C3 TP6 : Topologies des réseaux de neurones artificiels

Mathieu Oriol

Les topologies multicouches

Exercice

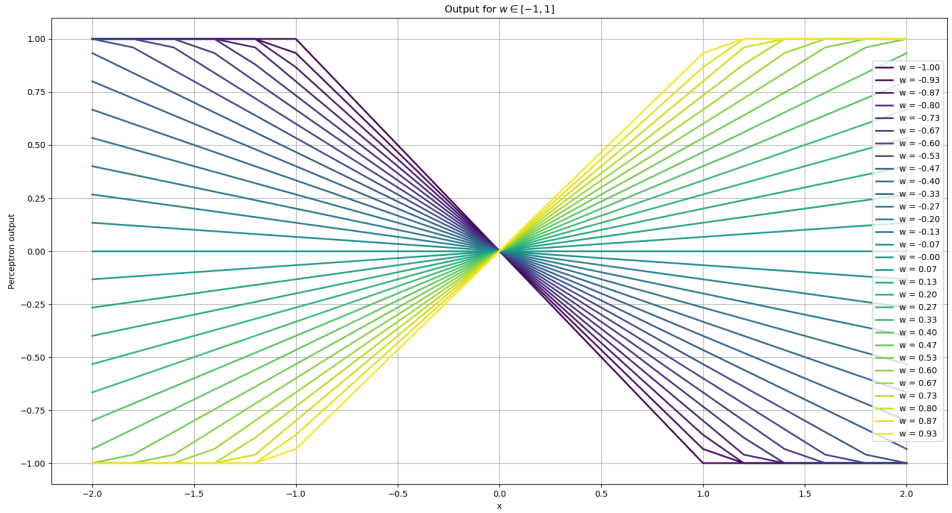

Fig. 1. – Graphe montrant différentes sorties du neurone pour $w \in [-1, 1]$

```python
def mlp(weights, x):
    """
    weigths is a 3D-array contaning layers,
    which are matrices containing perceptron weight
    """
    out = np.asarray([x])
    for layer in weights:
        out = np.clip(layer @ out, -1, 1)
    return out[0]
```

Liste 1. – Code de la fonction implémentation les réseux multicouches

Les sorties obtenus sont des graphes de fonctions toujours monotones. Un seul neurone ne peut donc représenter que des fonctions monotones.

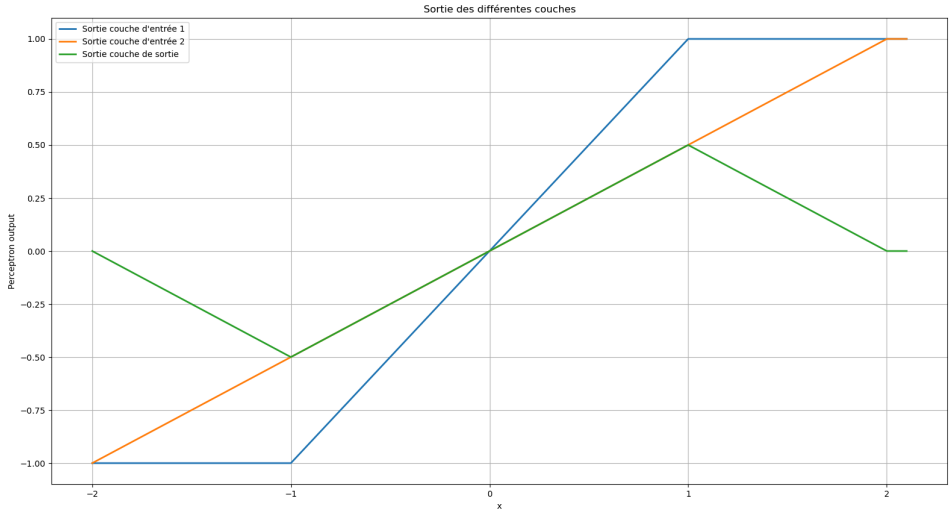

Fig. 2. – Graphe montrant la sortie du réseau ainsi que les sorties des couches d'entrée

Un réseau de neurone peut donc modéliser des fonctions plus complexes.

Exercice d'application robotique

On prend $w_{22} < 0$ pour faire tourner en arrière la roue qui n'est pas devant l'obstacle et l'éviter et $w_{21} > 0$, mais $w_{21} < |w_{22}|$ pour avancer et pouvoir contourner l'obstacle.

```
weights_q4 = [
    np.array([[1, 0], [0, 1]]),
    np.array([[0.2, -1], [-1, 0.2]]),
]
```

Liste 2. – Poids de la question 4

Le fichier `exo.webm` contient une démonstration vidéo de l'exercice.

La mémoire : les réseaux récurrents

Il a fallu fortement augmenter w_{pos} et w_{neg} afin de corriger le mouvement saccadé du robot. Une fois les poids réajustés, on a un parcours du labyrinthe comme dans la vidéo `parcours_mem.webm`, pour la version non récurrente voir la vidéo `parcours.webm`.

```
def mlp_rec(weights, memory, x):
  """
  weigths is a 3D-array contaning layers,
  which are matrices containing perceptron weight
  """
  out = np.array(x)
  for j, (layer, layer_mem) in enumerate(weights):
      out = clip(
          layer @ np.concatenate(([1], out)) + np.multiply(layer_mem, memory[j])
      )
      memory[j] = out
  return out
```

Liste 3. – Code du réseau récurent, qui inclut donc la de mémoire

Le filtre spatial

Lorsque le robot est face à un mur la sortie est : `[0. 0.35918304 0.5801218 0.28842237 0.]`

Lorsque le robot est face à un mur l'obstacle 1 (Fig. 3) la sortie est : `[0. 0. 1. 0.63760097 0.]`

Lorsque le robot est face à un mur l'obstacle 2 (Fig. 4) la sortie est : `[0. 0. 0. 0.92362892 0.]`

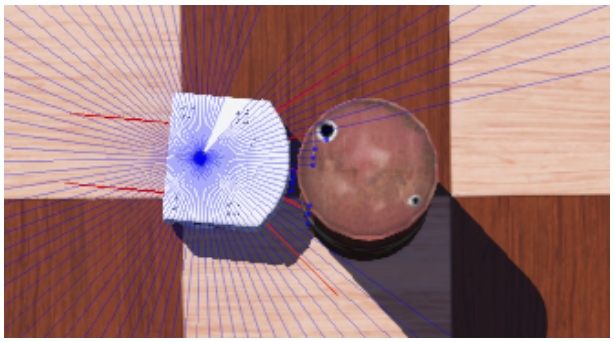

Fig. 3. – Obstacle 1

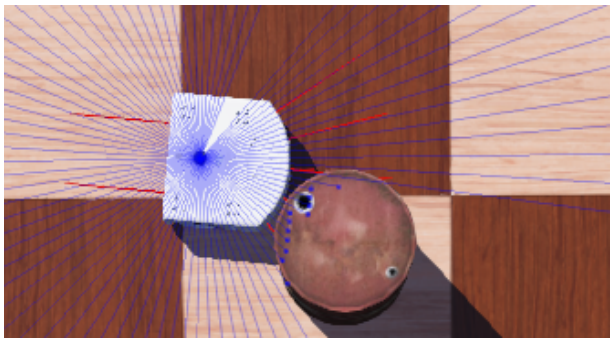

Fig. 4. – Obstacle 2

Le filtre spatial permet de ne conserver des valeurs positives sur moins de sorties que de capteurs en entrée lorsque de capteurs adjacents détectent un objet, les sorties en périphéries de l'objet détecté se retrouvent avec des valeurs négatives (qui sont ramenées à 0 ici).

```
weights_q8 = [
    np.array(
        [
            [0, 4, -4, 0, 0, 0],
            [0, -2, 4, -2, 0, 0],
            [0, 0, -2, 4, -2, 0],
            [0, 0, 0, -2, 4, -2],
            [0, 0, 0, 0, -4, 4],
        ]
    ),
    np.array(
        [
            [0, 0.4, 0.6, 0, 0, 0],
            [0, 0, 0, 1, 0, 0],
            [0, 0, 0, 0, 0.6, 0.4],
        ]
    ),
    np.array(
        [
            [0, -2.5, -0.2, 0],
            [0, 0, -0.2, -2.5],
        ]
    ),
]
```

Liste 4. – Poids utilisé pour le filtre spatial

Si les capteurs de proximité d'indice 1 et 2 (à gauche et devant) renvoient la valeur 0.5, on a (après calcul) la roue gauche qui ira à -1.45 et la droite à 1.3, donc le robot tourne à gauche où se situe l'obstacle. On a bien un robot qui suit les obstacles.

Les poids utilisés sont Liste 4. La vidéo `suivi.webm` montre le suivi d'objet et la vidéo `arret.webm` l'arrêt devant un mur.