Machine Learning Basics

Marcello Pelillo
University of Venice, Italy

Image and Video Understanding

a.y. 2018/19

What Is Machine Learning?

A branch of **Artificial Intelligence (AI)**.

Develops algorithms that can **improve their performance** using training data.

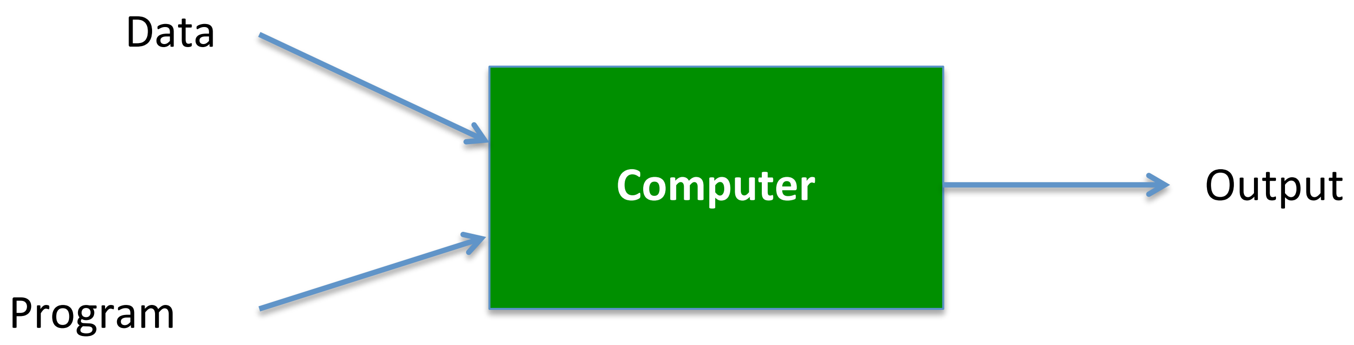
Typically ML algorithms have a (large) number of parameters whose values are learnt from the data.

Can be applied in situations where it is very challenging (= impossible) to define rules by hand, e.g.:

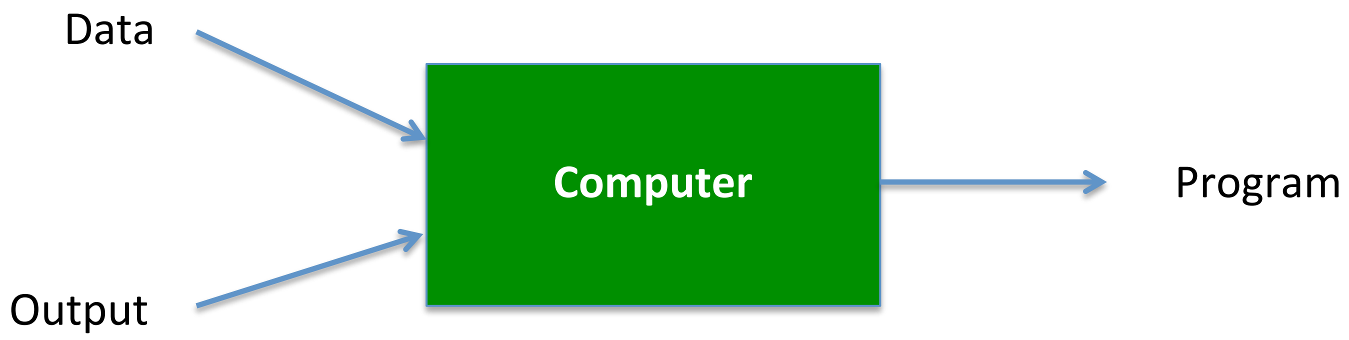
- Computer vision
- Speech recognition
- Stock prediction
- …

Machines that Learn?

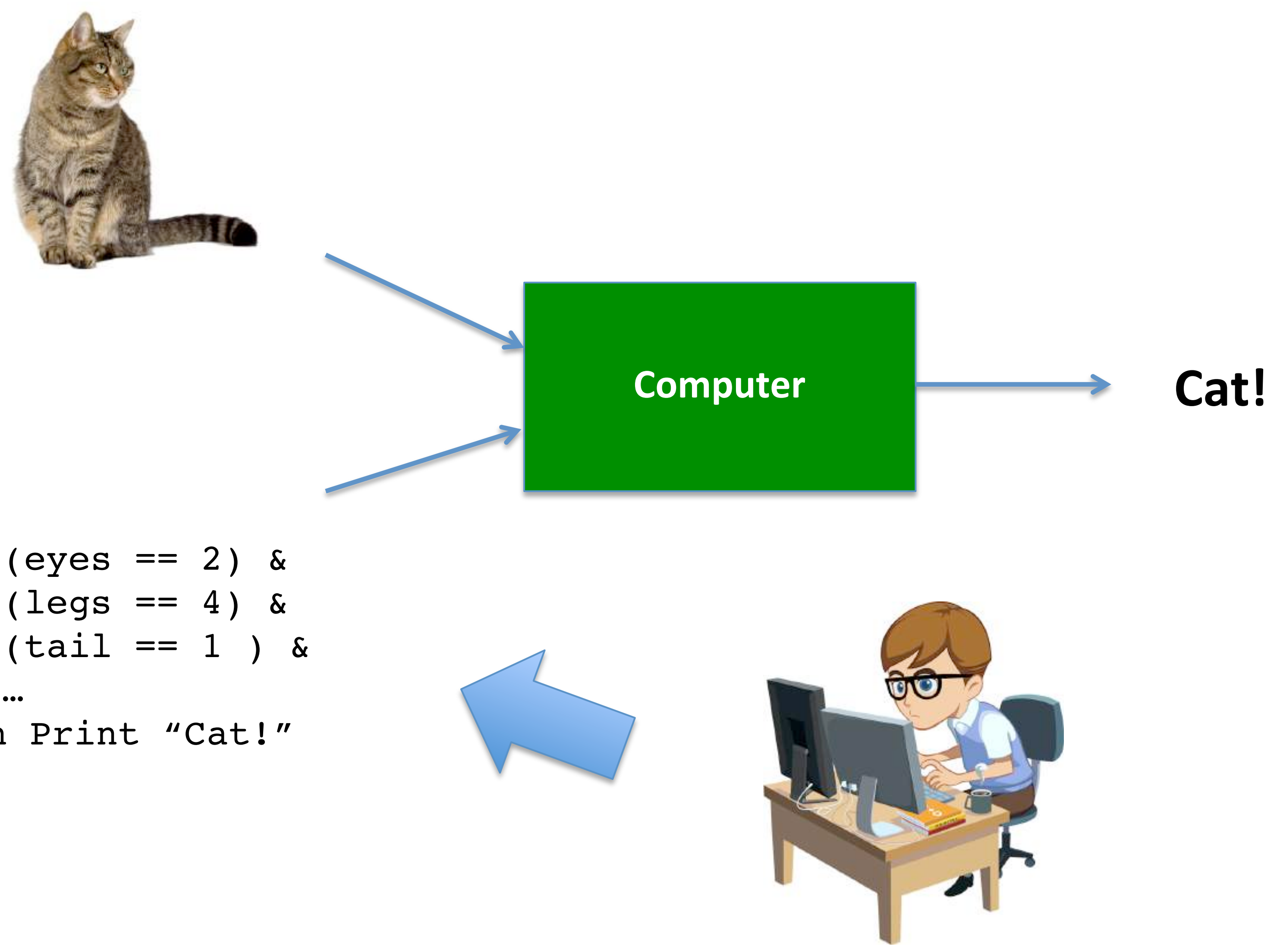
Traditional programming



Machine learning

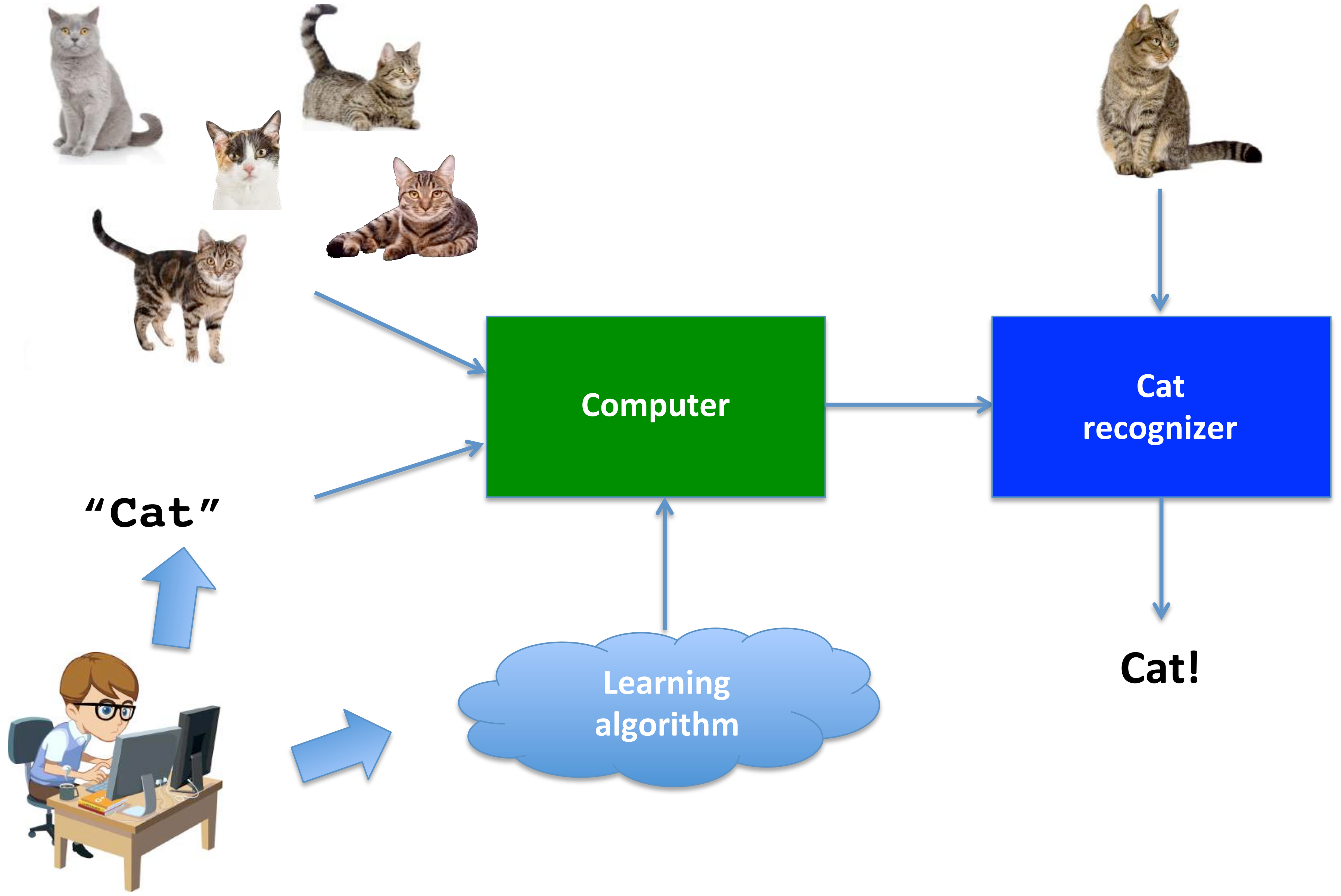


Traditional Programming



```
if  (eyes == 2) &
    (legs == 4) &
    (tail == 1 ) &
    …
then Print “Cat!”
```

Machine Learning



Data Beats Theory

«By the mid-2000s, with success stories piling up, the field had learned a powerful lesson: **data can be stronger than theoretical models**.

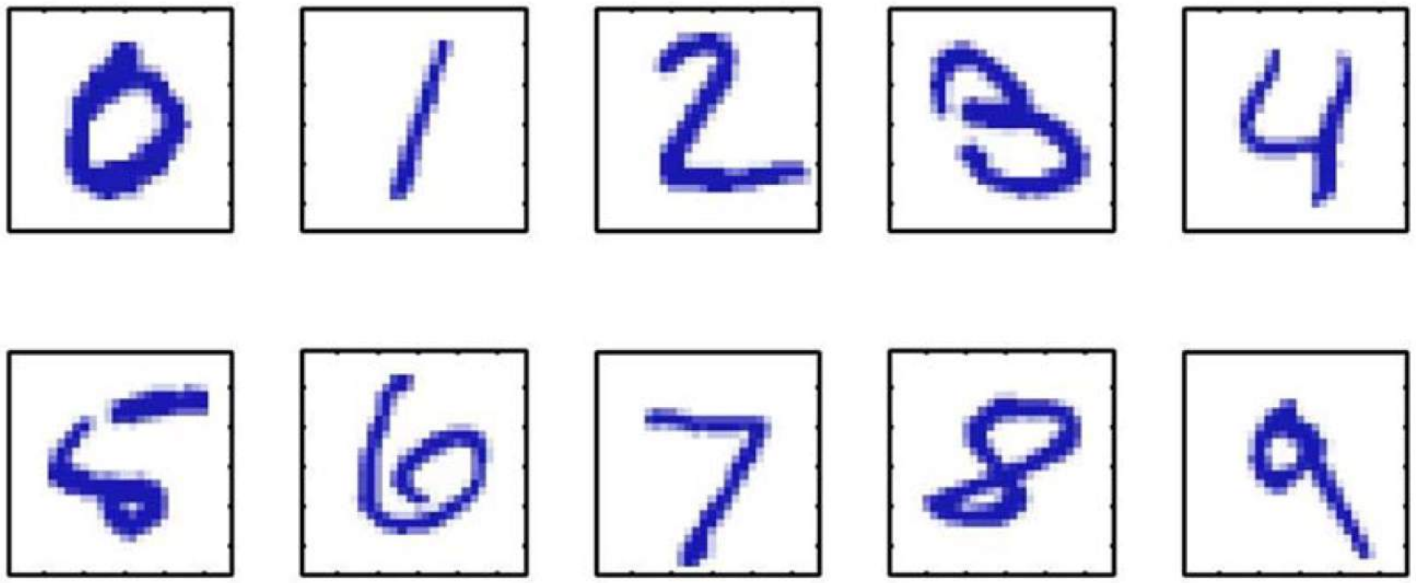
A new generation of intelligent machines had emerged, powered by a small set of statistical learning algorithms and large amounts of data.»

Nello Cristianini

The road to artificial intelligence: A case of data over theory

(New Scientist, 2016)

Example:
Hand-Written Digit Recognition



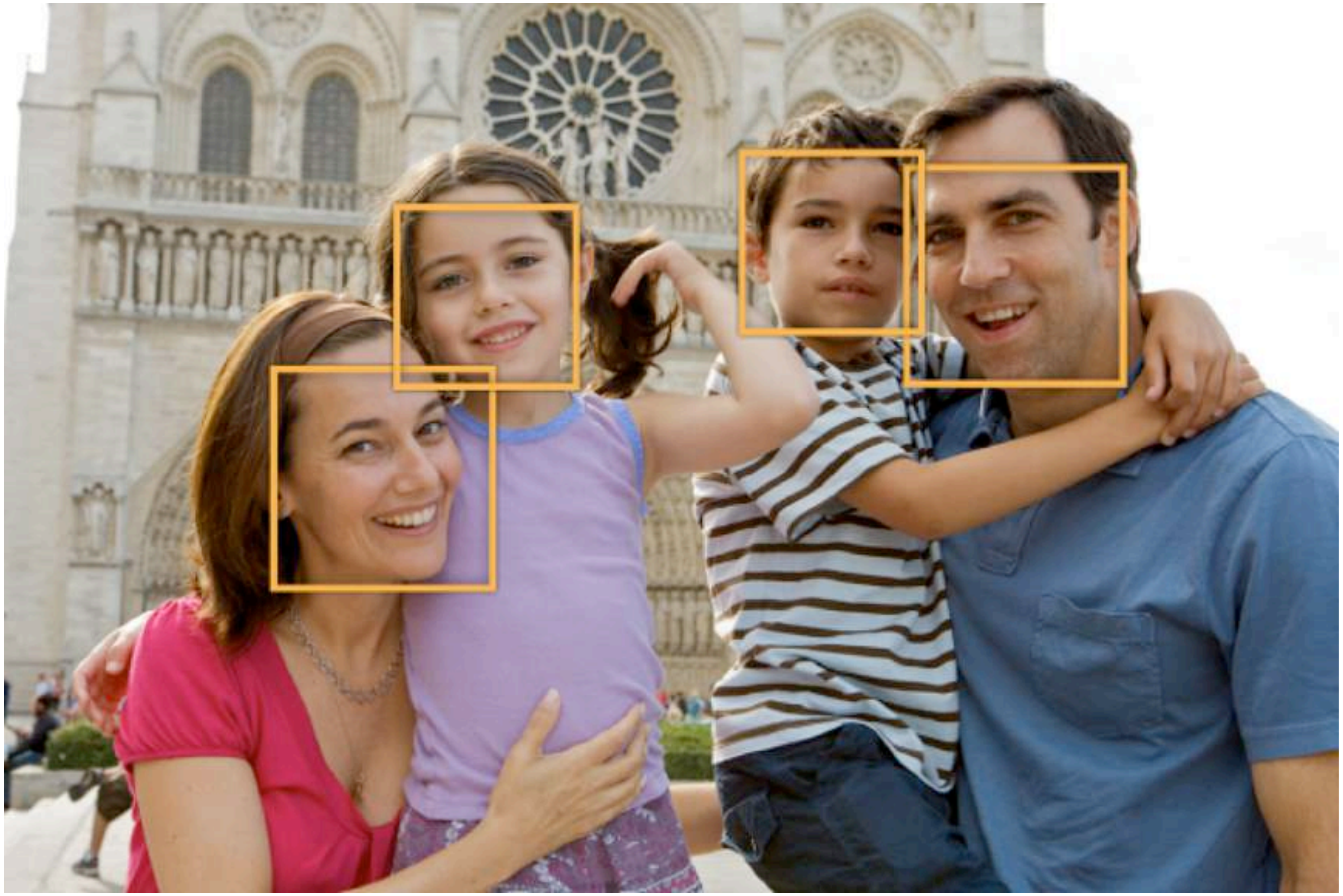
Images are 28 x 28 pixels

Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$

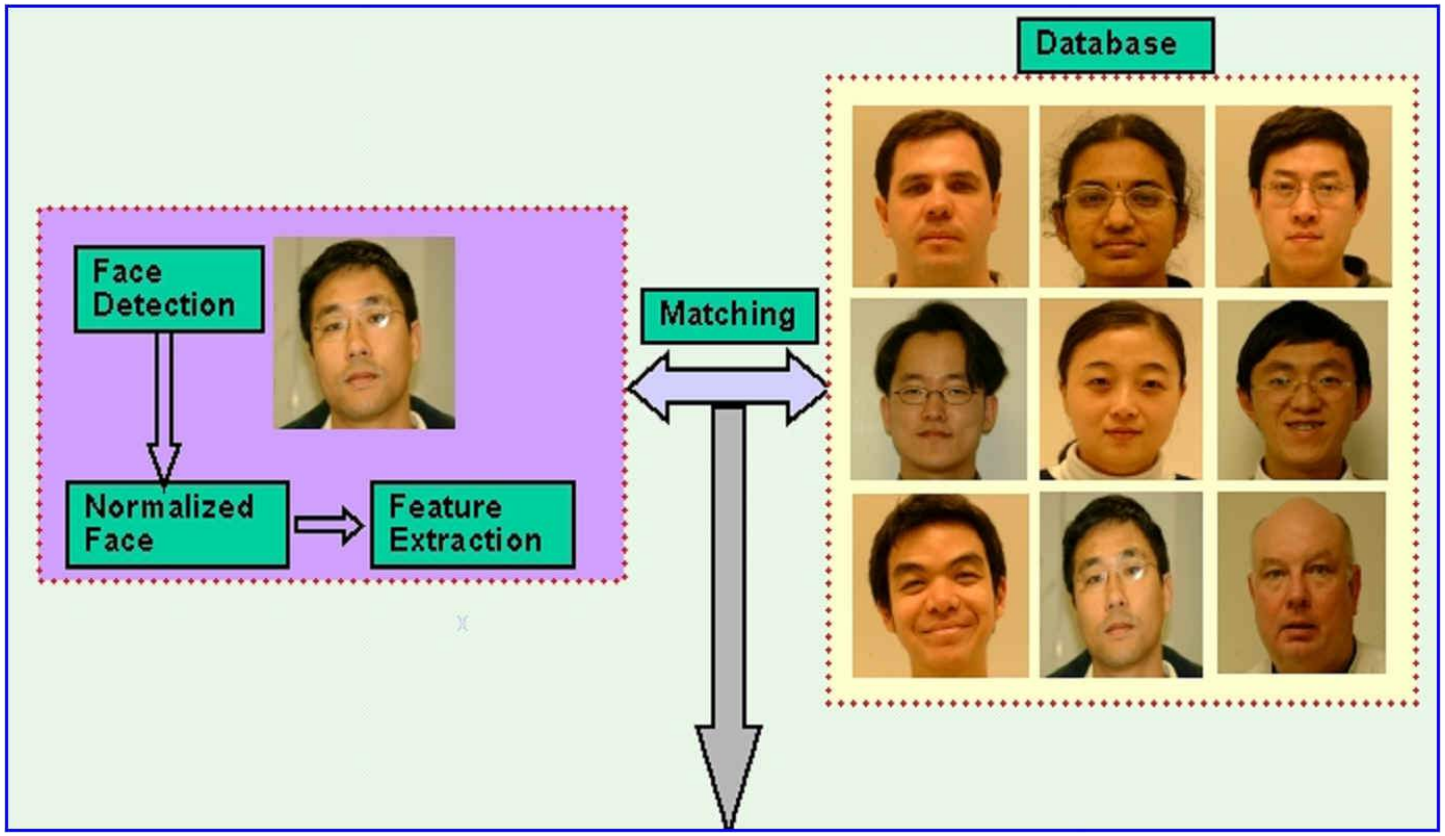
Learn a classifier $f(\mathbf{x})$ such that,

$$f : \mathbf{x} \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

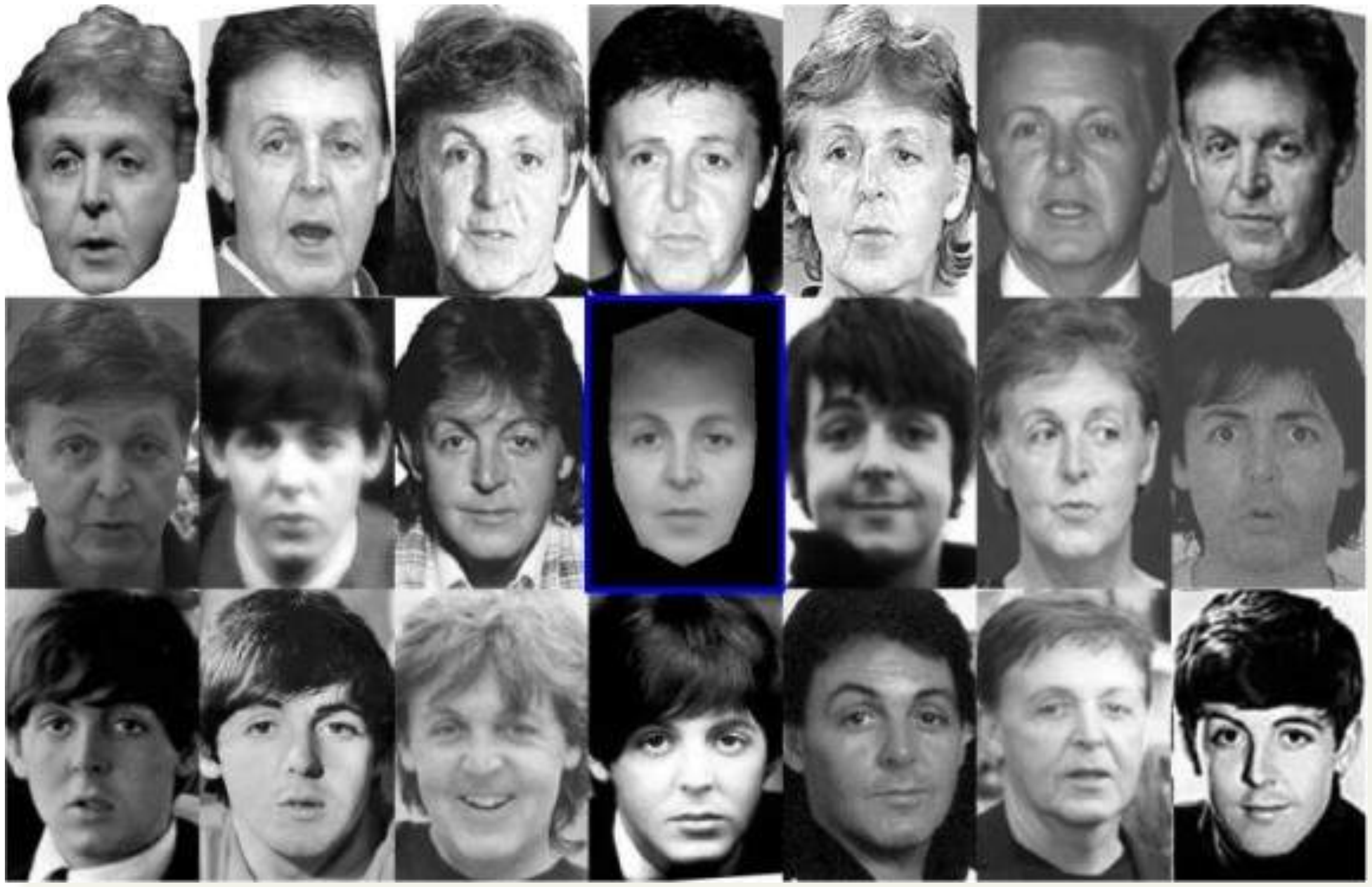
Example:
Face Detection



Example:
Face Recognition



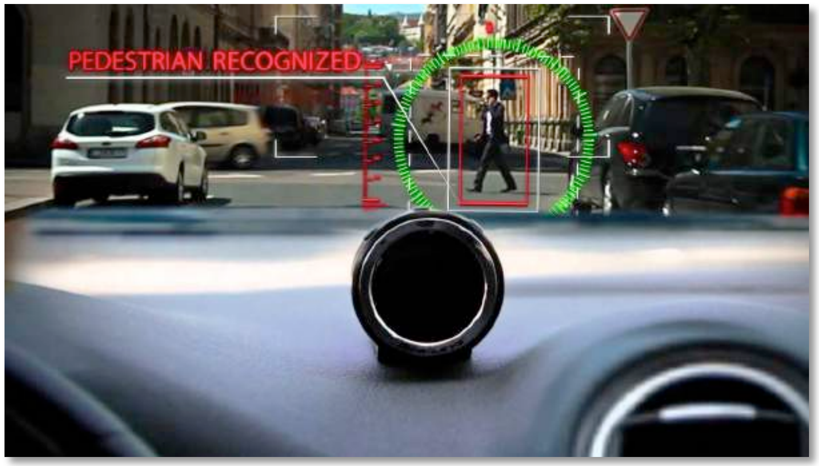
The Difficulty of Face Recognition

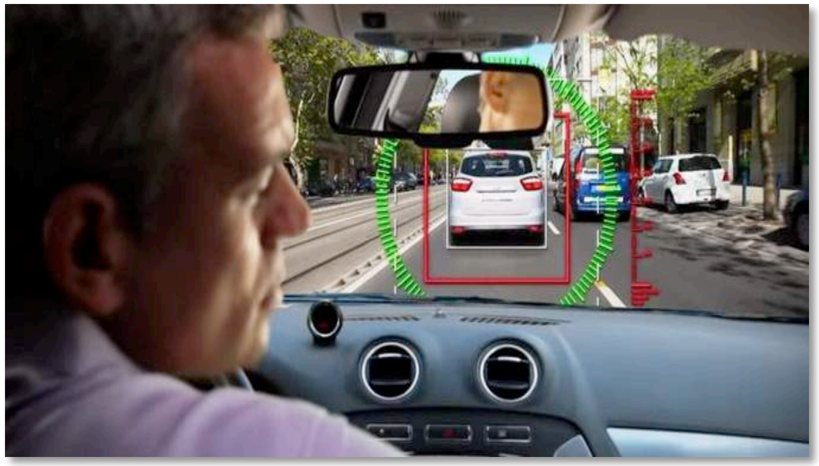


Example:
Fingerprint Recognition

$\xrightarrow{?}$

Assiting Car Drivers and Autonomous Driving





Assisting Visually Impaired People

Recommender Systems

amazon.com and you're done.

ebay

BARNES & NOBLE BOOKSELLERS

MOVIE GALLERY play on

iLike Beta / Artists / eighte8

Sign in or Create your own profile

HOLLYWOOD VIDEO

NETFLIX

bol.it

ibs.it internet bookshop

Three kinds of ML problems

- **Unsupervised learning (a.k.a. clustering)**
  - All available data are unlabeled
- **Supervised learning**
  - All available data are labeled
- **Semi-supervised learning**
  - Some data are labeled, most are not

Unsupervised Learning (a.k.a Clustering)

The clustering problem

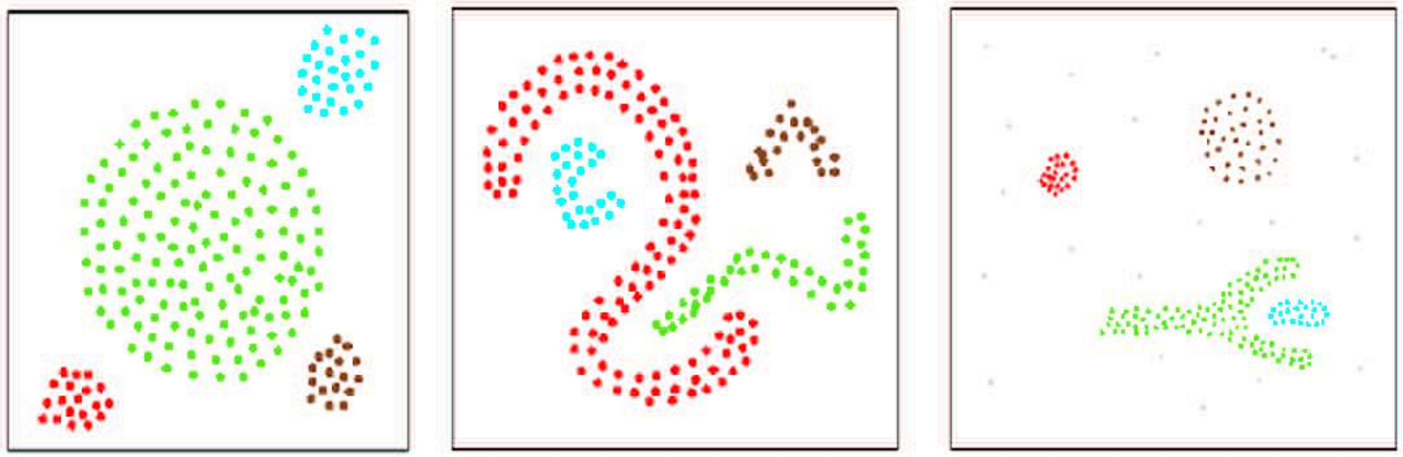
Given:

- ✓ a set of *n* "objects"
- ✓ an *n* × *n* matrix A of pairwise similarities

} = an edge-weighted graph G

Goal: *Partition* the vertices of the G into maximally homogeneous groups (i.e., clusters).

Usual assumption: symmetric and pairwise similarities (G is an undirected graph)



Applications

Clustering problems abound in many areas of computer science and engineering.

A short list of applications domains:

Image processing and computer vision
Computational biology and bioinformatics
Information retrieval
Document analysis
Medical image analysis
Data mining
Signal processing
…

For a review see, e.g., A. K. Jain, "Data clustering: 50 years beyond K-means,"
Pattern Recognition Letters 31(8):651-666, 2010.

Clustering

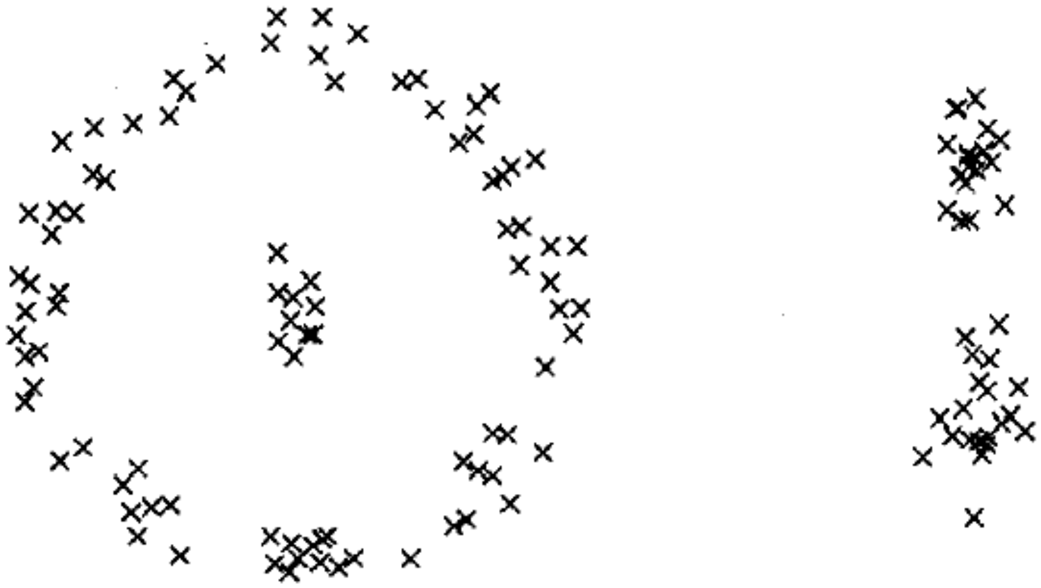
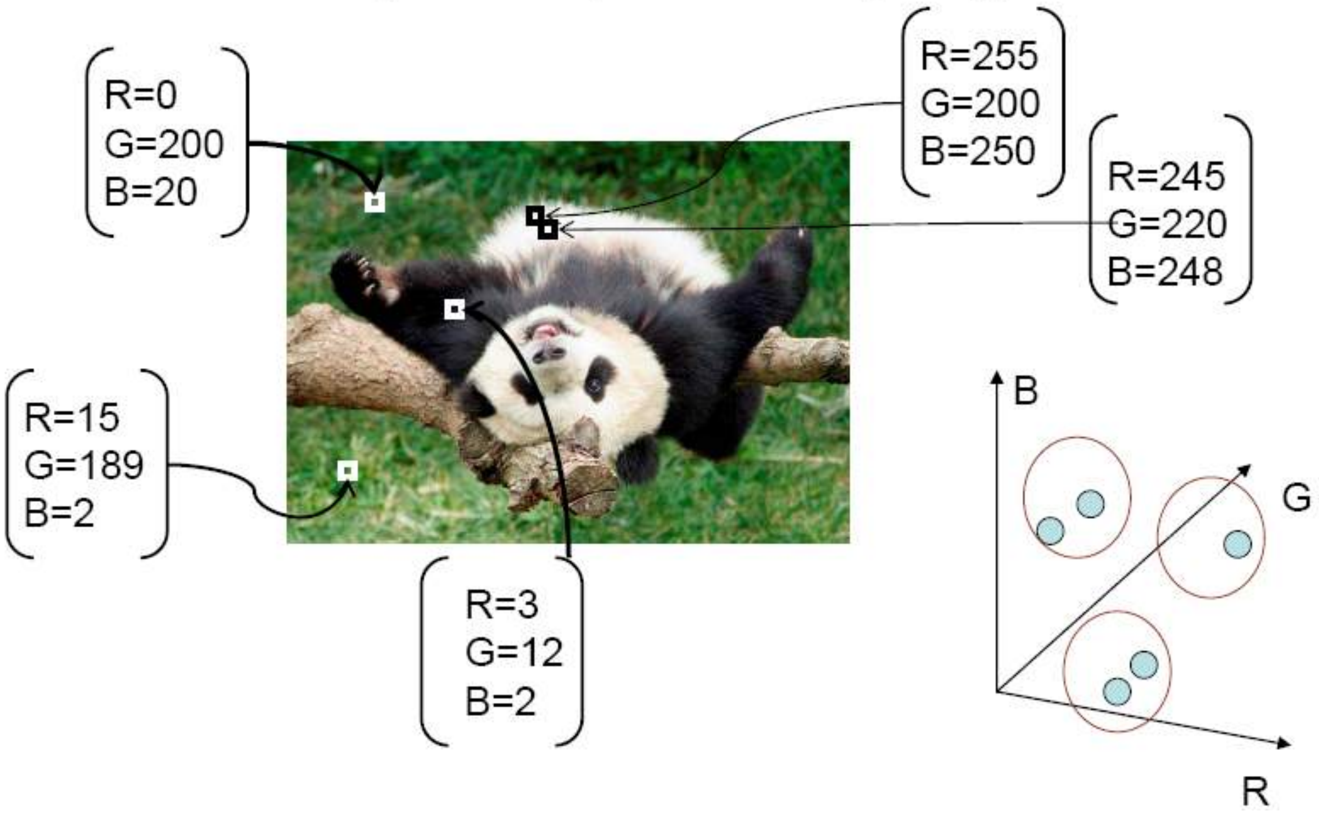


Figure 1: How many groups?

Image Segmentation as clustering

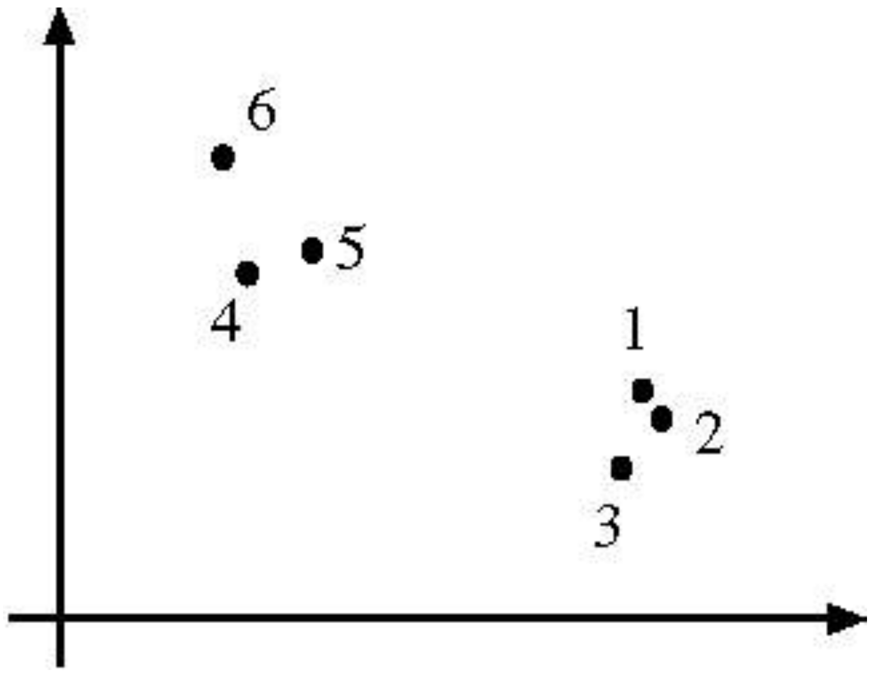
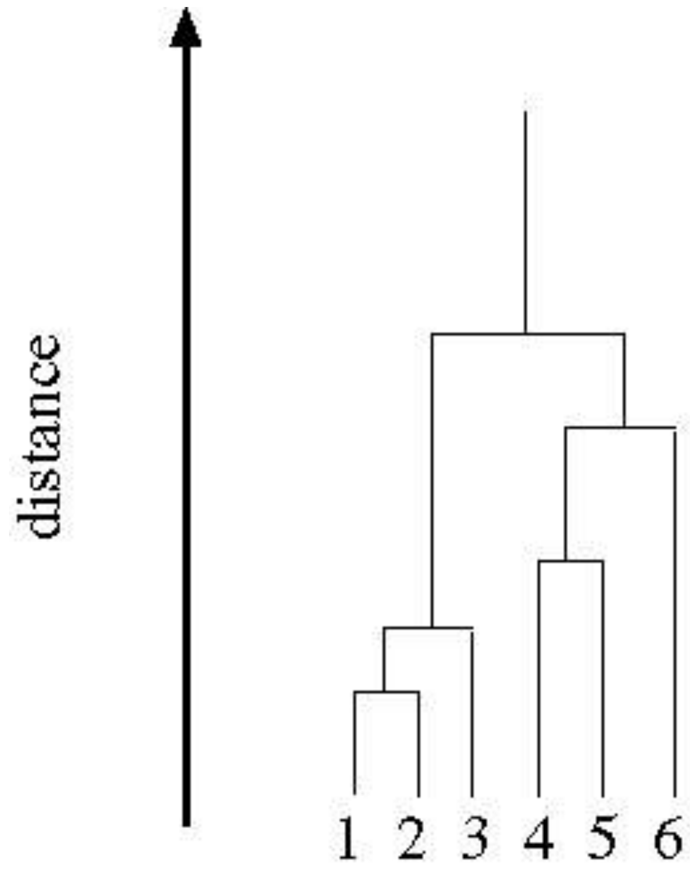
- Cluster similar pixels (features) together



Source: K. Grauman

Segmentation as clustering

- Cluster together (pixels, tokens, etc.) that belong together
- Agglomerative clustering
  - attach closest to cluster it is closest to
  - repeat
- Divisive clustering
  - split cluster along best boundary
  - repeat
- Point-Cluster distance
  - single-link clustering
  - complete-link clustering
  - group-average clustering
- Dendrograms
  - yield a picture of output as clustering process continues


6
5
4
1
2
3

distance
1
2
3
4
5
6

K-Means

An iterative clustering algorithm

– **Initialize:**

Pick *K* random points as cluster centers

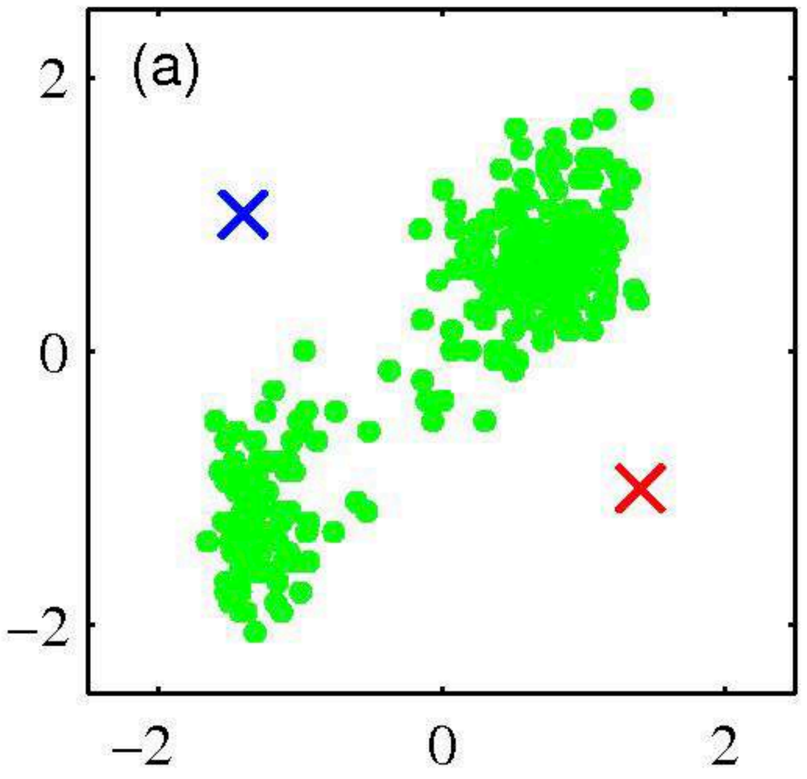
– **Alternate:**

1. Assign data points to closest cluster center
2. Change the cluster center to the average of its assigned points

– **Stop** when no points' assignments change

Note: Ensure that every cluster has at least one data point. Possible techniques for doing this include supplying empty clusters with a point chosen at random from points far from their cluster centers.

K-means clustering: Example

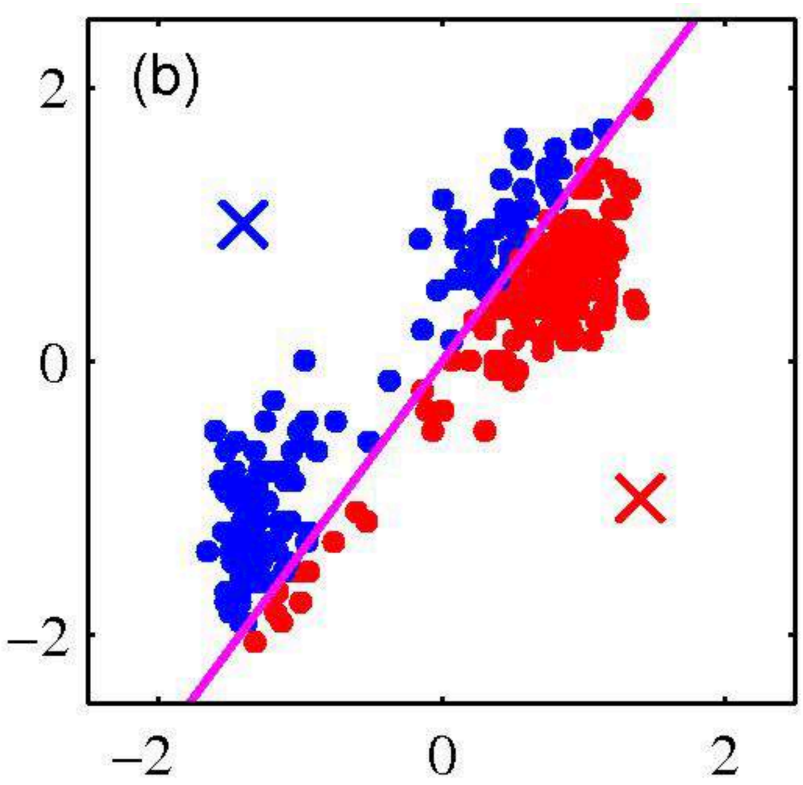


Initialization:
Pick K random points as cluster centers

Shown here for K=2

Adapted from D. Sontag

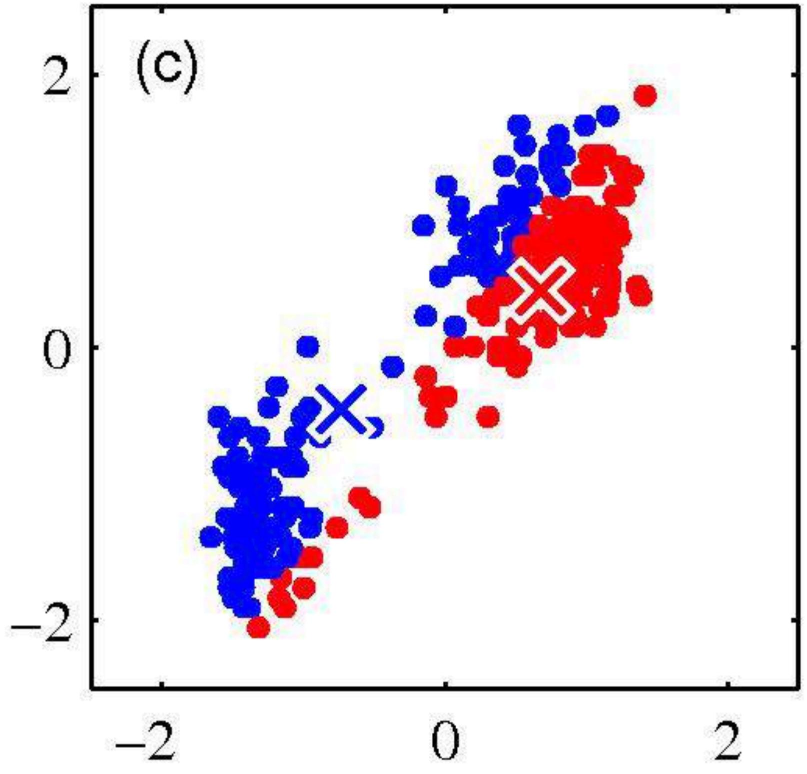
K-means clustering: Example



Iterative Step 1:
Assign data points to closest cluster center

Adapted from D. Sontag

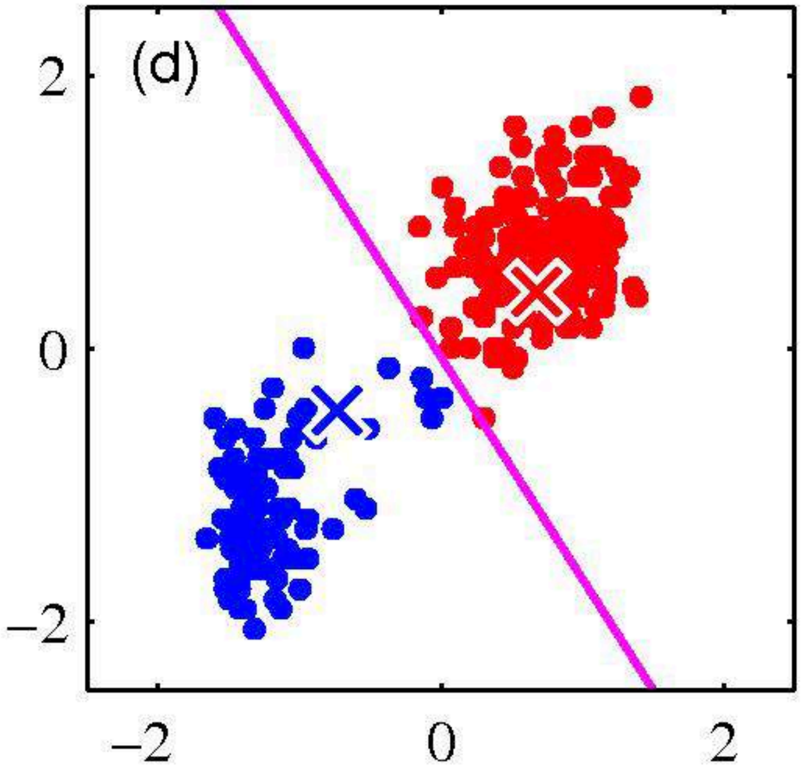
K-means clustering: Example



Iterative Step 2:
Change the cluster center to the average of the assigned points

Adapted from D. Sontag

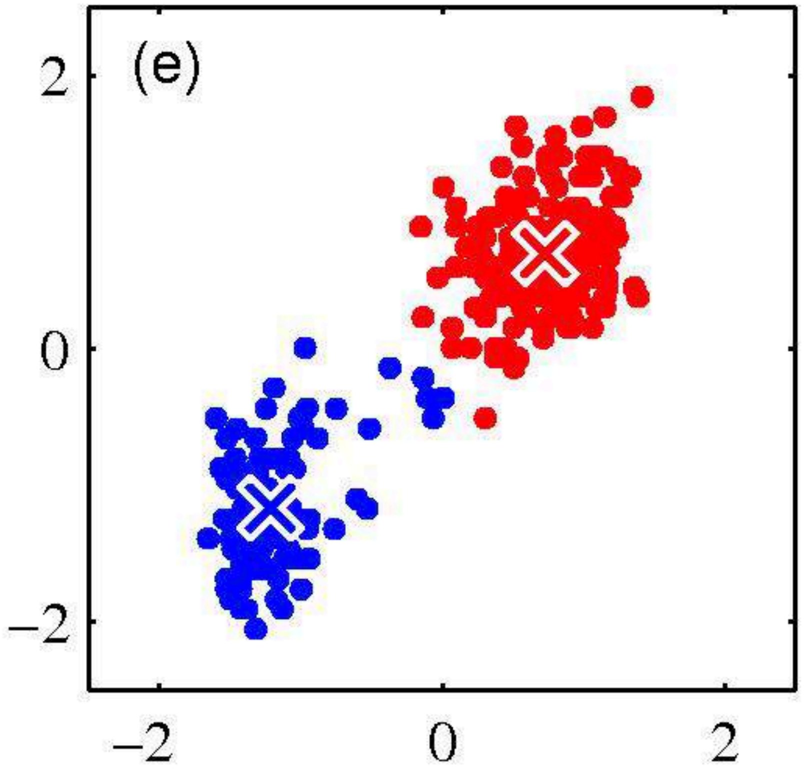
K-means clustering: Example



Repeat until convergence

Adapted from D. Sontag

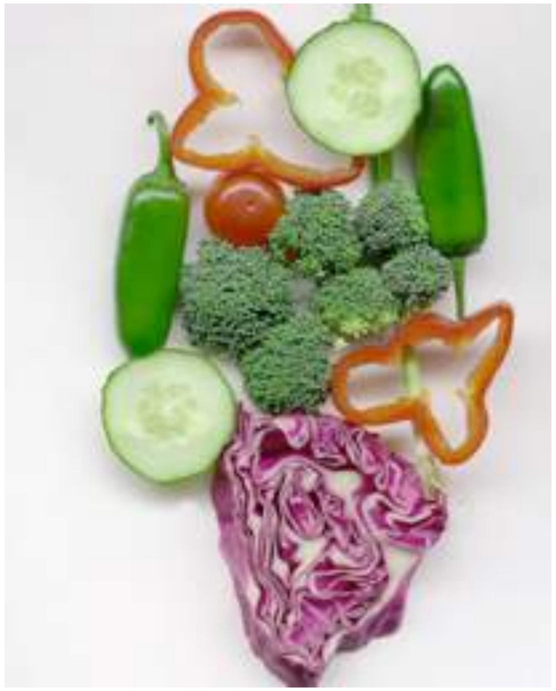
K-means clustering: Example



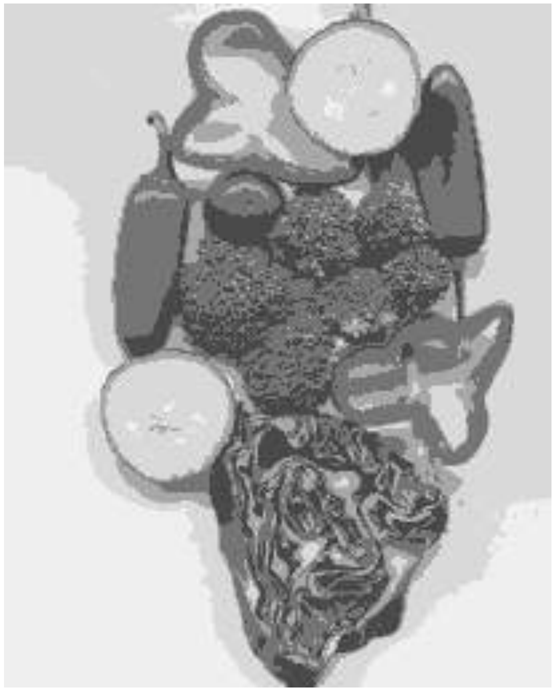
Final output

Adapted from D. Sontag

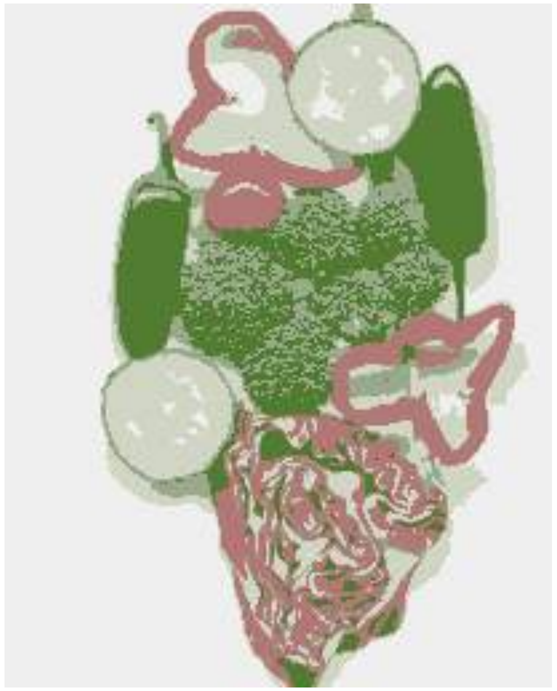
Image



Clusters on intensity



Clusters on color



K-means clustering using intensity alone and color alone

Properties of K-means

Guaranteed to converge in a finite number of steps.

Minimizes an objective function (compactness of clusters):

$$\sum_{i \in \text{clusters}} \left\{ \sum_{j \in \text{elements of i'th cluster}} \left\| x_j - \mu_i \right\|^2 \right\}$$

where μ_i is the center of cluster i.

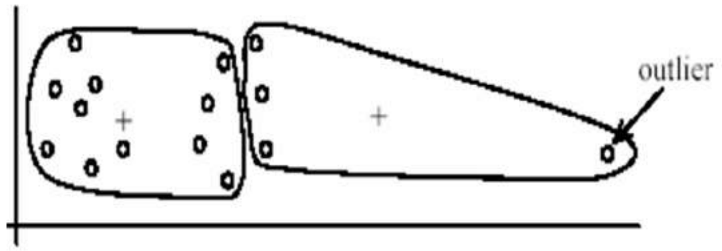
Running time per iteration:

- Assign data points to closest cluster center: $O(Kn)$ time
- Change the cluster center to the average of its points: $O(n)$ time

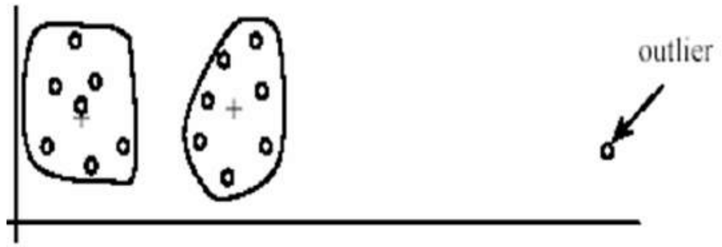
Properties of K-means

- Pros
  - Very simple method
  - Efficient

- Cons
  - Converges to a *local* minimum of the error function
  - Need to pick K
  - Sensitive to initialization
  - Sensitive to outliers
  - Only finds “spherical” clusters



(A): Undesirable clusters



(B): Ideal clusters

Supervised Learning (classification)

Classification Problems

Given :

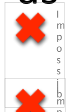
1) some "features": $f_1, f_2, \ldots., f_n$
2) some "classes": $c_1, \ldots., c_m$

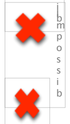
Problem :

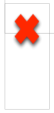
To classify an "object" according to its features

Example #1

To classify an "object" as :


= " watermelon "


= " apple "


= " orange "

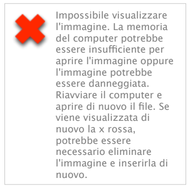
According to the following features :

f_1 = " weight "

f_2 = " color "

f_3 = " size "

Example :


Impossibile visualizzare l'immagine. La memoria del computer potrebbe essere insufficiente per aprire l'immagine oppure l'immagine potrebbe essere danneggiata. Riavviare il computer e aprire di nuovo il file. Se viene visualizzata di nuovo la x rossa, potrebbe essere necessario eliminare l'immagine e inserirla di nuovo.

$$\left.\begin{array}{l} \text{weight} = 80\text{ g} \\ \text{color} = \text{green} \\ \text{size} = 10\text{ cm}^3 \end{array}\right\} \Longrightarrow \text{“apple”}$$

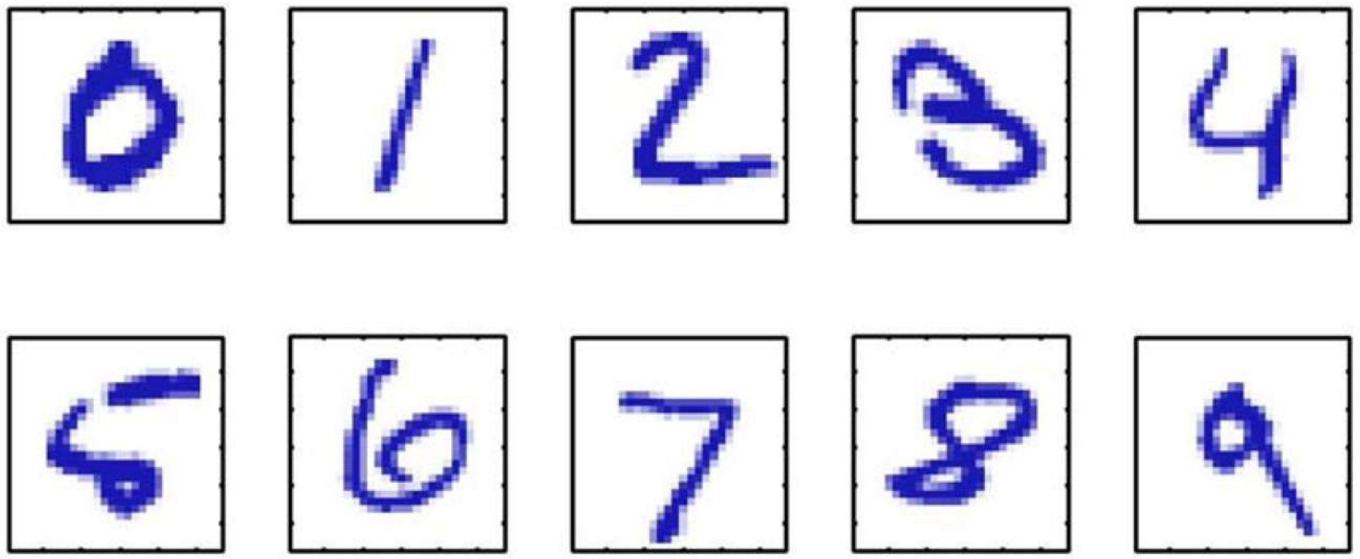
Example #2

Problem: Establish whether a patient got the flu

- Classes : { " flu " , " non-flu " }
- (Potential) Features :

f_1	:	Body temperature	
f_2	:	Headache ?	(yes / no)
f_3	:	Throat is red ?	(yes / no / medium)
f_4	:		

Example #3
Hand-written digit recognition



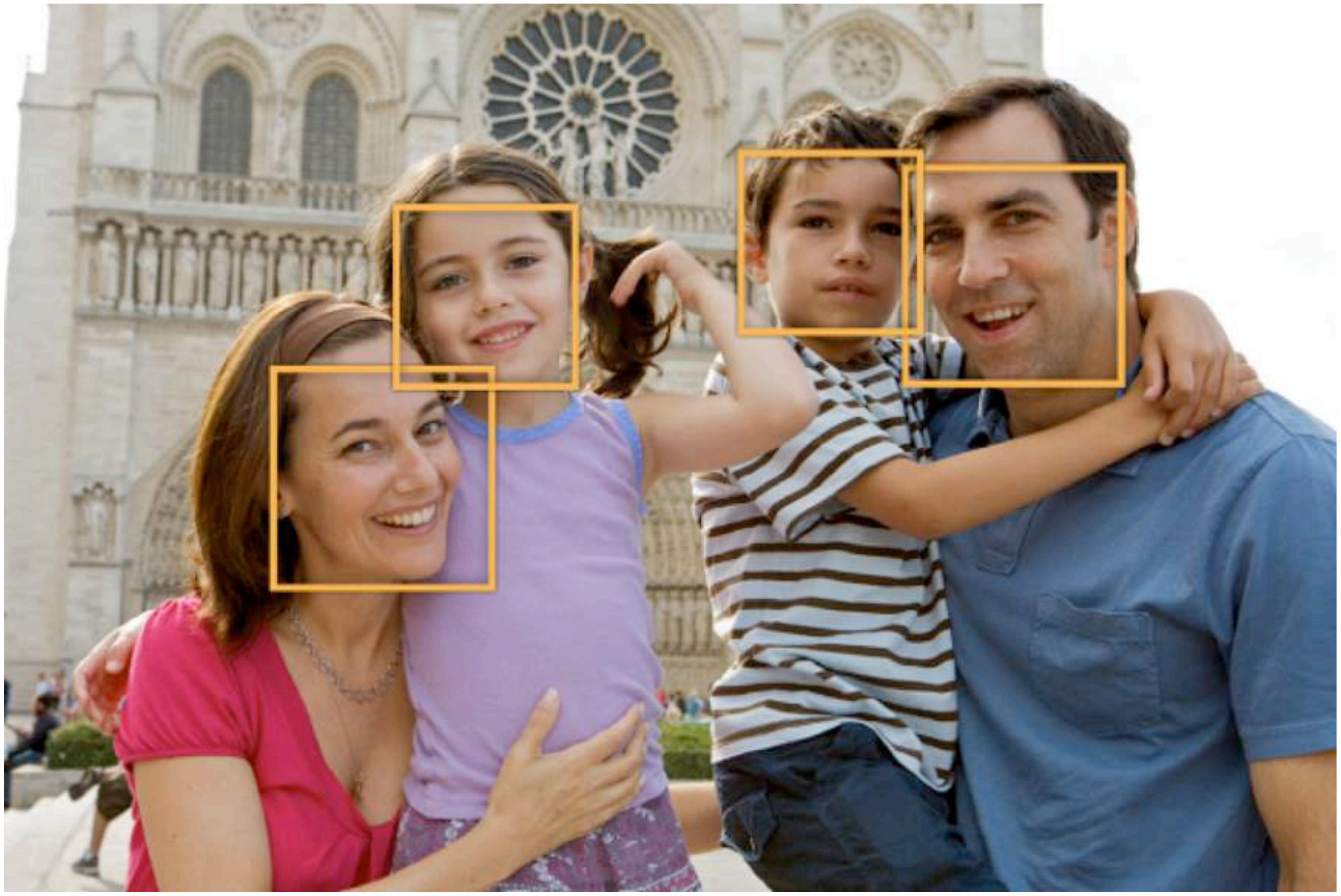
Images are 28 x 28 pixels

Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$

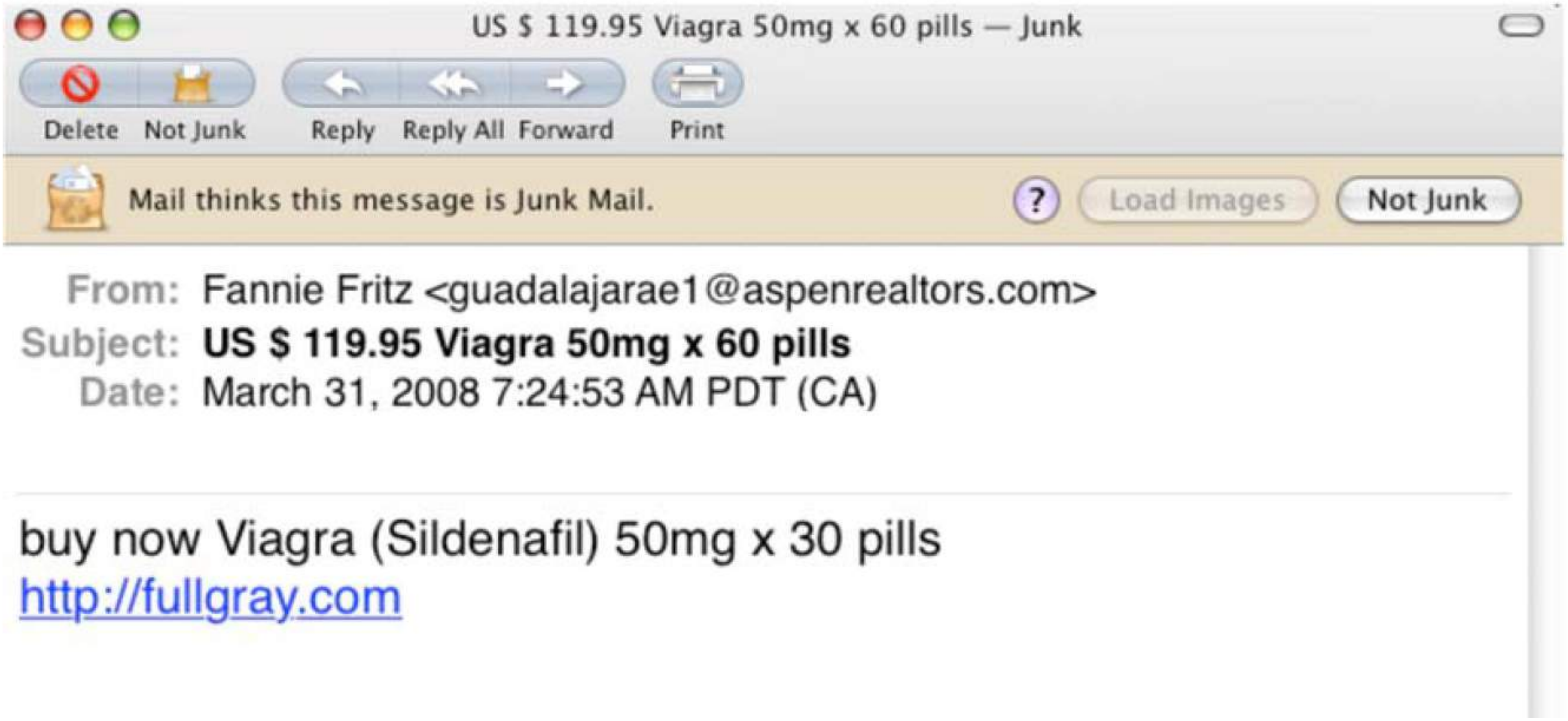
Learn a classifier $f(\mathbf{x})$ such that,

$$f : \mathbf{x} \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Example #4: Face Detection



Example #5:
Spam Detection



Mail thinks this message is Junk Mail.

From: Fannie Fritz <guadalajarae1@aspenrealtors.com>
Subject: **US $ 119.95 Viagra 50mg x 60 pills**
Date: March 31, 2008 7:24:53 AM PDT (CA)

buy now Viagra (Sildenafil) 50mg x 30 pills
http://fullgray.com

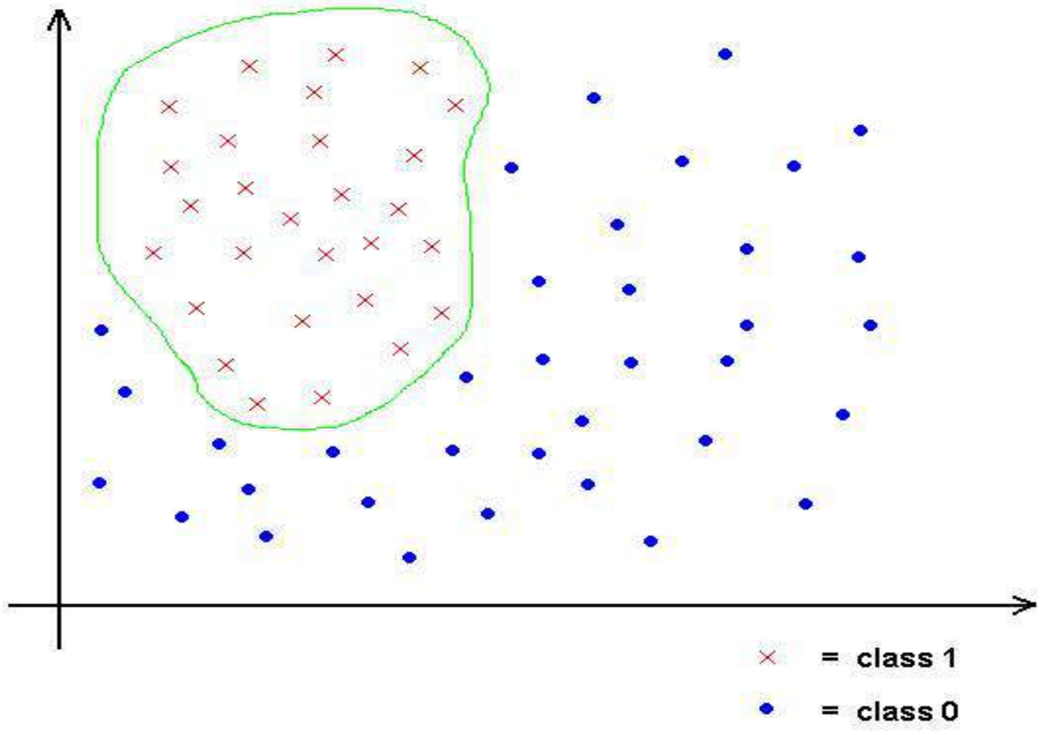
Geometric Interpretation

Example:

Classes = { 0 , 1 }

Features = x , y : both taking value in [0 , +∞ [

Idea: Objects are represented as “point” in a geometric space



The formal setup

SLT deals mainly with **supervised learning** problems.

Given:

- ✓ an input (feature) space: $\mathcal{X}$
- ✓ an output (label) space: $\mathcal{Y}$ (typically $\mathcal{Y} = \{ -1, +1 \}$)

the question of learning amounts to estimating a functional relationship between the input and the output spaces:

$$f : \mathcal{X} \to \mathcal{Y}$$

Such a mapping f is called a **classifier**.

In order to do this, we have access to some (labeled) training data:

$$(X_1, Y_1), \ldots , (X_n, Y_n) \in \mathcal{X} \times \mathcal{Y}$$

A **classification algorithm** is a procedure that takes the training data as input and outputs a classifier f.

Assumptions

In SLT one makes the following assumptions:

- ✓ there exists a joint probability distribution P on $\mathcal{X} \times \mathcal{Y}$
- ✓ the training examples (X_i, Y_i) are sampled independently from P (iid sampling).

In particular:

1. No assumptions on P
2. The distribution P is unknown at the time of learning
3. Non-deterministic labels due to label noise or overlapping classes
4. The distribution P is fixed

Losses and risks

We need to have some measure of "how good" a function f is when used as a classifier. A *loss function* measures the "cost" of classifying instance $X \in \mathcal{X}$ as $Y \in \mathcal{Y}$.

The simplest loss function in classification problems is the **0-1 loss** (or misclassication error):

$$\ell(X, Y, f(X)) = \begin{cases} 1 & \text{if } f(X) \neq Y \\ 0 & \text{otherwise.} \end{cases}$$

The *risk* of a function is the average loss over data points generated according to the underlying distribution P:

$$R(f) := E(\ell(X, Y, f(X)))$$

The *best classifier* is the one with the smallest risk $R(f)$.

Bayes classifiers

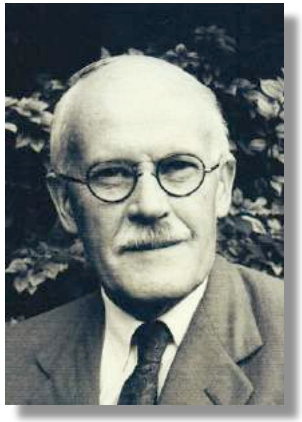
Among all possible classifiers, the “best” one is the *Bayes classifier*:

$$f_{Bayes}(x) := \begin{cases} 1 & \text{if } P(Y = 1 \mid X = x) \geq 0.5 \\ -1 & \text{otherwise.} \end{cases}$$

In practice, it is impossible to directly compute the Bayes classifier as the underlying probability distribution *P* is unknown to the learner.

The idea of estimating *P* from data doesn’t usually work …

Bayes' theorem



«[Bayes' theorem] is to the theory of probability what Pythagoras' theorem is to geometry.»

Harold Jeffreys
Scientific Inference (1931)

$$P(h \mid e) = \frac{P(e \mid h)P(h)}{P(e)} = \frac{P(e \mid h)P(h)}{P(e \mid h)P(h) + P(e \mid \neg h)P(\neg h)}$$

- ✓ $P(h)$: prior probability of hypothesis h
- ✓ $P(h \mid e)$: posterior probability of hypothesis h (in the light of evidence e)
- ✓ $P(e \mid h)$: "likelihood" of evidence e on hypothesis h

The classification problem

Given:

- ✓ a set training points $(X_1,Y_1), \ldots , (X_n,Y_n) \in \mathcal{X} \times \mathcal{Y}$ drawn iid from an *unknown* distribution P
- ✓ a loss functions

Determine a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ which has risk $R(f)$ as close as possible to the risk of the Bayes classifier.

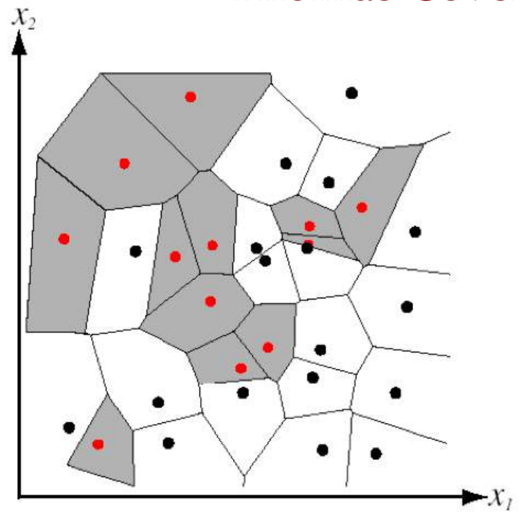
Caveat. Not only is it impossible to compute the Bayes error, but also the risk of a function f cannot be computed without knowing P.

A desperate situation?

An example:
The nearest neighbor (NN) rule

«Early in 1966 when I first began teaching at Stanford, a student, Peter Hart, walked into my office with an interesting problem. He said that Charles Cole and he were using a pattern classification scheme which, for lack of a better word, they described as the **nearest neighbor procedure**.
This scheme assigned to an as yet unclassified observation the classification of the nearest neighbor. Were there any good theoretical properties of this procedure?»

Thomas Cover (1982)



How good is the NN rule?

Cover and Thomas showed that:

$$R(f_{Bayes}) \le R_\infty \le 2R(f_{Bayes})$$

where R_∞ denotes the expected error rate of NN when the sample size tends to infinity.

We cannot say anything stronger as there are probability distributions for which the performance of the NN rule achieves either the upper or lower bound.

Variations:

- ✓ ***k*-NN rule:** use the k nearest neighbors and take a majority vote
- ✓ **k_n-NN rule:** the same as above, for k_n growing with n

Theorem (Stone, 1977) If $n \to \infty$ and $k \to \infty$, such that $k/n \to 0$, then for all probability distributions $R(k_n\text{-NN}) \to R(f_{Bayes})$ (that is, the k_n-NN rule is "universally Bayes consistent").

Back-Propagation Neural Networks

History

Early work (1940-1960)

- McCulloch & Pitts (Boolean logic)
- Rosenblatt (Learning)
- Hebb (Learning)

Transition (1960-1980)

- Widrow – Hoff (LMS rule)
- Anderson (Associative memories)
- Amari

Resurgence (1980-1990's)

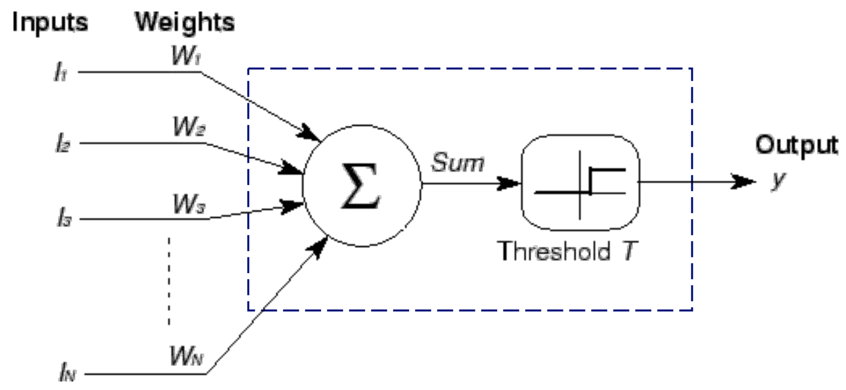
- Hopfield (Ass. mem. / Optimization)
- Rumelhart et al. (Back-prop)
- Kohonen (Self-organizing maps)
- Hinton , Sejnowski (Boltzmann machine)

New resurgence (2012 -)

- CNNs, Deep learning, GAN's ⋯.

The McCulloch and Pitts Model (1943)

The McCulloch-Pitts (MP) Neuron is modeled as a binary threshold unit

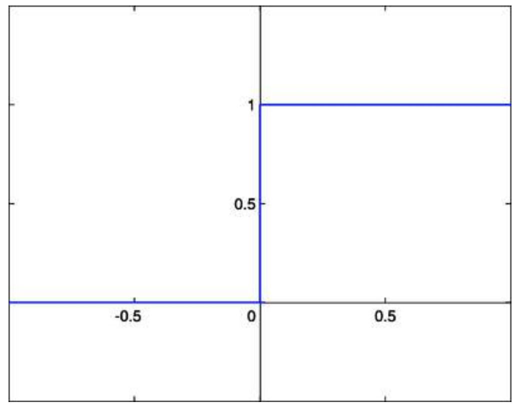


The unit "fires" if the **net input** $\sum_j w_j I_j$ reaches (or exceeds) the unit's threshold *T*:

$$y = g\left(\sum_j w_j I_j - T\right)$$

If neuron is firing, then its output y is 1, otherwise it is 0.

g is the unit step function: $g(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$

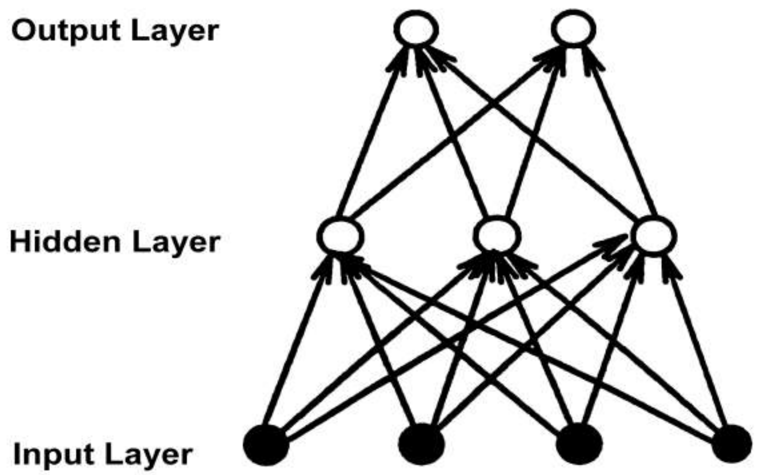


Weights w_{ij} represent the strength of the synapse between neuron *j* and neuron *i*

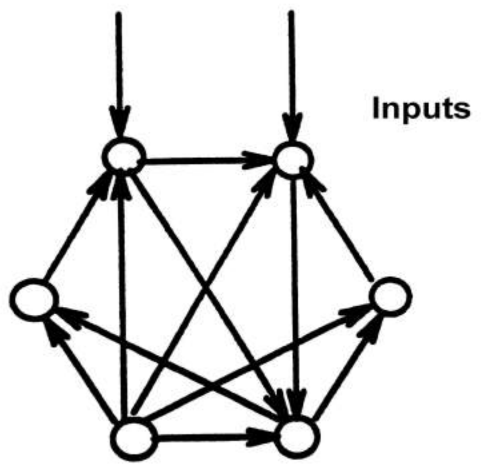
Network Topologies and Architectures

- Feedforward only *vs.* Feedback loop (Recurrent networks)
- Fully connected *vs.* sparsely connected
- Single layer *vs.* multilayer

Multilayer perceptrons, Hopfield networks,
Boltzman machines, Kohonen networks, …



(a)



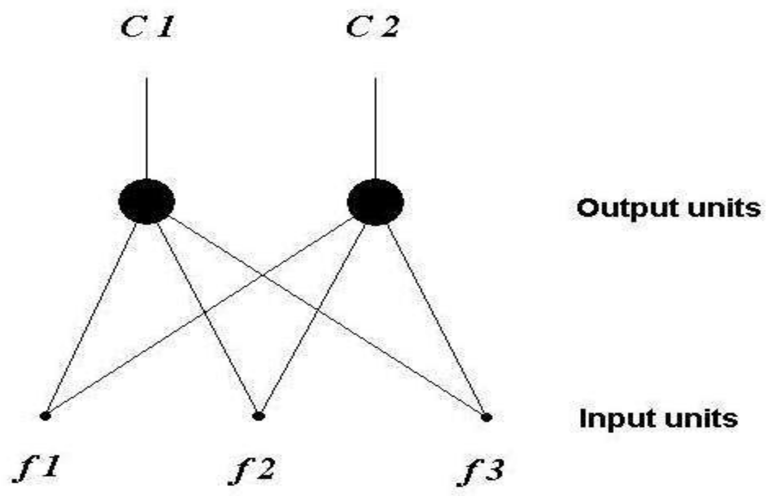
(b)

Neural Networks for Classification

A neural network can be used as a classification device .

Input	≡	features values
Output	≡	class labels

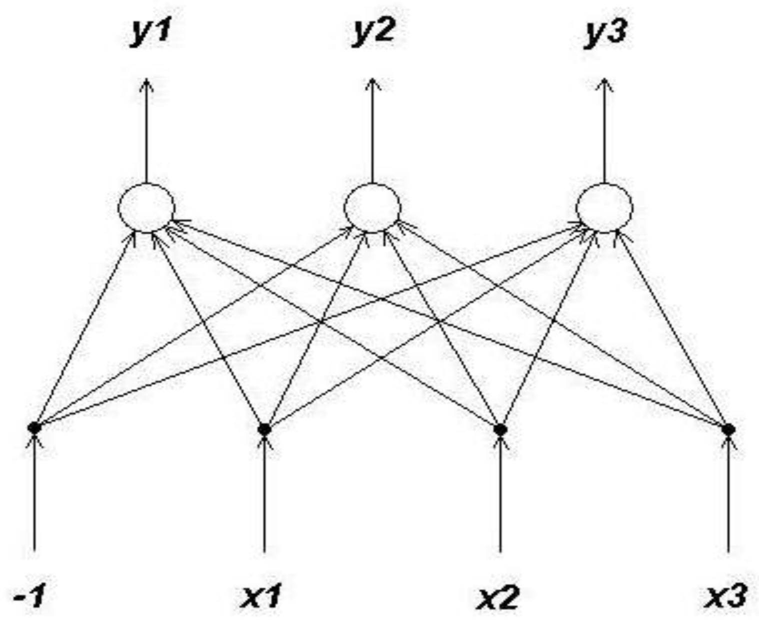
Example : 3 features , 2 classes



Thresholds

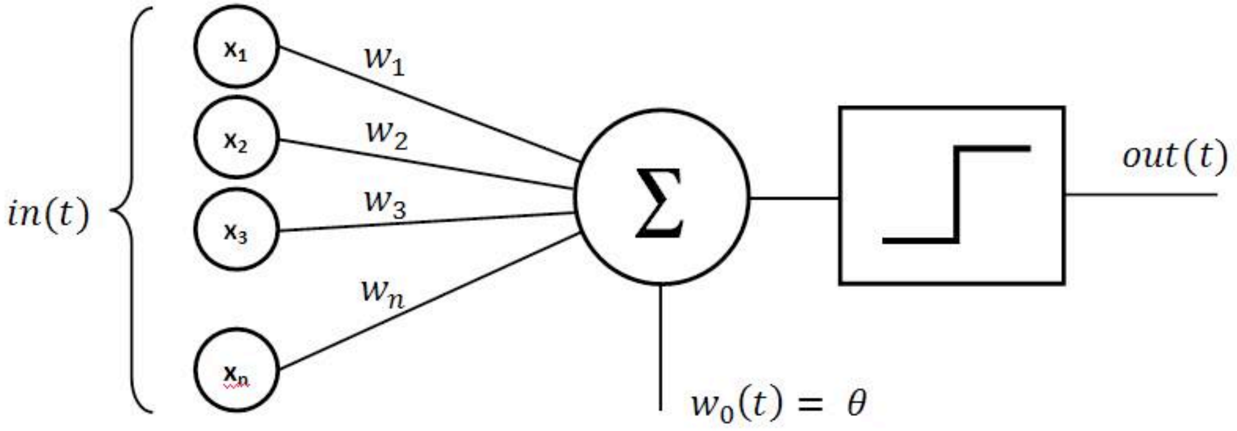
We can get rid of the thresholds associated to neurons by adding an extra unit **permanently clamped at -1** (or +1).

In so doing, thresholds become weights and can be adaptively adjusted during learning.



The Perceptron

A network consisting of one layer of M&P neurons connected in a feedforward way (i.e. no lateral or feedback connections).

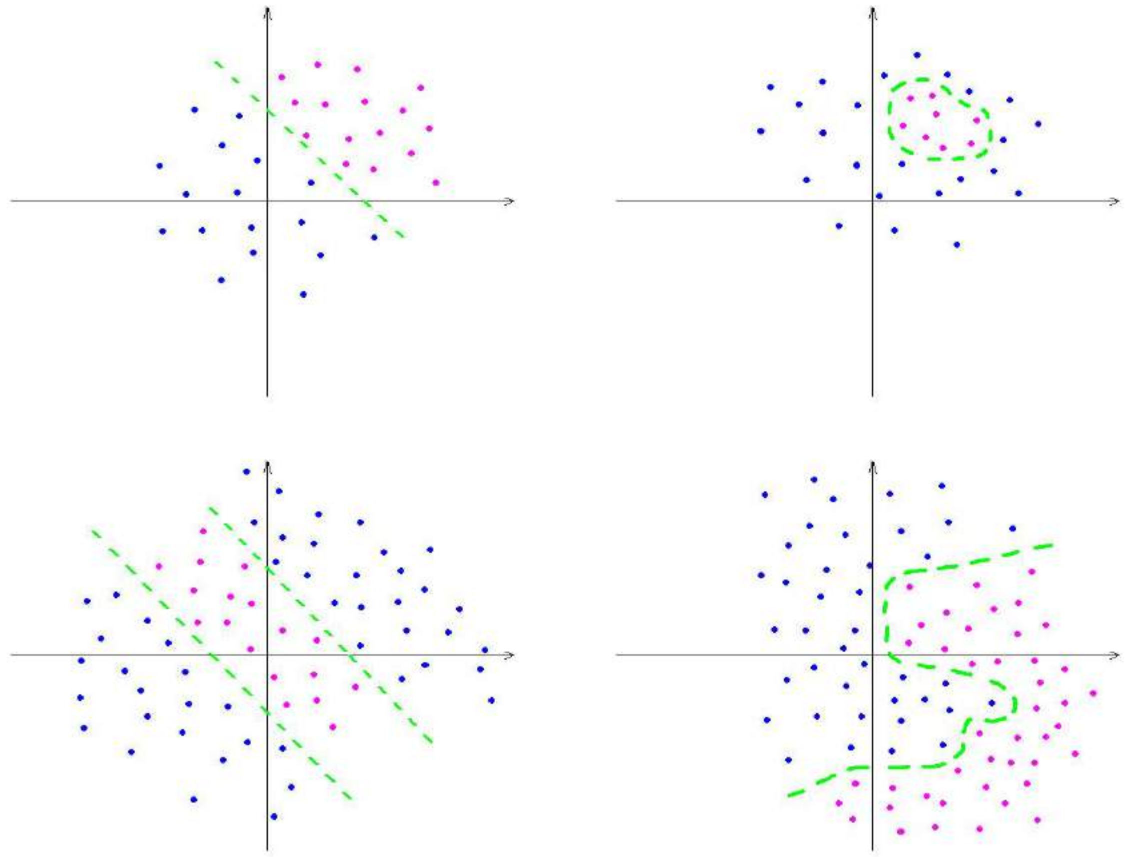


- Discrete output (+1 / -1)
- Capable of “learning” from examples (Rosenblatt)
- They suffer from serious computational limitations

Decision Regions

It’s an area wherein all examples of one class fall.

Examples:

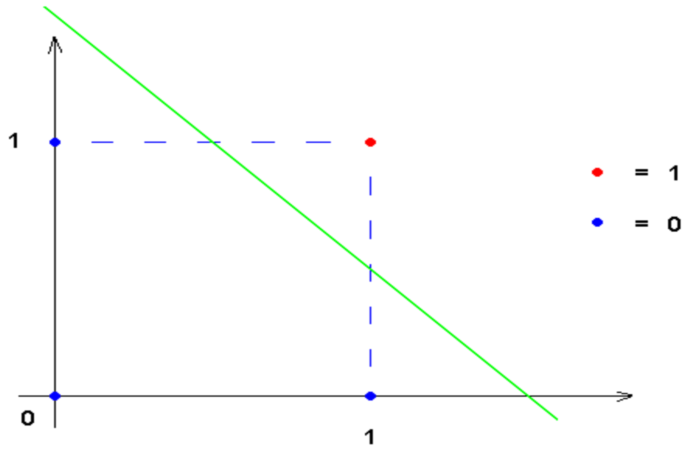


Linear Separability

A classification problem is said to be **linearly separable** if the decision regions can be separated by a hyperplane.

Example: AND

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

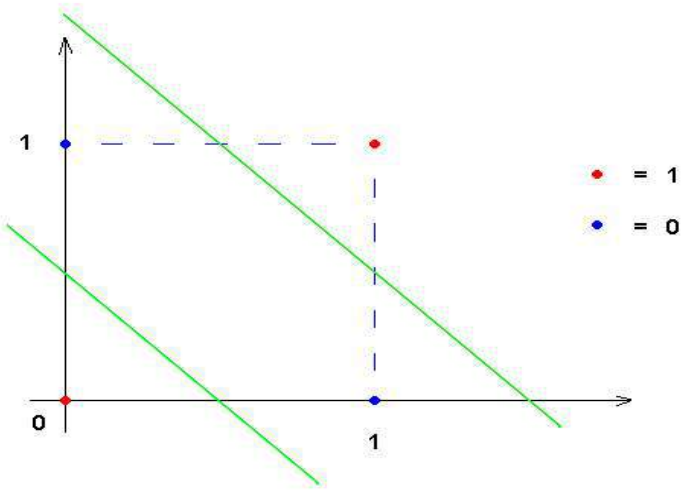


Limitations of Perceptrons

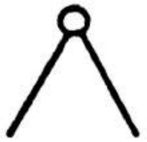
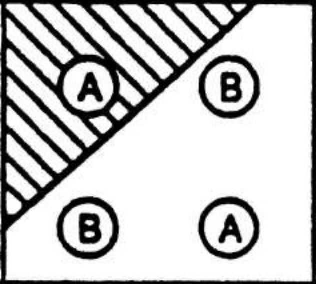
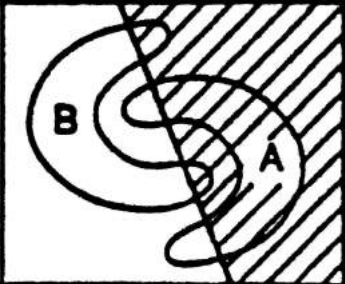
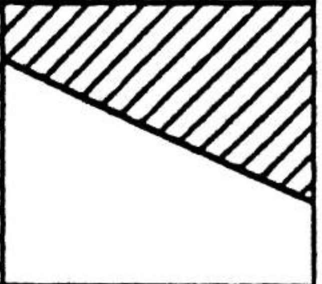
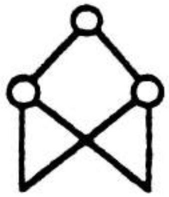
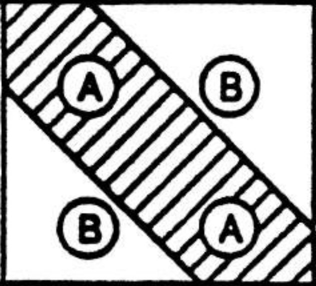
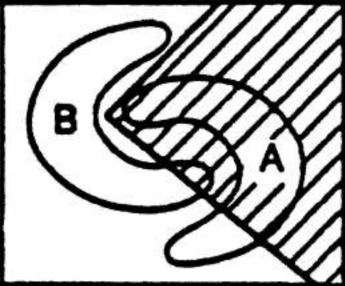
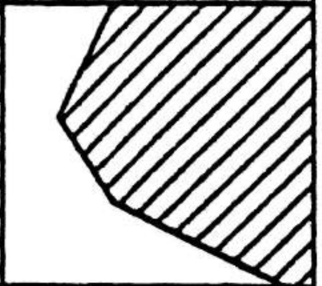
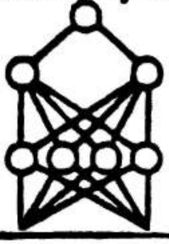
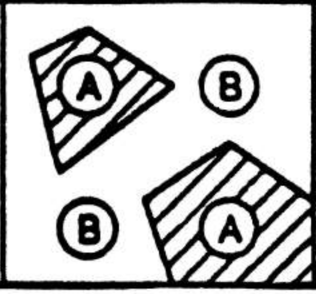
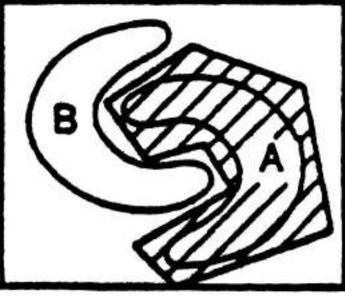
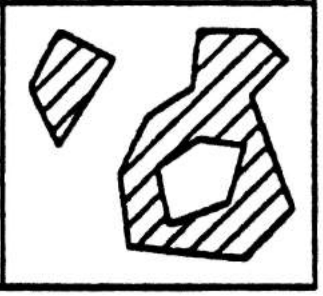
It has been shown that perceptrons can only solve linearly separable problems.

Example: XOR (exclusive OR)

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0



A View of the Role of Units

Structure	Type of Decision Regions	Exclusive-OR Problem	Classes with Meshed Regions	Most General Region Shapes
Single-layer 	Half plane bounded by hyperplane			
Two-layers 	Convex open or closed regions			
Three-layers 	Arbitrary (Complexity limited by number of nodes)			

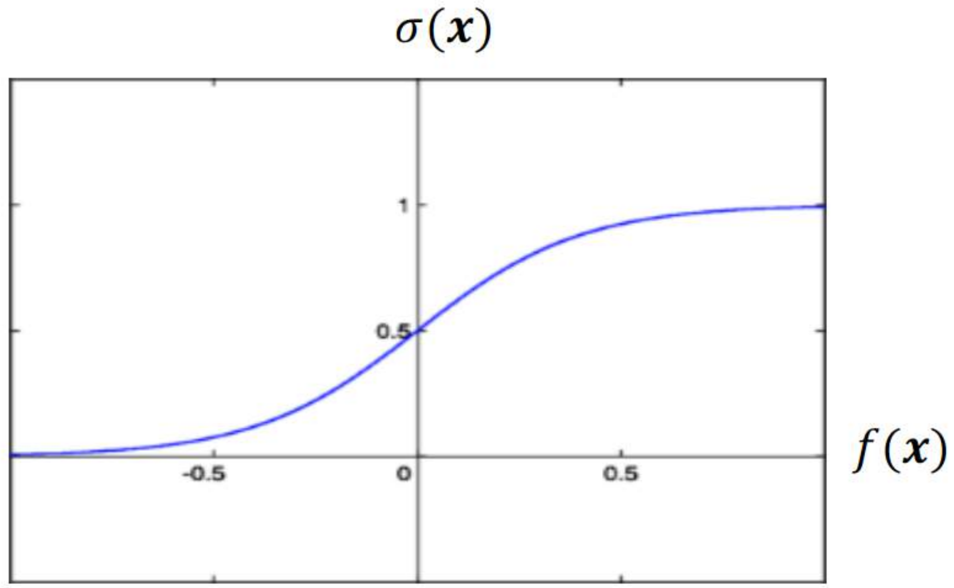
Multi–Layer Feedforward Networks

- Limitation of simple perceptron: can implement only linearly separable functions
- Add " hidden" layers between the input and output layer. A network with just one hidden layer can represent any Boolean functions including XOR
- Power of multilayer networks was known long ago, but algorithms for training or learning, e.g. back-propagation method, became available only recently (invented several times, popularized in 1986)
- **Universal approximation power:** Two-layer network can approximate any smooth function (Cybenko, 1989; Funahashi, 1989; Hornik, et al.., 1989)
- Static (no feedback)

Continuous-Valued Units

Sigmoid (or logistic)

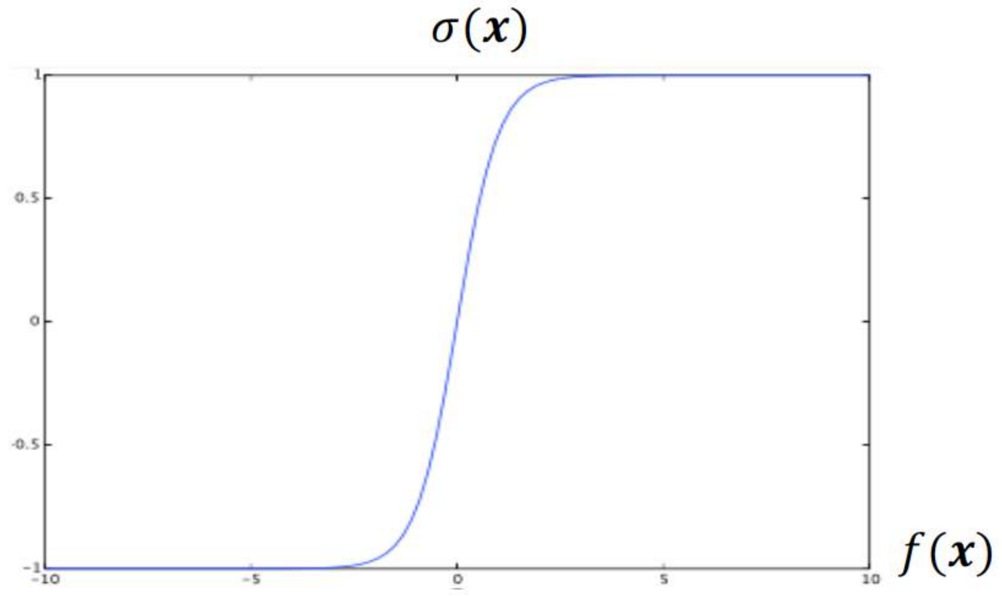
$$\sigma(\boldsymbol{x}) = \frac{1}{1 + e^{-f(\boldsymbol{x})}} \in (0,1)$$



Continuous-Valued Units

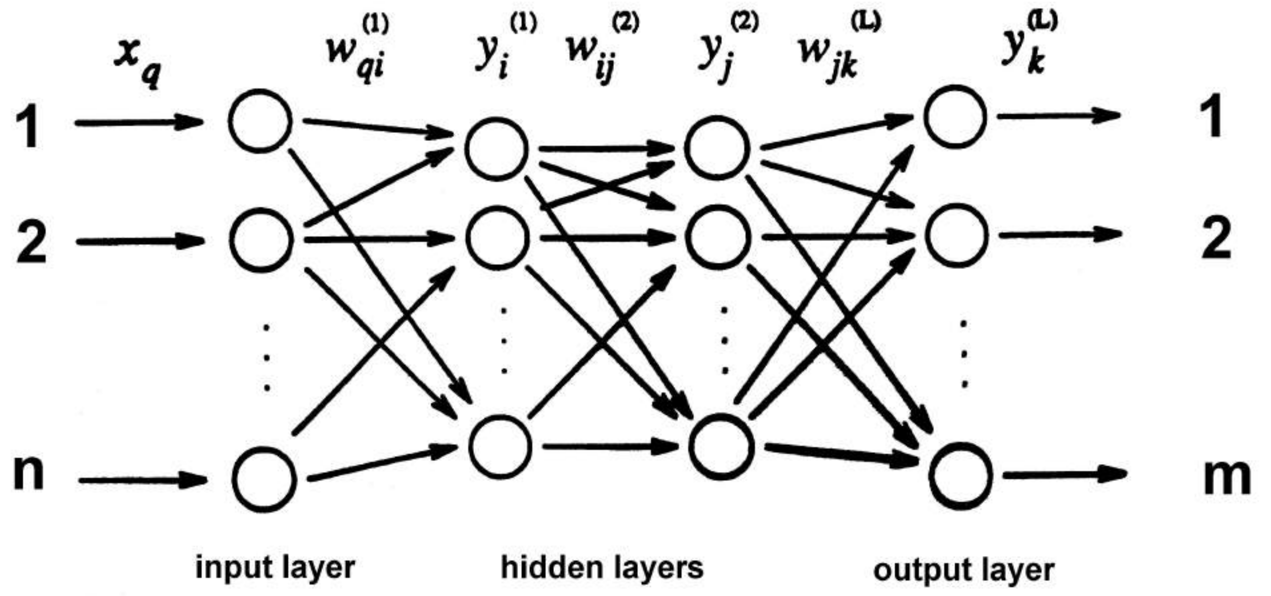
Hyperbolic tangent

$$\sigma(\boldsymbol{x}) = \frac{e^{f(\boldsymbol{x})} - e^{-f(\boldsymbol{x})}}{e^{f(\boldsymbol{x})} + e^{-f(\boldsymbol{x})}} \in (-1,1)$$



Back-propagation Learning Algorithm

- An algorithm for learning the weights in a feed-forward network, given a training set of input-output pairs
- The algorithm is based on gradient descent method.



Supervised Learning

Supervised learning algorithms require the presence of a "teacher" $\rightarrow$ who provides the right answers to the input questions.

Technically, this means that we need a **training set** of the form

$$L = \left\{ \left(\mathrm{x}^1, \mathrm{y}^1\right), \quad \ldots\ldots \quad \left(\mathrm{x}^p, \mathrm{y}^p\right) \right\}$$

where :

$\mathrm{x}^\mu \quad (\mu = 1 \ldots p)$ is the network input vector

$\mathrm{y}^\mu \quad (\mu = 1 \ldots p)$ is the **desired** network output vector

Supervised Learning

The learning (or training) phase consists of determining a configuration of weights in such a way that the network output be as close as possible to the desired output, for all the examples in the training set.

Formally, this amounts to minimizing an **error function** such as (not only possible one):

$$E = \frac{1}{2} \sum_{\mu} \sum_{k} \left(y_k^{\mu} - O_k^{\mu} \right)^2$$

where O_k^{μ} is the output provided by the output unit k when the network is given example μ as input.

Back-Propagation

To minimize the error function *E* we can use the classic **gradient-descent** algorithm:

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E_k}{\partial w_{ji}}$$

η = "learning rate"

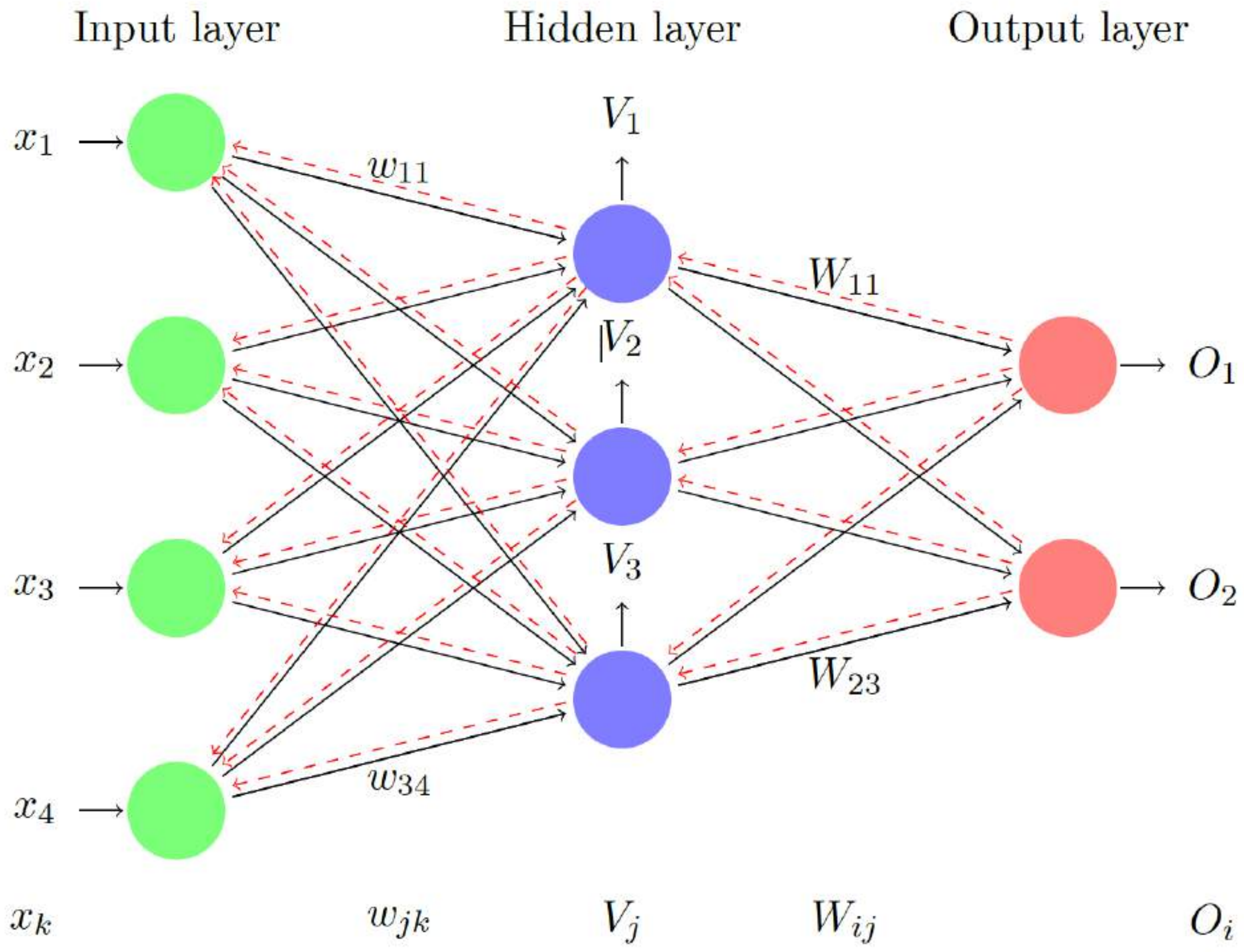
To compute the partial derivates we use the **error back propagation** algorithm.

It consists of two stages:

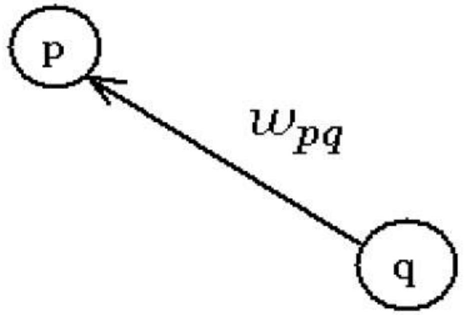
Forward pass : the input to the network is propagated layer after layer in forward direction

Backward pass : the "error" made by the network is propagated backward, and weights are updated properly

Error Back-Propagation



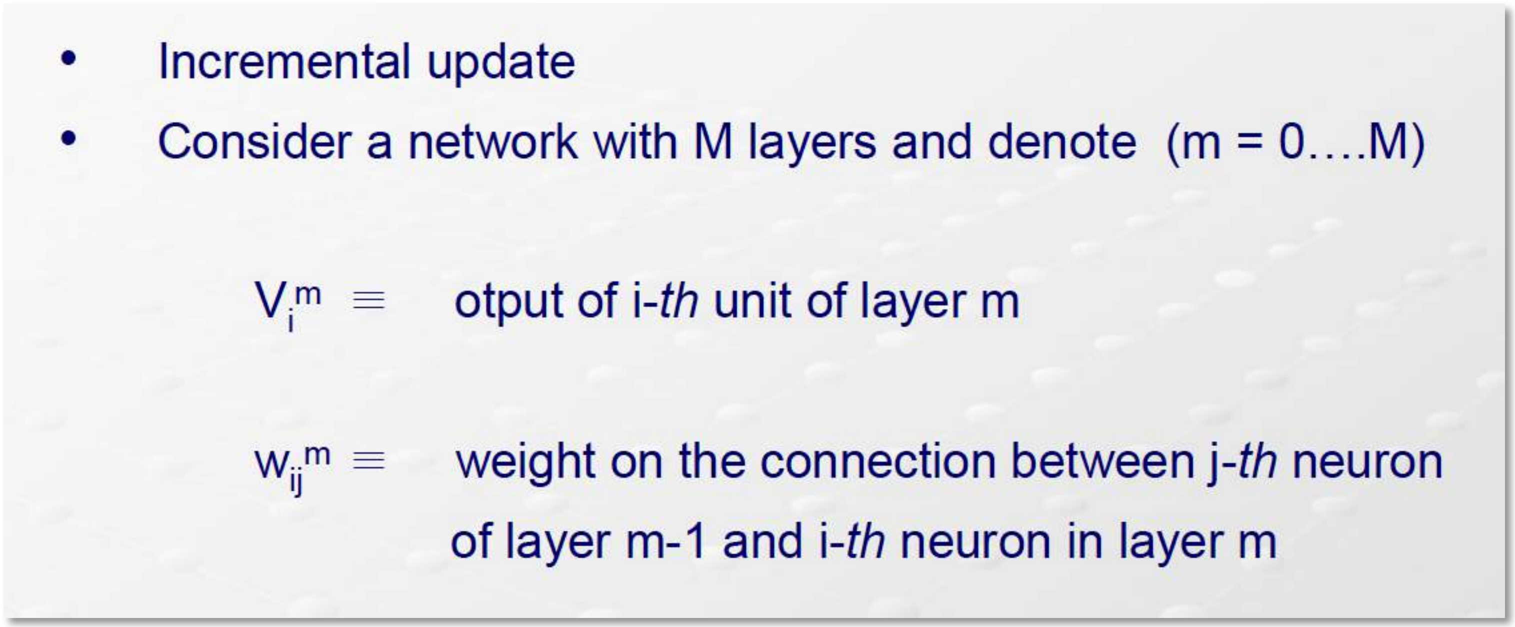
Locality of Back-Prop



$$\Delta\omega_{pq} = \eta \sum_{\mu} \delta_p^{\mu} V_q^{\mu}$$ off - line

$$\Delta\omega_{pq} = \eta\, \delta_p^{\mu} V_q^{\mu}$$ on - line

The Back-Propagation Algorithm



- Incremental update
- Consider a network with M layers and denote (m = 0….M)

$V_i^m \equiv$ otput of i-*th* unit of layer m

$w_{ij}^m \equiv$ weight on the connection between j-*th* neuron of layer m-1 and i-*th* neuron in layer m

The Back-Propagation Algorithm

1. Initialize the weight to (small) random values
2. Choose a pattern $\overline{x}^{\mu}$ and apply it to the input layer (m=0)

$$V_k^0 = x_k^{\mu} \qquad \forall k$$

3. Propagate the signal forward:

$$V_i^m = g\left(h_i^m\right) = g\left(\sum_j w_{ij} V_j^{m-1}\right)$$

4. Compute the δ's for the output layer:

$$\delta_i^M = g'\left(h_i^M\right)\left(y_i^M - V_i^M\right)$$

5. Compute the δ's for all preceding layers:

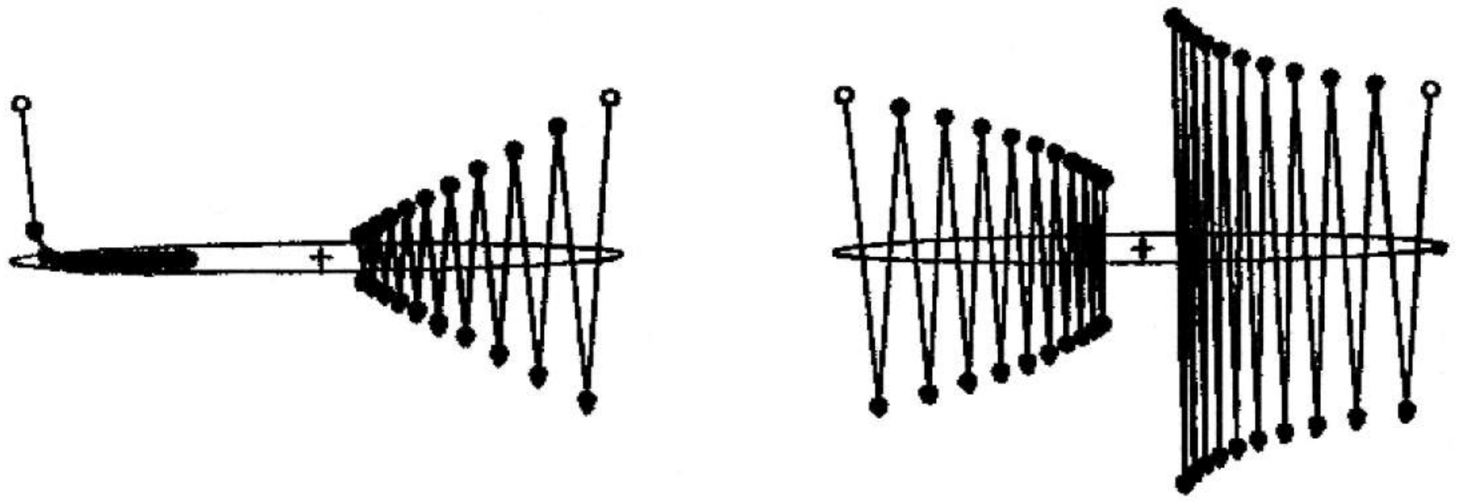
$$\delta_i^{m-1} = g'\left(h_i^{m-1}\right)\sum_j w_{ji}^m \delta_j^m$$

6. Update connection weights:

$$w_{ij}^{NEW} = w_{ij}^{OLD} + \Delta w_{ij} \qquad where \qquad \Delta w_{ij} = \eta\, \delta_i^m V_j^{m-1}$$

7. Go back to step 2 until convergence

The Role of the Learning Rate



Gradient descent on a simple quadratic surface (the left and right parts are copies of the same surface). Four trajectories are shown, each for 20 steps from the open circle. The minimum is at the + and the ellipse shows a constant error contour. The only significant difference between the trajectories is the value of *η*, which was 0.02, 0.0476, 0.049, and 0.0505 from left to right.

The Momentum Term

Gradient descent may:

- Converge too slowly if η is too small
- Oscillate if η is too large

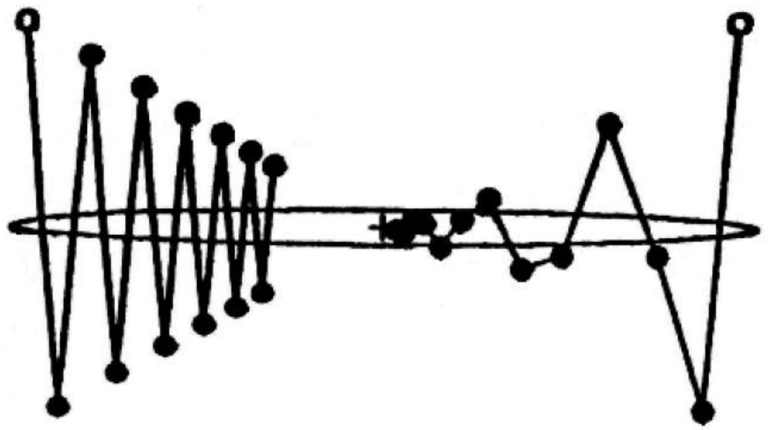
Simple remedy:

$$\Delta\omega_{pq}(t+1) = -\eta \frac{\partial E}{\partial w_{pq}} + \underbrace{\alpha\, \Delta w_{pq}(t)}_{momentum}$$

The momentum term allows us to use large values of η thereby avoiding oscillatory phenomena

Typical choice: $\alpha = 0.9, \eta = 0.5$

The Momentum Term



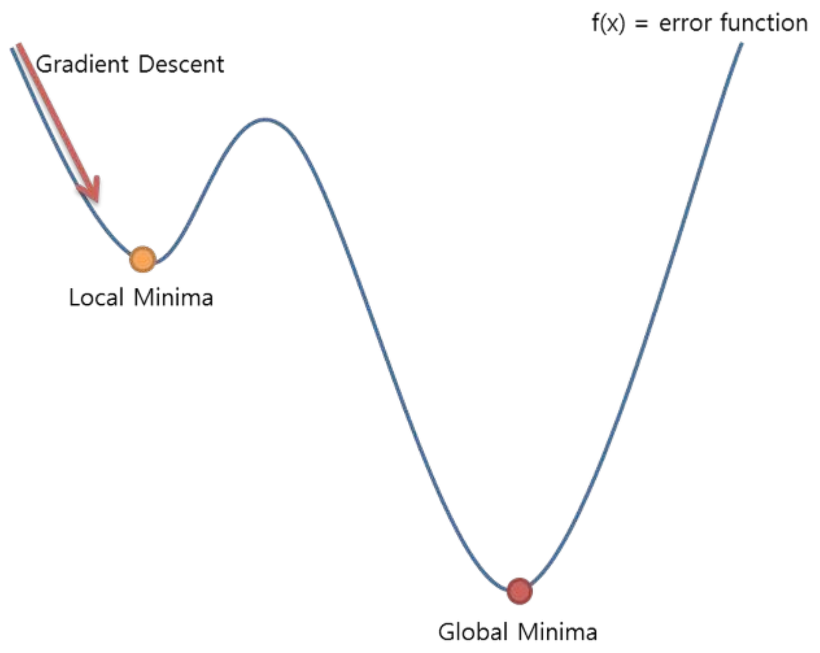
Gradient descent on the simple quadratic surface. Both trajectories are for 12 steps with $\eta = 0.0476$, the best value in the absence of momentum. On the left there is no momentum ($\alpha = 0$), while $\alpha = 0.5$ on the right.

The Problem of Local Minima

Back-prop **cannot avoid local minima**.

Choice of initial weights is important.

If they are too large the nonlinearities tend to saturate since the beginning of the learning process.



Heuristic $\Rightarrow$ Choose initial weights as $w_{ij} \cong 1 \big/ \sqrt{k_i}$

where k_i is the number of units that feed unit i (the "fan-in" of i)

Theoretical / Practical Questions

- How many layers are needed for a given task?
- How many units per layer?
- To what extent does representation matter?
- What do we mean by generalization?
- What can we expect a network to generalize?
  - Generalization: performance of the network on data not included in the training set
  - Size of the training set: how large a training set should be for "good" generalization?
  - Size of the network: too many weights in a network result in poor generalization

True *vs* Sample Error

Definition: The **true error** (denoted $error_{\mathcal{D}}(h)$) of hypothesis h with respect to target function f and distribution $\mathcal{D}$, is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$

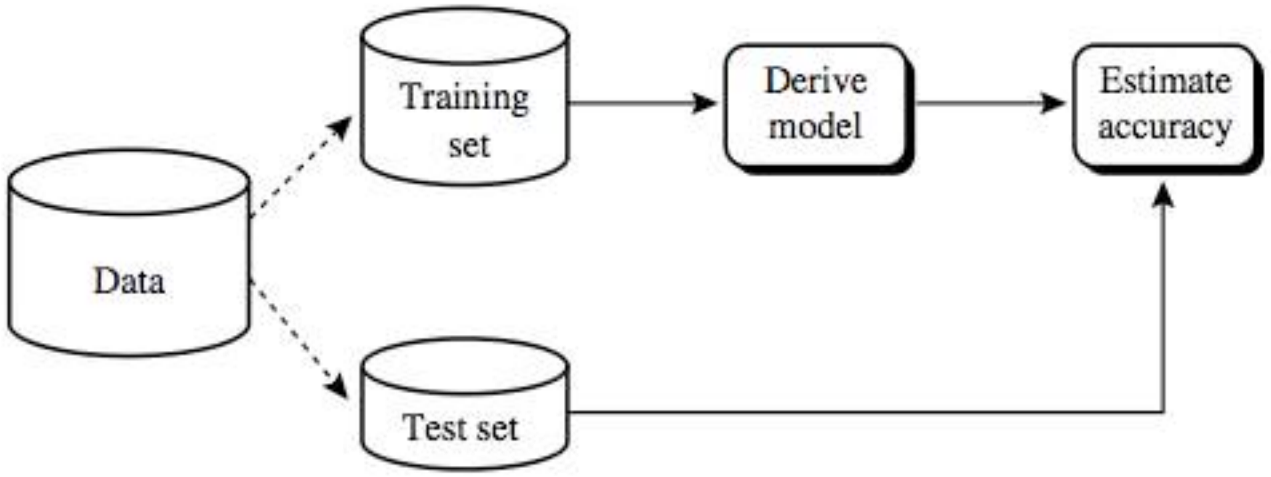
Definition: The **sample error** (denoted $error_S(h)$) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

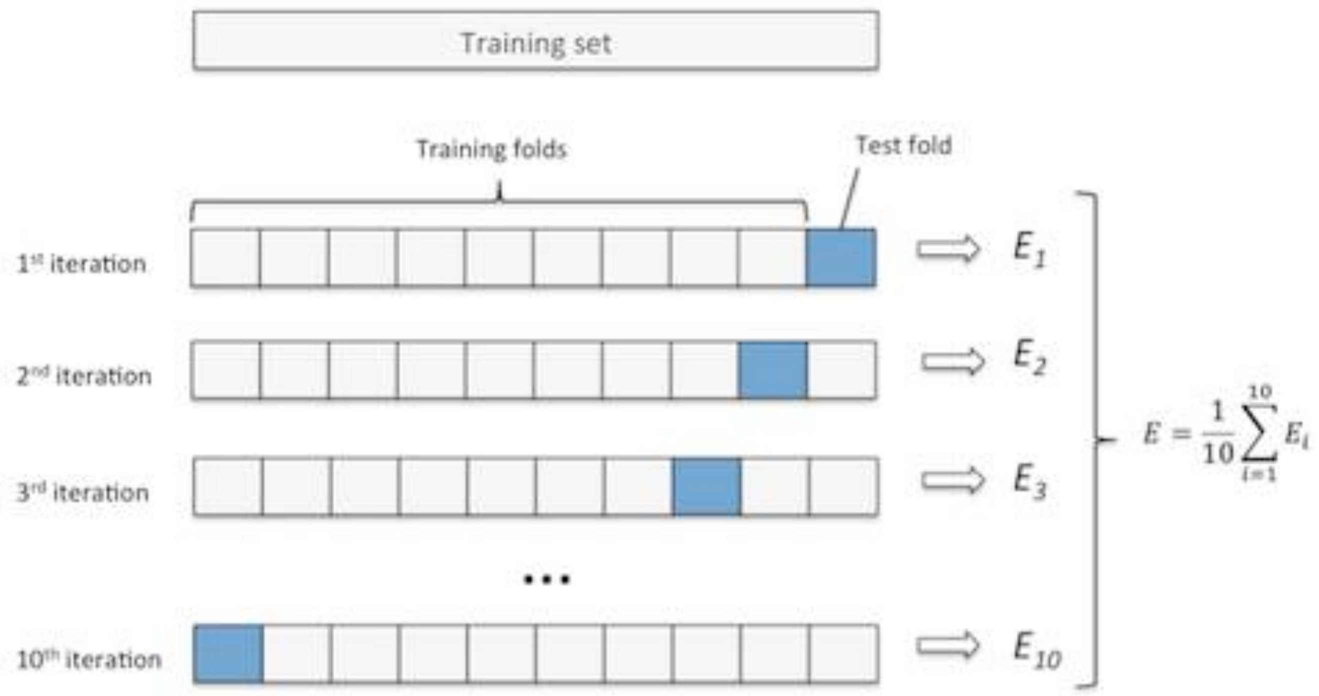
Where n is the number of examples in S, and the quantity $\delta(f(x), h(x))$ is 1 if $f(x) \neq h(x)$, and 0 otherwise.

The **true error** is unknown (and will remain so forever…).
On which sample should I compute the **sample error**?

Training *vs* Test Set

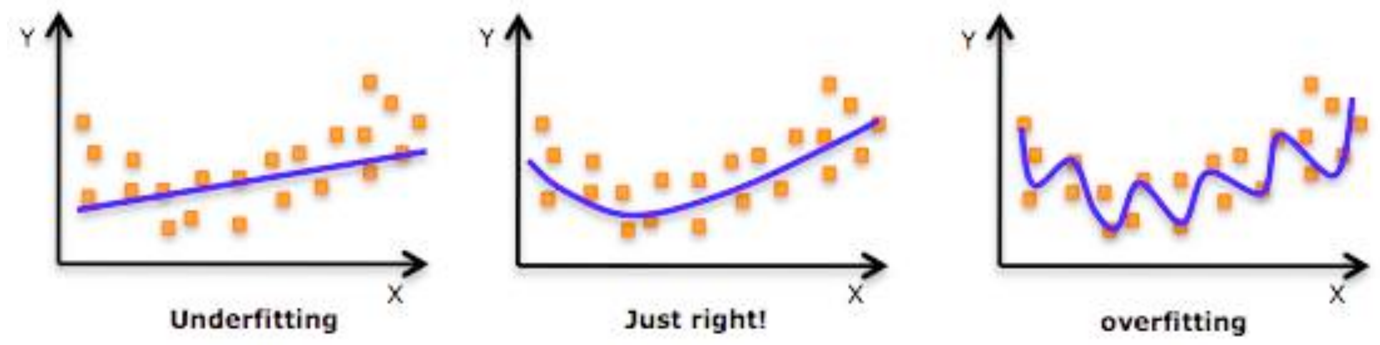


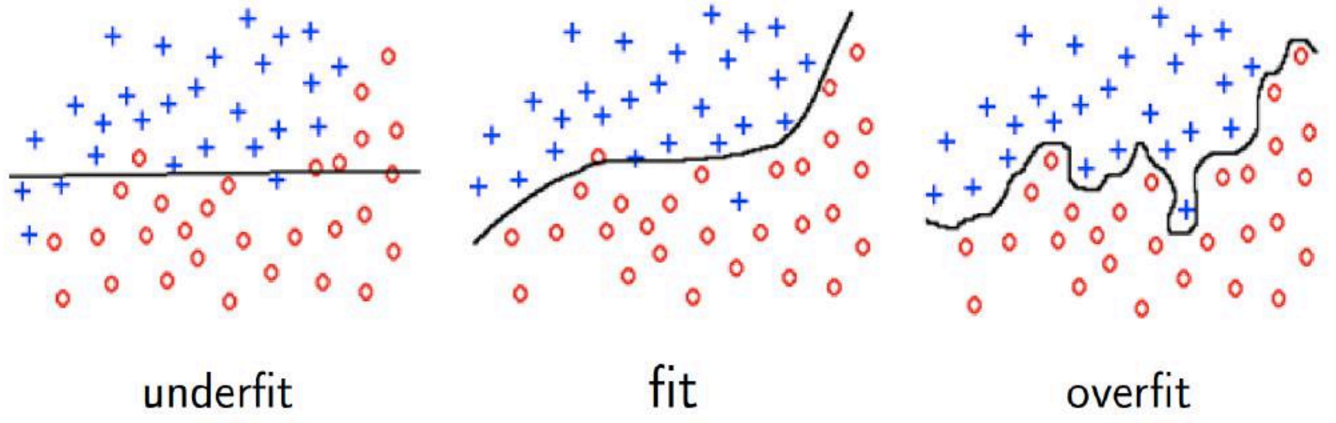
Cross-validation



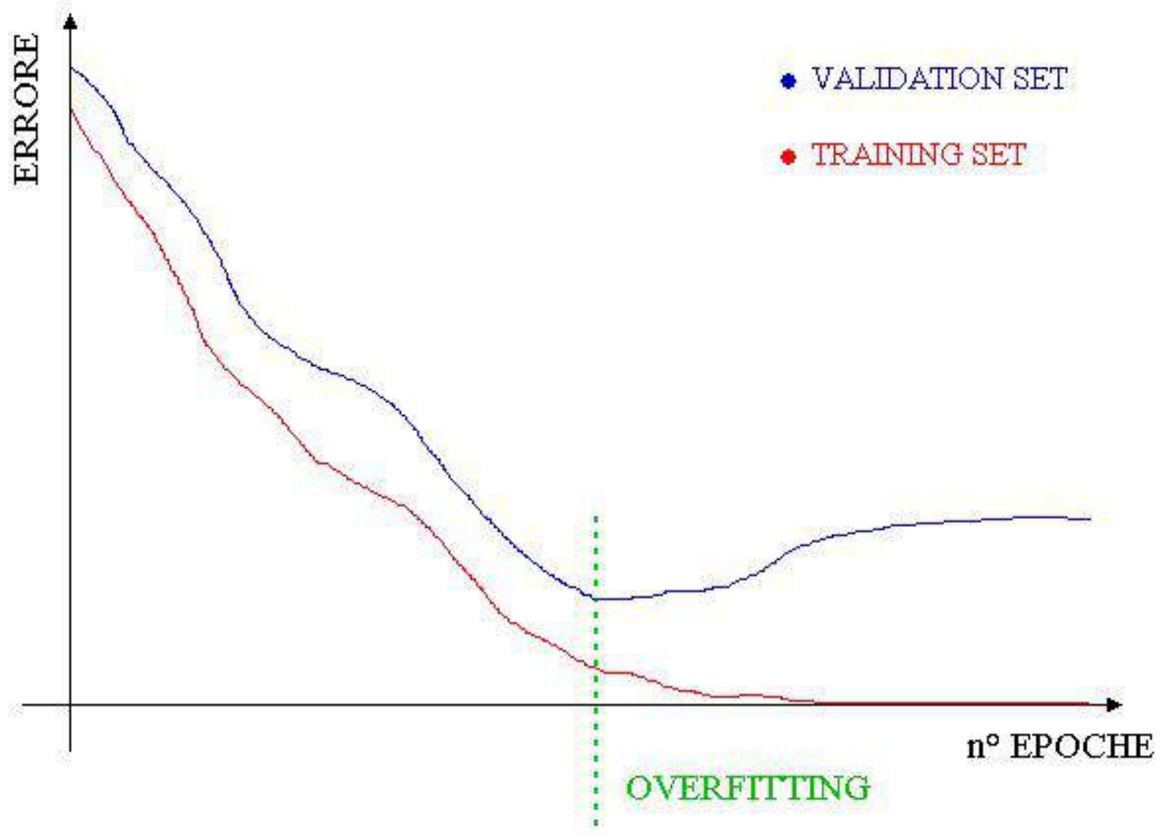
Leave-one-out: using as many test folds as there are examples (size of test fold = 1)

Model selection





Early Stopping



Size Matters

- The size (i.e. the number of hidden units and the number of weights) of an artificial neural network affects both its functional capabilities and its generalization performance

- Small networks could not be able to realize the desired input / output mapping

- Large networks lead to poor generalization performance