C3 TP5 : Introduction du modèle du réseau neuronal

Mathieu Oriol

Introduction au perceptron

Perceptron : implantation d'une porte logique

Voici le code du perceptron :

```
def sum(w, x):
    return w.T @ np.concatenate(([1], x))

def step(s):
    if s ≥ 0:
        return 1
    else:
        return 0

def perceptron(w, x):
    return step(sum(w, x))
```

Opérateur	w_0	w_1	w_2
ET	−1.5	1	1
OU	−0.5	1	1

Tableau 1. – Table des poids à utiliser pour les opérateur ET et OU

Il n'est pas possible de représenter la fonction XOR avec un seul perceptron car la fonction n'est pas séparable linéairement (Il faut au moins 2 droites).

Exercice d'application

Il faut d'abord s'assurer que les valeurs des capteurs sont de type tout ou rien, car on ne peut pas faire la différence entre une valeur de 3000 venant de 2 capteurs à 1500 d'un seul à 3000 d'après la courbe de réponse des capteurs Fig. 1.

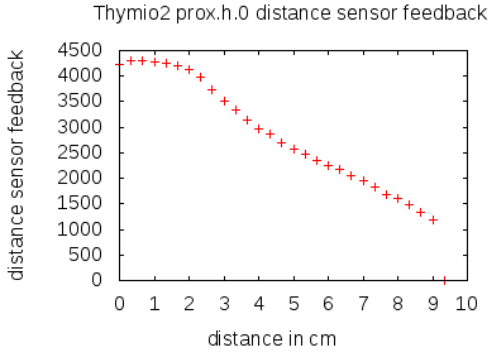

Fig. 1. – Courbe de réponse des capteurs, source https://cyberbotics.com/doc/guide/thymio2?version=R2021b

```
def get_prox_back():
    return np.array(
        [
```

```
            1 if distanceSensors[5].getValue() > 0 else 0,
            1 if distanceSensors[6].getValue() > 0 else 0,
        ]
    )


robot_speed = 2 * perceptron(
    w_and, get_prox_back()
)
```

Perceptron analogique

On utilise la fonction tangente hyperbolique comme fonction d'activation et $w_0 = 0, w_1 = 1$

```
speed = 9 * perceptron(w_analog, np.array([distanceVal[2]]), func_act=math.tanh)
```

Si on ajoute $w_3 = 0.1$, on voit l'effet des poids sur la vitesse comme dans la vidéo `Perceptron_analogique.webm`.

Application des réseaux de neurones artificiels

Véhicule de Braitenberg

1. Conception d'une stratégie d'évitement d'obstacles par réseau de neurones

On prend :

- $w_{\text{back}} > w_{\text{fwd}}$ pour pouvoir reculer si le robot et face à un mur.
- $w_{\text{fwd}} < w_{\text{pos}}$ et $w_{\text{fwd}} < w_{\text{neg}}$ pour pouvoir commencer à manœuvrer même si le robot est loin des murs et pour bien manœuvrer s'il est très proche.
- $w_{\text{back}} > w_{\text{pos}}$ et $w_{\text{back}} > w_{\text{neg}}$
- w_{neg} et w_{pos} assez proches pour ne pas déséquilibrer la rotation.
- J'ai pris $w_{\text{back}} = 2$, $w_{\text{fwd}} = 1$, $w_{\text{pos}} = 1.2$ et $w_{\text{neg}} = 1.3$
- Si le capteur de gauche et le capteur du centre détectent un obstacle simultanément (disons que les capteurs renvoient 0.5) alors la sortie du neurone de la roue droite est $w_{\text{fwd}} - 0.5w_{\text{neg}} - 0.5w_{\text{back}} = -0.65$, et de 0.6 pour la roue gauche, donc le robot tourne à droite (et recule très légèrement).
- La vidéo `evitement.mp4` montre le robot en action avec ces paramètres.

2. Conception d'une stratégie de suivi d'objet

On prend :

- $w_{\text{back}} > w_{\text{fwd}}$, mais avec un écart moins important entre les deux pour pouvoir reculer si le robot touche un mur et s'arrêter très proche.
- w_{pos} et w_{neg} sont pris négatif pour se diriger vers un obstacle.
- J'ai pris $w_{\text{back}} = 1.5$, $w_{\text{fwd}} = 1$, $w_{\text{pos}} = -0.5$ et $w_{\text{neg}} = -1$
- Si le capteur de gauche et le capteur du centre détectent un obstacle simultanément (disons que les capteurs renvoient 0.5) alors la sortie du neurone de la roue droite est 0.98, et de 0 pour la roue gauche, donc le robot tourne à gauche autour de sa roue gauche, donc ne fait que s'orienter vers l'objet.