Les Réseaux de Neurones Artificiels

Antoine Cornuéjols

AgroParisTech – INRA MIA 518

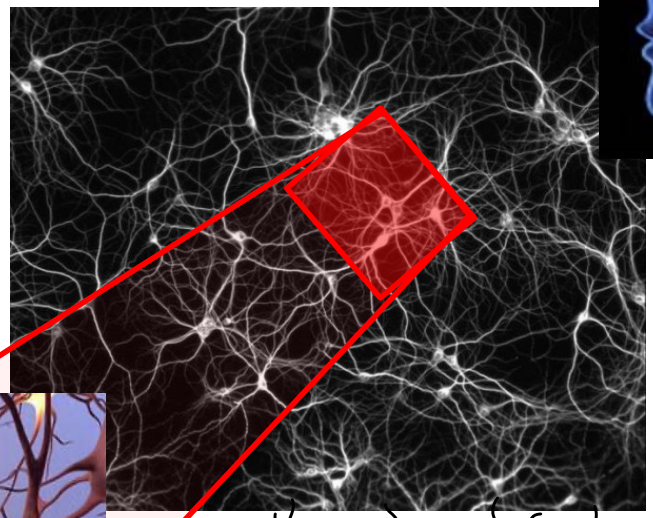
antoine.cornuejols@agroparistech.fr

Plan

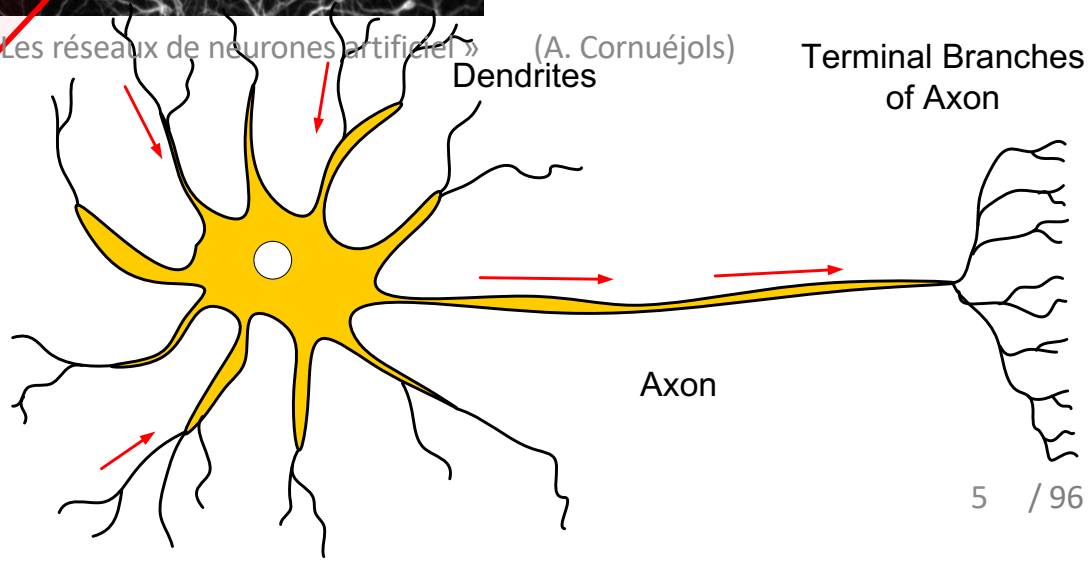
1. **Introduction**
2. Réseaux à une couche
3. Perceptrons multicouches
4. L'apprentissage de représentations

Introduction

Biological Neurons



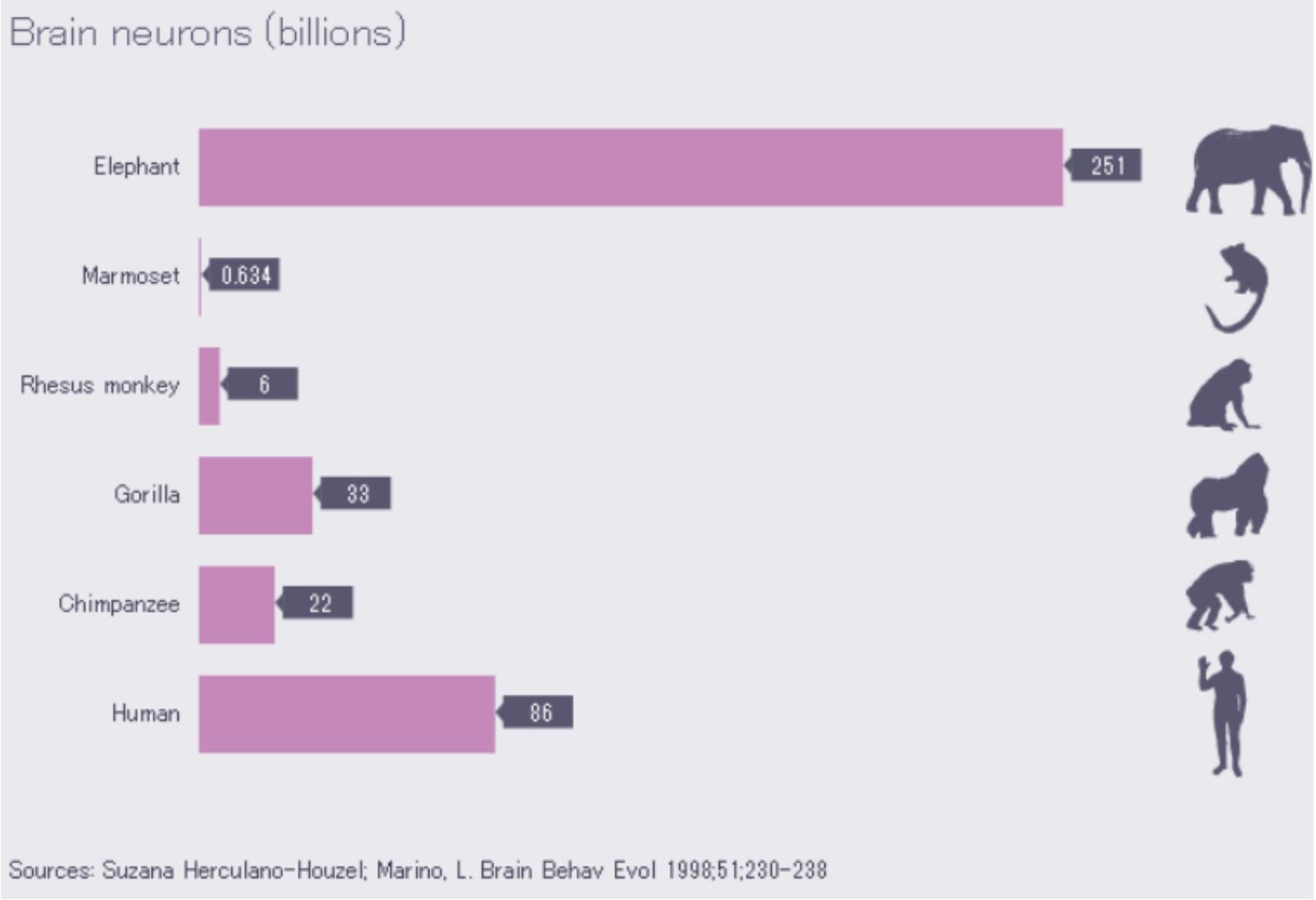
« Les réseaux de neurones artificiel » (A. Cornuéjols)



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**
    - ≈ **10^{11} neurones** dans le **cerveau humain**
      (950 x 10^3 dans celui de *l'abeille*)
    - ≈ **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)

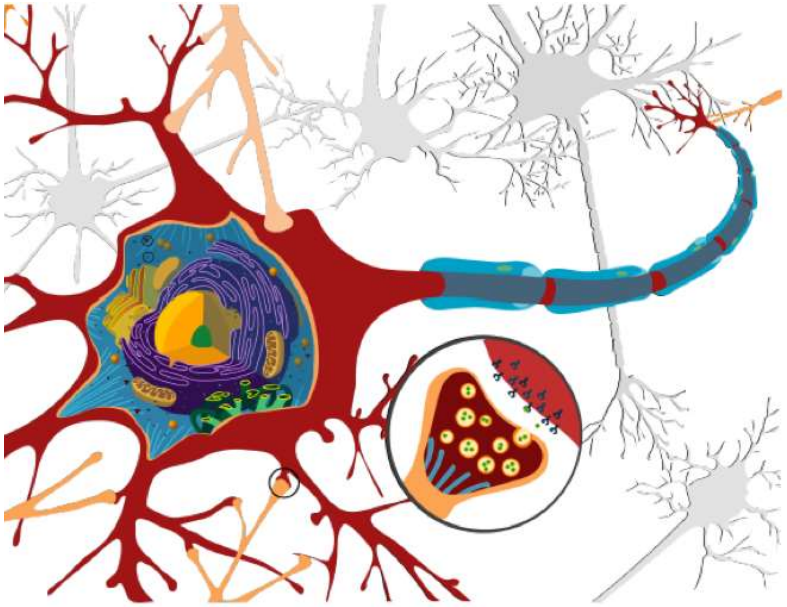
- **Mise en évidence des neurones**
  - Camillio Golgi et Ramon Y Cajal (~ 1870) ; les synapses (dans les années 1930)
- **Modélisation formelle**
  - Mc Culloch & Pitts (1943) : 1er modèle du neurone formel
  - Hebb (1949) : règle d'apprentissage par renforcement de couplage synaptique

Les réseaux à une couche

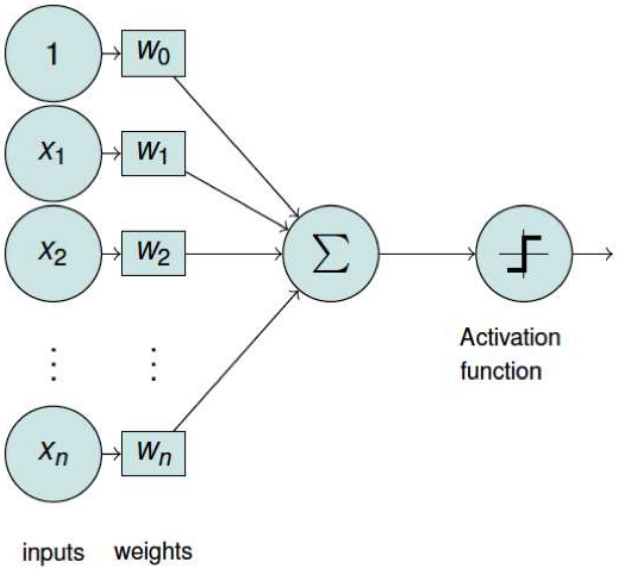
Historique (très limité)

- **Mise en évidence des neurones**
  - Camillio Golgi et Ramon Y Cajal (~ 1870) ; les synapses (dans les années 1930)
- **Modélisation formelle**
  - Mc Culloch & Pitt (1943) : 1er 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 »

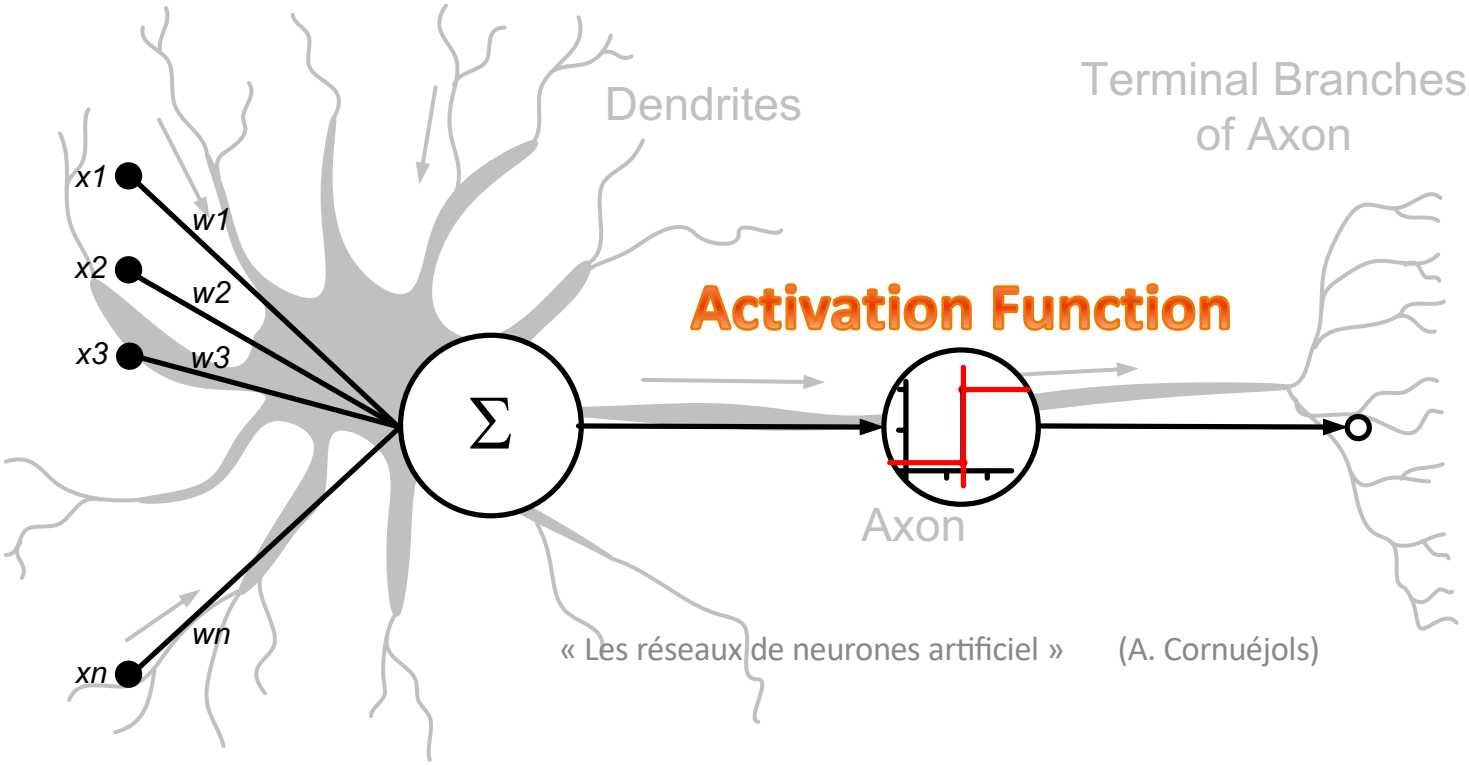


Biological Neuron



Computational Neuron

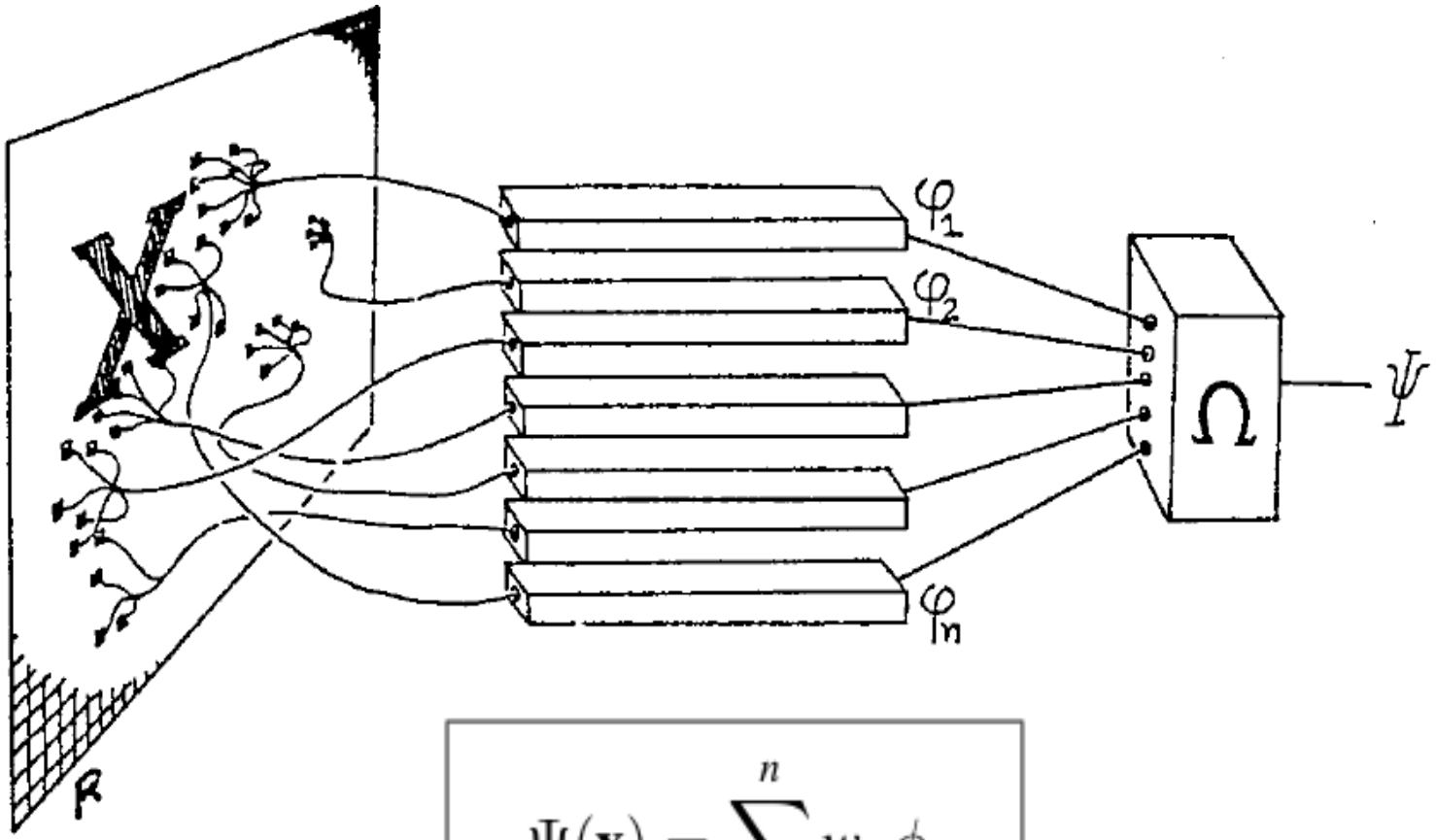
Artificial Neural Networks (ANN)



Slide credit : Andrew L. Nelson

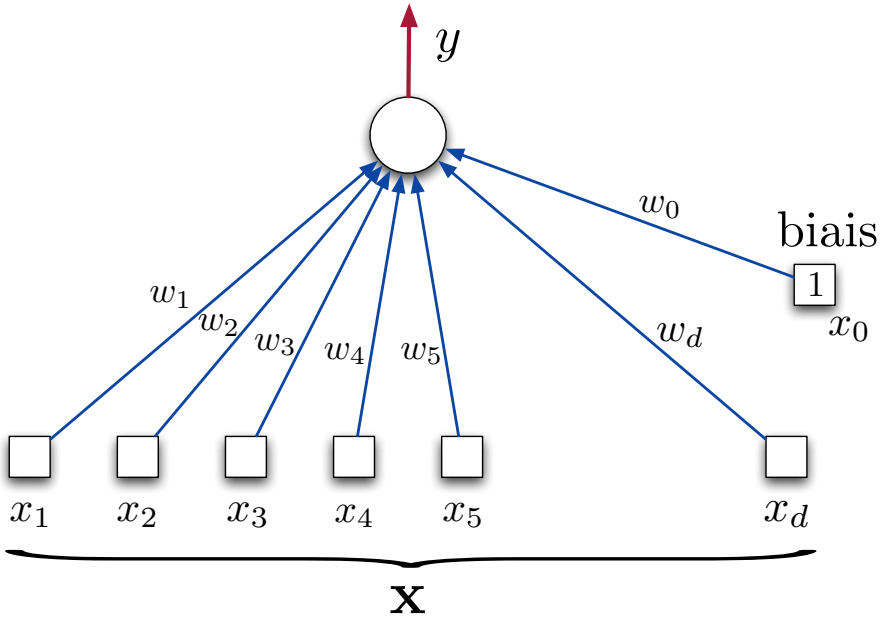
Le Perceptron

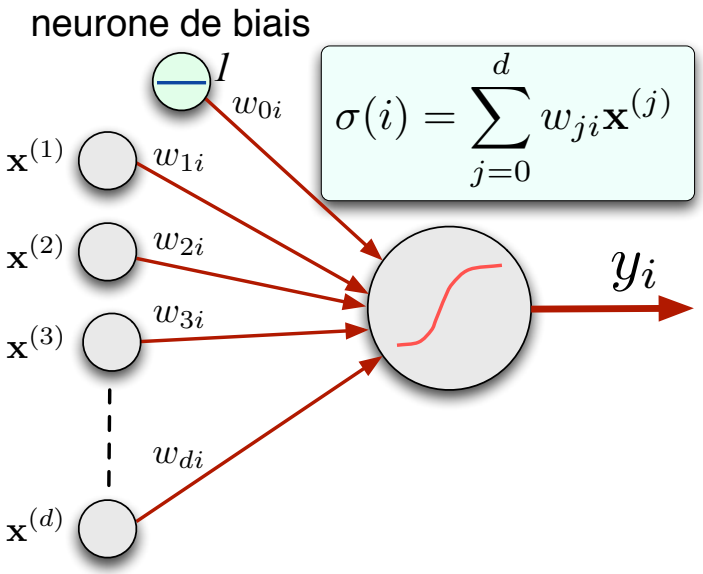
– Rosenblatt (1958-1962)



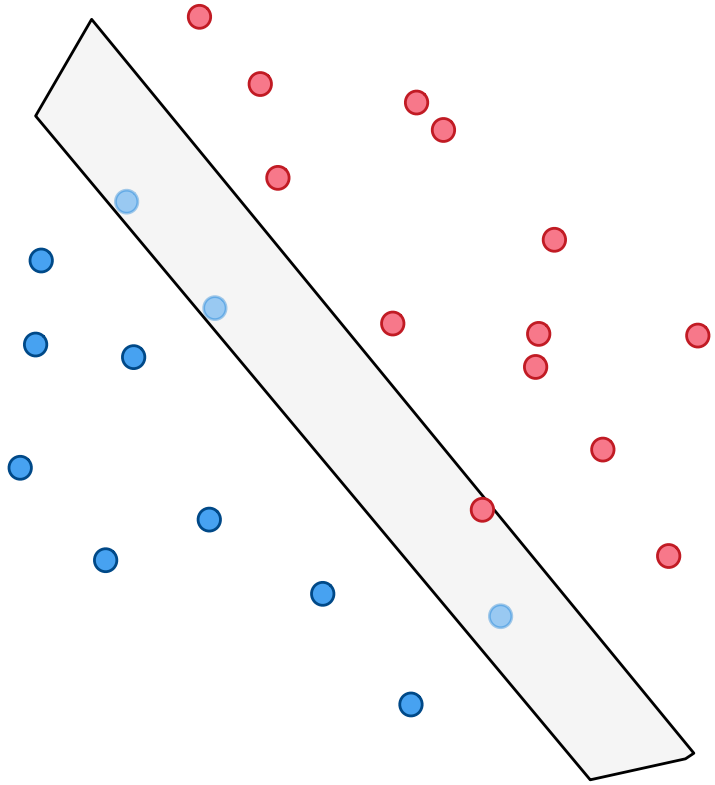
$$\Psi(\mathbf{x}) = \sum_{i=1}^{n} w_i \, \phi_i$$

Le perceptron





Le perceptron : un discriminant linéaire

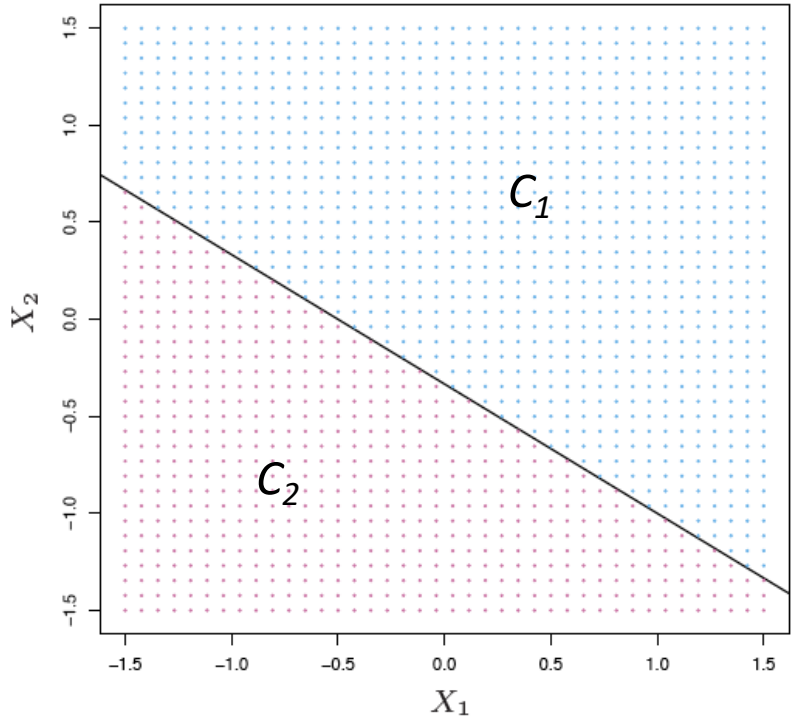


The perceptron: elementary algebra

- In the plan

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

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



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

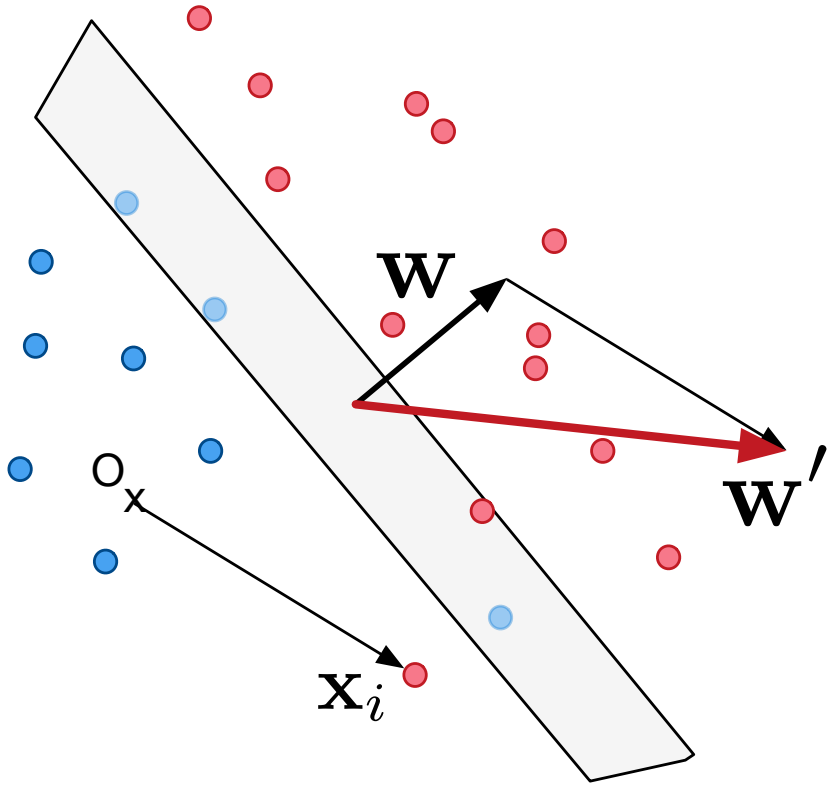
$$1 + 2\,x_1^* + 3\,x_2^* = \begin{cases} \geq 0, & \mathbf{x}^* \in \mathcal{C}_1 \\ \leq 0, & \mathbf{x}^* \in \mathcal{C}_2 \\ = 0, & \mathbf{x}^* \text{indéterminé} \end{cases}$$

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 $w^T.x_i < 0$ increase $w^T.x_i$

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

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

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

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

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

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

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

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

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  w ← 0 ;                          // Other initialisations of the weight vector are possible
2  converged ← false;
3  while converged = false do
4      converged ← true;
5      for i = 1 to |D| do
6          if y_i w · x_i ≤ 0        // i.e., ŷ_i ≠ y_i
7          then
8              w ← w + η y_i x_i ;
9              converged ← false;    // We changed 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 :

$$R_{Emp}(\boldsymbol{w}) = - \sum_{x_j \in \mathcal{M}} \boldsymbol{w}^T \boldsymbol{x}_j \, . \, u_j$$

Car nous voulons pour tous les exemples :

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

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 : $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta \; \boldsymbol{x}_i \; u_i$

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

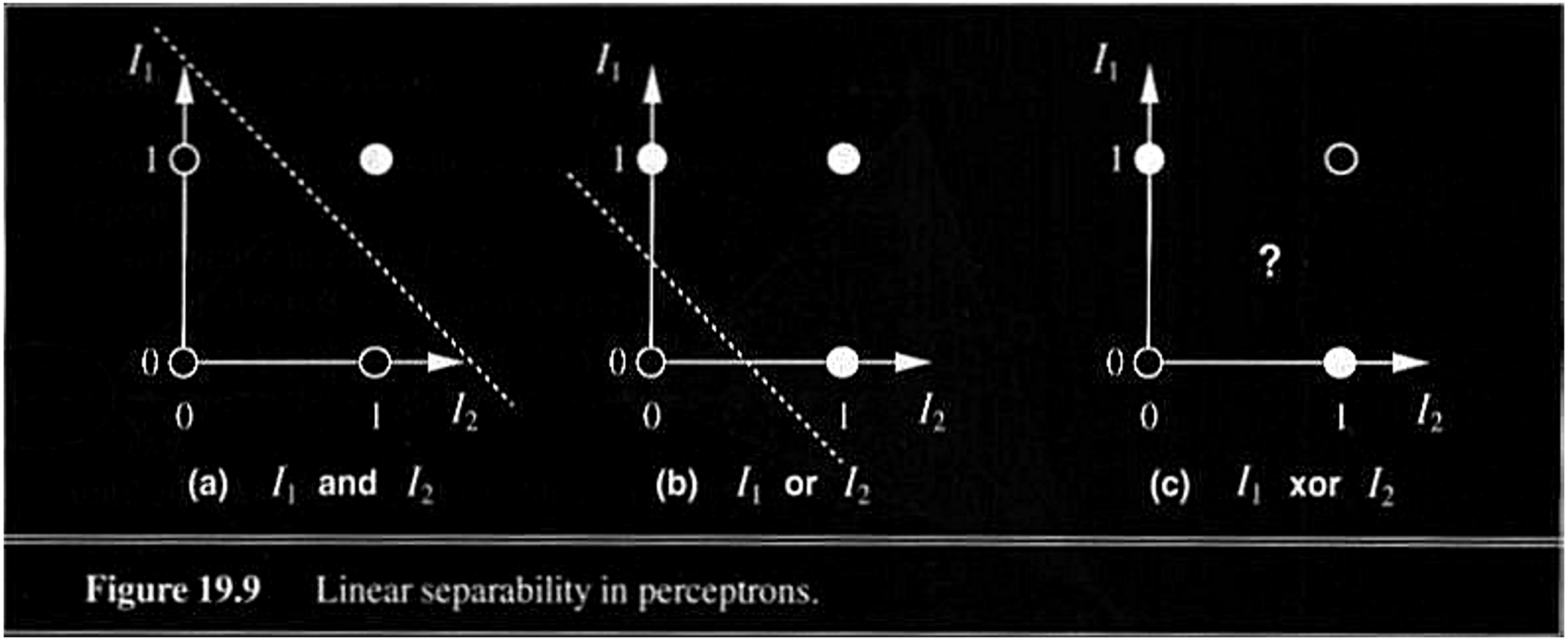
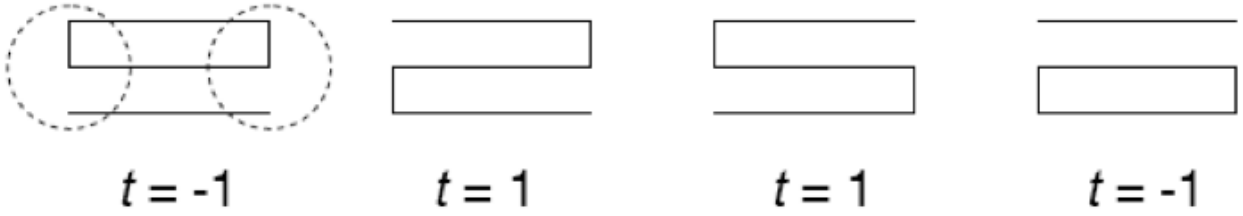


Figure 19.9 Linear separability in perceptrons.

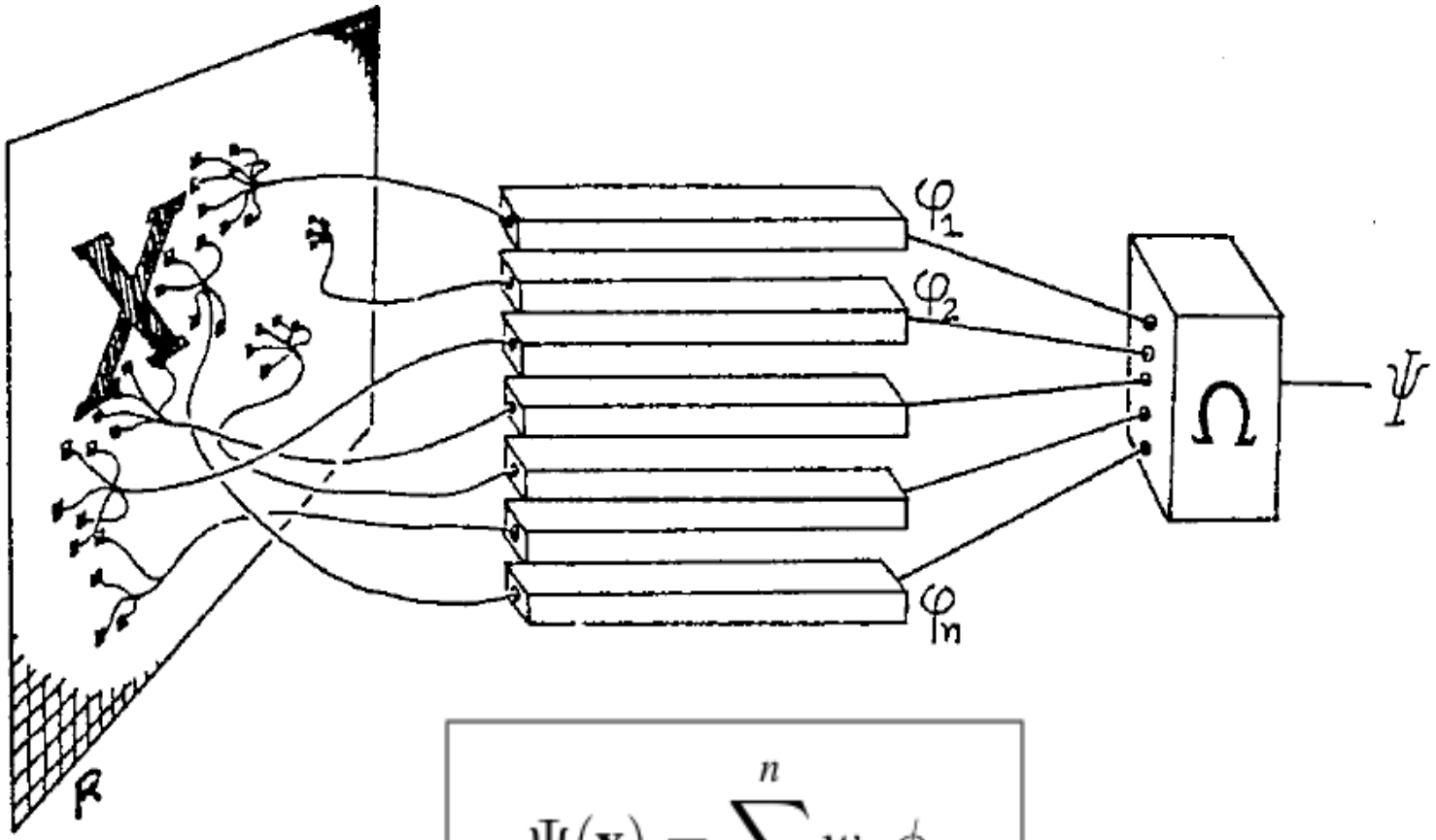
Limites du Perceptron

- 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

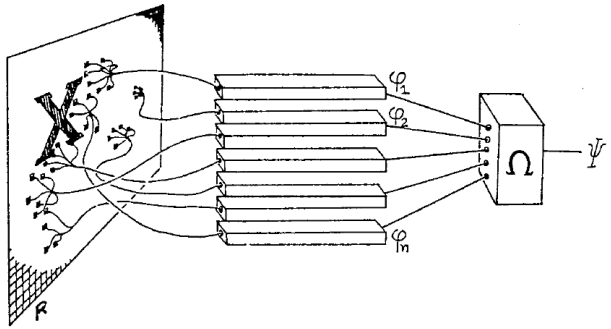
– Rosenblatt (1958-1962)



$$\Psi(\mathbf{x}) = \sum_{i=1}^{n} w_i \, \phi_i$$

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

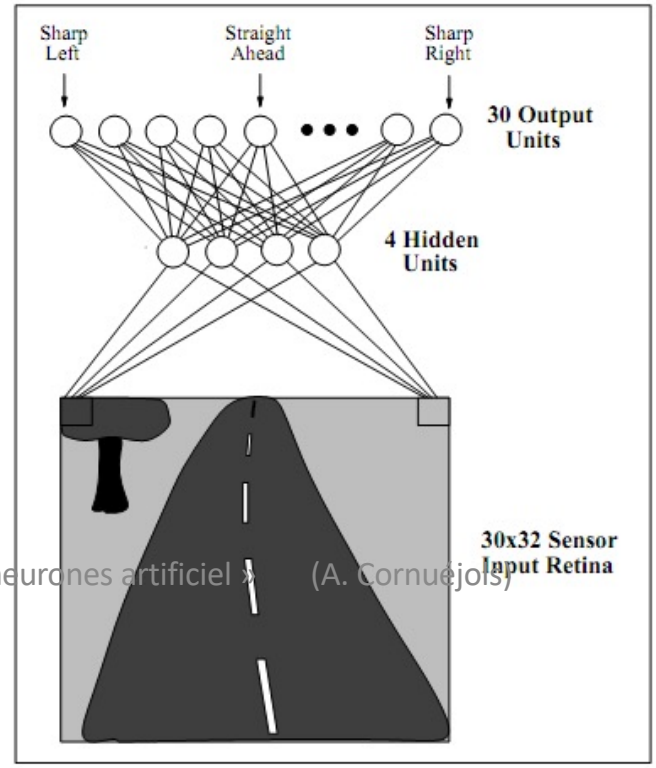
Historique (très limité)

- **Mise en évidence des neurones**
  - Camillio Golgi et Ramon Y Cajal (~ 1870) ; les synapses (dans les années 1930)
- **Modélisation formelle**
  - Mc Culloch & Pitt (1943) : 1er 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

Les perceptrons multi-couches

Neural network application

- ALVINN drives 70 mph on highways

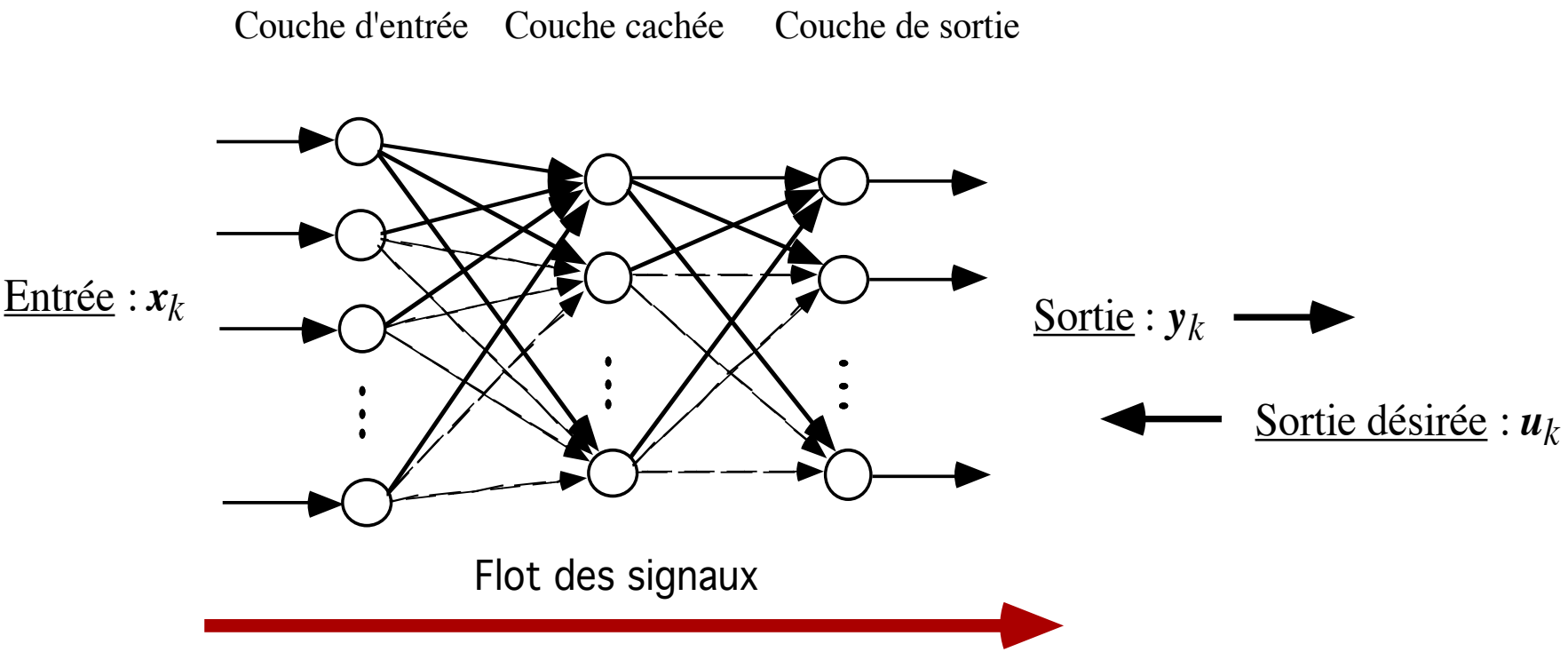


« Les réseaux de neurones artificiel » (A. Cornuéjols)

Composants et structure du PMC

Le perceptron multi-couche

- Topologie typique



Le Perceptron Multi-Couches : propagation

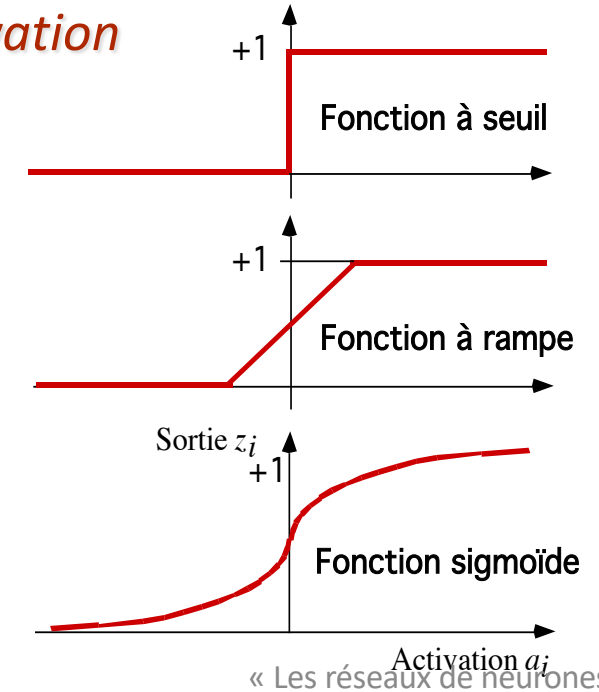
- Pour chaque neurone :

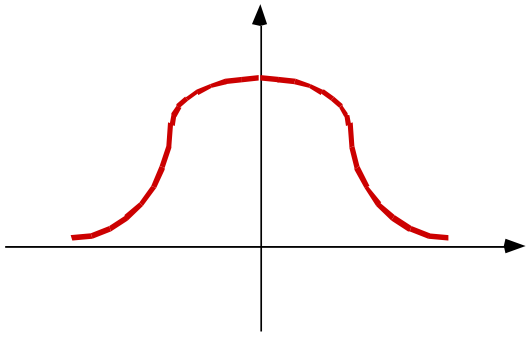
$$y_l = g\left(\sum_{j=0,d} w_{jk}\ \phi_j\right) = g(a_k)$$

  - 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*

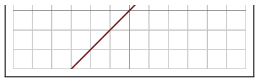
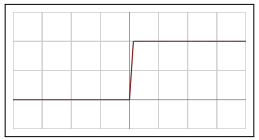
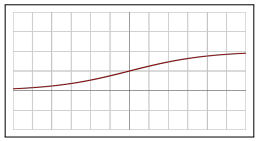
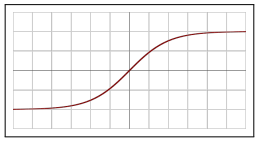
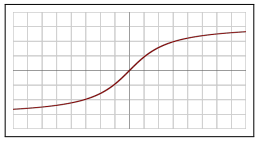
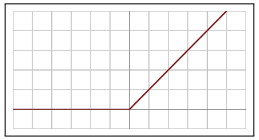
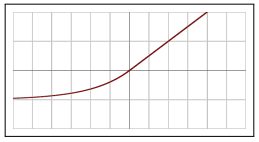
$$g(a) = \frac{1}{1 + e^{-a}}$$

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





Fonction à base radiale

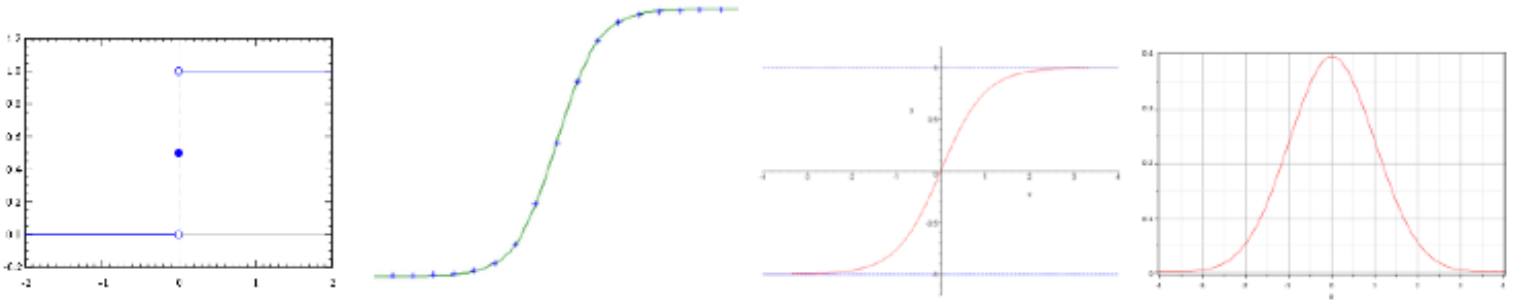
Nom	Graphe	Équation	Dérivée
Rampe		$f(x) = x$	$f'(x) = 1$
Heaviside		$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{si } x \neq 0 \\ ? & \text{si } x = 0 \end{cases}$
Logistique ou sigmoïde		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)\left(1 - f(x)\right)$
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]

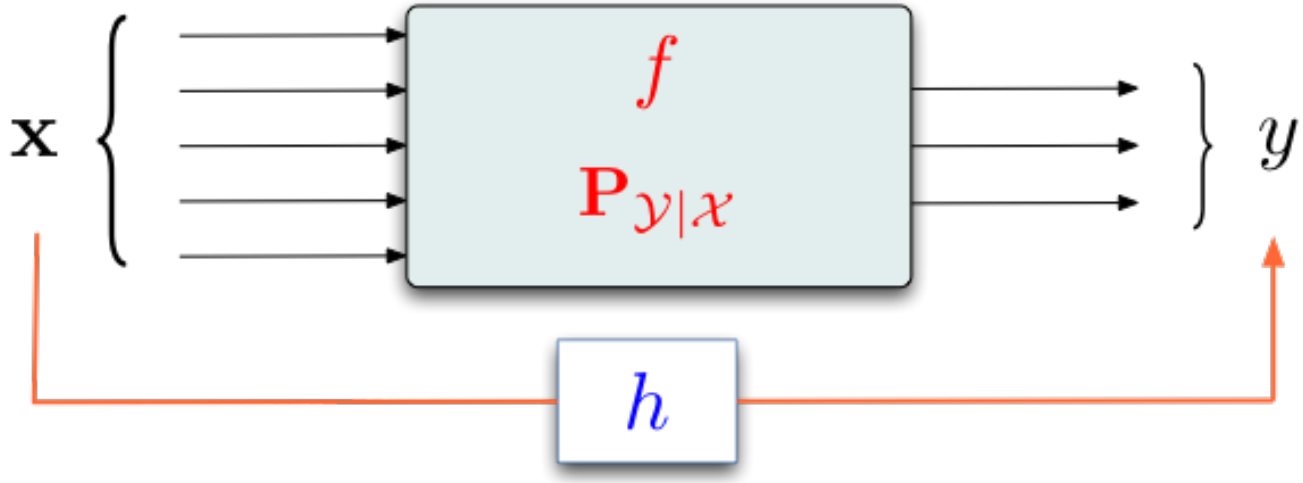
$$\hat{y}_j = \frac{e^{-z_j}}{\sum_{i=1}^{m} e^{-z_i}}$$

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

L'apprentissage

dans les PMC

Apprentissage supervisé



À partir :

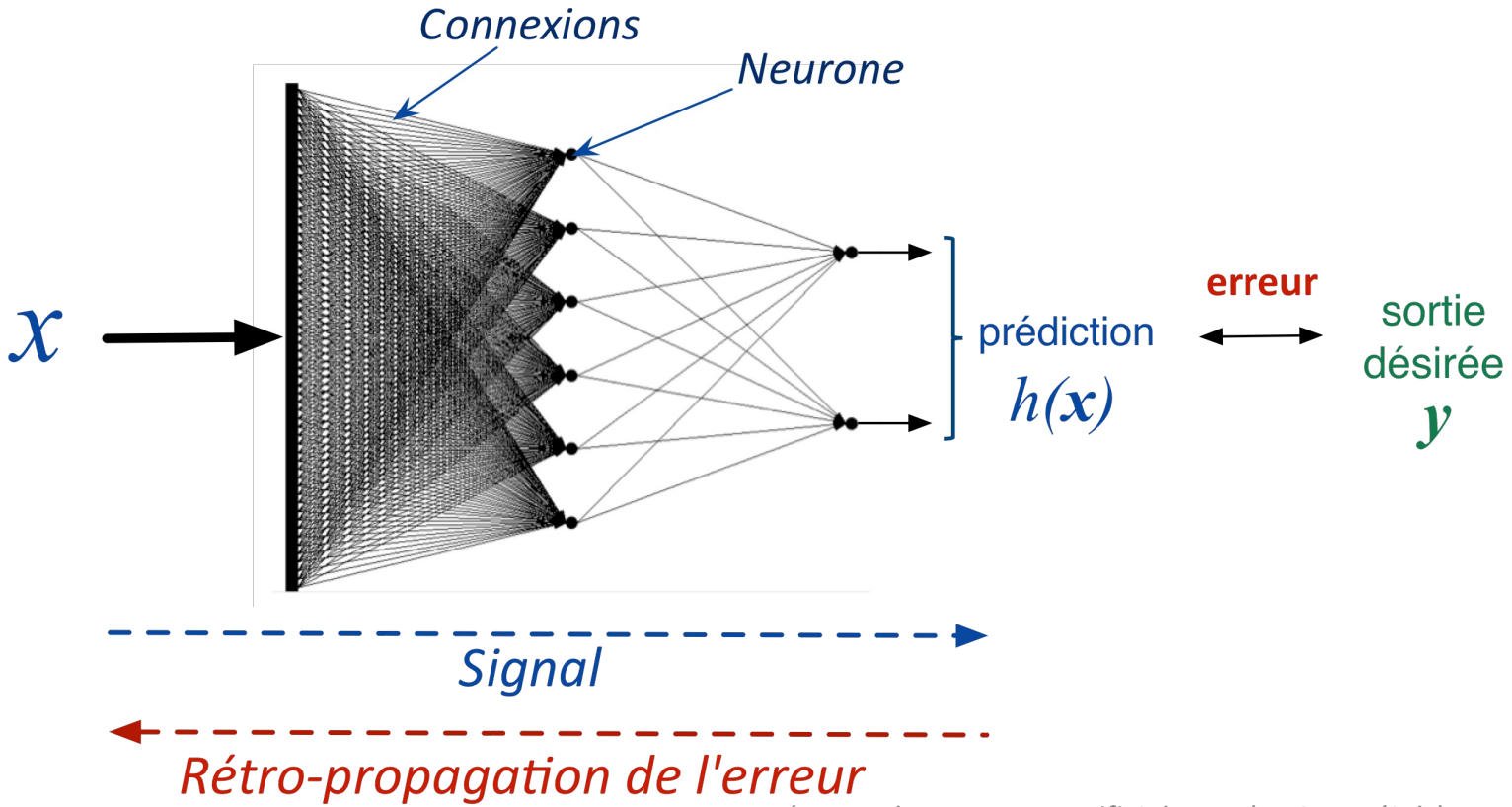
- d'un échantillon d'apprentissage $S_m = \langle (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m) \rangle$
- de connaissances préalables sur le type de dépendances sur $\mathcal{X} \times \mathcal{Y}$

Trouver :

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

Learning with Multi-Layer Perceptrons

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



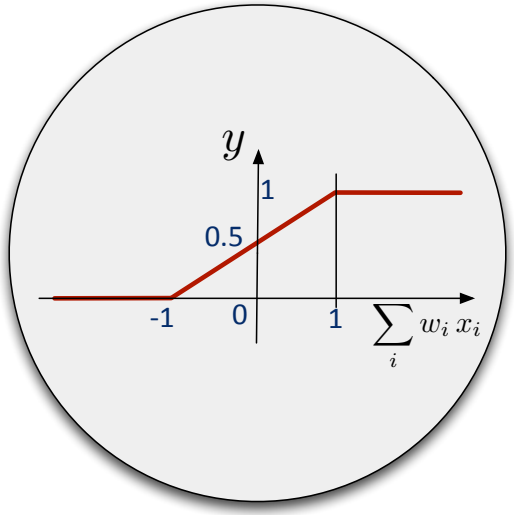
Analyse

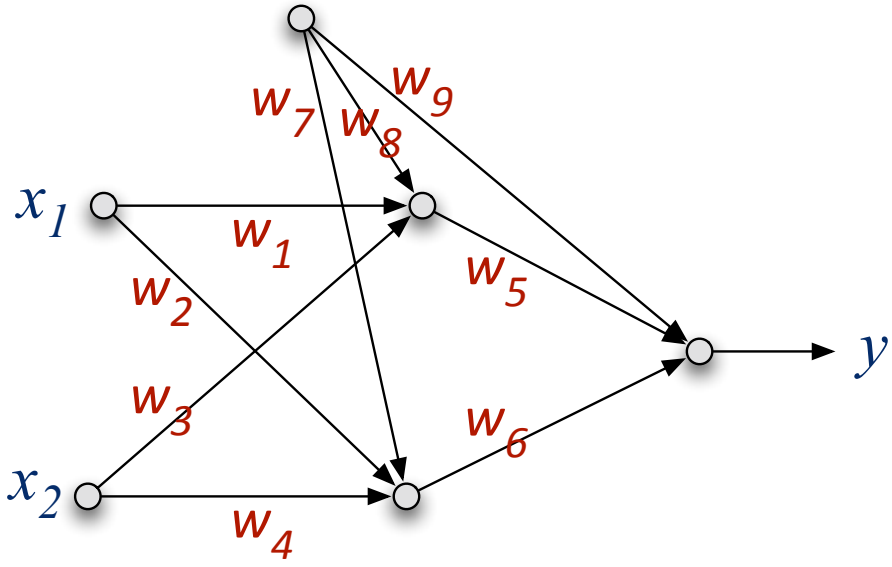
Rôle des couches cachées

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

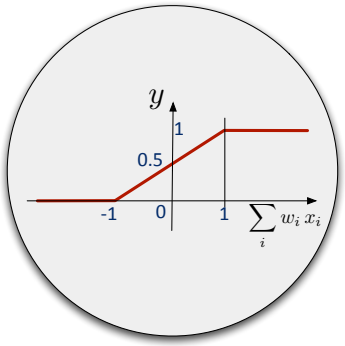




Saurez-vous trouver les poids w_i ?

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 =

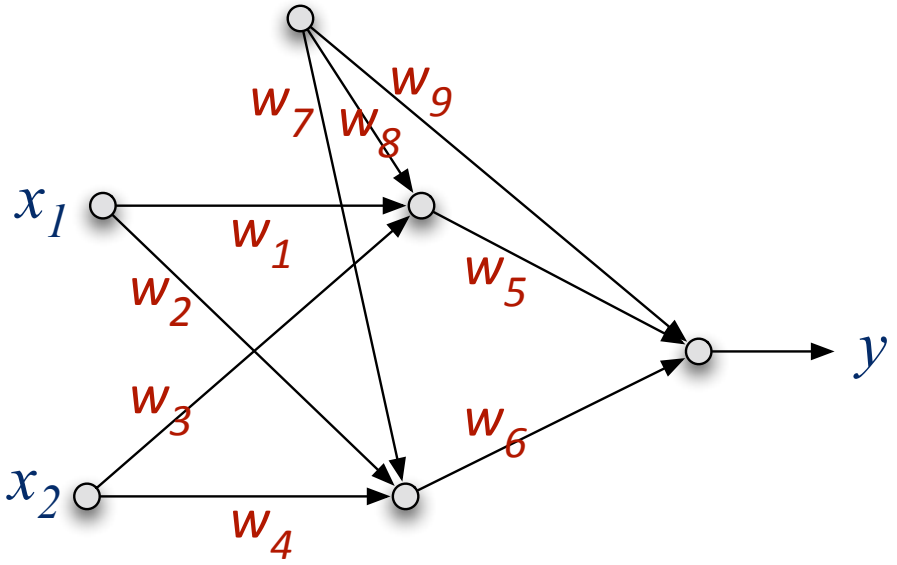
W_5 =

W_6 =

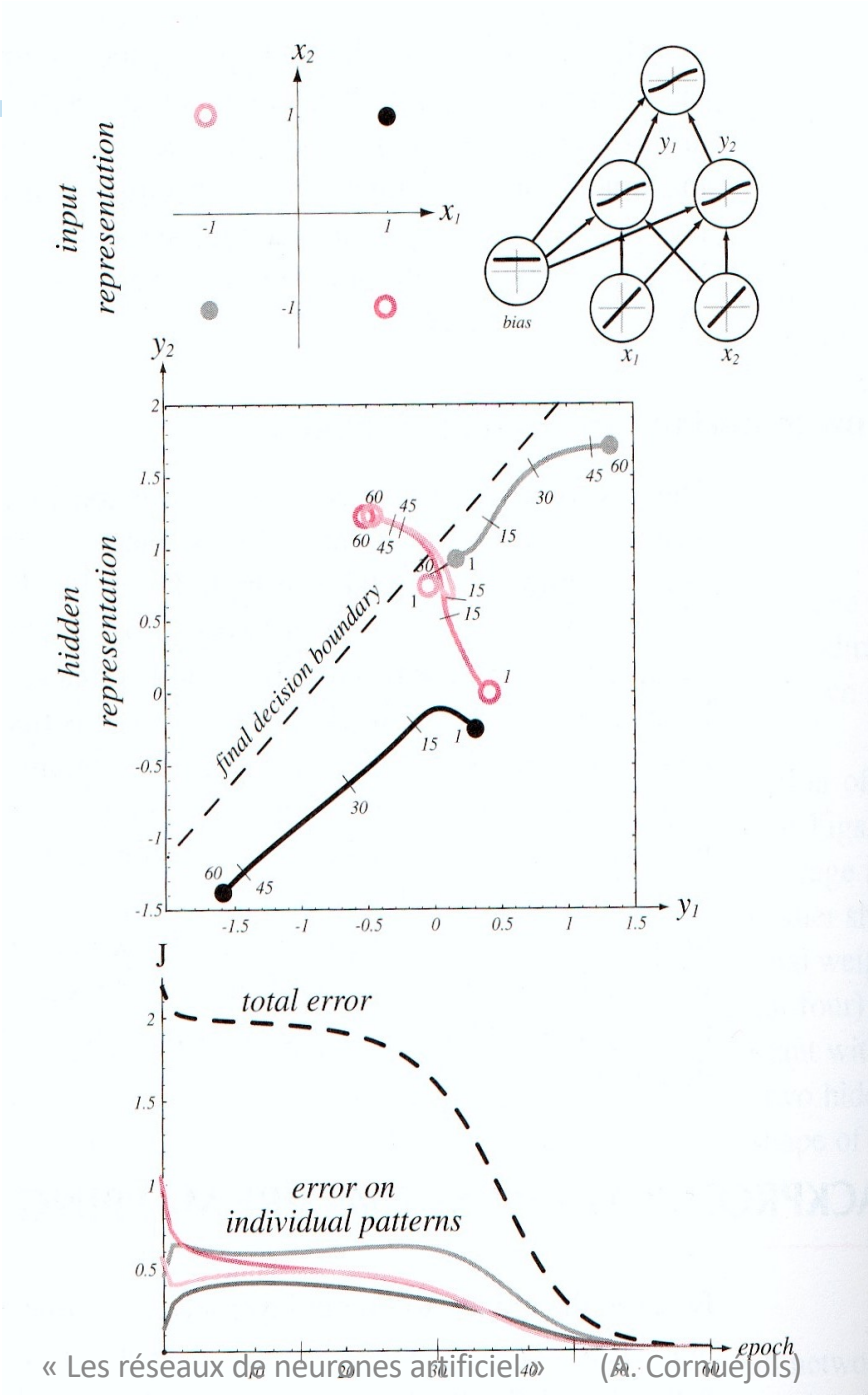
W_7 =

W_8 =

W_9 =

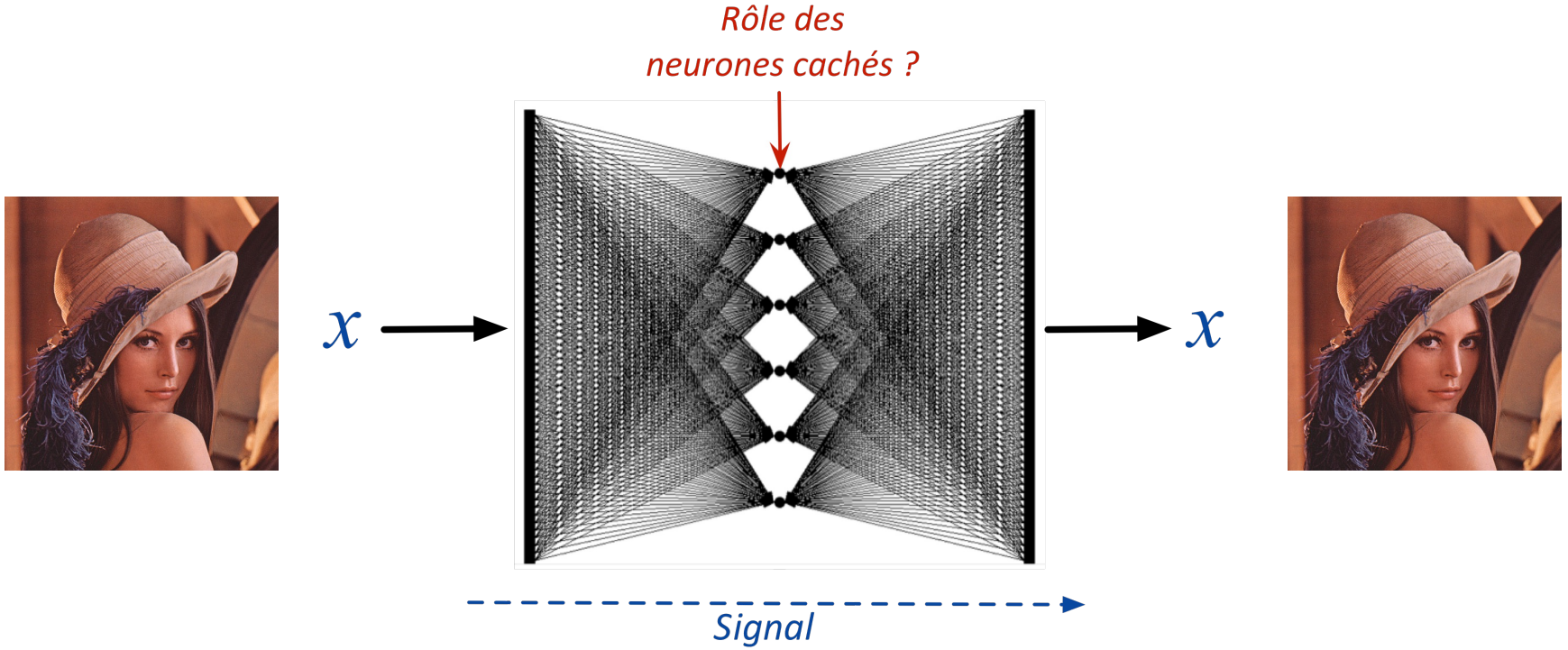


Rôle de la couche cachée



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

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



Rôle de la couche cachée

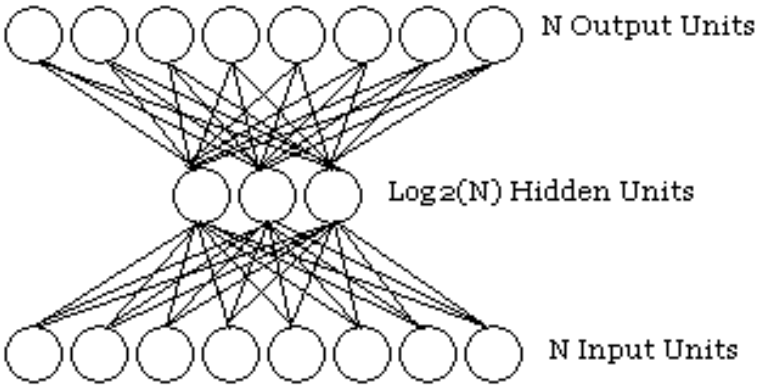
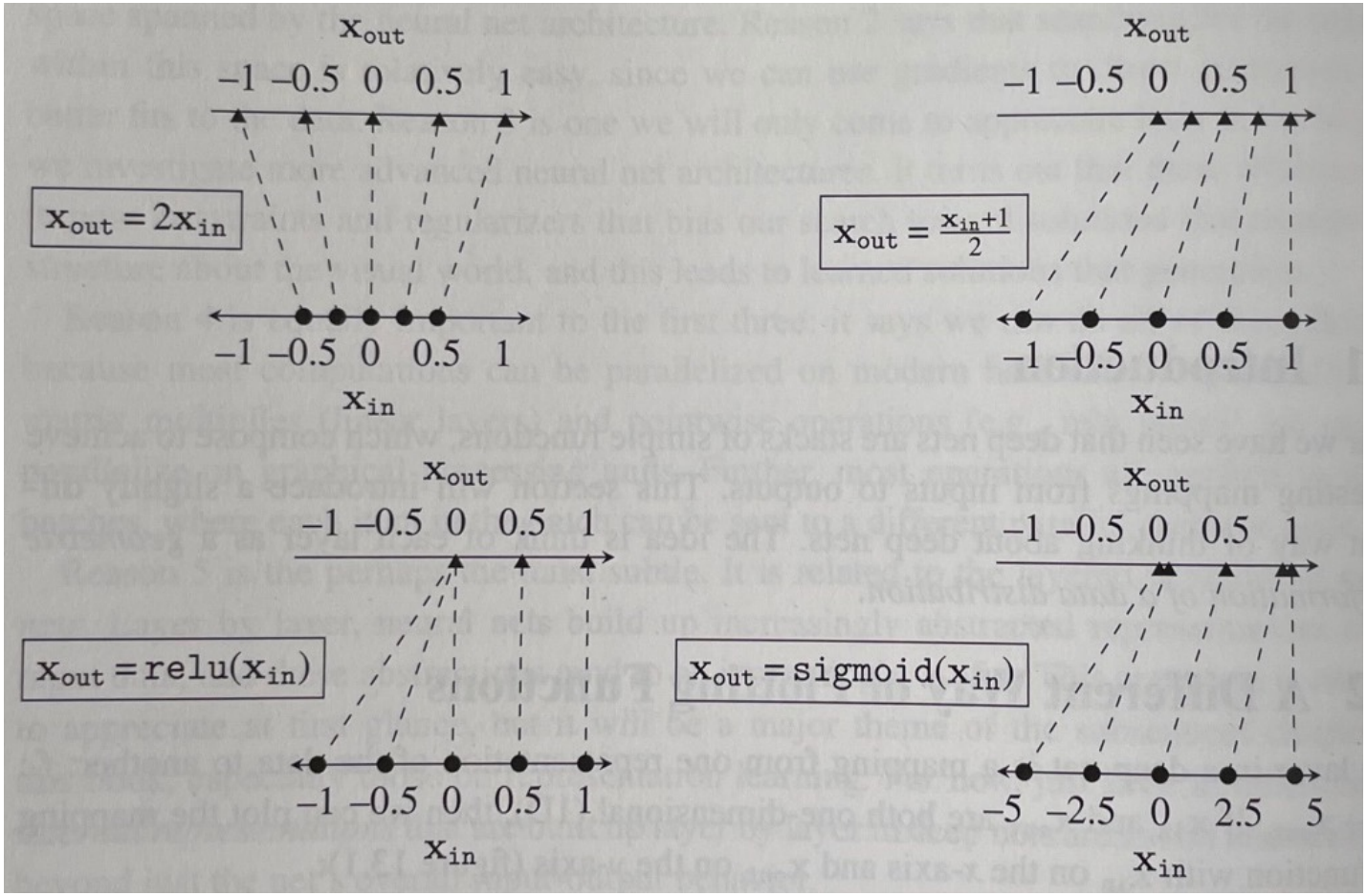
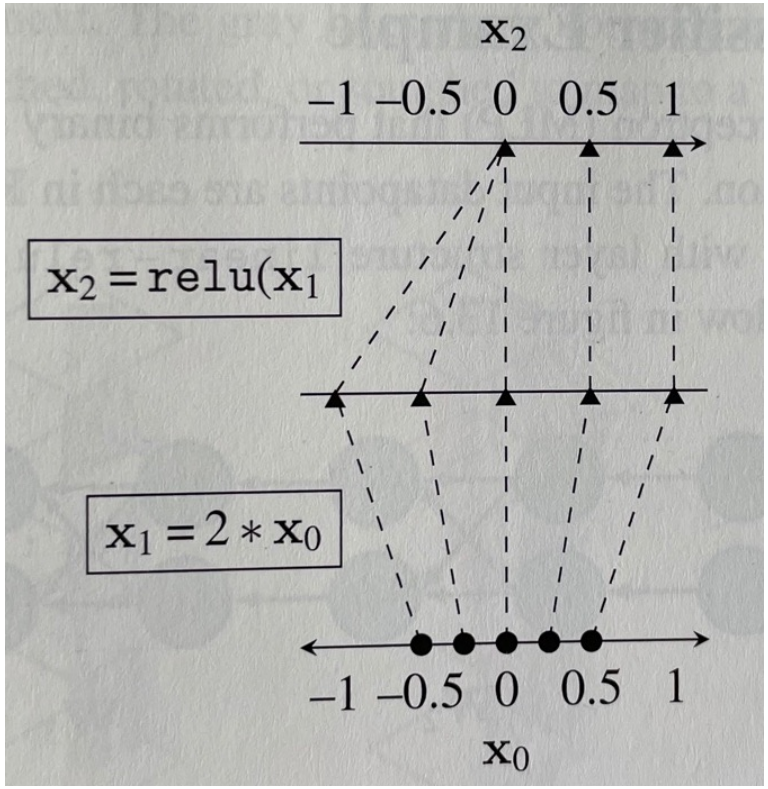


Figure 1. The encoding problem (Rumelhart, Hinton, & Williams, 1986)

- Exemples de transformations par un neurone

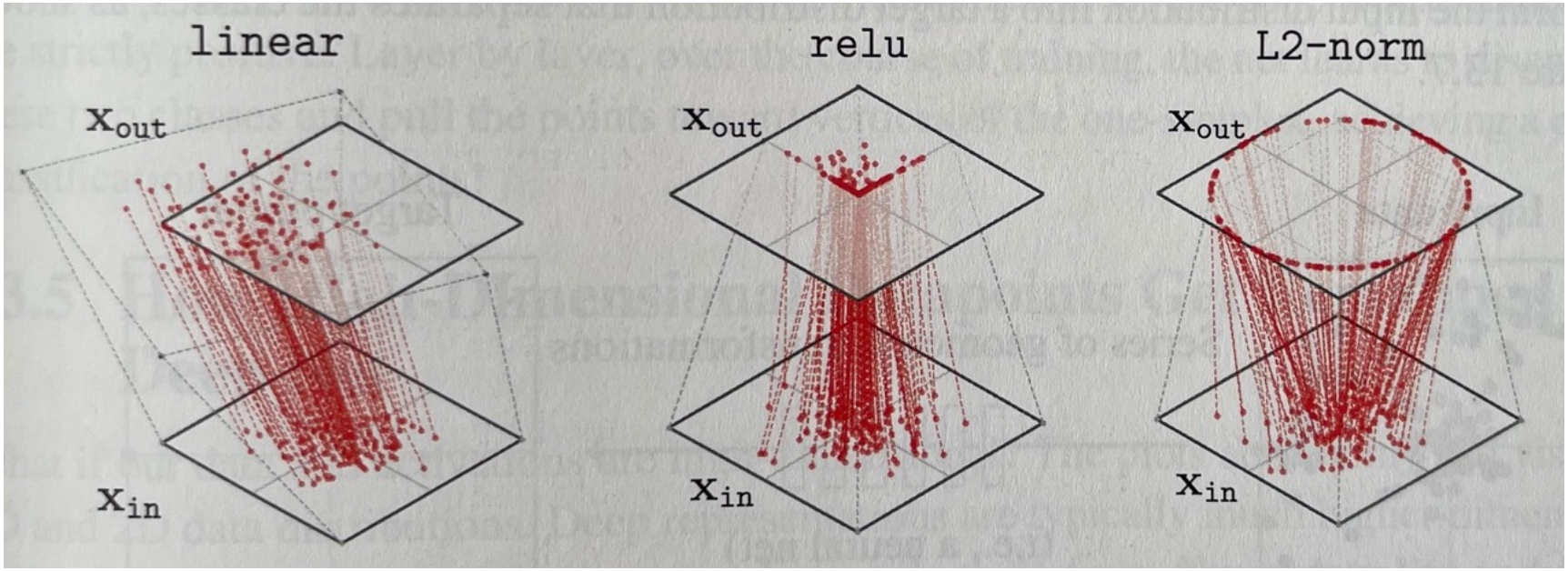


- Exemple de **séquence** de transformations (neurones)

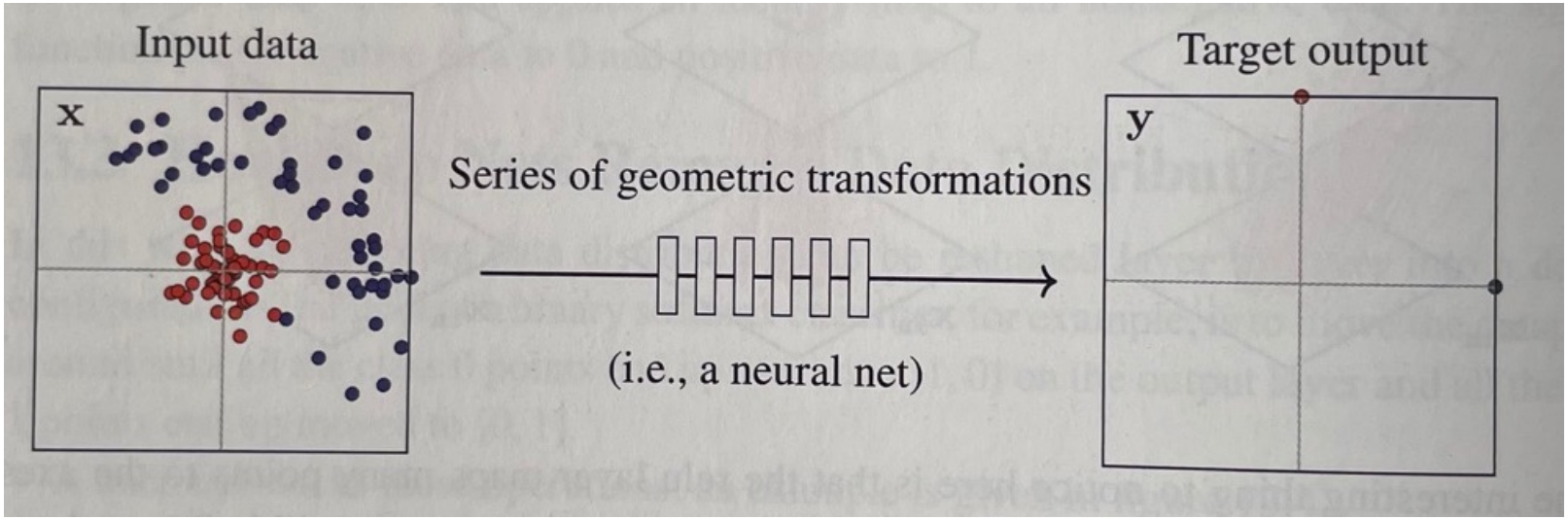


Exemples de transformations en 2D

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 » → (1,0) et classe « rouge » → (0,1)

- Illustration

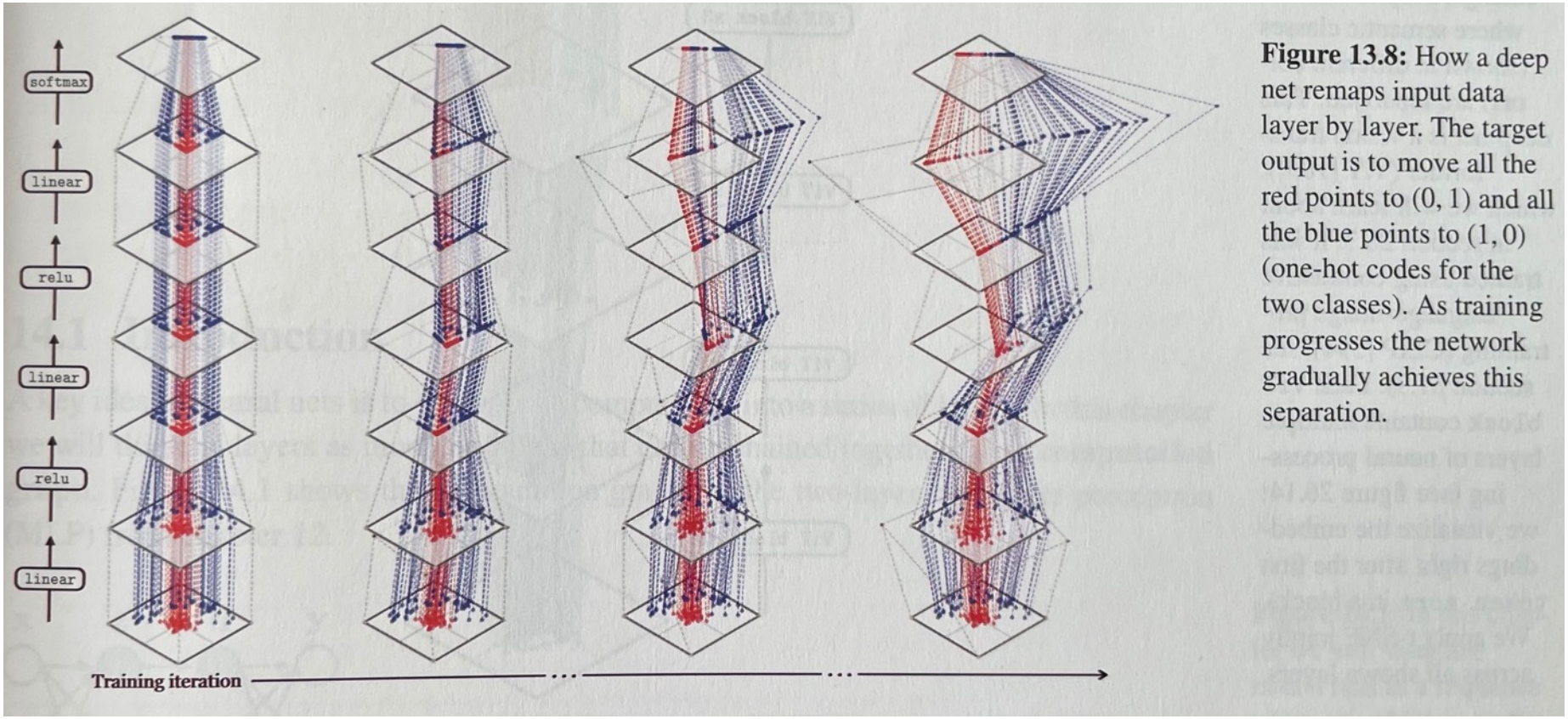
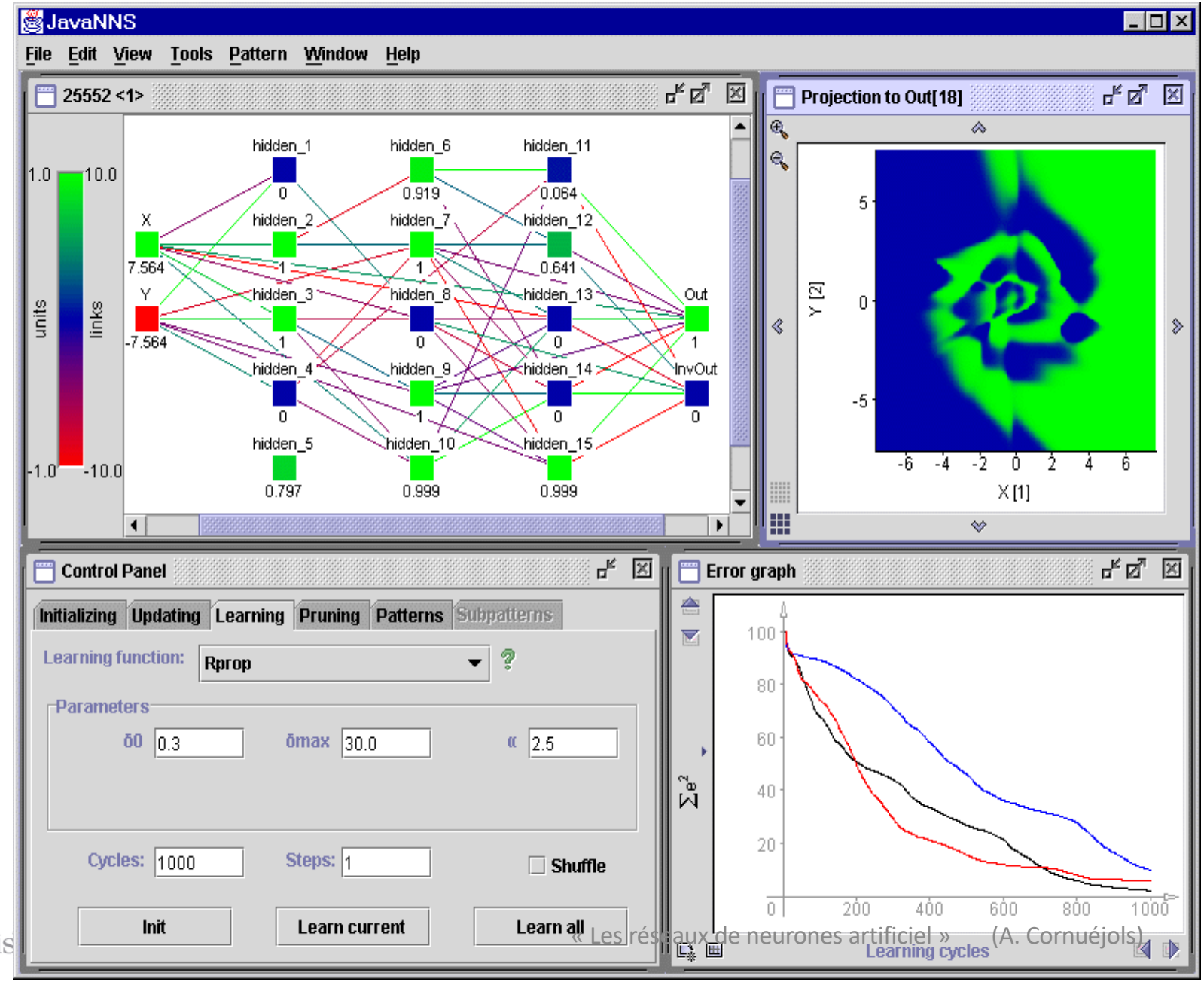


Figure 13.8: How a deep net remaps input data layer by layer. The target output is to move all the red points to (0, 1) and all the blue points to (1, 0) (one-hot codes for the two classes). As training progresses the network gradually achieves this separation.

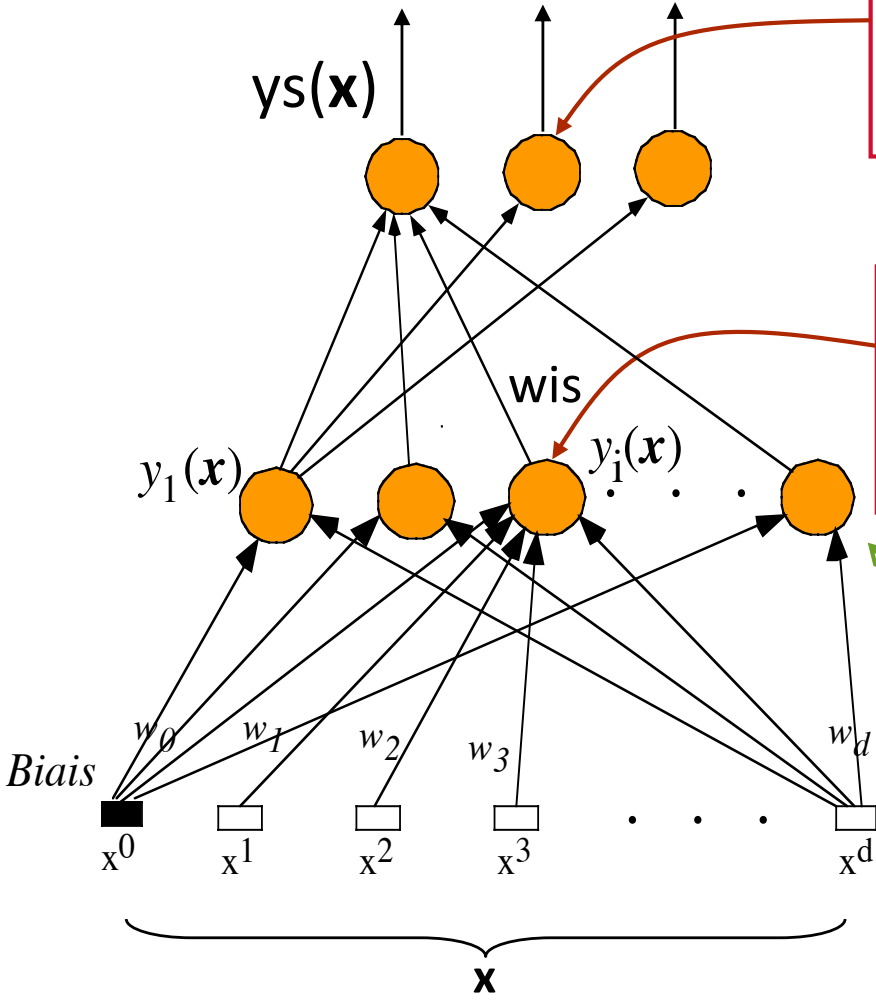
From [Antonio Torralba et al. (2024) « **Foundations of Computer Vision** »]

L'apprentissage de représentations

Exemple de réseau (simulateur JavaNNS)



MLPs: feed forward (summary)



$$y_s(\boldsymbol{x}) = \sum_{j=1}^{k} w_{js} y_j$$

$$a_i(\boldsymbol{x}) = \sum_{j=1}^{d} w_j x^j + w_0$$

$$y_i(\boldsymbol{x}) = g(a_i(\boldsymbol{x}))$$

k neurones sur la couche cachée

Le Perceptron Multi-Couches : apprentissage

Objectif :

$$\boldsymbol{w}^* = \arg\min_{w} \frac{1}{m} \sum_{l=1}^{m} \left[y(\boldsymbol{x}_l ; \boldsymbol{w}) - u(\boldsymbol{x}_l) \right]^2$$

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

Algorithme itératif :

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

Cas hors-ligne (gradient total) :

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

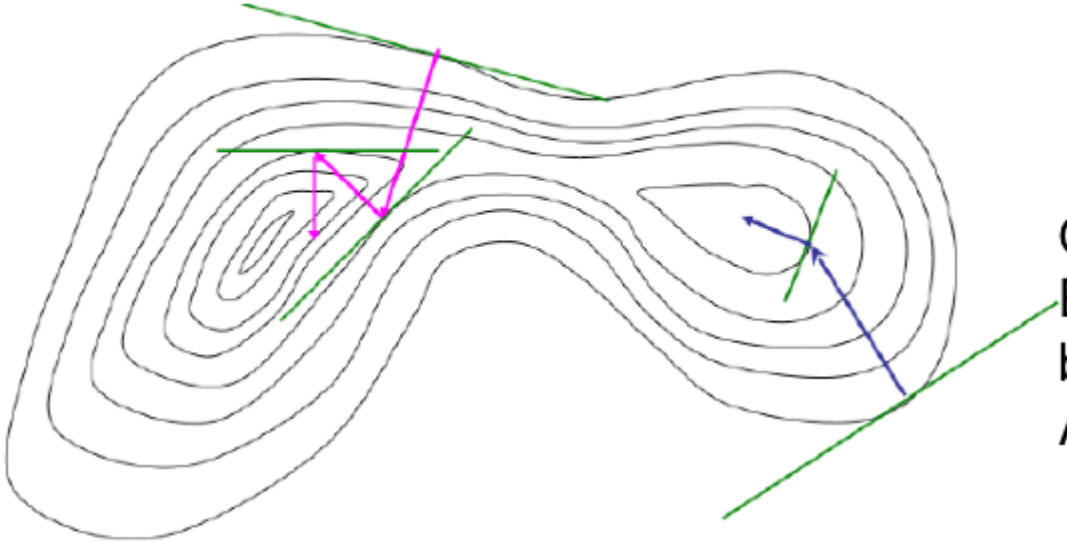
où :

$$R_E(x_k, \boldsymbol{w}) = [t_k - f(x_k, \boldsymbol{w})]^2$$

Cas en-ligne (gradient stochastique) :

$$w_{ij}(t) = w_{ij}(t-1) - \eta(t) \frac{\partial R_E(x_k, \boldsymbol{w})}{\partial w_{ij}}$$

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

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

avec :
W les paramètres ; η : le pas ; $\left. \frac{\mathrm{d}\mathcal{J}(\mathbf{W})}{\mathrm{d}\mathbf{W}} \right|_{\mathbf{W}_t}$: 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

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

– Pour les cellules de la couche de sortie :

$$\delta_k = \frac{\partial E^l}{\partial a_k} = g'(a_k) \frac{\partial E^l}{\partial y_k} = g'(a_k) \cdot \left(u_k(\boldsymbol{x}_l) - y_k\right)$$

– Pour les cellules d'une couche cachée :

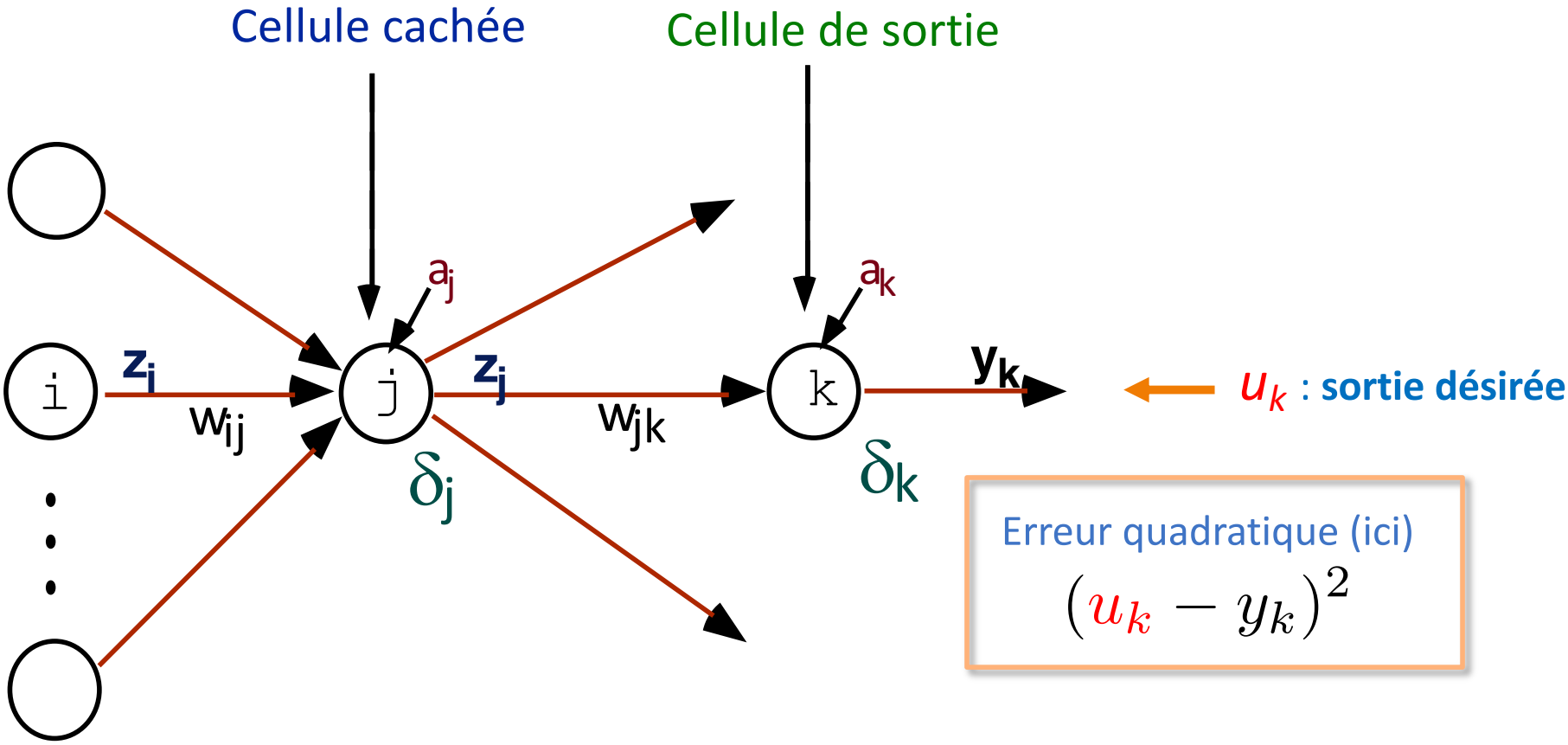
$$\delta_j = \frac{\partial E^l}{\partial a_j} = \sum_k \frac{\partial E^l}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} = g'(a_j) \cdot \sum_k w_{jk} \, \delta_k$$

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



- 2. Modification des poids
  - On suppose gradient à pas (constant ou non): η(t)
  - Si apprentissage stochastique (après présentation de chaque exemple)

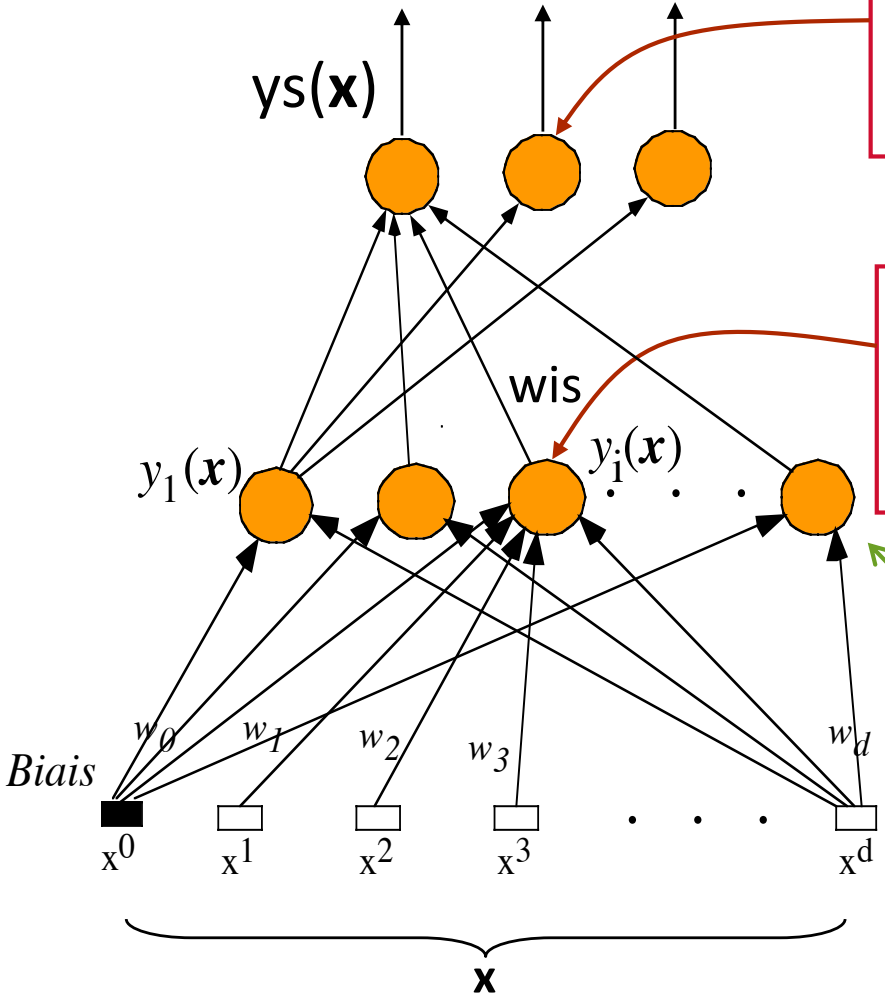
$$\Delta w_{ij} = \eta(t)\ \delta_j\ y_i$$

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)



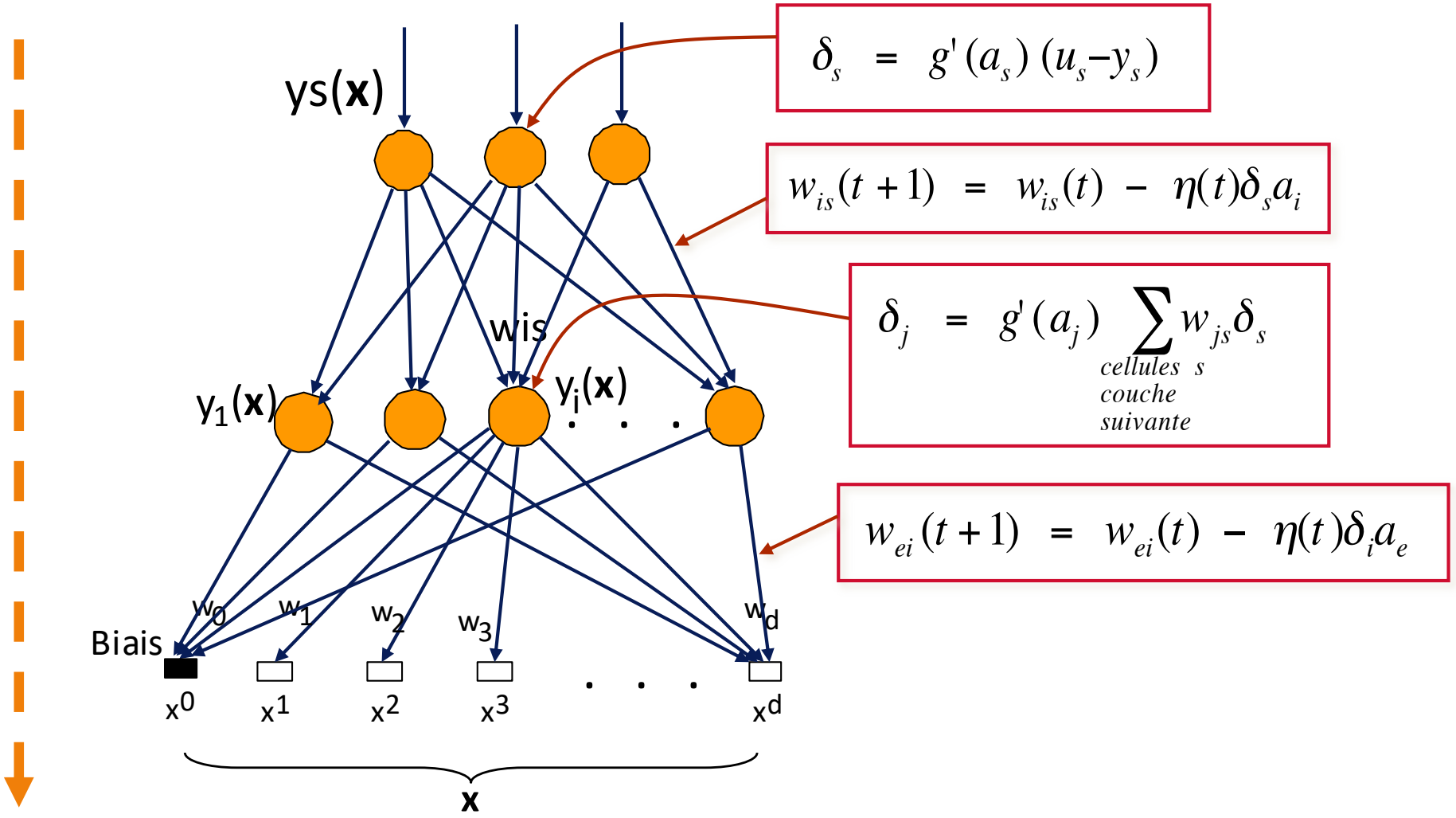
$$y_s(\boldsymbol{x}) = \sum_{j=1}^{k} w_{js} y_j$$

$$a_i(\boldsymbol{x}) = \sum_{j=1}^{d} w_j x^j + w_0$$

$$y_i(\boldsymbol{x}) = g(a_i(\boldsymbol{x}))$$

k neurones sur la couche cachée

Le PMC : passes avant et arrière (résumé)

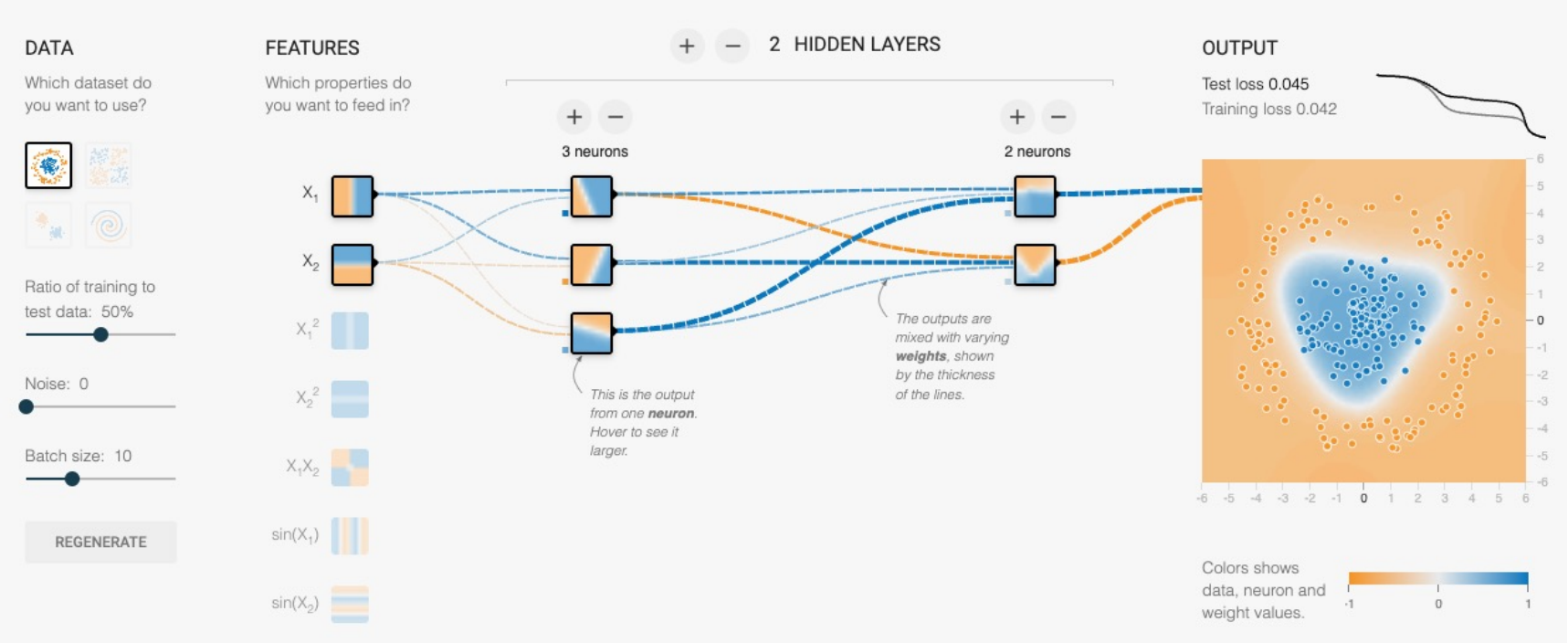


PMC : La rétro-propagation de gradient

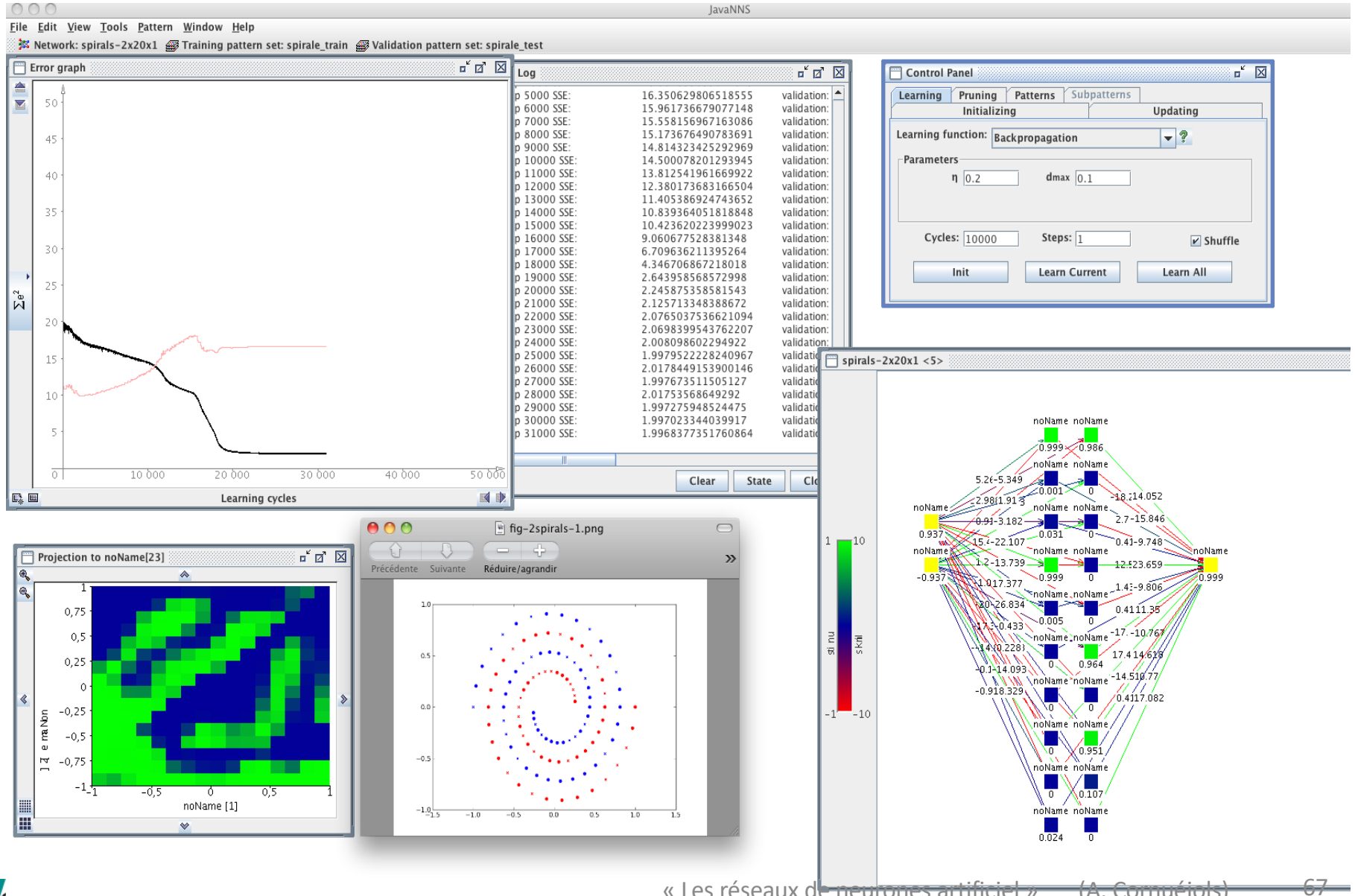
- Efficacité en apprentissage
  - En O(w) pour chaque passe d'apprentissage, w = nb de poids
  - Il faut typiquement plusieurs centaines de passes (voir plus loin)
  - Il faut typiquement recommencer plusieurs dizaines de fois un apprentissage en partant avec différentes initialisations des poids
- Efficacité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

Démo

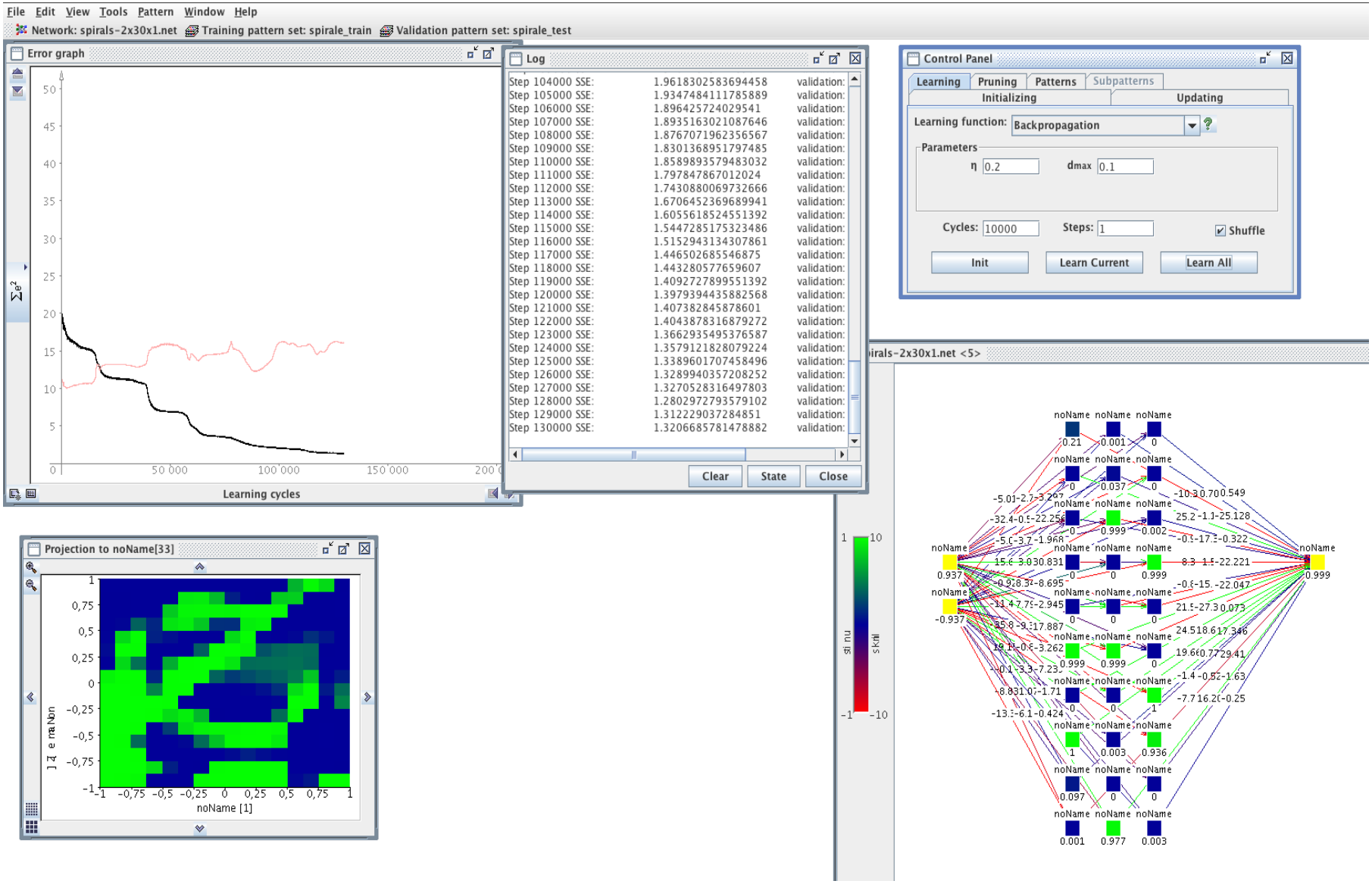
playground.tensorflow.org



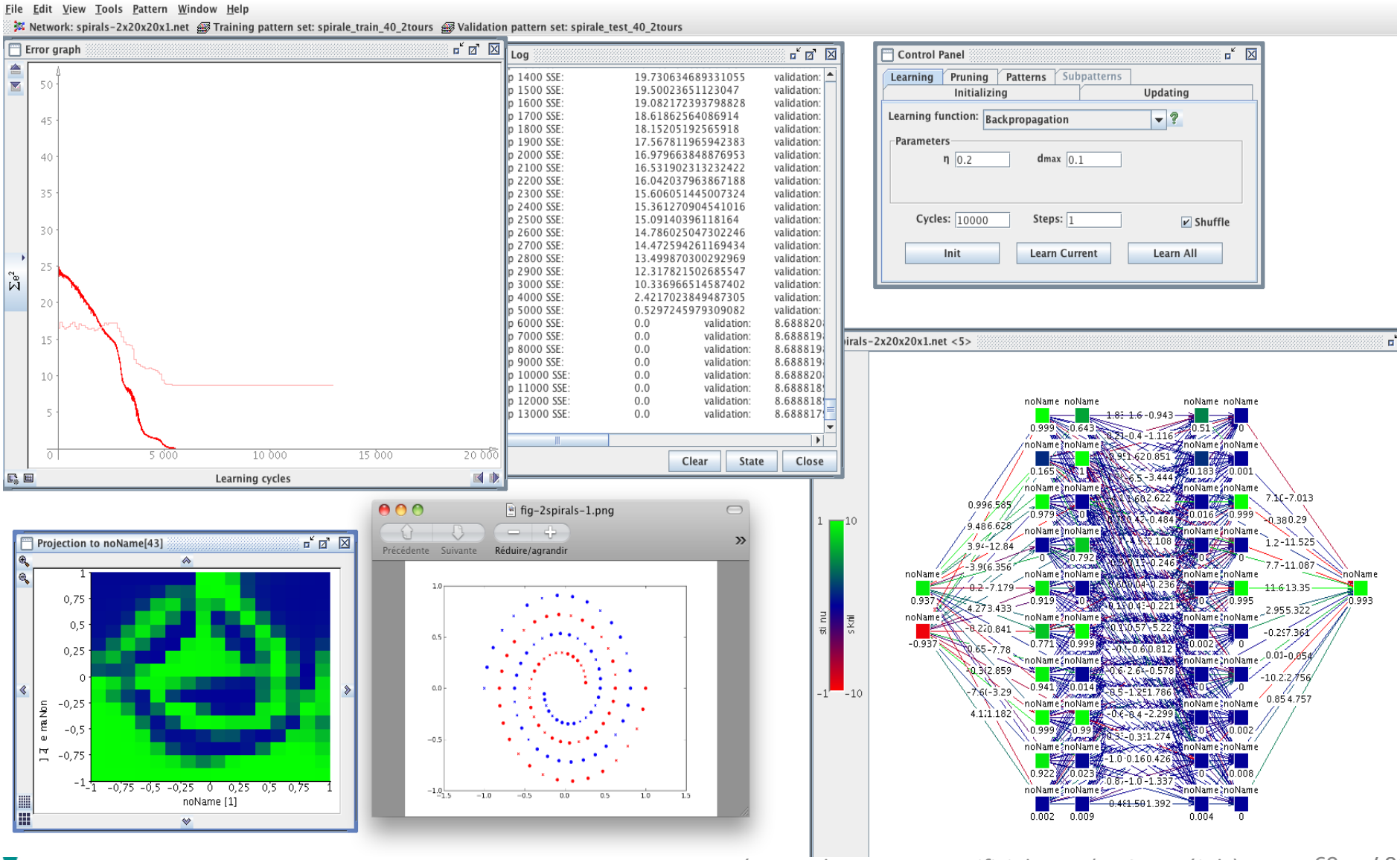
Illustration



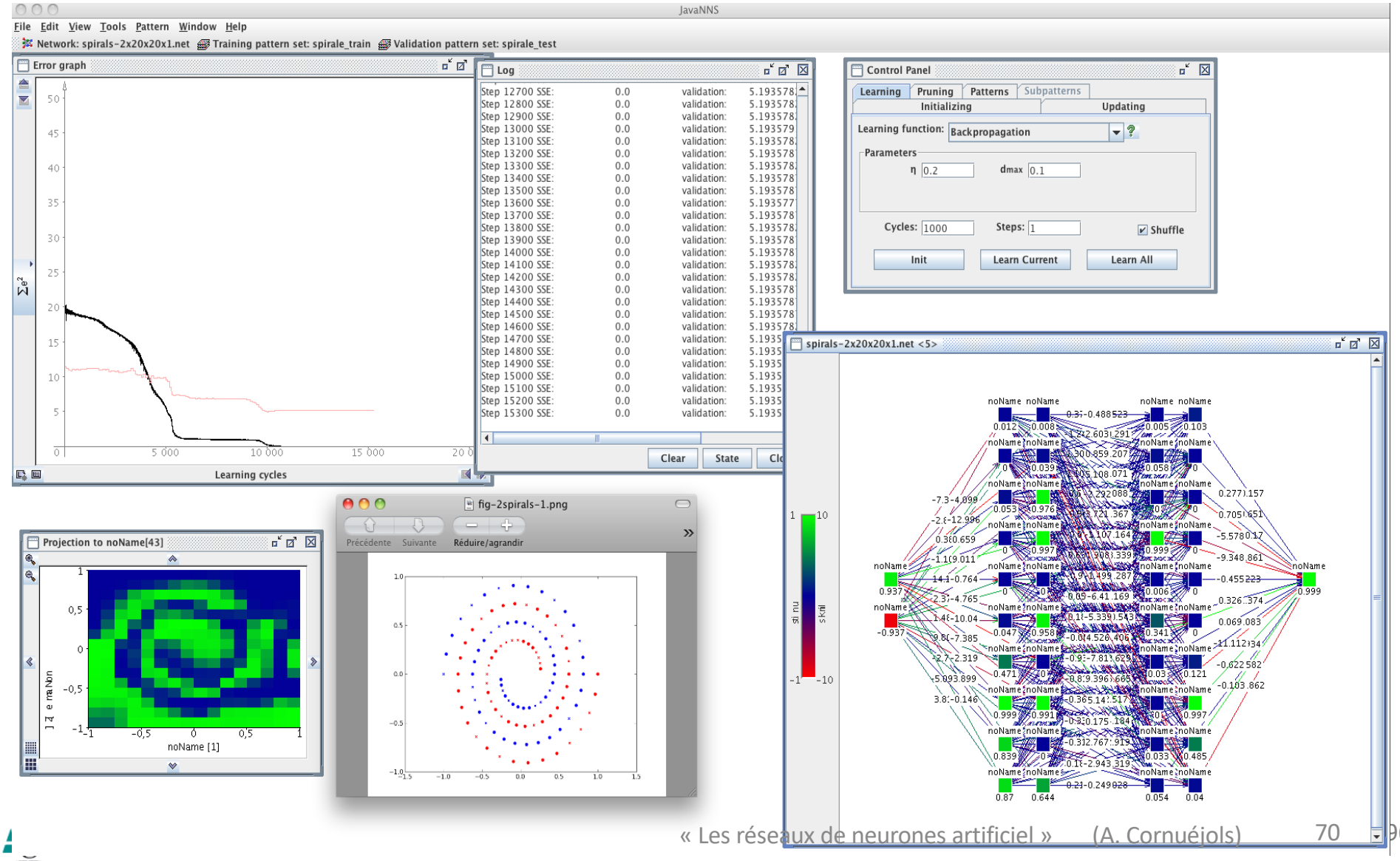
Illustration



Illustration



Illustration



Les **neurones** des couches cachés : **que représentent-ils** ?

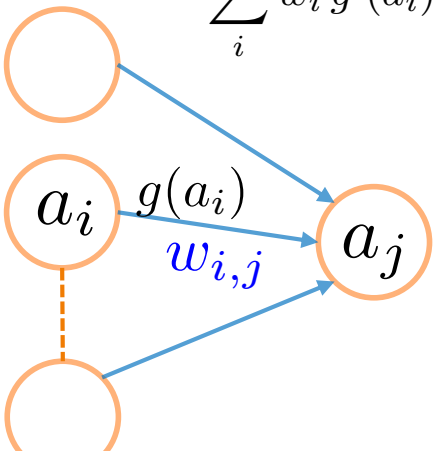
- Dans la descente de gradient par rétro-propagation, symétrie :
  - entre les **poids** des connexions
  - et les **exemples**
  - On peut donc optimiser sur les **exemples** plutôt que sur les poids des connexions

Couche **cachée** :

$$a_j = \sum_i w_{i,j}\, g(a_i)$$

$$\frac{\partial a_j}{\partial x^{(k)}} = \sum_i w_i \frac{\partial g(a_i)}{\partial x^{(k)}} = \sum_i w_i \frac{\partial g(a_i)}{\partial a_i} \frac{\partial a_i}{\partial x^{(k)}}$$

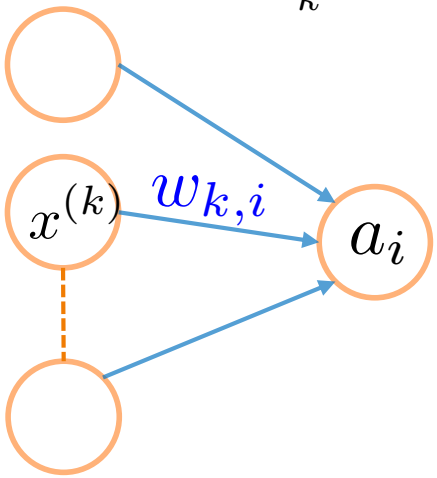
$$= \sum_i w_i\, g'(a_i) \frac{\partial a_i}{\partial x^{(k)}}$$



Couche **d'entrée** :

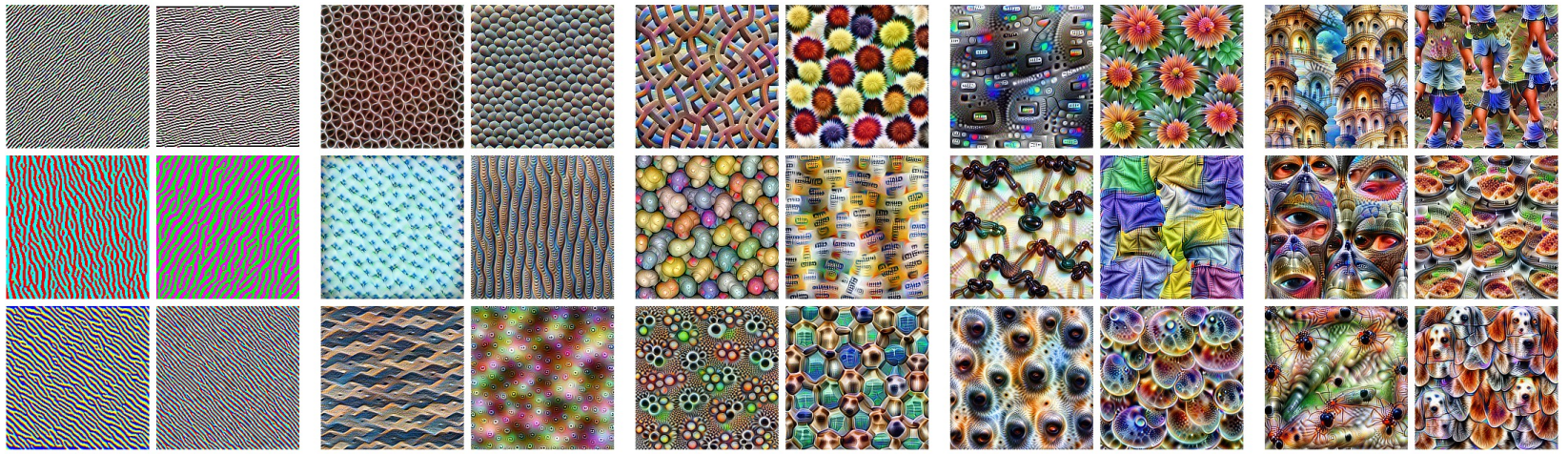
$$a_i = \sum_k w_{k,i}\, x^{(k)}$$

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



Les **neurones** des couches cachés : **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



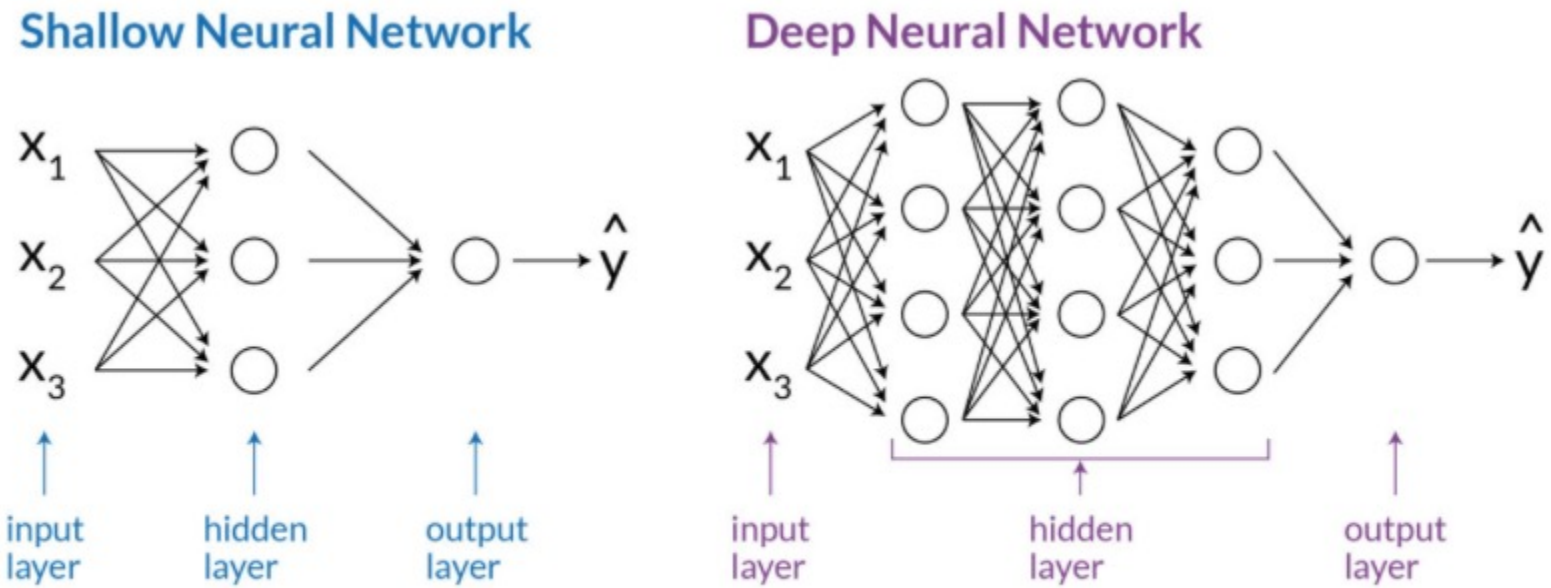
https://distill.pub/2017/feature-visualization/

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

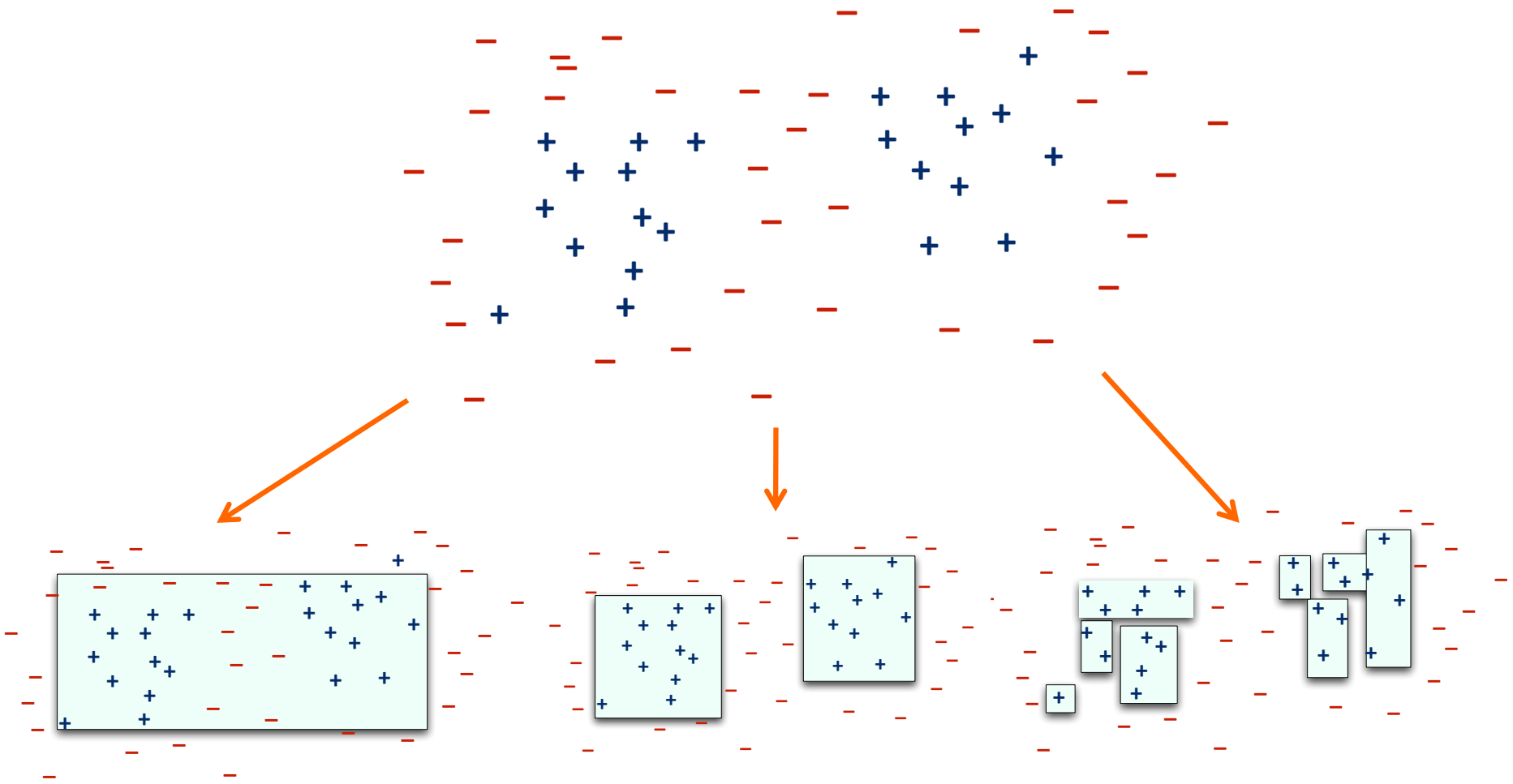
Comment choisir l'architecture du RN ?

- Contrôle la performance en généralisation



Shallow and Deep Neural Networks.

Illustration



Sous-apprentissage

Modèle « correct »

Sur-apprentissage

La régression et le sur-apprentissage

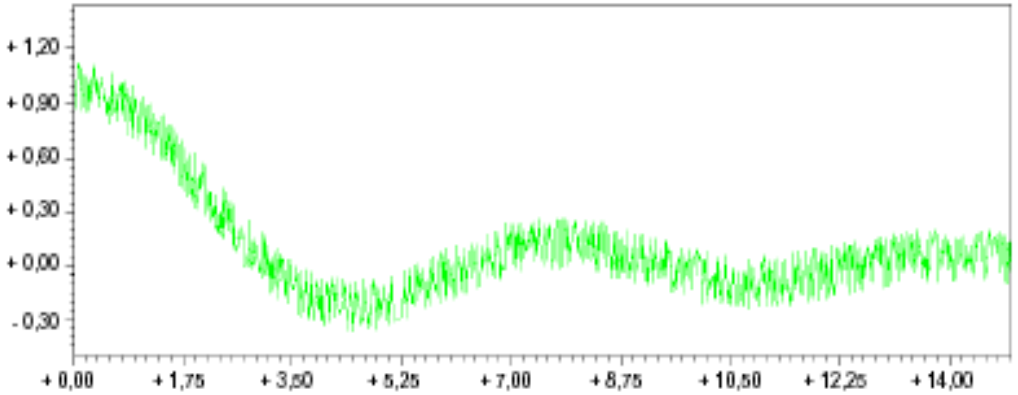


Figure 1-10. Un signal que l'on voudrait modéliser

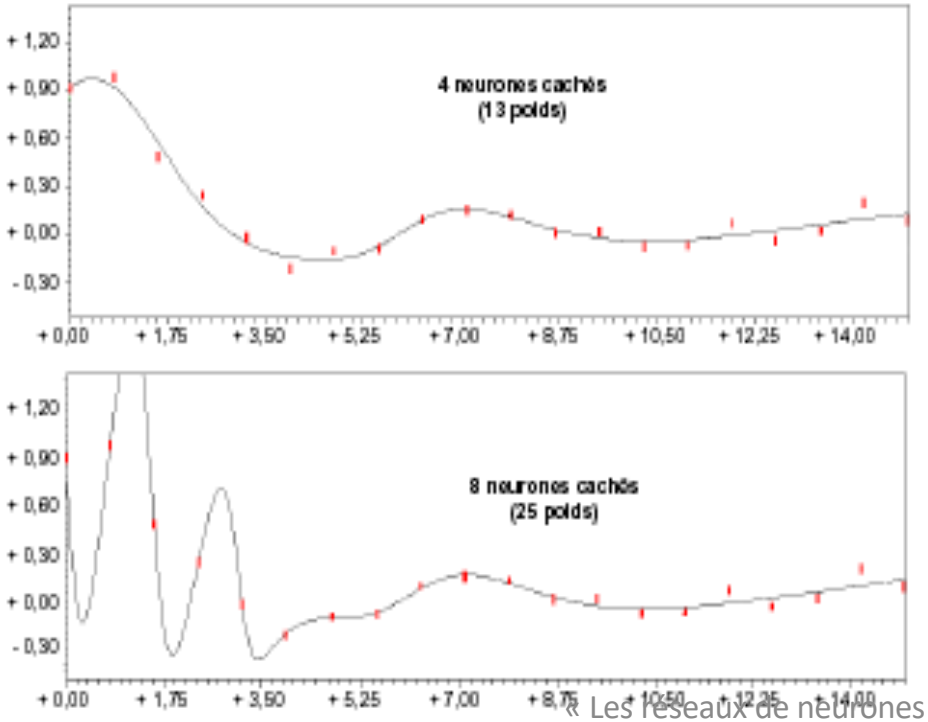
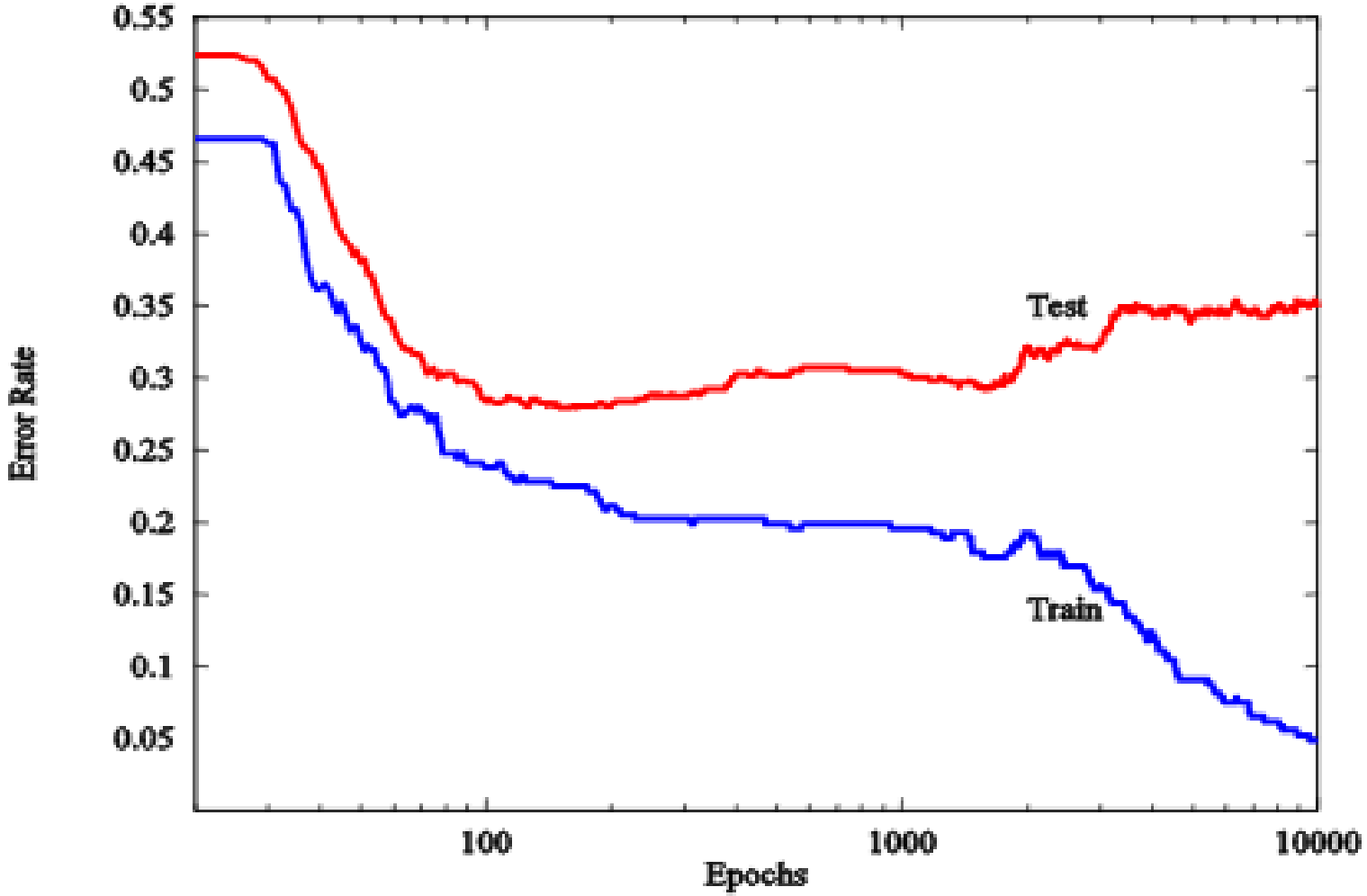
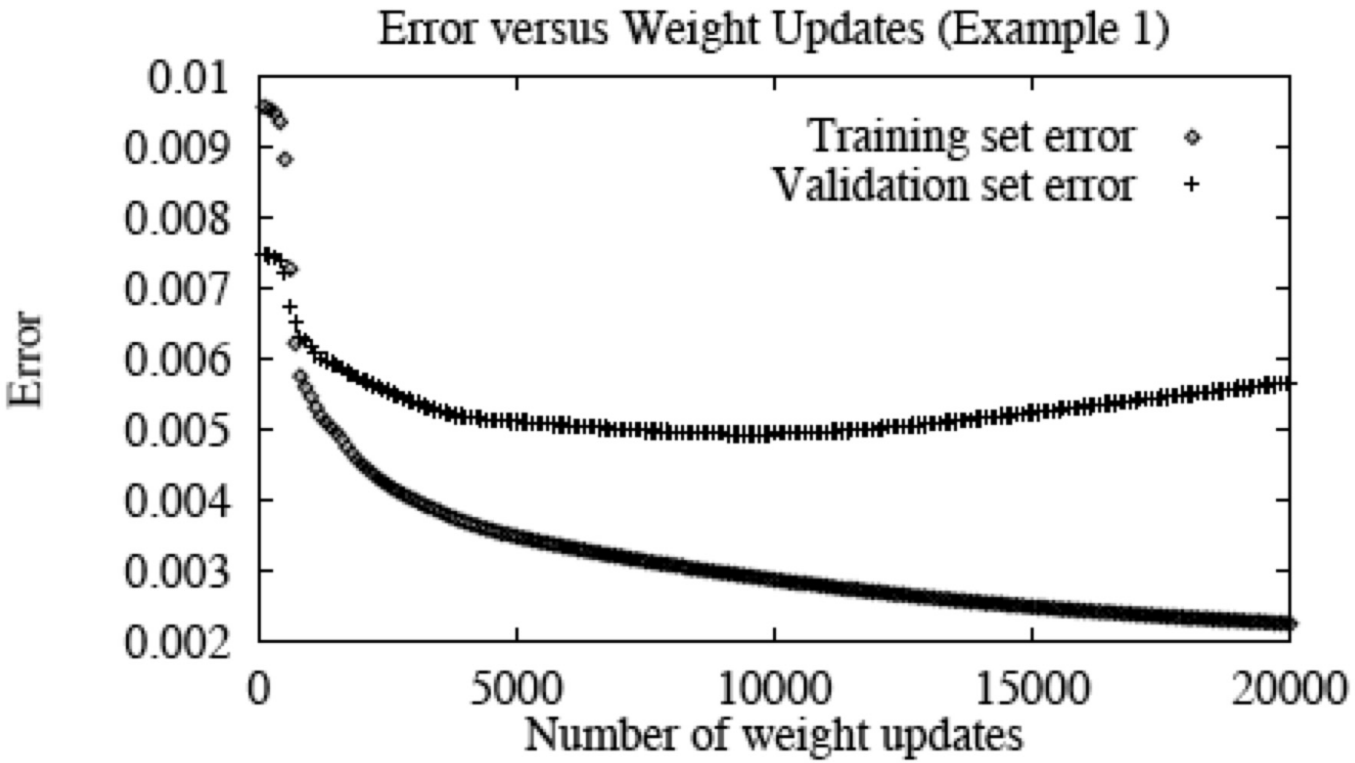


Figure 1-14. Toutes choses égales par ailleurs, le réseau de neurones le plus parcimonieux possède les meilleures propriétés de généralisation.

Sur-apprentissage



Sur-apprentissage (RN)

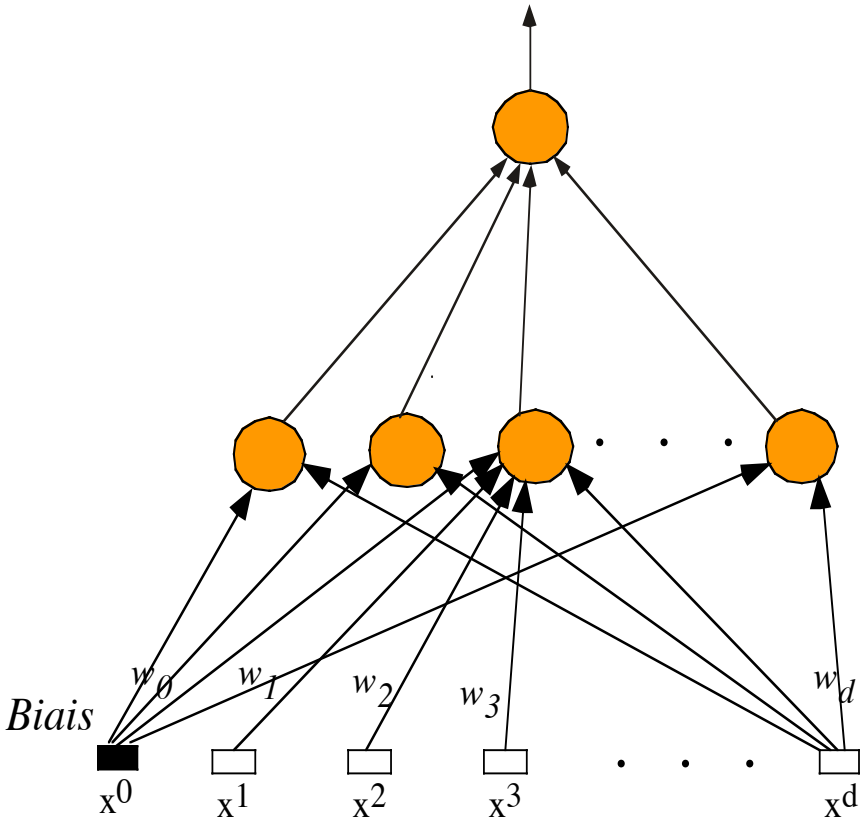


- Courbes pour 1 000 exemples

- *Quelles courbes si on avait 2 000 exemples ?*

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

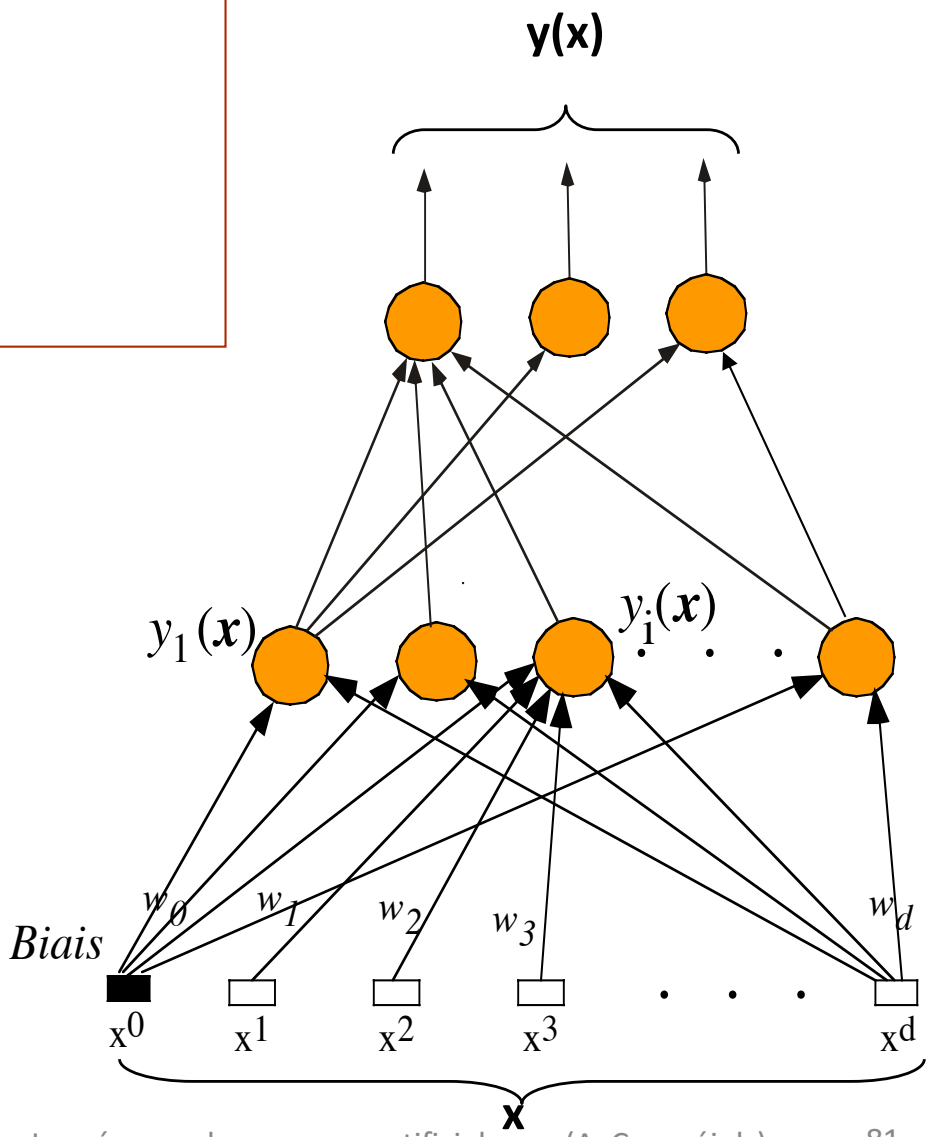
$\hat{y}_i$ is the output of the unit for the example x_i.
It can be considered as the probability that the input is of class *C*

Le codage de la **couche de sortie**

Exemple :

- Reconnaissance de caractères manuscrits
- Reconnaissance de locuteurs

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



ECOC

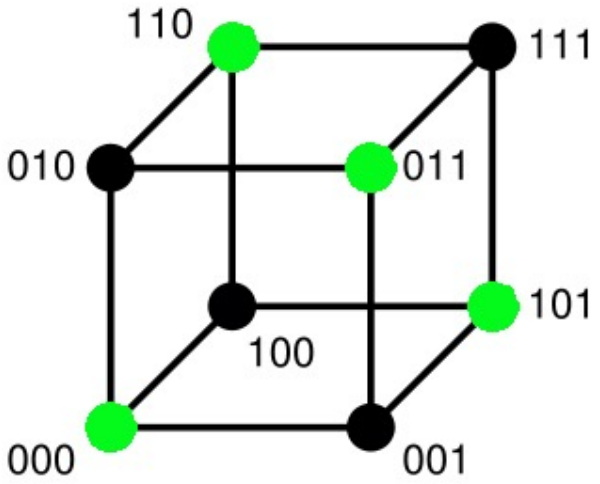
- Apprendre à distinguer 26 lettres
- Codage
  - Sur 5 bits ?
  - Sur 10 bits !

	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 4 6

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 $h_1(\boldsymbol{x})$, $h_2(\boldsymbol{x})$, $h_3(\boldsymbol{x})$, $h_4(\boldsymbol{x})$, $h_5(\boldsymbol{x})$, $h_6(\boldsymbol{x})$, $h_7(\boldsymbol{x})$, $h_8(\boldsymbol{x})$, $h_9(\boldsymbol{x})$, $h_{10}(\boldsymbol{x})$

- On décode $H(\boldsymbol{x})$ par plus proche voisin / au codage de A, B, C, …

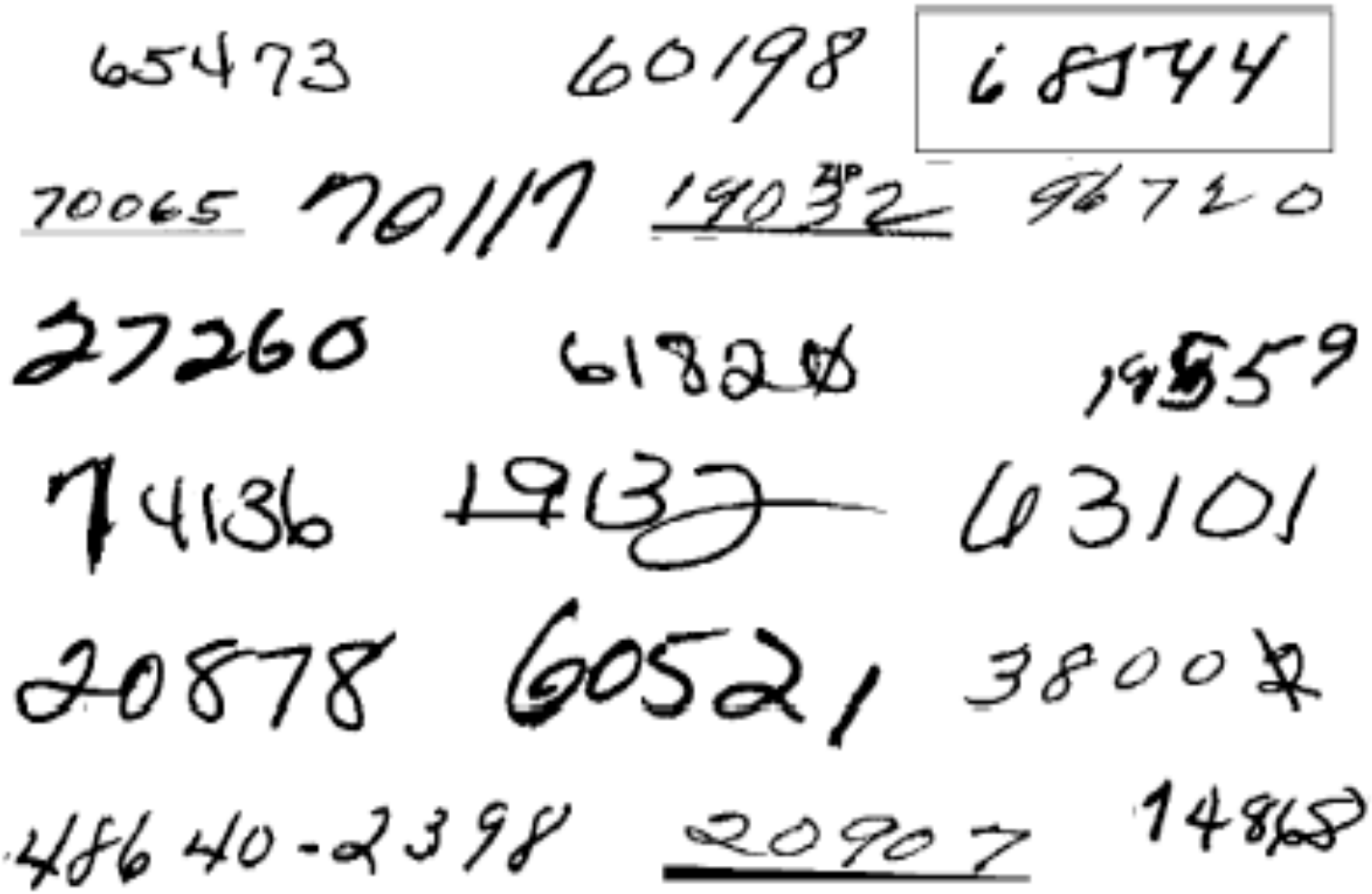
L'ancêtre des

réseaux convolutionnels

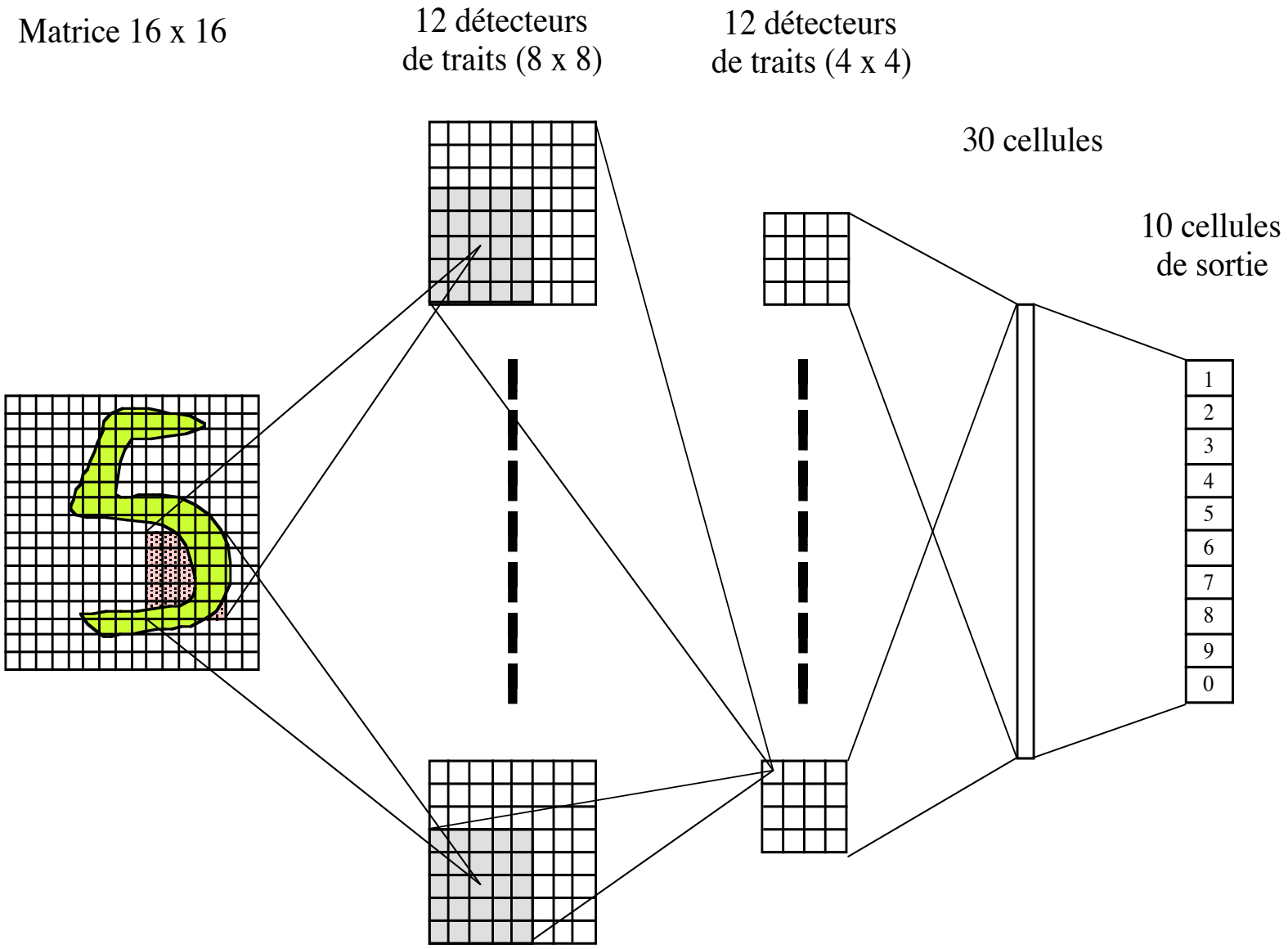
Application aux codes postaux (Zip codes)

- [Le Cun et al., 1989, ...] (ATT Bell Labs : très forte équipe)
- ≈ 10000 exemples de chiffres manuscrits
- Segmentés et redimensionnés sur matrice 16 x 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)

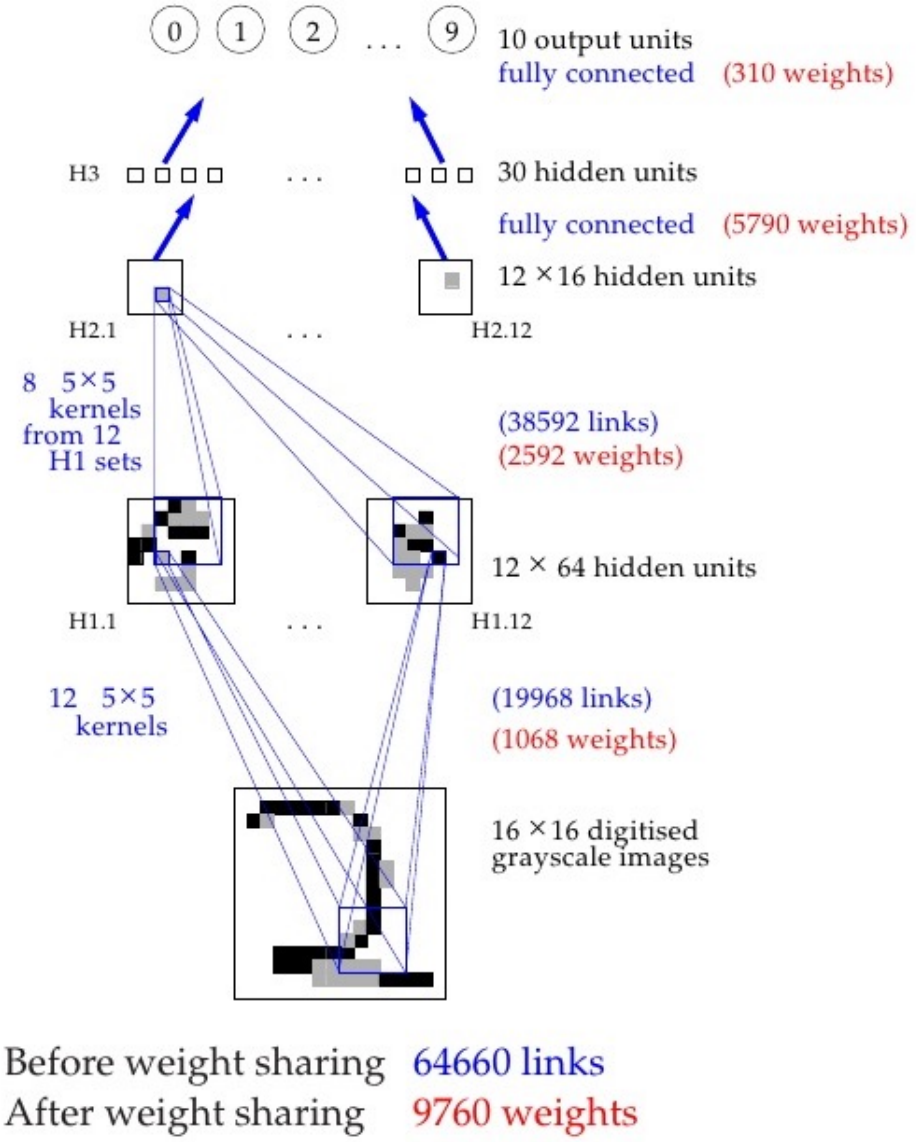
La base de données



Réseaux à convolution : Application aux codes postaux



Réseaux à convolution : Application aux codes postaux



- 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

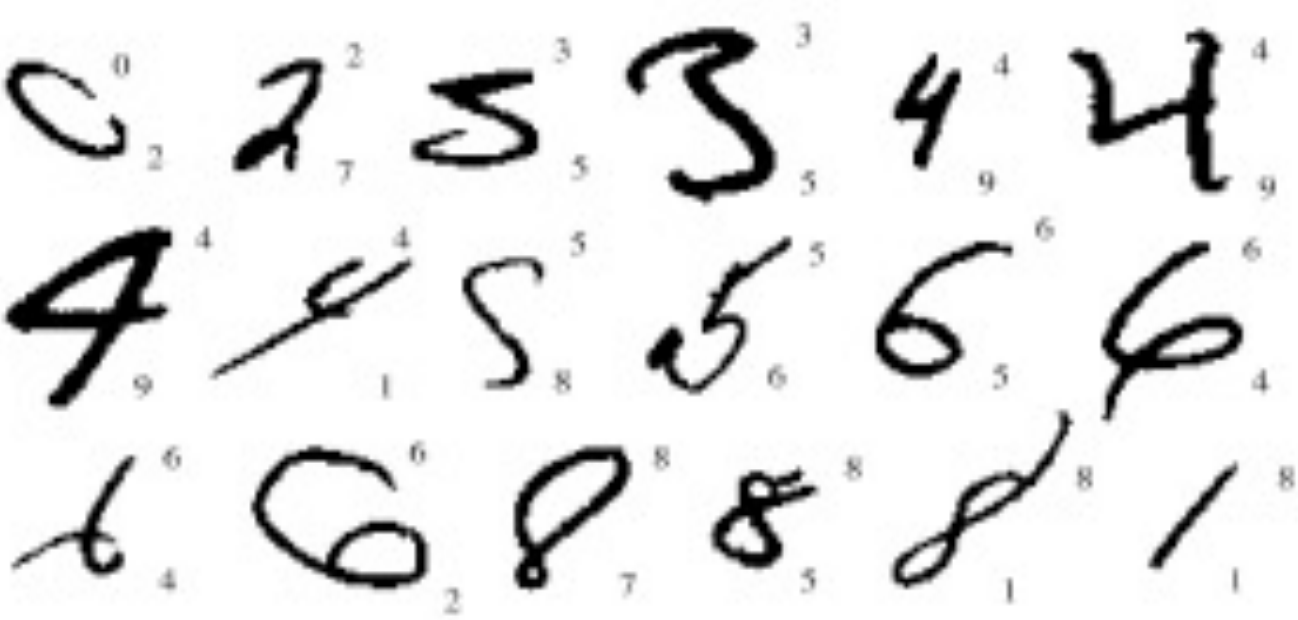


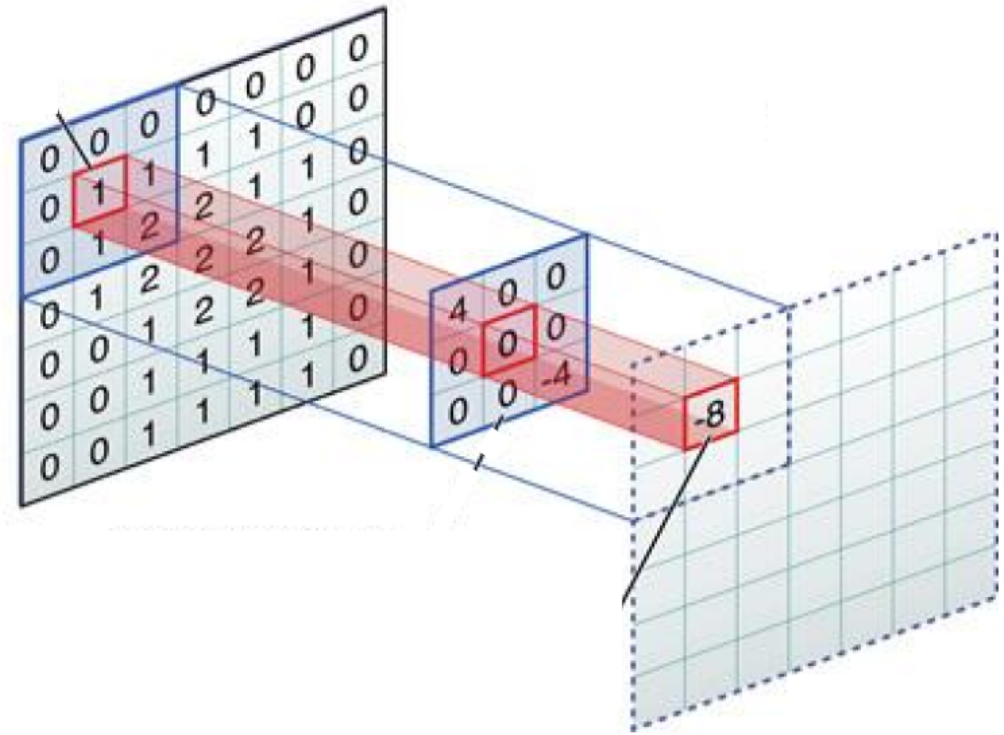
Figure 1-34. Les 18 erreurs de classification commises par séparation linéaire des classes deux à deux. Pour chaque chiffre manuscrit, l'indication en haut à droite est la classe d'appartenance du chiffre indiquée dans la base, et le chiffre en bas à droite est la classe affectée par le classifieur.

Examples of prediction error

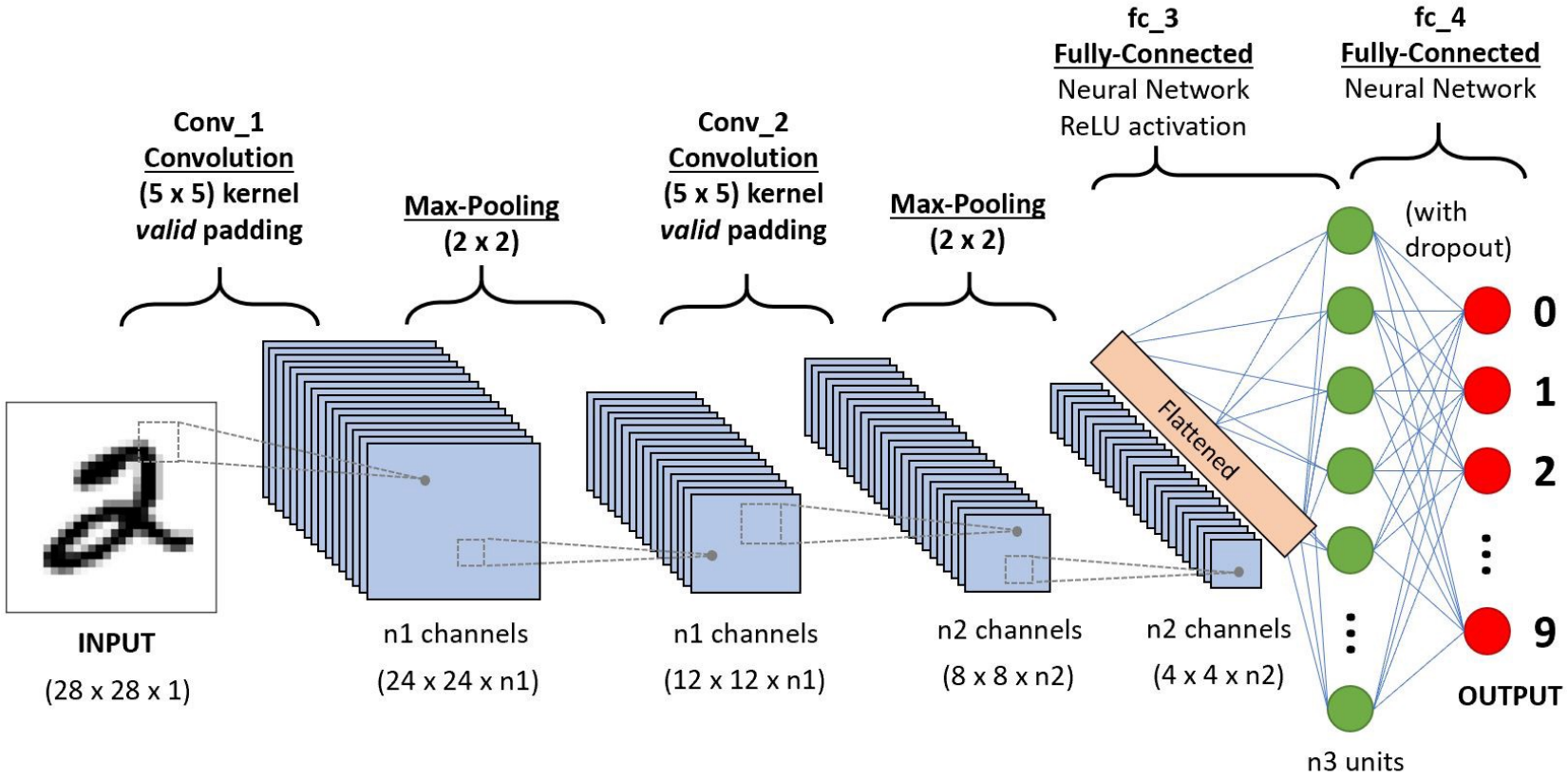
Convolutions

Same dimension between layers

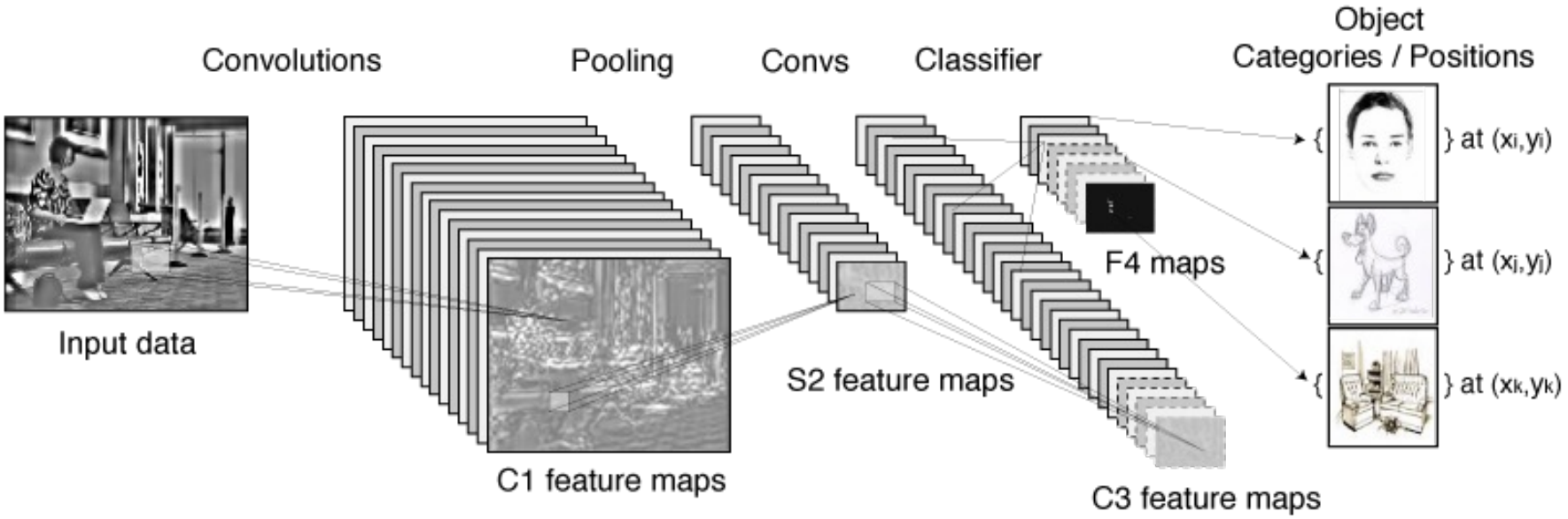
(4x0) + (0x0) + (0x0) + (0x0) + (0x1) + (0x1) + (0x0) + (0x1)+ (-4x2) = -8



Réseaux de neurones convolutionnels



Réseaux de neurones convolutionnels (2° exemple)



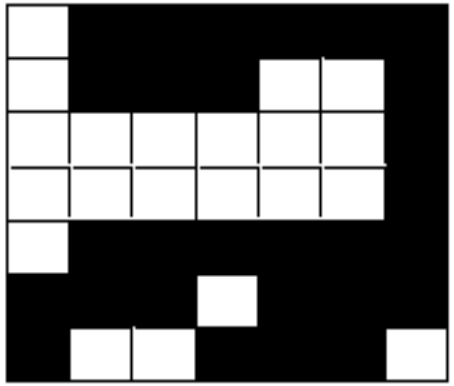
1) Le **pooling** consiste réduire la résolution des images filtrées, par exemple en moyennant les pixels contigus *(ex : 4 pixels deviennent 1 seul pixel qui contient une valeur moyenne)* - étape S2

2) **Convolution** : des filtres sont appris sur les nouvelles images - étape C3

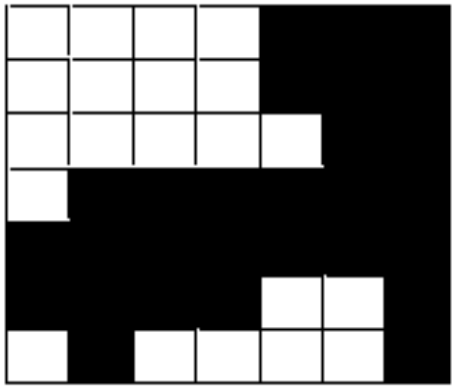
3) **Classifieur** : les dernières couches sont entièrement connectées *(i.e. MLP)* et apprennent à prédire la classe à partir des filtres appris *(i.e. des descripteurs générés automatiquement)* - étape F4

How to code the inputs

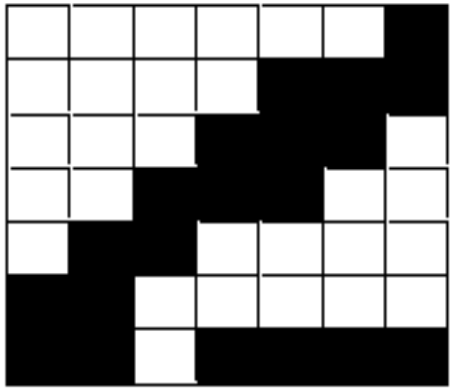
Learning is easy when we know what to look for



- Yes

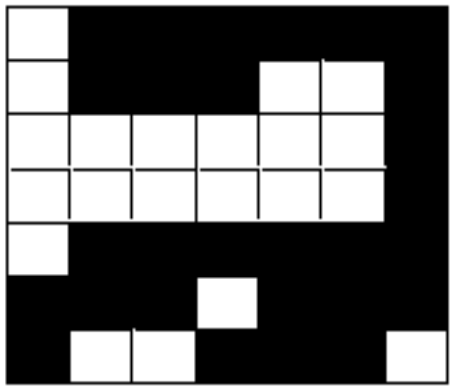


- Yes



- No

Inputs and prior knowledge



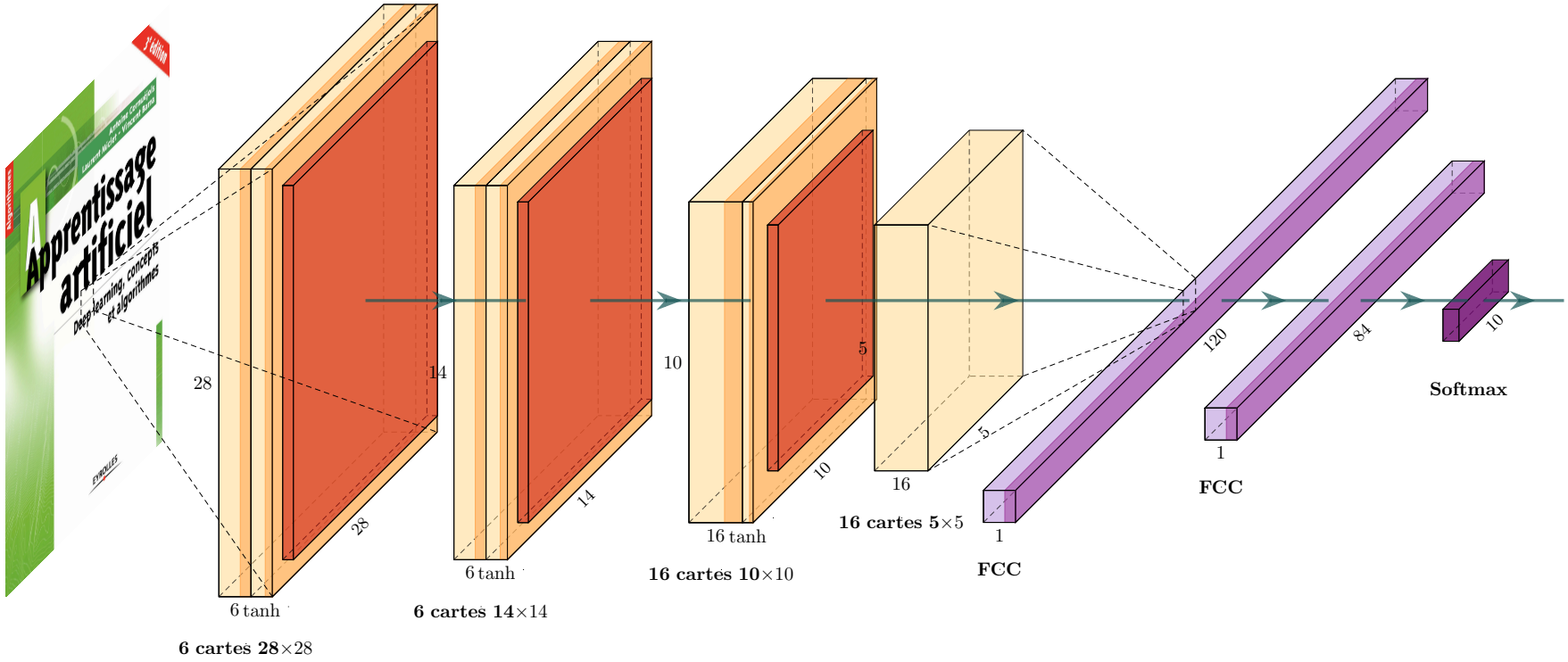
- Is it a pattern recognition task? A character recognition task? …
- *How to code the examples?*

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

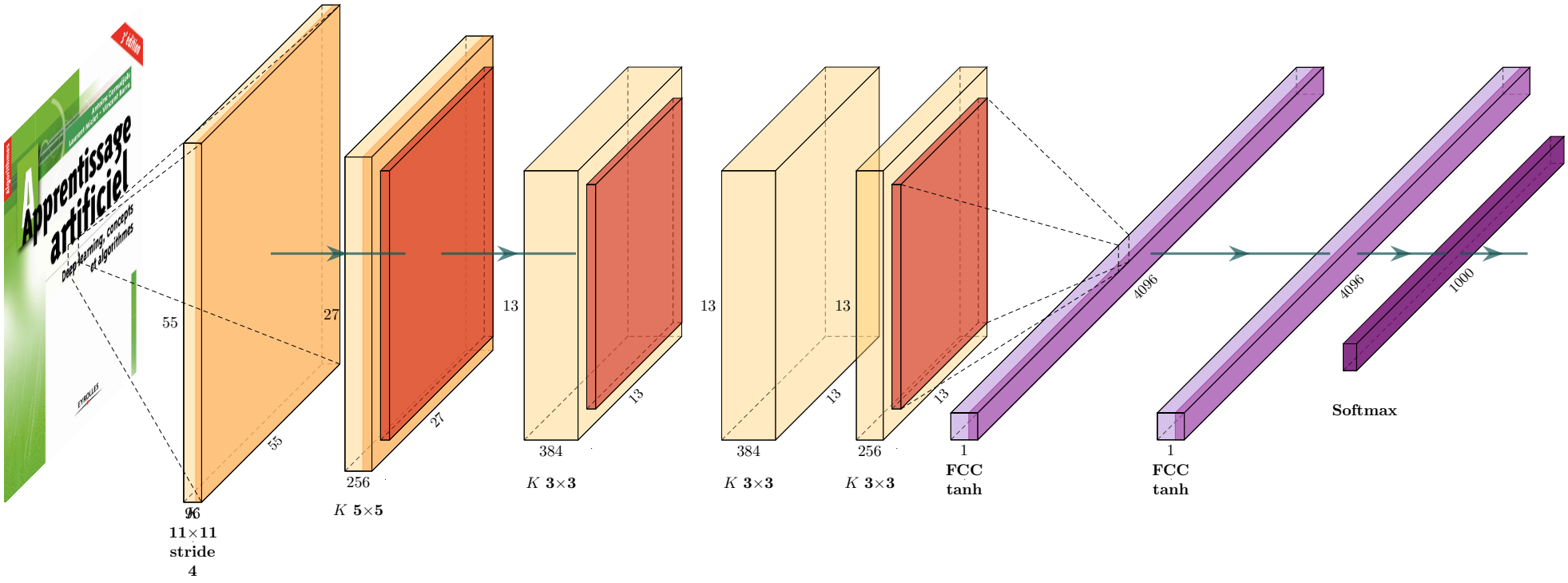
→ But how can we know the right representation?

LeNet-5



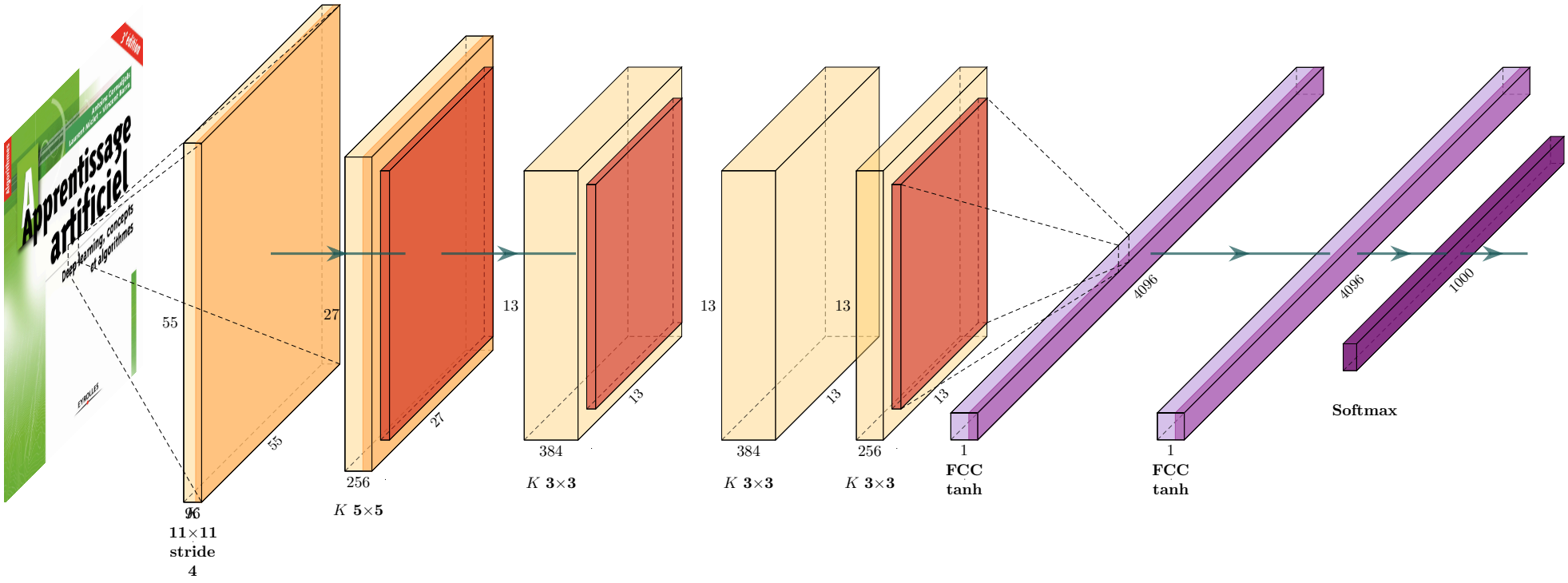
- Architecture du réseau LeNet-5.
  - Les couches de **convolution** et d'activation sont en orange clair.
  - Les couches d'**agrégation** en orange foncé
  - Les couches **complètement connectées** sont en violet

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**

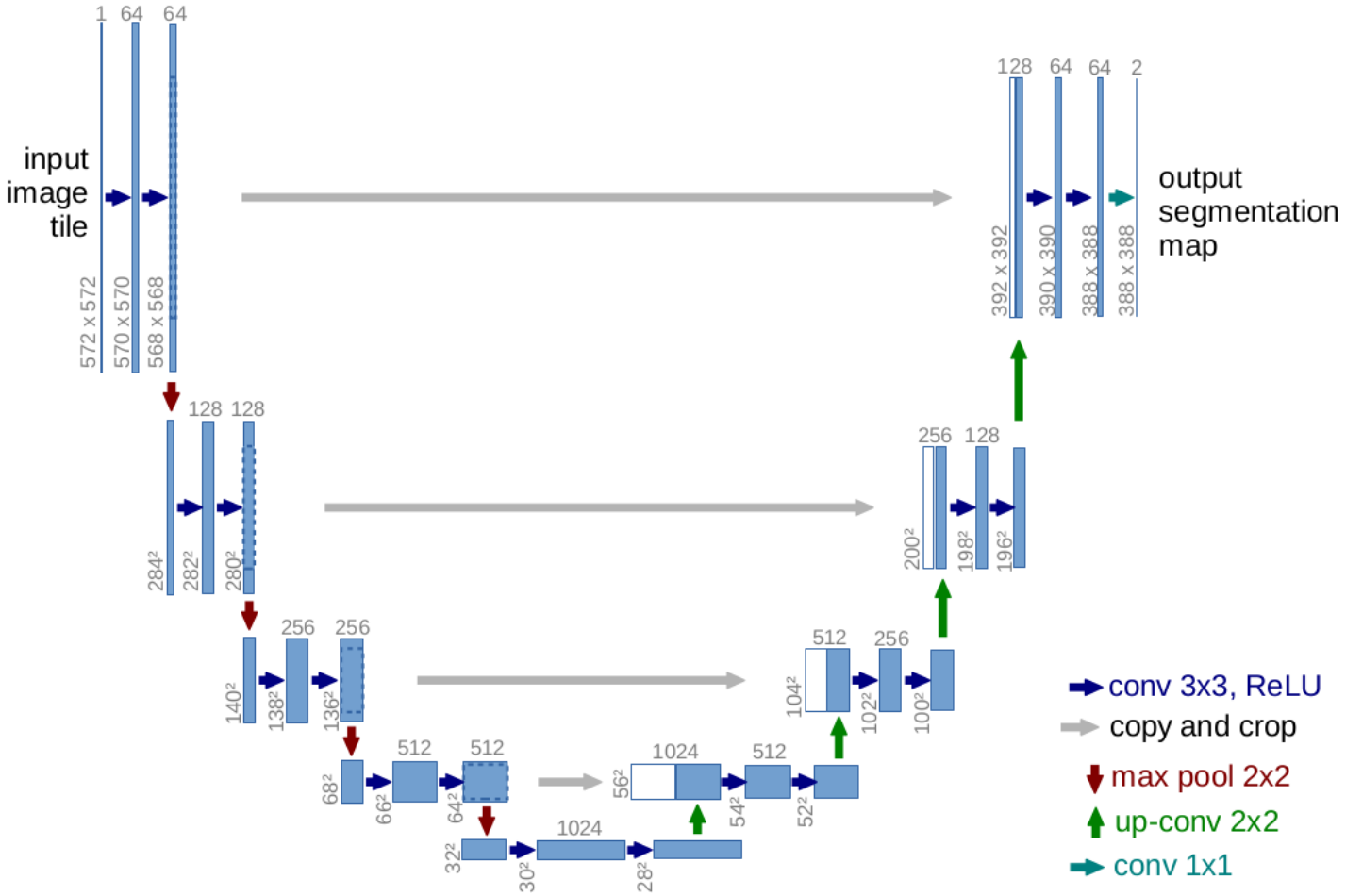
AlexNet



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

- En regardant uniquement la **première couche de convolution**, on constate que l'entrée est composée d'images 224 x 224 x 3, que les filtres de convolution sont de taille 11 et que le stride est de 4.
- Ainsi, 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.

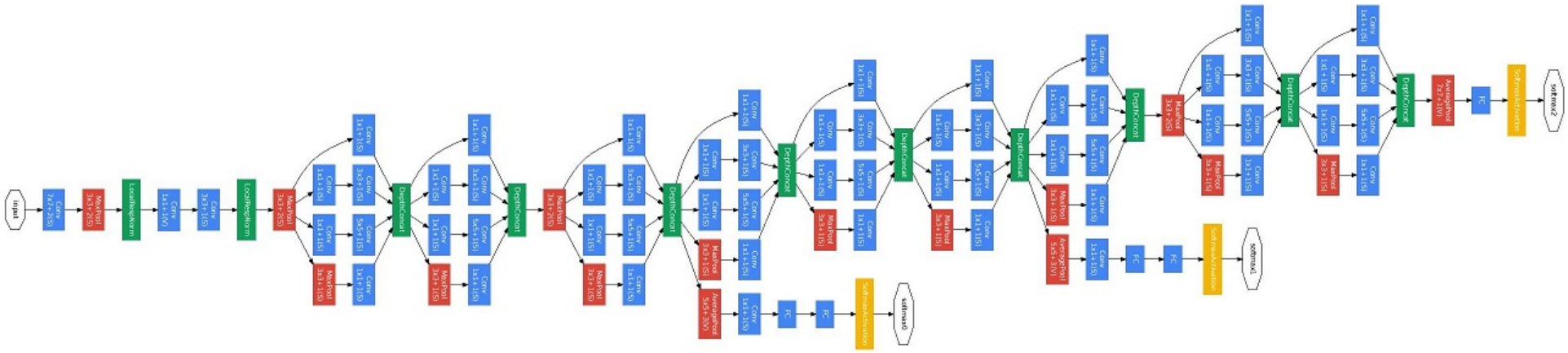
U-Net : pour la segmentation d'images



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



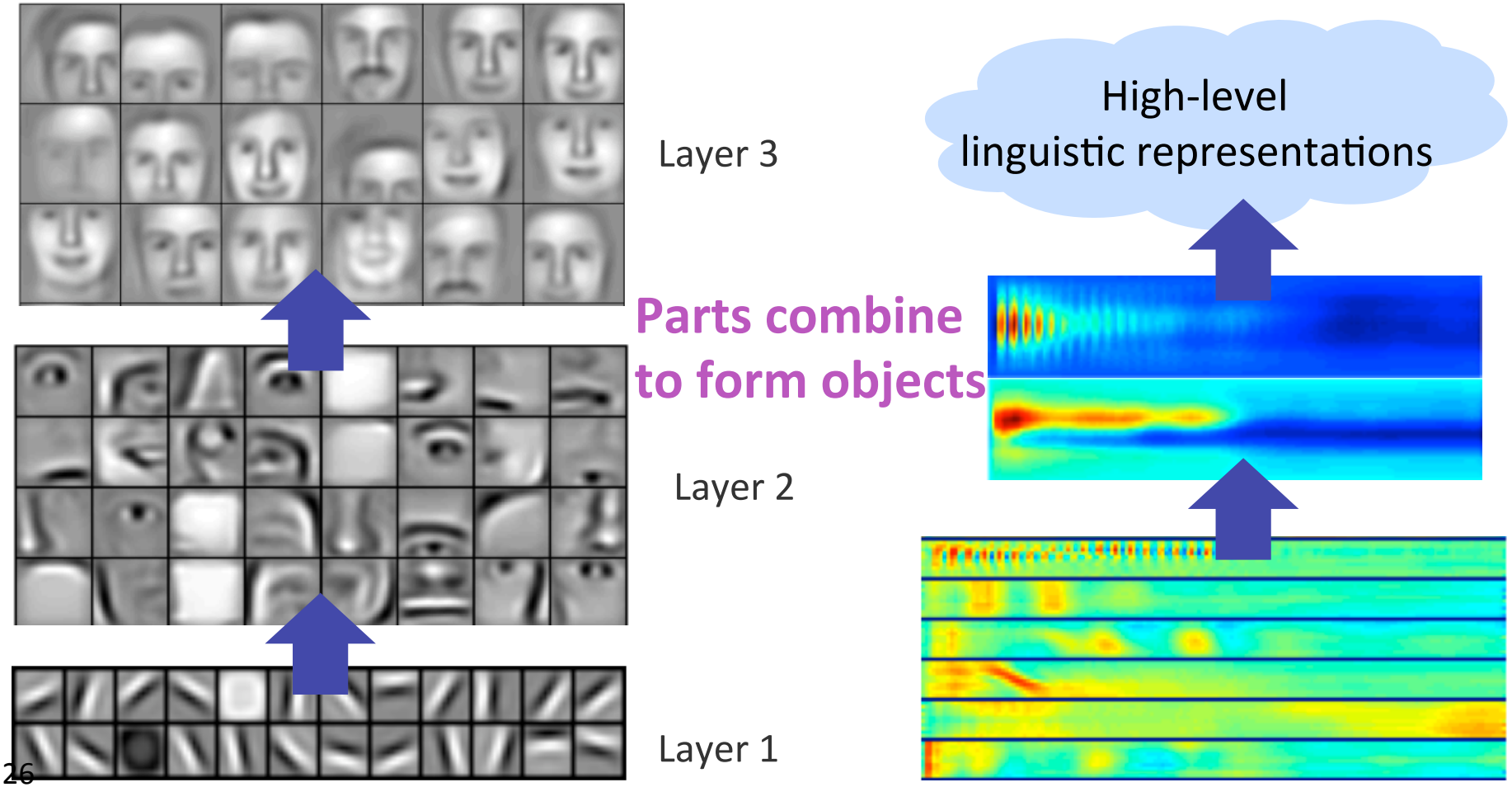
Illustration

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

Apprentissage de représentations hiérarchiques

- Apprentissage de représentations hiérarchiques



26

Illustration : ImageNet

La compétition ImageNet

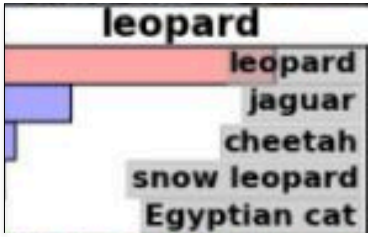
- Plus de **15M d'images** haute résolution étiquetées
- Environ **22K catégories**
- Récoltées sur le Web et étiquetées par Amazon Mechanical Turk

Illustration : ImageNet

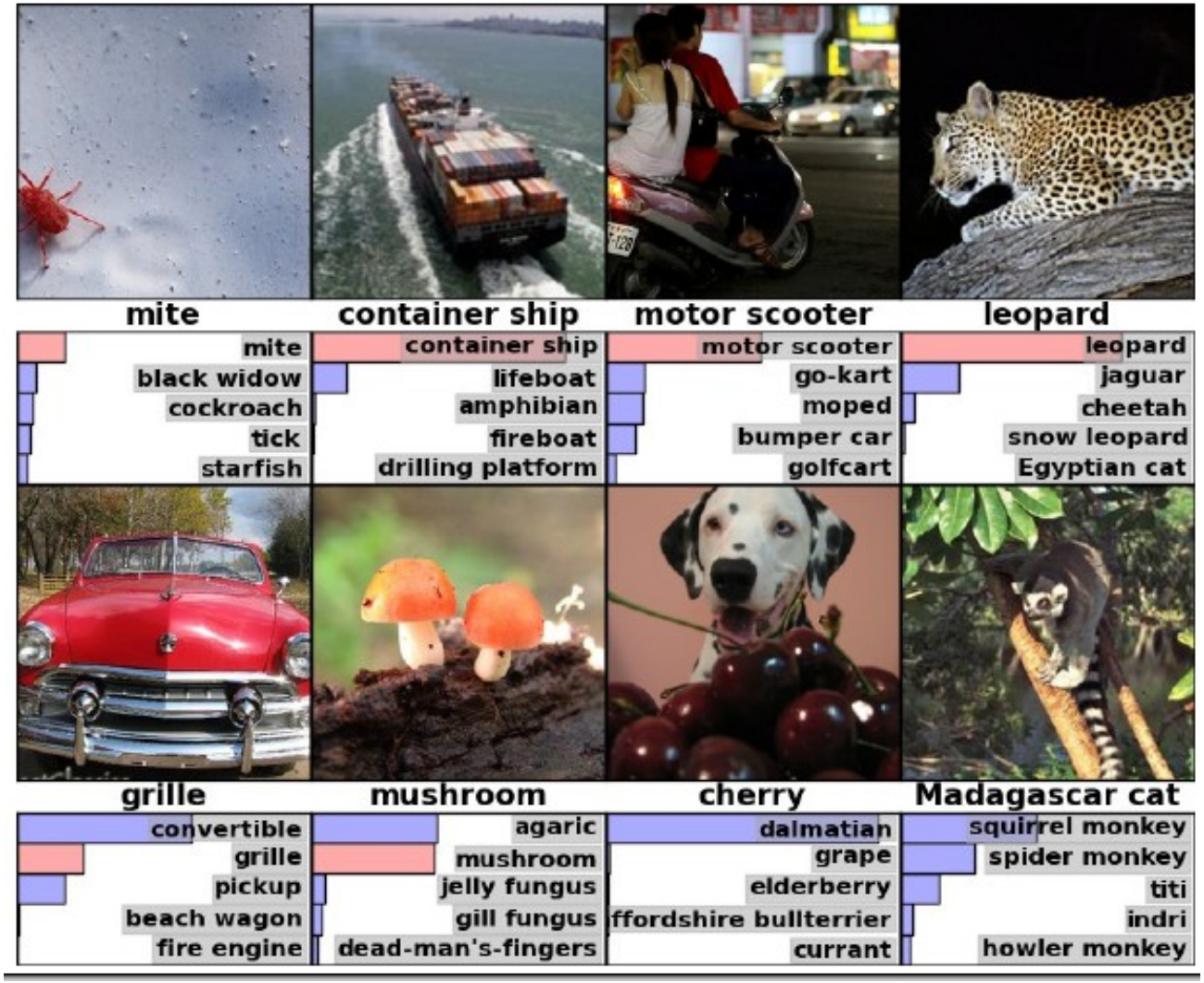
La compétition ImageNet

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

Classification



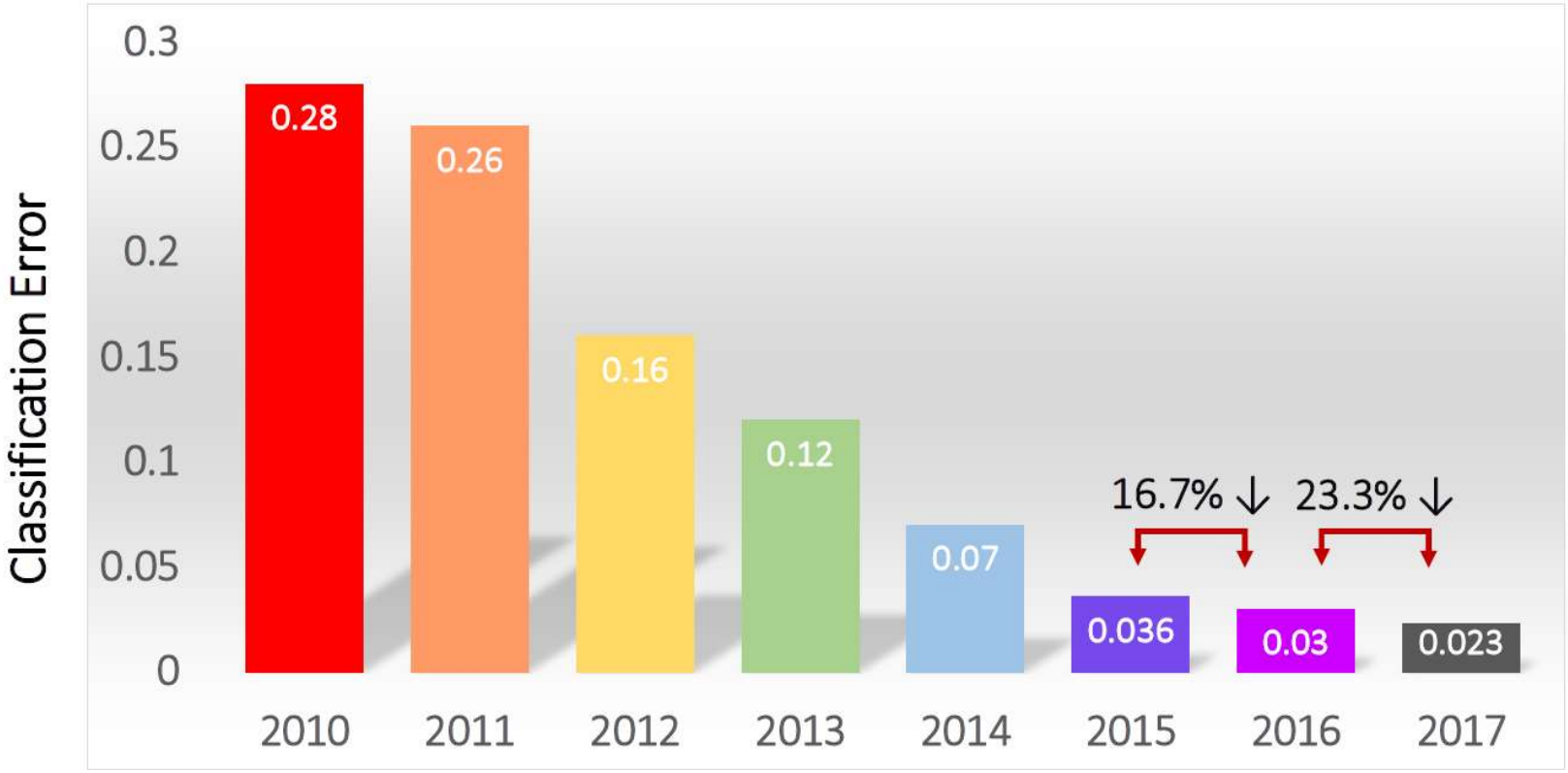
Object recognition



[Krizhevsky, Sutskever and Hinton (2012)]

Les performances sur les compétition

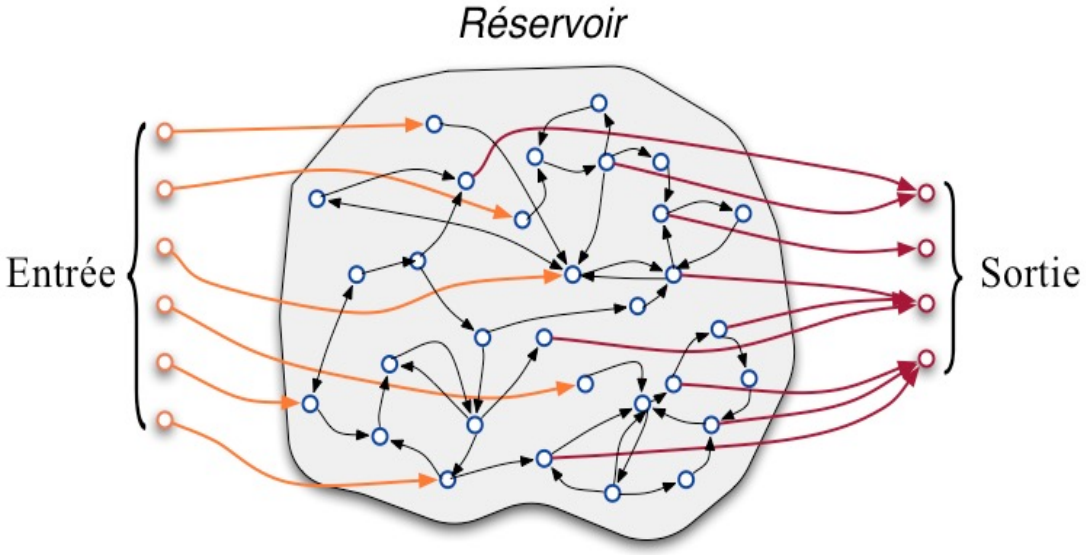
- Résultats en classification



Autres architectures

Une idée intrigante : le « reservoir computing »

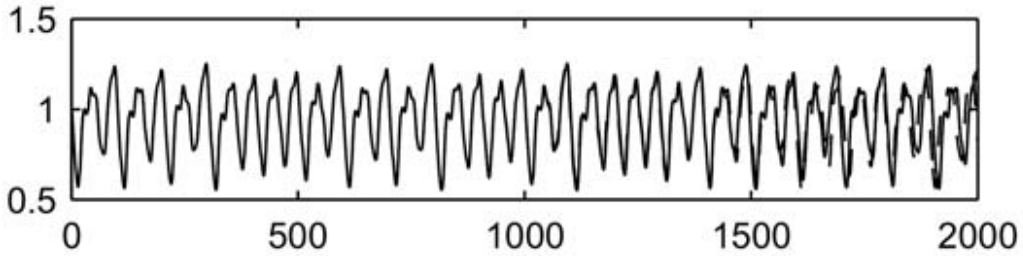
- Idée :
  - Utiliser un réseau récurrent sans l'entraîner explicitement
  - Entraîner une seule couche de sortie

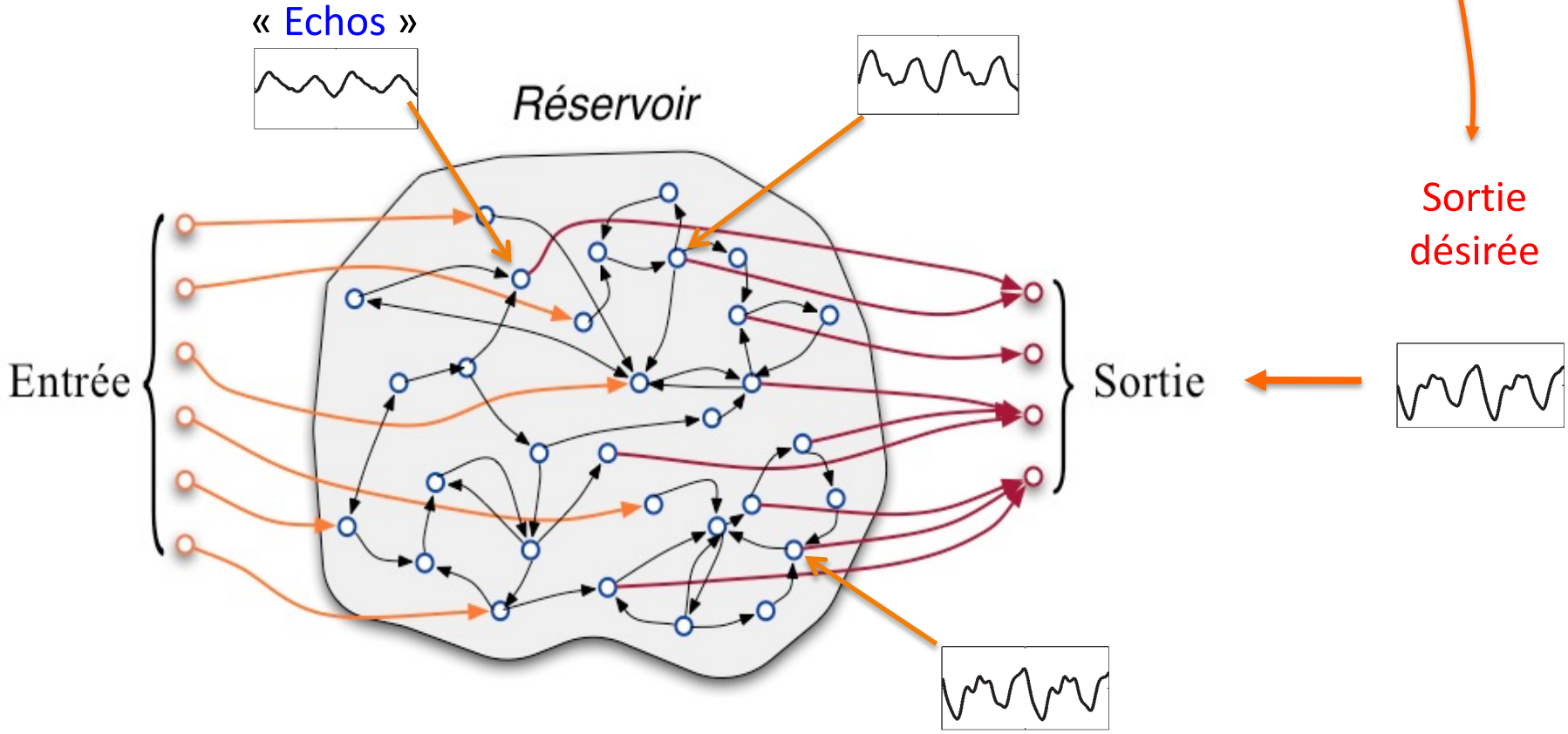


- Ré-introduit une « dynamique »
  - Séries temporelles

Prédiction : « reservoir computing »

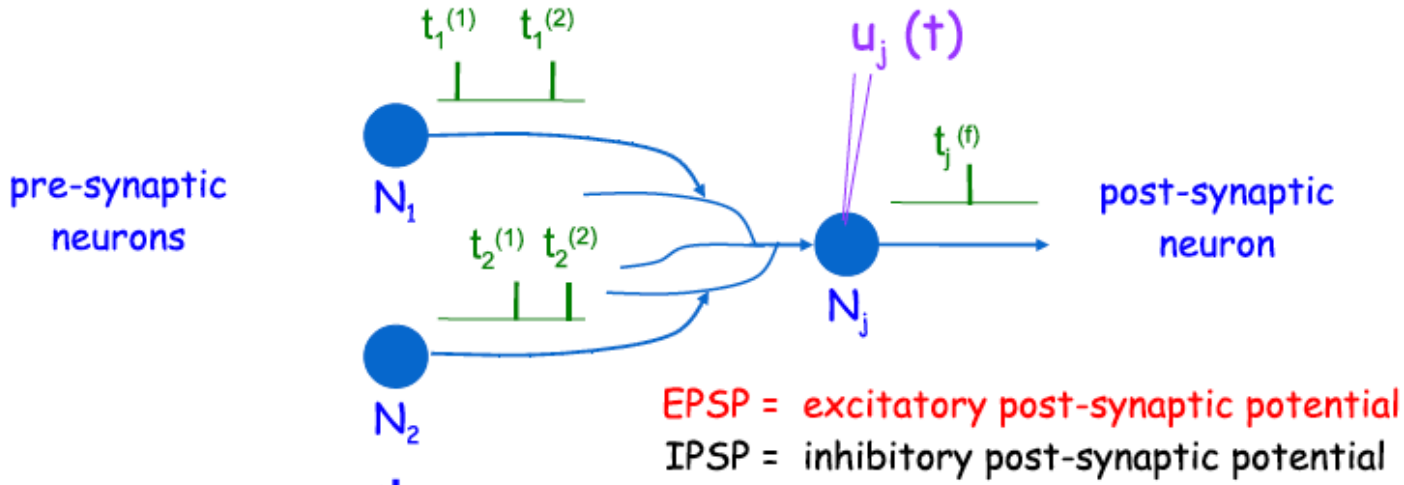
Signal à « apprendre »



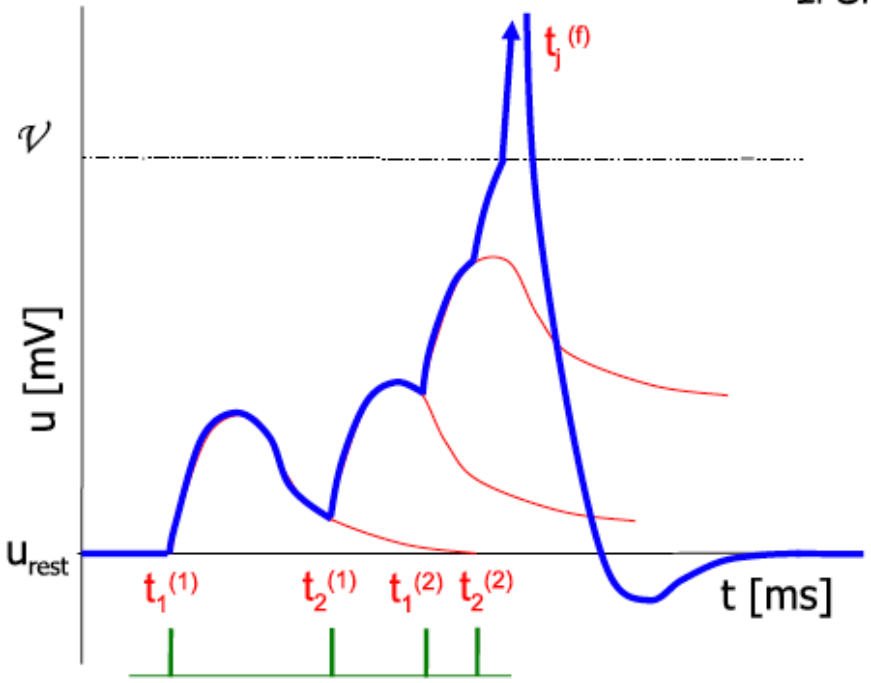


Les réseaux à neurones impulsionnels

- *Spiking neural networks*



EPSP = excitatory post-synaptic potential
IPSP = inhibitory post-synaptic potential



each pre-synaptic spike generates an EPSP (red curves) or IPSP in case of negative weight ;
they are all added (blue line) ;

when u_j (t) reaches the threshold $\mathcal{V}$ the post-synaptic neuron N_j emits a spike, in its turn.

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.