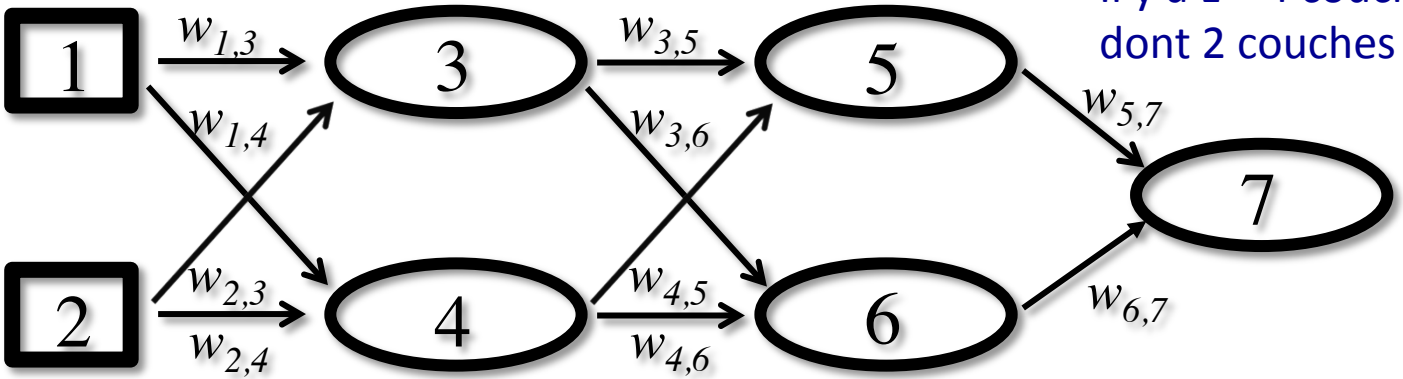
Réseau de neurones à L couches



Il y a L = 4 couches
dont 2 couches cachées

- On note a_j l'activité du *j*e « neurone », incluant les neurones d'entrée et de sortie. Donc on aura $a_i = x_i$
- On note in_j l'activité du *j*e neurone avant la non-linéarité logistique, c'est à dire

$$a_j = Logistic(in_j) = Logistic(\Sigma_i \; w_{i,j} a_i)$$

Dérivation de la règle d'apprentissage

- La dérivation de la règle d'apprentissage se fait encore avec les gradients

$$w_{i,j} \leftarrow w_{i,j} - \alpha \frac{\partial}{\partial w_{i,j}} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) \quad \forall i, j$$

- Calculer ces dérivées partielles peut paraître ardu

- Le calcul est grandement facilité en utilisant **la règle de dérivation en chaîne**

Dérivation en chaîne

- Si on peut écrire une fonction $f(x)$ à partir d'un résultat intermédiaire $g(x)$

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(x)}{\partial g(x)} \frac{\partial g(x)}{\partial x}$$

- Si on peut écrire une fonction $f(x)$ à partir de résultats intermédiaires $g_i(x)$, alors on peut écrire la dérivée partielle

$$\frac{\partial f(x)}{\partial x} = \sum_i \frac{\partial f(x)}{\partial g_i(x)} \frac{\partial g_i(x)}{\partial x}$$

Dérivation de la règle d'apprentissage

- La dérivation de la règle d'apprentissage se fait encore avec les gradients

$$w_{i,j} \leftarrow w_{i,j} - \alpha \frac{\partial}{\partial w_{i,j}} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) \quad \forall i, j$$

- Par l'application de la dérivée en chaîne, on a:

$$w_{i,j} \longleftarrow w_{i,j} - \alpha \underbrace{\frac{\partial}{\partial in_j} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t))}_{\text{gradient de la perte p/r à la somme des entrées du neurone}} \underbrace{\frac{\partial}{\partial w_{i,j}} in_j}_{\text{gradient de la somme p/r au poids } w_{i,j}}$$

- Par contre, un calcul naïf de toutes ces dérivées serait très inefficace
  - on utilise la procédure de **rétropropagation des gradients (ou erreurs)**

Rétropropagation des gradients

- Utiliser le fait que la dérivée pour un neurone à la couche l peut être calculée à partir de la dérivée des neurones connectés à la couche $l+1$

$$\frac{\partial Loss}{\partial in_j} = \frac{\partial Loss}{\partial a_j} \frac{\partial a_j}{\partial in_j}$$

$$= \left(\sum_k \frac{\partial Loss}{\partial in_k} \frac{\partial in_k}{\partial a_j} \right) \frac{\partial g(in_j)}{\partial in_j}$$

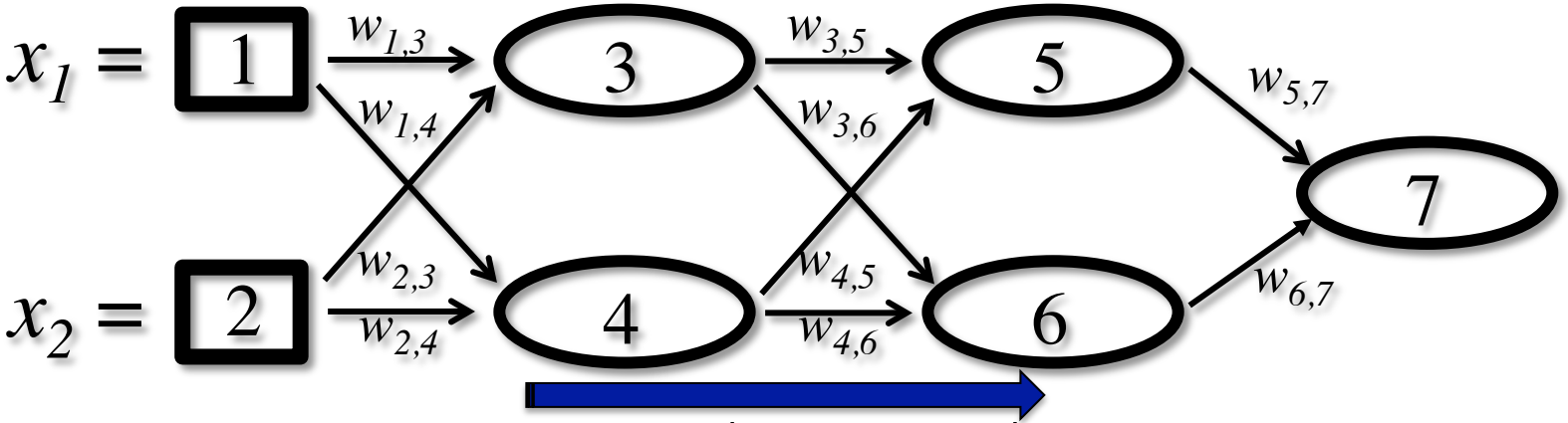
$$= \left(\sum_k \frac{\partial Loss}{\partial in_k} w_{j,k} \right) g(in_j)(1 - g(in_j))$$

k itère sur les neurones cachés de la couche $l+1$

où $in_k = \sum_i w_{i,k} a_i$ et $Logistic(\cdot) \equiv g(\cdot)$ (pour simplifier notation)

Visualisation de la rétropropagation

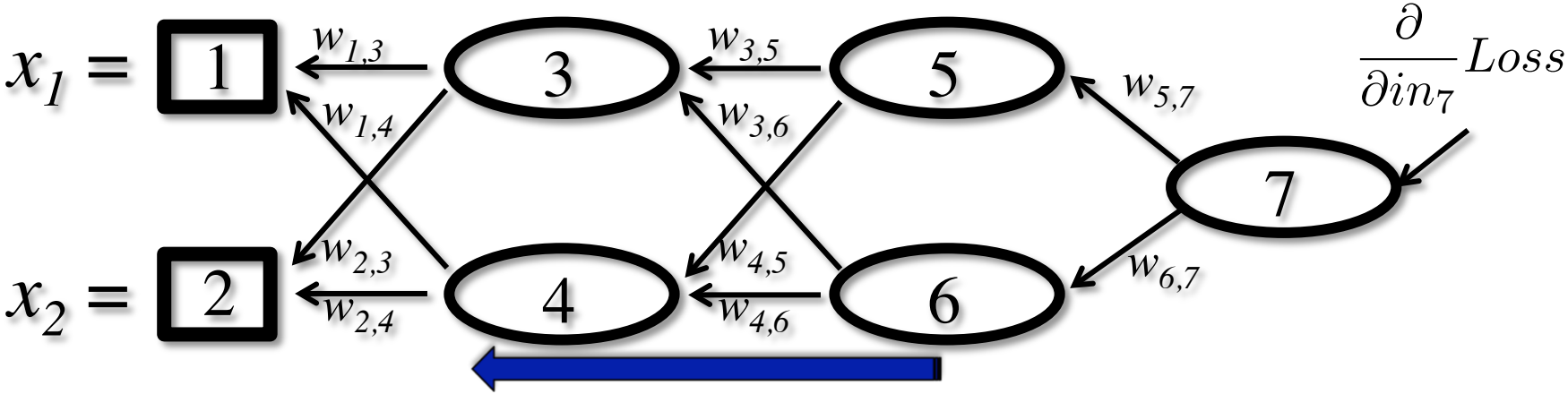
- L'algorithme d'apprentissage commence par une **propagation avant**



$$a_k = g\left(\sum_j w_{j,k} a_j\right)$$

Visualisation de la rétropropagation

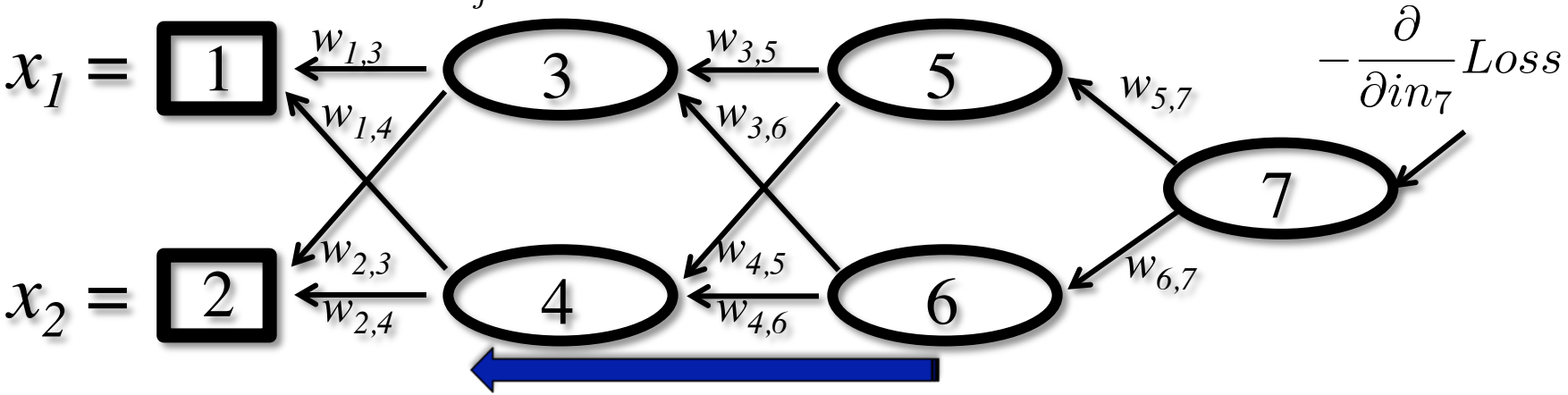
- Ensuite, le gradient sur la sortie est calculé, et le gradient rétropropagé



$$\frac{\partial}{\partial in_j} Loss = g(in_j)(1 - g(in_j)) \sum_k w_{j,k} \frac{\partial}{\partial in_k} Loss$$

Visualisation de la rétropropagation

- Peut propager $-\frac{\partial}{\partial in_j} Loss = \Delta[j]$ aussi (décomposition équivalente du livre)



$$\Delta[j] = g(in_j)(1 - g(in_j)) \sum_k w_{j,k} \Delta[k]$$

Retour sur la règle d'apprentissage

- La dérivation de la règle d'apprentissage se fait encore avec les gradients

$$w_{i,j} \longleftarrow w_{i,j} - \alpha \underbrace{\frac{\partial}{\partial in_j} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t))}_{-\Delta[j]} \underbrace{\frac{\partial}{\partial w_{i,j}} in_j}_{a_i}$$

- Donc la règle de mise à jour peut être écrite comme suite:

$$w_{i,j} \leftarrow w_{i,j} + \alpha \; a_i \; \Delta[j]$$

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
repeat
for each example (**x**, **y**) **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\, a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow y_j - a_j \quad (= -\partial Loss/\partial in_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g(in_i)(1 - g(in_i)) \sum_j w_{i,j}\Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

$Logistic(\cdot) \equiv g(\cdot)$
(pour simplifier notation)