Réseau de neurones

Cours 7
ARF Master DAC

Nicolas Baskiotis
nicolas.baskiotis@lip6.fr
http://webia.lip6.fr/~baskiotisn

équipe MLIA, Laboratoire d'Informatique de Paris 6 (LIP6)
Sorbonne Université - Université Pierre et Marie Curie (UPMC)

S2 (2017-2018)

Résumé des épisodes

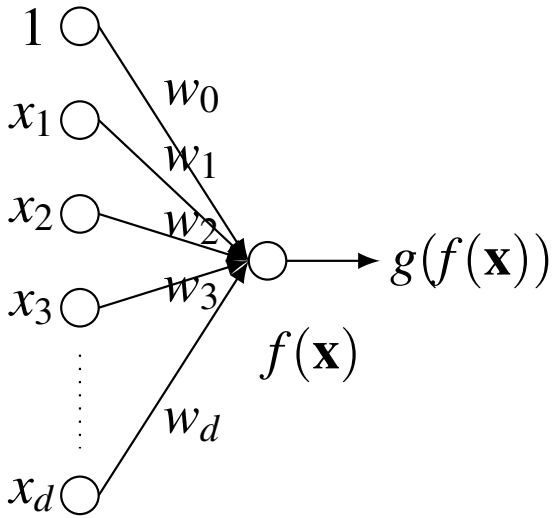
Problèmatique de l'apprentissage supervisé

- Ensemble d'apprentissage $\{(\mathbf{x}^i, y^i)\} \in X \times Y$, ensemble de fonctions $\mathcal{F}$
- un coût $L(\hat{y}, y) : Y \times Y \to \mathbb{R}^+$, trouver $f^* = argmin_{f \in F} \sum_i L(f(\mathbf{x}^i), y^i)$

Perceptron

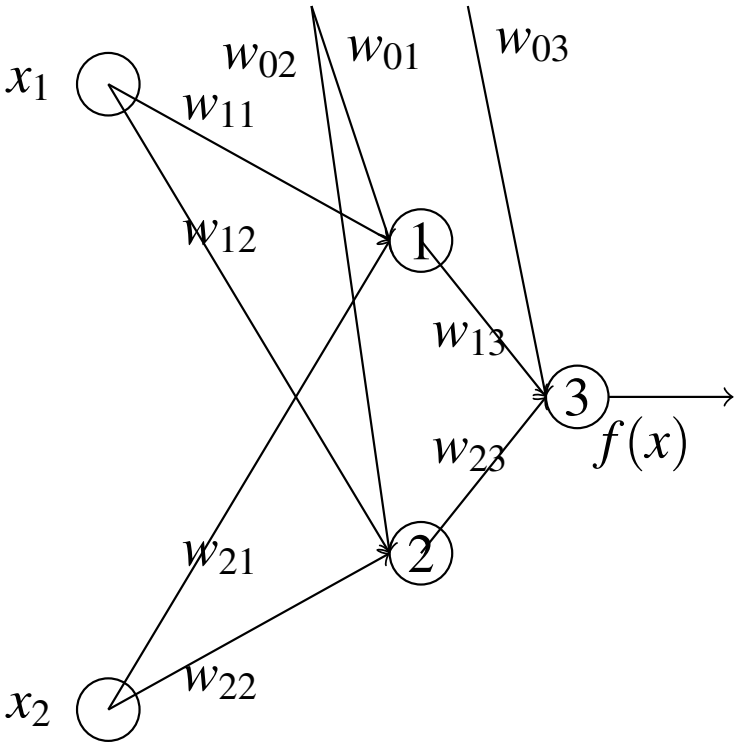
- Hypothèse linéaire : $f_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i$
- Coût perceptron : $L(f_{\mathbf{w}}(\mathbf{x}), y) = \max(0, -f_{\mathbf{w}}(\mathbf{x})y)$
- Gradient :

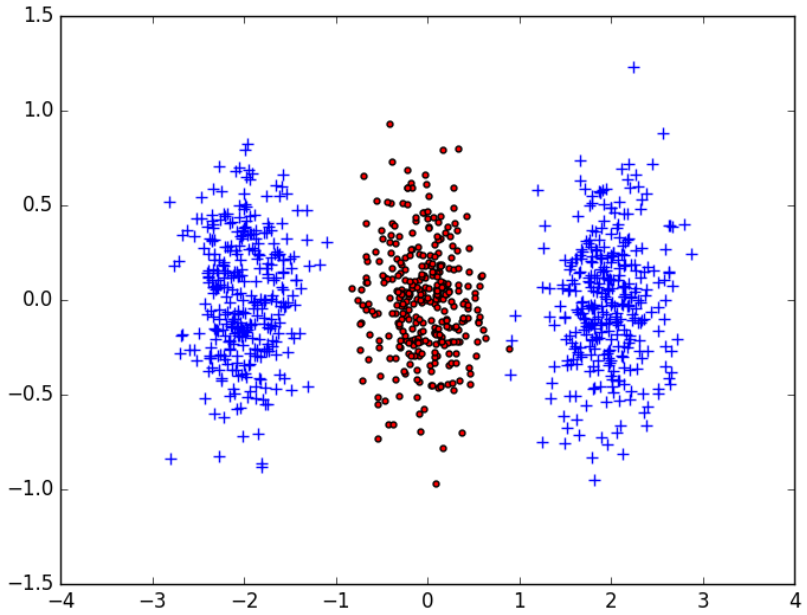
$$\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} 0 & \text{si } (-y < \mathbf{w}.\mathbf{x} >) < 0 \\ -y\mathbf{x} & \text{sinon} \end{cases}$$



SVM

Deux neurones

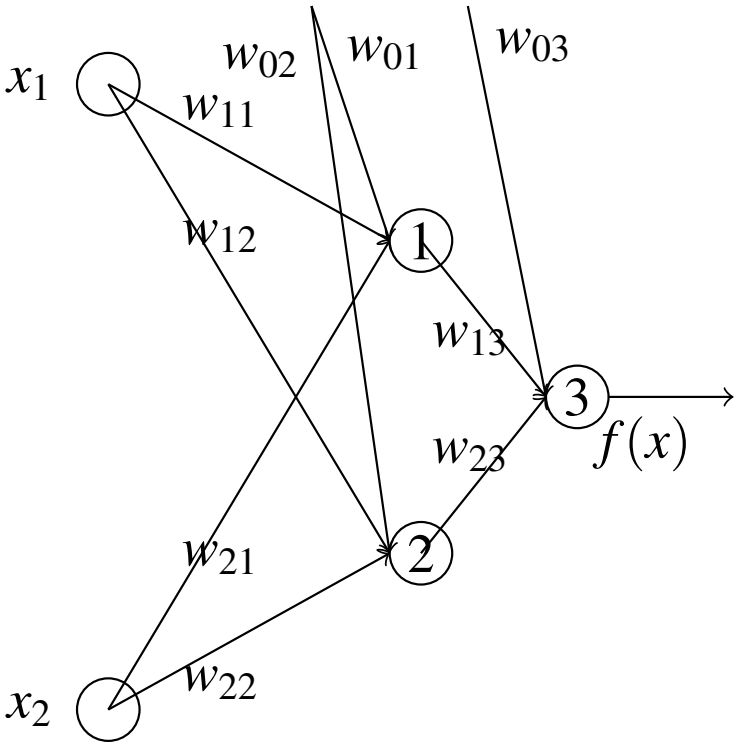


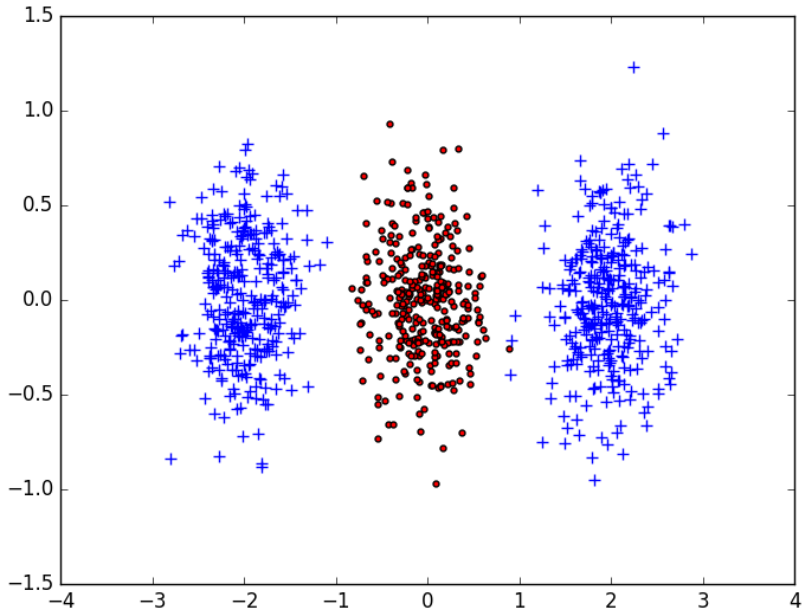


- Combiner des neurones → augmente l'expressivité
- Création de dimensions nouvelles, de nouveaux features

⇒ Est-il facile d'apprendre ses réseaux ?

Deux neurones

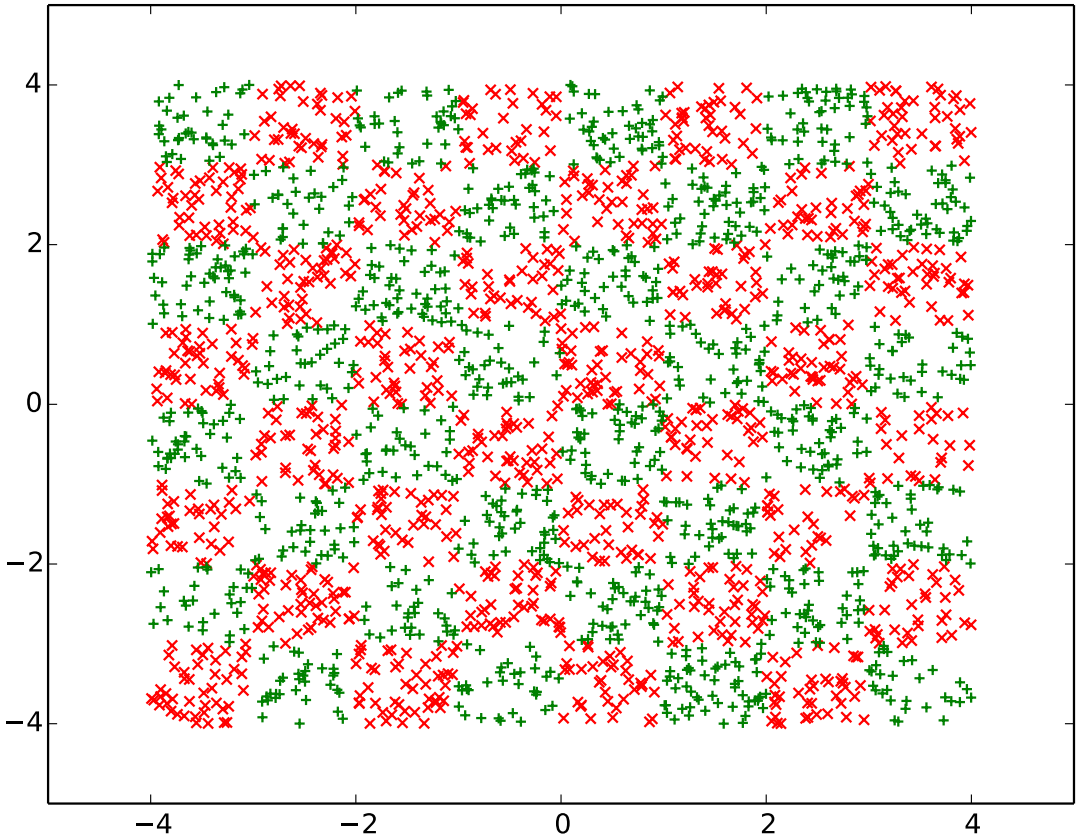




- Combiner des neurones $\rightarrow$ augmente l'expressivité
- Création de dimensions nouvelles, de nouveaux features

$\Rightarrow$ Est-il facile d'apprendre ses réseaux ?

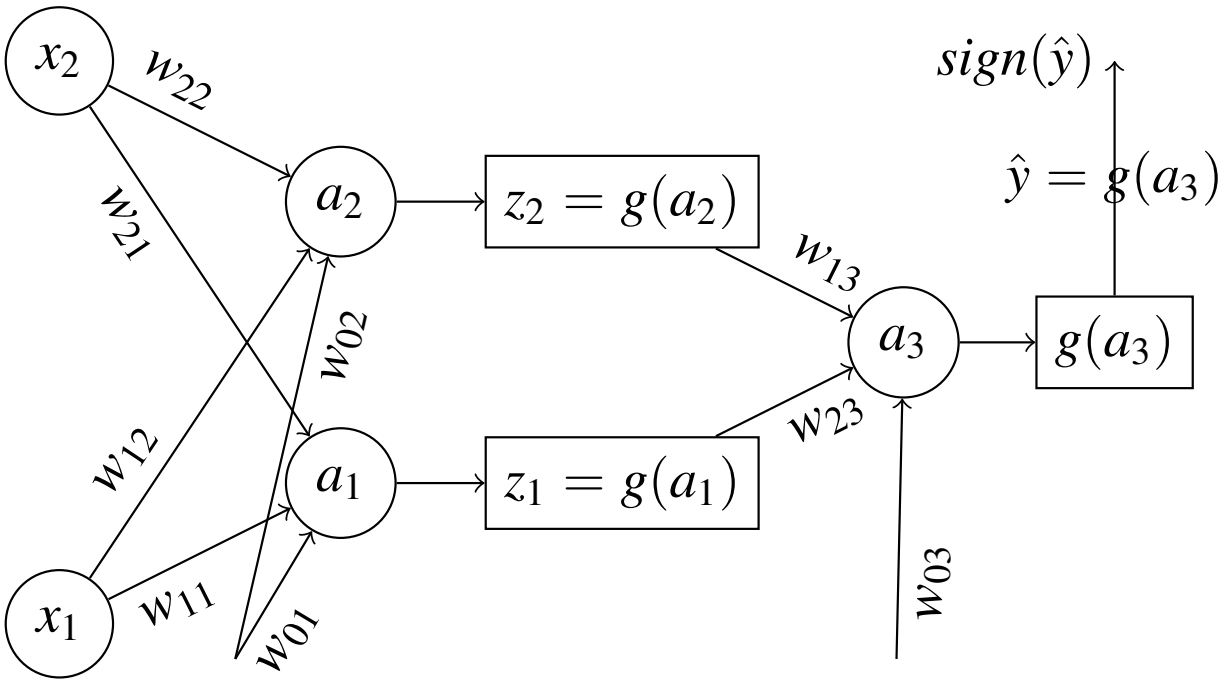
Et pour ce problème ?



Si vous aviez droit à n'importe quelle fonction dans un neurone ?

Plan

Pour l'inférence



Inférence

Avec $g(x) = tanh(x)$

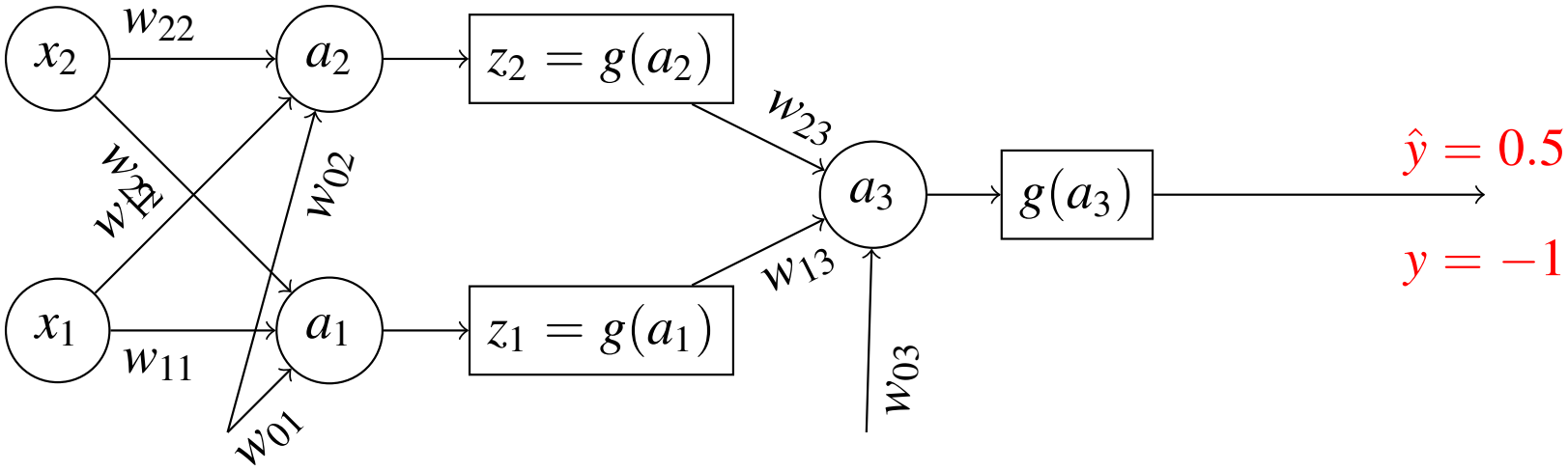
- $a_1 = w_{01} + w_{11}x_1 + w_{21}x_2$
- $a_2 = w_{02} + w_{12}x_1 + w_{22}x_2$
- $z_1 = g(a_1)$
- $z_2 = g(a_2)$
- $a_3 = w_{03} + w_{13}z_1 + w_{23}z_2$
- $\hat{y} = g(a_3)$

$\Rightarrow$ prédiction : $sign(\hat{y})$

Vocabulaire

- g fonction d'activation (non linéarité du réseau)
- a_i activation du neurone i
- z_i sortie du neurone i (transformé non linéaire de l'activation).

Pour l'apprentissage



Objectif : apprendre les poids

- Choix d'un coût : moindres carrés
  $L(\hat{y}, y) = (\hat{y} - y)^2$ (pourquoi est ce un bon choix ?)

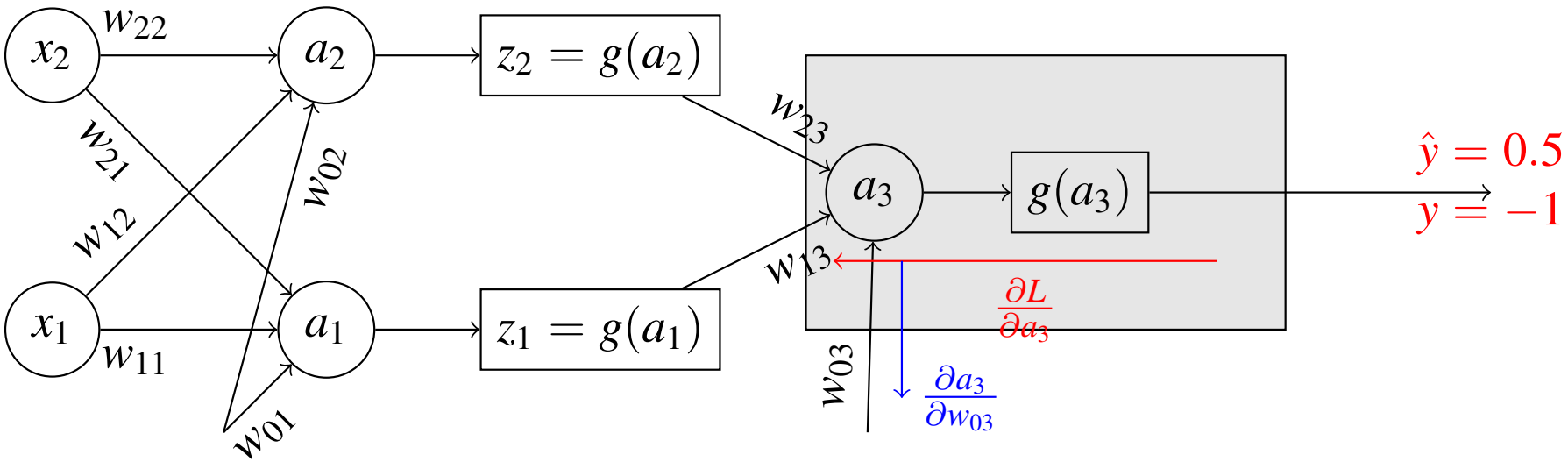
$\Rightarrow$ descente de gradient

- Mais à quel(s) neurone(s) et comment répartir l'erreur entre les poids ?

$\Rightarrow$ Rétro-propagation de l'erreur :

  - corriger un peu tous les poids . . .
  - en estimant la part de chacun dans l'erreur

Pour l'apprentissage



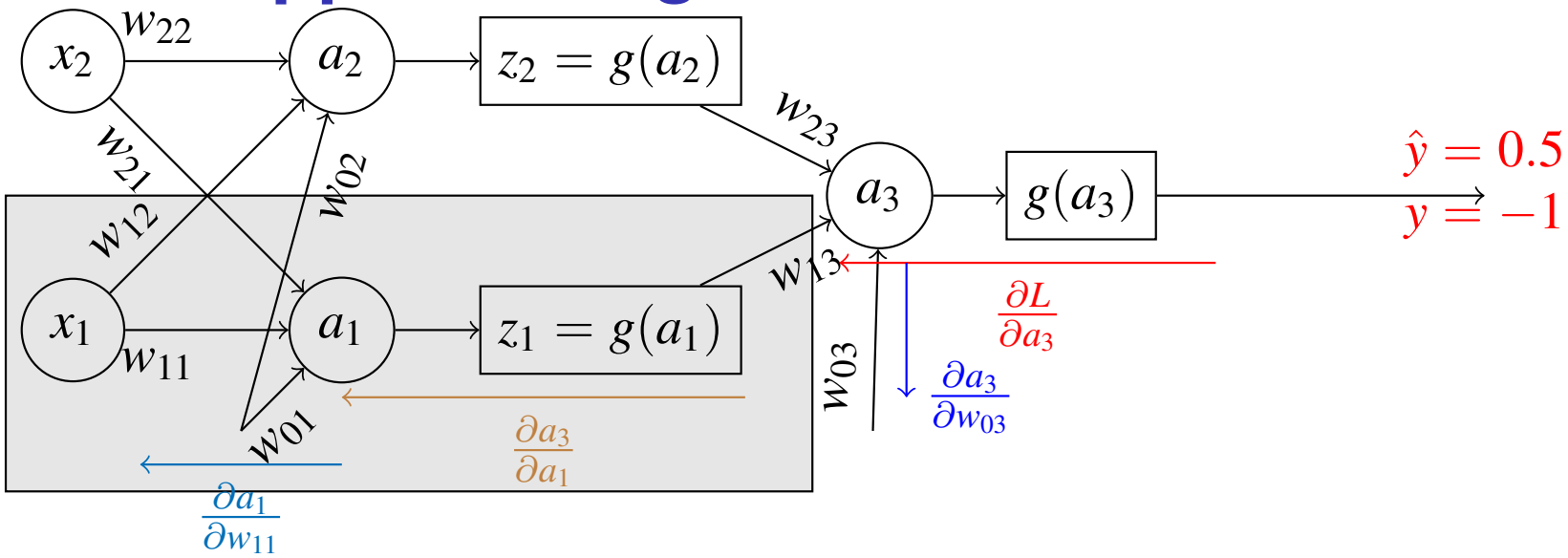
Pour les poids de la dernière couche : gradient de chaque élément

$$\frac{\partial L}{\partial w_{i3}} = \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial w_{i3}} \quad \text{avec} \quad \left| \begin{array}{lll} \frac{\partial L}{\partial a_3} & = \frac{\partial L}{\partial g(a_3)}\frac{\partial g(a_3)}{\partial a_3} = \frac{\partial (g(a_3)-y)^2}{\partial a_3} & = 2g'(a_3)(g(a_3)-y) \\ \frac{\partial a_3}{\partial w_{i3}} & = \frac{\partial w_{03}+w_{13}z_1+w_{23}z_2}{\partial w_{i3}} & = z_i \end{array} \right.$$

Soit

$$\frac{\partial L}{\partial w_{i3}} = 2g'(a_3)(\hat{y}-y)z_i$$

Pour l'apprentissage



Pour les poids de la première couche: w_{i1} par exemple

$$\frac{\partial L}{\partial w_{i1}} = \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial w_{i1}} \quad \text{avec} \quad \left| \begin{array}{lll} \frac{\partial L}{\partial a_1} & = \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial a_1} & = \frac{\partial L}{\partial a_3} g'(a_1) w_{13} \\ \frac{\partial a_1}{\partial w_{i1}} & = \frac{\partial w_{01} + w_{11}x_1 + w_{21}x_2}{\partial w_{i1}} & = x_i \end{array} \right.$$

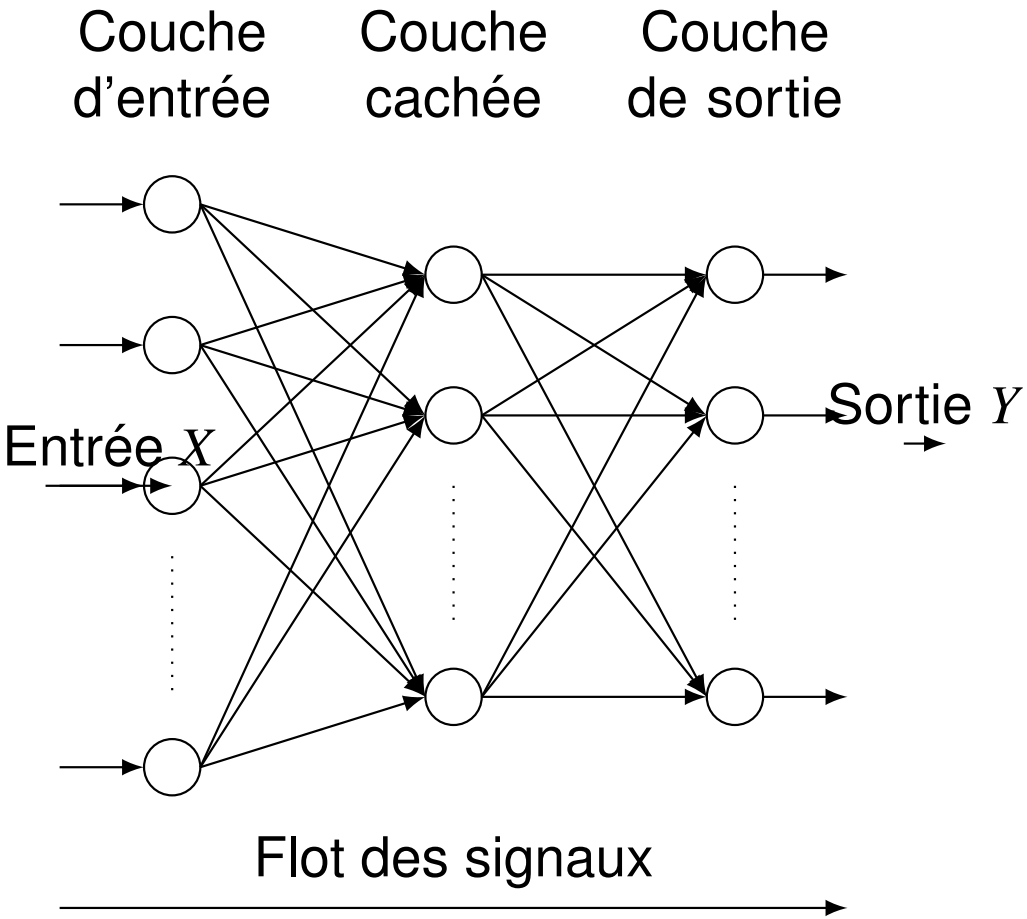
Soit

$$\underbrace{\frac{\partial L}{\partial w_{i1}}}_{\text{correction de } w_{i1}} = \frac{\partial L}{\partial a_1} x_i = \underbrace{\frac{\partial L}{\partial a_3}}_{\text{erreur à propager}} \underbrace{g'(a_1) w_{13}}_{\text{poids de la connexion}} \quad x_i$$

$\Rightarrow \frac{\partial L}{\partial a_i}$: joue le rôle de l'erreur à retro-propager dans les couches inférieures.

Plan

Topologie typique



Pour chaque neurone k, la sortie z_k

$$z_k = g\left(\sum_{j=0}^{d} w_{j,k} z_j\right) = g\left(a_k\right)$$

où

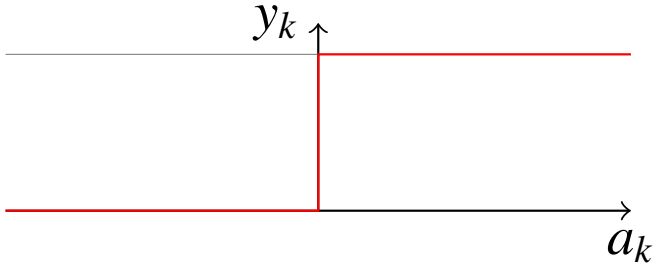
- $w_{j,k}$: poids de la connexion de la cellule j à la cellule k
- a_k : activation de la cellule k
  $a_k = \sum_{j=0}^{d} w_{j,k} z_j$
- g : fonction d'activation

Apprentissage :

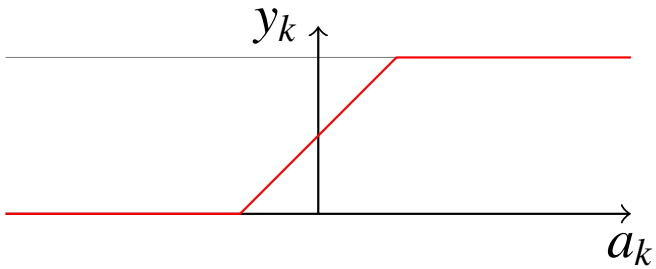
- Minimiser la fonction de coût $L\left(W, \{X, Y\}\right)$ en fonction du paramètre $W = (w_{i,j})$
- Algorithme de rétro-propagation de gradient $\Delta w_{i,j} \propto \frac{\partial L}{\partial w_{i,j}}$

Fonction d'activation

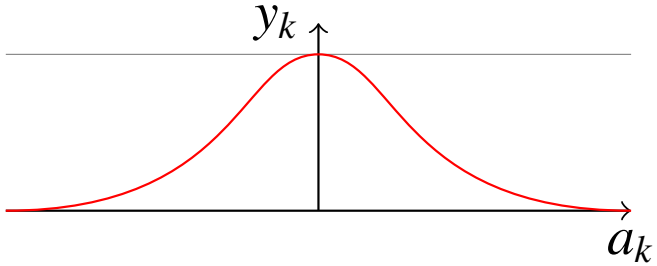
- Fonction à seuil



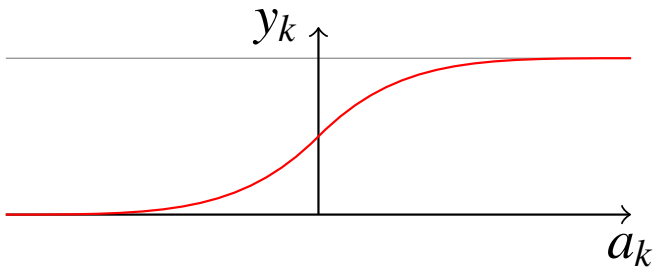
- Fonction à rampe



- Fonction radiale



- Fonction sigmoïde



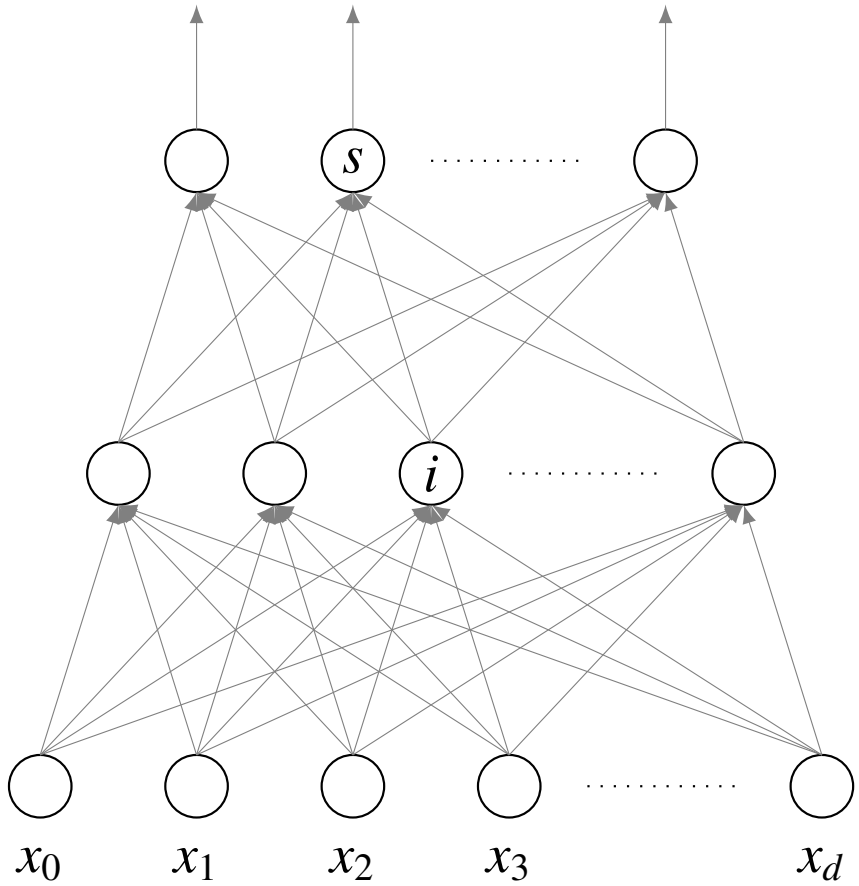
  - $g(a) = \frac{1}{1+\exp(-a)}$
  - $g'(a) = g(a)(1 - g(a))$

Passe avant (forward) Illustrations J.-N. Vittaut

Couche de sortie

Couche cachée

Couche d'entrée

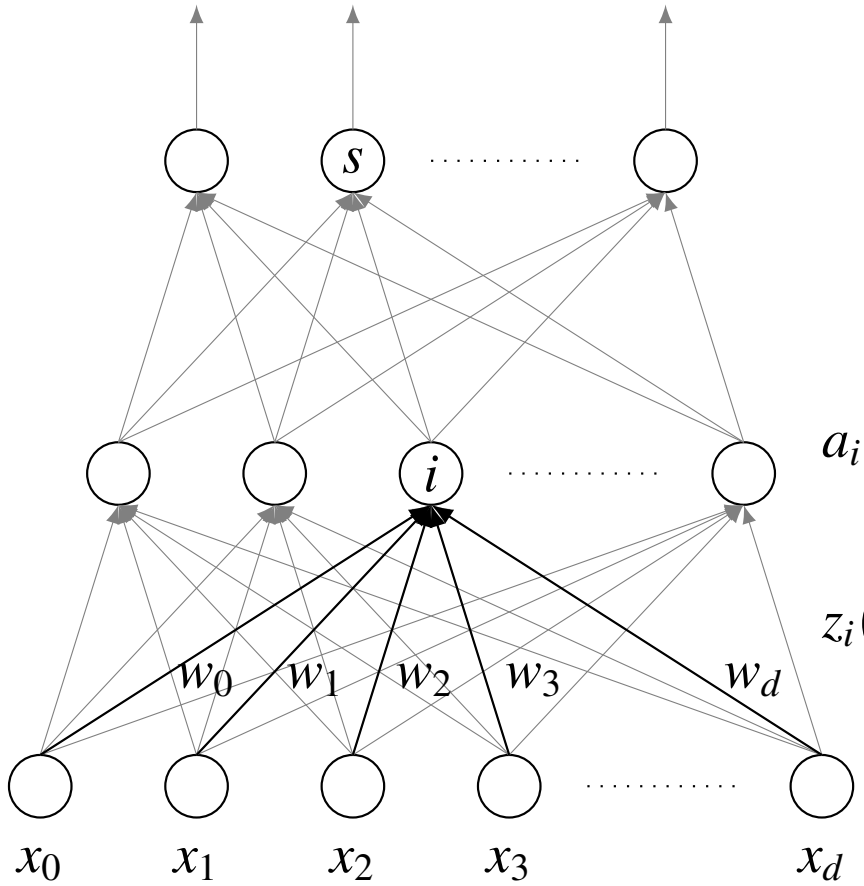


Passe avant (forward) Illustrations J.-N. Vittaut

Couche de sortie

Couche cachée

Couche d'entrée



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

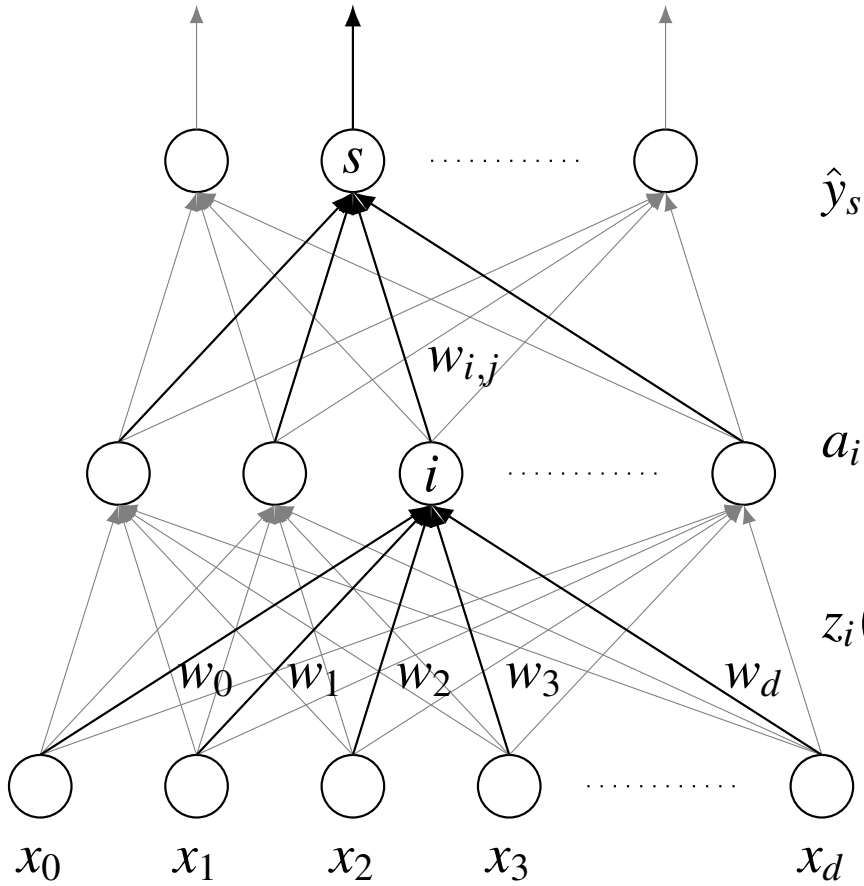
$$z_i(x) = g\left(a_i(x)\right)$$

Passe avant (forward) Illustrations J.-N. Vittaut

Couche de sortie

Couche cachée

Couche d'entrée

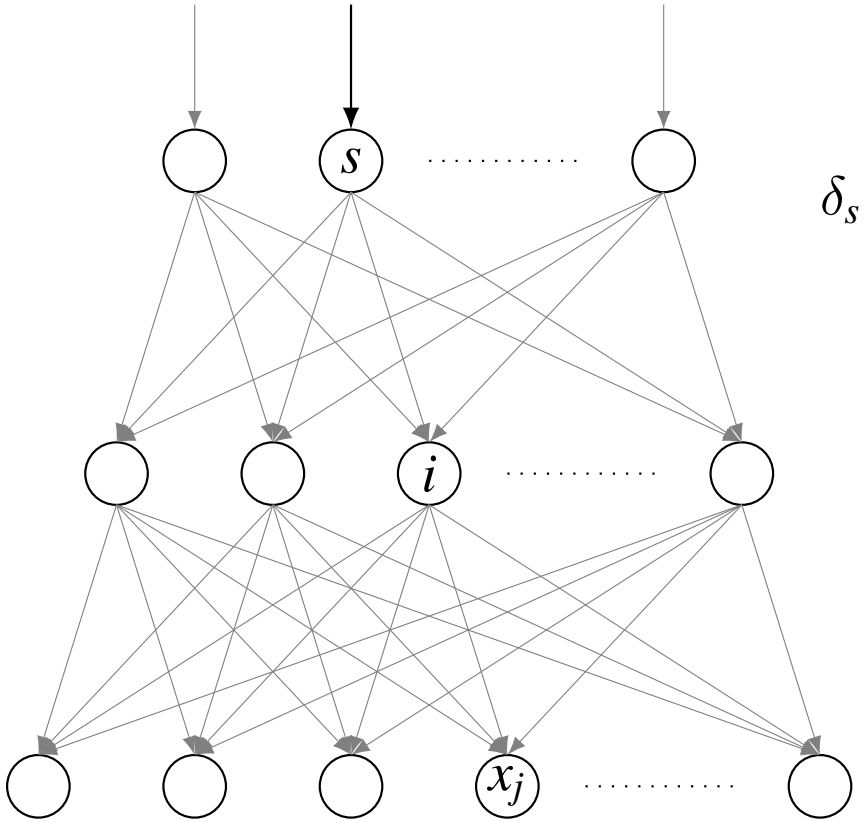


$$\hat{y}_s(x) = \sum_{i=1}^{k} w_{i,s} z_i$$

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

$$z_i(x) = g\left(a_i(x)\right)$$

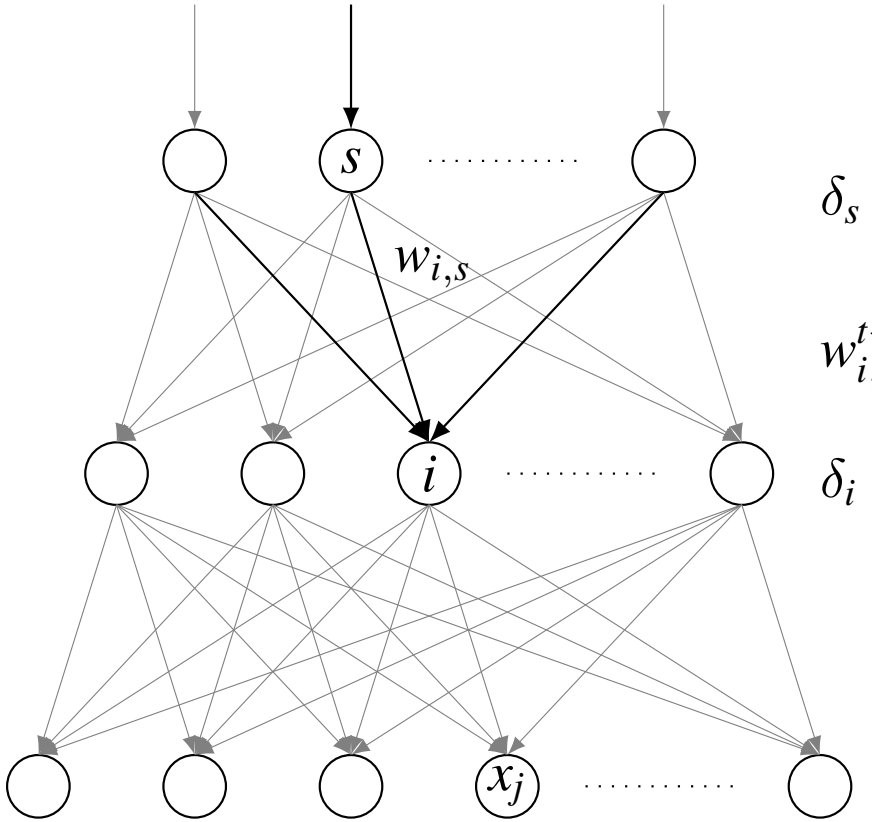
Passe arrière (backward)



$$\delta_s = g'(a_s)(y_s - \hat{y}_s)$$

- δ_s : erreur en sortie
- δ_i : somme des erreurs provenant des cellules suivantes

Passe arrière (backward)



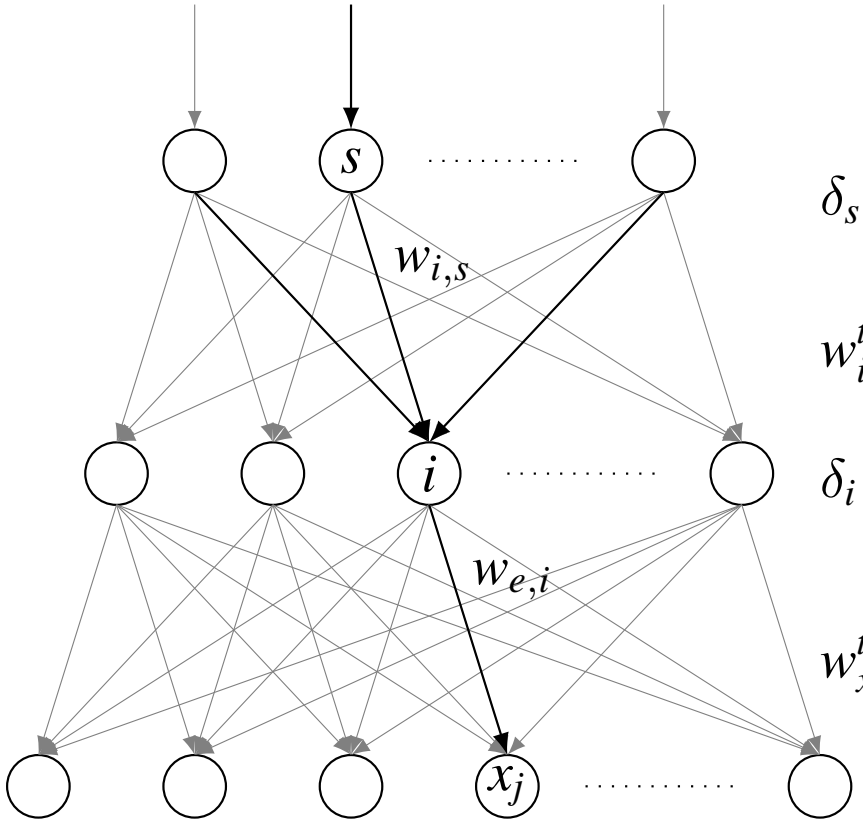
$$\delta_s = g'(a_s)(y_s - \hat{y}_s)$$

$$w_{i,s}^{t+1} = w_{i,s}^{t} - \eta(t)\delta_s a_i$$

$$\delta_i = g'(a_i) \sum_{s \in coucheSuiv.} w_{i,s}\delta_s$$

- δ_s : erreur en sortie
- δ_i : somme des erreurs provenant des cellules suivantes

Passe arrière (backward)



$$\delta_s = g'(a_s)(y_s - \hat{y}_s)$$

$$w_{i,s}^{t+1} = w_{i,s}^{t} - \eta(t)\delta_s a_i$$

$$\delta_i = g'(a_i) \sum_{s \in coucheSuiv.} w_{i,s}\delta_s$$

$$w_{x_j,i}^{t+1} = w_{x_j,i}^{t} - \eta(t)\delta_i x_j$$

- δ_s : erreur en sortie
- δ_i : somme des erreurs provenant des cellules suivantes

Algorithme

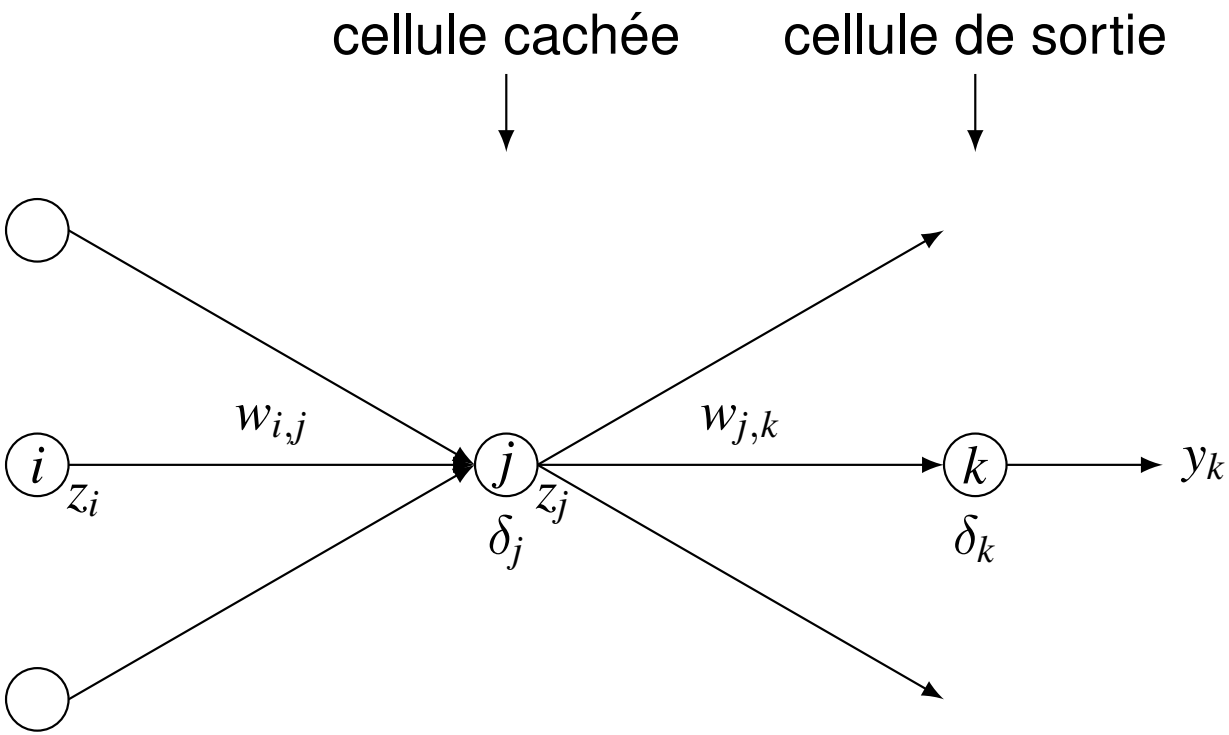
1. Présentation d'un/des exemple(s) parmi l'ensemble d'apprentissage
2. Calcul de l'état du réseau (phase forward)
3. Calcul de l'erreur avec un coût donné : e.g. = $(y - \hat{y})^2$)
4. Calcul des gradients (par l'algorithme de rétro-propagation du gradient)
5. Modification des poids
6. Critère d'arrêt (sur l'erreur, nombre de présentation d'exemples...)
7. Retour en 1

La rétro-propagation de gradient

- Le problème :
  - Détermination des responsabilités (credit assignment problem)
  - Quelle connexion est responsable, et de combien, de l'erreur ?
- Principe :
  - Calculer l'erreur sur une connexion en fonction de l'erreur sur la couche suivante
- Deux étapes :
  1. Evaluation des dérivées de l'erreur par rapport aux poids
  2. Utilisation de ces dérivées pour calculer la modification de chaque poids

La rétro-propagation de gradient

- a_i : activation de la cellule i
- z_i : sortie de la cellule i
- δ_i : erreur attachée à la cellule i



La rétro-propagation de gradient

- 1. Evaluation de l'erreur L due à chaque connexion : $\frac{\partial L}{\partial w_{i,j}}$
  - calculer l'erreur sur la connexion $w_{i,j}$ en fonction de l'erreur après la cellule j

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial a_j}\frac{\partial a_j}{\partial w_{i,j}} = \delta_j z_i$$

  - Pour les cellules de la couche de sortie :

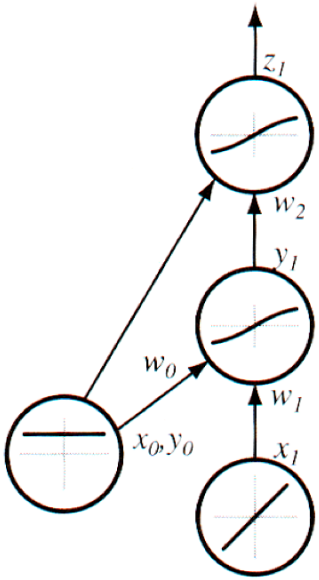
$$\delta_k = \frac{\partial L}{\partial a_k} = g'(a_k)(y_k - \hat{y}_k)$$

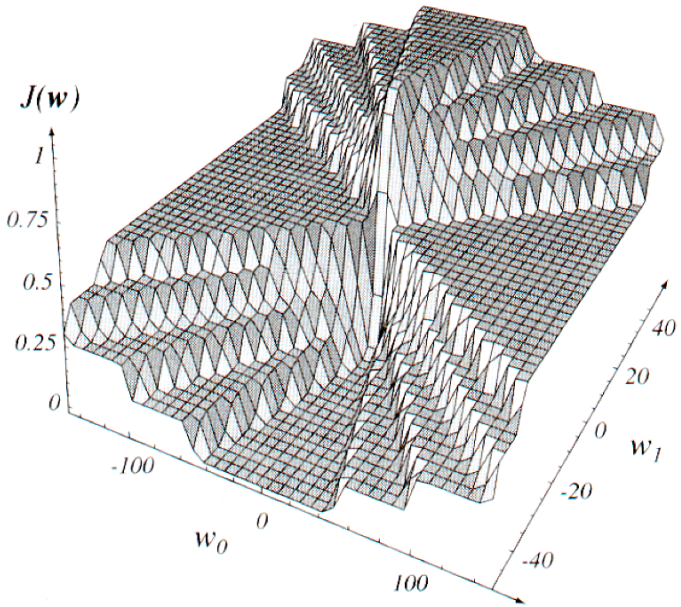
  - Pour les cellules d'une couche cachée :

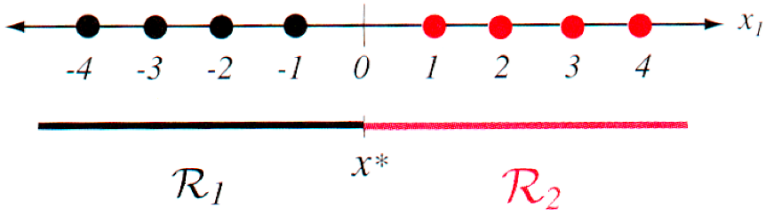
$$\delta_j = \frac{\partial L}{\partial a_j} = \sum_k \frac{\partial L}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j} = g'(a_j)\cdot\sum_k w_{j,k}\delta_k$$

Plan

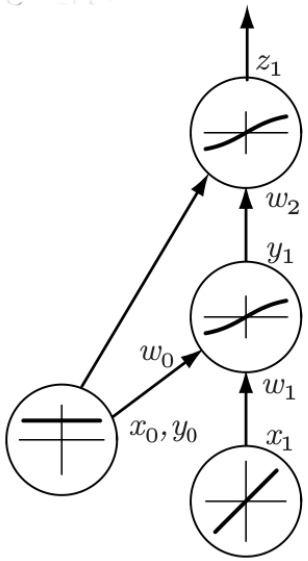
Analyse de la surface d'erreur

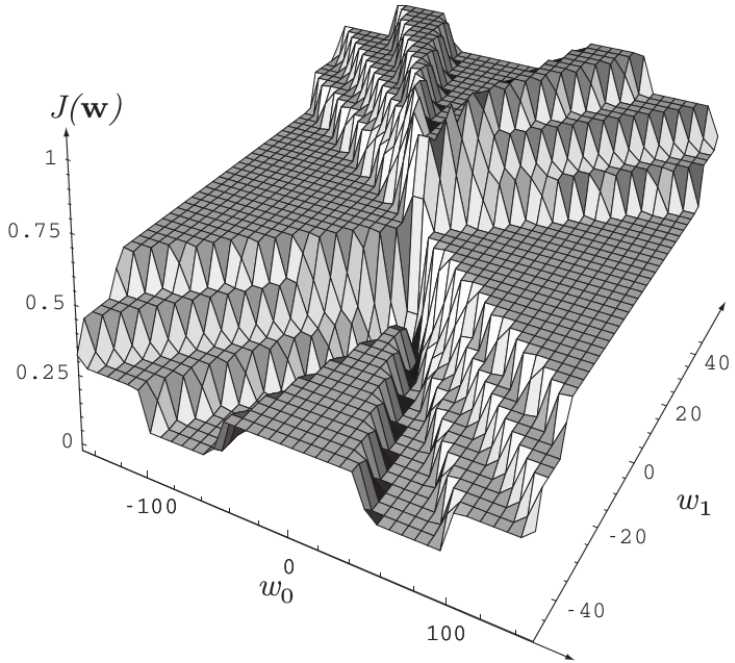


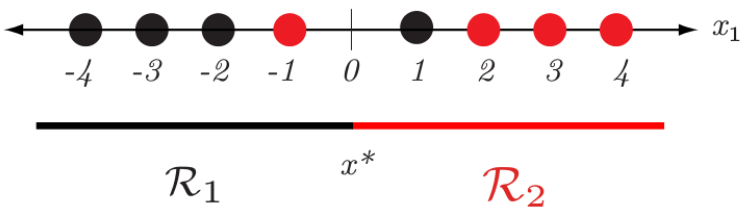




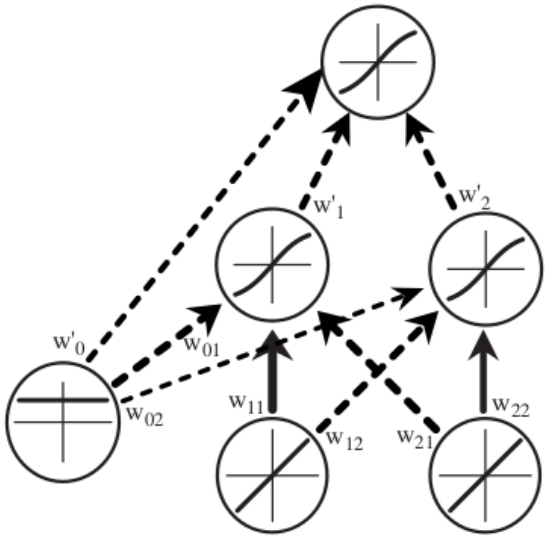
Analyse de la surface d'erreur

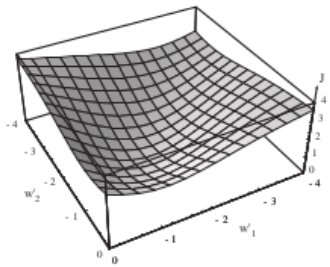


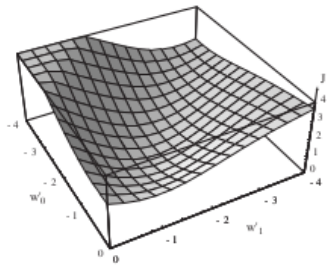


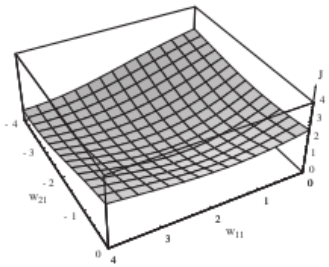


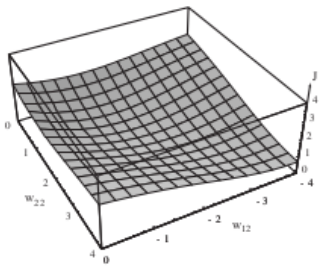
Analyse de la surface d'erreur





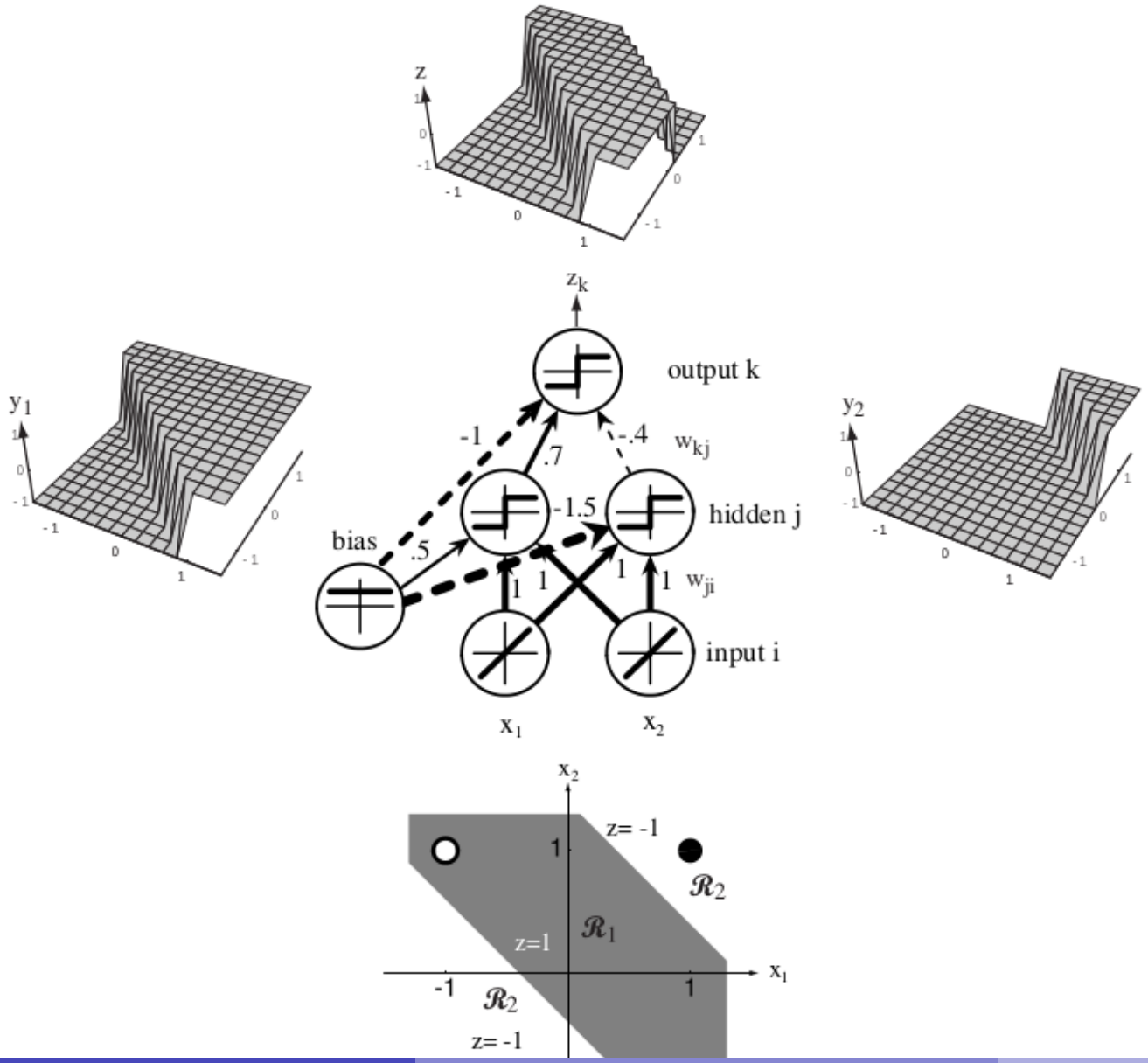






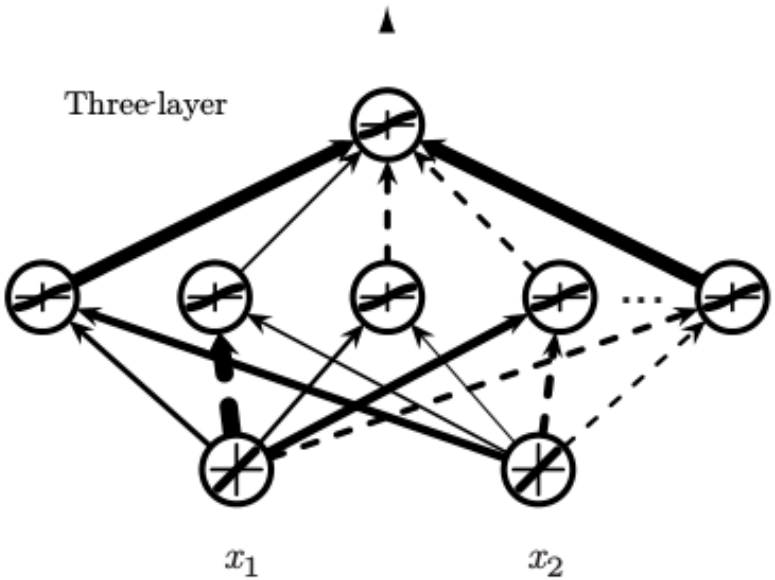
Exemple

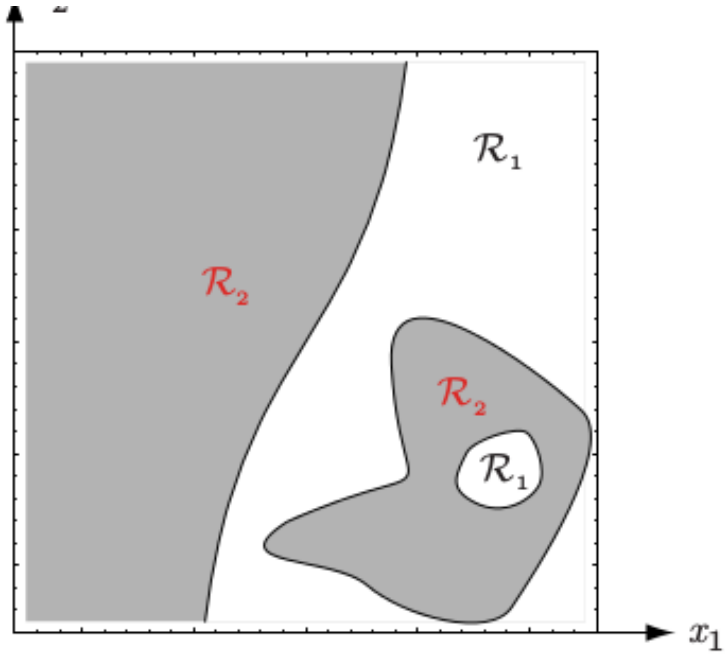
Le XOR selon [Duda et al 00]



Exemple

Non convexité des régions apprises [Duda et al 00]





Complexité et expressivité

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

Expressivité

- Quelle influence du nombre de couches ?
- du nombre de neurones par couche ?

$\Rightarrow$ 3 couches suffises pour un apprentissage universel ! Mais . . .

Plan

Multi-classes

Quand il faut prédire K classes

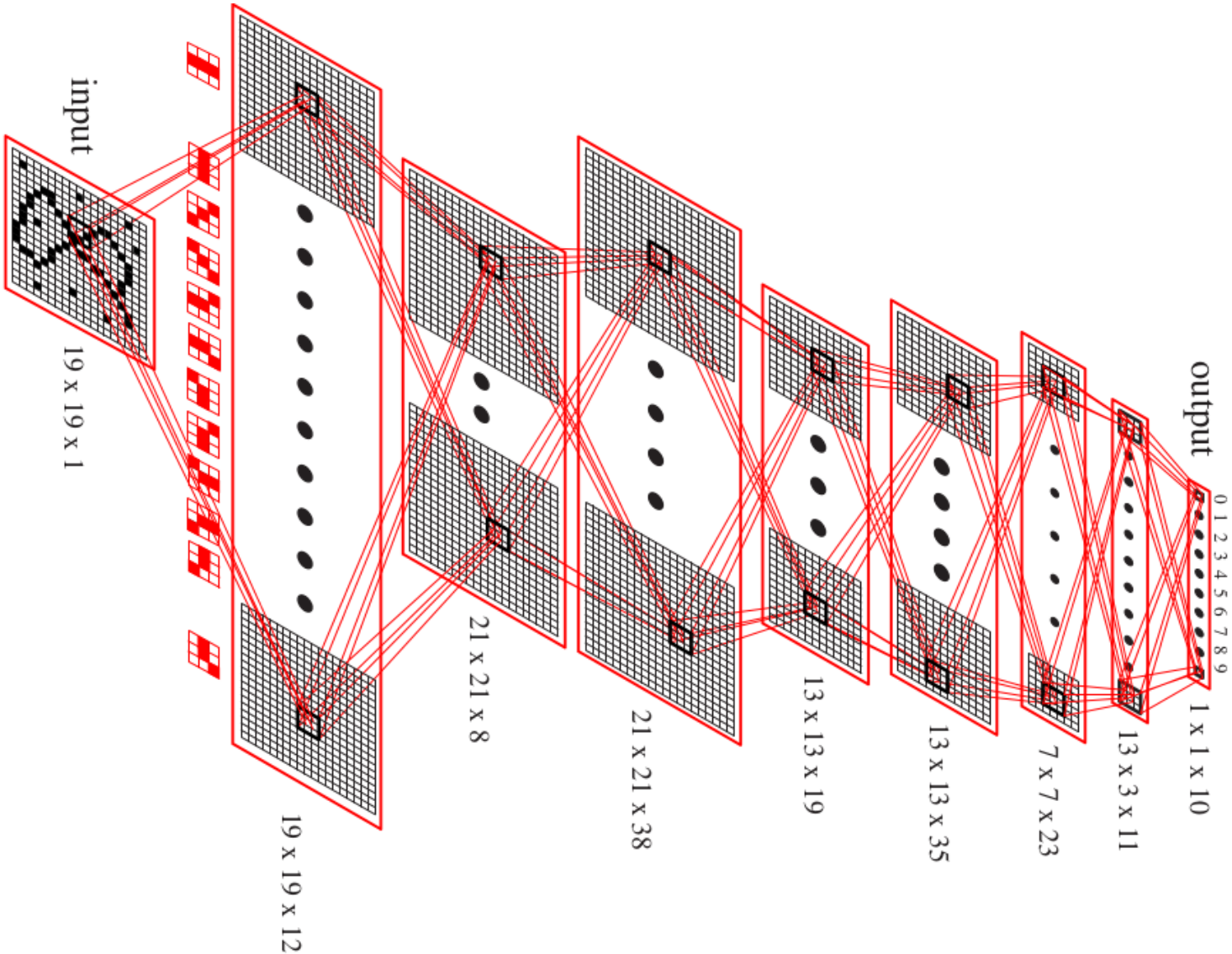
- K sorties
- Utilisation de vecteurs 1-hot : $(0, 0, \ldots, 1, \ldots, 0)$ pour l'apprentissage
- Mise à l'échelle des sorties par softmax : $\hat{y}_i = \frac{e^{z_i}}{\sum_j^K e^{z_j}}$
- Utilisation de la cross-entropie (similaire à la négative vraissemblance) : $\sum_i y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)$.

Plan

Réseaux convolutifs

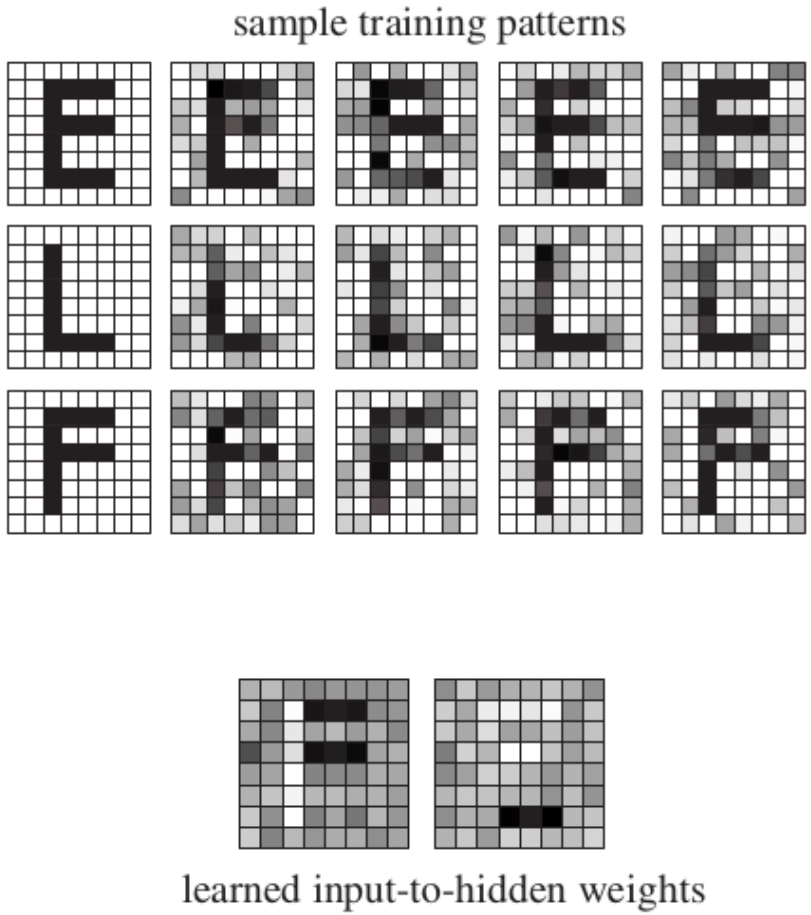
Reconnaissance de caractères

[Duda et al 00]



Exemple

Reconnaissance de caractères (couches internes) [Duda et al 00]



Réseaux convolutifs

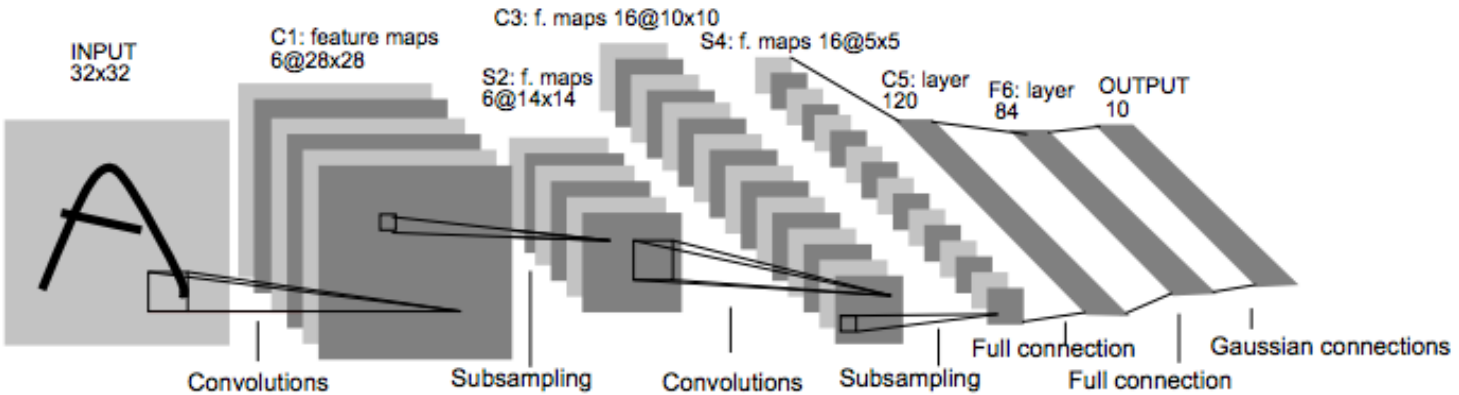
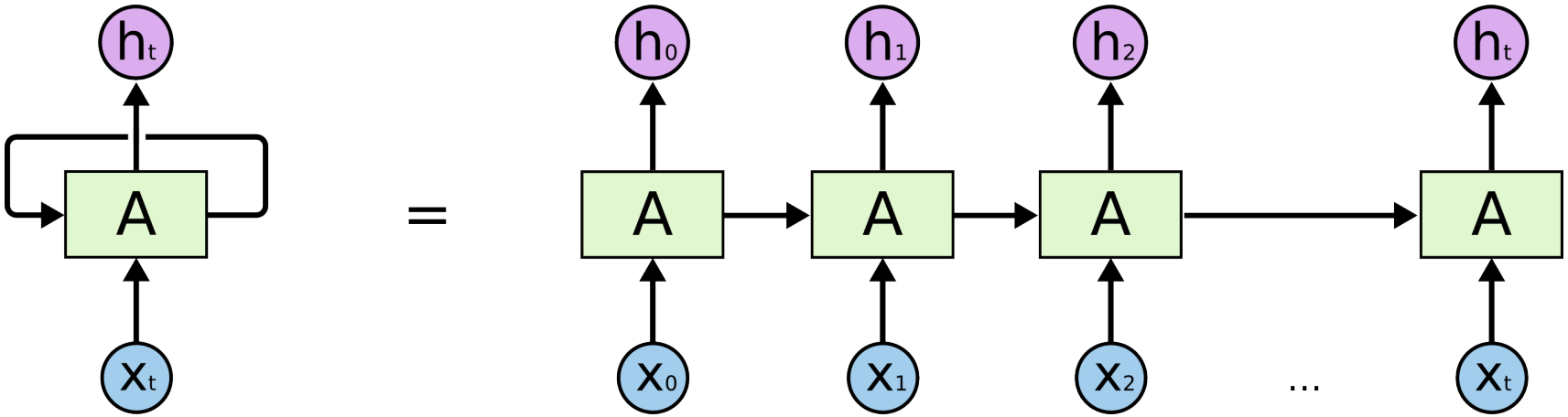


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

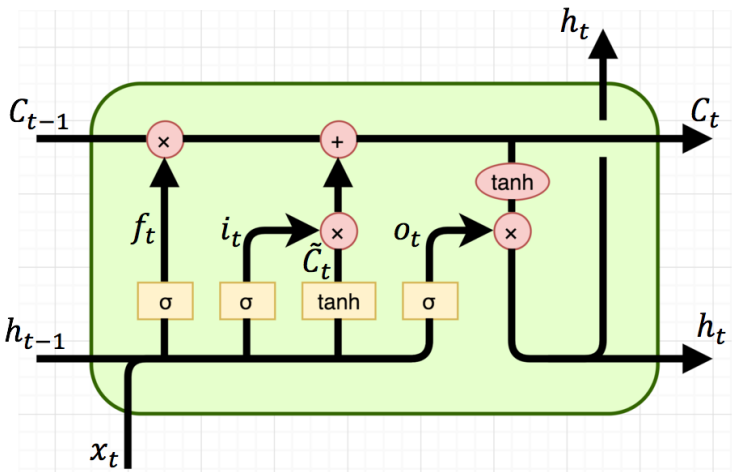
Réseaux récurrents



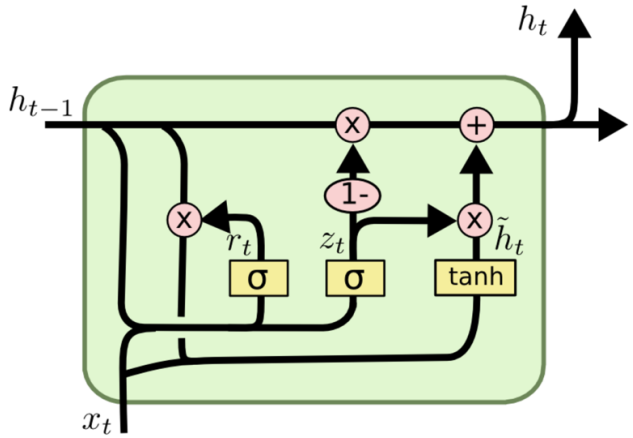
Pour les séquences

- L'objectif est de prédire l'élément suivant d'une séquence : $(x^0, x^1, \ldots, x^t)$
- Hypothèse : $x^{t+1} = f(x^t, h^t)$ l'élément suivant d'une séquence dépend des éléments précédents et d'un état mémoire latent.
- Le réseau prédit pour chaque passage l'état h^t latent.
- Un autre réseau est utilisé pour "décoder" l'état mémoire en la valeur x^t associée.

Long Short Term Memory (LSTM) et Gated Recurrent Unit (GRU)



(a) Long Short-Term Memory

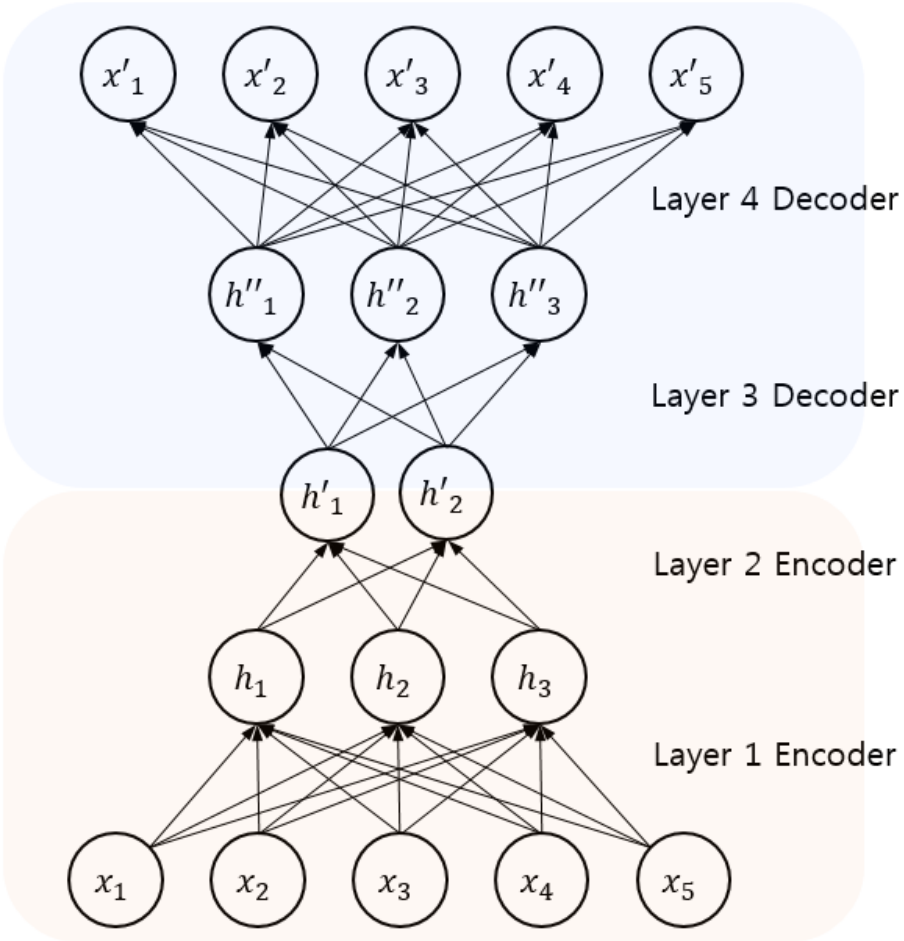


(b) Gated Recurrent Unit

Deux variantes des RNNs

- Un processus de mémoire mis-à-jour automatiquement
- Permet de se ”souvenir” (d’avoir une information persisente d’état en état)

Auto-encoders



Apprentissage non-supervisé

Objectif :

- apprendre une “compression” des données utile pour l’apprentissage
- $f^{-1}(f(x)) \approx x$: la sortie doit être proche de l’entrée

Intérêts :

- clustering des données
- lissage/débruitage
- visualisation
- ...

Et beaucoup d'autres

- RBMs (Restricted Boltzmann Machine)
- V.A.E. (Variational Auto-encoder)
- G.A.N. (Generative Adversial Network)
- Mécanisme d'attention
- Stochastic Unit (Reinforce, Gumbel-Softmax, Straight-Through estimator)

Vers les réseaux profonds

Problème : plus le réseau est profond plus il est dur à entraîner

- le gradient s'évapore (*vanishing*)
- le sur-apprentissage est très favorisé

Quelques solutions

- utiliser des architectures peu propices au sur-apprentissage (convolutives, RBM, . . .)
- Early-stopping
- Apprentissage en bruitant les données d'entrées

$\Rightarrow$ pas suffisant

- Première passe d'apprentissage pour "bien initialiser" les couches $\Rightarrow$ auto-encoders
- Drop-out : permet de limiter le sur-apprentissage (éteindre/supprimer un nombre de neurones aléatoirement pendant l'apprentissage)
- utilisation de fonctions d'activation spécifiques (Relu :$max(0, a)$ et dérivées)