UNIVERSITÉ LAVAL

GLO-4030/7030
APPRENTISSAGE PAR RÉSEAUX DE NEURONES PROFONDS

Réseaux en aval (*feedforward*)

Fonctions de perte

Graphe de calculs et rétropropagation (*backprop*)

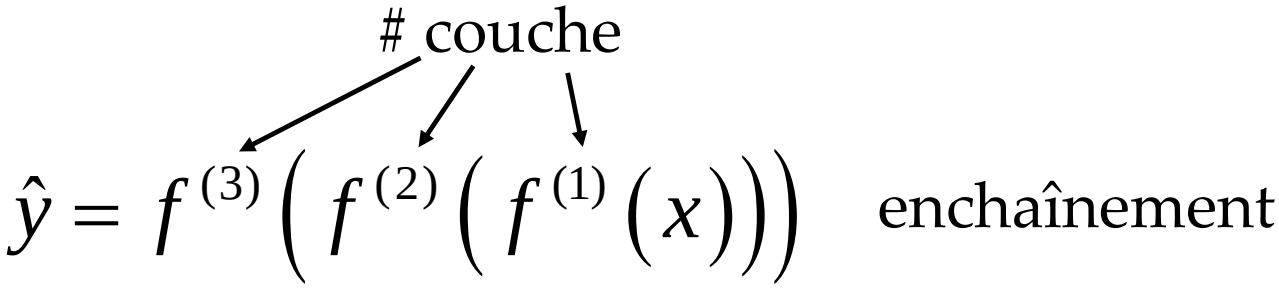
Hiver 2020

Philippe Giguère

Réseaux en aval (*feedforward*)

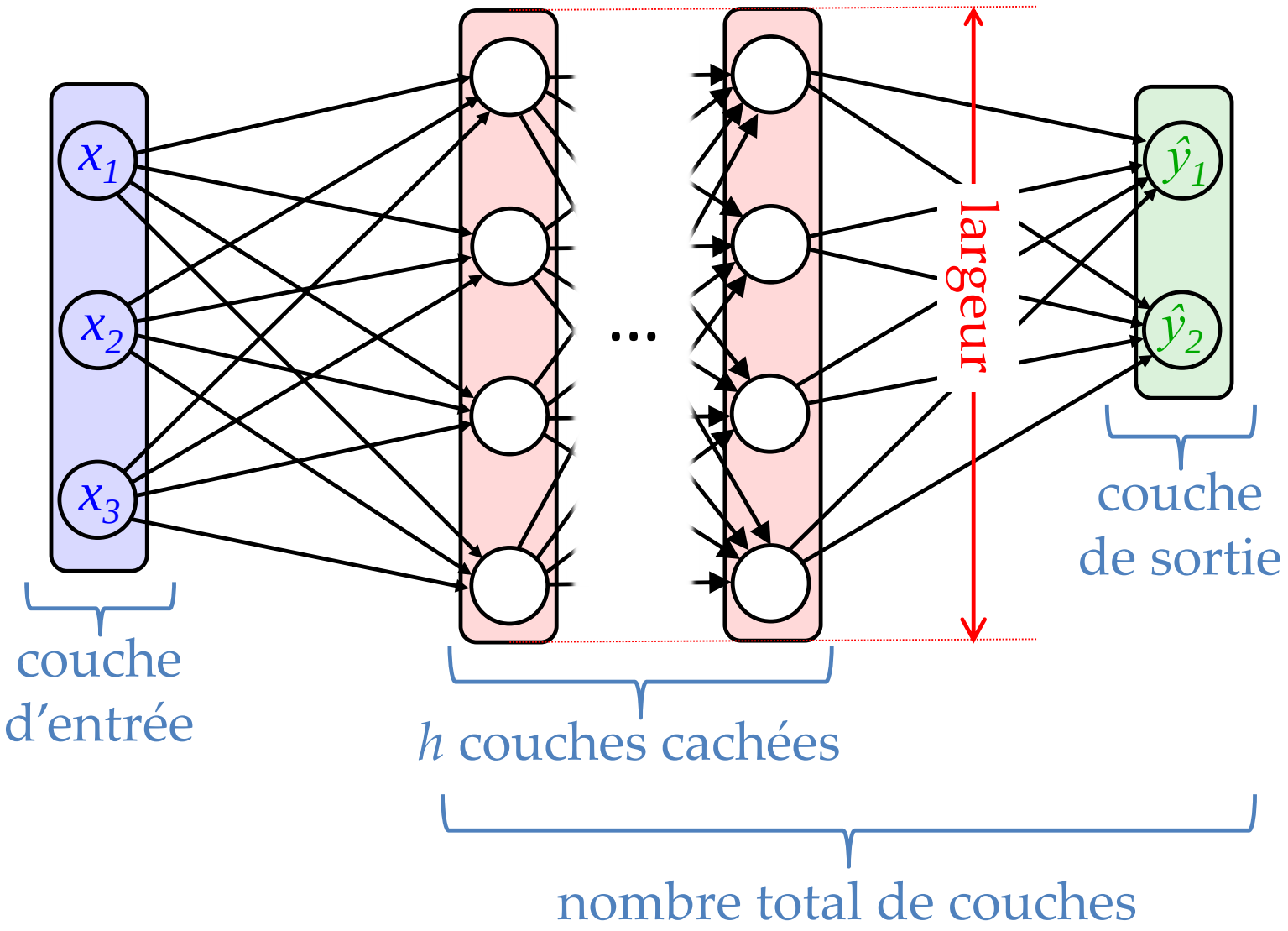
Réseau en aval

- Forment la base de beaucoup de systèmes
- Composition de plusieurs fonctions



  - exprimé par un graphe acyclique
  - par opposition : réseaux récurrents

Illustration et nomenclature



(Note : exemple pour réseau typique *fully-connected*)

Réseau en aval

- Cherche à approximer une fonction f^*
- Par exemple, un classificateur $y = f^*(x)$
- Appelé en aval (*feedfoward*) car l'information s'écoule de $x \rightarrow y$.
- Recherche les paramètres θ du réseau f qui donne la meilleure approximation :

$$\hat{y} = f(x;\theta)$$

Choix à faire

- Architecture (l'ensemble du cours)
  - # couches
  - # neurones (cachés) par couche
  - type de couche
- Forme de la sortie → fonction de sortie
- Fonction de perte
- Optimiseur (semaine prochaine)
  - et autres « *training details* »

« training details » : YOLO9000

Training for classification. We train the network on the standard ImageNet 1000 class classification dataset for 160 epochs using stochastic gradient descent with a starting learning rate of 0.1, polynomial rate decay with a power of 4, weight decay of 0.0005 and momentum of 0.9 using the Darknet neural network framework [13]. During training we use standard data augmentation tricks including random crops, rotations, and hue, saturation, and exposure shifts.

As discussed above, after our initial training on images at 224×224 we fine tune our network at a larger size, 448. For this fine tuning we train with the above parameters but for only 10 epochs and starting at a learning rate of 10^{-3}.

Training for detection. We modify this network for detection by removing the last convolutional layer and instead adding on three 3×3 convolutional layers with 1024 filters each followed by a final 1×1 convolutional layer with the number of outputs we need for detection.

We train the network for 160 epochs with a starting learning rate of 10^{-3}, dividing it by 10 at 60 and 90 epochs. We use a weight decay of 0.0005 and momentum of 0.9. We use a similar data augmentation to YOLO and SSD with random crops, color shifting, etc. We use the same training strategy on COCO and VOC.

Recettes parfois assez complexes…

Exemples de (fonction de) sortie

- softmax pour multiclasse exclusif
  - par rapport à *multilabel*
- sigmoïde pour 0 à 1
- tanh pour -1 à 1
- sin θ et cos θ, pour des angles (*grasping*)
- linéaire (*regression layer*)
  - sorties pré-activation $z = \boldsymbol{W}^T\boldsymbol{h}+\boldsymbol{b}$ de la dernière couche
- prédire la précision ($1/\sigma^2$) plutôt que la variance (section 6.2.2.4, p.182 du manuel)

Plus la sortie brute pour un réseau initialisé (au hasard) est proche de ce que l'on désire, plus facile sera la convergence lors de l'entraînement.

Quelques fonctions de perte

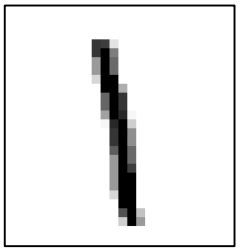
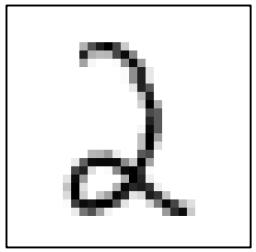
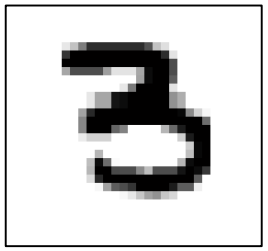
Fonction de perte

- On peut juger *qualitativement* de la qualité des réponses d'un réseau
- Mais besoin d'un formalisme *quantitatif* si on veut l'optimiser : **fonction de perte**
- Calculée sur les exemples d'entraînement (*perte empirique*)
- Entraînement : recherche dans l'espace des paramètres θ, de manière efficace, pour minimiser cette perte

Exemple de loss : SVM Multiclasse

(utilisé la semaine passé)

Scores $s_i = f(x) = Wx$

la cible est l'étiquette y_i

1	**2.3**	1.7	1.7
2	4.1	**5.4**	1.4
3	-2.1	2.3	**-1.3**

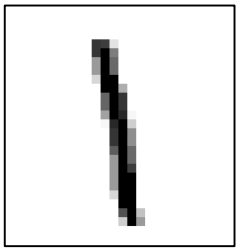
$$L = \sum_{j \neq cible} \begin{cases} 0 & \text{si } s_{cible} \geq s_j + 1 \\ s_j - s_{cible} + 1 & \text{autrement} \end{cases}$$

$$= \sum_{j \neq cible} \max(0, s_j - s_{cible} + 1)$$

Adapté de cs231n

Exemple de loss : SVM Multiclasse

Scores $s_i = f(x) = Wx$



1	2.3	1.7	1.7
2	4.1	5.4	1.4
3	-2.1	2.3	-1.3

$$L = \sum_{j \neq cible} \begin{cases} 0 & \text{si } s_{cible} \geq s_j + 1 \\ s_j - s_{cible} + 1 & \text{autrement} \end{cases}$$

$$= \sum_{j \neq cible} \max(0, s_j - s_{cible} + 1)$$

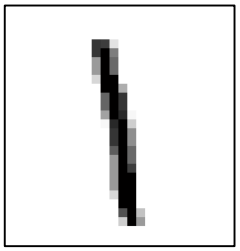
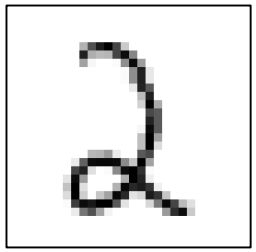
Perte : 2.8

```
= max(0,  4.1 - 2.3 + 1) +
  max(0, -2.1 - 2.3 + 1)
= max(0,2.8) + max(0,-3.4) = 2.8
```

Adapté de cs231n

Exemple de loss : SVM Multiclasse

Scores $s_i = f(x) = Wx$

1	2.3	1.7	1.7
2	4.1	5.4	1.4
3	-2.1	2.3	-1.3
Perte :	2.8	0	

$$L = \sum_{j \neq cible} \begin{cases} 0 & \text{si } s_{cible} \geq s_j + 1 \\ s_j - s_{cible} + 1 & \text{autrement} \end{cases}$$

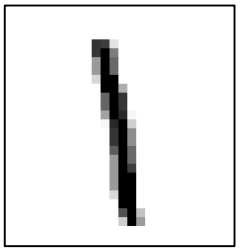
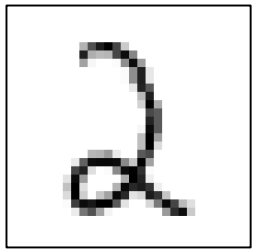
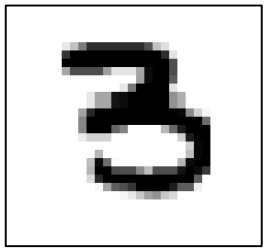
$$= \sum_{j \neq cible} \max(0, s_j - s_{cible} + 1)$$

```
= max(0,  1.7 - 5.4 + 1) +
  max(0,  2.3 - 5.4 + 1)
= max(0,-2.7) + max(0,-2.1) = 0
```

Adapté de cs231n

Exemple de loss : SVM Multiclasse

Scores $s_i = f(x) = Wx$

1	2.3	1.7	1.7
2	4.1	5.4	1.4
3	-2.1	2.3	-1.3
Perte :	2.8	0	7.7

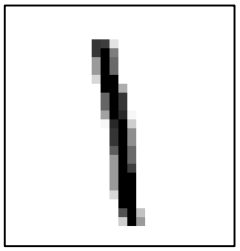
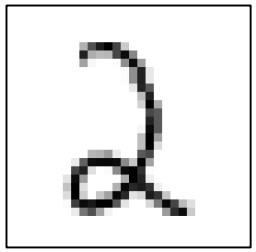
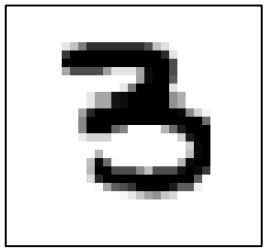
$$L = \sum_{j \neq cible} \begin{cases} 0 & \text{si } s_{cible} \geq s_j + 1 \\ s_j - s_{cible} + 1 & \text{autrement} \end{cases}$$

$$= \sum_{j \neq cible} \max(0, s_j - s_{cible} + 1)$$

```
= max(0,  1.7 - -1.3 + 1) +
  max(0,  1.4 - -1.3 + 1)
= max(0,4.0) + max(0,3.7) = 7.7
```

Exemple de loss : SVM Multiclasse

Scores $s_i = f(x) = Wx$

1	2.3	1.7	1.7
2	4.1	5.4	1.4
3	-2.1	2.3	-1.3
Perte :	2.8	0	7.7

$$L = \sum_{j \neq cible} \begin{cases} 0 & \text{si } s_{cible} \geq s_j + 1 \\ s_j - s_{cible} + 1 & \text{autrement} \end{cases}$$

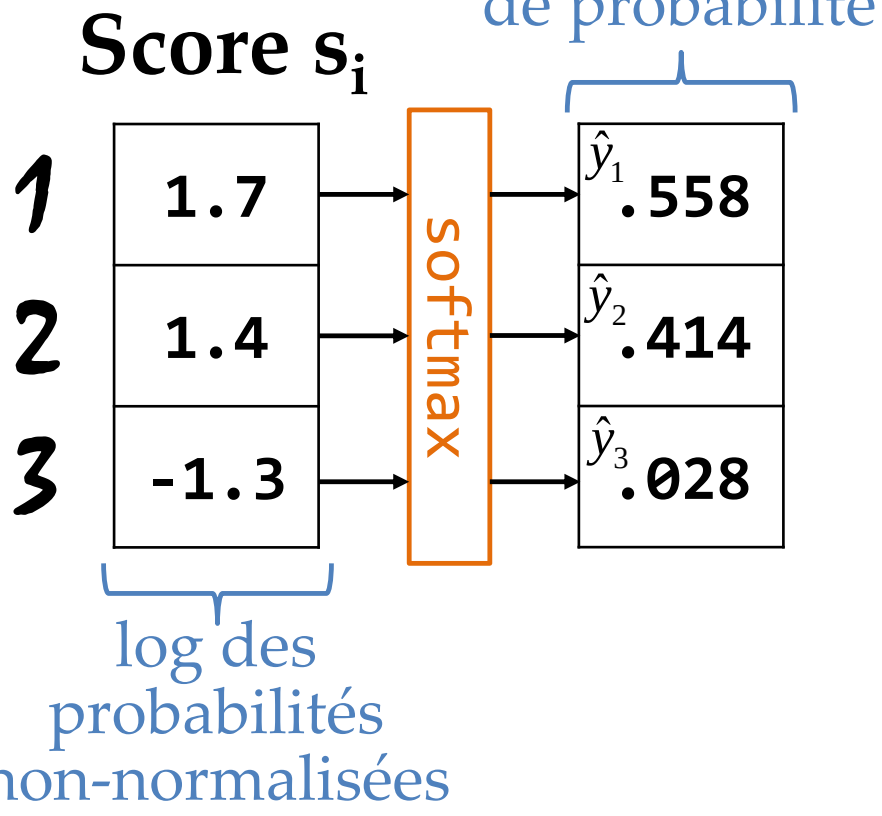
$$= \sum_{j \neq cible} \max(0, s_j - s_{cible} + 1)$$

Perte moyenne J : $\frac{2.8 + 0 + 7.7}{3} = 3.5$

Adapté de cs231n

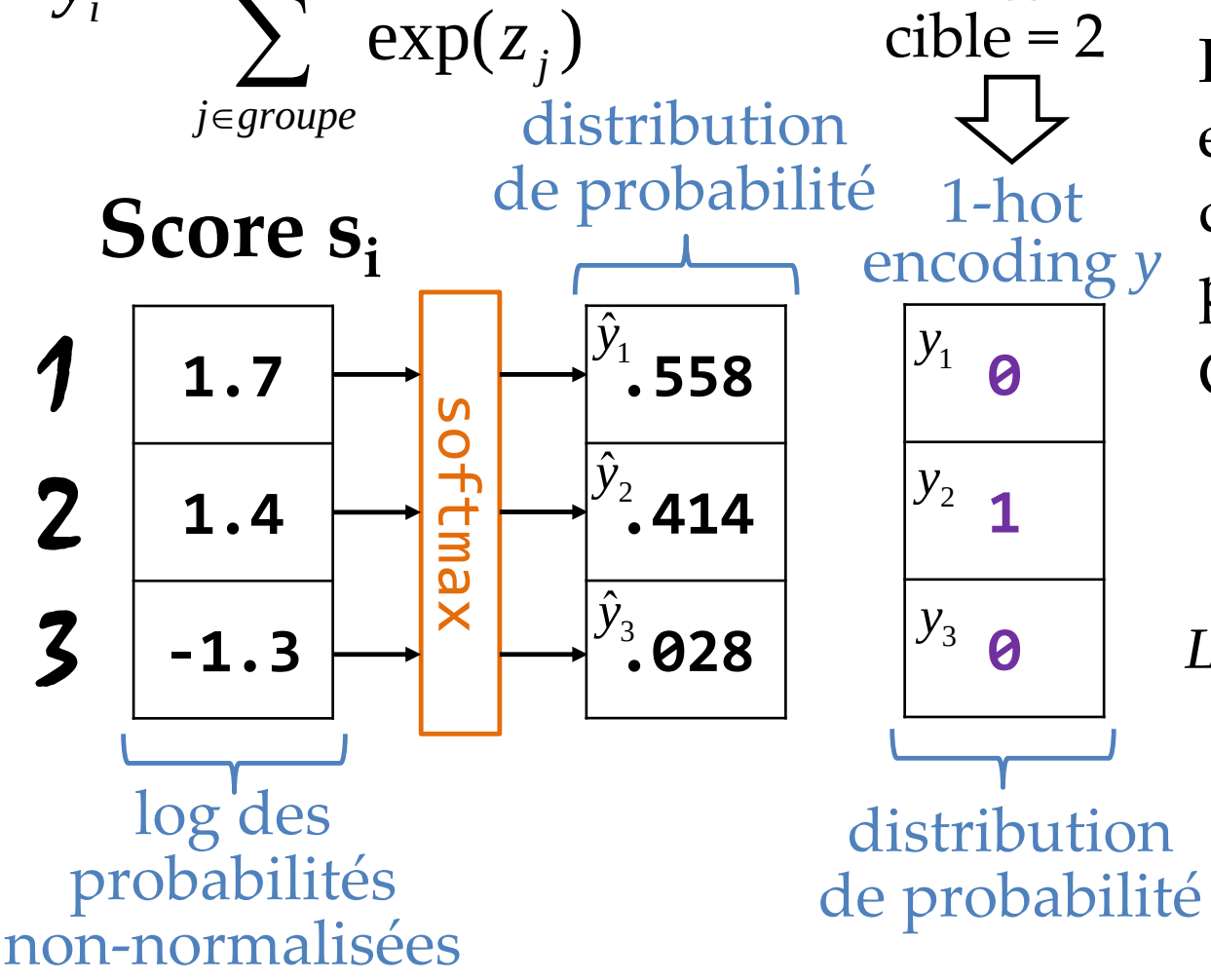
Softmax

$$\hat{y}_i = \frac{\exp(z_i)}{\sum_{j \in groupe} \exp(z_j)}$$



Perte pour softmax

$$\hat{y}_i = \frac{\exp(z_i)}{\sum_{j \in groupe} \exp(z_j)}$$



Perte sera une distance entre ces deux distributions de probabilité. Choix?

Cross-Entropy Loss!

1-hot encoding

$$L(f(x), y) = \sum_{j=1}^{c} -y_j \log \hat{y}_j$$

(se réduit à)

$$= -\log \hat{y}_{cible}$$

Cross-entropy

- Distance **asymétrique** entre deux distributions de probabilités

$$H(p,q) = \sum_{j}^{c} -p_j(x) \log q_j(x)$$

.558
.414
.028

$\hat{y} = f(x)$

0
1
0

y

Qu'arrive-t-il si on choisit :

$$H(\hat{y}, y) = \sum_{j}^{c} -\hat{y}_j \log y_j ?$$

$\log(y_1=0) = -\infty$ ☹

$\log(y_2=1) = 0$ ☹

Seul choix possible :

$$L(y, \hat{y}) = H(y, \hat{y}) = \sum_{j}^{c} -y_j \log \hat{y}_j$$

Si la cible est un *1-hot vector* , se réduit à : $L(y, \hat{y}) = -\log \hat{y}_{cible}$

Graphes de calculs et algorithme de rétropropagation `backprop`

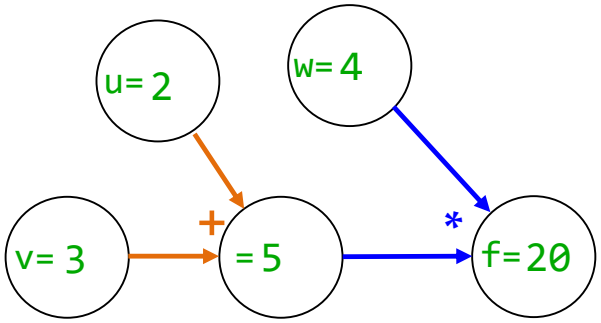
Pourquoi graphe de calcul?

- Pour faire du *lazy evaluation* :
  - découple la création du graphe de son évaluation
  - graphe créé sur CPU (avec votre code)
  - mais calculé sur GPU (10x-40x)
- Nécessaire pour calcul des gradients, via l'algorithme de rétropropagation `backprop`

Exemple graphe calcul

$$f = (u+v)w$$

nœud : variable
arête : opération



Instancie les variables
u=2
v=3
w=4
Évalue le graphe pour avoir *f* : **forward pass**

Pourquoi la backprop?

- Il faut une procédure pour trouver les paramètres θ (ex. poids W) du réseau qui minimisent la perte
- Recherche aléatoire? Bonne chance…
- Recherche informée : utilisation du gradient!

Descente du gradient

- Concept
  - Déterminer un vecteur directionnel $\boldsymbol{d}$ basé sur le gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
  - Mettre à jour les paramètres d'une fraction $\boldsymbol{a}$ du gradient $\boldsymbol{d}$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \boldsymbol{a}\boldsymbol{d}$$

  - Répéter jusqu'à convergence
- Implique de pouvoir calculer le gradient

Profil de la fonction de perte $L(\theta)$

Adapté de cs231n

Profil de la fonction de perte $L(\theta)$

Réalité : trouver le fond de la vallée embrumée, à tâtons…

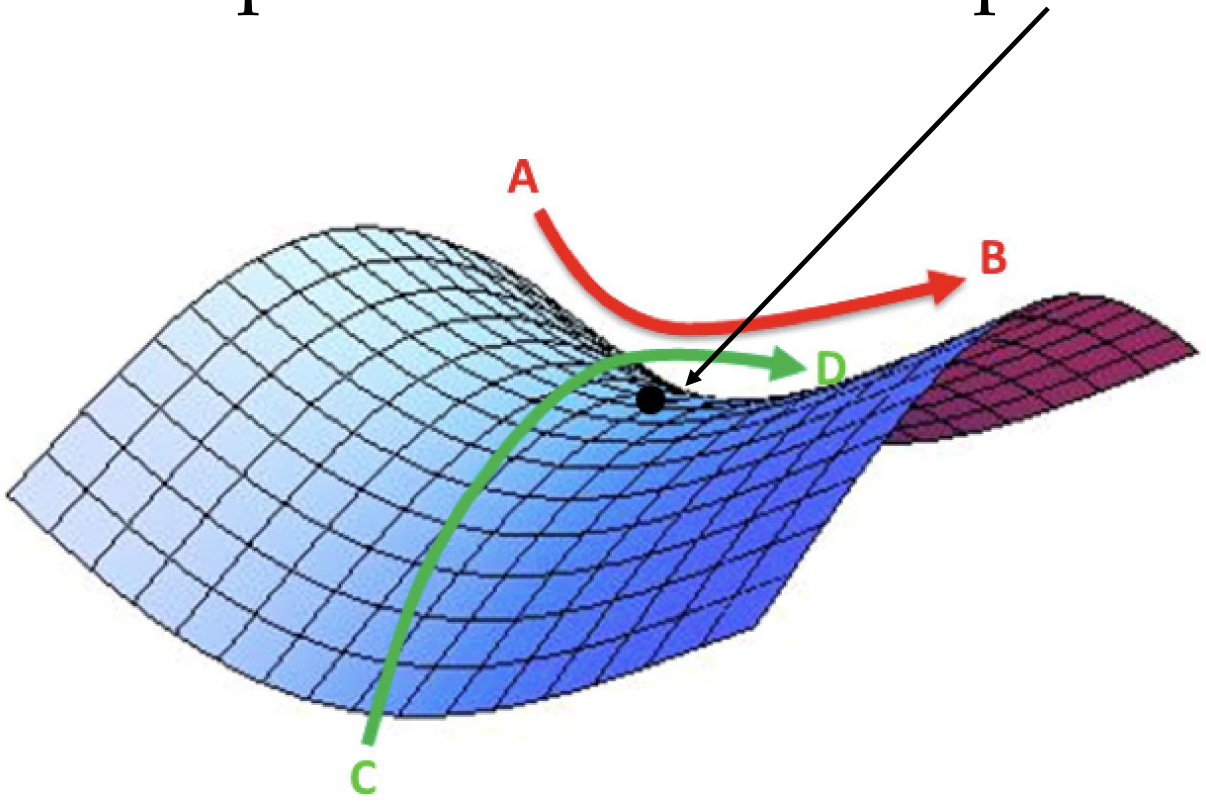
Comparaison avec autres méthodes

- Beaucoup de méthodes ML sont convexes
  - logistic regression
  - linear regression
  - SVM
- Réseaux de neurones sont non-convexes
  - demande un abandon de garanties théoriques
  - performance va dépendre de l'initialisation
  - peur historique des minimums locaux
  - réalisation graduelle que les solutions sont plus des points de selle
    - ratio (points de selle)/(minimum locaux) augmente exponentiellement avec nombre $|\theta|$ de paramètres

p. 278-280 du manuel

Exemple point de selle

- Dérivées partielles nulles au point de selle



https://www.safaribooksonline.com/library/view/fundamentals-of-deep/9781491925607/ch04.html

backprop

- Algorithme qui calcule tous les gradients dans un graphe de calcul
- **N'est pas l'algorithme d'apprentissage!**
- Mais tous les algos d'apprentissage utilisent les gradients calculés par *backprop*
- Basé sur la règle de dérivation en chaîne :

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Forme généralisée :

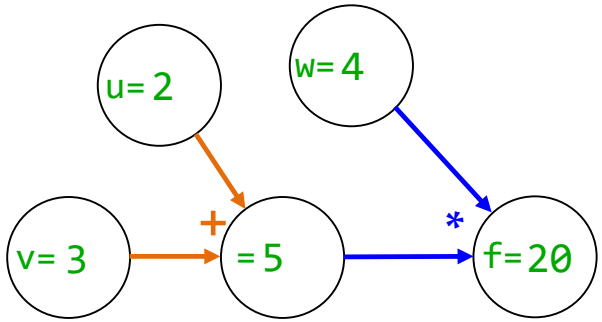
$$\frac{\partial z}{\partial x} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x}$$

Pourquoi étudier la backprop?

- Pour comprendre :
  - le gradients, donc l'apprentissage
  - les cas pathologique
  - les améliorations architecturales :
    - *Batch Normalisation*
    - *ResNet*

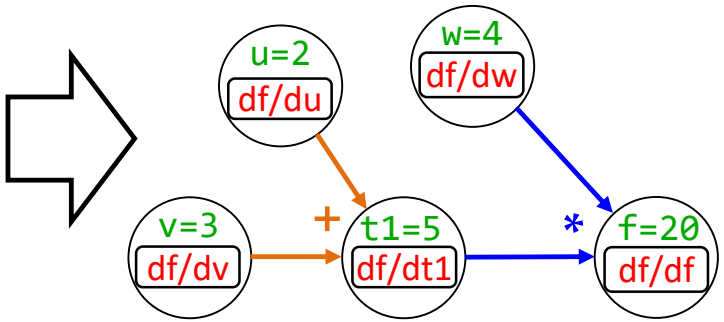
Exemple sur graphe calcul simple

On part d'un graphe de calcul évalué :



$$f=(u+v)w$$

Ajouter une case pour stocker les gradients :

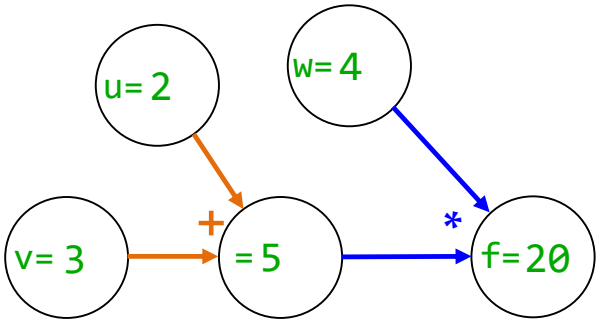


(Rappel : on cherche la sensibilité de la sortie en fonction des variables du graphe)

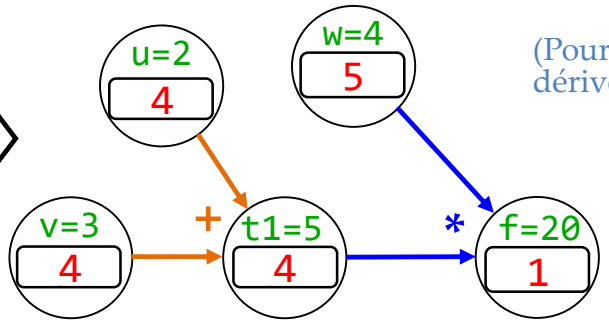
Note : certains placent les fonctions dans le nœuds. Je suis la convention du livre.

Exemple sur graphe calcul simple

Part d'un graphe de calcul évalué :



Ajouter une case pour stocker les gradients :



(Pourquoi /df? On cherche les dérivées partielle p.r. à la sortie, f)

(Rappel : on cherche la sensibilité de la sortie en fonction des variables du graphe)

$$f=(u+v)w$$

$$f=t_1 w, \quad t_1=u+v$$

$$\frac{\partial f}{\partial f}=1$$

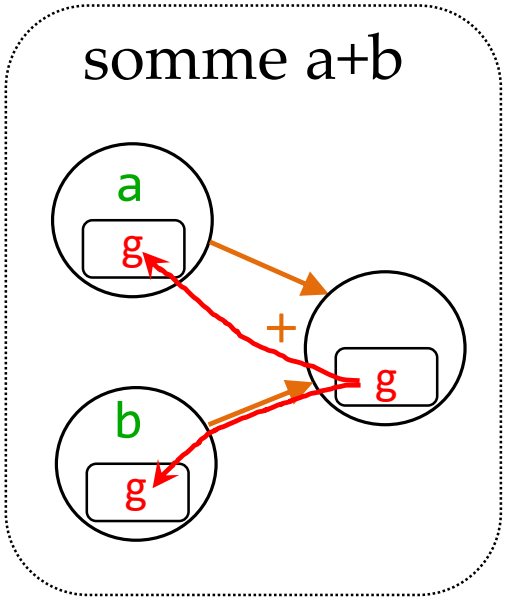
$$\frac{\partial f}{\partial w}=\frac{\partial}{\partial w}(t_1 w)\frac{\partial f}{\partial f}=t_1\cdot 1$$

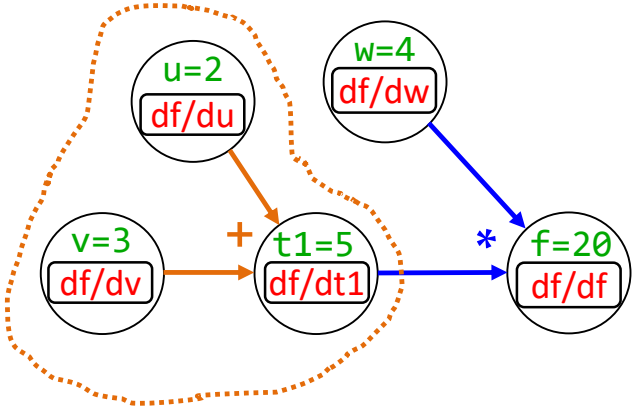
$$\frac{\partial f}{\partial t_1}=\frac{\partial}{\partial t_1}(t_1 w)\frac{\partial f}{\partial f}=w\cdot 1$$

$$\frac{\partial f}{\partial u}=\frac{\partial t_1}{\partial u}\frac{\partial f}{\partial t_1}=1\cdot\frac{\partial f}{\partial t_1}=4$$

$$\frac{\partial f}{\partial v}=\frac{\partial t_1}{\partial v}\frac{\partial f}{\partial t_1}=1\cdot\frac{\partial f}{\partial t_1}=4$$

Tirer des règles de base





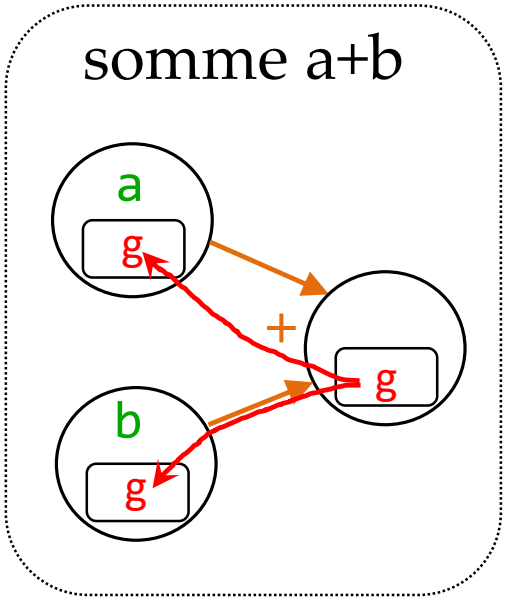
$$\frac{\partial f}{\partial w} = \frac{\partial}{\partial w}(t_1 w)\frac{\partial f}{\partial f} = t_1 \cdot 1$$

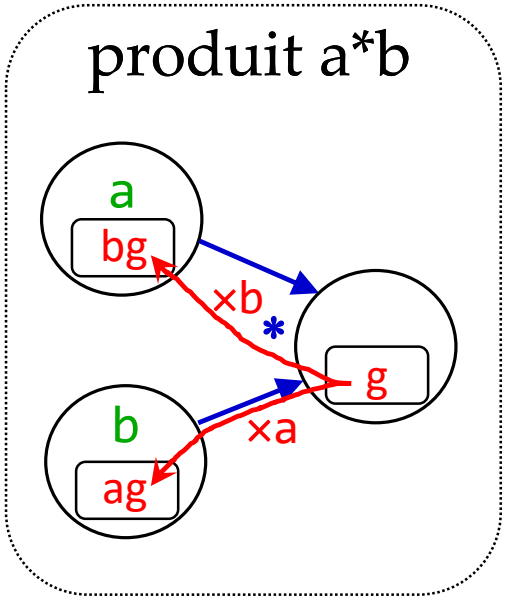
$$\frac{\partial f}{\partial t_1} = \frac{\partial}{\partial t_1}(t_1 w)\frac{\partial f}{\partial f} = w \cdot 1$$

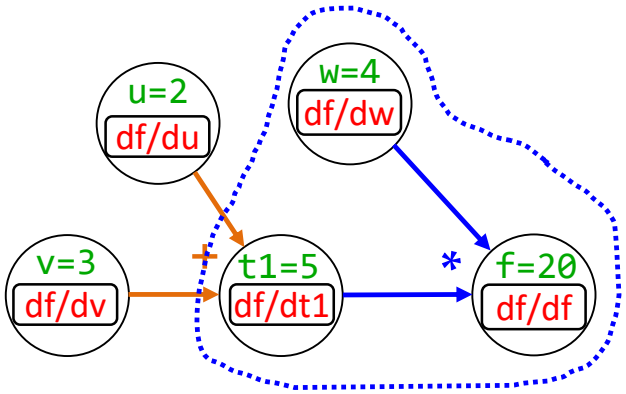
$$\frac{\partial f}{\partial u} = \frac{\partial t_1}{\partial u}\frac{\partial f}{\partial t_1} = 1 \cdot \frac{\partial f}{\partial t_1} = 4$$

$$\frac{\partial f}{\partial v} = \frac{\partial t_1}{\partial v}\frac{\partial f}{\partial t_1} = 1 \cdot \frac{\partial f}{\partial t_1} = 4$$

Tirer des règles de base







$$\frac{\partial f}{\partial w} = \frac{\partial}{\partial w}(t_1 w)\frac{\partial f}{\partial f} = t_1 \frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial t_1} = \frac{\partial}{\partial t_1}(t_1 w)\frac{\partial f}{\partial f} = w \frac{\partial f}{\partial f}$$

$$\frac{\partial f}{\partial u} = \frac{\partial t_1}{\partial u}\frac{\partial f}{\partial t_1} = 1 \cdot \frac{\partial f}{\partial t_1} = 4$$

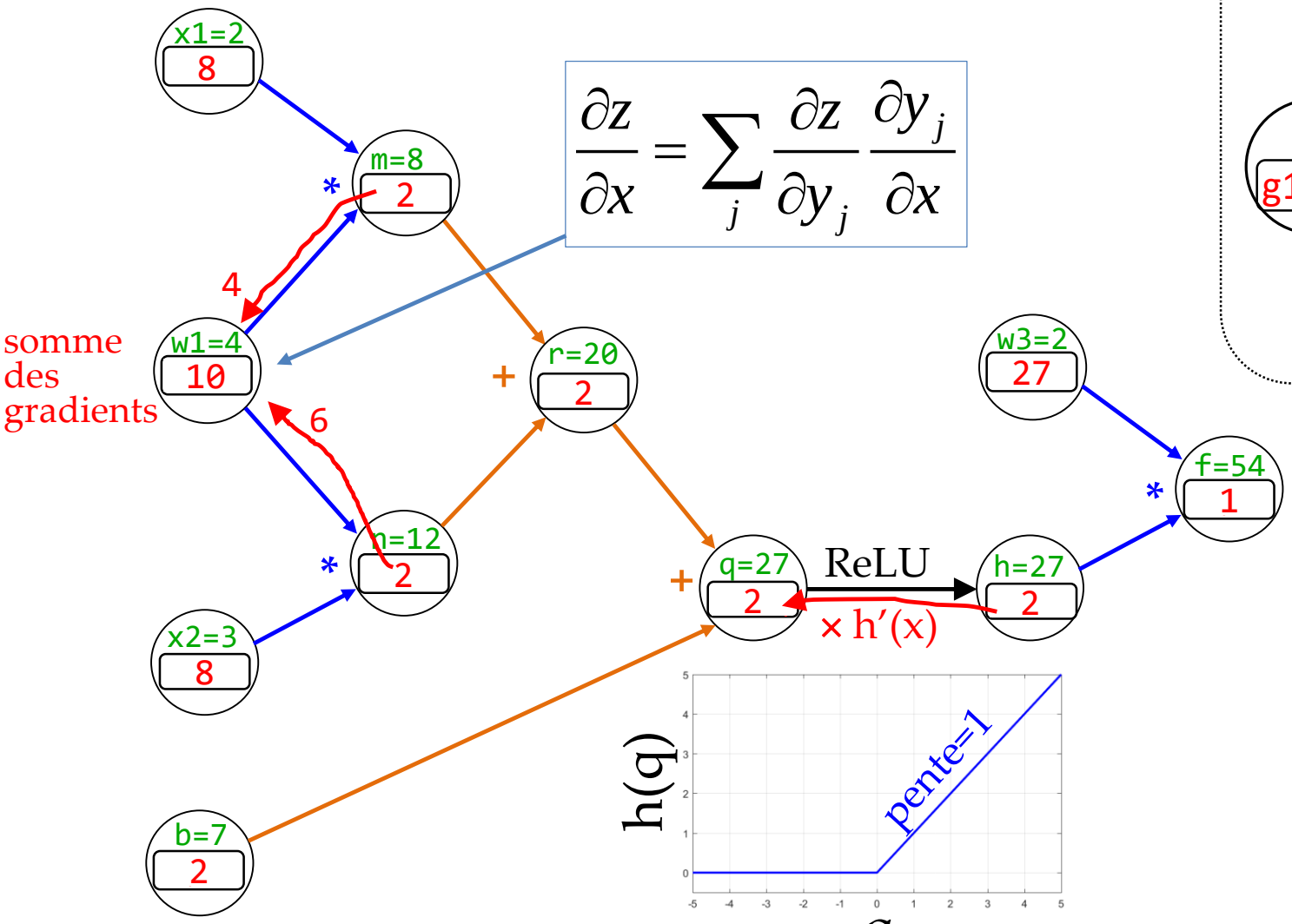
$$\frac{\partial f}{\partial v} = \frac{\partial t_1}{\partial v}\frac{\partial f}{\partial t_1} = 1 \cdot \frac{\partial f}{\partial t_1} = 4$$

Deuxième exemple

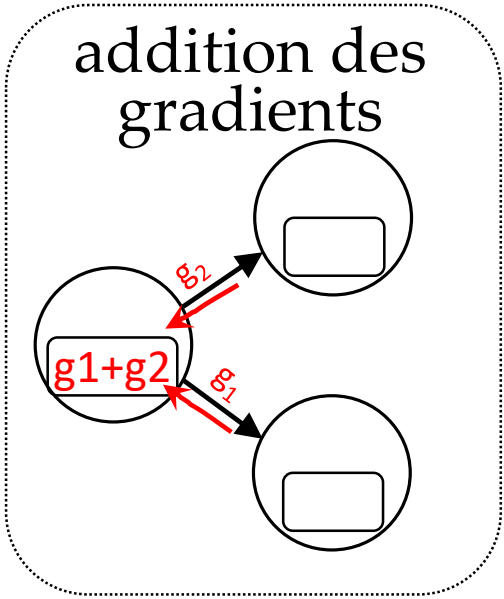
$$f = w_3 \mathrm{ReLU}(w_1 x_1 + w_1 x_2 + b)$$

Deuxième exemple

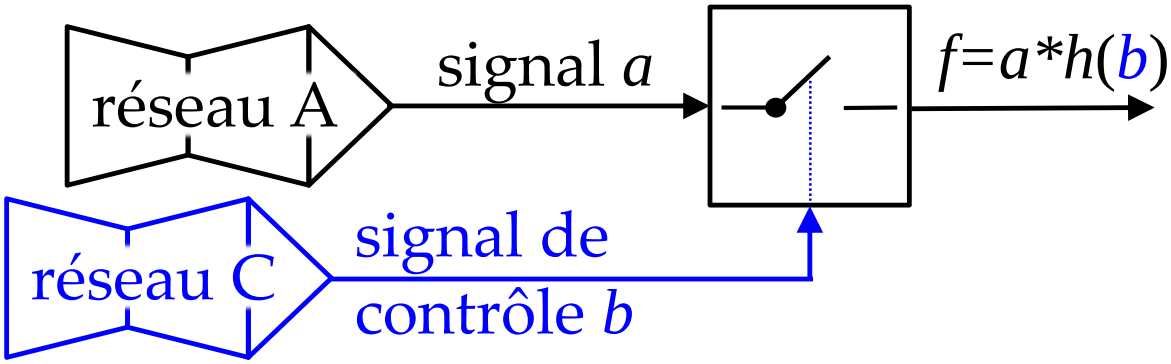
$$f = w_3 \mathrm{ReLU}(w_1 x_1 + w_1 x_2 + b)$$

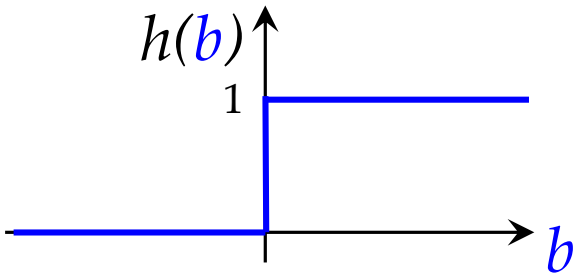


$$\frac{\partial z}{\partial x} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x}$$

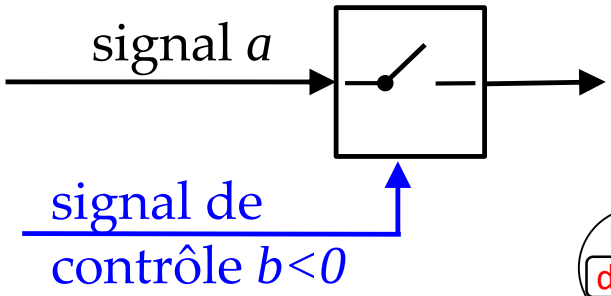


Gradient avec *gate* tout-ou-rien

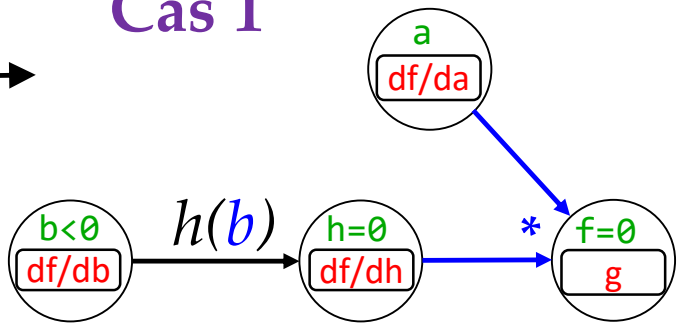


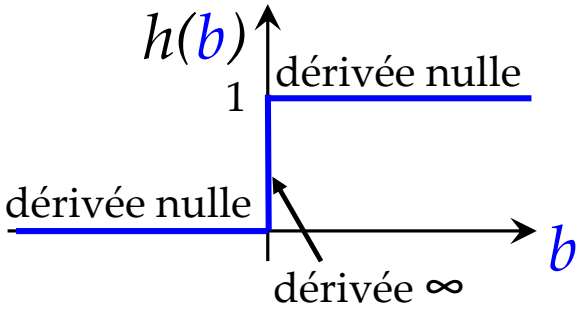


Gradient avec *gate* tout-ou-rien



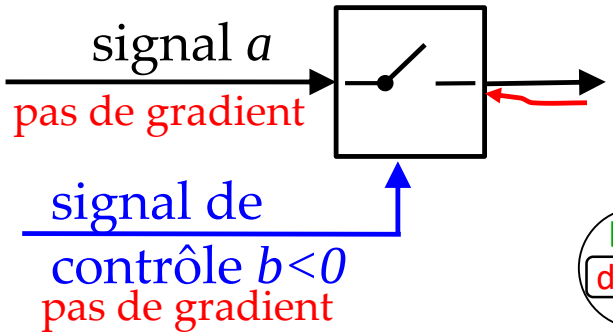
Cas 1



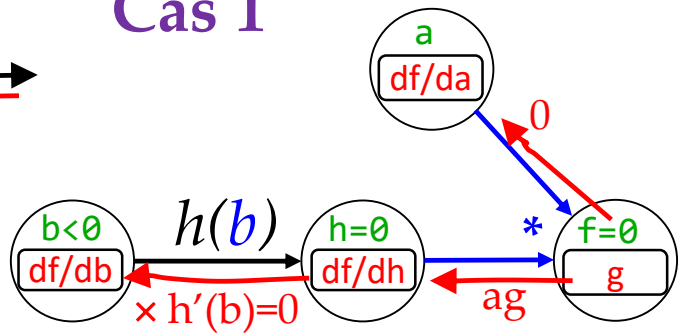


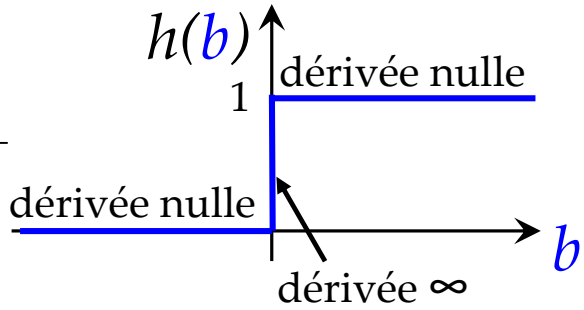
Gradient avec *gate* tout-ou-rien

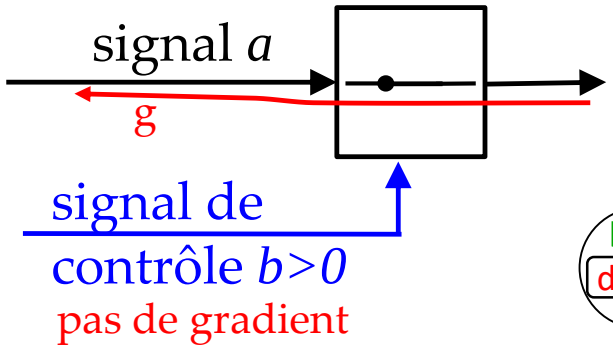
(cas similaire à la ReLU inactive)



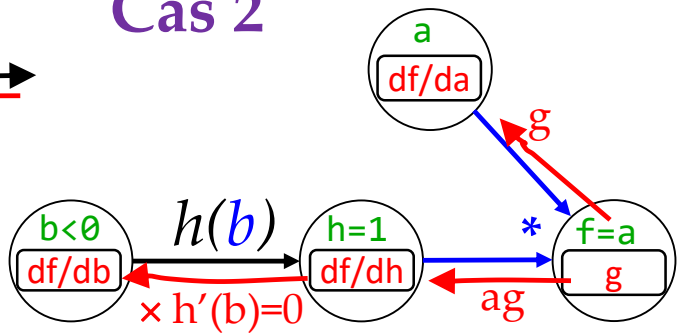
Cas 1

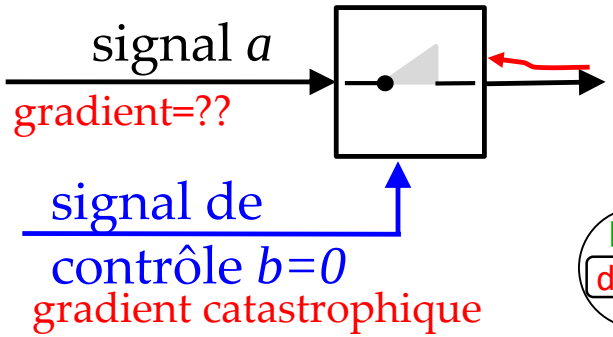




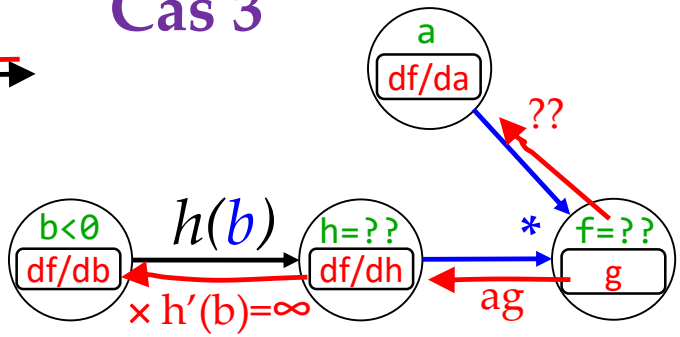


Cas 2

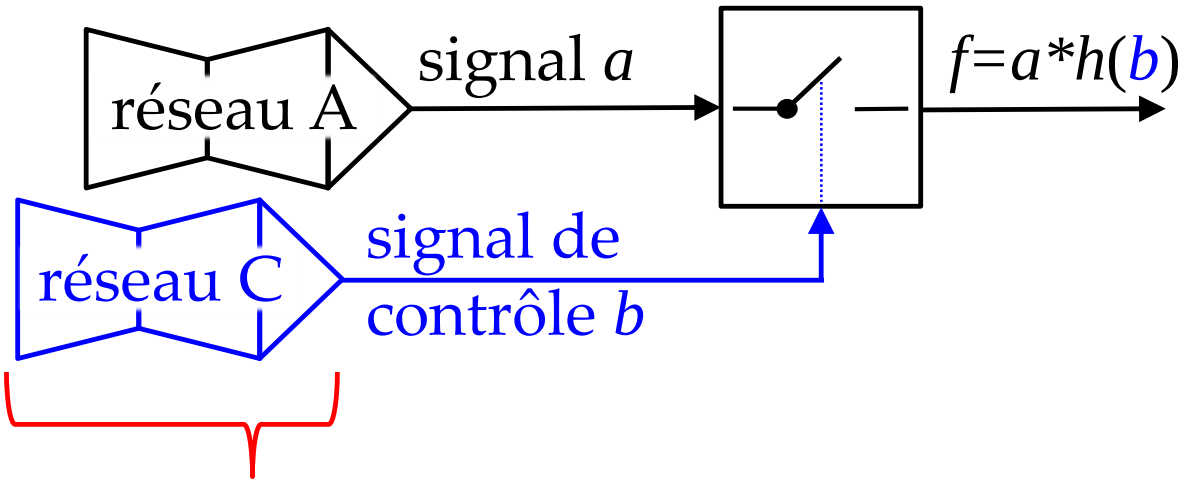




Cas 3

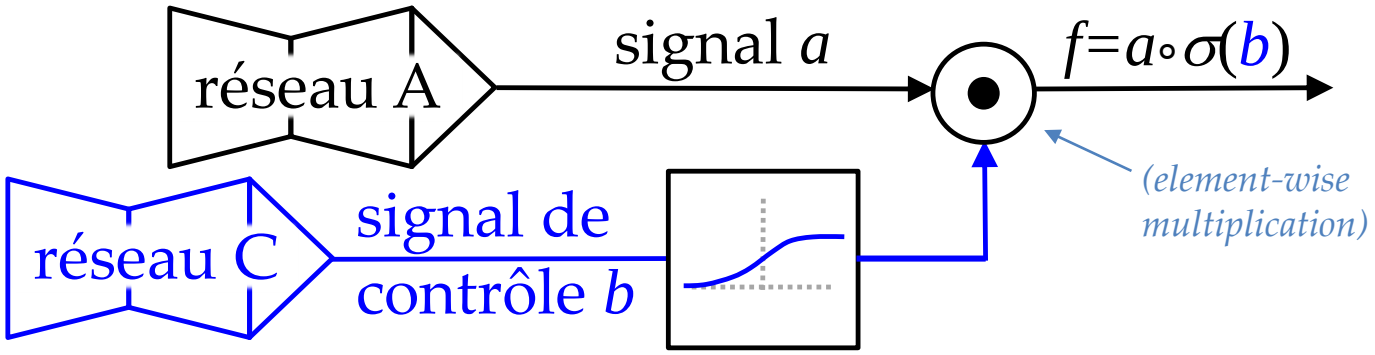


Gradient avec *gate* tout-ou-rien

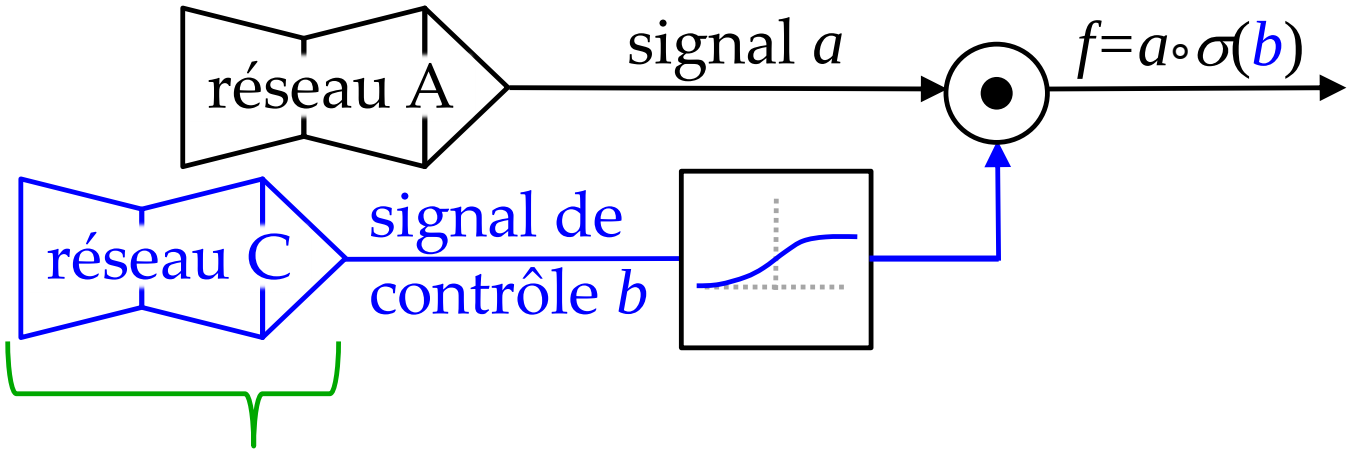


Le réseau C ne va
jamais apprendre ☹

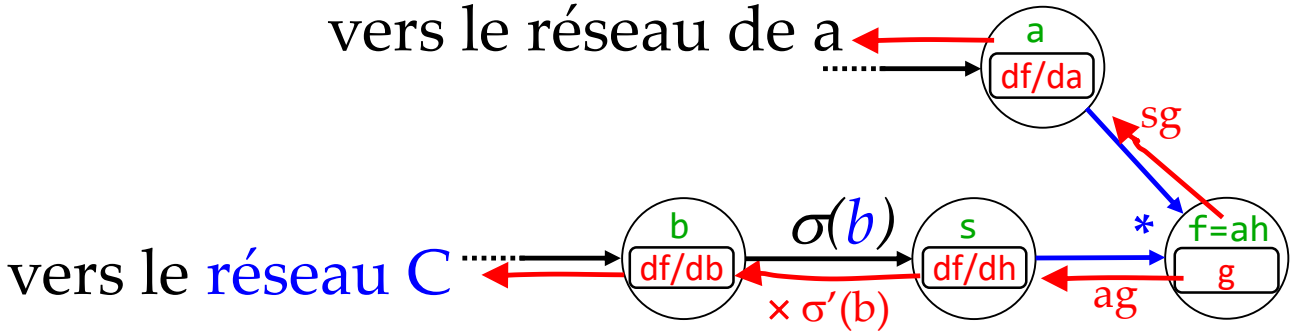
Gradient avec *gate* sigmoïde



Gradient avec *gate* sigmoïde



Ce réseau va apprendre ☺

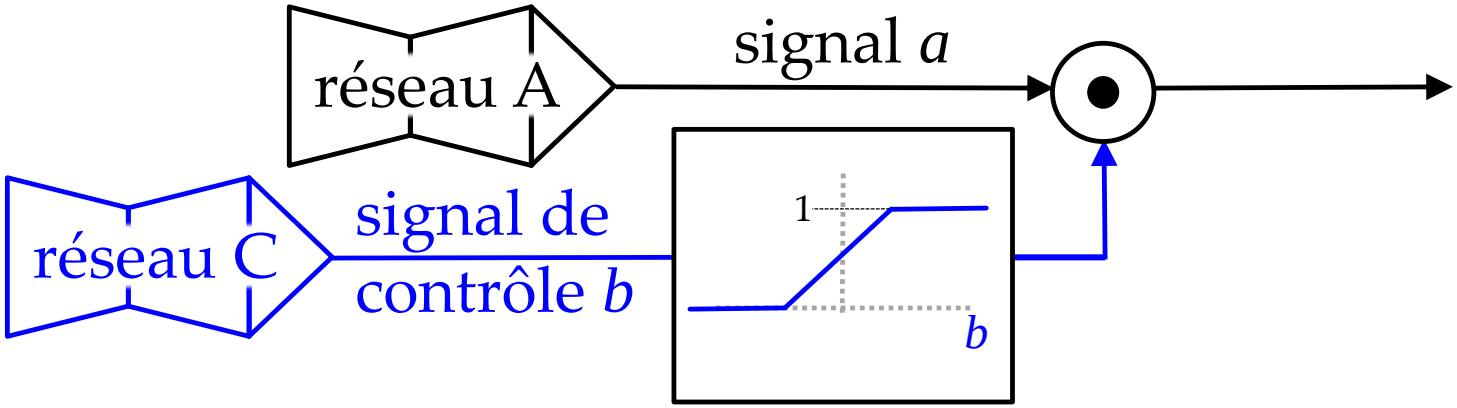


$$\sigma'(b)=\sigma(b)(1-\sigma(b))$$

Importance d'être dérivable/soft end-to-end

Question!

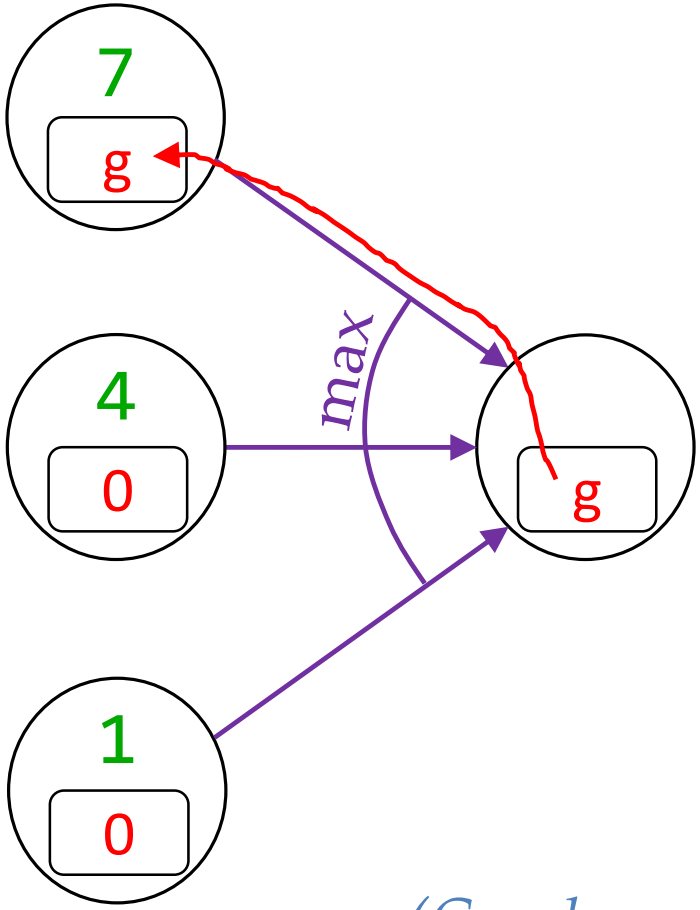
- Que pensez-vous de cette *gate*?



- On conserve encore le signal sortant entre 0 et *a* (c'est bien)
- Mais le réseau C n'apprendra que si le signal *b* est dans la zone linéaire!

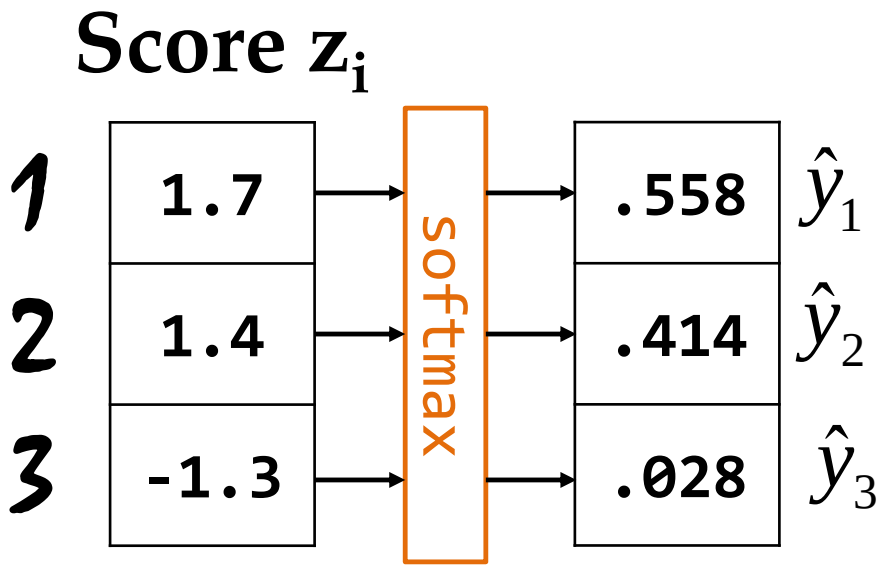
Règle pour max

- Fonction max(z1,z2,z3…)



(Couche max-pooling)

Gradient Softmax

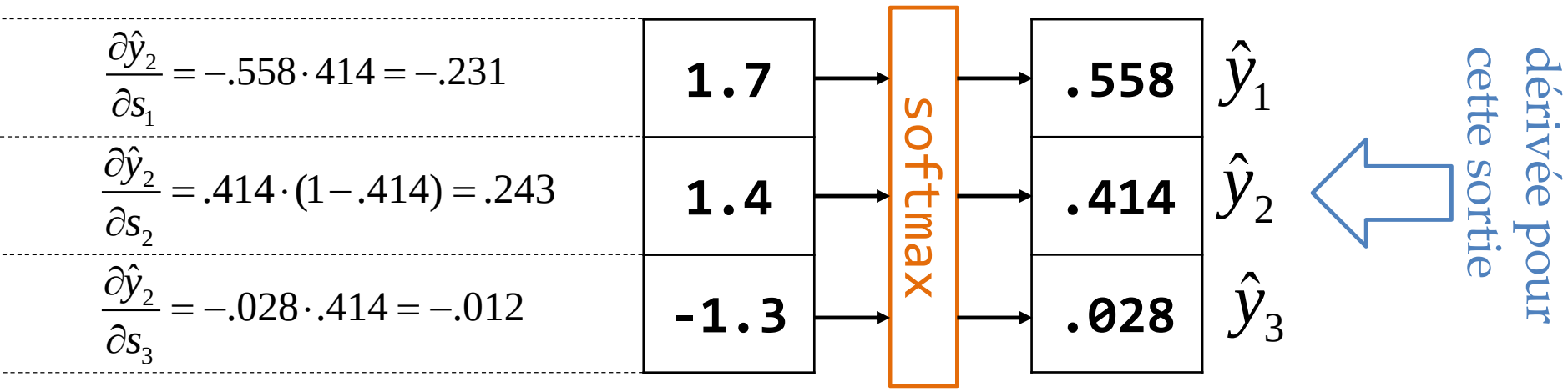


$$\hat{y}_i = \frac{\exp(z_i)}{\sum_{j \in groupe} \exp(z_j)}$$

$$\frac{\partial \hat{y}_j}{\partial s_i} = \begin{cases} \hat{y}_i(1-\hat{y}_i) & i = j \\ -\hat{y}_i \hat{y}_j & i \neq j \end{cases}$$

Voir https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/ pour la preuve.

Exemple calcul gradient Softmax

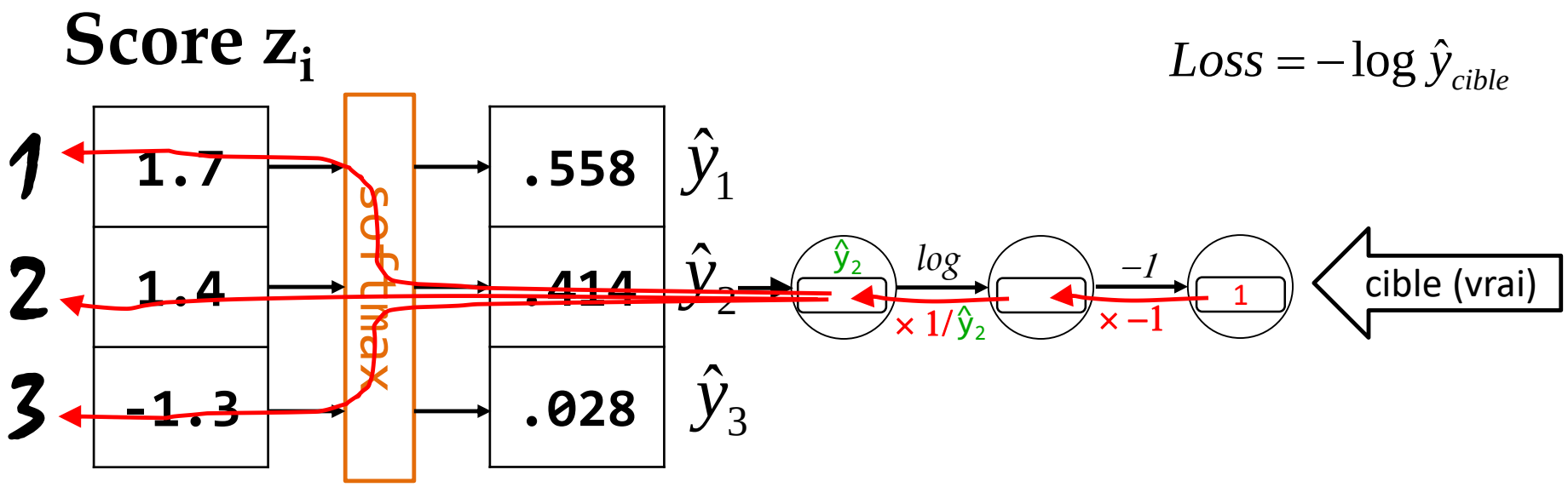


$$\hat{y}_i = \frac{\exp(z_i)}{\sum_{j \in groupe} \exp(z_j)}$$

$$\frac{\partial \hat{y}_j}{\partial s_i} = \begin{cases} \hat{y}_i(1-\hat{y}_i) & i = j \\ -\hat{y}_i \hat{y}_j & i \neq j \end{cases}$$

- Les gradients pour les s_j qui ne correspondent pas à i sont négatifs
- Plus le score s_j est élevé par rapport à s_i, plus la valeur absolue de son gradient le sera

Gradient Softmax + x-entropy loss



$$\hat{y}_i = \frac{\exp(z_i)}{\sum_{j \in groupe} \exp(z_j)}$$

$$\frac{\partial \hat{y}_j}{\partial s_i} = \begin{cases} \hat{y}_i(1-\hat{y}_i) & i = j \\ -\hat{y}_i \hat{y}_j & i \neq j \end{cases}$$

Intuition du livre

- *softmax* et perte avec *log* vont bien ensemble

$$-\log(\hat{y}_{cible}) = -\log\left(\frac{\exp(z_i)}{\sum_{j \in groupe} \exp(z_j)}\right)$$

$$= -\left(\log\left(\exp(z_{cible})\right) - \log\left(\sum_{j \in groupe} \exp(z_j)\right)\right)$$

plus z_{cible} augmente,
plus la perte diminue

$$= \underbrace{-z_{cible}} + \underbrace{\log\left(\sum_{j \in groupe} \exp(z_j)\right)}$$

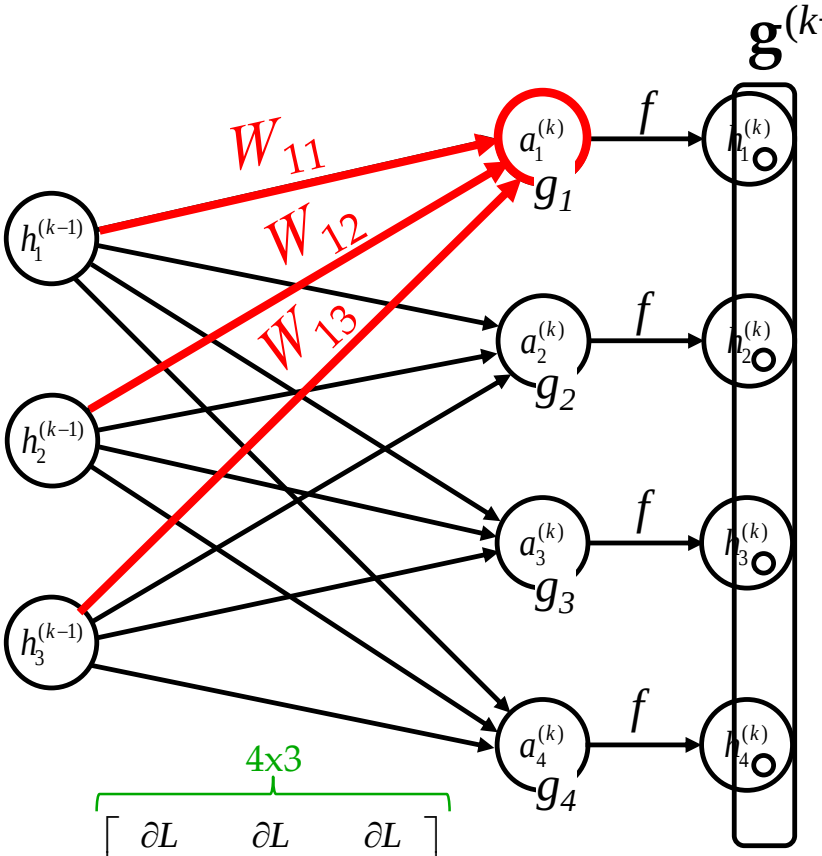
$\approx max(z_j)$
baisser le z_j de la classe
qui est la plus incorrecte

Section 6.2.2.3 du livre, p. 178-182

Notes sur Softmax, X-entropy loss

- Différence entre *multiclass* SVM et x-entr. :
  - La perte MC SVM pourrait atteindre 0, et le training va s'arrêter
  - Pas pour x-entr : les sorties $\hat{y} = f(x)$ ne seront jamais complètement 0 ou 1, algo va juste pousser les scores vers $+\infty$ pour bonne classe et $-\infty$ pour les autres.
- Sortie softmax s'entraîne mal avec une fonction de perte sans *log*, comme $(\bullet)^2$ (p.180)
- Lien avec *lateral inhibition* en biologie (*winner-takes-all*)

Backprop couche MLP sur $\boldsymbol{W}$



$$g = g^{(k+1)} \odot f'(a^{(k)})$$

Gradient $\nabla_W L$ des poids (comment les poids W affectent la perte L)

$$a^{(k)} = \overbrace{W^{(k)}}^{4x3} h^{(k-1)} + b^{(k)}$$

$$a_1^{(k)} = W_{11} h_1^{(k-1)} + W_{12} h_2^{(k-1)} + W_{13} h_3^{(k-1)}$$

$$\frac{\partial L}{\partial W_{11}} = \frac{\partial a_1^{(k)}}{\partial W_{11}} \frac{\partial L}{\partial a_1^{(k)}} = h_1^{(k-1)} g_1$$

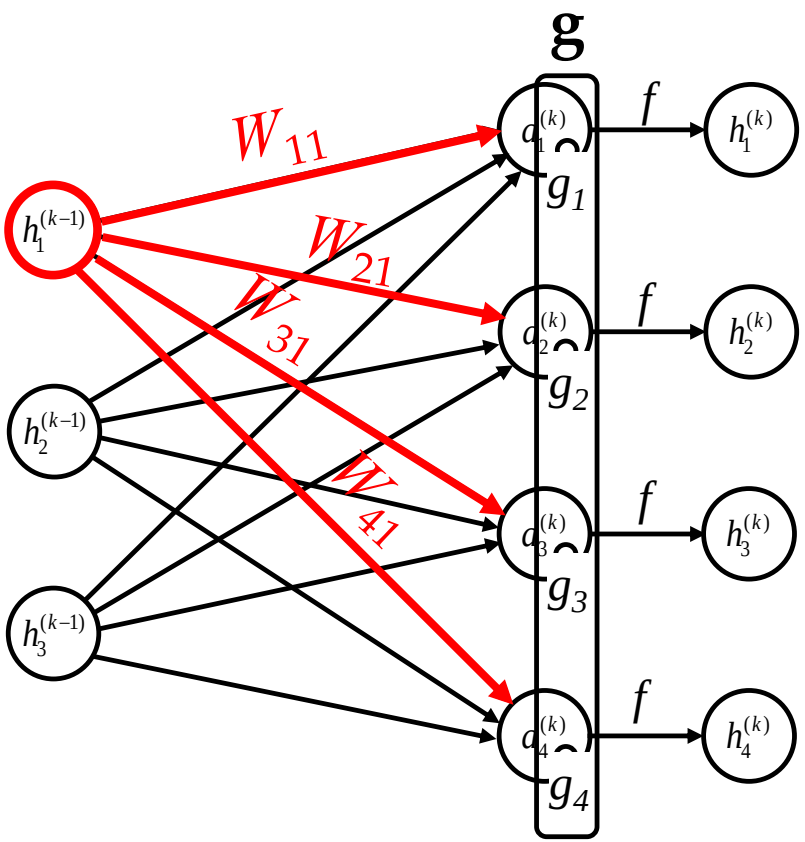
$$\frac{\partial L}{\partial W_{12}} = \frac{\partial a_1^{(k)}}{\partial W_{12}} \frac{\partial L}{\partial a_1^{(k)}} = h_2^{(k-1)} g_1$$

$$\frac{\partial L}{\partial W_{np}} = \frac{\partial a_n^{(k)}}{\partial W_{np}} \frac{\partial L}{\partial a_n^{(k)}} = h_p^{(k-1)} g_n$$

$$\nabla_W L = \overbrace{\begin{bmatrix} \frac{\partial L}{\partial W_{11}} & \frac{\partial L}{\partial W_{12}} & \frac{\partial L}{\partial W_{13}} \\ \frac{\partial L}{\partial W_{21}} & \frac{\partial L}{\partial W_{22}} & \frac{\partial L}{\partial W_{23}} \\ \frac{\partial L}{\partial W_{31}} & \frac{\partial L}{\partial W_{32}} & \frac{\partial L}{\partial W_{33}} \\ \frac{\partial L}{\partial W_{41}} & \frac{\partial L}{\partial W_{42}} & \frac{\partial L}{\partial W_{43}} \end{bmatrix}}^{4x3} = \begin{bmatrix} g_1 h_1^{(k-1)} & g_1 h_2^{(k-1)} & g_1 h_3^{(k-1)} \\ g_2 h_1^{(k-1)} & g_2 h_2^{(k-1)} & g_2 h_3^{(k-1)} \\ g_3 h_1^{(k-1)} & g_3 h_2^{(k-1)} & g_3 h_3^{(k-1)} \\ g_4 h_1^{(k-1)} & g_4 h_2^{(k-1)} & g_4 h_3^{(k-1)} \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix} \begin{bmatrix} h_1^{(k-1)} & h_2^{(k-1)} & h_3^{(k-1)} \end{bmatrix} = \boxed{g h^{(k-1)T}}$$

rapide à calculer !

Backprop couche MLP sur $h^{(k-1)}$



Propager gradient g vers $h^{(k-1)}$

$$a^{(k)} = \overbrace{W^{(k)}}^{4x3} h^{(k-1)} + b^{(k)}$$

$$a_1^{(k)} = W_{11} h_1^{(k-1)} + W_{12} h_2^{(k-1)} + W_{13} h_3^{(k-1)}$$

$$a_2^{(k)} = W_{21} h_1^{(k-1)} + W_{22} h_2^{(k-1)} + W_{23} h_3^{(k-1)}$$

$$\cdots$$

$$a_4^{(k)} = W_{41} h_1^{(k-1)} + W_{42} h_2^{(k-1)} + W_{43} h_3^{(k-1)}$$

W_{j1}

$$\frac{\partial L}{\partial h_i^{(k-1)}} = \sum_j \frac{\partial a_j^{(k)}}{\partial h_i^{(k-1)}} \overbrace{\frac{\partial L}{\partial a_j^{(k)}}}^{g_j} = \sum_j W_{ji} g_j$$

= produit scalaire entre $W_{j1}{}^T$ et $\mathbf{g}$

Forme de l'équation (6.49) du livre

$$\nabla_{h_i^{(k-1)}} L : g \leftarrow W^T g$$

rapide à calculer !

"Gradient check"

- Dérivée numérique (lente) :

$$\frac{\partial f}{\partial \theta_i} \approx \frac{f(\theta_1, ..., \theta_i + \varepsilon, ...) - f(\theta_1, ..., \theta_i - \varepsilon, ...)}{2\varepsilon}$$

- Dérivée analytique est rapide, mais facile de se tromper
- Vérifie l'implémentation de la dérivée analytique avec la dérivée numérique : *gradient check*
- Dans PyTorch, : **`autograd`** calculera automatiquement les gradients pour vous. Mais pas à l'examen ;)

Conclusion

- Utiliser fonction de perte pour :
  - quantifier l'accomplissement de la tâche
  - permettre de faire l'entraînement
  - SVM multiclass et cross-entropy pour classification
- Combiner Softmax et cross-entropy est naturel (*log* défait les *exp*)
- *Backpropagation* du gradient permet de trouver toutes les dérivées partielles
  - permet l'optimisation par descente de gradient
  - facile à calculer à partir du graphe de calcul
  - importance d'avoir des *gates* dites *soft*