Introduction à l'intelligence artificielle

F. Vasconcelos, G. Roux

1er novembre 2017

Table des matières

1 Prérequis mathématiques et equivalant en Python

Le package Numpy est indispensable pour manipuler convenablement les vecteurs et les matrices.

1.1 Déclaration d'un vecteur, d'une matrice

```
import numpy as np
X = np.array([[0,0],[0,1],[1,0],[1,1]])
print(X)
```

Ce qui produit la sortie :

```
>>> [[0 0]
 [0 1]
 [1 0]
 [1 1]]
```

Observons tout de suite la syntaxe pour transposer une matrice :

```
print(X.T)
```

Ce qui produit la sortie :

```
[[0 0 1 1]
 [0 1 0 1]]
```

1.2 Le produit matriciel grâce à la fonction dot :

```
L = np.array([ [10, 20, 30, 40] ])
C = np.array([ [2, 4, 6, 8] ]).T
print("L =", L)
print("C =", C)
print("L.C =", np.dot(L, C))
print("C.L =", np.dot(C, L))
```

Sortie :

```
>>> ('L =', array([[10, 20, 30, 40]]))
('C =', array([[2],
       [4],
       [6],
       [8]]))
('L.C =', array([[600]]))
('C.L =', array([[ 20,  40,  60,  80],
       [ 40,  80, 120, 160],
       [ 60, 120, 180, 240],
       [ 80, 160, 240, 320]]))
```

1.3 Génération aléatoire de matrices

```
A = np.random.uniform(size=(2, 5))
B = np.random.uniform(size=(3, 1))

print(A)
print(B)
```

Sortie :

```
>>> >>> [[ 0.84988774  0.84222283  0.71417327  0.54822657  0.45298536]
 [ 0.63462236  0.16929911  0.54541409  0.26605698  0.55474429]]
[[ 0.17312494]
 [ 0.176538  ]
 [ 0.64670851]]
```

1.4 Produit de Hadamard

Définition

Soient $A = \left(a_{ij}\right)$ et $B = \left(b_{ij}\right)$ deux matrices de même dimension. L produit de Hadamard de A et de B, noté $A \times B$ est défini par :

$$A \times B := \left(a_{ij} b_{ij}\right) \tag{1}$$

```
A = np.cumsum(np.ones((2, 5)), axis = 1)
B = 2 * np.ones((1, 5))

print(A)
print(B)
```

```
print(A * B)
```

Sortie :

```
>>> >>> [[ 1.  2.  3.  4.  5.]
 [ 1.  2.  3.  4.  5.]]
[[ 2.  2.  2.  2.  2.]]
>>> [[  2.   4.   6.   8.  10.]
 [  2.   4.   6.   8.  10.]]
```

2 Un premier exemple de réseau de neurones

2.1 Terminologie

Notation

Soit σ une fonction définie sur $\mathbb{R}$ et $M = \left(m_{ij}\right)_{\substack{1\leq i\leq n\\1\leq j\leq p}}$ une matrice à coefficients réels. On notera $\sigma(M)$ la matrice définie par :

$$\sigma(M) := \left(\sigma(m_{ij})\right)_{\substack{1\leq i\leq n\\1\leq j\leq p}} \tag{2}$$

Définition

Soit N un entier naturel. On appelle réseau de neurones à N couches toute suite finie de couples $(W^{(1)}, \sigma^{(1)})$, $(W^{(2)}, \sigma^{(2)})$, ..., $(W^{(N)}, \sigma^{(N)})$, telle que :

1. $W^{(n)} \in \mathcal{M}_{m_{j+1}m_j}((R)$, où $m_0, \ldots, m_N$ est une suite d'entiers positifs, où $\mathcal{M}_{np}((R))$ désigne l'ensemble des matrices à coefficients réels à n lignes et p colonnes.
2. $\sigma^{(n)}$ est une fonction définie (et dérivable) sur $\mathbb{R}$

n_i est la taille de la i^{eme} couche du réseau.

L'entrée du réseau est la colonne notée $X^{(0)} \in \mathcal{M}_{n_0,1}(\mathbb{R})$. On définit par récurrence la suite $X^{(1)}, \ldots, X^{(N)}$ des entrées des neurones ainsi que la suite $Y^{(0)}, \ldots, Y^{(N)}$ des sorties des neurones par :

$$\begin{cases} Y^{(0)} = X^{(0)} \\ \forall n \in \{1, ..., N\}, X^{(n)} = W^{(n)} Y^{(n-1)} \\ \forall n \in \{1, ..., N\}, Y^{(n)} = \sigma^n(X^{(n)}) \end{cases} \tag{3}$$

Si $X^{(0)}$ est l'entrée du réseau de neurones, on appelle sortie la matrice la matrice $Y^{(N)}$.

2.2 Expression de l'erreur

Définition

Notons $T \in \mathcal{M}_{m_N,1}(\mathbb{R})$ la sortie théorique du réseau. On appelle erreur commise par le réseau, et on la note E, le nombre défini par :

$$E := \frac{1}{2}\sum_{i=1}^{m_N}(T_i - Y_i^{(N)})^2 \tag{4}$$

Remarque :

$Y^{(N)}$ est une fonction (dérivable si les $\sigma^{(n)}$ le sont) des $m_1 \times m_0 + \cdots + m_N \times m_{N-1}$ variables $\left\{w_{ij}^{(n)}\right\}$.

2.3 Calcul des dérivées partielle de l'erreur

2.3.1 Par rapport aux coefficients de la dernière matrice

On calcule ici les dérivées partielles du type $\frac{\partial E}{\partial w_{ij}^{(N)}}$.

$$\frac{\partial E}{\partial w_{ij}^{(N)}} = \frac{\partial \frac{1}{2}\sum_{k=1}^{m_N}(t_k - y_k^{(N)})^2}{\partial w_{ij}^{(N)}} \tag{5}$$

$$= \frac{1}{2}\sum_{i=k}^{m_N}\frac{\partial (t_k - y_k^{(k)})^2}{\partial w_{ij}^{(N)}} \tag{6}$$

$$= \sum_{k=1}^{m_N}(t_k - t_k^{(N)})\frac{\partial (t_k - y_k^{(N)})}{\partial w_{ij}^{(N)}} \tag{7}$$

$$= -\sum_{k=1}^{m_N}(t_k - y_k^{(N)})\frac{\partial (\sigma^{(N)}(x_k^{(N)}))}{\partial w_{ij}^{(N)}} \tag{8}$$

$$= -(t_i - y_i^{(N)})\frac{\partial (\sigma^{(N)}(x_i^{(N)}))}{\partial w_{ij}^{(N)}} \tag{9}$$

$$= -(t_i - y_i^{(N)})\sigma^{(N)\prime}(x_i^{(N)})\frac{\partial (x_i^{(N)})}{\partial w_{ij}^{(N)}} \tag{10}$$

$$= -(t_i - y_i^{(N)})\sigma^{(N)\prime}(x_i^{(N)})\frac{\partial (\sum_{k=1}^{n_{N-1}} w_{ik}^{(N)} y_k^{(N-1)})}{\partial w_{ij}^{(N)}} \tag{11}$$

$$= -(t_i - y_i^{(N)})\sigma^{(N)\prime}(x_i^{(N)})y_j^{(N-1)} \tag{12}$$

$$\tag{13}$$

Remarques :
La règle de dérivation des fonctions composées donne :

$$\frac{\partial E}{\partial w_{ij}^{(N)}} = \sum_{k=1}^{m_N}\frac{\partial E}{\partial x_k^{(N)}}\frac{\partial x_k^{(N)}}{\partial w_{ij}^{(N)}} \tag{14}$$

$$= \sum_{k=1}^{m_N}\frac{\partial E}{\partial x_k^{(N)}}\frac{\partial (\sum_{k=1}^{n_{N-1}} w_{ik}^{(N)} y_k^{(N-1)})}{\partial w_{ij}^{(N)}} \tag{15}$$

$$= \frac{\partial E}{\partial x_i^{(N)}}y_j^{(N-1)} \tag{16}$$

Cela pousse à definir la grandeur, appelée signal d'erreur du neurone i de la n^{eme} couche, le nombre :

$$\delta_i^{(n)} = -\frac{\partial E}{\partial x_i^{(n)}} \tag{17}$$

Caculons $\delta_i^{(N)}$:

$$\delta_i^{(N)} = -\frac{\partial E}{\partial x_i^{(N)}} \tag{18}$$

$$= +\sum_{k=1}^{m_N}(t_k - y_k^{(N)})\frac{\partial(\sigma^{(N)}(x_k^{(N)}))}{\partial x_i^{(N)}} \tag{19}$$

$$= +(t_i - y_i^{(N)})\sigma^{(N)\prime}(x_i^{(N)}) \tag{20}$$

Cela permet la notation abrégée et généralisable suivante :

$$\frac{\partial E}{\partial w_{ij}^{(N)}} = -\delta_i^{(N)} y_j^{(N-1)} \tag{21}$$

2.3.2 Par rapport aux coefficients des matrices des couches cachées

Calculons δ_i^n pour $n < N$:

$$\delta_i^{(n)} = -\frac{\partial E}{\partial x_i^{(n)}} \tag{22}$$

$$= +\sum_{k=1}^{m_N}(t_k - y_k^{(n)})\frac{\partial(\sigma^{(N)}(x_k^{(N)}))}{\partial x_i^n} \tag{23}$$

$$= +\sum_{k=1}^{m_N}(t_k - y_k^{(n)})\sum_{l=1}^{m_{n+1}}\frac{\partial\left(\sigma^{(N)}(x_l^{(N)})\right)}{\partial x_l^{n+1}}\frac{\partial x_l^{n+1}}{\partial x_i^{(n)}} \tag{24}$$

$$= \sum_{l=1}^{m_{n+1}}\left(\sum_{k=1}^{m_N}(t_k - y_k^{(n)})\frac{\partial\left(\sigma^{(N)}(x_l^{(N)})\right)}{\partial x_l^{(n+1)}}\right)\frac{\partial x_l^{(n+1)}}{\partial x_i^{(n)}} \tag{25}$$

$$= \sum_{l=1}^{m_{n+1}}\delta_l^{(n+1)}\frac{\partial x_l^{(n+1)}}{\partial x_i^{(n)}} \tag{26}$$

$$= \sum_{l=1}^{m_{n+1}}\delta_l^{(n+1)}\frac{\partial\left(\sum_{k=1}^{m_n} w_{lk}^{(n+1)} y_k^{(n)}\right)}{\partial x_i^{(n)}} \tag{27}$$

$$= \sum_{l=1}^{m_{n+1}}\delta_l^{(n+1)}\frac{\partial\left(\sum_{k=1}^{m_n} w_{lk}^{(n+1)}\sigma^{(n)}(x_k^{(n)})\right)}{\partial x_i^{(n)}} \tag{28}$$

$$= \sum_{l=1}^{m_{n+1}} w_{li}^{n+1}\delta_l^{(n+1)}\sigma^{(n)\prime}(x_i^{(n)}) \tag{29}$$

$$= \sigma^{(n)\prime}(x_i^{(n)})\sum_{l=1}^{m_{n+1}} w_{li}^{n+1}\delta_l^{(n+1)} \tag{30}$$

La relation ci-dessus est très importante dans la propagation de l'erreur. Calculons $\frac{\partial E}{\partial w_{ij}^{(n)}}$:

$$\frac{\partial E}{\partial w_{ij}^{(n)}} = \sum_{k=1}^{m_N} \frac{\partial E}{\partial x_k^{(n)}} \frac{\partial x_k^{(n)}}{\partial w_{ij}^{(n)}} \tag{31}$$

$$= \sum_{k=1}^{m_N} \frac{\partial E}{\partial x_k^{(n)}} \frac{\partial (\sum_{k=1}^{n_{n-1}} w_{ik}^{(n)} y_k^{(n-1)})}{\partial w_{ij}^{(n)}} \tag{32}$$

$$= \frac{\partial E}{\partial x_i^{(n)}} y_j^{(n-1)} \tag{33}$$

$$= -\delta_i^{(n)} y_j^{(n-1)} \tag{34}$$

3 Mise à jour des matrices de poids - Rétropropagation

On chercha à trouver les coefficients $\left\{w_{ij}^{(n)}\right\}$ qui minimise l'erreur. On sait l'erreur E diminue le plus rapidement dans la direction donnée par l'opposé de son gradient.

Autrement dit, à chaque $w_{ij}^{(n)}$ on va ajouter $dw_{ij}^{(n)} = \delta_i^{(n)} y_j^{(n-1)}$.

3.1 Récapitulatif des formules en vue de l'implémentation

Les formules suivantes peuvent être implémentées quasiment telles quelles :

3.1.1 Version entrée en colonne

On note : $\delta^{(n)} = \begin{pmatrix} \delta_1^{(n)} \\ \delta_2^{(n)} \\ \dots \\ \delta_{m_n}^{(n)} \end{pmatrix}$, $L_i^{(n)}$ la i^{eme} de $W^{(n)}$ et $C_j^{(n)}$ sa j^{eme} colonne.

Dernière couche

1. $\delta_i^{(N)} = \sigma^{(N)\prime}(x_i^{(N)})(t_i - y_i^{(N)})$
2. $dw_{ij}^{(N)} = \sigma^{(N)\prime}(x_i^{(N)})(t_i - y_i^{(N)}) y_j^{(N-1)}$.

Couche intermédiaire

1.

$$\delta_i^{(n)} = \sigma^{(n)\prime}(x_i^{(n)}) \sum_{l=1}^{m_{n+1}} w_{li}^{(n+1)} \delta_l^{(n+1)} \tag{35}$$

$$= \sigma^{(n)\prime}(x_i^{(n)})^t C_i^{(n+1)}.\delta^{(n+1)} \tag{36}$$

2.

$$dw_{ij}^{(n)} = -\sigma^{(n)\prime}(x_i^{(n)}) \sum_{l=1}^{m_{n+1}} w_{li}^{(n+1)} \delta_l^{(n+1)} y_j^{(n-1)} \tag{37}$$

$$= \sigma^{(n)\prime}(x_i^{(n)})^t C_i^{(n+1)}.\delta^{(n+1)} y_j^{(n-1)} \tag{38}$$

Première couche

1.

$$\delta_i^{(1)} = \sigma^{(1)\prime}(x_i^{(1)}) \sum_{l=1}^{m_2} w_{li}^{(2)} \delta_l^{(2)} \tag{39}$$

$$= \sigma^{(1)\prime}(x_i^{(1)})^t C_i^{(2)}.\delta^{(2)} \tag{40}$$

2.

$$dw_{ij}^{(1)} = -\sigma^{(1)\prime}(x_i^{(1)}) \sum_{l=1}^{m_2} w_{li}^{(2)} \delta_l^{(2)} x_j^{(0)} \tag{41}$$

$$= \sigma^{(1)\prime}(x_i^{(1)})^t C_i^{(2)}.\delta^{(2)} x_j^{(0)} \tag{42}$$

3.1.2 Version transposée

Il peut être préférable de considéer l'entrée et les différentes couches du réseau comme des lignes plutôt que comme des colonnes. Il suffit pour cela de transposer toutes les matrices dans ce qui a été fait précedemment. On note alors que le produit à gauche devient un produit à droite, pour passer d'une couche à la suivante.

Les formules ci-dessus deviennent alors :

Dernière couche

1. $\delta_j^{(N)} = \sigma^{(N)\prime}(x_j^{(N)})(t_j - y_j^{(N)})$
2. $dw_{ij}^{(N)} = \sigma^{(N)\prime}(x_j^{(N)})(t_j - y_j^{(N)}) y_i^{(N-1)}$.

Couche intermédiaire

1. $\delta_j^{(n)} = \sigma^{(n)\prime}(x_j^{(n)}) \sum_{l=1}^{p_{n+1}} w_{jl}^{(n+1)} \delta_l^{(n+1)} = \sigma^{(n)\prime}(x_j^{(n)}) \delta^{(n)} . {}^t L_j^{(n)}$
2. $dw_{ij}^{(n)} = \sigma^{(n)\prime}(x_j^{(n)}) \sum_{l=1}^{p_{n+1}} w_{jl}^{(n+1)t} \delta_l^{(n+1)} y_i^{(n-1)} = \sigma^{(n)\prime}(x_j^{(n)}) \delta^{(n+1)} . {}^t L_j^{(n+1)} y_i^{(n-1)}$.

Première couche

1. $\delta_j^{(1)} = \sigma^{(1)\prime}(x_j^{(1)}) \sum_{l=1}^{p_2} w_{jl}^{(21)} \delta_l^{(2)} = \sigma^{(1)\prime}(x_i^{(1)}) \delta^{(1)} . {}^t L_j^{(1)}$
2. $dw_{ij}^{(1)} = \sigma^{(1)\prime}(x_j^{(1)}) \sum_{l=1}^{p_2} w_{jl}^{(2)} \delta_l^{(2)} x_i^{(0)} = \sigma^{(1)\prime}(x_i^{(1)}) \delta^{(2)} . {}^t L_j^{(2)} x_i^{(0)}$.

On peut exprimer cela matriciellement :

Dernière couche

1. $\delta^{(N)} = \sigma^{(N)\prime}(X^{(N)}) \times (T - Y^{(N)})$
2. $dw^{(N)} =^t Y^{(N-1)} . \delta^{(N)}$

Couche intermédiaire

1. $\delta^{(n)} = \sigma^{(n)\prime}(X^{(N)}) \times (\delta^{(n+1)} . {}^t W^{(n+1)})$
2. $dw^{(n)} =^t Y^{(n-1)} . \delta^{(n)}$

Première couche

1. $\delta^{(1)} = \sigma^{(1)\prime}(X^{(N)}) \times (\delta^{(2)} . {}^t W^{(2)})$
2. $dw^{(1)} =^t X^{(0)} . \delta^{(1)}$

3.1.3 Remarques

Le vecteur $\delta^{(n)}$ est l'opposé du gradient de l'erreur E, lorsque cette dernière est exprimée en fonction des $x_i^{(n)}$. Autrement dit :

$$\delta^{(n)} = -\nabla \left(E(x_1^{(n)}, \ldots, x_{m_n}^{(n)}) \right) \tag{43}$$

4 Exemple

4.1 Les mathématiques

On va utiliser la version transposée avec le cas suivant :

— $N = 2$
— $\sigma^n = \sigma$
— $(n_1, n_0) = (3, 2)$ donc $W^{(1)} \in \mathcal{M}_{2,3}(\mathbb{R})$
— $(n_2, n_1) = (1, 3)$ donc $W^{(2)} \in \mathcal{M}_{3,1}(\mathbb{R})$
— X_0 prendra successivement pour valeurs :
  — $(0, 0)$
  — $(0, 1)$

— $(1,0)$
— $(1,1)$

— T prendra successivement pour valeurs :
— 0
— 1
— 1
— 0

On obtient :

1. $\delta^{(2)} = \sigma^{(2)\prime}(X^{(2)}) \times (T - Y^{(2)})$
2. $\delta^{(1)} = \sigma^{(1)\prime}(X^{(2)}) \times (\delta^{(2)}.{}^{t}W^{(2)})$

1. $dw^{(2)} = -{}^{t}Y^{(1)}.\delta^{(2)}$
2. $dw^{(1)} = -{}^{t}X^{(0)}.\delta^{(1)}$

Remarque :

$$\sigma'(X^{(n)}) = \sigma(X^{(n)})(1 - \sigma(X^{(n)})) \tag{44}$$
$$= Y^{(n)}(1 - Y^{(n)}) \tag{45}$$
$$= \sigma\_(Y^{(n)}) \tag{46}$$

4.2 Implémentations en python

4.2.1 Version basique

```
# coding: utf-8
# XOR basique
import numpy as np

iterations = 6000                # Nombre d'itérations

tailleX0, tailleX1, tailleX2 = 2, 3, 1

X0 = np.array([[0,0], [0,1], [1,0], [1,1]])
T = np.array([ [0],   [1],   [1],   [0]])

def sigmoide (x):
    return 1/(1 + np.exp(-x))    # fonction d'activation
def sigmoide_(x):
    return x * (1 - x)           # dérivée de la fonction d'activation
```

```
# Poids
W1 = np.random.uniform(size=(tailleX0, tailleX1))
W2 = np.random.uniform(size=(tailleX1,tailleX2))

for i in range(iterations):

    X1 = np.dot(X0, W1)                 # entrée couche 1
    Y1 = sigmoide(X1)                   # activation couche 1
    X2 = np.dot(Y1, W2)                 # entrée couche 2
    Y2 = sigmoide(X2)                   # activation couche 2

    E = T - Y2                          # erreur

    d2 = sigmoide_(Y2) * E              # d2
    d1 = sigmoide_(Y1) * d2.dot(W2.T)   # d1

    dW1 = Y1.T.dot(d2)                  # somme sur les entrées des dW1
    dW2 = X0.T.dot(d1)                  # somme sur les entrées des dW2

    W2 += dW1                           # mise à jour des poides de la couche 2
    W1 += dW2                           # et des poids de la couche 1

print(Y2)
```

```
[[ 0.04418611]
 [ 0.97282793]
 [ 0.97282303]
 [ 0.01031442]]
```

4.2.2 Version améliorée

Le but de l'amélioration qui va suivre est d'écrire des fonctions réutilisables pour résoudre des problèmes plus compliqués.

```
# coding: utf-8
# XOR amélioré
import numpy as np

iterations = 6000                # Nombre d'itérations
```

```
tailleX0, tailleX1, tailleX2 = 2, 3, 1

X0 = np.array([[0,0], [0,1], [1,0], [1,1]])
T = np.array([ [0],   [1],   [1],   [0]])

def sigmoide (x):
    return 1/(1 + np.exp(-x))    # fonction d'activation
def sigmoide_(x):
    return x * (1 - x)           # dérivée de la fonction d'activation

# Poids
W1 = np.random.uniform(size=(tailleX0, tailleX1))
W2 = np.random.uniform(size=(tailleX1,tailleX2))

def traiter(X0):
    X1 = np.dot(X0, W1)                 # entrée couche 1
    Y1 = sigmoide(X1)                   # activation couche 1
    X2 = np.dot(Y1, W2)                 # entrée couche 2
    Y2 = sigmoide(X2)                   # activation couche 2
    return Y1, Y2

for i in range(iterations):
    Y1, Y2 = traiter(X0)

    E = T - Y2                          # erreur

    d2 = sigmoide_(Y2) * E              # d2
    d1 = sigmoide_(Y1) * d2.dot(W2.T)   # d1

    dW1 = Y1.T.dot(d2)                  # somme sur les entrées des dW1
    dW2 = X0.T.dot(d1)                  # somme sur les entrées des dW2

    W2 += dW1                           # mise à jour des poides de la couche 2
    W1 += dW2                           # et des poids de la couche 1

print(Y2)
```

```
[[ 0.04272965]
 [ 0.97370503]
 [ 0.97370852]
```

```
[ 0.01002294]]
```

4.3 Le OU exclusif

$$W^1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

4.4 La fonction d'activation : sigmoïde

On considère la fonction f définie sur $\mathbb{R}$ par $f(x) = \frac{1}{1+e^{-ax}}$, où a est un réel strictement positif.

f est de classe C^∞ et on a :

$$f'(x) = -\frac{-ae^{-ax}}{(1+e^{-ax})^{(2)}} \tag{47}$$

$$= a\frac{e^{-ax}}{(1+e^{-ax})^{(2)}} \tag{48}$$

On constate que f est solution de l'équation différentielle $y' = ay(1-y)$. En effet :

$$f(x)(1-f(x)) = \frac{1}{1+e^{-ax}}(1 - \frac{1}{1+e^{-ax}}) \tag{49}$$

$$= \frac{1}{1+e^{-ax}}(\frac{1+e^{-ax}-1}{1+e^{-ax}}) \tag{50}$$

$$= \frac{e^{-ax}}{(1+e^{-ax})^{(2)}} \tag{51}$$

$$= \frac{1}{a}f'(x) \tag{52}$$