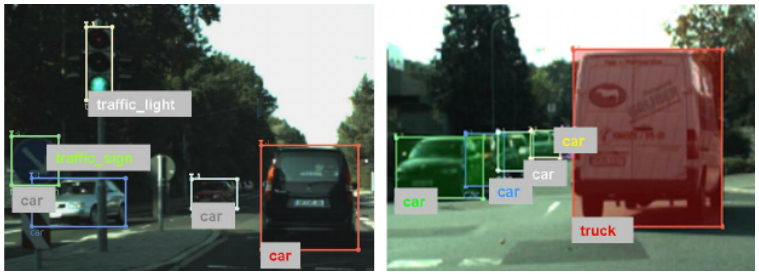
ENSTA 2e année - Cours MI 204
Classification et Apprentissage de Données Images

Antoine Manzanera
ENSTA Paris



Objectifs du cours

Ce cours est une introduction à l'apprentissage d'images à travers un choix partiel de techniques de classification et de modèles d'apprentissage utiles en vision par ordinateur. Le but est d'aborder les concepts clefs suivants :

- Réduction de dimension des descripteurs
- Apprentissage "basse dimension"
- Réseaux de Neurones Convolutionnels

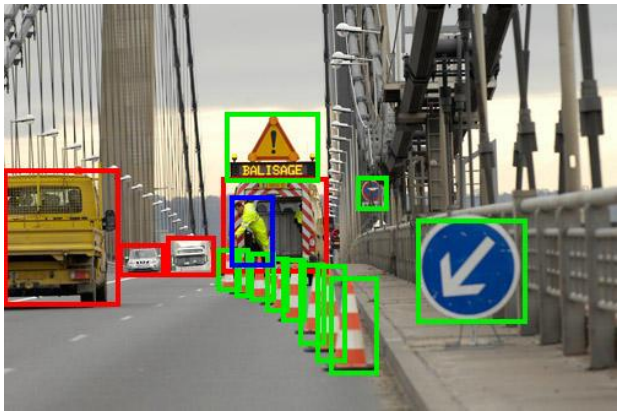
Plan du cours

Pourquoi l'apprentissage ?

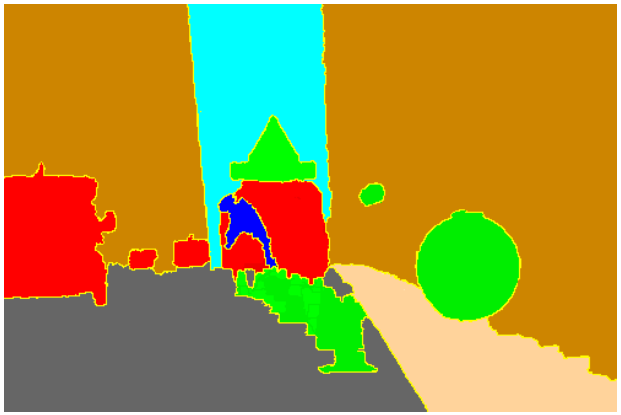
- En vision par ordinateur, la *décision* repose rarement sur une approche purement analytique et impérative ; elle s'appuie généralement sur une *connaissance préalable présente en mémoire* du système.
- Cette mémoire a été acquise par l'*expérience* (ou l'*entraînement*) subie par le système *avant* (apprentissage hors-ligne) et/ou *pendant* (apprentissage en ligne) sa mise en fonction.
- L'entraînement est réalisée à partir de données *exemplaires* pour lesquelles la décision attendue peut être fournie (apprentissage *supervisé*) ou absente (apprentissage *non supervisé*).

Quelles décisions en Vision par ordinateur ?

Classification



Détection = Localisation + Classification



Segmentation sémantique

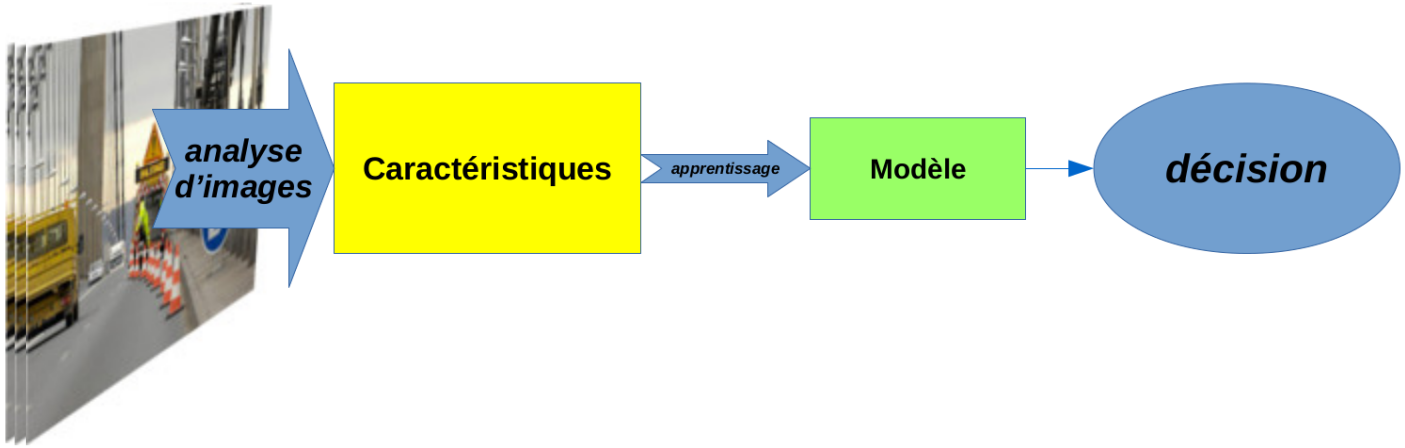
Légendage automatique

Classification *vs* Régression

- L'apprentissage peut être modélisé comme la construction d'une *fonction* d'un *espace d'observation* (données extraites de l'image) vers un *espace de décision*.
- L'espace de décision est souvent discret et dépourvu de métrique (ensemble de classes...) : on parle de *Classification*.
- L'espace de décision peut parfois être continu et structuré (par exemple une image, dans un problème de restauration ou de super-résolution...), on parle alors plutôt de *Régression*.

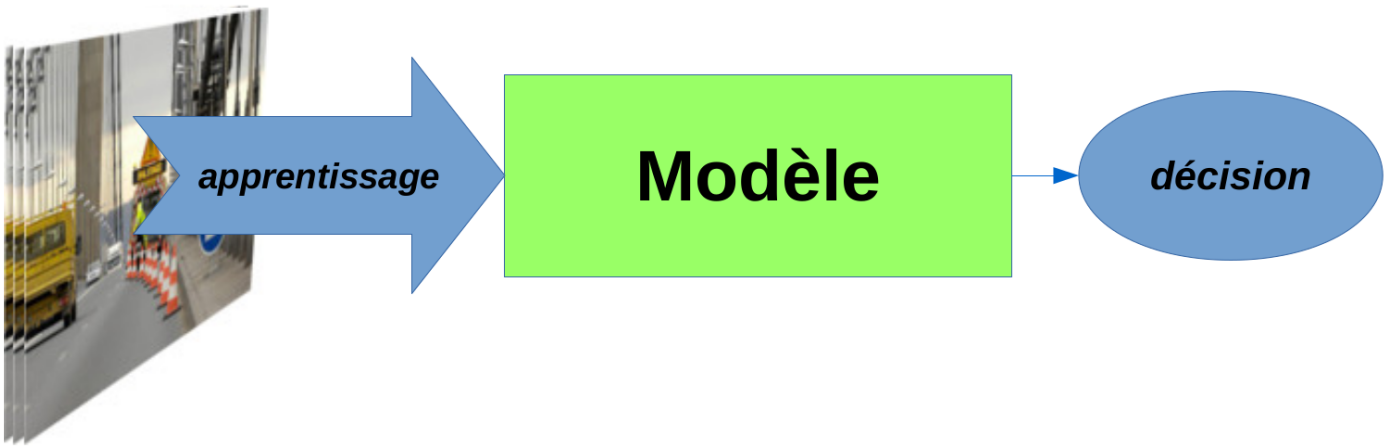
Caractéristiques explicites...

Dans une approche traditionnelle, l'espace d'observation est constitué de *caractéristiques ad hoc* extraites des images par traitement :



...ou Caractéristiques implicites

Depuis les années 2010, de plus en plus d'approches telles que les réseaux convolutionnels profonds appliquent un apprentissage *de bout en bout*, i.e. où l'espace d'observation est l'image elle-même. Les caractéristiques sont donc *apprises* au même titre que les autres niveaux de représentation :



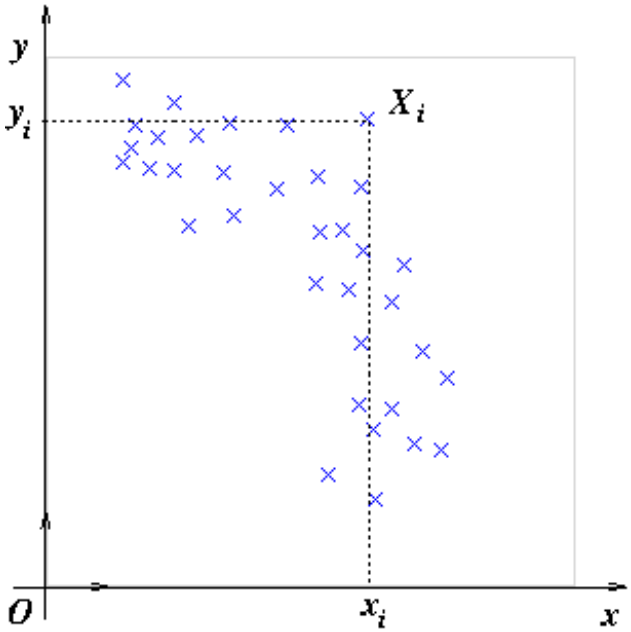
Plan du cours

Réduction de l'espace d'observation

Dans l'apprentissage "basse dimension", il est souvent nécessaire de réduire la dimension des caractéristiques pour : (1) Réduire la complexité et (2) Limiter la redondance entre les caractéristiques.

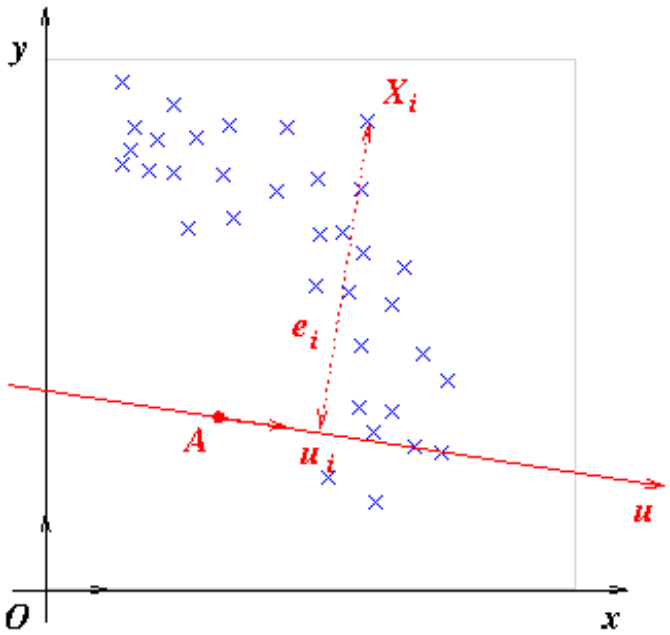
- **Analyse en Composantes Principales** (*ACP*) : Approche algébrique de *réduction de la dimension* aux directions de *variance maximale*.
- **Regroupement** (*Clustering*) : Approche métrique de *quantification du nombre de caractéristiques* en les limitant à un petit nombre de représentants.

Analyse en Composantes Principales



Soit $\{X_i\}_{i=1...n}$ un nuage de n points dans un espace vectoriel de dimension $d \geq 2$.

Analyse en Composantes Principales

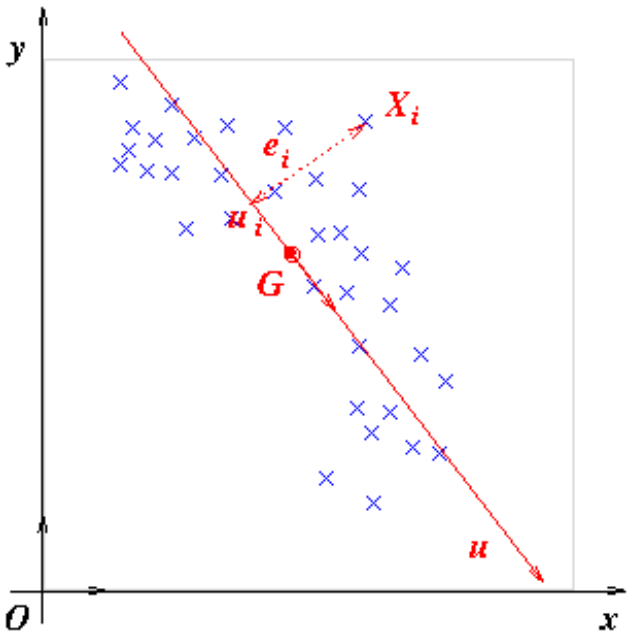


On cherche une représentation *mono-dimensionelle optimale* des vecteurs $\{X_i\}$, c'est-à-dire un repère $(A, \mathbf{u})$ qui minimise l'erreur totale :

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

où A est un point et $\mathbf{u}$ un vecteur unitaire de l'espace.

Analyse en Composantes Principales



- La solution est fournie par le repère $(G, \mathbf{u})$, où G est le *centre de gravité* (moyenne), et $\mathbf{u}$ est la *composante principale* des $\{X_i\}$.
- $\mathbf{u}$ est donné par le *vecteur propre* de la *matrice de covariance* des $\{X_i\}$ associé à *la plus grande valeur propre*.
- $\mathbf{u}$ correspond à la *direction de plus grande variance* des données $\{X_i\}$.

Analyse en Composantes Principales

Le procédé s'étend naturellement à plusieurs composantes principales : si on note $\mathbf{X}$ la matrice $(d \times n)$ composée de l'ensemble des n vecteurs d'observation $\{X_i\}$ rangés en colonnes :

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

La matrice $(d \times d)$ de covariance des observations s'écrit :

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

Analyse en Composantes Principales

Algorithme ACP

- Calcul de la matrice de covariance $\mathbf{C} = \frac{1}{n}\mathbf{X}\mathbf{X}^t - GG^t$
- Calcul et tri des valeurs propres $(\lambda_1, \ldots, \lambda_d)$, telles que $\lambda_1 > \cdots > \lambda_d$, et de leurs vecteurs propres associés $(U_1, \ldots, U_d)$.
- Les k $(k << d)$ premiers vecteurs propres $\{U_1, \ldots, U_k\}$ sont les *composantes principales*.

- Les composantes principales sont les dimensions théoriques (pas toujours de sens physique !) qui "expliquent le mieux" la répartition des $\{X_i\}$ dans l'espace d'observations.
- La matrice $\mathbf{C}$ est symétrique $\Rightarrow$ les composantes principales sont orthogonales.

ACP pour la reconnaissance de visages

Images 32×32 d'apprentissage $\{X_i\}$

Visage "moyen" G

Images de *Derek Hoiem*, University of Illinois.

ACP pour la reconnaissance de visages

U_k

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

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

"Eigenfaces for Recognition"

[Turk91]

Images de *Derek Hoiem*,

University of Illinois.

Regroupement : Algorithme K-means

- L'algorithme de regroupement (clustering) K-means réalise une partition $\mathcal{C}$ de l'ensemble des observations $\{X_i\}$ en K classes C_k $(1 \leq k \leq K)$.
- Soit $\Pi = \{\pi_k\}$ un ensemble de K prototypes choisis dans l'espace d'observation et représentant chacune des classes.
- Soit d une distance dans l'espace des observations.

L'objectif est de minimiser la fonction de coût suivante :

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

Regroupement : Algorithme K-means

L'objectif est de minimiser la fonction de coût suivante :

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

On connaît la solution :

- Si l'ensemble des prototypes Π est fixé :
  $C_k = \{X_i; \forall j \neq k, d(X_i, \pi_k) \leq d(X_i, \pi_j)\}$
- Si la partition en classes $\mathcal{C}$ est fixée : $\pi_k = \frac{1}{|C_k|} \sum_{X_i \in C_k} X_i$

Regroupement : Algorithme K-means

Le K-means est un algorithme itératif qui converge vers un minimum local de $J_{\mathcal{C}}^{\Pi}$ en ajustant alternativement $\mathcal{C}$ et Π :

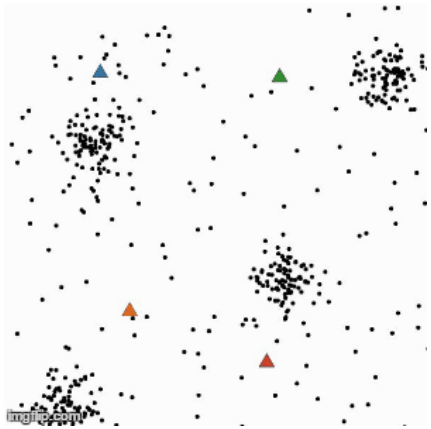
Algorithme des K-means

Initialisation : On choisit un ensemble de K prototypes $\Pi = \{\pi_k\}$.

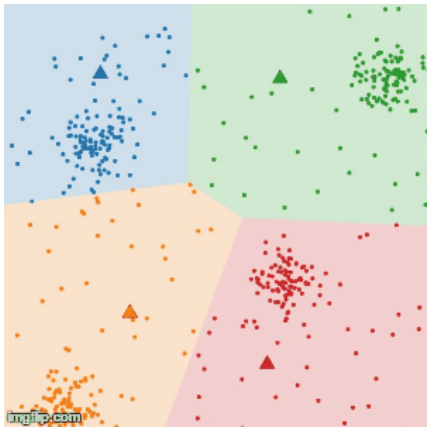
Répéter jusqu'à convergence :

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

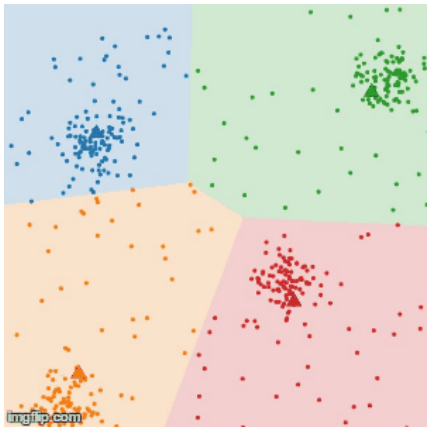
Regroupement : Algorithme K-means



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



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

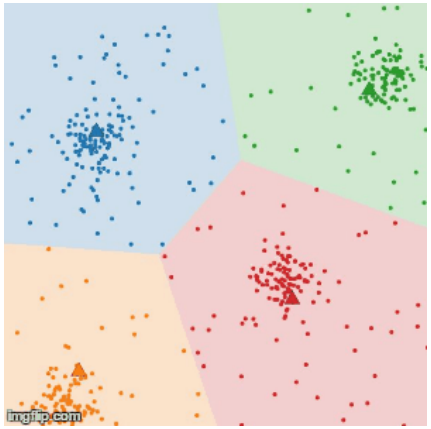


$[\Pi]^{(1)}$

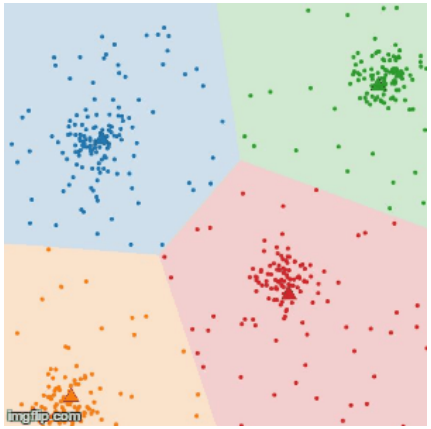
- Importance de l'initialisation !
- Partition $\mathcal{C} \leftrightarrow$ Diagramme de Voronoï

Images : *Wikipedia*

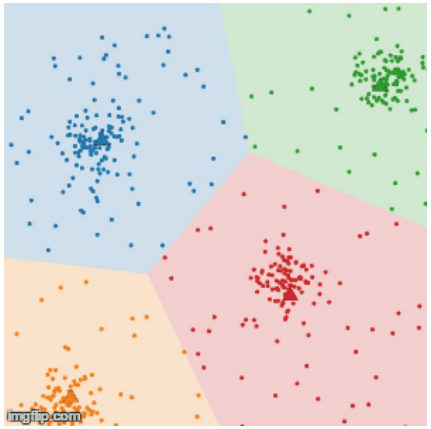
Regroupement : Algorithme K-means



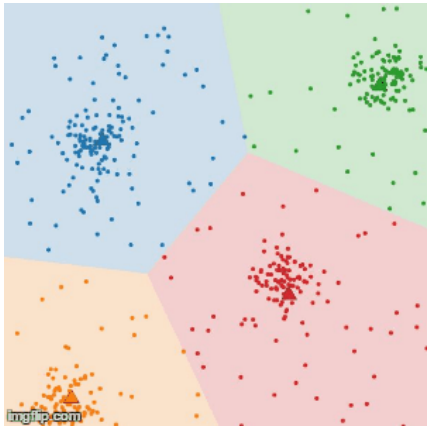
$[\mathcal{C}]^{(2)}$



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



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



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

- Importance de l'initialisation !
- Partition $\mathcal{C} \leftrightarrow$ Diagramme de Voronoï

Images : *Wikipedia*

Plan du cours

Classification bayésienne

Les données d'observation X, ainsi que les classes c sont modélisées comme des vecteurs (resp. variables) aléatoires. On appelle :

- $P(X)$ et $P(c)$ les probabilités *a priori*.
- $P(X, c)$ la probabilité *jointe*.
- $P(X/c)$ la *vraisemblance* conditionnelle de la donnée X.
- $P(c/X)$ la probabilité *a posteriori* de la classe c.
- **Apprentissage** : Construire les lois de probabilités à partir d'un ensemble de données étiquetées $\{X_i, c(X_i)\}$.
- **Inférence** : Associer à une donnée X de test sa classe $c^\star$.

Inférence bayésienne

- **Inférence** : Associer à une donnée X de test sa classe $c^\star$.

Critère du Maximum de Vraisemblance (ML)

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

Inférence bayésienne : Critère du Maximum a Posteriori (MAP)

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

Apprentissage bayésien

- **Apprentissage** : On estime la loi jointe $P(X, c)$, et donc la loi *a posteriori* à partir des données d'apprentissage $\{X_i\}$.

En effet $P(X, c) = P(X).P(c/X) = P(c).P(X/c)$, et donc :

Théorème de Bayes

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

- $P(c)$ est la probabilité d'occurrence de la classe, fixée arbitrairement, ou par des statistiques sur les données d'apprentissage.
- $P(X/c)$ est fournie par les statistiques sur les données *étiquetées* (i.e. dont la classe est connue = apprentissage supervisée).

Classification bayésienne

- Le critère du MAP devient:

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

- Finalement c'est la pondération par la loi *a priori* qui distingue le MAP du ML.
- Le modèle de classification consiste à estimer empiriquement $P(X/c)$ par des statistiques sur $\{X_i\}$.

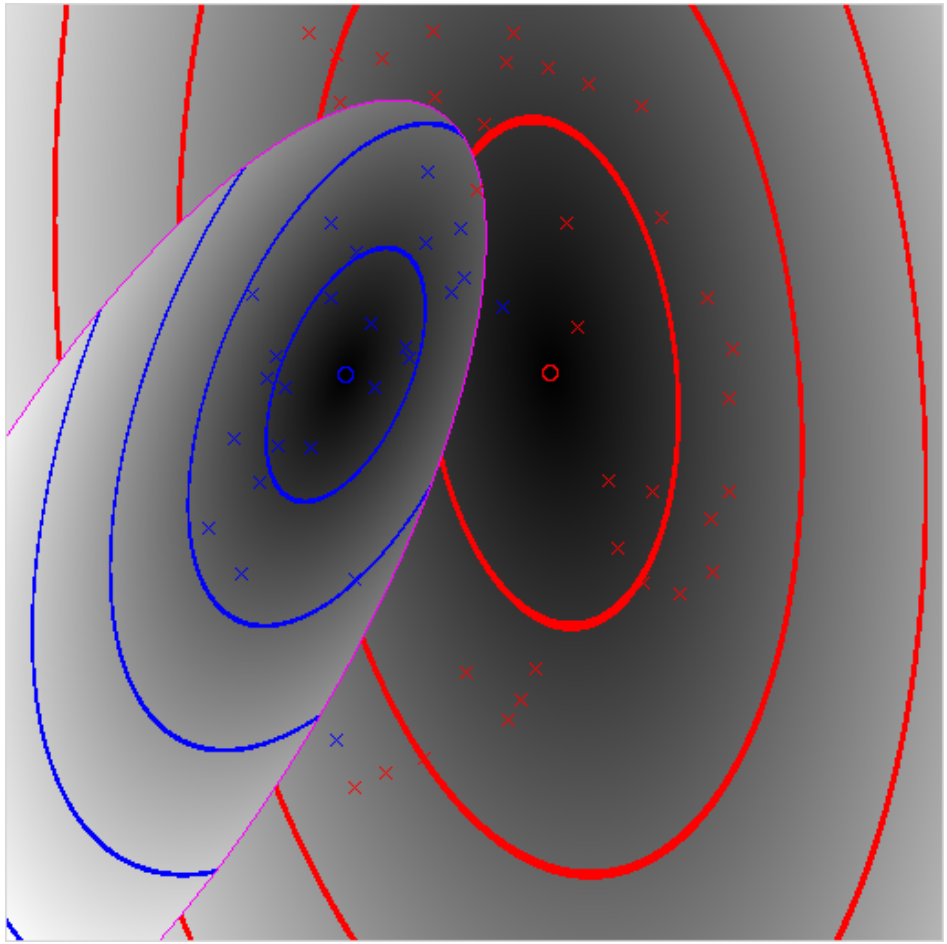
Modèle bayésien multivarié

On utilise souvent un modèle probabiliste simplifié de la distribution de $P(X/c)$, comme la gaussienne :

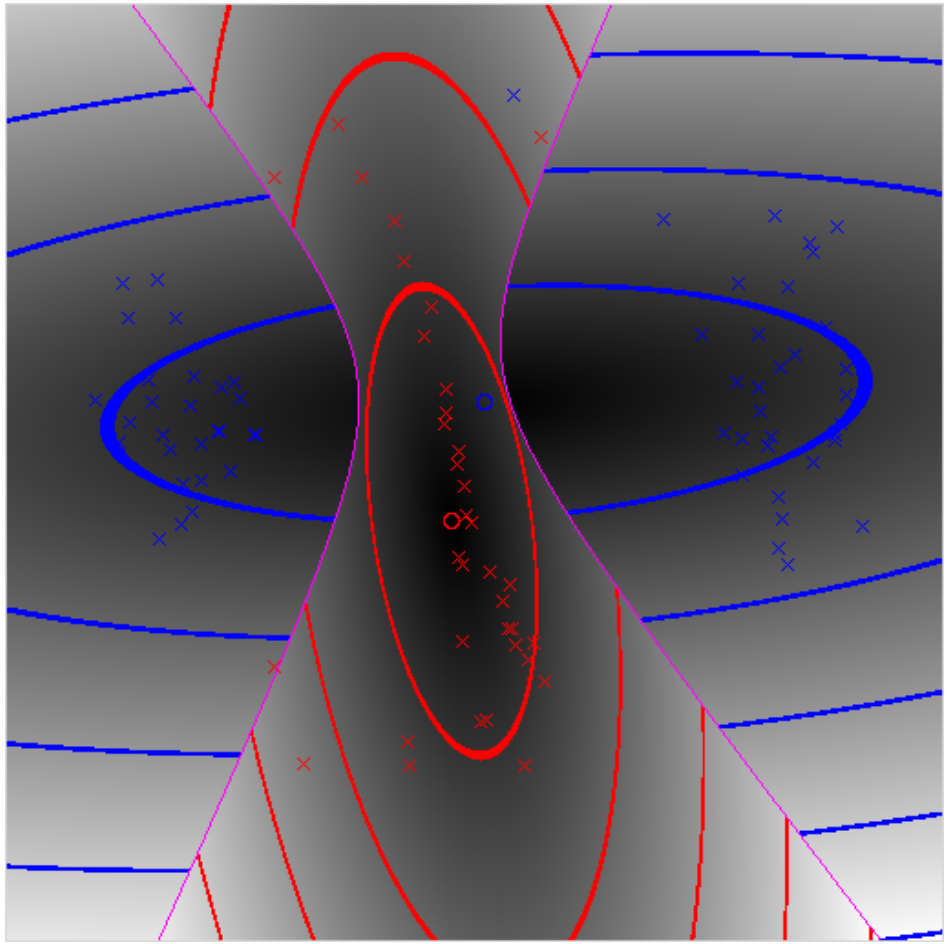
$$P(X/c) = f_c(X) = \frac{1}{\sqrt{(2\pi)^d \det \mathbf{C}_c}} \exp\left(-\frac{1}{2}(X - G_c)^t \mathbf{C}_c^{-1}(X - G_c)\right)$$

où G_c et $\mathbf{C}_c$ sont respectivement le vecteur moyen et la matrice de covariance des données d'apprentissage étiquetées à la classe c.

Classification Bayésienne : Modèles Gaussiens



Classification Bayésienne : Modèles Gaussiens



Modèle bayésien naïf

Le modèle bayésien naïf fait l'hypothèse de l'indépendance des différentes dimensions de X conditionnellement à la classe c :

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

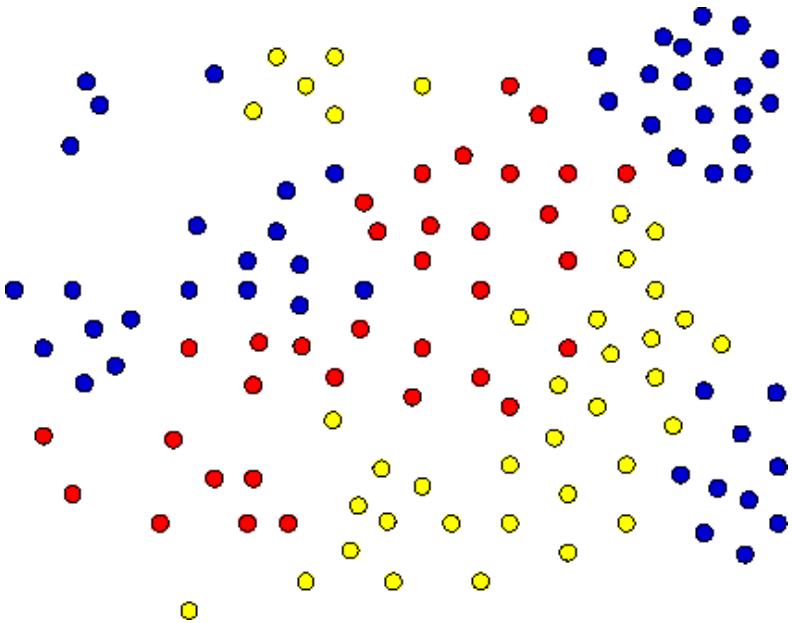
et finalement :

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

- Permet d'estimer des distribution scalaires, potentiellement complexes (mélanges de lois, histogrammes, ...).
- Peut parfois mieux fonctionner qu'un modèle multivarié unimodal.

Plan du cours

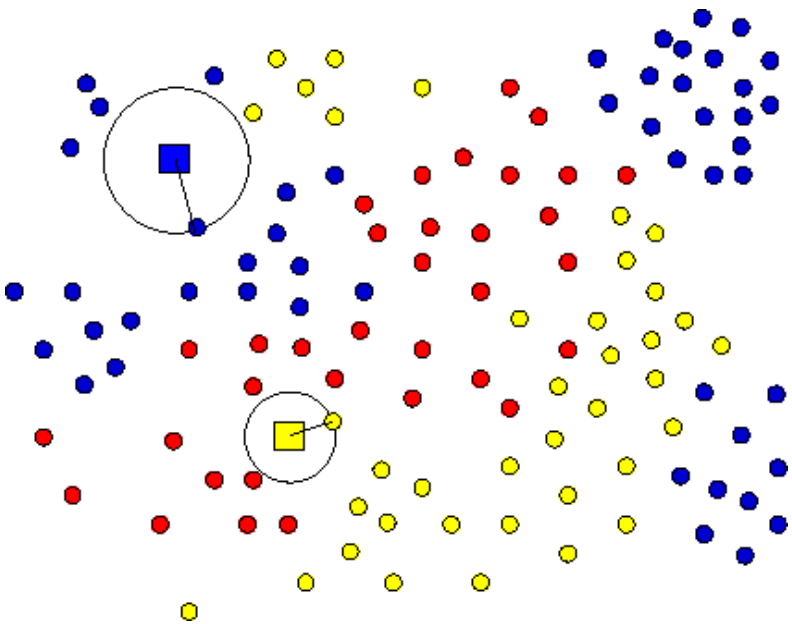
Classification aux plus proches voisins



Dans la classification aux plus proches voisins (k-ppv, ou k-NN), on considère :

- un ensemble de points *étiquetés* dans l'espace d'observation $\{X_i, c(X_i)\}$,
- la classe $c(X_i)$ est fournie par supervision,
- d une distance dans l'espace d'observation.

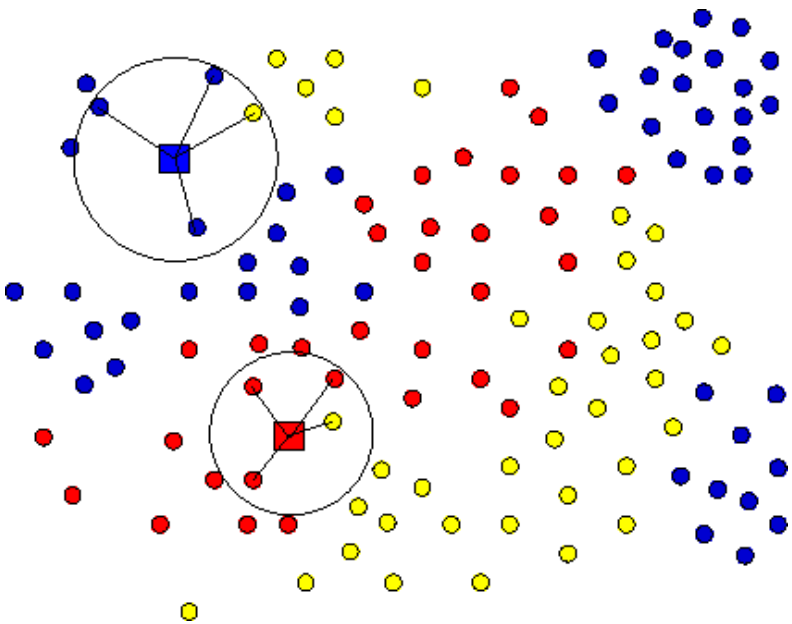
1-ppv



La classification 1-ppv d'un vecteur X inconnu consiste à lui attribuer la classe de son plus proche voisin parmi les vecteurs d'apprentissage :

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

k-ppv



La classification k-ppv d'un vecteur X inconnu consiste à lui attribuer la classe la plus représentée parmi ses k plus proches voisins dans les vecteurs d'apprentissage.

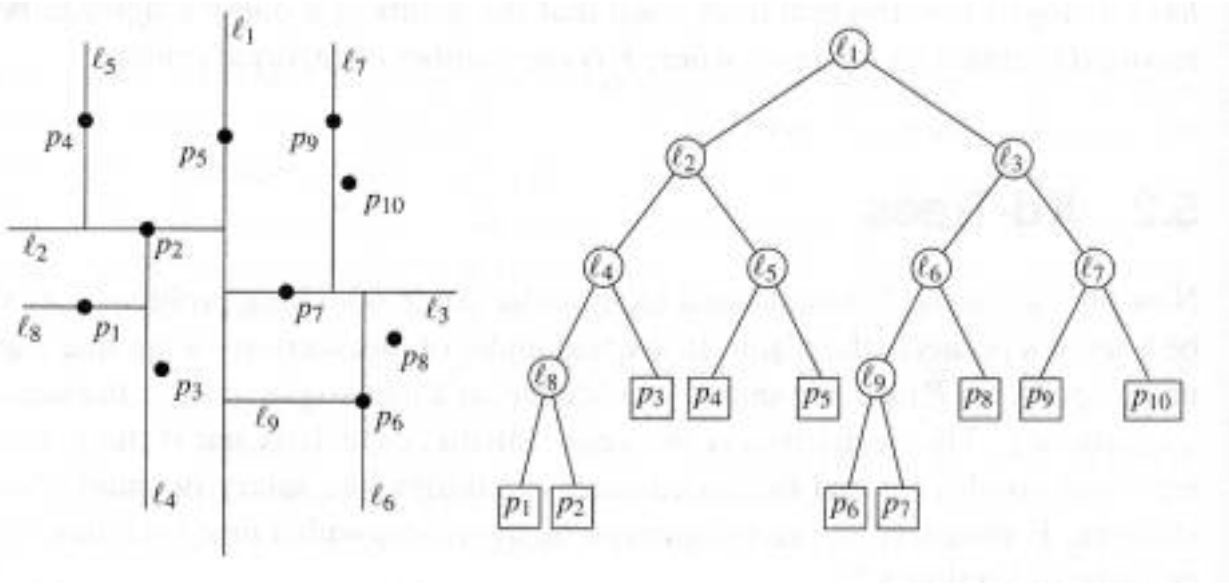
Avantages des k-ppv

- *Flexible :* s'adapte à tous types de distribution dans l'espace des observations, sans *a priori* probabiliste.
- *Simple :* le modèle est simplement l'ensemble des vecteurs et leurs classes.
- *Général :* s'étend trivialement à la *régression* d'une fonction :

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

Limitations des k-ppv

- *Coût en mémoire :* le modèle peut être de très grande taille.
- *Coût en calcul :* la complexité de la recherche exhaustive de ppv dépend linéairement du nombre de données n, et de leur dimension d. La complexité moyenne peut être réduite en $\mathcal{O}(\log n)$ par l'utilisation de Kd-trees :

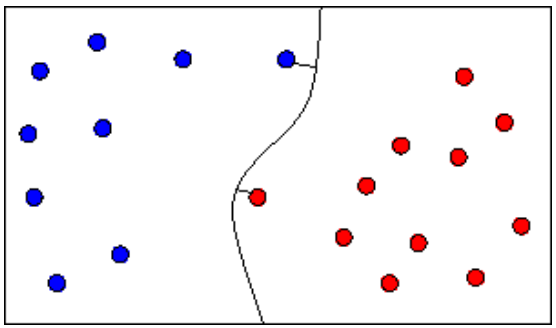


Limitations des k-ppv

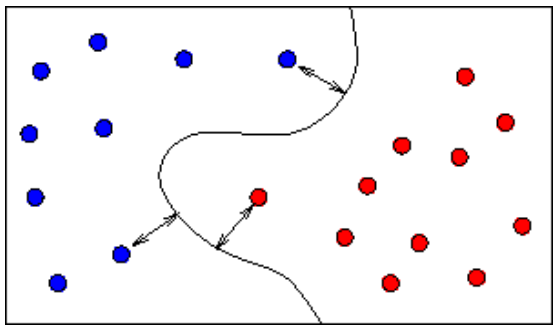
- *Malédiction de la dimensionalité :* le nombre d'échantillons nécessaires pour estimer avec la même fiabilité une distribution dans l'espace des observations *augmente exponentiellement avec la dimension*. Il en va de même vis-à-vis du nombre d'exemples d'apprentissage pour les approches qui n'utilisent aucun *a priori* probabiliste, comme les k-ppv. Solutions possibles :
  - Utiliser un modèle probabiliste ! (cf Section 3).
  - Réduire la dimension ! (cf Section 2).

Survol des SVM

- Les *Séparateurs à Vaste Marge* (Support Vector Machines) sont des approches *discriminatives* qui construisent dans l'espace d'observation des *surfaces de séparation optimales entre deux classes* sur des vecteurs d'apprentissage etiquetés.
- La surface optimale est celle qui maximise la *marge* de séparation, c'est à dire la distance aux vecteurs les plus proches de la surface (= les vecteurs *supports*) :



Faible marge



Grande marge

Survol des SVM

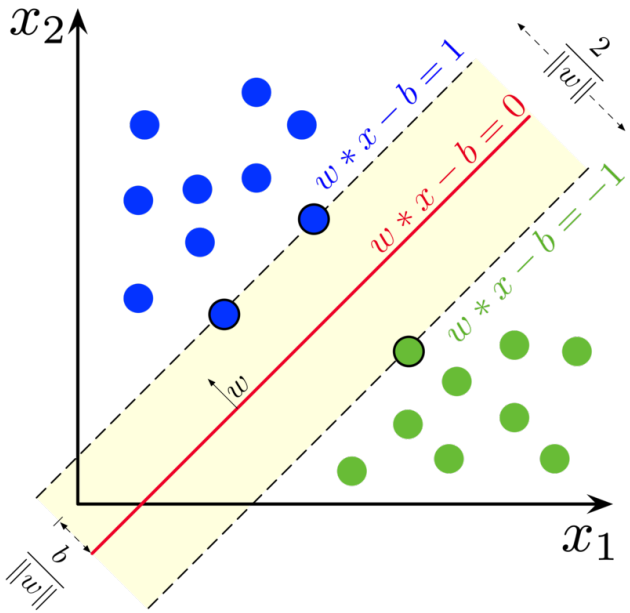


Image : *Wikipedia*

Dans le cas linéaire la surface de séparation est un hyperplan d'équation :

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

- **w** est un vecteur normal à l'hyperplan.
- b est proportionnel à $d(\mathcal{P}, O)$, distance de l'hyperplan à l'origine.

Survol des SVM

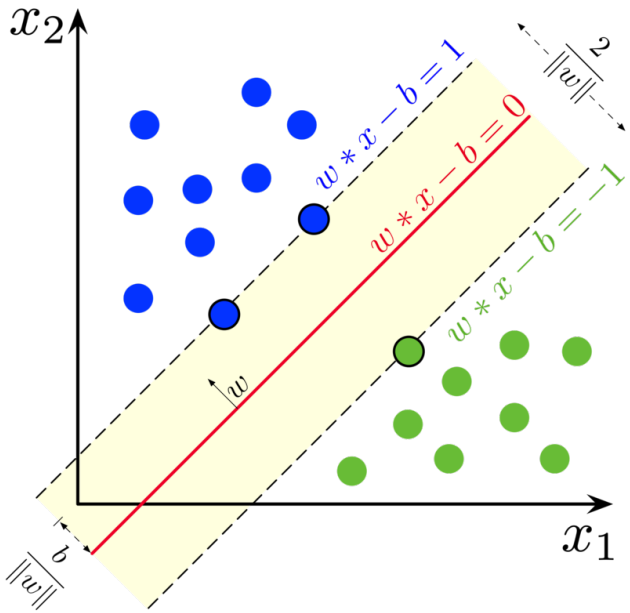


Image : *Wikipedia*

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

$\mathbf{w}$ et b sont *normalisés* par la marge m (distance de $\mathcal{P}$ aux vecteurs supports) :

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

Classification par SVM

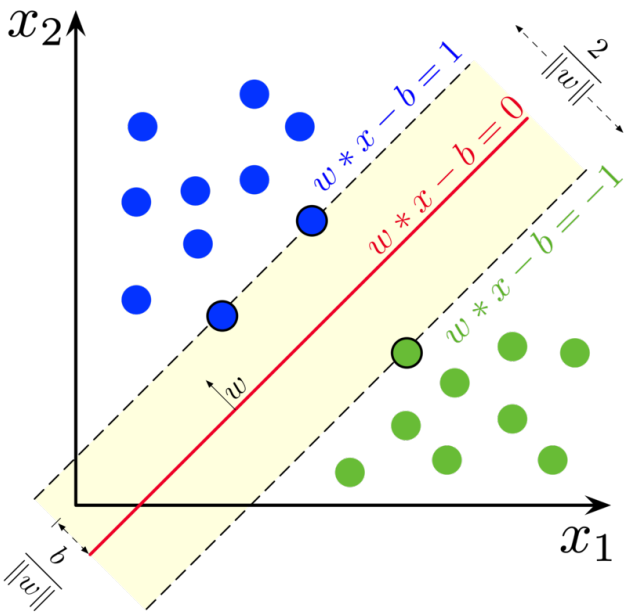


Image : *Wikipedia*

- La classification d'un vecteur inconnu X se fait simplement en comparant $X.\mathbf{w}$ à b :

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

- L'indice de *confiance* de la classification est donné par la valeur absolue (= distance de X à l'hyperplan en "nombre de marges") :

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

Apprentissage d'un SVM

L'entraînement d'un SVM linéaire sur des données d'apprentissage étiquetées $\{X_i, c(X_i)\}$, avec $c(X_i) = +1$ ou $c(X_i) = -1$ est un problème d'optimisation quadratique sous contraintes linéaires :

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

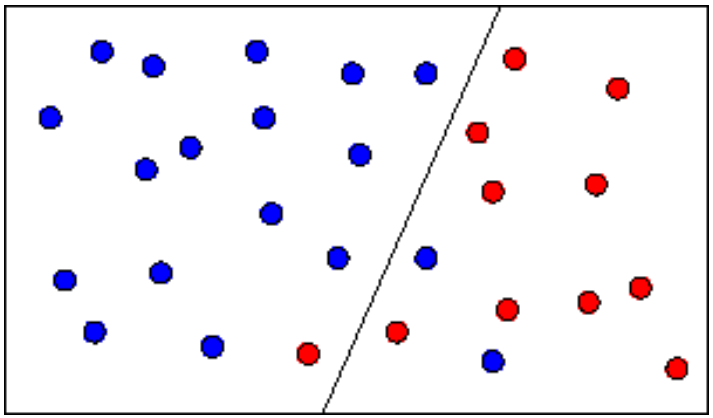
Les vecteurs supports sont les vecteurs X_s tels que :

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

Soft Margin SVM

Le SVM linéaire peut être étendu au cas où les deux classes ne sont pas linéairement séparables, en relativisant la marge (soft margin). L'optimisation suivante peut être utilisée :

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



SVM non linéaire

L'idée des SVM non linéaires est de transformer l'espace d'observations (vecteurs X) en un espace de plus grande dimension (vecteurs $\Phi(X)$) où les vecteurs seraient linéairement séparables.

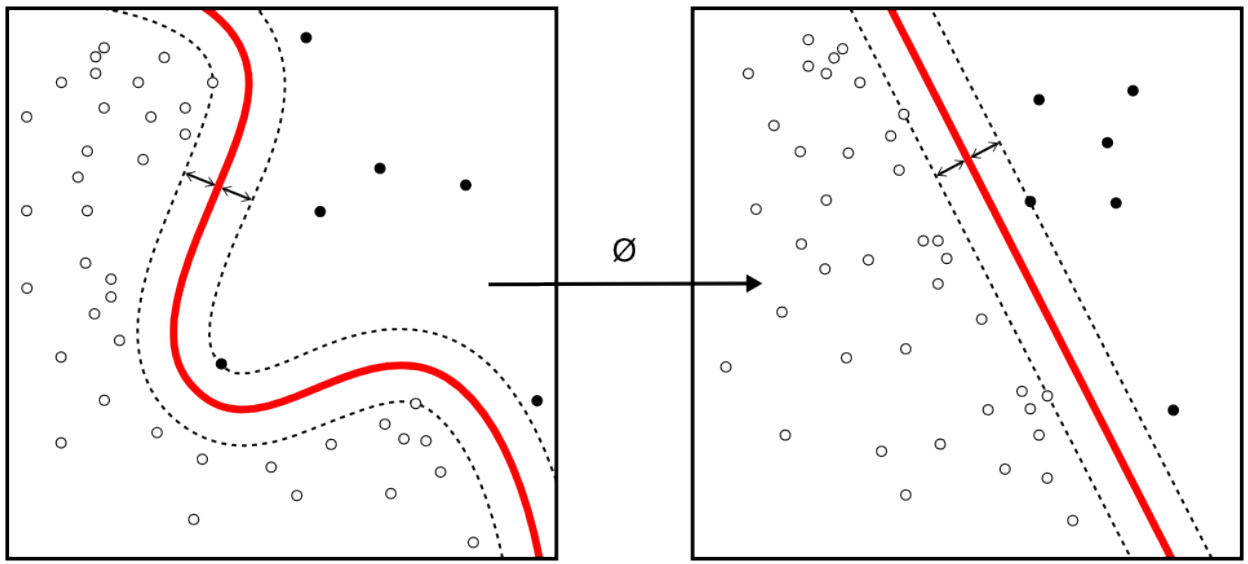


Image : *Wikipedia*

SVM non linéaire

L'astuce (*Kernel trick*) consiste à ne pas calculer explicitement Φ, mais à utiliser un fonction à valeur réelle (noyau) non linéaire pour remplacer le produit scalaire : $K(X, Y) = \Phi(X).\Phi(Y)$.

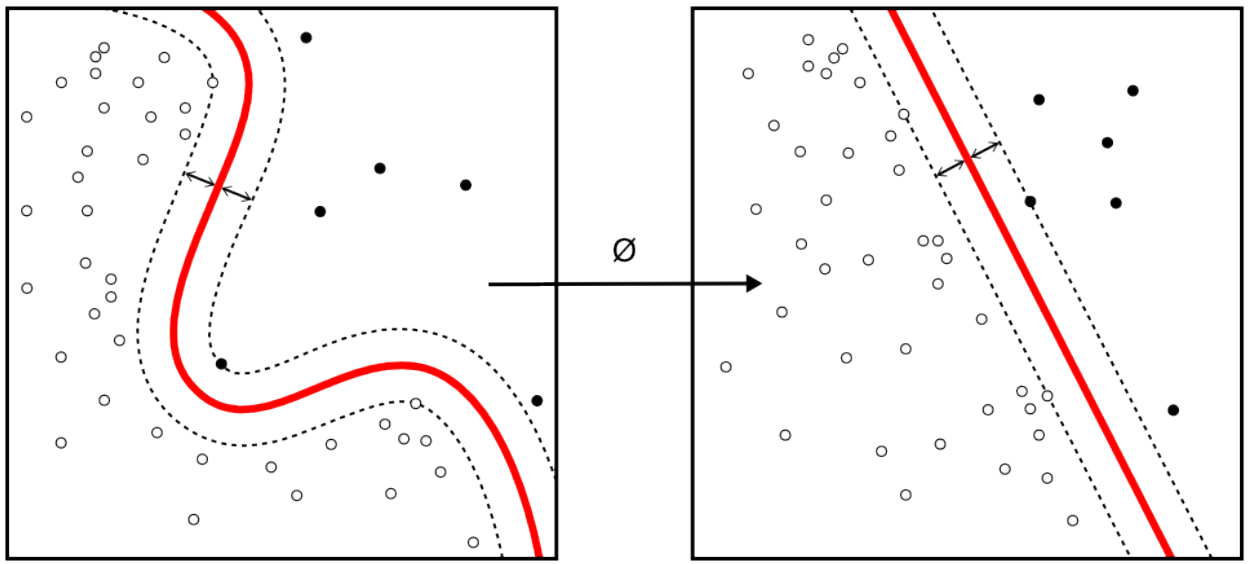


Image : *Wikipedia*

Conclusion sur les SVM

- Les SVM sont des modèles extrêmement compacts : un hyperplan séparateur $(\mathbf{w}, b)$ par SVM.
- La classification est extrêmement rapide : un seul produit scalaire pour classer un vecteur selon les deux classes.
- Les SVM sont seulement des classifieurs binaires. Pour les étendre à une classification en N classes, deux approches possibles :
  - *1-against-all*: N classifications binaires.
  - *1-against-1*: $\frac{N(N-1)}{2}$ classifications binaires.

Forêts Aléatoires

Le principe des forêts aléatoires (*Random Forests*, RF) repose sur la classification par arbres de décision :

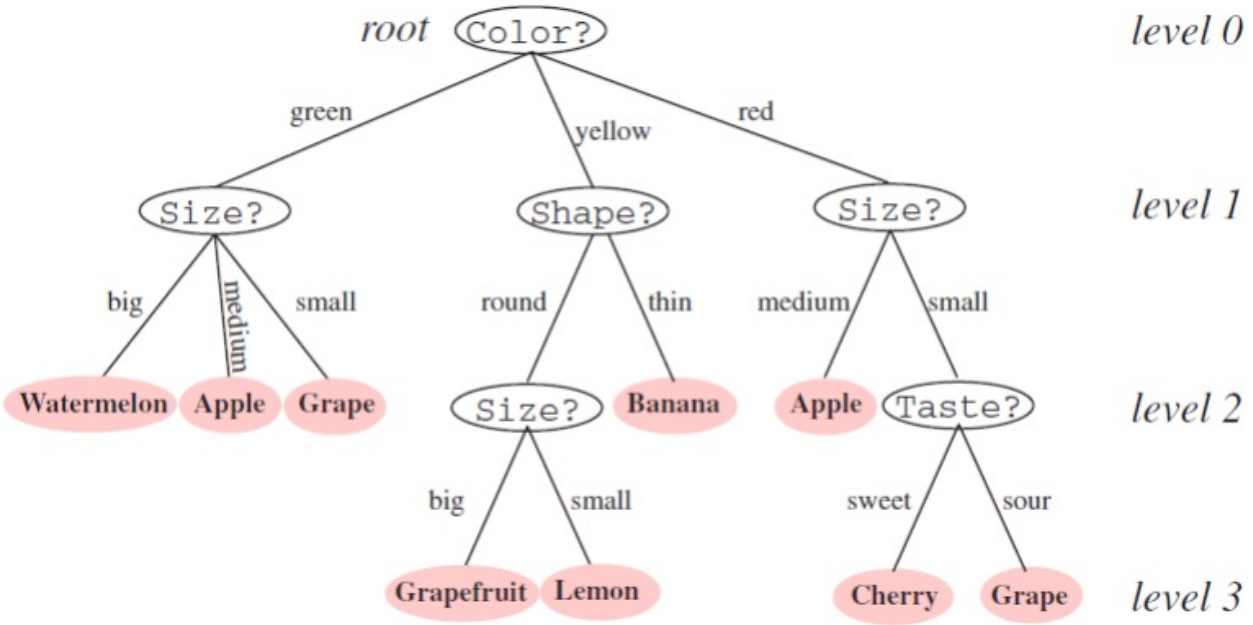


Figure d'*Anil K. Jain*

Entraînement d'un arbre aléatoire

L'apprentissage (création d'un arbre) consiste à partitionner récursivement l'ensemble d'apprentissage $\{X_i, c(X_i)\}$ en deux sous-ensembles, jusqu'à un critère d'arrêt, lié typiquement à :

- la *cardinalité* de la feuille (taille minimale du sous-ensemble, en deça de laquelle la feuille n'est plus représentative).
- la *qualité* de la feuille (ex : faible entropie de classes du sous-ensemble, signifiant que la feuille a correctement "isolé" une ou plusieurs classes).

Entraînement d'un arbre aléatoire

La création de chaque nœud de l'arbre consiste à :

- tirer au hasard un grand nombre de *prédicats binaires* liés à une des composantes des vecteurs $\{X_i\}$ (par exemple "$X_i^k < t$").
- évaluer la qualité des feuilles (provisoires) résultant de la partition correspondante (par exemple en calculant leur entropie de classes).
- conserver le prédicat qui réalise le meilleur score de qualité.

NB : Noter la parenté avec les Kd-trees.

Classification par RF

- La classification par un arbre aléatoire consiste à soumettre un vecteur inconnu X aux prédicats rencontrés dans son parcours depuis la racine (cf Kd-trees et recherche de ppv), jusqu'à parvenir à une feuille.
- L'histogramme des classes représentées dans la feuille d'arrivée est interprété comme un vecteur de probabilité des classes du vecteur $P(c/X)$.
- Le processus est répété sur plusieurs arbres et les décisions sont combinées (vote majoritaire, ou moyennes des histogrammes...)

Classification par RF

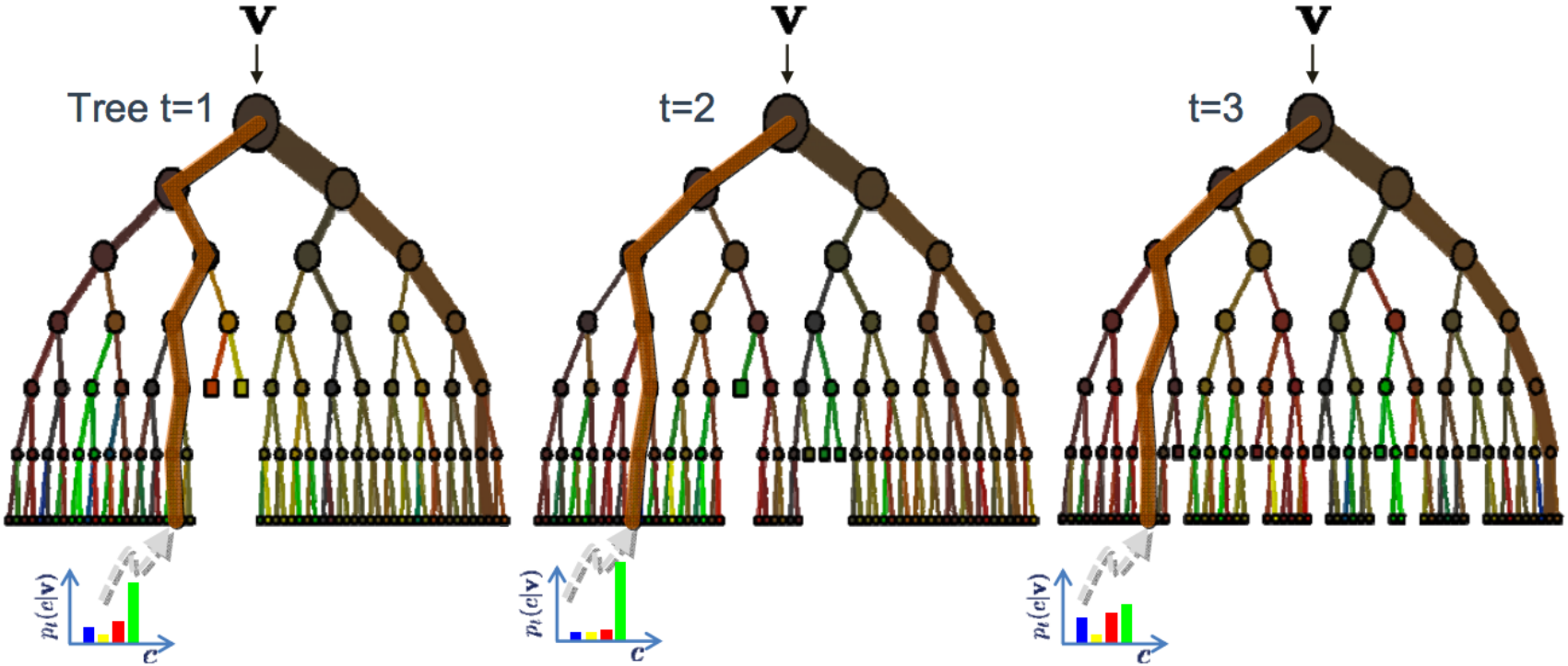


Figure de *Raghav Aggiwal*

Conclusion sur les Forêts Aléatoires

+ Possibilité d'utiliser des composantes de nature très différentes (continue ou discrète, avec ou sans métrique).
+ Grande lisibilité (*"explicabilité"*) du modèle.
+ Rapidité de la classification.
- Lourdeur du modèle en taille mémoire.
- Chute des performances avec l'augmentation du nombre de données d'apprentissage, ainsi que de leur dimension (= nombre de caractéristiques).

Plan du cours

Réseaux de neurones artificiels

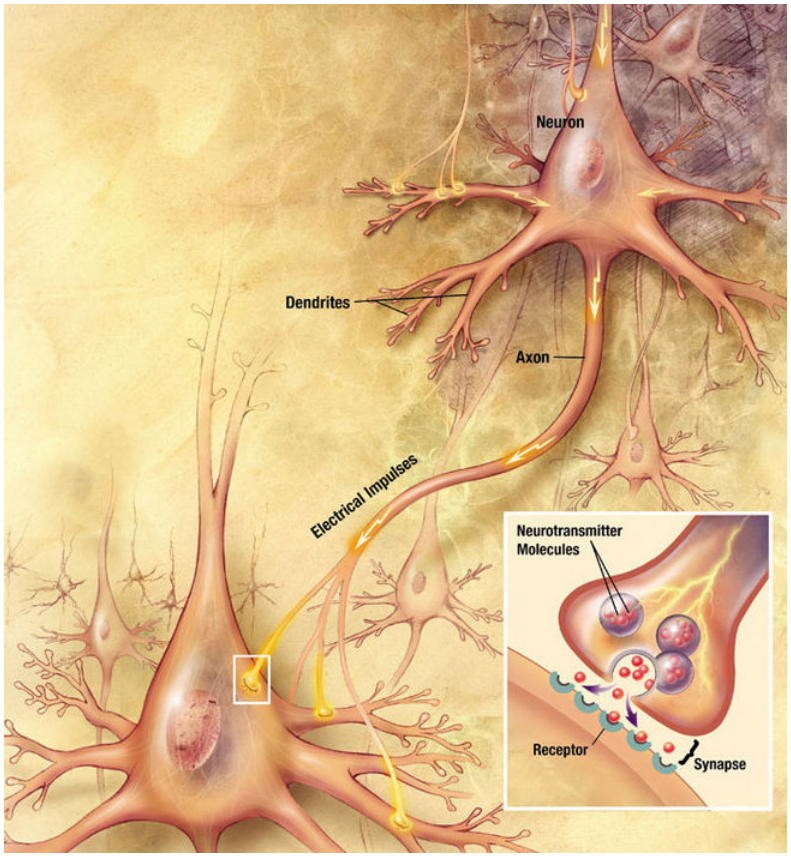
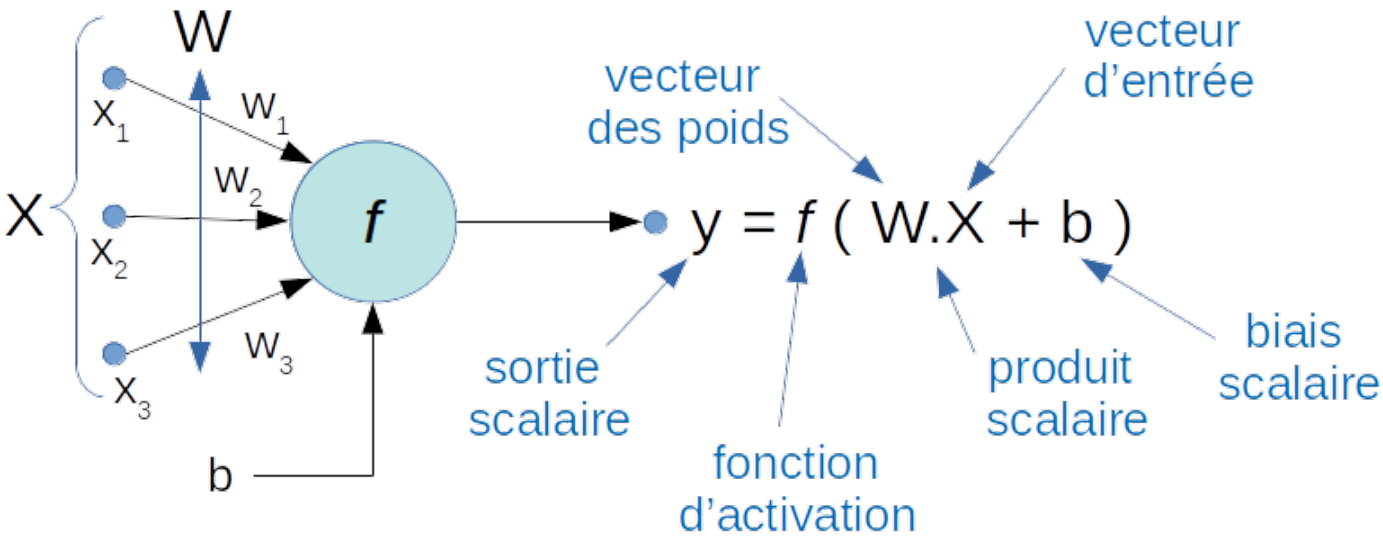


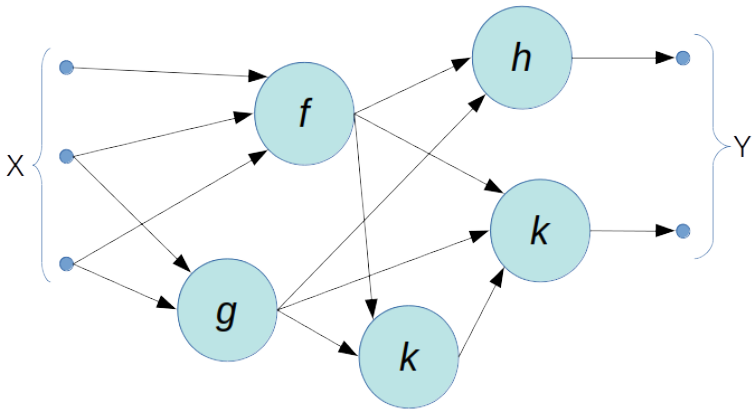
Image : *Wikipedia*

- Années 1940 : Modèle de Neurone Formel - Règle de Hebb.
- Années 1950-60 : Travaux sur le Perceptron.
- Années 1980 : Réseaux de Hopfield, Machines de Boltzmann.
- Années 1980 : Perceptrons Multicouches, Rétropropagation du Gradient.
- Années 2010 : Retour en force *via* les réseaux profonds.

Modèle de Neurone Formel



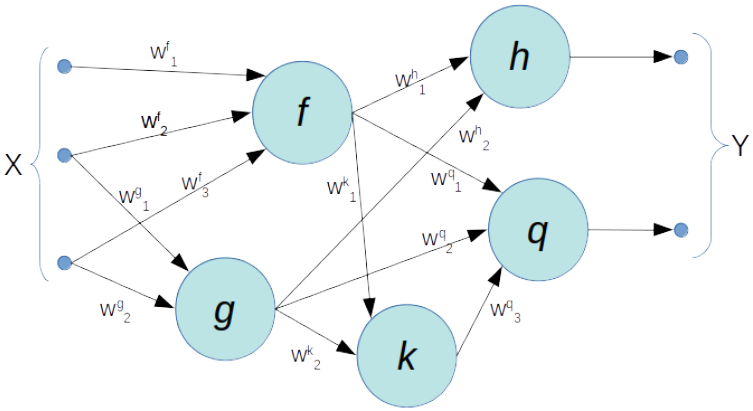
Réseaux de neurones



Un réseau de neurones formels est un graphe orienté, dont :

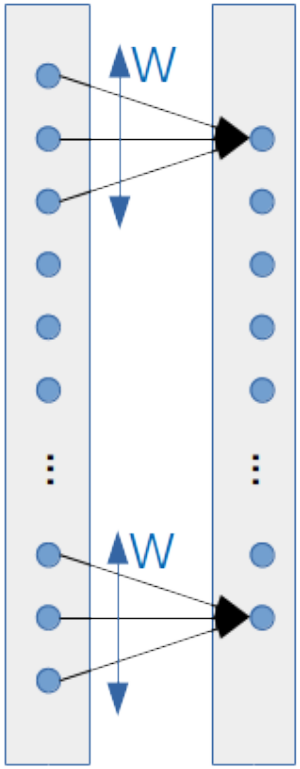
- Les sources forment le vecteur d'entrée X, qui représente la donnée, voire qui est la donnée elle-même (apprentissage de bout en bout).
- Les puits forment le vecteur de sortie Y, qui est interprété comme le résultat de la classification ou de la régression.

Réseaux de neurones



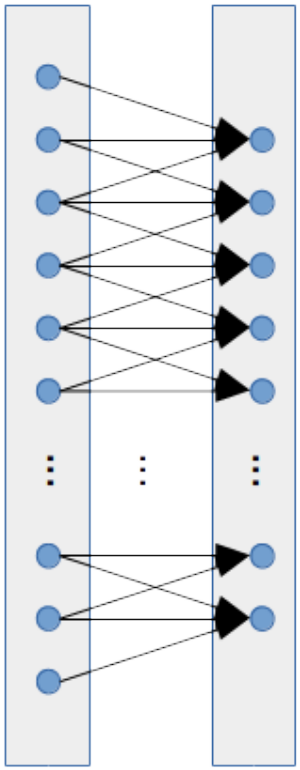
- L'architecture (i.e. le graphe), et les fonctions d'activation sont en général définies *a priori* et *statiques*.
- Les poids des connexions W (et les valeurs de biais b) sont *adaptatifs* et sont modelés par l'apprentissage.

Réseaux de neurones convolutifs



Dans un réseau de neurones convolutif (CNN), un même neurone (i.e. mêmes vecteur de poids et fonction d'activation) est utilisé pour toutes les parties du vecteur d'entrée associé à chaque couche.

Réseaux de neurones convolutifs

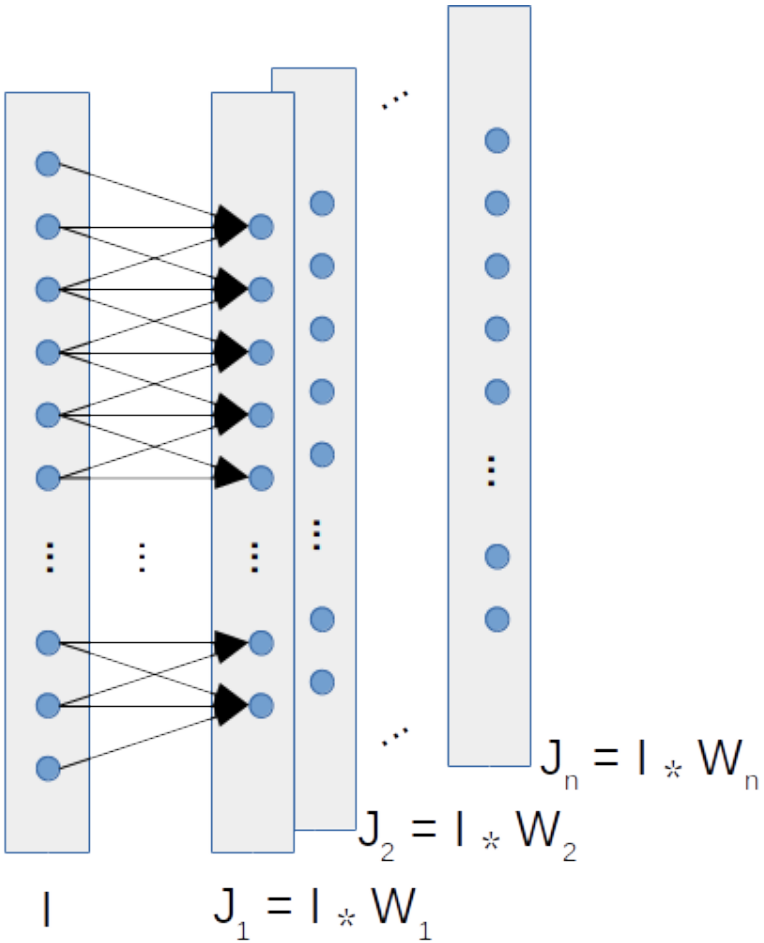


I

$J = I * W$

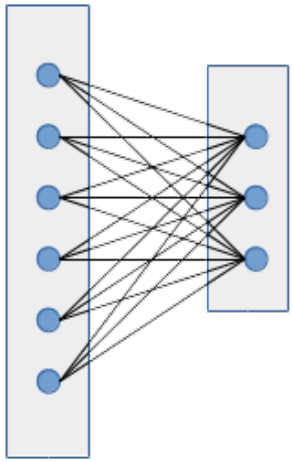
L'opération réalisée entre deux couches I et J est donc une application linéaire invariante par translation, c'est-à-dire une convolution...

Réseaux de neurones convolutifs



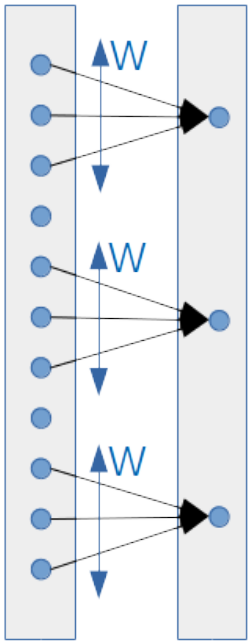
En réalité, ce sont en général plusieurs neurones qui sont ainsi appliquées à chaque couche, ce qui correspond à un banc de filtres de convolution...

Couches entièrement connectées

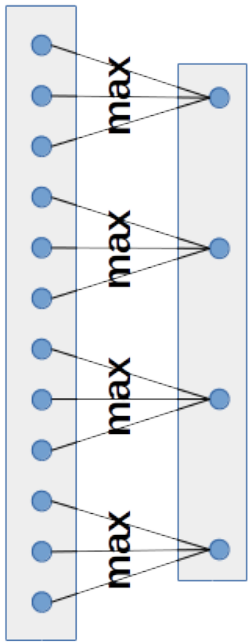


Un cas particulier important : lorsque la taille des vecteurs de poids coïncide avec la taille du vecteur d'entrée, on a affaire à une couche entièrement connectée.

CNN : Réduction de la dimension



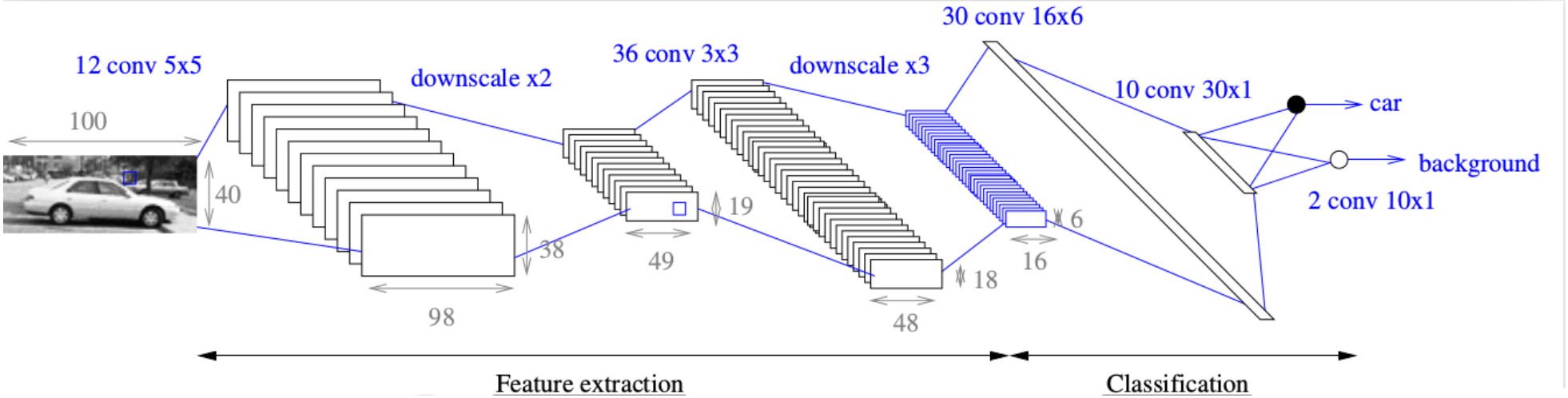
(1)



(2)

1. **stride :** Augmentation du pas d'échantillonnage dans l'application de la convolution.
2. **pooling :** Sous-échantillonnage par combinaison (moyenne, maximum...) locale des valeurs.

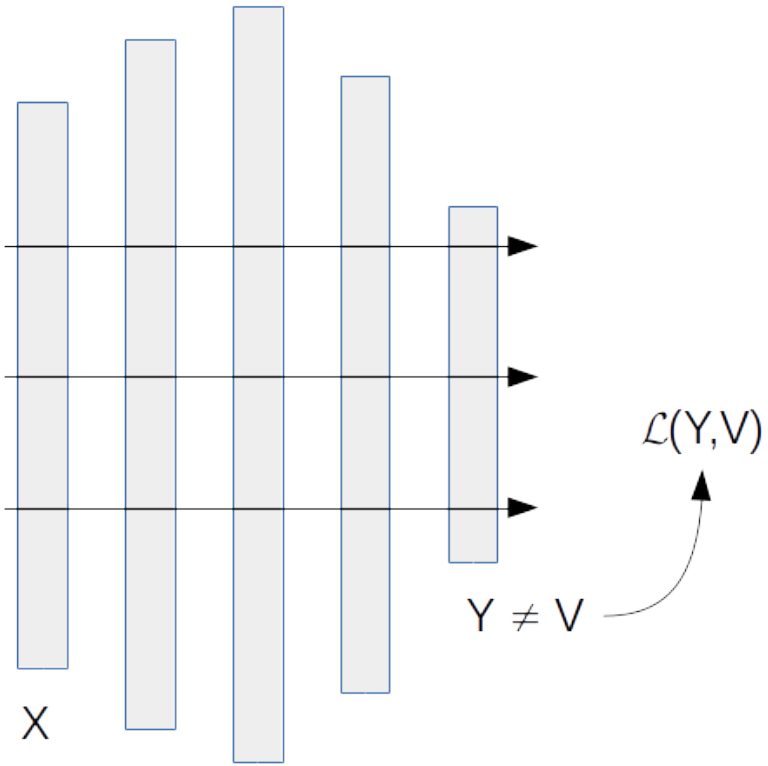
Exemple de réseau convolutionnel



Entraînement supervisé d'un CNN

- Un CNN est entraîné à partir d'un ensemble d'apprentissage $\{X_i, V_i\}$ où les X_i sont les données d'entraînement et les V_i les sorties attendues (supervision).
- On considère une fonction de coût (loss) $\mathcal{L}(Y, V) \geq 0$, qui mesure la différence (erreur) entre une sortie calculée Y et une sortie attendue V.
- L'objectif de l'entraînement est de minimiser l'erreur globale sur l'ensemble d'apprentissage : $\sum_i \mathcal{L}(\mathcal{O}(X_i), V_i)$, où $\mathcal{O}(X)$ est la sortie calculée par le réseau sur l'entrée X.

Entraînement d'un CNN (forward)



Dans la passe avant, la donnée X est soumise au réseau et on compare la sortie produite Y à la sortie attendue V en utilisant la fonction d'erreur $\mathcal{L}(Y, V)$.

Propagation Avant

Pour simplifier l'écriture on suppose que toutes les fonctions d'activation sont égales à g. La passe avant s'écrit :

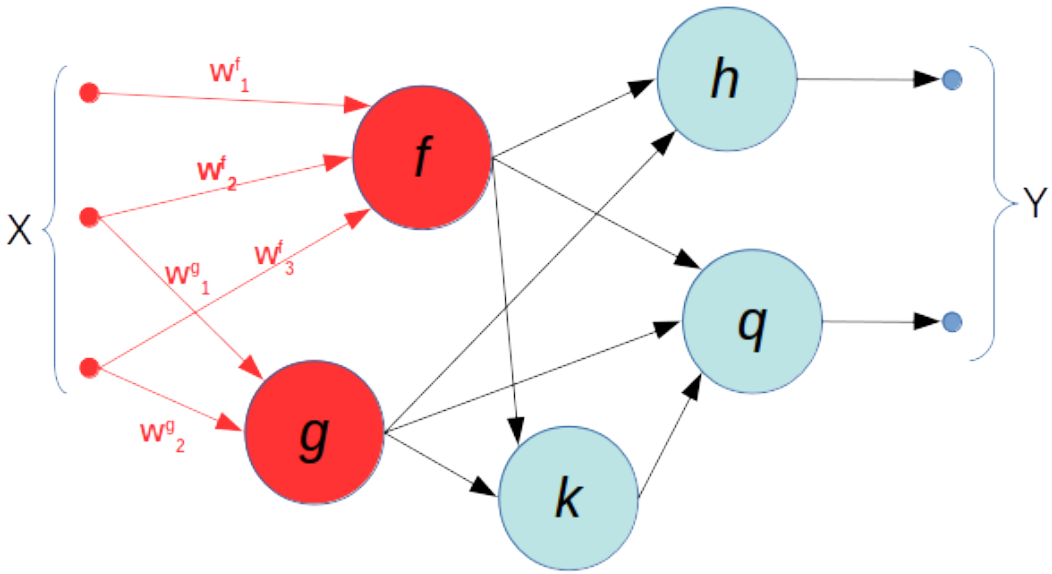
Propagation avant $Y \leftarrow \text{ForwardProp}(W, X)$

- Pour tous les neurones i de la couche d'entrée $s_i = x_i$.
- Pour toutes les couches l suivantes :
  - Pour tous les neurones j de la couche l :

$$s_j = g\left(\sum_k w_{kj} s_k + b_j\right).$$

- Pour tous les neurones j de la couche de sortie $y_j = s_j$.

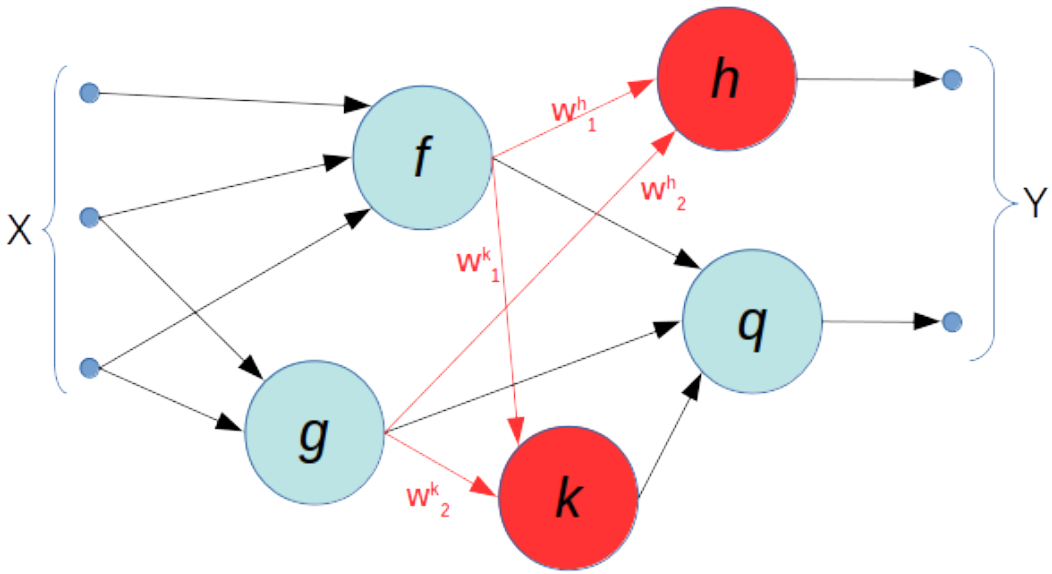
Exemple de passe avant



Calculer, *en parallèle* :

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

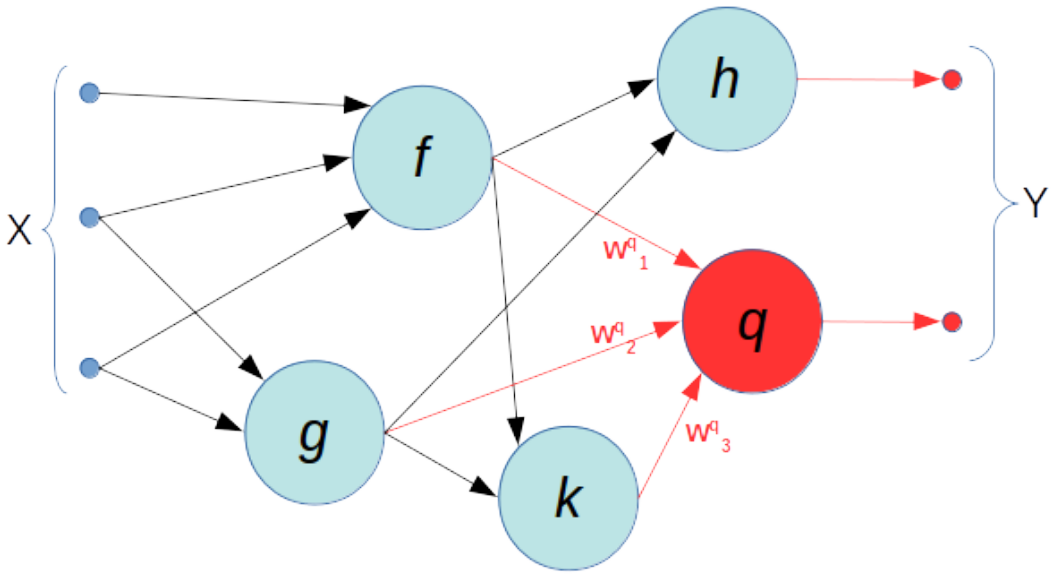
Exemple de passe avant



Calculer, *en parallèle* :

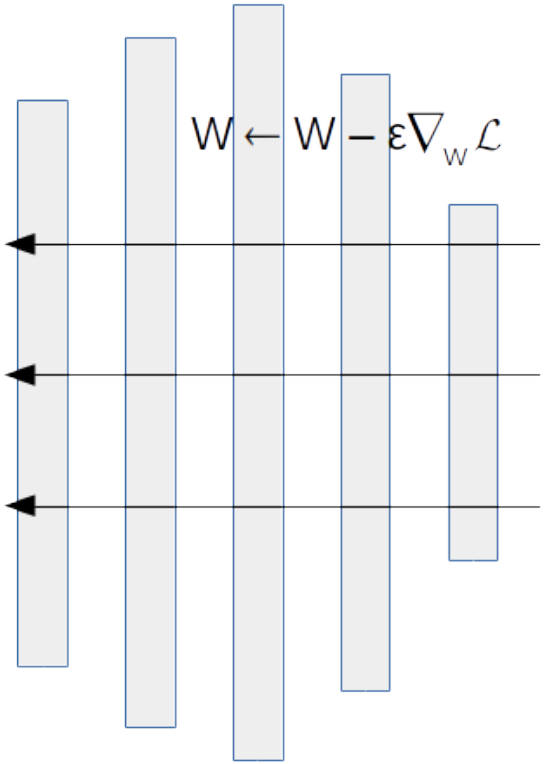
- $s_h = h\left(w_1^h s_f + w_2^h s_g\right)$
- $s_k = k\left(w_1^k s_f + w_2^k s_g\right)$

Exemple de passe avant



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

Entraînement d'un CNN (backward)



Dans la passe arrière, l'erreur commise $\mathcal{L}(Y, V)$ est rétro-propagée à l'ensemble des neurones, et les poids des connexions sont ajustés en fonction de leur contribution à l'erreur commise :

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

où ε est le taux d'apprentissage (learning rate).

Calcul effectif des contributions

Mais la contribution à l'erreur des poids de connexion $\frac{\partial \mathcal{L}}{\partial w_{ij}}$ n'est calculable directement que pour les connexions de la couche de sortie. On va donc utiliser les règles de dérivation en chaîne pour calculer récursivement les contributions des poids des précédentes couches. Commençons par quelques notations / conventions :

- On note w_{ij} le poids de la connexion du neurone i vers le neurone j.
- On note s_i la valeur de sortie du neurone i.
- On suppose toutes les fonctions d'activation égales à g.
- On note $a_j = \sum_i w_{ij} s_i + b_j$ la valeur d'activation du neurone j.

  Sa sortie est donc $s_j = g(a_j)$.

Calcul récursif des contributions

L'équation de mise à jour s'écrit :

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

En introduisant a_j, la valeur d'activation du neurone j, on a :

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

Puisque $a_j = \sum_i w_{ij} s_i + b_j$. D'autre part on a :

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

Puisque $s_j = g(a_j)$.

Calcul récursif des contributions

Enfin, en développant l'impact de la sortie du neurone j sur toutes les entrées k de la couche qui suit celle de j :

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

Puisque $a_k = \sum_j w_{jk} s_j + b_k$. En conclusion, **il suffit** de savoir calculer le gradient de l'erreur **sur la dernière couche** pour le calculer **sur toutes les couches précédentes par récursivité**.

Calcul sur la dernière couche

Or pour la dernière couche le calcul est immédiat.
Supposons pour simplifier une fonction d'erreur quadratique :

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

Pour la dernière couche on a $s_j = y_j$ et donc : $\mathcal{L} = \frac{1}{2}\sum_j (s_j - v_j)^2$,

on a donc :

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

Mise à jour des biais

Il faut également mettre à jour les biais :

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

Il suffit alors de remarquer que :

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

Donc sur la dernière couche :

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

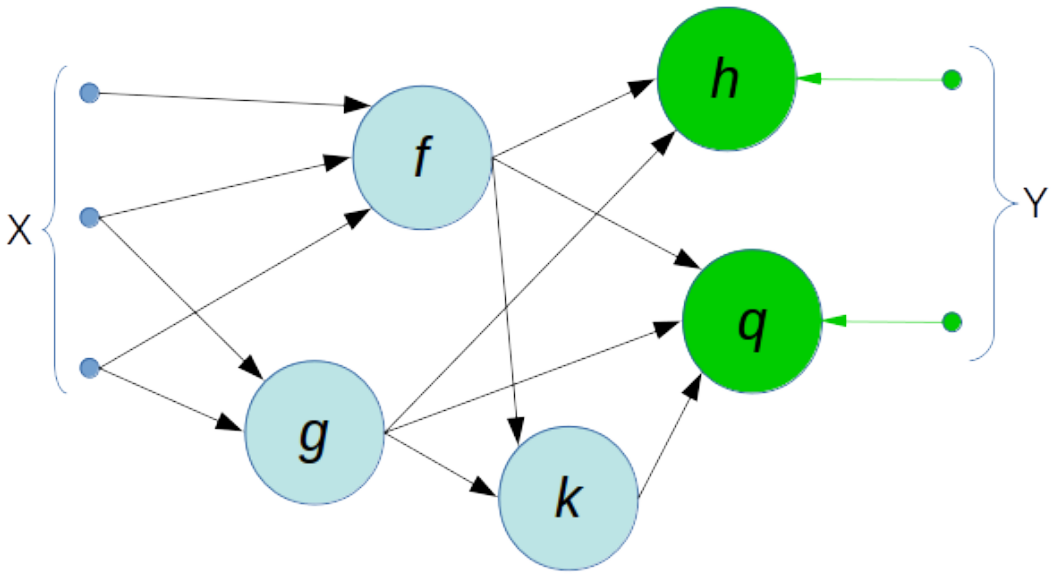
Algorithme de Rétropropagation du Gradient

Finalement, l'algorithme de la passe arrière s'écrit (Pour une fonction d'erreur $\mathcal{L}$ quadratique) :

Rétropropagation du gradient $W \leftarrow \text{BackProp}(W, Y, V)$

- Pour tous les neurones j de la couche de sortie :
  - calculer l'erreur $\Delta_j = (s_j - v_j) \times g'(a_j)$
- Pour toutes les couches l précédentes :
  - Pour tous les neurones i de la couche l :
    - calculer l'erreur $\Delta_i = \left(\sum_k \Delta_k w_{ik} \right) \times g'(a_i)$
- Pour toutes les connexions (i, j) du réseau :
  - mettre à jour le poids $w_{ij} \leftarrow w_{ij} - \varepsilon \times s_i \times \Delta_j$
  - mettre à jour le biais $b_j \leftarrow b_j - \varepsilon \times \Delta_j$

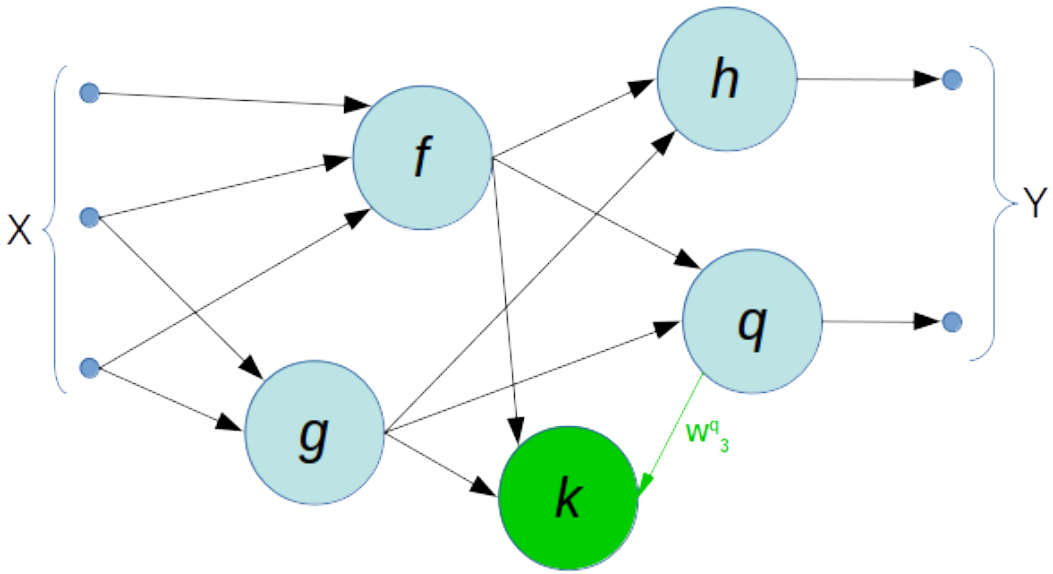
Exemple de passe arrière



Calculer l'erreur sur la couche de sortie, *en parallèle* :

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

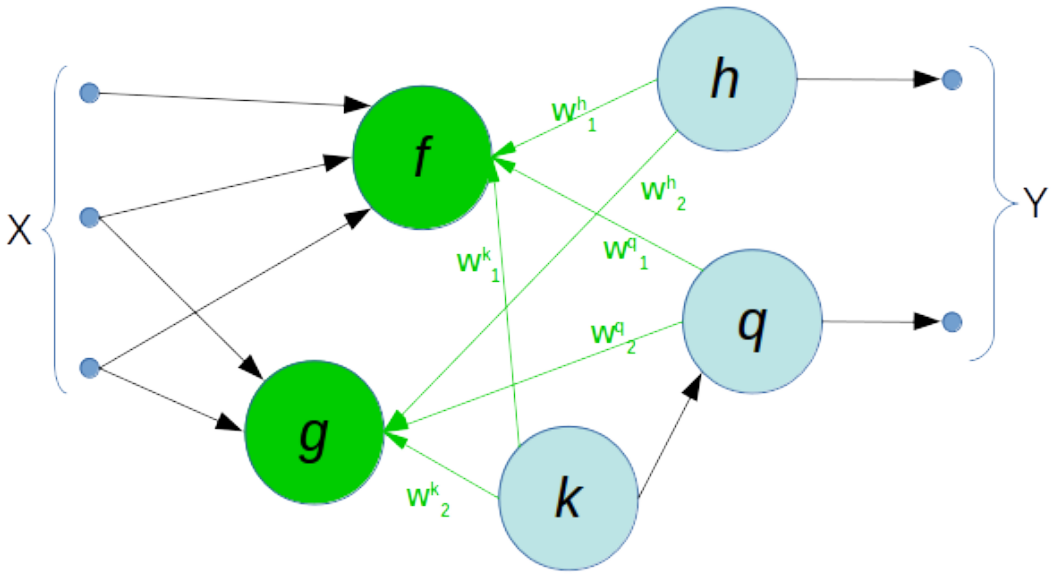
Exemple de passe arrière



Couche intermédiaire :

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

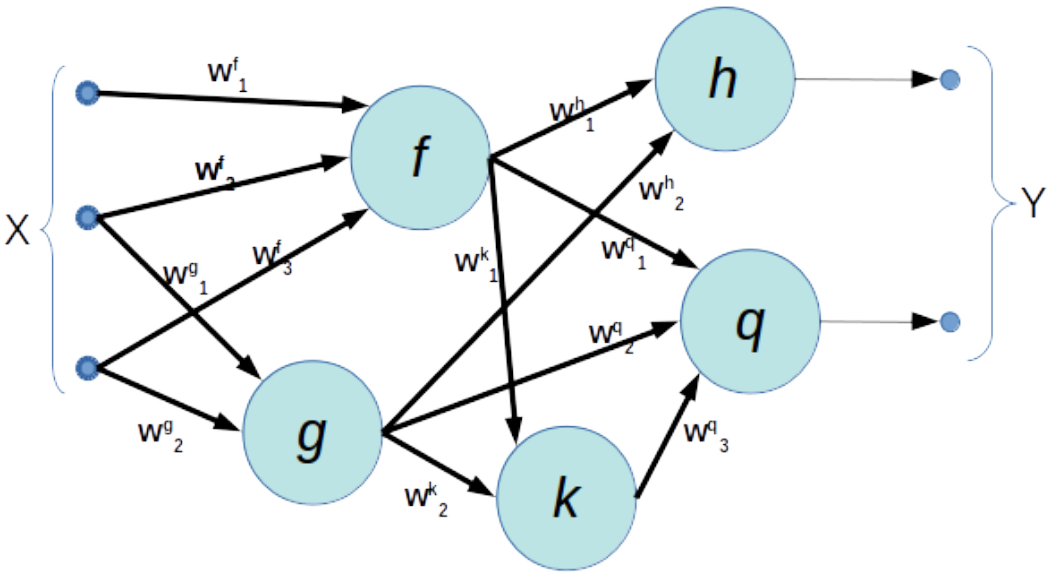
Exemple de passe arrière



Couche d'entrée, *en parallèle* :

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

Exemple de passe arrière



Finalement mettre à jour tous les poids w_{ij}, *en parallèle* :

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

Algorithme d'apprentissage : Gradient stochastique

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

- Initialisation : $W \leftarrow \text{RANDOM}(]-\eta, +\eta[)$
- Répéter jusqu'à convergence $(\star)$:
  - Pour chaque exemple d'apprentissage k :
    - $Y_k \leftarrow \text{ForwardProp}(W, X_k)$
    - $W \leftarrow \text{BackProp}(W, Y_k, V_k)$

- Apprentissage plus lent (1 BackProp par exemple)
- Convergence plus rapide (Moins d'itérations "epochs" $\star$ sur la base)
- Progression irrégulière

Algorithme d'apprentissage : Gradient par lots (batches)

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

- Initialisation : $W \leftarrow \text{RANDOM}(]-\eta, +\eta[)$
- Diviser la base d'apprentissage $\mathcal{A} = \{X_i, V_i\}$ en lots $\mathcal{A}_b$
- Répéter jusqu'à convergence $(\star)$:
  - Pour chaque lot b :
    - $\mathcal{L}_b \leftarrow \sum_{(X_k, V_k) \in \mathcal{A}_b} \mathcal{L}(\text{ForwardProp}(W, X_k), V_k)$
    - $W \leftarrow \text{BackProp}(W, \mathcal{L}_b)$

- Apprentissage plus rapide (1 BackProp par lot)
- Convergence plus lente (Plus d'"epochs" $\star$)
- Progression plus régulière

Quelques fonctions d'erreur classiques

Erreur L_2

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

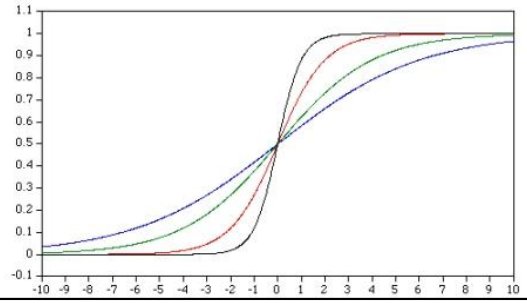
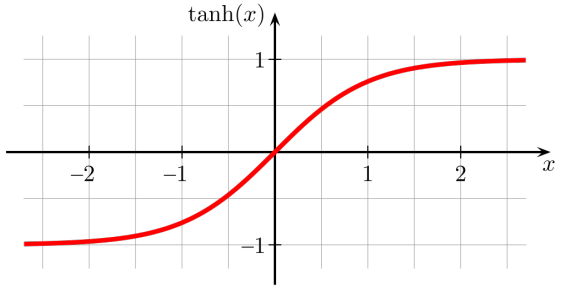
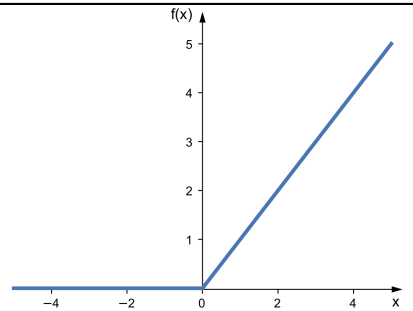
Erreur Logarithmique Moyenne ($V_i, Y_i \geq 0$)

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

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

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

Quelques fonctions d'activation usuelles

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

Quelques variantes de schémas d'optimisation

Quelques variantes au schéma d'optimisation de base (SGD)
$w_{ij} \leftarrow w_{ij} - \varepsilon \frac{\partial \mathcal{L}}{\partial w_{ij}}$:

SGD avec Momentum

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

- Le Momentum limite les fluctuations de directions du gradient entre 2 itérations (batches) $t-1$ et t.
- $\beta \to O$: SGD classique.
- $\beta \to 1$: Le gradient conserve sa direction.

Quelques variantes de schémas d'optimisation

Quelques variantes au schéma d'optimisation de base (SGD)
$w_{ij} \leftarrow w_{ij} - \varepsilon \frac{\partial \mathcal{L}}{\partial w_{ij}}$:

Adam

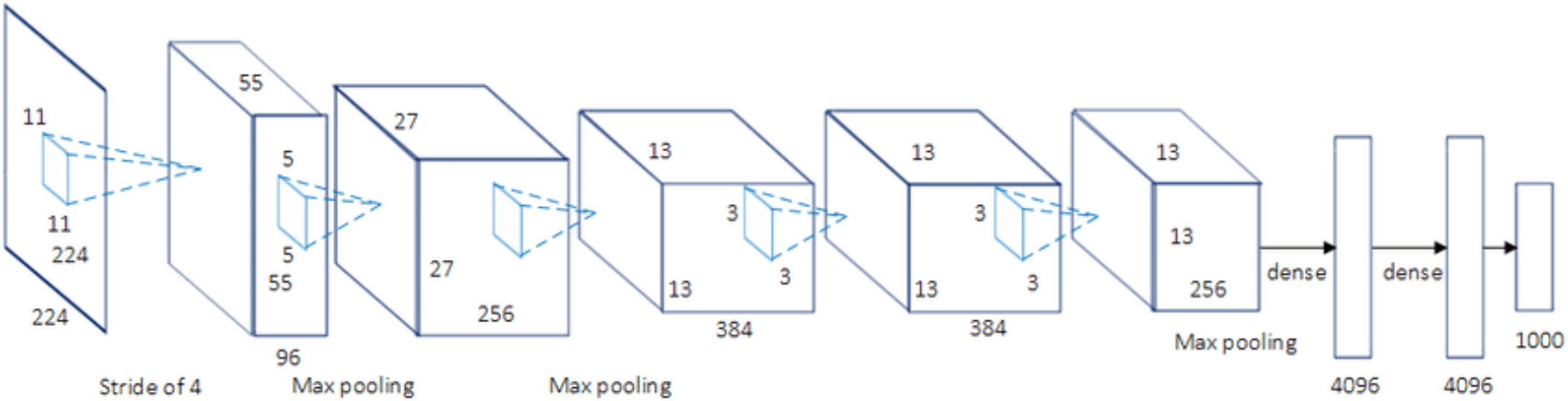
$$G_t = \beta_1 G_{t-1} + (1-\beta_1)\frac{\partial \mathcal{L}}{\partial w_{ij}}; \beta_1 \in [0,1]$$

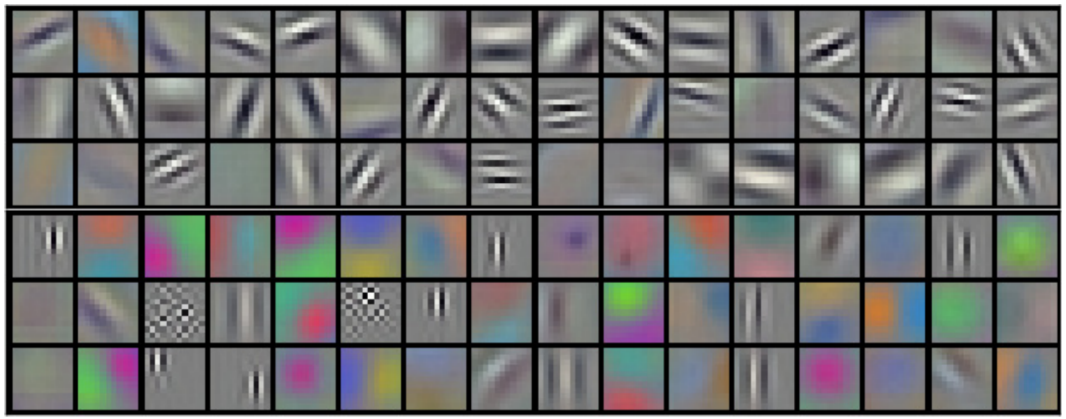
$$K_t = \beta_2 K_{t-1} + (1-\beta_2)\left(\frac{\partial \mathcal{L}}{\partial w_{ij}}\right)^2; \beta_2 \in [0,1]$$

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

- Adam limite les fluctuations temporelles de directions du gradient, tout en adaptant la vitesse à la courbure (dérivée seconde) de la fonction de coût.
- 2 hyperparamètres supplémentaires : $\{\beta_1, \beta_2\}$.

Deep CNN pour la classification d'images



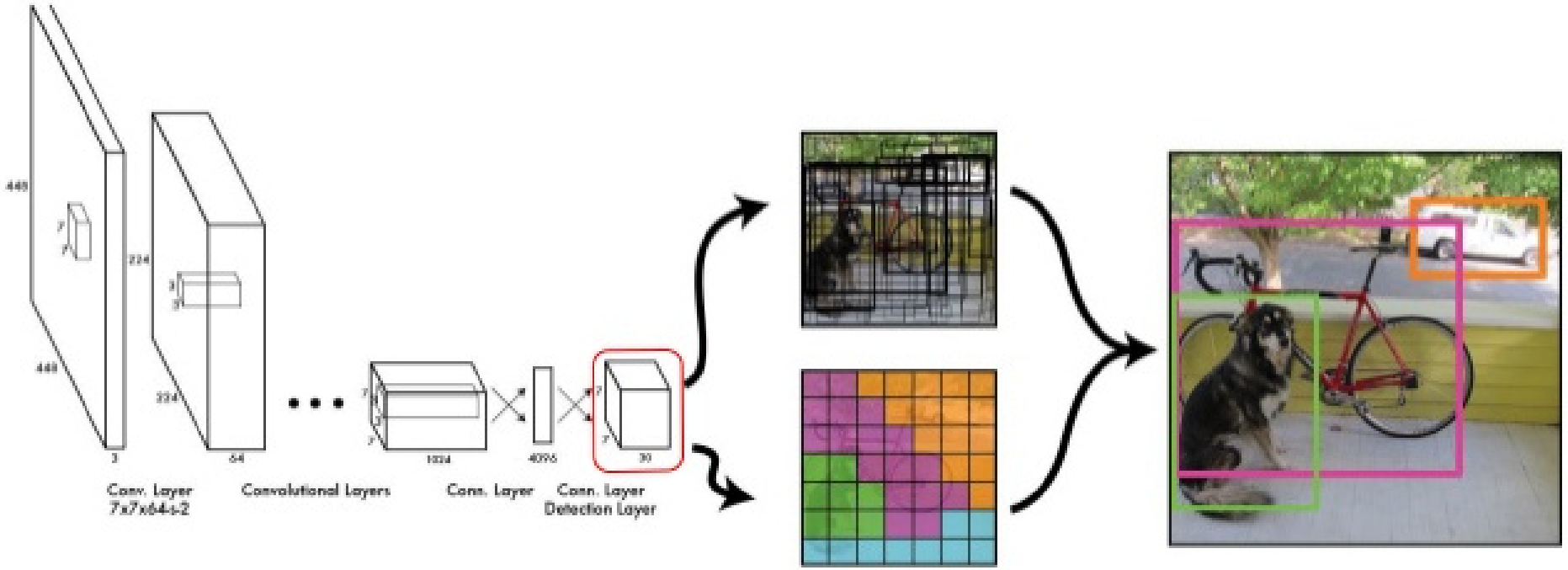


Les 96 premières convolutions 11×11

AlexNet **[Krizhevsky12]**:

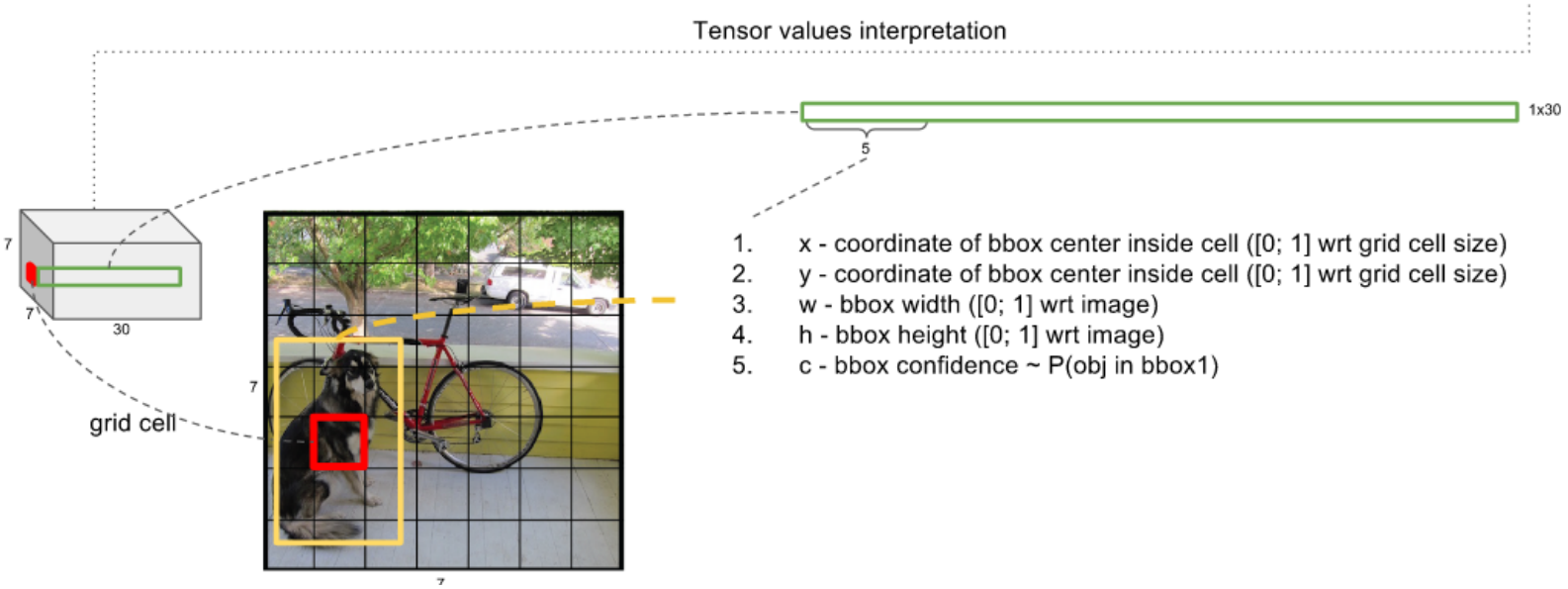
- Entraîné sur ImageNet ILSVRC-2010
- 1,2M Images d'apprentissage
- 1000 Classes

Deep CNN pour la détection d'objets



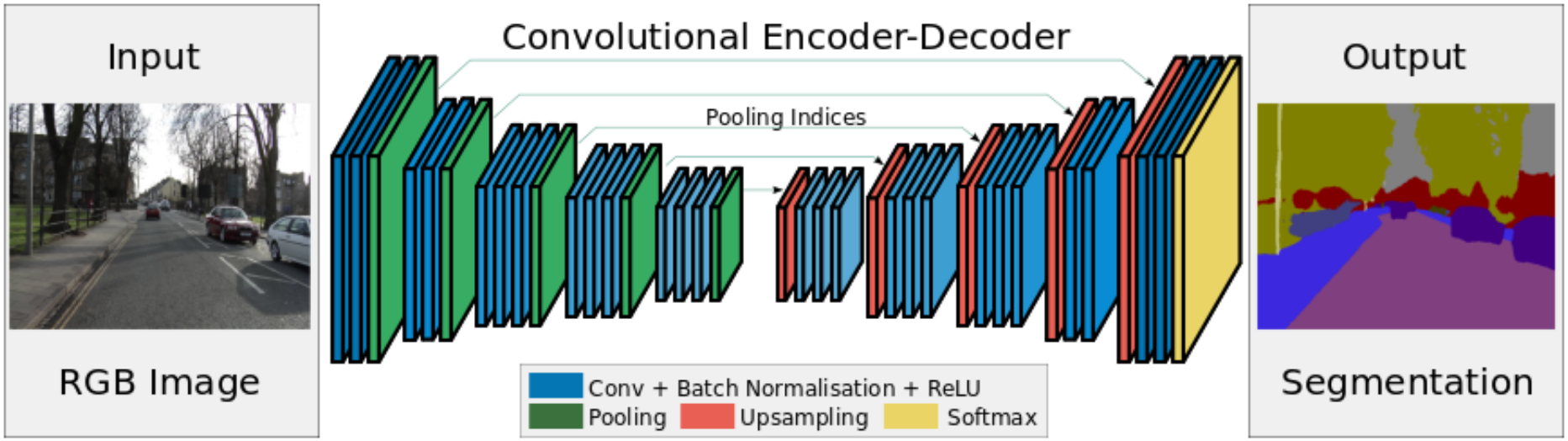
YOLO **[Redmon16]**

Deep CNN pour la détection d'objets



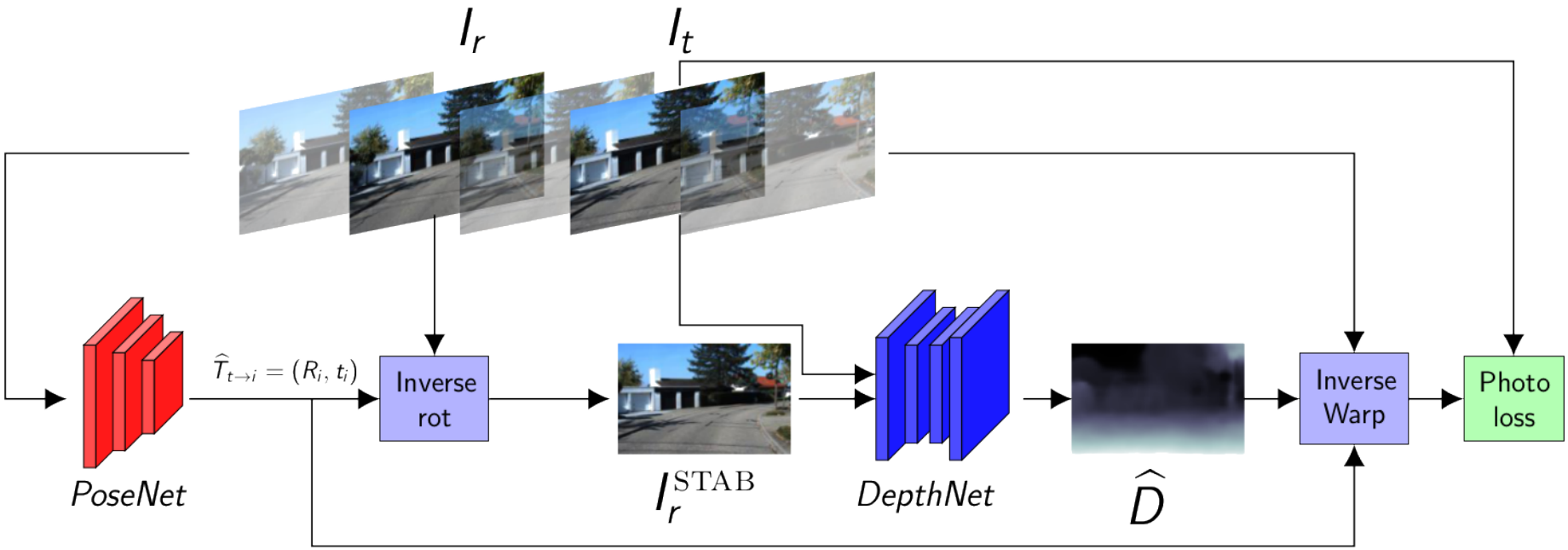
YOLO **[Redmon16]**

Deep CNN pour la segmentation sémantique



SegNet **[Badrinarayanan15]**

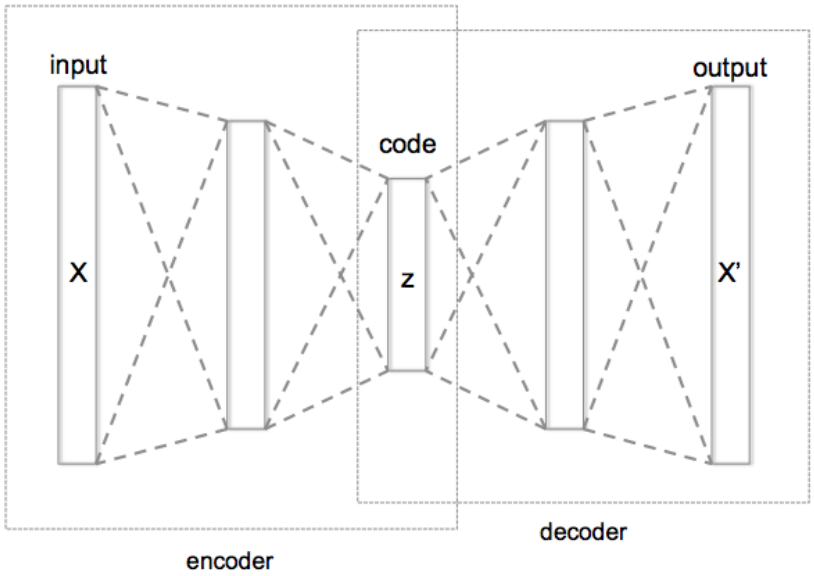
Deep CNN pour la reconstruction 3d



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

PoseNet + DepthNet **[Pinard18]**

Deep CNN auto-supervisé



Auto-encodeur (Image : Wikipedia)

En utilisant une fonction d'erreur $\mathcal{L}(\hat{X}, X)$, l'auto-encodeur apprend à reconstruire une copie $\hat{X}$, voire une version améliorée (débruitée, restaurée...) de son entrée X.

Conclusion sur les CNN

Free lunch? Not yet!

- Empirisme / Artisanat !!
- Bases d'images annotées !!
- Explicabilité...?
- Choix de $\mathcal{L}$...?
- Adaptation de domaine...?
- Oubli catastrophique...?

Plan du cours

Conclusion générale

A retenir pour ce cours :

- Réduction de la dimensionalité
  - Statistique : ACP
  - Métrique : K-means
- Apprentissage supervisé
  - Apprentissage bayésien
  - K-plus-proches voisins
  - SVM
  - Random Forest
  - CNN / $\mathcal{L}(Y, V)$
- Apprentissage non supervisé
  - K-means (VBoW)
  - CNN / $\mathcal{L}(\hat{X}, X)$

Conclusion générale

A étudier en complément :

- Pratique de l'apprentissage
  - Partition Bases d'Apprentissage / de Validation / de Test
  - Validation croisée
- Évaluation
  - Classification : Taux d'erreur
  - Classification : Matrice de confusion
  - Classification : Courbe Précision / Rappel...
  - Régression : Norme L_n...
  - Détection : Indice de Jaccard...
- Surapprentissage et Régularisation des CNN : Voir TP !!
  - Dropout - Batch Normalisation - Weight Decay

Bibliographie

[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

Bibliographie

[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

Bibliographie

[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

Bibliographie

[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