



## Rapport de Projet

MASTER 2 ROBOTIQUE

---

# Évitement d'obstacles pour le robot mobile e-puck en utilisant un réseau de neurones artificiels

---

Réalisé par :  
CHAKRAA Hamza  
ARGUI Imane

Encadrant :  
Pr.CHERUBINI Andrea

Année universitaire :  
2020-2021

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Matériels utilisés</b>	<b>5</b>
2.1	Le robot e-puck . . . . .	5
2.2	CoppeliaSim . . . . .	7
<b>3</b>	<b>Les réseaux de neurones</b>	<b>8</b>
3.1	Le système nerveux . . . . .	8
3.2	Le réseau de neurones artificiels . . . . .	8
3.2.1	Principe de fonctionnement . . . . .	9
3.2.2	Fonctions d'activation . . . . .	10
3.2.3	Types d'apprentissage . . . . .	11
<b>4</b>	<b>Algorithmes d'évitement d'obstacles</b>	<b>12</b>
4.1	La règle hebbienne . . . . .	12
4.1.1	Définition . . . . .	12
4.1.2	L'algorithme pour l'évitement d'obstacles . . . . .	13
4.2	Algorithme avec descente du gradient . . . . .	15
4.2.1	Fonction Feedward . . . . .	15
4.2.2	Fonction d'activation . . . . .	15
4.2.3	Descente du gradient . . . . .	16
4.2.4	Fonction Backpropagation . . . . .	16
4.2.5	Apprentissage . . . . .	17
<b>5</b>	<b>Simulation</b>	<b>18</b>
<b>6</b>	<b>Conclusion</b>	<b>19</b>
	<b>Bibliographie</b>	<b>20</b>
	<b>A Script : Algorithme de Hebb</b>	<b>21</b>
	<b>B Script : Algorithme avec descente du gradient</b>	<b>22</b>

## Table des figures

1	Obstacle autour d'un champ de potentiels [8] . . . . .	3
2	Le robot e-puck . . . . .	5
3	Capteurs du robot e-puck [12] . . . . .	6
4	Caractéristiques d'un capteur ultrason [3] . . . . .	6
5	Scène CoppeliaSim . . . . .	7
6	Structure d'un neurone [15] . . . . .	8
7	Aperçu simplifié d'un réseau artificiel de neurones [16] . . . . .	9
8	Un réseau de neurones multicouches . . . . .	9
9	Schéma d'apprentissage supervisé . . . . .	11
10	Réseau de perceptron . . . . .	12
11	Réseau de perceptron pour l'évitement d'obstacles de l'e-puck . . . . .	13
12	Apprentissage d'une machine . . . . .	15
13	Schéma de commande . . . . .	17
14	Trajectoire dans le cas du couloir . . . . .	18
15	Trajectoire dans le cas où l'obstacle est devant . . . . .	18

## Liste des tableaux

1	Caractéristiques du robot e-puck . . . . .	5
2	Les différentes fonctions d'activation . . . . .	10
3	Apprentissage du robot . . . . .	17

## 1 Introduction

Dans le vaste domaine de la robotique, les robots mobiles n'occupent qu'une place relativement modeste qui peut s'expliquer par leur grande complexité comparée à celle des robots manipulateurs. En effet, outre la difficulté de leur conception s'ajoutent d'autres contraintes : leur alimentation, modélisation et commande. Un robot mobile peut accomplir différentes tâches selon l'environnement dans lequel il évolue : Il peut être télécommandé comme il peut évoluer d'une manière autonome dans un environnement connu ou inconnu.

La mobilité par roues est la structure mécanique la plus communément appliquée. Cette technique assure selon l'agencement et les dimensions des roues un déplacement dans toutes les directions avec une accélération et une vitesse importantes.

La tâche principale qui est attribuée à un robot mobile est la navigation de manière autonome. Les deux principaux scénarios sont :

1. Le robot évolue dans un environnement connu d'une manière autonome, dans ce cas, il connaît les obstacles présents et le but est de choisir le chemin le plus court.
2. Le robot évolue dans un environnement inconnu et doit éviter la collision avec les obstacles à l'aide de ses capteurs ou ses caméras.

Dans le cadre de la navigation d'un robot mobile autonome, il existe différentes méthodes d'évitement d'obstacles imprévus [8] :

1. **Champs de potentiels artificiels (APF)** : construire un champ de potentiels sur l'environnement de navigation pour atteindre des points où le champ est minimal (le champ est trop élevé dans les zones où il y a des obstacles). La figure 1 présente une carte des champs potentiels autour d'un obstacle.

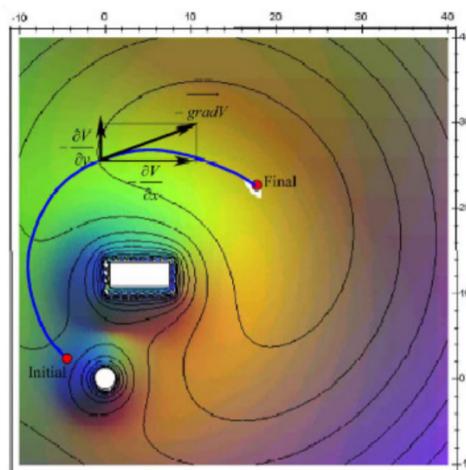


FIGURE 1 – Obstacle autour d'un champ de potentiels [8]

2. **Logique floue** : Lors de la détermination de la proximité d'un obstacle, la logique floue peut donner des concepts comme "assez proche" ou "très loin", plutôt que de se limiter à une définition binaire "d'obstacle ou pas d'obstacle".
3. **Réseau de neurones** : Un système du fonctionnement des neurones biologiques qui s'est approché à des méthodes mathématiques.

Les réseaux de neurones artificiels sont une approche standard pour résoudre différents problèmes, mais nécessitent un processus lourd et coûteux d'estimation des paramètres pendant la phase d'apprentissage qui permet de tenir en compte le maximum de situations possibles pour l'évitement d'obstacles dans un environnement inconnu. [7]

Parmi les différents types des réseaux de neurones, il y a les ANNs (réseaux de neurones artificiels) et les CNNs (réseaux de neurones convolutifs). Le premier est basé sur un modèle avec des entrées et des sorties sous forme de signaux, et le deuxième modèle inspiré par le cortex visuel des animaux, traite les images et les vidéos.

Le CNN est une méthode qui permet de résoudre le problème d'évitement d'obstacles en utilisant la caméra du robot pour analyser s'il voit un obstacle ou pas. L'ANN procède de la même façon en utilisant les données reçues par les capteurs.

Ce projet présente une description de la commande d'un robot mobile à roues en utilisant un réseau de neurones artificiels (ANN). Ce choix repose sur le fait de manipuler les capteurs du robot.

La suite du rapport présentera en premier temps un aperçu du robot e-puck et de l'environnement de simulation utilisé, le troisième chapitre contiendra la description du système nerveux de l'être humain et du réseau ANN, ensuite le quatrième chapitre traitera les différentes étapes de l'application de cette méthode pour l'évitement d'obstacles du robot mobile e-puck en indiquant les algorithmes et les fonctions nécessaires. Et enfin, les résultats de la simulation sur CoppeliaSim seront présentés.

## 2 Matériels utilisés

### 2.1 Le robot e-puck

Les robots mobiles à roues peuvent être présentés sous différents aspects, selon leur taille, forme, caractéristique ... Le robot sur lequel nous allons travailler durant ce projet est le robot e-puck V1. Il est représenté dans la figure 2. C'est un robot qui a été développé pour l'enseignement et la recherche. Il est caractérisé par sa taille, ayant un diamètre de 70 mm, un poids de 200 g et une hauteur de 50 mm. Son design flexible et sa structure lui fournissent un large domaine d'utilisation. [14]



FIGURE 2 – Le robot e-puck

Le tableau 1 décrit les principales caractéristiques du robot e-puck :

Vitesse maximale	13 cm/s
Autonomie de déplacement	2 heures
Processeur	DsPIC30F6014A
RAM	8 KB
Flash	144 KB
Moteurs	2 moteurs pas à pas
Capteurs	8 capteurs infrarouges
Caméra	Couleur VGA (640 x 480)
Leds	Anneau de 8 Leds, 1 Led frontale, 1 Led centrale.
Accéléromètre	3D
Microphones	3
Haut-parleur	1

TABLE 1 – Caractéristiques du robot e-puck

Ce travail se base sur l'utilisation des capteurs du robot. Comme décrit dans le tableau, c'est des capteurs ultrasons. La figure 3 représente l'emplacement des 8 capteurs et les angles entre chacun d'eux (dans la suite du rapport, le capteur 8 correspondra au capteur 0 de la figure 3).

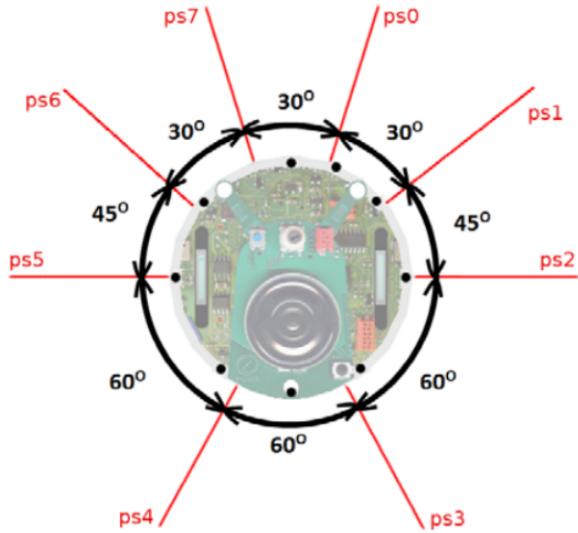


FIGURE 3 – Capteurs du robot e-puck [12]

Les capteurs ultrasons émettent des impulsions à une fréquence donnée. Ces impulsions se propagent à la vitesse de la lumière dans l'environnement. Si un obstacle est présent, l'impulsion est alors réfléchie au capteur, et c'est ce qui lui permet de détecter sa présence. Le capteur peut déterminer la distance le séparant de l'obstacle en calculant le temps de propagation des ultrasons à travers la relation :  $d = \frac{vt}{2}$  [3]

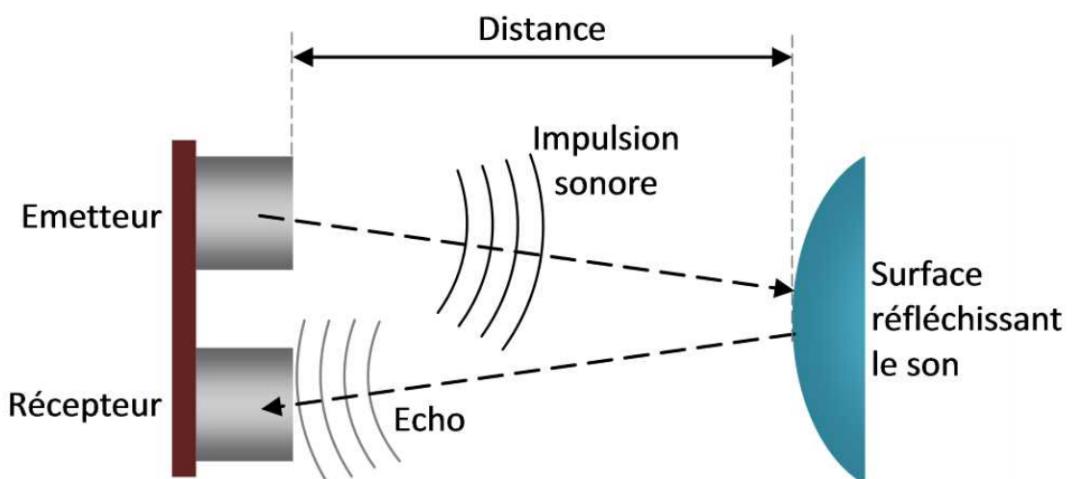


FIGURE 4 – Caractéristiques d'un capteur ultrason [3]

## 2.2 CoppeliaSim

Le simulateur de robot CoppeliaSim avec environnement de développement intégré est basé sur une architecture de contrôle distribué. C'est-à-dire que chaque objet ou modèle peut être contrôlé via un script. [4]. C'est un simulateur de robots 3D permettant aux robots virtuels d'agir dans un monde avec gravité, frottements et collisions. Les algorithmes de contrôle peuvent être écrits en différents langages de programmation (C / C ++, Python, Java, Matlab ou Octave).

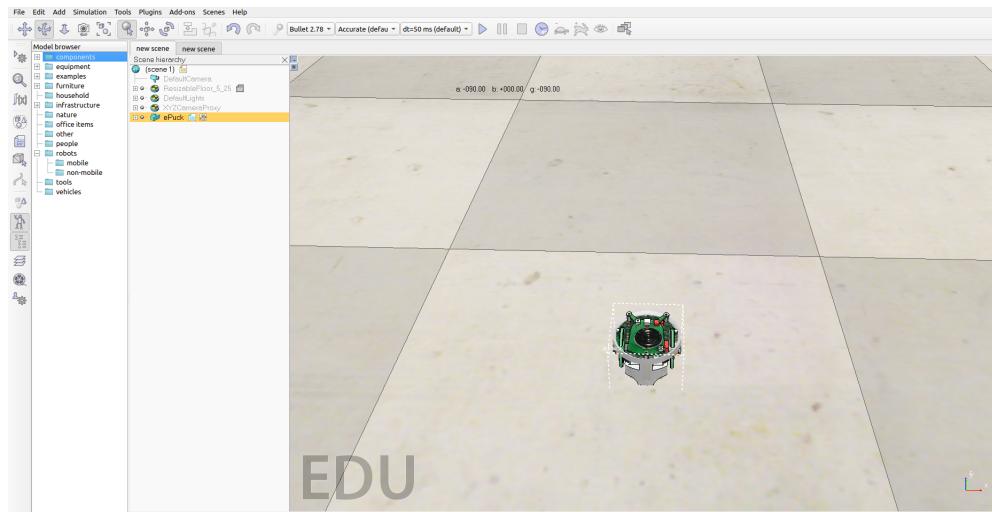


FIGURE 5 – Scène CoppeliaSim

CoppeliaSim a le plus de fonctions et l'API (Interface de programmation ) la plus puissante. De même, dans le sens où il peut être installé et utilisé sur tous les systèmes d'exploitation, le logiciel peut parfaitement interagir. Enfin, l'outil intègre un plug-in permettant de gérer les nœuds ROS, ce qui permet par la suite d'établir l'interconnexion de plusieurs simulateurs. La figure 5 représente le robot e-puck dans l'espace de travail CoppeliaSim.

## 3 Les réseaux de neurones

### 3.1 Le système nerveux

Le système nerveux est un système biologique qui est responsable de la coordination des actions avec l'environnement extérieur et de la communication rapide entre les différentes parties du corps humain.

Les neurones sont les principales cellules du système nerveux. Ces cellules excitables peuvent transmettre des signaux électrochimiques d'un point du corps à un autre. Ce signal est formé par la propagation de la dépolarisation de la membrane plasmique et la libération de molécules chimiques au niveau des points de connexion avec d'autres cellules. La fonction importante des neurones est de traiter ces différentes formes d'informations dans de multiples réseaux afin qu'ils puissent effectuer des tâches complexes et diverses. [13]

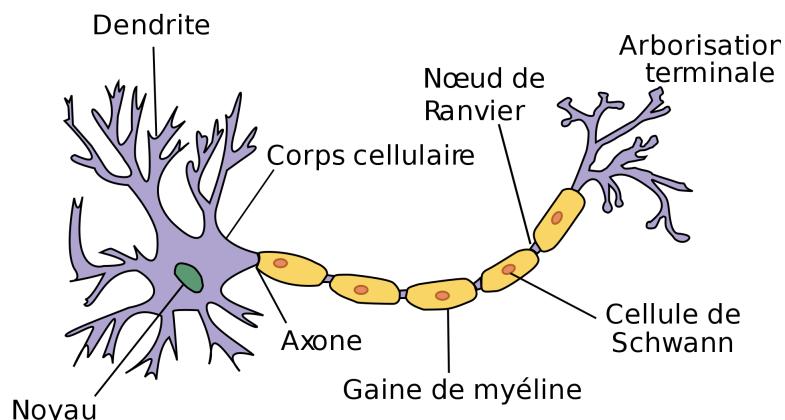


FIGURE 6 – Structure d'un neurone [15]

Le neurone est constitué principalement d'un noyau et d'une substance appelée axone qui rend la communication possible entre deux cellules. En fait, les axones de différents neurones se connectent au dendrite situé à l'extrémité du corps à l'aide des synapses (Figure 6).

Cependant, ce processus peut être simplifié en se basant sur des impulsions envoyées d'un neurone à un autre. Une entrée va être reçue au niveau de la dendrite, et puis des opérations seront effectuées au niveau du noyau afin de produire une impulsion de sortie à l'axone. [13]

### 3.2 Le réseau de neurones artificiels

Le réseau de neurones artificiels est un modèle mathématique qui s'inspire principalement du comportement d'un réseau de neurones biologiques. De nombreux neurones disposés dans une structure interconnectée forment un réseau. Chacun d'entre eux est relié par les informations entrantes et sortantes des autres neurones, en fonction de son emplacement. L'imitation du comportement d'un réseau biologique est ce qui permet de résoudre le problème d'apprentissage des machines. [16]

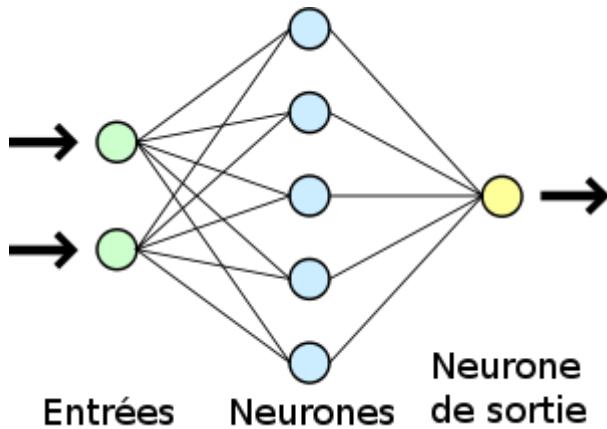


FIGURE 7 – Aperçu simplifié d'un réseau artificiel de neurones [16]

Les réseaux de neurones peuvent comporter au minimum 2 couches de neurones : une couche d'entrée et une couche de sortie. En général, plus le problème à résoudre est compliqué, plus le réseau comporte de couches. La figure 7 est une représentation d'un réseau simple de neurones artificiels.

### 3.2.1 Principe de fonctionnement

Quel que soit le type du réseau, l'information est traitée généralement de la même façon. Le réseau reçoit des informations sous forme de signaux. A chaque signal est attribué un poids qui symbolise son importance. Ces poids définissent les informations qui peuvent entrer dans le système pour que la fonction d'activation associée à un seuil puisse calculer la sortie du neurone. La figure 8 schématise le principe de fonctionnement d'un réseau à plusieurs couches.

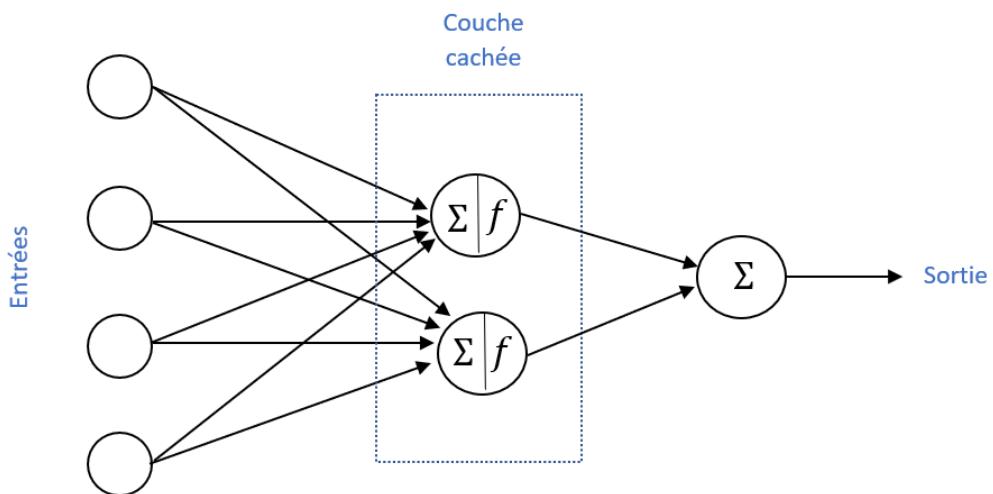


FIGURE 8 – Un réseau de neurones multicouches

### 3.2.2 Fonctions d'activation

Une fonction d'activation sert à modéliser la manière dont les neurones traitent les signaux électriques qu'ils reçoivent, c'est la fonction mathématique qui permet de traiter l'information qui arrive à un neurone artificiel en machine learning. Le choix de cette fonction diffère selon le problème à résoudre [5]. Les fonctions d'activation les plus courantes sont représentées dans le tableau 2 :

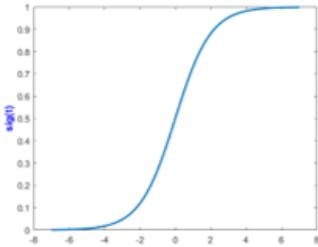
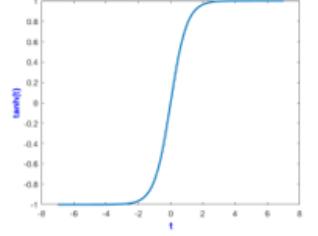
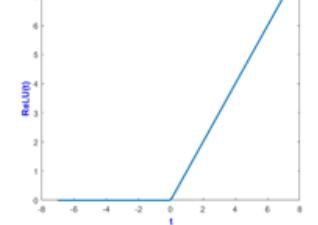
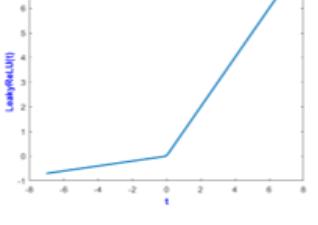
Fonction	Expression	Représentation	Caractéristique
<b>Fonction sigmoïde</b>	$f(t) = \frac{1}{1 + e^{-t}}$		Transforme une valeur réelle d'entrée en une valeur comprise entre 0 et 1.
<b>Tangente hyperbolique</b>	$f(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$		Transforme la valeur d'entrée en une valeur comprise entre -1 et 1.
<b>Fonction ReLU</b>	$f(t) = \begin{cases} 0 & \text{si } t < 0 \\ t & \text{si } t \geq 0 \end{cases}$		Remplace toute entrée négative par 0 et toute entrée positive par elle-même.
<b>Fonction Leaky ReLU</b>	$f(t) = \begin{cases} \alpha t & \text{si } t < 0 \\ t & \text{si } t \geq 0 \end{cases}$		Remplace toute entrée négative par $\alpha t$ afin d'éviter que certains gradients s'annulent et toute entrée positive par elle-même.

TABLE 2 – Les différentes fonctions d'activation

Notons qu'il existe bien d'autres façons de choisir la manière de s'activer (max-pool, softmax...).

### 3.2.3 Types d'apprentissage

Pour que le robot puisse apprendre par lui-même, il existe différents modes d'apprentissage. Voici les trois principaux types [1] :

- **Apprentissage par renforcement** : L'apprentissage par renforcement est inspiré par l'apprentissage humain ou animal. En effet, lors de la réalisation des expériences, le résultat obtenu est observé ; si le résultat est positif et encourageant, ça veut dire que l'expérience est bénéfique et peut être réitérée, or si le résultat est négatif, l'expérience ne doit plus se reproduire. Il en est de même pour l'algorithme, il apprend par interaction avec l'environnement, et selon le résultat de ses actions, il arrive à définir le comportement idéal à suivre. Pour ce faire, un simple retour des résultats est nécessaire pour apprendre comment il doit agir.
- **Apprentissage non supervisé** : Dans ce cas, la machine devra apprendre par elle-même, c'est-à-dire qu'elle ne dispose pas au préalable d'un classement déterminé. Le nombre de labels et les caractéristiques sont inconnus, c'est à la machine de regrouper les données selon leurs similarités. Par exemple, dans une série de photos donnée, la machine peut regrouper les images d'un oiseau en repérant ses caractéristiques : bec, ailes, plumes ...
- **Apprentissage supervisé** : Le système apprend à travers un modèle prédéterminé. Dans un premier lieu, il est demandé de classer des données et de les associer à un label. Ensuite, le robot (ou l'algorithme) doit être en mesure de déterminer à quel label appartient une nouvelle donnée en se basant sur le classement fait. Par exemple, pour une série d'images d'objets classés selon leurs types : Table, chaise, porte, la machine est supposée reconnaître l'image entrante selon ce classement.  
D'une manière générale, cet apprentissage peut être schématisé comme le montre la figure 9.

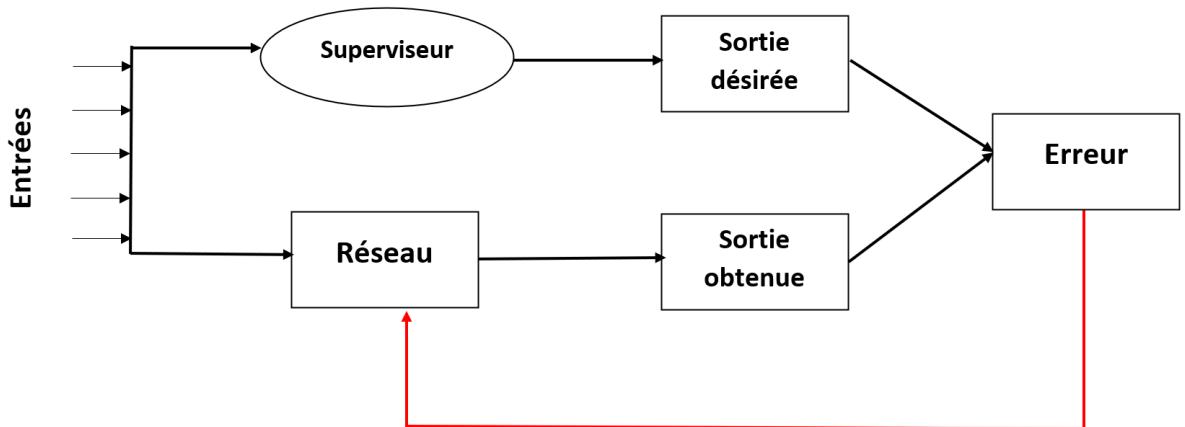


FIGURE 9 – Schéma d'apprentissage supervisé

## 4 Algorithmes d'évitement d'obstacles

Ce chapitre décrit les deux algorithmes implémentés dans ce projet. Le premier est basé sur l'apprentissage en utilisant la règle de Hebb, et le deuxième est basé sur la méthode d'optimisation : la descente du gradient.

Ces deux algorithmes sont basés sur le perceptron (un réseau de neurones sans aucune couche cachée). Il contient une couche d'entrée et une couche de sortie. [9] L'utilisation du perceptron se résume dans de nombreux scénarios. Bien qu'il soit principalement utilisé pour une prise de décision simple, il peut également être combiné dans des programmes informatiques volumineux pour résoudre des problèmes plus complexes.

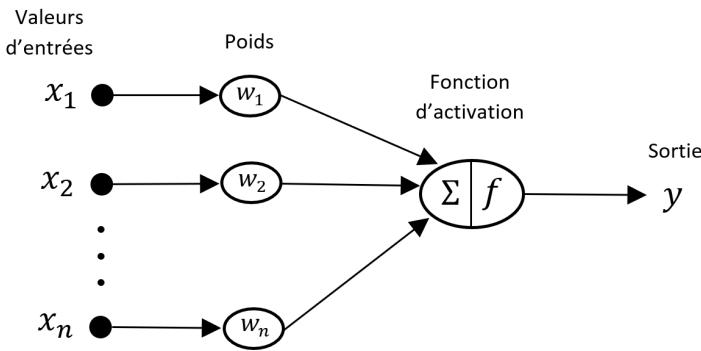


FIGURE 10 – Réseau de perceptron

Les entrées seront multipliées par les poids puis additionnées avant qu'une fonction d'activation soit appliquée (Figure 10)

$$y = f \left( \sum_{i=1}^n x_i w_i \right)$$

### 4.1 La règle hebbienne

#### 4.1.1 Définition

La règle de Hebb est une technique d'apprentissage simple pour un ANN. C'est une forme d'entraînement améliorée qui change le poids des connexions entre les neurones du réseau. En fait, si le comportement du robot est bon, l'algorithme renforcera les poids du réseau.[1]

La variation du poids entre un neurone  $k$  et un neurone  $j$  s'écrit sous la forme :

$$\Delta w_{kj} = \alpha y_k x_j$$

Avec  $w_{kj}$  le poids entre  $k$  et  $j$ ,  $\Delta w_{kj}$  son taux de variation,  $y_k$  la sortie du neurone  $k$ ,  $x_j$  l'entrée du neurone  $j$ , et  $\alpha$  la constante qui définit l'apprentissage (elle sera choisie petite pour que la convergence des poids prenne un peu de temps afin de ne pas dépasser des valeurs intéressantes).

L'application de cette règle sur une machine reflète l'attitude attendue d'un humain. C'est le comportement du robot qui est évalué et non pas son état. Dans ce projet, le robot e-puck va apprendre à éviter les obstacles à partir des données issues des capteurs.

#### 4.1.2 L'algorithme pour l'évitement d'obstacles

L'algorithme implémente un ANN afin de lire les entrées des capteurs et calculer les sorties pour les moteurs.

Le réseau est constitué de huit entrées (huit capteurs) et de deux sorties (vitesses des deux roues).

Les entrées seront représentées par le vecteur  $x = [x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8]^T$  et les sorties par le vecteur  $y = [y_1 \ y_2]^T$ .

Les entrées sont modélisées sous une forme binaire (0 si le capteur ne détecte rien, et 1 si le capteur détecte un obstacle) et les valeurs désirées des vitesses sont données selon les informations reçues par les capteurs. Par exemple, si le capteur de gauche détecte un obstacle, les valeurs  $y_1 = -1$  et  $y_2 = 1$  sont affectées aux roues pour tourner à droite.

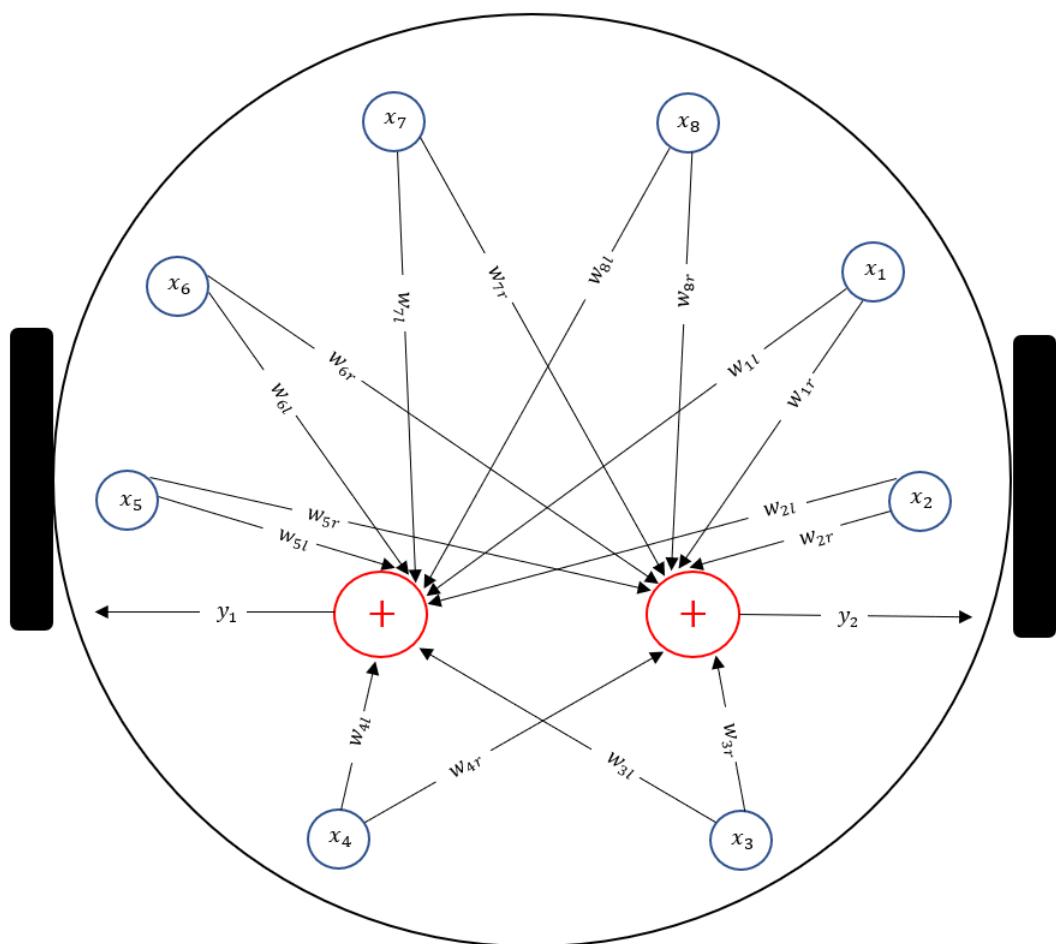


FIGURE 11 – Réseau de perceptron pour l'évitement d'obstacles de l'e-puck

En réalité, cet algorithme est inspiré du véhicule de Braitenberg qui utilise des rétroactions capteur-moteur de base pour produire des comportements cognitifs. Le véhicule est équipé de capteurs primitifs pour mesurer l'excitation en un point donné et les roues (chacune entraînée par son propre moteur) font office d'actionneurs. Dans la configuration la plus simple, un capteur est connecté directement à un jeu d'effets, de sorte qu'un signal détecté induit instantanément le mouvement de la roue. [2] Selon la connexion capteur-moteur, le robot semble essayer d'atteindre certaines situations et en éviter d'autres, en se détournant à mesure que la situation change.

À partir de la figure 11, la sortie du réseau s'écrit sous la forme suivante :

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} w_{1l} & w_{2l} & w_{3l} & w_{4l} & w_{5l} & w_{6l} & w_{7l} & w_{8l} \\ w_{1r} & w_{2r} & w_{3r} & w_{4r} & w_{5r} & w_{6r} & w_{7r} & w_{8r} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = Wx$$

Pour faire l'apprentissage, l'algorithme mettra à jour les poids  $w$  du réseau en utilisant un retour d'informations à partir des données des capteurs et des sorties désirées (apprentissage supervisé) pour une valeur de  $j$  allant de 1 à 8. Le but est de trouver la matrice  $W$  qui permet de minimiser cette erreur.

$$\begin{aligned} w_{jl} &\leftarrow w_{jl} + \alpha y_1 x_j \\ w_{jr} &\leftarrow w_{jr} + \alpha y_2 x_j \end{aligned}$$

$y_1$  et  $y_2$  sont définis comme étant la différence entre la sortie calculée et la sortie voulue (La règle de Hebb prend en compte l'erreur observée en sortie).

$$\begin{aligned} y_1 &\leftarrow y_{1d} - y_1 \\ y_2 &\leftarrow y_{2d} - y_2 \end{aligned}$$

Avec  $y_d = \begin{bmatrix} y_{1d} \\ y_{2d} \end{bmatrix}$  le vecteur de sortie désiré.

Initialement, la matrice  $W$  est nulle ou bien aléatoire, le réseau est prédisposé à déclencher un 1. Alors, la notion du biais "b" sera ajoutée avec  $b = 1$  (biais constant). Dans l'algorithme suivant, l'utilisation du biais sera mise en place avec une variation après chaque itération.

## 4.2 Algorithme avec descente du gradient

Cette section présente un algorithme stochastique qui est beaucoup sollicité dans le "Machine learning". La figure 12 représente la démarche globale de l'apprentissage d'une machine.

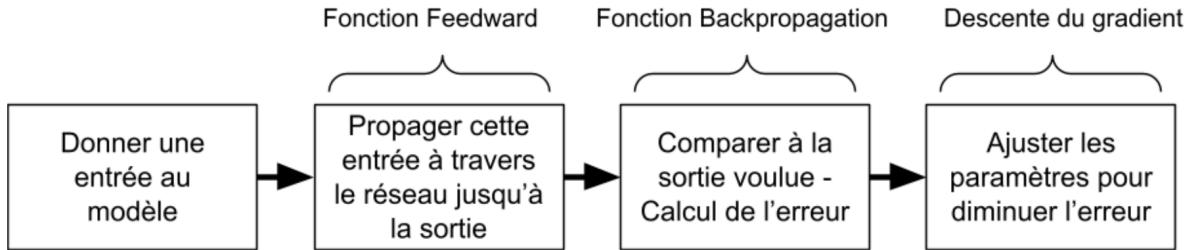


FIGURE 12 – Apprentissage d'une machine

### 4.2.1 Fonction Feedward

$x$  représente l'entrée du système, qui peut être aussi la sortie de la couche précédente pour un réseau à plusieurs couches. Elle introduit les données nécessaires qui vont se propager dans le réseau afin de produire la sortie finale  $y$ . La notion du biais "b" s'ajoute au calcul de la sortie du réseau établi précédemment à partir de la figure 11.

$$\begin{cases} y_1 = b + \sum_{i=1}^8 x_i w_{il} \\ y_2 = b + \sum_{i=1}^8 x_i w_{ir} \end{cases}$$

L'usage du biais est nécessaire afin de réaliser un réseau de neurones robuste. En effet, il contrôle la façon dont le neurone est prédisposé à déclencher un 1 ou un 0, quel que soit le poids.

Notons que les poids sont donnés aléatoirement, ils seront optimisés durant la rétropropagation.

### 4.2.2 Fonction d'activation

La fonction d'activation utilisée dans ce réseau est la tangente hyperbolique qui va permettre de saturer les valeurs des sorties pour une valeur comprise entre  $-1$  et  $1$ .

$$\begin{cases} f(y) = \tanh y \\ y \leftarrow f(y) \end{cases}$$

### 4.2.3 Descente du gradient

La descente du gradient est un algorithme qui procède de manière itérative pour trouver les valeurs optimales des paramètres du réseau de neurones. C'est une méthode d'optimisation non linéaire intuitive destinée à trouver le minimum local d'une fonction réelle différentiable.

La formule générale pour trouver les paramètres optimaux est la suivante [9] :

$$W \leftarrow W - l_r \frac{\partial y}{\partial W}$$

Avec  $W$  la matrice des poids,  $l_r$  le taux d'apprentissage, et  $y$  la sortie après avoir appliqué la fonction d'activation.

Pour adapter l'équation précédente à notre problème,  $y$  est remplacée par l'erreur  $E$ .

$$W \leftarrow W - l_r \frac{\partial E}{\partial W}$$

Il existe de nombreuses façons d'exprimer l'erreur du réseau qui mesure la performance pour une entrée donnée. Dans ce contexte, la méthode utilisée pour identifier cette erreur est l'une des plus fréquentes : MSE - Mean Square Error (L'erreur moyenne quadratique). [11]

$$E = \sum \frac{1}{2} (y_d - y)^2$$

$y_d$  est la sortie cible introduite à travers l'apprentissage supervisé.

### 4.2.4 Fonction Backpropagation

Il est possible de calculer  $\frac{\partial E}{\partial W}$  en utilisant la règle de dérivation des fonctions composées afin d'ajuster les paramètres de la couche par la suite.

Le but est de voir comment une valeur donnée du poids affecte l'erreur. [9]

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial f} \frac{\partial f}{\partial y} \frac{\partial y}{\partial W}$$

Avec :

$$\begin{cases} \frac{\partial E}{\partial f} = y - y_d \\ \frac{\partial f}{\partial y} = 1 - \tanh y \\ \frac{\partial y}{\partial W} = x \end{cases}$$

À la fin de l'algorithme, le biais sera optimisé dans une boucle allant de  $j = 0$  à  $\frac{\partial E}{\partial W}$ .

$$b \leftarrow b - l_r j$$

#### 4.2.5 Apprentissage

La partie d'entraînement du réseau est la plus compliquée puisque le robot comporte 8 capteurs, facteur qui mène à 256 cas possibles durant la navigation de ce dernier. Dans ce projet, une vingtaine de cas ont été considérés (les cas les plus fréquents).

Le tableau 3 démontre quelques cas envisagés durant le projet : L'idée est de considérer le plus de scénarios possibles : quand un capteur ou un ensemble de capteurs détectent un obstacle, le robot devrait réagir d'une façon déterminée.

Détection d'obstacles	Réaction du robot
Aucun	Robot va tout droit
Capteur 3	Robot va tout droit
Capteur 5, 6 et 7	Robot tourne à droite
Capteur 1 et 2	Robot tourne à gauche
Capteur 6, 7 et 0	Robot tourne à droite
Capteur 0 et 1	Robot tourne à gauche

TABLE 3 – Apprentissage du robot

La figure 13 regroupe toutes les étapes précédentes dans un schéma de contrôle qui explique bien cet algorithme.

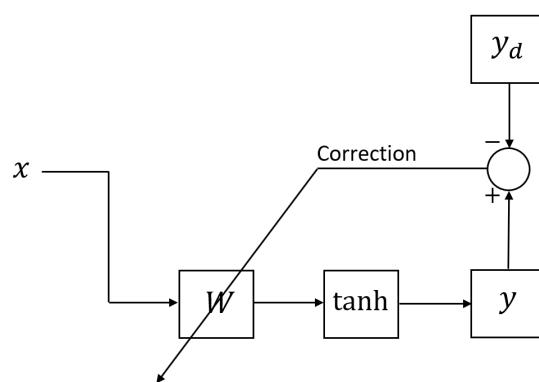


FIGURE 13 – Schéma de commande

## 5 Simulation

Lors de l'implémentation des algorithmes précédents sur CoppeliaSim, le robot a pu éviter les obstacles imprévus pour les différentes situations renseignées dans la phase de l'apprentissage.

Les figures 14 et 15 montrent quelques situations durant la simulation.

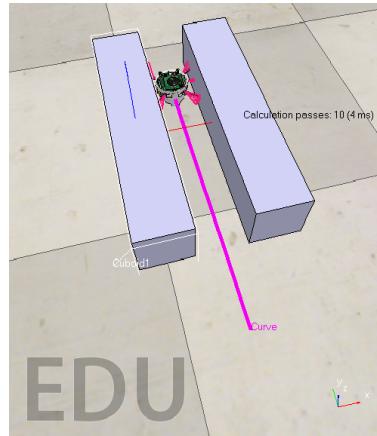


FIGURE 14 – Trajectoire dans le cas du couloir

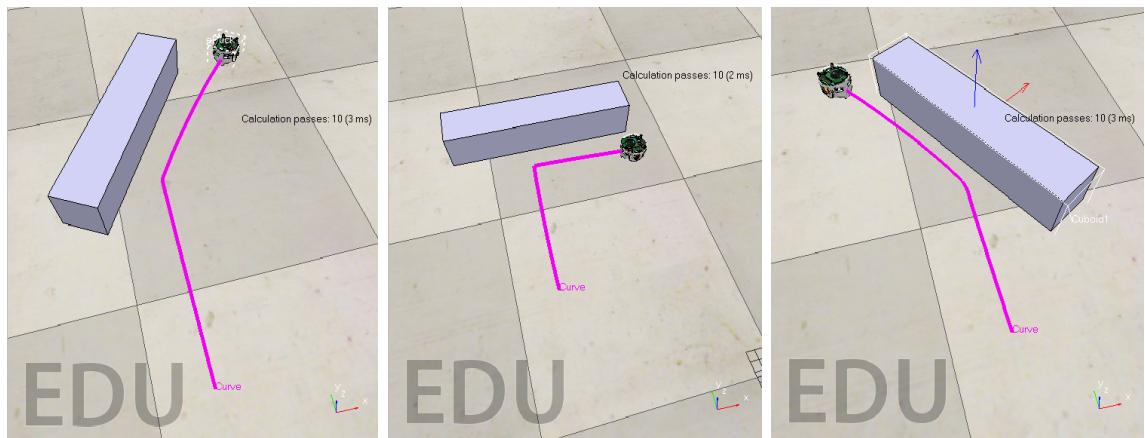


FIGURE 15 – Trajectoire dans le cas où l'obstacle est devant

Les résultats de la simulation ont permis de voir l'optimalité de l'algorithme implémenté. Dans la figure 14, le robot continue son chemin puisque les obstacles sont sur le côté et ne posent aucun problème. Par contre, et comme le présente la figure 15, le robot évite les obstacles d'une façon à suivre une trajectoire parallèle par rapport à ces derniers.

## 6 Conclusion

Depuis l'apparition de l'intelligence artificielle, le monde de l'informatique a été révolutionné. Aujourd'hui toutes les grandes entreprises, notamment les géants du web se basent sur les algorithmes de l'apprentissage artificiel afin de traiter de lourds calculs ou de résoudre des problématiques complexes. Par exemple, Google se base principalement sur la reconnaissance vocale afin de répondre à des requêtes, Facebook utilise la recherche de similarité afin de reconnaître les vidéos ou les photos qui sont sous le coup d'un copyright, et bien plus encore d'exemples.

L'intelligence artificielle s'inspire du comportement de l'humain et de son cerveau, et les réseaux de neurones artificiels sont en plein essor. Le but principal est de permettre à la machine d'apprendre d'une manière indépendante. Il existe différents types d'apprentissages : apprentissage supervisé, non supervisé et par renforcement. Le choix varie selon le besoin et la problématique.

Durant ce projet, les algorithmes utilisés se basent principalement sur l'apprentissage supervisé. C'est d'ailleurs le plus populaire en Machine learning et en Deep learning. Comme son nom l'indique, un superviseur guide la machine durant son apprentissage en lui fournissant des exemples de tâches qu'elle doit réaliser. Deux algorithmes ont été étudiés, puis appliqués afin de résoudre la problématique posée : "L'évitement d'obstacles d'un robot mobile dans un environnement incertain". Le premier algorithme se base sur la règle d'Hebb, qui donne un moyen pratique d'associer les données d'entrée aux données de sortie en prenant en compte l'erreur observée en sortie. Le deuxième algorithme se base sur la descente du gradient et permet de retrouver la valeur minimale des paramètres du réseau. Les résultats obtenus étaient similaires : le robot évitait toute collision avec les obstacles présents dans différents scénarios. Cependant, le réseau utilisé est un réseau assez simple basé sur le principe du perceptron. Il comporte deux couches et tous les neurones présents sont connectés à la couche de sortie. Dans des réseaux plus complexes, il est possible alors d'envisager l'appel à une ou plusieurs couches cachées, on parle alors de perceptrons multicouches.

Les réseaux de neurones convolutifs peuvent être décrits comme étant un empilement de perceptrons multicouches [6], qui sont appelés principalement pour la reconnaissance d'images, de vidéos ou de traitement naturel de langage. Ce type de réseau peut aussi être utilisé pour l'évitement d'obstacles où l'entrée du réseau serait l'image obtenue de la caméra du robot, et les pixels seront alors convertis en commandes. Dans ce cas, il serait alors nécessaire de faire appel à des librairies qui regroupent un grand nombre de modèles et d'algorithmes indispensables au Machine Learning. Les deux librairies les plus compétitives sont "Pytorch" (développée par Facebook) et "Tensorflow" (développée par Google), le débat de quelle librairie choisir n'a pas de réponse évidente [10], c'est deux "frameworks" qui sont en constante évolution et innovation.

## Bibliographie

- [1] Mordechai BEN-ARI et Francesco MONDADA. *Elements of Robotics*. Gewerbestrasse 11, 6330 Cham, Switzerland : Springer International Publishing AG, 2017.
- [2] Valentino BRAITENBERG. *Véhicules : expériences en psychologie synthétique*. Lausanne, Switzerland : PPUR presses polytechniques, 1991.
- [3] *Capteur de distance à Ultrasons*. URL : [arduino.blaisepascal.fr/capteur-de-distance-a-ultrasons](http://arduino.blaisepascal.fr/capteur-de-distance-a-ultrasons).
- [4] COPPELIAROBOTICS. URL : [coppeliarobotics.com](http://coppeliarobotics.com).
- [5] *Fonction d'activation*. URL : [deeplearning.fr/cours-theoriques-deep-learning/fonction-dactivation](https://deeplearning.fr/cours-theoriques-deep-learning/fonction-dactivation).
- [6] Canglong LIU et al. “CNN-Based Vision Model for Obstacle Avoidance of Mobile Robot”. In : *MATEC Web of Conferences* 139 (jan. 2017), p. 00007. DOI : [10.1051/matecconf/201713900007](https://doi.org/10.1051/matecconf/201713900007).
- [7] Mohamed MORCHID, Richard DUFOUR et Georges LINARÉS. “Initialisation de Réseaux de Neurones à l'aide d'un Espace Thématisqué”. In : (juin 2015), p. 28-33.
- [8] Nicolas MORETTE. “Contribution à la Navigation de robots mobiles : approche par modèle direct et commande prédictive”. Thèse. Université d'Orléans, 2009.
- [9] Shukla PRATIK et Roberto IRIONDO. “Neural Networks from Scratch with Python Code and Math in Detail — I”. In : *Towards AI* (juin 2020). URL : [towardsai.net/neural-networks-with-python](https://towardsai.net/neural-networks-with-python).
- [10] *PyTorch vs TensorFlow*. URL : <https://medium.com/neoxia/pytorch-vs-tensorflow-9768d24626c6>.
- [11] *Réseaux de neurones en partant de zéro en Python*. URL : [medium.com/france-school-of-ai/math%C3%A9matiques-des-r%C3%A9seaux-de-neurones-code-python-613d8e83541](https://medium.com/france-school-of-ai/math%C3%A9matiques-des-r%C3%A9seaux-de-neurones-code-python-613d8e83541).
- [12] Pablo TARQUINO et Kevin NICKELS. “Programming an E-Puck Robot to Create Maps of Virtual and Physical Environments”. In : (jan. 2014).
- [13] WIKIPEDIA. *Biological neuron model*. URL : [en.wikipedia.org/wiki/Biological\\_neuron\\_model](https://en.wikipedia.org/wiki/Biological_neuron_model).
- [14] WIKIPEDIA. *E-Puck*. URL : [fr.wikipedia.org/wiki/E-Puck](https://fr.wikipedia.org/wiki/E-Puck).
- [15] WIKIPEDIA. *Neuromorphologie*. URL : <https://fr.wikipedia.org/wiki/Neuromorphologie>.
- [16] WIKIPEDIA. *Réseau de neurones artificiels*. URL : [fr.wikipedia.org/wiki/R%C3%A9seau\\_de\\_neurones\\_artificiels](https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels).

## A Script : Algorithme de Hebb

```
# Valeur du biais
bias = 1

# Taux d'apprentissage
alpha = 0.03

# Le vecteur de sortie
outputs = np.dot(weights,inputs) + bias

# Fonction d'activation
outputs = np.tanh(outputs)

# Erreur
E = target_outputs - outputs

# Algorithme de Hebb
for j in range(7) :
    weights[0][j] = weights[0][j] + alpha*E[0]*inputs[j]
    weights[1][j] = weights[1][j] + alpha*E[1]*inputs[j]
```

## B Script : Algorithme avec descente du gradient

```
for i in range(10000):

    # Valeur du biais
    bias = 3

    # Taux d'apprentissage
    lr = 0.02

    # Fonction Feedward
    in_f = np.dot(weights,inputs) + bias

    # Fonction d'activation
    out_f = np.tanh(in_f)

    # Dérivée de l'erreur
    error = out_f - outputs

    # Dérivée de la fonction d'activation
    d = 1 - (np.tanh(out_f))**2

    # Dérivée de la sortie
    i = inputs.T

    # Calcul de la dérivée finale
    deriv = np.dot(d,i)
    deriv_final = error * deriv

    # Mise à jour des poids
    weights = weights - lr * deriv_final

    # Mise à jour du bias
    for j in deriv:
        bias -= lr * deriv
```