BACKPROPAGATION

Jérémie Cabessa Laboratoire DAVID, UVSQ

CHAIN RULE

CHAIN RULE

•00000

On rappelle le théorème des fonctions composées (chain rule).

$$\mathbb{R} \xrightarrow{f} \mathbb{R} \xrightarrow{g} \mathbb{R}$$

$$x \xrightarrow{f} y = f(x) \xrightarrow{g} z = g(y)$$

$$= g(f(x))$$

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

•00000

On rappelle le théorème des fonctions composées (chain rule).

Soient les fonctions suivantes:

$$\mathbb{R} \xrightarrow{f} \mathbb{R} \xrightarrow{g} \mathbb{R}$$

$$x \xrightarrow{f} y = f(x) \xrightarrow{g} z = g(y)$$

$$= g(f(x))$$

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

CHAIN RULE

•00000

On rappelle le théorème des fonctions composées (chain rule).

Soient les fonctions suivantes:

$$\mathbb{R} \xrightarrow{f} \mathbb{R} \xrightarrow{g} \mathbb{R}$$

$$x \xrightarrow{f} y = f(x) \xrightarrow{g} z = g(y)$$

$$= g(f(x))$$

alors on a:

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

CHAIN RULE

000000

Exemple:

$$\frac{\partial z}{\partial x} = \frac{\partial \left[5(x^2+1)\right]}{\partial x} = 10x = 5 \cdot 2x = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

CHAIN RULE

000000

Exemple:

$$\mathbb{R} \xrightarrow{f} \mathbb{R} \xrightarrow{g} \mathbb{R}$$

$$x \xrightarrow{f} y = x^2 + 1 \xrightarrow{g} z = 5y$$

$$= 5(x^2 + 1)$$

On a:

$$\frac{\partial z}{\partial x} \ = \ \frac{\partial \left[5(x^2 + 1) \right]}{\partial x} \ = \ 10x \ = \ 5 \cdot 2x \ = \ \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

000000

Généralisation multidimensionnelle

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^n \frac{\partial z}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_i}, \quad i = 1, \dots, m$$

CHAIN RULE

000000

Généralisation multidimensionnelle

Soient les fonctions suivantes:

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^n \frac{\partial z}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_i}, \quad i = 1, \dots, n$$

CHAIN RULE

000000

Généralisation multidimensionnelle

Soient les fonctions suivantes:

$$\mathbb{R}^m \stackrel{f}{\longrightarrow} \mathbb{R}^n \stackrel{g}{\longrightarrow} \mathbb{R}$$
 $x \stackrel{f}{\longmapsto} y = f(x) \stackrel{g}{\longmapsto} z = g(y)$
 $= g(f(x))$

alors on a:

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^n \frac{\partial z}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_i}, \quad i = 1, \dots, m$$

Exemple:

$$\mathbb{R}^{2} \xrightarrow{f} \mathbb{R}^{3} \xrightarrow{g} \mathbb{R}$$

$$\begin{pmatrix} x_{1} \\ x_{2} \end{pmatrix} \xrightarrow{f} \begin{pmatrix} y_{1} \\ y_{2} \\ y_{3} \end{pmatrix} = \begin{pmatrix} x_{1}x_{2} \\ x_{1}^{2} \\ x_{2}^{3} \end{pmatrix} \xrightarrow{g} \qquad z = y_{1}y_{2} + y_{3}$$

$$= (x_{1}x_{2})x_{1}^{2} + x_{2}^{3}$$

$$\begin{array}{cccc} \frac{\partial z}{\partial x_1} &=& \frac{\partial \left[(x_1x_2)x_1^2 + x_2^3 \right]}{\partial x_1} &=& x_1^2x_2 + x_1x_22x_1 + 0 \\ &&=& \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_1} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_1} + \frac{\partial z}{\partial y_3} \cdot \frac{\partial y_3}{\partial x_1} \\ \\ \frac{\partial z}{\partial x_2} &=& \frac{\partial \left[(x_1x_2)x_1^2 + x_2^3 \right]}{\partial x_2} &=& x_1^2x_1 + 3x_2^2 + 0 \\ &&=& \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_2} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_2} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_3}{\partial x_2} \end{array}$$

000000

Exemple:

$$\mathbb{R}^{2} \xrightarrow{f} \mathbb{R}^{3} \xrightarrow{g} \mathbb{R}$$

$$\begin{pmatrix} x_{1} \\ x_{2} \end{pmatrix} \xrightarrow{f} \begin{pmatrix} y_{1} \\ y_{2} \\ y_{3} \end{pmatrix} = \begin{pmatrix} x_{1}x_{2} \\ x_{1}^{2} \\ x_{2}^{3} \end{pmatrix} \xrightarrow{g} \qquad z = y_{1}y_{2} + y_{3}$$

$$= (x_{1}x_{2})x_{1}^{2} + x_{2}^{3}$$

On a:

$$\frac{\partial z}{\partial x_1} = \frac{\partial \left[(x_1 x_2) x_1^2 + x_2^3 \right]}{\partial x_1} = x_1^2 x_2 + x_1 x_2 2 x_1 + 0$$

$$= \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_1} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_1} + \frac{\partial z}{\partial y_3} \cdot \frac{\partial y_3}{\partial x_1}$$

$$\frac{\partial z}{\partial x_2} = \frac{\partial \left[(x_1 x_2) x_1^2 + x_2^3 \right]}{\partial x_2} = x_1^2 x_1 + 3 x_2^2 + 0$$

$$= \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_2} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_3} + \frac{\partial z}{\partial y_3} \cdot \frac{\partial y_3}{\partial x_3}$$

CHAIN RULE

000000

Exemple:

$$\mathbb{R}^{2} \xrightarrow{f} \mathbb{R}^{3} \xrightarrow{g} \mathbb{R}$$

$$\begin{pmatrix} x_{1} \\ x_{2} \end{pmatrix} \xrightarrow{f} \begin{pmatrix} y_{1} \\ y_{2} \\ y_{3} \end{pmatrix} = \begin{pmatrix} x_{1}x_{2} \\ x_{1}^{2} \\ x_{2}^{3} \end{pmatrix} \xrightarrow{g} \qquad z = y_{1}y_{2} + y_{3}$$

$$= (x_{1}x_{2})x_{1}^{2} + x_{2}^{3}$$

On a:

$$\frac{\partial z}{\partial x_1} = \frac{\partial \left[(x_1 x_2) x_1^2 + x_2^3 \right]}{\partial x_1} = x_1^2 x_2 + x_1 x_2 2 x_1 + 0$$

$$= \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_1} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_1} + \frac{\partial z}{\partial y_3} \cdot \frac{\partial y_3}{\partial x_1}$$

$$\frac{\partial z}{\partial x_2} = \frac{\partial \left[(x_1 x_2) x_1^2 + x_2^3 \right]}{\partial x_2} = x_1^2 x_1 + 3 x_2^2 + 0$$

$$= \frac{\partial z}{\partial y_1} \cdot \frac{\partial y_1}{\partial x_2} + \frac{\partial z}{\partial y_2} \cdot \frac{\partial y_2}{\partial x_2} + \frac{\partial z}{\partial y_3} \cdot \frac{\partial y_3}{\partial x_2}$$

CHAIN RULE

000000

Formulation vectorielle

$$\mathbb{R}^{m} \xrightarrow{f} \mathbb{R}^{n} \xrightarrow{g} \mathbb{R}$$
 $\mathbf{x} \xrightarrow{f} \mathbf{y} = f(\mathbf{x}) \xrightarrow{g} z = g(\mathbf{y})$
 $= g(f(\mathbf{x}))$

$$abla_{m{x}}z = \left[[
abla_{m{y}}z]^T \left[rac{\partial m{y}}{\partial m{x}}
ight]
ight]^T = m{J}_f^T
abla_{m{y}}z$$

$$\begin{pmatrix} \frac{\partial z}{\partial x_1} \\ \vdots \\ \frac{\partial z}{\partial x_m} \end{pmatrix} = \begin{bmatrix} \begin{pmatrix} \frac{\partial z}{\partial y_1} & \cdots & \frac{\partial z}{\partial y_n} \end{pmatrix} \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_m} \end{pmatrix} \end{bmatrix}^T$$

<ロ > ← □

CHAIN RULE

000000

Formulation vectorielle

Soient $\nabla_{x}z$ et le gradient de z par rapport à x, $\nabla_{y}z$ et le gradient de z par rapport à ${m y}$, et ${m J}_f:=\left[rac{\partial {m y}}{\partial {m x}}
ight]$ le jacobien de la fonction f:

$$abla_{m{x}}z = \left[[
abla_{m{y}}z]^T \left[rac{\partial m{y}}{\partial m{x}}
ight]
ight]^T = m{J}_f^T
abla_{m{y}}z$$

$$\begin{pmatrix} \frac{\partial z}{\partial x_1} \\ \vdots \\ \frac{\partial z}{\partial x_m} \end{pmatrix} = \begin{bmatrix} \begin{pmatrix} \frac{\partial z}{\partial y_1} & \cdots & \frac{\partial z}{\partial y_n} \end{pmatrix} \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_m} \end{pmatrix} \end{bmatrix}^T$$

000000

Formulation vectorielle

$$\begin{array}{cccc} \mathbb{R}^m & \xrightarrow{f} & \mathbb{R}^n & \xrightarrow{g} & \mathbb{R} \\ \boldsymbol{x} & \stackrel{f}{\longmapsto} & \boldsymbol{y} = f(\boldsymbol{x}) & \stackrel{g}{\longmapsto} & z = g(\boldsymbol{y}) \\ & & = g\left(f(\boldsymbol{x})\right) \end{array}$$

Soient $\nabla_{x}z$ et le gradient de z par rapport à x, $\nabla_{y}z$ et le gradient de z par rapport à y, et $J_{f}:=\left[\frac{\partial y}{\partial x}\right]$ le jacobien de la fonction f:

$$abla_{m{x}}z = \left[\left[
abla_{m{y}}z
ight]^T \left[rac{\partial m{y}}{\partial m{x}}
ight]
ight]^T = m{J}_f^T
abla_{m{y}}z$$

$$\begin{pmatrix} \frac{\partial z}{\partial x_1} \\ \vdots \\ \frac{\partial z}{\partial x_m} \end{pmatrix} = \begin{bmatrix} \begin{pmatrix} \frac{\partial z}{\partial y_1} & \cdots & \frac{\partial z}{\partial y_n} \end{pmatrix} \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_m} \end{pmatrix} \end{bmatrix}^T$$

CHAIN RULE

000000

Formulation vectorielle

$$\mathbb{R}^m \stackrel{f}{\longrightarrow} \mathbb{R}^n \stackrel{g}{\longrightarrow} \mathbb{R}$$
 $x \stackrel{f}{\longmapsto} y = f(x) \stackrel{g}{\longmapsto} z = g(y)$
 $= g(f(x))$

Soient $\nabla_x z$ et le gradient de z par rapport à x, $\nabla_y z$ et le gradient de z par rapport à y, et $J_f := \left\lceil \frac{\partial y}{\partial x} \right\rceil$ le jacobien de la fonction f:

$$abla_{m{x}}z = \left[\left[
abla_{m{y}}z
ight]^T \left[rac{\partial m{y}}{\partial m{x}}
ight]
ight]^T = m{J}_f^T
abla_{m{y}}z$$

$$\begin{pmatrix} \frac{\partial z}{\partial x_1} \\ \vdots \\ \frac{\partial z}{\partial x_m} \end{pmatrix} = \begin{bmatrix} \begin{pmatrix} \frac{\partial z}{\partial y_1} & \cdots & \frac{\partial z}{\partial y_n} \end{pmatrix} \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_m} \end{pmatrix} \end{bmatrix}^T$$

Exemple (suite):

$$\mathbb{R}^{m} \xrightarrow{f} \mathbb{R}^{n} \xrightarrow{g} \mathbb{R}$$

$$\begin{pmatrix} x_{1} \\ x_{2} \end{pmatrix} \xrightarrow{f} \begin{pmatrix} y_{1} \\ y_{2} \\ y_{3} \end{pmatrix} = \begin{pmatrix} x_{1}x_{2} \\ x_{1}^{2} \\ x_{2}^{3} \end{pmatrix} \xrightarrow{g} z = y_{1}y_{2} + y_{3}$$

$$= (x_{1}x_{2})x_{1}^{2} + x_{2}^{3}$$

$$\nabla_{\boldsymbol{x}}z = \begin{pmatrix} \frac{\partial z}{\partial x_1} \\ \frac{\partial z}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 3x_1^2x_2 \\ x_1^3 + 3x_2^2 \end{pmatrix} = \begin{bmatrix} (y_2 \quad y_1 \quad 1) \begin{pmatrix} x_2 & x_1 \\ 2x_1 & 0 \\ 0 & 3x_2^2 \end{pmatrix} \end{bmatrix}^T = \begin{bmatrix} [\nabla_{\boldsymbol{y}}z]^T \begin{bmatrix} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \end{bmatrix} \end{bmatrix}^T$$

Exemple (suite):

$$\mathbb{R}^{m} \xrightarrow{f} \mathbb{R}^{n} \xrightarrow{g} \mathbb{R}$$

$$\begin{pmatrix} x_{1} \\ x_{2} \end{pmatrix} \xrightarrow{f} \begin{pmatrix} y_{1} \\ y_{2} \\ y_{3} \end{pmatrix} = \begin{pmatrix} x_{1}x_{2} \\ x_{1}^{2} \\ x_{2}^{3} \end{pmatrix} \xrightarrow{g} z = y_{1}y_{2} + y_{3}$$

$$= (x_{1}x_{2})x_{1}^{2} + x_{2}^{3}$$

On a:

$$\nabla_{\boldsymbol{x}}z = \begin{pmatrix} \frac{\partial z}{\partial x_1} \\ \frac{\partial z}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 3x_1^2x_2 \\ x_1^3 + 3x_2^2 \end{pmatrix} = \begin{bmatrix} (y_2 \quad y_1 \quad 1) \begin{pmatrix} x_2 & x_1 \\ 2x_1 & 0 \\ 0 & 3x_2^2 \end{pmatrix} \end{bmatrix}^T = \begin{bmatrix} [\nabla_{\boldsymbol{y}}z]^T \begin{bmatrix} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \end{bmatrix} \end{bmatrix}^T$$

- lacksquare Soit $S=\{(oldsymbol{x_i},oldsymbol{y_i})\in\mathbb{R}^{d_1} imes\mathbb{R}^{d_2}:i=1,\ldots,N\}$ un dataset.

$$\mathbf{\Theta} := \left\{ \left(\mathbf{W}^{[l]}, \mathbf{b}^{[l]} \right) : l = 1, \dots, L \right\}$$

$$egin{cases} m{a^{[0]}} &= m{x} \ m{z^{[l]}} &= m{W^{[l]}}m{a^{[l-1]}} + m{b^{[l]}}, \ m{a^{[l]}} &= m{\sigma\left(m{z^{[l]}}
ight)} \end{cases}$$

CHAIN BILLE

NEURAL NETWORK AS A FUNCTION

- $lackbox{
 ightharpoonup}$ Soit $S=\left\{(oldsymbol{x_i},oldsymbol{y_i})\in\mathbb{R}^{d_1} imes\mathbb{R}^{d_2}:i=1,\ldots,N
 ight\}$ un dataset.
- Soit \mathcal{N}_{Θ} un réseau de neurones (MLP) à L couches donné par les paramètres (poids et biais)

$$\mathbf{\Theta} := \left\{ \left(\mathbf{W}^{[l]}, \mathbf{b}^{[l]} \right) : l = 1, \dots, L \right\}$$

et par la dynamique

$$egin{cases} oldsymbol{a^{[0]}} &= oldsymbol{x} \ oldsymbol{z^{[l]}} &= oldsymbol{W^{[l]}} oldsymbol{a^{[l-1]}} + oldsymbol{b^{[l]}}, \ oldsymbol{a^{[l]}} &= oldsymbol{\sigma}\left(oldsymbol{z^{[l]}}
ight) \end{cases}$$

NEURAL NETWORK AS A FUNCTION

ightharpoonup Remarque: le réseau \mathcal{N}_{Θ} peut-être naturellement associé à la fonction

$$egin{array}{lll} f_{m{\Theta}}: \mathbb{R}^{d_1} & \longrightarrow & \mathbb{R}^{d_2} \ & m{x} & \longmapsto & f_{m{\Theta}}(m{x}) := m{a}^{[m{L}]} \end{array}$$

- $ightharpoonup f_{\Theta}(x)$ est la *prédiction* (output) de \mathcal{N}_{Θ} associée à l'input x.
- ightharpoonup Chaque jeu de paramètres Θ donne lieu à une fonction f_{Θ} différente.

Remarque: le réseau \mathcal{N}_{Θ} peut-être naturellement associé à la fonction

$$egin{array}{lll} f_{m{\Theta}}: \mathbb{R}^{d_1} & \longrightarrow & \mathbb{R}^{d_2} \ x & \longmapsto & f_{m{\Theta}}(x) := m{a^{[L]}} \end{array}$$

- $ightharpoonup f_{\Theta}(x)$ est la prédiction (output) de \mathcal{N}_{Θ} associée à l'input x.

NEURAL NETWORK AS A FUNCTION

ightharpoonup Remarque: le réseau \mathcal{N}_{Θ} peut-être naturellement associé à la fonction

$$egin{array}{lll} f_{m{\Theta}}: \mathbb{R}^{d_1} & \longrightarrow & \mathbb{R}^{d_2} \ x & \longmapsto & f_{m{\Theta}}(x) := m{a}^{[L]} \end{array}$$

- lacksquare $f_{m{\Theta}}(x)$ est la *prédiction* (output) de $\mathcal{N}_{m{\Theta}}$ associée à l'input x.
- lacktriangle Chaque jeu de paramètres Θ donne lieu à une fonction f_{Θ} différente.

Soit une fonction de coût (cost or loss function) qui mesure l'erreur entre la prédiction \hat{y}_i et la réalité y_i :

$$egin{array}{lll} \ell: \mathbb{R}^{d_2} imes \mathbb{R}^{d_2} & \longrightarrow & \mathbb{R} \ (\hat{m{y_i}}, m{y_i}) & \longmapsto & \ell\left(\hat{m{y_i}}, m{y_i}
ight) \end{array}$$

▶ Typiquement, la fonction de coût pourrait être l'erreur quadratique (distance Euclidienne au carré)

$$\ell\left(\hat{\boldsymbol{y}}_{i}, \boldsymbol{y}_{i}\right) = \frac{1}{2} \left\| \hat{\boldsymbol{y}}_{i} - \boldsymbol{y}_{i} \right\|_{2}^{2}$$

Soit une fonction de coût (cost or loss function) qui mesure l'erreur entre la prédiction \hat{y}_i et la réalité y_i :

$$egin{array}{lll} \ell: \mathbb{R}^{d_2} imes \mathbb{R}^{d_2} & \longrightarrow & \mathbb{R} \ (\hat{m{y_i}}, m{y_i}) & \longmapsto & \ell\left(\hat{m{y_i}}, m{y_i}
ight) \end{array}$$

► Typiquement, la fonction de coût pourrait être l'erreur quadratique (distance Euclidienne au carré)

$$\ell\left(\hat{\boldsymbol{y}_i}, \boldsymbol{y_i}\right) = \frac{1}{2} \left\|\hat{\boldsymbol{y}_i} - \boldsymbol{y_i}\right\|_2^2$$

► La fonction de coût peut être naturellement généralisée à un ensemble de *prédictions* et de *réalités*:

$$egin{array}{lll} \mathcal{L}: \mathbb{R}^{d_2} imes \cdots & \mathbb{R}^{d_2} & \longrightarrow & \mathbb{R} \ (\hat{m{y_1}}, \ldots, \hat{m{y_N}}, m{y_i} \ldots, m{y_N}) & \longmapsto & \mathcal{L}\left(\hat{m{y_1}}, \ldots, \hat{m{y_N}}, m{y_i} \ldots, m{y_N}
ight) \end{array}$$

► Typiquement, la fonction de coût pourrait être l'erreur quadratique moyenne (mean squared error MSE)

$$\mathcal{L}(\hat{\mathbf{y}}_{1}, \dots, \hat{\mathbf{y}}_{N}, \mathbf{y}_{i}, \dots, \mathbf{y}_{N}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_{i}, \mathbf{y}_{i})$$
$$= \frac{1}{2N} \sum_{i=1}^{N} ||\hat{\mathbf{y}}_{i} - \mathbf{y}_{i}||$$

► La fonction de coût peut être naturellement généralisée à un ensemble de *prédictions* et de *réalités*:

$$egin{array}{lll} \mathcal{L}: \mathbb{R}^{d_2} imes \cdots & \mathbb{R}^{d_2} & \longrightarrow & \mathbb{R} \ (\hat{m{y_1}}, \ldots, \hat{m{y_N}}, m{y_i} \ldots, m{y_N}) & \longmapsto & \mathcal{L}\left(\hat{m{y_1}}, \ldots, \hat{m{y_N}}, m{y_i} \ldots, m{y_N}
ight) \end{array}$$

► Typiquement, la fonction de coût pourrait être l'erreur quadratique moyenne (mean squared error MSE)

$$\mathcal{L}(\hat{\boldsymbol{y}}_{1}, \dots, \hat{\boldsymbol{y}}_{N}, \boldsymbol{y}_{i}, \dots, \boldsymbol{y}_{N}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\boldsymbol{y}}_{i}, \boldsymbol{y}_{i})$$
$$= \frac{1}{2N} \sum_{i=1}^{N} \|\hat{\boldsymbol{y}}_{i} - \boldsymbol{y}_{i}\|_{2}^{2}$$

Pour un réseau de neurones \mathcal{N}_{Θ} , l'erreur entre les prédictions et les réalités est

$$\mathcal{L}\left(f_{\mathbf{\Theta}}\left(\boldsymbol{x_{1}}\right),\ldots,f_{\mathbf{\Theta}}\left(\boldsymbol{x_{N}}\right),\boldsymbol{y_{1}},\ldots,\boldsymbol{y_{N}}\right).$$

- Pour différents paramètres Θ , on aura différentes prédictions $f_{\Theta}(x_1), \ldots, f_{\Theta}(x_N)$, et donc différentes erreurs $\mathcal{L}(\ldots)$.
- ightharpoonup Ainsi, $\mathcal L$ est une fonction des paramètres Θ du réseau

où $|\Theta|$ est le nombre de paramètres Θ (poids et biais, souvent plusieurs millions).

Pour un réseau de neurones \mathcal{N}_{Θ} , l'erreur entre les prédictions et les réalités est

$$\mathcal{L}\left(f_{\Theta}\left(\boldsymbol{x_{1}}\right),\ldots,f_{\Theta}\left(\boldsymbol{x_{N}}\right),\boldsymbol{y_{1}},\ldots,\boldsymbol{y_{N}}\right).$$

- \triangleright Pour différents paramètres Θ , on aura différentes prédictions $f_{\Theta}(x_1), \ldots, f_{\Theta}(x_N)$, et donc différentes erreurs $\mathcal{L}(\ldots)$.

4 D > 4 D > 4 E > 4 E > E 900

CHAIN BILLE

Pour un réseau de neurones \mathcal{N}_{Θ} , l'erreur entre les prédictions et les réalités est

$$\mathcal{L}\left(f_{\mathbf{\Theta}}\left(\boldsymbol{x_{1}}\right),\ldots,f_{\mathbf{\Theta}}\left(\boldsymbol{x_{N}}\right),\boldsymbol{y_{1}},\ldots,\boldsymbol{y_{N}}\right).$$

- \triangleright Pour différents paramètres Θ , on aura différentes prédictions $f_{\Theta}(x_1), \ldots, f_{\Theta}(x_N)$, et donc différentes erreurs $\mathcal{L}(\ldots)$.
- \triangleright Ainsi, \mathcal{L} est une fonction des paramètres Θ du réseau:

$$\mathcal{L}: \mathbb{R}^{|\Theta|} \longrightarrow \mathbb{R}$$

$$\Theta \longmapsto \mathcal{L}\left(f_{\Theta}\left(x_{1}\right), \dots, f_{\Theta}\left(x_{N}\right), y_{1}, \dots, y_{N}\right).$$

où $|\Theta|$ est le nombre de paramètres Θ (poids et biais, souvent plusieurs millions).

 \triangleright L'entraînement du réseau \mathcal{N}_{Θ} consiste à déterminer des paramètres Θ qui minimisent l'erreur

$$\mathcal{L}(f_{\Theta}(x_1),\ldots,f_{\Theta}(x_N),y_1,\ldots,y_N).$$

 $lackbox{L'entraı̂nement}$ du réseau \mathcal{N}_{Θ} consiste à déterminer des paramètres Θ qui minimisent l'erreur

$$\mathcal{L}(f_{\Theta}(x_1),\ldots,f_{\Theta}(x_N),y_1,\ldots,y_N).$$

- ▶ Pour cela, on utilise une descente de gradient: *mini-batch stochas-tic gradient descent*.
- La backpropagation est un algorithme qui permet de calculer les gradients $\nabla_{\Theta} \mathcal{L}$ de manière efficiente.
- ► [Werbos, 1982, Rumelhart et al., 1986]

 $lackbox{L'entraînement}$ du réseau \mathcal{N}_{Θ} consiste à déterminer des paramètres Θ qui minimisent l'erreur

$$\mathcal{L}(f_{\Theta}(x_1),\ldots,f_{\Theta}(x_N),y_1,\ldots,y_N).$$

- ▶ Pour cela, on utilise une descente de gradient: *mini-batch stochas-tic gradient descent*.
- La backpropagation est un algorithme qui permet de calculer les gradients $\nabla_{\Theta} \mathcal{L}$ de manière efficiente.
- ► [Werbos, 1982, Rumelhart et al., 1986]

 $lackbox{L'entraı̂nement}$ du réseau \mathcal{N}_{Θ} consiste à déterminer des paramètres Θ qui minimisent l'erreur

$$\mathcal{L}(f_{\Theta}(x_1),\ldots,f_{\Theta}(x_N),y_1,\ldots,y_N).$$

- ▶ Pour cela, on utilise une descente de gradient: *mini-batch stochas-tic gradient descent*.
- La backpropagation est un algorithme qui permet de calculer les gradients $\nabla_{\Theta} \mathcal{L}$ de manière efficiente.
- ► [Werbos, 1982, Rumelhart et al., 1986]

$$\mathcal{L}\left(f_{\Theta}\left(x_{1}\right),\ldots,f_{\Theta}\left(x_{N}\right),y_{1},\ldots,y_{N}\right)=\mathcal{L}\left(\Theta,\ldots\right)$$

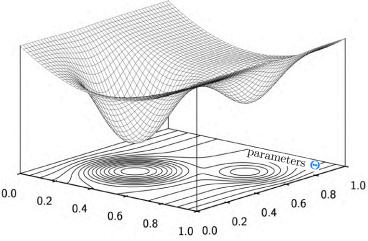
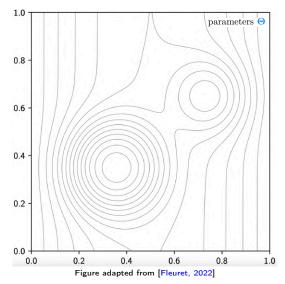
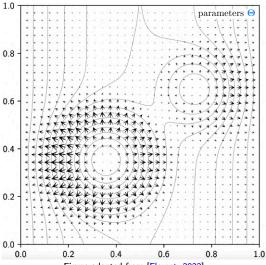


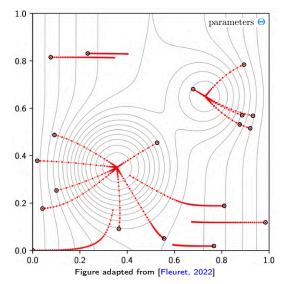
Figure adapted from [Fleuret, 2022]



TRAINING



TRAINING



TRAINING

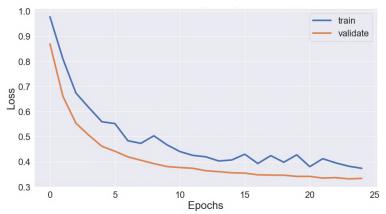
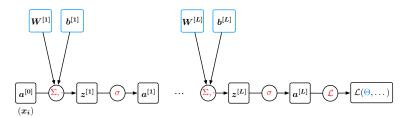


Figure taken from towardsdatascience.com

Graphe computationnel d'un réseau de NEURONES (FORWARD PASS)



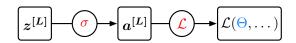
BACKPROPAGATION •00000000000000000

On veut calculer les gradients:

$$\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) := \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{W}^{[l]}} \ \ \text{et} \ \ \nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) := \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial \boldsymbol{b}^{[l]}}$$

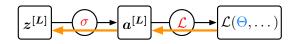
pour
$$l = 1, \ldots, M$$

CHAIN BILLE

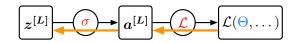


$$\begin{split} \delta_j^{[L]} &:= \frac{\partial \mathcal{L}(\Theta)}{\partial z_j^{[L]}} &= \sum_{k=1}^{|\mathbf{a}^{[L]}|} \frac{\partial \mathcal{L}(\Theta)}{\partial a_k^{[L]}} \cdot \frac{\partial a_k^{[L]}}{\partial z_j^{[L]}} \\ &= \sum_{k=1}^{|\mathbf{a}^{[L]}|} \frac{\partial \mathcal{L}(\Theta)}{\partial a_k^{[L]}} \cdot \frac{\partial \sigma(z_k^{[L]})}{\partial z_j^{[L]}} \\ \left(\frac{\partial \sigma(z_k^{[L]})}{\partial z_j^{[L]}} = 0 \text{ for } k \neq j \right) &= \frac{\partial \mathcal{L}(\Theta)}{\partial a_k^{[L]}} \cdot \sigma'(z_j^{[L]}) \end{split}$$

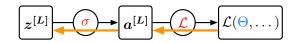
Calcul des gradients: équation 1



$$\begin{split} \delta_{\pmb{j}}^{[L]} &:= \frac{\partial \mathcal{L}(\pmb{\Theta})}{\partial z_{\pmb{j}}^{[L]}} &= \sum_{k=1}^{|a^{[L]}|} \frac{\partial \mathcal{L}(\pmb{\Theta})}{\partial a_{k}^{[L]}} \cdot \frac{\partial a_{k}^{[L]}}{\partial z_{j}^{[L]}} \\ &= \sum_{k=1}^{|a^{[L]}|} \frac{\partial \mathcal{L}(\pmb{\Theta})}{\partial a_{k}^{[L]}} \cdot \frac{\partial \sigma(z_{k}^{[L]})}{\partial z_{j}^{[L]}} \\ \left(\frac{\partial \sigma(z_{k}^{[L]})}{\partial z_{j}^{[L]}} = 0 \text{ for } k \neq j \right) &= \frac{\partial \mathcal{L}(\pmb{\Theta})}{\partial a_{k}^{[L]}} \cdot \sigma'(z_{j}^{[L]}) \end{split}$$

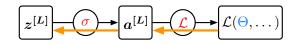


$$\begin{split} \delta_j^{[L]} &:= \frac{\partial \mathcal{L}(\Theta)}{\partial z_j^{[L]}} &= \sum_{k=1}^{|a^{[L]}|} \frac{\partial \mathcal{L}(\Theta)}{\partial a_k^{[L]}} \cdot \frac{\partial a_k^{[L]}}{\partial z_j^{[L]}} \\ &= \sum_{k=1}^{|a^{[L]}|} \frac{\partial \mathcal{L}(\Theta)}{\partial a_k^{[L]}} \cdot \frac{\partial \sigma(z_k^{[L]})}{\partial z_j^{[L]}} \\ \left(\frac{\partial \sigma(z_k^{[L]})}{\partial z_j^{[L]}} = 0 \text{ for } k \neq j \right) &= \frac{\partial \mathcal{L}(\Theta)}{\partial a_k^{[L]}} \cdot \sigma'(z_j^{[L]}) \end{split}$$



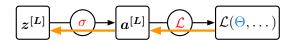
$$\begin{split} \delta_j^{[L]} &:= \frac{\partial \mathcal{L}(\Theta)}{\partial z_j^{[L]}} &= \sum_{k=1}^{|\boldsymbol{a}^{[L]}|} \frac{\partial \mathcal{L}(\Theta)}{\partial a_k^{[L]}} \cdot \frac{\partial a_k^{[L]}}{\partial z_j^{[L]}} \\ &= \sum_{k=1}^{|\boldsymbol{a}^{[L]}|} \frac{\partial \mathcal{L}(\Theta)}{\partial a_k^{[L]}} \cdot \frac{\partial \sigma(z_k^{[L]})}{\partial z_j^{[L]}} \\ \left(\frac{\partial \sigma(z_k^{[L]})}{\partial z_j^{[L]}} = 0 \text{ for } k \neq j \right) &= \frac{\partial \mathcal{L}(\Theta)}{\partial a_k^{[L]}} \cdot \sigma'(z_j^{[L]}) \end{split}$$

Calcul des gradients: équation 1



$$\begin{split} \delta_j^{[L]} &:= \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial z_j^{[L]}} &= \sum_{k=1}^{|\mathbf{a}^{[L]}|} \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial a_k^{[L]}} \cdot \frac{\partial a_k^{[L]}}{\partial z_j^{[L]}} \\ &= \sum_{k=1}^{|\mathbf{a}^{[L]}|} \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial a_k^{[L]}} \cdot \frac{\partial \sigma(z_k^{[L]})}{\partial z_j^{[L]}} \\ \left(\frac{\partial \sigma(z_k^{[L]})}{\partial z_j^{[L]}} = 0 \text{ for } k \neq j \right) &= \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial a_k^{[L]}} \cdot \sigma'(z_j^{[L]}) \end{split}$$

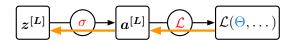
Formulation vectorielle:



$$\delta^{[L]} \; := \; \nabla_{z^{[L]}} \mathcal{L}(\Theta) \;\; = \;\; \nabla_{a^{[L]}} \mathcal{L}(\Theta) \odot \sigma'(z^{[L]})$$

CHAIN BILLE

Formulation vectorielle:

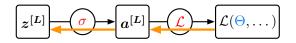


$$\boldsymbol{\delta^{[L]}} \; := \; \nabla_{\boldsymbol{z}^{[L]}} \mathcal{L}(\boldsymbol{\Theta}) \;\; = \;\; \nabla_{\boldsymbol{a}^{[L]}} \mathcal{L}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma}'(\boldsymbol{z}^{[L]})$$

CHAIN BILLE

CALCUL DES GRADIENTS: ÉQUATION 1

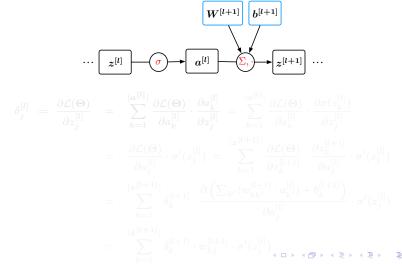
Formulation vectorielle:



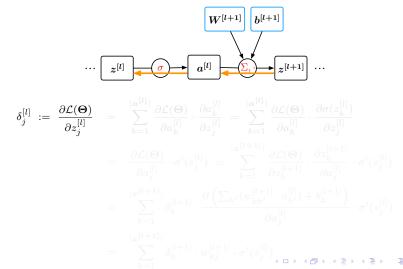
$$\delta^{[L]} \; := \; \nabla_{\boldsymbol{z}^{[L]}} \mathcal{L}(\boldsymbol{\Theta}) \;\; = \;\; \nabla_{\boldsymbol{a}^{[L]}} \mathcal{L}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[L]})$$

où ⊙ est le produit de Hadamard (composante par composante).

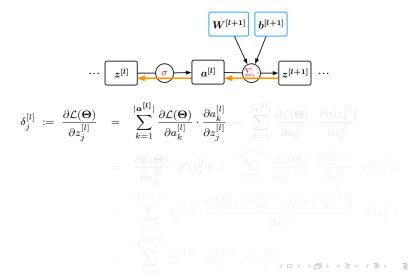
Supposons que les $\delta_k^{[l+1]}$ ont été calculés pour tous k:



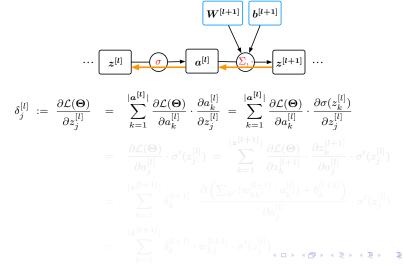
Supposons que les $\delta_k^{[l+1]}$ ont été calculés pour tous k:



Supposons que les $\delta_k^{[l+1]}$ ont été calculés pour tous k:

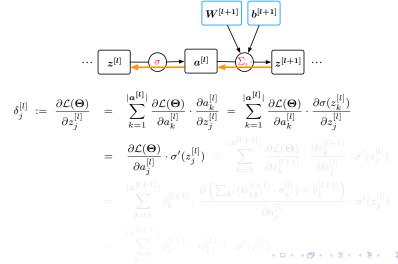


Supposons que les $\delta_k^{[l+1]}$ ont été calculés pour tous k:



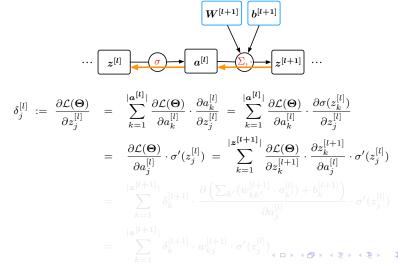
CALCUL DES GRADIENTS: ÉQUATION 2

Supposons que les $\delta_k^{[l+1]}$ ont été calculés pour tous k:



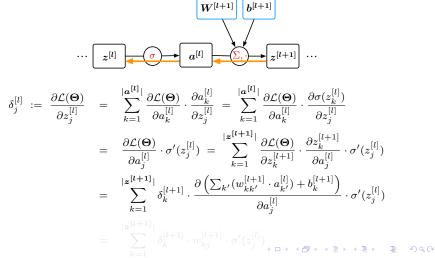
Calcul des gradients: équation 2

Supposons que les $\delta_k^{[l+1]}$ ont été calculés pour tous k:



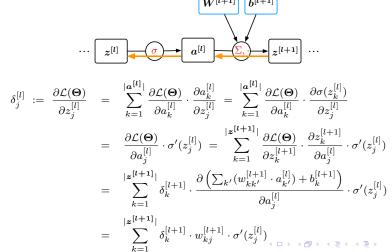
Calcul des gradients: équation 2

Supposons que les $\delta_k^{[l+1]}$ ont été calculés pour tous k:



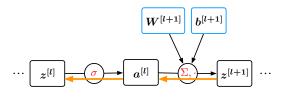
Calcul des gradients: équation 2

Supposons que les $\delta_k^{[l+1]}$ ont été calculés pour tous k:



CALCUL DES GRADIENTS: ÉQUATION 2

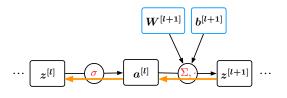
Formulation vectorielle:



$$\begin{split} \boldsymbol{\delta}^{[l]} &:= & \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \\ &= & \left[\boldsymbol{\delta}^{[l+1]}\right]^T \boldsymbol{W}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}) \\ &= & \left[\boldsymbol{W}^{[l+1]}\right]^T \boldsymbol{\delta}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}) \end{split}$$

◆□▶ ◆圖▶ ◆園▶ ◆園▶ ■

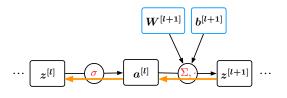
Formulation vectorielle:



$$egin{array}{lll} \boldsymbol{\delta}^{[l]} &:= & \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \ &= & \left[\boldsymbol{\delta}^{[l+1]}
ight]^T W^{[l+1]} \odot \boldsymbol{\sigma}'(\boldsymbol{z}^{[l]}) \ &= & \left[\boldsymbol{W}^{[l+1]}
ight]^T \boldsymbol{\delta}^{[l+1]} \odot \boldsymbol{\sigma}'(\boldsymbol{z}^{[l]}) \end{array}$$

◆ロト ◆園 ▶ ◆夏 ▶ ◆夏 ▶ □ ● の Q №

Formulation vectorielle:

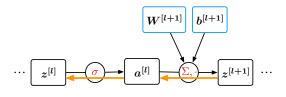


$$\begin{split} \boldsymbol{\delta}^{[l]} &:= & \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \\ &= & \left[\boldsymbol{\delta}^{[l+1]}\right]^T \boldsymbol{W}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}) \\ &= & \left[\boldsymbol{W}^{[l+1]}\right]^T \boldsymbol{\delta}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}) \end{split}$$

<ロ > ← □

Formulation vectorielle:

CHAIN RULE

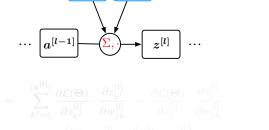


$$\begin{split} \boldsymbol{\delta}^{[l]} &:= & \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \\ &= & \left[\boldsymbol{\delta}^{[l+1]} \right]^T \boldsymbol{W}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}) \\ &= & \left[\boldsymbol{W}^{[l+1]} \right]^T \boldsymbol{\delta}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}) \end{split}$$

où \odot est le produit de Hadamard (composante par composante).

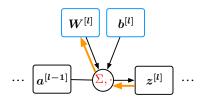
Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :

 $W^{[l]}$



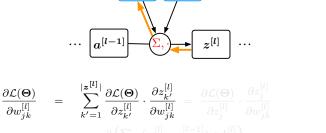
 $b^{[l]}$

Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :



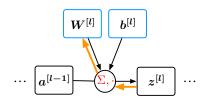
$$\begin{split} \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial w_{jk}^{[l]}} &= \sum_{k'=1}^{\lfloor z-1 \rfloor} \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial z_{k'}^{[l]}} \cdot \frac{\partial z_{k'}^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial z_{j}^{[l]}} \cdot \frac{\partial z_{j}^{[l]}}{\partial w_{jk}^{[l]}} \\ &= \delta_{j}^{[l]} \cdot \frac{\partial \left(\sum_{k'} (w_{jk'}^{[l]} \cdot a_{k'}^{[l-1]}) + b_{j}^{[l]}\right)}{\partial w_{jk}^{[l]}} = \delta_{j}^{[l]} \cdot a_{k}^{[l-1]} \end{split}$$

Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :



 $b^{[l]}$

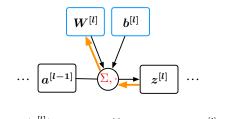
Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :



$$\frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial w_{jk}^{[l]}} = \sum_{k'=1}^{|\boldsymbol{z}^{[l]}|} \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial z_{k'}^{[l]}} \cdot \frac{\partial z_{k'}^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial z_{j}^{[l]}} \cdot \frac{\partial z_{j}^{[l]}}{\partial w_{jk}^{[l]}}$$

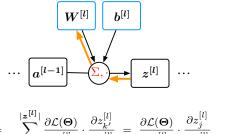
$$= \delta_{j}^{[l]} \cdot \frac{\partial \left(\sum_{k'} (w_{jk'}^{[l]} \cdot a_{k'}^{[l-1]}) + b_{j}^{[l]}\right)}{\partial w_{jk}^{[l]}} = \delta_{j}^{[l]} \cdot a_{k}^{[l-1]}$$

Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :



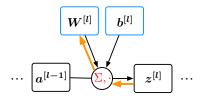
$$\begin{array}{ll} \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial w_{jk}^{[l]}} & = & \sum_{k'=1}^{|\boldsymbol{z}^{[l]}|} \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial z_{k'}^{[l]}} \cdot \frac{\partial z_{k'}^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial z_{j}^{[l]}} \cdot \frac{\partial z_{j}^{[l]}}{\partial w_{jk}^{[l]}} \\ & = & \delta_{j}^{[l]} \cdot \frac{\partial \left(\sum_{k'} (w_{jk'}^{[l]} \cdot a_{k'}^{[l-1]}) + b_{j}^{[l]}\right)}{\partial w_{jk}^{[l]}} = \delta_{j}^{[l]} \cdot a_{k}^{[l-1]} \end{array}$$

Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :



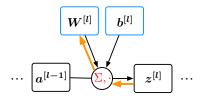
$$\begin{array}{ll} \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial w_{jk}^{[l]}} & = & \sum_{k'=1}^{|\boldsymbol{z}^{[l]}|} \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial z_{k'}^{[l]}} \cdot \frac{\partial z_{k'}^{[l]}}{\partial w_{jk}^{[l]}} = \frac{\partial \mathcal{L}(\boldsymbol{\Theta})}{\partial z_{j}^{[l]}} \cdot \frac{\partial z_{j}^{[l]}}{\partial w_{jk}^{[l]}} \\ & = & \delta_{j}^{[l]} \cdot \frac{\partial \left(\sum_{k'} (w_{jk'}^{[l]} \cdot a_{k'}^{[l-1]}) + b_{j}^{[l]}\right)}{\partial w_{jk}^{[l]}} = \delta_{j}^{[l]} \cdot a_{k}^{[l-1]} \end{array}$$

Formulation vectorielle:



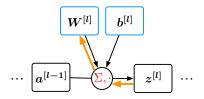
$$abla_{W^{[l]}} \mathcal{L}(\mathbf{\Theta}) = \delta^{[l]} \left[a^{[l-1]} \right]^{T}$$

Formulation vectorielle:



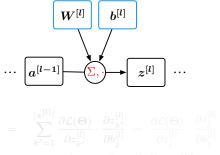
$$abla_{oldsymbol{W}^{[l]}} \mathcal{L}(oldsymbol{\Theta}) \ = \ \delta^{[l]} \left[a^{[l-1]}
ight]^T$$

Formulation vectorielle:



$$abla_{oldsymbol{W}^{[l]}} \mathcal{L}(oldsymbol{\Theta}) \ = \ oldsymbol{\delta}^{[l]} \left[oldsymbol{a}^{[l-1]}
ight]^T$$

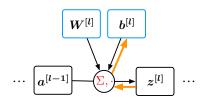
Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :



$$= \sum_{k'=1} \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial z_{k'}^{[l]}} \cdot \frac{\partial z_{k'}}{\partial b_j^{[l]}} = \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial z_j^{[l]}} \cdot \frac{\partial z_j}{\partial b_j^{[l]}}$$

$$= \delta_j^{[l]} \cdot \frac{\partial \left(\sum_{k'} (w_{jk'}^{[l]} \cdot a_{k'}^{[l-1]}) + b_j^{[l]}\right)}{\partial b_j^{[l]}} = \delta_j^{[l]}$$

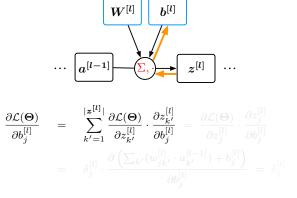
Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :



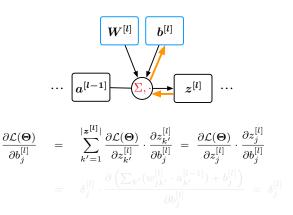
$$\frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial b_{j}^{[l]}} = \sum_{k'=1}^{|\mathbf{z}^{[l]}|} \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial z_{k'}^{[l]}} \cdot \frac{\partial z_{k'}^{[l]}}{\partial b_{j}^{[l]}} = \frac{\partial \mathcal{L}(\mathbf{\Theta})}{\partial z_{j}^{[l]}} \cdot \frac{\partial z_{j}^{[l]}}{\partial b_{j}^{[l]}}$$

$$= \delta_{j}^{[l]} \cdot \frac{\partial \left(\sum_{k'} (w_{jk'}^{[l]} \cdot a_{k'}^{[l-1]}) + b_{j}^{[l]}\right)}{\partial b_{j}^{[l]}} = \delta_{j}^{[l]}$$

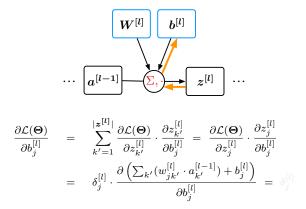
Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :



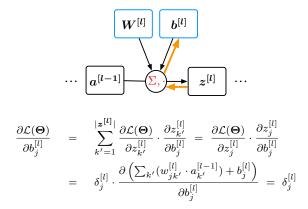
Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :



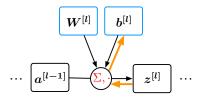
Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :



Calcul des gradients proprement dits en utilisant les erreurs δ_i^l :

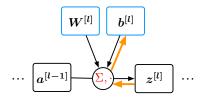


Formulation vectorielle:



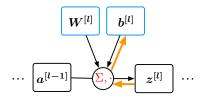
$$\nabla_{b^{[l]}} \mathcal{L}(\mathbf{\Theta}) = \delta^{[l]}$$

Formulation vectorielle:



$$abla_{m{b}^{[l]}} \mathcal{L}(m{\Theta}) = \delta^{[l]}$$

Formulation vectorielle:



$$\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) = \boldsymbol{\delta}^{[l]}$$

Calcul des erreurs $\delta^{[l]} := \nabla_{z^{[l]}} \mathcal{L}(\Theta)$ et des gradients $\nabla_{W^{[l]}} \mathcal{L}(\Theta)$ et $\nabla_{\mathbf{h}^{[l]}} \mathcal{L}(\mathbf{\Theta})$, pour toute couche $l = L, \ldots, 1$:

$$\delta^{[l]} = \begin{cases} \nabla_{a^{[l]}} \mathcal{L}(\mathbf{\Theta}) \odot \sigma'(z^{[l]}), & \text{si } l = L \\ \left[\mathbf{W}^{[l+1]} \right]^T \delta^{[l+1]} \odot \sigma'(z^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(1)

$$\nabla_{W^{[l]}} \mathcal{L}(\Theta) = \delta^{[l]} \left[a^{[l-1]} \right]^T \tag{2}$$

$$\nabla_{b^{[l]}} \mathcal{L}(\mathbf{\Theta}) = \delta^{[l]} \tag{3}$$

Calcul des erreurs $\boldsymbol{\delta}^{[l]} := \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et des gradients $\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et $\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$, pour toute couche $l = L, \dots, 1$:

$$\boldsymbol{\delta}^{[l]} = \begin{cases} \nabla_{\boldsymbol{a}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } l = L \\ \left[W^{[l+1]} \right]^T \boldsymbol{\delta}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(1)

Pour $l = L, \ldots, 1$:

$$\nabla_{W^{[l]}} \mathcal{L}(\Theta) = \delta^{[l]} \left[a^{[l-1]} \right]^T \tag{2}$$

BACKPROPAGATION

$$\nabla_{b^{[l]}} \mathcal{L}(\Theta) = \delta^{[l]} \tag{3}$$

Calcul des erreurs $\boldsymbol{\delta}^{[l]} := \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et des gradients $\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et $\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$, pour toute couche $l = L, \ldots, 1$:

$$\boldsymbol{\delta}^{[l]} = \begin{cases} \nabla_{\boldsymbol{a}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } l = L \\ \left[\boldsymbol{W}^{[l+1]} \right]^T \boldsymbol{\delta}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(1)

$$\nabla_{W^{[l]}} \mathcal{L}(\Theta) = \delta^{[l]} \left[a^{[l-1]} \right]^T \tag{2}$$

$$\nabla_{b^{[l]}} \mathcal{L}(\Theta) = \delta^{[l]} \tag{3}$$

Calcul des erreurs $\boldsymbol{\delta}^{[l]} := \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et des gradients $\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et $\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$, pour toute couche $l = L, \dots, 1$:

$$\boldsymbol{\delta}^{[l]} = \begin{cases} \nabla_{\boldsymbol{a}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } l = L \\ \left[\boldsymbol{W}^{[l+1]} \right]^T \boldsymbol{\delta}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(1)

$$\nabla_{W^{[l]}} \mathcal{L}(\mathbf{\Theta}) = \delta^{[l]} \left[a^{[l-1]} \right]^T \tag{2}$$

$$\nabla_{b^{[l]}} \mathcal{L}(\Theta) = \delta^{[l]} \tag{3}$$

Calcul des erreurs $\boldsymbol{\delta}^{[l]} := \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et des gradients $\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et $\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$, pour toute couche $l = L, \dots, 1$:

$$\boldsymbol{\delta}^{[l]} = \begin{cases} \nabla_{\boldsymbol{a}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } l = L \\ \left[\boldsymbol{W}^{[l+1]} \right]^T \boldsymbol{\delta}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(1)

$$\nabla_{W^{[l]}} \mathcal{L}(\mathbf{\Theta}) = \delta^{[l]} \left[a^{[l-1]} \right]^T \tag{2}$$

$$\nabla_{b^{[l]}} \mathcal{L}(\mathbf{\Theta}) = \delta^{[l]} \tag{3}$$

Calcul des erreurs $\boldsymbol{\delta}^{[l]} := \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et des gradients $\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et $\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$, pour toute couche $l = L, \dots, 1$:

$$\boldsymbol{\delta}^{[l]} = \begin{cases} \nabla_{\boldsymbol{a}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } l = L \\ \left[\boldsymbol{W}^{[l+1]} \right]^T \boldsymbol{\delta}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(1)

$$\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) = \boldsymbol{\delta}^{[l]} \left[\boldsymbol{a}^{[l-1]} \right]^{T}$$
 (2)

$$\nabla_{b^{[l]}} \mathcal{L}(\mathbf{\Theta}) = \delta^{[l]} \tag{3}$$

Calcul des erreurs $\boldsymbol{\delta}^{[l]} := \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et des gradients $\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$ et $\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}(\boldsymbol{\Theta})$, pour toute couche $l = L, \dots, 1$:

$$\boldsymbol{\delta}^{[l]} = \begin{cases} \nabla_{\boldsymbol{a}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } l = L \\ \left[\boldsymbol{W}^{[l+1]} \right]^T \boldsymbol{\delta}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(1)

Pour $l = L, \ldots, 1$:

$$\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) = \delta^{[l]} \left[\boldsymbol{a}^{[l-1]} \right]^T \tag{2}$$

BACKPROPAGATION

$$\nabla_{b^{[l]}} \mathcal{L}(\mathbf{\Theta}) = \delta^{[l]} \tag{3}$$

Une fois gradients calculés, on effectue l'update des poids et biais (cf. gradient descent algo):

$$W^{[l]} := W^{[l]} - \eta \cdot \nabla_{W^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \tag{4}$$

$$\boldsymbol{b}^{[l]} := \boldsymbol{b}^{[l]} - \eta \cdot \nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}(\boldsymbol{\Theta}) \tag{5}$$

Une fois gradients calculés, on effectue l'update des poids et biais (cf. gradient descent algo):

Pour
$$l = L, ..., 1$$
:
$$\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \eta \cdot \nabla_{\mathbf{W}^{[l]}} \mathcal{L}(\mathbf{\Theta})$$

$$b^{[l]} := b^{[l]} - \eta \cdot \nabla_{\mathbf{W}^{[l]}} \mathcal{L}(\mathbf{\Theta})$$
(5)

où η est le *learning rate*.

Une fois gradients calculés, on effectue l'update des poids et biais (cf. gradient descent algo):

Pour
$$l = L, ..., 1$$
:

$$\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \eta \cdot \nabla_{\mathbf{W}^{[l]}} \mathcal{L}(\mathbf{\Theta})$$

$$\mathbf{b}^{[l]} := \mathbf{b}^{[l]} - \eta \cdot \nabla_{\mathbf{b}^{[l]}} \mathcal{L}(\mathbf{\Theta})$$
(5)

où η est le *learning rate*.

CHAIN RULE

Remarque: on peut déduire une version "batched" des équations.

Remarque: on peut déduire une version "batched" des équations.

Soit B = (X, Y) un batch composé de B inputs et outputs x_k et y_k alignés en deux matrices:

$$egin{aligned} egin{aligned} egin{aligned} egin{aligned} egin{aligned} dallet & dallet & \dots & dallet \ m{x_1} & m{x_2} & \cdots & m{x_B} \ dallet & dallet & \dots & dallet \end{aligned} \end{aligned} egin{aligned} dallet & egin{aligned} dallet & dallet & \dots & dallet \ m{y_1} & m{y_2} & \cdots & m{y_B} \ dallet & dallet & \dots & dallet \end{aligned}$$

$$A^{[L]} = \begin{pmatrix} \vdots & \vdots & \dots & \vdots \\ a_1^{[L]} & a_2^{[L]} & \cdots & a_B^{[L]} \\ \vdots & \vdots & \dots & \vdots \end{pmatrix}$$

4□ → 4回 → 4 回 → 4 回 → 9 へ ○

CHAIN BILLE

Remarque: on peut déduire une version "batched" des équations.

Soit B = (X, Y) un batch composé de B inputs et outputs x_k et y_k alignés en deux matrices:

BACKPROPAGATION

$$egin{aligned} egin{aligned} egin{aligned} egin{aligned} egin{aligned} dall & dall & dall & \dots & dall \ m{x_1} & m{x_2} & \cdots & m{x_B} \ dall & dall & \dots & dall \end{aligned} \end{aligned} egin{aligned} egin{aligned} dall & all & all & all & \dots & dall \ m{y_1} & m{y_2} & \cdots & m{y_B} \ dall & egin{aligned} dall & all & all & \dots & dall \end{aligned} \end{aligned}$$

Soit $A^{[L]}$ les outputs du réseau associés aux inputs X:

$$A^{[L]} = egin{pmatrix} dots & dots & \ldots & dots \ a_1^{[L]} & a_2^{[L]} & \cdots & a_B^{[L]} \ dots & dots & \ldots & dots \end{pmatrix}$$

4□ → 4回 → 4 回 → 4 回 → 9 へ ○

CHAIN BILLE

Remarque: on peut déduire une version "batched" des équations.

Soit B = (X, Y) un batch composé de B inputs et outputs x_k et y_k alignés en deux matrices:

BACKPROPAGATION

$$egin{aligned} X = egin{pmatrix} dots & dots & \ldots & dots \ x_1 & x_2 & \cdots & x_B \ dots & dots & \ldots & dots \end{pmatrix} & ext{et } Y = egin{pmatrix} dots & dots & \ldots & dots \ y_1 & y_2 & \cdots & y_B \ dots & dots & \ldots & dots \end{pmatrix} \end{aligned}$$

Soit $A^{[L]}$ les outputs du réseau associés aux inputs X:

$$A^{[L]} = egin{pmatrix} dots & dots & \ldots & dots \ a_1^{[L]} & a_2^{[L]} & \cdots & a_B^{[L]} \ dots & dots & \ldots & dots \end{pmatrix}$$

lackbrack Soit $\mathcal{L}_k(m{\Theta}) := \mathcal{L}(m{\Theta}, m{a_k^{[L]}}, m{y_k})$ la loss associée à l'exemple k, pour $k = 1, \dots, B$. 4□ ト 4 □ ト 4 □ ト 4 □ ト 9 0 ○

Calcul des erreurs $\delta_{\mathbf{k}}^{[l]} := \nabla_{\mathbf{z}^{[l]}} \mathcal{L}_k(\mathbf{\Theta})$ et des gradients $\nabla_{\mathbf{W}[l]} \mathcal{L}_k(\mathbf{\Theta})$ et $\nabla_{\mathbf{h}[l]} \mathcal{L}_k(\mathbf{\Theta})$, pour tout exemple $k = 1, \dots, B$ pour toute couche $l = L, \ldots, 1$:

Pour $k = 1, \ldots, B$:

$$\delta_{k}^{[l]} = \begin{cases} \nabla_{a^{[L]}} \mathcal{L}_{k}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma}'(\boldsymbol{z}_{k}^{[L]}), & \text{si } l = L \\ \left[\boldsymbol{W}^{[l+1]} \right]^{T} \delta_{k}^{[l+1]} \odot \boldsymbol{\sigma}'(\boldsymbol{z}_{k}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(6)

$$\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta}) = \delta_k^{[l]} \left[a_k^{[l-1]} \right]^T \tag{7}$$

$$\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta}) = \boldsymbol{\delta}^{[l]} \tag{8}$$

CHAIN BILLE

Calcul des erreurs $oldsymbol{\delta}_{k}^{[l]} :=
abla_{oldsymbol{z}^{[l]}} \mathcal{L}_{k}(oldsymbol{\Theta})$ et des gradients $\nabla_{\mathbf{W}[l]} \mathcal{L}_k(\mathbf{\Theta})$ et $\nabla_{\mathbf{h}[l]} \mathcal{L}_k(\mathbf{\Theta})$, pour tout exemple $k = 1, \dots, B$ pour toute couche $l = L, \ldots, 1$:

Pour $k = 1, \ldots, B$:

$$\boldsymbol{\delta}_{k}^{[l]} = \begin{cases} \nabla_{\boldsymbol{a}^{[L]}} \mathcal{L}_{k}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma}'(\boldsymbol{z}_{k}^{[L]}), & \text{si } l = L \\ \left[W^{[l+1]} \right]^{T} \boldsymbol{\delta}_{k}^{[l+1]} \odot \boldsymbol{\sigma}'(\boldsymbol{z}_{k}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(6)

$$\nabla_{W^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta}) = \delta_k^{[l]} \left[a_k^{[l-1]} \right]^T \tag{7}$$

$$\nabla_{b^{[l]}} \mathcal{L}_k(\mathbf{\Theta}) = \delta^{[l]} \tag{8}$$

CHAIN BILLE

Calcul des erreurs $oldsymbol{\delta}_{k}^{[l]} :=
abla_{oldsymbol{z}^{[l]}} \mathcal{L}_{k}(oldsymbol{\Theta})$ et des gradients $\nabla_{\mathbf{W}[l]} \mathcal{L}_k(\mathbf{\Theta})$ et $\nabla_{\mathbf{h}[l]} \mathcal{L}_k(\mathbf{\Theta})$, pour tout exemple $k = 1, \dots, B$ pour toute couche $l = L, \ldots, 1$:

Pour $k = 1, \ldots, B$:

$$\boldsymbol{\delta}_{k}^{[l]} = \begin{cases} \nabla_{\boldsymbol{a}^{[L]}} \mathcal{L}_{k}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma'}(\boldsymbol{z}_{k}^{[L]}), & \text{si } l = L \\ \left[\boldsymbol{W}^{[l+1]} \right]^{T} \boldsymbol{\delta}_{k}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}_{k}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(6)

$$\nabla_{W^{[l]}} \mathcal{L}_k(\mathbf{\Theta}) = \delta_k^{[l]} \left[a_k^{[l-1]} \right]^T \tag{7}$$

$$\nabla_{b^{[l]}} \mathcal{L}_k(\mathbf{\Theta}) = \delta^{[l]} \tag{8}$$

Calcul des erreurs $\delta_k^{[l]} := \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta})$ et des gradients $\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta})$ et $\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta})$, pour tout exemple $k=1,\ldots,B$ pour toute couche $l=L,\ldots,1$:

Pour $k = 1, \ldots, B$:

$$\boldsymbol{\delta}_{k}^{[l]} = \begin{cases} \nabla_{\boldsymbol{a}^{[L]}} \mathcal{L}_{k}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma}'(\boldsymbol{z}_{k}^{[L]}), & \text{si } l = L \\ \left[\boldsymbol{W}^{[l+1]} \right]^{T} \boldsymbol{\delta}_{k}^{[l+1]} \odot \boldsymbol{\sigma}'(\boldsymbol{z}_{k}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(6)

Pour $l = L, \ldots, 1$ et pour $k = 1, \ldots, B$:

$$\nabla_{\mathbf{W}^{[l]}} \mathcal{L}_k(\mathbf{\Theta}) = \delta_k^{[l]} \left[a_k^{[l-1]} \right]^T \tag{7}$$

$$\nabla_{b^{[l]}} \mathcal{L}_k(\mathbf{\Theta}) = \delta^{[l]} \tag{8}$$

Calcul des erreurs $\delta_k^{[l]} := \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta})$ et des gradients $\nabla_{\mathbf{W}[l]} \mathcal{L}_k(\mathbf{\Theta})$ et $\nabla_{\mathbf{h}[l]} \mathcal{L}_k(\mathbf{\Theta})$, pour tout exemple $k = 1, \dots, B$ pour toute couche $l = L, \ldots, 1$:

Pour $k = 1, \ldots, B$:

$$\boldsymbol{\delta_{k}^{[l]}} = \begin{cases} \nabla_{\boldsymbol{a}^{[L]}} \mathcal{L}_{k}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma'}(\boldsymbol{z}_{k}^{[L]}), & \text{si } l = L \\ \left[\boldsymbol{W}^{[l+1]} \right]^{T} \boldsymbol{\delta}_{k}^{[l+1]} \odot \boldsymbol{\sigma'}(\boldsymbol{z}_{k}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(6)

Pour $l = L, \ldots, 1$ et pour $k = 1, \ldots, B$:

$$\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta}) = \delta_k^{[l]} \left[a_k^{[l-1]} \right]^T \tag{7}$$

$$\nabla_{b^{[l]}} \mathcal{L}_k(\mathbf{\Theta}) = \delta^{[l]} \tag{8}$$

CHAIN BILLE

Calcul des erreurs $\delta_k^{[l]} := \nabla_{\boldsymbol{z}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta})$ et des gradients $\nabla_{\mathbf{W}[l]} \mathcal{L}_k(\mathbf{\Theta})$ et $\nabla_{\mathbf{h}[l]} \mathcal{L}_k(\mathbf{\Theta})$, pour tout exemple $k = 1, \dots, B$ pour toute couche $l = L, \ldots, 1$:

Pour $k = 1, \ldots, B$:

$$\boldsymbol{\delta}_{k}^{[l]} = \begin{cases} \nabla_{\boldsymbol{a}^{[L]}} \mathcal{L}_{k}(\boldsymbol{\Theta}) \odot \boldsymbol{\sigma}'(\boldsymbol{z}_{k}^{[L]}), & \text{si } l = L \\ \left[\boldsymbol{W}^{[l+1]} \right]^{T} \boldsymbol{\delta}_{k}^{[l+1]} \odot \boldsymbol{\sigma}'(\boldsymbol{z}_{k}^{[l]}), & \text{si } L > l \ge 1 \end{cases}$$
(6)

Pour $l = L, \ldots, 1$ et pour $k = 1, \ldots, B$:

$$\nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta}) = \delta_k^{[l]} \left[a_k^{[l-1]} \right]^T \tag{7}$$

$$\nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta}) = \boldsymbol{\delta}^{[l]} \tag{8}$$

Une fois gradients calculés pour tout k = 1, ..., B, on effectue l'update des poids et biais (cf. stochastic gradient descent algo):

$$\mathbf{W}^{[l]} := \mathbf{W}^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{\mathbf{W}^{[l]}} \mathcal{L}_k(\mathbf{\Theta})$$
 (9)

$$\boldsymbol{b}^{[l]} := \boldsymbol{b}^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta})$$
 (10)

Une fois gradients calculés pour tout $k = 1, \ldots, B$, on effectue l'update des poids et biais (cf. stochastic gradient descent algo):

Pour
$$l = L, \ldots, 1$$
:

$$\boldsymbol{W}^{[l]} := \boldsymbol{W}^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta})$$
 (9)

$$\boldsymbol{b}^{[l]} := \boldsymbol{b}^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{D} \nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta})$$
 (10)

Une fois gradients calculés pour tout k = 1, ..., B, on effectue l'update des poids et biais (cf. stochastic gradient descent algo):

Pour
$$l = L, \ldots, 1$$
:

$$\boldsymbol{W}^{[l]} := \boldsymbol{W}^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{\boldsymbol{W}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta})$$
 (9)

$$\boldsymbol{b}^{[l]} := \boldsymbol{b}^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{\boldsymbol{b}^{[l]}} \mathcal{L}_k(\boldsymbol{\Theta})$$
 (10)

où η est le *learning rate*.



FORWARD PASS AND BACKWARD PASS

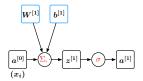


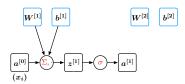


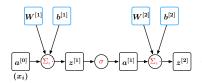


FORWARD PASS AND BACKWARD PASS

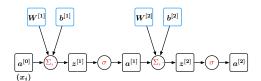






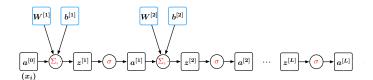


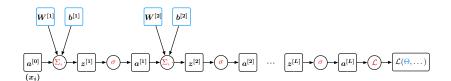
FORWARD PASS AND BACKWARD PASS



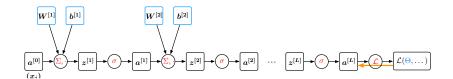
BACKPROPAGATION

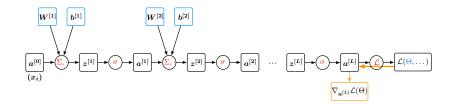
FORWARD PASS AND BACKWARD PASS



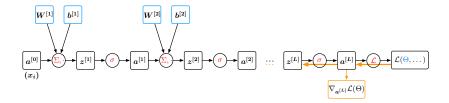


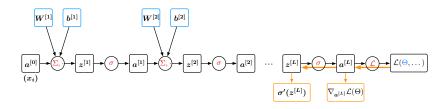
BACKPROPAGATION



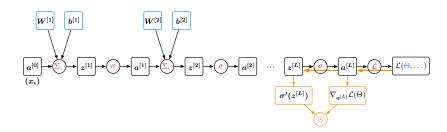


BACKPROPAGATION

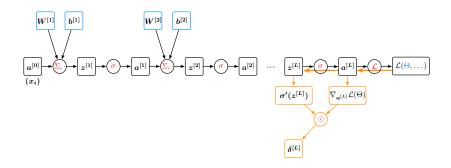


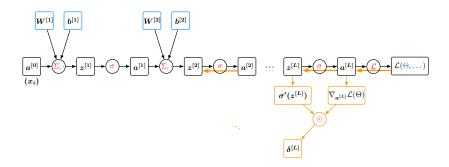


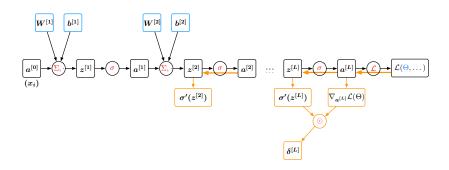
BACKPROPAGATION



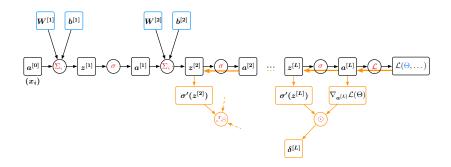
BACKPROPAGATION

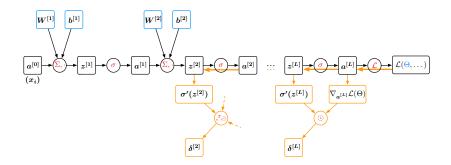


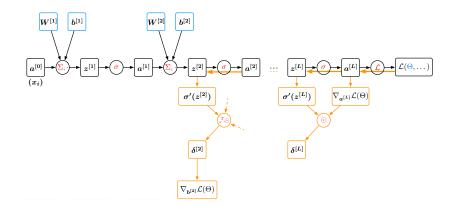




BACKPROPAGATION

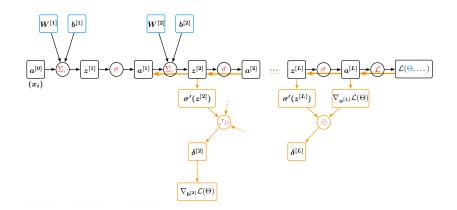


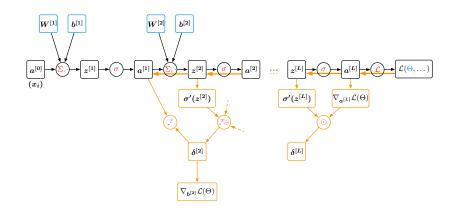


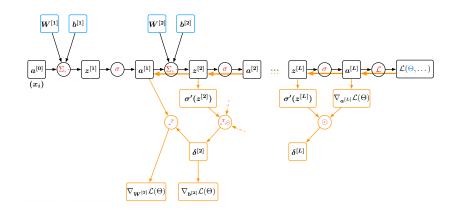


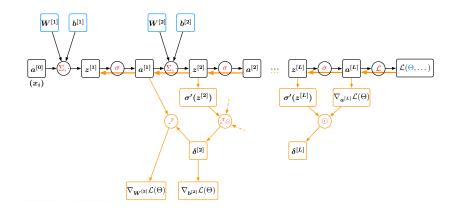
BACKPROPAGATION

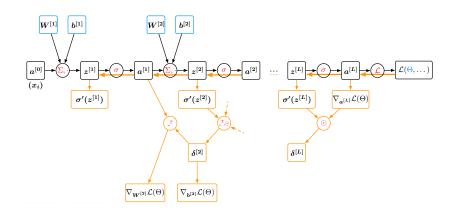
FORWARD PASS AND BACKWARD PASS

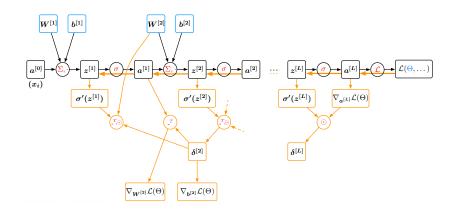


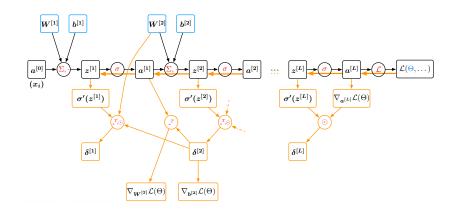


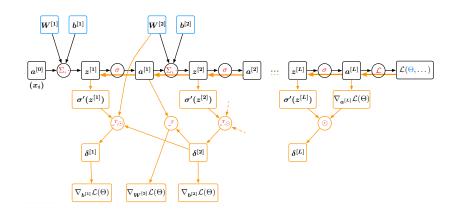


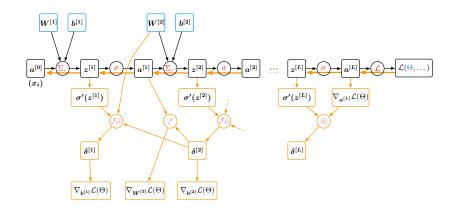


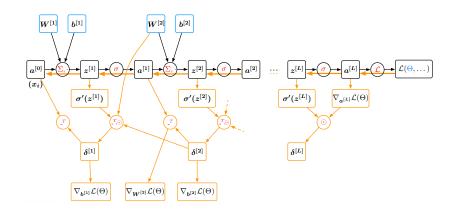


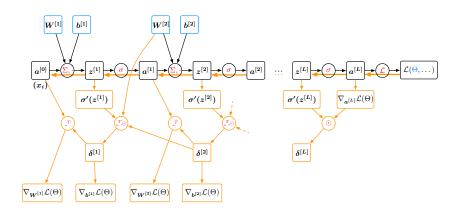












```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, ..., \bar{B}\}
Inputs: MLP = \{(W^{[l]}, b^{[l]}) : l = 1, ..., L\} initialized randomly
```

Algorithm 1: Backpropagation (stochastic gradient descent)

```
Inputs: MLP = \{(W^{[l]}, b^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
```

Data: dataloader = $\{B_b = (X_b, Y_b) : b = 1, ..., \bar{B}\}$

```
// loop over epochs
```

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, ..., \bar{B}\}
Inputs: MLP = \{(W^{[l]}, b^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                   // loop over epochs
     for k = 1 to \bar{B} do
                                                                                 // loop over batches
```

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, ..., \bar{B}\}
Inputs: MLP = \{(W^{[l]}, b^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                // loop over epochs
     for k=1 to \bar{B} do
                                                                               // loop over batches
           for l=1 to L do
                                                                                     // forward pass
```

BACKPROPAGATION

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, ..., \bar{B}\}
Inputs: MLP = \{(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                       // loop over epochs
      for k = 1 to \bar{B} do
                                                                                      // loop over batches
            for l = 1 to L do
                                                                                            // forward pass
```

Algorithm 1: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \left\{ oldsymbol{B_b} = \left( oldsymbol{X_b}, oldsymbol{Y_b} \right) : b = 1, \dots, ar{B} 
ight\}
Inputs: MLP = \{(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                                      // loop over epochs
       for k=1 to \bar{B} do
                                                                                                    // loop over batches
               for l = 1 to L do
                                                                                                           // forward pass
               for l = L to 1 do
                                                                                                          // backward pass
```

BACKPROPAGATION 000000000000000000

Algorithm 1: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \left\{ oldsymbol{B_b} = \left( oldsymbol{X_b}, oldsymbol{Y_b} \right) : b = 1, \dots, ar{B} 
ight\}
Inputs: MLP = \{(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                                     // loop over epochs
       for k = 1 to \bar{B} do
                                                                                                   // loop over batches
              for l = 1 to L do
                                                                                                          // forward pass
              for l = L to 1 do
                                                                                                         // backward pass
                                                                                                         // compute error
```

BACKPROPAGATION

000000000000000000

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, ..., \bar{B}\}
Inputs: MLP = \{(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                     // loop over epochs
      for k = 1 to \bar{B} do
                                                                                   // loop over batches
            for l = 1 to L do
                                                                                         // forward pass
            for l = L to 1 do
                                                                                        // backward pass
                                                                                        // compute error
                                                                                      // update gradient
```

Algorithm 1: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, ..., \bar{B}\}
Inputs: MLP = \{(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                    // loop over epochs
      for k = 1 to \bar{B} do
                                                                                   // loop over batches
            for l = 1 to L do
                                                                                         // forward pass
            for l = L to 1 do
                                                                                       // backward pass
                                                                                       // compute error
                                                                                     // update gradient
                                                                                     // update gradient
```

BACKPROPAGATION 0000000000000000000

Algorithm 1: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \left\{ \boldsymbol{B_b} = (\boldsymbol{X_b}, \boldsymbol{Y_b}) : b = 1, \dots, \bar{B} \right\}
Inputs: MLP = \{(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                                // loop over epochs
      for k = 1 to \bar{B} do
                                                                                               // loop over batches
              for l = 1 to L do
                                                                                                      // forward pass
              for l = L to 1 do
                                                                                                    // backward pass
                                                                                                    // compute error
                                                                                                  // update gradient
                                                                                                  // update gradient
      end
```

BACKPROPAGATION 000000000000000000

end

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \{(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}) : l = 1, ..., L\} initialized randomly
```

CHAIN RULE

《□》《圖》《意》《意》 毫

Algorithm 2: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \left\{\left(oldsymbol{W}^{oldsymbol{[l]}}, oldsymbol{b^{[l]}}\right): l=1,\ldots,L\right\} initialized randomly
for e = 1 to E do
                                                                                                             // loop over epochs
```

CHAIN RULE

◆□▶ ◆圖▶ ◆園▶ ◆園▶ ■園

Algorithm 2: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \{(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                           // loop over epochs
      for b = 1 to \bar{B} do
                                                                                         // loop over batches
```

CHAIN RULE

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \left\{\left(oldsymbol{W}^{[l]}, oldsymbol{b}^{[l]}
ight): l=1,\ldots,L\right\} initialized randomly
for e = 1 to E do
                                                                                                // loop over epochs
      for b = 1 to \bar{B} do
                                                                                               // loop over batches
              A^{[0]} = X_h
              for l = 1 to L do
                                                                                                     // forward pass
```

Algorithm 2: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \{(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                              // loop over epochs
      for b = 1 to \bar{B} do
                                                                                            // loop over batches
             A^{[0]} = X_h
             for l = 1 to L do
                                                                                                   // forward pass
                    Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}
                    A^{[l]} = \sigma\left(Z^{[l]}
ight)
             end
```

CHAIN RULE

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \{(\mathbf{W}^{[l]}, \mathbf{b}^{[l]}) : l = 1, ..., L\} initialized randomly
for e = 1 to E do
                                                                                             // loop over epochs
      for b = 1 to \bar{B} do
                                                                                           // loop over batches
             A^{[0]} = X_h
             for l = 1 to L do
                                                                                                  // forward pass
                    Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}
                    A^{[l]} = \sigma \left( Z^{[l]} \right)
             end
             for l = L to 1 do
                                                                                                 // backward pass
```

Algorithm 2: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \left\{\left(oldsymbol{W}^{[l]}, oldsymbol{b}^{[l]}
ight): l=1,\ldots,L\right\} initialized randomly
for e = 1 to E do
                                                                                                              // loop over epochs
       for b = 1 to \bar{B} do
                                                                                                            // loop over batches
                A^{[0]} = X_h
                for l = 1 to L do
                                                                                                                    // forward pass
                        Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}
                       A^{[l]} = \sigma \left( Z^{[l]} \right)
                end
                for l = L to 1 do
                                                                                                                   // backward pass
                        if l = L then
                                                                                                                   // compute error
                               \delta_k^{[l]} = \nabla_{\sigma[l]} \mathcal{L}_k(\Theta) \odot \sigma'(z_k^{[l]})
                                                                                             for k = 1, \ldots, B
                        else if L > l > 1 then
                               \delta_{k}^{[l]} = \begin{bmatrix} W^{[l+1]} \end{bmatrix}^{T} \delta_{k}^{[l+1]} \odot \sigma'(z_{k}^{[l]})
                                                                                             for k = 1, \dots, B
                        end
```

Algorithm 2: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \left\{\left(oldsymbol{W}^{[l]}, oldsymbol{b}^{[l]}
ight): l=1,\ldots,L\right\} initialized randomly
for e = 1 to E do
                                                                                                                    // loop over epochs
        for b = 1 to \bar{B} do
                                                                                                                  // loop over batches
                A^{[0]} = X_h
                 for l = 1 to L do
                                                                                                                          // forward pass
                         Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}
                         A^{[l]} = \sigma \left( Z^{[l]} \right)
                 end
                 for l = L to 1 do
                                                                                                                         // backward pass
                         if l = L then
                                                                                                                         // compute error
                                 \delta_k^{[l]} = \nabla_{\sigma[l]} \mathcal{L}_k(\Theta) \odot \sigma'(z_k^{[l]})
                                                                                                  for k = 1, \ldots, B
                         else if L > l > 1 then
                                 \delta_{k}^{[l]} = \begin{bmatrix} W^{[l+1]} \end{bmatrix}^T \delta_{k}^{[l+1]} \odot \sigma'(z_{k}^{[l]})
                                                                                                  for k = 1, \dots, B
                         end
                         \nabla_{\mathbf{W}[l]} \mathcal{L}_k(\mathbf{\Theta}) = \delta_k^{[l]} \left[ a_k^{[l-1]} \right]^T
                                                                                                   for k = 1, \dots, B
```

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \left\{\left(oldsymbol{W}^{[l]}, oldsymbol{b}^{[l]}
ight): l=1,\ldots,L\right\} initialized randomly
for e = 1 to E do
                                                                                                                                   // loop over epochs
         for b = 1 to \bar{B} do
                                                                                                                                  // loop over batches
                   A^{[0]} = X_h
                   for l = 1 to L do
                                                                                                                                           // forward pass
                            Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}
                            A^{[l]} = \sigma \left( Z^{[l]} \right)
                   end
                   for l = L to 1 do
                                                                                                                                         // backward pass
                            if l = L then
                                                                                                                                         // compute error
                                     \delta_k^{[l]} = \nabla_{\sigma[l]} \mathcal{L}_k(\Theta) \odot \sigma'(z_k^{[l]})
                                                                                                                for k = 1, \ldots, B
                            else if L > l > 1 then
                                     \delta_{k}^{[l]} = \begin{bmatrix} W^{[l+1]} \end{bmatrix}^T \delta_{k}^{[l+1]} \odot \sigma'(z_{k}^{[l]})
                                                                                                               for k = 1, \dots, B
                             end
                            \nabla_{\boldsymbol{W}}[l] \mathcal{L}_k(\boldsymbol{\Theta}) = \boldsymbol{\delta}_{\boldsymbol{k}}^{[l]} \left[ \boldsymbol{a}_{\boldsymbol{k}}^{[l-1]} \right]^T
                                                                                                                for k = 1, \dots, B
                            \nabla_{\mathbf{b}}[l] \mathcal{L}_k(\mathbf{\Theta}) = \delta_{i}^{[l]}
                                                                                                                for k = 1, \ldots, B
                            W^{[l]} := W^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{W^{[l]}} \mathcal{L}_k(\Theta)
```

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \left\{\left(oldsymbol{W}^{[l]}, oldsymbol{b}^{[l]}
ight): l=1,\ldots,L\right\} initialized randomly
for e = 1 to E do
                                                                                                                                            // loop over epochs
          for b = 1 to \bar{B} do
                                                                                                                                          // loop over batches
                    A^{[0]} = X_h
                    for l = 1 to L do
                                                                                                                                                    // forward pass
                              Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}
                              A^{[l]} = \sigma \left( Z^{[l]} \right)
                    end
                    for l = L to 1 do
                                                                                                                                                  // backward pass
                              if l = L then
                                                                                                                                                  // compute error
                                       egin{aligned} oldsymbol{\delta_k^{[l]}} &= 
abla_{oldsymbol{a}[l]} \mathcal{L}_k(oldsymbol{\Theta}) \odot oldsymbol{\sigma'}(oldsymbol{z}_k^{[l]}) \end{aligned}
                                                                                                                       for k = 1, \ldots, B
                              else if L > l > 1 then
                                        \delta_{k}^{[l]} = \begin{bmatrix} W^{[l+1]} \end{bmatrix}^{T} \delta_{k}^{[l+1]} \odot \sigma'(z_{k}^{[l]})
                                                                                                                      for k = 1, \dots, B
                               end
                              \nabla_{\boldsymbol{W}}[l] \mathcal{L}_k(\boldsymbol{\Theta}) = \boldsymbol{\delta}_{\boldsymbol{k}}^{[l]} \left[ \boldsymbol{a}_{\boldsymbol{k}}^{[l-1]} \right]^T
                                                                                                                       for k = 1, \dots, B
                              \nabla_{\mathbf{b}[l]} \mathcal{L}_k(\mathbf{\Theta}) = \delta_{\mathbf{b}}^{[l]}
                                                                                                                       for k = 1, \ldots, B
                              W^{[l]} := W^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{W^{[l]}} \mathcal{L}_k(\Theta)
                                                                                                                                              // update gradient
```

Algorithm 2: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \left\{\left(oldsymbol{W}^{[l]}, oldsymbol{b}^{[l]}
ight): l=1,\ldots,L\right\} initialized randomly
for e = 1 to E do
                                                                                                                                    // loop over epochs
         for b = 1 to \bar{B} do
                                                                                                                                  // loop over batches
                   A^{[0]} = X_h
                   for l = 1 to L do
                                                                                                                                           // forward pass
                            Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}
                            A^{[l]} = \sigma \left( Z^{[l]} \right)
                   end
                   for l = L to 1 do
                                                                                                                                          // backward pass
                            if l = L then
                                                                                                                                          // compute error
                                     \delta_{k}^{[l]} = \nabla_{[l]} \mathcal{L}_{k}(\Theta) \odot \sigma'(z_{k}^{[l]})
                                                                                                                for k = 1, \ldots, B
                            else if L > l > 1 then
                                      \boldsymbol{\delta_k^{[l]}} = \begin{bmatrix} W^{[l+1]} \end{bmatrix}^T \boldsymbol{\delta_k^{[l+1]}} \odot \boldsymbol{\sigma'(z_k^{[l]})}
                                                                                                               for k = 1, \dots, B
                             end
                            \nabla_{\mathbf{W}[l]} \mathcal{L}_k(\mathbf{\Theta}) = \delta_k^{[l]} \left[ a_k^{[l-1]} \right]^T
                                                                                                                for k = 1, \dots, B
                            \nabla_{\mathbf{b}[l]} \mathcal{L}_k(\mathbf{\Theta}) = \delta_{\mathbf{b}}^{[l]}
                                                                                                                for k = 1, \ldots, B
                            m{W}^{[l]} := m{W}^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{m{W}[l]} \mathcal{L}_{k}(\Theta)
                                                                                                                                      // update gradient
                            b^{[l]} := b^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{\mathbf{r}[l]} \mathcal{L}_{k}(\boldsymbol{\Theta})
                                                                                                                                      // update gradient
```

Algorithm 2: Backpropagation (stochastic gradient descent)

```
Data: dataloader = \{B_b = (X_b, Y_b) : b = 1, \dots, \bar{B}\}
Inputs: MLP = \left\{\left(oldsymbol{W}^{[l]}, oldsymbol{b}^{[l]}
ight): l=1,\ldots,L\right\} initialized randomly
for e = 1 to E do
                                                                                                                                // loop over epochs
         for b = 1 to \bar{B} do
                                                                                                                              // loop over batches
                  A^{[0]} = X_h
                  for l = 1 to L do
                                                                                                                                       // forward pass
                            Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}
                           A^{[l]} = \sigma \left( Z^{[l]} \right)
                  end
                  for l = L to 1 do
                                                                                                                                     // backward pass
                            if l = L then
                                                                                                                                     // compute error
                                    \delta_{k}^{[l]} = \nabla_{[l]} \mathcal{L}_{k}(\Theta) \odot \sigma'(z_{k}^{[l]})
                                                                                                            for k = 1, \ldots, B
                            else if L > l > 1 then
                                    \delta_{h}^{[l]} = \begin{bmatrix} W^{[l+1]} \end{bmatrix}^{T} \delta_{h}^{[l+1]} \odot \sigma'(z_{h}^{[l]})
                                                                                                            for k = 1, \dots, B
                            end
                            \nabla_{\mathbf{W}[l]} \mathcal{L}_k(\mathbf{\Theta}) = \delta_k^{[l]} \left[ a_k^{[l-1]} \right]^T
                                                                                                             for k = 1, \dots, B
                           \nabla_{\mathbf{b}[l]} \mathcal{L}_k(\mathbf{\Theta}) = \delta_{\mathbf{b}}^{[l]}
                                                                                                            for k = 1, \ldots, B
                            m{W}^{[l]} := m{W}^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{m{W}[l]} \mathcal{L}_{k}(\Theta)
                                                                                                                                 // update gradient
                           b^{[l]} := b^{[l]} - \frac{\eta}{B} \cdot \sum_{k=1}^{B} \nabla_{\mathbf{r}[l]} \mathcal{L}_k(\boldsymbol{\Theta})
                                                                                                                                 // update gradient
                  end
         end
end
```

EXEMPLE DE TRAINING VIA BACKROP

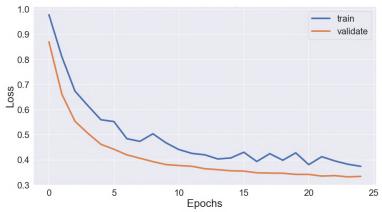


Figure taken from towardsdatascience.com

CHAIN RULE

BIBLIOGRAPHIE



CHAIN RULE

Fleuret, F. (2022).

Deep Learning Course.



Goodfellow, I. J., Bengio, Y., and Courville, A. C. (2016).

Deep Learning.

Adaptive computation and machine learning. MIT Press.



Nielsen, M. A. (2018).

Neural networks and deep learning.



Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986).

Learning representations by back-propagating errors.

Nature. 323(6088):533-536.



Werbos, P. J. (1982).

Applications of advances in nonlinear sensitivity analysis.

In Drenick, R. F. and Kozin, F., editors, System Modeling and Optimization, pages 762-770, Berlin, Heidelberg. Springer Berlin Heidelberg.



Wikipedia contributors (2022).

Backpropagation — Wikipedia, the free encyclopedia.