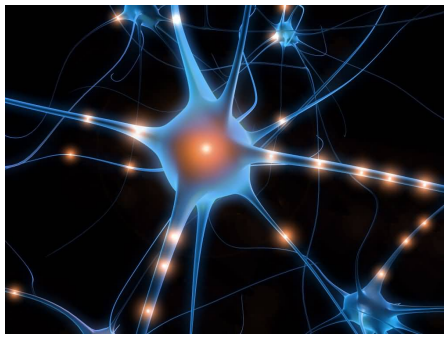
M2 STIM - ≪ Identification & aide à la décision ≫

Réseaux de Neurones

clement.chatelain@insa-rouen.fr

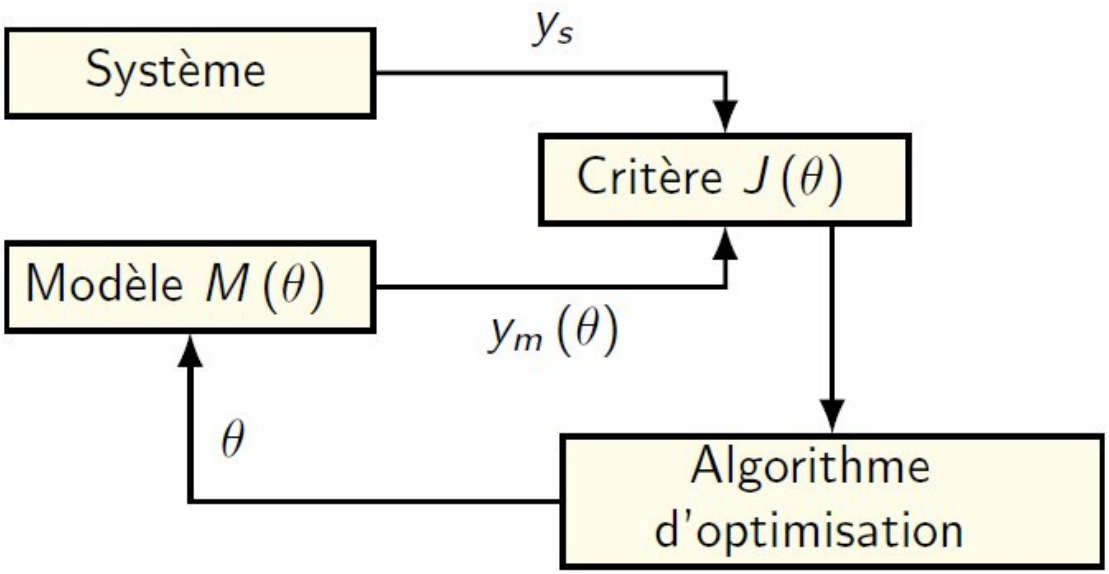
26 janvier 2017



INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Sommaire

Introduction



Avantage

- Fonctionne $\forall$ le nombre d'entrées, $\forall$ le nombre de sorties
- Modèle pas forcément linéaire par rapport aux paramètres
- Et surtout ... ça marche très bien !

Inconvénients

- Le critère doit être dérivable
- Le paramétrage et l'apprentissage demandent un peu d'éxpérience ...

Introduction

Les réseaux de neurones permettent d'estimer une fonction f :

$$f : x \rightarrow y$$

avec $x^T = [x_1, x_2, \ldots, x_E] \in \mathbb{R}^E$

- Si $y \in \mathbb{R}^S$, on parle de **régression**
- Si $y \in \{C_1, C_2, \ldots, C_S\}$, on parle de **classification**
  Dans ce cas, autant de neurones de sortie que de classe
  $\rightarrow$ Sorties désirées de la forme : ${y^d}^T = [0, 0, \ldots, 1, \ldots, 0]$

Estimation de f :

- Apprentissage des poids de connexion entre neurones
- Sur une base étiquetée de N couples
  $(\{x(1), y(1)\}, \ldots, \{x(n), y(n)\}, \ldots, \{x(N), y(N)\})$

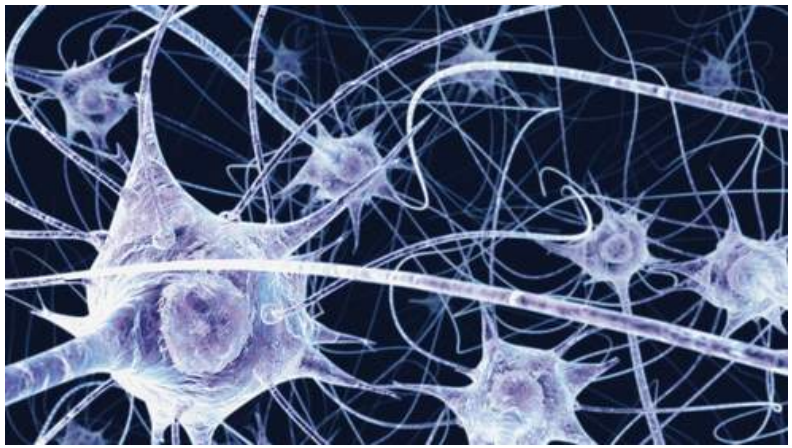
Principes généraux

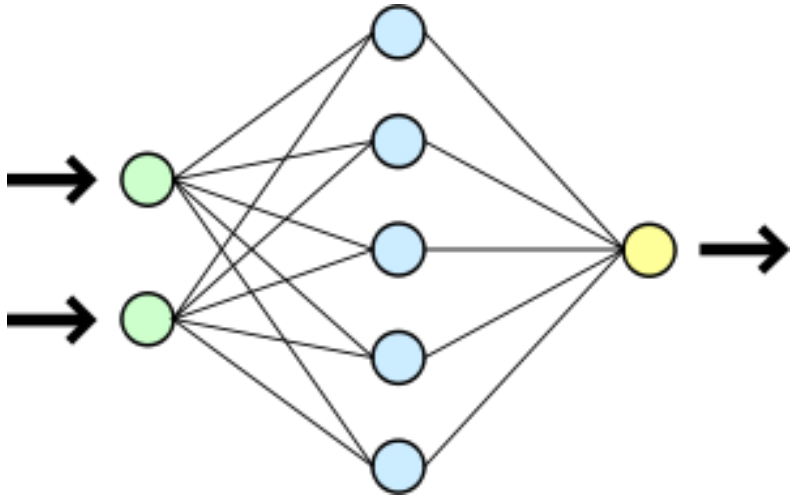
Idée générale des Réseaux de neurones :

- combiner de nombreuses fonctions élémentaires pour former des fonctions complexes.
- Apprendre les liens entre ces fonctions simples à partir d'exemples étiquetés

Analogie (un peu commerciale) avec le cerveau :

- Fonctions élémentaires = neurones
- Connexion = synapse
- Apprentissage des connexions = la connaissance





Le neurone formel [McCulloch et Pitts, 1943]

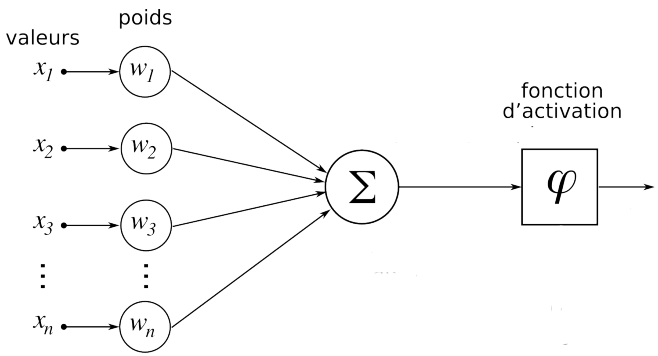
Unité élémentaire : neurone formel

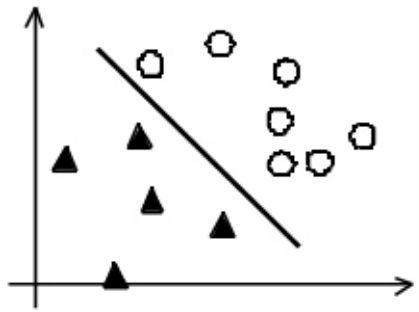
- E entrées x_e, sortie y
- Somme des entrées x_e pondérée par des **poids** w_e :

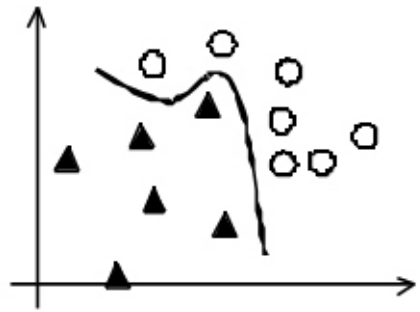
$$\alpha = \sum_{e=1}^{E} w_e x_e + b = \sum_{e=0}^{E} w_e x_e \quad \text{avec} \quad x_0 = 1$$

- Une fonction d'activation φ, linéaire ou non :
  $y = \varphi(\alpha) = \varphi\left(\sum_{e=0}^{E} w_e x_e\right)$

φ linéaire : hyperplan séparateur ; φ non linéaire : hyperbole dimension E





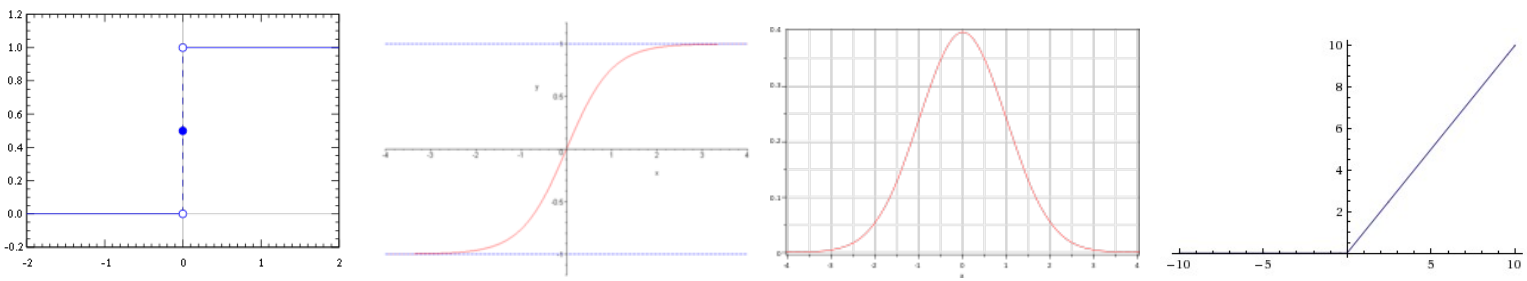


Le neurone formel [McCulloch et Pitts, 1943]

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
- sigmoide : $\varphi(x) = \frac{1}{1+e^{-x}}$
- tanh : $\varphi(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1}$
- fonction noyau (gaussienne)
- ReLU



heaviside - tanh - gaussienne - ReLU

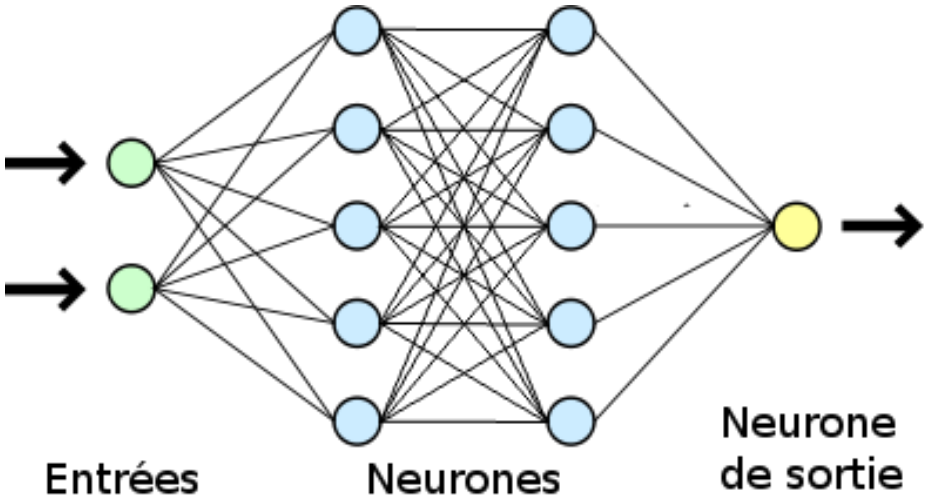
Topologies

Il existe de nombreuses manière d'organiser les neurones en réseau :

- Réseau en couches (adaline, perceptron, perceptron multicouches, RBF)
- Réseau totalement interconnecté (Hopfield, Boltzmann)
- Réseau récurrent (LSTM)
- Réseau à convolution (TDNN, SDNN)
- Réseau avec beaucoup de couches ! (architectures profondes)

Réseaux en couches (1)

- Chaque neurone d'une couche est connecté à tous les neurones des couches précédentes et suivantes
- Réseaux dits « feedforward » : propagation des entrées de couches en couches vers la sortie

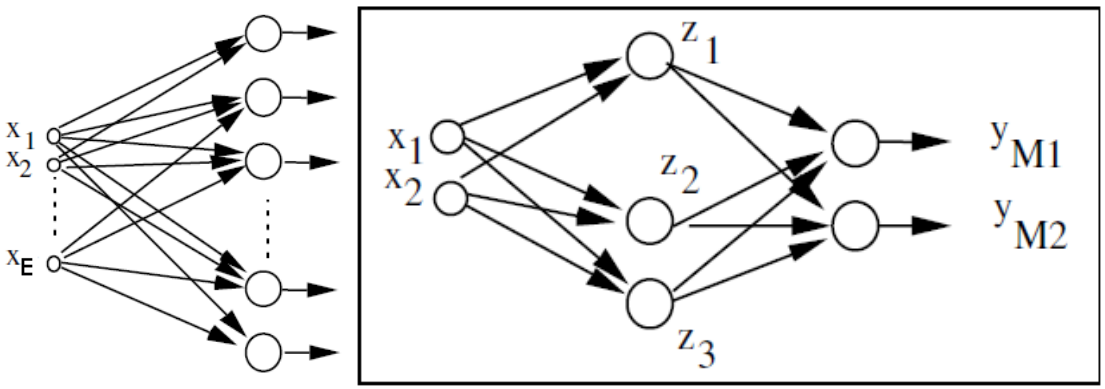


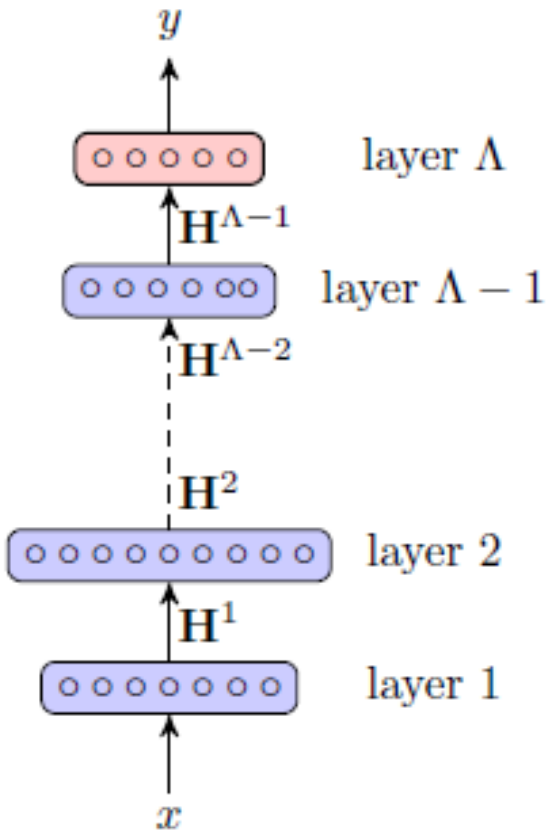
- Structure la plus répandue
- Algorithmes d'apprentissage des poids efficaces

Réseaux en couches (2)

Variantes

- Une couche, fonction d'activation heaviside, une sortie : *perceptron* [Rosenblatt 1957]
- Si plus d'une couche : couches dites « cachées », perceptron multicouches
- Si beaucoup de couches : architectures profondes

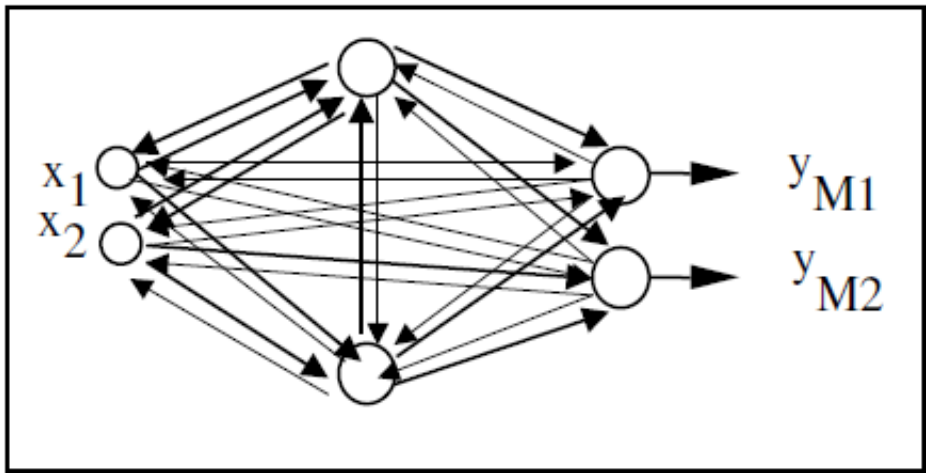




Réseaux totalement interconnectés

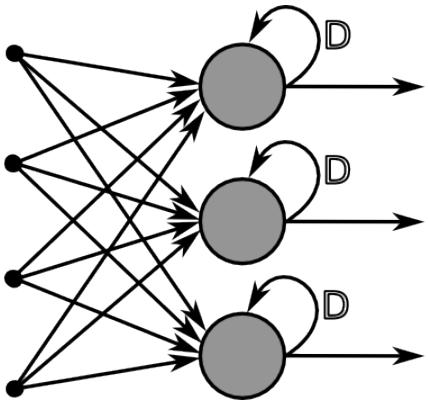
Réseaux de Hopfield, Machines de Boltzmann

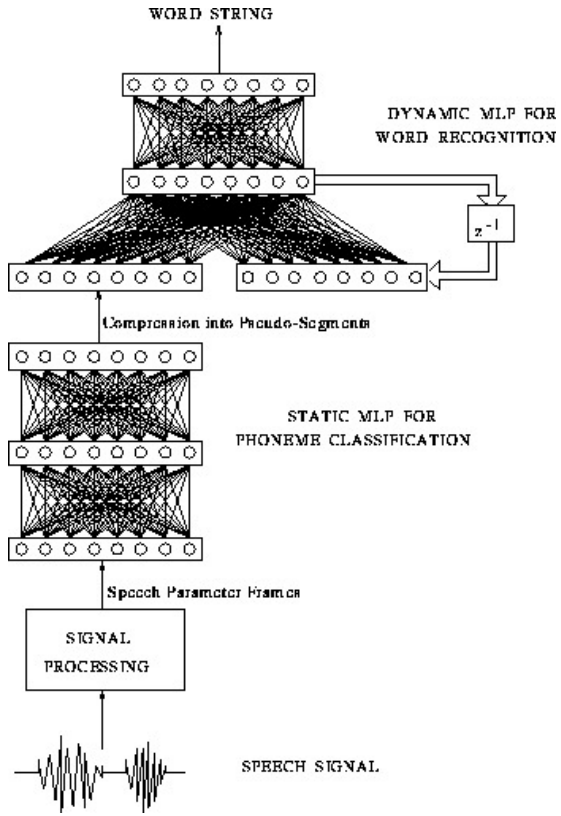
- Tous les neurones sont connectés entre eux
- Difficile à entrainer
- N'a jamais vraiment prouvé son utilité sur des problèmes réels
  $\rightarrow$ intérêt essentiellement théorique



Réseaux récurrents

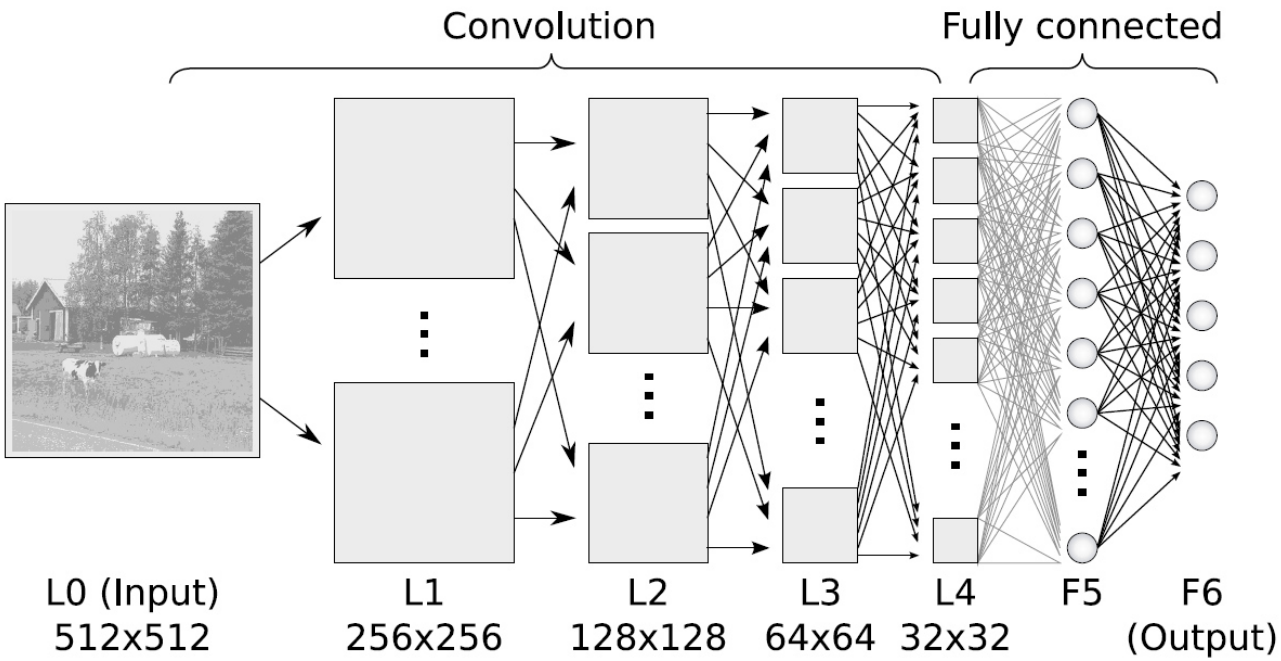
- Adapté aux séquences
- Permet de prendre en compte le contexte
- On calcule $y(n)$ à partir de :
  - $x(n)$ les entrées courantes
  - $y(n-1)$ les sorties de l'exemple précédent
    (provenant d'une même séquence)
- Hypothèse $\simeq$ Markovienne





Réseaux convolutionnels

- Poids partagés, connexions locales
- Apprentissage de configurations particulières



Plan

Notations (1)

Les données

- On dispose d'une base étiquetée de N couples $\{x(n), y^d(n)\}$
- $\mathbf{X} \in \mathbb{R}^{E \times N}$, $\mathbf{Y}^d \in \mathbb{R}^{S \times N}$

$$\mathbf{X} = \{x(n)\} = \left\{ \begin{bmatrix} x_1(n) \\ \vdots \\ x_E(n) \end{bmatrix} \right\} = \begin{bmatrix} x_1(1) & \dots & \dots & x_1(N) \\ \vdots & \ddots & x_e(n) & \vdots \\ x_E(1) & \dots & \dots & x_E(N) \end{bmatrix}$$

$$\mathbf{Y}^d = \{y^d(n)\} = \left\{ \begin{bmatrix} y_1^d(n) \\ \vdots \\ y_E^d(n) \end{bmatrix} \right\} = \begin{bmatrix} y_1^d(1) & \dots & \dots & y_1^d(N) \\ \vdots & \ddots & y_s^d(n) & \vdots \\ y_S^d(1) & \dots & \dots & y_S^d(N) \end{bmatrix}$$

Notations(2)

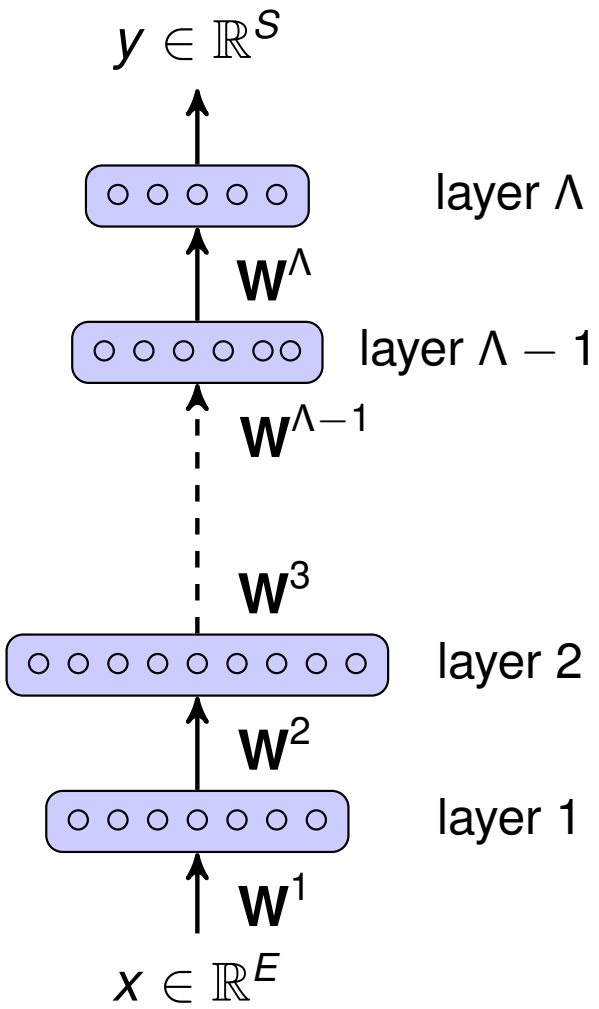
Le réseau (en couche)

- E entrée, S sorties
- Le réseau comporte Λ couches
- W^λ matrice des poids entre couches $\lambda - 1$ et λ
- On appelera $y^d(n)$ la sortie désirée pour l'exemple n

Si :
- la couche $\lambda - 1$ contient n_i neurones
- la couche λ n_j neurones,

alors :

$$\mathbf{W}^\lambda = \{w_{ji}\} = \begin{bmatrix} w_{11} & \dots & w_{1i} & \dots & w_{1n_i} \\ \vdots & & \vdots & & \\ w_{j1} & & w_{ji} & & w_{jn_i} \\ \vdots & & \vdots & & \\ w_{n_j1} & \dots & w_{n_ji} & \dots & w_{n_jn_i} \end{bmatrix}$$



Le problème

Rappel : on souhaite estimer f :

- Apprentissage sur la base des poids de connexion entre neurones W

$\rightarrow$ Critère

Critère des moindres carrés (dérivable) :

$$\mathcal{J}(W) = \sum_{n=1}^{N} e(n)^T e(n) \quad \text{avec} \quad e(n) = \Big(y(n) - y^d(n)\Big)$$

Qu'on peut réécrire en sommant sur les sorties :

$$\mathcal{J}(W) = \sum_{s=1}^{S} \sum_{n=1}^{N} (e_s(n))^2 = \sum_{s=1}^{S} \mathcal{J}(W_s) \tag{1}$$

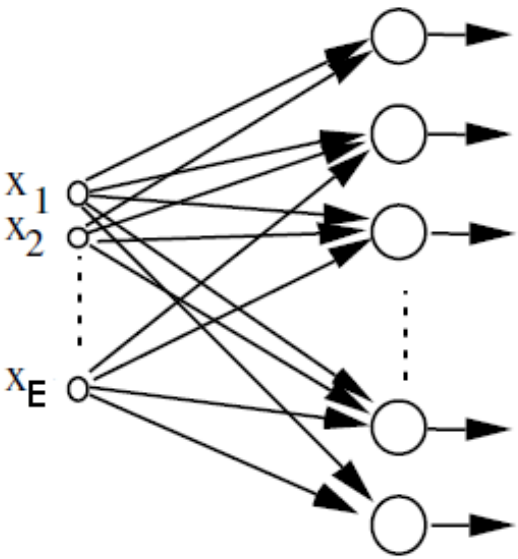
Avec $e_s(n) = \big(y_s(n) - y_s^d(n)\big)$

Plan

Réseau linéaire à une couche (1)

- Une couche de neurones avec une fonction d'activation φ = identité

$$\text{poids} : \mathbf{W} = \{w_{se}\} = \begin{bmatrix} W_1^T \\ \vdots \\ W_S^T \end{bmatrix} = \begin{bmatrix} w_{11} & \dots & w_{1E} \\ \vdots & w_{se} & \vdots \\ w_{S1} & \dots & w_{SE} \end{bmatrix}$$



Propagation 1 ex. sur une sortie : $y_s(n) = W_s^T x(n)$

$$\begin{bmatrix} y_s \end{bmatrix} = \begin{bmatrix} w_{s1} & \dots & w_{sE} \end{bmatrix} \times \begin{bmatrix} x_1 \\ \vdots \\ x_E \end{bmatrix}$$

Propagation 1 ex. sur toutes les sorties : $y(n) = \mathbf{W}x(n)$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_S \end{bmatrix} = \begin{bmatrix} w_{11} & \dots & w_{1E} \\ \vdots & w_{se} & \vdots \\ w_{S1} & \dots & w_{SE} \end{bmatrix} \times \begin{bmatrix} x_1 \\ \vdots \\ x_E \end{bmatrix}$$

Propagation N ex. sur S sortie : $\mathbf{Y} = \mathbf{WX}$

$$\begin{bmatrix} y_1^d(1) & \dots & \dots & y_1^d(N) \\ \vdots & \ddots & y_s^d(n) & \vdots \\ y_S^d(1) & \dots & \dots & y_S^d(N) \end{bmatrix} = \begin{bmatrix} w_{11} & \dots & w_{1E} \\ \vdots & w_{se} & \vdots \\ w_{S1} & \dots & w_{SE} \end{bmatrix} \times \begin{bmatrix} x_1(1) & \dots & \dots & x_1(N) \\ \vdots & \ddots & x_e(n) & \vdots \\ x_E(1) & \dots & \dots & x_E(N) \end{bmatrix}$$

Réseau linéaire à une couche (3)

- Si $\mathbf{Y} = \mathbf{WX}$, alors $\mathbf{Y}^T = \mathbf{X}^T\mathbf{W}^T$ ...
- ... qui est de la forme $Y = X\Theta$, en remplacant $\mathbf{X}$ et $\mathbf{Y}$ par leur transposées, et Θ par $\mathbf{W}^T$
- En appliquant les MC, on obtient $\mathbf{W}_{MC}^T = (XX^T)^{-1}XY^T$
- D'ou : $\mathbf{W}_{MC} = YX^T(X^TX)^{-1}$

Conclusion

Apprentissage OK avec les MC, mais :

- Pas de non linéarité = pas terrible
- Une seule couche = pas terrible
- (X^TX) à inverser : potentiellement très lourd (mais MC récursifs possibles)

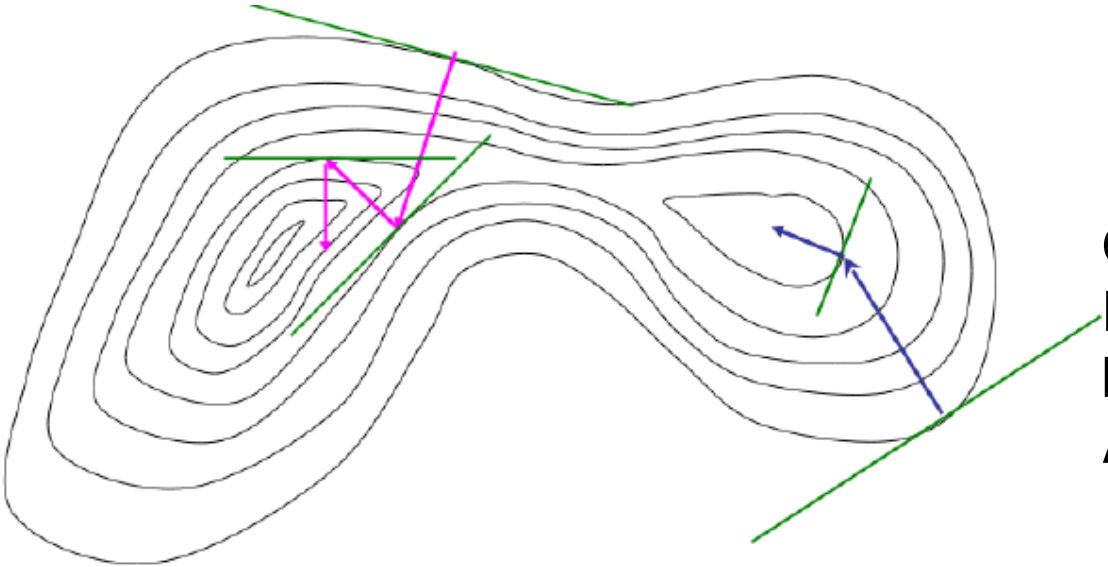
Introduction d'une fonction φ non linéaire

Plan

Réseau non linéaire à une couche (1)

Introduction d'une fonction φ non linéaire

- On a donc $\mathbf{Y} = \varphi(\mathbf{WX})$, et les MC ne sont plus applicables
- On va appliquer une méthode de descente de gradient $\rightarrow$ rappels !



- Algorithme itératif :

On choisit un $\mathbf{W}_{t=0}$ aléatoire
Bonne direction = celle oú 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 :
$\mathbf{W}$ les paramètres ; η : le pas ; $\left.\frac{\mathrm{d}\mathcal{J}(\mathbf{W})}{\mathrm{d}\mathbf{W}}\right|_{\mathbf{W}_t}$: la « bonne »direction

Réseau non linéaire à une couche (2)

Critère

- Pour un réseau à une couche contenant une FNL φ : $y(n) = \varphi(\mathbf{W}x(n))$
- Le critère s'écrit donc :

$$\mathcal{J}(W) = \sum_{n=1}^{N} \left(y^d(n) - \varphi(\mathbf{W}x(n)) \right)^2$$

- On dérive pour appliquer le gradient : $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t - \eta \left. \frac{\mathrm{d}\mathcal{J}(\mathbf{W})}{\mathrm{d}\mathbf{W}} \right|_{\mathbf{W}_t}$

$$\begin{aligned} \frac{\mathrm{d}\mathcal{J}(\mathbf{W})}{\mathrm{d}\mathbf{W}} &= -2 \sum_{n=1}^{N} \left(y^d(n) - \varphi(\mathbf{W}x(n)) \right) \times \frac{\mathrm{d}\varphi(\mathbf{W}x(n))}{\mathrm{d}\mathbf{W}} \\ &= -2 \sum_{n=1}^{N} \left(y^d(n) - \varphi(\mathbf{W}x(n)) \right) \times \varphi'(\mathbf{W}x(n))\, x(n) \end{aligned}$$

Réseau non linéaire à une couche (3)

Application du gradient

$$\frac{\mathrm{d}\mathcal{J}(\mathbf{W})}{\mathrm{d}\mathbf{W}} = -2\sum_{n=1}^{N}\left(y^d(n) - \varphi\left(\mathbf{W}x(n)\right)\right) \times \varphi'\left(\mathbf{W}x(n)\right)x(n)$$

Deux cas de figure :

Si φ n'est pas dérivable (ex. heaviside) : ça ne marche pas !

- Approximation linéaire de la dérivée : algo adaline [Widrow & Hoff 1960]

$$\frac{\mathrm{d}\mathcal{J}(\mathbf{W})}{\mathrm{d}\mathbf{W}} = -2\sum_{n=1}^{N}\left(y^d(n) - \varphi\left(\mathbf{W}x(n)\right)\right) \times x(n)$$

Si φ est dérivable (ex. sigmoide, tanh) : ça marche !

- Dans ce cas on applique la descente de gradient
- Remarque : dans le cas d'une fonction identité, $\varphi' = 1$: ça marche

OK pour une couche, et pour plusieurs ?

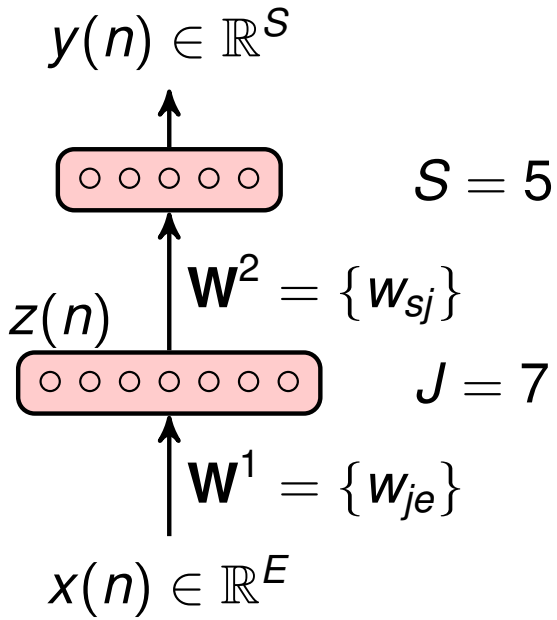
Plan

Réseaux multicouches (1)

- C'est le perceptron multicouches (PMC ou MLP)
  $\rightarrow$ Couches dites cachées
- FNL φ, peuvent être $\neq$ suivant les couches

Exemple d'un réseau à deux couches

- E entrée, S sorties, J neurones en couche cachée
- $\mathbf{W}^1 \in \mathbb{R}^{J \times E}$ poids entre les x et la couche 1
- $\mathbf{W}^2 \in \mathbb{R}^{S \times J}$ poids entre les couches 1 et 2
- $z(n) \in \mathbb{R}^J$: variable intermédiaire
- φ^1 : sigmoide ; φ^2 : softmax



$$\mathbf{W}^1 = \{w_{je}\} = \begin{bmatrix} w_{11} & \dots & w_{1E} \\ w_{j1} & \ddots & w_{jE} \\ w_{J1} & \dots & w_{JE} \end{bmatrix} \qquad \mathbf{W}^2 = \{w_{sj}\} = \begin{bmatrix} w_{11} & \dots & w_{1J} \\ w_{s1} & \ddots & w_{sJ} \\ w_{S1} & \dots & w_{SJ} \end{bmatrix}$$

Réseaux multicouches (2)

Propagation d'un exemple :

- couche 1 : somme pondérée $\alpha_j^1 = \sum_e w_{je} x_e$, puis $z_j = \varphi^1\left(\alpha_j^1\right)$
- couche 2 : somme pondérée $\alpha_s^2 = \sum_j w_{sj} z_j$, puis $y_s = \varphi^2\left(\alpha_s^2\right)$

Apprentissage : Rétropropagation du gradient [Rumelhart 86]

Initialiser les $\mathbf{W}^\lambda$ au hasard

1. Propager un exemple $x(n)$ pour calculer $y(n)$
2. Calculer l'erreur $(y^d(n) - y(n))$
3. Rétropropager le critère $\mathcal{J} = (y^d(n) - y(n))^2$ à travers $\mathbf{W}_2$ ...
4. ... puis à travers $\mathbf{W}_1$

On passe tous les exemples de la base, et on itère tant qu'on n'est pas satisfait

Réseaux multicouches (3)

Chaque sortie s'écrit :

$$y_s = \varphi^2 \left(\sum_j w_{sj} \varphi^1 \left(\sum_e w_{je} x_e \right) \right)$$

Donc le critère $\mathcal{J} = 1/2 \sum_s (y_s^d - y_s)^2$ dépend de $(w_{je}, \varphi^1, w_{sj}, \varphi^2)$.

Apprentissage de w_{je} et w_{sj} :

- Descente de gradient :

$$w_{je}{}_{t+1} \leftarrow w_{je}{}_t - \eta \left. \frac{\partial \mathcal{J}}{\partial w_{je}} \right|_{w_{je}{}_t} \quad \text{et} \quad w_{sj}{}_{t+1} \leftarrow w_{sj}{}_t - \eta \left. \frac{\partial \mathcal{J}}{\partial w_{sj}} \right|_{w_{sj}{}_t}$$

- Problème : comment calculer les dérivées partielles du critère ?

Réseaux multicouches (3)

On commence par le calcul de : $\frac{\partial \mathcal{J}}{\partial w_{sj}} = \frac{\partial \mathcal{J}}{\partial y_s} \times \frac{\partial y_s}{\partial \alpha_s^2} \times \frac{\partial \alpha_s^2}{\partial w_{sj}}$

$$\begin{aligned}
\frac{\partial \mathcal{J}}{\partial y_s} &= \frac{\partial}{\partial y_s} \frac{1}{2} \sum_{s=1}^{S} (y_s^d - y_s)^2 = -(y_s^d - y_s) \\
\frac{\partial y_s}{\partial \alpha_s^2} &= \frac{\partial}{\partial \alpha_s^2} \varphi^2(\alpha_s^2) = \varphi^{2\prime}(\alpha_s^2) \\
\frac{\partial \alpha_s^2}{\partial w_{sj}} &= \frac{\partial}{\partial w_{sj}} \sum_{j=1}^{J} w_{sj} z_j = z_j
\end{aligned}$$

Finalement : $\frac{\partial \mathcal{J}}{\partial w_{sj}} = -(y_s^d - y_s) \times \varphi^{2\prime}(\alpha_s^2) \times z_j = \mathit{Erreur}_s z_j$

Cette quantité représente l'erreur sur la sortie s due au neurone j

Réseaux multicouches (4)

On enchaine avec le calcul de : $\frac{\partial \mathcal{J}}{\partial w_{je}} = \frac{\partial \mathcal{J}}{\partial z_j} \times \frac{\partial z_j}{\partial \alpha_j^1} \times \frac{\partial \alpha_j^1}{\partial w_{je}}$

$$\frac{\partial \mathcal{J}}{\partial z_j} = \sum_s \left[\frac{\partial \mathcal{J}}{\partial \alpha_s^2} \times \frac{\partial \alpha_s^2}{\partial z_j} \right] \quad \text{1er terme : rouge*vert du slide précédent}$$

$$= \sum_s \left[-(y_s^d - y_s) \times \varphi^{2\prime}(\alpha_s^2) \times \frac{\partial}{\partial z_j} \sum_j w_{sj} z_j \right] = -\sum_s (y_s^d - y_s) \times \varphi^{2\prime}(\alpha_s^2) \times w_s$$

$$\frac{\partial z_j}{\partial \alpha_j^1} = \frac{\partial}{\partial \alpha_j^1} \varphi^1(\alpha_j^1) = \varphi^{1\prime}(\alpha_j^1) \qquad \frac{\partial \alpha_j^1}{\partial w_{je}} = \frac{\partial}{\partial w_{je}} \sum_e w_{je} x_e = x_e$$

Finalement :
$\frac{\partial \mathcal{J}}{\partial w_{je}} = -\sum_s (y_s^d - y_s) \times \varphi^{2\prime}(\alpha_s^2) \times w_{sj} \times \varphi^{1\prime}(\alpha_j^1) \times x_e = Erreur_j x_e$
Cette quantité représente l'erreur sur le neurone j due à l'entrée e

Réseaux multicouches (5)

On récapitule :

Algorithm 1 Backpropagation algorithm

$\eta \leftarrow 0.001$
W1 $\leftarrow$ rand(J,E)
W2 $\leftarrow$ rand(S,J)
while (erreurApp $\leq \epsilon$) **do**
  for $n = 1 \rightarrow N$ **do**
    propagate $x(n)$: compute $z(n)$ and $y(n)$
    compute $Error_S$
    $\mathbf{W}^2 \leftarrow \mathbf{W}^2 - \eta * Error_S * z(n)$
    compute $Erreur_J$
    $\mathbf{W}^1 \leftarrow \mathbf{W}^1 - \eta * Error_J * x(n)$
  end for
end while

Réseaux multicouches (5')

On récapitule, en matlab :

```
Function grad = retropropag(x,yd,W1,W2)
:
a1 = [x ones(n,1)]*W1 ; x1 = tanh(a1) ;
a2 = [x1 ones(n,1)]*W2 ; y = a2 ;
errorS = -(yd-y).*(1-y.*y) ;
GradW2 = [x1 ones(n,1)]'* errorS ;
errorJ = (w2(1 :n2-1, :)*errorS')'.*(1-x1.*x1) ;
GradW1 = [x ones(n,1)]'* errorJ ;
w1 = w1 - pas1 .* GradW1 ;
w2 = w2 - pas2 .* GradW2 ;
```

Généralisation (Y.Lecun, voir cours du 12/02/16 [1])

Réseau Feedforward :

Empilement de couches F_i calculant une sortie H_i à partir d'une entrée H_{i-1}, et éventuellement de paramètres W_i et de la sortie Y. Exemples :

- Couche linéaire :

$$H_i = F_i(H_{i-1}, W_i) = W_i H_{i-1}$$

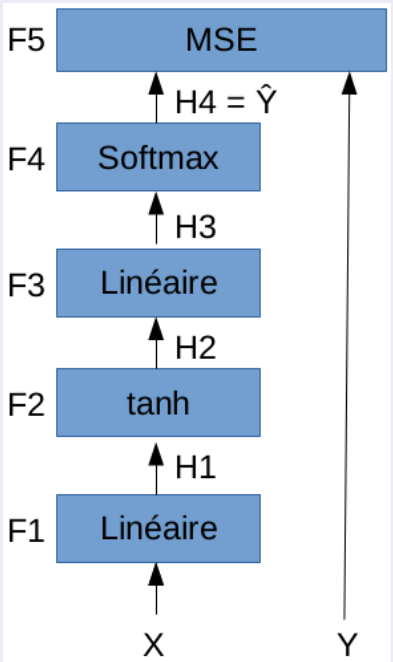
- Couche fonction d'activation (f : *tanh*, ReLU, Softmax, etc.) :

$$H_i = F_i(H_{i-1}) = f(H_{i-1})$$

- Couche MSE :

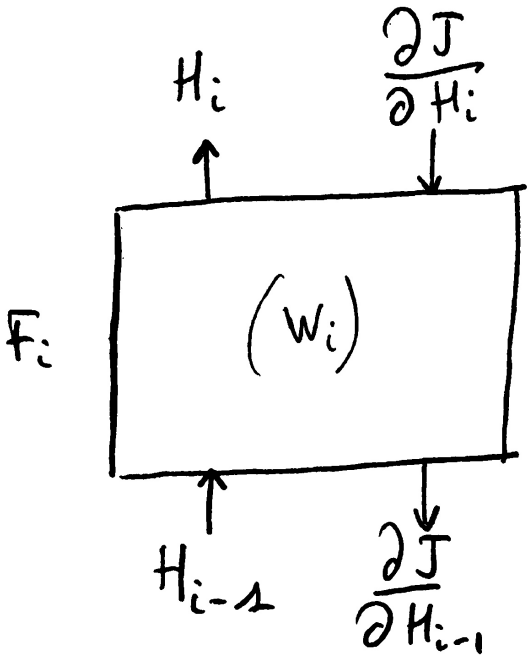
$$H_i = F_i(H_{i-1}, Y) = ||W_i H_{i-1} - Y||^2$$

MLP classique :



1. https://www.college-de-france.fr/site/yann-lecun/course-2016-02-12-14h30.htm

Généralisation (2)



Gradients et Backprop

- Ainsi, on peut faire abstraction des couches
- Calcul des gradients :

$$\frac{\partial J}{\partial H_{i-1}} = \frac{\partial J}{\partial H_i} \times \frac{\partial H_i}{\partial H_{i-1}}$$

- Pour les couches F_i qui en comportent : Mise à jour des poids

$$\frac{\partial J}{\partial W_i} = \frac{\partial J}{\partial H_i} \times \frac{\partial H_i}{\partial W_i}$$

- On applique ces deux équations de manière récursive :
  - le terme en bleu étant calculé précédemment
  - les termes en magenta et vert sont à instancier suivant le type de couche

Généralisation (3)

Instanciation des F_i les plus courants :

- Couche linéaire : $H_i = F_i(H_{i-1}, W_i) = W_i H_{i-1}$ donc :

$$\frac{\partial H_i}{\partial H_{i-1}} = W_i \quad ; \quad \frac{\partial H_i}{\partial W_i} = H_{i-1}$$

- Couche activation : $H_i = F_i(H_{i-1}) = f(H_{i-1})$
avec f : *tanh*, sigmoide, ReLU, Softmax, etc. donc :

$$\frac{\partial H_i}{\partial H_{i-1}} = f'(H_{i-1})$$

- Couche MSE : $H_i = F_i(H_{i-1}, Y) = ||W_i H_{i-1} - Y||^2$ donc :

$$\frac{\partial H_i}{\partial H_{i-1}} = 2W_i \times (W_i H_{i-1} - Y)$$

Généralisation (4)

Algo de backprop generique pour L couches

foreach sample

for $i = 1$ to L

$\frac{\partial J}{\partial H_{i-1}} \leftarrow \frac{\partial J}{\partial H_i} \times \frac{\partial H_i}{\partial H_{i-1}}$

$\frac{\partial J}{\partial W_i} \leftarrow \frac{\partial J}{\partial H_i} \times \frac{\partial H_i}{\partial W_i}$ // si nécessaire

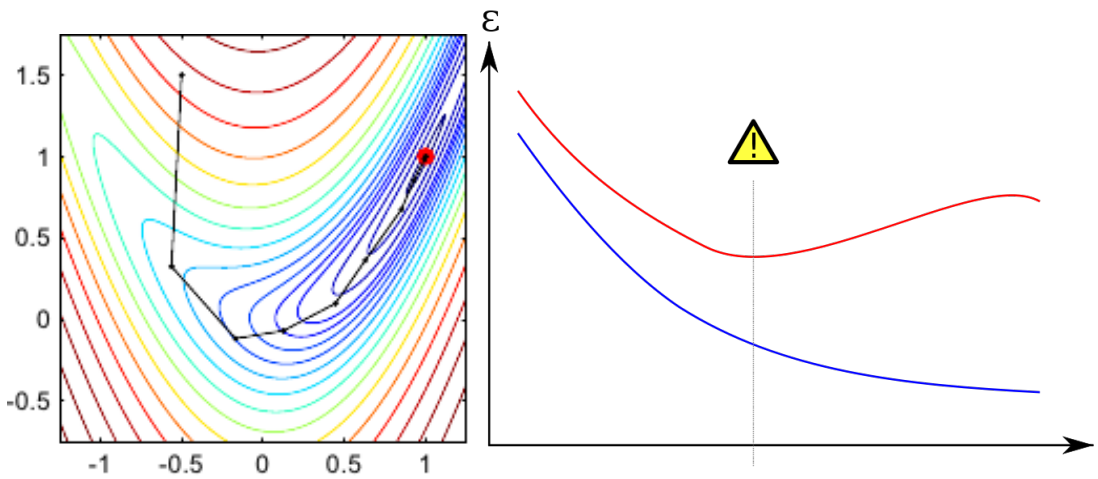
endfor

end foreach

Réseaux multicouches (6)

Remarques :

- On peut sommer les erreurs de plusieurs exemples et rétropropager une seule fois
  $\rightarrow$ mode online/Batch/minibatch
- Question du pas ... 2ème ordre ? Cf. cours gradient
- Quand stopper l'algorithme ? Attention au surapprentissage



Réseaux multicouches (7)

Dimensionnement

- Combien de neurones par couches / Combien de couches ?
- Une seul couche suffit pour estimer n'importe quelle fonction f, pourvu que : $J \to \infty$ et $N \to \infty$ [Lippman 87]

Solution : rajoutons des couches !

☺Backprop généralisable avec plusieurs couches cachées
☺Frontières de décision plus complexes
☺Représentation de haut niveau des données
☹Mais l'énergie de l'erreur est trop faible pour modifier les couches basses

$\to$ Deep learning !

Plan

MLP vs. SVM vs. Architectures profondes

1985 - 1995 : l'essor des réseaux de neurones

- Emergence de nombreuses applications industrielles :
- Reconnaissance d'écriture, de la parole, etc.

1995 - 2005 : La supprématie des Support Vector Machines

- Classifieurs aux bases théoriques fortes
- Excellentes capacités de généralisation, perf. à l'état de l'art
- Réseaux de neurones = has been . . .

2006 - 20 ? ? : Le retour des réseaux de neurones

Hinton, G. E., Osindero, S. and Teh, Y. A fast learning algorithm for deep belief nets. Neural Computation, 18, pp 1527-1554 (2006)

- Réseaux de neurones profonds
- Architectures connues, nouveaux algo d'apprentissage
- Performances permettant d'envisager de nouvelles applications

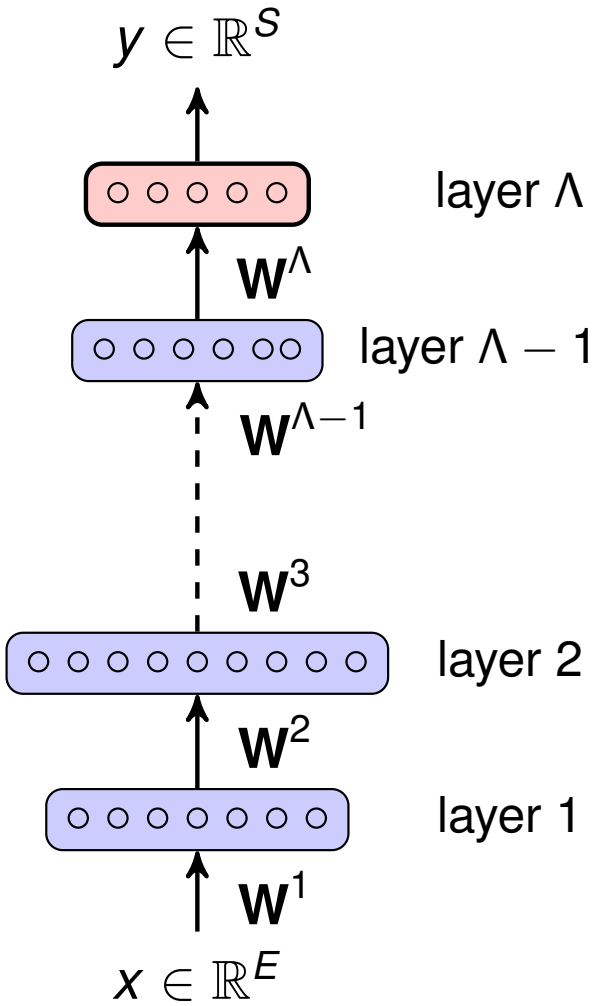
Architectures profondes (1)

Principe

- Réseau feedforward comportant Λ couches, avec $\Lambda > 2$
- W^λ matrice des poids entre couches $\lambda - 1$ et λ
- backprop insuffisante $\rightarrow$ comment faire ?

Apprentissage en deux temps

- Apprentissage des couches dites basses, en non supervisé
  - Utilisation des autoencodeurs
  - Couches dites de modèles
- Apprentissage des dernières couches en supervisé
  - Backpropagation
  - Couches dites de decision

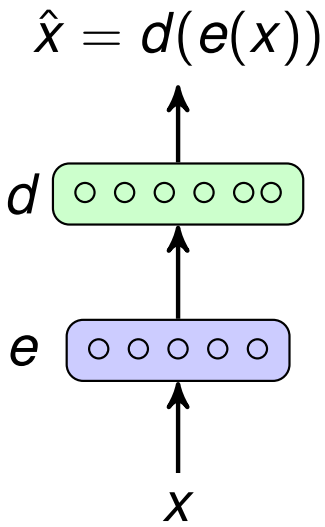


Architectures profondes (2)

Auto Associateurs (AA)

- Un AA cherche à apprendre ses propres entrées : on veut $y^d = x$
- Apprentissage d'un encodeur $e(x)$ et d'un décodeur $d((e(x))$
- Réseau à une couche cachée e et une couche de sortie d
- Critère : $\mathcal{J} = (\hat{x} - x)^2 = (d(e(x)) - x)^2$

Si le nombre de neurones de e est $< E$:
Compression, représentation parcimonieuse de x



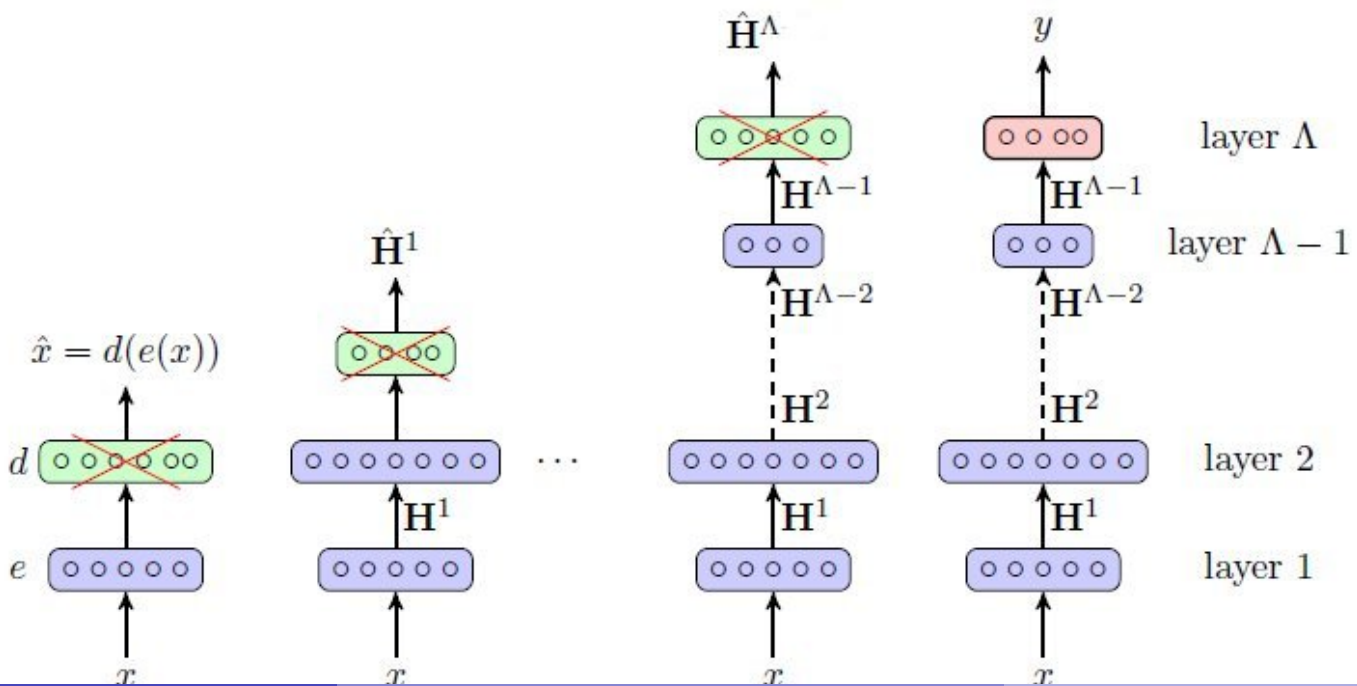
Architectures profondes (3) : Apprentissage

« pre-training »

- Apprendre un AA sur x.
- Garder $e^1(x) = H^1$, jeter $d^1(e^1(x))$
- Apprendre un nouvel AA sur $e^1(x)$
- Garder $e^2(e^1(x)) = H^2$, jeter d^2
- etc.

« fine-tuning »

- déverrouiller tous les H^λ
- rajouter une ou plusieurs couches
- backpropagation sur l'ensemble du réseau



Plan

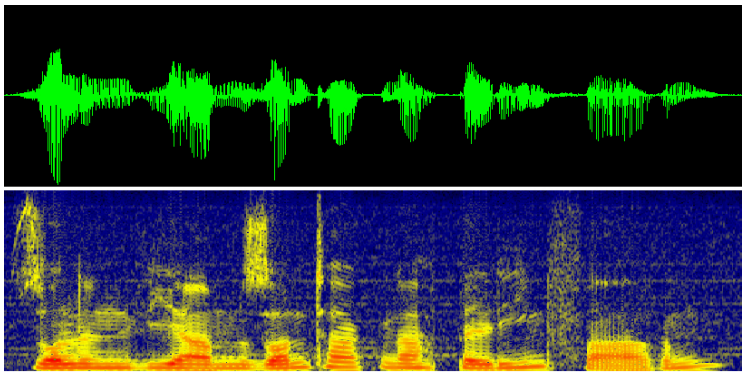
Réseaux de neurones et séquences

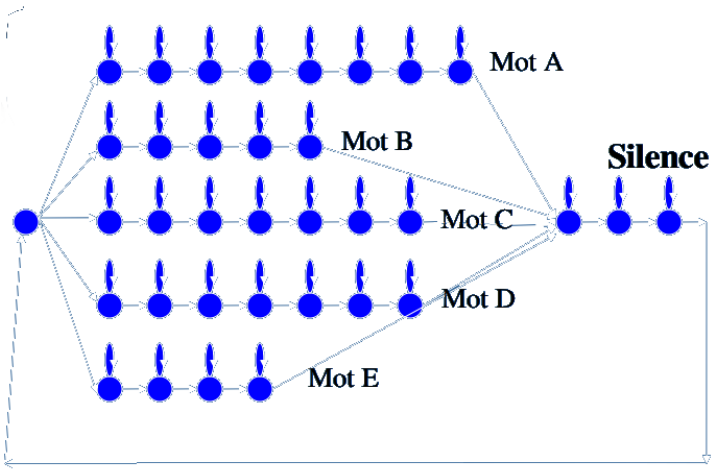
Comment traiter des **Séquences** avec des réseaux de neurones ?

- Parole, écriture, cours de la bourse, image (2D), etc.
- Signaux de taille **variable** $\rightarrow$ nécessité de classifieurs **dynamiques**

1ère solution : réseaux de neurones / classifieur dynamique

- Réseaux de neurones / Hidden Markov Model
- = classification locale / modélisation de séquence

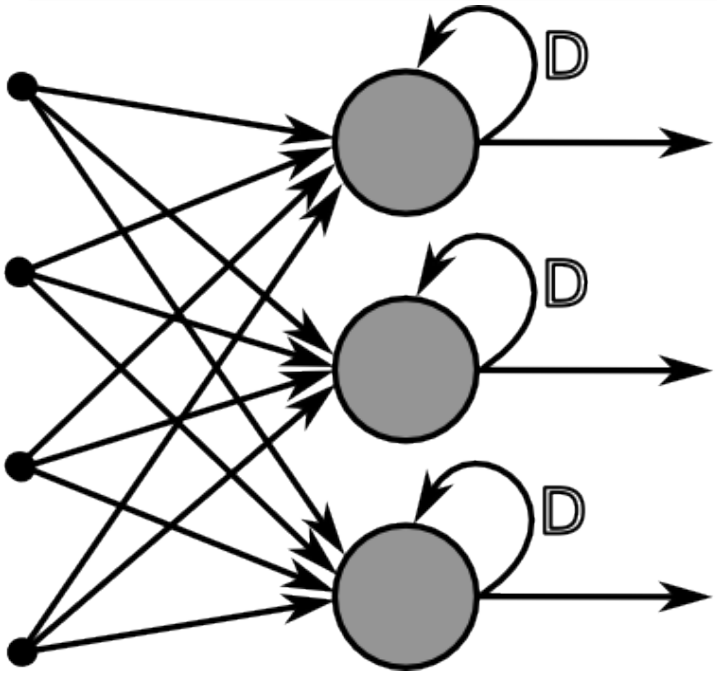




2ème solution : Réseaux récurrents (1)

Connexions récurrentes

- Permet de prendre en compte le contexte
- On calcule $y(n)$ à partir de $x(n)$ et $y(n-1)$ les sorties de l'observation précédente



Question :

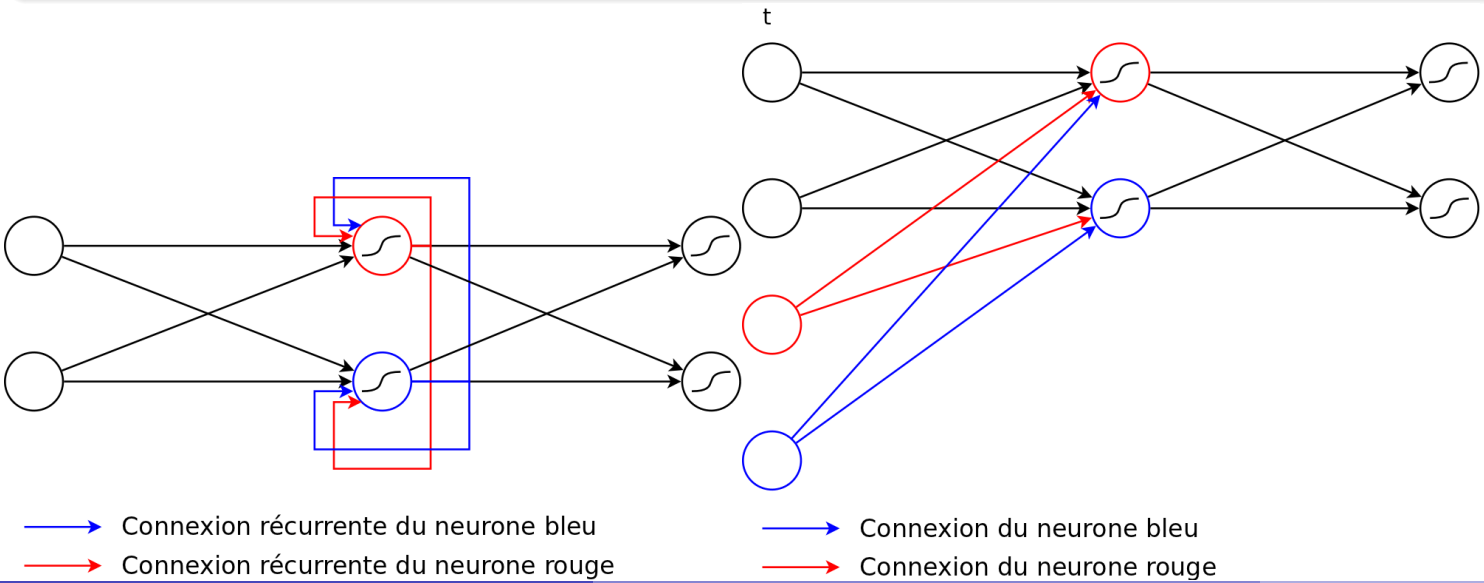
Comment apprendre les poids des connexions récurrentes ?

1. BackProp Through Time (BPTT)
2. Real Time Recurrent Learning (RTRL)

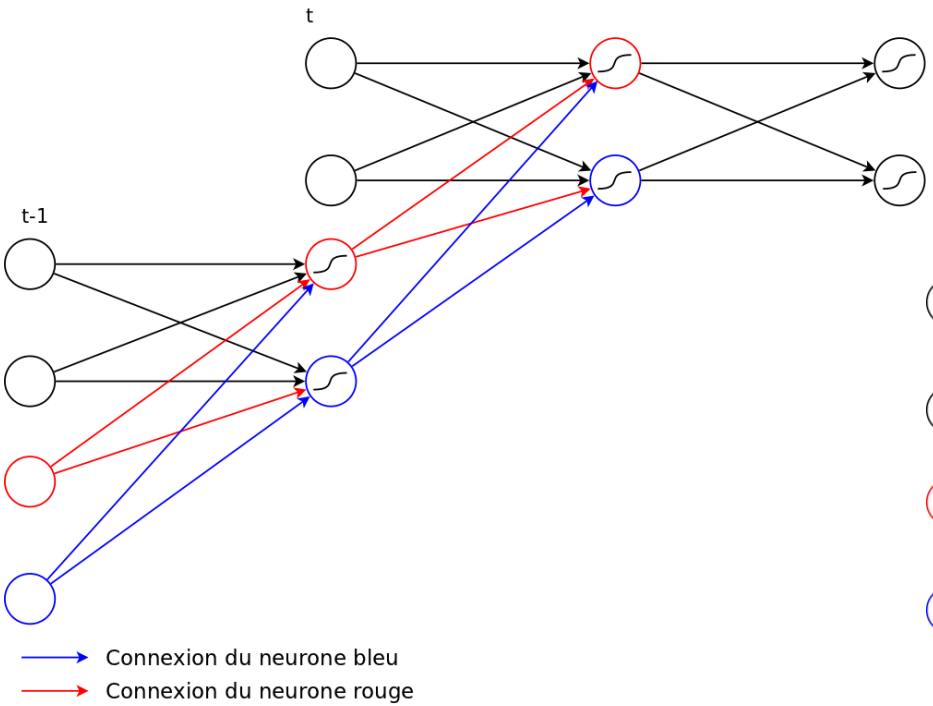
Réseaux récurrents (2)

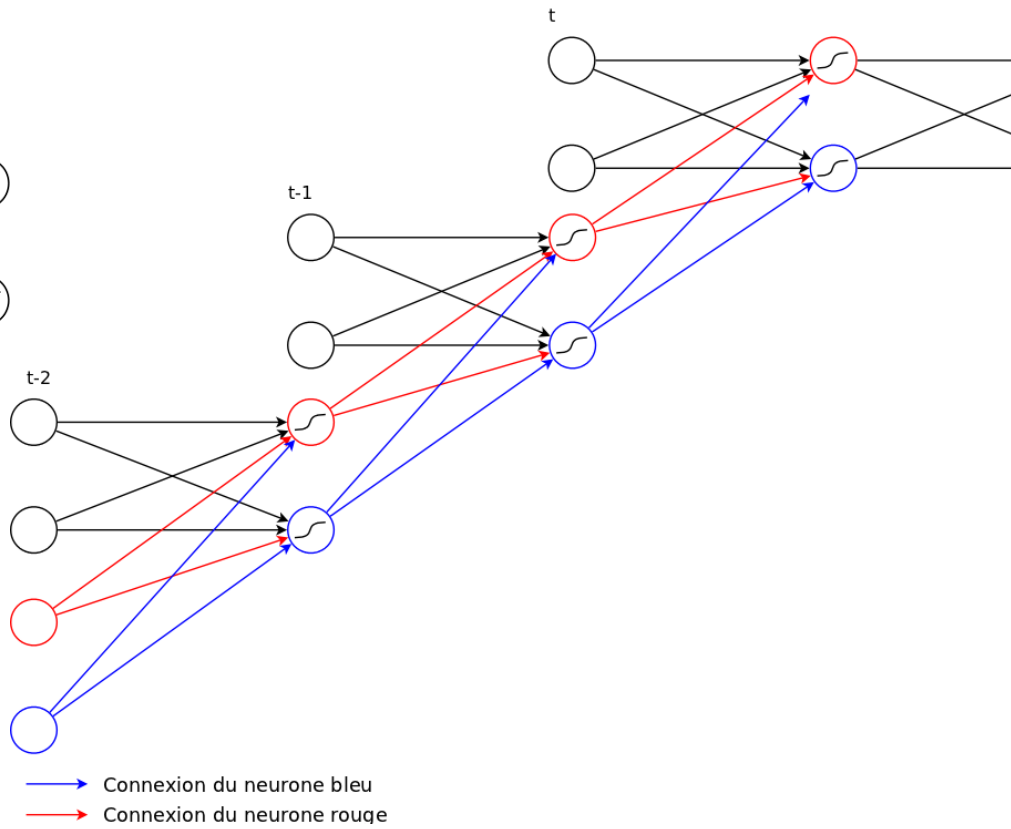
Backpropagation Through Time (BPTT)

- Idée : déplier le réseau R pour l'approximer par un réseau non récurrent R^*
- Structure temporelle $\rightarrow$ structure spatiale sur k pas
- Les poids des connexions récurrentes sont copiés et attribués à des connexions non récurrentes de R^*.
- Les copies des connexions possèdent toutes la même pondération.
- Les premiers neurones rouges et bleus sont initialisés au hasard



Réseaux récurrents (3)





Backpropagation Through Time (BPTT), suite

- Une fois déplié, on applique une backprop classique
- $\rightarrow$ k limité = contexte limité ...

Réseaux récurrents (4)

Real Time Recurrent Learning (RTRL) [Williams 1989]

La sortie du neurone j recoit :

- tous les $x(t)$ de la couche précédente pondérés par w_{je}
- tous les $y(t-1)$ de sa couche pondérés par des $w_{jj'}$

$$y_j(t) = \varphi\left(\sum_{e=0}^{E} w_{je} x_e(t) + \sum_{j'=0}^{J} w_{jj'} y_{j'}(t-1)\right)$$

RTRL : apprentissage

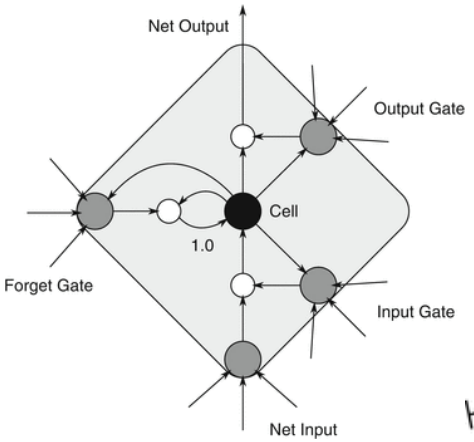
- Critère $\mathcal{J} = (y_j^d - y_j)^2$
- Calcul des $\frac{\partial \mathcal{J}}{\partial w_{sj}}$, $\frac{\partial \mathcal{J}}{\partial w_{je}}$ et $\frac{\partial \mathcal{J}}{\partial w_{jj'}}$ pour appliquer le gradient « classique »
- deuxième ordre possible
- Complexité importante $\mathcal{O}(N^4)$

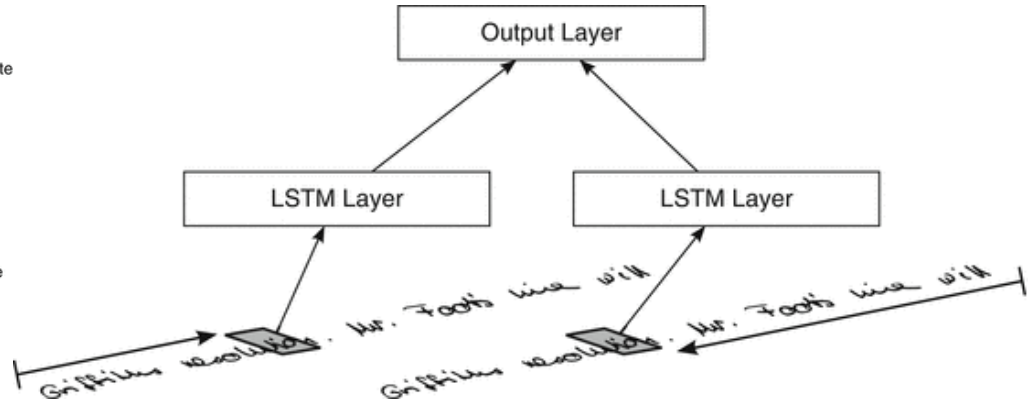
BLSTM

Bidirectionnal Long Short Term Memory

A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. NIPS, 2009

- Modélisation des dépendences à court/long terme
- Neurone formel avec mémoire (cell) + gates
- Apprentissage par BPTT
- Ex. de Perf HWR : 83% $\rightarrow$ 91% WER





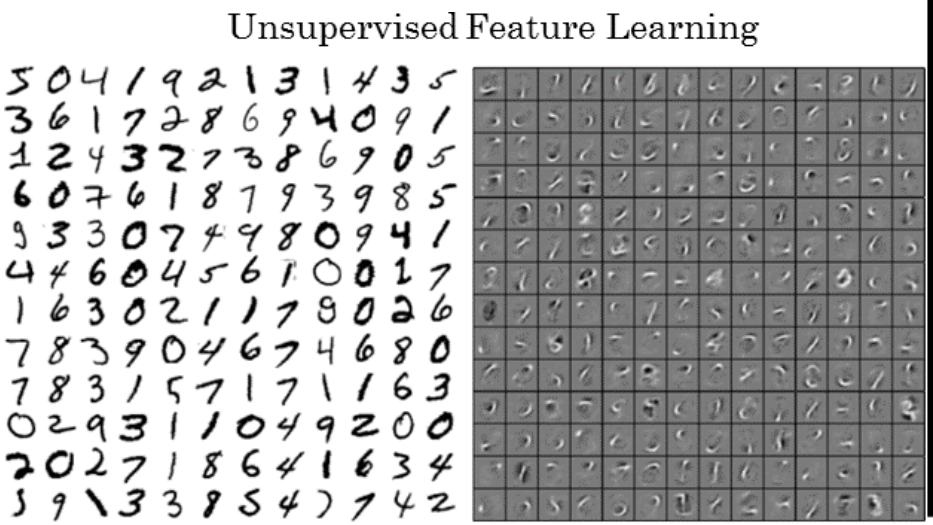
Et aussi : Utilisation en génération. Démo :
`http://www.cs.toronto.edu/~graves/handwriting.html`

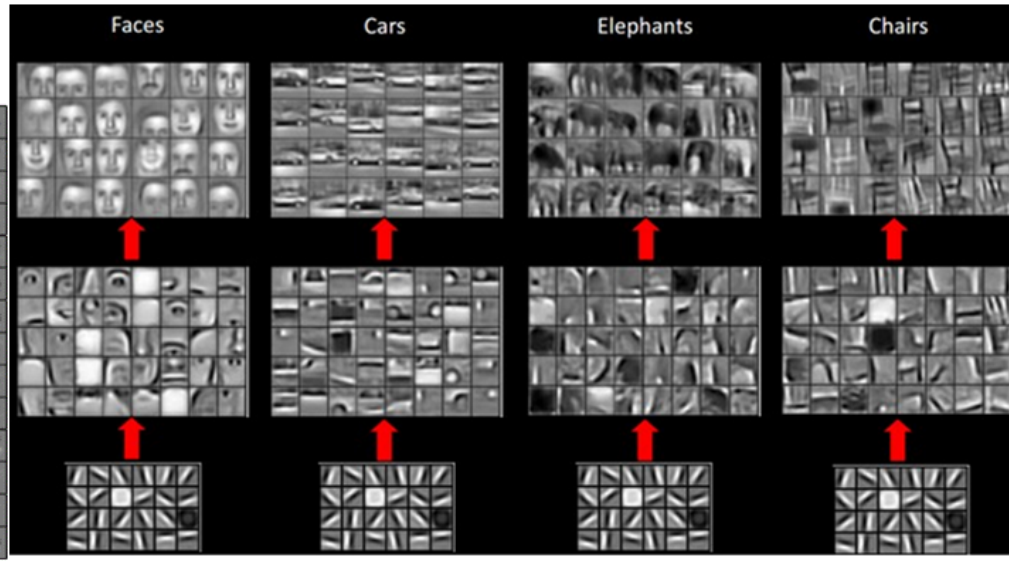
Plan

Convolutional neural network (1)

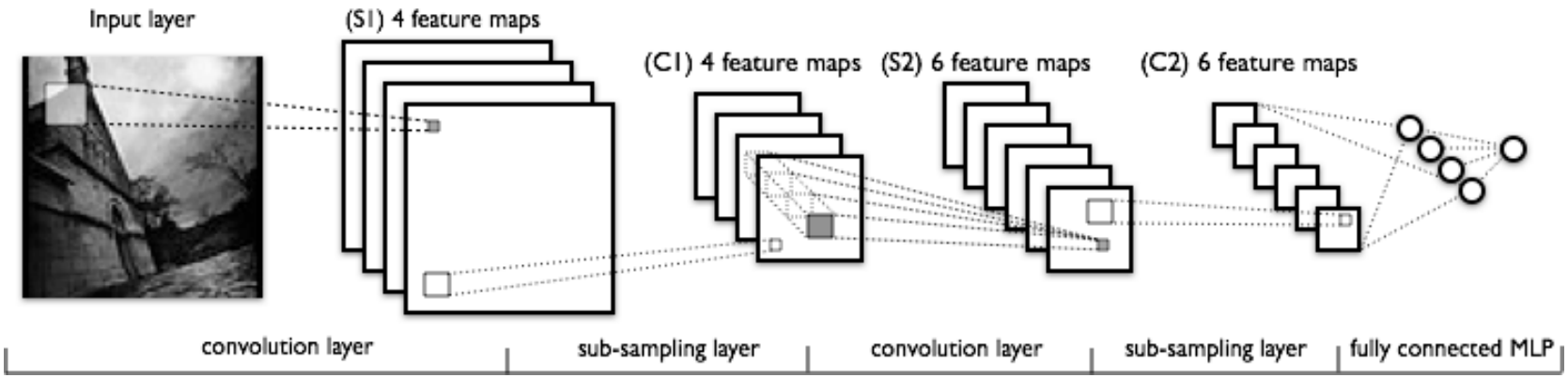
Réseaux de neurones convolutionnels

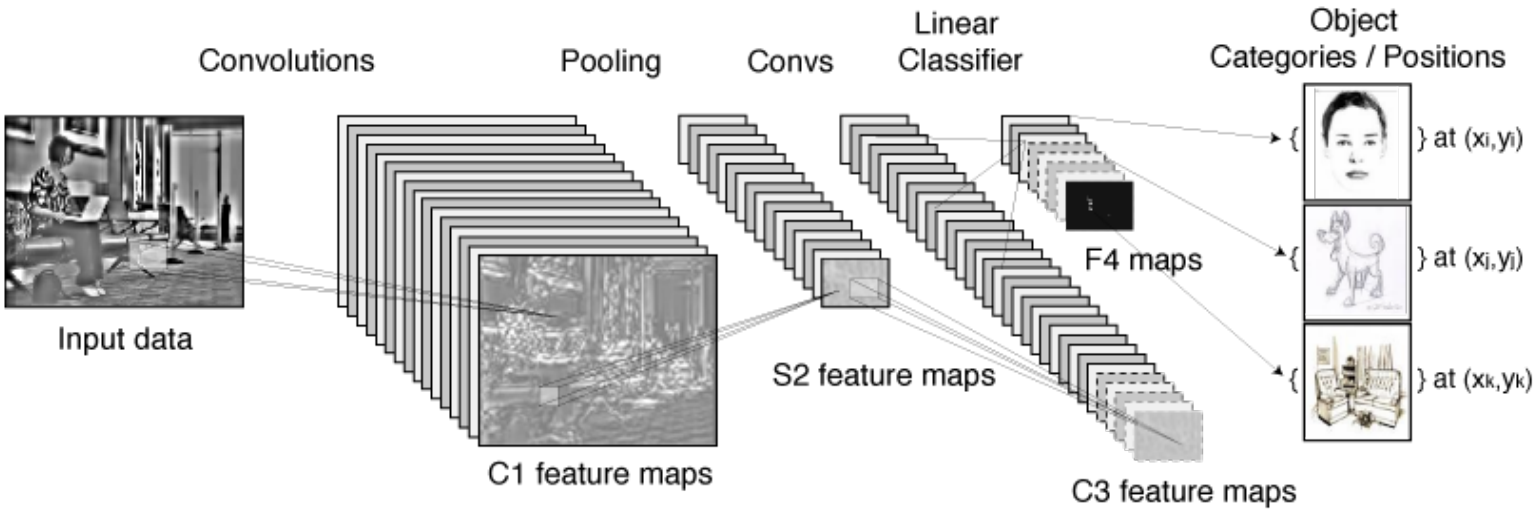
- Destiné à traiter les images
- Mécanisme de poids partagés $\rightarrow$ moins de paramètres, meilleure généralisation
- Apprentissage de filtres par backprop classique





Convolutional neural network (2)



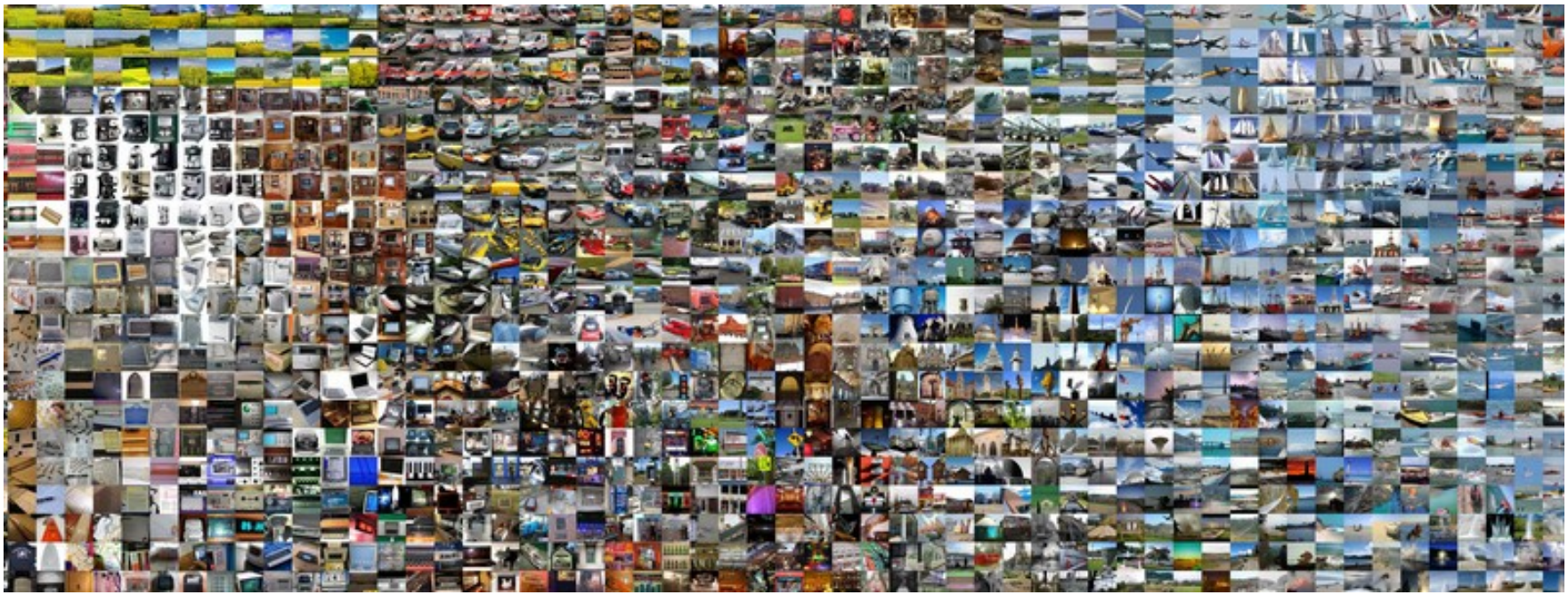


Plan

La compétition ImageNet

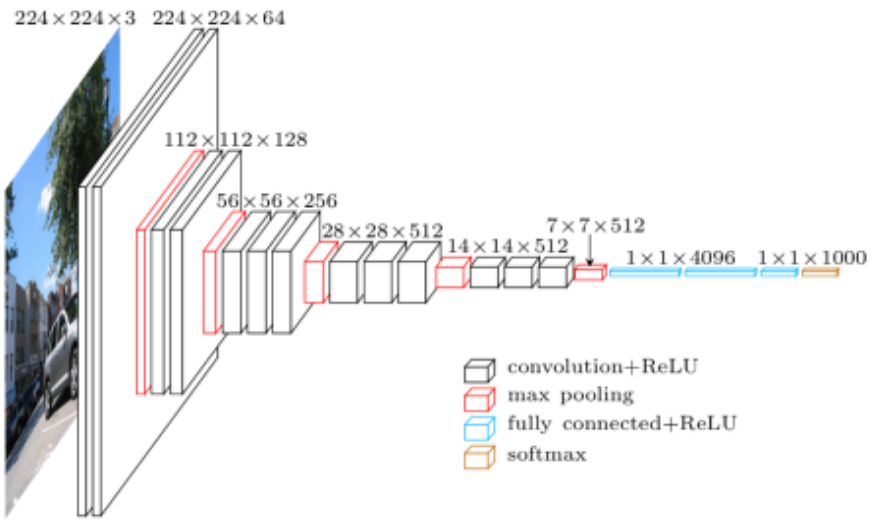
ImageNet

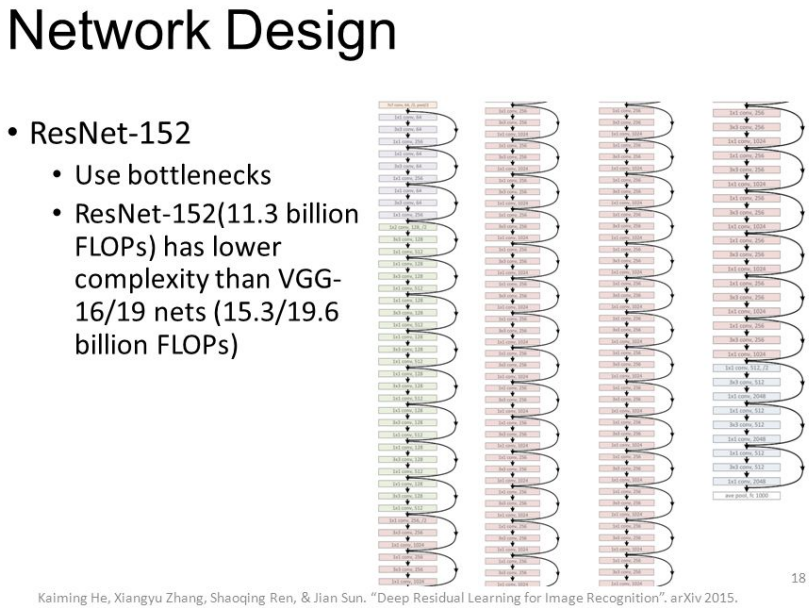
- $>$ 14M d'images, 1000 classes (objets, animaux, scènes, etc.)
- Images couleur $512 * 512$



Les architectures pour traiter ImageNet

VGG16, VGG19, AlexNet, GoogleNet, ResNet ($L > 150$!), etc.



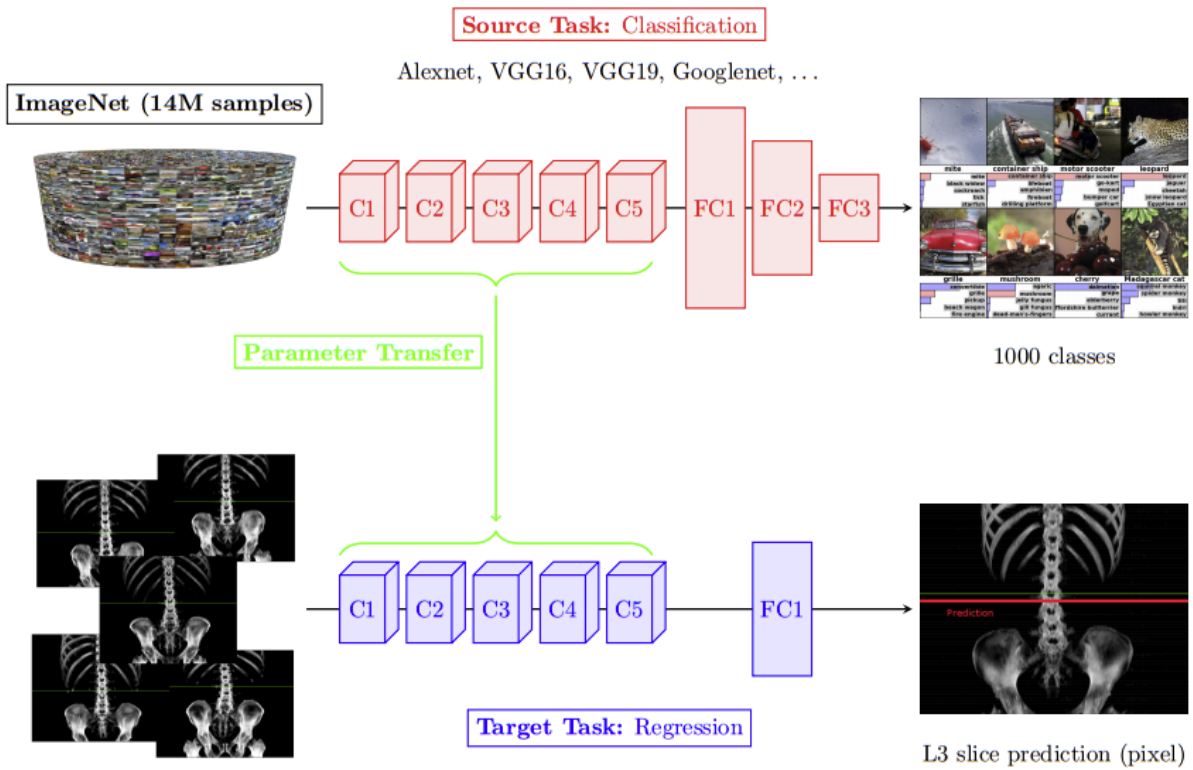


performance : de l'ordre de qq % d'erreur en 2015 (3.52% pour resnet)

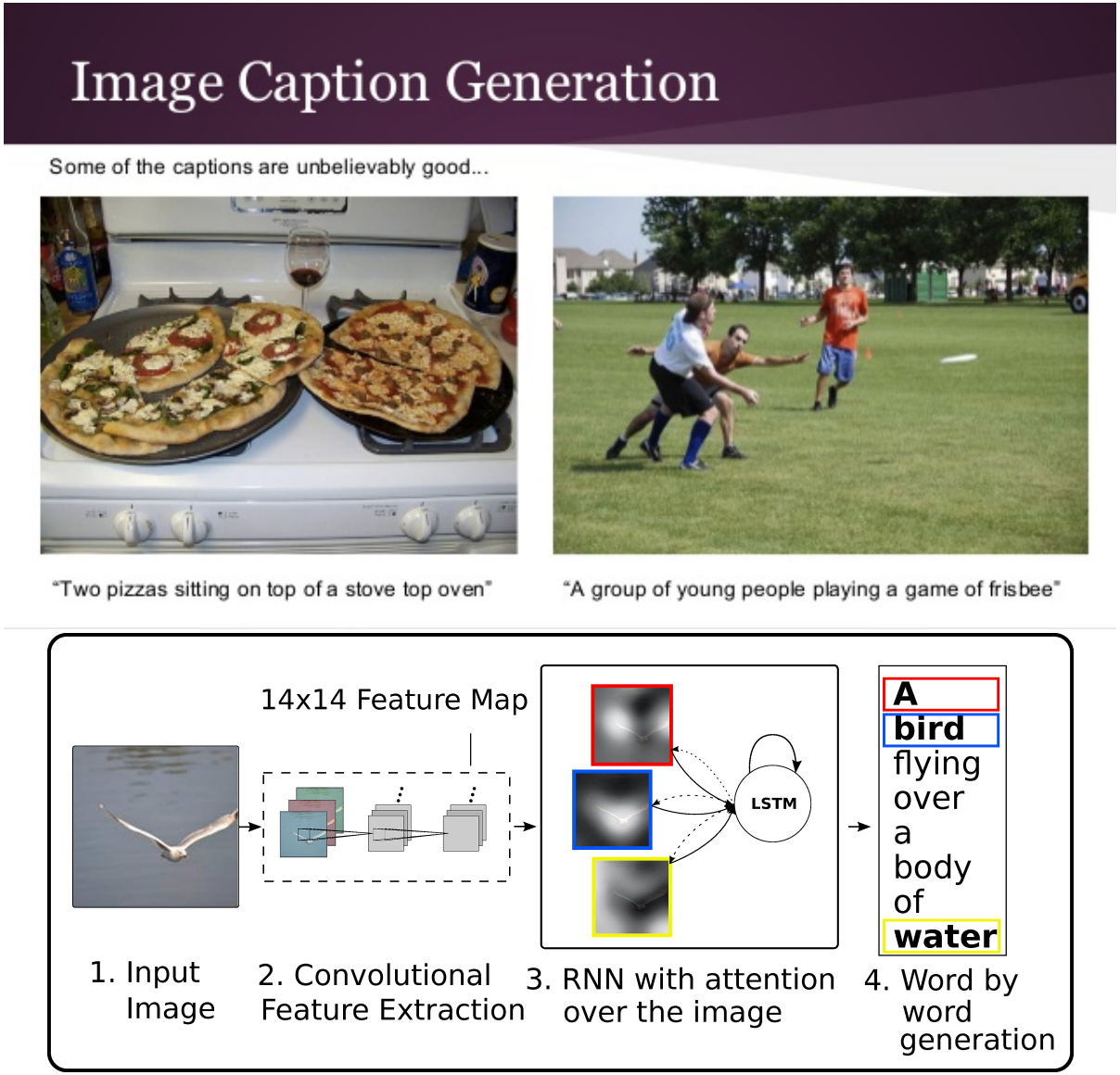
Transfer learning

Comment faire quand on a peu de données ?

- Utiliser un réseau pré-appris (AlexNet, VGG16, etc.) sur une très grosse base (ImageNet)
- Fit des données et des couches de sorties
- Réapprentissage sur le nouveau jeu de données



Caption Generation

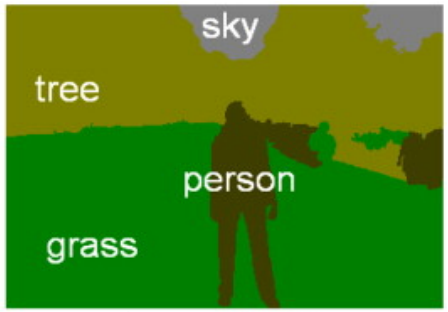


Semantic Segmentation (1)

Étiquetage des pixels d'une image

- Nombreuses applications en CV : automobile, médical, etc.
- Problème à sorties structurées (ciel est souvent au dessus de l'herbe)
- Tâche difficile car dimension des entrées et des sorties importantes

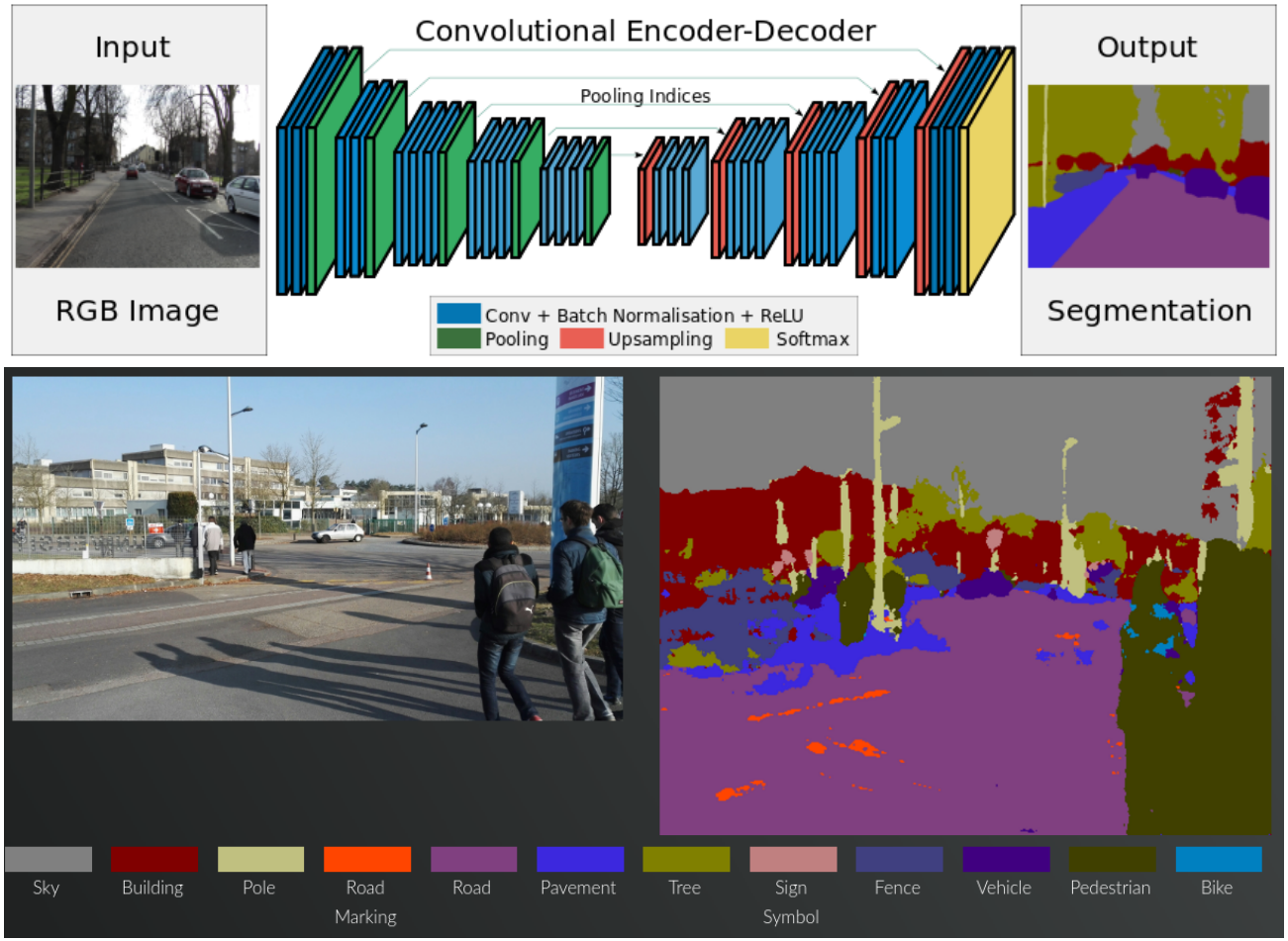
Input



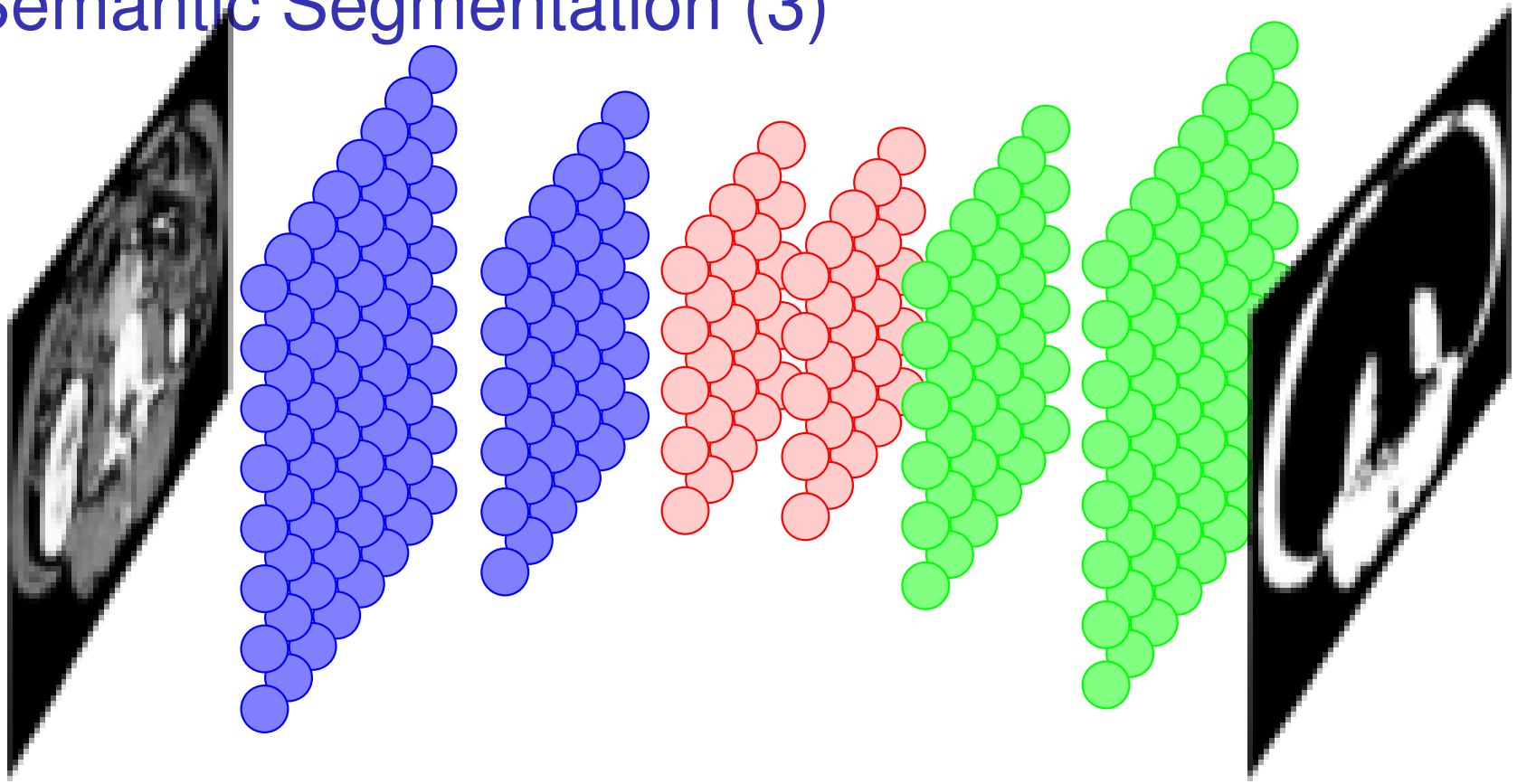
Segmentation [9]

Semantic Segmentation (2)

SegNet (démo sur `http://mi.eng.cam.ac.uk/projects/segnet/`), detectNet, etc.



Semantic Segmentation (3)



Input/Output Deep Architecture [4, 5]

1. Préapprentissage des entrées = représentation des données
2. Préapprentissage des sorties = app. des connaissances a priori
3. Fine tuning = apprentissage classique du lien entre entrées et sorties

Plan

Avantages et Inconvénients

Avantages

- Un RdN approxime des probabilités à posteriori $p(C_i/x)$
- Très rapide en décision
- Supporte très bien les grandes dimensions ($E >$ qq centaines)
- Performances : architectures profondes à l'état de l'art sur plusieurs problèmes

Inconvénients

- Paramétrisation
- Apprentissage long et parfois difficile à contrôler (minimum locaux)
- Nécessite bcp de données

Paramétrisation/choix du réseau

Nombre de couches

- Problème "simple", carac disponibles : MLP avec 1 ou 2 couches
- Sinon : Architecture profonde : Pas de caractéristiques à extraire ☺, mais plus d'hyperparamètres ☹
- Si image/vidéo : CNN / LSTM
- Si séquence (texte, signal, etc.) : LSTM

Nombre de neurones couches cachées

- Classique : moyenne géométrique ou arithmétique de (E,S)
- Avec des deep : + difficile, première couche + grande que E

Fonction d'activation φ

Old school : *tanh*, sigmoide ; new : ReLU (Rectified Linear Unit)

Paramétrisation/choix du réseau

Réglage du pas (voir cours Gradient)

- pas fixe : petit (10^{-3}, 10^{-4}, 10^{-5}, ...)
- pas adaptatif : diminue avec les itérations
- line search : calcul du pas "idéal" à chaque itération
- Méthode du deuxième ordre (gradient conjugué), + de calculs

Les données

- Centrées réduites : c'est mieux
- Taille de la base d'app : the more, the better ($> E^2$ / classe)
- Attention aux bases non balancées
- Mélanger les données

Choix du critère

- Classification : plutôt cross entropy
- Regression : plutôt MSE

Les données

DATA DATA DATA DATA DATA DATA DATA DATA
The more, the better !

- Centrées réduites : c'est mieux
- Attention aux bases non balancées
- Mélanger les données
- Online/batch/mini batch
- App/Valid/Test

Datasets publics

- Assez rares
- Souvent différents de notre problème
- Mais peuvent être utilisés à travers le transfer learning
- ImageNet, Rimes, MNIST, STREET dataset (numéros google), etc.

Mise en œuvre, pointeurs

∃ de nombreuses librairies

La plupart sont basées sur Theano (python, Montreal) [a] et TensorFlow (Google) [b]

- Keras (python) `https://keras.io/`
- Torch7 (lua) (NEC) `http://torch.ch/`
- pybrain (python, TUM Munich) `http://pybrain.org/`
- Caffe (Berkeley) `http://caffe.berkeleyvision.org/`

a. `http://deeplearning.net/software/theano/`
b. `https://www.tensorflow.org`

Pointeurs intéressants :

- Les cours de Yann Lecun au collège de France `https://www.college-de-france.fr/site/yann-lecun/`
- Chaine Youtube de H. Larochelle : `http://tinyurl.com/lpkvjm4`

Exercices

Exercice A : MLP from scratch

- Coder en matlab/octave un MLP à 1 couche cachée, sans librairie
- Tester les hyperparamètres : η ; nb d'itération, nb de neurones, etc.
- Base ? MNIST

Exercice B : Utilisation de tensorFlow + Keras

- Tester les CNN pour une tâche de classif
- Tester le transfer learning
- Tester les RNN
- ...

Exercice C : Utilisation sur un pb perso

- Prédiction résultats sportifs, de température
- Apprendre au réseau à parler
- ...

Bibliographie

- F. Rosenblatt. Principles of Neurodynamics. New York : Spartan, 1962.
- C.M. Bishop. Neural networks for pattern recognition, Oxford : Oxford University Press, 1995.
- D.E. Rumelhart, G.E. Hinton and R.J. Williams. Learning internal representations by error propagation. Parallel Distributed Processing Explorations in the Microstructure of Cognition. MIT Press, Bradford Books, vol. 1, pp. 318-362, 1986.
- J. Lerouge, R. Herault, C. Chatelain, F. Jardin, and R. Modzelewski, ”Ioda : an input output deep architecture for image labeling”, Pattern recognition, vol. 48, iss. 9, p. 2847-2858, 2015.
- Soufiane Belharbi, Clément Chatelain, Romain Hérault, Sébastien Adam : Input/Output Deep Architecture for Structured Output Problems. CoRR abs/1504.07550 (2015)