*From Data Science to Machine Learning*

# De l'analyse de données à l'apprentissage automatique

Jean-Yves Ramel

ramel@univ-tours.fr

Laboratoire d'Informatique Fondamentale
et Appliquée de Tours

LABORATOIRE D'INFORMATIQUE FONDAMENTALE ET APPLIQUÉE DE TOURS

# 2 – Les principaux modèles

# 1er Exemple : prédiction de l'infarctus du myocarde

On sait que la probabilité de développer un infarctus du myocarde (IM) augmente avec l'âge et avec le taux de cholestérol LDL.

- Comment développer un programme qui prédise le risque d'IM à partir
- de ces deux variables ?

- **Approche de type « système expert » :**
  - Modéliser la connaissance d'un médecin par des règles de la forme :
  - SI âge > 60 ET ldl > 10 ALORS risque élevé
  - SI 50 < âge  60 ET 8 < ldl  10 ALORS risque moyen
  - ...

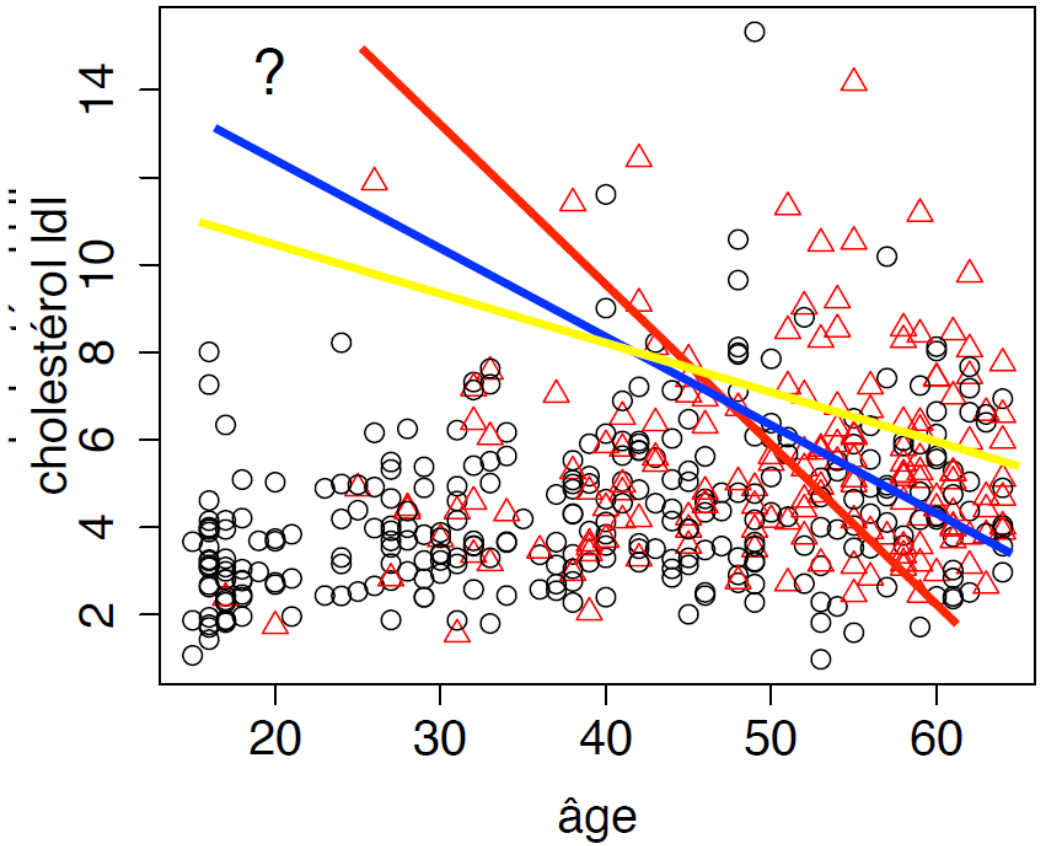
- Approche peu fiable et difficile à mettre en oeuvre, d'autant plus que le
- Nombre de variables explicatives est important.

# 1er Exemple : prédiction de l'infarctus du myocarde

**Approche de type ML ➔ Premier modèle : Régression logistique**

**Préparation des données**

- Constituer une base d'apprentissage
- Comment distinguer / séparer au mieux les deux classes ?
- Trouver la frontière
- Modèle linéaire :
  la droite séparatrice optimale ?



# 1er Exemple : prédiction de l'infarctus du myocarde

**Approche de type ML ➔ Premier modèle : Régression logistique**

- On ne peut pas prédire à coup sûr l'occurence d'un IM à partir de l'âge et du taux de cholestérol, mais on peut chercher à estimer sa probabilité

  notée p(x) = $\mathbb{P}(\underbrace{\text{IM}}_{y=1} \mid \underbrace{\text{âge, ldl}}_{\mathbf{x}})$

- Modèle linéaire, on pose : $\ln \frac{p(\mathbf{x})}{1-p(\mathbf{x})} = w_0 + w_1 \times \text{âge} + w_2 \times \text{ldl}$

- Formulation équivalente : $p(\mathbf{x}) = \frac{1}{1+\exp(-w_0 - w_1 \times \text{âge} - w_2 \times \text{ldl})}$

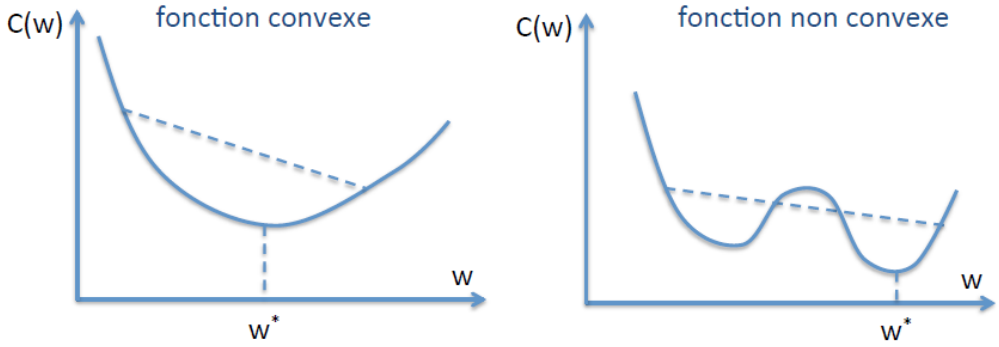
- Problème : comment déterminer les coefficients w0, w1 et w2 ?
  - Si y = 1, on veut avoir p(x) aussi grand que possible.
    On définit l'erreur dans ce cas par *- ln (p(x))*
  - Symétriquement, si y = 0, on veut avoir p(x) aussi petit que possible.
    L'erreur est alors *- ln (1 - p(x))*
  - Formule générale : $\text{erreur} = -y \ln p(\mathbf{x}) - (1-y) \ln (1 - p(\mathbf{x}))$
  - Erreur totale pour un ensemble d'apprentissage {(x1; y1), … ,(xn; yn) } mesurée par l'**entropie-croisée** : $C(\underbrace{w_0, w_1, w_2}_{\mathbf{w}}) = \sum_{i=1}^{n} \text{erreur}_i$

# 1er Exemple : prédiction de l'infarctus du myocarde

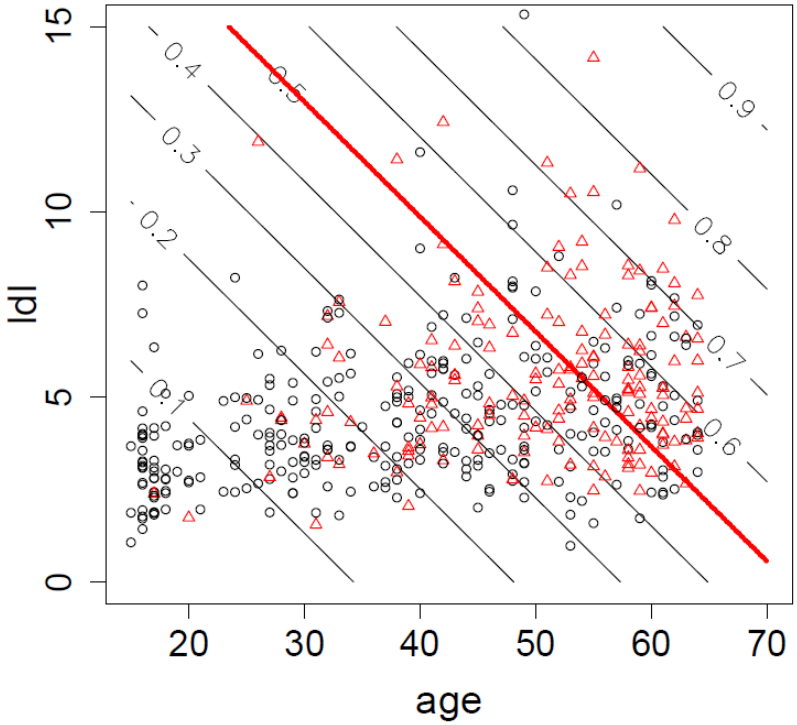
## Approche de type ML ➔ Premier modèle : Régression logistique

### Apprentissage

- Une fois définie une fonction d'erreur, le problème de l'apprentissage devient un problème d'optimisation : rechercher le vecteur de coefficient w* qui minimise l'erreur.
- Dans le cas de la régression logistique, ce vecteur est unique car la fonction d'erreur est convexe.




- Le vecteur solution w* peut être obtenu par un algorithme itératif.



# 1er Exemple : prédiction de l'infarctus du myocarde

**Approche de type ML ➔ Premier modèle : Régression logistique**

**Exploitation**

- Une fois déterminé le vecteur de coefficient w optimal, on dispose d'un **programme classifieur permettant de classer de nouveaux individus**.

**Evaluation des performances**

- Pour estimer la probabilité d'erreur du classifieur, il faut disposer d'un ensemble de test indépendant (**base de test**).
- On construit alors la **matrice de confusion** pour cet ensemble
- Avec 100 exemples

| | | Vraie classe | |
|---|---|---|---|
| | | Positif | Négatif |
| Prediction | Positif | 14 | 10 |
| | Négatif | 21 | 55 |

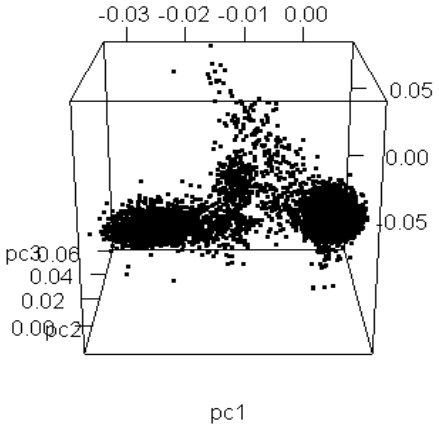
- Taux d'erreur = (10+21)/100=31%.

# 2 – Les Principaux modèles

**RAPPEL ➔ Selon les données disponibles et les objectifs**

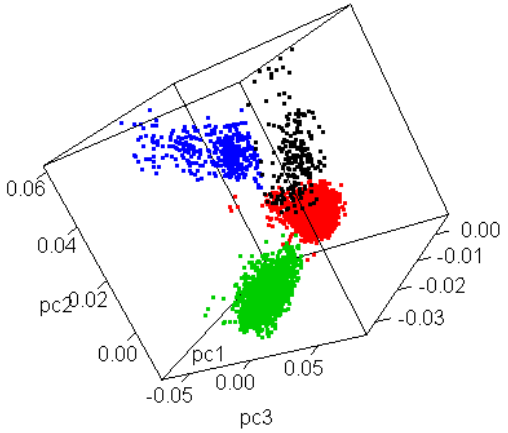
## Apprentissage non supervisé

$\mathbf{Voxel_1} = [F_1\ F_2\ F_3\ \ldots]$
$\mathbf{Voxel_2} = [F_1\ F_2\ F_3\ \ldots]$
$\mathbf{Voxel_3} = [F_1\ F_2\ F_3\ \ldots]$
$\mathbf{Voxel_4} = [F_1\ F_2\ F_3\ \ldots]$
$\mathbf{Voxel_5} = [F_1\ F_2\ F_3\ \ldots]$
.....
$\mathbf{G} = [F_1\ F_2\ F_3\ \ldots]$



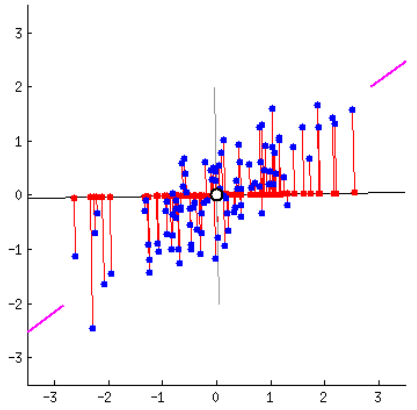

## Apprentissage supervisé

$\mathbf{Voxel_1} = [F_1\ F_2\ F_3\ \ldots]$ [Tumor]
$\mathbf{Voxel_2} = [F_1\ F_2\ F_3\ \ldots]$ [Normal]
$\mathbf{Voxel_3} = [F_1\ F_2\ F_3\ \ldots]$ [Tumor]
$\mathbf{Voxel_4} = [F_1\ F_2\ F_3\ \ldots]$ [Tumor]
$\mathbf{Voxel_5} = [F_1\ F_2\ F_3\ \ldots]$ [Normal]
$\mathbf{G_G} = [F_1\ F_2\ F_3\ \ldots]$ ↑
$\mathbf{G_R} = [F_1\ F_2\ F_3\ \ldots]$ **Label**




## Régression

$\mathbf{Voxel_1} = [F_1]$
$\mathbf{Voxel_2} = [F_1]$
$\mathbf{Voxel_3} = [F_1]$
$\mathbf{Voxel_4} = [F_1]$
$\mathbf{Voxel_5} = [F_1]$
.....
$\mathbf{G} = [F_1]$

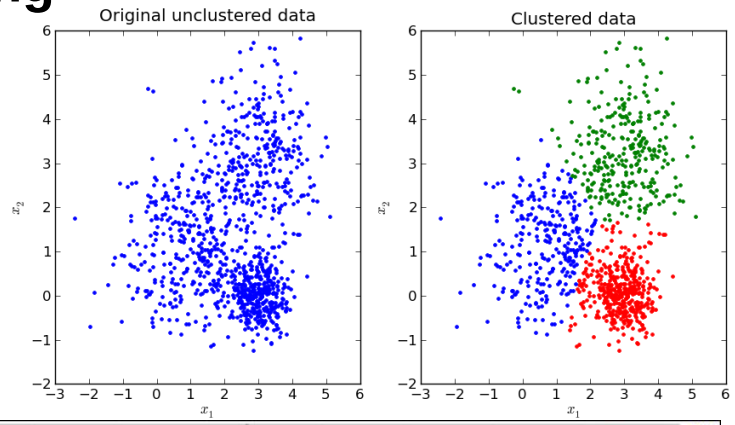


# 2 – Les Principaux modèles

## Unsupervised classification : k-means clustering

- No tagged data available ⇨ learning impossible
- We look for k classes starting from k centers ($G_i$)

**Objectif :** minimising the intra-class variance



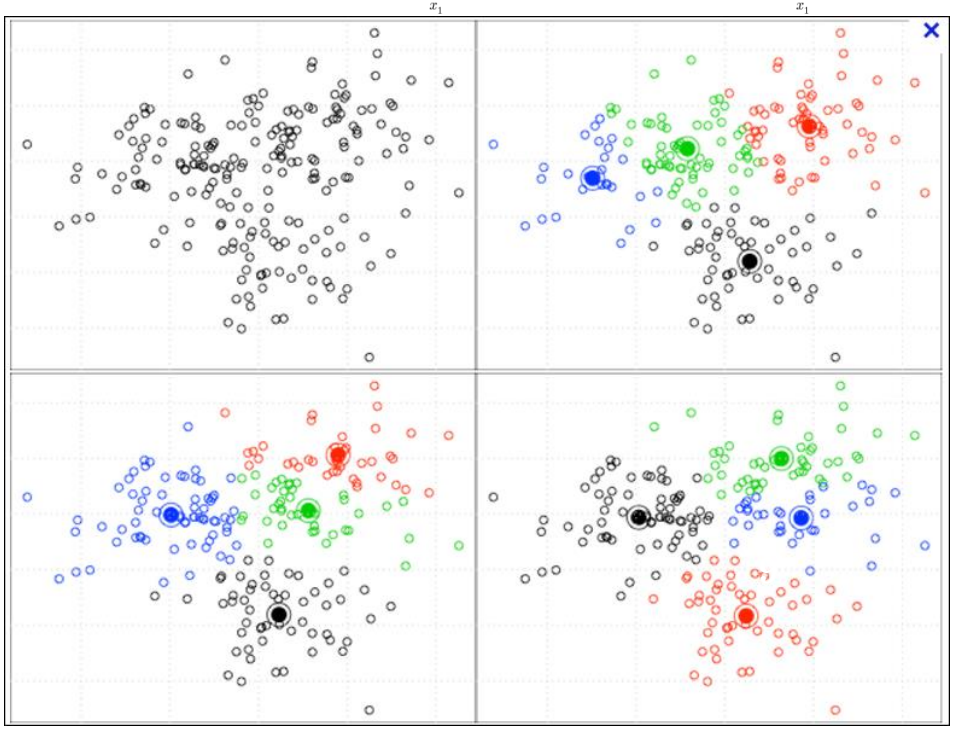

**Algorithm:**

**1 - Choose K centers randomly**

**2 – Repeat :**

**a/ Allocate each x to the closest center Gi**

**b/ Compute the new Gi until stabilization**



**Dependent from initialization →**

# 2 – Les Principaux modèles

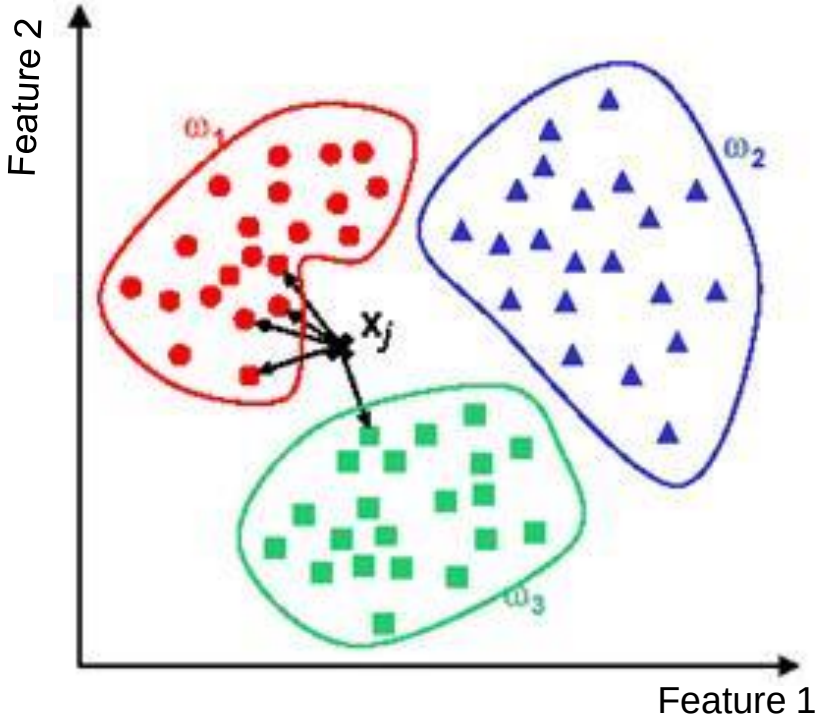
## Clustering on images

Group together pixels by color, automatic segmentation: k-means, k = 2

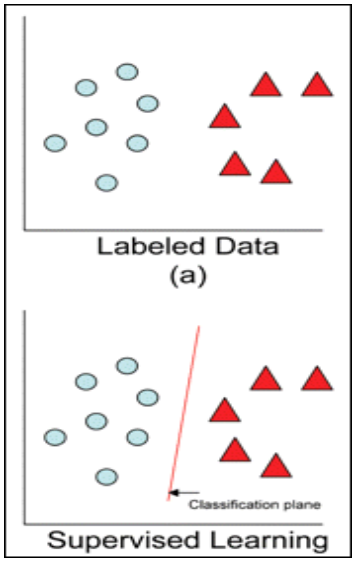
# 2 – Les Principaux modèles

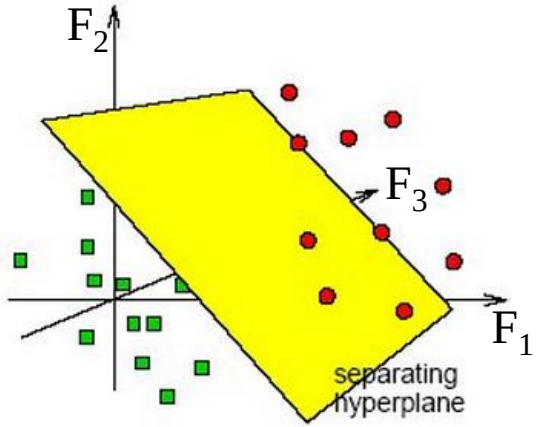
**Supervised classification : k-Nearest-Neighbors (kNN)**

- We have a training set with feature vectors tagged with the corresponding classes ($w_i$)
- The unknown vector $X_j$ is classified with/inside the most represented class among its k nearest neighbors




**Or other sophisticate techniques for classifier definition = ML**



# 2 – Les Principaux modèles

**Supervised classification : k-Nearest-Neighbors (kNN)**

- We have a training set with feature vectors tagged with the corresponding classes ($w_i$)
- The unknown vector $X_j$ is classified with/inside the most represented class among its k nearest neighbors

**Algorithme 1 :** Algorithme implémentant la règle des k-ppv

**Données :**
- *appbase* : base d'exemples de référence, contenant les valeurs des descripteurs
- *etiquettes* : étiquettes des exemples de la base de référence
- *individu* : exemple à classer
- $k$ : nombre de voisins à prendre en compte

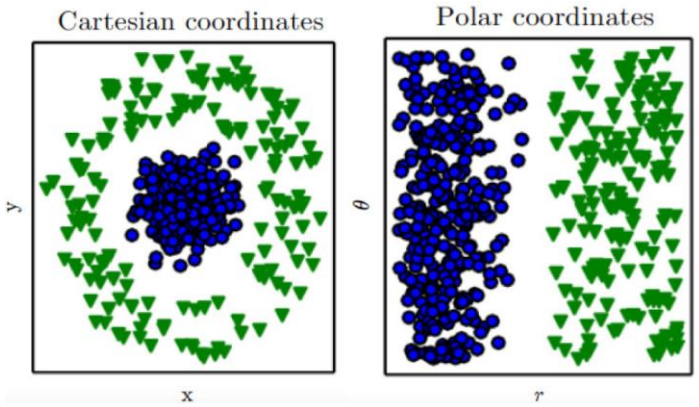
**Résultat :**
- *classe* : étiquette de la classe proposée

**début**
  **pour** *chaque individu* $I_r \in$ *dans appbase* **faire**
    dist[r] ← distance (individu, $I_{app}$)
  **end**
  Trier conjointement (*dist*, *etiquettes*) sur la valeur de *dist* croissante
  *classe* ← étiquette majoritaire dans *etiquettes*$[0 : k]$
  **retourner** *classe*
**end**

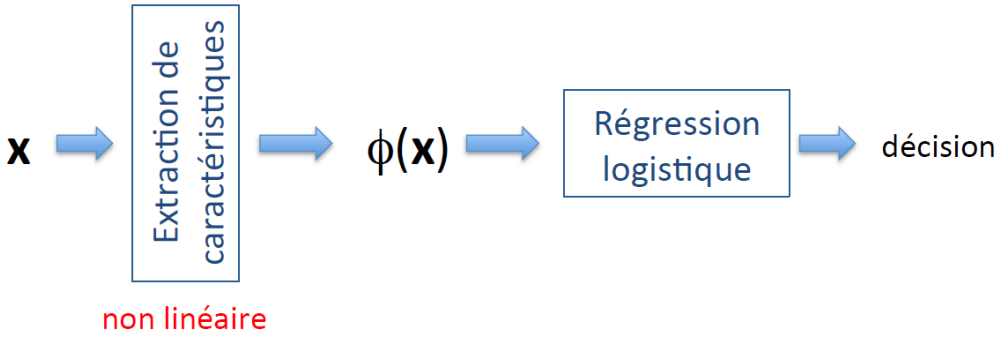
# Problème du choix de l'espace de representation

➔ **Pour aller plus loin que les modèles linéaires…**

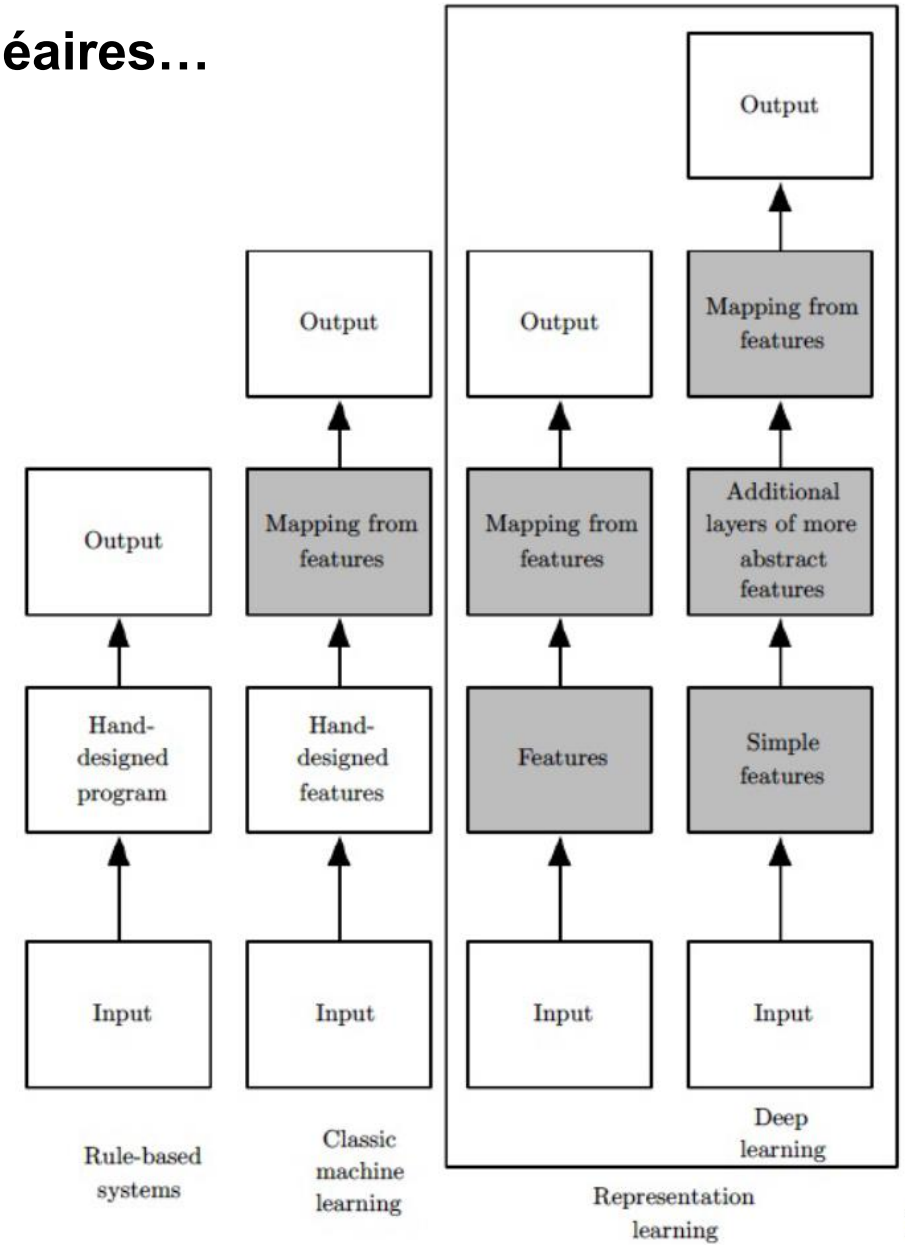
- Changement de representation




- Classifieur linéaire généralisé



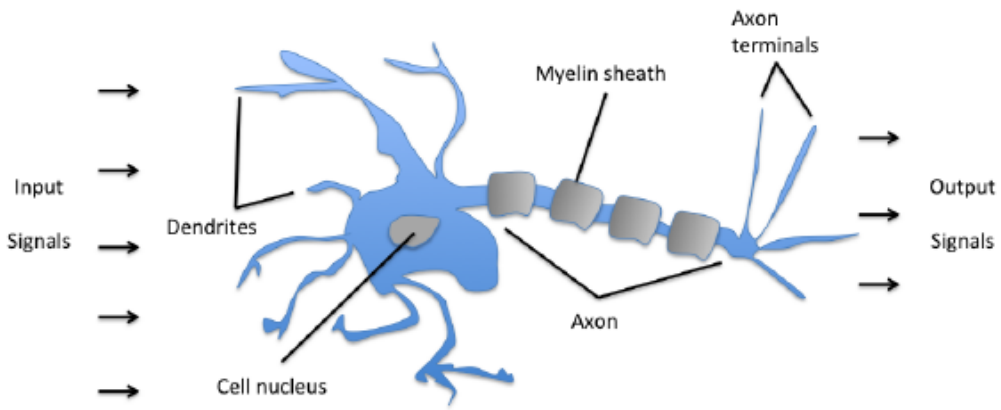

- Architectures plus profondes



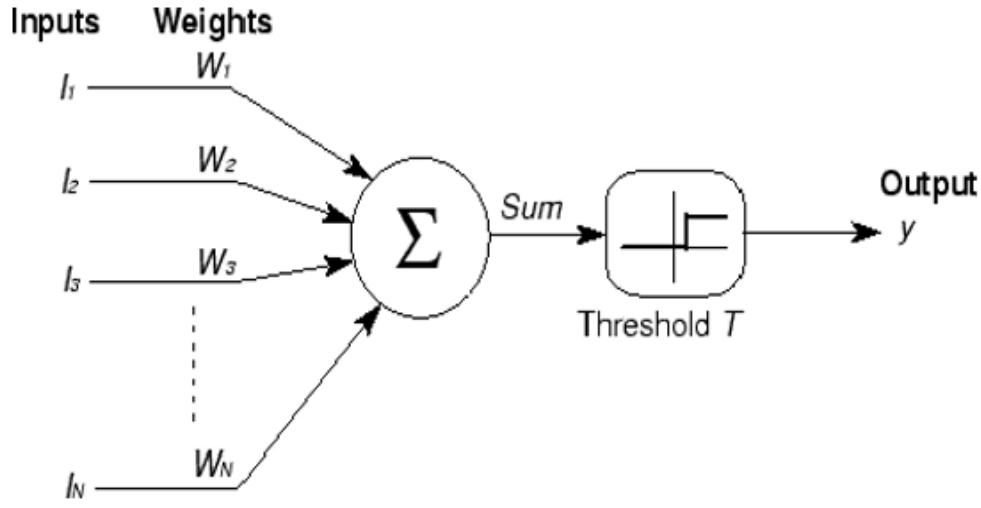
# Réseaux de neurones

## Le modèle de McCulloch et Pitts

- W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, 5(4) :115-133, 1943.
- Idée : neurones biologiques vus comme des portes logiques effectuant des opérations de la logique booléenne



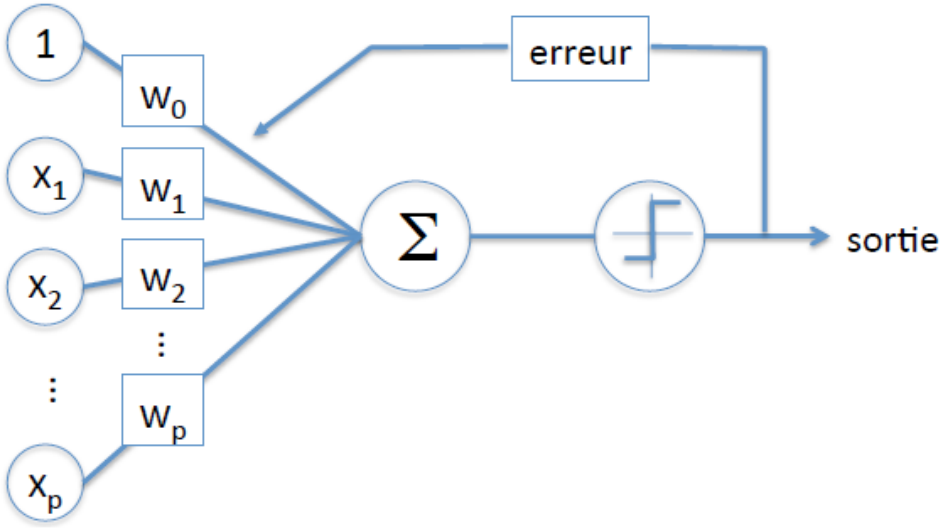

Schematic of a biological neuron.



# Réseaux de neurones

## Le Perceptron

- F. Rosenblatt. The perceptron, a perceiving and recognizing automaton (Project PARA). Cornell Aeronautical Laboratory, 1957.
- Idée : une algorithme qui apprend les poids pour résoudre des problèmes de classification binaire.

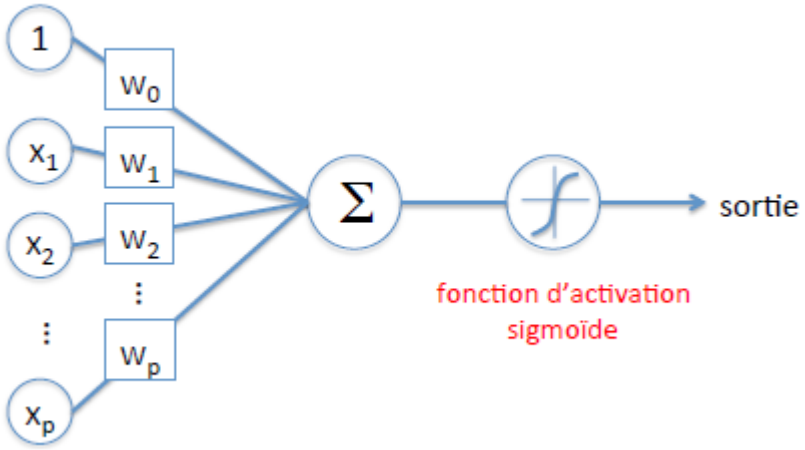



Limites :

- L'algorithme ne converge que si les deux classes sont bien séparées
- Difficilement généralisable à plus de deux classes

# Réseaux de neurones

## Version moderne du perceptron



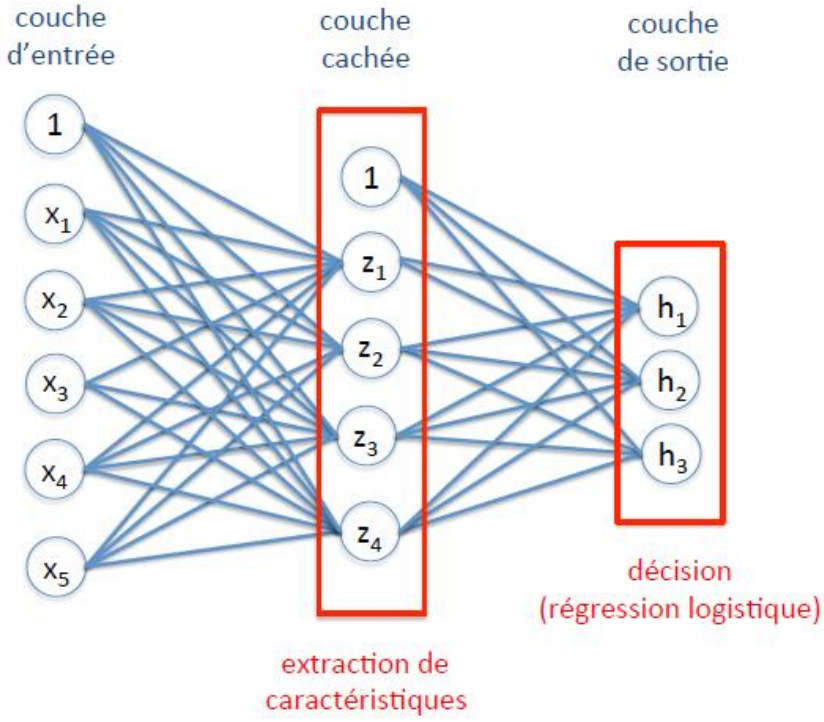

Sortie : $$g(\mathbf{x}) = \frac{1}{1 + \exp\left[-(w_0 + w_1 x_1 + \ldots w_p x_p)\right]}$$

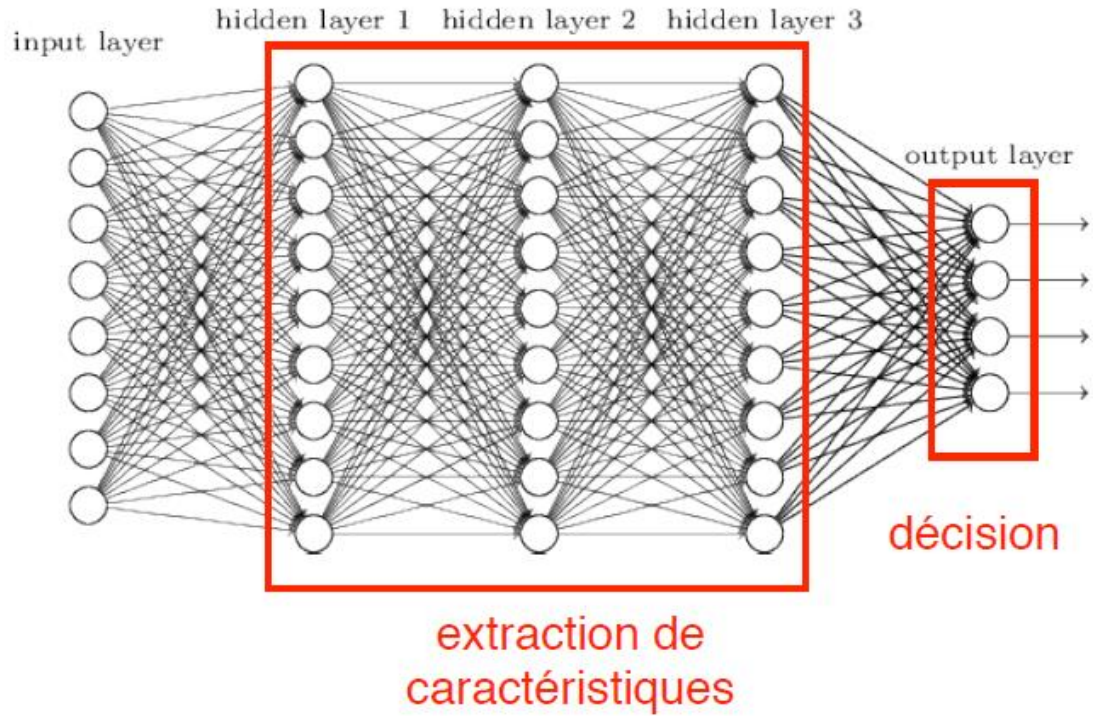
Apprentissage des poids par minimisation de l'entropie croisée

C'est exactement le modèle de la régression logistique !

# Réseaux de neurones

## Vers les reseaux profonds




Comme pour la régression logistique, l'apprentissage des poids se fait en minimisant une fonction d'erreur telle que l'entropie-croisée.

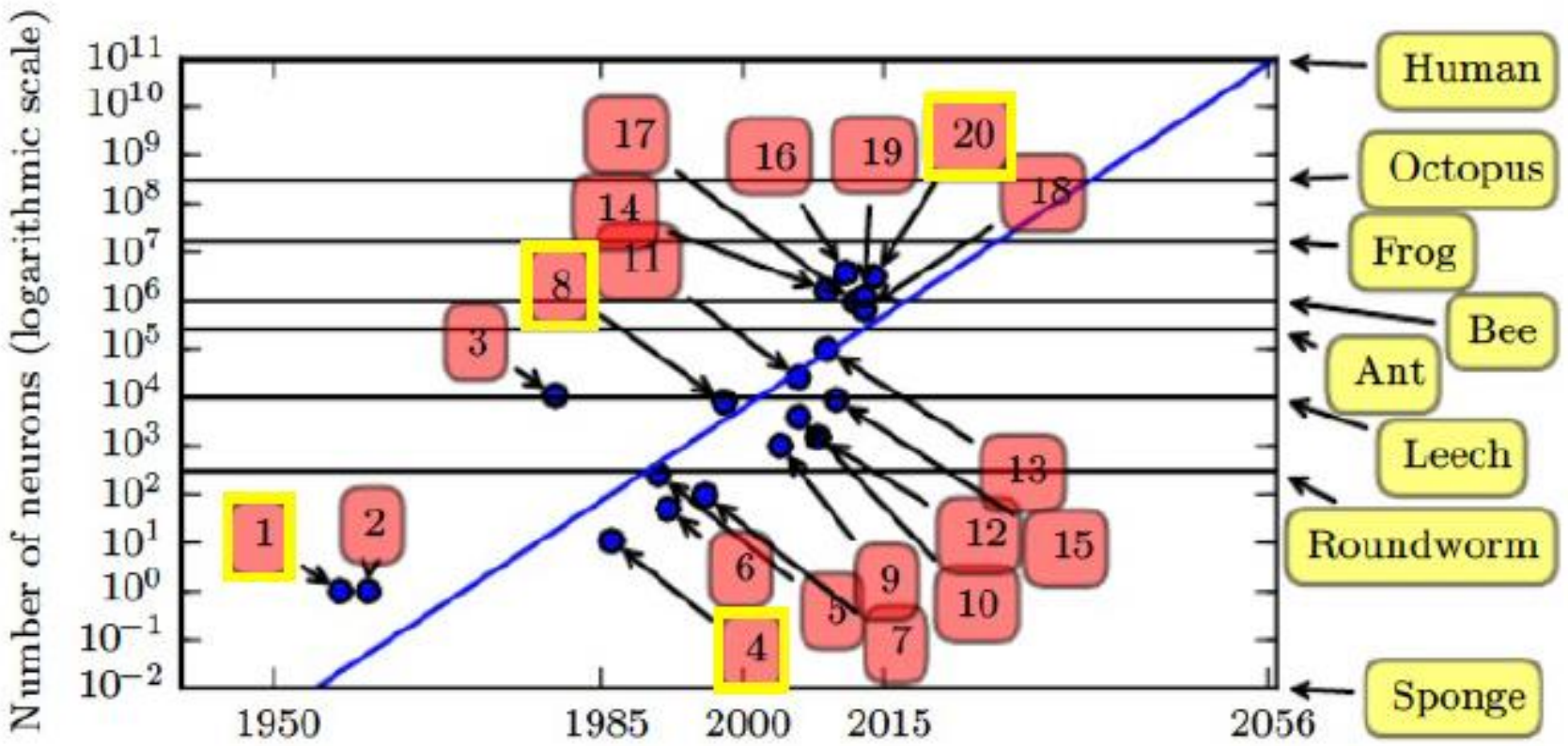
**Problèmes:** Très grand nombre de variables (poids du réseau) - Fonction d'erreur non-convexe - beaucoup de minima locaux

➔ Nécessité de grosses capacités de calcul.

# Réseaux de neurones

## Vers les reseaux profonds

Depuis leur introduction dans les années 1950, la taille des réseaux de neurones a doublé en moyenne toutes les 2:4 années.

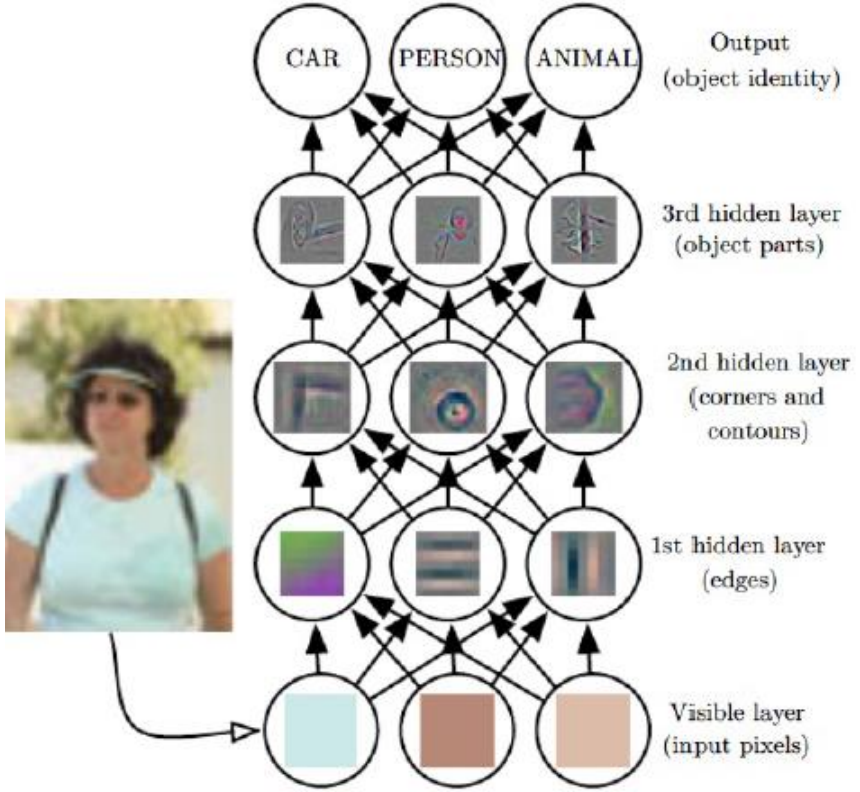


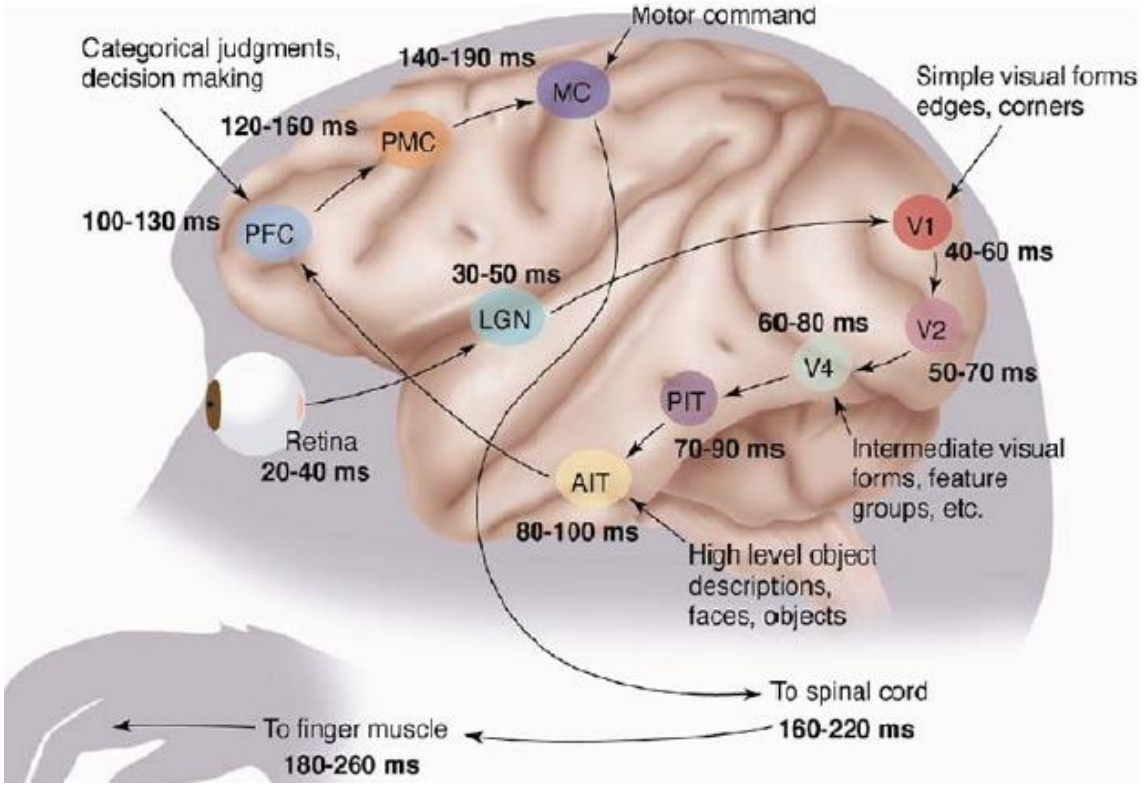

1 : Perceptron ; 4 : Premiers perceptrons multi-couches ; 8 : LeNet-5 (LeCun et al., 1998b) ; 20 : GoogLeNet

# Réseaux de neurones

## Vers les réseaux profonds

Mais on est très loin de répliquer la complexité du cerveau humain !!!



# Bilan

L’IA : un des grands défis scientifiques et technologiques de notre temps

- Questions scientifiques :
  - Intégration de techniques symboliques (raisonnement, gestion des connaissances) et connexionnistes
  - Explication des raisonnements et des décisions
  - Quantification des incertitudes
  - Connaissances a priori en apprentissage (apprendre à partir de peu d’exemples)
  - Etc.
- Questions éthiques (protection de la vie privée, robots autonomes, etc.)
- Questions économiques (automatisation : impacts négatifs et positifs sur l’emploi, transformation de certains secteurs économiques – transports, services, etc.)

# ANNEXE

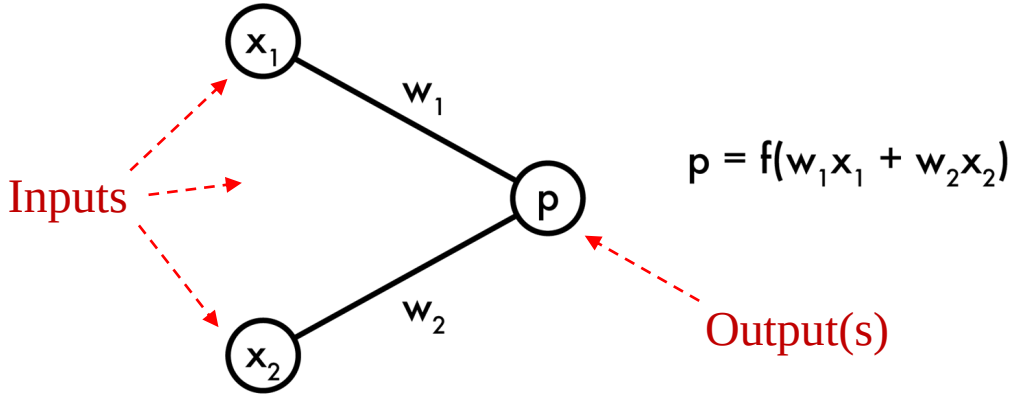
# Overcoming the limitation of ML…

## What is feature engineering?

- Arguably the **core** problem of machine learning (especially in practice)
- ML work well if there is a clear relationship between the inputs (features representing objects: $x_i$) and the outputs ($p$: the function you are trying to model / learn)

## Classifier = Linear model

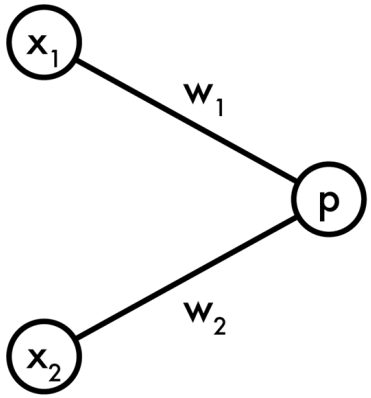
- Compute a probability for each possible class according the selected features
- Linear model means output = linear combination of the inputs
- Learning the weights



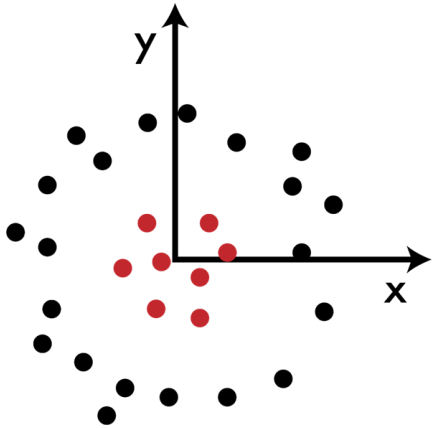
# Overcoming the limitation of ML…

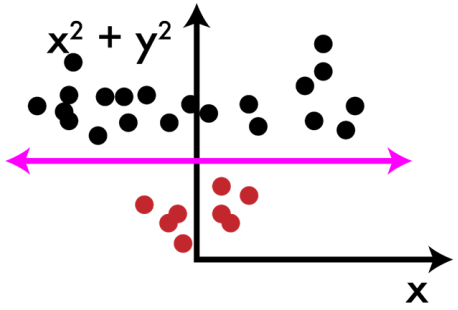
**What is feature engineering?**

- Even with more sophisticate models, generally, feature engineering is just coming up with combinations of the features we already have
- Cannot learn transformations of features, only use existing features. Human must create good features



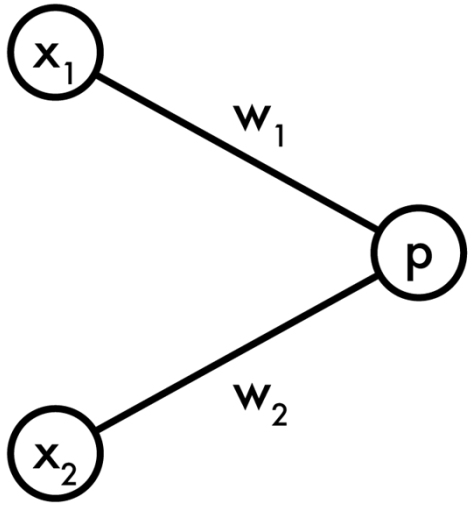

$$p = f(w_1x_1 + w_2x_2)$$



# Overcoming the limitation of ML…

**What if we added more processing?**

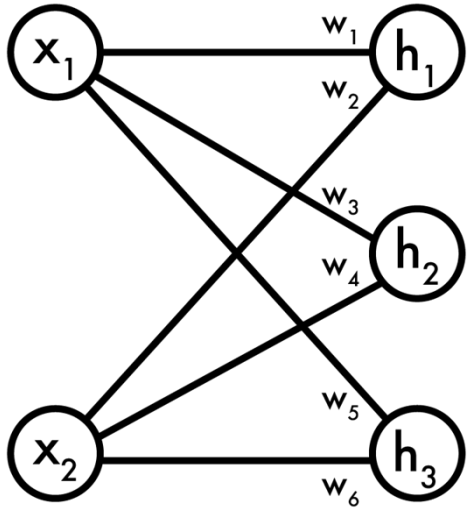



$$p = f(w_1x_1 + w_2x_2)$$

# Overcoming the limitation of ML…

## What if we added more processing?

Create “new” features using old ones. We’ll call **H** our *hidden layer*

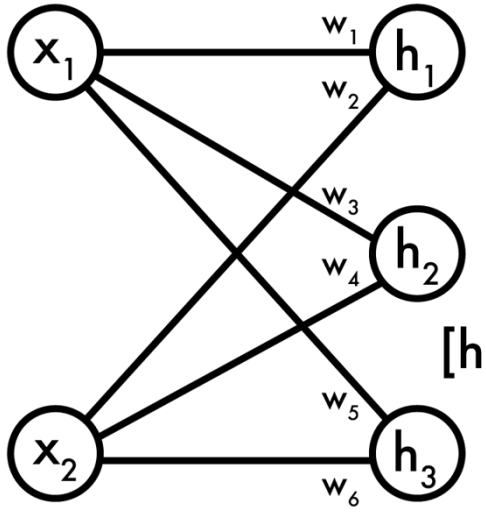



$h_1 = \varphi(w_1 x_1 + w_2 x_2)$
$h_2 = \varphi(w_3 x_1 + w_4 x_2)$
$h_3 = \varphi(w_5 x_1 + w_6 x_2)$

# Overcoming the limitation of ML…

## What if we added more processing?

As with linear model, **H** can be expressed in matrix operations




$$h_1 = \varphi(w_1 x_1 + w_2 x_2)$$
$$h_2 = \varphi(w_3 x_1 + w_4 x_2)$$
$$h_3 = \varphi(w_5 x_1 + w_6 x_2)$$

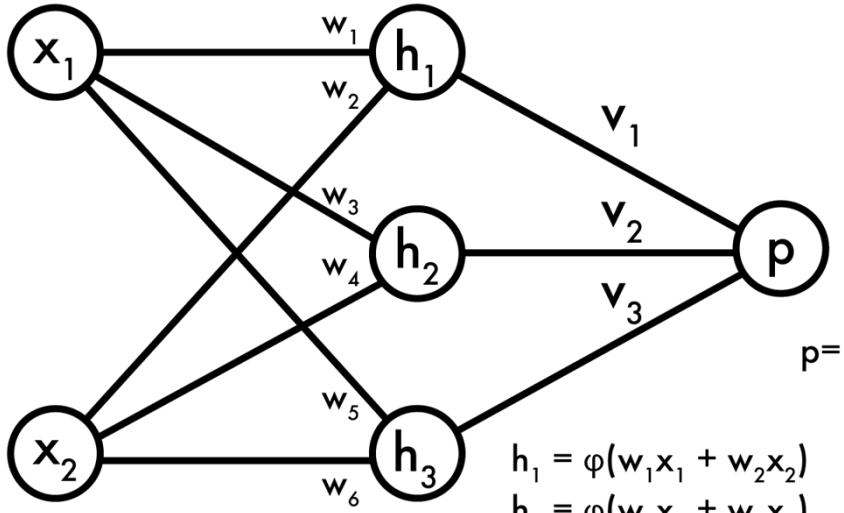
$$[h_1\ h_2\ h_3] = \varphi([x_1\ x_2]\begin{bmatrix} w_1 & w_3 & w_6 \\ w_2 & w_4 & w_5 \end{bmatrix})$$

$$\mathbf{H} = \varphi(\mathbf{X}\mathbf{w})$$

# Overcoming the limitation of ML…

## What if we added more processing?

Now our prediction p is a function of our hidden layer




$p = \varphi(v_1h_1 + v_2h_2 + v_3h_3)$

$h_1 = \varphi(w_1x_1 + w_2x_2)$

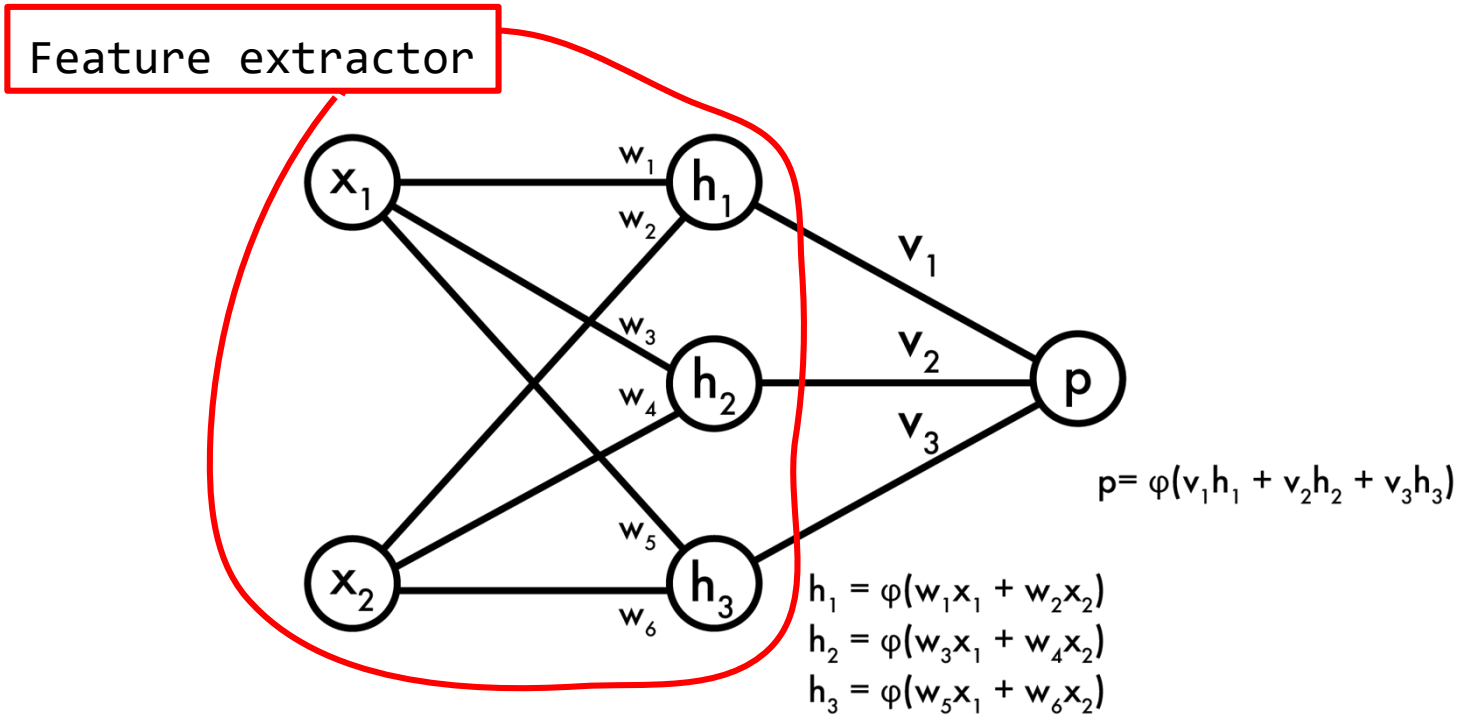
$h_2 = \varphi(w_3x_1 + w_4x_2)$

$h_3 = \varphi(w_5x_1 + w_6x_2)$

# Overcoming the limitation of ML…

## What if we added more processing?

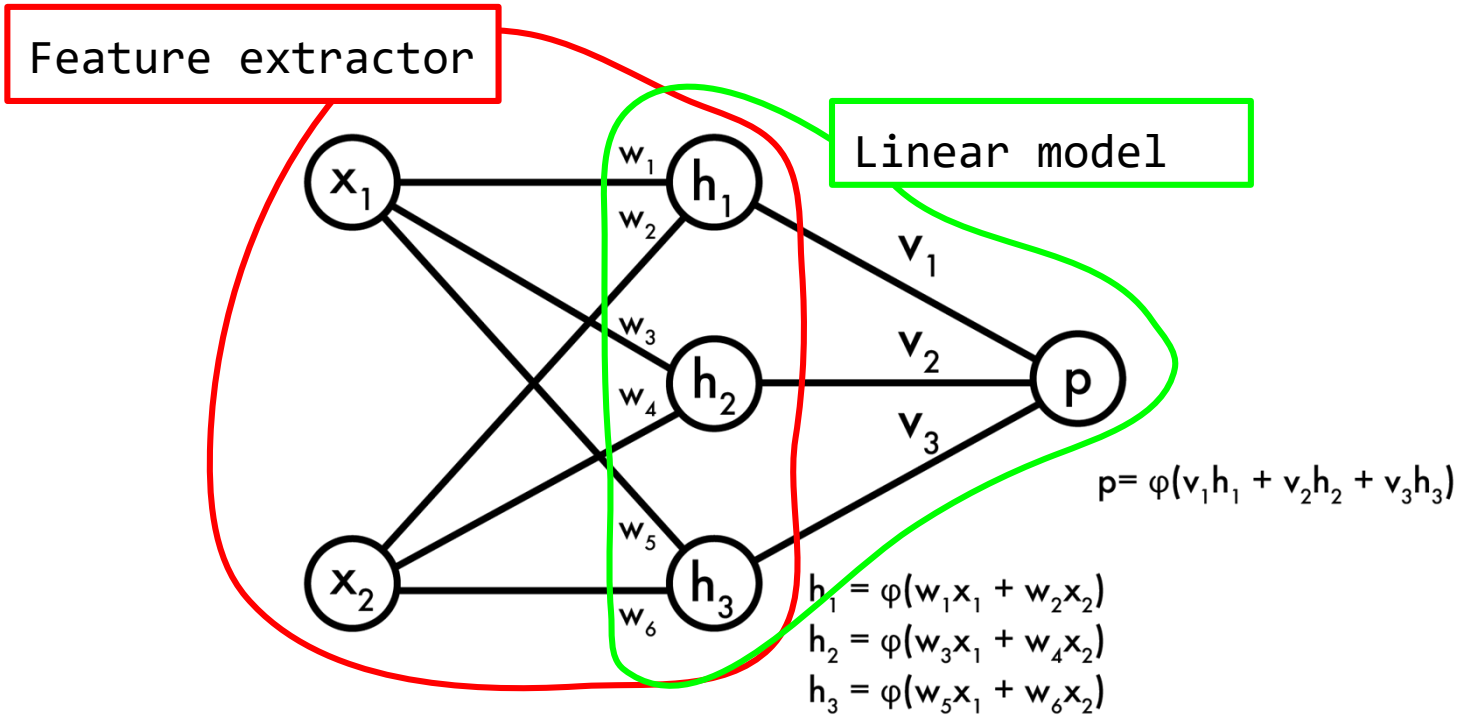
Now our prediction p is a function of our hidden layer



# Overcoming the limitation of ML…

## What if we added more processing?

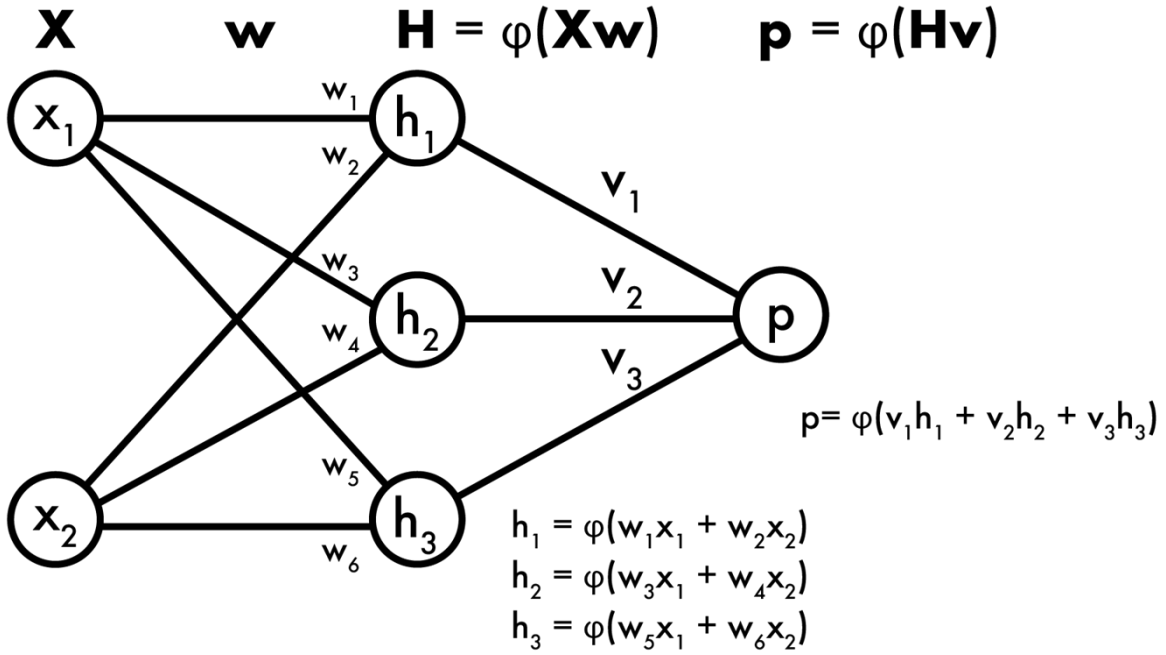
Now our prediction p is a function of our hidden layer



# Overcoming the limitation of ML…

## What if we added more processing?

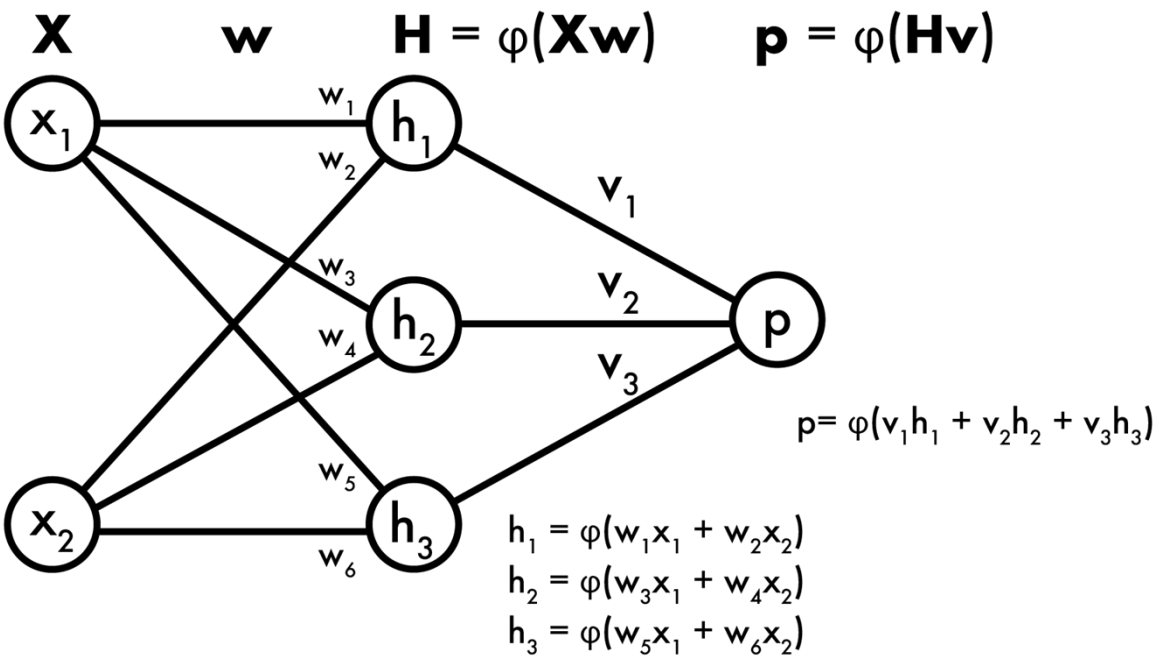
Can still express the whole process in matrix notation! Nice because matrix ops are fast



# Overcoming the limitation of ML…

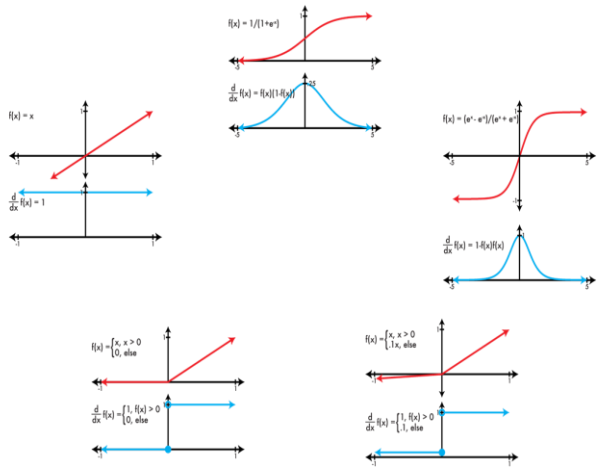
## This is a neural network! (the starting point of Deep Learning)

- This one has 1 hidden layer, but can have **way** more
- Each layer is just **some functions φ** applied to linear combination of the previous layer



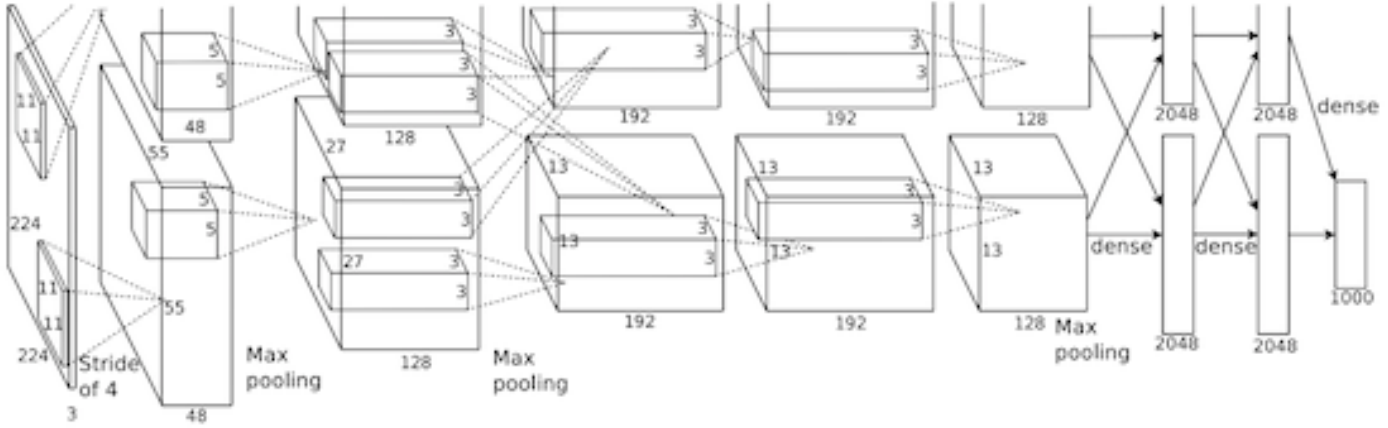

**What about activation functions φ ?**

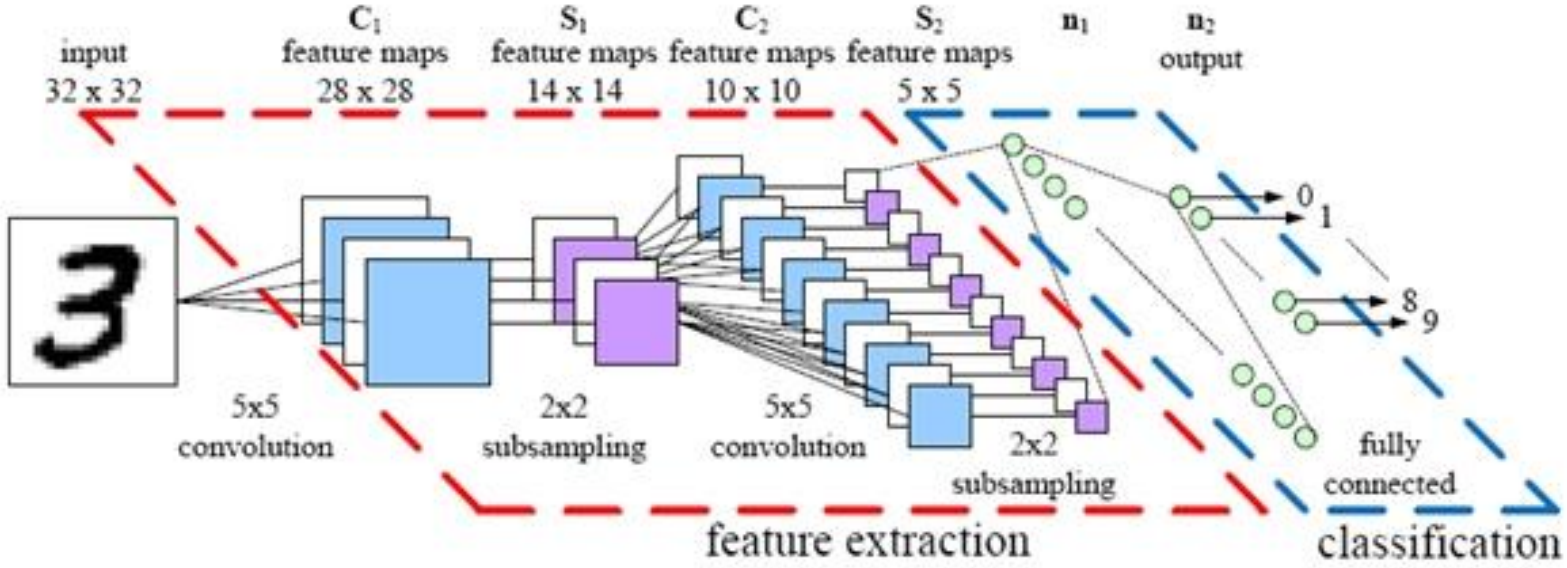
- So many possible options
- want them to be easy to take derivative



# First idea of Deep Learning…

## Deep Learning (Convolutional Neural Networks - CNN)

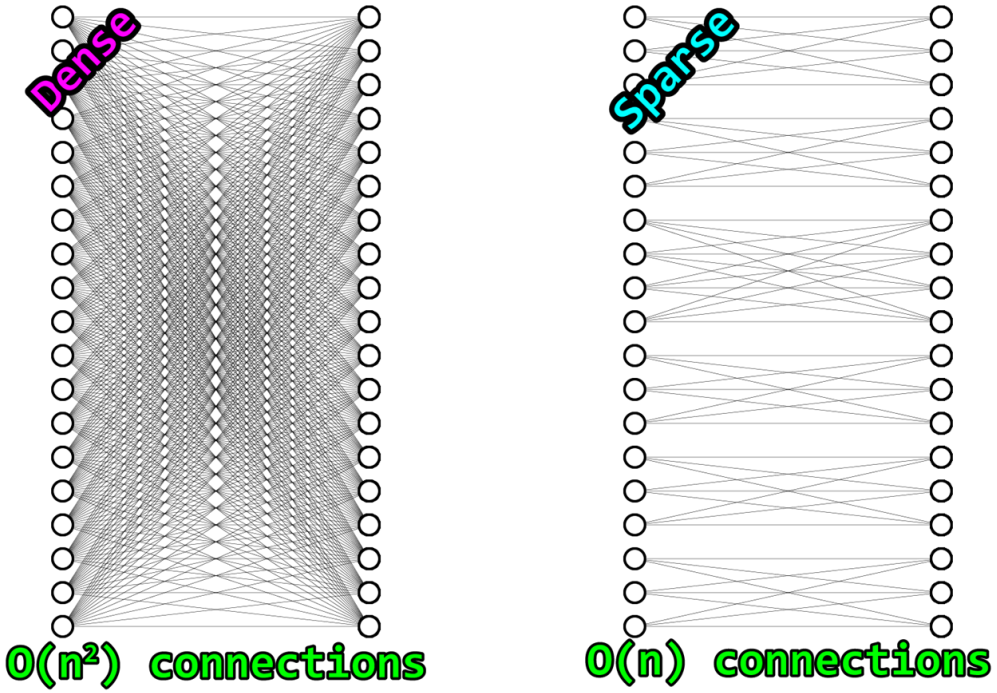


# First idea of Deep Learning…

## Too many weights!

- Would rather have sparse connections
  - Fewer weights
  - Nearby regions - related
  - Far apart - not related
- Convolutions!
  - Just weighted sums of small areas in image
- Weight sharing in different locations in image

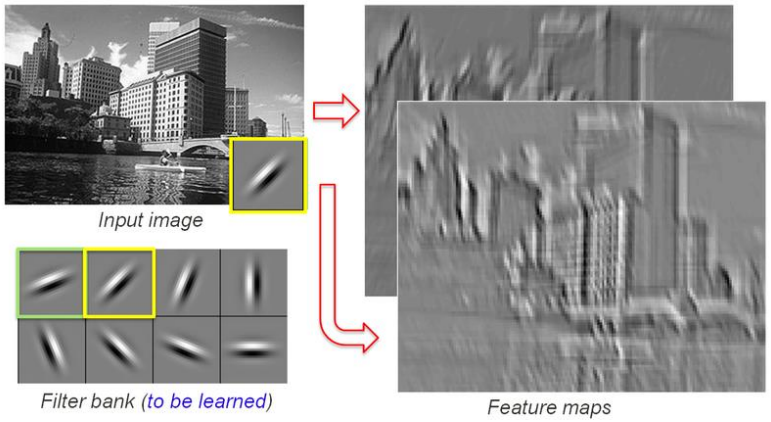


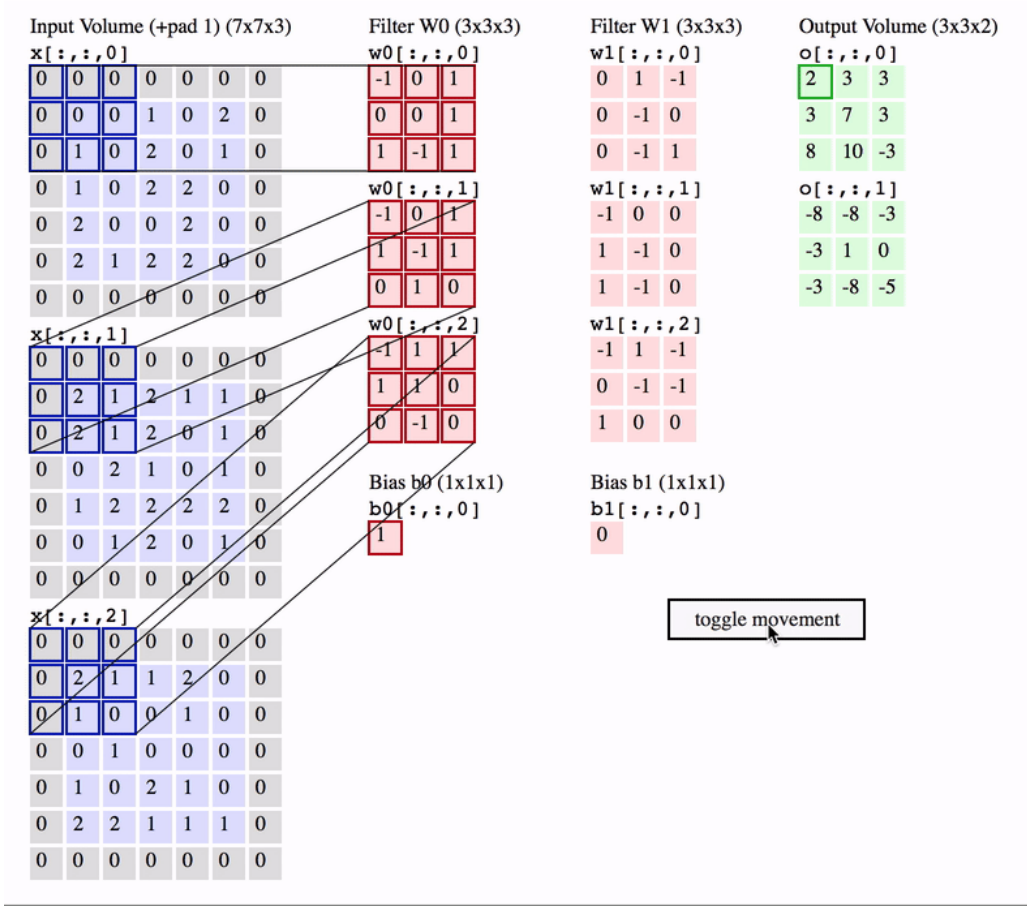
# First idea of Deep Learning…

## Convolutional Layer

- **Input:** an image
  **Processing**: convolution with multiple filters
  **Output**: an image, # channels = # filters

- Output still weighted sum of input (w/ activation)



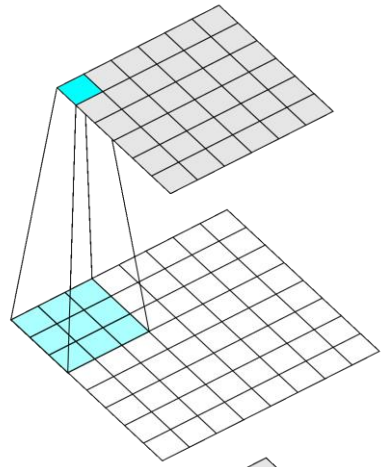




Example of a Conv layer with K = 2 filters, each with a spatial extent F = 3 , moving at a stride S = 2, and input padding P = 1

# First idea of Deep Learning…

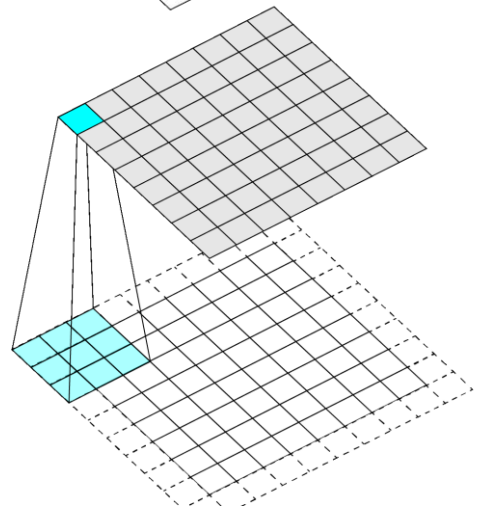
## Padding

- Convolutions have problems on edges
- Do nothing: output a little smaller than input
- Pad: add extra pixels on edge



## Stride

- How far to move filter between applications
- We've done stride 1 convolutions up until now, approximately preserves image size
- Could move filter further, downsample image



## ReLU Layer

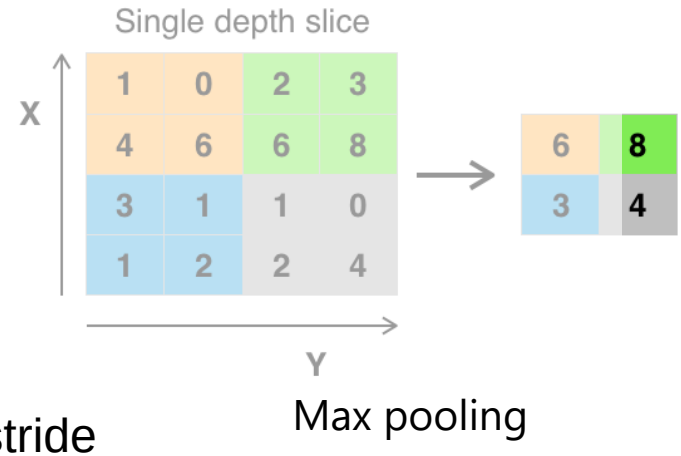
- Applies an elementwise activation function max(0,x)
  ➔ turns negative values to zeros

# First idea of Deep Learning…

## Pooling Layer

- **Input:** an image
  **Processing**: pool pixel values over region
  **Output**: an image, shrunk by a factor of the stride

- Hyperparameters:
  - What kind of pooling? Average, mean, max, min
  - How big of stride? Controls downsampling
  - How big of region? Usually not much bigger than stride




- Most common: 2x2 or 3x3 maxpooling, stride of 2

## Fully Connected Layer

- The standard neural network layer where every input neuron connects to every output neuron

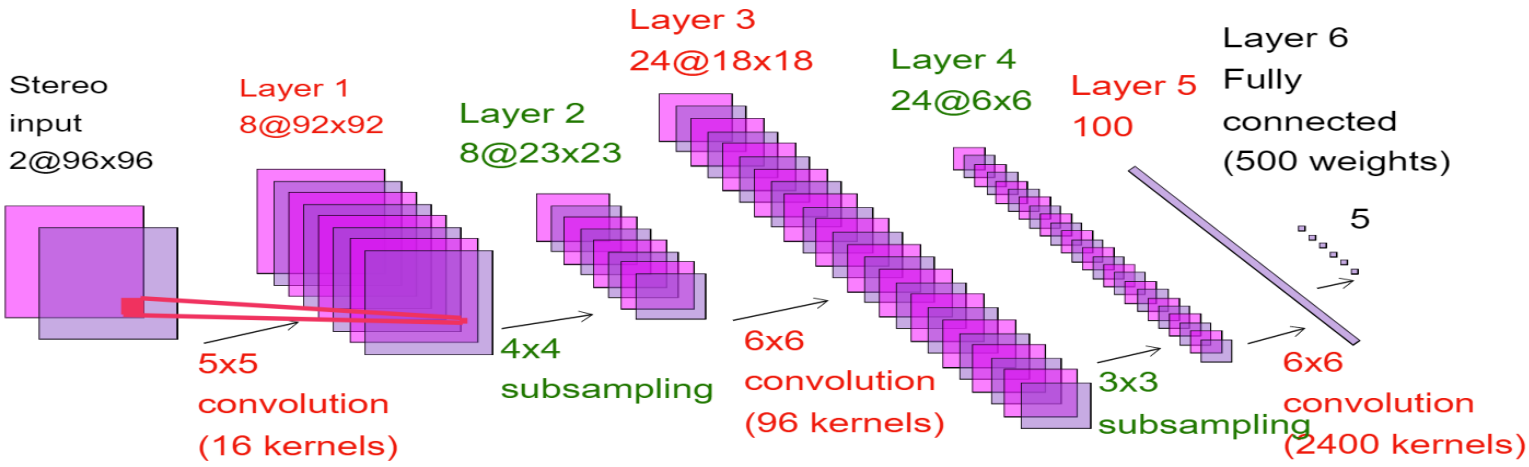
- Often used to go from image feature map -> final output or map image features to a single vector

- Eliminates spatial information

# First idea of Deep Learning…

## Example of Convnet Building Blocks

- Convolutional layers:
  - Connections are convolutions
  - Used to extract features
- Pooling layers:
  - Used to downsample feature maps, make processing more efficient
  - Most common: maxpool, avgpool sometimes used at end
- Connected layers:
  - Often used as last layer, to map image features → prediction
  - No spatial information
  - lots of weights, no weight sharing ➔ Inefficient ?

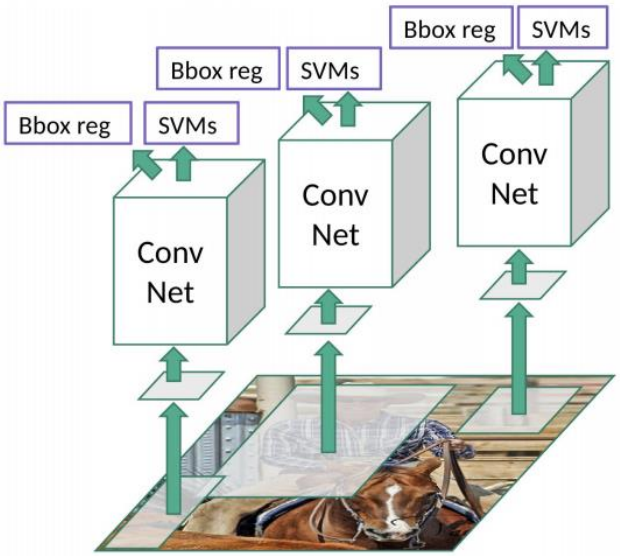


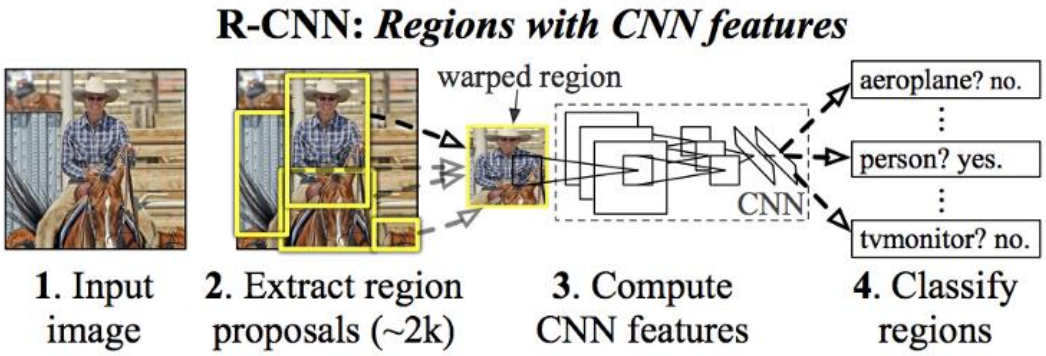
# First idea of Deep Learning…

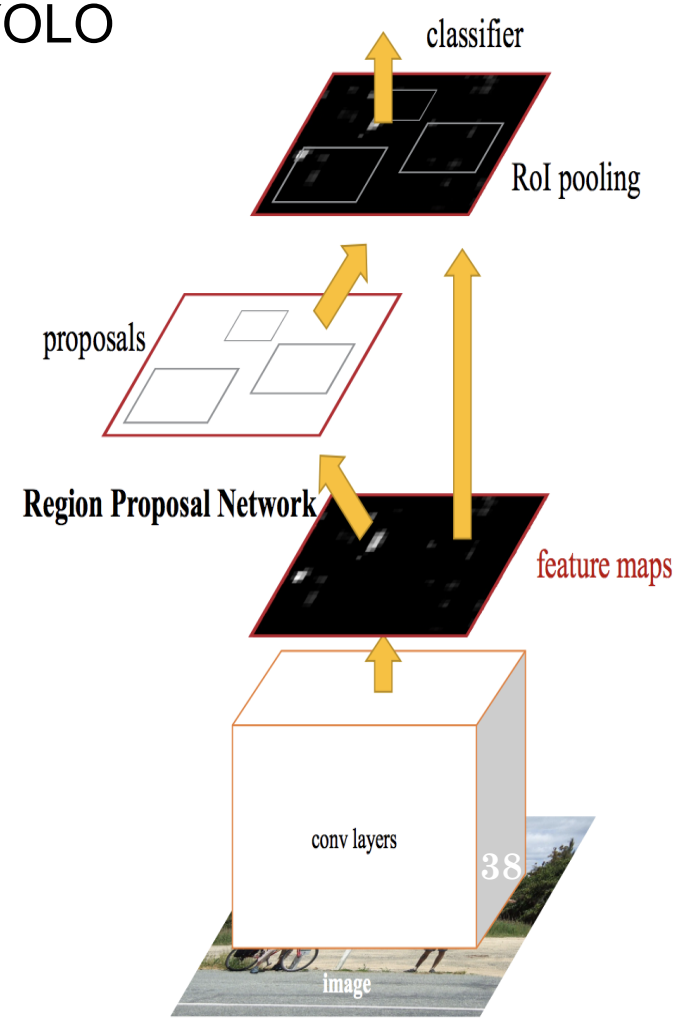
**So many architectures & hyperparameters….**

**➔ Architecture engineering replaces Feature engineering**

- CONVnet, R-CNN, Fast R-CNN, Faster R-CNN, YOLO



# That All for today…

MERCI…

## Support de cours :

- TP : http://www.rfai.lifat.univ-tours.fr/PagesPerso/jyramel/PDF/TP2cesr_eleves.ipynb
- Slides : http://www.rfai.lifat.univ-tours.fr/PagesPerso/jyramel/PDF/HNcesr_ADML2019part2.pdf