

Introduction à l'intelligence artificielle

F. Vasconcelos, G. Roux

3 novembre 2017

Table des matières

1	Introduction	2
2	Le OU exclusif	2
3	Prérequis mathématiques et equivalent en Python	2
3.1	Déclaration d'un vecteur, d'une matrice	2
3.2	Le produit matriciel grâce à la fonction dot :	3
3.3	Génération aléatoire de matrices	3
3.4	Produit de Hadamard	3
4	Un premier exemple : un réseau de neurones de type "feed forward"	4
4.1	Terminologie	4
4.2	Expression de l'erreur	4
4.3	Calcul des dérivées partielles de l'erreur	5
4.3.1	Par rapport aux coefficients de la dernière matrice	5
4.3.2	Par rapport aux coefficients des matrices des couches cachées	6
5	Optimisation des matrices de poids - Rétropropagation	8
5.1	Récapitulatif des formules en vue de l'implémentation	8
5.1.1	Version entrée en colonne	8
5.1.2	Version transposée	9
5.1.3	Remarques	10
5.2	Sigmoïdes - Fonctions d'activation usuelles	10

6	Exemple	11
6.1	Les mathématiques	11
6.2	Implémentations en python	12
6.2.1	Version basique	12
6.2.2	Version améliorée	13
7	Améliorations possibles	13
7.1	Taux d'apprentissage	13
7.2	Biais	13
8	Exercices	14
8.1	Quadrants	14
8.2	Implémentation du biais	14

1 Introduction

2 Le OU exclusif

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

3 Prérequis mathématiques et équivalent en Python

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

3.1 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 :

Observons tout de suite la syntaxe pour transposer une matrice :

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

Ce qui produit la sortie :

3.2 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 :

3.3 Génération aléatoire de matrices

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

print(A)
print(B)
```

Sortie :

3.4 Produit de Hadamard

Définition

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

$$A \times B := (a_{ij}b_{ij}) \quad (1)$$

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

print(A)
print(B)

print(A * B)
```

Sortie :

4 Un premier exemple : un réseau de neurones de type "feed forward"

4.1 Terminologie

Notation

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

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

Définition

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

1. $W^{(n)} \in \mathcal{M}_{m_{j+1}m_j}(\mathbb{R})$, où m_0, \dots, m_N est une suite d'entiers positifs, où $\mathcal{M}_{np}(\mathbb{R})$ 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 $X^{(1)}, \dots, X^{(N)}$ des entrées des neurones ainsi que la suite $Y^{(0)}, \dots, Y^{(N)}$ des sorties des neurones par :

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

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

4.2 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 :

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

Remarque :

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

4.3 Calcul des dérivées partielles de l'erreur

4.3.1 Par rapport aux coefficients de la dernière matrice

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

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

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

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

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

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

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

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

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

$$(13)$$

Remarques :

La règle de dérivation des fonctions composées donne :

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

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

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

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

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

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

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

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

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

Cela permet la notation abrégée et généralisable suivante :

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

4.3.2 Par rapport aux coefficients des matrices des couches cachées

Calculons δ_i^n pour $n < N$:

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

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

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

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

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

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

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

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

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

La relation ci-dessus est très importante dans la propagation de l'erreur.

Calculons $\frac{\partial E}{\partial w_{ij}^{(n)}}$:

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

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

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

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

5 Optimisation des matrices de poids - Rétropropagation

On chercha à trouver les coefficients $\{w_{ij}^{(n)}\}$ 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 $dw_{ij}^{(n)} = \delta_i^{(n)} y_j^{(n-1)}$.

5.1 Récapitulatif des formules en vue de l'implémentation

Les formules suivantes peuvent être implémentées quasiment telles quelles :

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

Couche intermédiaire

1.

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

$$= \sigma^{(n)'}(x_i^{(n)})^t C_i^{(n+1)} \cdot \delta^{(n+1)} \quad (36)$$

2.

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

$$= \sigma^{(n)'}(x_i^{(n)})^t C_i^{(n+1)} \cdot \delta^{(n+1)} y_j^{(n-1)} \quad (38)$$

Première couche

1.

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

$$= \sigma^{(1)'}(x_i^{(1)})^t C_i^{(2)} \cdot \delta^{(2)} \quad (40)$$

2.

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

$$= \sigma^{(1)'}(x_i^{(1)})^t C_i^{(2)} \cdot \delta^{(2)} x_j^{(0)} \quad (42)$$

5.1.2 Version transposée

Il peut être préférable de considérer 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 précédemment. 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. $\delta_j^{(N)} = \sigma^{(N)'}(x_j^{(N)})(t_j - y_j^{(N)})$
2. $dw_{ij}^{(N)} = \sigma^{(N)'}(x_j^{(N)})(t_j - y_j^{(N)}) y_i^{(N-1)}$.

Couche intermédiaire

1. $\delta_j^{(n)} = \sigma^{(n)'}(x_j^{(n)}) \sum_{l=1}^{p_{n+1}} w_{jl}^{(n+1)} \delta_l^{(n+1)} = \sigma^{(n)'}(x_j^{(n)}) \delta^{(n)} \cdot {}^t L_j^{(n)}$
2. $dw_{ij}^{(n)} = \sigma^{(n)'}(x_j^{(n)}) \sum_{l=1}^{p_{n+1}} w_{jl}^{(n+1)t} \delta_l^{(n+1)} y_i^{(n-1)} = \sigma^{(n)'}(x_j^{(n)}) \delta^{(n+1)} \cdot {}^t L_j^{(n+1)} y_i^{(n-1)}$.

Première couche

1. $\delta_j^{(1)} = \sigma^{(1)'}(x_j^{(1)}) \sum_{l=1}^{p_2} w_{jl}^{(21)} \delta_l^{(2)} = \sigma^{(1)'}(x_i^{(1)}) \delta^{(1)} \cdot {}^t L_j^{(1)}$
2. $dw_{ij}^{(1)} = \sigma^{(1)'}(x_j^{(1)}) \sum_{l=1}^{p_2} w_{jl}^{(2)} \delta_l^{(2)} x_i^{(0)} = \sigma^{(1)'}(x_i^{(1)}) \delta^{(2)} \cdot {}^t L_j^{(2)} x_i^{(0)}$.

On peut exprimer cela matriciellement :

Dernière couche

1. $\delta^{(N)} = \sigma^{(N)'}(X^{(N)}) \times (T - Y^{(N)})$
2. $dw^{(N)} = {}^t Y^{(N-1)} \cdot \delta^{(N)}$

Couche intermédiaire

1. $\delta^{(n)} = \sigma^{(n)'}(X^{(N)}) \times (\delta^{(n+1)} \cdot {}^t W^{(n+1)})$
2. $dw^{(n)} = {}^t Y^{(n-1)} \cdot \delta^{(n)}$

Première couche

1. $\delta^{(1)} = \sigma^{(1)'}(X^{(N)}) \times (\delta^{(2)} \cdot {}^t W^{(2)})$
2. $dw^{(1)} = {}^t X^{(0)} \cdot \delta^{(1)}$

5.1.3 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 :

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

5.2 Sigmoïdes - Fonctions d'activation usuelles

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 :

$$f'(x) = -\frac{-ae^{-ax}}{(1+e^{-ax})^2} \quad (44)$$

$$= a \frac{e^{-ax}}{(1+e^{-ax})^2} \quad (45)$$

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

$$f(x)(1 - f(x)) = \frac{1}{1 + e^{-ax}} \left(1 - \frac{1}{1 + e^{-ax}}\right) \quad (46)$$

$$= \frac{1}{1 + e^{-ax}} \left(\frac{1 + e^{-ax} - 1}{1 + e^{-ax}}\right) \quad (47)$$

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

$$= \frac{1}{a} f'(x) \quad (49)$$

6 Exemple

6.1 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. $\delta^{(2)} = \sigma^{(2)'}(X^{(2)}) \times (T - Y^{(2)})$
2. $\delta^{(1)} = \sigma^{(1)'}(X^{(2)}) \times (\delta^{(2)}.{}^tW^{(2)})$
1. $dw^{(2)} = -{}^tY^{(1)}.\delta^{(2)}$
2. $dw^{(1)} = -{}^tX^{(0)}.\delta^{(1)}$

Remarque :

$$\sigma'(X^{(n)}) = \sigma(X^{(n)})(1 - \sigma(X^{(n)})) \quad (50)$$

$$= Y^{(n)}(1 - Y^{(n)}) \quad (51)$$

$$= \sigma_-(Y^{(n)}) \quad (52)$$

6.2 Implémentations en python

6.2.1 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
```

```

dW2 = Y1.T.dot(d2)                # somme sur les entrées des dW1
W2 += dW2                          # mise à jour des poids de la couche 2

d1 = sigmoide_(Y1) * d2.dot(W2.T) # d1
dW1 = X0.T.dot(d1)                 # somme sur les entrées des dW2
W1 += dW1                           # et des poids de la couche 1

print(Y2)

```

6.2.2 Version améliorée

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

7 Améliorations possibles

7.1 Taux d'apprentissage

7.2 Biais

Introduire un biais signifie transformer l'écriture $Y^{(n)} = \sigma^n(X^{(n)})$ en $Y^{(n)} = \sigma^n(X^{(n)} + B^{(n)})$.

A nouveau, calculons, en notations colonnes, les dérivées partielles de l'erreur en fonctions des biais de la dernière couche ($n = N$) :

$$\frac{\partial E}{\partial b_i^{(N)}} = \sum_{k=1}^{m_N} \frac{\partial E}{\partial x_k^{(N)}} \frac{\partial x_k^{(N)}}{\partial b_i^{(N)}} \quad (53)$$

$$= \sum_{k=1}^{m_N} \frac{\partial E}{\partial x_k^{(N)}} \frac{\partial (\sum_{l=1}^{n_N-1} w_{il}^{(N)} y_l^{(N-1)} + b_i^{(N)})}{\partial b_i^{(N)}} \quad (54)$$

$$= \sum_{k=1}^{m_N} \frac{\partial E}{\partial x_k^{(N)}} \cdot 1 \quad (55)$$

$$= - \sum_{k=1}^{m_N} \delta_k^N \quad (56)$$

L'expression ci-dessus ne dépend pas de i .
Pour une couche quelconque :

$$\frac{\partial E}{\partial b_i^{(n)}} = \sum_{k=1}^{m_{n+1}} \frac{\partial E}{\partial x_k^{(n)}} \frac{\partial x_l^{(n)}}{\partial b_i^{(n)}} \quad (57)$$

$$= \sum_{k=1}^{m_{n+1}} \frac{\partial E}{\partial x_k^{(n)}} \frac{\partial \left(\sum w_{kl}^{(n-1)} y_l^{(n-1)} + b_l^{(n)} \right)}{\partial b_i^{(n)}} \quad (58)$$

$$= \sum_{k=1}^{m_{n+1}} \frac{\partial E}{\partial x_k^{(n)}} \cdot 1 \quad (59)$$

$$= - \sum_{k=1}^{m_{n+1}} \delta_k^{(n)} \quad (60)$$

8 Exercices

8.1 Quadrants

Implémenter un réseau de neurones capable de déterminer si un point généré aléatoirement se situe dans le premier, le deuxième, le troisième ou le quatrième quadrant du plan.

Le réseau doit produire les sorties suivantes :

- [1,0,0,0] si le point est dans le premier quadrant ($x > 0$ et $y > 0$)
- [0,1,0,0] si le point est dans le deuxième quadrant ($x < 0$ et $y > 0$)
- [0,0,1,0] si le point est dans le troisième quadrant ($x < 0$ et $y < 0$)
- [0,0,0,1] si le point est dans le quatrième quadrant ($x > 0$ et $y < 0$)

8.2 Implémentation du biais