \perp\!\!\!\perp T$ . From the example of drugs we can deduce that if we analyze serious cases separately we could understand if, given the same severity, the drug has some effect.

## Bias: how to avoid it?

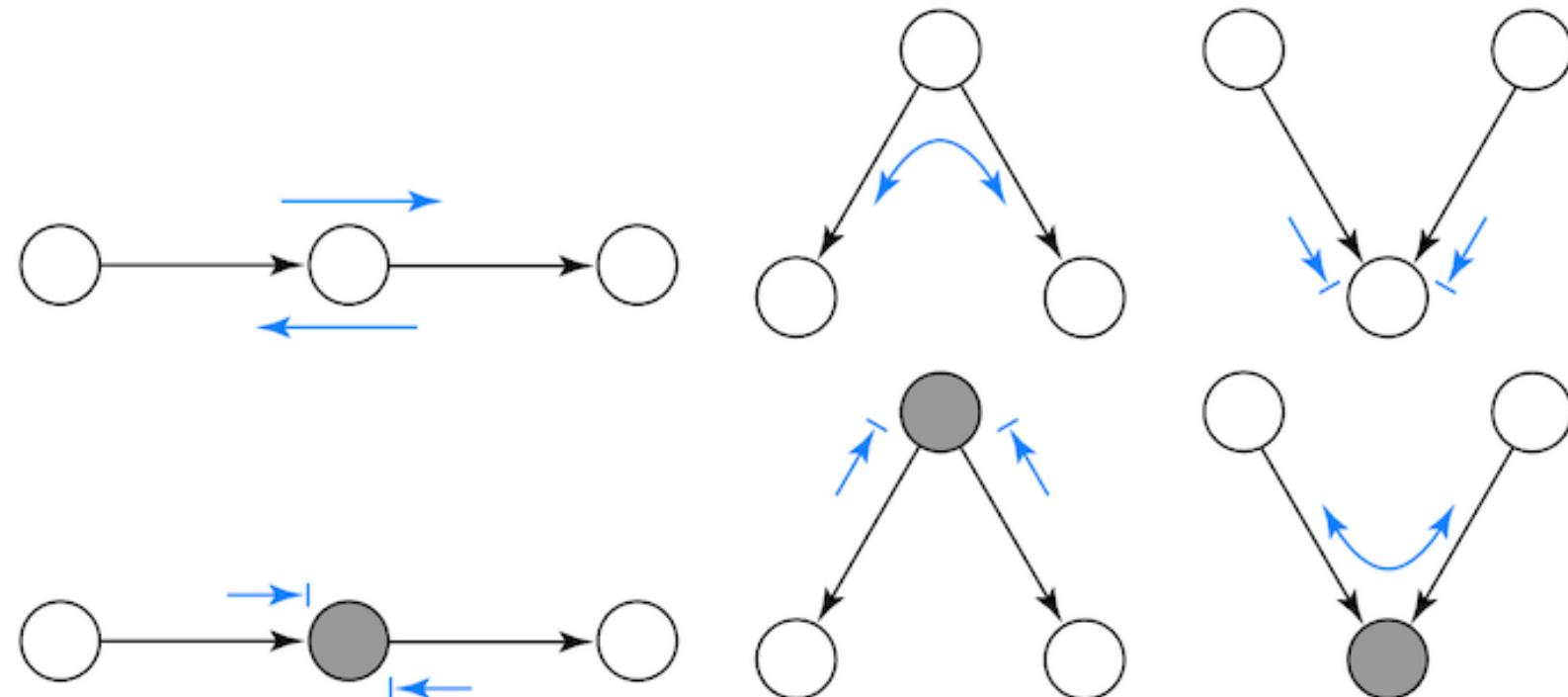
The action of separating the effect of



In this case we say that the counterfactuals are independent with respect to  $T$  if we condition on  $X$ :  $(Y(0), Y(1)) \perp\!\!\!\perp T | X$ .

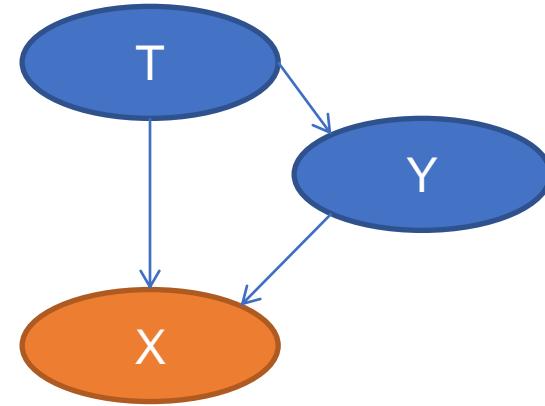
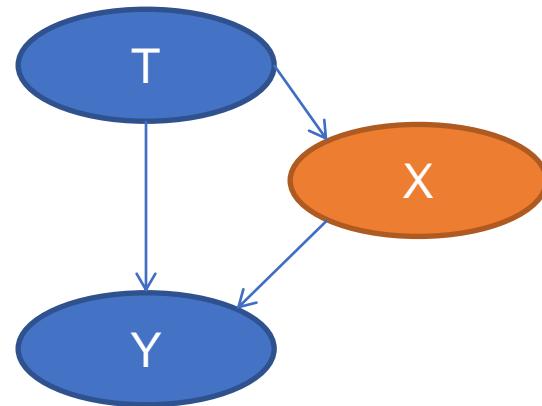
## Parentheses: *probability flow*

In the afternoon we will delve deeper into the concept of "probability flow" but for now we can take these situations as valid (the gray balls indicate a conditional variable):



## Bias: where does it come from? (pt. 2)

Since we have resolved the confounding bias by controlling the so-called confounding variables, we may be tempted to condition on all the variables we find in our model. Before that, let's consider the following structures.



In these cases we even have  $(Y(0), Y(1)) \perp\!\!\!\perp T$ , but conditioning on  $T$  in the second because it opens a new path for the probability from  $T$  towards  $Y$ .

# Methods

## Randomized Controlled Trials (RCTs)

The cleanest method to eliminate the bias is to ensure that the T=0 and T=1 groups are statistically identical.

If we take them from the same population and take enough of them we can be pretty sure that  $Y(0)$  for each group will be the same (i.e., in the absence of treatment both groups would have the same outcome).

That is to say,

$$E[Y_i(0) | T = 1] - E[Y_i(0) | T = 0] = 0$$

## *Matching and Propensity scores*

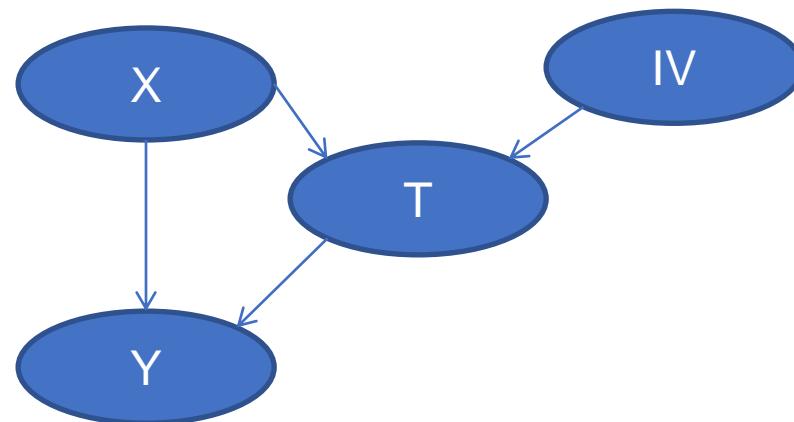
It is not always feasible (or ethical) to organize an RCT. If we want to monitor the effects of smoking during pregnancy, it seems rather inadvisable to have a group of pregnant women smoke.

Fortunately, there are techniques that allow us to “match” participants in the test group with those in the control group to ensure that the ATT estimate is reliable and reduce bias.

## *Instrumental variables*

We saw in the previous section that confounding variables can introduce strong biases. Conditioning them would seem to be a very simple and elegant solution, but what to do when these variables are not observable?

In these cases it is necessary to propose a variable (called instrumental) that is not correlated with the confounding ones (X) but still has an effect on the treatment (and therefore on the outcome Y).



## *Difference in differences*

When there is an intervention marked by a clear temporal character it is often possible to check how the trend changes in the treated group and compare it with the change in the control group.

Seeing the “difference in differences” can often give us a measure of the effect of our intervention.