

Les Réseaux de Neurones Artificiels

Antoine Cornuéjols

AgroParisTech – INRAE MIA paris-Saclay

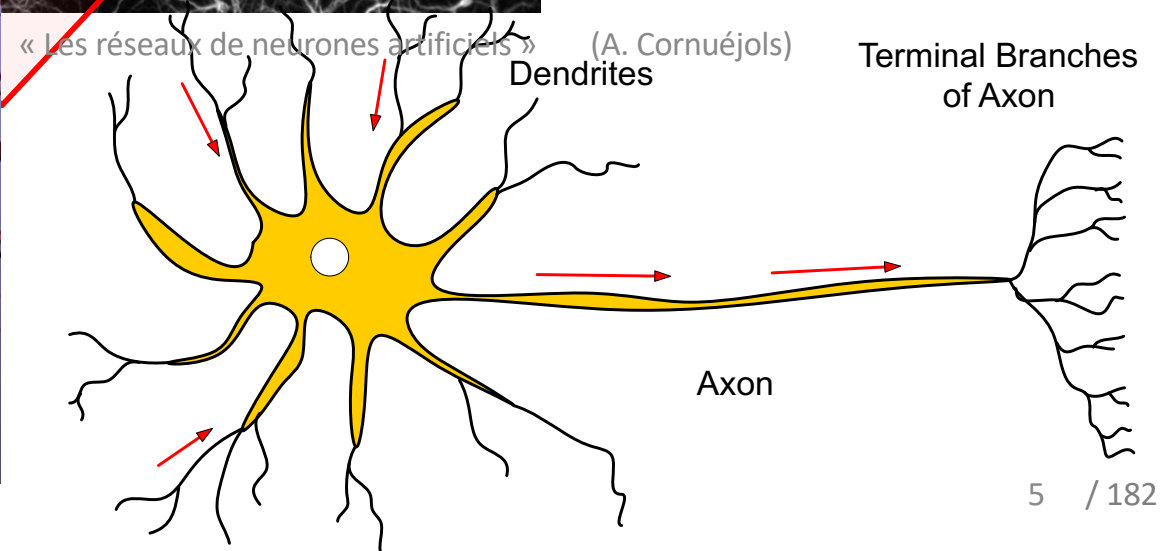
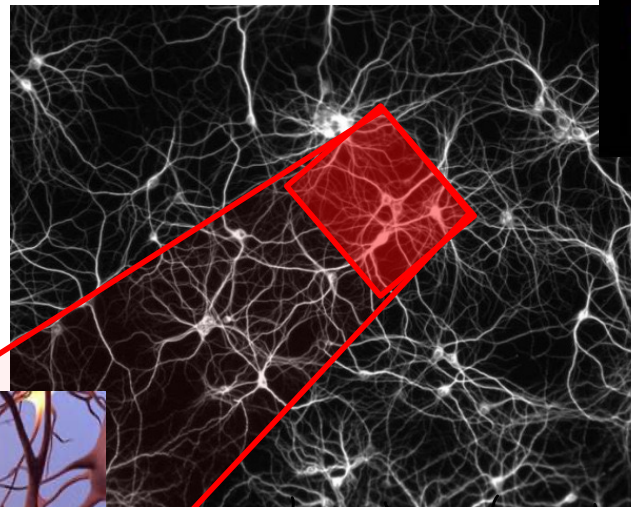
antoine.cornuejols@agroparistech.fr

Plan

1. Introduction et réseaux de neurones à une couche
2. Les perceptrons multi-couches
3. L'apprentissage dans les réseaux de neurones
4. Une introduction aux réseaux de neurones profonds
5. Autres architectures

Introduction

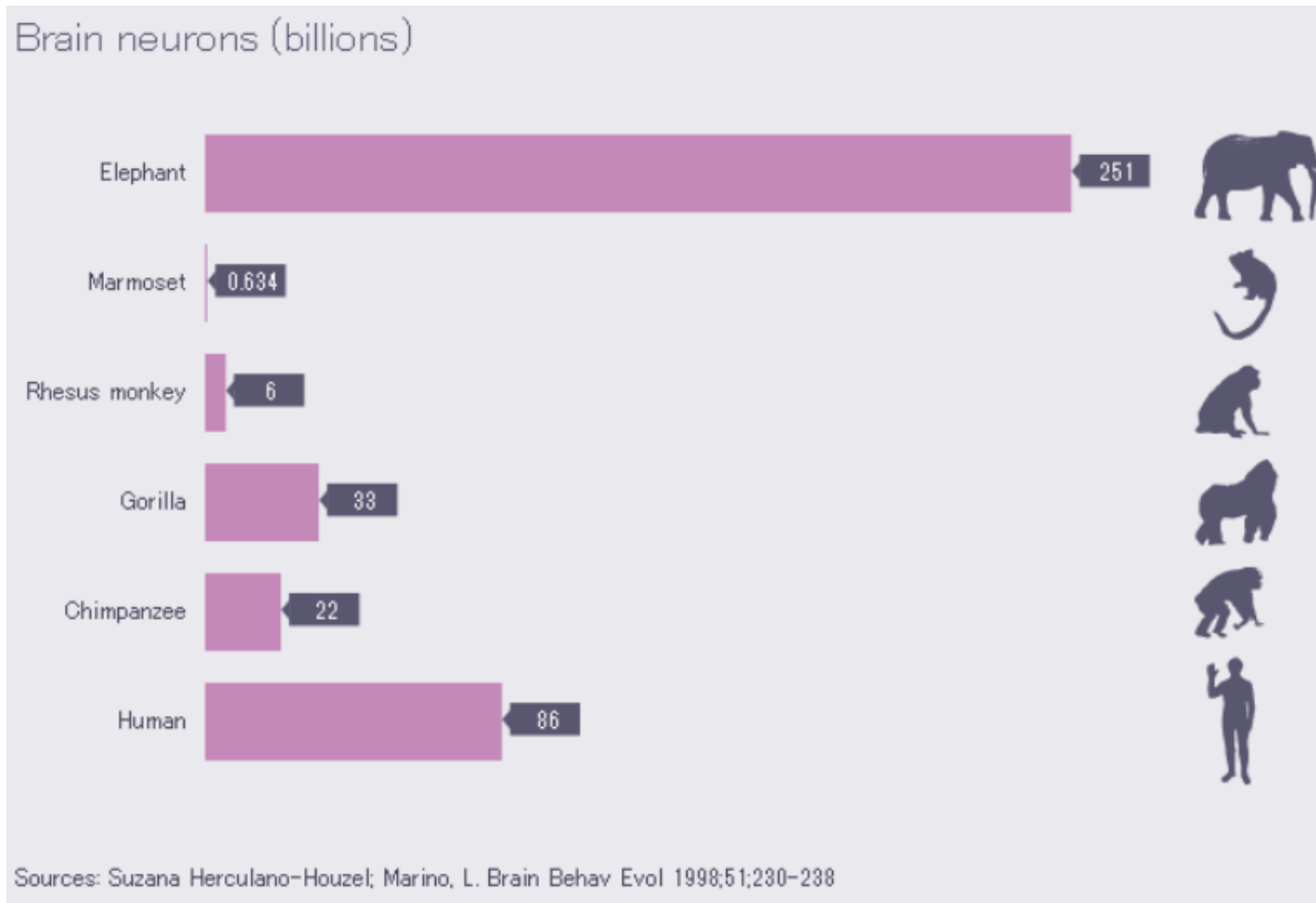
Biological Neurons



Pourquoi considérer des réseaux de neurones ?

- Inspiration biologique
 - Le **cerveau** naturel : un modèle très séduisant
 - Capable d'apprentissage
 - Robuste et tolérant aux fautes
 - S'accommode d'informations incomplètes, incertaines, imprécises
 - Massivement parallèle
 - **Neurones**
 - $\approx 10^{11}$ neurones dans le **cerveau humain**
(950×10^3 dans celui de *l'abeille*)
 - $\approx 10^3$ à 10^4 **connexions / neurone**
 - *Potentiel d'action / période réfractaire / neuro-transmetteurs*
 - *Signaux excitateurs / inhibiteurs*

Sizes of the brains of different species



...

Vers des réseaux de neurones artificiels

- **Attraits**

- Calculs parallélisables
- Robustes et tolérants aux fautes (distribué)
- Algorithmes simples
- D'emploi très général

- **Défauts**

- Opacité des raisonnements
- Et de l'hypothèse produite

Historique (très limité)

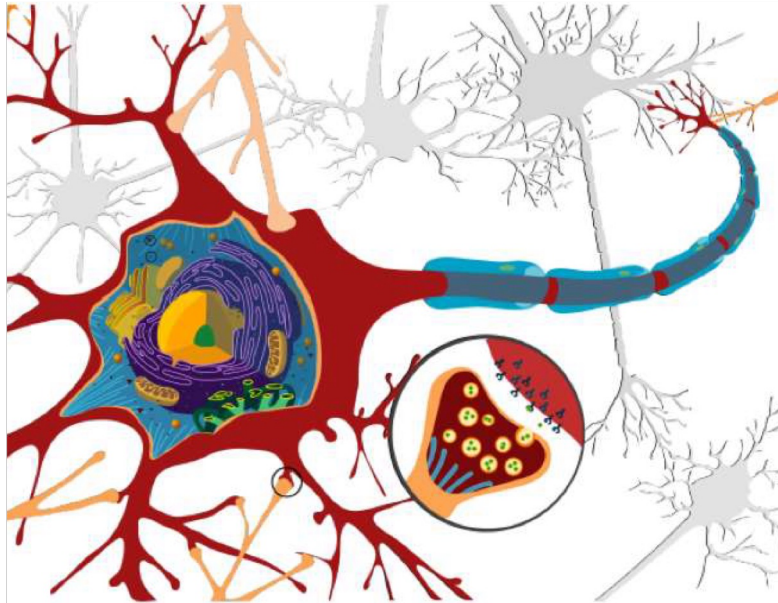
- **Mise en évidence des neurones**
 - Camillio Golgi et Ramon Y Cajal (~ 1870) ; les synapses (dans les années 1930)
- **Modélisation formelle**
 - Mc Culloch & Pitts (1943) : 1^{er} modèle du neurone formel
 - Hebb (1949) : règle d'apprentissage par renforcement de couplage synaptique

Les réseaux à une couche

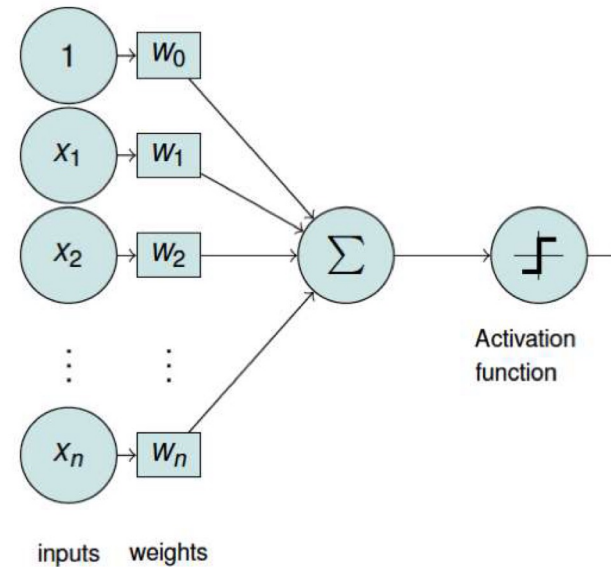
Historique (très limité)

- **Mise en évidence des neurones**
 - Camillio Golgi et Ramon Y Cajal (~ 1870) ; les synapses (dans les années 1930)
- **Modélisation formelle**
 - Mc Culloch & Pitt (1943) : 1^{er} modèle du neurone formel
 - Hebb (1949) : règle d'apprentissage par renforcement de couplage synaptique
- **Premiers modèles informatiques**
 - ADALINE (Widrow-Hoff, 1960)
 - Perceptron (Rosenblatt, 1958-1962)
 - Analyse par Minsky et Papert (1969)

Neurone biologique et « neurone formel »

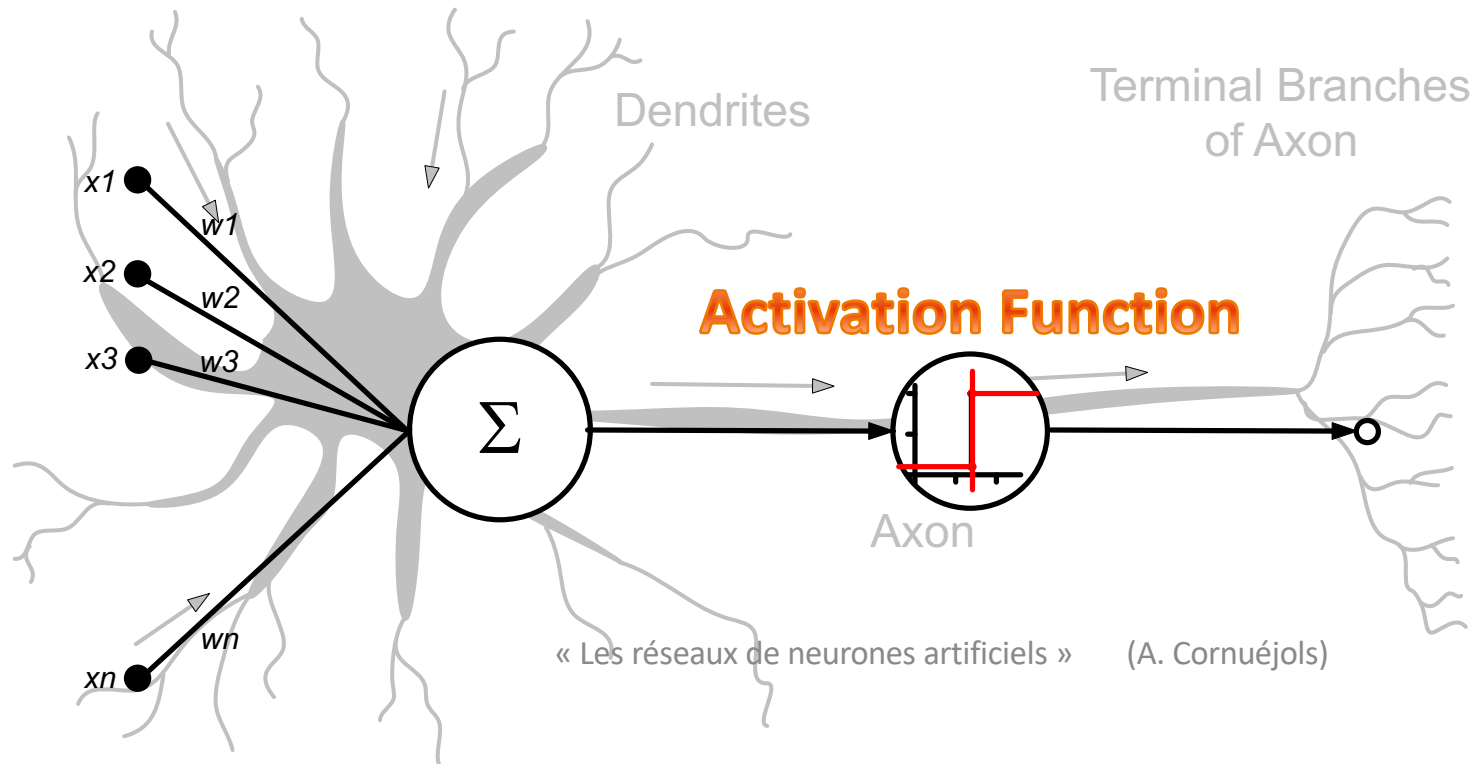


Biological Neuron

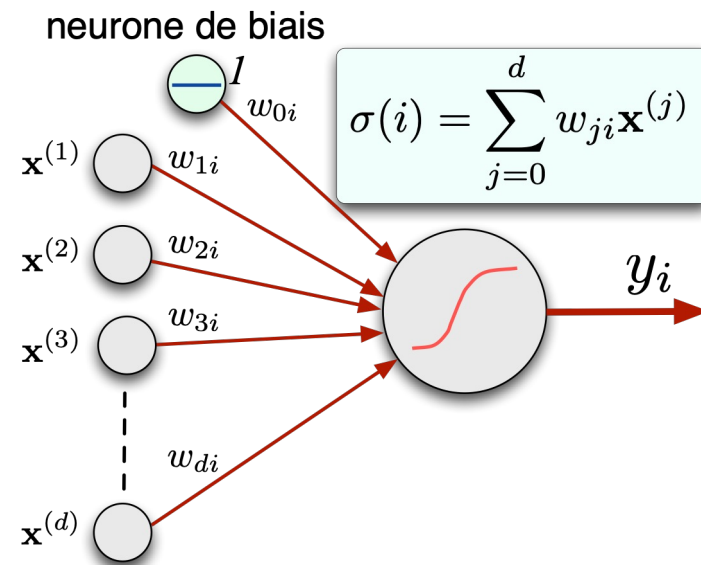
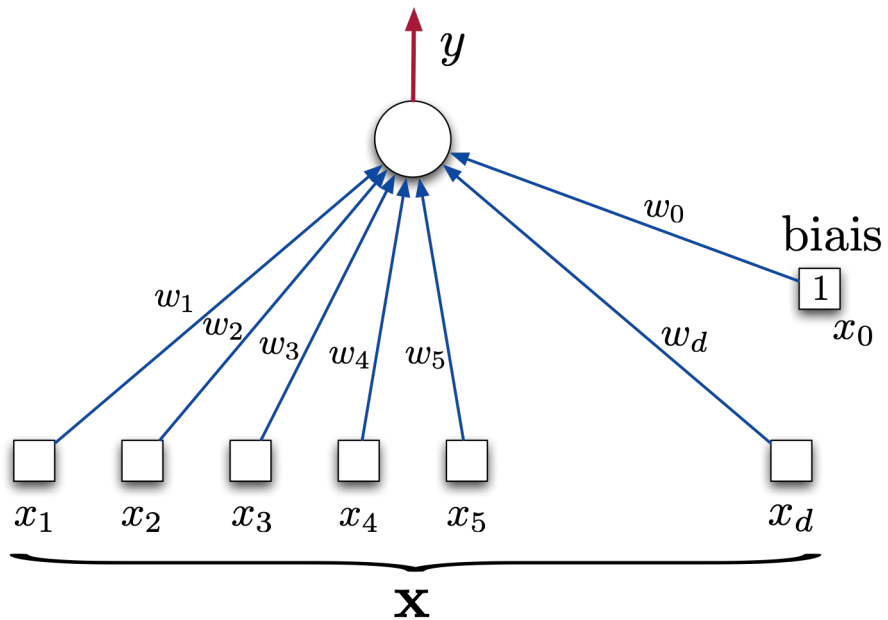


Computational Neuron

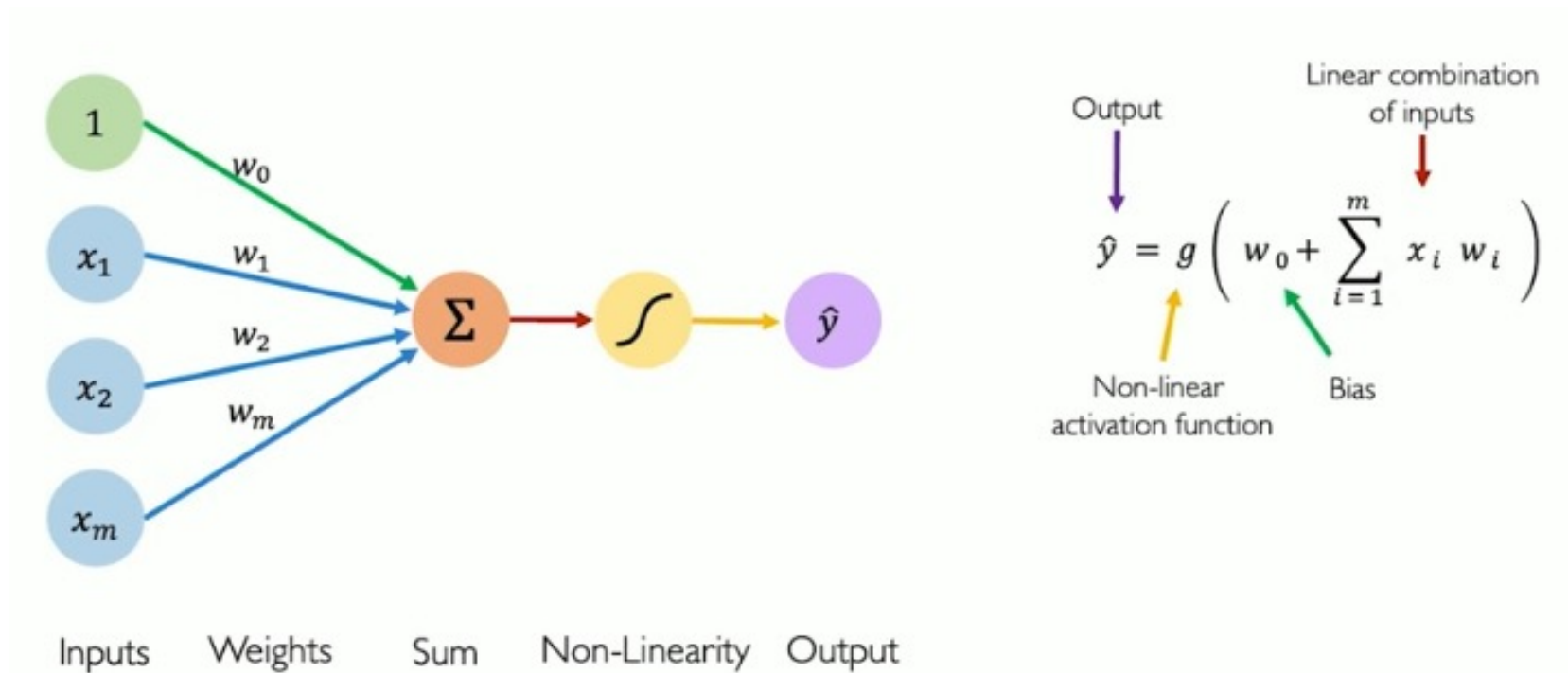
Artificial Neural Networks (ANN)



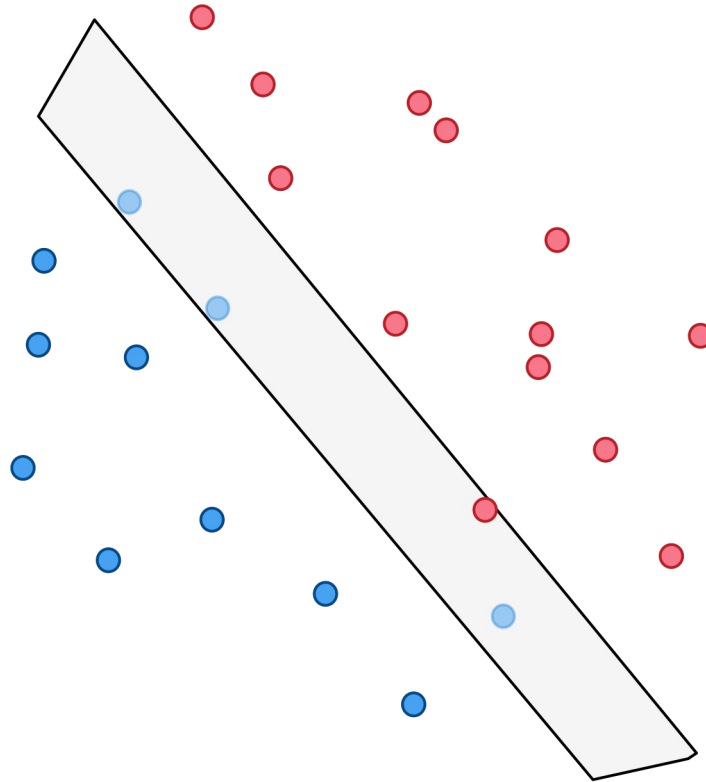
Le perceptron



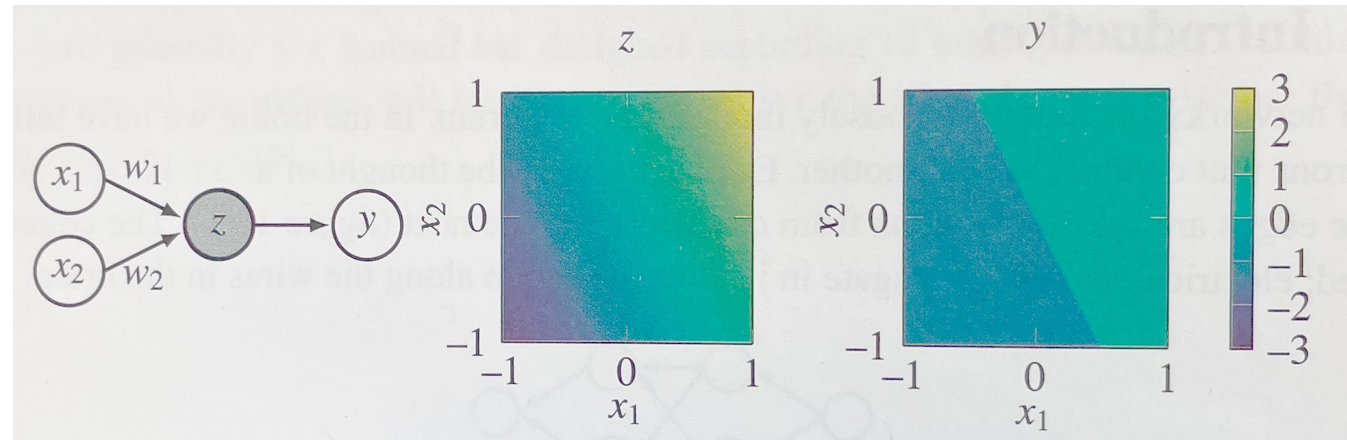
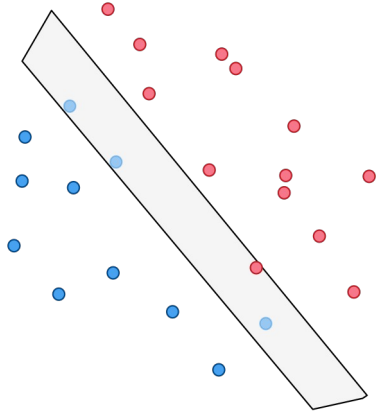
Le perceptron



Le perceptron : un discriminant **linéaire**



Le perceptron : un discriminant **linéaire**



The perceptron: elementary algebra

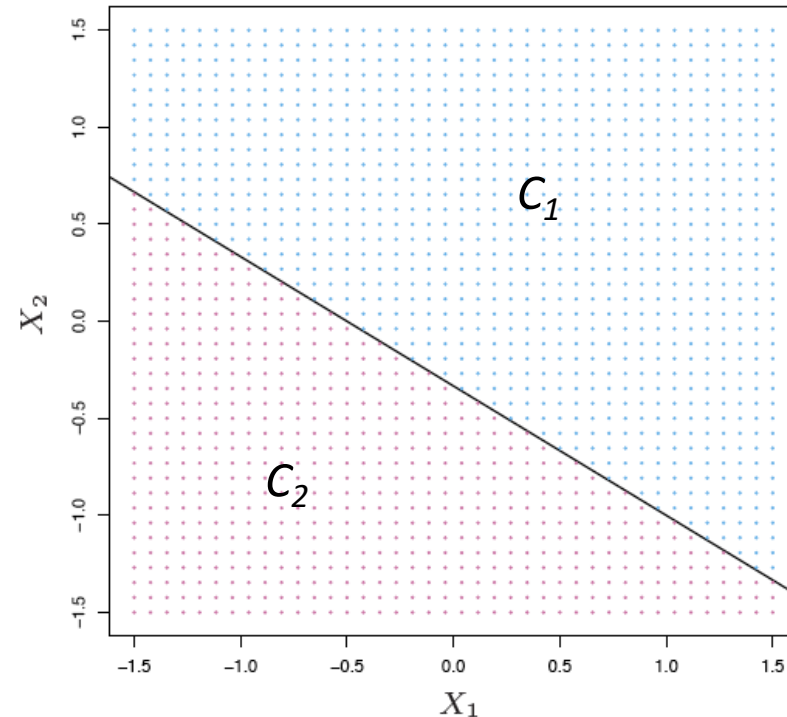
- In the plan

$$1 + 2x_1 + 3x_2 = 0$$

or:
$$x_2 = -\frac{2}{3}x_1 - \frac{1}{3}$$

Given a new point: $\mathbf{x}^* = [x_1^*, x_2^*]^\top$

$$1 + 2x_1^* + 3x_2^* = \begin{cases} \geq 0, & \mathbf{x}^* \in \mathcal{C}_1 \\ \leq 0, & \mathbf{x}^* \in \mathcal{C}_2 \\ = 0, & \mathbf{x}^* \text{ indéterminé} \end{cases}$$



The perceptron: elementary algebra

- In a space of dimension d

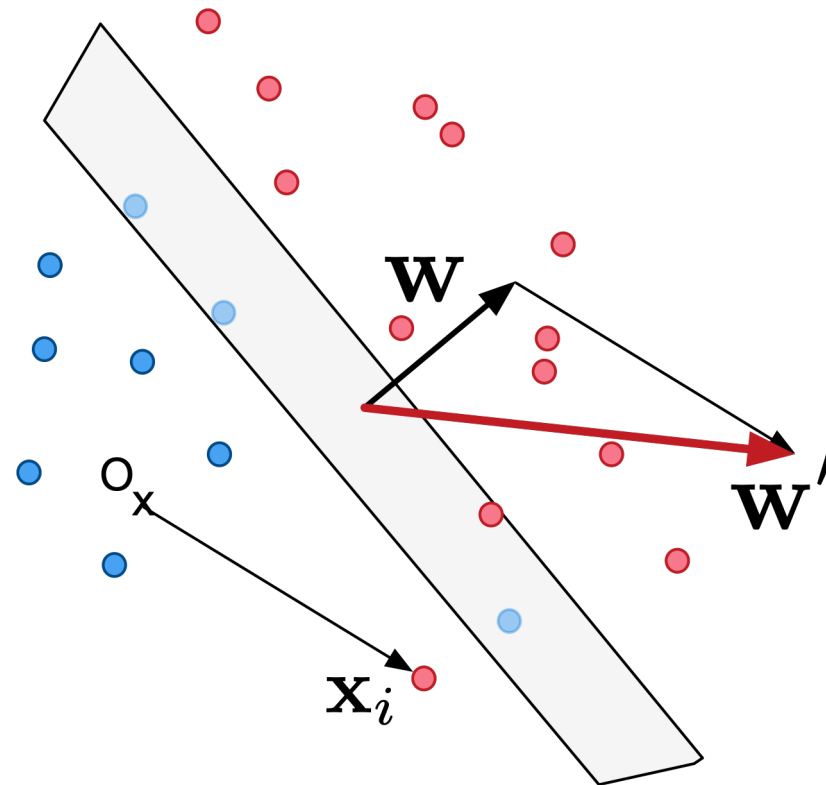
- Equation of an *hyperplan* (of dimension $d-1$)

$$\sum_{j=1}^d w_j x_j + w_0 = 0$$

- Expression of an **hypothesis** (perceptron) :

$$h(\mathbf{x}) = \sum_{j=1}^d w_j x_j + w_0$$

The perceptron learning algorithm: intuition



$$\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$$

Le perceptron

- Apprentissage des poids w_i

Historiquement, une règle ad hoc

Règle du perceptron : apprendre seulement en cas d'échec

Algorithme 1 : Algorithme d'apprentissage du perceptron

```
tant que non convergence faire
|   si la forme d'entrée est correctement classée alors
|   |   ne rien faire
|   sinon
|   |   
$$\mathbf{w}(t + 1) = \mathbf{w}(t) + \eta \mathbf{x}_i y_i$$

|   fin
|   Passer à la forme d'apprentissage suivante
fin
```

The perceptron learning algorithm: intuition

1. If $y_i = +1$ and $w^T \cdot x_i < 0$ increase $w^T \cdot x_i$

(minimize the risk of having $y_i \cdot (w^T x_i) < 0$ (wrong decision))

2. If $y_i = -1$ and $w^T \cdot x_i > 0$ decrease $w^T \cdot x_i$

(minimize the risk of having $y_i \cdot (w^T x_i) < 0$ (wrong decision))

For (1), we want $w' \cdot x_i > w \cdot x_i$

With $w' = w + \eta x_i$, we have $w' \cdot x_i = w \cdot x_i + \eta x_i x_i > w \cdot x_i$

For (2), we want $w' \cdot x_i < w \cdot x_i$

With $w' = w - \eta x_i$, we have $w' \cdot x_i = w \cdot x_i - \eta x_i x_i < w \cdot x_i$



$$\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$$

The perceptron learning algorithm

Algorithm 7.1: *Perceptron*(D, η) – train a perceptron for linear classification.

Input : labelled training data D in homogeneous coordinates;
learning rate η .

Output : weight vector $\mathbf{w}$ defining classifier $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$.

```
1  $\mathbf{w} \leftarrow \mathbf{0}$ ; // Other initialisations of the weight vector are possible
2 converged  $\leftarrow$  false;
3 while converged = false do
4   converged  $\leftarrow$  true;
5   for  $i = 1$  to  $|D|$  do
6     if  $y_i \mathbf{w} \cdot \mathbf{x}_i \leq 0$  // i.e.,  $\hat{y}_i \neq y_i$ 
7       then
8          $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$ ;
9         converged  $\leftarrow$  false; // We changed  $\mathbf{w}$  so haven't converged yet
10      end
11   end
12 end
```



Apprentissage des poids des connexions

Algorithme par descente du **gradient** minimisant un **risque empirique**

- On veut minimiser :

$$R_{Emp}(\mathbf{w}) = - \sum_{\mathbf{x}_j \in \mathcal{M}} \mathbf{w}^T \mathbf{x}_j \cdot u_j$$

Car nous voulons pour tous les exemples :

$$\mathbf{w}^T \mathbf{x} \begin{cases} \geq 0 \\ < 0 \end{cases} \Rightarrow \mathbf{x} \in \begin{cases} \omega_1 \\ \omega_2 \end{cases}$$

Le perceptron : **algorithme de gradient**

- Méthode d'exploration de $\mathcal{H}$
 - Recherche par gradient
 - Minimisation de la fonction d'erreur

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \nabla_{\mathbf{w}} E(\mathbf{w}(t))$$

$$\eta \ x_i \ u_i$$

- Algorithme historique :

si la forme est correctement classée : ne rien faire

sinon : $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \ x_i \ u_i$

boucler sur les formes d'apprentissage jusqu'à critère d'arrêt

Des propriétés remarquables !!

- **Convergence** en un **nombre fini d'étapes**
 - Indépendamment du **nombre** d'exemples
 - Indépendamment de la **distribution** des exemples
 - Indépendamment de la **dimension** de l'espace d'entrée

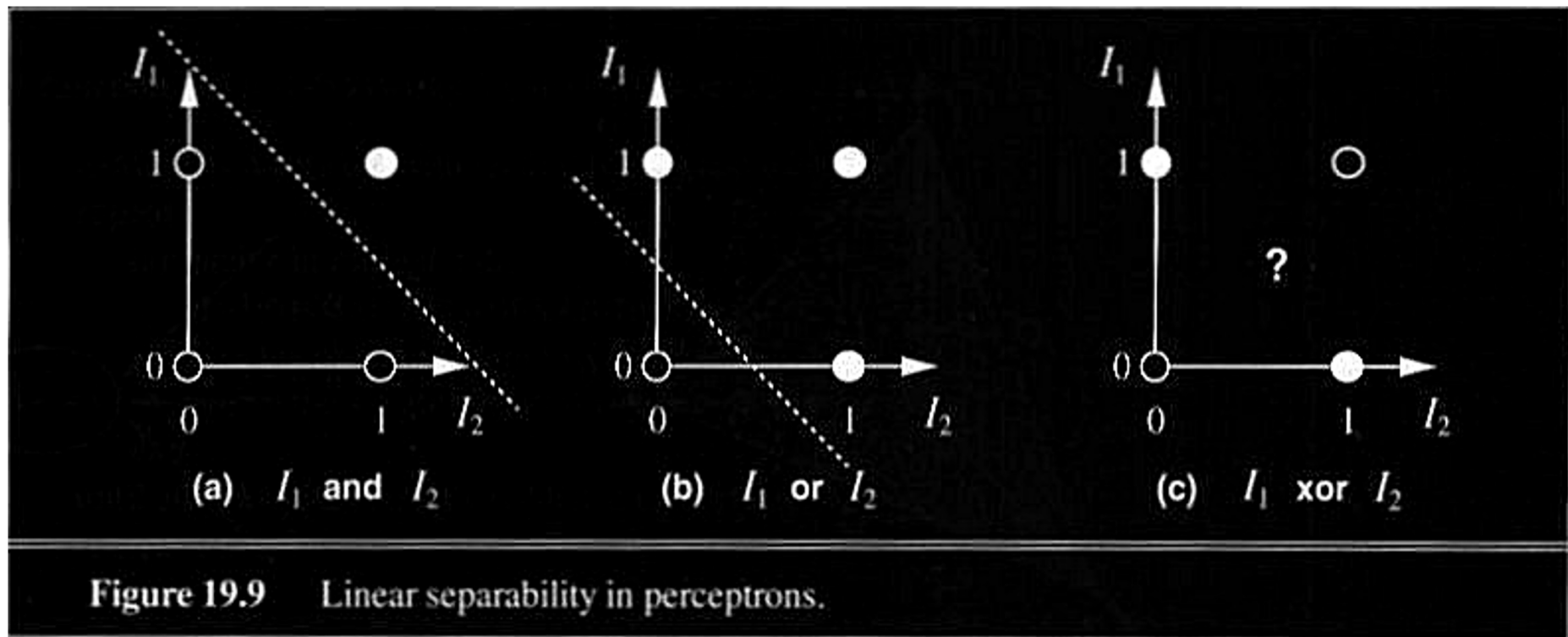


Si il existe au moins *une séparatrice linéaire des exemples*

Garantie de généralisation ??

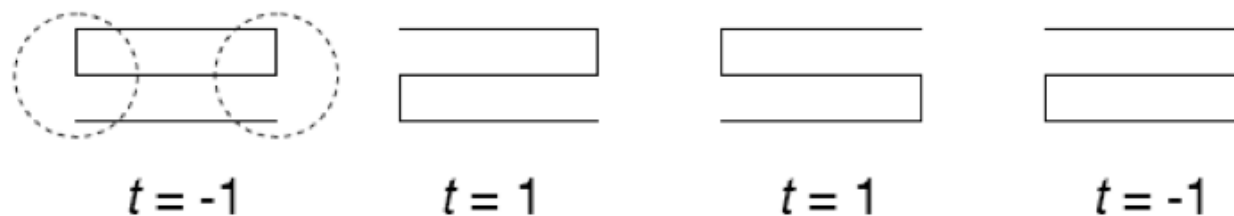
- Théorèmes sur la performance
par rapport à l'échantillon d'apprentissage
- Mais qu'en est-il pour des **exemples à venir** ?

Capacité expressive : Séparations linéaires



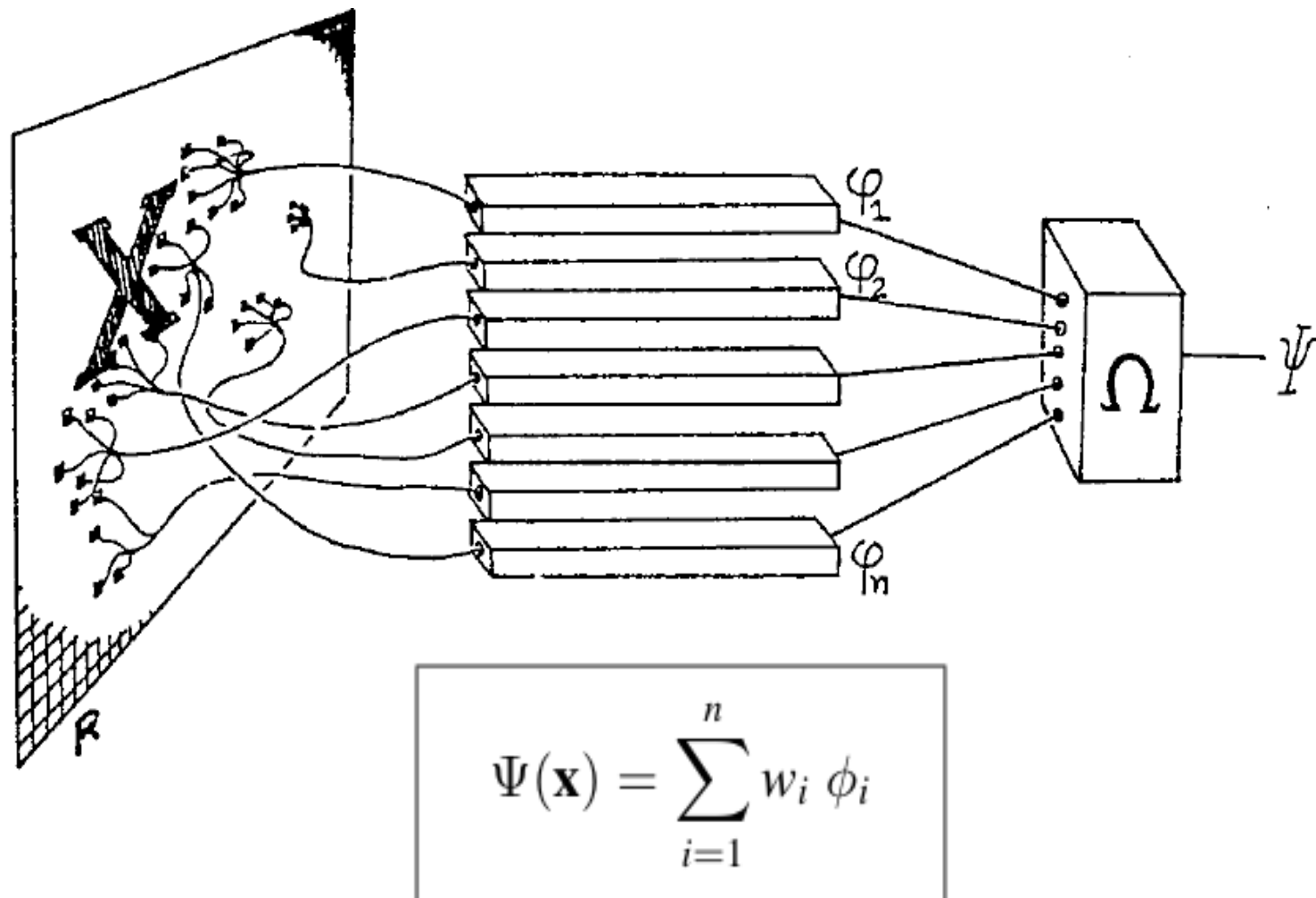
Limites du Perceptron

- Le Perceptron ne peut apprendre **que des séparatrices linéaires** (e.g. pas le XOR)
- Pas tout à fait vrai si on change d'espace d'entrée [Minsky et Papert, 1969]
- Mais le prétraitement est difficile
- Exemple (Bishop)



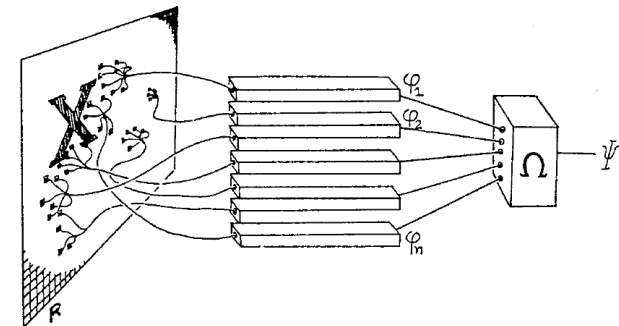
Le Perceptron

- Rosenblatt (1958-1962)



Premier connexionnisme : le perceptron

- *Propriétés*
 - Algorithme **en-ligne**
 - **Ne pouvait pas tout apprendre !?**
 - Car ne peut pas tout représenter
 - Il faut avoir de bonnes fonctions de base (détecteurs locaux)
 - Il faut savoir les combiner
 - *Ok sur la couche de sortie*



→ **Blocage**

Historique (très limité)

- **Mise en évidence des neurones**

- Camillio Golgi et Ramon Y Cajal (~ 1870) ; les synapses (dans les années 1930)

- **Modélisation formelle**

- Mc Culloch & Pitt (1943) : 1^{er} modèle du neurone formel
- Hebb (1949) : règle d'apprentissage par renforcement de couplage synaptique

- **Premiers modèles informatiques**

- ADALINE (Widrow-Hoff, 1960)
- Perceptron (Rosenblatt, 1958-1962)
- Analyse par Minsky et Papert (1969)

- **Nouveau connexionnisme**

- Hopfield (1982) : réseau bouclé
- Perceptron Multi-Couche (1985)
- « Reservoir computing » (2003)
- Les réseaux de neurones profonds
- « Spiking Neurons » networks

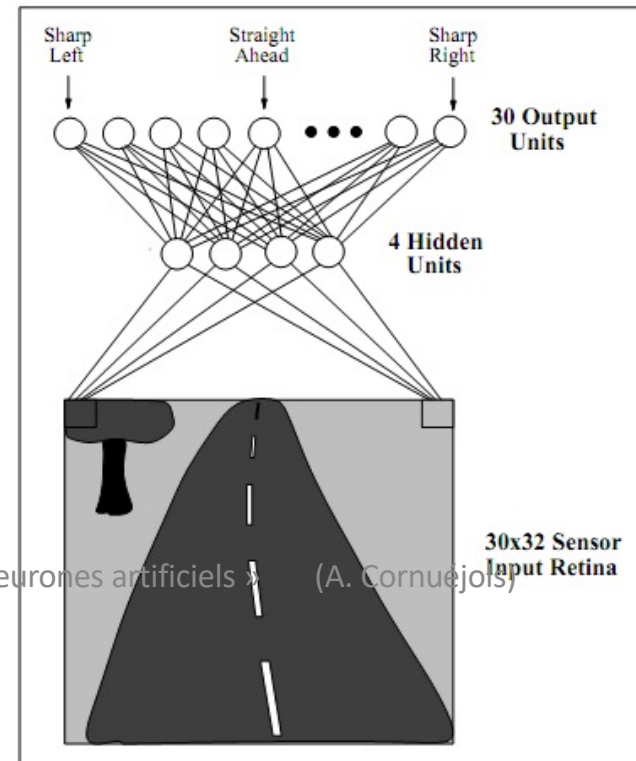
Plan

1. Introduction et réseaux de neurones à une couche
2. Les perceptrons multi-couches
3. L'apprentissage dans les réseaux de neurones
4. Une introduction aux réseaux de neurones profonds
5. Autres architectures

Les perceptrons multi-couches

Neural network application

- ALVINN drives 70 mph on highways

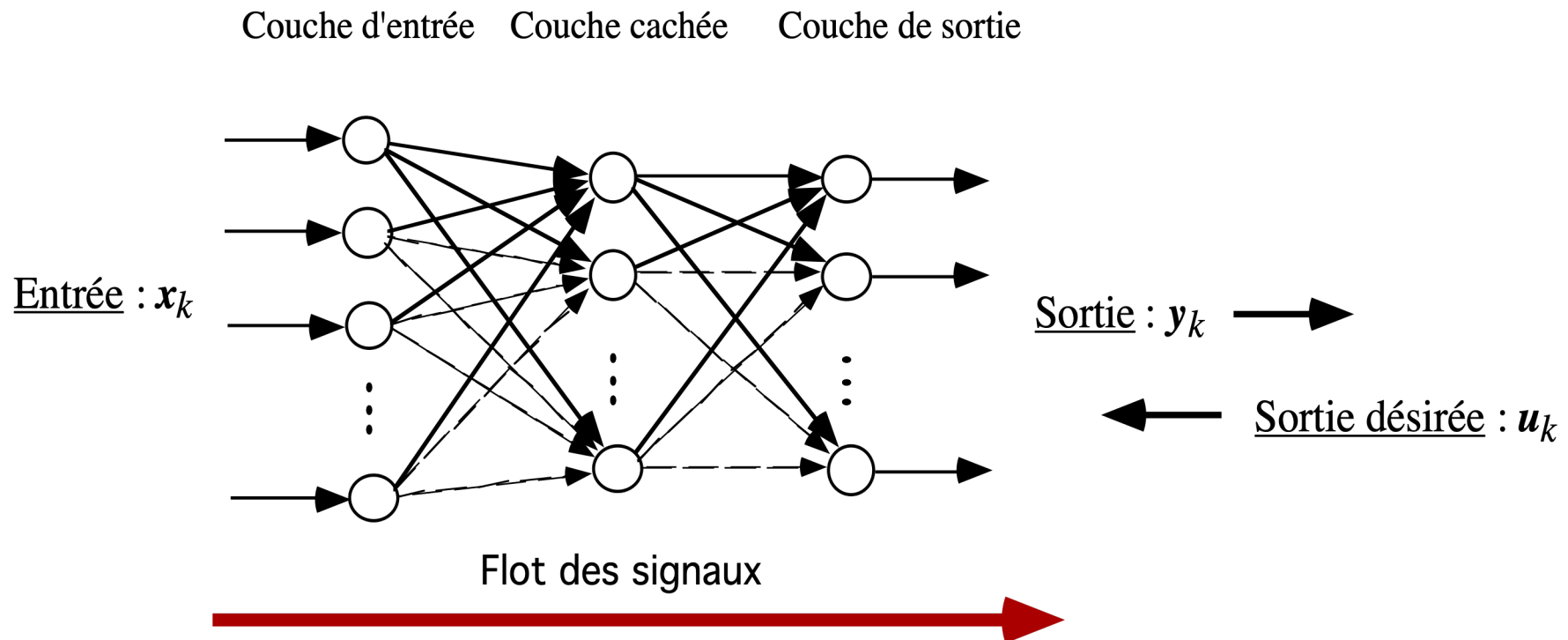


« Les réseaux de neurones artificiels » (A. Cornuéjols)

Composants et structure du PMC

Le perceptron multi-couche

- Topologie typique



Le Perceptron Multi-Couches : propagation

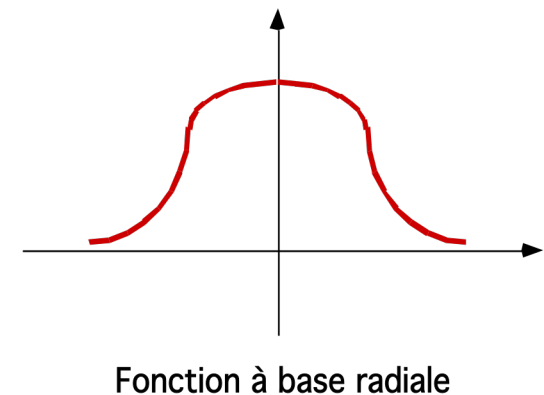
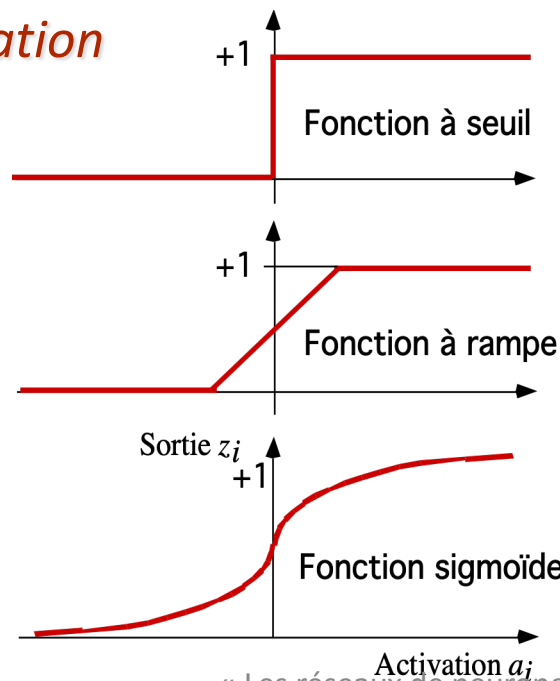
- Pour chaque neurone :

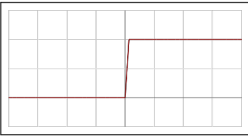
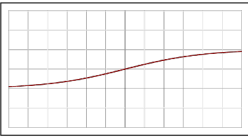
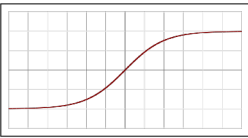
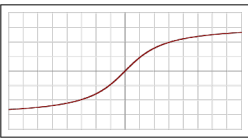
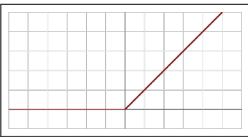
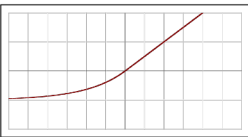
$$y_l = g\left(\sum_{j=0,d} w_{jk} \phi_j\right) = g(a_k)$$

- w_{jk} : *poids* de la connexion de la cellule j à la cellule k
- a_k : *activation* de la cellule k
- g : *fonction d'activation*

$$g(a) = \frac{1}{1 + e^{-a}}$$

$$g'(a) = g(a)(1-g(a))$$



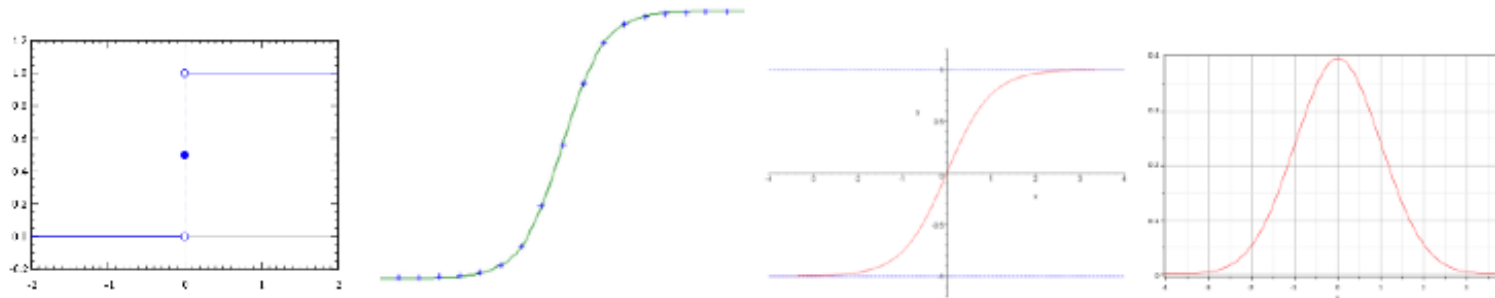
Nom	Graphe	Équation	Dérivée
Rampe		$f(x) = x$	$f'(x) = 1$
Heaviside		$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{si } x \neq 0 \\ ? & \text{si } x = 0 \end{cases}$
Logistique ou sigmoïde		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tangente hyperbolique		$f(x) = \tanh(x)$ $= \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f^2(x)$
Arc Tangente		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Unité ReLU		$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{si } x \neq 0 \\ 1 & \text{si } x = 0 \end{cases}$
Unité Exponentielle Linéaire		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$

Fonctions d'activation

Différentes fonctions d'activation

Elles introduisent un intervalle sur lequel le neurone est activé

- fonction identité
- heaviside : $\varphi(x) = 0$ si $x < 0$, 1 sinon
- sigmoïde : $\varphi(x) = \frac{1}{1+e^x}$
- tanh : $\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$
- fonction noyau (gaussienne)



heaviside - sigmoïde - tanh - gaussienne

Fonctions d'activation de la couche de sortie

- Tâche de **classification**

- Couches **cachées** : sigmoïde [0,1], tanh [-1,1], ReLU [0, ∞], ...
- Couche de **sortie** : **sigmoïde** dans [0,1] ; **softmax** dans [0,1]

$$\hat{y}_j = \frac{e^{-z_j}}{\sum_{i=1}^m e^{-z_i}}$$

- Tâche de **régression**

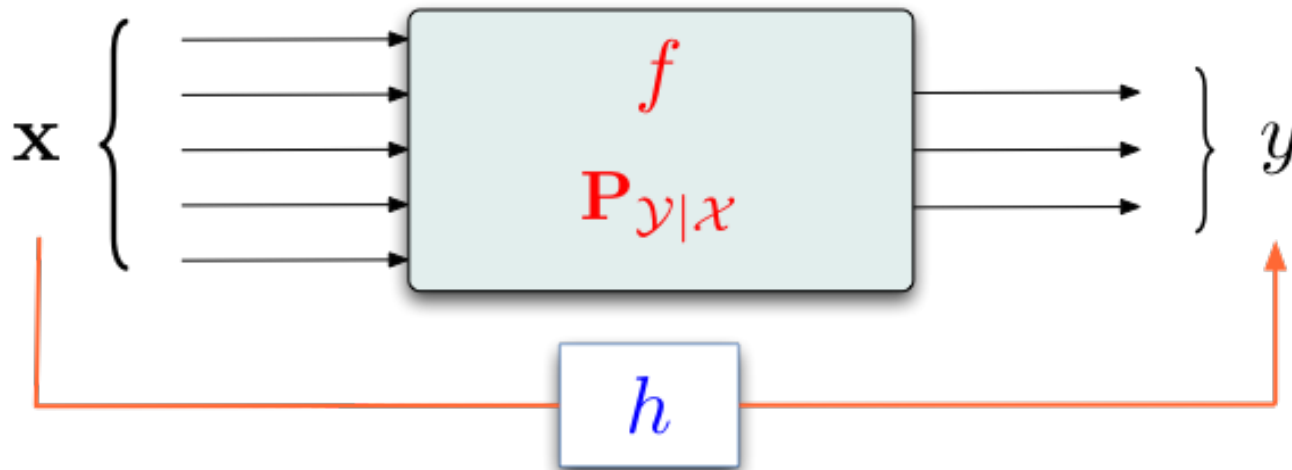
- Couches **cachées** : sigmoïde [0,1], tanh [-1,1], ReLU [0, ∞], ...
- Couche de **sortie** : **linéaire**

Plan

1. Introduction et réseaux de neurones à une couche
2. Les perceptrons multi-couches
3. L'apprentissage dans les réseaux de neurones
4. Une introduction aux réseaux de neurones profonds
5. Autres architectures

L'apprentissage dans les PMC

Apprentissage supervisé



À partir :

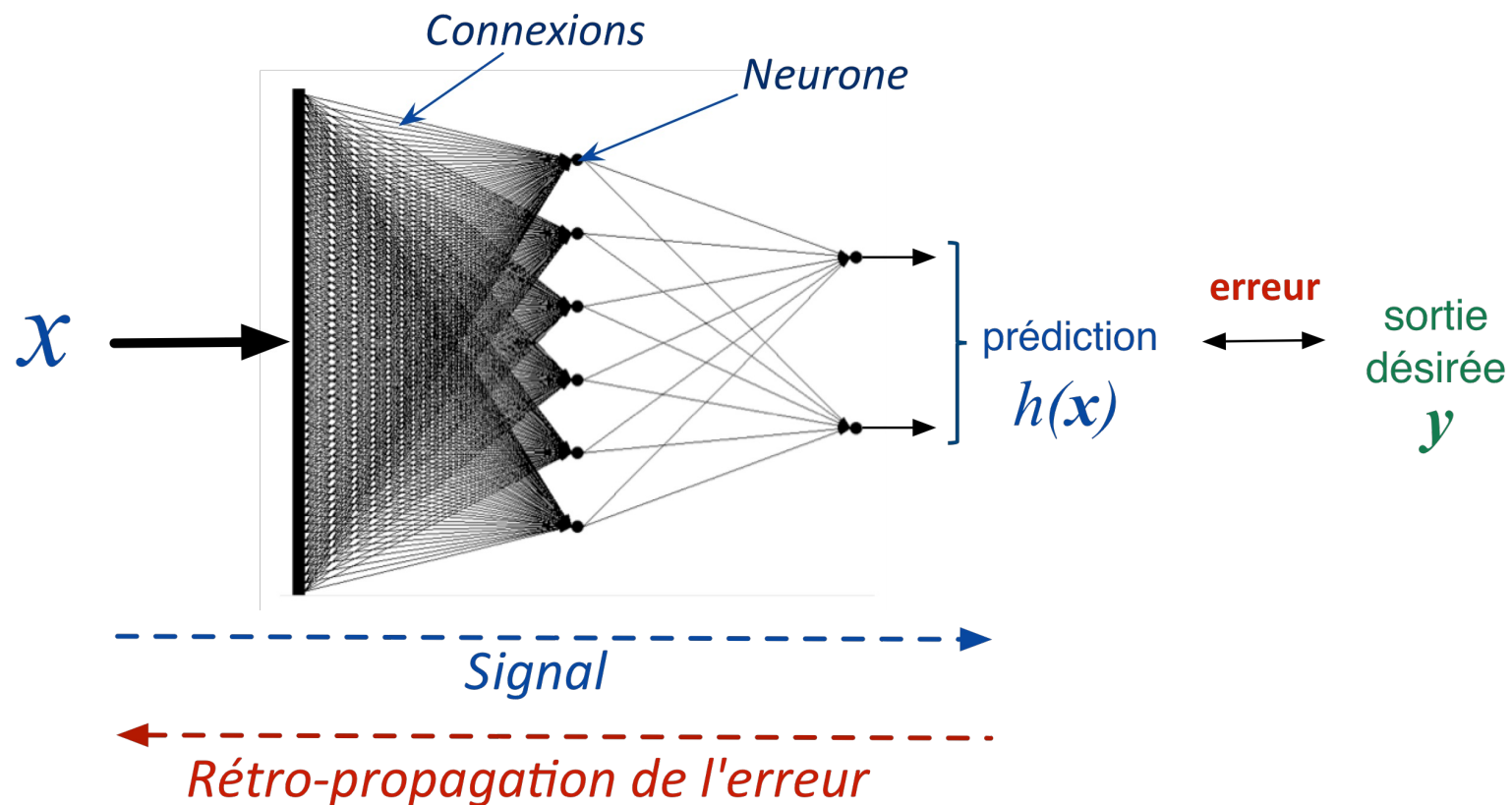
- d'un échantillon d'apprentissage $S_m = \langle (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \rangle$
- de connaissances préalables sur le type de dépendances sur $\mathcal{X} \times \mathcal{Y}$

Trouver :

- une fonction h
- permettant la prédiction de y pour une nouvelle entrée $\mathbf{x}$ $h(\mathbf{x}) \approx y \quad (= f(\mathbf{x}))$

Learning with Multi-Layer Perceptrons

- Questions:
 - How to learn the **parameters** (weights of the connections) ?
 - How to set the **architecture** of the network?



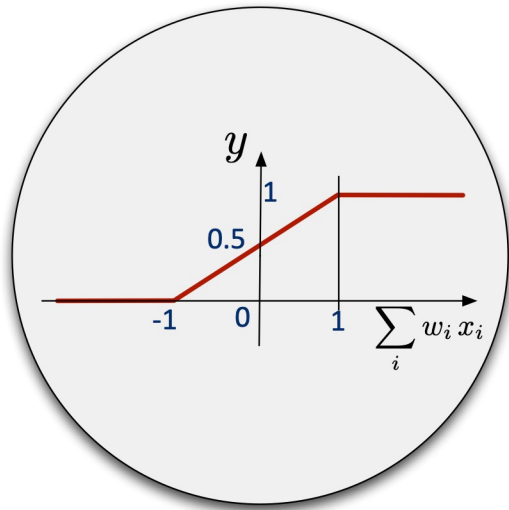
Analyse

Rôle des couches cachées

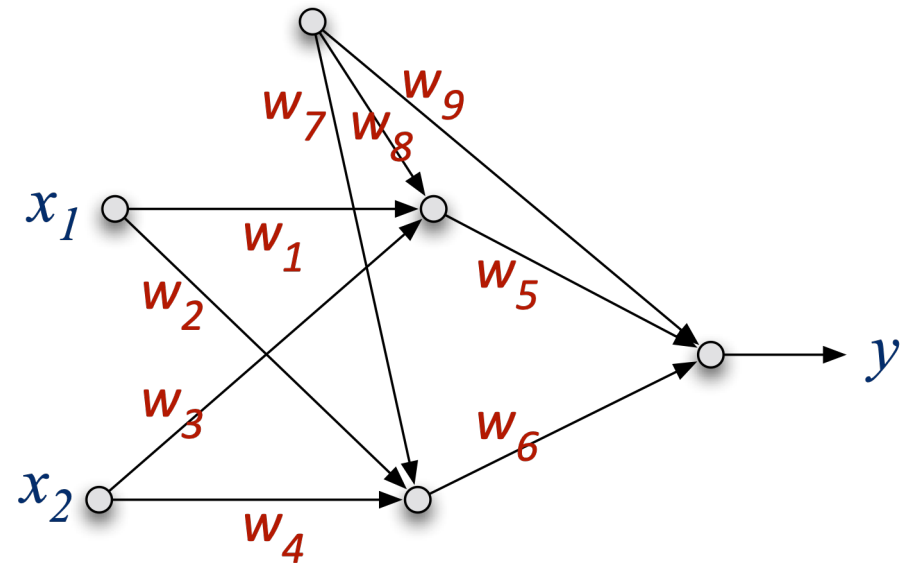
Calcul de poids

Calcul de la fonction XOR

Entrée x_1	Entrée x_2	Sortie y
0	0	0
0	1	1
1	0	1
1	1	0

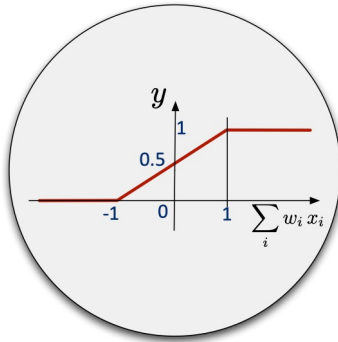


Saurez-vous trouver les poids w_i ?



Compute the weights such that ...

Computation of the XOR function



Entrée x_1	Entrée x_2	Sortie y
0	0	0
0	1	1
1	0	1
1	1	0

$W_1 =$

$W_2 =$

$W_3 =$

$W_4 =$

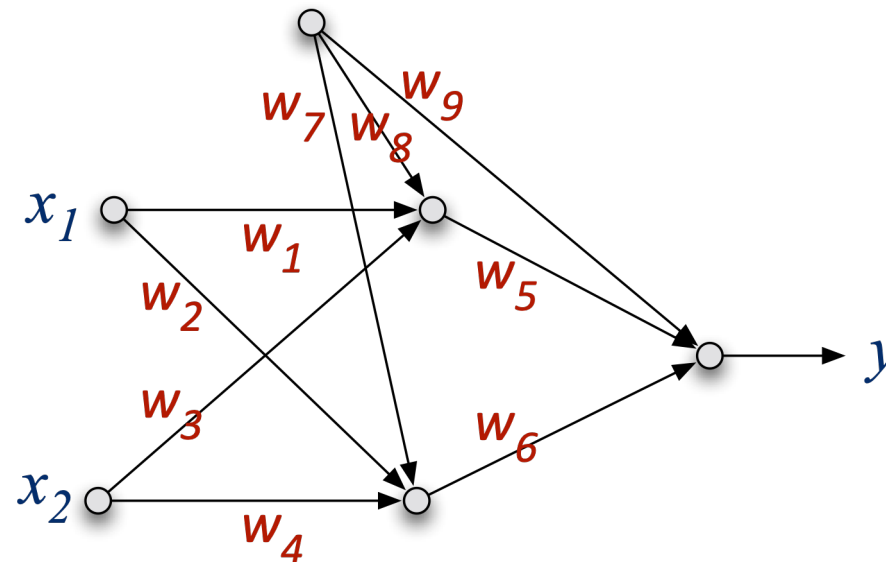
$W_5 =$

$W_6 =$

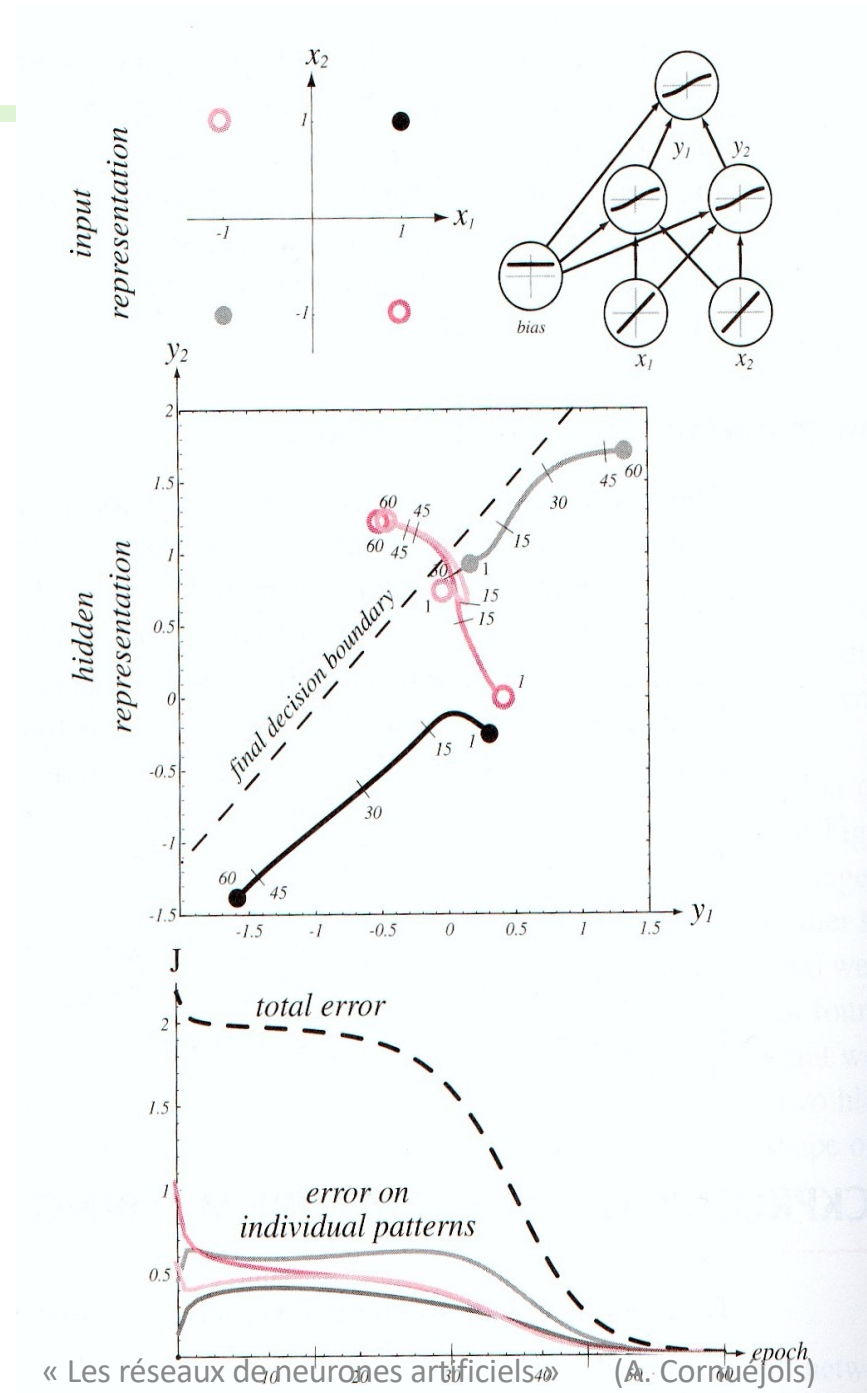
$W_7 =$

$W_8 =$

$W_9 =$

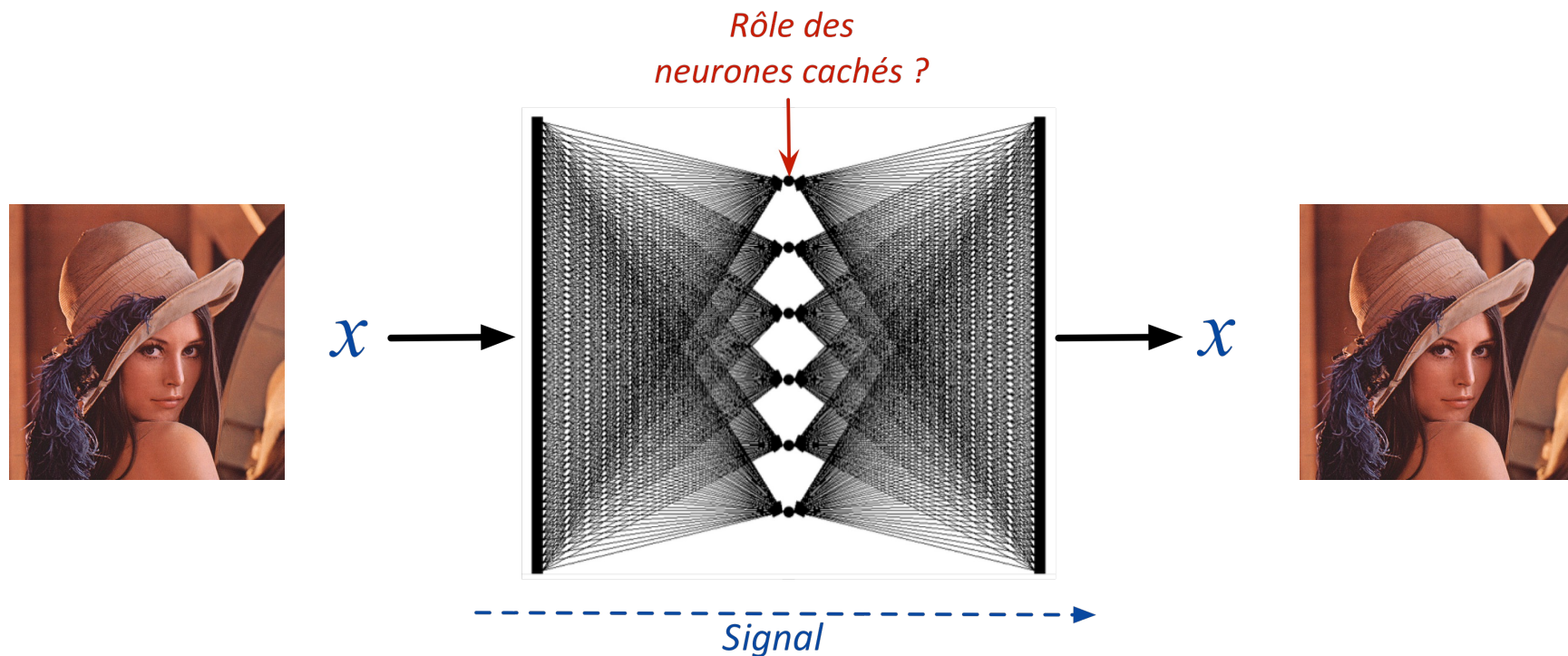


Rôle de la couche cachée

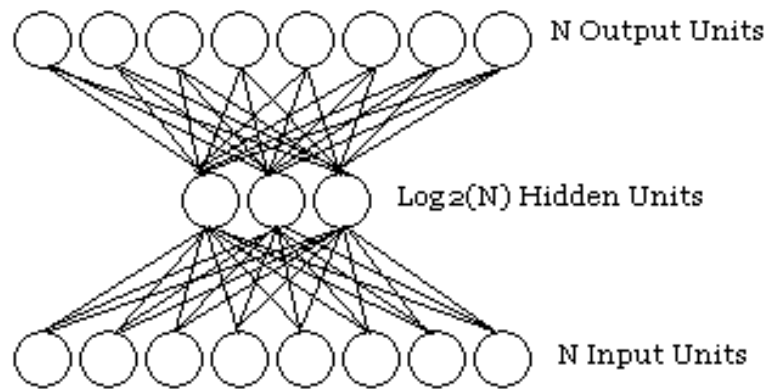


Change of representation: the role of hidden layer(s)

- Which new representation (latent variables)?
- How to choose the architecture?



Rôle de la couche cachée



Output: 00000001 00000010 00000100 ...

↑ ↑ ↑

Input: 00000001 00000010 00000100 ...

Figure 1. The encoding problem (Rumelhart, Hinton, & Williams, 1986)



(a)



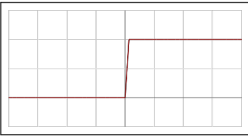
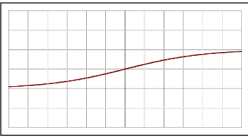
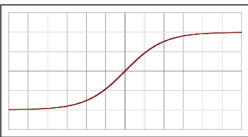
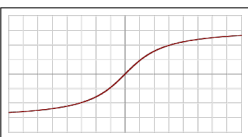
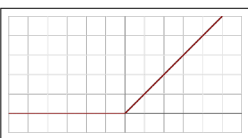
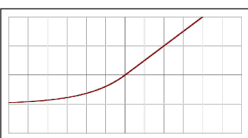
(b)



(c)



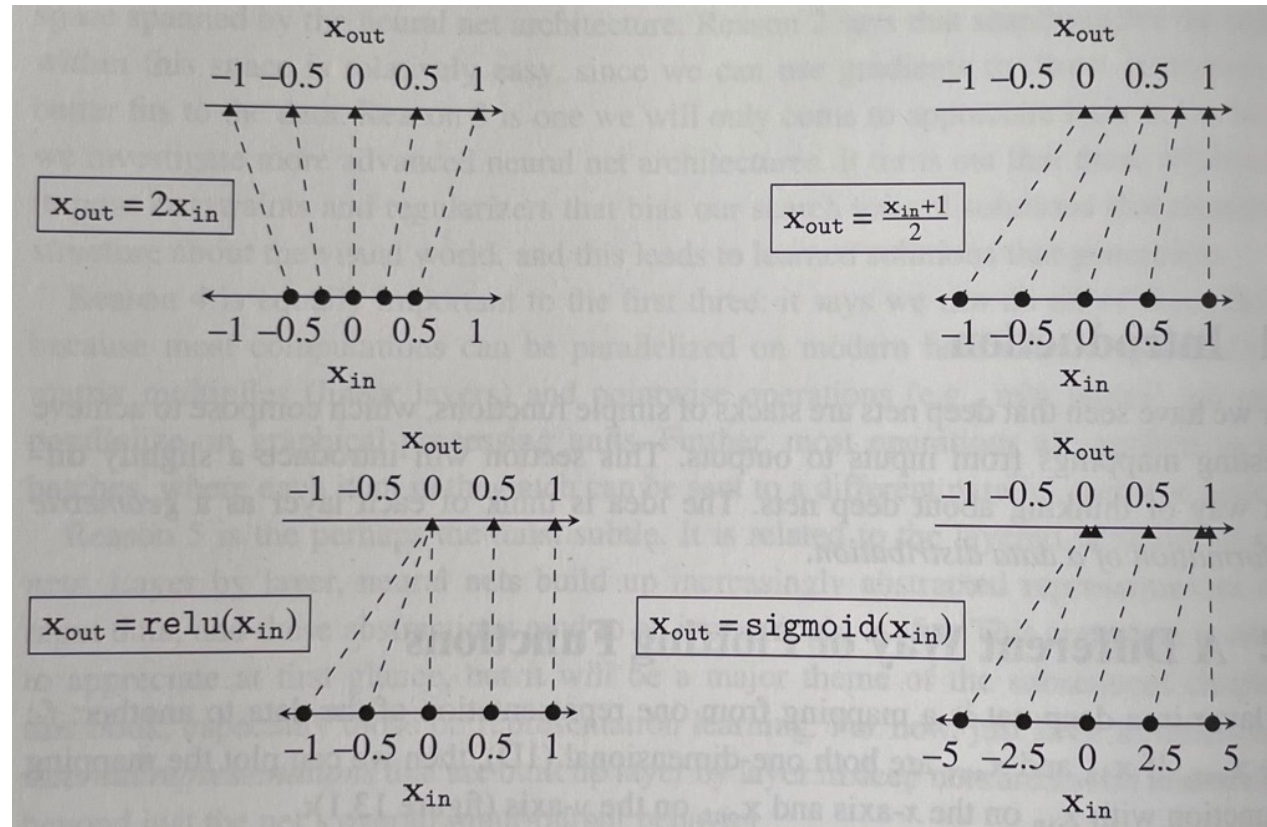
(d)

Nom	Graphe	Équation	Dérivée
Rampe		$f(x) = x$	$f'(x) = 1$
Heaviside		$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{si } x \neq 0 \\ ? & \text{si } x = 0 \end{cases}$
Logistique ou sigmoïde		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tangente hyperbolique		$f(x) = \tanh(x)$ $= \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f^2(x)$
Arc Tangente		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Unité ReLU		$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{si } x \neq 0 \\ 1 & \text{si } x = 0 \end{cases}$
Unité Exponentielle Linéaire		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$

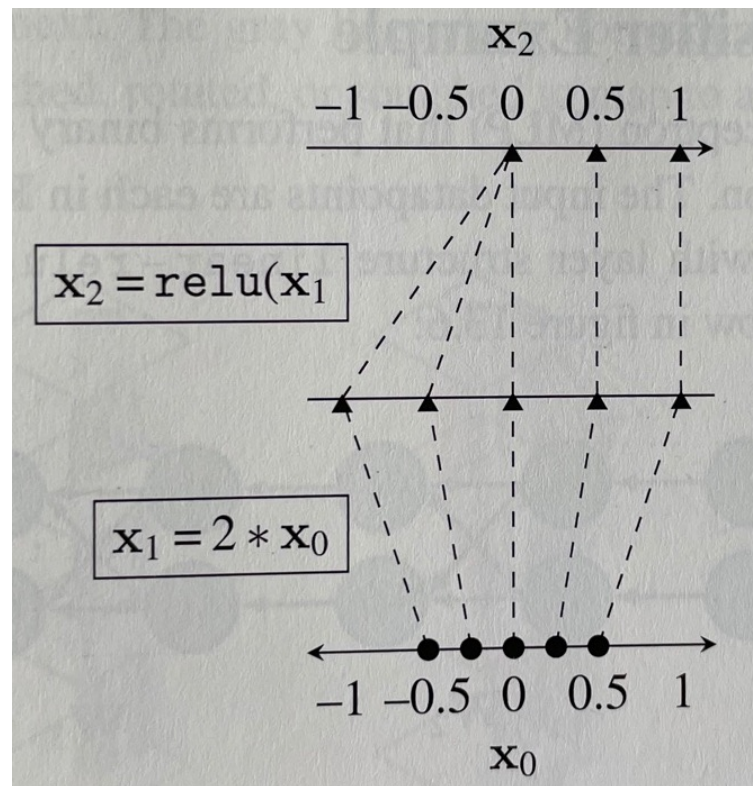
- Exemples de transformations par un neurone

Fonctions d'activation
linéaires

Fonctions d'activation
non linéaires

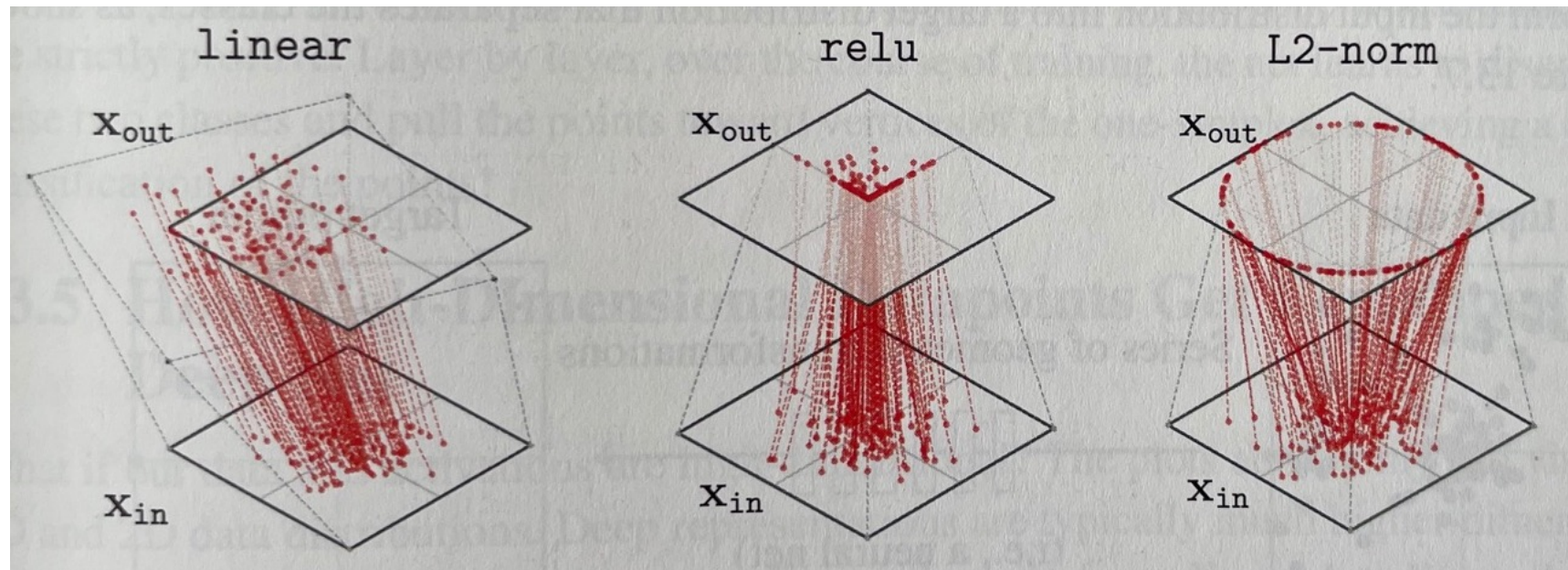


- Exemple de **séquence** de transformations (neurones)

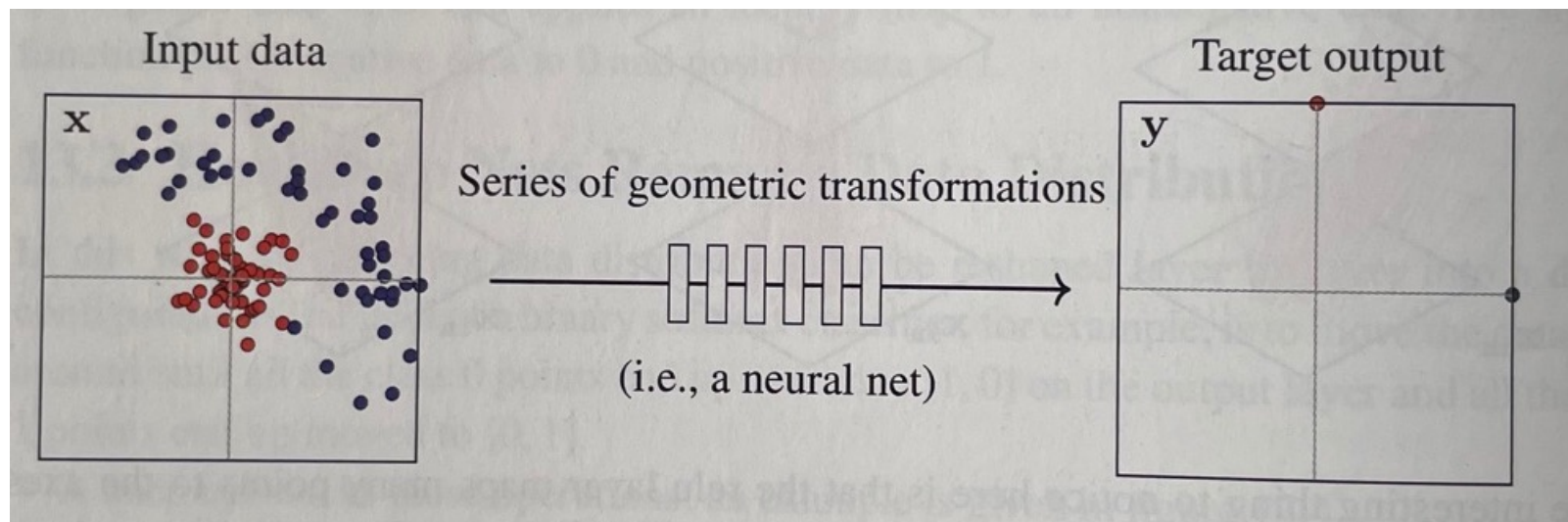


Exemples de transformations en 2D

Entrées de neurones en 2D et **sortie** en 2D

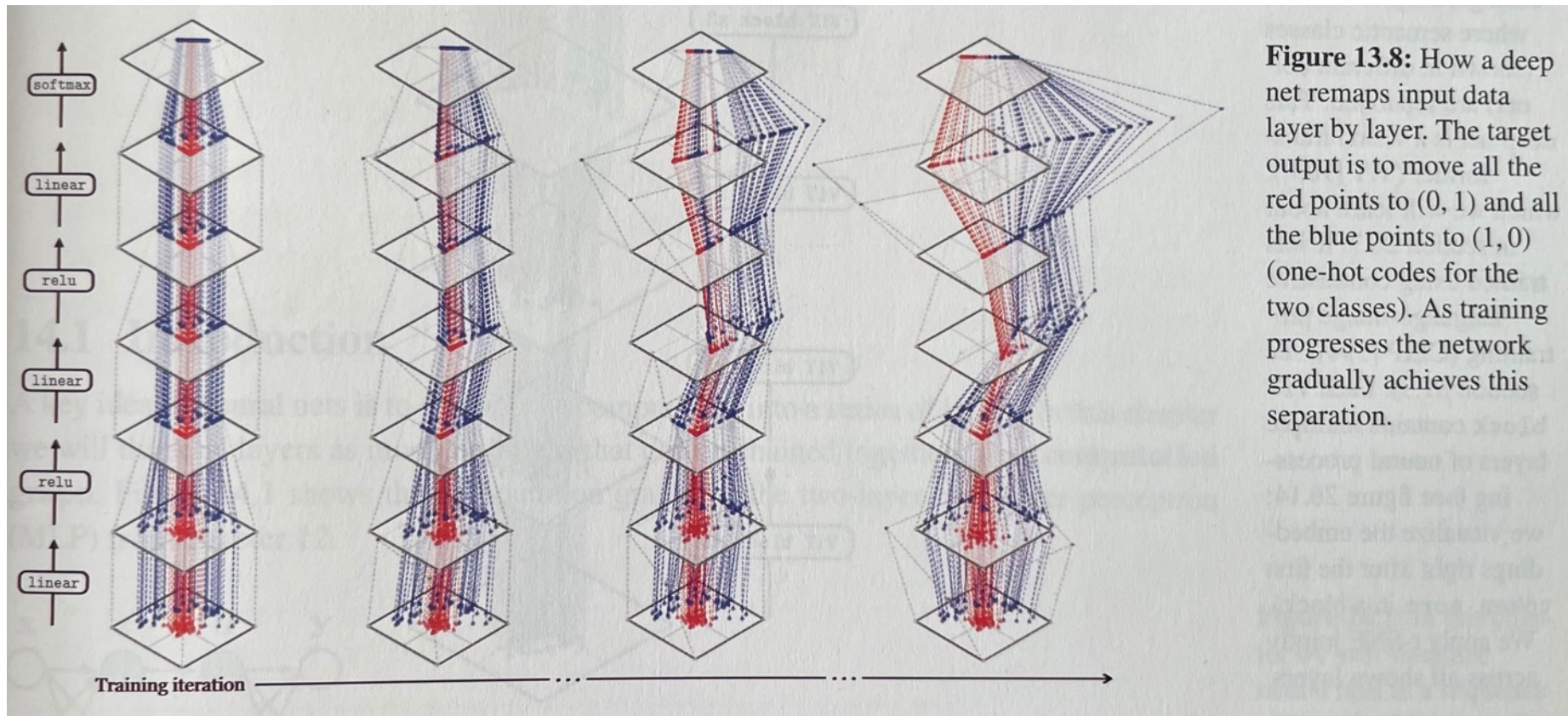


- Le but des transformations successives est de **rapprocher** la **distribution en sortie** du réseau de la **distribution cible**



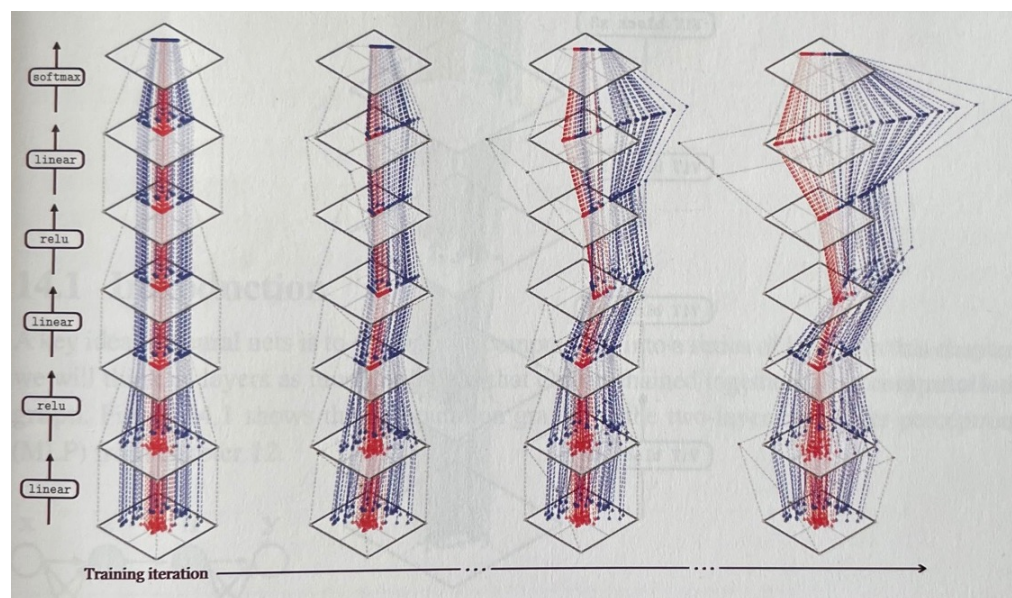
Ici, on veut séparer les deux classes et utiliser le « *one hot encoding* » :
classe « **bleue** » $\rightarrow$ (1,0) et classe « **rouge** » $\rightarrow$ (0,1)

- Illustration



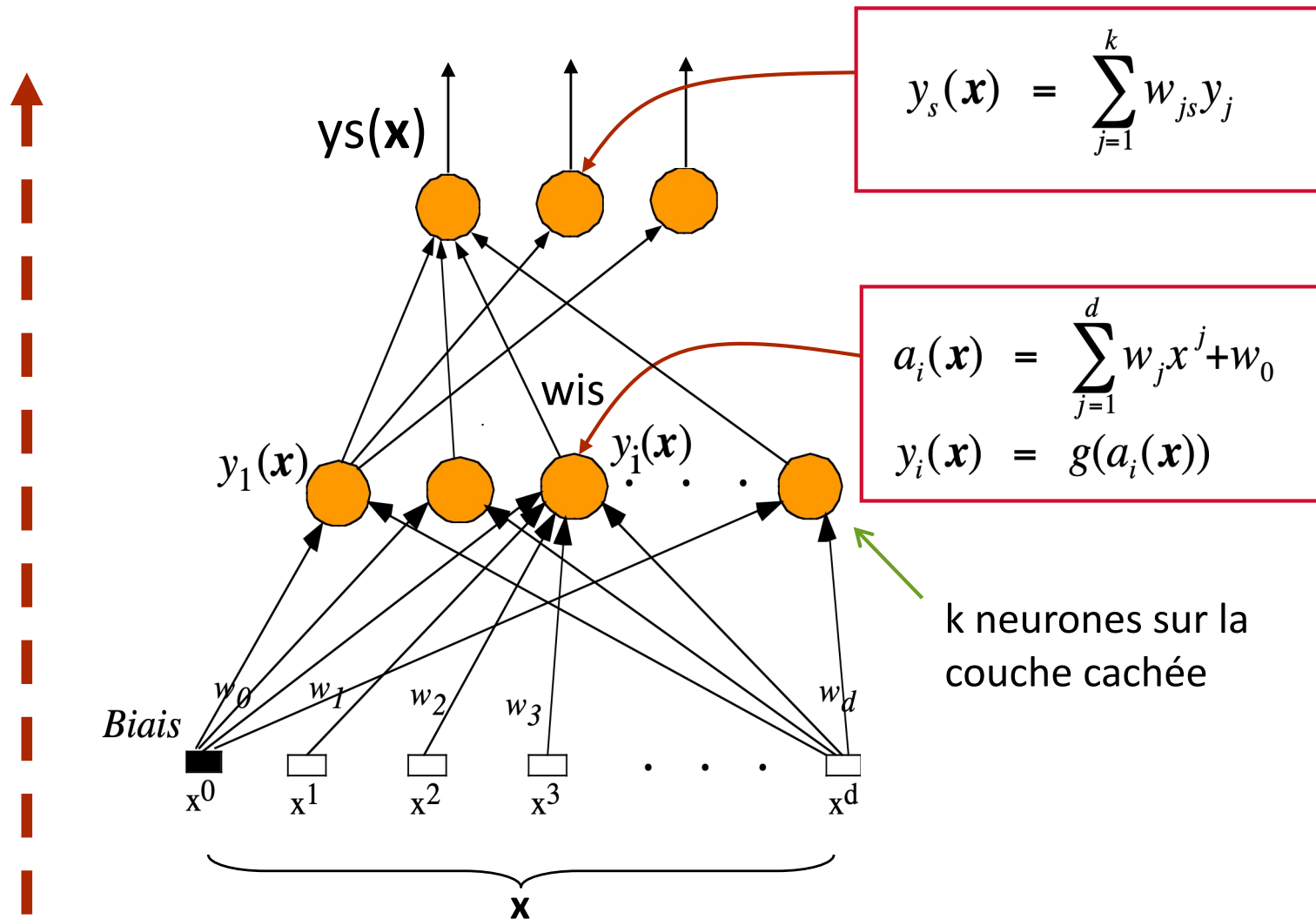
From [Antonio Torralba et al. (2024) « **Foundations of Computer Vision** »]

Mais comment **apprendre** les poids ?



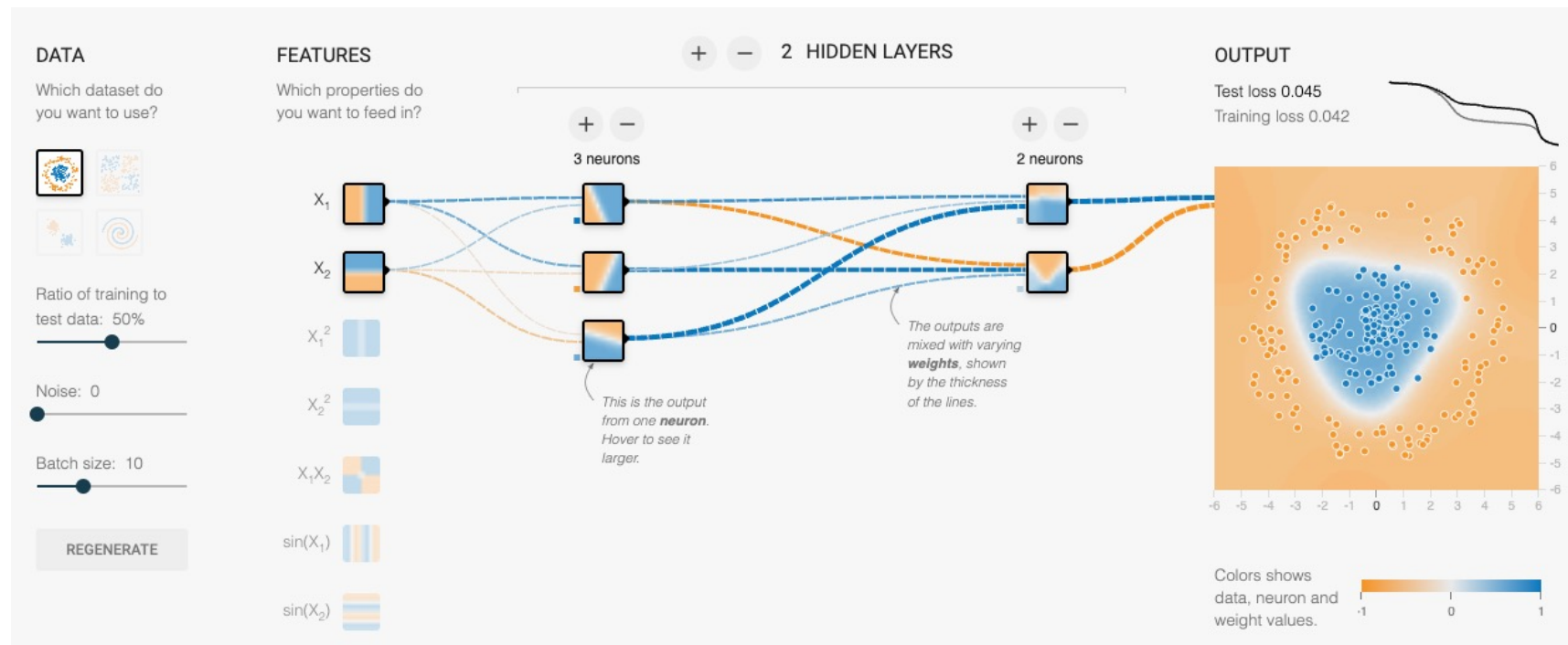
L'apprentissage de représentations **par** rétro-propagation de gradient

MLPs: feed forward (summary)



Démo

playground.tensorflow.org



Le Perceptron Multi-Couches : apprentissage

Objectif :

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{m} \sum_{l=1}^m \left[y(\mathbf{x}_l ; \mathbf{w}) - u(\mathbf{x}_l) \right]^2$$

- Algorithme (rétro-propagation de gradient) : descente de gradient

Algorithme itératif :

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \eta \nabla_E \mathbf{w}^{(t)}$$

Cas hors-ligne

(gradient total) :

$$w_{ij}(t) = w_{ij}(t-1) - \eta(t) \frac{1}{m} \sum_{k=1}^m \frac{\partial R_E(\mathbf{x}_k, \mathbf{w})}{\partial w_{ij}}$$

où :

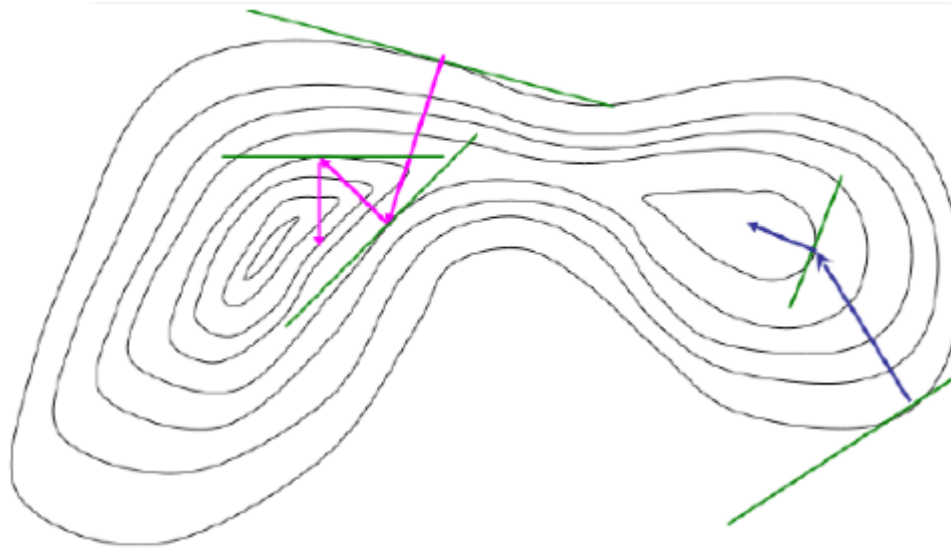
$$R_E(\mathbf{x}_k, \mathbf{w}) = [t_k - f(\mathbf{x}_k, \mathbf{w})]^2$$

Cas en-ligne

(gradient stochastique) :

$$w_{ij}(t) = w_{ij}(t-1) - \eta(t) \frac{\partial R_E(\mathbf{x}_k, \mathbf{w})}{\partial w_{ij}}$$

Descente de gradient



- Algorithme itératif :

On choisit un $\mathbf{W}_{t=0}$ aléatoire

Bonne direction = celle où le critère baisse

Avancer un peu, mais pas trop

$$\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \left. \frac{d\mathcal{J}(\mathbf{W})}{d\mathbf{W}} \right|_{\mathbf{W}_t}$$

avec :

$\mathbf{W}$ les paramètres ; η : le pas ; $\left. \frac{d\mathcal{J}(\mathbf{W})}{d\mathbf{W}} \right|_{\mathbf{W}_t}$: la « bonne » direction

Le Perceptron Multi-Couches : apprentissage

1. Présentation d'un exemple parmi l'ensemble d'apprentissage

Séquentielle, aléatoire, en fonction d'un critère donné

2. Calcul de l'état du réseau

3. Calcul de l'erreur = fct(sortie - sortie désirée) (e.g. $= (y^l - u^l)^2$)

4. Calcul des gradients

Par l'algorithme de rétro-propagation de gradient

5. Modification des poids synaptiques

6. Critère d'arrêt

Sur l'erreur. Nombre de présentation d'exemples, ...

7. Retour en 1

PMC : la **rétro-propagation de gradient**

1. **Evaluation** de l'erreur E^l (ou E) due à chaque connexion : $\frac{\partial E^l}{\partial w_{ij}}$

Idée : calculer l'erreur sur la connexion w_{ji} en fct de l'erreur après la cellule j

$$\frac{\partial E^l}{\partial w_{ij}} = \frac{\partial E^l}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = \delta_j z_i$$

Sortie du neurone i

- Pour les cellules de la couche de sortie :

$$\delta_k = \frac{\partial E^l}{\partial a_k} = \frac{\partial E^l}{\partial y_k} \frac{\partial y_k}{\partial a_k} = (y_k^l - \hat{y}_k^l) \cdot g'(a_k)$$

- Pour les cellules d'une couche cachée :

$$\delta_j = \frac{\partial E^l}{\partial a_j} = \sum_k \frac{\partial E^l}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \cdot \frac{\partial a_k}{\partial z_j} \cdot \frac{\partial z_j}{\partial a_j} = g'(a_j) \cdot \sum_k w_{jk} \cdot \delta_k$$

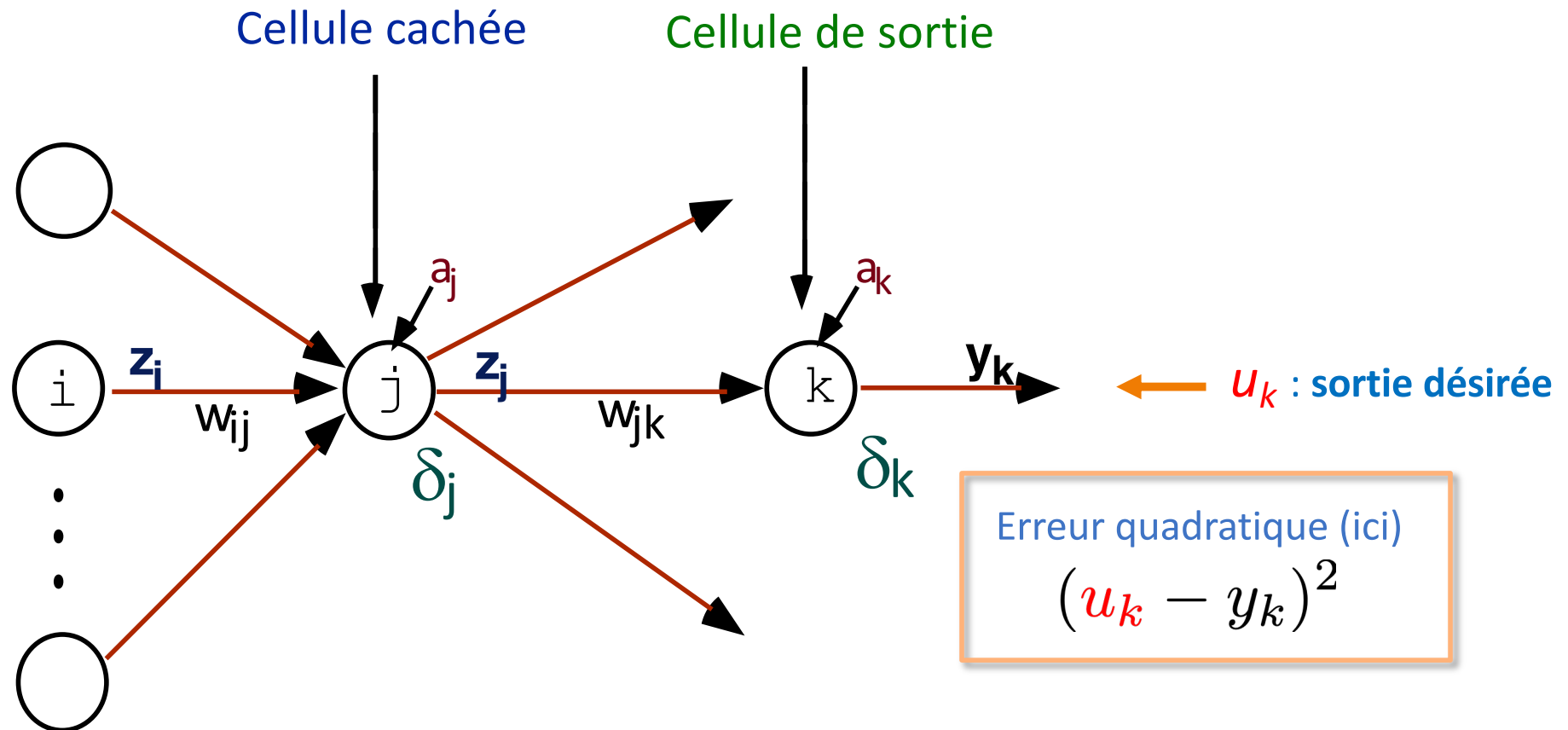
PMC : la rétro-propagation de gradient

a_i : activation de la cellule i

z_i : sortie de la cellule i

δ_i : erreur attachée à la cellule i

$$\delta_j = \frac{\partial E^l}{\partial a_j} = \sum_k \frac{\partial E^l}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \cdot \frac{\partial a_k}{\partial z_j} \cdot \frac{\partial z_j}{\partial a_j} = g'(a_j) \cdot \sum_k w_{jk} \cdot \delta_k$$



PMC : la rétro-propagation de gradient

- 2. Modification des poids

- On suppose gradient à pas (constant ou non) : $\eta(t)$
- Si apprentissage stochastique (après présentation de chaque exemple)

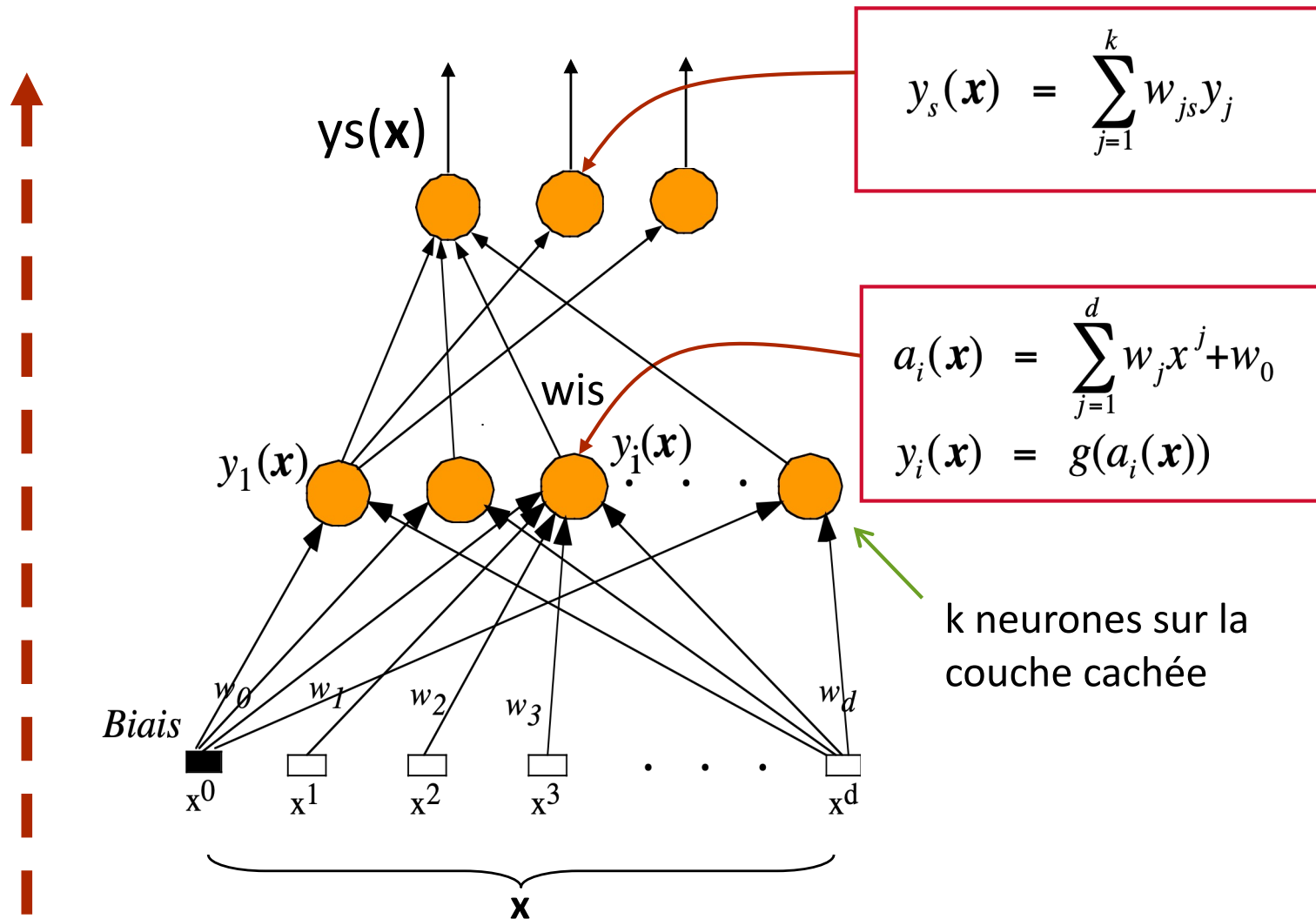
$$\Delta w_{ij} = \eta(t) \delta_j y_i$$

j : neurone de sortie

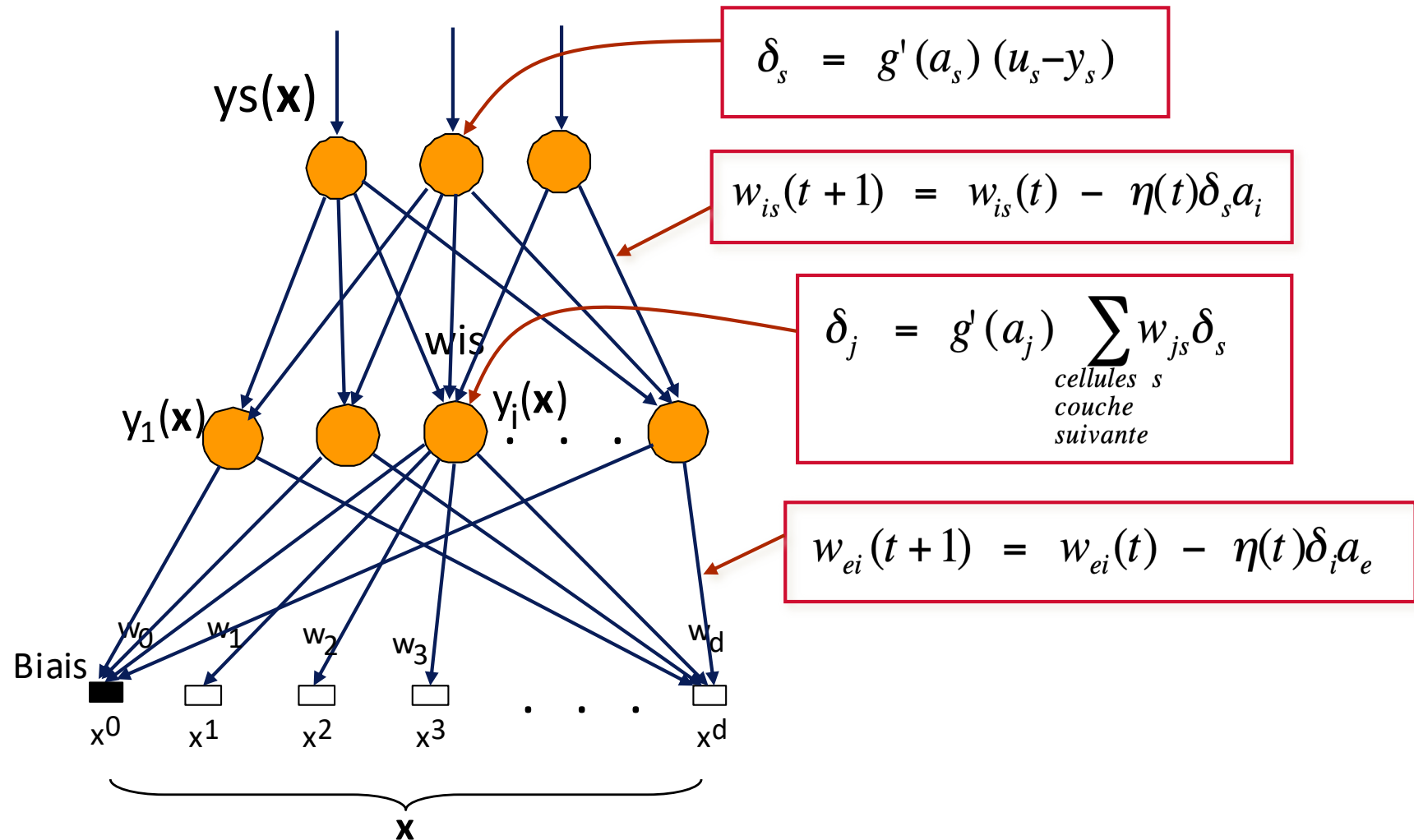
- Si apprentissage total (après présentation de l'ensemble des exemples)

$$\Delta w_{ij} = \eta(t) \sum_{n=1}^m \delta_j^n z_i^n$$

Le PMC : passes avant et arrière (résumé)

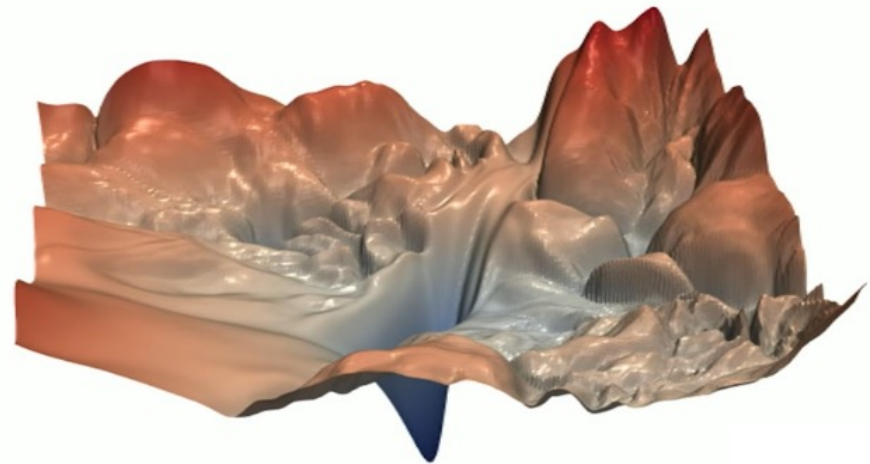
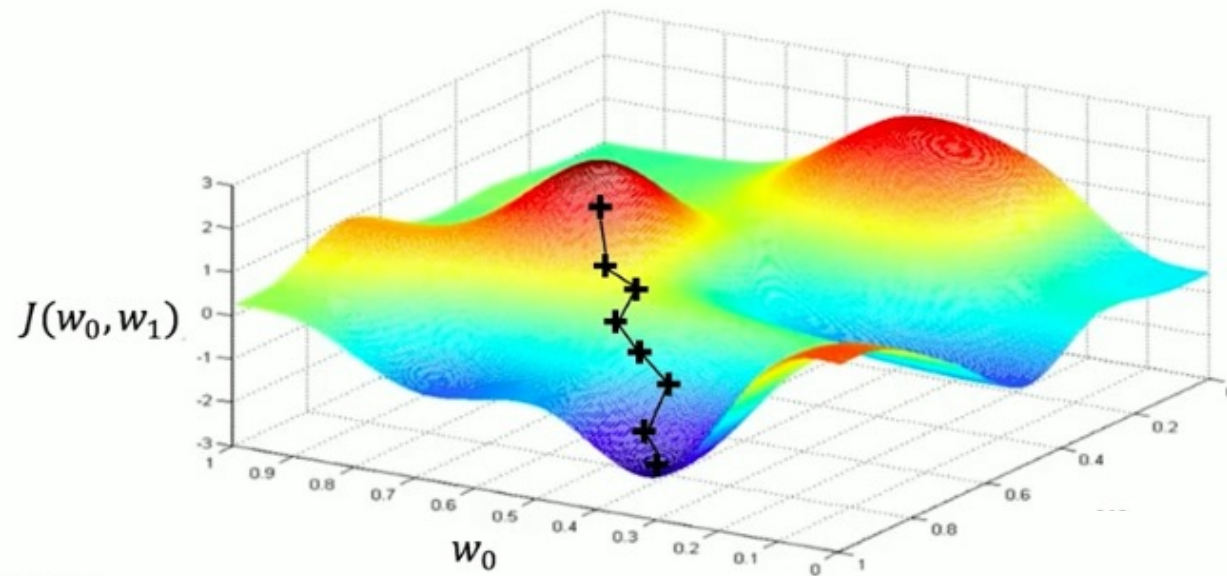


Le PMC : passes avant et arrière (résumé)



Optimisation de la fonction de coût

Repeat until convergence

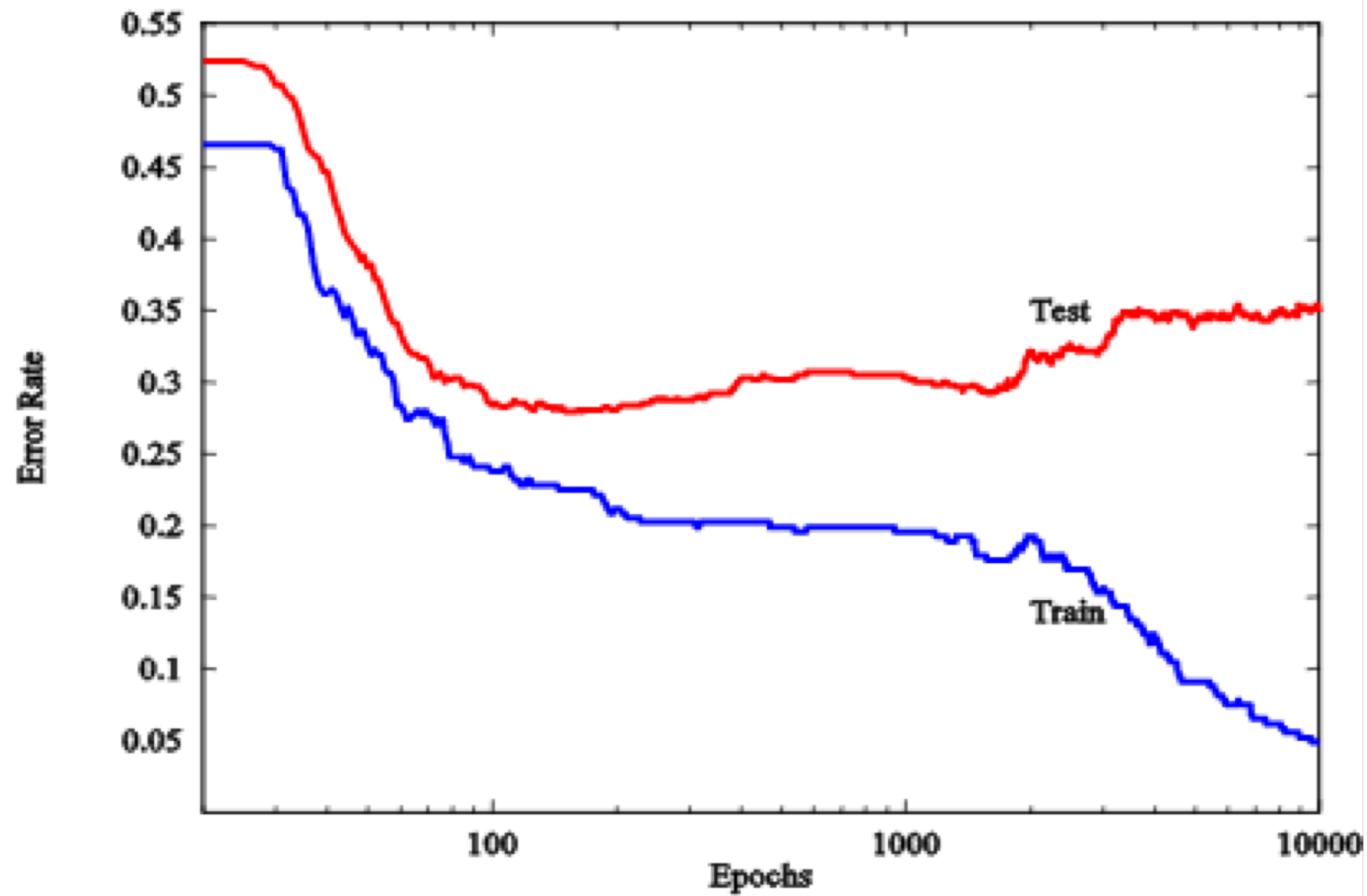


...

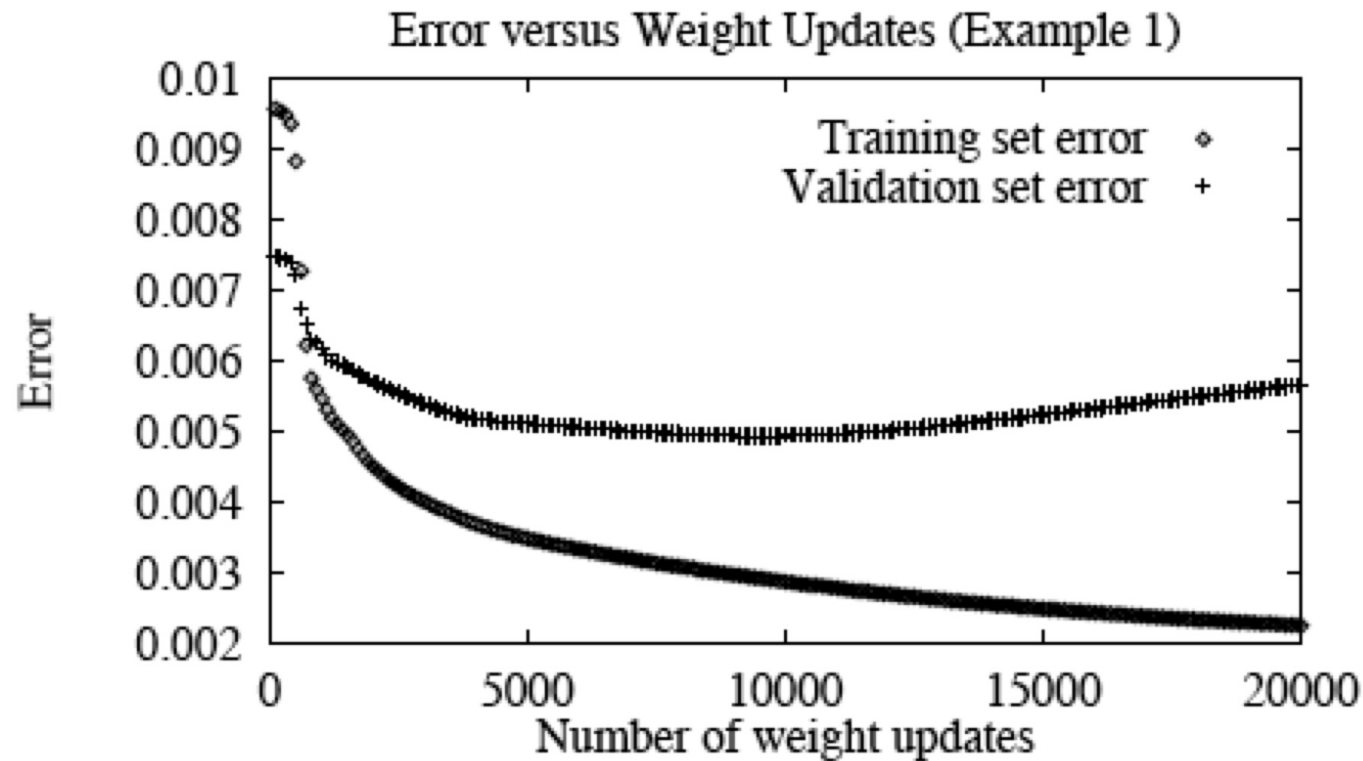
PMC : La rétro-propagation de gradient

- Efficacité en apprentissage
 - En $O(w)$ pour chaque passe d'apprentissage, w = nb de poids
 - Il faut typiquement plusieurs centaines de passes (voir plus loin)
 - Il faut typiquement recommencer plusieurs dizaines de fois un apprentissage en partant avec différentes initialisations des poids
- Efficacité en reconnaissance
 - Possibilité de temps réel

Sur-apprentissage



Sur-apprentissage (RN)



- Courbes pour 1 000 exemples
- *Quelles courbes si on avait 2 000 exemples ?*

Remarque sur les règles d'apprentissage

- La **retro-propagation** de gradient
 - Est une règle « **top-down** »
 - Les erreurs constatées en fin de calcul sont corrigées par retro-propagation
 - **Peu plausible** biologiquement
- La règle de **Hebb**
 - « *Fire together, wire together* »
Augmenter le poids des connexions entre deux neurones qui sont actifs pour le même exemple
 - Règle “**bottom-up**”. Locale sans coordination globale
 - Ne minimise pas explicitement une fonction de coût, mais apprend des représentations info-max (capturent dans les neurones autant d'information que possible)
 - **Plausibilité** biologique

Critère d'arrêt

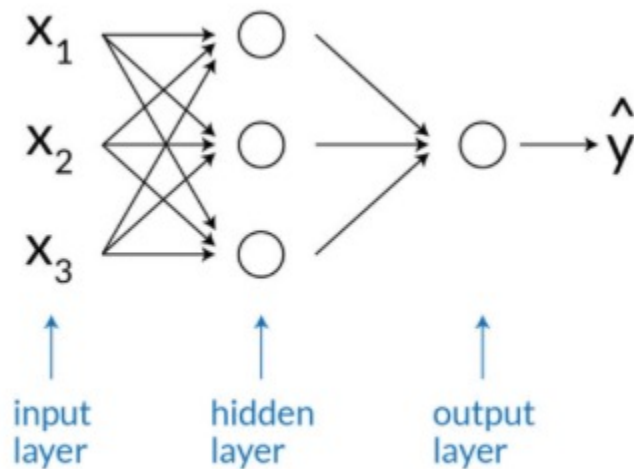
- Le problème des optima **locaux**
- Différence entre « **batch** » et « **en-ligne** »
 - La minimisation stochastique permet (en partie) d'échapper aux minima locaux
- Danger : le **sur-apprentissage** (« over-fitting »)
 - Early stopping rule

Le **choix** de l'architecture du réseau de neurones

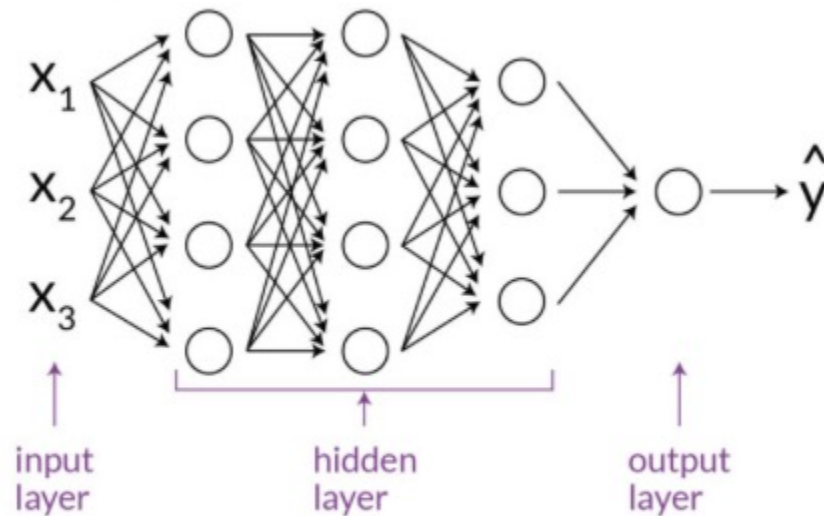
Comment choisir l'architecture du RN ?

- Contrôle la performance en généralisation

Shallow Neural Network

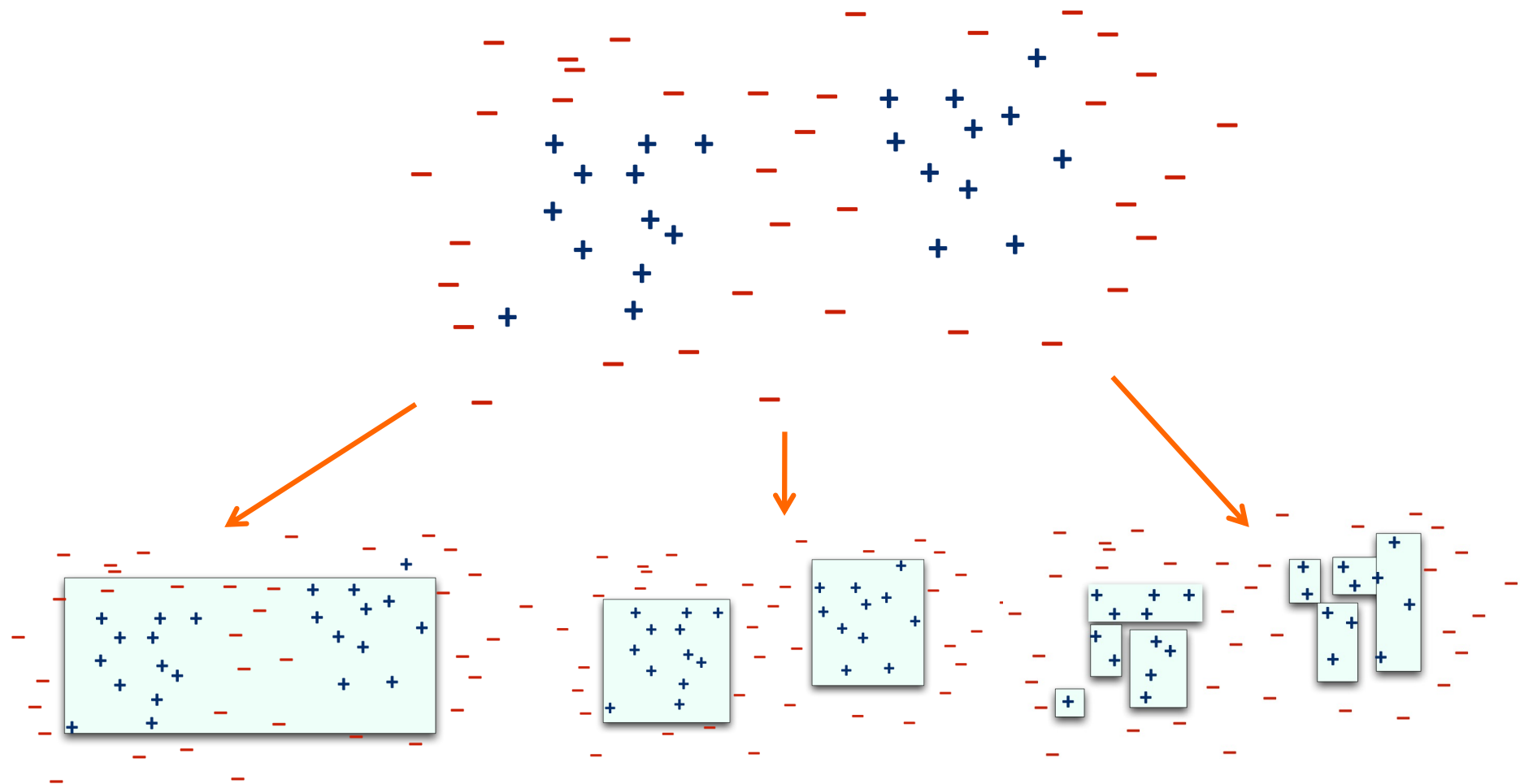


Deep Neural Network



Shallow and Deep Neural Networks.

Illustration



Sous-apprentissage

Modèle « correct »

Sur-apprentissage

La régression et le sur-apprentissage

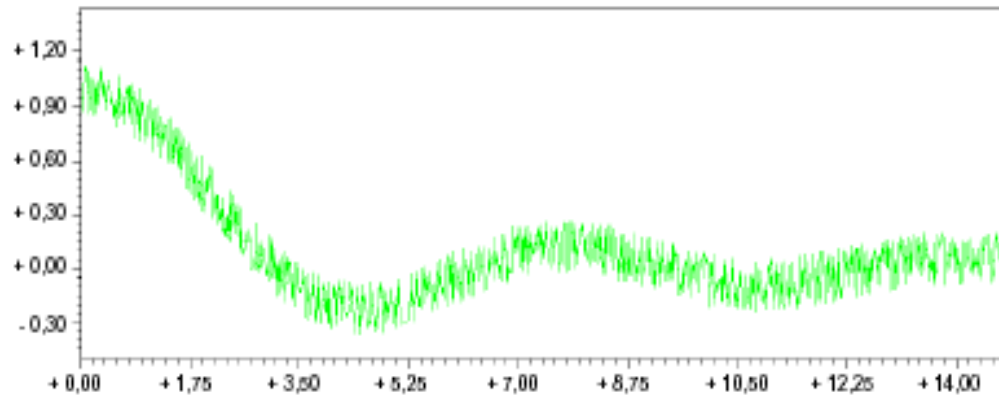


Figure 1-10. Un signal que l'on voudrait modéliser

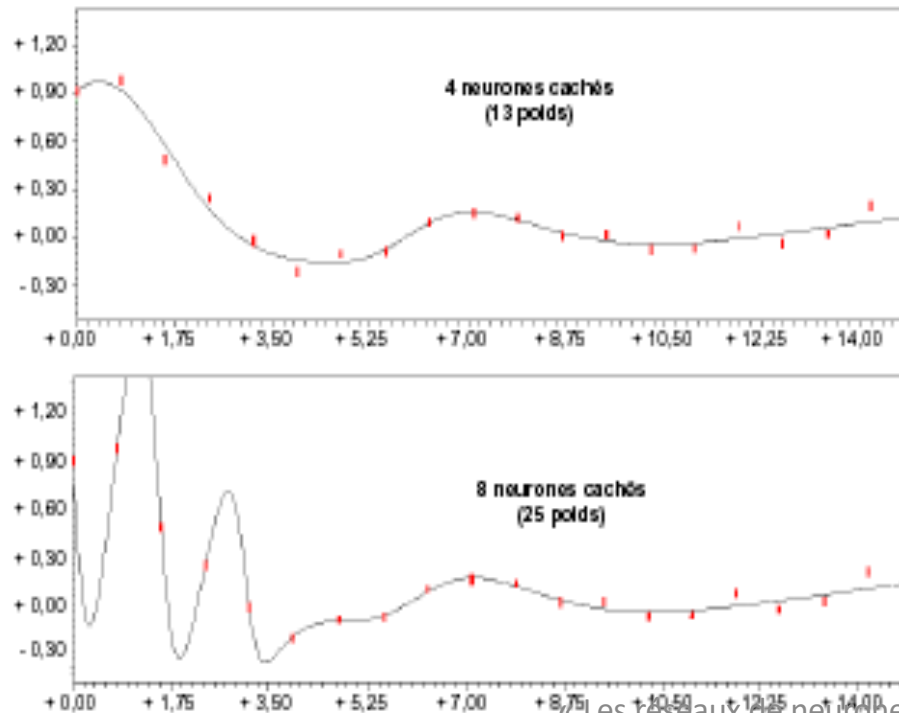
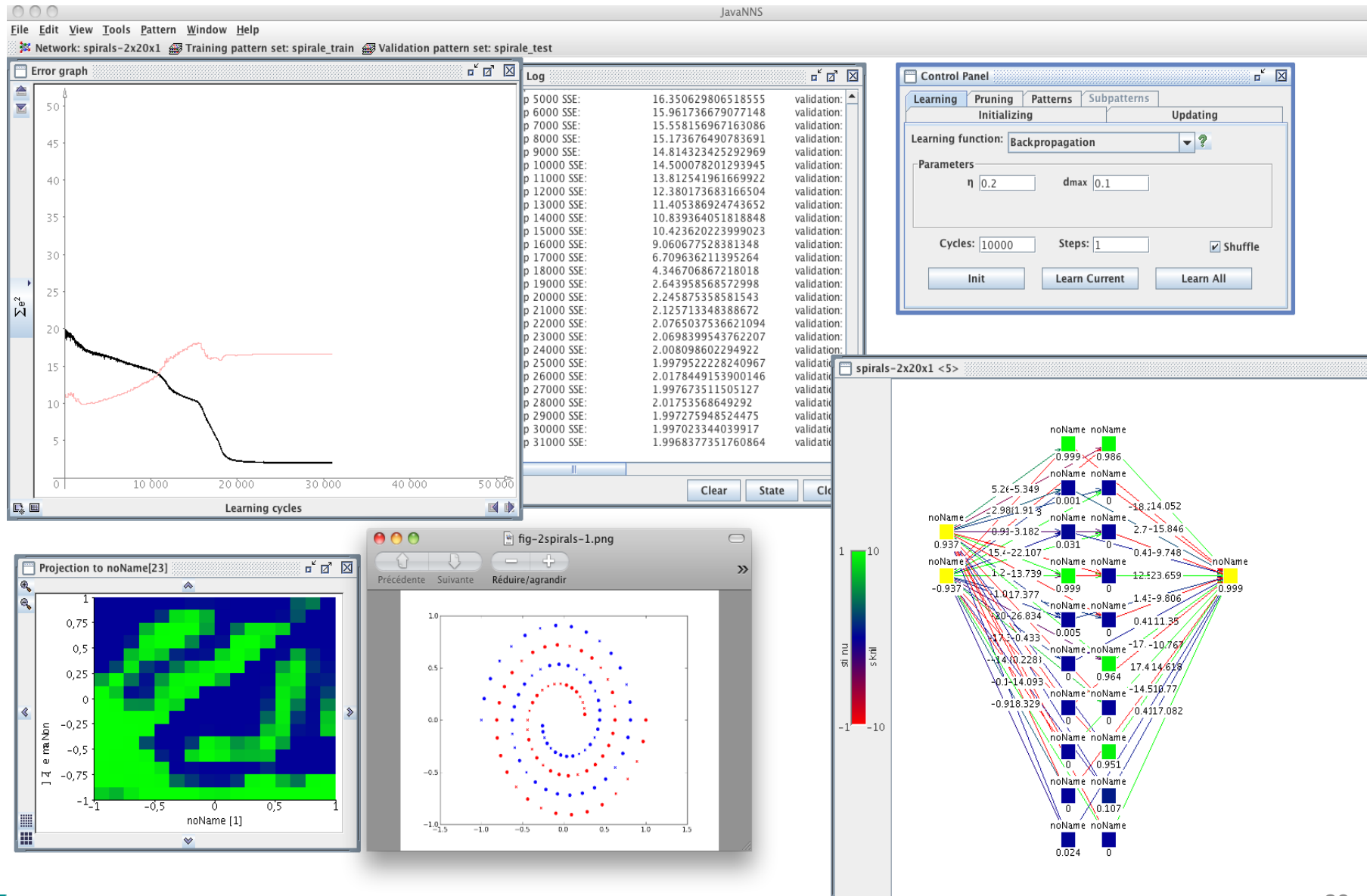
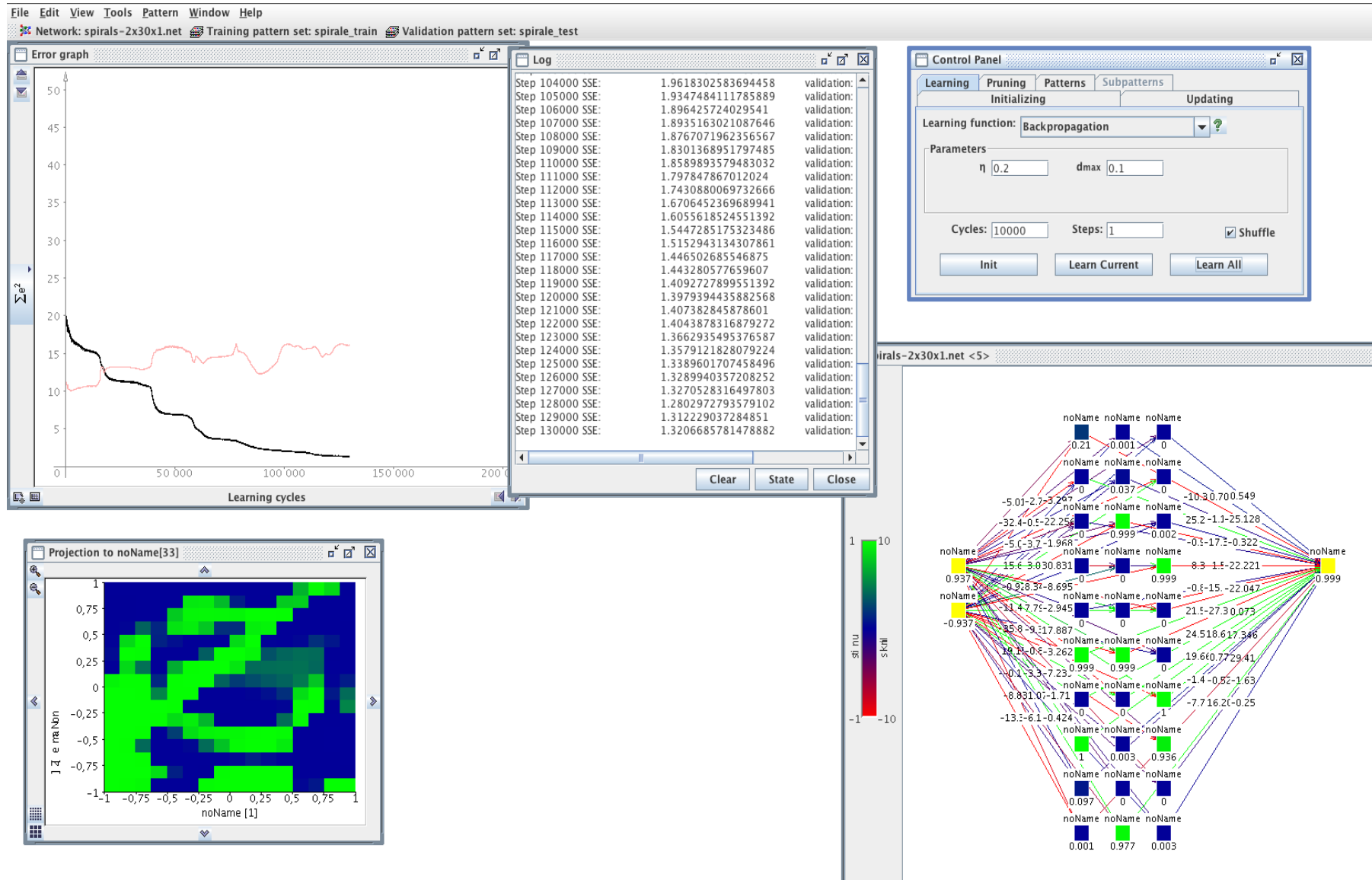


Figure 1-14.
Toutes choses
égales par
ailleurs, le
réseau de
neurones le plus
parcimonieux
possède les
meilleures
propriétés de
généralisation.

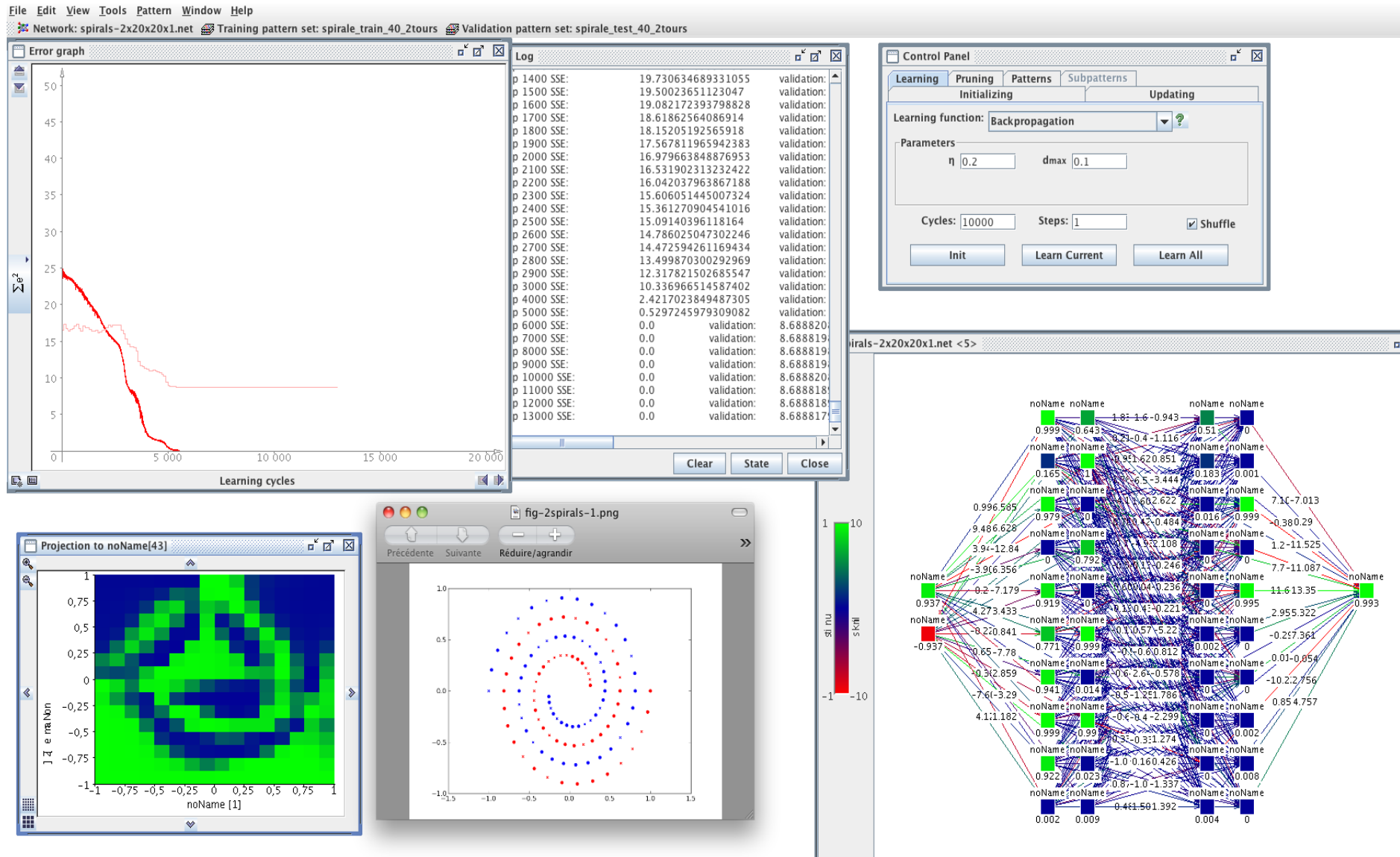
Illustration



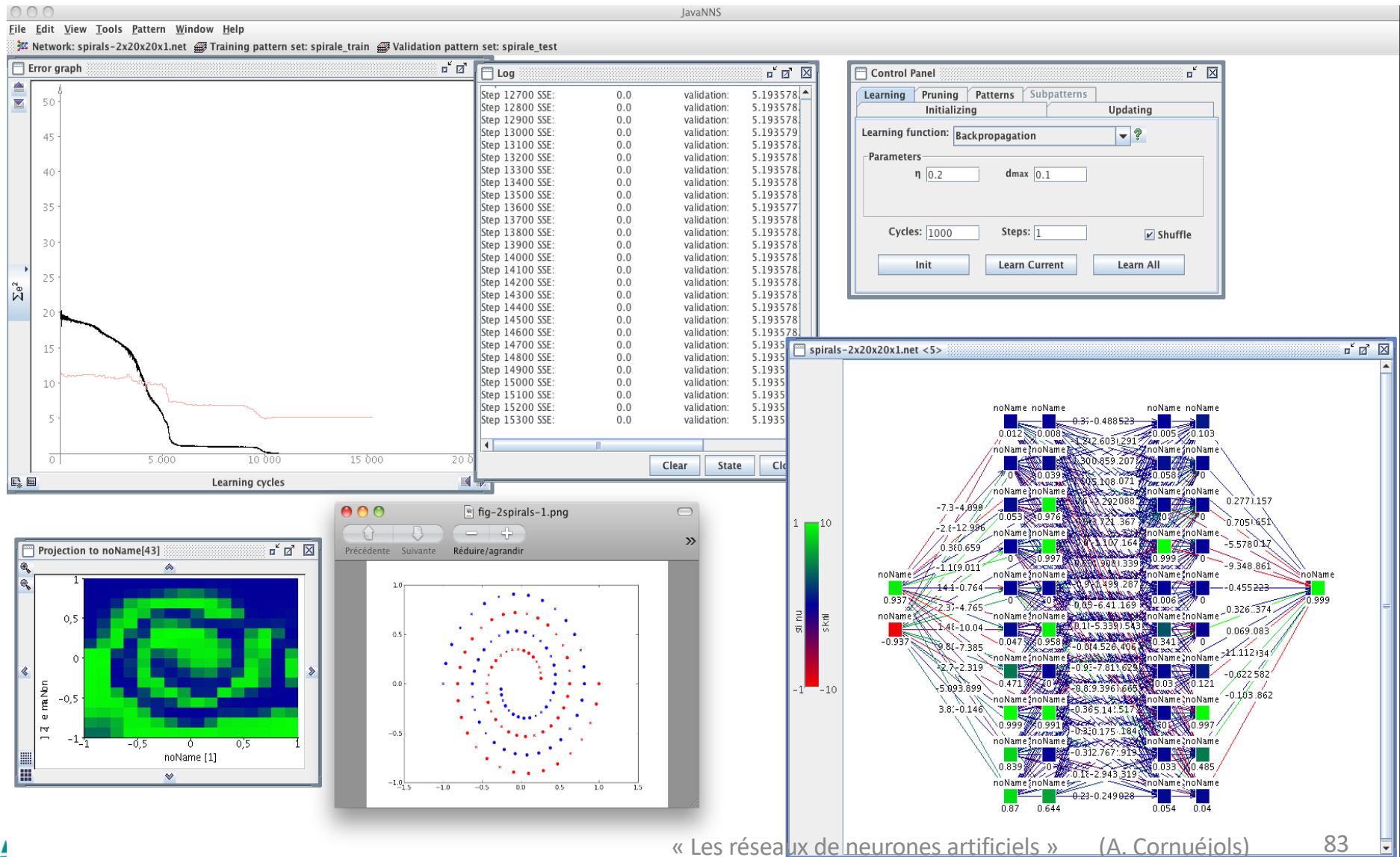
Illustration



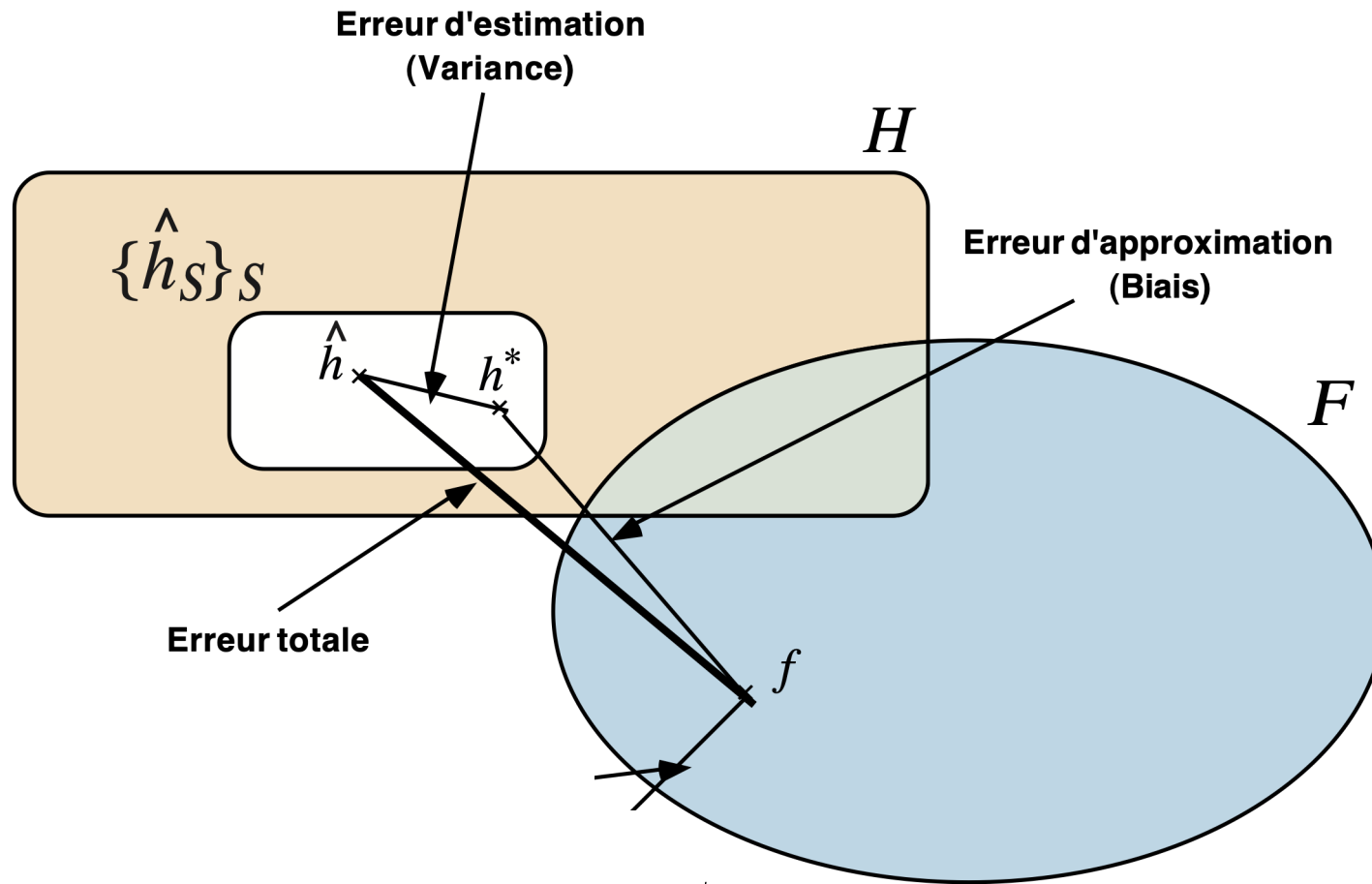
Illustration



Illustration



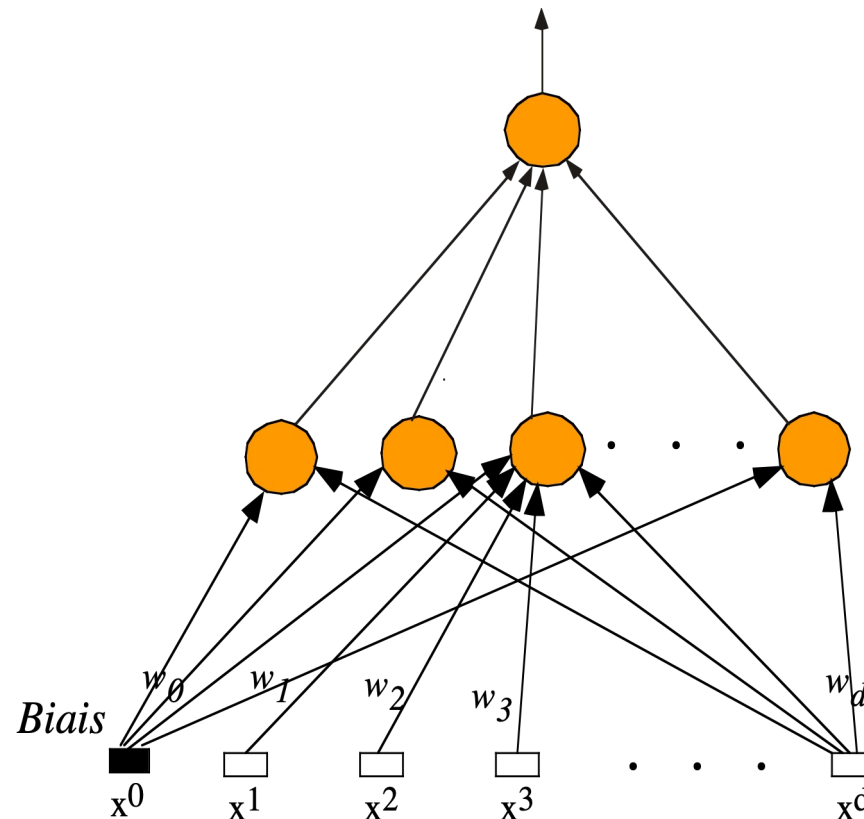
Le compromis biais-variance



- Illustration

Le codage de la couche de sortie

- **Un seul** neurone de sortie
 - $\{0,1\}$
 - $[0,1]$
 - Erreur quadratique
 - Probabilité $[0,1]$
 - Critère entropique
- **Plusieurs** neurones de sortie
 - Sortie considérée comme une **probabilité d'appartenance**
Voir **critère entropique**
(tr. suivant)



La « calibration » de la sortie

- Comment obtenir **une probabilité en sortie** ?
- Une solution :
 - Utilisation d'un signal d'erreur par **entropie croisée**

$$l(y_i, \hat{y}_i) = -[y_i * \log \hat{y}_i + (1 - y_i) * \log(1 - \hat{y}_i)],$$

where $\hat{y}_i = \frac{1}{1 + \exp(-w^T x_i)}, y_i \in \{0,1\}.$

$\hat{y}_i$ is the output of the unit for the example x_i .

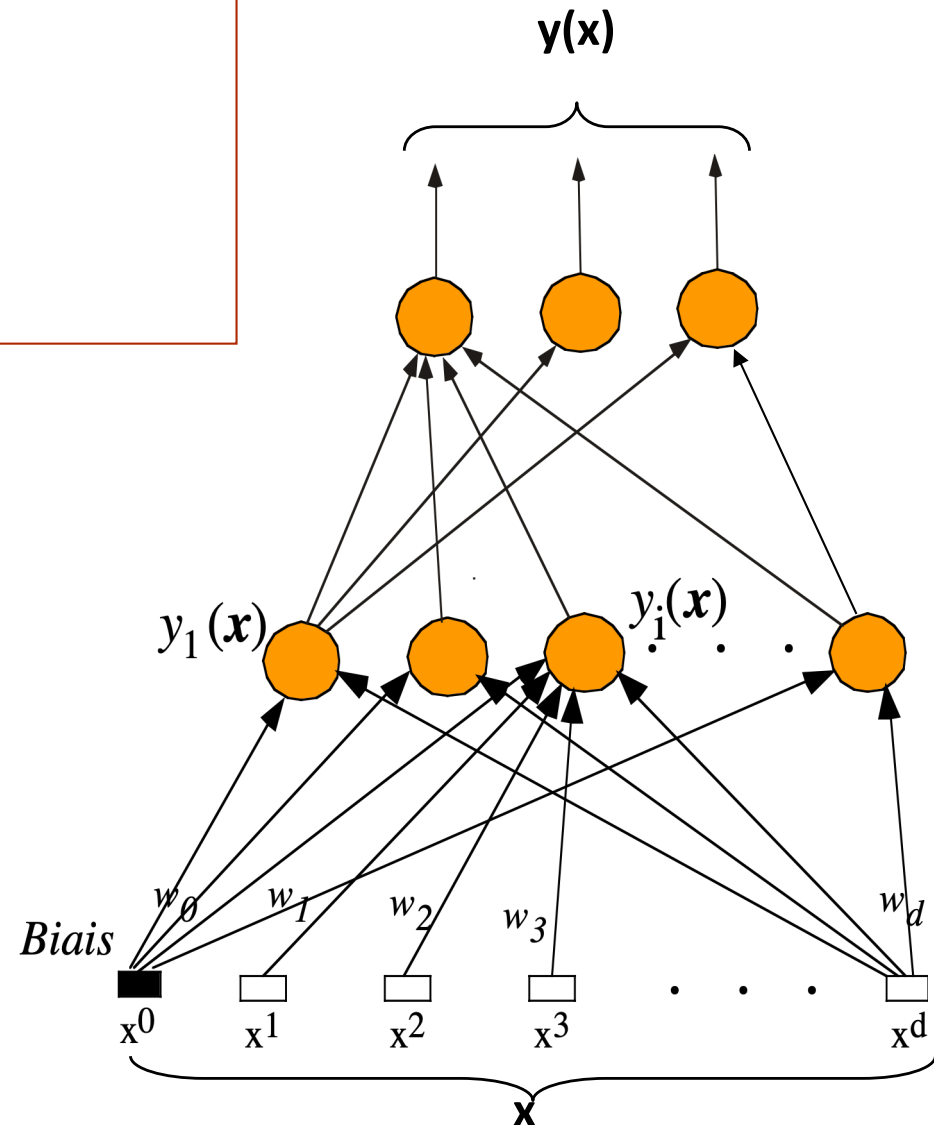
It can be considered as the probability that the input is of class C

Le codage de la couche de sortie

Exemple :

- Reconnaissance de caractères manuscrits
- Reconnaissance de locuteurs

- $c-1$ problèmes de discrimination
- 1 neurone de sortie
 - $\{0, 1, \dots, c\}$
 - $[0, 1]$
- $n (\leq c)$ neurones de sortie
 - 1 neurone / classe
 - Code correcteur d'erreur



ECOC

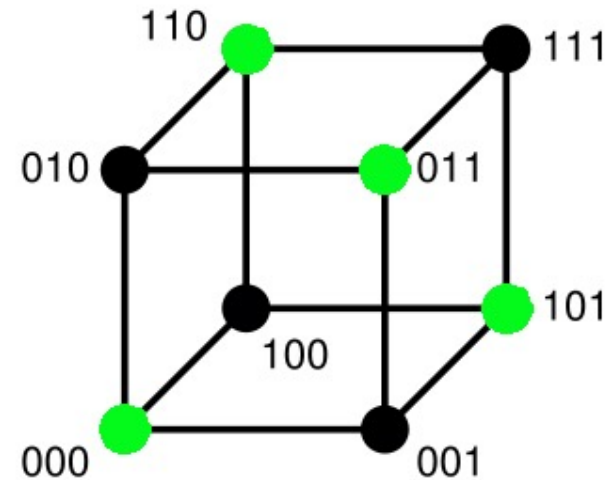
- Apprendre à distinguer 26 lettres
- Codage
 - Sur 5 bits ?
 - Sur 10 bits !

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}
A	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	1	0	1	0	1	1
C	0	0	0	1	0	1	0	1	0	1

4
6
4

Codes correcteurs d'erreur

On cherche à maximiser
le nombre de bits
différents entre codes



Message = E	Code = F(E)
0 0	0 0 0
0 1	1 0 1
1 0	1 1 0
1 1	0 1 1

ECOC

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}
A	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	1	0	1	0	1	1
C	0	0	0	1	0	1	0	1	0	1

- On apprend 10 classifieurs binaires

$$h_5(\mathbf{x}) = \begin{cases} 0 & \text{si } y \in \{A, C, E, G, I, K, M, O, Q, S, U, W, Y\} \\ 1 & \text{sinon} \end{cases}$$

$$h_4(\mathbf{x}) = \begin{cases} 0 & \text{si } y \in \{A, B, E, F, I, J, M, N, Q, R, U, V, Y, Z\} \\ 1 & \text{sinon} \end{cases}$$

- On calcule $h_1(\mathbf{x}), h_2(\mathbf{x}), h_3(\mathbf{x}), h_4(\mathbf{x}), h_5(\mathbf{x}), h_6(\mathbf{x}), h_7(\mathbf{x}), h_8(\mathbf{x}), h_9(\mathbf{x}), h_{10}(\mathbf{x})$
- On décode $H(\mathbf{x})$ par plus proche voisin / au codage de A, B, C, ...

Bilan : pourquoi les reseaux de neurons sont séduisants

1. L'espace des hypothèses associé est de **grande capacité**
 - Approximateurs universels à la limite
 - Pose des questions sur l'induction
2. Les paramètres peuvent être appris par **descente de gradient**
3. Les calculs sont **parallélisables**
4. On peut y mettre de la **structure** pour refléter l'organisation du monde
5. Ils apprennent des **abstractions**

Plan

1. Introduction et réseaux de neurones à une couche
2. Les perceptrons multi-couches
3. L'apprentissage dans les réseaux de neurones
4. Une introduction aux réseaux de neurones profonds
5. Autres architectures

L'ancêtre des réseaux convolutionnels

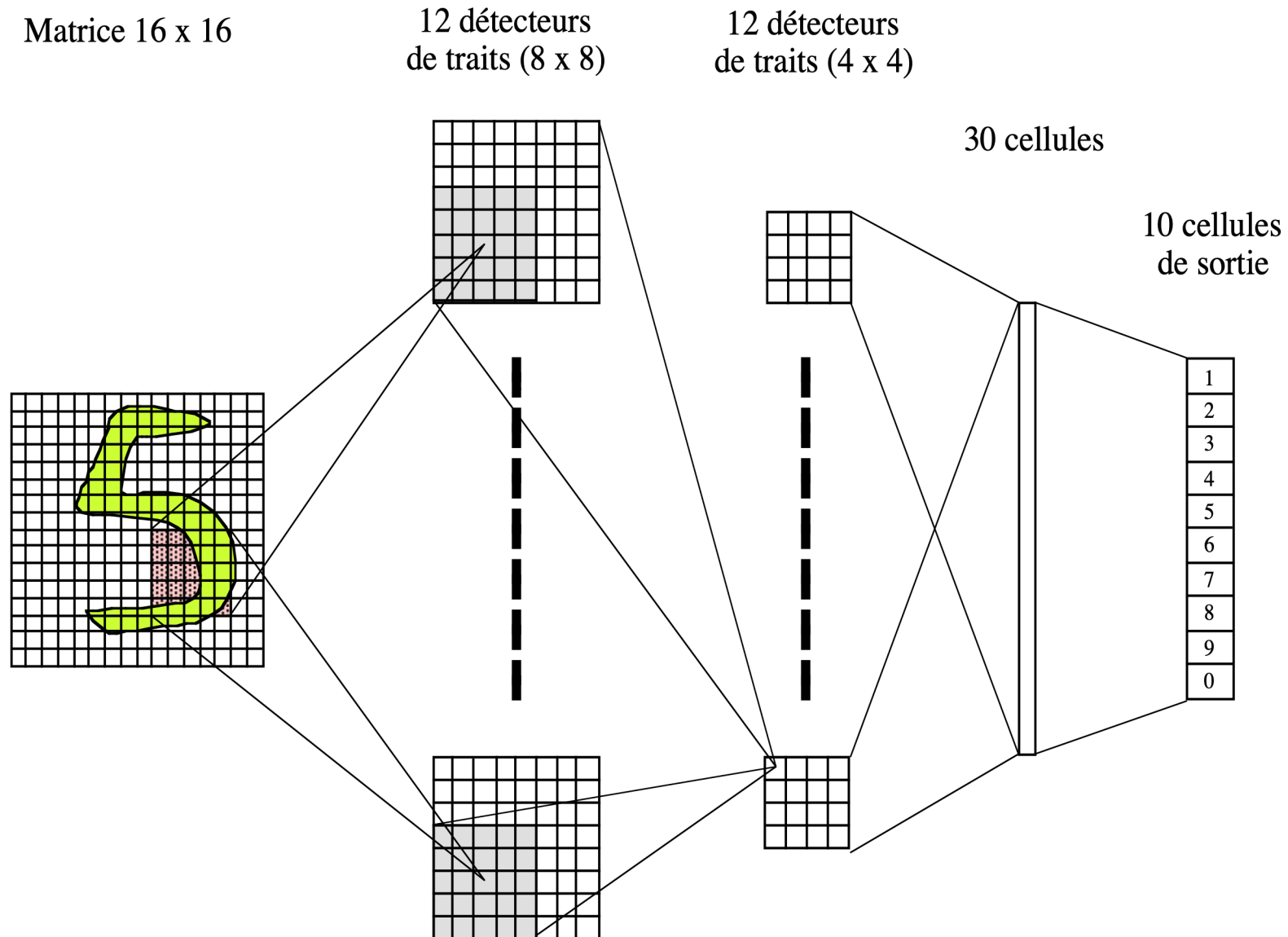
Application aux codes postaux (Zip codes)

- [Le Cun et al., 1989, ...] (ATT Bell Labs : très forte équipe)
- ≈ 10000 exemples de chiffres manuscrits
- Segmentés et redimensionnés sur matrice 16×16
- Technique des poids partagés (“weight sharing”)
- Technique du optimal brain damage
- 99% de reconnaissance correcte (sur l’ensemble d’apprentissage)
- 9% de rejet (pour reconnaissance humaine)

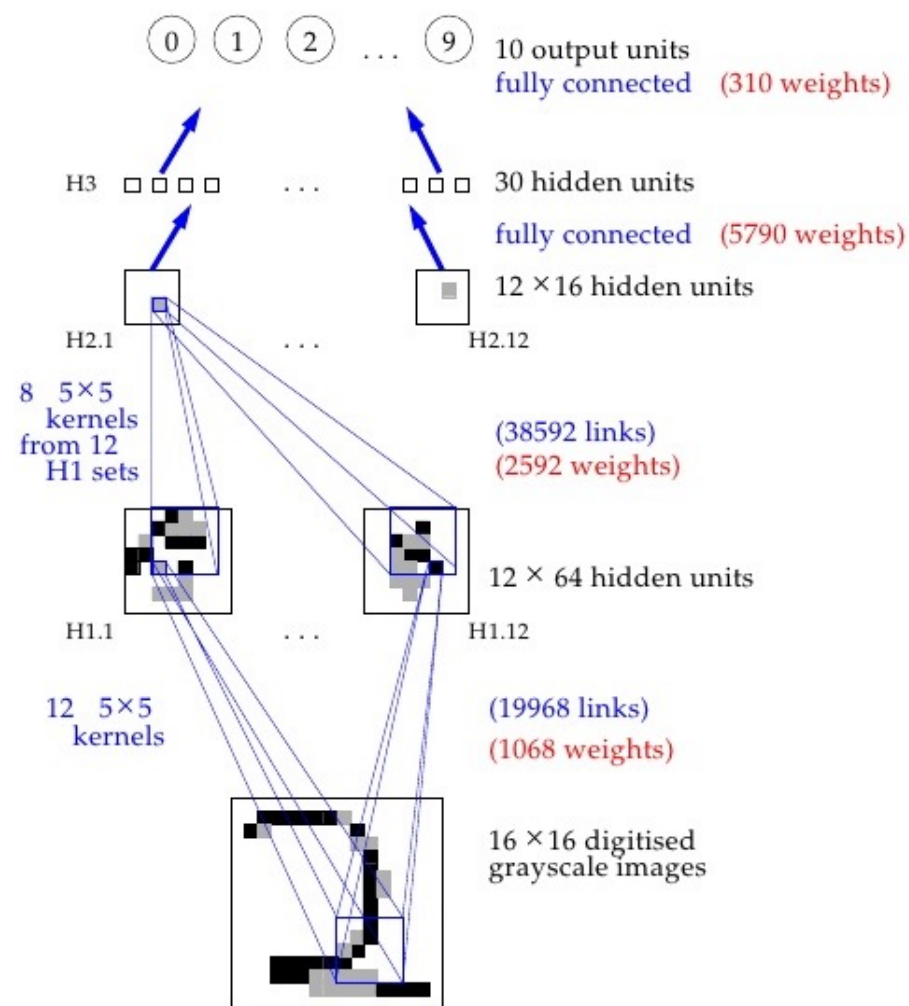
La base de données

65473 60198 68544
70065 70117 19032 96720
27260 61820 19559
74136 ~~19137~~ 63101
20878 60521 38002
48640-2398 20907 14868

Réseaux à convolution : Application aux codes postaux



Réseaux à convolution : Application aux codes postaux



Before weight sharing 64660 links
After weight sharing 9760 weights

- Taken from <http://image.slidesharecdn.com/bp2slides-090922011749-phpapp02/95/the-back-propagation-learning-algorithm-10-728.jpg?cb=1253582278>

Exemples d'erreurs de prédiction

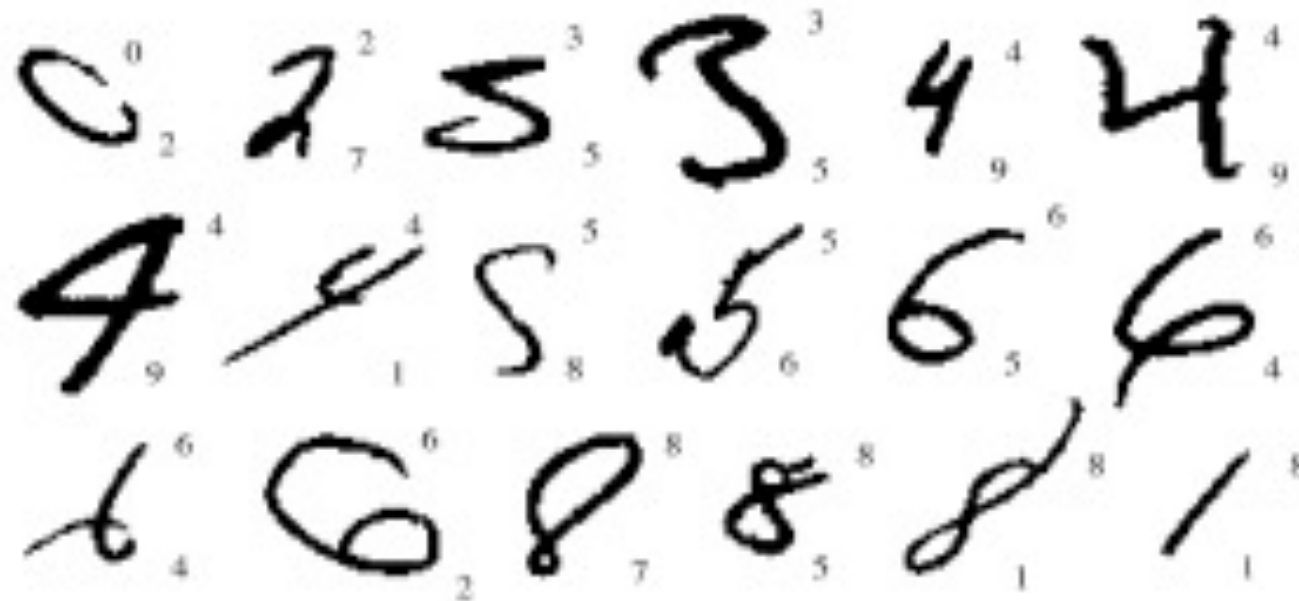


Figure 1-34. Les 18 erreurs de classification commises par séparation linéaire des classes deux à deux. Pour chaque chiffre manuscrit, l'indication en haut à droite est la classe d'appartenance du chiffre indiquée dans la base, et le chiffre en bas à droite est la classe affectée par le classifieur.

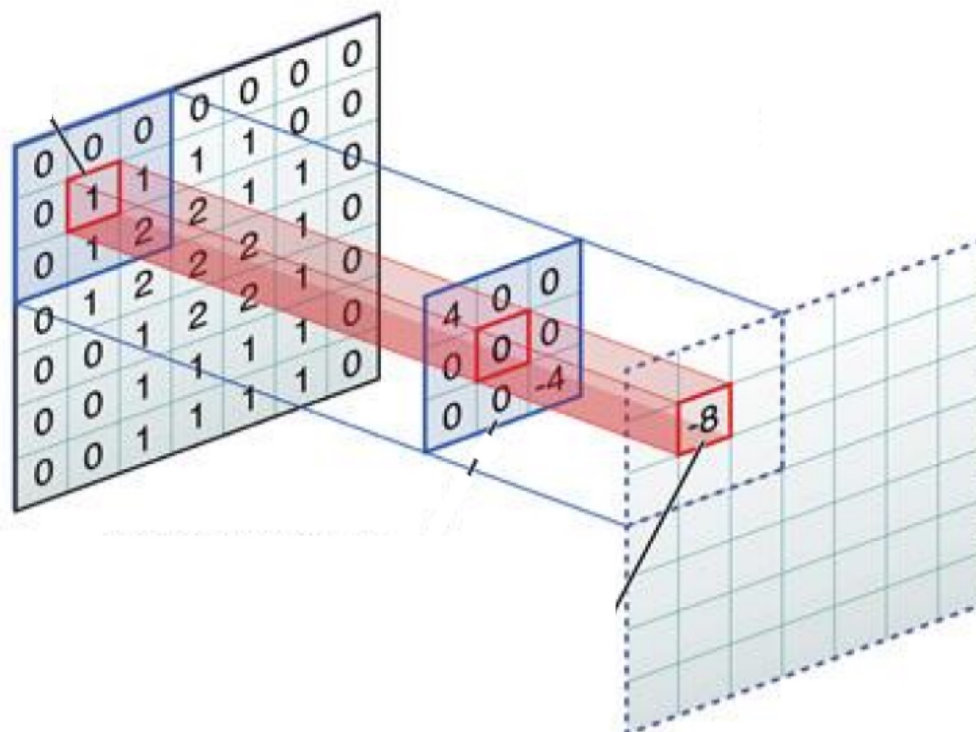
Examples of prediction error

 4→6	 3→5	 8→2	 2→1	 5→3	 4→8	 2→8	 3→5	 6→5	 7→3
 9→4	 8→0	 7→8	 5→3	 8→7	 0→6	 3→7	 2→7	 8→3	 9→4
 8→2	 5→3	 4→8	 3→9	 6→0	 9→8	 4→9	 6→1	 9→4	 9→1
 9→4	 2→0	 6→1	 3→5	 3→2	 9→5	 6→0	 6→0	 6→0	 6→8
 4→6	 7→3	 9→4	 4→6	 2→7	 9→7	 4→3	 9→4	 9→4	 9→4
 8→7	 4→2	 8→4	 3→5	 8→4	 6→5	 8→5	 3→8	 3→8	 9→8
 1→5	 9→8	 6→3	 0→2	 6→5	 9→5	 0→7	 1→6	 4→9	 2→1
 2→8	 8→5	 4→9	 7→2	 7→2	 6→5	 9→7	 6→1	 5→6	 5→0
 4→9	 2→8								

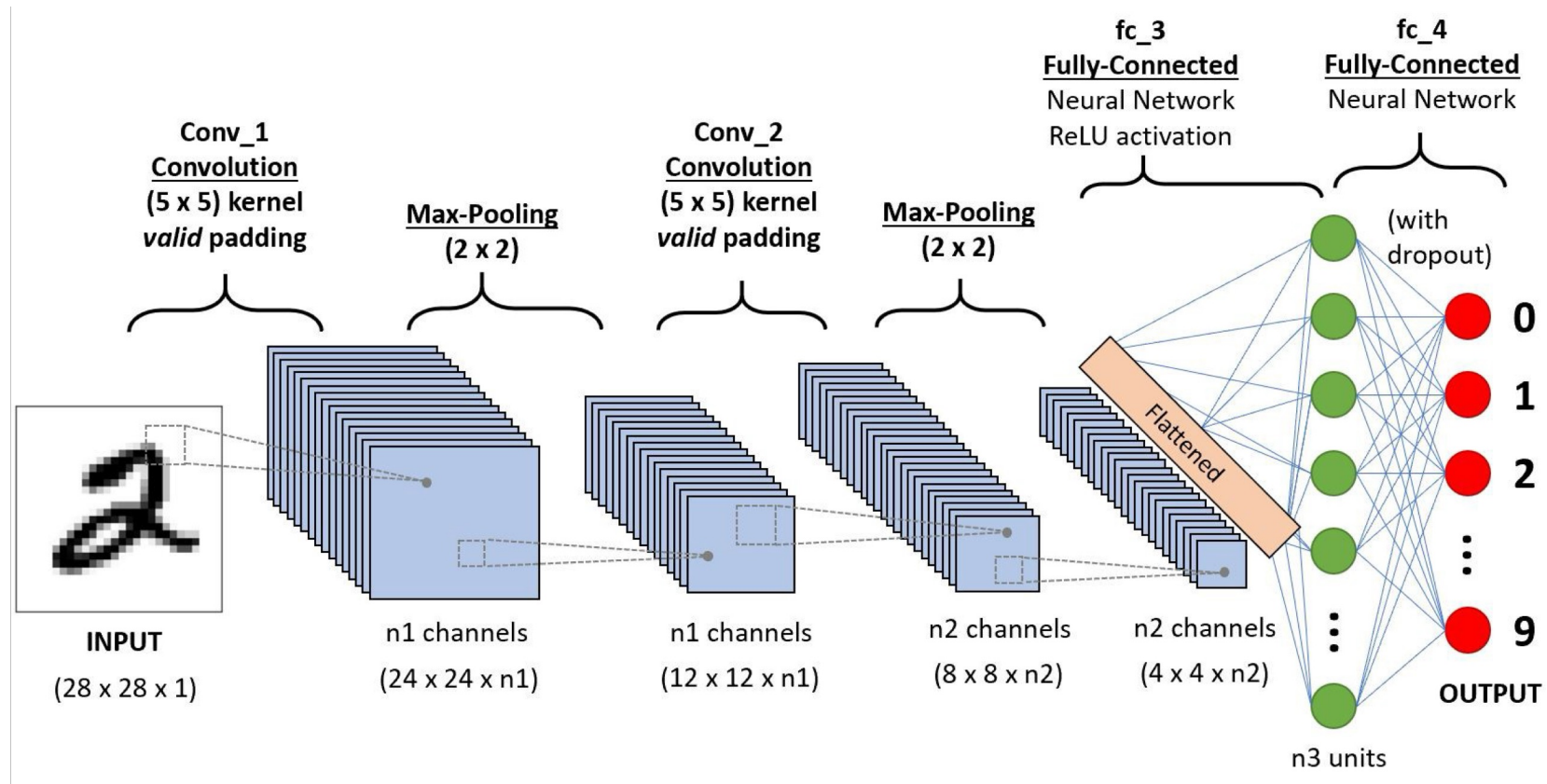
Convolutions

Same dimension between layers

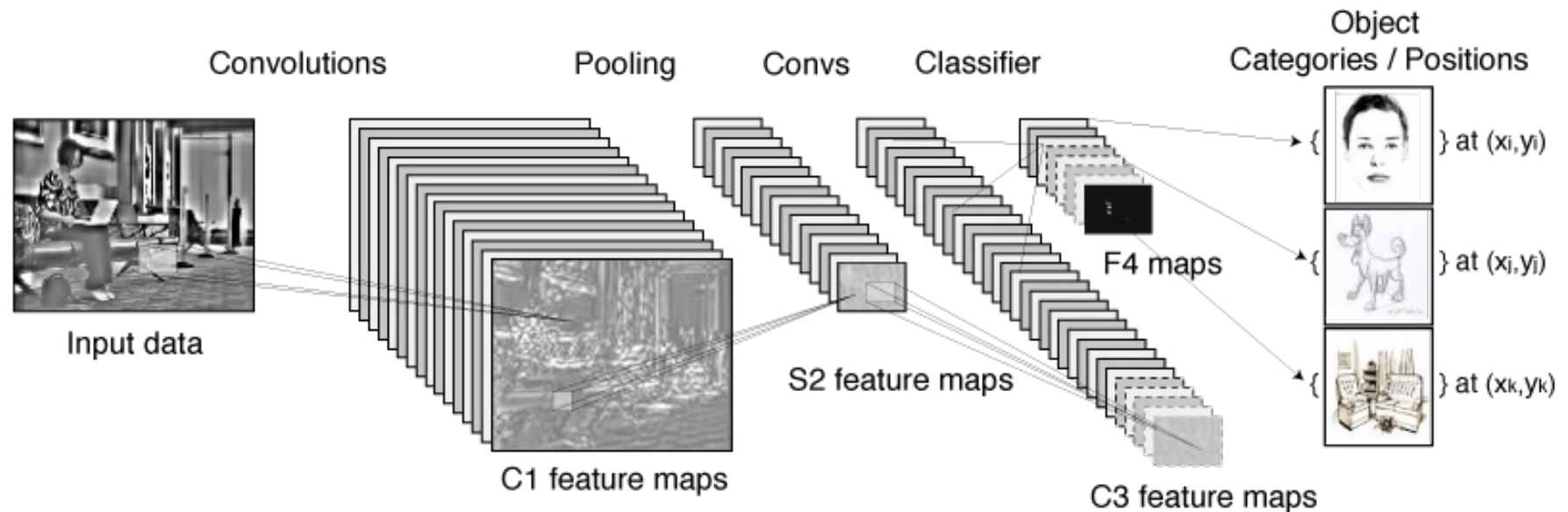
$$(4 \times 0) + (0 \times 0) + (0 \times 0) + (0 \times 0) + (0 \times 1) + (0 \times 1) + (0 \times 0) + (0 \times 1) + (-4 \times 2) = -8$$



Réseaux de neurones convolutionnels



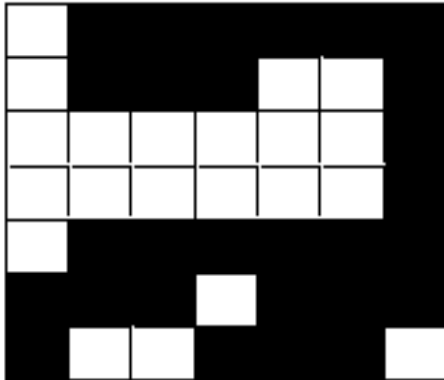
Réseaux de neurones convolutionnels (2° exemple)



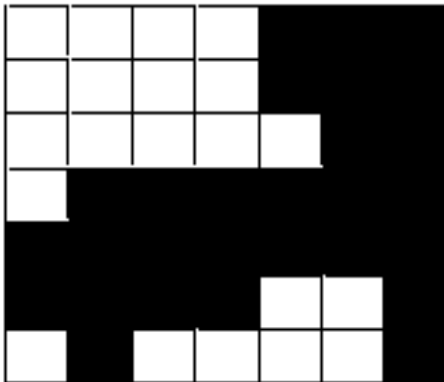
- 1) Le **pooling** consiste réduire la résolution des images filtrées, par exemple en moyennant les pixels contigus (*ex : 4 pixels deviennent 1 seul pixel qui contient une valeur moyenne*) - étape S2
- 2) **Convolution** : des filtres sont appris sur les nouvelles images - étape C3
- 3) **Classifieur** : les dernières couches sont entièrement connectées (*i.e. MLP*) et apprennent à prédire la classe à partir des filtres appris (*i.e. des descripteurs générés automatiquement*) - étape F4

How to code the inputs

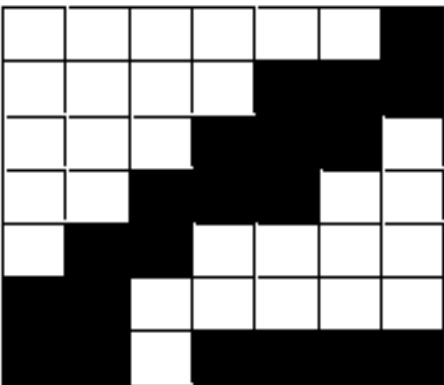
Learning is easy when we know what to look for



- Yes

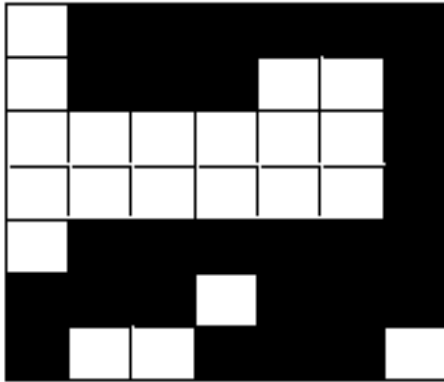


- Yes



- No

Inputs and prior knowledge

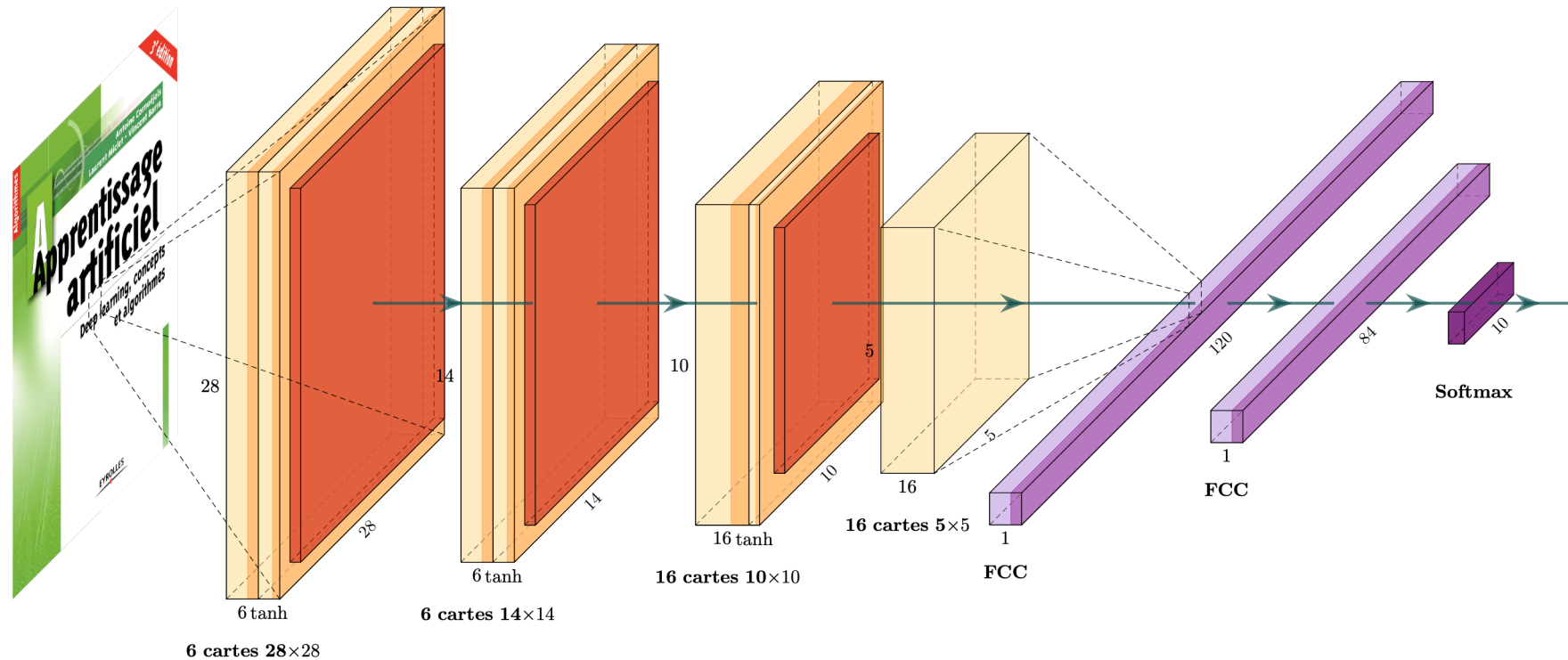


- Is it a pattern recognition task? A character recognition task? ...
- *How to code the examples?*

0 1 1 1 1 1 1 0 1 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 1 1 0

- A right choice of representation can render the learning task trivial
 - But how can we know the right representation?

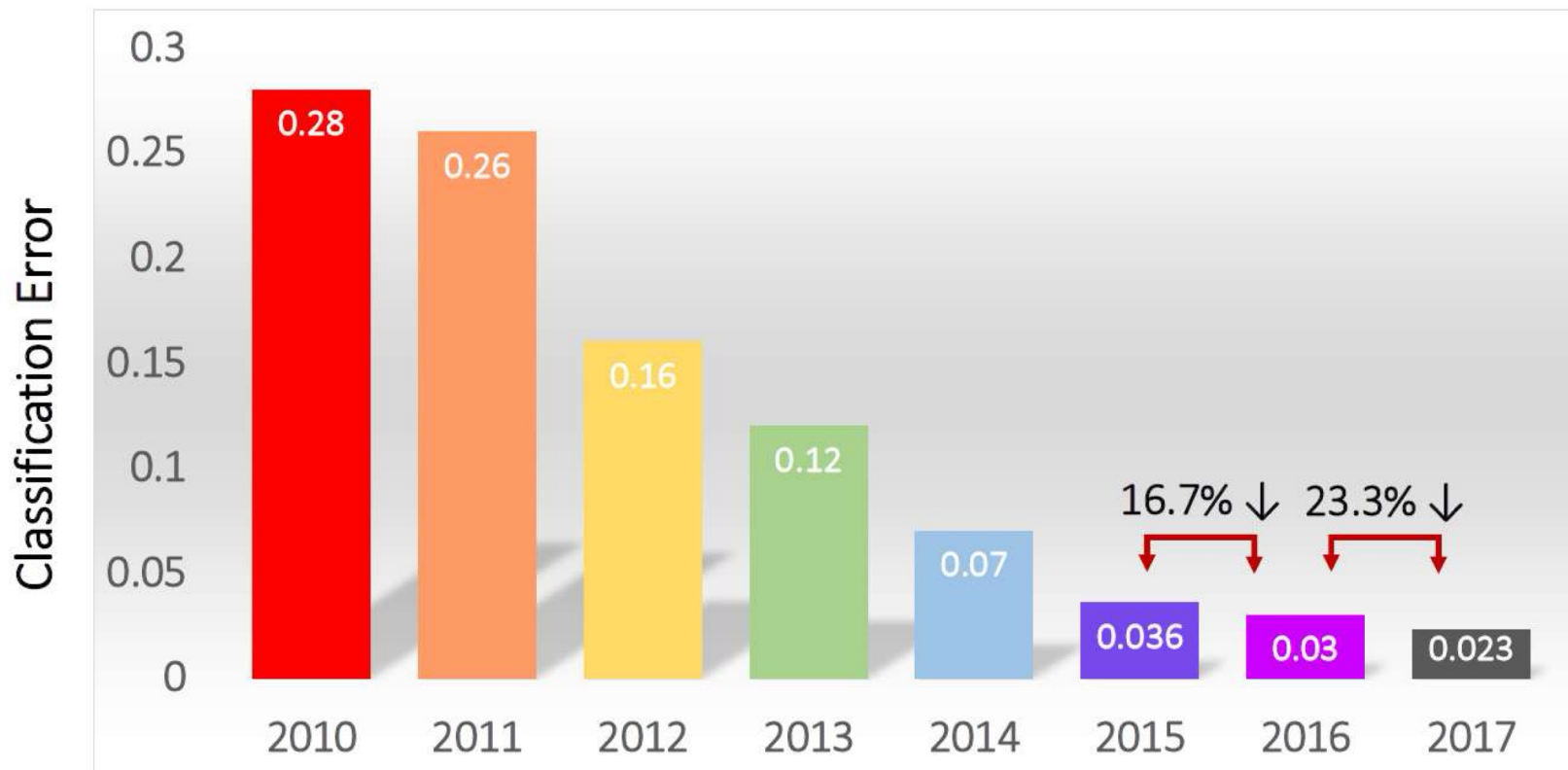
LeNet-5



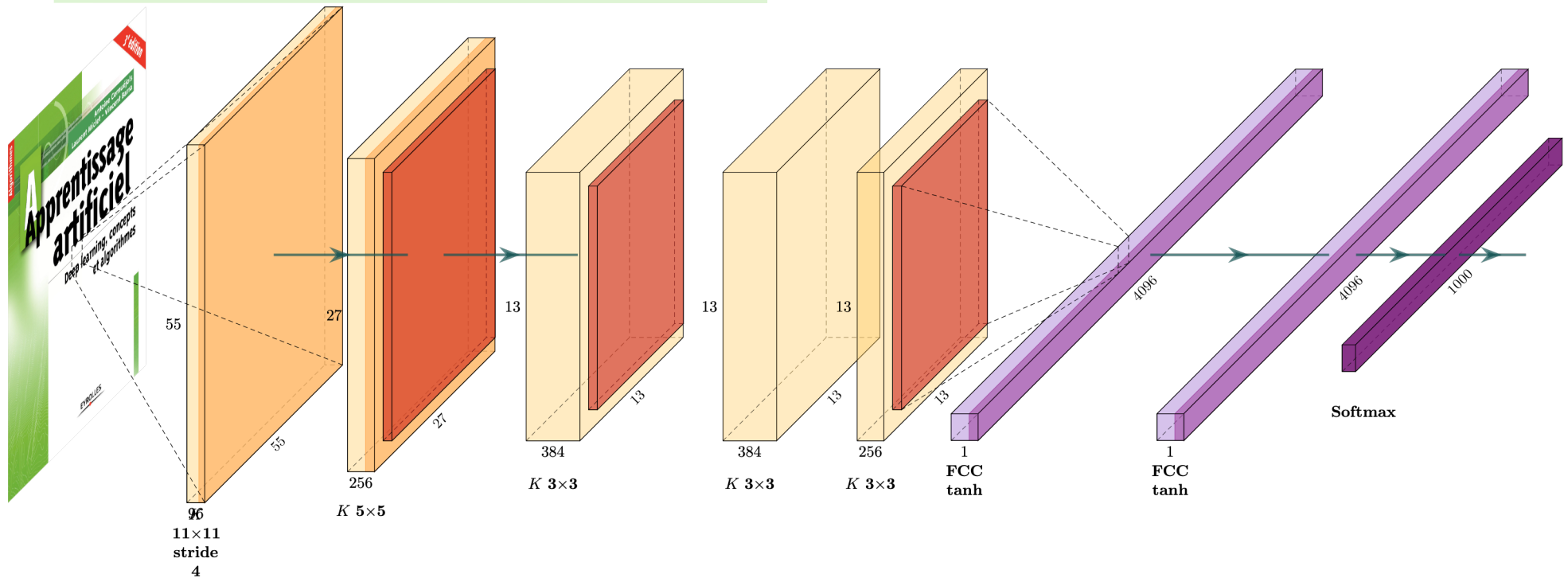
- Architecture du réseau LeNet-5.
 - Les couches de **convolution** et d'activation sont en **orange clair**.
 - Les couches d'**agrégation** en **orange foncé**
 - Les couches **complètement connectées** sont en **violet**

Les performances sur les compétition

- Résultats en classification

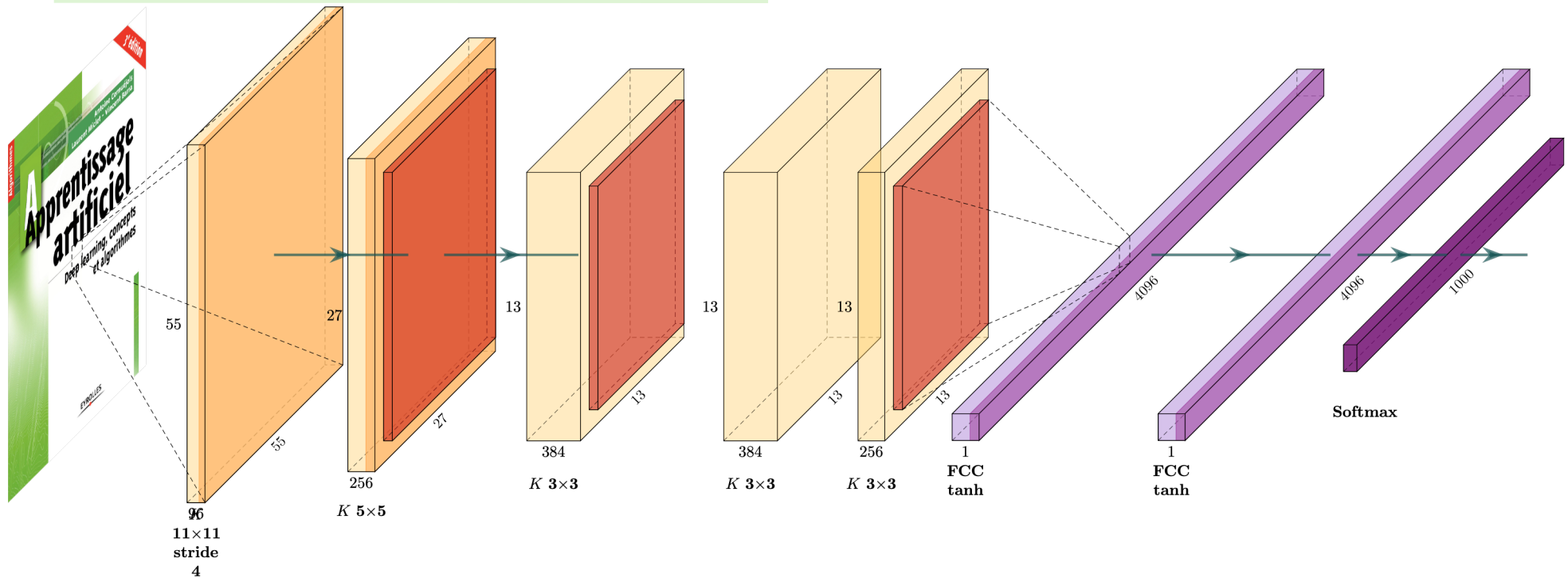


AlexNet



- Architecture du réseau AlexNet.
 - Les couches de **convolution** et d'activation sont en **orange clair**.
 - Les couches d'**agrégation** en **orange foncé**
 - Les couches **complètement connectées** sont en **violet**

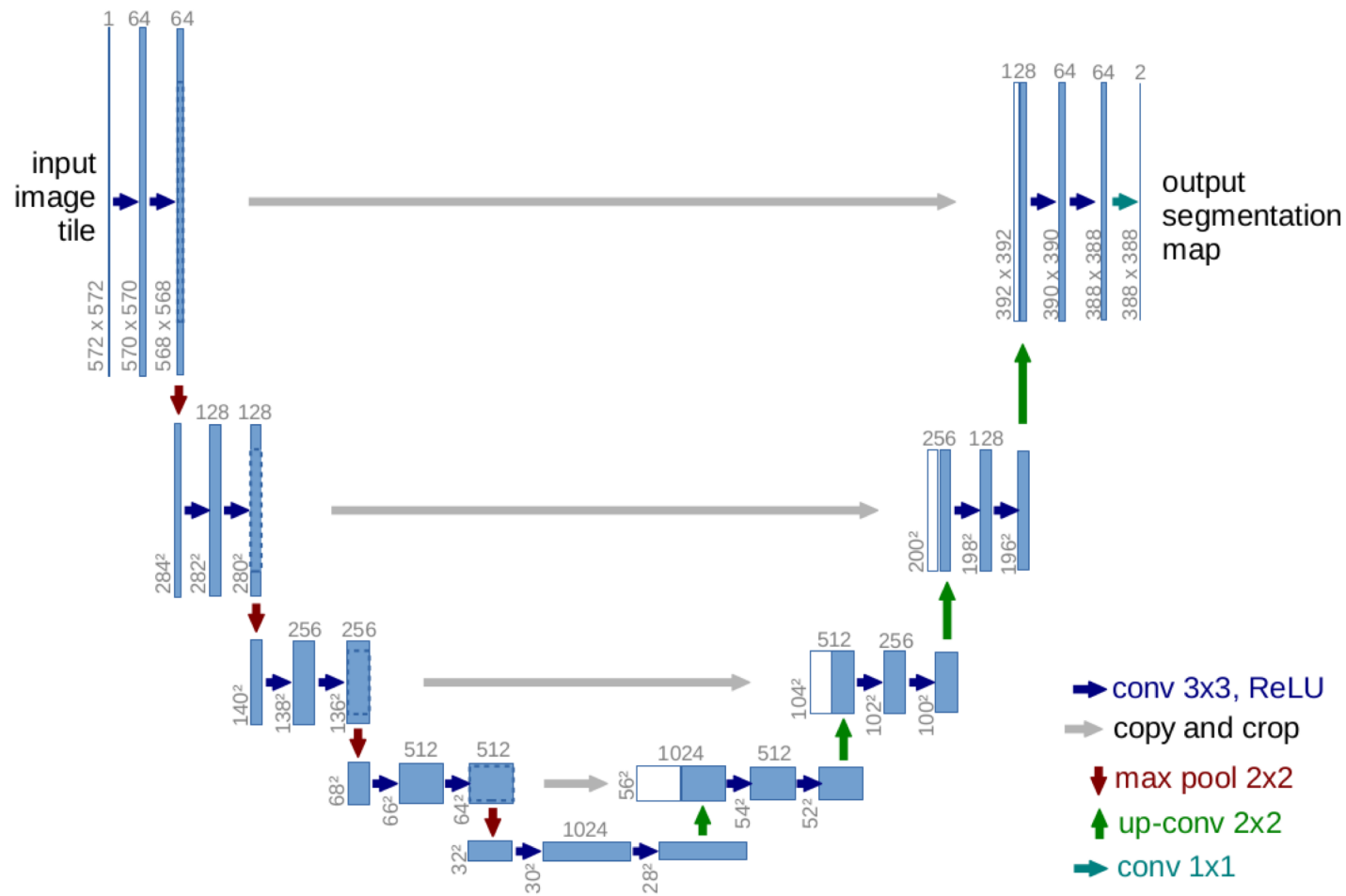
AlexNet



Si la profondeur du réseau reste faible, **le nombre de paramètres est déjà important.**

- En regardant **uniquement** la **première couche de convolution**, l'entrée est composée d'images **227 x 227 x 3** (3 canaux), les filtres de convolution sont de taille **11** et le stride (décalage entre fenêtres) est de **4** (d'où 55 fenêtres en largeur et hauteur : $4 * 55 = 220$) . (Rq : $55 = (227 - \text{kernel width } 11) / \text{stride } 4 + 1$).
- La sortie de la couche de convolution est de taille **55 x 55 x 96 = 290 400** neurones, chacun ayant **11 x 11 x 3 = 363** poids et un biais. Cela implique, **sur cette couche** de convolution seulement, **105 705 600 paramètres** à ajuster.

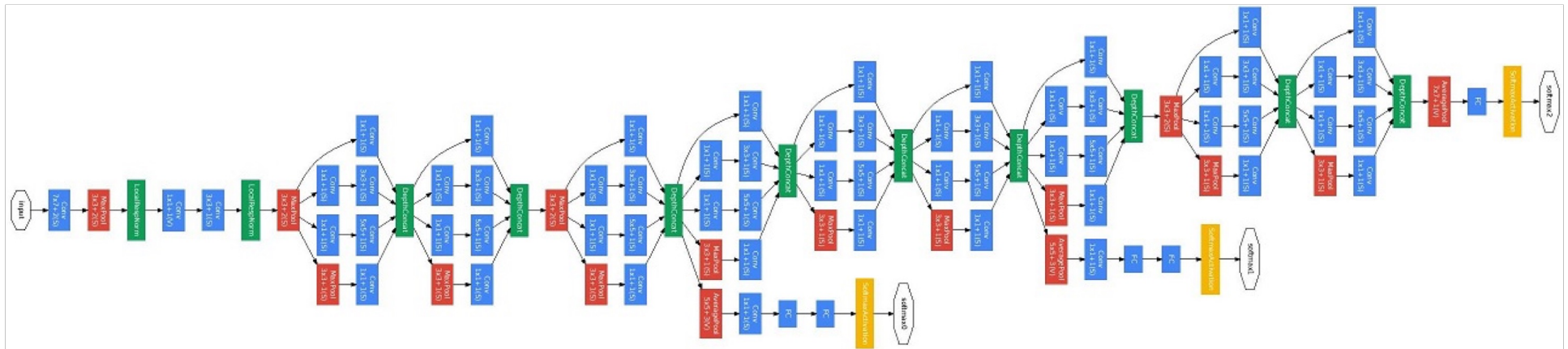
U-Net : pour la segmentation d'images



Cette architecture est intégralement convolutive et symétrique. Dans la première partie de l'architecture, la taille de la sortie diminue et le nombre de filtres augmente. La seconde partie inverse le processus avec une carte de prédictions en sortie.

GoogleNet

- Un **mécano** de réseaux de neurones



Illustration

Système développé par Google et U. de Stanford

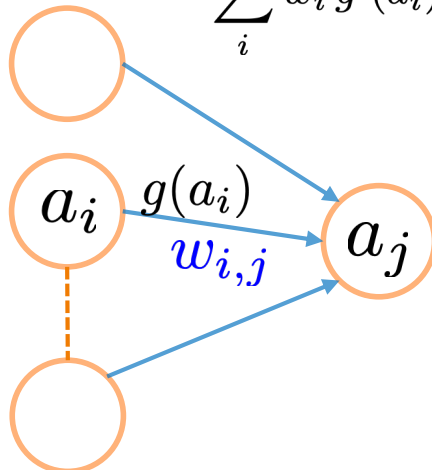
- Reconnaissance de visages
 - Sous conditions de lumière diverses
 - Sous tout angle
- Apprentissage non supervisé
 - 9 couches ; 10^9 connexions
 - 10 millions d'images
 - 3 jours de calcul sur 16 000 processeurs
- Amélioration des performances de 70% / état de l'art

Les neurones des couches cachées : que représentent-ils ?

- Dans la descente de gradient par rétro-propagation, **symétrie** :
 - entre les **poids** des connexions
 - et les **exemples**
- On peut donc **optimiser sur les exemples** plutôt que sur les poids des connexions

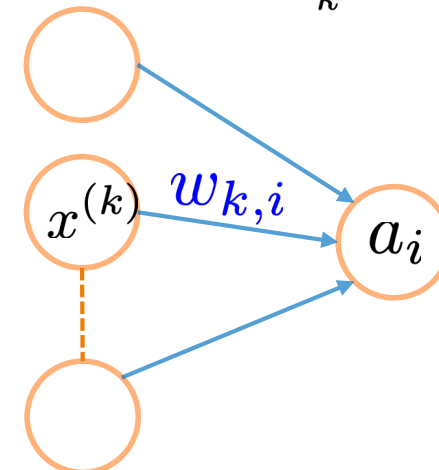
Couche **cachée** :

$$a_j = \sum_i w_{i,j} g(a_i)$$
$$\frac{\partial a_j}{\partial x^{(k)}} = \sum_i w_{i,j} \frac{\partial g(a_i)}{\partial x^{(k)}} = \sum_i w_{i,j} \frac{\partial g(a_i)}{\partial a_i} \frac{\partial a_i}{\partial x^{(k)}}$$
$$= \sum_i w_{i,j} g'(a_i) \frac{\partial a_i}{\partial x^{(k)}}$$



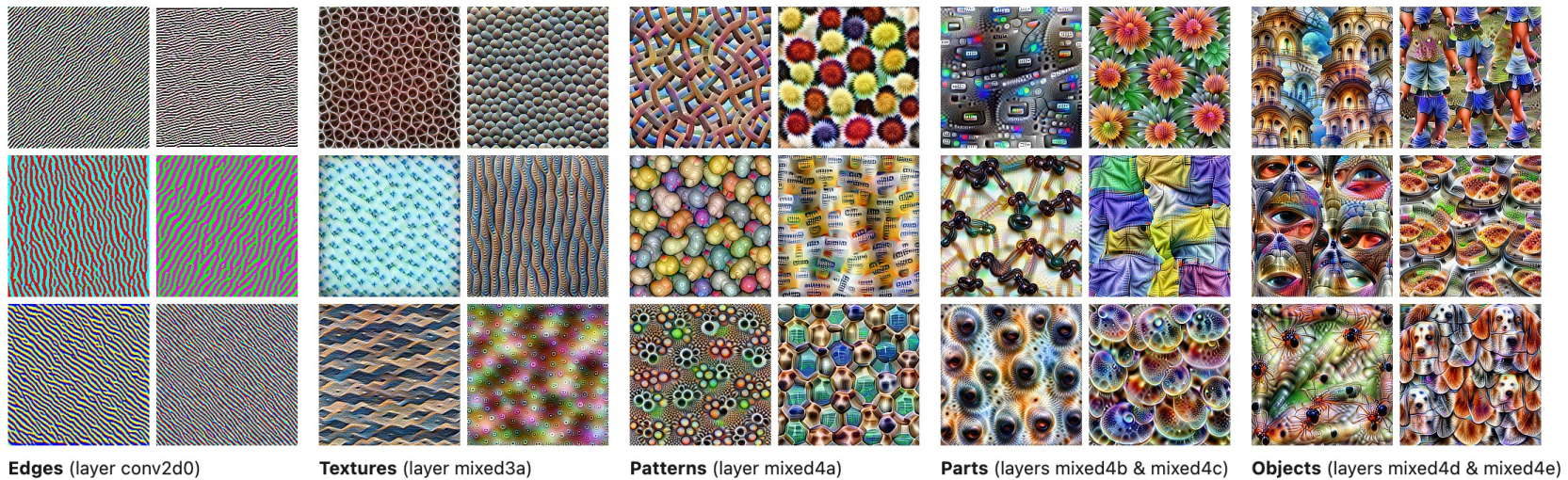
Couche **d'entrée** :

$$a_i = \sum_k w_{k,i} x^{(k)}$$
$$\frac{\partial a_i}{\partial x^{(k)}} = w_{k,i}$$



Les neurones des couches cachées : que représentent-ils ?

- Dans la descente de gradient par rétro-propagation, **symétrie** :
 - entre les **poids** des connexions
 - et les **exemples**
- On peut donc **optimiser sur les exemples** plutôt que sur les poids des connexions



<https://distill.pub/2017/feature-visualization/>

Apprentissage de représentations hiérarchiques

- Apprentissage de représentations hiérarchiques

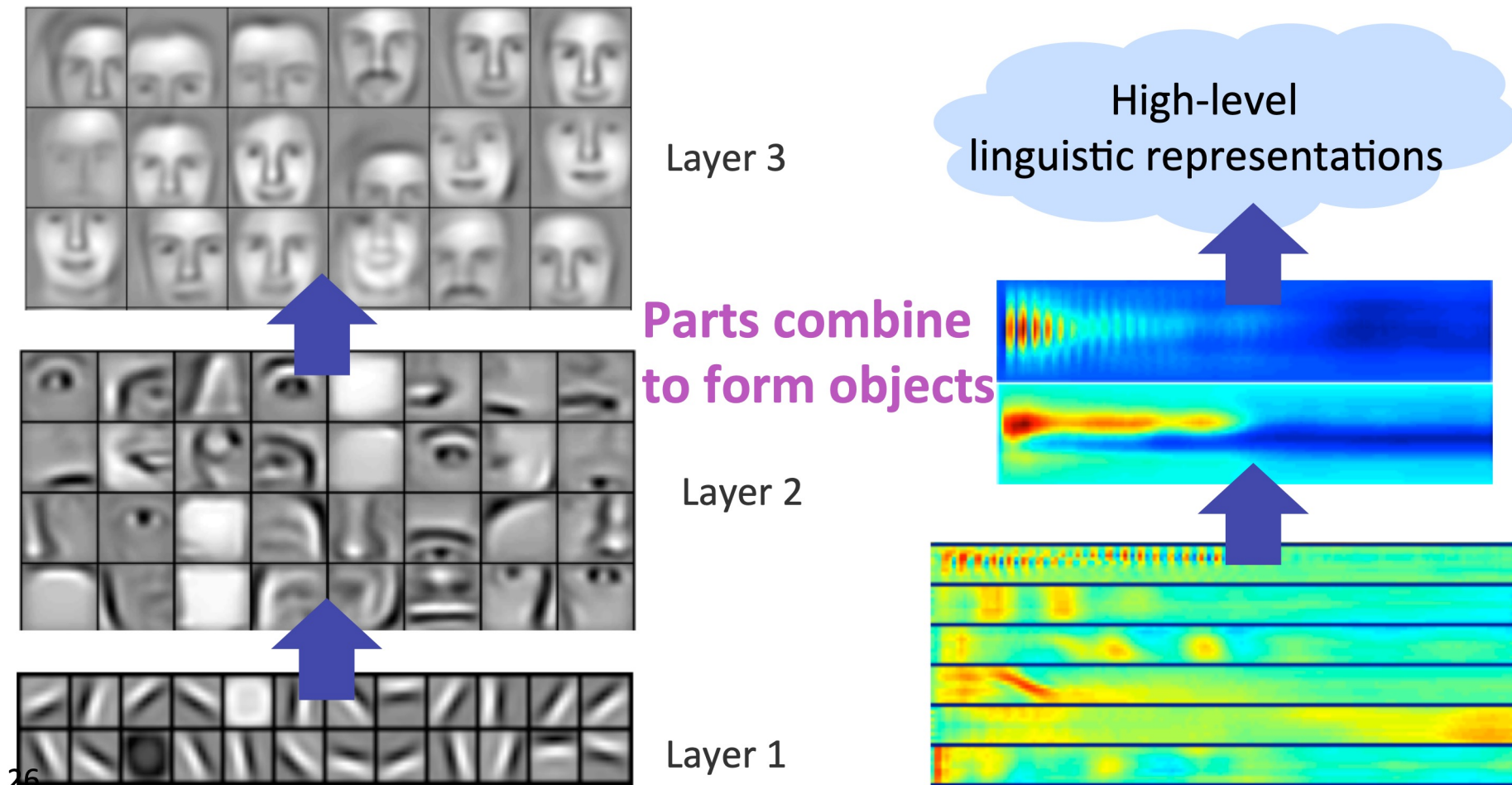


Illustration : ImageNet

La compétition ImageNet

- Plus de **15M d'images** haute résolution étiquetées
- Environ **22K catégories**
- Récoltées sur le Web et étiquetées par Amazon Mechanical Turk



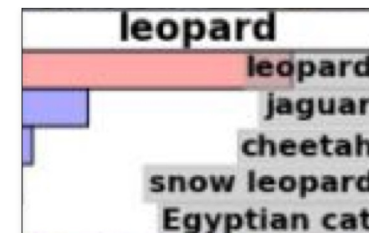
Illustration : ImageNet

La compétition ImageNet

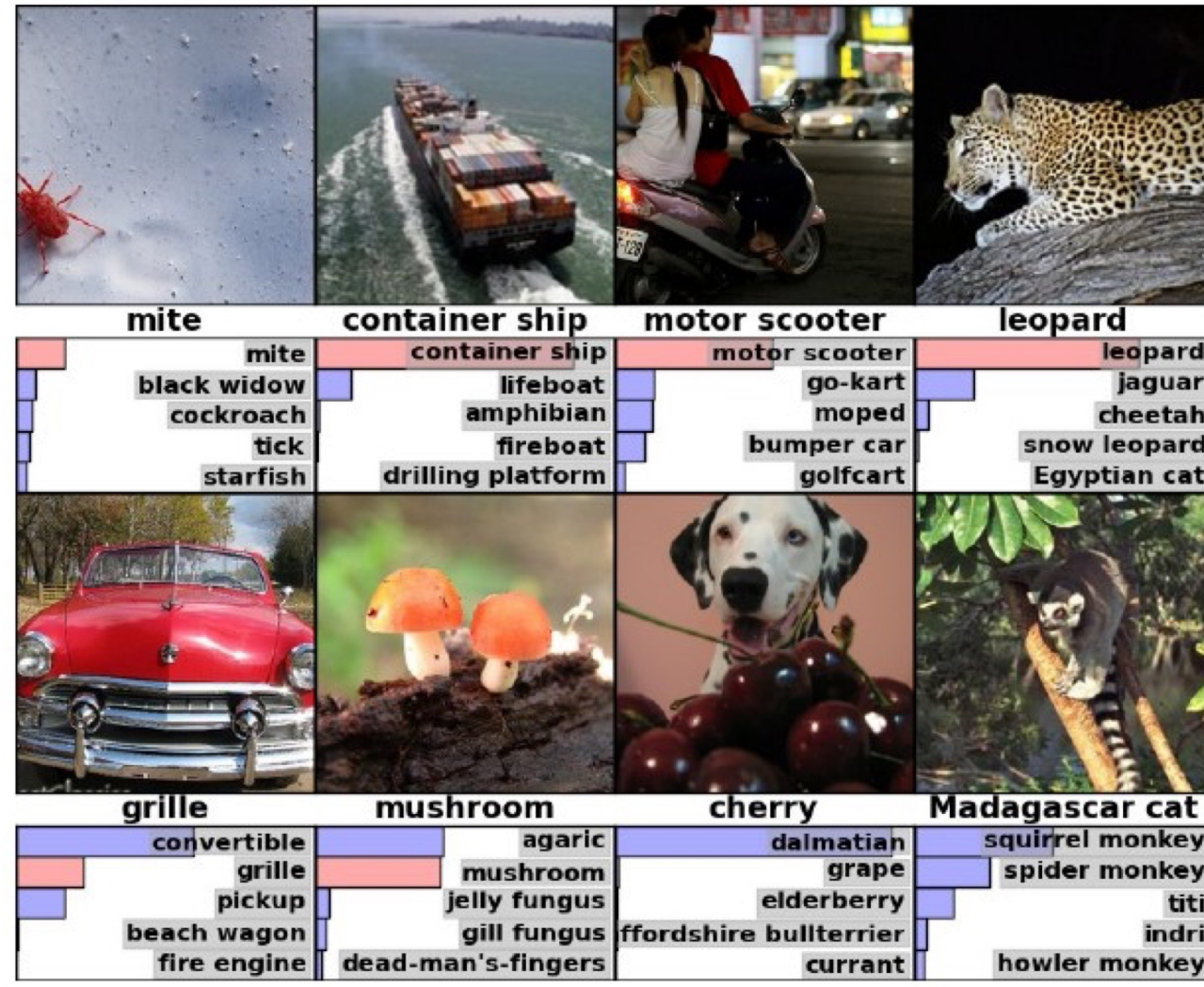
- Plus de **15M d'images** haute résolution étiquetées
- Environ **22K catégories**
- Récoltées sur le Web et étiquetées par Amazon Mechanical Turk



Classification



Object recognition



Il faut énormément d'exemples étiquetés

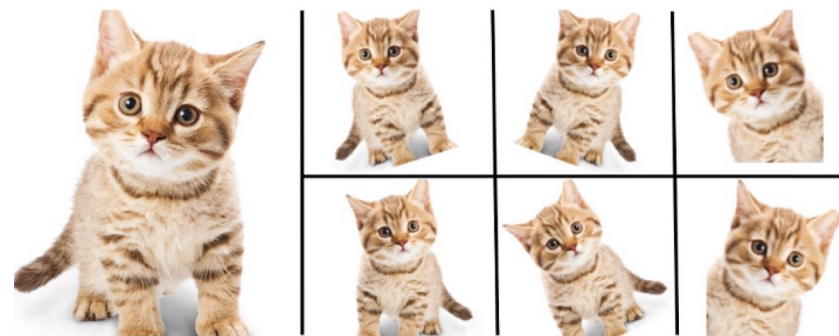
Comment faire ? ...

L' « augmentation » de données

Génération de **nouveaux exemples** à partir des exemples étiquetés disponibles

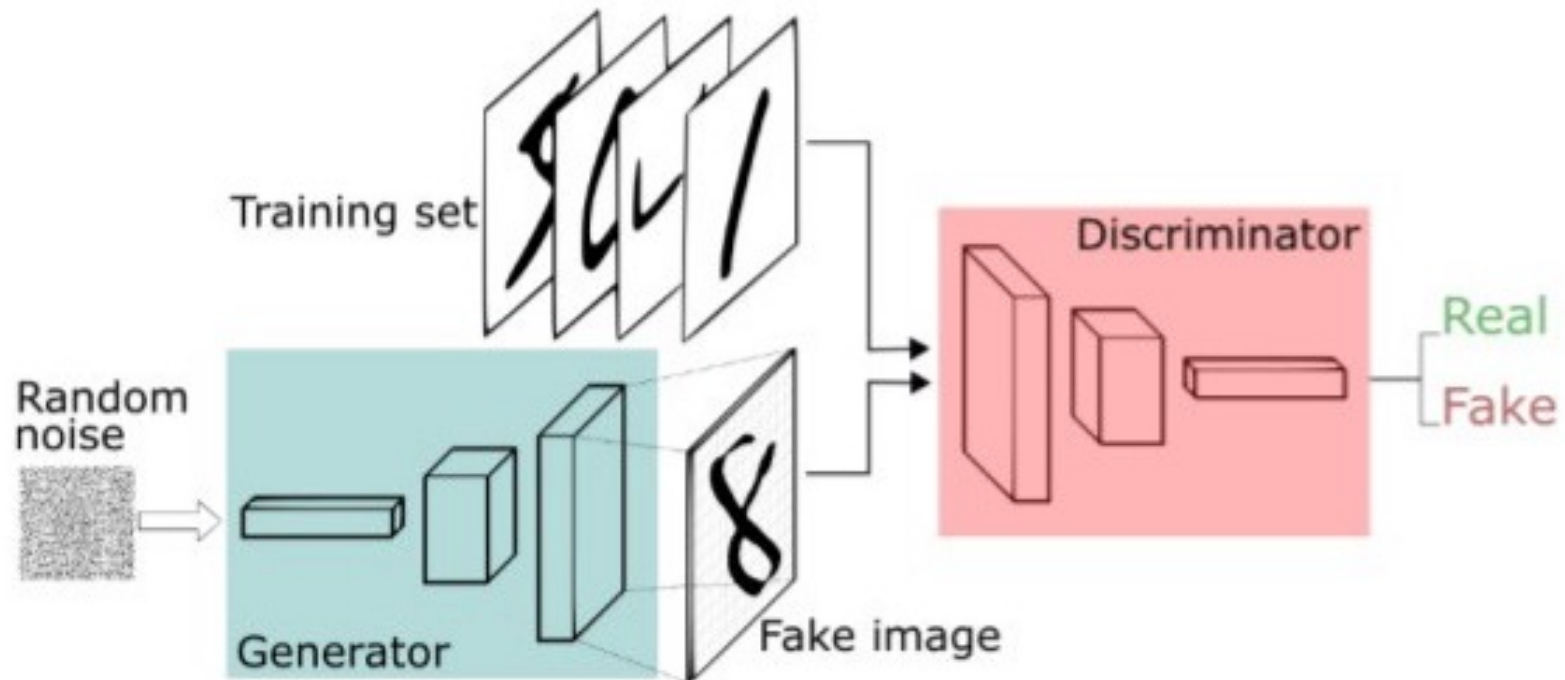
1. Transformation de données

- Bruitage / floutage / modification de contraste / changement de luminosité / autres effets spéciaux
- Rotation / dilatation / translation



L' « augmentation » de données

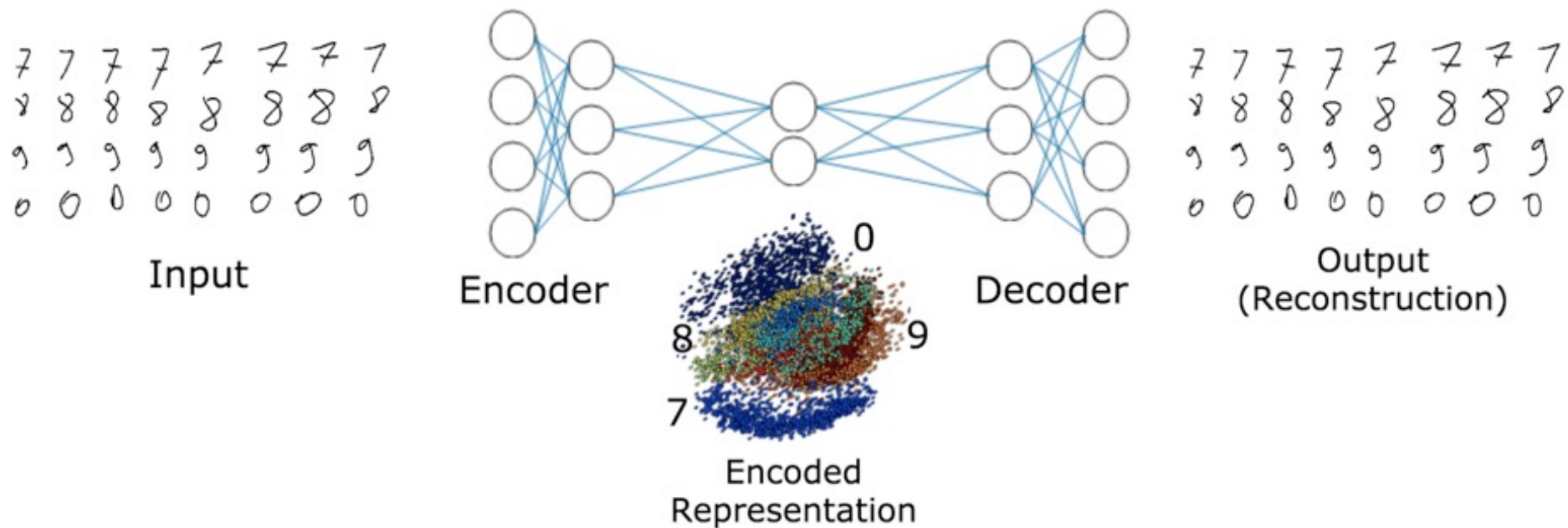
2. Génération par réseaux GANs (Generative Adversarial Networks)



Attention : Pas évident du tout à utiliser !

L' auto-encodage

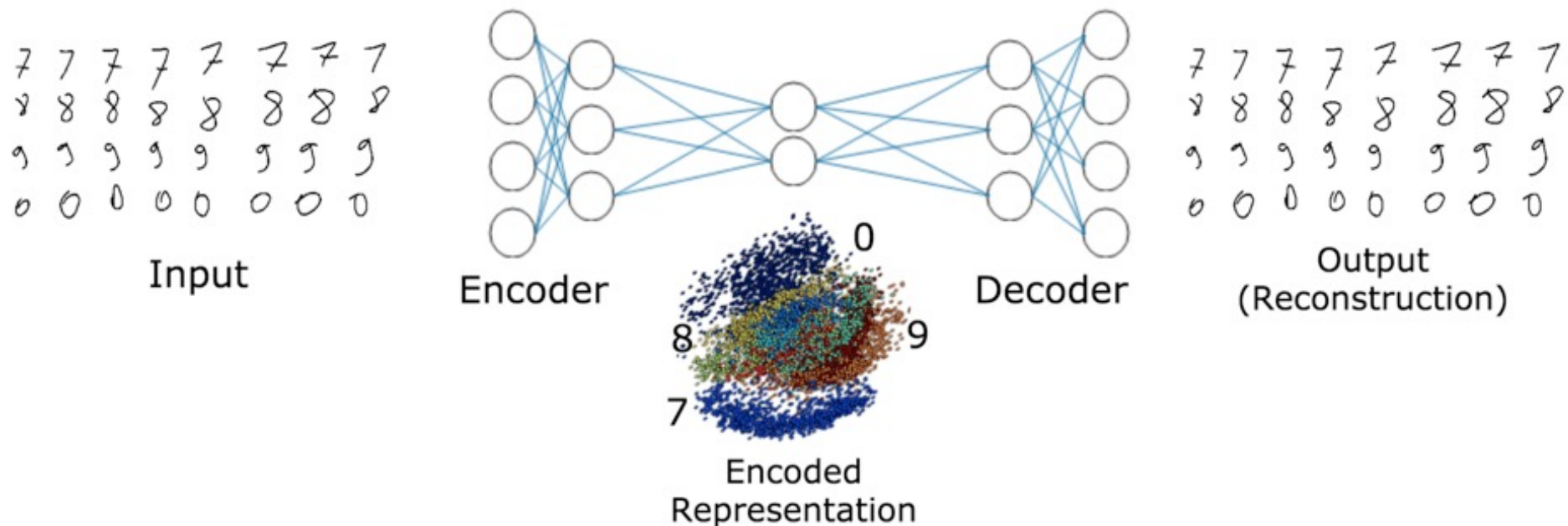
- Une vieille idée: l'auto-association



Représentation interne : l'*embedding*

L' auto-encodage

- Une vieille idée: l'auto-association

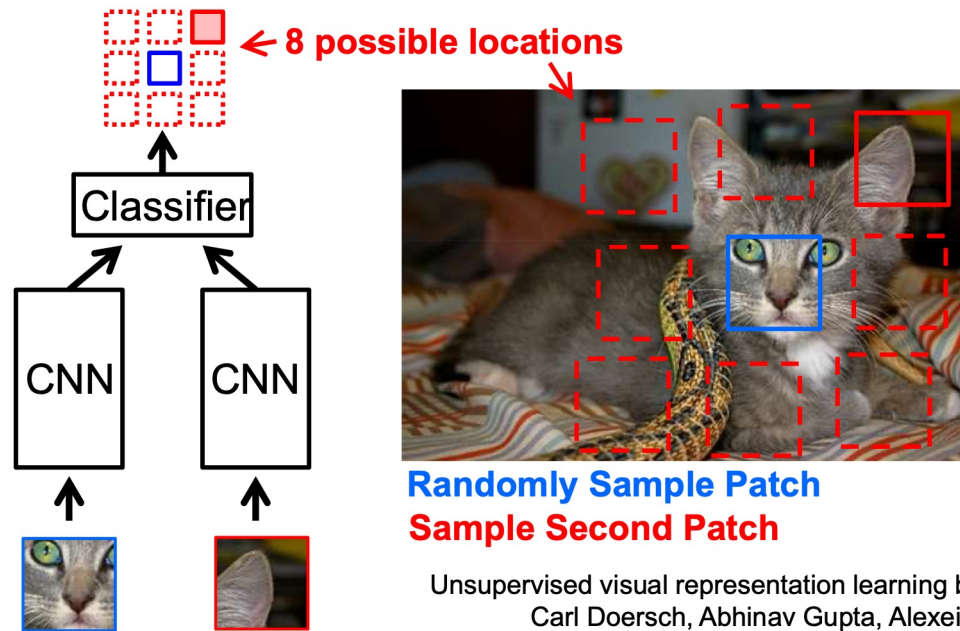


Un apprentissage supervisé ...
... sans étiquette !

L'auto-apprentissage (self-supervised learning)

- Une forme d'apprentissage **non supervisée** (pas d'étiquette nécessaire) mais qui permet un apprentissage **supervisé** !
 - Exemple : entraîner un réseau pour qu'il **prédise la position relative** de deux sous-images

On espère ainsi apprendre au réseau des **connaissances relationnelles** sur ce qui constitue une image



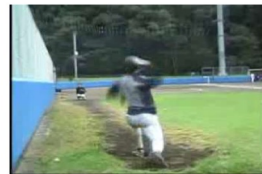
L'auto-apprentissage (self-supervised learning)

Dans des **séquences**

- Est-ce une séquence **valide** ?



Original video



Sun and Giles, 2001; Sun et al., 2001; Cleermans 1993; Reber 1989
Arrow of Time - Pickup et al., 2014

Slide credit: Ishan Misra

L'auto-apprentissage (self-supervised learning)

Temporally Correct order



Original video

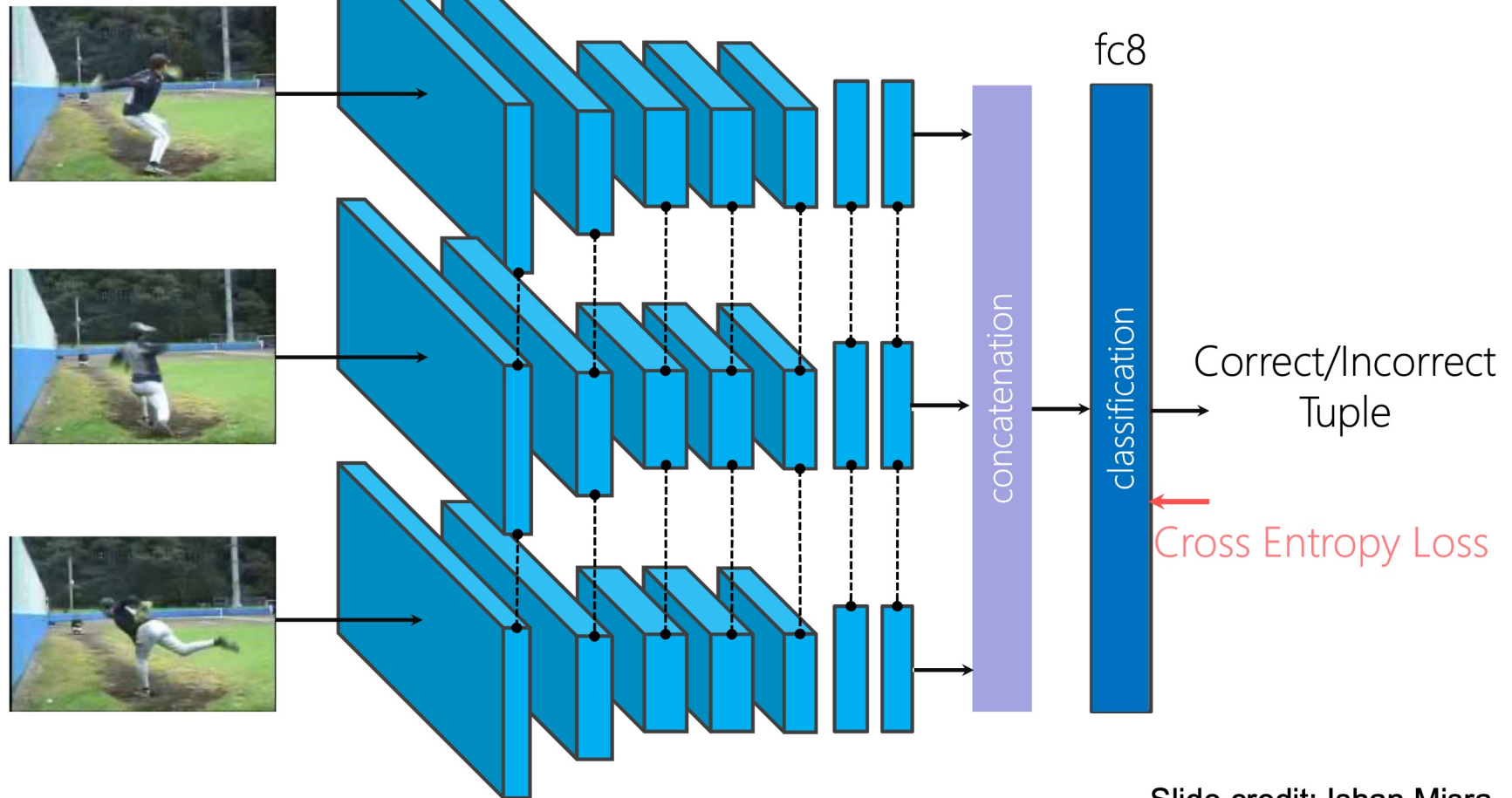


Temporally Incorrect order

Slide credit: Ishan Misra

L'auto-apprentissage (self-supervised learning)

Input Tuple

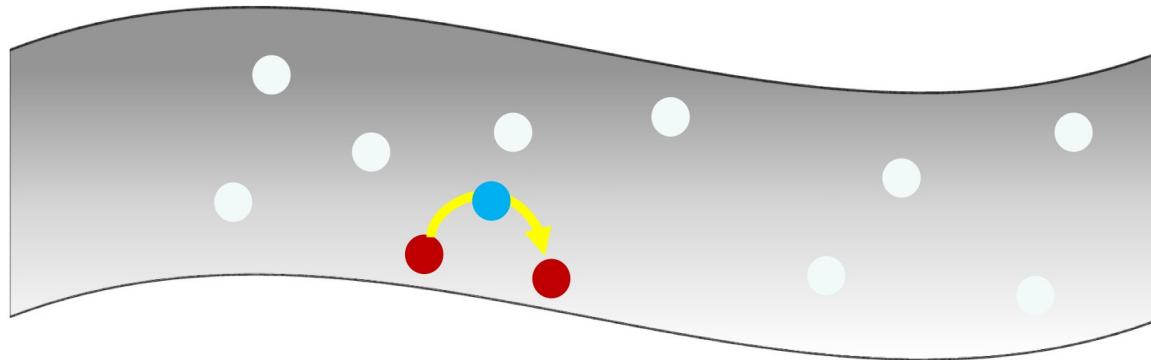


Slide credit: Ishan Misra

L'auto-apprentissage (self-supervised learning)

...

Images



Given a start and an end, can this point lie in between?

Shuffle and Learn – I. Misra, L. Zitnick, M. Hebert – ECCV 2016

Slide credit: Ishan Misra

Les couches de normalisation

Modifient la sortie des neurones en fonction de l'activité des autres neurones

- Batchnorm

$$x_{\text{out}}[i] = \gamma \frac{x_{\text{in}}[i] - \mathbb{E}[x_{\text{in}}[i]]}{\sqrt{\text{Var}[x_{\text{in}}[i]]}} + \beta$$

- L2-norm

$$x_{\text{out}}[i] = \frac{x_{\text{in}}[i]}{\|x_{\text{in}}\|_2}$$

- Layernorm

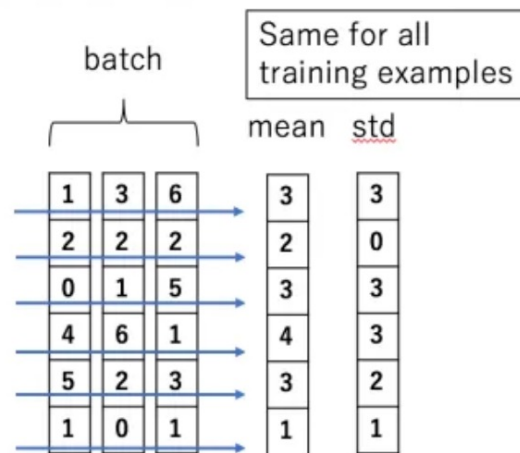
$$x_{\text{out}}[i] = \gamma \frac{x_{\text{in}}[i] - \mu}{\sigma} + \beta$$

$$\mu = \frac{1}{n} \sum_{i=1}^n x_{\text{in}}[i] \quad \sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_{\text{in}}[i] - \mu)^2$$

Les couches de normalisation

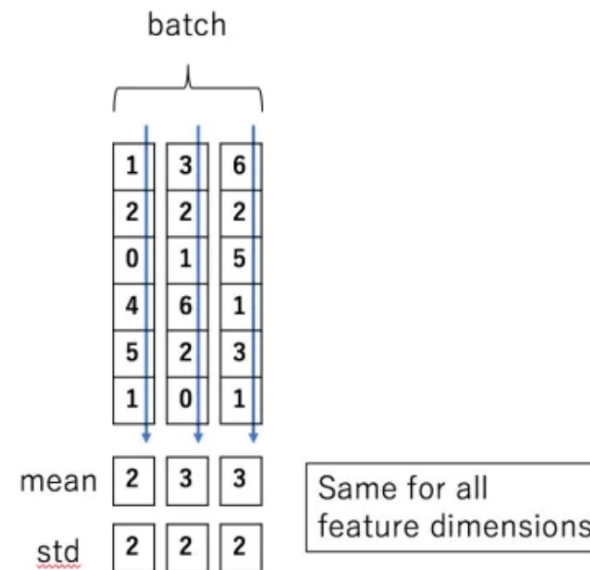
Modifient la sortie des neurones en fonction de **l'activité des autres neurones**

Batch Normalization



Normalisation pour chaque **descripteur** indépendamment

Layer Normalization



Normalisation pour chaque **exemple** indépendamment

Le « Drop Out »

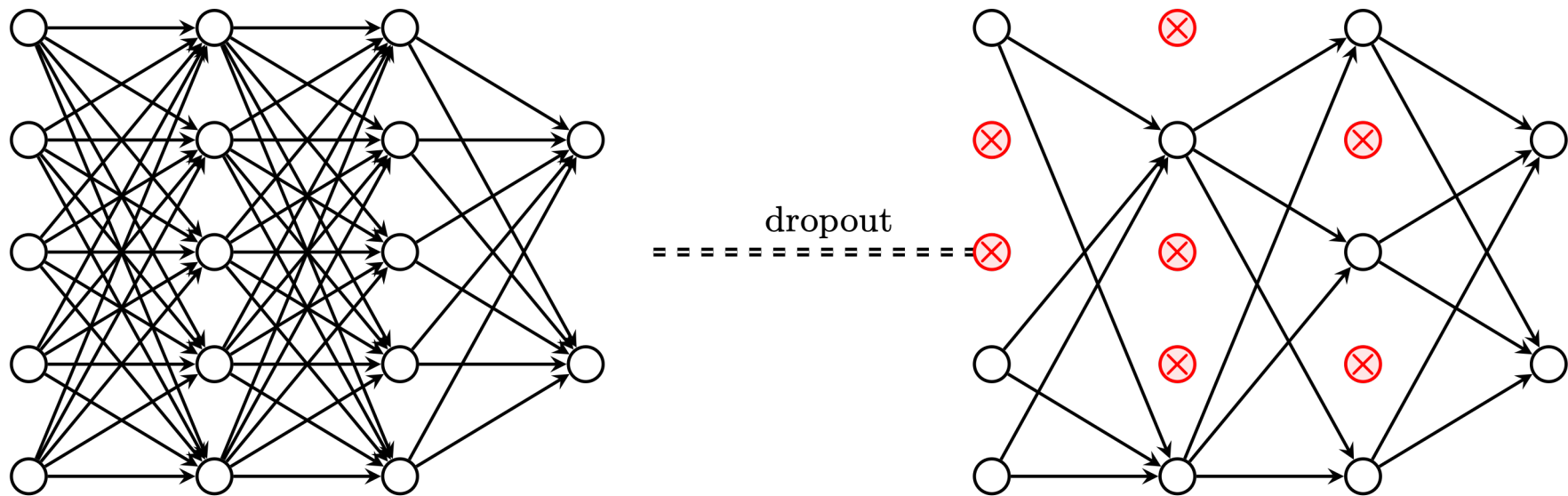


FIGURE 9: Schéma descriptif du dropout

...

Les comportements étranges

Sait-on pourquoi ça marche ... Quand ça marche

Quelque chose de troublant

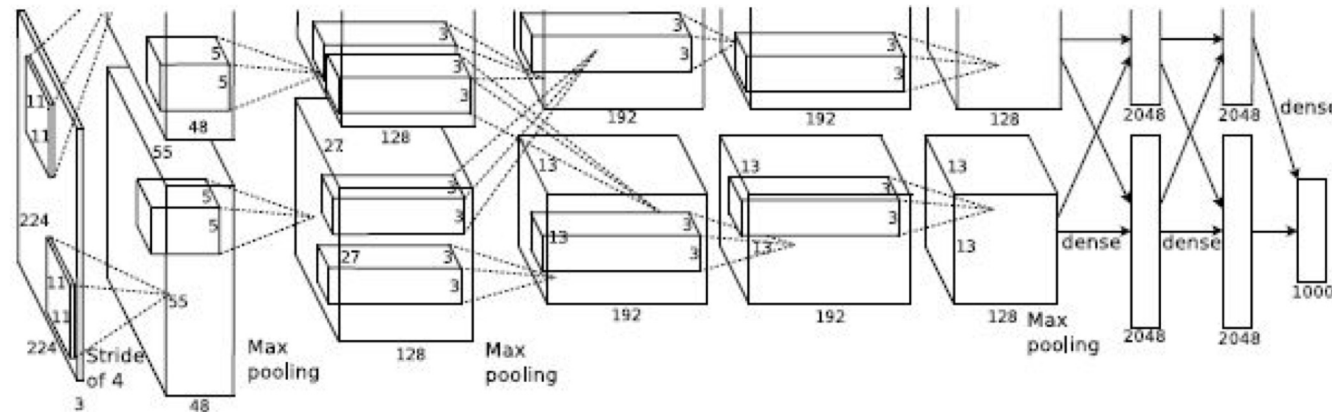
- C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals (ICLR, **May 2017**).
“Understanding deep learning requires rethinking generalization”

Quelque chose de troublant

- C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals (ICLR, **May 2017**).
“Understanding deep learning requires rethinking generalization”

Extensive experiments on the classification of images

- The AlexNet (> **1,000,000 parameters**) + 2 other architectures



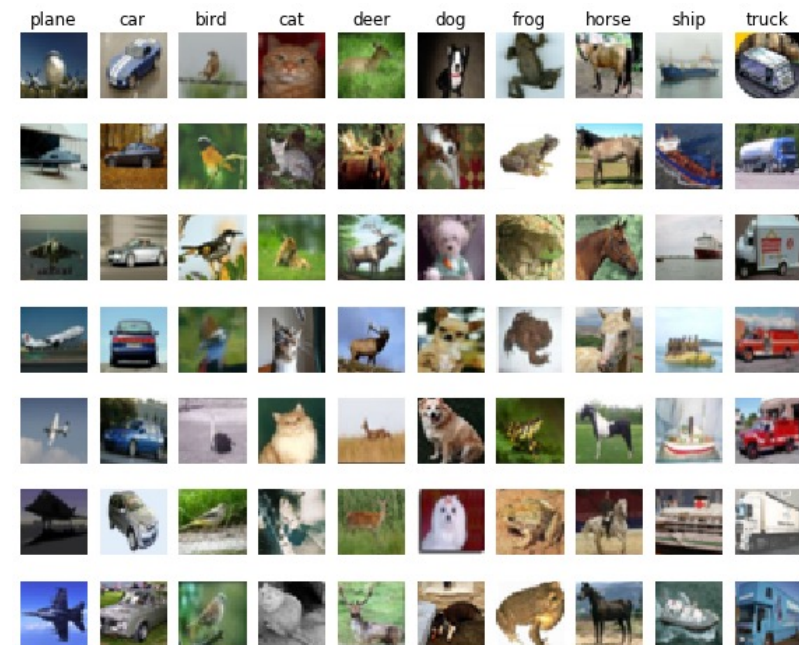
- The **CIFAR-10 data set**:
 - **60,000** images categorized in **10 classes** (50,000 for training and 10,000 for testing)
 - Images: 32x32 pixels in 3 color channels

Quelque chose de troublant

Experiments

1. Original dataset without modification

- Results ?
 - **Training** accuracy = **100%** ; **Test** accuracy = **89%**
 - Speed of convergence ~ 5,000 steps



Quelque chose de troublant

Experiments

1. Original dataset without modification

- Results ?
 - **Training** accuracy = 100% ; **Test** accuracy = 89%
 - Speed of convergence ~ 5,000 steps

Expected behavior if the **capacity** of the hypothesis space is **limited**

i.e. the system **cannot** fit any (arbitrary) training data

$$\forall h \in \mathcal{H}, \forall \delta \leq 1 : P^m \left[R(h) \leq \hat{R}(h) + 2 \widehat{Rad}_m(\mathcal{H}) + 3 \sqrt{\frac{\ln(2/\delta)}{m}} \right] > 1 - \delta$$

Troubling findings

Experiments

1. Original dataset without modification

- Results ?

- **Training** accuracy = 100% ; **Test** accuracy = 89%
- Speed of convergence ~ 5,000 steps

2. Random labels

!!!



- **Training** accuracy = 100% !!?? ; **Test** accuracy = 9.8%
- Speed of convergence = similar behavior (~ 10,000 steps)

Troubling findings

Experiments

1. Original dataset without modification

- Results ?

- **Training** accuracy = 100% ; **Test** accuracy = 89%
- Speed of convergence ~ 5,000 steps

2. Random labels

- **Training** accuracy = 100% !!?? ; **Test** accuracy = 9.8%
- Speed of convergence = similar behavior (~ 10,000 steps)

3. Random pixels

- **Training** accuracy = 100% !!?? ; **Test** accuracy ~ 10%
- Speed of convergence = similar behavior (~ 10,000 steps)

Now, we
are in
trouble!!

Troubling findings

- Deep NNs can accommodate ANY training set

Can grow without limit!!

$$\forall h \in \mathcal{H}, \forall \delta \leq 1 : P^m \left[R(h) \leq \hat{R}(h) + 2 \widehat{Rad}_m(\mathcal{H}) + 3 \sqrt{\frac{\ln(2/\delta)}{m}} \right] > 1 - \delta$$

But then,

why are deep NNs so good on image classification tasks?

Ce que l'on sait faire.

Sait-on d'ailleurs vraiment le faire ?

Ce qui interroge.

Ce qui reste à faire.

Un peu de recul :
**Que sait-on faire
et où sont les limites ?**

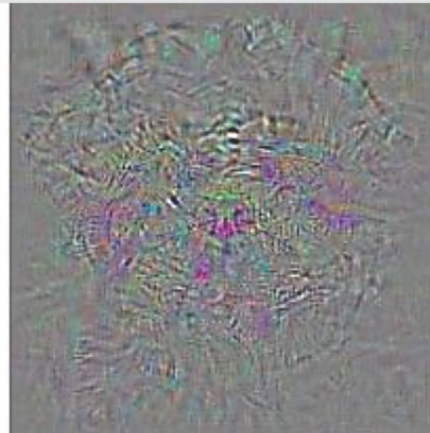
Ce que l'on sait faire

Quoique !?

Adversarial learning



School bus + tiny adversarial perturbation = "ostrich"



Dog + tiny adversarial perturbation = "ostrich"

Adversarial input can fool a machine-learning algorithm into misperceiving images.

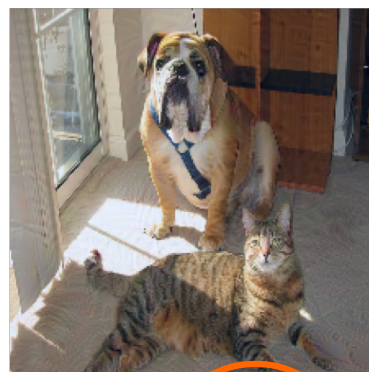
Explanations and deep neural networks

Optical illusions: how to explain them?



Boxer: 0.40 Tiger Cat: 0.18

(a) Original image



Airliner 0.9999

(b) Adversarial image

!!??

[Selvaraju et al. (2017) « *Grad-CAM: Visual explanations from deep networks via gradient-based localization* »]

Annotation d'images



Figure 2.11: “A group of young people playing a game of frisbee”—that caption was written by a computer with no understanding of people, games or frisbees.

Automated image-captioning

- Not always so good!



A dog is jumping to catch a frisbee

Exemple en médecine

MACHINE LEARNING

Adversarial attacks on medical machine learning

Emerging vulnerabilities demand new conversations

22 March 2019

Science

The anatomy of an adversarial attack

Demonstration of how adversarial attacks against various medical AI systems might be executed without requiring any overtly fraudulent misrepresentation of the data.

Original image



Dermatoscopic image of a benign melanocytic nevus, along with the diagnostic probability computed by a deep neural network.



Model confidence

Diagnosis: Benign

The patient has a history of back pain and chronic alcohol abuse and more recently has been seen in several...

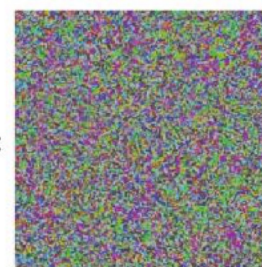
Opioid abuse risk: High

277.7 Metabolic syndrome
429.9 Heart disease, unspecified
278.00 Obesity, unspecified

Reimbursement: Denied

+ 0.04 ×

Adversarial noise



Perturbation computed by a common adversarial attack technique. See (7) for details.

=

Adversarial example



Combined image of nevus and attack perturbation and the diagnostic probabilities from the same deep neural network.



Model confidence

Diagnosis: Malignant

The patient has a history of lumbago and chronic alcohol dependence and more recently has been seen in several...

Opioid abuse risk: Low

401.0 Benign essential hypertension
272.0 Hypercholesterolemia
272.2 Hyperglyceridemia
429.9 Heart disease, unspecified
278.00 Obesity, unspecified

Reimbursement: Approved

Adversarial rotation (8)

Adversarial text substitution (9)

Adversarial coding (13)

Voiture dans une piscine

- ... ou **pas de voiture** ... ?



Is this less of a car
because the context is wrong?

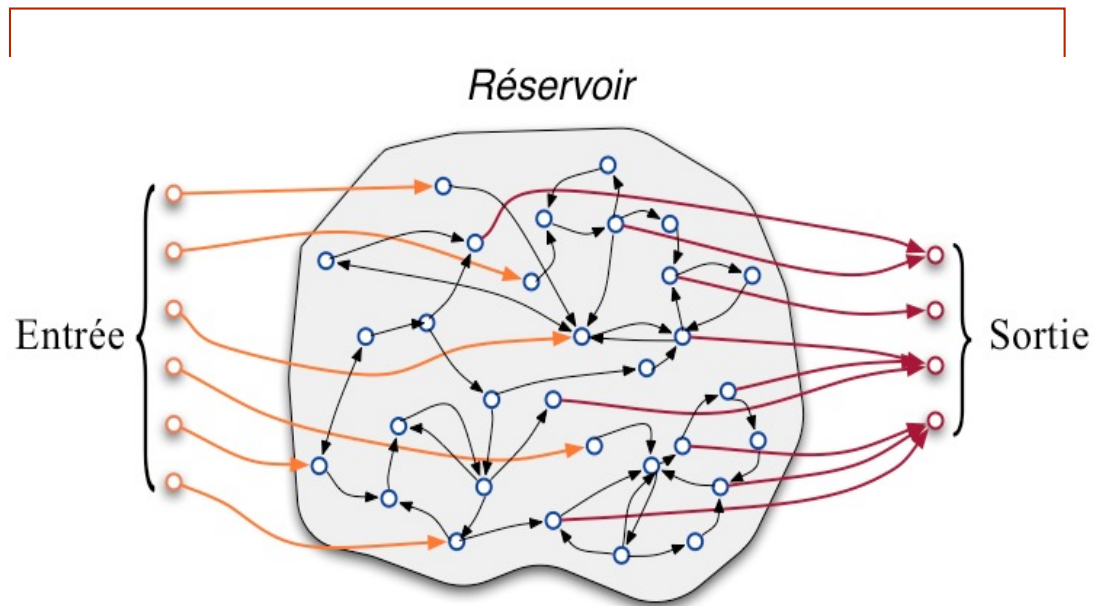
[Léon Bottou (ICML-2015, invited talk) « *Two big challenges in Machine Learning* »]

Plan

1. Introduction et réseaux de neurones à une couche
2. Les perceptrons multi-couches
3. L'apprentissage dans les réseaux de neurones
4. Une introduction aux réseaux de neurones profonds
5. Autres architectures

Une idée intrigante : le « reservoir computing »

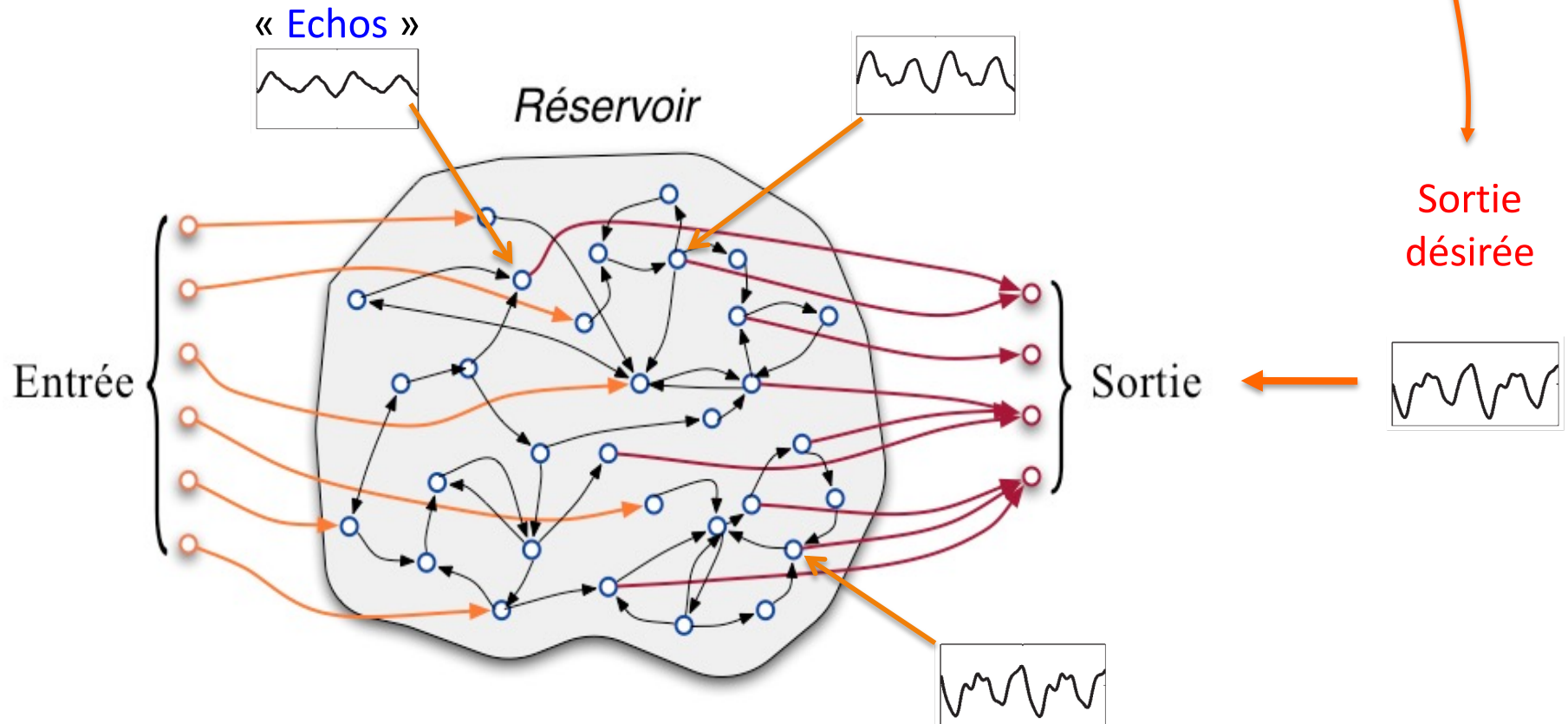
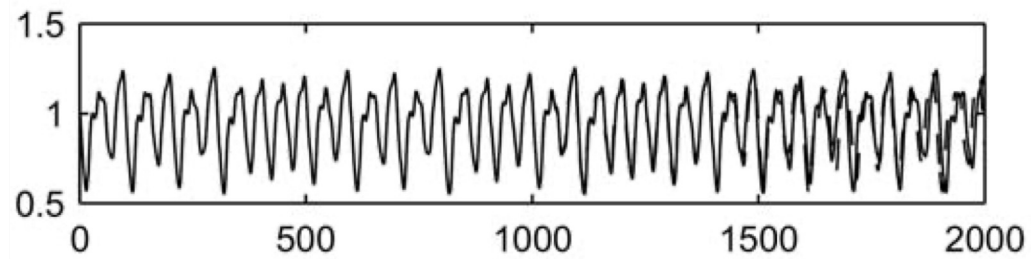
- Idée :
 - Utiliser un réseau récurrent sans l'entraîner explicitement
 - Entraîner une seule couche de sortie



- Ré-introduit une « dynamique »
 - Séries temporelles

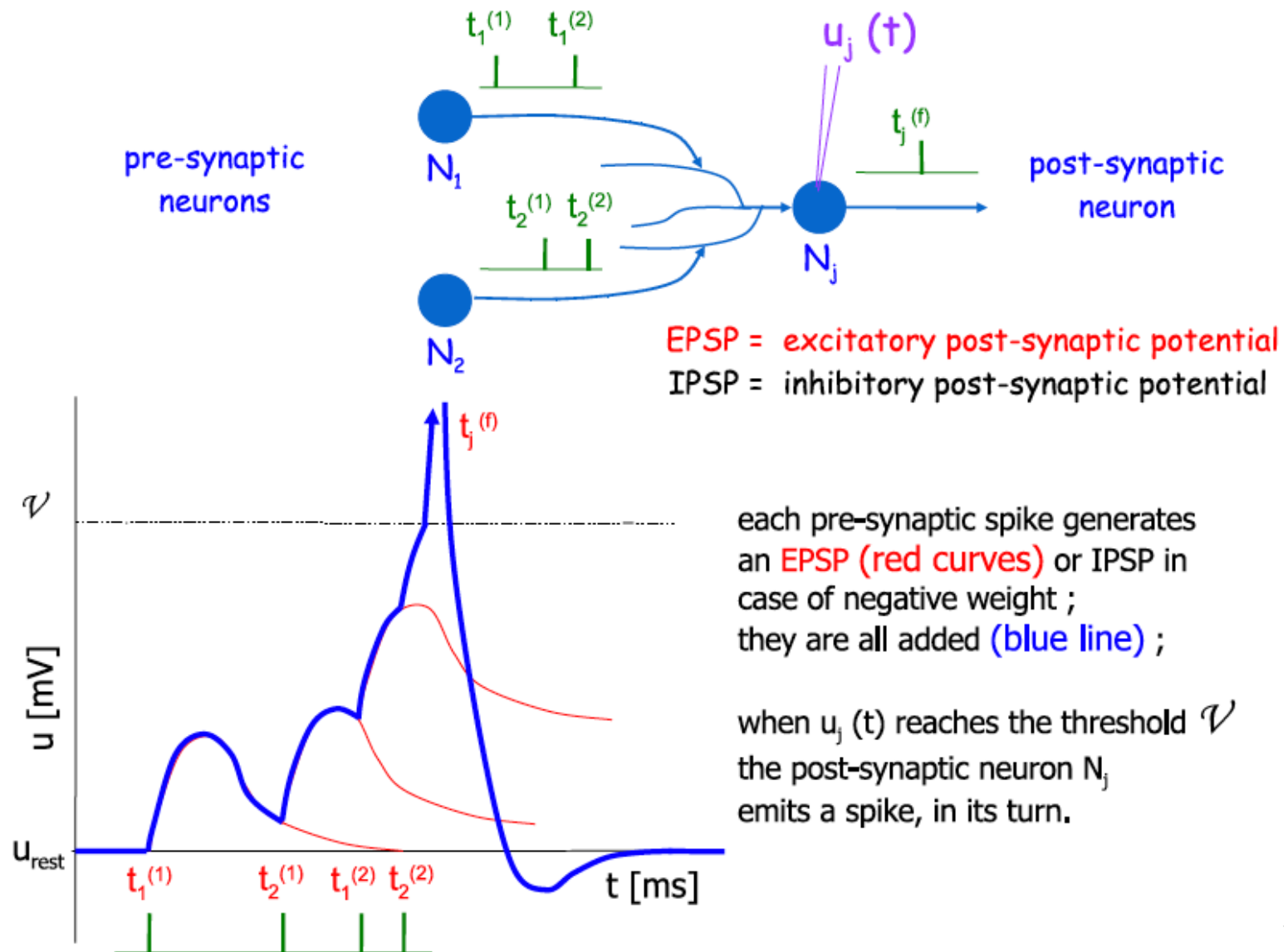
Prédiction : « reservoir computing »

Signal à « apprendre »



Les réseaux à neurones impulsionnels

- *Spiking neural networks*



A retenir

1. Méthode d'induction supervisée très versatile

- Très grand champ d'application. Famille de modèles très souple.
- Certainement pas un modèle biologique

2. Apprentissage demandant du soin

- Choix de l'architecture
- Apprentissage lent, sujet à minima locaux

3. « Opacité »

- Difficile d'interpréter les réseaux appris
- Difficile de mettre de la connaissance *a priori*

Références bibliographiques

- Vincent Barra, Antoine CORNUÉJOLS et Laurent MICLET
Apprentissage artificiel. Concepts et algorithmes. De Bayes et Hume au Deep Learning
Eyrolles, 2021
- Christopher Bishop & Hugh Bishop
Deep Learning: Foundations and Concepts
Springer, 2023
- Simon HAYKIN
Neural Networks. A comprehensive foundation
Prentice Hall, 1999
- Sebastian RASCHKA, Yuxi LIU & Vahid MIRJALILI
Machine Learning with PyTorch and Scikit-Learn
Packt>, 2022
- Antonio TORRALBA, Philip ISOLA & William FREEMAN
Foundations of Computer Vision
MIT Press, 2024.

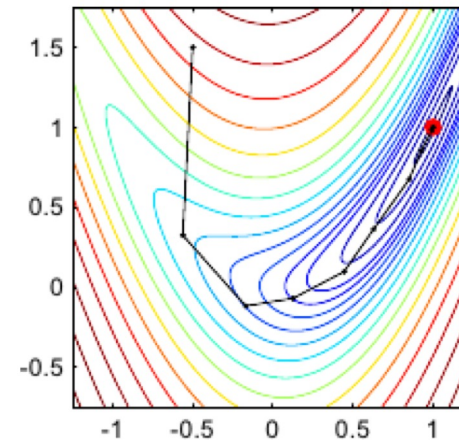
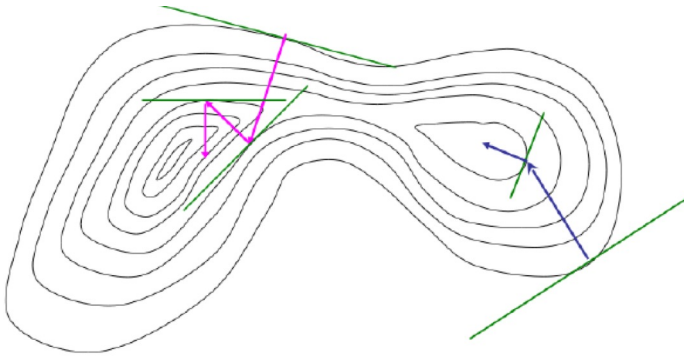
Questions pratiques

Questions

- « Batch » ou « en-ligne » ?
- Réglage du pas d'apprentissage
 - Fixe
 - Variable : méthode du moment
 - Méthodes de 2^{ème} ordre : calcul du Hessien, ...
- Critère d'arrêt ?
- La « calibration » de la sortie
- Codage de la couche de sortie ?
- Quelle architecture ?

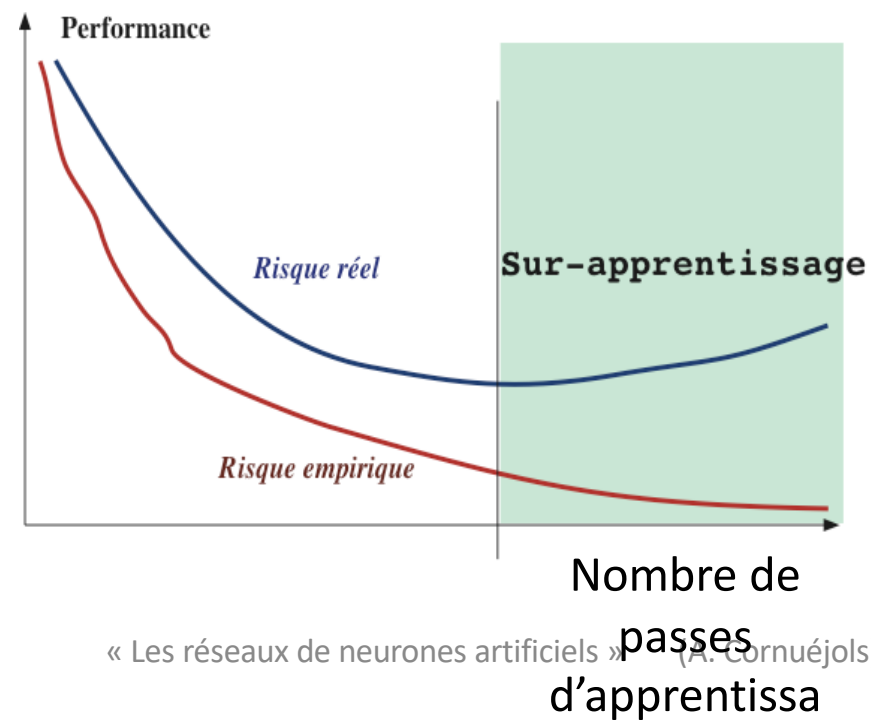
Réglage du pas d'apprentissage

- Techniques
 - Ajout d'un moment
 - Méthode de 2^{ème} ordre



Critère d'arrêt

- Nombre d'« époques » déterminé
- À l'arrêt de la baisse de l'erreur empirique
- Règle d'arrêt prématuré (« *early stopping rule* »)
 - À l'arrêt de la baisse de l'erreur sur l'ensemble de validation



Prétraitement des exemples

1. **Transformation** des variables d'un type à l'autre
2. Traitement du **bruit** dans les données
3. Traitement des **valeurs manquantes**
4. **Normalisation** des valeurs des attributs
5. **Réduction** de l'espace de description
6. Données **mal équilibrées** entre classes
7. Les **points aberrants**

Transformation des variables d'un type à l'autre

- **Catégoriel vers numérique** ou booléen
 - Certains systèmes d'apprentissage demandent des entrées numériques
 - Le type numérique induit une relation d'ordre entre ses éléments
 - Le type catégoriel est souvent dénué d'une telle relation d'ordre
 - E.g. marque de voiture (Renault, Peugeot, Fiat, ...)
 - Une solution : le *one-hot-encoding*

Couleur de carte	Trèfle	Carreau	Coeur	Pique
Carreau	0	1	0	0
Pique	0	0	0	1
Coeur	0	0	1	0
Coeur	0	0	1	0
Trèfle	1	0	0	0
Pique	0	0	0	1
Trèfle	1	0	0	0
Carreau	0	1	0	0

« Les réseaux de neurones artificiels » (A. Cornuéjols)

Types de données

- **Nominales**

- *Sexe, profession, ...*
- Nombre de valeurs dénombrable
- Aucune relation d'ordre
- Opérateurs arithmétiques inapplicables

- **Ordinales**

- *Taille (petite, moyenne, grande)*
- Des modalités
- Relation d'ordre entre modalités
- Calculs sur des rangs

- **Numériques** ou **continues**

- *Age, poids*
- Nombre de valeurs théoriquement infini
- Relation d'ordre entre les valeurs
- Notion de distance et d'écart
- Calculs arithmétiques possibles

Transformation des variables d'un type à l'autre

- **Discrétisation** d'un **attribut numérique**
 - Certains algorithmes d'apprentissage sont incapables de traiter directement des attributs à valeur continue. Il est nécessaire de les transformer en attributs à valeur discrète.
 - **Nombreuses méthodes** de discrétisation
 - Intervalles **égaux** (e.g. [0, 10], [11, 20], [21, 30], ...)
 - Intervalles de **même fréquence**
 - ...

« Bruit » dans les données

- Bruit **aléatoire**
- Bruit fonction de la **classe**
- Bruit fonction de la **source**
- Données **fiables** vs. **Non fiables**
- Données **censurées**
- ...

Traitement des valeurs manquantes

- Illustrations
 - Tous les examens médicaux possibles n'ont pas été réalisés
 - On n'a pas osé demander l'âge de la personne
 - Les acheteurs n'achètent qu'une toute petite fraction des produits en vente
- De nombreux systèmes d'apprentissage ne peuvent traiter des données incomplètes

Valeurs manquantes

Vache	Sexe	Age	Conso. 1 ^{er} jour	Conso 2 ^{ème} jour	Poids
Bérénice	F	3	3.56 kg	3.87 kg	630 kg
Marguerite	?	5	4.78 kg	?	?
Blanchette	F	?	?	5.72 kg	435 kg
Furie	M	6	6.02 kg	?	875 kg

- Valeurs souvent indispensables
- Comment faire ?

Traitement des **valeurs manquantes**

- Techniques variées d'**imputation** des valeurs manquantes
 - Remplacement par la **valeur moyenne**
 - Remplacement par la **valeur la plus fréquente**
 - Remplacement par la **valeur moyenne** de la **classe considérée**
 - Imputation par une **fonction de prédiction**
 - Suppose une interdépendance entre les variables
 - Cf. aussi les systèmes de recommandation
 - Pose la question de l'utilité de la variable considérée

Normalisation

- Cas de **domaines de variations très différents**
 - Fausse les régularités trouvées
 - Fréquemment on utilise une *normalisation centrée réduite*
 - **Centrée** : on ramène tous les domaines de variations autour de 0
 - **Réduite** : on divise les valeurs par l'écart-type
 - Toutes les dimensions dans $[0, 1]$
 - Discrétisation en un même nombre de valeurs
- Attention : ce n'est **pas forcément une bonne idée**

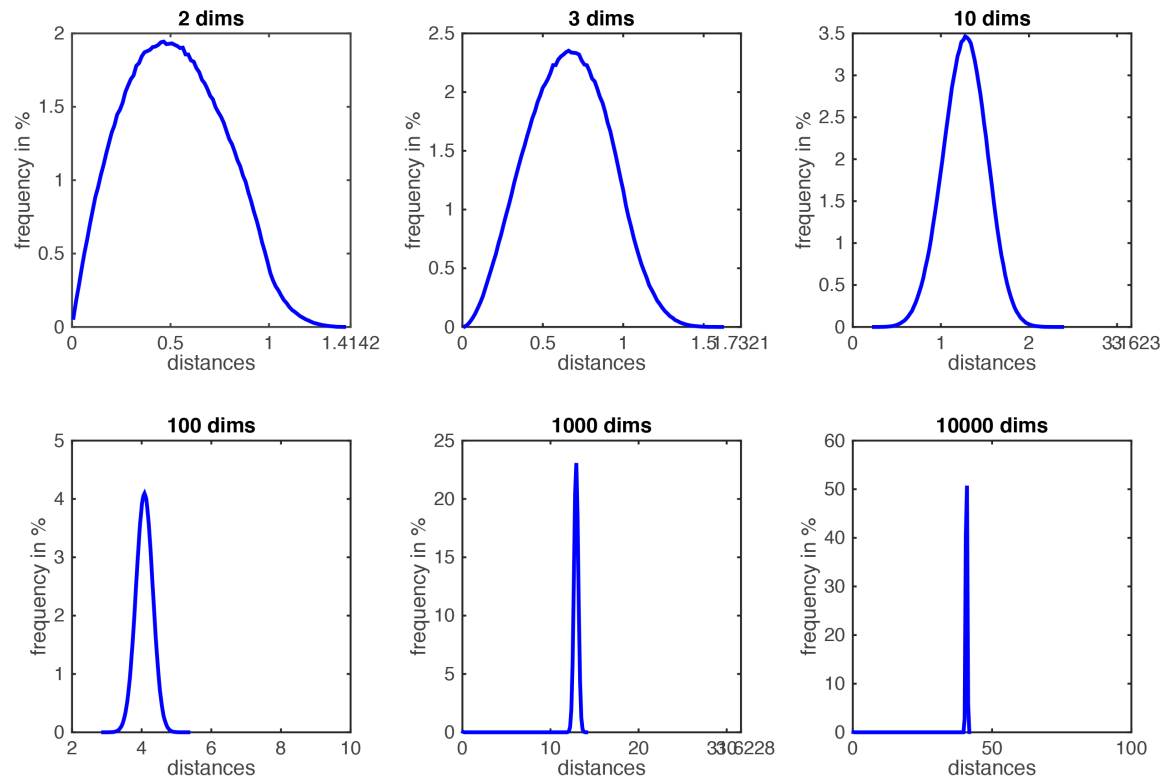
Réduction de l'espace d'entrée

- Attributs **redondants**
- Attributs **non pertinents**
- Attributs **peu informatifs**

Réduction de l'espace de description

- La **malédiction de la dimensionnalité**

- En stat : il faut un **nombre exponentiel de données** (en la dimension de l'espace) pour approcher une distribution
- En apprentissage fondé sur les distances : un **phénomène de concentration**



Distributions des distances
de paires de points tirées
aléatoirement dans le cube
unité en fonction de **d**

Réduction de l'espace de description

Deux grandes classes de méthodes

1. Sélection de variables

- Le plus souvent en contexte d'apprentissage supervisé
- Exemples
 - ANOVA
 - RELIEF
 - Méthodes de régularisation : Ridge, LASSO, Elastic Net, ...

2. Changement d'espace de représentation

- **Analyses en composantes** : principales (ACP), indépendantes (ACI), composantes non négatives (NMF), Analyse sémantique latente (LSA), ...
- **Manifold learning** : ISOMAP, tSNE

Sélection de variables

1. Méthodes **intégrées** (*embedded*)

- La méthode d'apprentissage **sélectionne** les descripteurs **par elle même**
- E.g. Arbres de décision

2. Méthodes « **symbiose** » (*wrapper*)

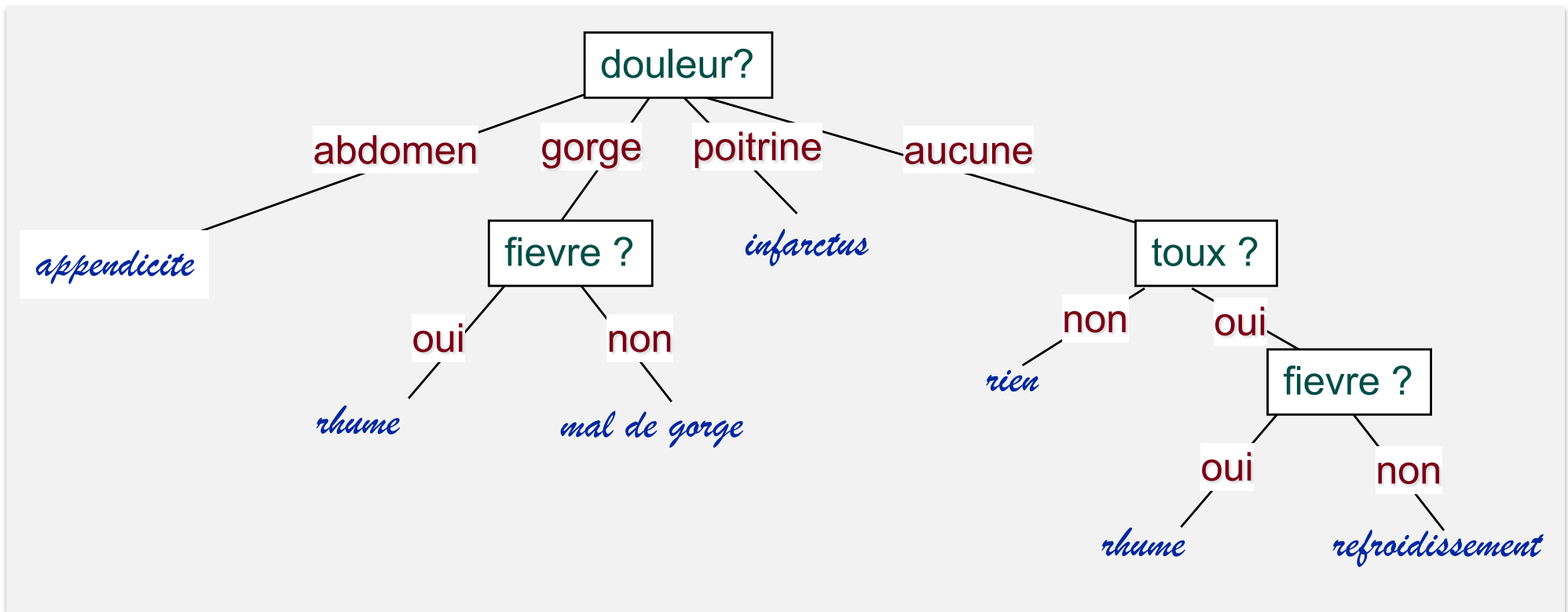
- **Exploration des sous-ensembles de descripteurs** en observant la performance d'une méthode de classification

3. Méthodes de **filtre** (*filter*)

- Sélection des variables **indépendamment** les unes des autres
- ANOVA, RELIEF, ...

Méthodes « intégrées »

- L'apprentissage conduit à distinguer les attributs pertinents
 - Ex : arbres de décision



Méthode « symbiose » (*wrapper*)

- On utilise un algorithme d'apprentissage pour **évaluer la pertinence de l'ensemble d'attributs** sélectionné
 - **Avantage :** on optimise bien ce que l'on veut optimiser
 - **Inconvénient :** espace de recherche gigantesque
 - Il faut avoir recours à une exploration heuristique
 - Ascendante (« *forward selection* »)
 - Descendante (« *backward selection* »)
 - Coûteux

Classification : Méthodes de filtres

- Beaucoup de techniques
 - **ANOVA** (ANalysis Of VAriance)
 - Comparer la variabilité intra-classe à la variabilité inter-classe
 - Méthode paramétrique : suppose des lois normales

$$S_{int}^2 = \frac{SC_{int}}{\# \text{ degrés de liberté}} = \frac{\sum_{i=1}^k \sum_{j=1}^{n_i} (X_{ij} - \bar{X}_i)^2}{n - k}$$

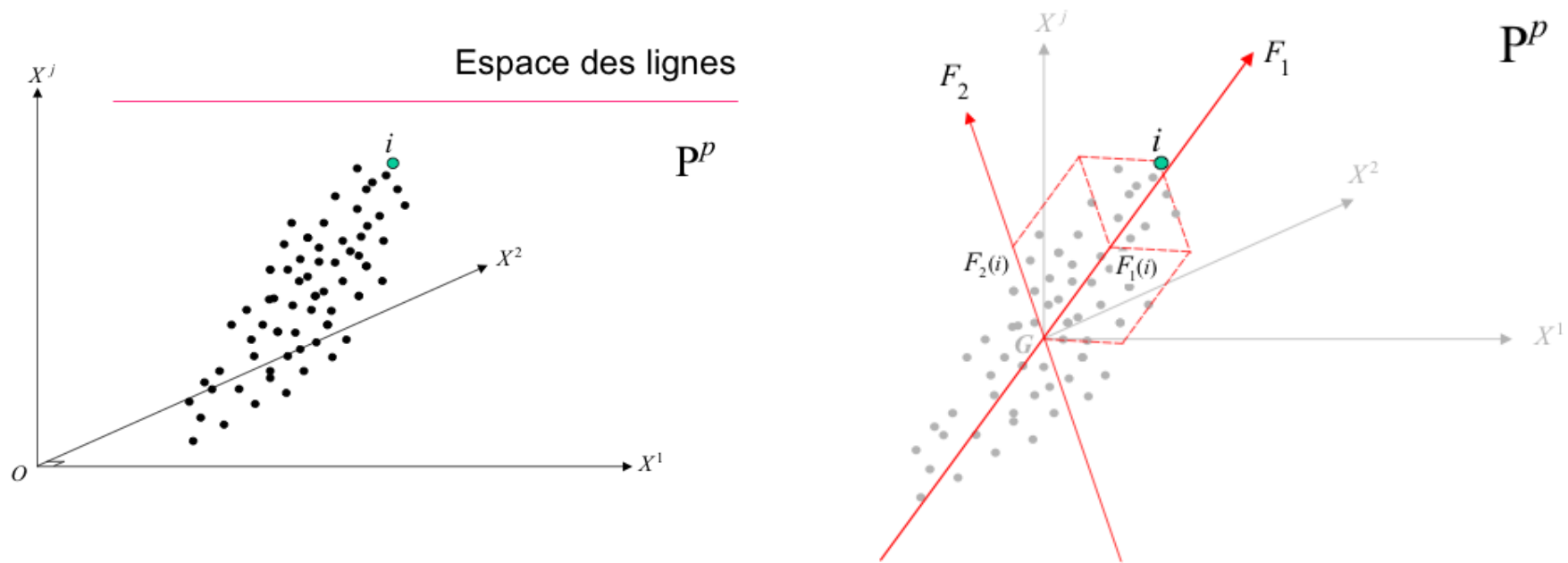
$$S_{ent}^2 = \frac{\sum_{i=1}^k n_i (\bar{X}_i - \bar{X})^2}{k - 1}$$

Plus S_{ent} grand par rapport à S_{int} , moins la variance peut-être expliquée par la variabilité de l'échantillonnage et plus c'est indicatif d'un effet classe.

- **RELIEF**

- Principe sous-jacent similaire : mesurer en quoi la connaissance de l'attribut apporte une information sur la classe des exemples
- Méthode non paramétrique

Analyse en Composantes Principales (ACP)



- Méthode linéaire
- Optimisant un critère quadratique
 - Très sensible aux points aberrants

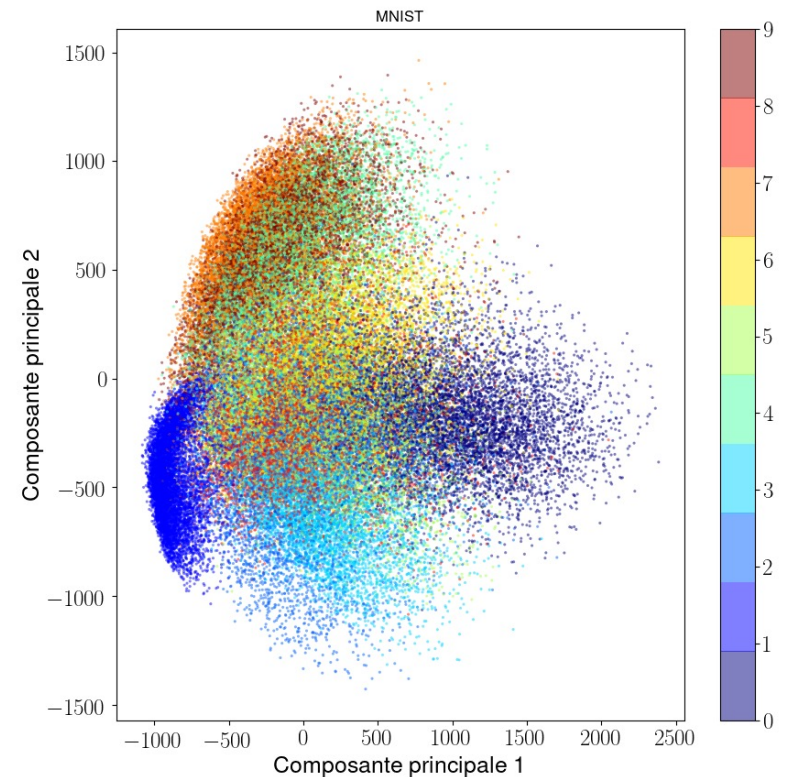
Analyse en composantes principales

- Identifier les axes principaux d'inertie
- Limite : méthode linéaire



$d = 584$

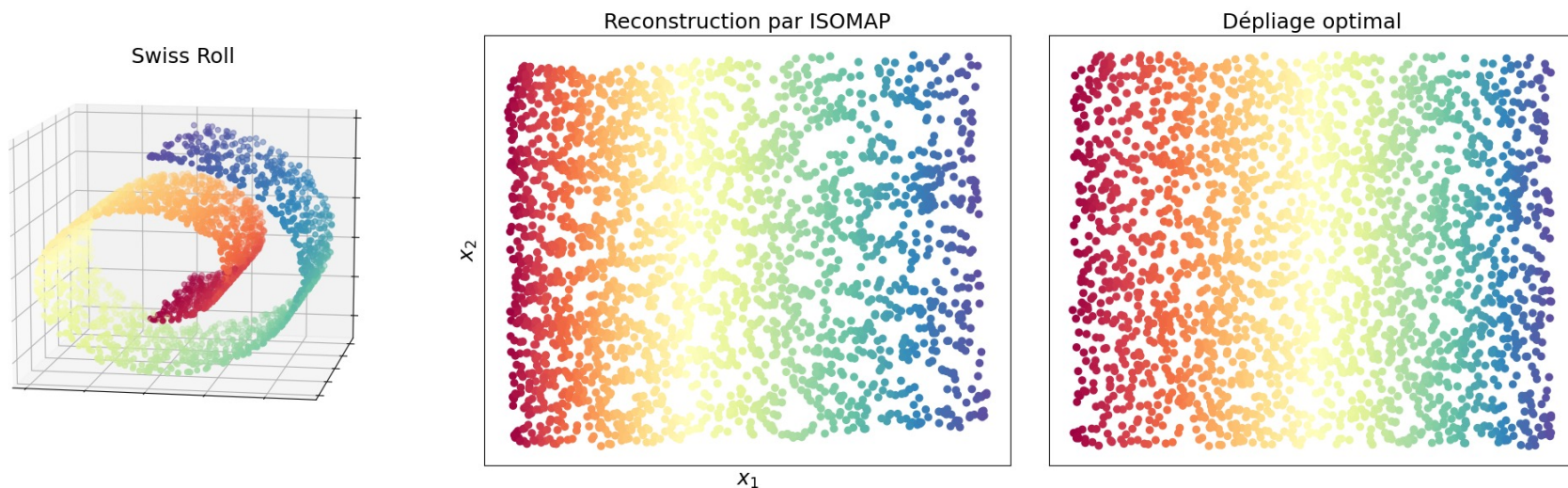
60 000 images



$d = 2$

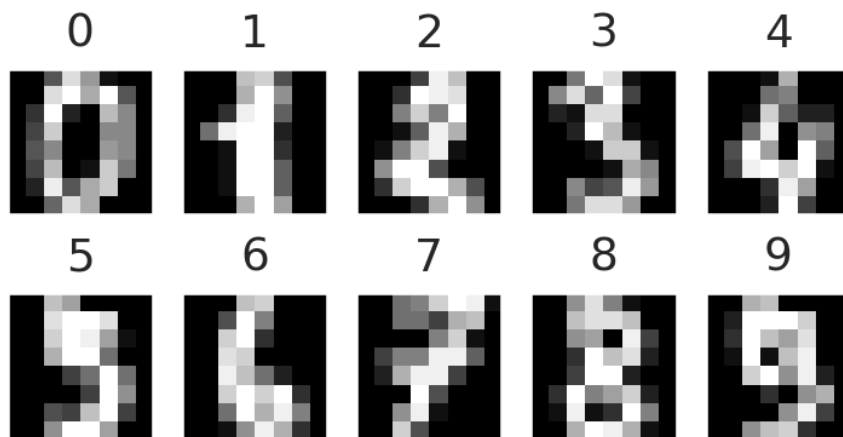
Cas non linéaire : ISOMAP

- Tente de préserver les propriétés de la variété sous-jacente en identifiant les paires de points proches et en reportant leur distance euclidienne

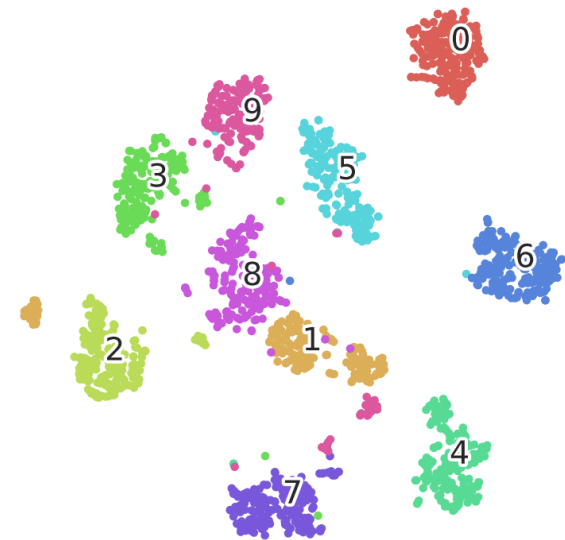


Cas non linéaire : tSNE

- À partir de calcul de probabilités conditionnelles
- Projette en **dimension 2** ou **3**
- Complexité en m^2 : donc recourir à des optimisations (m : nb d'exemples)



$d = 8 \times 8 = 64$



$d = 2$

Données **déséquilibrées** selon les classes

1. **Sous-échantillonner** la classe **majoritaire**
 - Pas adapté si peu de données
2. Créer des **exemples fictifs** de la classe **minoritaire**
 - E.g. par bruitage des exemples existants
3. **Modifier les coûts** de mauvaise classification
 - E.g. $\text{coût}(\text{faux positif}) \gg \text{coût}(\text{faux négatif})$

Les points aberrants

- Des **erreurs** ou des **exceptions**
 - Exemples :
 - salaire de diplômés Agro à la sortie
 - Fraudes
 - Attaques sur réseau
- Méthodes
 - Expertise
 - Contraintes d'intégrité
 - Détection par méthode d'apprentissage
 - One-class SVM
 - Boosting
 - ...