# De nouveaux paradigmes pour les réseaux de neurones : *Deep Learning* et *Reservoir Computing*

Hélène Paugam-Moisy

INRIA Saclay - LRI, Orsay / LIRIS, Université Lyon 2

CAp 2012

LORIA, Nancy - 25 mai 2012

⇒ séminaire LAMIA, Pointe-à-Pitre - 6 janvier 2016

# INTRODUCTION

# Les réseaux de neurones à travers les âges

# La décennie des MLP : les années 90

Multi-Layer Perceptron (MLP) = réseau multicouche dont les poids sont appris par rétro-propagation, méthode supervisée par descente en gradient.

*Théorie :*

- Calculabilité: outperforms a common Turing machine [Siegelman,1995]
- Complexité: the "loading problem" is NP-complete [Judd,1990]
- Capacité: MLP are universal approximators [Cybenko,1988 ; Hornik,1990]

*Pratique :* une seule couche cachée ; fonction *tanh* ou logistique ; connexion complète d'une couche à la suivante

*Applications :* innombrables..... classification ou régression

# La décennie des SVM : les années 2000

Support Vector Machine (SVM) = *kernel trick* pour expansion vers un espace virtuel de très grande dimension + marge optimale pour séparation linéaire (de type Perceptron) dans cet espace.

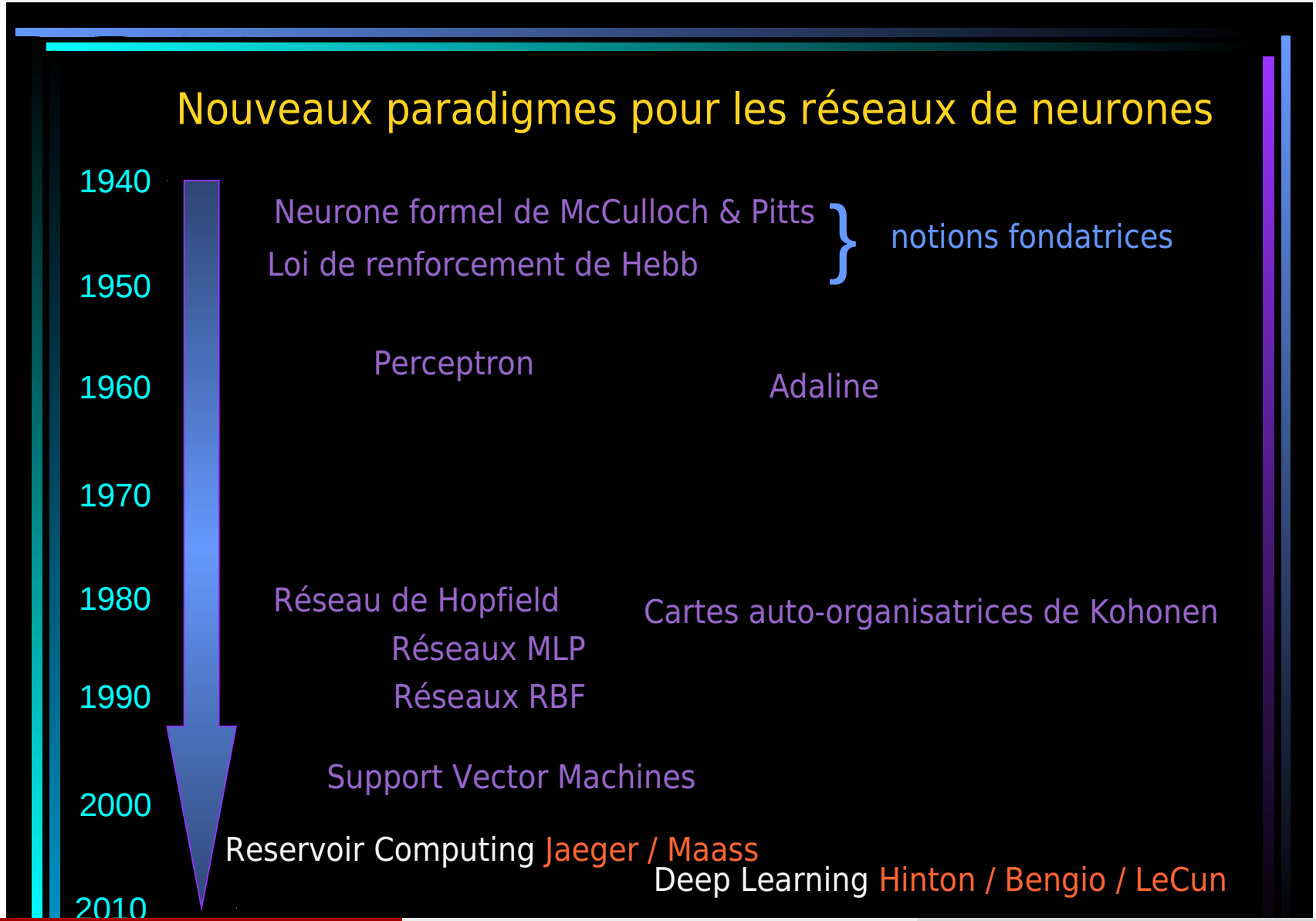
*Théorie :*

- VC-dimension: capacité d'apprentissage [Vapnik,1995]
- Théorie statistique de l'apprentissage: risque structurel [Vapnik,1998]
- Méthodes à noyaux: *kernel PCA*, etc... [Smola ; Burges ; et al.]

*Pratique :* noyaux gaussiens ; méthodes d'optimisation quadratique

*Applications :* tout aussi innombrables.....

# Émergence de nouveaux paradigmes de calculs



# DEEP LEARNING

# Les leçons du passé

Les *Deep Networks* sont des réseaux multicouches, qui sont... profonds.

*shallow* vs. *deep* architecture pour les réseaux *feedforward*

On savait depuis longtemps qu'il était plus facile de capter certaines non-linéarités avec plusieurs couches de petites tailles plutôt qu'avec une seule très grande couche (ex. parity).

mais.... comment adapter l'apprentissage ? (pb des minima locaux)

Il s'est avéré que la rétropropagation marchait mal dans les réseaux à 3, 4 ou plus... couches cachées, sauf si on savait "mettre de la connaissance dans le réseau" (ex. LeNet de LeCun).

# Les motivations

Pourquoi des Deep Networks ?

- capturer des non-linéarités d'ordre supérieur
- transformer les données d'entrée en une succession de *features* de niveau de plus en plus abstrait
- obtenir des réseaux ayant des architectures plus compactes
- pouvoir apprendre à partir d'un plus petit nombre d'exemples
- pouvoir inclure dans l'apprentissage des exemples sans étiquette

# Les nouvelles idées

*Deep Learning* = apprentissage couche par couche + non supervisé

Deux familles de modèles ont été proposées simultanément :

## RBM empilées

inspiré par la *Boltzmann Machine*
modèle génératif
basé sur des calculs de probabilités

Hinton, Salakhutdinov, *Science*, 2006

Hinton, Osindero, Teh, *Neural Comp.*, 2006

## Auto-Associateurs empilés

inspiré par le réseau "diabolo"
modèle discriminatif
déterministe, par reconstruction

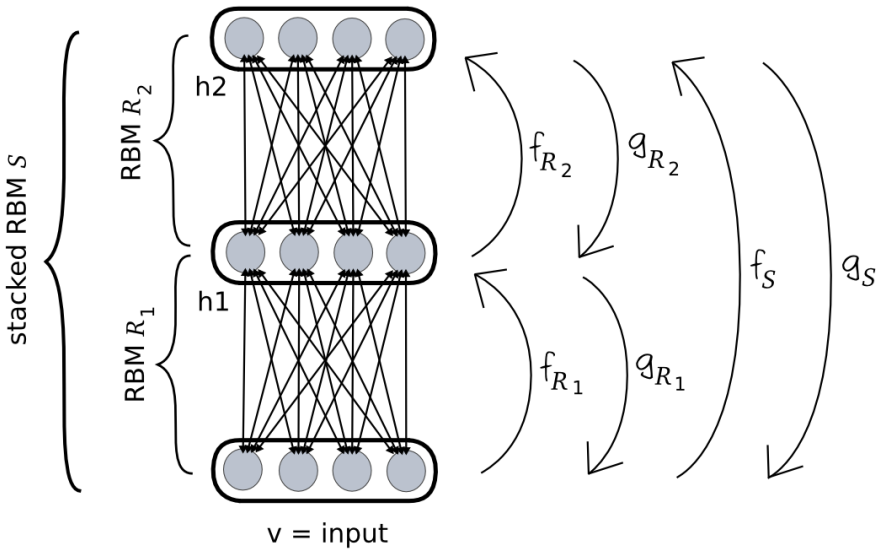
Bengio, Lamblin, Popovici, Larochelle, *Nips'06*

Ranzato, Poultney, Chopra, LeCun, *Nips'06*

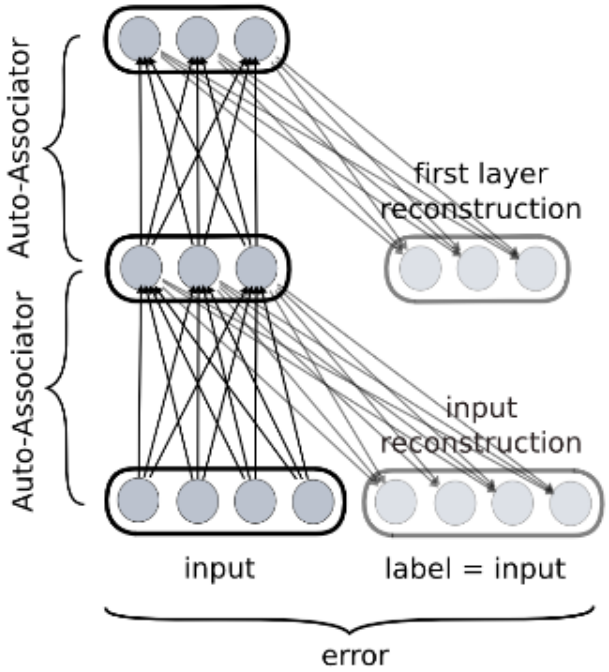
# Les deux principales familles de modèles

*Principe :* Construction couche par couche par apprentissage non supervisé

## Deep Belief Networks (RBM empilées)

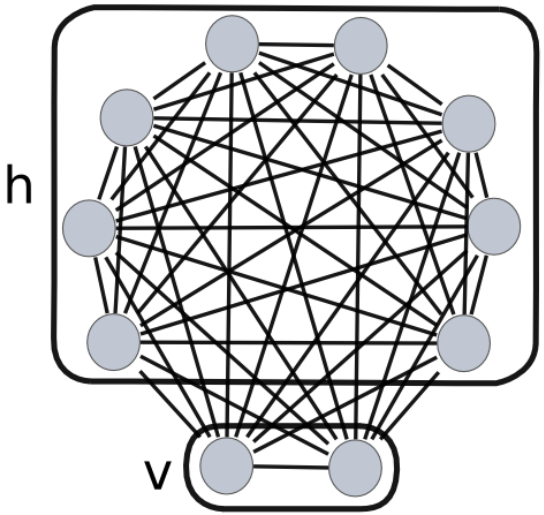



## Auto-associateurs empilés



# Restricted Boltzmann Machine (RBM)

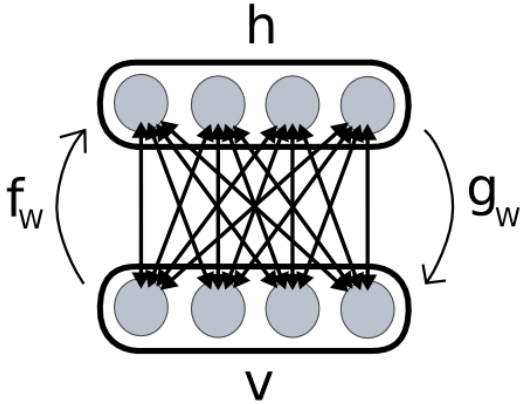
La machine de Boltzmann d'origine [Ackley, Hinton, Sejnowski - Cognitive Science, 1985] a été modifiée : graphe biparti, non orienté et pondéré, entre couche visible et couche cachée, vu comme un réseau à satisfaction de contraintes, et caractérisé par une énergie $E(v,h) = -h^T W v$




Machine de Boltzmann

probabilité d'une configuration

$$P(v,h) = \frac{e^{-E(v,h)}}{Z} = P(v|h)P(h)$$

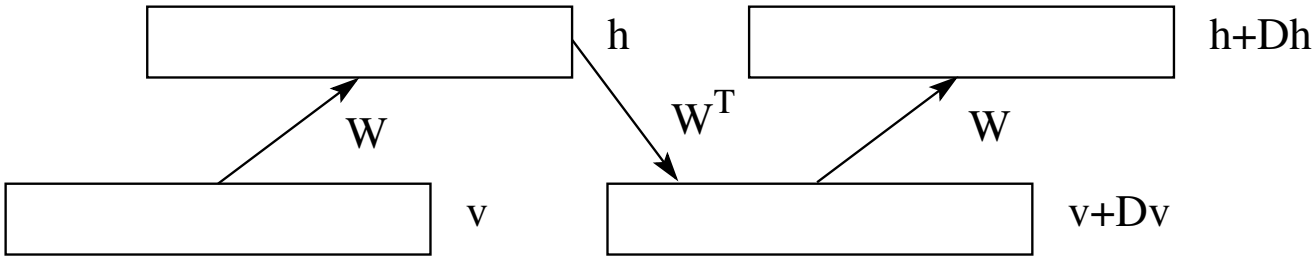



Restricted Boltzmann Machine (RBM)

# Apprentissage dans une RBM (1)

*Principe :* approximation du maximum de vraisemblance - Gibbs sampling
*Méthode :* descente en gradient pour minimiser la divergence KL

Les pixels de l'image (binaires) = visible units $v_i$
Les "feature detectors" = hidden units $h_j$




$< v_i h_j >$, produit scalaire de valeurs binaires, représente le nombre de fois où le pixel $v_i$ et le feature detector $h_j$ sont simultanément à 1.

# Apprentissage dans une RBM (2)

## Cheminement du procédé : aller-retour d'une couche à l'autre

- On met $h_j$ à 1 selon la probabilité $\sigma(\sum_i w_{ij} v_i + b_j)$ où $\sigma$ est une fonction sigmoïde
- On calcule une confabulation en mettant $v_i$ à 1 selon la probabilité $\sigma(\sum_j w_{ij} h_j + b_i)$
- On met à jour les états des $h_j$ (même méthode, à partir de la confabulation)
- On modifie les poids par $\Delta w_{ij} = \epsilon(< v_i h_j >_{data} - < v_i h_j >_{recon})$ où $\epsilon$ est un learning rate

$< v_i h_j >_{data}$ pour les données de l'image initiale et les $1^{res}$ valeurs des $h_j$
$< v_i h_j >_{recon}$ pour les valeurs de confabulation et les $2^{ndes}$ valeurs des $h_j$

# Apprentissage dans une RBM (3)

*Objectif :* descente en gradient pour minimiser $KL(P_{data} \parallel P_{model})$

$$\frac{\partial KL(P_{data} \parallel P_{model})}{\partial w_{ij}} = -\frac{1}{T}(p_{ij} - p'_{ij})$$

**Algorithm 0.2** Algorithme d'apprentissage d'une machine de Boltzmann restreinte.

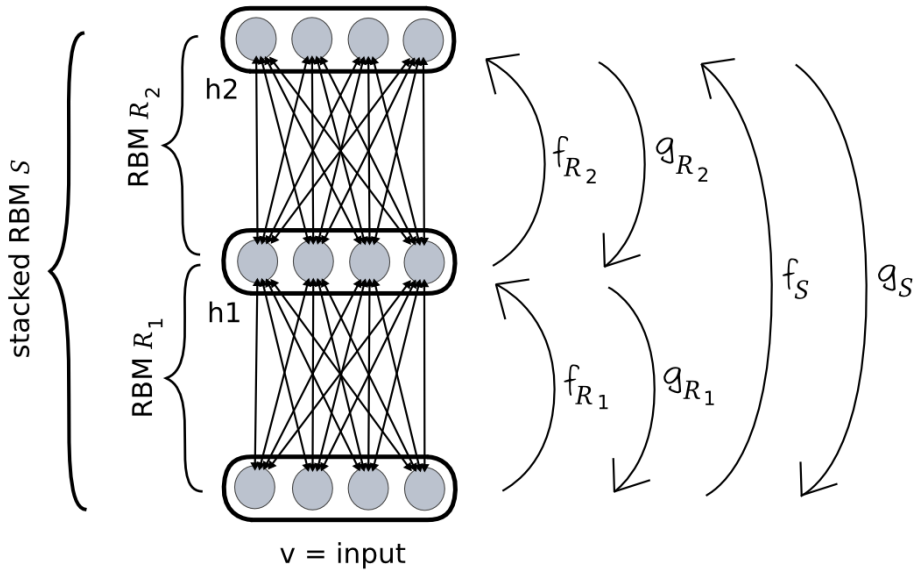
Pour chaque vecteur d'entrée $\mathbf{v}$ :

1. Obtenir $p_{ij}$ (en faisant un tirage aléatoire selon $P_{data}(\mathbf{x})$) de la manière suivante :
   (a) fixer les unités visibles $\mathbf{v}$ à un vecteur d'entrée
   (b) tirer aléatoirement $h_i$ selon $P(h_i|\mathbf{v})$ en une étape
2. Obtenir $p'_{ij}$ (en faisant un tirage aléatoire selon $P_{model}(\mathbf{x})$) de la manière suivante :
   (a) Laisser le réseau tourner librement
   (b) minimiser l'énergie avec le Gibbs Sampling
   (c) attendre convergence
3. déduire la mise à jour des poids

*Contrastive Divergence* : approximation, pour diminuer le temps de calcul

# Deep Learning : apprendre un empilement de RBMs

*Principe :* décomposition successive de la probablilité d'une configuration




$$P(v) = \sum_{h^1} P_{W_1}(v|h^1) P_{W_2}(h^1)$$
$$= \sum_{h^1} P_{W_1}(v|h^1) \sum_{h^2} P_{W_2}(h^1, h^2)$$

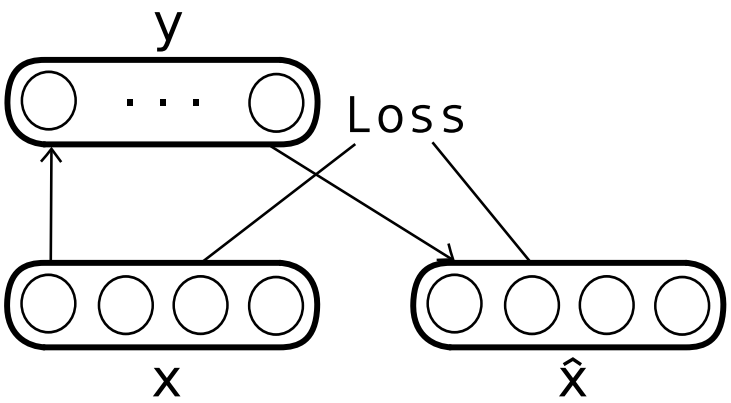
puis, de proche en proche :

$$P_{W_i}(h^{i-1}) = \sum_{h^i} P_{W_i}(h^{i-1}|h^i) P_{W_{i+1}}(h^i)$$

RBM empilées (schéma à 2 couches)

# Apprentissage d'un auto-associateur

De $x$ vers $y$ : encodeur – De $y$ vers $\hat{x}$ : décodeur



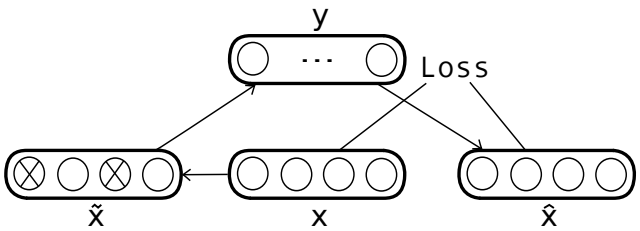

$$\begin{cases} y = f^{E}(x) = a^{E}(W^{E}x + b^{E}) \\ \hat{x} = f^{D}(y) = a^{D}(W^{D}y + b^{D}) \\ \hat{x} \simeq x \end{cases}$$

On apprend, par rétro-propagation, à reconstruire les données d'entrée ; on n'utilise donc aucune information de nature supervisée.

$$\phi(\Theta) = \sum_{x \in \mathcal{S}} \mathcal{L}(x, f^{D} \circ f^{E}(x))$$

# Variantes

## *Denoising Auto-Associator*

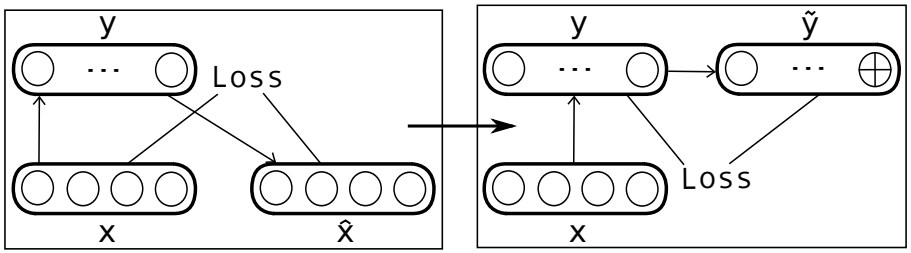



$$\phi^{\mathsf{D}}(\Theta) = \sum_{x \in \mathcal{S}} \mathcal{L}(x, f^{\mathsf{D}} \circ f^{\mathsf{E}}(\tilde{x}))$$

$$\text{with} \quad \tilde{x} = \text{corrupt}(x)$$

## *Sparse Auto-Associator*

$$\phi^{\mathsf{S,accuracy}}(\Theta) = \sum_{x \in \mathcal{S}} \mathcal{L}(x, f^{\mathsf{D}} \circ f^{\mathsf{E}}(x))$$

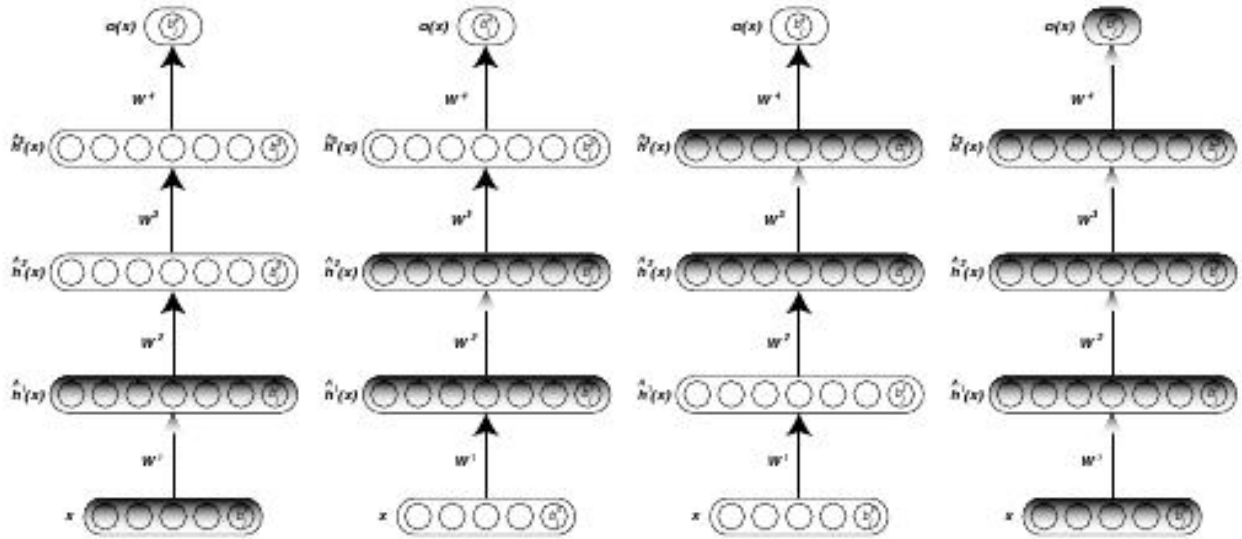



$$\phi^{\mathsf{S,sparsity}}(\Theta^{\mathsf{E}}) = \sum_{x \in \mathcal{S}} \mathcal{L}(f^{\mathsf{E}}(x), \tilde{y})$$

$$\text{with} \quad \tilde{y} = \text{sparsify} \circ f^{\mathsf{E}}(x)$$

# Deep Learning : apprendre une pile d'auto-associateus

On apprend à reconstruire chaque couche successivement, puis, comme pour les RBM empilées, on opère un *fine-tuning*, i.e. une rétro-propagation globale et supervisée sur l'ensemble des couches.



(a) First hidden layer pre-training (b) Second hidden layer pre-training (c) Third hidden layer pre-training (d) Fine-tuning of whole network

Figure 2: Unsupervised greedy layer-wise training procedure.

# Applications

Pourquoi des Deep Networks ? Parce que... "ça marche" !

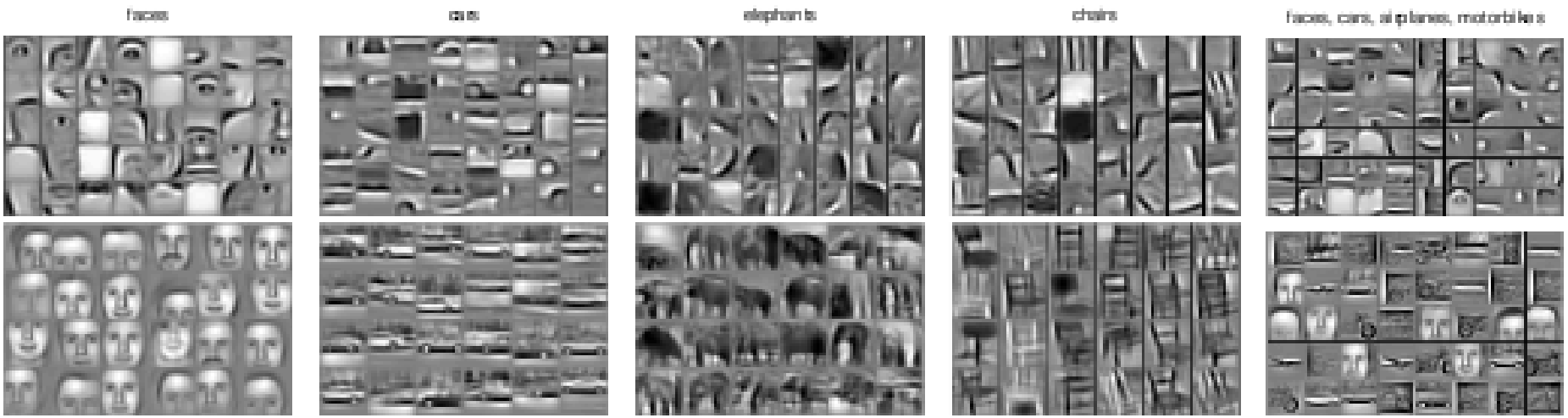
Exemples d'applications :

- images or faces recognition (Salakhutdinov, Hinton, 2008)
- real-world objects (Ranzato et al. 2007)
- text data (Hinton, Salakhutdinov, 2006 ; Collobert, Weston, 2008)
- regression (Salakhutdinov, Hinton, 2008)
- information retrieval (Salakhutdinov, Hinton, 2007)
- robotics (Hadsell et al., 2008)
- collaborative filtering (Salakhutdinov et al., 2007)
- etc...

# Codage des images, dans les couches profondes

Pour la plupart des auteurs : similarité entre les features de la 1ère couche cachée et les edge detectors réalisés par les cellules de l'aire V1.

Andrew Y. Ng et al vont plus loin en cherchant à interpréter les features des 2nde et 3ème couches in Lee, Grosse, Ranganath, Ng - ICML'2009 :



*Deep Learning modifié :* convolutional RBM + probabilistic max-pooling

# Pourquoi ça marche ?

Le "pre-training", qu'il soit supervisé ou non, permet de démarrer le "fine-tuning" par rétro-propagation dans une configuration de poids plus propice à conduire l'algorithme vers un *meilleur* minimum local.

H. Larochelle et al. (JMLR, 2009) montrent expérimentalement qu'un pré-apprentissage non supervisé semble plus efficace. Mais ils montrent aussi que la meilleure configuration consisterait à prendre des couches cachées successives de tailles identiques (simplificateur, mais contestable).

Les *features de features* sont supposées extraire des non-linéarités empilées qui permettraient de mieux adapter la méthode à une fonction complexe, présentant de brusques variations.

# Encore de nombreuses questions...

Il reste bien des mystères à élucider :

- Combien de couches cachées (un vieux serpent de mer...) ?
  ← il a été constaté qu'un réseau *trop* profond nuisait à la performance.
- Discriminative or generative method ?
- Supervised or unsupervised learning ?
- Comment réduire la complexité des méthodes d'apprentissage ?
  ← les algorithmes actuels sont *assez* coûteux en temps de calcul.
- Complexité : "Loading deep networks is hard" [J. Sima, 2009]

# Éléments de bibliographie sur *Deep Learning*

Y. Bengio and Y. Le Cun. Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, Large Scale Kernel Machines. MIT Press, 2007.

Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In NIPS 19, pages 153–160. MIT Press, 2007.

R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proc. of ICML 2008, pages 160–167, 2008.

R. Hadsell, A. Erkan, P. Sermanet, M. Scoffier, U. Muller, and Y. LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. In Proc. of IROS'08.

G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 313(5786):504–507, July 2006.

G.E. Hinton, S. Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. Neural Computation, 18:1527–1554, 2006.

H. Larochelle and Y. Bengio. Classification using discriminative Restricted Boltzmann Machines. In Proc. of ICML 2008, pages 536–543. Omnipress, 2008.

H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In Proc. of ICML 2007.

M'A. Ranzato, Y-L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In NIPS 20. MIT Press, Cambridge, MA, 2008.

R. Salakhutdinov and I. Murray. On the quantitative analysis of deep belief networks. In Proc. of ICML 2008, volume 25, 2008.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In Proc of ICML 2008, pages 1096–1103.

# Quelques pointeurs intéressants

Les sites des Workshops de NIPS'07 et de NIPS'11 sur le Deep Learning :
http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepLearningWorkshopNIPS2007

Un rapport de la NSF de juillet 2007 : Future Challenges for the Science and Engineering of Learning : http://www.cnl.salk.edu/Media/NSFWorkshopReport.v4.pdf

La page web "recherche" de Yoshua Bengio :
http://www.iro.umontreal.ca/ bengioy/yoshua_fr/recherche.html
et son livre :
Learning Deep Architectures for AI, Foundations and Trends in ML, now Publishers Inc., 2009

# RESERVOIR COMPUTING

# Les leçons du passé

Les Réservoirs sont des réseaux récurrents, qui sont *sparses* et aléatoires.

## Réseaux récurrents

On savait depuis longtemps que les réseaux récurrents étaient nécessaires pour capter les aspects temporels de certaines données (reconnaissance de la parole ; séries temporelles).

## mais.... comment superviser l'apprentissage d'une dynamique ?

Les performances des modèles tels que TDNN, BPTT, RTRL ou EKF n'étaient pas très satisfaisantes et les chaînes de Markov cachées (HMM) étaient largement préférées aux réseaux de neurones.

# Les motivations

Pourquoi le Reservoir Computing ?

- gérer une dynamique alimentée par des entrées successives, sans attendre de convergence
- conserver du passé une mémoire évanescente : *fading memory*
- simplifier au maximum la règle d'apprentissage
- créer un réseau qui puisse avoir une dynamique la plus riche possible : *edge of chaos*

# Les nouvelles idées

*Reservoir Computing* = libre auto-organisation du réservoir + apprentissage des sorties, les *readout*

Deux familles de modèles ont été proposées simultanément :

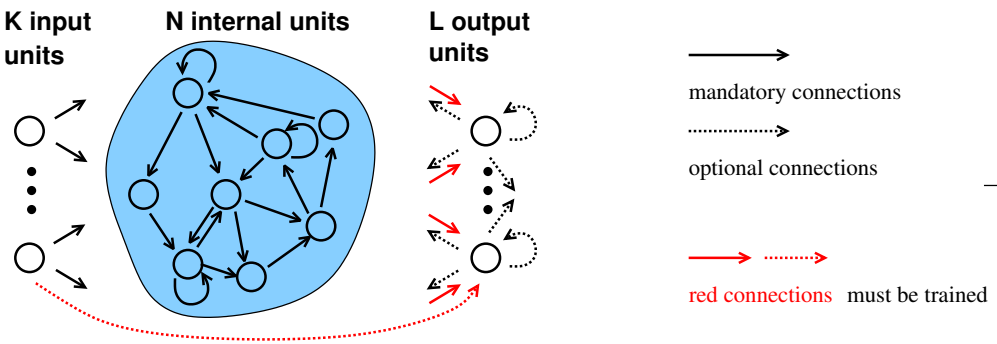
| Echo State Network (ESN) | Liquid State Machine (LSM) |
|---|---|
| inspiré par la résonance d'un écho | les ondes propagées sur l'eau |
| neurones sigmoïdes ou LIF | Leaky Integrate-and-Fire (LIF) |
| destiné aux séries temporelles | basé sur l'organisation du cortex |
| Jaeger, Report TR-GMD-148, 2001 | Maass, Natschläger, Markram, *N. Comp.*, 2002 |

# Les deux principales familles de modèles

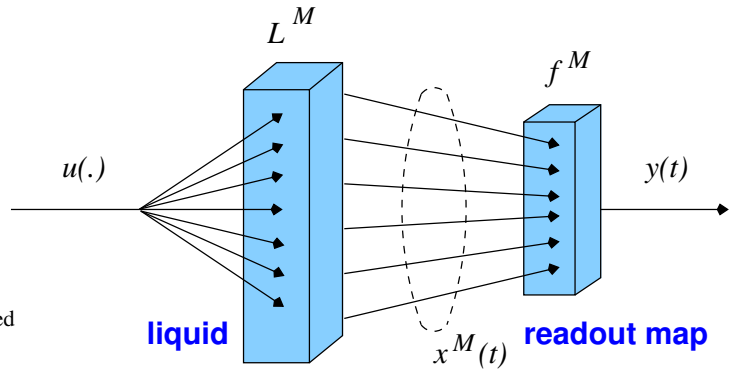
*Terminologie :* le nom de *Reservoir Computing* est apparu en 2007, pour unifier le concept : apprendre à extraire des informations d'un "réservoir".

## Architectures des deux modèles fondateurs

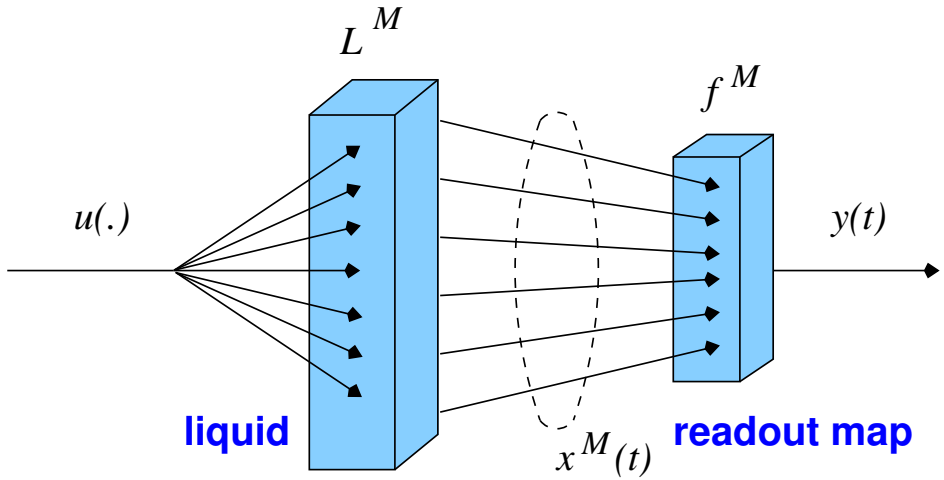
### Echo State Network




### Liquid State Machine



# LSM = Liquid State Machine




*Reservoir*

point-wise Separation Property (SP)

neurones LIF, avec synapses dynamiques - éventuellement unités à seuil

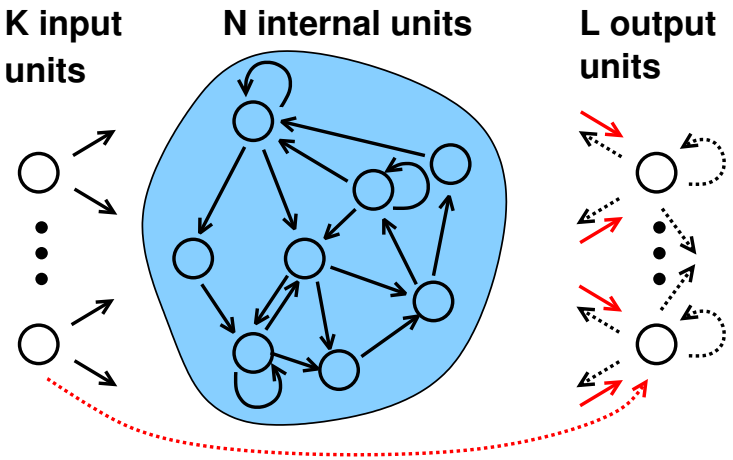
1 ou plusieurs colonnes 3D de neurones, connectivité sparse et aléatoire dependant de la distance entre neurones

*Readout*

Approximation Property (AP)

neurones LIF

# ESN = Echo State Network




mandatory connections

optional connections

red connections must be trained

*Reservoir*

Echo State property (ES)

neurones sigmoïdes - éventuellement neurones LIF

sans structure dans le réservoir de neurones, connectivité sparse et aléatoire

*Readout*

apprentissage très simple (ex. Least Mean Squares)

éventuellement connexions feedback : reservoir $\leftarrow$ readouts

# Contrôle du réservoir ?

C'est simple : peu de règles de contrôle.

Connectivité le plus souvent de l'ordre de 10% et poids tirés aléatoirement.

Dans le cas des ESN à neurones sigmoïdes :

- plus grande valeur singulière: $\sigma_{max} < 1$ est une condition suffisante
- rayon spectral: $\mid \lambda_{max} \mid < 1$ est une condition nécessaire

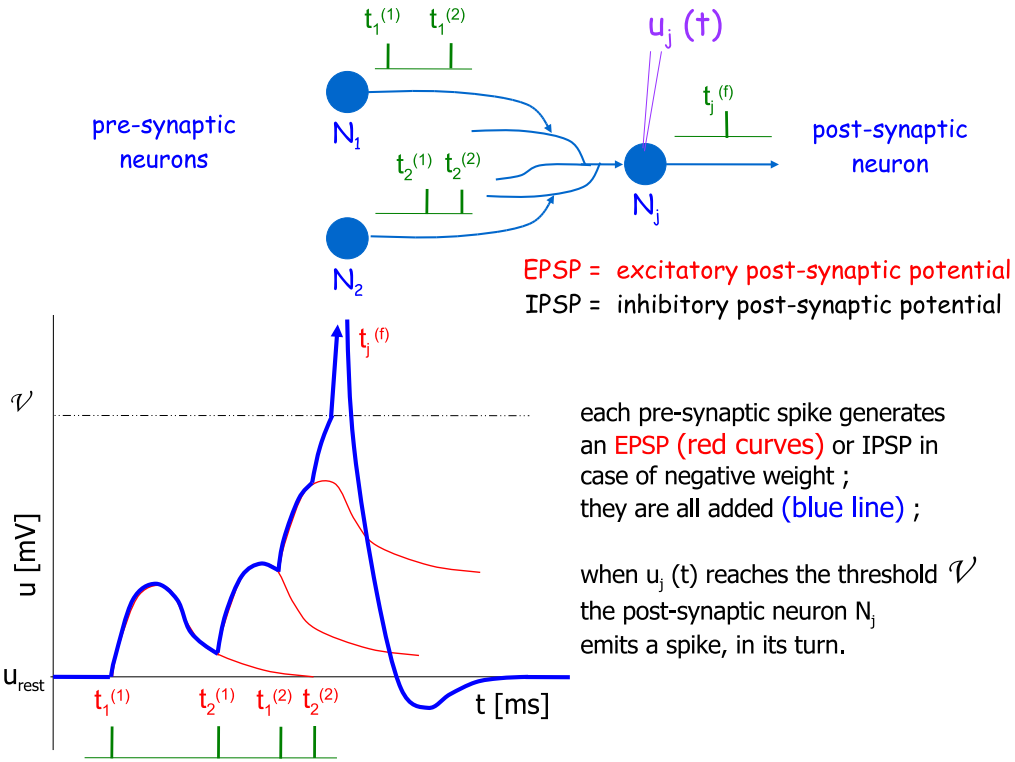
$\Rightarrow$ en pratique, "on" réajuste la matrice de poids $W$ afin que son rayon spectrale soit $\simeq 0.95$, mais cela n'est pas nécessairement judicieux...

**Et surtout !** cela n'a aucun sens dans un réservoir de *spiking neurons*.

# Réservoirs de *spiking neurons*

Les réservoirs sont des modèles dynamiques destinés à traiter des données temporelles $\Rightarrow$ les neurones les mieux appropriés sont les *spiking neurons*.

## Spiking neurons




L'instant précis du *spike* joue un rôle fondamental dans le codage de l'information

**action potential**
**= pulse**
**= spike**

# Modèles de *spiking neurons*

- Modèle de Hodgkin-Huxley (HH model) [depuis 1952]

# Modèles de *spiking neurons*

- Modèle de Hodgkin-Huxley (HH model) [depuis 1952]

$$C\frac{du}{dt} = -g_{Na}m^3h(u - E_{Na}) - g_K n^4(u - E_K) - g_L(u - E_L) + I(t)$$

$$\tau_n \frac{dn}{dt} = -[n - n_0(u)]$$

$$\tau_m \frac{dm}{dt} = -[m - m_0(u)]$$

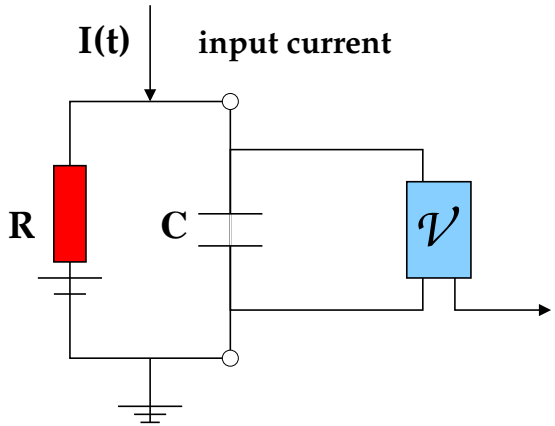
$$\tau_h \frac{dh}{dt} = -[h - h_0(u)]$$

# Modèles de *spiking neurons*

- Modèle de Hodgkin-Huxley (HH model) [depuis 1952]
- Modèles Integrate-and-Fire (IF, LIF, QIF...) [depuis Lapicque, 1907!]

# Modèles de *spiking neurons*

- Modèle de Hodgkin-Huxley (HH model) [depuis 1952]
- Modèles Integrate-and-Fire (IF, LIF, QIF...) [depuis Lapicque, 1907!]




$$C\frac{du}{dt} = -\frac{1}{R}u(t) + I(t) \qquad \text{spike}$$

timing $t^{(f)}$ defined by $u(t^{(f)}) = \vartheta$ with $u'(t^{(f)}) > 0$

# Modèles de *spiking neurons*

- Modèle de Hodgkin-Huxley (HH model) [depuis 1952]
- Modèles Integrate-and-Fire (IF, LIF, QIF...) [depuis Lapicque, 1907!]
- Gerstner's Spike Response Model (SRM, $SRM_0$) [≈1998]

# Modèles de *spiking neurons*

- Modèle de Hodgkin-Huxley (HH model) [depuis 1952]
- Modèles Integrate-and-Fire (IF, LIF, QIF...) [depuis Lapicque, 1907!]
- Gerstner's Spike Response Model (SRM, $SRM_0$) [≈1998]

$$u_j(t) = \sum_{t_j^{(f)} \in \mathcal{F}_j} \eta_j \left(t - t_j^{(f)}\right) + \sum_{i \in \Gamma_j} \sum_{t_i^{(f)} \in \mathcal{F}_i} w_{ij} \epsilon_{ij} \left(t - t_i^{(f)}\right) + \underbrace{\int_0^\infty \kappa_j(r) I(t-r) dr}_{\text{if external input current}}$$

or simpler model $SRM_0$ (very close to traditional formula)

$$u_j(t) = \sum_{i \in \Gamma_j} w_{ij} \epsilon(t - t_i^{(f)} - \delta_{ij}^{ax})$$

# Modèles de *spiking neurons*

- Modèle de Hodgkin-Huxley (HH model) [depuis 1952]
- Modèles Integrate-and-Fire (IF, LIF, QIF...) [depuis Lapicque, 1907!]
- Gerstner's Spike Response Model (SRM, $SRM_0$) [≈1998]
- Izhikevich's neuron model [2003]

# Modèles de *spiking neurons*

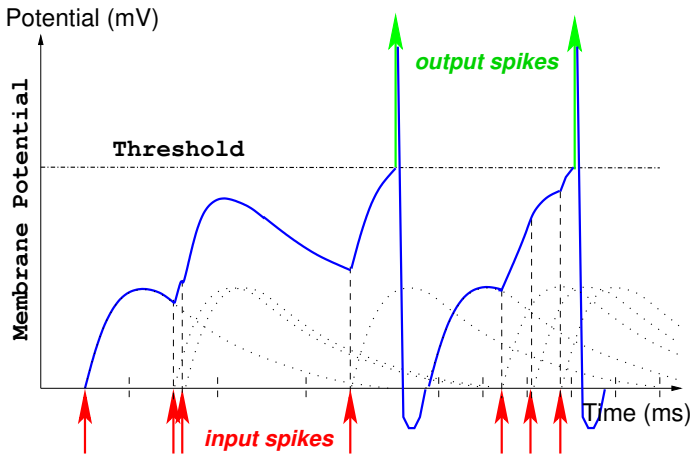
- Modèle de Hodgkin-Huxley (HH model) [depuis 1952]
- Modèles Integrate-and-Fire (IF, LIF, QIF...) [depuis Lapicque, 1907!]
- Gerstner's Spike Response Model (SRM, $SRM_0$) [≈1998]
- Izhikevich's neuron model [2003]

$$\frac{du}{dt} = 0.04u(t)^2 + 5u(t) + 140 - w(t) + I(t)$$

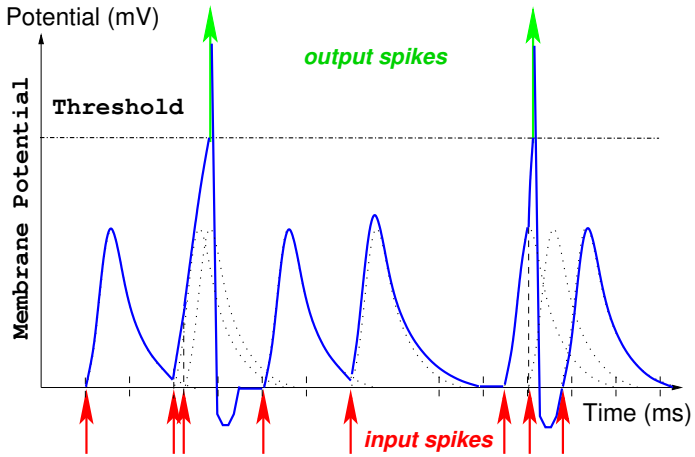
$$\frac{dw}{dt} = a\left(bu(t) - w(t)\right)$$

# Propriétés temporelles des *spiking neurons*

Selon les paramètres du neurone, le potentiel de membrane peut suivre différentes lois ⇒ différents comportements pour le neurone :




**Intégrateur**



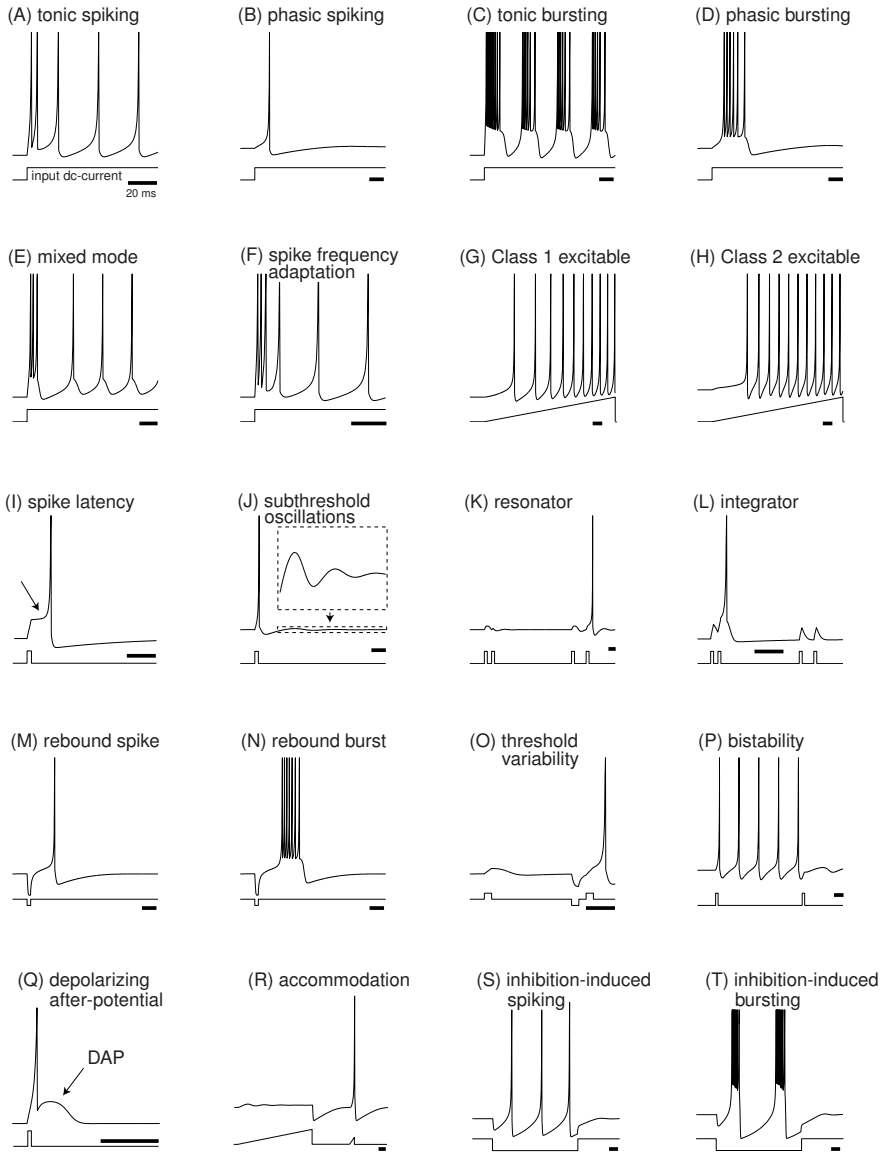

**Détecteur de coïncidence**

[left] most all input spikes participate to postsynaptic firing, whereas [right] only quasi-synchronously incoming spikes trigger an output.

# Reproduction des propriétés neuro-computationelles

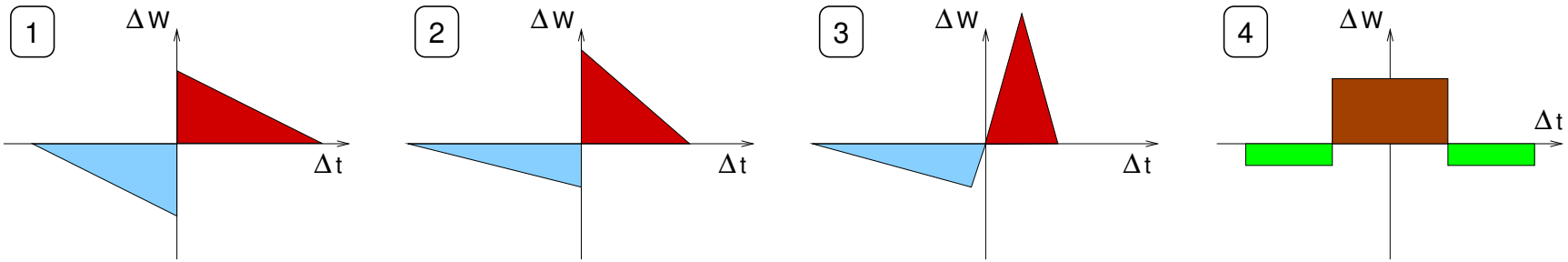
Le neurone d'Izhikevich est capable de reproduire 20 comportements différents parmi ceux observés pour les neurones biologiques.

Le LIF ne peut en reproduire que 3.



# Apprentissage dans un réservoir ?

OUI, il est possible de mettre un apprentissage local et non supervisé, par plasticité synaptique, à l'intérieur d'un résrvoir de *spiking neurons*.




Pour appliquer les fenêtres STDP, la règle d'apprentissage multiplicative est conseillée :

si $DW \leq 0$ diminuer le poids:

$$w_{ij} \leftarrow w_{ij} + \alpha \times (w_{ij} - w_{min}) \times \Delta W$$

si $DW \geq 0$ augmenter le poids:

$$w_{ij} \leftarrow w_{ij} + \alpha \times (w_{max} - w_{ij}) \times \Delta W$$

# Computational learning theory

L'application de la STDP comme règle d'apprentissage local et non supervisé se justifie dans le cadre de la théorie de l'apprentissage :

- Variables cachées et maximum log-likelihood (Barber, 2003) (Pfister, Barber, Gerstner, 2003)
- Maximisation de l'information mutuelle (Chechik, 2003) (Toyoizumi, Pfister, Aihara, Gerstner, 2005a)
- On retrouve le modèle BCM (Izhikevich, Desai, 2003) (Toyoizumi, Pfister, Aihara, Gerstner, 2005b)
- Minimisation de l'entropy (Bohte, Mozer, 2006)

# Applications

Pourquoi le Reservoir Computing ? Parce que... "ça marche" !

Exemples d'applications :

- *control:* Echo State Networks used for motor control
  M. Salmen, P.G. Plöger - ICRA'2005
- *prediction:* Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless telecommunication
  H. Jaeger, H. Hass - Science (2004)
- *pattern recognition:* Isolated word recognition using a LSM
  D. Verstraeten, B. Schrauwen, D. Stroobandt - ESANN'2005
- etc...

# Pourquoi ça marche ?

Le *Reservoir Computing* permet de capter dans les données des aspects temporels ou des caractéristiques de séquence.

Ces modèles, en particulier avec des *spiking neurons*, offrent une meilleure résistance au bruit que les méthodes classiques.

Des résultats qui font progresser l'état de l'art, tel que :
learning the Mackey-Glass chaotic attractor [Jaeger, 2002],
handwritten character / speech recognition [ORGANIC FP-7 project, 2011].

# De récentes avancées

De nombreuses pistes à explorer, pour suivre des avancées prometteuses :

- Apprentissage des poids mais aussi des délais.
  [Paugam-Moisy et al., 2008]
- Hiérarchie de réservoirs $\Rightarrow$ plusieurs échelles de temps
  [Jaeger et al. 2010]
- Couplage de réservoirs avec des HMMs $\Rightarrow$ dépasse l'état de l'art.
- Notion de recurrent kernels.
- Un modèle de reservoir computing pour communiquer oralement avec un robot. [Hinaut, Dominey, 2013]

# Éléments de bibliographie sur *Reservoir Computing*

Jaeger, H.. The "echo state" approach to analysing and training recurrent neural networks. TR-GMD-148, 2001.
Maass, W. and Natschläger, T. and Markram, H.. Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation, 14(11):2531–2560, 2002.
Schrauwen, B. and Verstraeten, D. and Van Campenhout, J.. An overview of reservoir computing: theory, applications and implementations. In Proc. of ESANN'2007.
L. Büsing and W. Maass. Simplified rules and theoretical analysis for information bottleneck optimization and PCA with spiking neurons. In Proc. of NIPS'2007.
Paugam-Moisy, H. and Martinez, R. and Bengio, S.. Delay learning and polychronization for reservoir computing. Neurocomputing, 71(7-9):1134–11587, 2008.
Schrauwen, B. and Büsing, L. and Legenstein, R.. On computational power and the order-chaos phase transition in Reservoir Computing. In Proc. of NIPS'2009.
M. Lukoševičius, H. Jaeger. Reservoir computing approaches to recurrent neural network training. Computer Science Review, 3:127–149, 2009.
Paugam-Moisy and Bohte. Computing with Spiking Neuron Networks. In Rozenberg, Bck and Kok, Eds, *Handbook of Natural Computing*, 335376. Springer, 2012.

# Quelques pointeurs intéressants

Bibliothèque logicielle pour implémenter le *Reservoir Computing* :
OGER (OrGanic Environment for Reservoir computing)
http://organic.elis.ghent.be/organic/engine

Les tutoriels, démonstrations et scripts qui sont sur le site du projet ORGANIC : http://organic.elis.ghent.be/organic

La page web "recherche" de Herbert Jaeger (avec un simulateur d'ESN)

Plus généralement, sur les réseaux de *spiking neurons*, le chapitre *survey* :
"Computing with Spiking Neuron Networks" (2012), Paugam-Moisy and Bohte.
In the "Handbook of Natural Computing", Springer

# Conclusion

Deux nouveaux courants de recherche en réseaux de neurones ont émergé en 2006/2007 : Deep Learning et Reservoir Computing

Le premier est majoritairement nord-américain (Canada, USA) et le second est surtout nord-européen (Pays-Bas, Allemagne, Autriche, Suisse, France)

Le premier développe de nouveaux paradigmes d'apprentissage pour les réseaux *feedforward* et le second concerne les réseaux récurrents.

Dans les deux cas, de nombreux résultats dans les domaines d'applications courants (ex. reconnaissance des formes) améliorent l'état de l'art.

Les deux domaines vont sûrement se rencontrer puisque...
on commence à empiler des réservoirs !