Master Recherche IAC
TC2: Apprentissage Statistique & Optimisation

Alexandre Allauzen – Anne Auger – Michèle Sebag
LIMSI – LRI

Oct. 4th, 2012

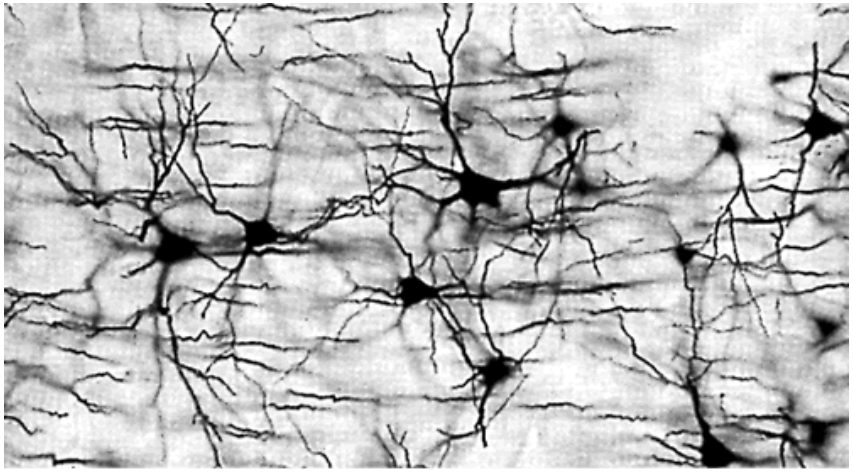
This course

Bio-inspired algorithms

Classical Neural Nets

- History
- Structure

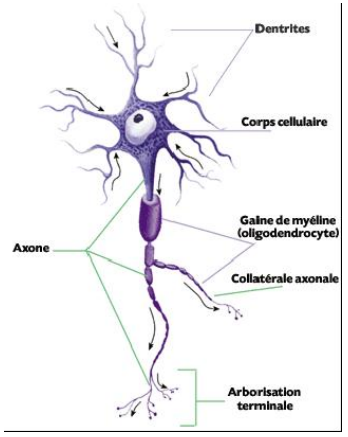
Bio-inspired algorithms



Facts

- 10^{11} neurons
- 10^4 connexions per neuron
- Firing time: $\sim 10^{-3}$ second

10^{-10} computers



Bio-inspired algorithms, 2

Human beings are the best !

- How do we do ?
  - What matters is not the number of neurons
    as one could think in the 80s, 90s...
  - Massive parallelism ?
  - Innate skills ? = anything we can't yet explain
  - Is is the training process ?

Synaptic plasticity

Hebb 1949 Conjecture

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

Learning rule

Cells that fire together, wire together

If two neurons are simultaneously excitated, their connexion weight increases.

Remark: unsupervised learning.

This course

Bio-inspired algorithms

Classical Neural Nets

- History
- Structure

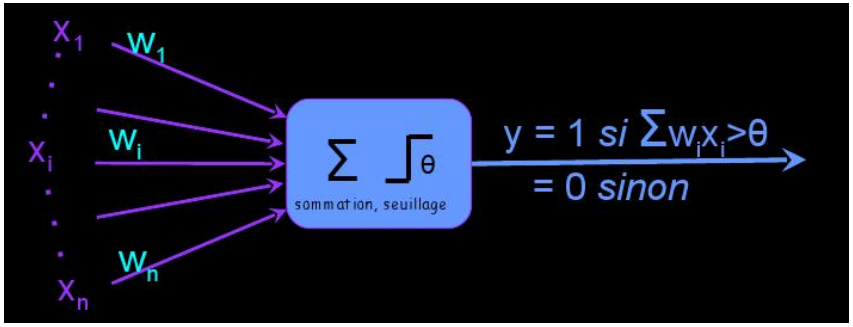
History of artificial neural nets (ANN)

1. Non supervised NNs and logical neurons
2. Supervised NNs: Perceptron and Adaline algorithms
3. The NN winter: theoretical limitations
4. Multi-layer perceptrons.

History


Historique du connexionnisme
1940
1950
1960
1970
1980
1990
2000
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

Thresholded neurons Mc Culloch et Pitt 1943



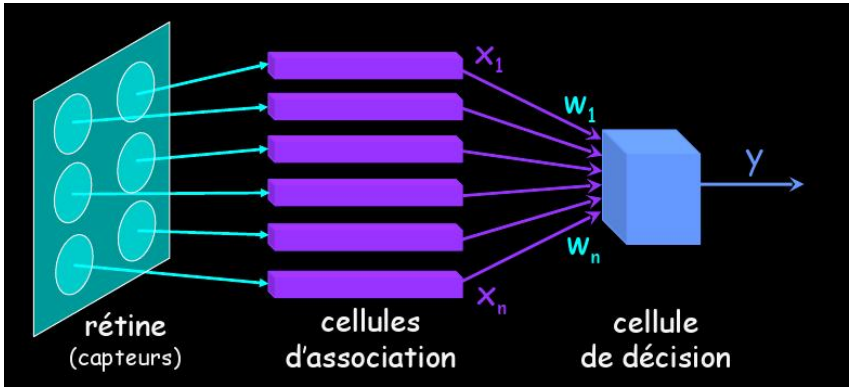
Ingredients

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

Remarks

- Neurons $\rightarrow$ Logics $\rightarrow$ Reasoning $\rightarrow$ Intelligence
- Logical NNs: can represent any boolean function
- No differentiability.

Perceptron Rosenblatt 1958



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

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

$$y = sign(\langle \mathbf{w}, \mathbf{x} \rangle)$$

Learning a Perceptron

Given

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

For $i = 1 \dots n$, do

- If no mistake, do nothing

$$\text{no mistake} \Leftrightarrow \langle \mathbf{w}, \mathbf{x} \rangle \text{ same sign as } y$$
$$\Leftrightarrow y \langle \mathbf{w}, \mathbf{x} \rangle > 0$$

- If mistake

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

Enforcing algorithmic stability:

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

α_t decreases to 0 faster than $1/t$.

Convergence: upper bounding the number of mistakes

Assumptions:

- $\mathbf{x}_i$ belongs to $\mathcal{B}(\mathbb{R}^d, C)$ $\quad ||\mathbf{x}_i|| < C$
- $\mathcal{E}$ is separable, i.e.
  exists solution $\mathbf{w}^*$ s.t. $\forall i = 1 \dots n,\ y_i \langle \mathbf{w}^*, \mathbf{x}_i \rangle > \delta > 0$

Convergence: upper bounding the number of mistakes

Assumptions:

- $\mathbf{x}_i$ belongs to $\mathcal{B}(\mathbb{R}^d, C)$ $\quad ||\mathbf{x}_i|| < C$
- $\mathcal{E}$ is separable, i.e.
  exists solution $\mathbf{w}^*$ s.t. $\forall i = 1 \dots n,\ y_i \langle \mathbf{w}^*, \mathbf{x}_i \rangle > \delta > 0$
  with $||w^*|| = 1$.

Convergence: upper bounding the number of mistakes

Assumptions:

- $\mathbf{x}_i$ belongs to $\mathcal{B}(\mathbb{R}^d, C)$ $\quad ||\mathbf{x}_i|| < C$
- $\mathcal{E}$ is separable, i.e.
  exists solution $\mathbf{w}^*$ s.t. $\forall i = 1 \dots n,\ y_i \langle \mathbf{w}^*, \mathbf{x}_i \rangle > \delta > 0$
  with $||w^*|| = 1$.

Then The perceptron makes at most $(\frac{C}{\delta})^2$ mistakes.

Bouding the number of misclassifications

Proof

Upon the k-th misclassification for some $\mathbf{x}_i$

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

In the meanwhile:

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

Therefore:

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

Going farther...

Remark: Linear programming: Find $\mathbf{w}, \delta$ such that

$$\textit{Max } \delta, \quad \textit{subject to}$$
$$\forall\, i = 1 \ldots n,\ y_i \left\langle \mathbf{w}, \mathbf{x}_i \right\rangle > \delta$$

gives the floor to Support Vector Machines...

Adaline Widrow 1960

Adaptive Linear Element

Given

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

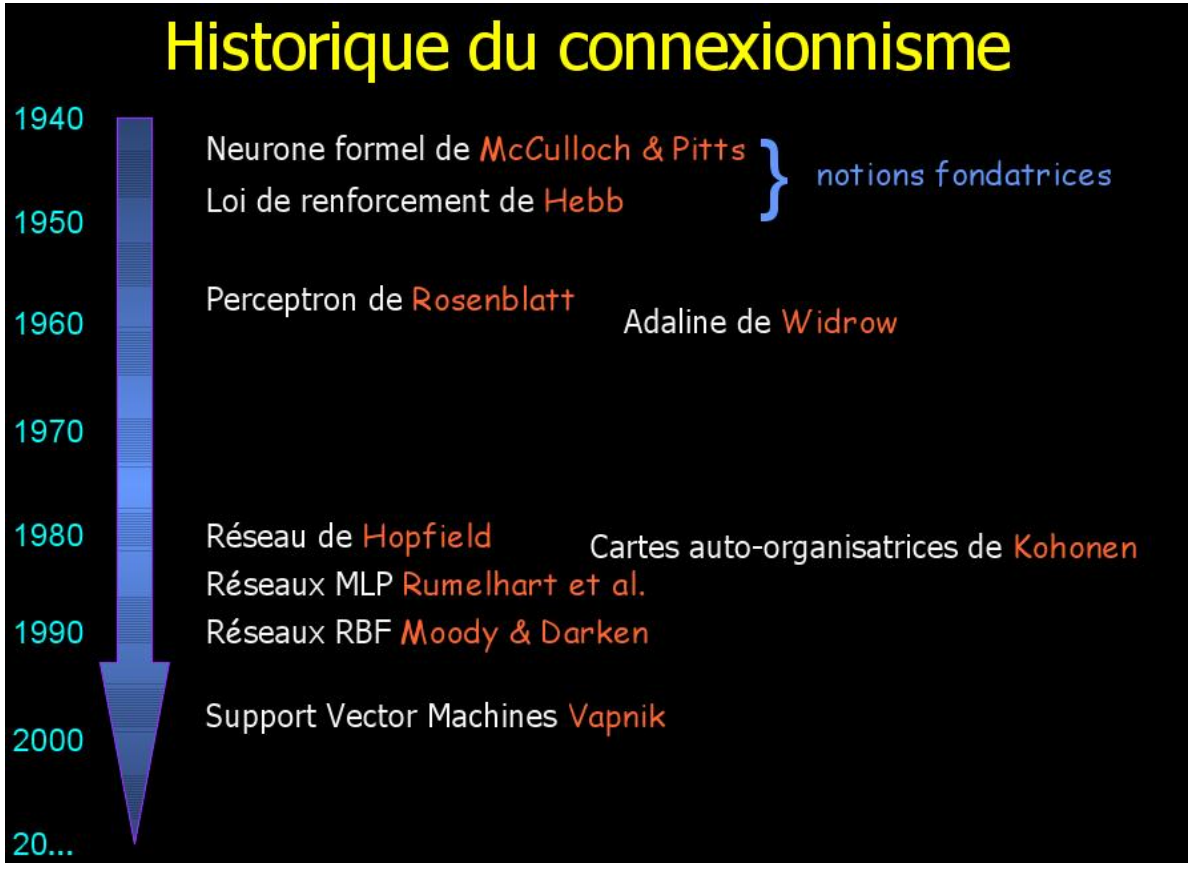
Learning Minimization of a quadratic function

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

Gradient algorithm

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

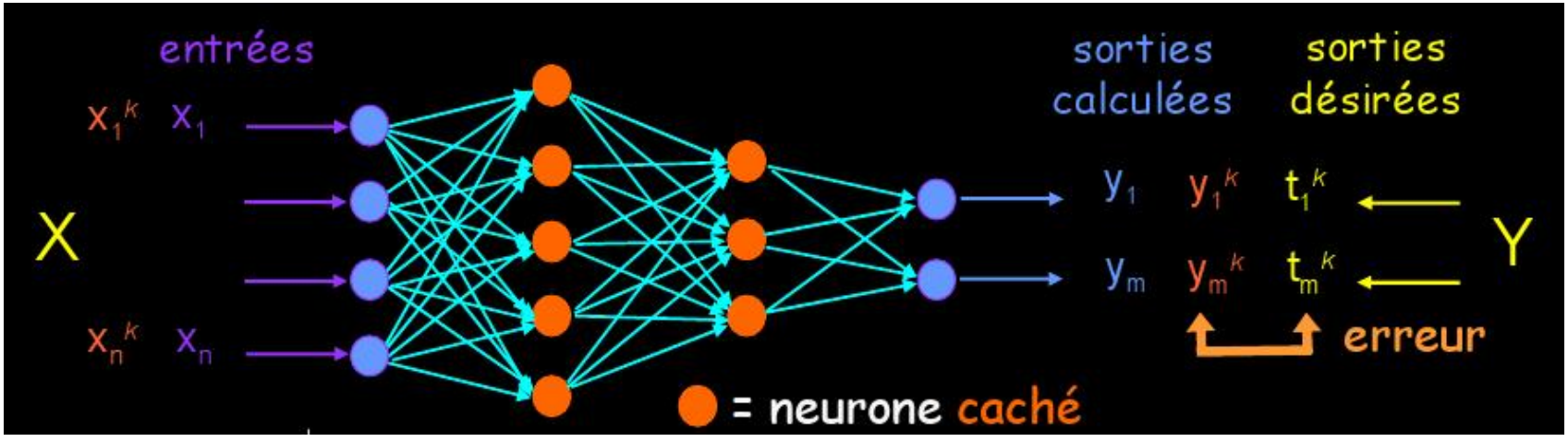
The NN winter



Limitation of linear hypotheses Minsky Papert 1969

The XOR problem.

Multi-Layer Perceptrons, Rumelhart McClelland 1986



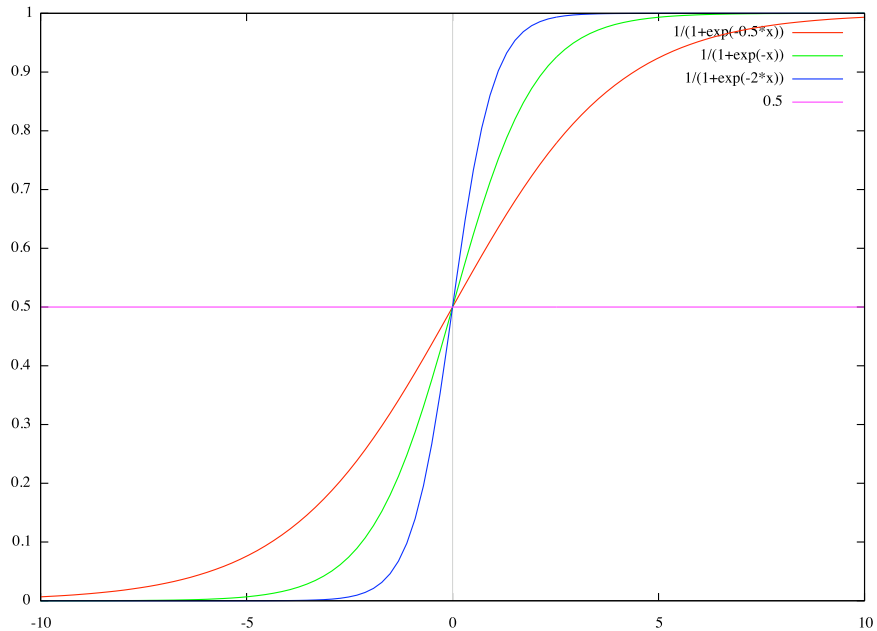
Issues

- Several layers, non linear separation, addresses the XOR problem
- A **differentiable** activation function

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

The sigmoid function

- $\sigma(t) = \frac{1}{1+exp(-a.t)}, a > 0$
- approximates step function (binary decision)
- linear close to 0
- Strong increase close to 0
- $\sigma'(x) = a\sigma(x)(1 - \sigma(x))$



Back-propagation algorithm, Rumelhart McClelland 1986; Le Cun 1986

Intuition

- Given $(\mathbf{x}, y)$ a training sample uniformly randomly drawn
- Set the d entries of the network to $x_1 \dots x_d$
- Compute iteratively the output of each neuron until final layer: output $\hat{y}$;
- Compare $\hat{y}$ and y $Err(w) = (\hat{y} - y)^2$
- Modify the NN weights on the last layer based on the gradient value
- Looking at the previous layer: we know what we would have liked to have as output; infer what we would have liked to have as input, i.e. as output on the previous layer. And back-propagate...
- Errors on each i-th layer are used to modify the weights used to compute the output of i-th layer from input of i-th layer

Back-propagation of the gradient

Notations

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

From input to the first hidden layer

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

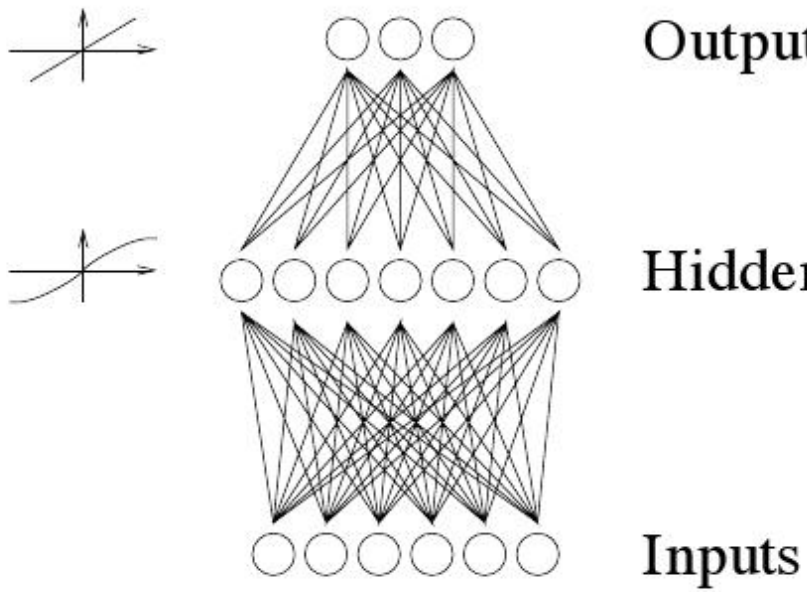
$$x_j^{(1)} = f(z_j^{(1)})$$

From layer i to layer $i+1$

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

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

(f: e.g. sigmoid)



Back-propagation of the gradient

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

Phase 1 Propagate information forward

- For layer $i = 1 \dots \ell$

  For every neuron j on layer i

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

Phase 2 Compare

- Error: difference between $\hat{y}_j = x_j^{(\ell)}$ and y_j.
  Modify $\hat{y}_j$ by

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

Back-propagation of the gradient

Phase 3 retro-propagate the errors

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

Phase 4: Update weights on all layers

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

where α is the learning rate ($< 1.$)

This course

Bio-inspired algorithms

Classical Neural Nets

- History
- Structure

Neural nets

Ingredients

- Activation function
- Connexion topology = directed graph
  feedforward (≡ DAG, directed acyclic graph) or recurrent
- A (scalar, real-valued) weight on each connexion

Activation(z)

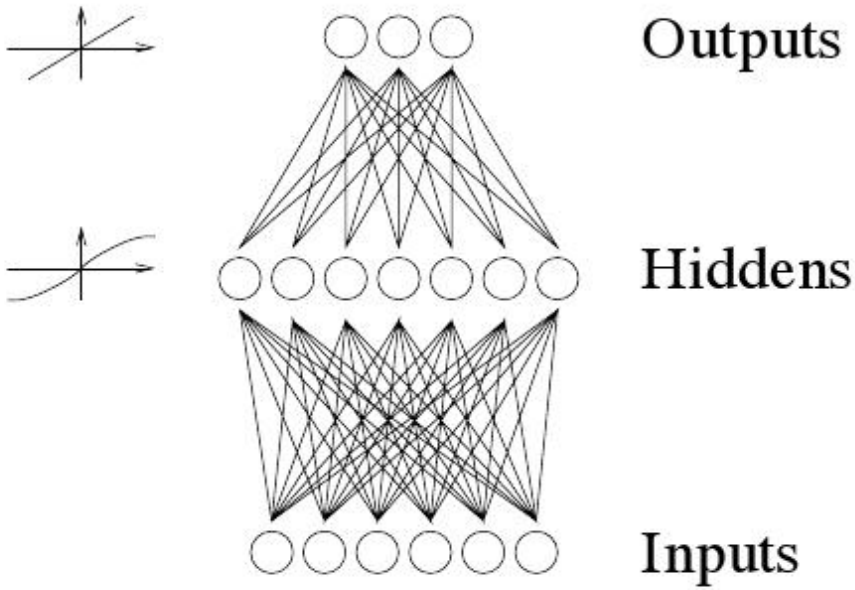
- thresholded — 0 si $z < \mathit{seuil}$, 1 sinon
- linear — z
- sigmoid — $1/(1+e^{-z})$
- Radius-based — e^{-z^2/σ^2}

Neural nets

Ingredients

- Activation function
- Connexion topology = directed graph
  feedforward (≡ DAG, directed acyclic graph) or recurrent
- A (scalar, real-valued) weight on each connexion

Feedforward NN



Labex DigiCosme

Neural nets

Ingredients

- Activation function
- Connexion topology = directed graph
  feedforward (≡ DAG, directed acyclic graph) or recurrent
- A (scalar, real-valued) weight on each connexion

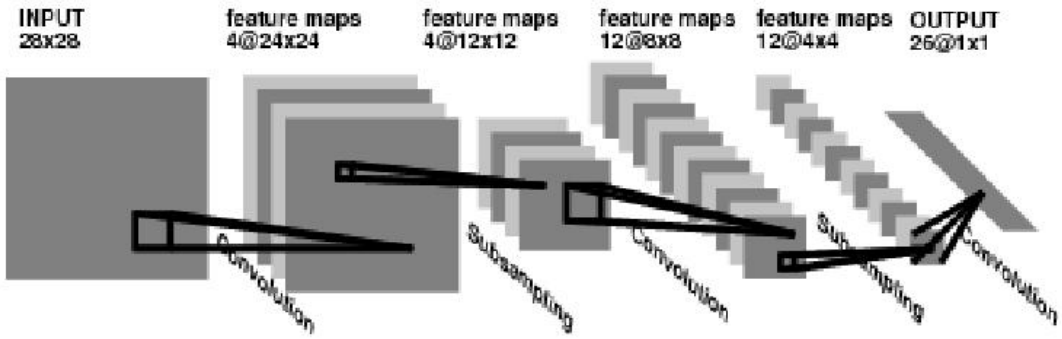
Recurrent NN

- Propagate until stabilisation
- Back-propagation does not apply
- Memory of the recurrent NN: value of hidden neurons
  Beware that memory fades exponentially fast
- Dynamic data (audio, video)

Structure / Connexion graph / Topology

Prior knowledge

- Invariance under translation, rotation,.. *op*
- $\rightarrow$ Complete $\mathcal{E}$ consider $(op(\mathbf{x}_i), y_i)$
- or use weight sharing: convolutionnal networks



100,000 weights $\rightarrow$ 2,600 parameters **Details**

- http://yann.lecun.com/exdb/lenet/ Demos
- http://deeplearning.net/tutorial/lenet.html

Labex DigiCosme

Hubel & Wiesel 1968

Visual cortex of the cat

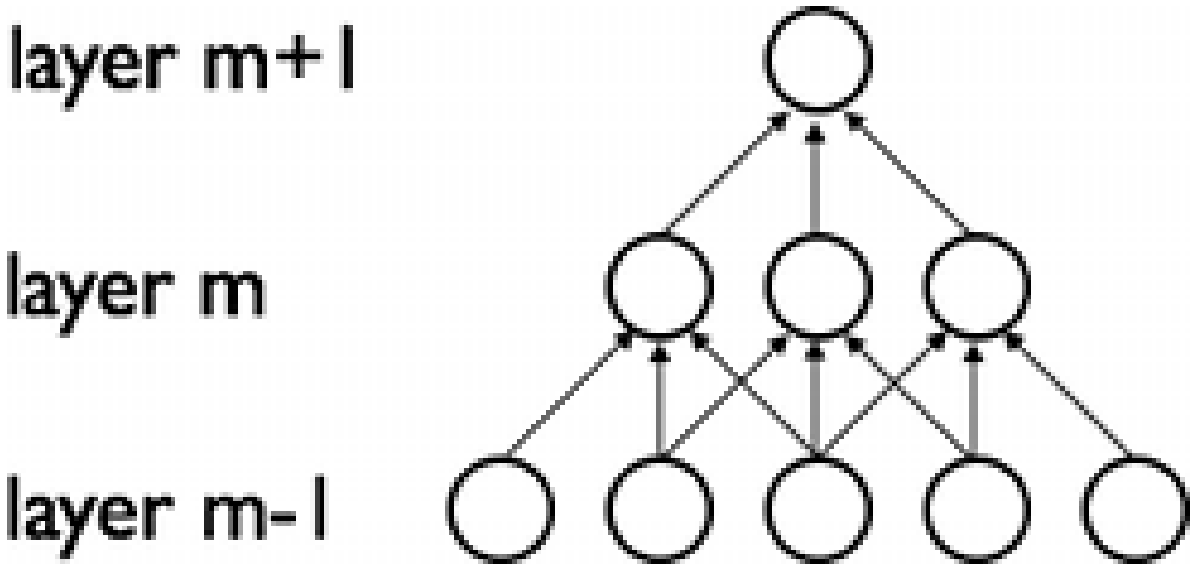
- cells arranged in such a way that
- ... each cell observes a fraction of the visual field

  receptive field
- the union of which covers the whole field

Characteristics

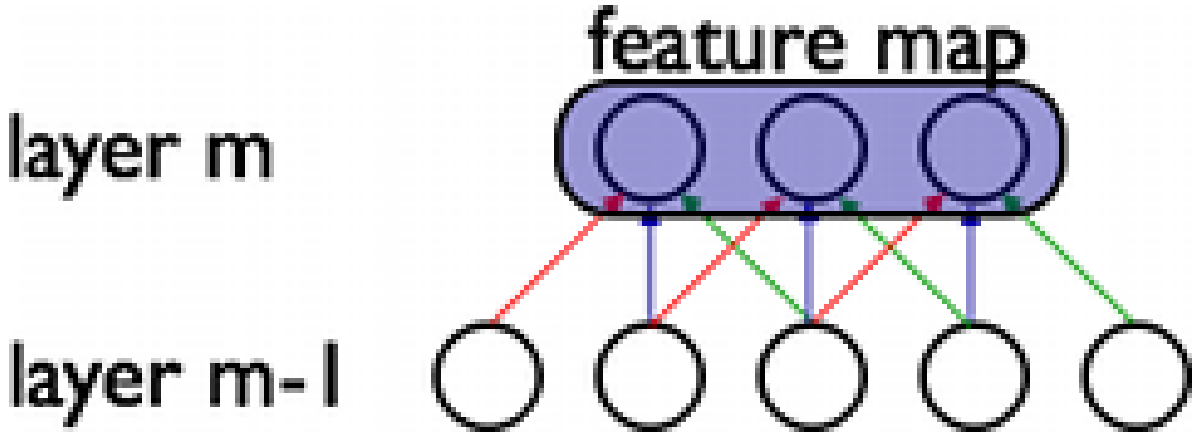
- Simple cells check the presence of a pattern
- More complex cells consider a larger receptive field, detect the presence of a pattern up to translation/rotation

Sparse connectivity



- Reducing the number of weights
- Layer m: detect local patterns
- Layer $m+1$: non linear aggregation, more global field

Convolutional NN: shared weights

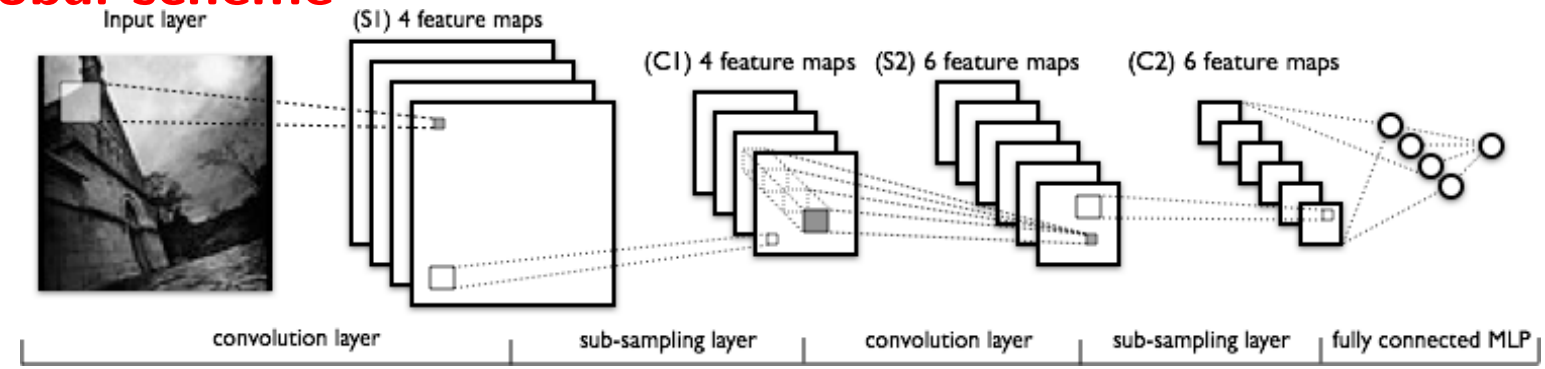


- Reducing the number of weights
- through adapting the gradient-based update: the update is averaged over all occurrences of the weight.

Max pooling: reduction et invariance

- Partitioning
- Return the max value in the subset invariance

Global scheme



Properties

Good news

- MLP, RBF: universal approximators

For every (decent) function f ($= f^2$ has a finite integral on every compact of $\mathbb{R}^d$) for every $\epsilon > 0$, there exists some MLP/RBF g such that $||f - g|| < \epsilon$.

Bad news

- Not a constructive proof (the solution exists, and then ?)
- Everything is possible; $\rightarrow$ no guarantee (overfitting).

Key issues

Model selection

- Selecting number of neurons, connexion graph
- Which learning criterion overfitting

More $\not\Rightarrow$ Better

Algorithmic choices

a difficult optimization problem

- Initialisation **w** small !
- Decrease the learning rate with time
- Enforce stability through relaxation

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

- Stopping criterion

Start by normalization of data

$$X \mapsto \frac{x - average}{variance}$$

The curse of NNs

- **The NIPS community has suffered of an acute convexivitis epidemic**
  - ML applications seem to have trouble moving beyond logistic regression, SVMs, and exponential-family graphical models.
  - For a new ML model, convexity is viewed as a virtue
  - Convexity is sometimes a virtue
  - But it is often a limitation

  - ML theory has essentially never moved beyond convex models
    - the same way control theory has not really moved beyond linear systems

  - Often, the price we pay for insisting on convexity is an unbearable increase in the size of the model, or the scaling properties of the optimization algorithm [O(n^2), O(n^3)...]

http://videolectures.net/eml07_lecun_wia/

Pointers

URL

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

Also see

- NEAT & HyperNEAT — Stanley, U. Texas
  When no examples available: e.g. robotics.