# Master 2 Recherche
# Apprentissage Statistique et Optimisation

François Yvon – Michèle Sebag
Alexandre Allauzen – Marc Schoenauer

http://www.limsi.fr/Individu/allauzen/wiki/index.php/TSI09 – http://tao.lri.fr/tiki-index.php

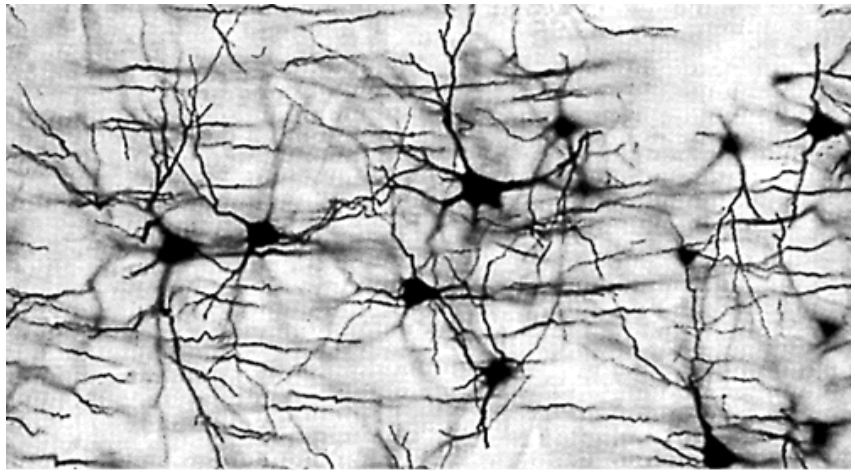
7 octobre 2010: Réseaux Neuronaux

# Réseaux Neuronaux

1. Pourquoi ?
2. Réseaux classiques
3. Quelques applications
4. Réseaux modernes: profonds, à échos, impulsionnels, liquides...
5. Pistes de recherche fraiches

Emprunts: Hélène Paugam-Moisy, Francois Yvon, Yann Le Cun, Yoshua Bengio.

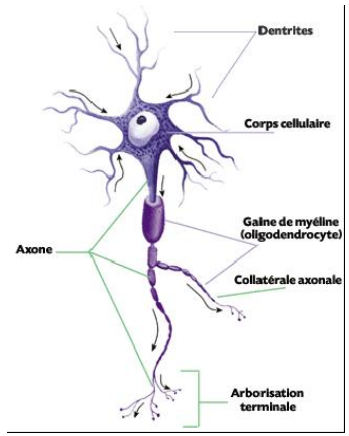
# Pourquoi ? 1. La fascination biologique



## Faits

- $10^{11}$ neurones
- Chacun connecté à $10^4$ autres
- Temps déclenchement $\sim 10^{-3}$ seconde

$10^{-10}$ computers



# La fascination biologique, 2

Croyances

- Le cerveau humain est inégalé
  Hum. Pour Sebastien Thrun, le programme de l'IA est résolu
  - à 80% pour le raisonnement
  - à 40% pour le dialogue
  - à 10% pour la perception
- Mais comment fait-il ?
  - La question n'est pas la quantité de neurones
    on pouvait le croire en 80, en 90...
  - Est-ce le massivement parallèle ?
  - Est-ce l'inné ? Tout ce qu'on ne sait pas expliquer
  - Est-ce la procédure/nature d'entrainement ?
    Chats et rayures

# La fascination biologique, 3

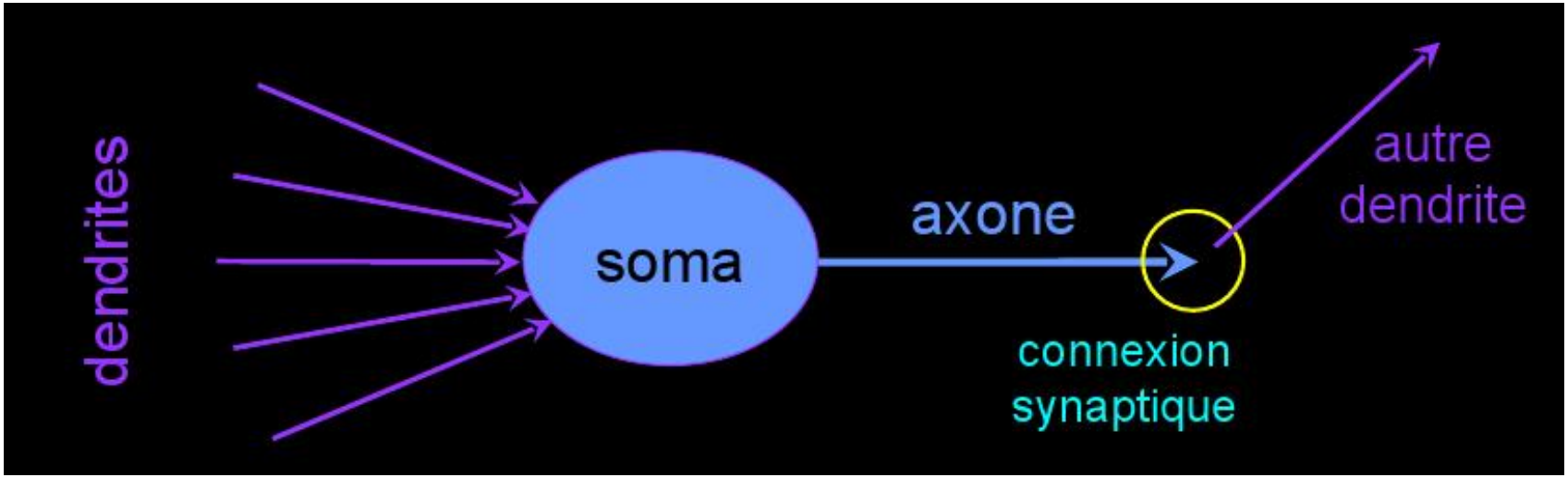
Limites des connaissances

- Finitude de la population de neurones
- Le neurone grand-mère ?
- Neurones miroirs

Savoir qu'un savoir incomplet peut être un leurre misleading

- Imiter les oiseaux pour voler

# Quelques détails




Output

Axone. Un neurone emet des impulsions sur son axone

Transmission

Impulsions transmises aux dendrites via des synapses (poids)

Processing

Corps du neurone: soma

# Plasticité synaptique

Hebb 1949 Conjecture

Quand un axone $A$ excite un neurone $B$ de manière répétée, un processus de croissance ou un changement métabolique s'installe dans $A$, dans $B$ ou dans les deux. tel que la capacité de $A$ à exciter $B$ augmente.

Règle d'apprentissage

Si deux neurones sont excités simultanément, le poids de leur interaction augmente.

Remarque: apprentissage non supervisé.

# Réseaux Neuronaux

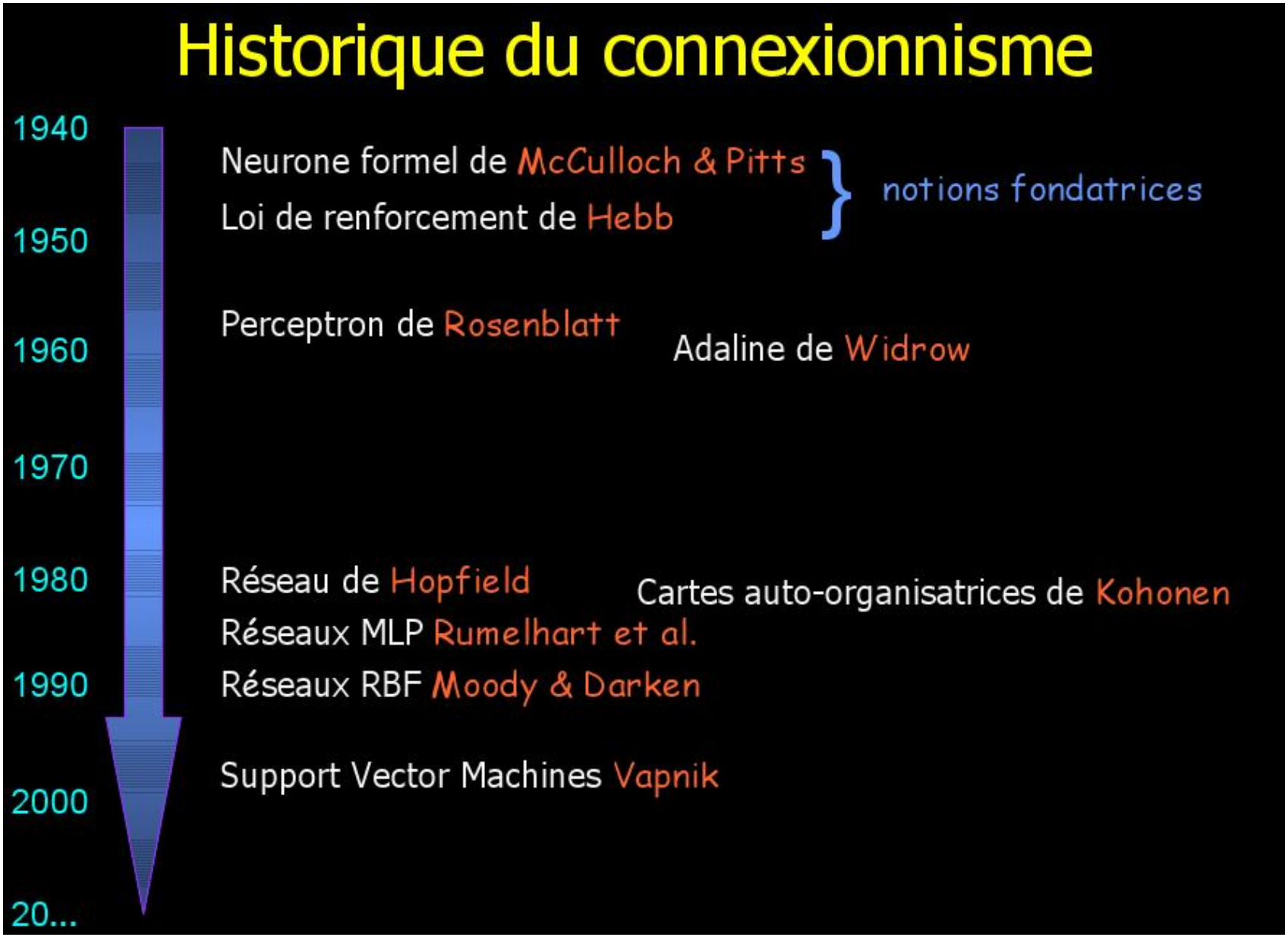
1. Pourquoi ?
2. Réseaux classiques
3. Quelques applications
4. Réseaux modernes: profonds, à échos, impulsionnels, liquides...
5. Pistes de recherche fraiches

# Réseaux Neuronaux Artificiels

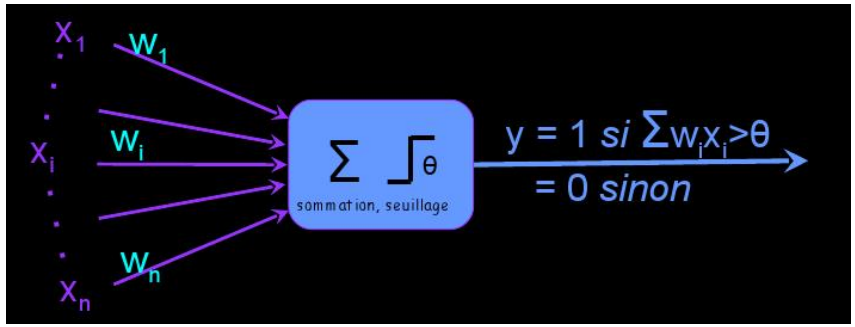
## Réseaux classiques

1. Histoire
   - Neurones logiques/non supervisés
   - Perceptron, Adaline: algorithmes
   - L'hiver des NN: limites théoriques
   - Perceptron Multi-couches: algorithme et limites algorithmiques
2. Structure du réseau et apprentissage

# Histoire


Historique du connexionnisme
1940
1950
1960
1970
1980
1990
2000
20...
Neurone formel de McCulloch & Pitts
Loi de renforcement de Hebb
notions fondatrices
Perceptron de Rosenblatt
Adaline de Widrow
Réseau de Hopfield
Cartes auto-organisatrices de Kohonen
Réseaux MLP Rumelhart et al.
Réseaux RBF Moody & Darken
Support Vector Machines Vapnik

# Neurones à seuil Mc Culloch et Pitt 1943



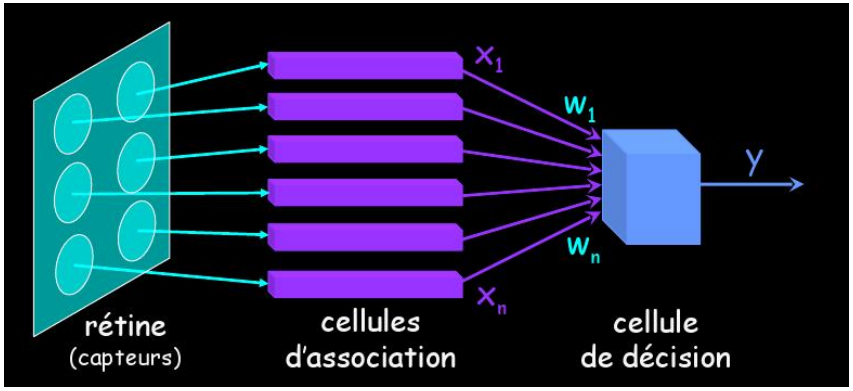

## Ingrédients

- Input (dendrites) $x_i$
- Weights $w_i$
- Threshold $\theta$
- Output: 1 iff $\sum_i w_i x_i > \theta$

## Remarques

- Neurones $\longrightarrow$ Logique $\longrightarrow$ Raisonnement $\longrightarrow$ Intelligence
- Réseau de neurones logiques: toute fonction booléenne
- Pas dérivable... (?!)

# Perceptron Rosenblatt 1958




$$y = signe(\sum w_i x_i - \theta)$$

$\mathbf{x} = (x_1, \ldots, x_d) \mapsto (x_1, \ldots, x_d, 1).$
$\mathbf{w} = (w_1, \ldots, w_d) \mapsto (w_1, \ldots w_d, -\theta)$

$$y = signe(< \mathbf{w}, \mathbf{x} >)$$

# Perceptron, Apprentissage

Etant donné

- $\mathcal{E} = \{(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{1, -1\}, i = 1 \dots n\}$

Pour $i = 1 \dots n$, Faire

- Si pas d'erreur ne rien faire

  pas d'erreur $\Leftrightarrow < \mathbf{w}, \mathbf{x} >$ meme signe que $y$

  $\Leftrightarrow y < \mathbf{w}, \mathbf{x} > \quad > 0$

- Si erreur

$$\mathbf{w} \rightarrow \mathbf{w} + y.\mathbf{x}_i$$

Pour stabilité algorithmique

$$\mathbf{w}_{t+1} \rightarrow \mathbf{w}_t + \alpha_t y.\mathbf{x}_\ell$$

$w_t$ décroit plus vite que $1/t$.

# Convergence de l'algorithme: borner le nombre d'erreurs

Hypothèses; Si

- $\mathbf{x}_i$ dans une boule $\mathcal{B}(C)$ $\quad ||\mathbf{x}_i|| < C$
- $\mathcal{E}$ est séparable, i.e.
  il existe une solution $\mathbf{w}^*$ s.t.
  $\forall i = 1 \dots n,\ y_i\ < \mathbf{w}^*, \mathbf{x}_i > > \delta > 0$

# Convergence de l'algorithme: borner le nombre d'erreurs

Hypothèses; Si

- $\mathbf{x}_i$ dans une boule $\mathcal{B}(C)$ $\quad ||\mathbf{x}_i|| < C$
- $\mathcal{E}$ est séparable, i.e.
  il existe une solution $\mathbf{w}^*$ s.t.
  $\forall i = 1 \ldots n,\ y_i < \mathbf{w}^*, \mathbf{x}_i > > \delta > 0$
  avec $||w^*|| = 1$.

# Convergence de l'algorithme: borner le nombre d'erreurs

Hypothèses; Si

- $\mathbf{x}_i$ dans une boule $\mathcal{B}(C)$ $\quad ||\mathbf{x}_i|| < C$
- $\mathcal{E}$ est séparable, i.e.
  il existe une solution $\mathbf{w}^*$ s.t.
  $\forall i = 1 \ldots n,\ y_i < \mathbf{w}^*, \mathbf{x}_i > > \delta > 0$
  avec $||w^*|| = 1$.

Alors L'algorithme du perceptron commet au plus $(\frac{C}{\delta})^2$ erreurs.

# Borner le nombre d'erreurs

Preuve

A la $k$-ieme erreur arrive pour un $\mathbf{x}_i$ donné

$$\begin{aligned}
\mathbf{w}_{k+1} &= \mathbf{w}_k + y_i \mathbf{x}_i \\
< \mathbf{w}_{k+1}, \mathbf{w}^* > &= < \mathbf{w}_k, \mathbf{w}^* > + y_i < \mathbf{x}_i, \mathbf{w}^* > \\
&\geq < \mathbf{w}_k, \mathbf{w}^* > + \delta \\
&\geq < \mathbf{w}_{k-1}, \mathbf{w}^* > + 2\delta \\
&\geq k\delta
\end{aligned}$$

Dans le meme temps:

$$\begin{aligned}
||\mathbf{w}_{k+1}||^2 &= ||\mathbf{w}_k + y_i \mathbf{x}_i||^2 \leq ||\mathbf{w}_k||^2 + C^2 \\
&\leq kC^2
\end{aligned}$$

Donc:

$$\sqrt{k}C > k\delta$$

# On pourrait demander plus...

Remarque: Programmation linéaire. Trouver $\mathbf{w}, \delta$ tel que:

$$Max\ \delta, \quad subject\ to$$
$$\forall\, i = 1 \ldots n,\ y_i\ < \mathbf{w}, \mathbf{x}_i > > \delta$$

Ici, se brancheront les machines à vecteurs supports

# Adaline Widrow 1960

## Adaptive Linear Element

Etant donné

$$\mathcal{E} = \{(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}, i = 1 \dots n\}$$

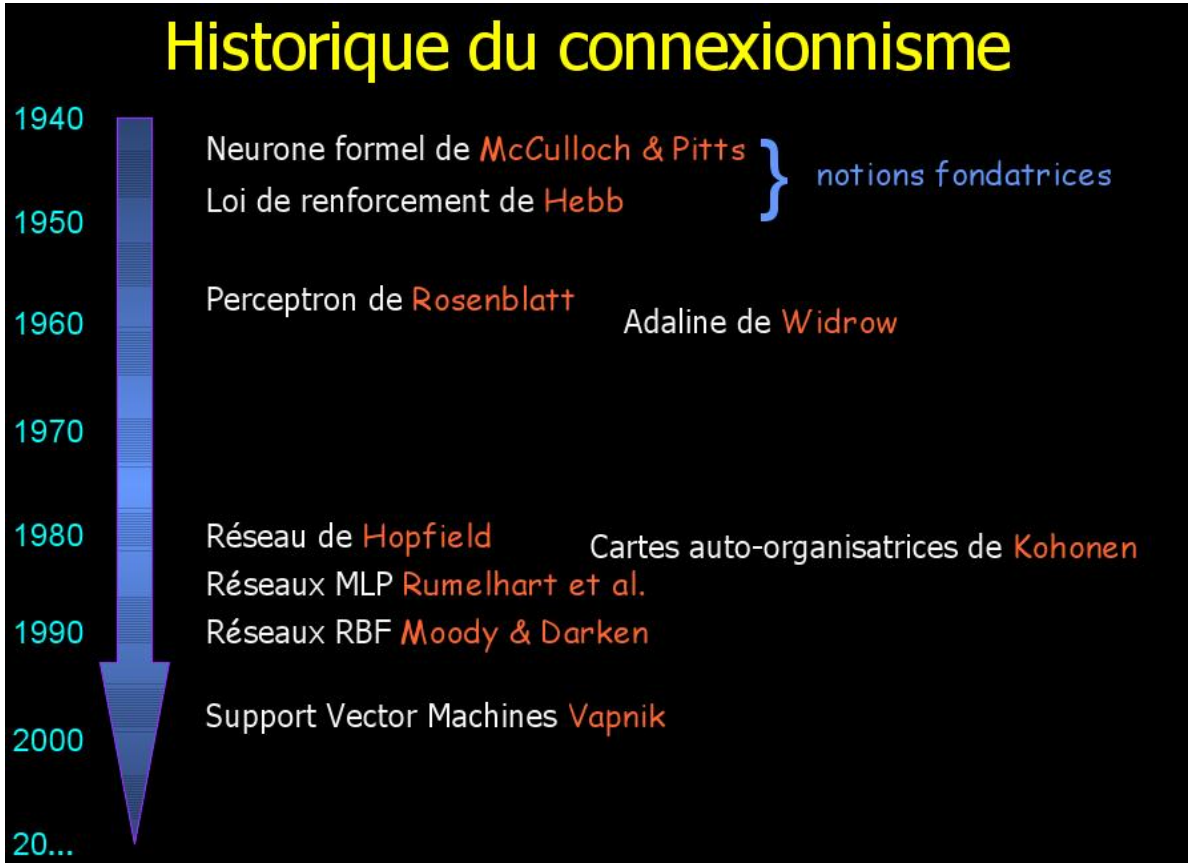
Apprentissage Minimisation d'une fonction quadratique

$$\mathbf{w}^* = argmin\{Err(\mathbf{w}) = \sum (y_i - < \mathbf{w}, \mathbf{x}_i >)^2\}$$

## Algorithme de gradient

$$\mathbf{w}_i = \mathbf{w}_{i-1} + \alpha_i \nabla Err(\mathbf{w}_i)$$

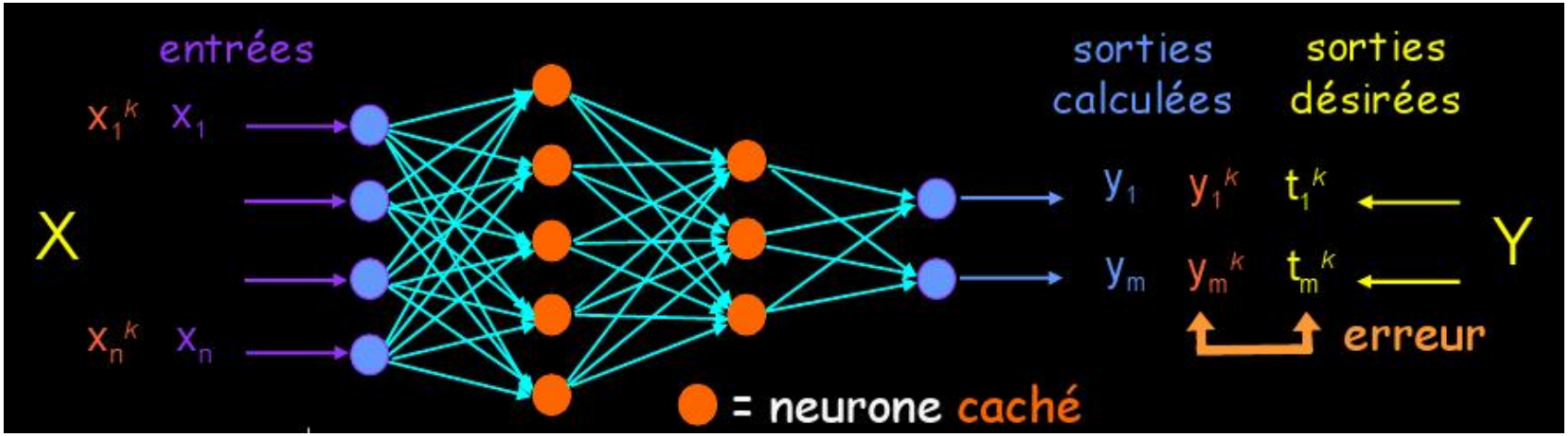
# L'hiver des NNs




Limite des séparateurs linéaires Minsky Papert 1969

Le problème du XOR.

# Perceptrons Multi-Couches, Rumelhart McClelland 1986



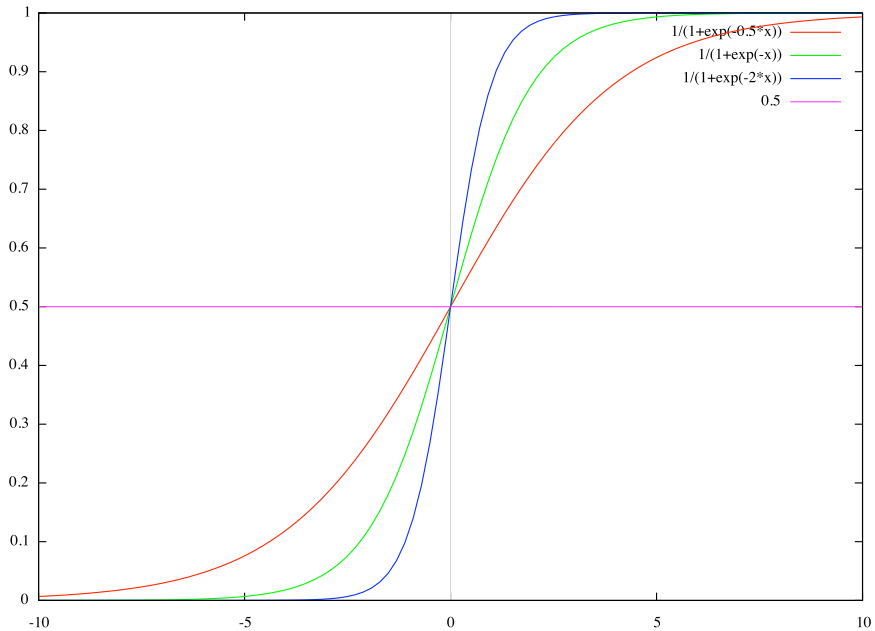

Apports

- Plusieurs couches, une représentation plus complexe, résolution du XOR
- Une fonction d'activation dérivable

$$ouput(\mathbf{x}) = \frac{1}{1 + exp\{- < \mathbf{w}, \mathbf{x} >\}}$$

# La fonction sigmoide

- $\sigma(t) = \frac{1}{1+exp(-a.t)}, a > 0$
- approxime une décision binaire (fonction en escalier)
- linéaire au voisinage de 0
- croit fortement au voisinage de 0 (dérivée “en cloche”)
- $\sigma'(x) = a\sigma(x)(1 - \sigma(x))$



# Rétro-propagation du gradient, Rumelhart McClelland 1986; Le Cun 1986

## Intuition

- Soit $(\mathbf{x}, y)$ un exemple d'apprentissage tiré au hasard
- On met les $d$ entrées du réseau à $x_1 \dots x_d$
- On calcule itérativement les valeurs des sorties des neurones
- Jusqu'à la sortie du réseau: $\hat{y}$;
- Comparer $\hat{y}$ et $y$ $Err(w) = (\hat{y} - y)^2$
- On modifie les poids de la dernière couche en utilisant le gradient.
- Regardons l'avant dernière couche; on sait ce qu'on aurait aimé avoir; on en déduit ce qu'on aurait aimé avoir sur la couche d'avant; on remonte.
- Les erreurs (sur chaque couche $i$) sont utilisées pour modifier les poids menant de la couche $i-1$ à la couche $i$.

# Rétro-propagation du gradient

## Notations

Input $\mathbf{x} = (x_1, \ldots x_d)$

Des input à la premiere couche cachée

$$z_j^{(1)} = \sum w_{jk} x_k$$

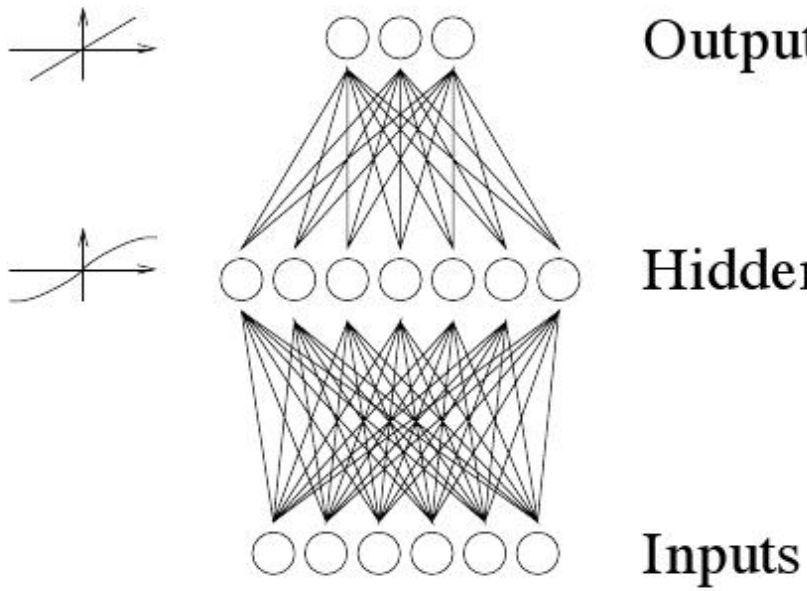
$$x_j^{(1)} = f(z_j^{(1)})$$

De la couche $i$ à la couche $i+1$

$$z_j^{(i+1)} = \sum w_{jk}^{(i)} x_k^{(i)}$$

$$x_j^{(i+1)} = f(z_j^{(i+1)})$$

($f$: e.g. sigmoide)



# Algorithme de rétro-propagation du gradient

Input$(\mathbf{x}, y)$, $\mathbf{x} \in \mathbb{R}^d$, $y \in \{-1, 1\}$

Phase 1 Propager l'information vers l'avant

- Pour toute couche $i = 1 \dots \ell$

  Pour tout neurone $j$ de la couche $i$

  $$z_j^{(i)} = \sum_k w_{j,k}^{(i)} x_k^{(i-1)}$$
  $$x_j^{(i)} = f(z_j^{(i)})$$

Phase 2 Comparer

- Erreur: difference entre $\hat{y}_j = x_j^{(\ell)}$ et $y_j$.
  On aimerait modifier $\hat{y}_j$ de

$$e_j^{sortie} = \frac{\partial h}{\partial t}(z_j^{sortie})[\hat{y}_j - y_j]$$

# Algorithme de rétro-propagation du gradient

Phase 3 Propager les erreurs vers l'arrière

$$e_j^{(i-1)} = \frac{\partial h}{\partial t}(z_j^{(i-1)}) \sum_k w_{kj}^{(i)} e_k^{(i)}$$

Phase 4: On met à jour les poids dans toutes les couches :

$$\Delta w_{ij}^{(k)} = \alpha e_i^{(k)} x_j^{(k-1)}$$

où $\alpha$ est le taux d'apprentissage $(< 1.)$

# Réseaux Neuronaux Artificiels

## Réseaux classiques

1. Histoire
   - Neurones logiques/non supervisés
   - Perceptron, Adaline: algorithmes
   - L'hiver des NN: limites théoriques
   - Perceptron Multi-couches: algorithme et limites algorithmiques
2. Structure du réseau et apprentissage

# Types de réseaux neuronaux

## Ingrédients

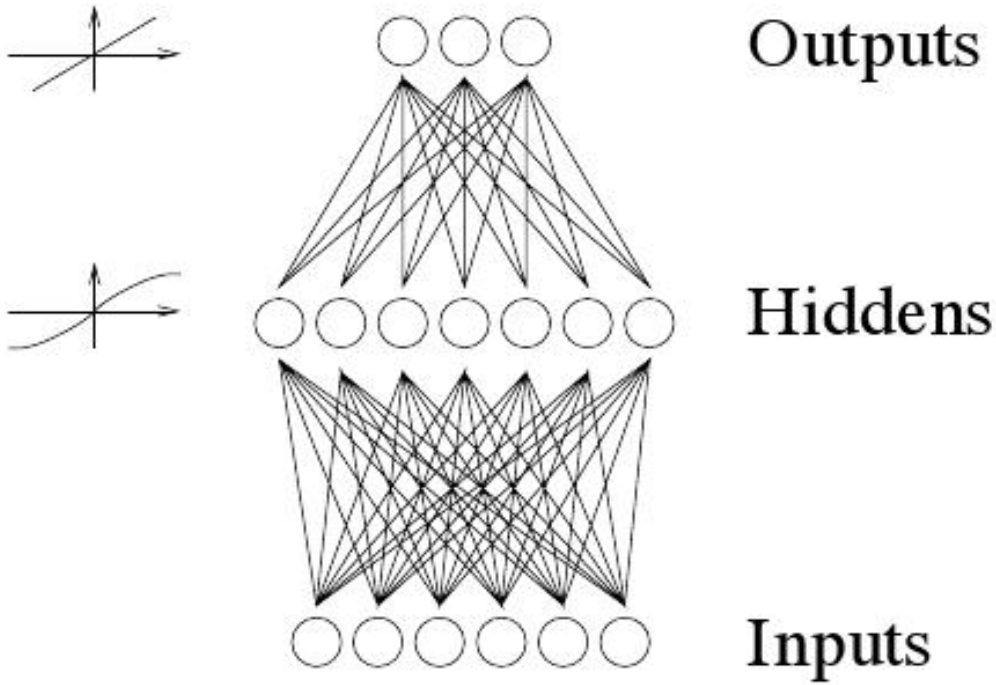
- Un ensemble de neurones connectés (graphe orienté) feedforward ($\equiv$ DAG) ou récurrent
- Un poids sur chaque connexion
- Une fonction d'activation

## Fonction d'activation(z)

- à seuil — 0 si $z < \mathit{seuil}$, 1 sinon
- linéaire — z
- sigmoide — $1/(1+e^{-z})$
- Radius — $e^{-z^2/\sigma^2}$

# Structure / Graphe de connexion / Topologie

## Architecture multi-couche




(C) David McKay - Cambridge Univ. Press

# Structure / Graphe de connexion / Topologie, suite

## Récurrent

- Propager ; attendre la stabilisation
- Retro-propagation inapplicable
- Le système a une mémoire (valeur des neurones cachés)
  Hum. La mémoire s'efface exponentiellement vite
- Données dynamiques (video)

## Connaissances a priori

- Invariance par translation, rotation,.. $op$
- $\longrightarrow$ Etendre $\mathcal{E}$ considérer $(op(\mathbf{x}_i), y_i)$
- ou partager les poids: Réseau convolutionnel

# Propriétés

## Good news

- MLP, RBF: approximateurs universels

Pour toute fonction $f$ raisonnable (de carré intégrable sur un compact), pour tout $\epsilon > 0$, il existe un MLP/RBF approchant $f$ avec la précision $\epsilon$.

## Bad news

- Ce résultat n'est pas constructif. (i.e. il existe; so what).
- On peut avoir tout ce qu'on veut; $\rightarrow$ on peut avoir n'importe quoi (sur-apprentissage).

# Questions centrales

## Sélection de modèles

- Choix de nombre de neurones, de la topologie, ...
- Choix de critère overfitting

$$\text{PLUS} \not\Rightarrow \text{MIEUX}$$

## Choix algorithmiques un pb d'optim. difficile

- Initialisation **w** petit !
- Décroissance du pas
- Utilisation d'un momentum relaxation

$$\mathbf{w}_{neo} \leftarrow (1-\alpha)\mathbf{w}_{old} + \alpha\mathbf{w}_{neo}$$

- Critère d'arrêt

Commencer par centrer normer les données, naturellement

$$x \mapsto \frac{x - \textit{moyenne}}{\textit{variance}}$$

# Quelques bonnes adresses

## Quelques liens

- un cours:
  `http://neuron.tuke.sk/math.chtf.stuba.sk/pub/vlado/NN_books_texts/Krose_Smagt_neuro-intro.pdf`
- une FAQ: `http://www.faqs.org/faqs/ai-faq/neural-nets/part1/preamble.html`
- des applets
  `http://www.lri.fr/~marc/EEAAX/Neurones/tutorial/`
- des implÃ©mentations: PDP++/Emergent (`www.cnbc.cmu.edu/PDP++/`); SNNS `http://www-ra.informatik.uni-tuebingen.de/SNNS/`...

## Voir aussi

- NEAT & HyperNEAT — Stanley, U. Texas

# Réseaux Neuronaux

1. Pourquoi ?
2. Réseaux classiques
3. Quelques applications
4. Réseaux modernes: profonds, à échos, impulsionnels, liquides...
5. Pistes de recherche fraiches

# Applications

1. Reconnaissance de chiffres manuscrits
2. Contrôle (conduite de véhicule,..)
3. Language

# Philosophie

- Partitioning a system into individual modules is a time-honored engineering practice.
- The partition is performed so that each module can be individually specified, designed, and tested. function
- However, there have been several recent demonstration that, with enough training data, learning algorithms are much better at building complex systems than humans: speech and handwriting recognition.
- Relying more on learning and less on hand-crafting can yield better results.

# Reconnaissance de chiffres manuscrits

Fig. 4. Size-normalized examples from the MNIST database.

## Caractéristiques

- Taille input $d$: quelques centaines
- $\longrightarrow$ beaucoup de poids :-(
- Connaissance a priori: la classe ne change pas par (petites) translation, rotation, homothétie, des données

# Réseaux convolutionnels

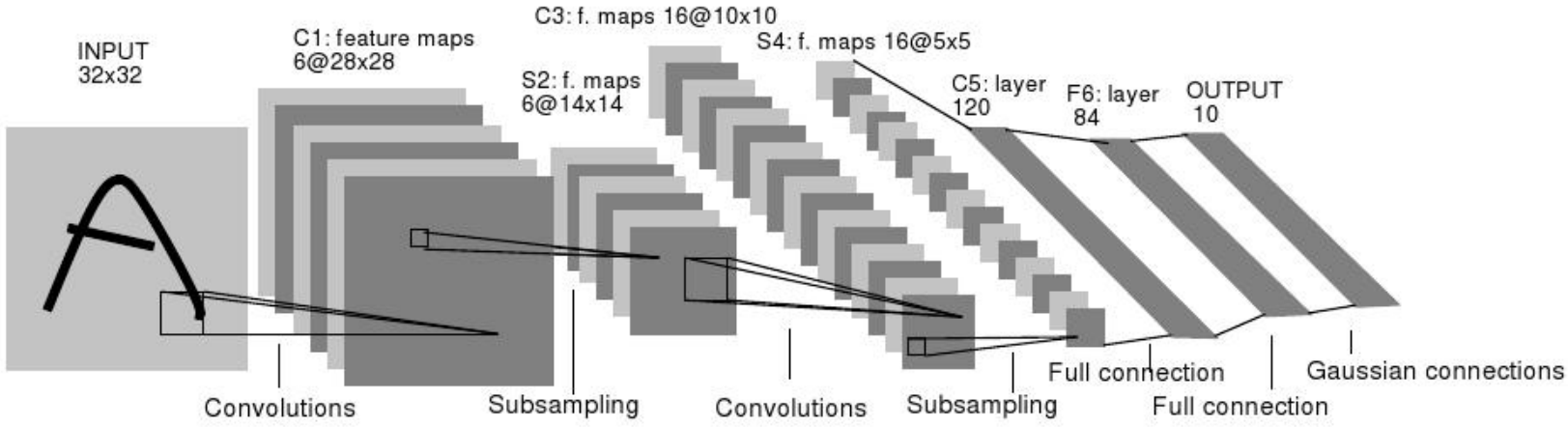
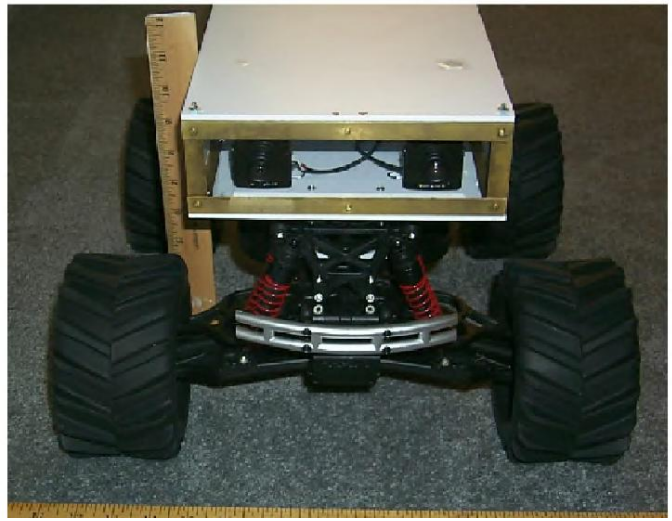


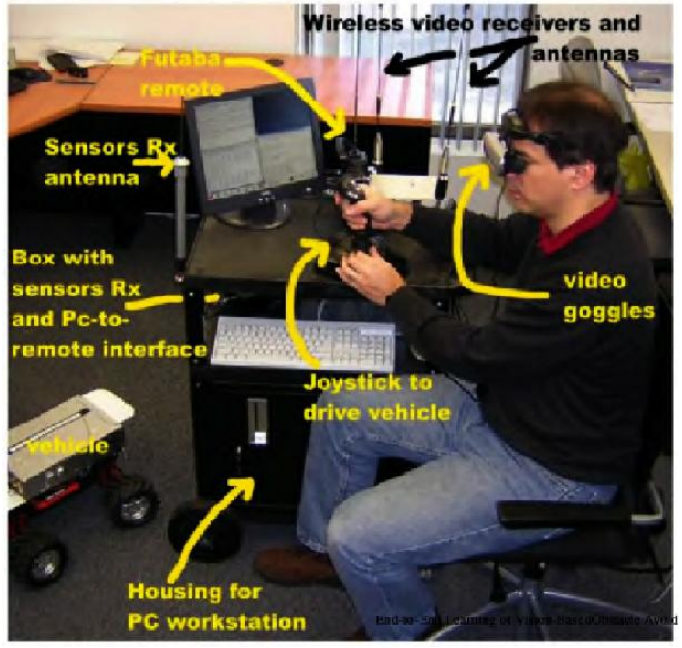

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Lecture http://yann.lecun.com/exdb/lenet/

- Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, The Handbook of Brain Theory and Neural Networks. MIT Press, 1995.

# Controle de véhicule






Lectures, Video

http://www.cs.nyu.edu/ yann/research/dave/index.html

# Modèles continus de langage

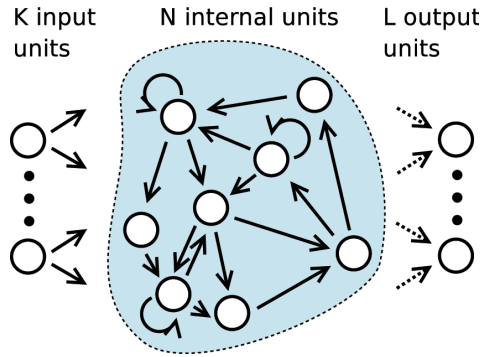
## Principe

- Input: 10,000 input booleens les mots de la phrase
- Neurones cachés: 500 neurones numériques
- But: En fonction des mots d'une fenêtre $m_1, ...m_{2k+1}$ , apprendre
  - Le genre grammatical du mot central $m_k$
  - Le mot suivant $m_{2k+2}$
- Rq: La première couche: $\{$ fragments de phrase $\}$ $\mapsto \mathbb{R}^{500}$

# Réseaux Neuronaux

1. Pourquoi ?
2. Réseaux classiques
3. Quelques applications
4. Réseaux modernes: profonds, à échos, impulsionnels, liquides...
5. Pistes de recherche fraiches

# Echo State Networks Jaeger 2001

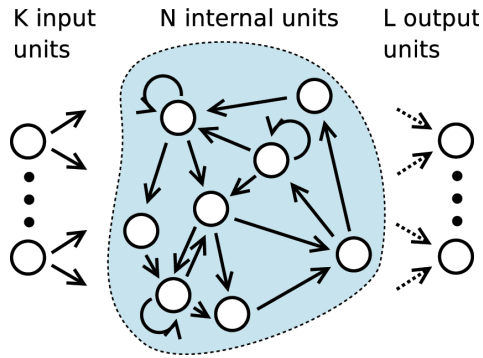



NIPS 2005 Wshop, Reservoir Computing
$=$ Echo State Network, [Jaeger 2001]
$\bigcup$ Liquid State Machine, [Maas 2002]

## Structure

- $N$ hidden nodes
- Random connexion matrix $\mathcal{G}$, connectivity $\rho$
- Stability: highest eigenvalue $\mathcal{G}$ (damping factor) $< 1$
- Output nodes: linear combination of hidden nodes

# Echo State Networks, 2




A revolution

- The end of micro-management for NN (only $\rho$ and $\lambda$)
- Training (e.g. for regression) through quadratic optimization

More:

On computational power and the Order Chaos: Phase Transition in Reservoir Computing, Benjamin Schrauder, Lars Büsing and Robert Legenstein, NIPS 2008.

# Réseaux profonds Hinton et al. 2006; Bengio et al. 2006

Profond $\longrightarrow$ Réduction de complexité

- Fonction parité de $d$ bits $\mathcal{X} = \{0,1\}^d$
- 1 seule couche cachée

# Réseaux profonds Hinton et al. 2006; Bengio et al. 2006

Profond $\longrightarrow$ Réduction de complexité

- Fonction parité de $d$ bits — $\mathcal{X} = \{0,1\}^d$
- 1 seule couche cachée — $2^d$ neurones cachés
- $log_2(n)$ couches cachées

# Réseaux profonds Hinton et al. 2006; Bengio et al. 2006

Profond $\longrightarrow$ Réduction de complexité

- Fonction parité de $d$ bits — $\mathcal{X} = \{0,1\}^d$
- 1 seule couche cachée — $2^d$ neurones cachés
- $log_2(n)$ couches cachées — $d$ neurones cachés

Profond $\longrightarrow$ Apprentissage supervisé terrible

optima locaux de mauvaise qualité, difficulté de régulariser

# Réseaux profonds Hinton et al. 2006; Bengio et al. 2006

Profond $\longrightarrow$ Réduction de complexité

- Fonction parité de $d$ bits — $\mathcal{X} = \{0,1\}^d$
- 1 seule couche cachée — $2^d$ neurones cachés
- $log_2(n)$ couches cachées — $d$ neurones cachés

Profond $\longrightarrow$ Apprentissage supervisé terrible

optima locaux de mauvaise qualité, difficulté de régulariser

Apprentissage profond **non supervisé !**

http://www.iro.umontreal.ca/∼bengioy/yoshua_en/talks.html

# Réseaux Neuronaux

1. Pourquoi ?
2. Réseaux classiques
3. Quelques applications
4. Réseaux modernes: profonds, à échos, impulsionnels, liquides...
5. Pistes de recherche

# Le point: On veut construire un système

## Ce qu'on a compris

- Combinaison d'entités simples: arbitrairement complexe

# Le point: On veut construire un système

## Ce qu'on a compris

- Combinaison d'entités simples: arbitrairement complexe pourvu que les entités ne soient pas linéaires.
- Le réseau est une fonction de redescription des données.

# Le point: On veut construire un système

## Ce qu'on a compris

- Combinaison d'entités simples: arbitrairement complexe pourvu que les entités ne soient pas linéaires.
- Le réseau est une fonction de redescription des données.

## On veut construire un système

1. Dans quel espace habite-t-il ?
   Décrit par un ensemble de poids ? par des macro-descripteurs (topologie, poids) ?
2. Comment l'entrainer ?
   Quel critère ? Quel algorithme d'optimisation ?
3. Comment coder la connaissance du domaine ?
   Dans les exemples ? Dans la structure de la solution ?