Royaume du maroc
ⵜⴰⴳⵍⴷⵉⵜ ⵏ ⵍⵎⵖⵔⵉⴱ

Ministère de l'Education Nationale
du préscolaire & des sports
ⵜⴰⵎⴰⵡⴰⵜ ⵏ ⵜⵓⵙⵎⵔⵙ ⴰⵏⴰⵎⵓⵔ
ⴰⵏ ⵜⵓⵍⴰ ⴰⵎⵣⵡⴰⵓ ⴰ ⵜⵜⵍⵜ

esith
Constructeur de compétences

Thème: Jeux et sport

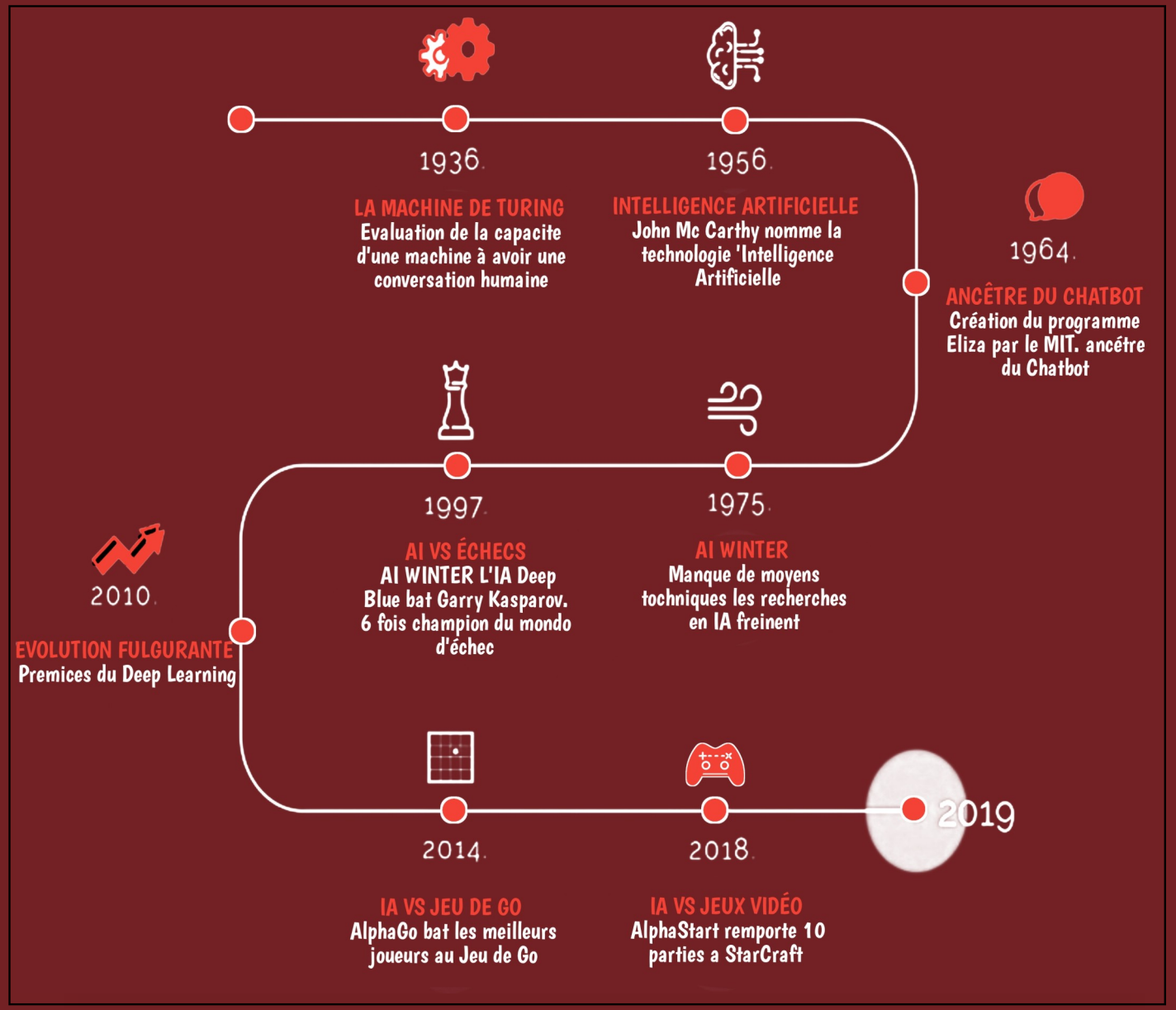
Année scolaire: 2023-2024

Optimisation des Stratégies de Jeu par Neuro-évolution: Une Application au Jeu du Serpent

Réalisé par: Youssef El Antraoui

L'INTELLIGENCE ARTIFICIELLE

L'intelligence artificielle (IA) est un domaine de l'informatique dédié à la création de machines capables d'effectuer des tâches qui nécessitent normalement l'intelligence humaine. Ces tâches incluent l'apprentissage, la résolution de problèmes, la reconnaissance de la parole et la prise de décisions.



LES DATES CLÉS DE L'INTELLIGENCE ARTIFICIELLE

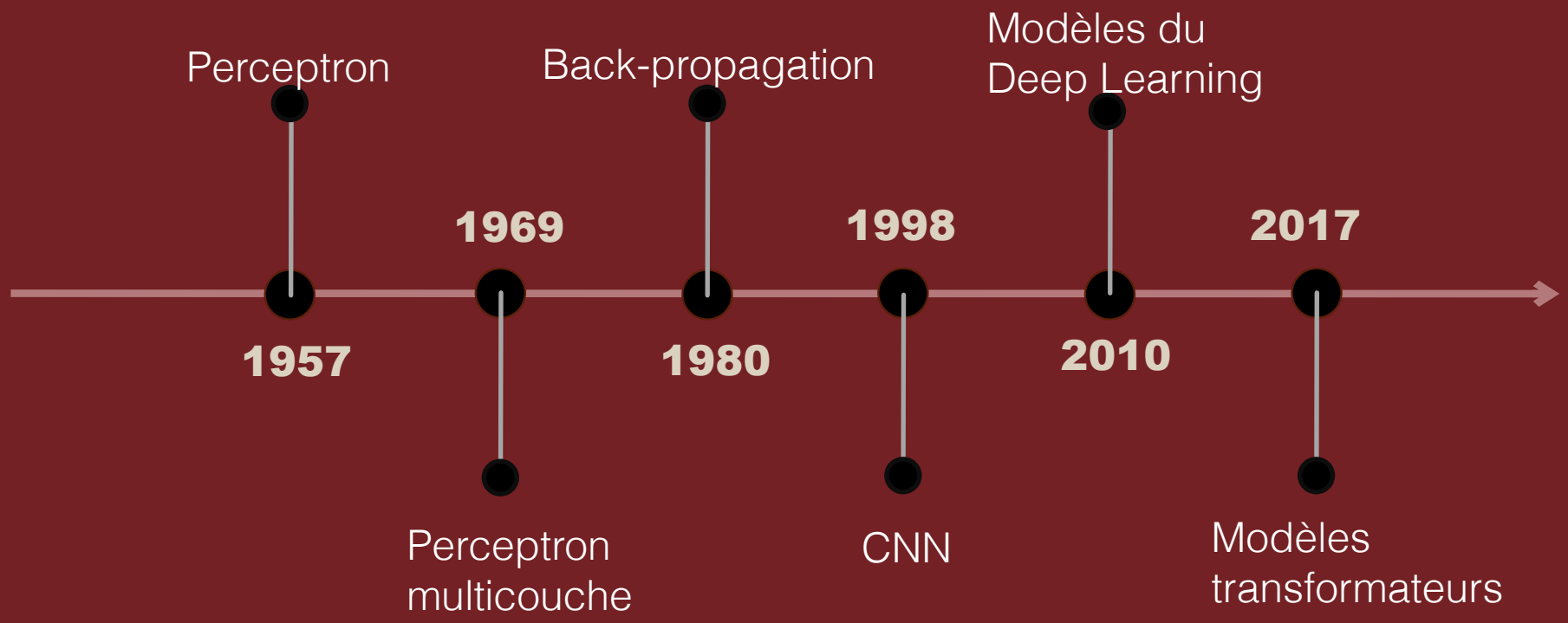
L'histoire de l 'IA

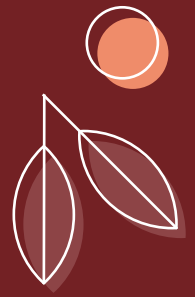
LE MACHINE LEARNING

"Le machine Learning fait référence à la capacité des systèmes informatiques à apprendre et à s'améliorer à partir de l'expérience sans être explicitement programmés. ." - Arthur Samuel.

Cette capacité d'auto-apprentissage, où les systèmes peuvent s'adapter et améliorer leurs performances au fil du temps, distingue le machine Learning de la programmation traditionnelle basée sur des règles.

Etapes importantes en Machine Learning





DEEP LEARNING

Le Deep Learning, ou apprentissage profond, est une sous-discipline de l'intelligence artificielle et du machine Learning qui utilise des réseaux de neurones artificiels pour modéliser et comprendre des motifs complexes dans de grandes quantités de données. En imitant la structure et le fonctionnement du cerveau humain, ces réseaux de neurones peuvent apprendre de manière autonome à partir de données non structurées comme des images, des textes ou des sons. Le Deep Learning est à la base de nombreuses applications modernes, notamment la reconnaissance d'image, la traduction automatique et les assistants virtuels.

LE JEU DU SERPENT


pygame window
Episode: 3000
Score: 4

- Nom : Jeu du Serpent (Snake)
- Genre : Jeu d'arcade classique
- Histoire et Origines
  - Origines : Créé dans les années 1970
  - Popularité : Popularisé par Nokia sur ses téléphones mobiles dans les années 1990
- Objectif du Jeu
  - Contrôler un serpent qui grandit en mangeant des objets (généralement des pommes) sans entrer en collision avec les murs ou son propre corps.
- Mécanismes de Jeu
  - Contrôles : Flèches directionnelles ou touches WASD
  - Croissance : Le serpent grandit avec chaque objet mangé, augmentant la difficulté.

LES RÉSEAUX NEURONAUX

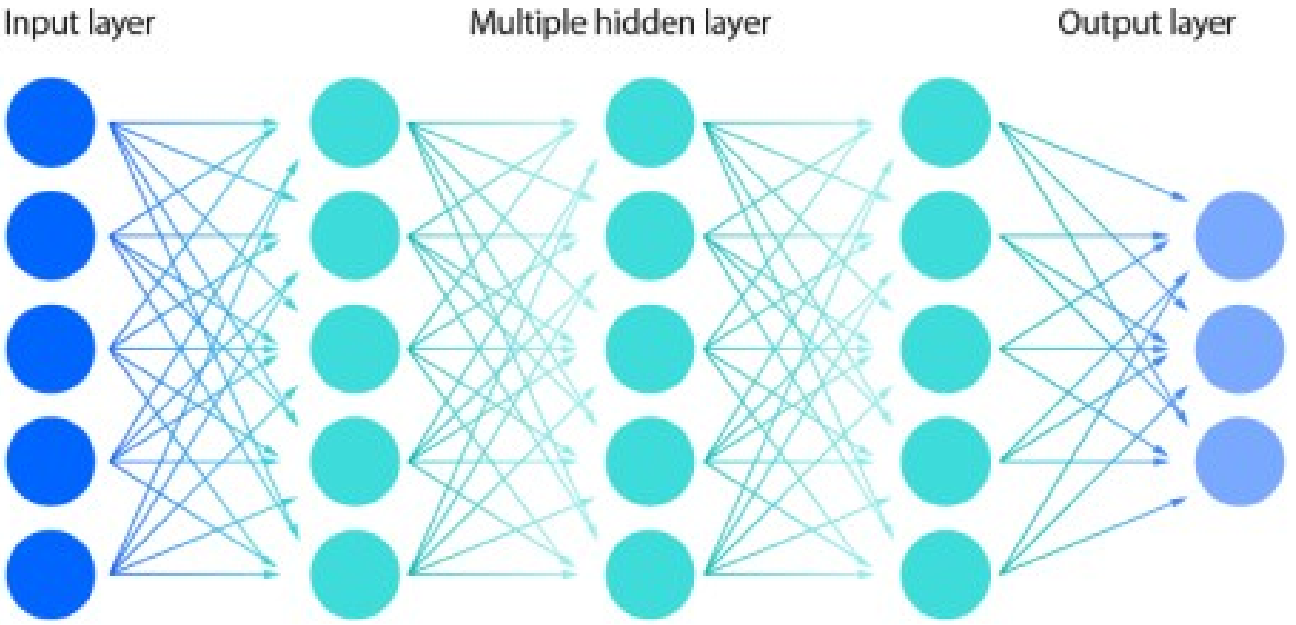
C’EST QUOI UN RÉSEAU DE NEURONES ARTIFICIELS ?

- Un réseau neuronal est une méthode d'intelligence artificielle qui apprend à des ordinateurs à traiter les données d'une manière inspirée par le cerveau humain.

- C'est un type de processus de machine learning appelé deep learning, qui exploite des nœuds, ou neurones, interconnectés dans une structure à plusieurs couches similaire au cerveau humain.

- Il crée un système adaptatif utilisé par les ordinateurs pour apprendre de leurs erreurs et s'améliorer en continu. Les réseaux neuronaux artificiels tentent de résoudre des problèmes complexes tels que résumer des documents ou reconnaître des visages, avec davantage de précision.

Deep neural network

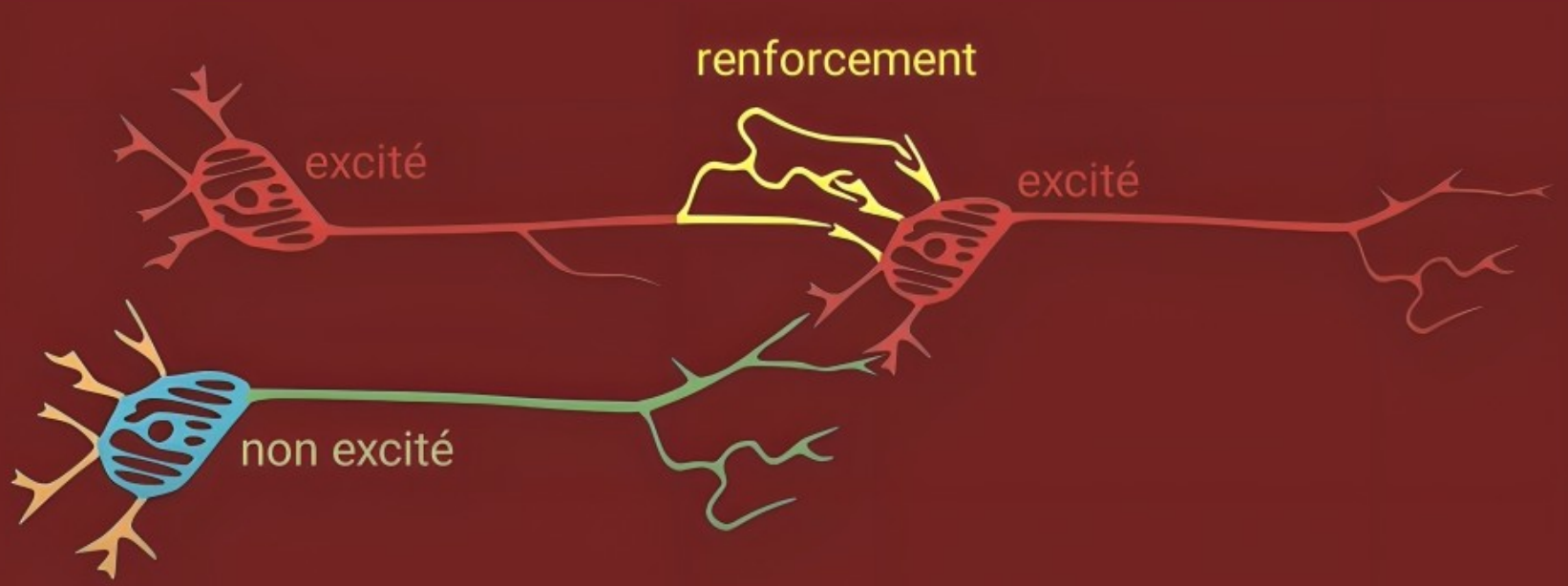


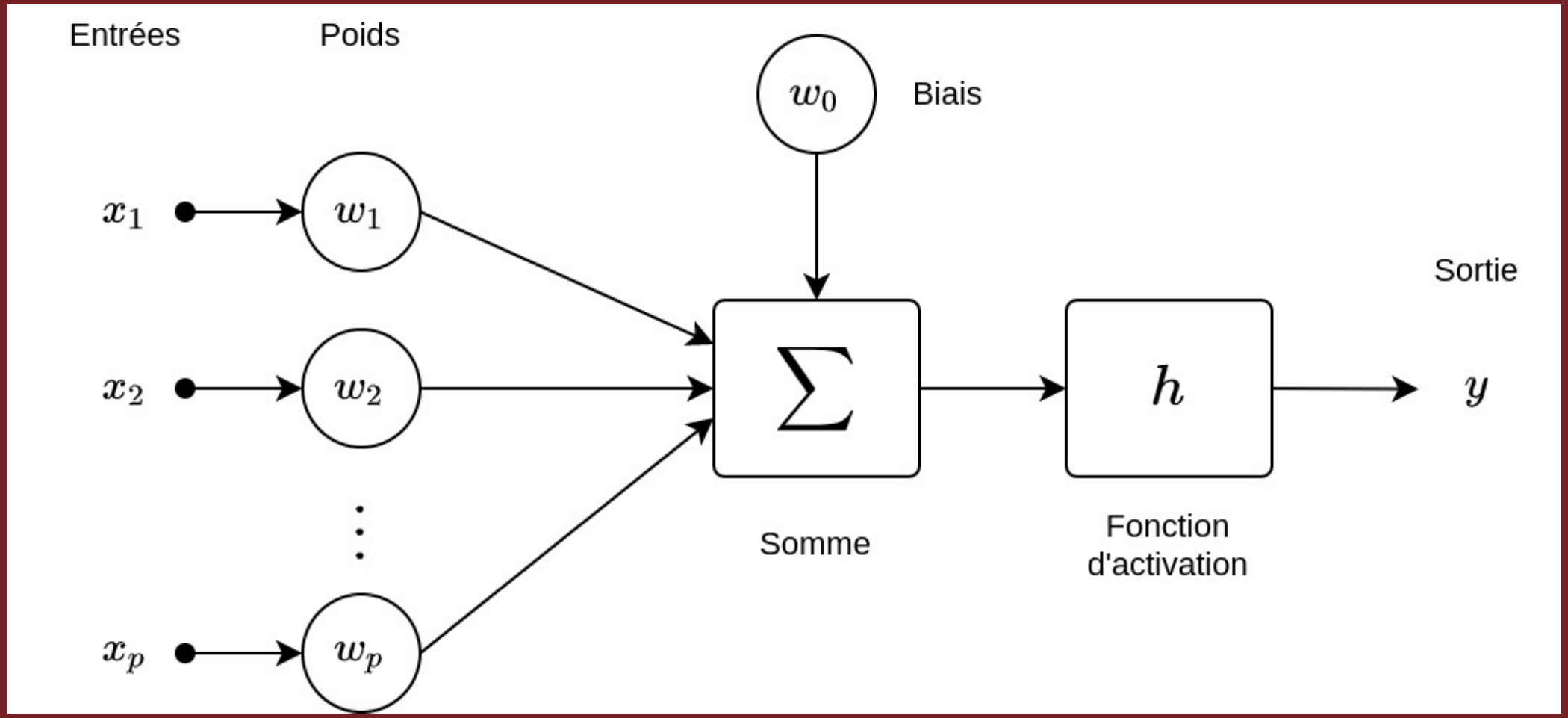
LE PERCEPTRON

Les perceptrons sont les éléments fondamentaux des réseaux neuronaux artificiels, simulant la façon dont les neurones biologiques reçoivent et traitent les entrées pour produire une sortie.

THÉORIE DE HEBB

Lorsque 2 neurons biologiques sont excités conjointement, alors ils renforcent leur lien synaptique .





$$y = h(\sum_{i=1}^{p} w_i.x_i + w_0)$$

ALGORITHME GRADIENT DESCENT

- Beaucoup d'algorithmes de Machine Learning obtiennent de bonnes performances grâce à leur entraînement.
- Cependant, une grande majorité des entraînements repose sur l'optimisation d'une fonction de perte.
- Plus sa valeur est petite, meilleurs sont les résultats de l'algorithme.
- Ainsi, l'algorithme de descente de gradient est un des nombreux moyens qui permettent de trouver le minimum (ou maximum) d'une fonction.

- La descente du gradient est un des nombreux algorithmes dits de descente. La formule générale est la suivante :

$$x_{t+1} = x_t - \eta \Delta f(x_t)$$

Le taux d'apprentissage

La direction de descente

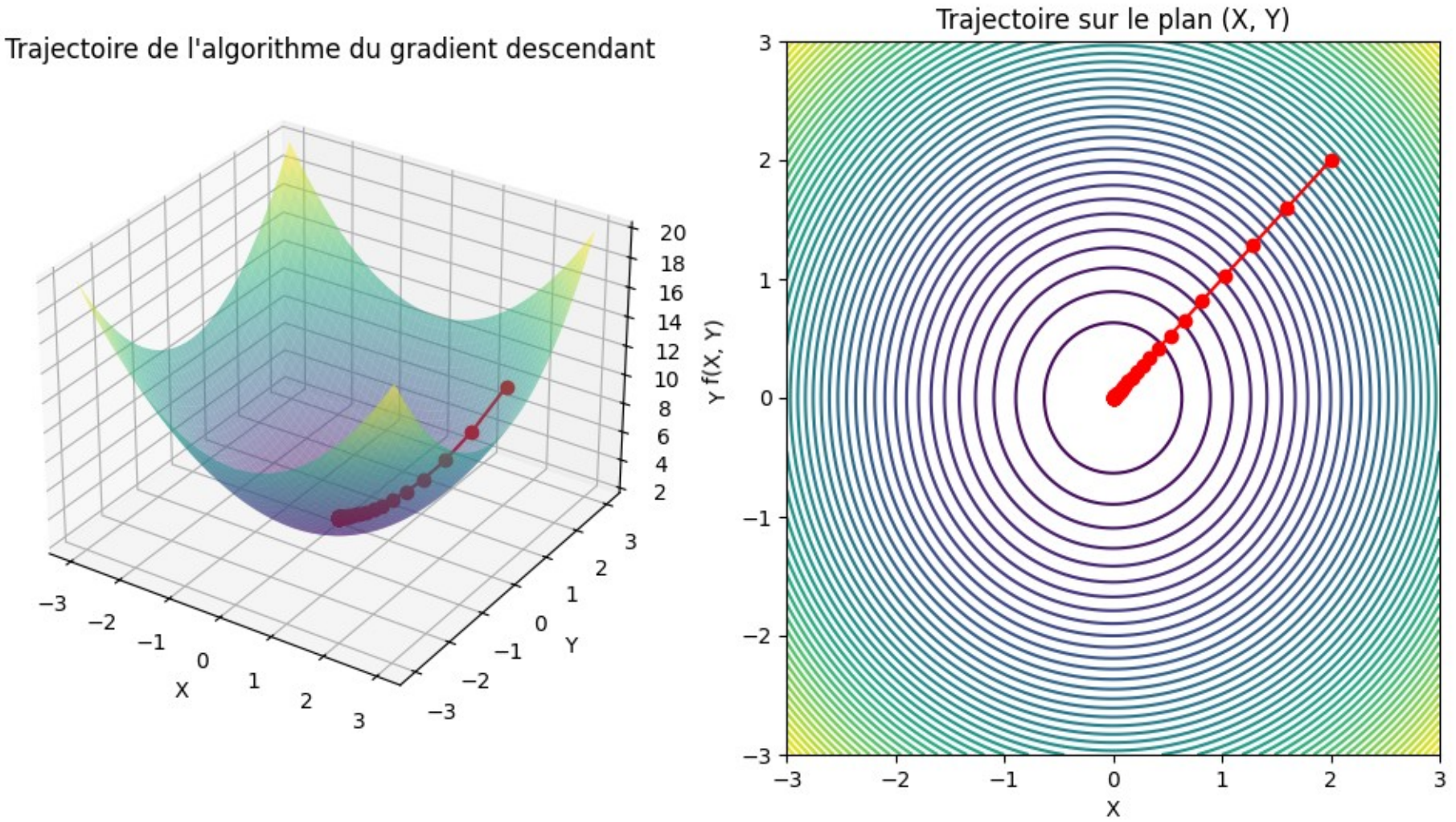
La fonction convexe que l'on souhaite minimiser.

- L'algorithme de descente du gradient décide de suivre comme direction de descente l'opposé du gradient d'une fonction convexe f.

Voici son fonctionnement:

1. Soit un point d'initialisation x_0 appartenant au domaine de f
2. Calculer $\Delta f(x_t)$
3. Mettre à jour les coordonnées: $x_{t+1} = x_t - \eta \Delta f(x_t)$
4. Répéter 2 et 3 jusqu'au critère d'arrêt

- Généralement, le critère d'arrêt est de la forme $|f(x_{t+1}) - f(x_t)| \leq \varepsilon$, où est très petit.
- Ce critère semble logique puisqu'on décide d'arrêter quand la différence entre les valeurs de la fonction prise en deux points différents est très peu significative, signe qu'on a convergé.
- Un critère d'arrêt reconnu dans le machine learning est celui du nombre d'itérations ; on réalise un nombre fini d'optimisation (étape 2 et 3).


Trajectoire de l'algorithme du gradient descendant
X
Y
Y f(X, Y)
Trajectoire sur le plan (X, Y)
X

LE Q-LEARNING-APPRENTISSAGE PAR RENFORCEMENT

Le Q-learning est une méthode d'apprentissage par renforcement où un agent apprend à prendre des décisions optimales en interagissant avec un environnement.

Voici comment le Q-learning peut être appliqué au jeu de Snake :

1) Initialisation: On initialise une table de Q-valeurs (Q-table) qui stocke les valeurs Q pour chaque paire état-action possible. L'agent de Q-learning utilise cette table pour évaluer la qualité des actions dans différents états du jeu.

État	Action Haut	Action Bas	Action Gauche	Action Droite
(Tête, Nourriture, Dir.)	Q	Q	Q	Q
(3, 5, "up", 7, 8)	0.5	-0.2	0.1	0.0
(4, 5, "down", 7, 8)	-0.1	0.3	0.0	0.2
...	...	...	...	...

2) États et Actions:

- États: Dans le jeu de Snake, un état peut être défini par la position de la tête du serpent, la position de la nourriture, et la direction actuelle du serpent.
- Actions: Les actions possibles sont les mouvements du serpent (haut, bas, gauche, droite).

3) Choix de l'Action:

À chaque étape du jeu, l'agent choisit une action basée sur une politique. Cette politique peut être une politique ε-greedy, où l'agent choisit l'action avec la meilleure valeur Q avec une probabilité de 1-ε, et une action aléatoire avec une probabilité de ε. Cela permet à l'agent d'explorer de nouvelles actions tout en exploitant les connaissances existantes.

4) Mise à Jour des Valeurs Q:

Après avoir effectué une action, l'agent observe la récompense et le nouvel état résultant de cette action. Il met à jour la valeur Q pour la paire état-action précédente en utilisant la formule :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

où :

- $Q(s,a)$ est la valeur Q actuelle pour l'état s et l'action a.
- α est le taux d'apprentissage, un paramètre entre 0 et 1.
- r est la récompense reçue après avoir effectué l'action a dans l'état s.
- γ est le facteur de discount (ou facteur d'actualisation), également un paramètre entre 0 et 1, qui mesure l'importance des récompenses futures.
- s' est le nouvel état après avoir pris l'action a.
- $\max_{a'} Q(s',a')$ est la valeur Q maximale pour toutes les actions possibles a' dans le nouvel état s'.

5) Récompenses:

Dans le jeu de Snake, les récompenses peuvent être définies de différentes manières :

- +1 pour manger de la nourriture,
- 0 pour un mouvement normal,
- -1 pour une collision (perte du jeu).

6) Apprentissage:

L'agent joue plusieurs épisodes du jeu, apprenant à chaque étape en mettant à jour les valeurs Q. Avec le temps, il apprend à éviter les obstacles, à trouver la nourriture efficacement et à prolonger la durée de vie du serpent.

Exemple d'initialisation d'une Q-table en Python pour le Jeu du Serpent (Snake)

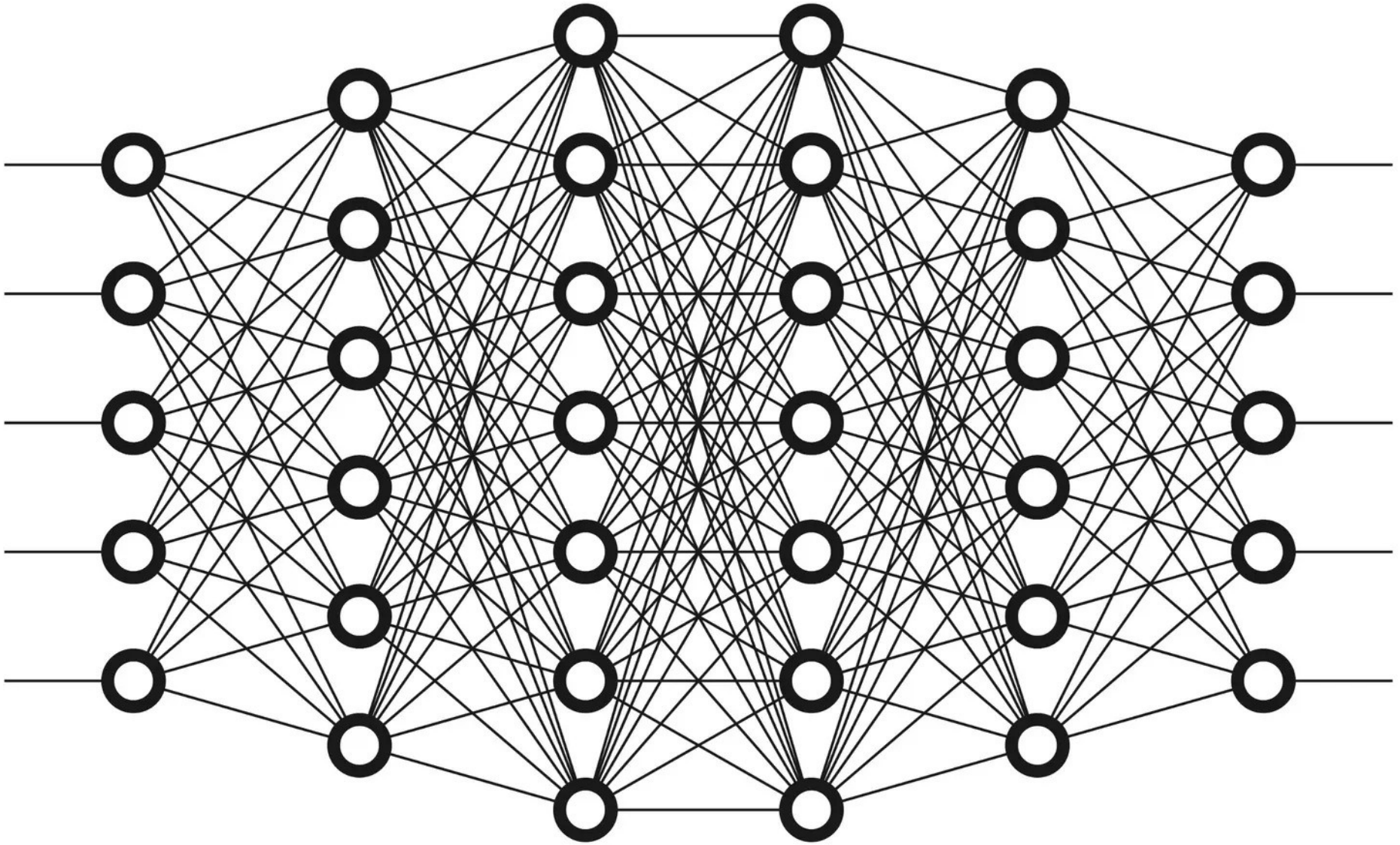
```python
import numpy as np

# Définir les états possibles et les actions
states = [(x, y, dir, fx, fy) for x in range(10) for y in range(10)
          for dir in ["up", "down", "left", "right"] for fx in range(10) for fy in range(10)]
actions = ["up", "down", "left", "right"]

# Initialiser la Q-table avec des zéros
q_table = {}
for state in states:
    q_table[state] = {action: 0.0 for action in actions}

# Exemple d'accès à une valeur Q
state = (3, 5, "up", 7, 8)
action = "up"
print(f"Valeur Q pour l'état {state} et l'action {action}: {q_table[state][action]}")
```

LE DEEP Q-LEARNING (DQL)



Le Deep Q-Learning (DQL) est une extension du Q-Learning traditionnel qui utilise des réseaux de neurones pour approximer la fonction de valeur Q, rendant l'algorithme capable de gérer des environnements avec des espaces d'état très grands ou continus.

Le processus de DQL pourrait ressembler à ceci :

1) L'état s est une représentation de la position de l'agent et des obstacles dans la grille.

2) Le réseau de neurones prend cet état comme entrée et sort des valeurs Q pour les actions "haut", "bas", "gauche", "droite".

3) L'agent choisit une action basée sur ces valeurs Q (avec une certaine exploration aléatoire).

4) Le jeu fournit une récompense basée sur le résultat de l'action (atteindre l'objectif, heurter un obstacle, etc.).

5) Le réseau de neurones est mis à jour pour minimiser la différence entre la valeur Q prédite et la récompense observée.

LE CODE PYTHON DU JEU DU SERPENT


pygame window
Episode: 3000
Score: 4

```python
import random
import numpy as np
import pygame
import pickle
import time

# color class
class Color:
    def __init__(self):
        self.white = (255, 255, 255)
        self.black = (0, 0, 0)
        self.red = (255, 0, 0)
        self.blue = (50, 150, 255)
        self.green = (200, 255, 0)
```

```python
class VisualSnake:
    def __init__(self):
        # whether to show episode number at the top
        self.show_episode = False
        self.episode = None

        # scale adjusts size of whole board (use 1.0 or 2.0)
        self.scale = 2
        self.game_width = int(600 * self.scale)
        self.game_height = int(400 * self.scale)

        # padding for score & episode
        self.padding = int(30 * self.scale)
        self.screen_width = self.game_width
        self.screen_height = self.game_height + self.padding
```

```
32          self.snake_size = int(10 * self.scale)
33          self.food_size = int(10 * self.scale)
34          self.snake_speed = 40
35
36          self.snake_coords = []
37          self.snake_length = 1
38          self.dir = "right"
39          self.board = np.zeros((self.game_height // self.snake_size, self.game_width // self.snake_size))
40
41          self.game_close = False
42
43
44          # starting location for the snake
45          self.x1 = self.game_width / 2
46          self.y1 = self.game_height / 2 + self.padding
47
48          self.r1, self.c1 = self.coords_to_index(self.x1, self.y1)
49          self.board[self.r1][self.c1] = 1
```

```
51          self.c_change = 1
52          self.r_change = 0
53
54          self.food_r, self.food_c = self.generate_food()
55          self.board[self.food_r][self.food_c] = 2
56          self.survived = 0
57          pygame.init()
58          self.color = Color()
59
60          self.screen = pygame.display.set_mode((self.screen_width, self.screen_height))
61          self.clock = pygame.time.Clock()
62
63          self.font = pygame.font.SysFont( name: "bahnschrift", int(18 * self.scale))
64          self.last_dir = None
65          self.step()
66
```

```
67      def print_score(self, score):
68          value = self.font.render( text: f"Score: {score}",  antialias: True, self.color.white)
69          self.screen.blit(value,  dest: [500 * self.scale, 10])
70
71      def print_episode(self):
72          if self.show_episode:
73              value = self.font.render( text: f"Episode: {self.episode}",  antialias: True, self.color.white)
74              self.screen.blit(value,  dest: [10, 10])
75
76      def draw_snake(self):
77          for i in range(len(self.snake_coords) - 1, -1, -1):
78              r, c = self.snake_coords[i]
79              x, y = self.index_to_coords(r, c)
80              if i == len(self.snake_coords) - 1:
81                  # head square color
82                  pygame.draw.rect(self.screen, self.color.blue,  rect: [x, y, self.snake_size, self.snake_size])
83              else:
84                  pygame.draw.rect(self.screen, self.color.green,  rect: [x, y, self.snake_size, self.snake_size])
```

```
86      def game_end_message(self):
87          mesg = self.font.render( text: "Game over!",  antialias: True, self.color.red)
88          self.screen.blit(mesg,  dest: [2 * self.game_width / 5, 2 * self.game_height / 5 + self.padding])
89
90      # is there danger in this square (body or wall)
91      def is_unsafe(self, r, c):
92          if self.valid_index(r, c):
93              if self.board[r][c] == 1:
94                  return 1
95              return 0
96          else:
97              return 1
98
```

```
    # returns tuple of 12 features
    def get_state(self):
        head_r, head_c = self.snake_coords[-1]
        state = []
        state.append(int(self.dir == "left"))
        state.append(int(self.dir == "right"))
        state.append(int(self.dir == "up"))
        state.append(int(self.dir == "down"))
        state.append(int(self.food_r < head_r))
        state.append(int(self.food_r > head_r))
        state.append(int(self.food_c < head_c))
        state.append(int(self.food_c > head_c))
        state.append(self.is_unsafe(head_r + 1, head_c))
        state.append(self.is_unsafe(head_r - 1, head_c))
        state.append(self.is_unsafe(head_r, head_c + 1))
        state.append(self.is_unsafe(head_r, head_c - 1))
        return tuple(state)
```

```
118     def valid_index(self, r, c):
119         return 0 <= r < len(self.board) and 0 <= c < len(self.board[0])
120 
121     # board coordinates <==> row, column conversions
122     def index_to_coords(self, r, c):
123         x = c * self.snake_size
124         y = r * self.snake_size + self.padding
125         return (x, y)
126     def coords_to_index(self, x, y):
127         r = int((y - self.padding) // self.snake_size)
128         c = int(x // self.snake_size)
129         return (r, c)
130 
131     # randomly place food
132     def generate_food(self):
133         food_c = int(round(random.randrange( start: 0, self.game_width - self.food_size) / self.food_size))
134         food_r = int(round(random.randrange( start: 0, self.game_height - self.food_size) / self.food_size))
135         if self.board[food_r][food_c] != 0:
136             food_r, food_c = self.generate_food()
137         return food_r, food_c
138 
139     def game_over(self):
140         return self.game_close
```

```
143     def step(self, action="None"):
144         if action == "None":
145             action = random.choice(["left", "right", "up", "down"])
146         else:
147             action = ["left", "right", "up", "down"][action]
148 
149         for event in pygame.event.get():
150             pass
151 
152         # take action
153         self.last_dir = self.dir
154         if action == "left" and (self.dir != "right" or self.snake_length == 1):
155             self.c_change = -1
156             self.r_change = 0
157             self.dir = "left"
158         elif action == "right" and (self.dir != "left" or self.snake_length == 1):
159             self.c_change = 1
160             self.r_change = 0
161             self.dir = "right"
162         elif action == "up" and (self.dir != "down" or self.snake_length == 1):
163             self.r_change = -1
164             self.c_change = 0
165             self.dir = "up"
166         elif action == "down" and (self.dir != "up" or self.snake_length == 1):
167             self.r_change = 1
168             self.c_change = 0
169             self.dir = "down"
```

```
172         if self.c1 >= self.game_width // self.snake_size or self.c1 < 0 or self.r1 >= self.game_height // self.snake_size or self.r1 < 0:
173             self.game_close = True
174         self.c1 += self.c_change
175         self.r1 += self.r_change
176 
177         self.screen.fill(self.color.black)
178         pygame.draw.rect(self.screen, color: (255, 255, 255), rect: (0, self.padding, self.game_width, self.game_height), width: 1)
179 
180 
181         food_x, food_y = self.index_to_coords(self.food_r, self.food_c)
182         pygame.draw.rect(self.screen, self.color.red, rect: [food_x, food_y, self.food_size, self.food_size])
183 
184         self.snake_coords.append((self.r1, self.c1))
185 
186         if self.valid_index(self.r1, self.c1):
187             self.board[self.r1][self.c1] = 1
188 
189         if len(self.snake_coords) > self.snake_length:
190             rd, cd = self.snake_coords[0]
191             del self.snake_coords[0]
192             if self.valid_index(rd, cd):
193                 self.board[rd][cd] = 0
```

```
199         self.draw_snake()
200         self.print_score(self.snake_length - 1)
201         self.print_episode()
202         pygame.display.update()
203 
204         # snake ate the food
205         if self.c1 == self.food_c and self.r1 == self.food_r:
206             self.food_r, self.food_c = self.generate_food()
207             self.board[self.food_r][self.food_c] = 2
208             self.snake_length += 1
209         self.survived += 1
```

```
211     def run_game(self, episode):
212         self.show_episode = True
213         self.episode = episode
214         self.print_episode()
215         pygame.display.update()
216 
217         # pass in pickle file with q table (stored in directory pickle with file name being episode #.pickle)
218         filename = f"pickle/{episode}.pickle"
219         with open(filename, 'rb') as file:
220             table = pickle.load(file)
221         time.sleep(5)
222         current_length = 2
223         steps_unchanged = 0
224         while not self.game_over():
225             if self.snake_length != current_length:
226                 steps_unchanged = 0
227                 current_length = self.snake_length
228             else:
229                 steps_unchanged += 1
```

```
232             state = self.get_state()
233             action = np.argmax(table[state])
234             if steps_unchanged == 1000:
235                 # stop if snake hasn't eaten anything in 1000 episodes (stuck in a loop)
236                 break
237             self.step(action)
238             self.clock.tick(self.snake_speed)
239         if self.game_over() == True:
240             # snake dies
241             self.screen.fill(self.color.black)
242             pygame.draw.rect(self.screen, color: (255, 255, 255), rect: (0, self.padding, self.game_width, self.game_height), width: 1)
243             self.game_end_message()
244             self.print_episode()
245             self.print_score(self.snake_length - 1)
246             pygame.display.update()
247             time.sleep(5)
248             pygame.quit()
249         return self.snake_length
250
251
252 VisualSnake().run_game("3000")
```

LE CODE PYTHON DE L'ENTRAÎNEMENT DU JEU DU SERPENT

```python
import numpy as np
import random
from snake_no_visual import LearnSnake
import pickle

class SnakeQAgent():
    def __init__(self):
        # define initial parameters
        self.discount_rate = 0.95
        self.learning_rate = 0.01
        self.eps = 1.0
        self.eps_discount = 0.9992
        self.min_eps = 0.001
        self.num_episodes = 10000
        self.table = np.zeros((2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4))
        self.env = LearnSnake()
        self.score = []
        self.survived = []
```

```
    # epsilon-greedy action choice
    def get_action(self, state):
        # select random action (exploration)
        if random.random() < self.eps:
            return random.choice([0, 1, 2, 3])

        # select best action (exploitation)
        return np.argmax(self.table[state])

    def train(self):
        for i in range(1, self.num_episodes + 1):
            self.env = LearnSnake()
            steps_without_food = 0
            length = self.env.snake_length

            # print updates
            if i % 25 == 0:
                print(f"Episodes: {i}, score: {np.mean(self.score)}, survived: {np.mean(self.survived)},`"
                      f" eps: {self.eps}, lr: {self.learning_rate}")
                self.score = []
                self.survived = []
```

```
        # occasionally save latest model
        if (i < 500 and i % 10 == 0) or (i >= 500 and i < 1000 and i % 200 == 0) or (i >= 1000 and i % 500 == 0):
            with open(f'pickle/{i}.pickle', 'wb') as file:
                pickle.dump(self.table, file)

        current_state = self.env.get_state()
        self.eps = max(self.eps * self.eps_discount, self.min_eps)
        done = False
        while not done:
            # choose action and take it
            action = self.get_action(current_state)
            new_state, reward, done = self.env.step(action)

            # Bellman Equation Update
            self.table[current_state][action] = (1 - self.learning_rate)\
                * self.table[current_state][action] + self.learning_rate\
                * (reward + self.discount_rate * max(self.table[new_state]))
            current_state = new_state

            steps_without_food += 1
            if length != self.env.snake_length:
                length = self.env.snake_length
                steps_without_food = 0
            if steps_without_food == 1000:
                # break out of loops
                break

        # keep track of important metrics
        self.score.append(self.env.snake_length - 1)
        self.survived.append(self.env.survived)
```


pygame window
Episode: 3000
Score: 4


pygame window
Episode: 3000
Score: 12

CONCLUSION

LES RÉFÉRENCES

- https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/
- https://robankhood.com/histoire-intelligence-artificielle/
- https://www.geeksforgeeks.org/introduction-deep-learning/
- https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53
- https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-feed-forward-network-in-deep-learning/

MERCI POUR VOTRE ATTENTION