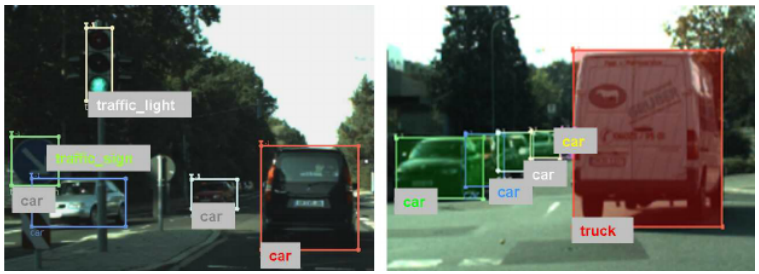
Master 2 - Image Mining Course

Institut Polytechnique de Paris - Université Paris-Saclay

Image Classification and Visual Machine Learning

Antoine Manzanera - ENSTA Paris



Objectives of the lecture

This lecture is an introduction to visual learning through a partial selection of classification techniques and learning models, that are useful / popular in computer vision. The aim is to address the following key concepts:

- Dimension reduction of descriptors
- "Low dimension" machine learning
- Convolutional Neural Networks

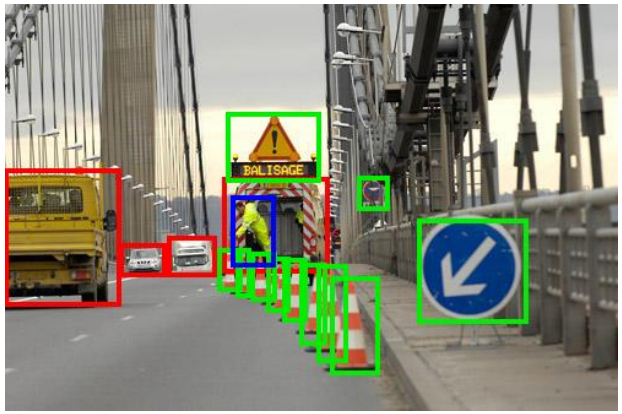
Outline of the lecture

Images and Machine Learning

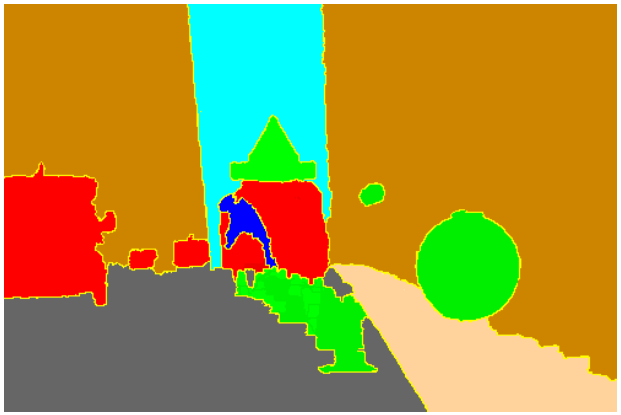
- In computer vision, *decision* rarely relies on a purely analytical and imperative approach; it is generally based on *prior knowledge* that is represented within the system's memory.
- This memory was acquired by the *experience* (or the *training*) undergone by the system *before* (offline learning) and / or *during* (online learning) its activation.
- The training is carried out from *exemplary* data for which the expected decision can be provided (*supervised* learning) or absent (*unsupervised* learning).

Which decisions in Computer Vision?

Classification



Detection = Localisation + Classification



Semantic segmentation

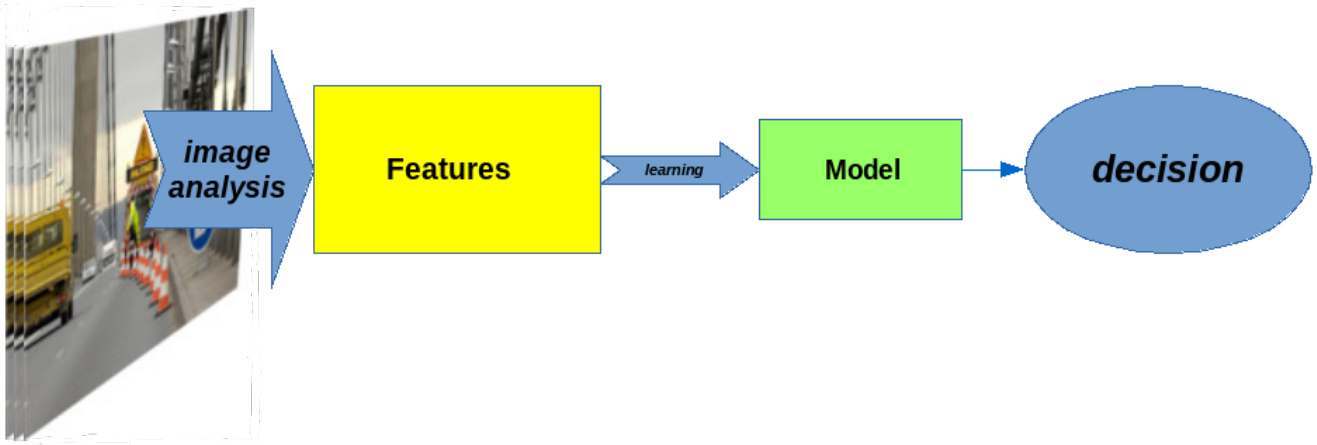
Automatic Captioning

Classification *vs* Regression

- Learning can be modelled as the construction of a *function* from an *observation space* (data extracted from the image) to a *decision space*.
- The decision space is often discrete and devoid of metrics (set of . . . classes): we then speak of *Classification*.
- The decision space can sometimes be continuous and structured (for example an image, in a restoration or super-resolution problem . . .), so we speak of *Regression*.

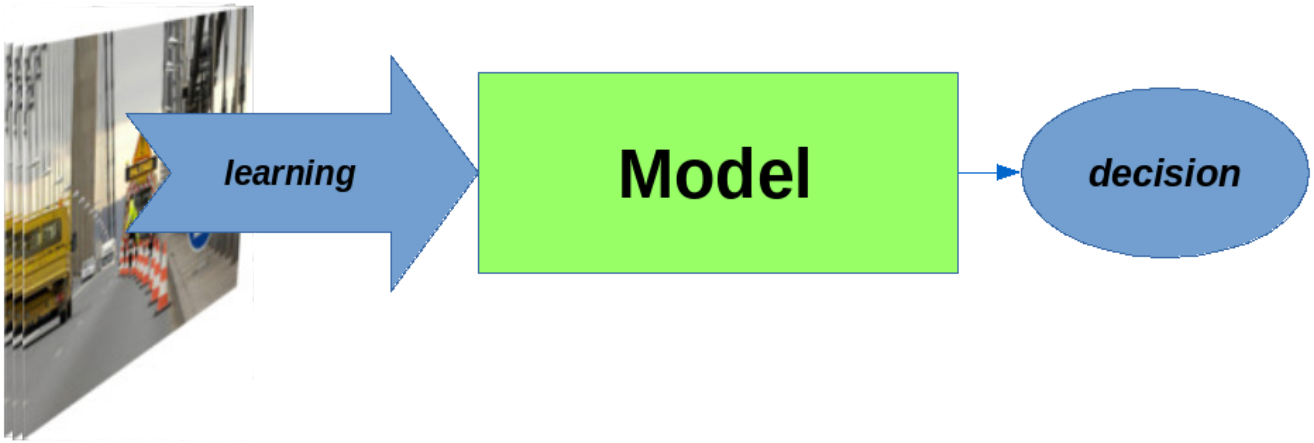
Explicit Features...

In a traditional approach, the observation space consists of *ad hoc features* extracted from the images by a dedicated processing:



...or Implicit Features

Since the 2010s, more and more approaches such as Deep Convolutional Networks apply *end-to-end* learning, i.e. where the observation space is the image itself. The features are therefore *learned* in the same way as the other levels of representation:



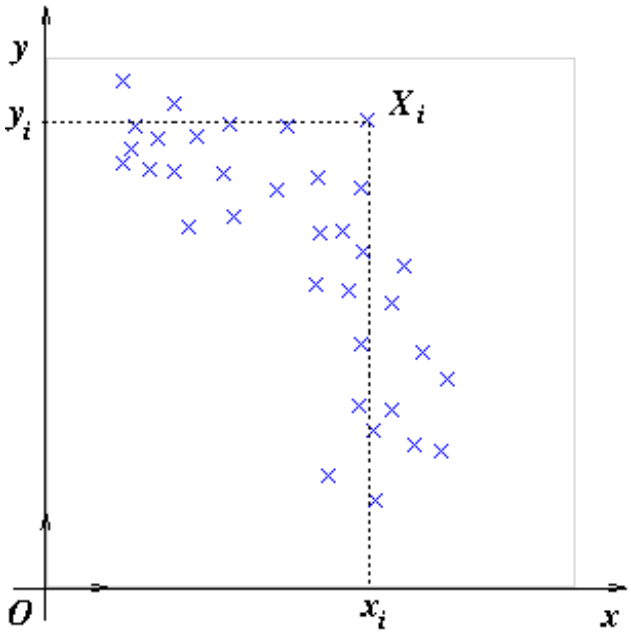
Outline of the lecture

Observation space Reduction

In "low dimension" learning, it is often necessary to reduce the dimension of features to: (1) Reduce complexity and (2) Limit redundancy between features.

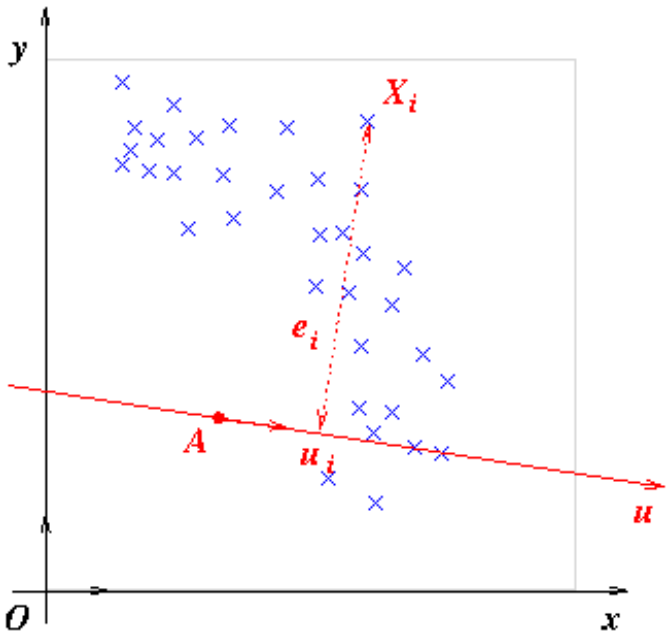
- **Principal Component Analysis** (*PCA*): Algebraic approach of *dimension reduction* to directions of *maximum variance*.
- **Clustering**: Metric approach to *quantify the number of features* by limiting them to a small number of representatives.

Principal Component Analysis



Let $\{X_i\}_{i=1...n}$ be a set of n points in a vector space of dimension $d \geq 2$.

Principal Component Analysis

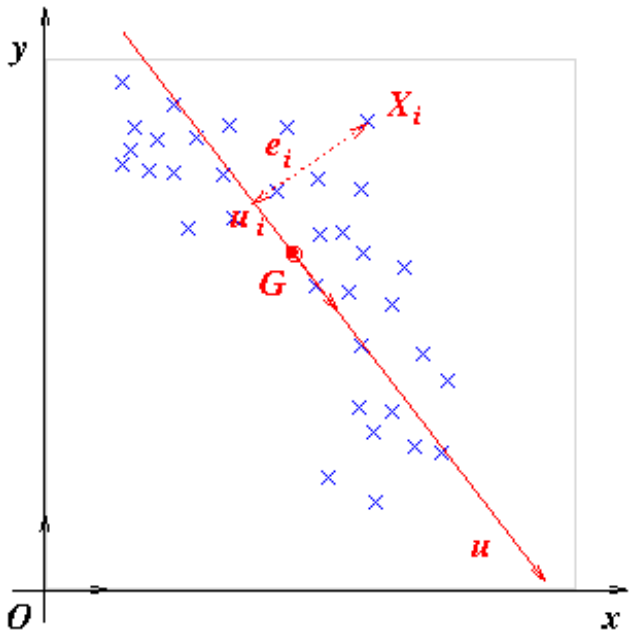


We are looking for an *one-dimensional optimal* representation of vectors $\{X_i\}$, i.e. a coordinate system $(A, \mathbf{u})$ that minimises the total error:

$$e = \sum_{i=1}^{n} ||X_i - (A + u_i \mathbf{u})||$$

where A is a point and $\mathbf{u}$ a unit vector of the space.

Principal Component Analysis



- The solution is provided by the coordinate system $(G, \mathbf{u})$, where G is the *centre of mass* (mean), and $\mathbf{u}$ is the *principal component* of $\{X_i\}$.
- $\mathbf{u}$ corresponds to the *direction with largest variance* of data $\{X_i\}$.
- $\mathbf{u}$ is given by the *eigen vector* of the *covariance matrix* of the $\{X_i\}$ associated to *the greatest eigen value*.

Principal Component Analysis

The process naturally extends to several principal components: if we denote $\mathbf{X}$ the matrix $(d \times n)$ composed of all the n observation vectors $\{X_i\}$ organised in columns:

$$\mathbf{X} = \begin{pmatrix} X_1^1 & X_2^1 & \dots & X_n^1 \\ \vdots & \vdots & \ddots & \vdots \\ X_1^d & X_2^d & \dots & X_n^d \end{pmatrix}, \text{ et } G = \frac{1}{n}\sum_{i=1}^{n} X_i$$

The $(d \times d)$ covariance matrix of the observations writes:

$$\mathbf{C} = \frac{1}{n}\mathbf{X}\mathbf{X}^t - GG^t$$

Principal Component Analysis

PCA Algorithm

- Calculate the covariance matrix $\mathbf{C} = \frac{1}{n}\mathbf{X}\mathbf{X}^t - GG^t$
- Calculate and sort the eigen values $(\lambda_1, \ldots, \lambda_n)$, such that $\lambda_1 > \cdots > \lambda_n$, and their associated eigen vectors $(U_1, \ldots, U_n)$.
- The k ($k << d$) first eigen vectors $\{U_1, \ldots, U_k\}$ are the *principal components*.

- The principal components are the *theoretical* dimensions (i.e they don't have a physical meaning in general!) that "best explain" the repartition of the $\{X_i\}$ in the observation space.
- The matrix $\mathbf{C}$ is symmetric $\Rightarrow$, so the principal components are orthogonal.

PCA for Face Recognition

32×32 Learning Images $\{X_i\}$

"Mean" Face G

Images from *Derek Hoiem*, University of Illinois.

PCA for Face Recognition

U_k

$G + 3\sqrt{\lambda_k}U_k$

$G - 3\sqrt{\lambda_k}U_k$

"Eigenfaces for Recognition"

[Turk91]

Images from *Derek Hoiem*, University of Illinois.

Clustering: K-means Algorithm

- The clustering algorithm K-means performs a partition $\mathcal{C}$ of the observation set $\{X_i\}$ into K classes C_k $(1 \leq k \leq K)$.
- Let $\Pi = \{\pi_k\}$ be a set of K prototypes chosen in the observation space, and representing each one of the classes.
- Let d be a distance in the observation space.

The objective is to minimise the following cost function:

$$J_{\mathcal{C}}^{\Pi} = \sum_{k=1}^{K} \sum_{X_i \in C_k} d(X_i, \pi_k)^2$$

Clustering: K-means Algorithm

The objective is to minimise the following cost function:

$$J_{\mathcal{C}}^{\Pi} = \sum_{k=1}^{K} \sum_{X_i \in C_k} d(X_i, \pi_k)^2$$

The solution is trivial in any of those two cases:

- If the set of prototypes Π is fixed:
  $C_k = \{X_i; \forall j \neq k, d(X_i, \pi_k) \leq d(X_i, \pi_j)\}$
- If the partition into classes $\mathcal{C}$ is fixed: $\pi_k = \frac{1}{|C_k|} \sum_{X_i \in C_k} X_i$

Clustering: K-means Algorithm

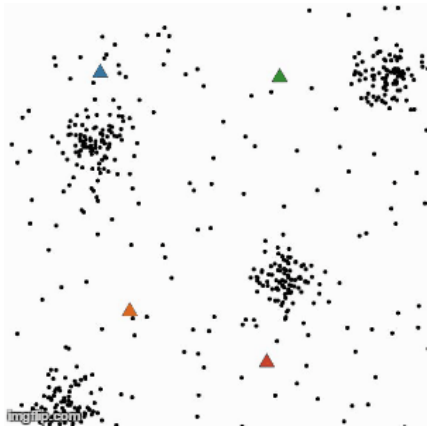
K-means is an iterative algorithm that converges to a local minimum of $J_{\mathcal{C}}^{\Pi}$, by alternatively adjusting $\mathcal{C}$ and Π :

K-means Algorithm

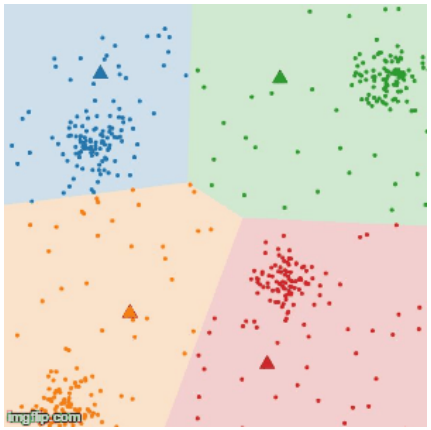
Initialisation: A set of K prototypes $\Pi = \{\pi_k\}$ is chosen.
Repeat until convergence:

1. Update $\mathcal{C}$: $C_k = \{X_i; \forall j \neq k, d(X_i, \pi_k) \leq d(X_i, \pi_j)\}$
2. Update Π : $\pi_k = \frac{1}{|C_k|} \sum_{X_i \in C_k} X_i$

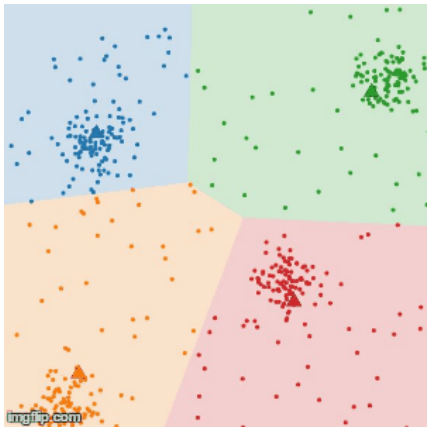
Clustering: K-means Algorithm



$[\Pi]^{(0)}$



$[\mathcal{C}]^{(1)}$

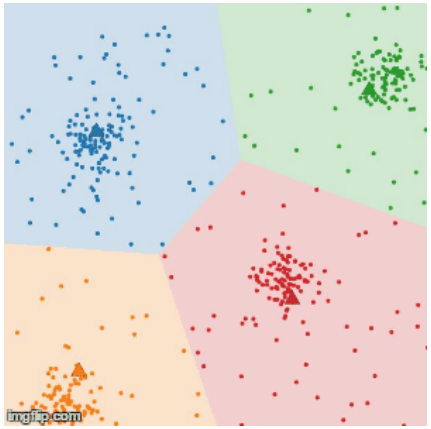


$[\Pi]^{(1)}$

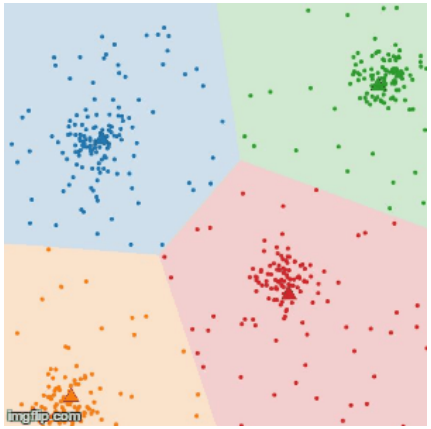
- Initialisation is critically important!
- Partition $\mathcal{C} \leftrightarrow$ Voronoi Diagram

Images: *Wikipedia*

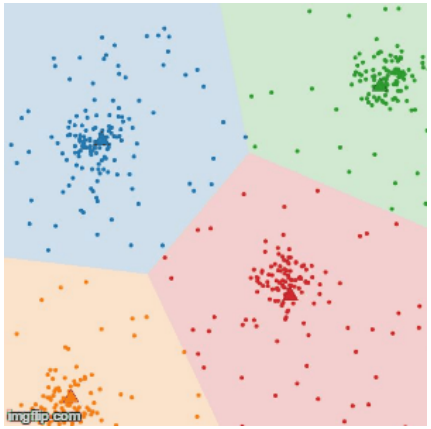
Clustering: K-means Algorithm



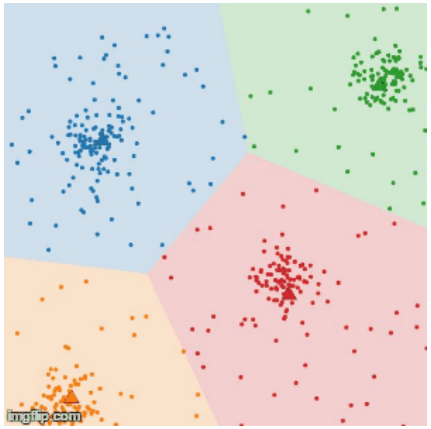
$[\mathcal{C}]^{(2)}$



$[\Pi]^{(2)}$



$[\mathcal{C}]^{(3)}$



$[\Pi]^{(3)}$

- Initialisation is critically important!
- Partition $\mathcal{C} \leftrightarrow$ Voronoi Diagram

Images: *Wikipedia*

Outline of the lecture

Bayesian Classification

The observation data X,and the classes c are modelled as random vectors (resp. variables). We denote:

- $P(X)$ and $P(c)$ the *prior* probabilities.
- $P(X, c)$ the *joint* probability.
- $P(X/c)$ the conditional *likelihood* of data X.
- $P(c/X)$ the *posterior* probability of class c.
- **Learning**: Build the probability laws from a collection of labelled data $\{X_i, c(X_i)\}$.
- **Inference**: Associate to a test data X its class $c^\star$.

Bayesian Inference

- **Inference**: Associate to a test data X its class $c^\star$.

Maximum Likelihood (ML) criterion

$$c^\star(X) = \arg\max_c P(X/c)$$

Bayesian Inference: Maximum a Posteriori (MAP) criterion

$$c^\star(X) = \arg\max_c P(c/X)$$

Bayesian Learning

- **Learning**: The joint distribution $P(X, c)$, and then the *posterior* law, is estimated from the learning data $\{X_i\}$.

Indeed $P(X, c) = P(X).P(c/X) = P(c).P(X/c)$, and so:

Bayes Theorem

$$P(c/X) = \frac{P(X/c).P(c)}{P(X)}$$

- $P(c)$ is the occurrence probability of class c, arbitrarily set, or empirically estimated from statistics over the learning data.
- $P(X/c)$ is provided by statistics over the *labelled* data (i.e. whose class is known = supervised learning).

Bayesian Classification

- The MAP criterion then writes:

$$c^\star(X) = \arg\max_c P(X/c).P(c)$$
$$= \arg\max_c \underbrace{\log P(X/c)}_{\text{(Data)}} + \underbrace{\log P(c)}_{\text{(Modèle)}}$$

- Finally it is the weighting by the *a prior* law that distinguishes the MAP from the ML criterion.
- The classification model then consists in empirically estimating $P(X/c)$ from statistics over $\{X_i\}$.

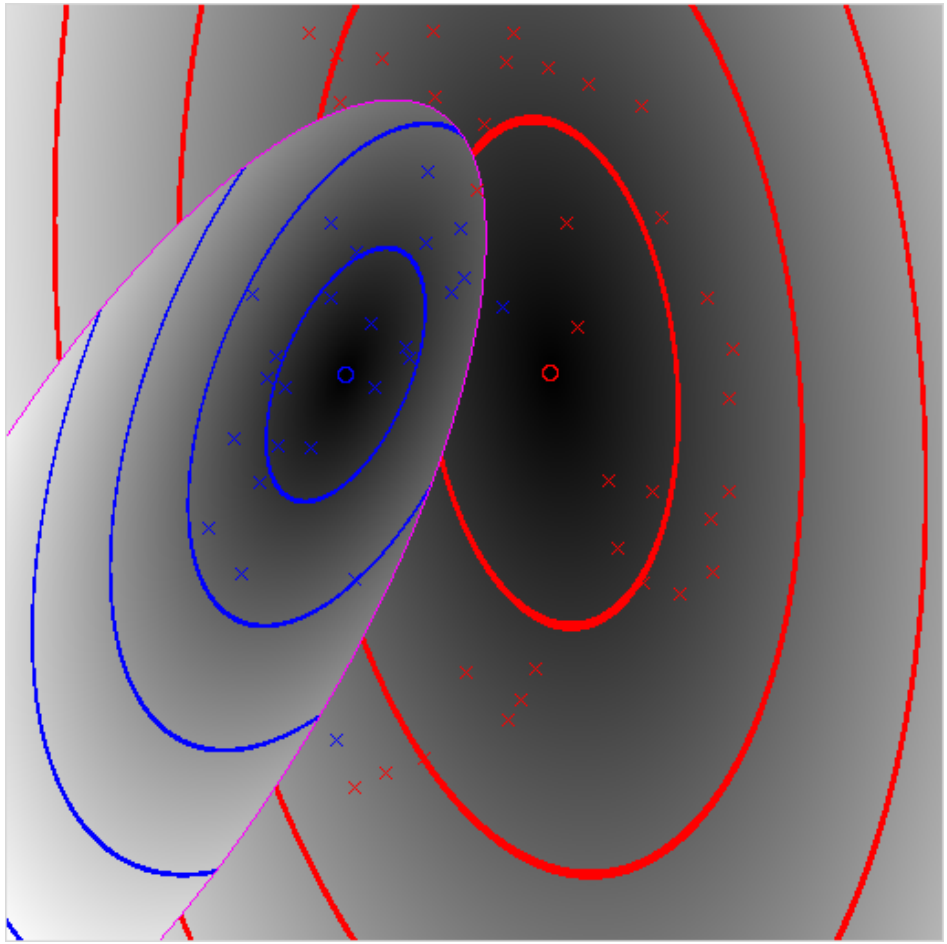
Multivariate Bayesian Model

A simple parametric model of the distribution of $P(X/c)$, like the Gaussian, is often used:

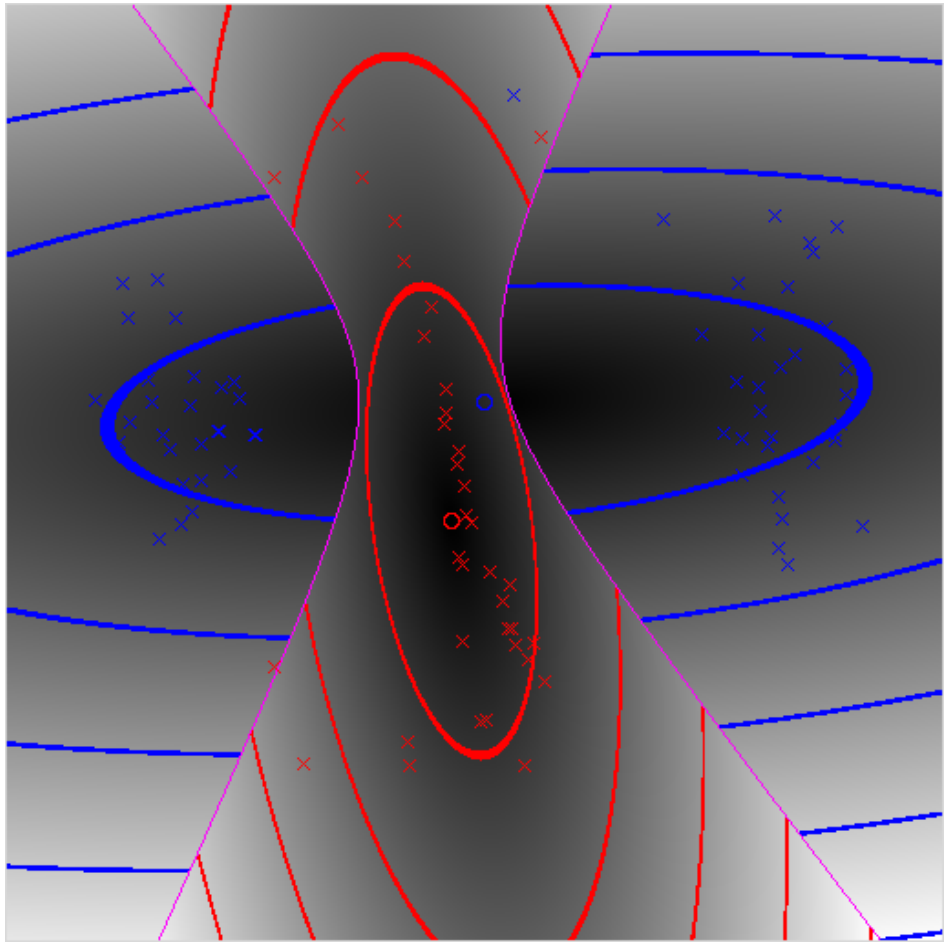
$$P(X/c) = f_c(X) = \frac{1}{\sqrt{(2\pi)^d \det \mathbf{C}_c}} \exp\left(-\frac{1}{2}(X - G_c)^t \mathbf{C}_c^{-1}(X - G_c)\right)$$

where G_c and $\mathbf{C}_c$ are respectively, the mean vector and the covariance matrix of the learning data labelled at class c.

Bayesian Classification: Gaussian Models



Bayesian Classification: Gaussian Models



Naive Bayesian Model

The naive Bayesian model makes the hypothesis of independence of the different dimensions of X conditionally to the class c:

$$P(X/c) = P(x_1, \ldots, x_d/c) = \prod_{j=1}^{d} P(x_j/c)$$

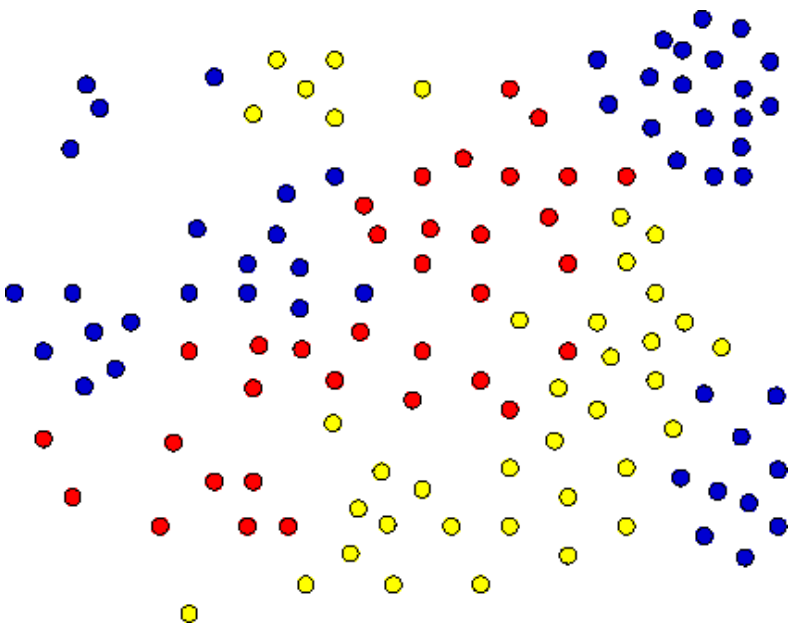
and finally:

$$c^\star(X) = \arg\max_c \log P(c) + \sum_{j=1}^{d} \log P(x_j/c)$$

- Estimates only scalar distributions, maybe more complex than unimodal Gaussian (Law mixtures, histograms, ...).
- May work better than a unimodal multivariate model.

Outline of the lecture

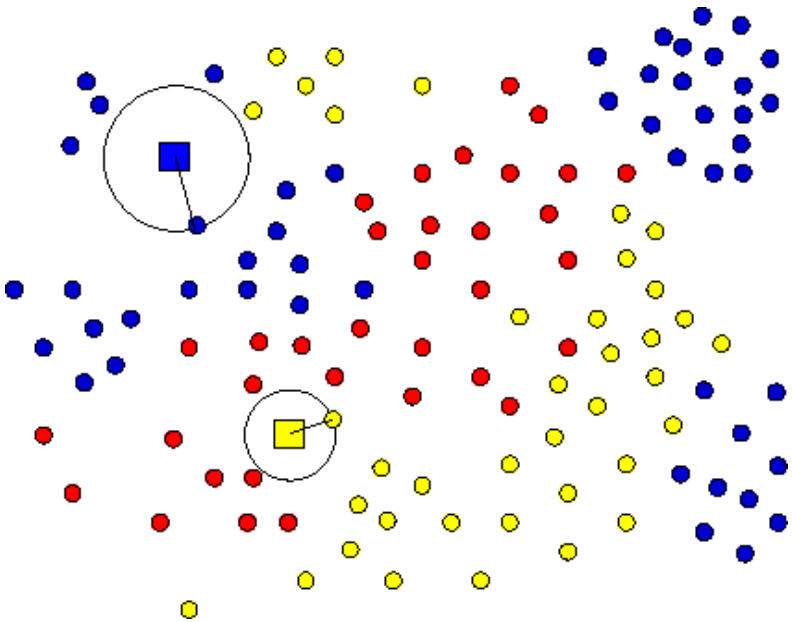
Nearest Neighbours Classification



In Nearest Neighbours classification (k-NN), we consider:

- a set of *labelled* points in an observation space $\{X_i, c(X_i)\}$,
- the class $c(X_i)$ is provided by supervision,
- d a distance in the observation space.

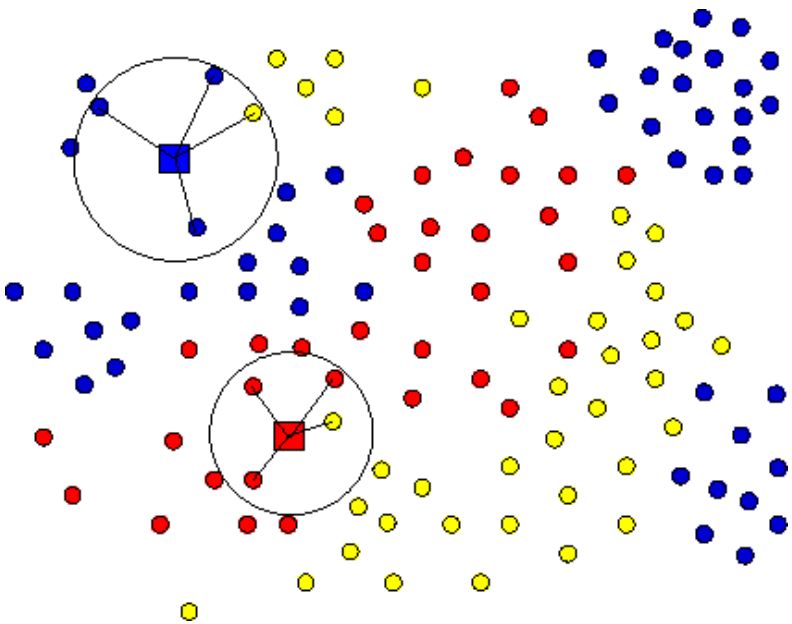
1-NN



The 1-NN classification of an unknown vector X consists in assigning it to *the class of its nearest neighbour* among the learning vectors:

$$c^\star(X) = c\left(\arg\min_{X_i} d(X, X_i)\right)$$

k-NN



The k-NN classification of an unknown vector X consists in assigning it to *the most frequent class among its k nearest neighbours* in the learning vectors.

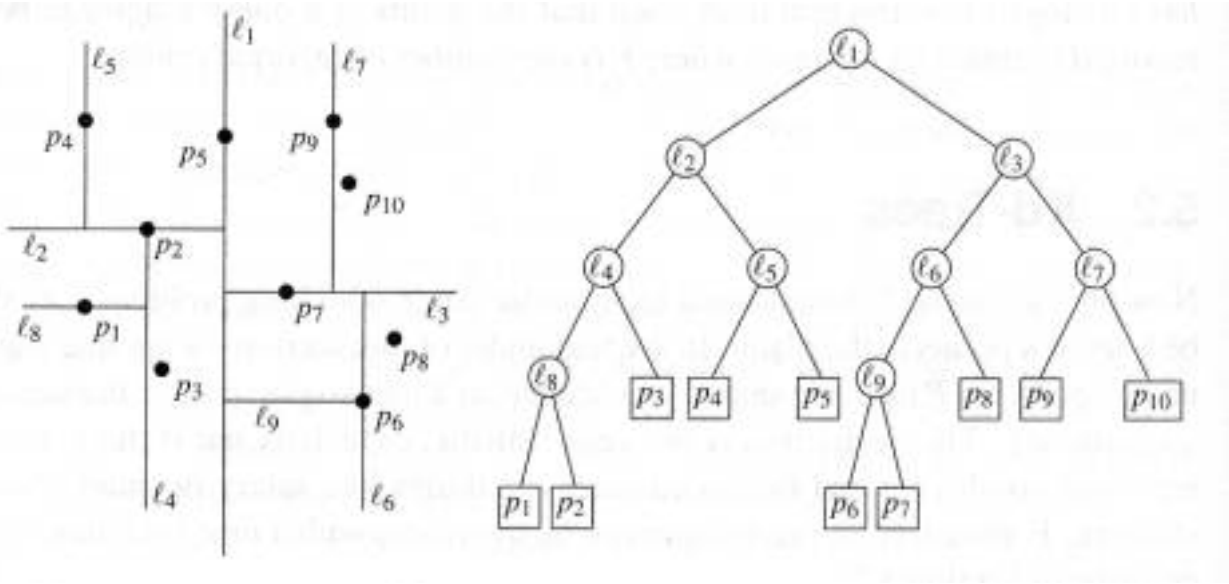
Advantages of k-NN

- *Versatile:* can be adapted to ane kind of distribution in the observation space, without probabilistic prior.
- *Simple:* the model is simply the set of learning vectors and their classes.
- *General:* it trivially extends to the *regression* of a function:

$$f^\star(X) = f\left(\arg\min_{X_i} d(X, X_i)\right).$$

Limitations of k-NN

- *Memory cost:* the model size can be huge.
- *Computation cost:* the complexity of the exhaustive search of NN linearly depends on the number of data n, and on their dimension d. The average complexity can be reduced to $\mathcal{O}(\log n)$ by using kd-trees:

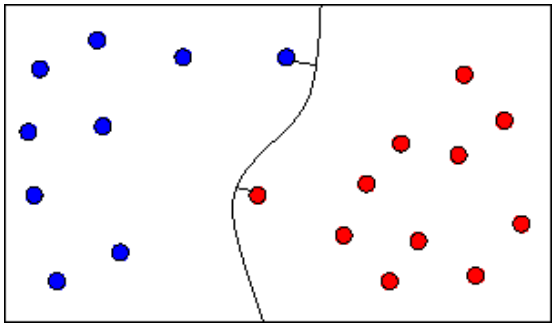


Limitations of k-NN

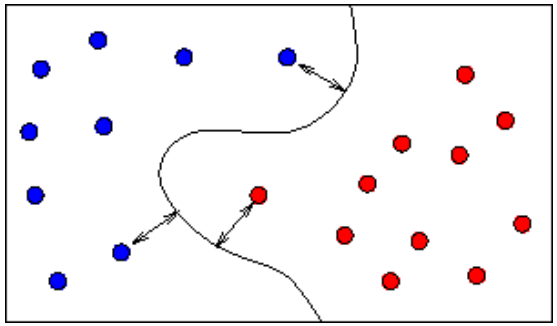
- *Curse of dimensionality:* the number of samples needed to estimate with the same reliability a distribution in the observation space *increases exponentially with the dimension*. The same goes regarding the number of learning examples for the methods without probabilistic prior like k-NN. Possible solutions are:
  - Use a probabilistic model! (cf Section 3).
  - Reduce the dimension! (cf Section 2).

Overview of SVMs

- *Support Vector Machines* are a *discriminative* approach that builds *optimal separation surfaces between two classes* in the observation space, from labelled learning example vectors.
- The optimal surface is the one that maximises the separation *margin*, i.e. the distance of the nearest vectors from the surface (= the *support vectors*):



Small margin



Large margin

Overview of SVMs

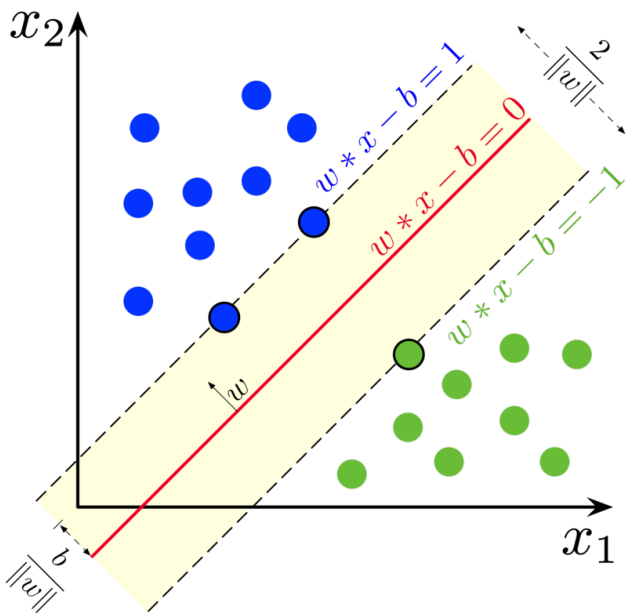


Image : *Wikipedia*

In the linear case the separation surface is a hyperplane of equation:

$$X.\mathbf{w} - b = 0$$

- $\mathbf{w}$ is a vector normal to the hyperplane.
- b is proportional to $d(\mathcal{P}, O)$, the distance of the hyperplane to the origin.

Overview of SVMs

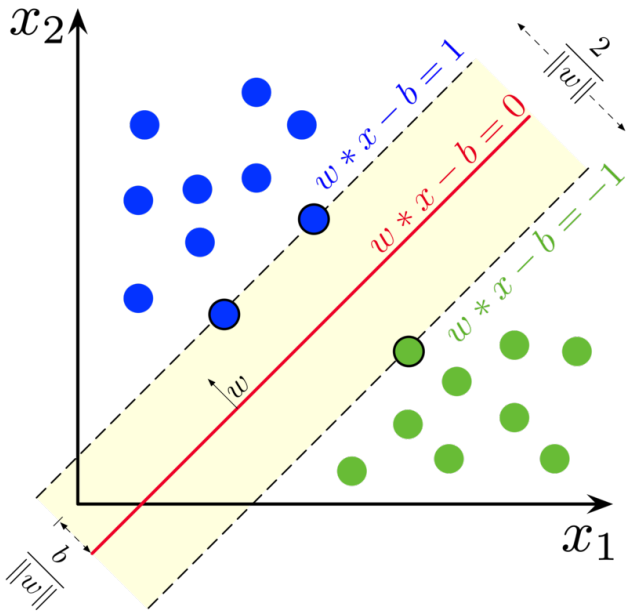


Image : *Wikipedia*

$$X.\mathbf{w} - b = 0$$

$\mathbf{w}$ and b are *normalised* by the margin m (distance from $\mathcal{P}$ to the support vectors) :

- $||\mathbf{w}|| = \frac{1}{m}$
- $b = \frac{d(\mathcal{P},O)}{m}$

Classification by SVM

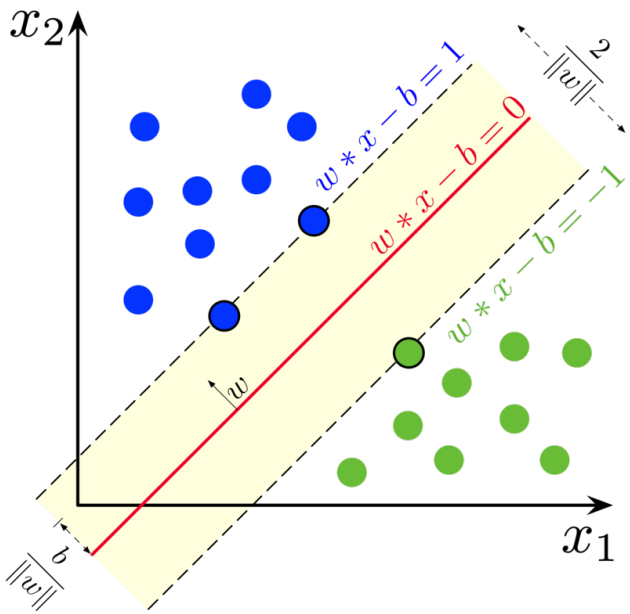


Image : *Wikipedia*

- The classification of an unknown vector X is simply done by comparing $X.\mathbf{w}$ to b:

$$c^\star(X) = \text{sign}(X.\mathbf{w} - b)$$

- The *confidence* of the classification can be quantified by the absolute value (= distance from X to the hyperplane, in "number of margins") :

$$\kappa(X) = |X.\mathbf{w} - b|$$

Training a SVM

Training a linear SVM on labelled learning data $\{X_i, c(X_i)\}$, with $c(X_i) = +1$ or $c(X_i) = -1$ is a quadratic optimisation problem under linear constraints:

$$\min_{(\mathbf{w},b)} ||\mathbf{w}||^2 \text{ u.c. } \forall i, c(X_i)(X_i.\mathbf{w} - b) \geq 1$$

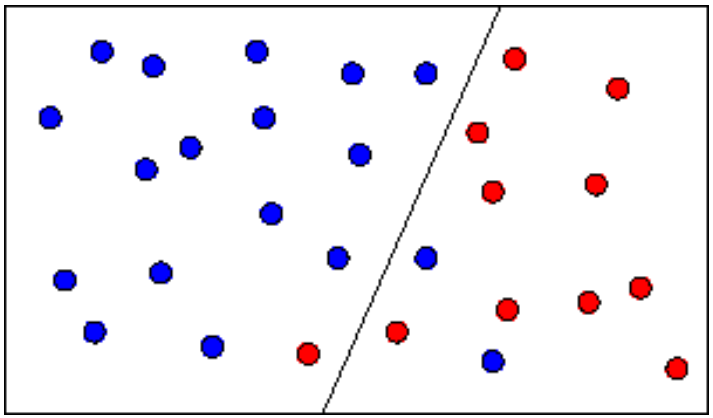
The support vectors X_s are those which satisfy:

$$c(X_s)(X_s.\mathbf{w} - b) = 1$$

Soft Margin SVM

The linear SVM can be extended to the case where the two classes are not linearly separable, by softening the margin. The following optimisation can be used:

$$\min_{(\mathbf{w},b)} \left\{ \left[\frac{1}{n} \sum_{i=1}^{n} \max\left(0, 1 - c(X_i)(X_i.\mathbf{w} - b)\right) \right] + \lambda ||\mathbf{w}||^2 \right\}$$



Non linear SVM

The idea of the non linear SVMs is to transform the observation space (vectors X) into a higher dimension space (vectors $\Phi(X)$) where the vectors would be linearly separable.

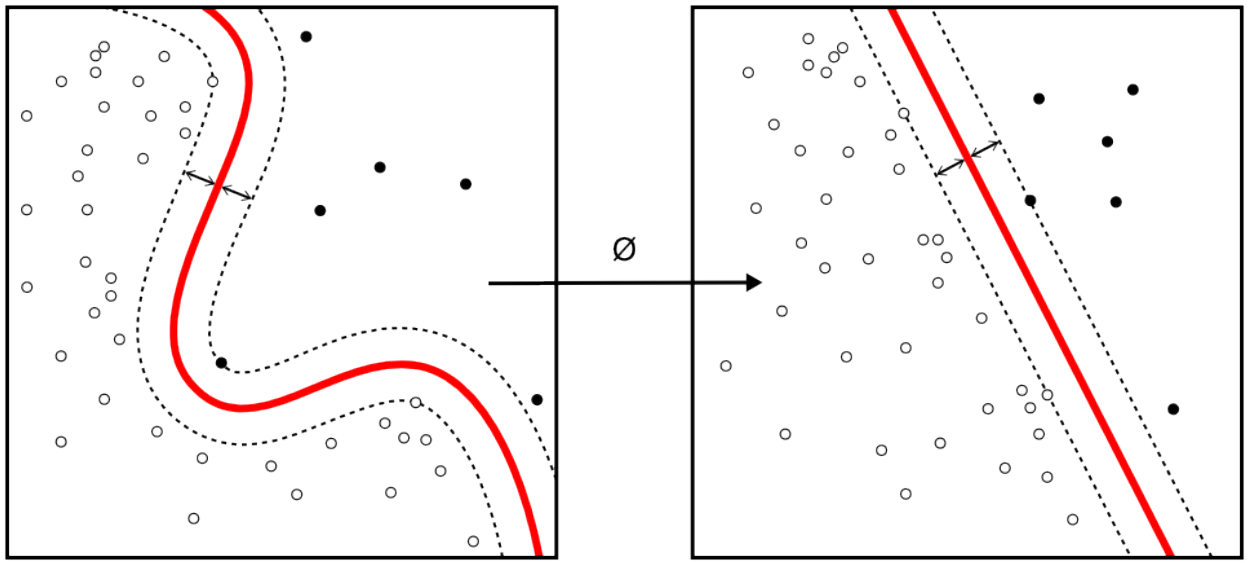


Image : *Wikipedia*

Non linear SVM

The so-called *Kernel trick* consists in not calculating explicitly Φ, but using a non linear real-valued function (kernel) to replace the dot product: $K(X, Y) = \Phi(X).\Phi(Y)$.

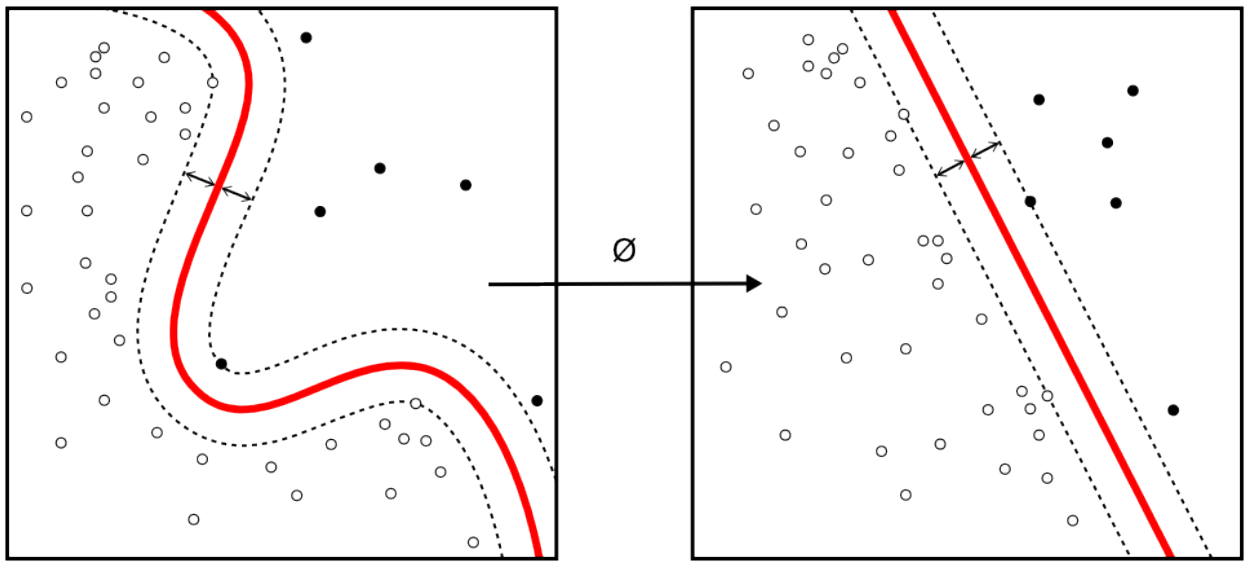


Image : *Wikipedia*

Conclusion on the SVMs

- The SVMs are extremely compact models: one separating hyperplane $(\mathbf{w}, b)$ by SVM.
- The classification is extremely fast: one single dot product to classify a vector between the two classes.
- The SVMs are exclusively binary classifiers. To extend them to a N-class classification, two approaches are possibles:
  - *1-against-all*: N binary classifiers.
  - *1-against-1*: $\frac{N(N-1)}{2}$ binary classifiers.

Random Forests

The principle of *Random Forests*, (RF) is based on classification by decision trees:

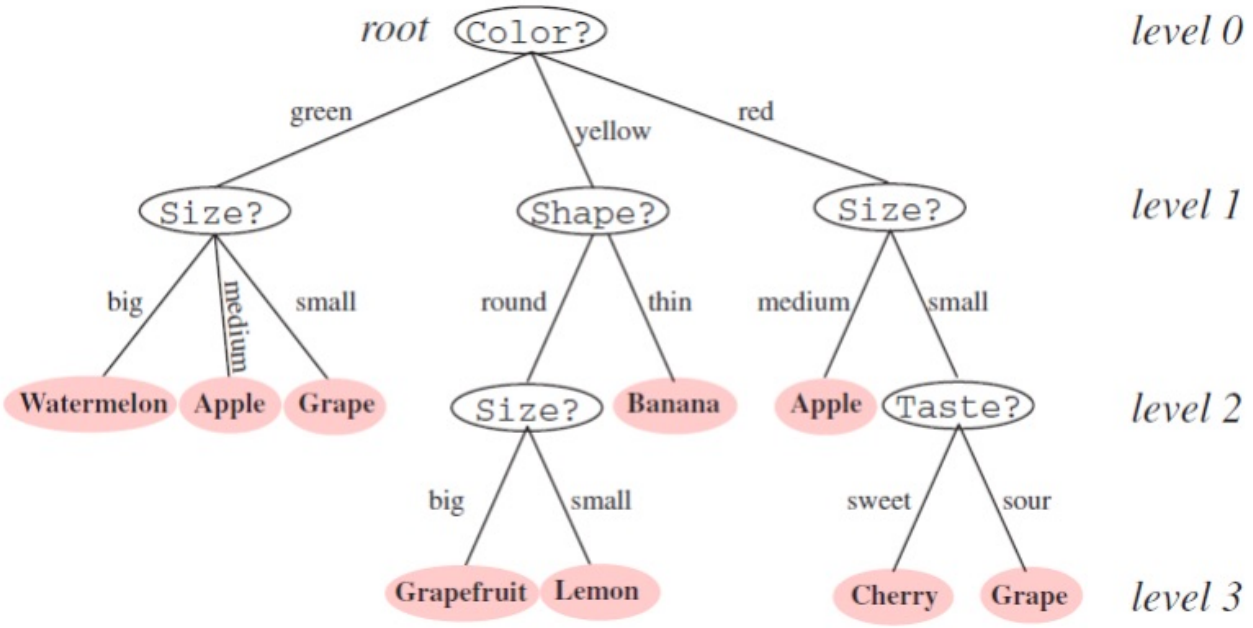


Figure by *Anil K. Jain*

Training of a Random Tree

The learning (creation of a tree) consists in partitioning recursively the learning set $\{X_i, c(X_i)\}$ into two subsets, until some stop criterion, typically related to:

- the *cardinality* of the leaf (i.e. minimal size of the subset, below which it is no longer considered representative).
- the *quality* of the leaf (e.g. low class entropy of the subset, meaning that the leaf correctly "isolated" one or more classes).

Training of a Random Tree

The creation of each node of the tree is made as follows:

- randomly draw a large number of *binary predicates* typically related to one component of the vectors $\{X_i\}$ (e.g. "$X_i^k < t$").
- evaluate the (provisional) quality of the leaves resulting from the corresponding partition (e.g. by calculating their class entropy).
- keep the predicate that realises the best quality score.

NB: Note the kinship with kd-trees.

Classification by RF

- The classification by a Random Tree consists in submitting an unknown vector X to the sequence of predicates met during the traversal of the tree from the root to a leaf (see kd-trees and k-NN search).
- The histogram of classes represented in the arrival leaf is interpreted as a probability density for the class of the vector: $P(c/X)$.
- The process is repeated on many trees of the forest, and the decisions are combined (majority vote, or average of the histograms...).

Classification by RF

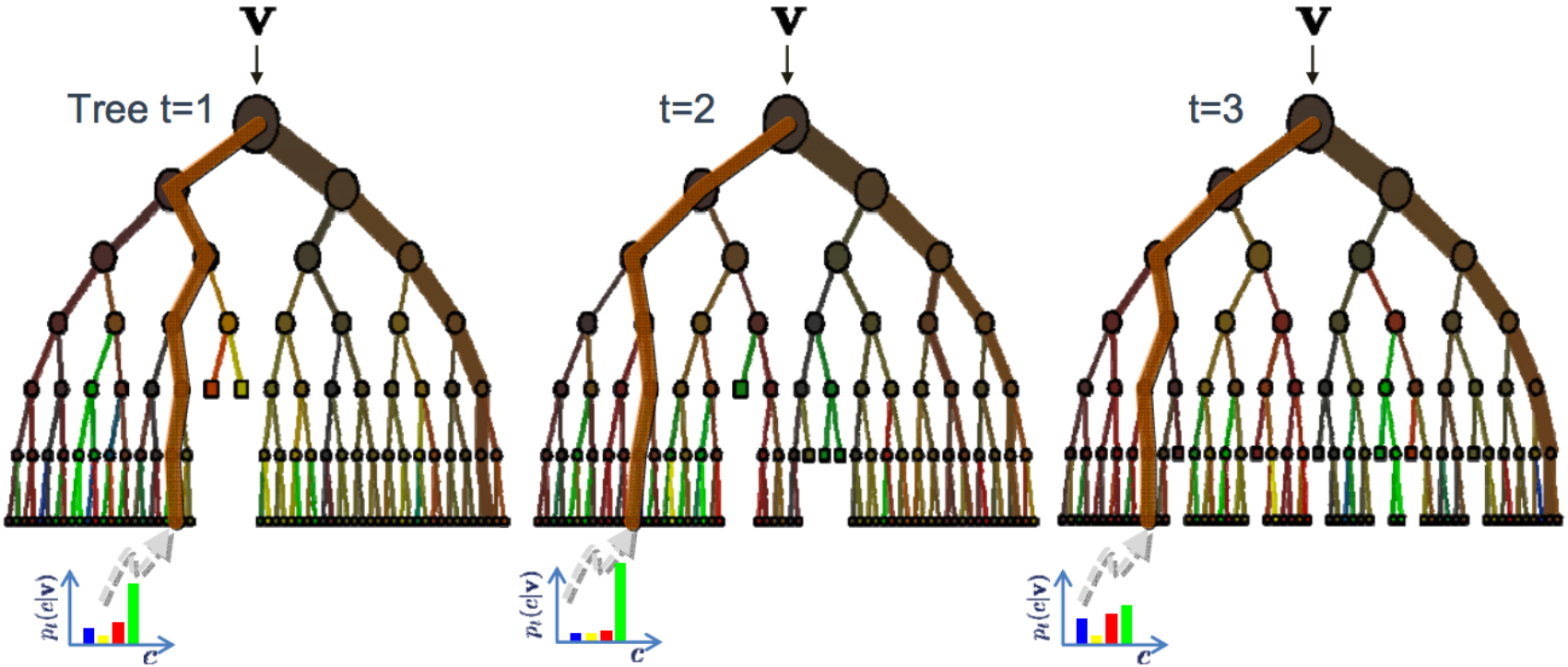


Figure by *Raghav Aggiwal*

Conclusion on Random Forests

+ Possibility to combine components of very different nature (continuous or discrete, with or without metrics).
+ Readability ("*Explainability*") of the model.
+ Speed of classification.
- Large memory cost of the model.
- Performance drop with the increase in the number of training data, as well as their dimension (i.e. number of features).

Outline of the lecture

Artificial Neural Networks

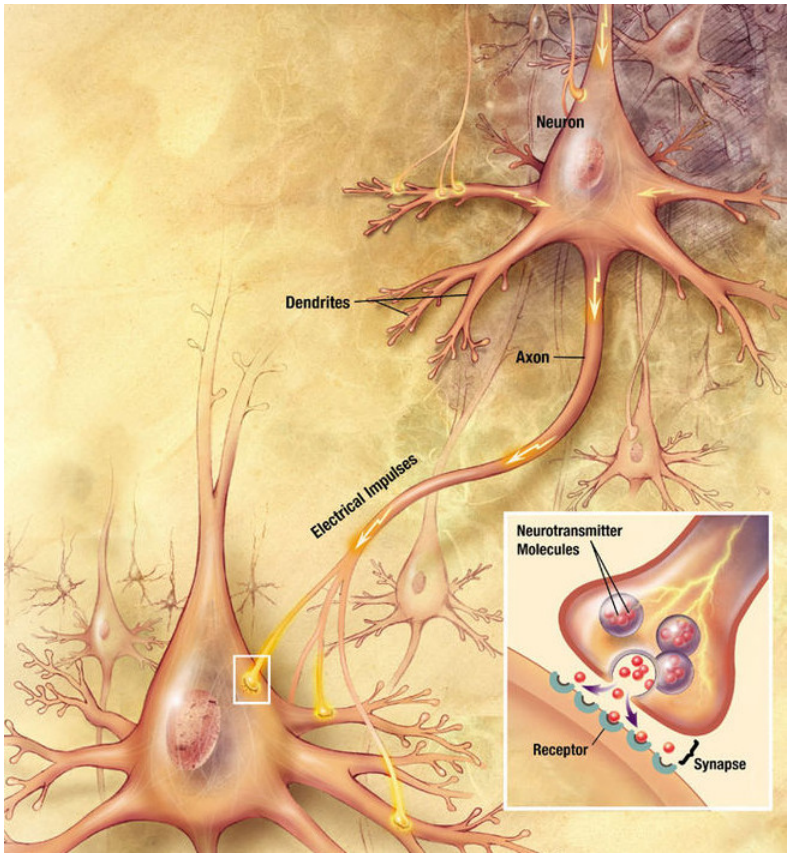
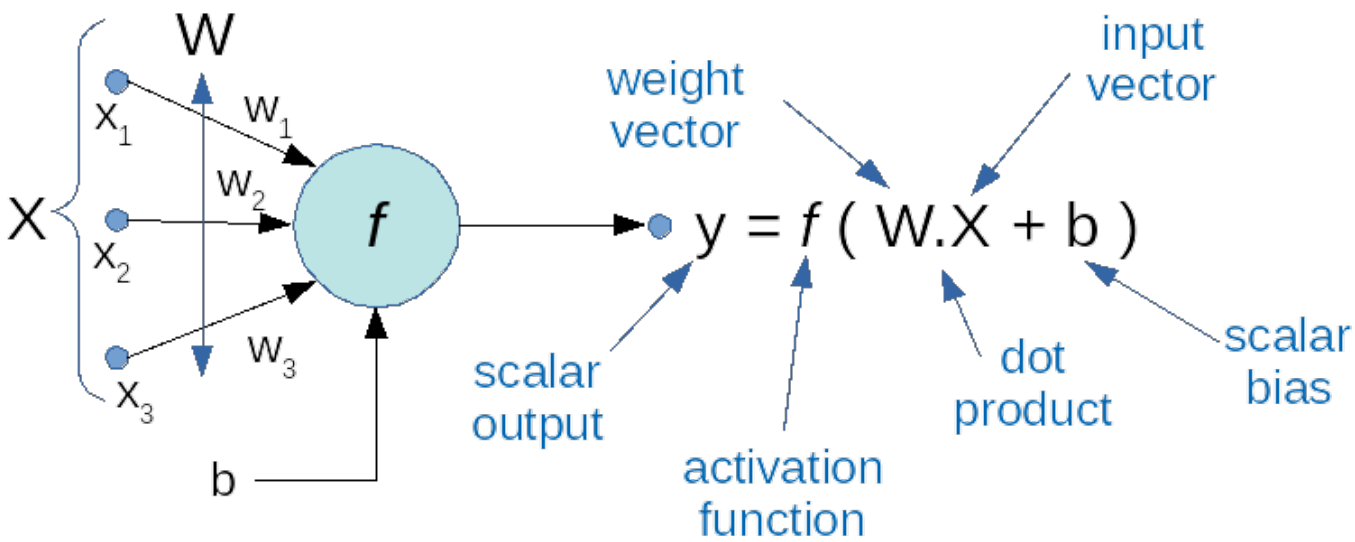


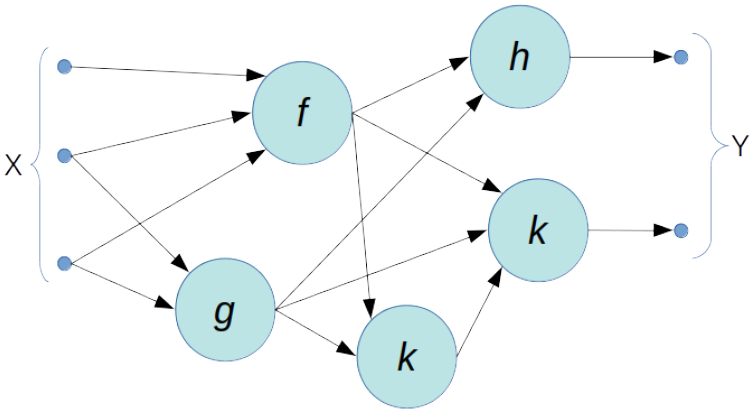
Image: *Wikipedia*

- 1940s: Formal Neuron Model - Hebb's Rule.
- 1950-60s: Works on Perceptron.
- 1980s: Hopfield's networks, Boltzmann's machines.
- 1980s: Multilayers Perceptrons, Gradient Back-propagation Algorithm.
- 2010s: Strong Comeback *via* the Deep Networks.

Formal Neuron Model



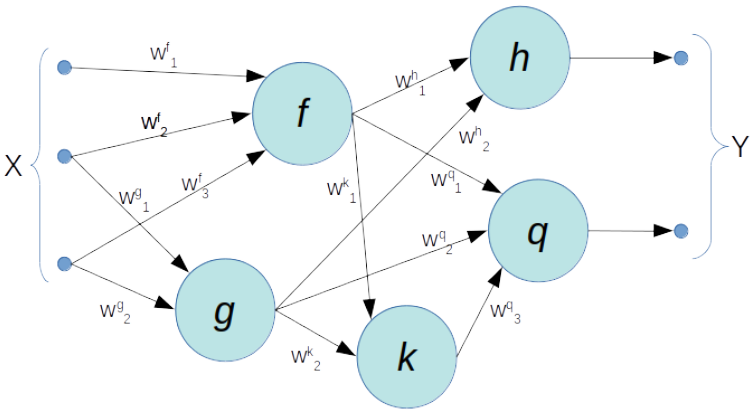
Neural Networks



A formal Neural Network is an oriented graph, where:

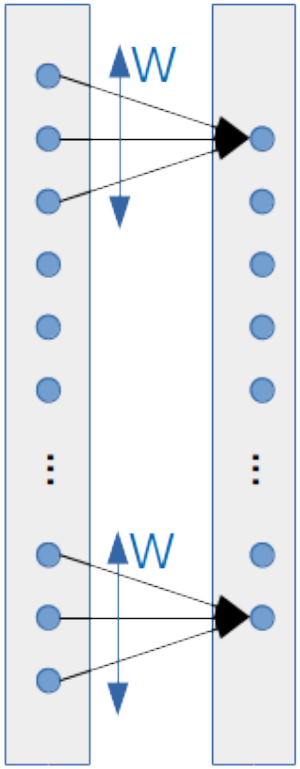
- The *source* nodes form the input vector X, that represents the data, or is the data itself (end-to-end learning).
- The *sink* nodes form the output vector Y, which is interpreted as the result of the classification or regression.

Neural Networks



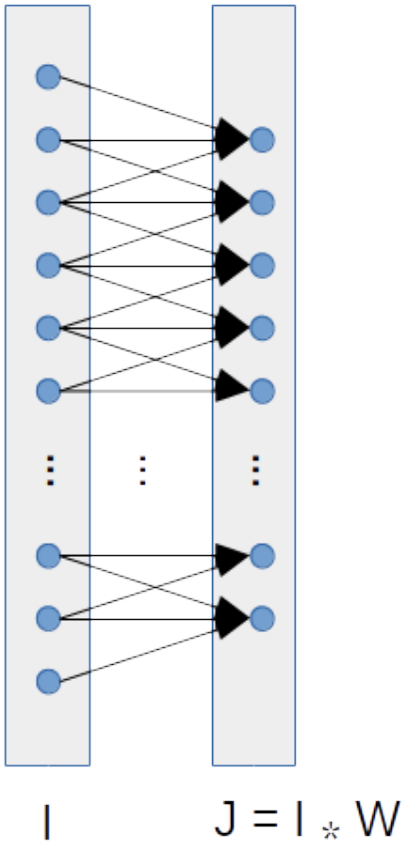
- The architecture (i.e. the graph), and the activation functions are generally defined *a priori* and *static*.
- The weights of the connexions W (and the bias values b) are *adaptive* and modelled by the learning process.

Convolutional Neural Networks



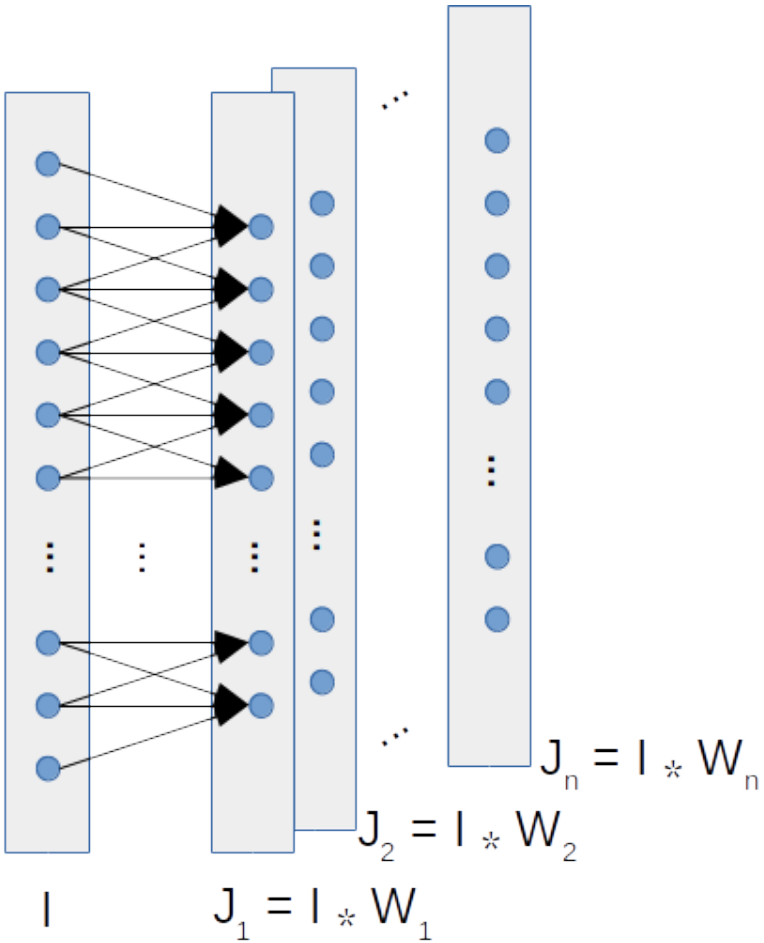
In a Convolutional Neural Networks (CNN), a same neuron (i.e. same weight vector and activation function) is used for all the parts of the input vector associated to each layer.

Convolutional Neural Networks



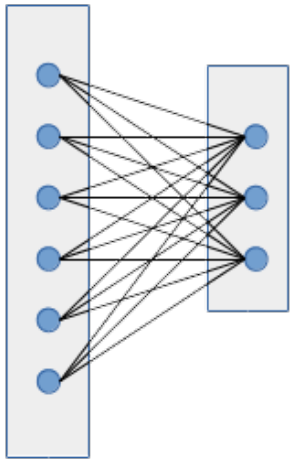
The operation performed between two layers I and J is then a translation invariant linear mapping, i.e. a convolution...

Convolutional Neural Networks



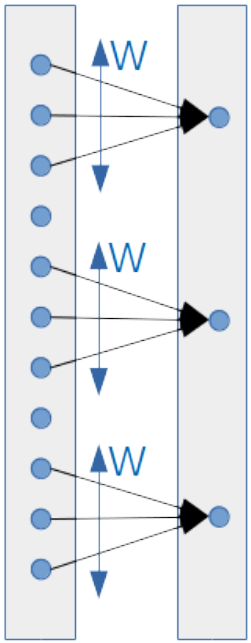
In fact, there are generally several neurons that are applied this way to each layer, which corresponds to a convolution filter bank...

Fully Connected Layers

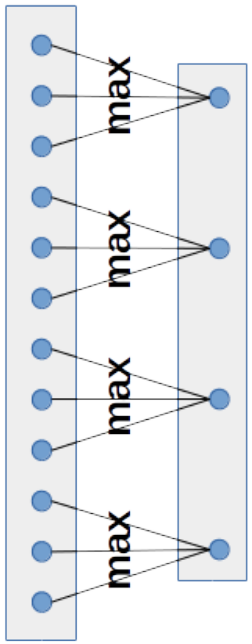


An important special case: when the size of the weight vectors is the same as the size of the input vector, this corresponds to a Fully Connected (FC) Layer.

CNN: Dimension Reduction Mechanisms



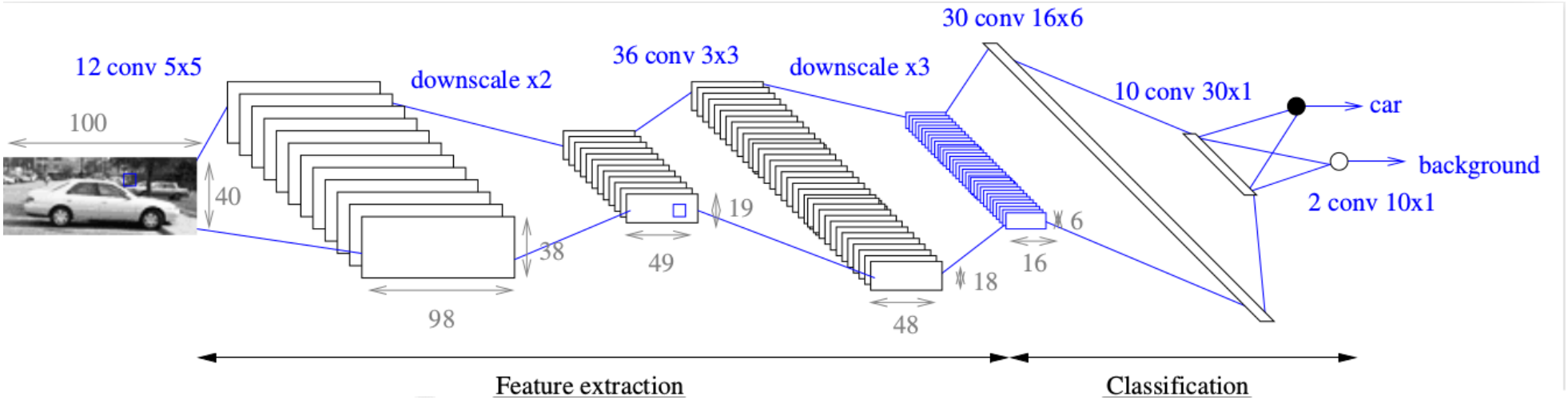
(1)



(2)

1. **stride:** Use of an increased sampling step while applying the convolution.
2. **pooling:** Sub-sample the data after applying a local combination (average, maximum...) of its values.

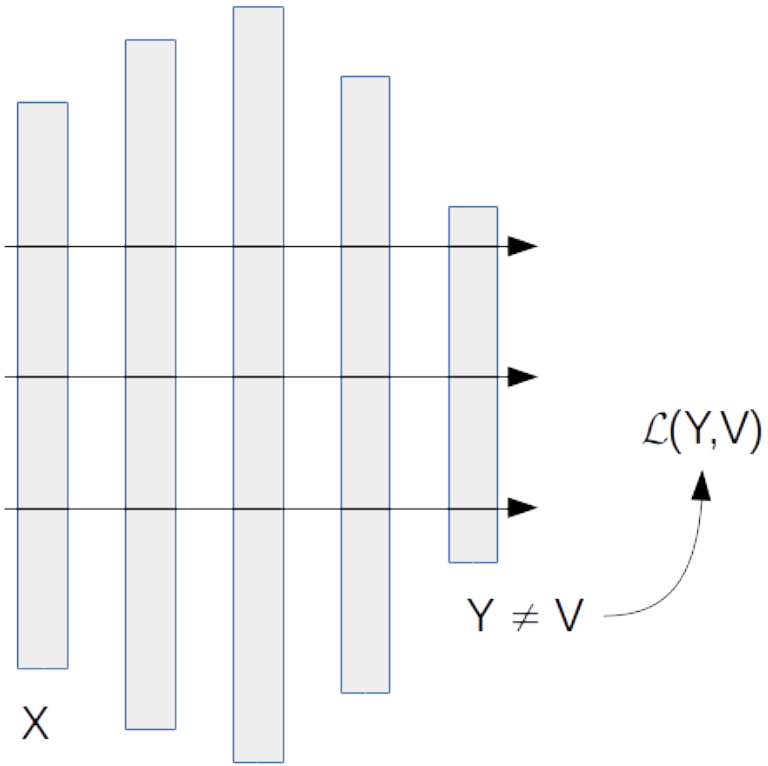
Example of a Convolutional Network



Supervised Training of a NN

- A NN is trained from a learning set $\{X_i, V_i\}$ where X_i are the training data and V_i the expected outputs (*Ground Truth*).
- A loss function $\mathcal{L}(Y, V) \geq 0$ is designed, that measures the difference (error) between the predicted output Y and the expected output V.
- The objective of the training is to minimise the global error over the training set: $\sum_i \mathcal{L}(\mathcal{O}(X_i), V_i)$, where $\mathcal{O}(X)$ is the output predicted by the network on the input X.

Supervised Training of a NN (forward)



In the forward pass, the data X is submitted to the network, and its predicted output Y is compared to the expected output V using the loss function $\mathcal{L}(Y, V)$.

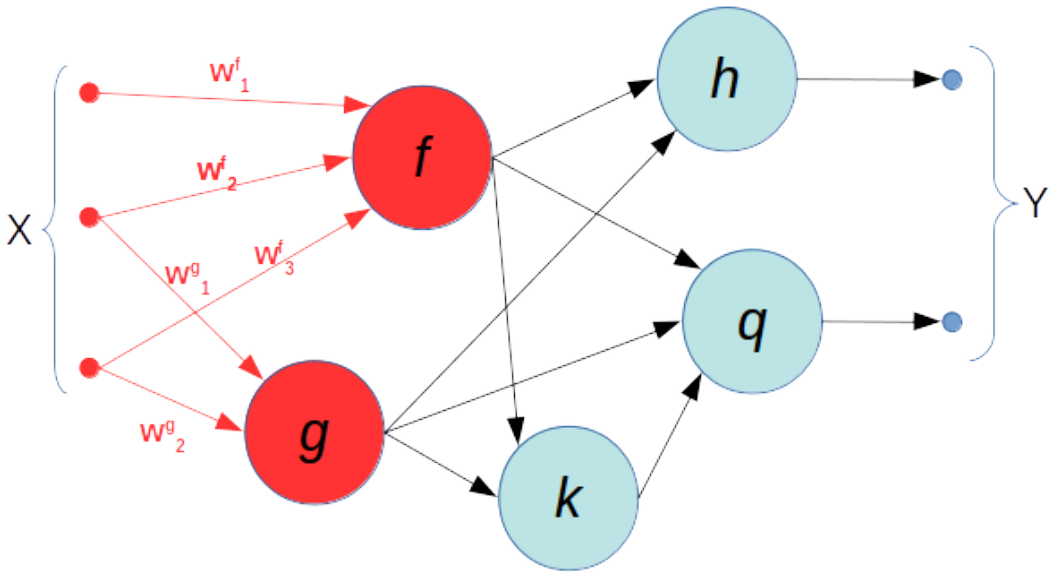
Forward Propagation

To simplify the notation, all activation functions are assumed equal to g. w_{kj} denotes the weight of the connection from neuron k to neuron j. The forward pass is then written:

Forward Propagation $Y \leftarrow \text{ForwardProp}(W, X)$

- For all neuron i from the input layer: $s_i = x_i$.
- For all following layer l:
  - For all neuron j from layer l: $s_j = g\left(\sum_k w_{kj} s_k + b_j\right)$.
- For all neuron j from the output layer: $y_j = s_j$.

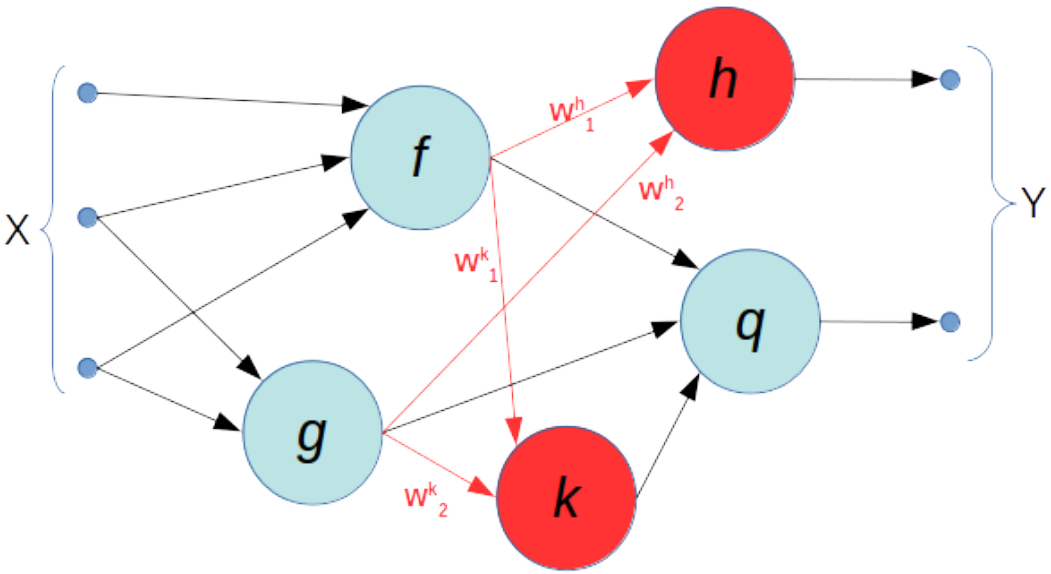
Example of forward pass



Compute, *in parallel*:

- $s_f = f\left(w_1^f x_1 + w_2^f x_2 + w_3^f x_3\right)$
- $s_g = g\left(w_1^g x_2 + w_2^g x_3\right)$

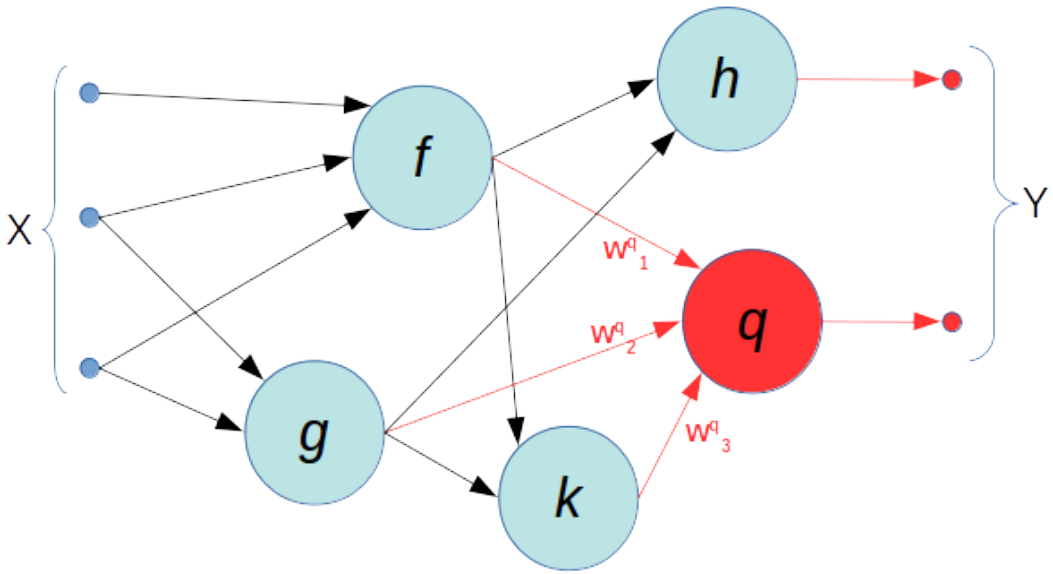
Example of forward pass



Compute, *in parallel*:

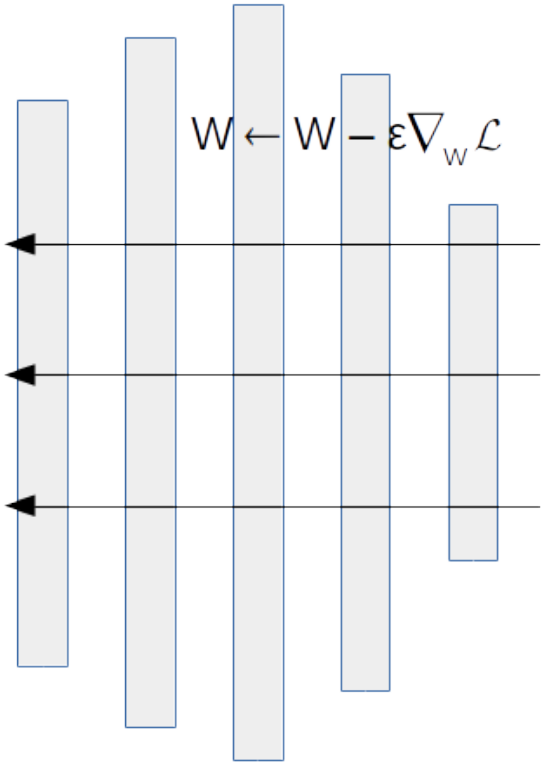
- $s_h = h\left(w_1^h s_f + w_2^h s_g\right)$
- $s_k = k\left(w_1^k s_f + w_2^k s_g\right)$

Example of forward pass



- $s_q = q\left(w_1^q s_f + w_2^q s_g + w_3^q s_k\right)$
- $y_1 = s_h$
- $y_2 = s_q$

Training a NN (backward)



In the backward pass, the computed error $\mathcal{L}(Y, V)$ is back-propagated to all the neurons, and the connexion weights are adjusted, depending on their contribution to the error:

$$w_{ij} \leftarrow w_{ij} - \varepsilon \frac{\partial \mathcal{L}}{\partial w_{ij}}$$

where ε is the learning rate.

Effective calculus of the contributions

But the contribution to the error of each connection weight $\frac{\partial \mathcal{L}}{\partial w_{ij}}$ is only directly computable for the connections to the output layer. We then use the chain derivation rule to recursively calculate the weights contributions for the previous layers. Let us first define our notations / conventions:

- We denote by w_{ij} the weight of the connection from neuron i to neuron j.
- We denote by s_i the output value of neuron i.
- We suppose all activation functions equal to g.
- We denote by $a_j = \sum_i w_{ij} s_i + b_j$ the activation value of neuron j, so that its output is $s_j = g(a_j)$.

Recursive calculus of the contributions

The update equation is written:

$$w_{ij} \leftarrow w_{ij} - \varepsilon \frac{\partial \mathcal{L}}{\partial w_{ij}}$$

By introducing a_j, the activation value of neuron j, we get:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial a_j} \times \frac{\partial a_j}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial a_j} \times s_i,$$

since $a_j = \sum_i w_{ij} s_i + b_j$. On the other hand we have:

$$\frac{\partial \mathcal{L}}{\partial a_j} = \frac{\partial \mathcal{L}}{\partial s_j} \times \frac{\partial s_j}{\partial a_j} = \frac{\partial \mathcal{L}}{\partial s_j} \times g'(a_j),$$

since $s_j = g(a_j)$.

Recursive calculus of the contributions

Finally, by developing the impact of the output of neuron j on all the inputs k of the next layer:

$$\frac{\partial \mathcal{L}}{\partial s_j} = \sum_k \frac{\partial \mathcal{L}}{\partial a_k} \frac{\partial a_k}{\partial s_j} = \sum_k \frac{\partial \mathcal{L}}{\partial a_k} w_{jk},$$

since $a_k = \sum_j w_{jk} s_j + b_k$. As a conclusion, **it is sufficient** to know the error gradient **on the last layer** to compute it **on all the previous layers by recursivity**.

Calculus for the last layer

Now for the last layer the calculus is straightforward.
Let us assume to simplify a quadratic loss function:

$$\mathcal{L}(Y, V) = \frac{1}{2}||Y - V||^2.$$

For the last layer we get $s_j = y_j$ and then: $\mathcal{L} = \frac{1}{2}\sum_j (s_j - v_j)^2$, we then get:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial a_j} = & \frac{\partial \mathcal{L}}{\partial s_j}\frac{\partial s_j}{\partial a_j} = & (s_j - v_j) \times g'(a_j) \\ \frac{\partial \mathcal{L}}{\partial w_{ij}} = & \frac{\partial \mathcal{L}}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = & (s_j - v_j) \times g'(a_j) \times s_i \end{cases}$$

Updating the biases

The biases need also to be updated:

$$b_j \leftarrow b_j - \varepsilon \frac{\partial \mathcal{L}}{\partial b_j}$$

It then suffices to notice that:

$$\frac{\partial \mathcal{L}}{\partial b_j} = \frac{\partial \mathcal{L}}{\partial a_j} \times \frac{\partial a_j}{\partial b_j} = \frac{\partial \mathcal{L}}{\partial a_j} \text{ because } a_j = \sum_i w_{ij} s_i + b_j$$

Then on the last layer:

$$\frac{\partial \mathcal{L}}{\partial b_j} = (s_j - v_j) \times g'(a_j)$$

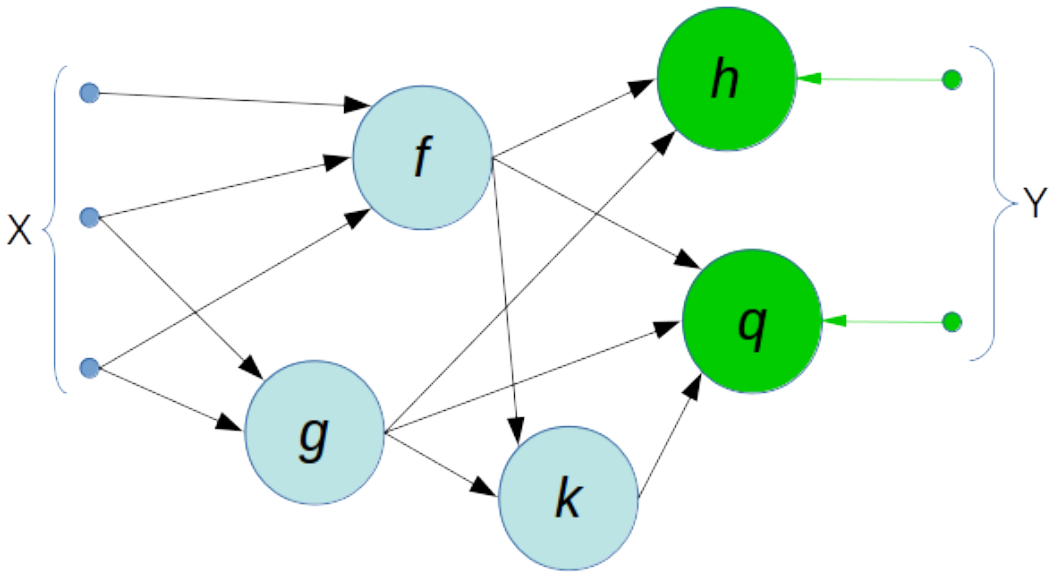
Gradient Backpropagation Algorithm

Finally, the backward pass algorithm is written (for a quadratic loss function $\mathcal{L}$):

Gradient Backpropagation $W \leftarrow \text{BackProp}(W, Y, V)$

- For all neuron j from the output layer:
  - compute the error $\Delta_j = (s_j - v_j) \times g'(a_j)$
- For all previous layer l:
  - For all neuron i from layer l :
    - compute the error $\Delta_i = \left(\sum_k \Delta_k w_{ik} \right) \times g'(a_i)$
- For all connection (i, j) of the network:
  - update the weight $w_{ij} \leftarrow w_{ij} - \varepsilon \times s_i \times \Delta_j$
  - update the bias $b_j \leftarrow b_j - \varepsilon \times \Delta_j$

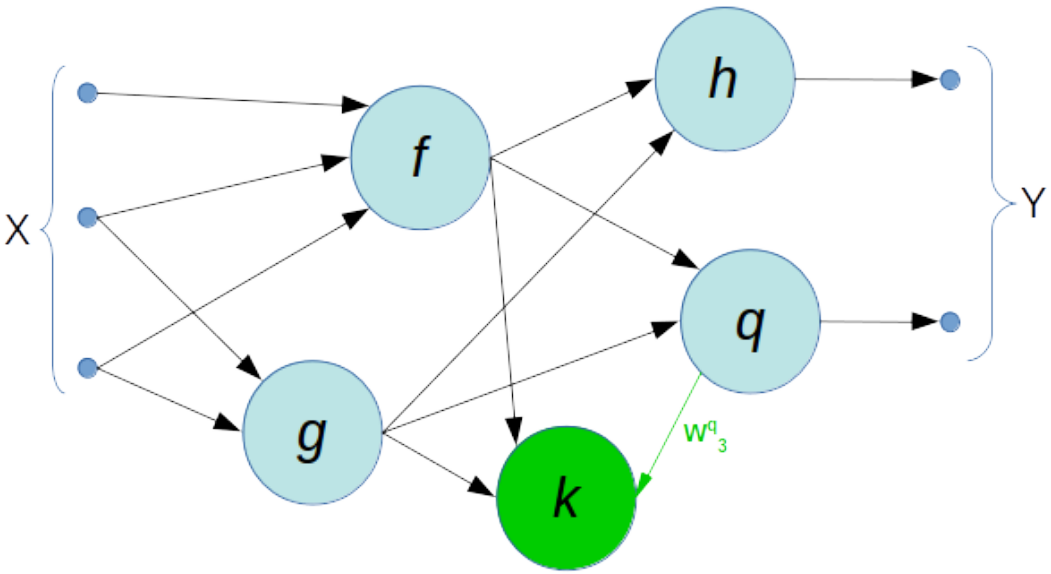
Example of backward pass



Calculate the error on the output layer, *in parallel*:

- $\Delta_h = (y_1 - v_1) \times h'(a_h)$
- $\Delta_q = (y_2 - v_2) \times q'(a_q)$

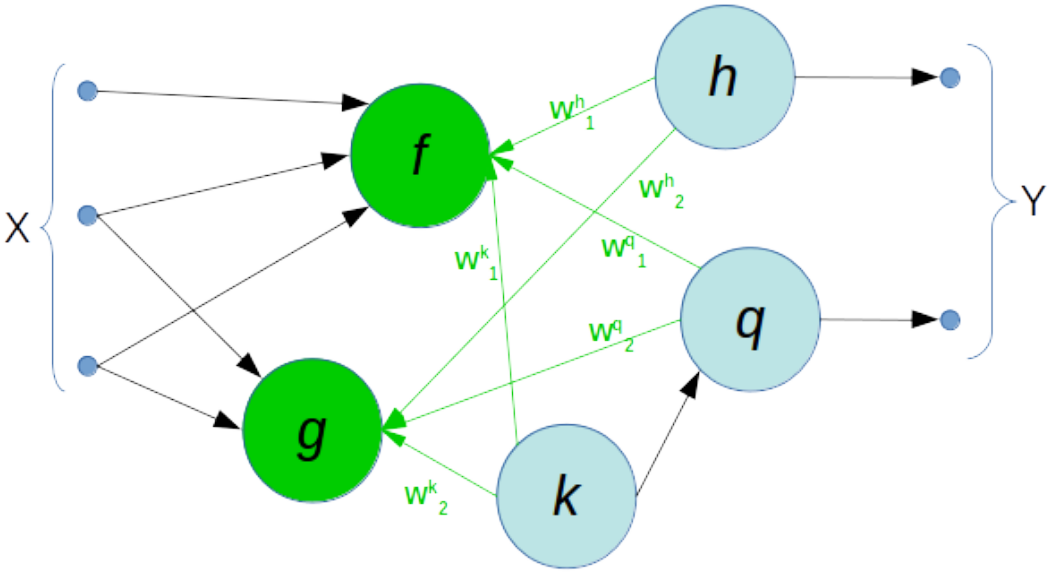
Example of backward pass



Intermediate layer:

- $\Delta_k = (\Delta_q w_3^q) \times k'(a_k)$

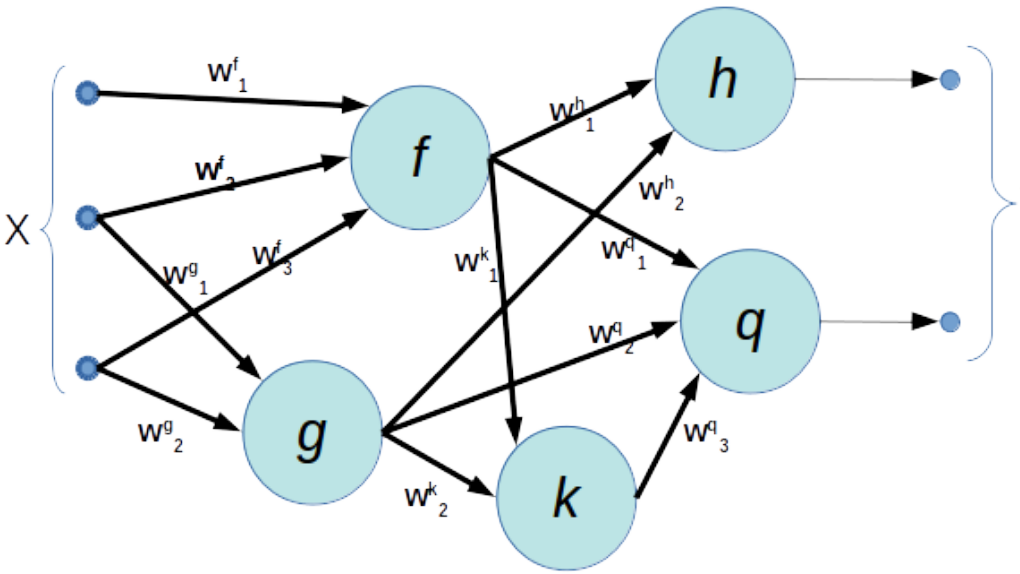
Example of backward pass



Input layer, *in parallel*:

- $\Delta_f = \left(\Delta_h w_1^h + \Delta_q w_1^q + \Delta_k w_1^k\right) \times f'(a_f)$
- $\Delta_g = \left(\Delta_h w_2^h + \Delta_q w_2^q + \Delta_k w_2^k\right) \times g'(a_g)$

Example of backward pass



Finally, update all weights w_{ij}, *in parallel*:

- $w_{ij} \leftarrow w_{ij} - \varepsilon s_i \Delta_j$

Learning Algorithm: Stochastic Gradient Descent

SGD $W \leftarrow \text{Train}(W, \{X_i, V_i\})$

- Initialisation: $W \leftarrow \text{RANDOM}(]-\eta, +\eta[)$
- Repeat until convergence ($\star$):
  - For each learning sample k:
    - $Y_k \leftarrow \text{ForwardProp}(W, X_k)$
    - $W \leftarrow \text{BackProp}(W, Y_k, V_k)$

- Training is longer (1 BackProp per training example)
- Convergence is faster, i.e. less iterations $\star$ over the entire base ("epochs")
- Progression is irregular

Learning Algorithm: Batch Gradient Descent

BGD $W \leftarrow \text{Train}(W, \{X_i, V_i\})$

- Initialisation: $W \leftarrow \text{RANDOM}(]-\eta, +\eta[)$
- Divide the learning base $\mathcal{A} = \{X_i, V_i\}$ into batches $\mathcal{A}_b$
- Repeat until convergence $(\star)$:
  - For each batch b:
    - $\mathcal{L}_b \leftarrow \sum_{(X_k, V_k) \in \mathcal{A}_b} \mathcal{L}(\text{ForwardProp}(W, X_k), V_k)$
    - $W \leftarrow \text{BackProp}(W, \mathcal{L}_b)$

- Training is faster (1 BackProp per batch)
- Convergence is slower (More "epochs" $\star$)
- Progression is more regular

Some popular loss functions

L_2 Error

$$\mathcal{L}(Y, V) = \frac{1}{2}||Y - V||^2$$

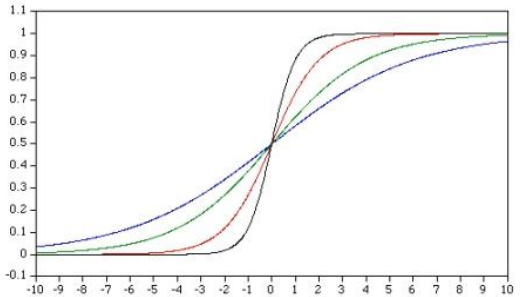
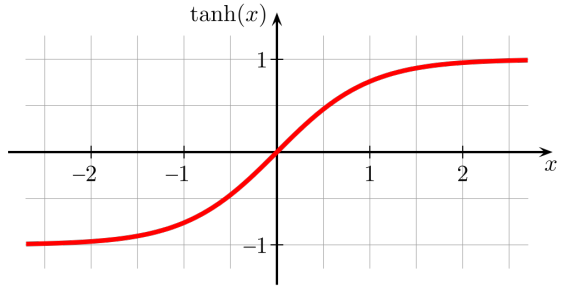
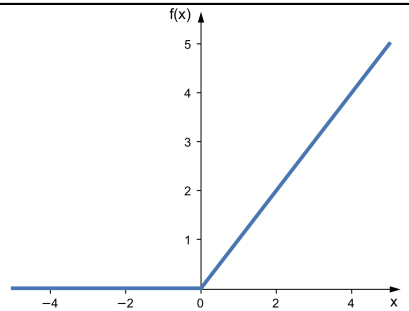
Mean Logarithmic Error ($V_i, Y_i \geq 0$)

$$\mathcal{L}(Y, V) = \frac{1}{n}\sum_{i=1}^{n}(\log(1 + V_i) - \log(1 + Y_i))^2$$

Kullback-Leibler Divergence, ($0 < V_i, Y_i < 1$)

$$\mathcal{L}(Y, V) = \sum_i V_i \frac{\log V_i}{\log Y_i}$$

Some usual activation functions

Sigmoid	$f_\lambda(x) = \frac{1}{1+e^{-\lambda x}}$	
	$f'_\lambda(x) = \lambda f_\lambda(x)(1 - f_\lambda(x))$	
TanH	$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$	
	$f'(x) = 1 - f(x)^2$	
ReLU	$f(x) = \frac{x+\lvert x\rvert}{2}$	
	$f'(x) = \frac{x+\lvert x\rvert}{2x}$	

Some variants to the optimisation scheme

Some variants to the Gradient Descent (GD) $w_{ij} \leftarrow w_{ij} - \varepsilon \frac{\partial \mathcal{L}}{\partial w_{ij}}$:

GD with Momentum

$$G_t = \beta G_{t-1} + (1-\beta)\frac{\partial \mathcal{L}}{\partial w_{ij}}; \beta \in [0,1]$$
$$w_{ij}^t = w_{ij}^{t-1} - \varepsilon G_t$$

- The Momentum limits direction fluctuations of the gradient between 2 iterations (batches) $t-1$ et t.
- $\beta \to O$: Classic GD.
- $\beta \to 1$: The gradient keeps its direction.

Some variants to the optimisation scheme

Some variants to the Gradient Descent (GD) $w_{ij} \leftarrow w_{ij} - \varepsilon \frac{\partial \mathcal{L}}{\partial w_{ij}}$:

Adam

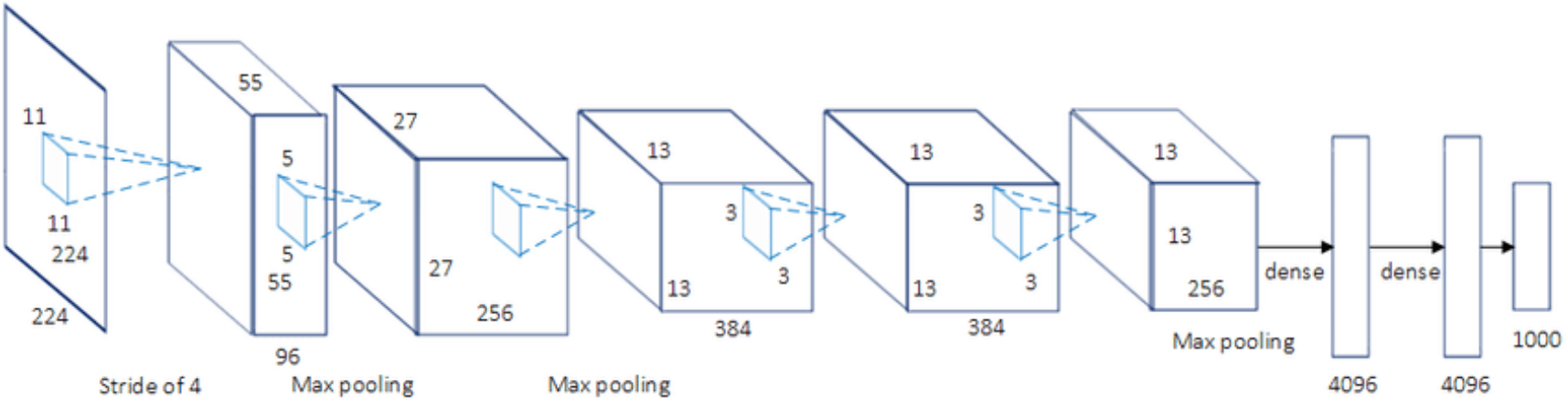
$$G_t = \beta_1 G_{t-1} + (1-\beta_1)\frac{\partial \mathcal{L}}{\partial w_{ij}}; \beta_1 \in [0,1]$$

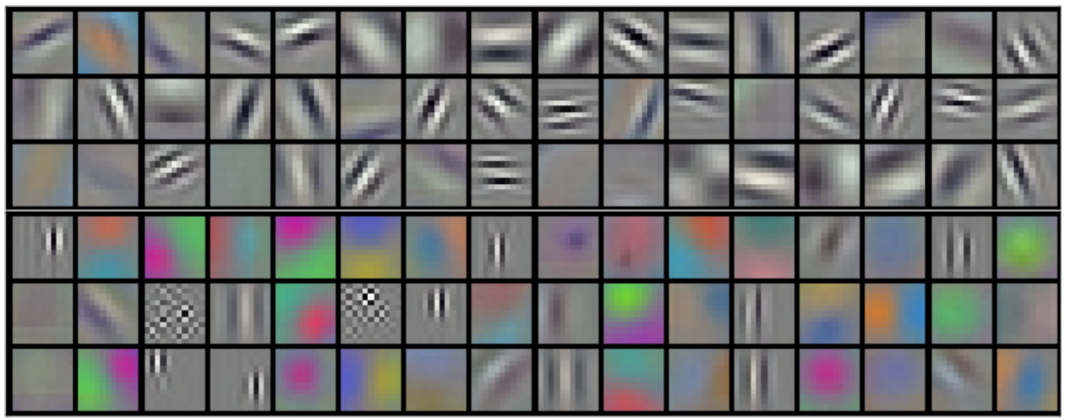
$$K_t = \beta_2 K_{t-1} + (1-\beta_2)\left(\frac{\partial \mathcal{L}}{\partial w_{ij}}\right)^2; \beta_2 \in [0,1]$$

$$w_{ij}^t = w_{ij}^{t-1} - \varepsilon \frac{G_t}{\sqrt{K_t + \eta}}$$

- Adam also limits the temporal direction fluctuations of the gradient, and also adapts the learning speed to the curvature (second derivative) of the loss function.
- 2 additional hyperparameters: $\{\beta_1, \beta_2\}$.

Deep CNN for image classification



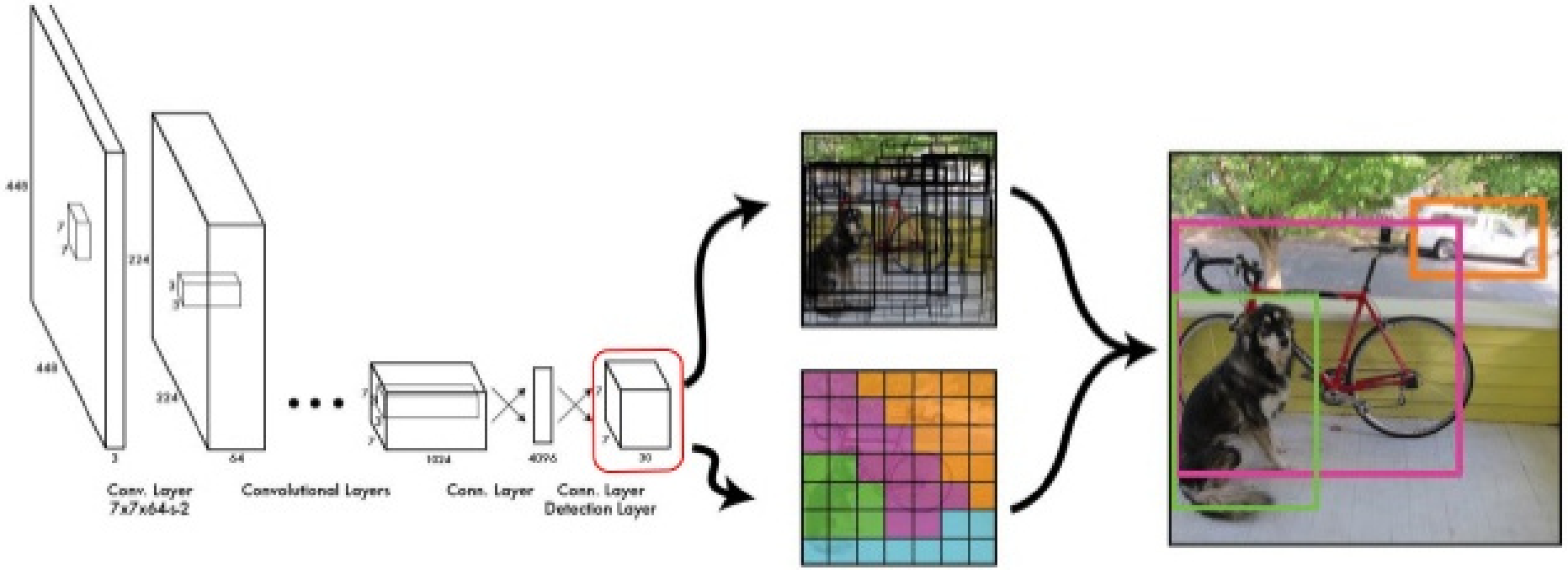


The 96 first 11×11 convolutions

AlexNet **[Krizhevsky12]**:

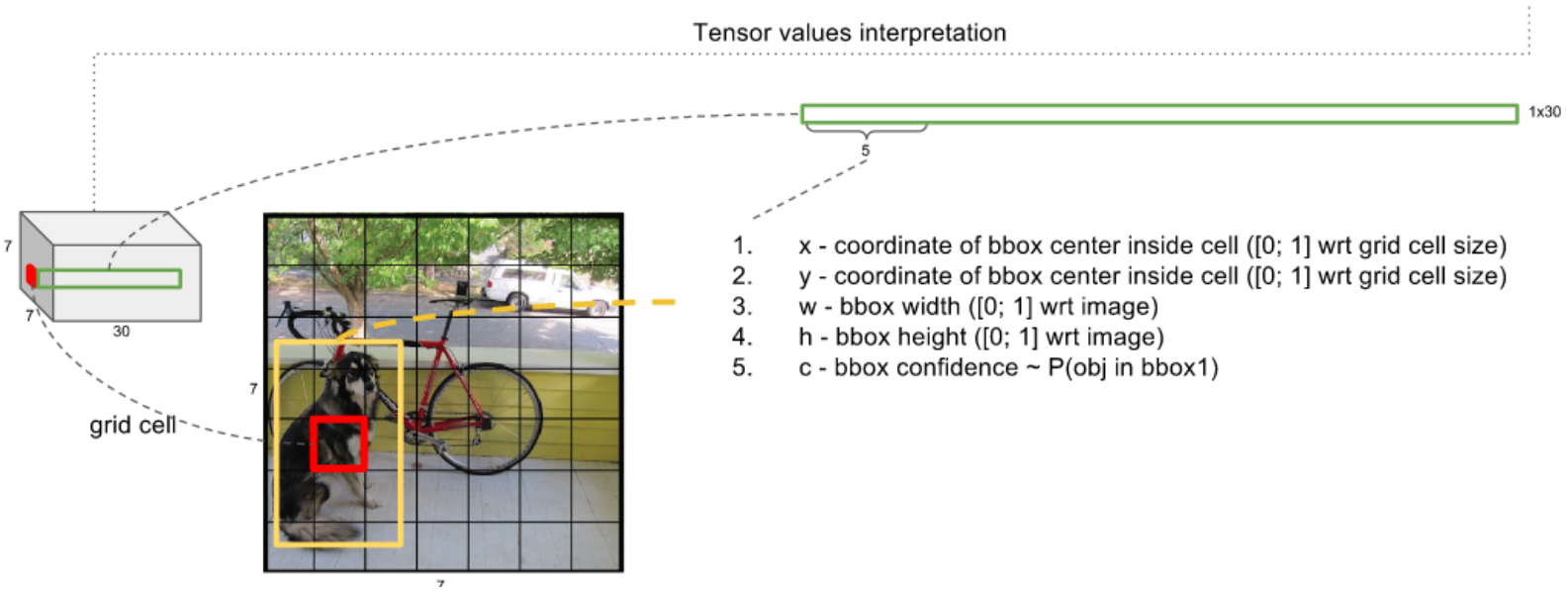
- Trained on ImageNet ILSVRC-2010
- 1,2M learning images
- 1000 Classes

Deep CNN for object detection



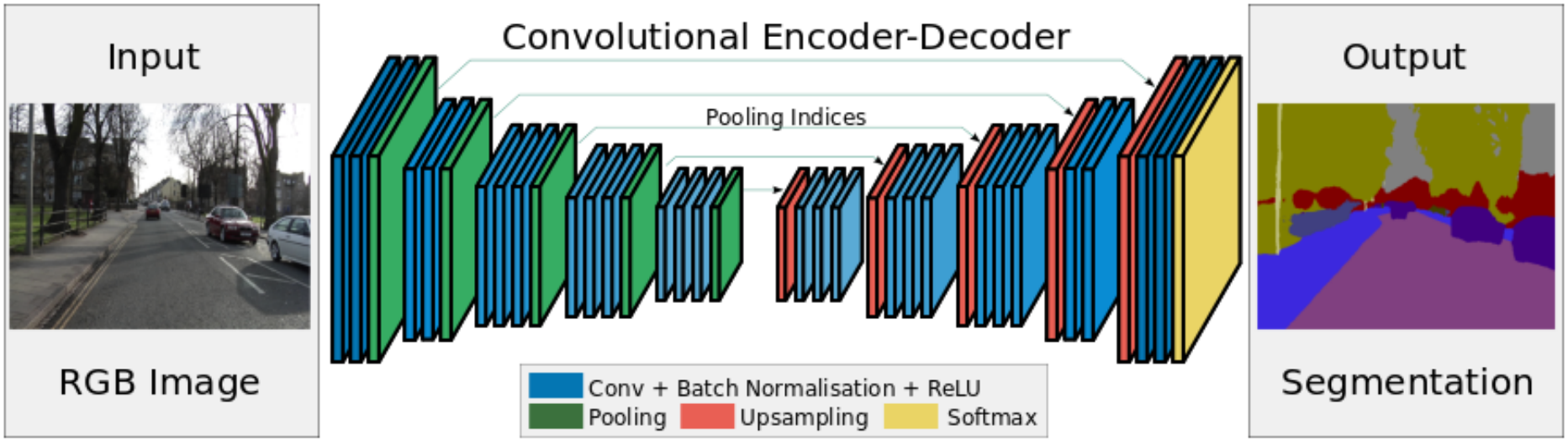
YOLO **[Redmon16]**

Deep CNN for object detection



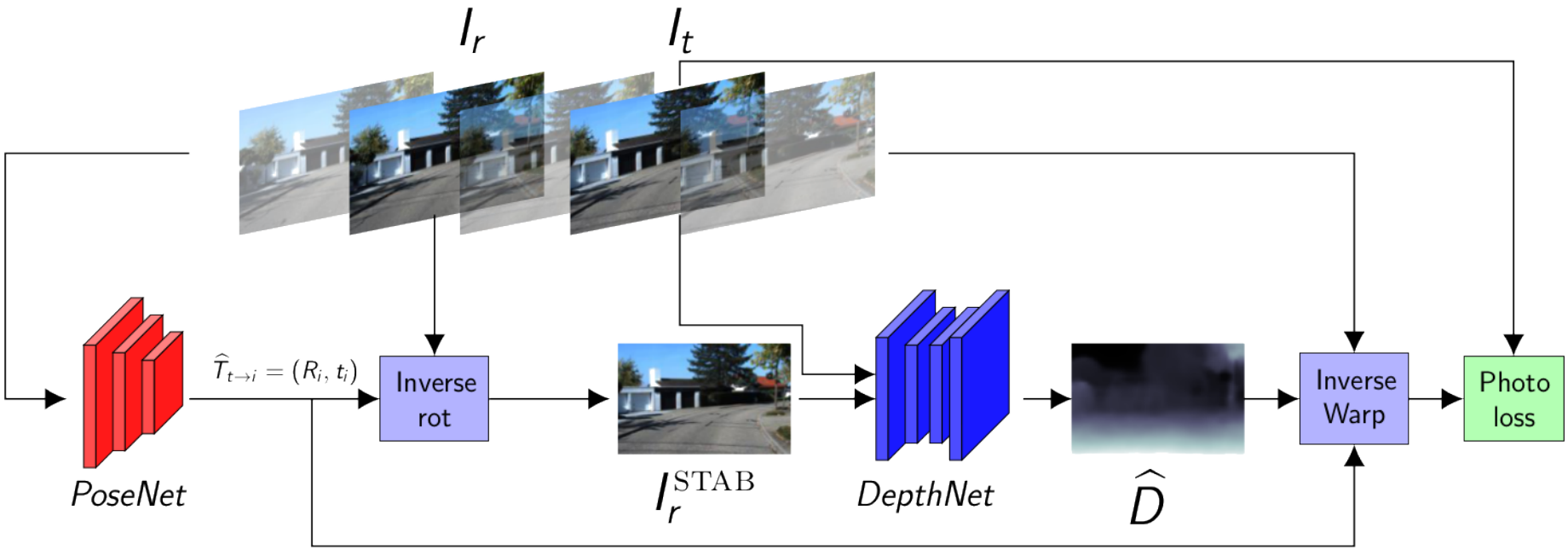
YOLO **[Redmon16]**

Deep CNN for semantic segmentation



SegNet **[Badrinarayanan15]**

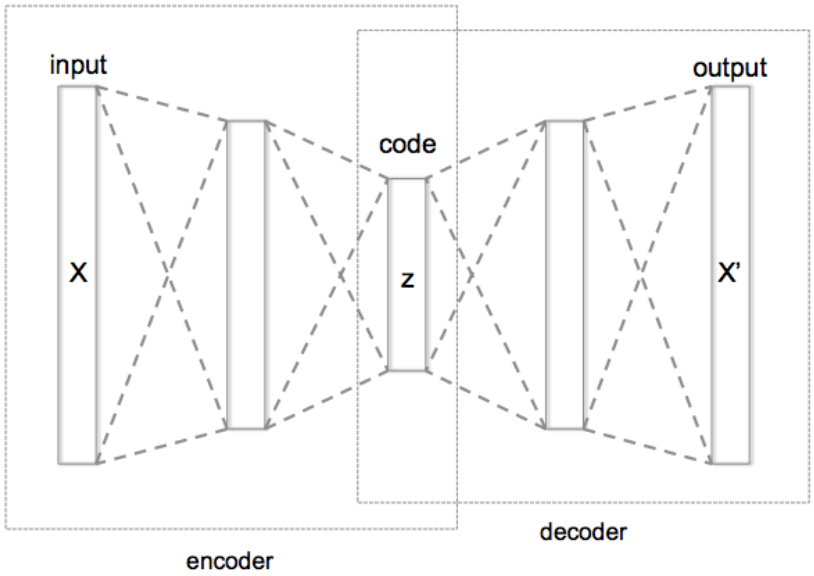
Deep CNN for 3d reconstruction



$$\forall i,\ t_i^{\text{NORM}} = t_i \frac{T_0}{\epsilon + \|t_r\|}$$

PoseNet + DepthNet **[Pinard18]**

Self-supervised Deep CNN



Autoencoder (Figure: Wikipedia)

By using a loss function $\mathcal{L}(\hat{X}, X)$, the autoencoder learns to reconstruct a copy $\hat{X}$, or a enhanced (denoised, restored...) version of its input X.

Conclusion on the NN

Free lunch? Not yet!

- Empirism / Handcraft!
- Annotated Image Datasets!
- Explainability...?
- Choice for $\mathcal{L}$...?
- Domain adaptation...?
- Online learning *vs* catastrophic Forgetting...

Outline of the lecture

General Conclusion

Takeaway from this lecture:

- Dimension Reduction
  - Statistical: PCA
  - Metric: K-means
- Supervised Learning
  - Bayesian Learning
  - K-nearest neighbours
  - SVM
  - Random Forest
  - CNN / $\mathcal{L}(Y, V)$
- Unsupervised Learning
  - K-means (VBoW)
  - CNN / $\mathcal{L}(\hat{X}, X)$

General Conclusion

To be studied as complement:

- Pratical Learning
  - Dataset Partition into Learning / Validation / Test sets
  - Cross-validation
- Evaluation
  - Classification: Error rate
  - Classification: Confusion Matrix
  - Classification: Accuracy / Recal Curves...
  - Regression: L_n Norms...
  - Detection: Jaccard Index, IoU...
- Overfitting and Regularisation
  - Dropout - Batch Normalisation - Weight Decay

References

[Bishop06] C. BISHOP
Pattern recognition and machine learning
Information science and statistics, Springer, 2006

[Barber12] D BARBER
Bayesian Reasoning and Machine Learning
Cambridge University Press New York, 2012

[Shlens05] J. SHLENS
A Tutorial on Principal Component Analysis
Salk Institute for Biological Studies

References

[Turk91] M. TURK, A. PENTLAND
Eigenfaces for recognition
Journal of Cognitive Neuroscience Vol.3(1), pp 71-86, 1991

[Burges98] C.J.C BURGES
A Tutorial on Support Vector Machines for Pattern Recognition
Data Mining and Knowledge Discovery, 2, pp121–167, 1998

[Breiman01] L. BREIMAN
Random Forests
Machine Learning 45(1), pp5-32, 2001

References

[Goodfellow16] I. GOODFELLOW, Y. BENGIO, A. COURVILLE
Deep Learning
MIT Press, http://www.deeplearningbook.org, 2016

[Krizhevsky12] A. KRIZHEVSKY, I SUTSKEVER, G.E. HINTON
Imagenet classification with deep convolutional neural networks
Advances in Neural Information Processing Systems (NIPS), 2012

[Redmon16] J. REDMON *et al.*
You Only Look Once: Unified, Real-Time Object Detection
IEEE Conf. on Computer Vision and Pattern Recognition, pp779-788, 2016

References

 **[Badrinarayanan15]** V. BADRINARAYANAN, A. KENDALL, R. CIPOLLA
SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation
CoRR abs 2015

 **[Pinard18]** C. PINARD, L. CHEVALLEY, A. MANZANERA, D. FILLIAT
Learning Structure-from-Motion from Motion
ECCV 2018 Workshops - LNCS vol.11131 Springer, 2019