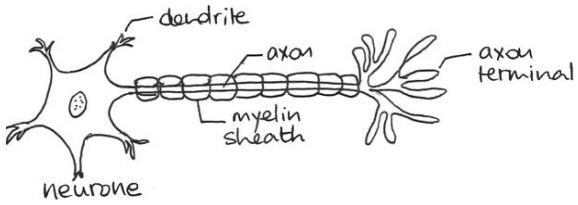
Machine Learning : le perceptron un premier réseau de neurones

Résumé :

Le perceptron est un réseau de neurones très simple qui permet de résoudre des problèmes de classifications à deux classes. Il est à la base de réseaux plus complexes.

Son principe est 'd'imiter' le fonctionnement d'un vrai neurone.



Les réponses aux questions seront renseignées sur la feuille réponse.

Sommaire :

Pour introduire le sujet deux vidéos :

(1) Naissance_du_premier_neurone_artificiel_IA_en_5_min.mp4

(2) LedeeplearningScienceetonnante27.mp4

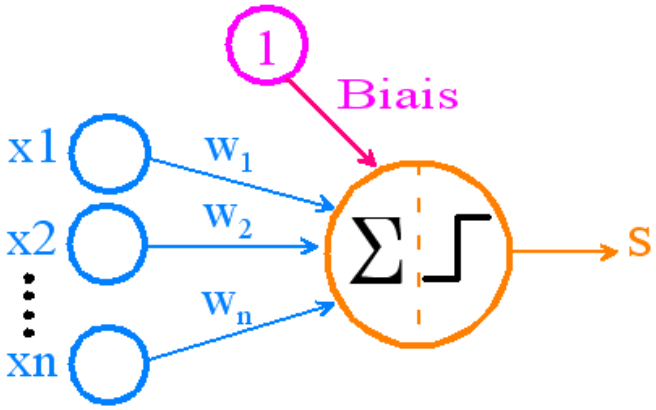
Et après un mini projet :

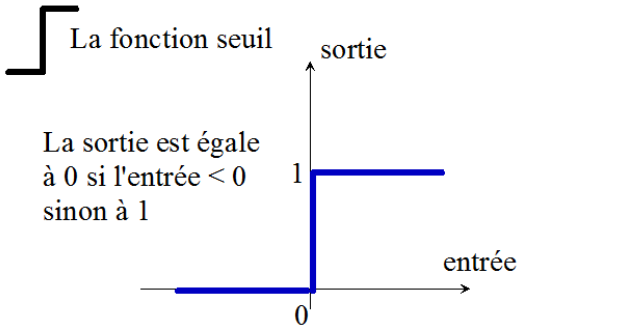
Machine Learning : utilisation du perceptron appliquée à la chasse aux mines marines

1 Le neurone artificiel

1.1 Présentation

Une réalisation du neurone artificiel est présentée ci-dessous :





La somme pondérée

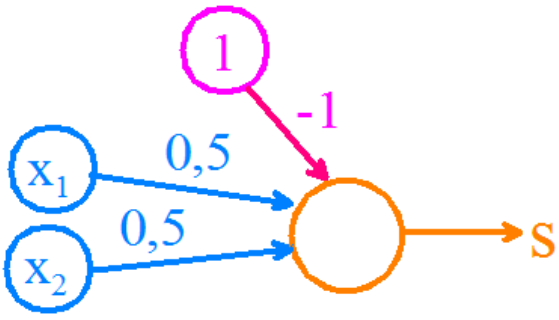
$$\sum = Biais + x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots + x_n \cdot w_n$$

Le neurone possède n entrées x_1 -- x_n. Chaque entrée intervient dans le calcul de l'entrée de la fonction d'activation pondérée par un poids w_i. Un biais intervient (également noté w_0). Si la somme calculée est supérieure à 0 alors la sortie du neurone est égale 1 sinon elle est égale à 0.

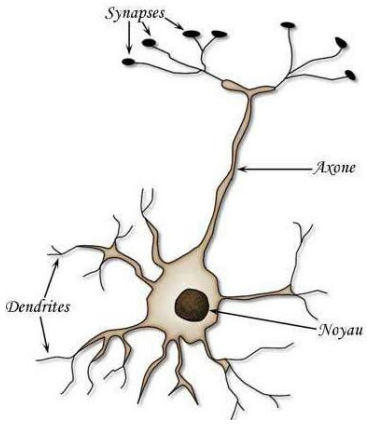
A partir de la vidéo 1[1]

Q1. Expliquer en quoi le fonctionnement de ce 'neurone' artificiel est relativement similaire au fonctionnement du neurone biologique.

Q2. x1 et x2 sont deux variables logiques. Quelle est la fonction réalisée par le neurone ci-dessous :



x_1	x_2	S
0	0	
0	1	
1	0	
1	1	



⇦

[1] A voir sur youtube : https://www.youtube.com/watch?v=RPpq9EncIQs

1.2 Implémentation

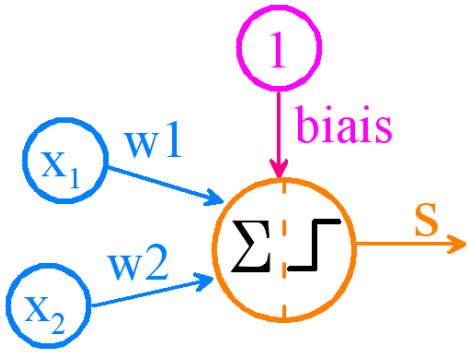
Pour implanter ce premier neurone référons-nous au schéma ci-contre, le calcul se fait en deux étapes :

Étape 1 : Le calcul de la somme pondérée appelée également activation s'écrit :

$$activation = bias + \sum_{i=1}^{n} weight_i \times x_i$$

Étape 2 : La sortie du neurone S qui réalisera la prédiction de notre modèle est calculée par :

$$prediction = 1.0 \text{ IF } activation \geq 0.0 \text{ ELSE } 0.0$$



Notre unique neurone pourra réaliser des prédictions si les valeurs des poids *weights* w_i sont définies, en fonction du problème à résoudre, par un processus d'entrainement que nous verrons par la suite.

Pour le moment nous allons implanter ce neurone avec les poids suivants :

weights = [w0, w1, w2] weights = [-0.1, 0.206, -0.234]

Les données *row* seront présentées sous la forme : row = [1, x1, x2, y]

Où x1 et x2 sont les valeurs des entrées et y la sortie attendue. Notre modèle de neurone devra prédire cette valeur de y.

Calcul de quelques valeurs 'à la main' :

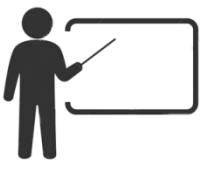
Q3. Calculer la valeur activation et prédiction pour les données suivantes :
weights = [-0.1, 0.206, -0.234]
row = [1, 1.465, 2.362, 0]

Q4. La valeur prédite est-elle bien égale à celle attendue ?

Q5. Calculer la valeur activation et prédiction pour les données suivantes :
weights = [-0.1, 0.206, -0.234]
row = [1, 7.673, 3.508, 1]

Q6. La valeur prédite est-elle bien égale à celle attendue ?

Script_Perceptron_1. Compléter le script Script_Perceptron_1.py pour faire fonctionner votre premier perceptron.

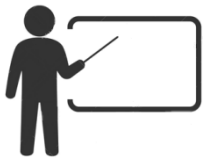


1.3 Travail sur un jeu de test

Maintenant nous allons utiliser notre fonction *predict* sur un jeu de 10 vecteurs de tests.

```
# test predictions sur un jeux de 10 échantillons
dataset = [
    [1, 2.781,2.550,0],
    [1, 1.465,2.362,0],
    [1, 3.396,4.401,0],
    [1, 1.388,1.851,0],
    [1, 3.064,3.005,0],
    [1, 7.627,2.759,1],
    [1, 5.332,2.089,1],
    [1, 6.922,1.771,1],
    [1, 8.675,-0.242,1],
    [1, 7.673,3.508,1]
    ]
```

Script_Perceptron_2. Compléter le script Script_Perceptron_2.py pour traiter le lot de données et obtenir l'affichage des résultats comme ci-dessous.

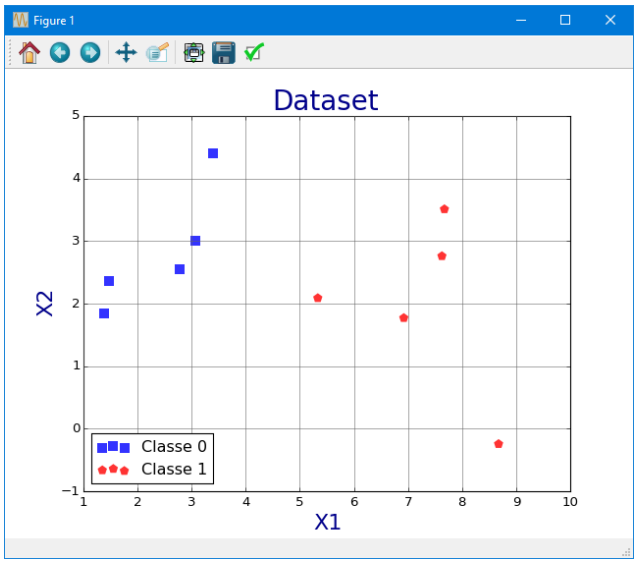


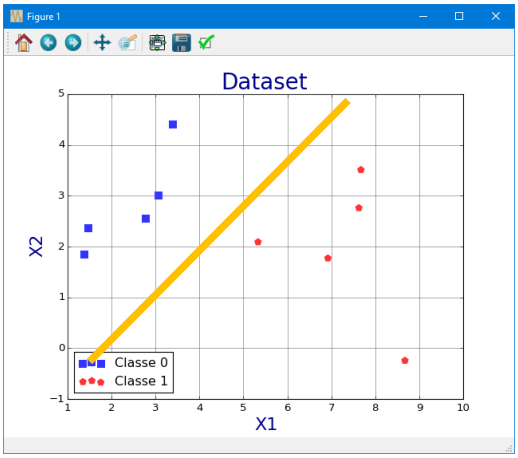
```
Essais du perceptron sur un groupe de 10 vecteurs de tests

 2.781  2.550  0  Sortie attendue=0, Prédite=0 Prédiction:  OK
 1.465  2.362  0  Sortie attendue=0, Prédite=0 Prédiction:  OK
 3.396  4.401  0  Sortie attendue=0, Prédite=0 Prédiction:  OK
 1.388  1.851  0  Sortie attendue=0, Prédite=0 Prédiction:  OK
 3.064  3.005  0  Sortie attendue=0, Prédite=0 Prédiction:  OK
 7.627  2.759  1  Sortie attendue=1, Prédite=1 Prédiction:  OK
 5.332  2.089  1  Sortie attendue=1, Prédite=1 Prédiction:  OK
 6.922  1.771  1  Sortie attendue=1, Prédite=1 Prédiction:  OK
 8.675 -0.242  1  Sortie attendue=1, Prédite=1 Prédiction:  OK
 7.673  3.508  1  Sortie attendue=1, Prédite=1 Prédiction:  OK
```

1.4 Visualisation des données

Les données sont réparties en deux classes. Nous pouvons donc les visualiser avec l'aide de matplotlib.

Voilà le résultat à obtenir :

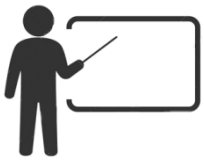




A vous de réaliser le tracé :

Le perceptron peut réaliser une classification linéaire en deux classes d'un jeu de données. Le plan est découpé par une droite en deux parties.

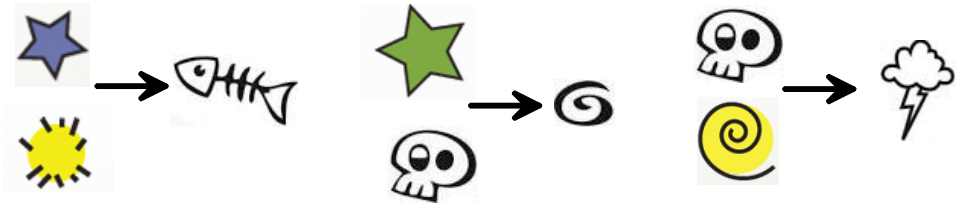
Script_Perceptron_3. Compléter le script Script_Perceptron_3.py pour traiter le lot de données et obtenir l'affichage des résultats comme ci-dessous.



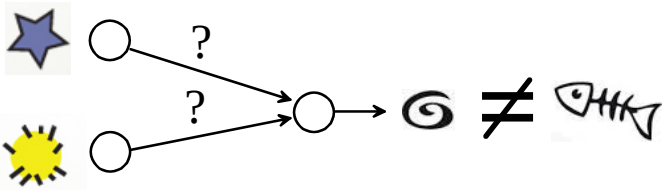
2 Apprentissage du neurone calcul des poids

2.1 Principe de la rétro propagation pour le calcul des poids

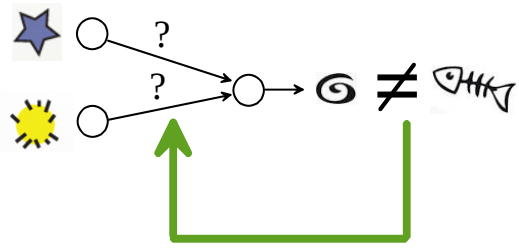
Nous voulons 'enseigner' le réseau pour qu'il puisse résoudre le problème ci-dessous :



Pour cela nous présentons successivement toutes les combinaisons d'entrées :



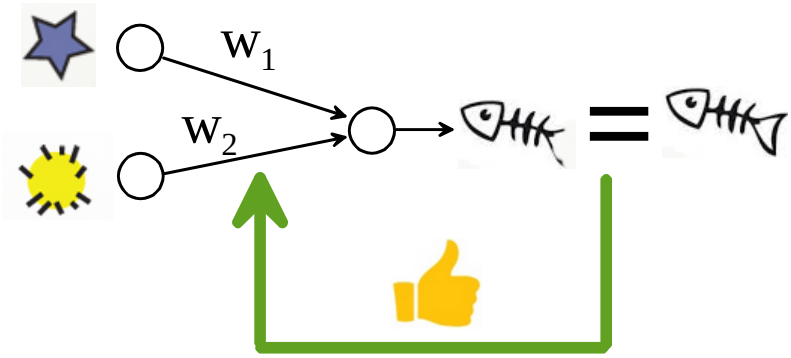
Pour chacune de ces combinaisons on connait la sortie souhaitée donc on peut calculer la différence entre celle-ci et la sortie calculée du réseau. On se sert de cette erreur pour modifier les valeurs des poids w_i du réseau **c'est le mécanisme de rétro propagation**.



Le processus est répété pour toutes les combinaisons d'entrées cela s'appelle une itération.

❒ **Epoch** — Dans le contexte de l'entraînement d'un modèle, l'*epoch* est un terme utilisé pour référer à une itération où le modèle voit tout le training set pour mettre à jour ses coefficients.

Un certain nombre d'itérations sont effectuées jusqu'à ce que le critère d'erreur devienne suffisamment faible :



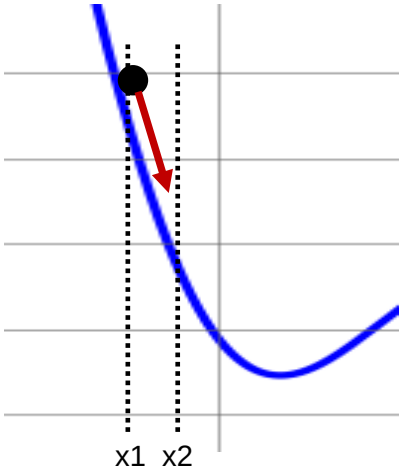
Les poids sont déterminés, le réseau est maintenant opérationnel.

2.2 Apprentissage des poids : descente de gradient

La descente de gradient est une méthode permettant de manière itérative de déterminer le minimum d'une fonction.

Principe de fonctionnement de la descente de gradient[2]

Problématique : trouver le minimum de la fonction f(x) ci-dessous.



Visuellement 'il suffit de se laisser glisser le long de la courbe' pour arriver vers le minimum.

Mathématiquement **c'est la valeur de la pente** au point étudié qui nous indiquera la direction. Donc le calcul de la dérivée de la fonction au point concerné.

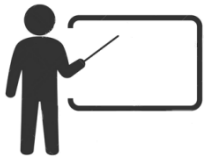
Pour calculer la nouvelle abscisse x2 qui permet de progresser vers le minimum recherché on utilise une relation du type :

[2] Voir sur le site https://www.charlesbordet.com/fr/gradient-descent/#et-les-maths-dans-tout-ca-

$x2 = x1 - \eta \cdot f'(x1)$

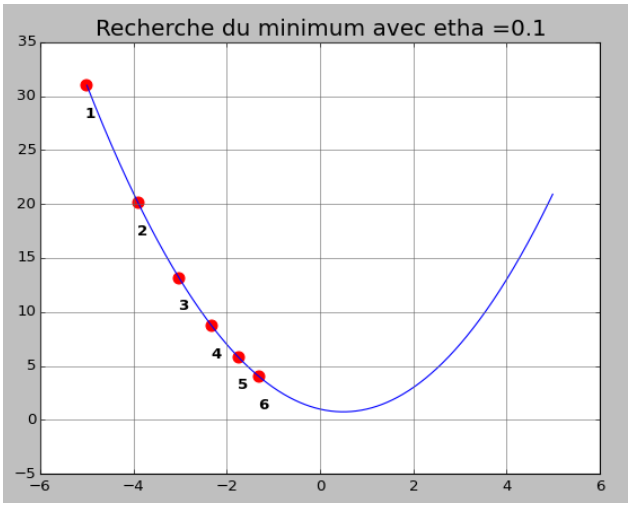
Le processus se répète de proche en proche jusqu'à aboutir au minimum recherché. Ce processus s'appelle descente de gradient. (*Gradient = dérivé*). Le paramètre η permet de régler la rapidité de réaction vous allez étudier son influence en pratique avec les scripts ci-dessous.

Script_Descente_Gradient_1. Compléter Descente de gradient_1.py pour obtenir le calcul des points successifs comme ci-dessous avec les valeurs par défaut.



Résultats attendus

```
Point n° 0 x= -5.00 y= 31.00
Point n° 1 x= -3.90 y= 20.11
Point n° 2 x= -3.02 y= 13.14
Point n° 3 x= -2.32 y=  8.68
Point n° 4 x= -1.75 y=  5.83
Point n° 5 x= -1.30 y=  4.00
```



Utilisez le script précédent avec etha = 0.9, les autres valeurs restant par défaut. On obtient la courbe ci-contre :

Q7. Numérotez les points.

Q8. Que se passe-t-il ?

Q9. Pourquoi ?

Utilisez le script précédent avec etha = 0.01, les autres valeurs restant par défaut.

Q10. Que se passe-t-il ?

Q11. Pourquoi ?

Fonction f(x) = 2·x²·cos(x) - 5·x

Pour appliquer la descente de gradient il nous faut l'expressions de la dérivée. Nous allons calculer la dérivée en utilisant le calcul formel. Le résultat sera l'expression analytique de la fonction.

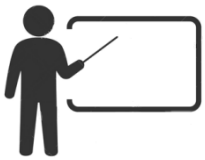
calcul_formel_1.py

```
Calcul formel en Python module sympy
f= 2*x**2*cos(x) - 5*x
df/dx= -2*x**2*sin(x) + 4*x*cos(x) - 5
```

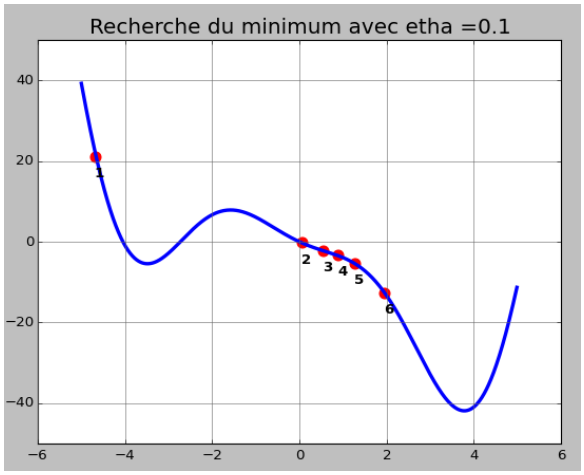
Q12. En modifiant le script ci-dessus déterminer les expressions des fonctions dérivées. Remplir le tableau ci-dessous :

Fonction f(x)	Fonction f'(x)
`f= x**3`	
`f= 1/x`	
`f= sqrt(x)`	
`f= x*cos(x)`	

Script_Descente_Gradient_2. Compléter Descente_de_gradient_2.py pour obtenir le calcul des points successifs comme ci-dessous avec les valeurs par défaut.



```
Point n° 0 x= -4.66 y= 21.03
Point n° 1 x=  0.08 y= -0.39
Point n° 2 x=  0.55 y= -2.23
Point n° 3 x=  0.89 y= -3.46
Point n° 4 x=  1.29 y= -5.54
Point n° 5 x=  1.97 y=-12.88
```



Q13. Que remarque-t-on sur cette courbe ?

Q14. Est-on sûr d'obtenir le minimum le plus grand : minimum minimorum ?

2.3 Application à notre perceptron

Ici il s'agira de calculer le minimum d'une fonction d'erreur calculée en comparant la sortie prédite par notre réseau de neurones avec la sortie prévue.

Deux paramètres pour caractériser notre apprentissage :

- Le taux d'apprentissage ou *learning rate* qui module l'amplitude de la correction appliquée à chacun des poids en fonction de l'erreur.
- Epochs : le nombre d'itération réalisée dans le processus d'apprentissage avec le jeu de données en entier.

Les relations sont les suivantes :

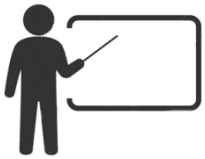
Apprentissage de chaque poids :

$$w(t+1) = w(t) + \text{learning rate} \times (expected(t) - predicted(t)) \times x(t)$$

Apprentissage du biais :

$$bias(t+1) = bias(t) + \text{learning rate} \times (expected(t) - predicted(t))$$

Script_Perceptron_4. Compléter le script pour obtenir les résultats ci-dessous. Script_Perceptron_4.py



Voilà l'algorithme de la fonction *train_weights* en métalangage :

```
Fonction train_weights(data, taux_apprentissage, nombre_epochs)
debut fonction
        initialiser les poids à 0
        pour tous les epochs faire
        debut pour
            somme_erreur = 0.0
            pour tous les vecteurs de tests row dans data faire
            debut pour
                prediction = predict(row, weights)
                erreur = valeur_prévue – prédiction
                somme_erreur = somme_erreur + erreur²
                poids[0] = poids[0] + taux_apprentissage ▪ erreur
                pour i de 1 à len(row)-1 faire
                debut pour
                    poids[i] = poids[i] + taux_apprentissage ▪ erreur ▪ row[i]
                fin pour
            fin pour
        fin pour
return poids
```

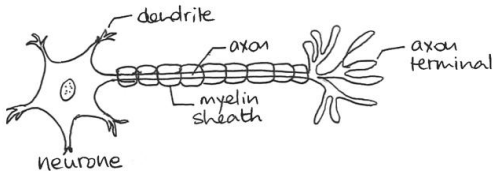
```
>epoch=0, lrate=0.100, error=2.000
>epoch=1, lrate=0.100, error=1.000
>epoch=2, lrate=0.100, error=0.000
>epoch=3, lrate=0.100, error=0.000
>epoch=4, lrate=0.100, error=0.000
[-0.1, 0.20653640140000007, -0.23418117710000003, 0.0]
```

Et la fonction Python à compléter :

```
# Estimatation des poids du Perceptron
def train_weights(train, l_rate, n_epoch):
    '''
    Apprentissage des poids du perceptron
    Entrées
        train    : l'ensemble des vecteurs de tests chacun étant de la forme [ 1, x1, x2, y]
        l_rate   : le taux d'apprentissage
        n_epochs : nombre d'itérations sur le jeu de tests complet
    Sortie
        Les poids du perceptron après apprentissage
    '''
    # Initialisation des poids à 0
    weights = [''' à compéter ''']
    # Apprentissage
    for epoch in range(n_epoch):
        sum_error = 0.0
        for row in train:
            prediction =
            error =
            sum_error += error**2
            weights[0] =
            for i in range(1,len(row)-1):
                weights[i] =
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    return weights
```

Le réseau est entrainé on peut maintenant l'utiliser pour faire des prédictions.

3 Perceptron feuille réponse

Nom :	**Note : / 20**
	**Classe :**

Q1. Expliquer en quoi le fonctionnement de ce 'neurone' artificiel est relativement similaire au fonctionnement du neurone biologique.

Q2. x1 et x2 sont deux variables logiques. Quelle est la fonction réalisée par le neurone.

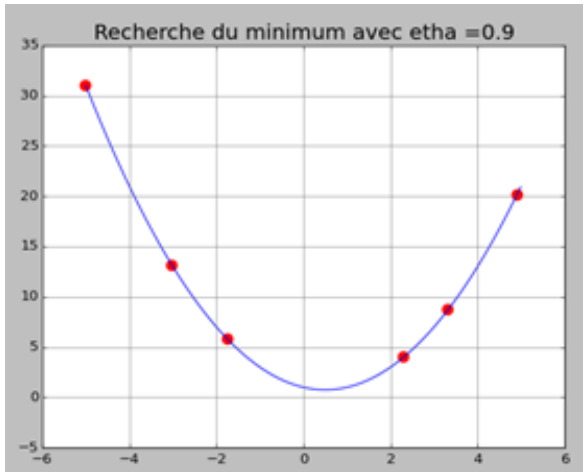
Q3. Calculer la valeur activation et prédiction pour les données suivantes :
weights = [-0.1, 0.206, -0.234]
row = [1, 1.465, 2.362, 0]

Q4. La valeur prédite est-elle bien égale à celle attendue ?

Q5. Calculer la valeur activation et prédiction pour les données suivantes :
weights = [-0.1, 0.206, -0.234]
row = [1, 7.673, 3.508, 1]

Q6. La valeur prédite est-elle bien égale à celle attendue ?

Q7. Numérotez les points.

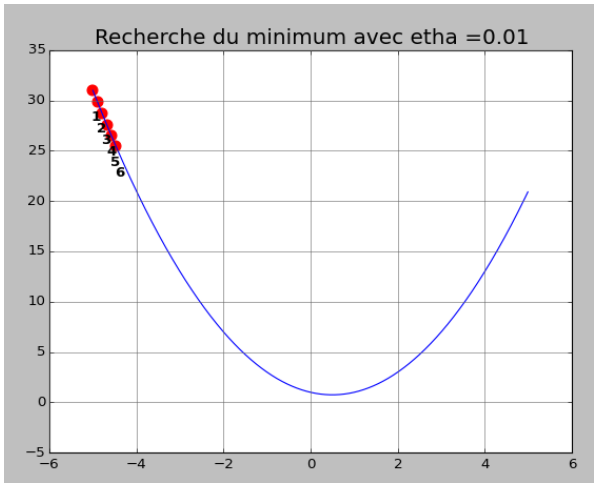


Q8. Que se passe-t-il ?

Q9. Pourquoi ?

Utilisez le script précédent avec etha = 0.01, les autres valeurs restant par défaut.

Q10. Que se passe-t-il ?



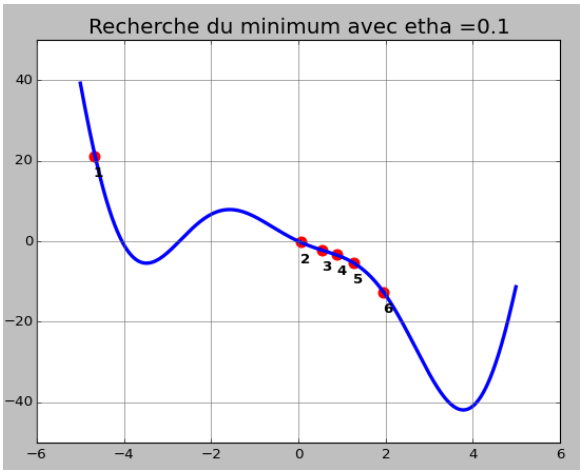
Q11. Pourquoi ?

Q12. En modifiant le script calcul_formel_1.py déterminé les expressions des fonctions dérivées. Remplir le tableau ci-dessous :

Fonction f(x)	Fonction f'(x)
`f= x**3`	
`f= 1/x`	
`f= sqrt(x)`	
`f= x*cos(x)`	

Fonction f(x) = $2 \cdot x^2 \cdot \cos(x) - 5 \cdot x$

Q13. Que remarque-t-on sur cette courbe ?



Q14. Est-on sûr d'obtenir le minimum le plus grand : minimum minimorum ?