ÉCOLE CENTRALE LYON

APPRO INF
RAPPORT

Rapport Stratégie de Résolution de Problèmes : Reinforcement Learning

---

Groupe (F2a_3) :
Aurélien VU NGOC
Alexandre BERNARD-MICHINOV

Enseignant(s) :
Alexandre SAÏDI

7 novembre 2019

Table des matières

Table des figures

Introduction

Le Machine Leaning, comme on en entend si bien parler, semble être un enjeu majeur de XXIe siècle. Toutes les entreprises exploite cette nouvelle méthode et ses applications sont toujours plus nombreuses. Le principe est simple : la machine "apprend" à répondre à un problème après s'être entraînée sur de nombreux exemples types (plus ou moins simples). Elle est ensuite mise en situation (en phase de test) et est capable de répondre à des problèmes complexes que la théorie ne parvenait pas parfaitement à résoudre par ordinateur. Cette approche de Machine Learning s'oppose directement à la notion de modèle, puisqu'on construit une intelligence uniquement à partir de l'expérience. En d'autres termes, les connaissances initiales sont très réduites.

Le jeu que nous étudions ici est simple : une souris cherche à trouver le fromage le plus rapidement possible dans un espace donné tout en évitant les murs, les pièges à souris et les bordures. D'autres éléments sont ajoutés comme des gouttes d'eau mais n'ont normalement pas d'influence ici. L'objectif ici est de faire apprendre à la machine à se diriger de la manière la plus optimale possible.

Il existe plusieurs façon de faire apprendre à une machine à jouer, mais le but de ce bureau d'étude est de mieux comprendre comment fonctionne **l'apprentissage par renforcement**. Nous allons donc utiliser ici cette méthode : le reinforcement learning ou Qlearning en référence à la matrice Q qu'elle utilise.

Chapitre 1

Reinforcement Learning / Q-learning

1.1 Qu'est-ce que l'apprentissage par renforcement ?

1.1.1 Matrice de récompenses R

Il s'agit d'une méthode par laquelle l'utilisateur connaît d'une part ce qu'il faut faire (de bien), et connaît toutes les inconnues du jeu. Le renforcement vient du fait qu'à chaque action qu'entreprend la machine, l'homme lui accorde une **récompense** (ou au contraire une pénalité). Nous allons donc se baser sur une matrice des récompense R :

Cette matrice se lit de la manière suivante : en partant d'une case i (parmi les n^2 cases de la grille), j'associe une action j (en l'occurrence le déplacement vers une autre case parmi les n^2) qui est récompensée par la valeur $R_{i,j}$. De cette manière, à chaque action qu'elle choisit, la machine est récompensée d'une certaine valeur (ou pénalisée bien entendu).

1.1.2 Matrice de probabilités Q

On entraîne la machine à travers la matrice de probabiltés Q. On va faire évoluer cette matrice à partir des récompenses obtenues successivement par les actions de la machine.

Cette matrice se lit de la manière suivante : en partant d'une case i (parmi les n^2 cases de la grille), j'associe une action j (en l'occurrence le déplacement vers une autre case parmi les n^2) qui doit arriver avec la probabilité $Q_{i,j}$. De cette manière, il suffit de choisir la plus grande probabilité pour savoir où aller à un moment donné.

1.2 Comment entraîne-t-on la machine ? Comment mettre à jour la matrice des probabilités

Une formule régit l'évolution des probabilités de la matrice Q.
Plusieurs facteur rentrent en jeu dans cette formule :

— $\underbrace{Q(s_t, a_t)}_{\text{old value}}$: l'ancienne valeur d'où l'évolution de la matrice

$$Q^{new}(s_t, a_t) = \overbrace{(1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

FIGURE 1.1 – Qlearning formula - *source : wikipedia*

— $\underbrace{\alpha}_{\text{learning rate}}$: le facteur d'apprentissage donne la vitesse d'évolution de la valeur dans Q. Plus la valeur de α est élevée, plus cette valeur dans Q change (mais risque de diverger) et plus la machine apprend vite. Lorsque $\alpha = 1$, est théoriquement optimale mais pose certains problèmes que nous ne détaillons pas ici. Lorsque $\alpha = 0$, la machine n'apprend rien ! On restera sur une valeur intermédiaire de $\alpha = 0.1$ et on étudiera par la suite l'influence de cette valeur.

— $\underbrace{r_t}_{\text{reward}}$: la récompense donnée par la matrice R de récompenses

— $\underbrace{\gamma}_{\text{discount factor}}$: le facteur de réduction pour changer l'influence de la récompense dans la mise à jour de la valeur dans Q. On veut le maximiser à 1, mais lorsque $\gamma = 1$, on observe des problèmes de propagation d'erreurs que l'on veut éviter. Pour cela on garde une valeur légèrement inférieure à 1 : $\gamma = 0.8$. On étudiera aussi par la suite l'influence de ce paramètre dans l'apprentissage de la machine.

— $\underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}$: une estimation de la valeur recherchée à partir des valeurs déjà présentes dans la matrice (on effectue un max sur toutes les actions possibles en étant sur la nouvelle case)

Chapitre 2

Présentation du code commenté

Dans l'optique de faire un code simple et lisible, nous avons choisit de programmer en utilisant la programmation orientée objet qui se prête particulièrement bien à ce genre d'exercice.

2.1 Une classe Grille ...

```
1  # -*- coding: utf-8 -*-
2  import numpy as np
3  import time
4
5  RWD_DECHARGE = -50
6  RWD_EAU = 50
7  RWD_FROMAGE = 100
8  GAMMA = 0.8 # discount factor, see https://en.wikipedia.org/wiki/Q-learning
9  ALPHA = 0.1 # learning rate, see https://en.wikipedia.org/wiki/Q-learning
10
11 #
       ================================================================================
12 #                                Classe Grille
13 #
       ================================================================================
14 class Grille :
15     #
       ================================================================================
16     #                                __init__
17     #
       ================================================================================
18     def __init__(self, dimension, fromage=None, lstMurs=None, lstDecharges=
       None, lstGouttesEau=None):
19         """ On initialise tous les éléments que l'on va trouver dans la
       grille """
20         # self.__grille = np.zeros((self.__dim,self.__dim))
21         self.__dim = dimension
22
23         # Souris dans la grille
```

```
24          # positionInit = (0,0)
25          # self.__souris = Souris(positionInit)
26          self.__positionSouris = (0,0)
27
28          # Eléments de la grille
29          self.__murs = lstMurs if lstMurs != None else []
30          self.__decharges = lstDecharges if lstDecharges != None else []
31          self.__gouttesEau = lstGouttesEau if lstGouttesEau != None else []
32          self.__fromage = fromage if fromage != None else (np.random.randint
        (self.__dim), np.random.randint(self.__dim))
33
34          # Matrice de récompenses
35          self.__rewardMatrix = np.matrix(np.ones((self.__dim**2, self.__dim
        **2))) # matrice de n^2*n^2 car on peut (potentiellement) aller de chaque
        case à chaque case,
36                                                  # ie n^2 états possibles vers n^2
         autre états possibles
37          self.__rewardMatrix *= -1 # matrice de -1 partout
38          self.updateRewardMatrix()
39
40
41          # Matrice de Proba
42          self.__gamma = GAMMA # discount factor, see https://en.wikipedia.
        org/wiki/Q-learning
43          self.__alpha = ALPHA # learning rate, see https://en.wikipedia.org/
        wiki/Q-learning
44          self.__probabilityMatrix = np.matrix(np.zeros([self.__dim**2, self.
        __dim**2])) # que des 0 initialement
45          print("INFO: Qmatrix init - {}".format(self.__probabilityMatrix))
46
47
48      #
        ============================================================================

49      #                                   add...()
50      #
        ============================================================================

51      def addGouttesEau(self, lstCases):
52          """ Rajoute des gouttes d'eau aux emplacements lstCases """
53          for case in lstCases:
54              self.__gouttesEau.append(case)
55          self.__gouttesEau = set(self.__gouttesEau) # on enlève les doublons
56          self.updateRewardMatrix()
57          # DEBUG: print("DEBUG: Eau ajoutée à la grille !")
58
59      def addMurs(self, lstCases):
60          """ Rajoute des murs aux emplacements lstCases """
61          for case in lstCases:
62              self.__murs.append(case)
63          self.__murs = set(self.__murs) # on enlève les doublons
64          self.updateRewardMatrix()
65          # DEBUG: print("DEBUG: Mur ajouté à la grille !")
66
67      def addDecharges(self, lstCases):
68          """ Rajoute des décharges aux emplacements lstCases """
69          for case in lstCases:
```

```
70              self.__decharges.append(case)
71          self.__decharges = set(self.__decharges) # on enlève les doublons
72          self.updateRewardMatrix()
73          # DEBUG: print("DEBUG: Decharge ajoutée à la grille !")
74
75      def addFromage(self, case):
76          """ Rajoute le fromage à l'emplacement case """
77          self.__fromage = case
78          self.updateRewardMatrix()
79          # DEBUG: print("DEBUG: Fromage ajoutée à la grille !")
80
81      #
        ================================================================================
82      #                                    set...()
83      #
        ================================================================================
84      def setPositionSouris(self, case):
85          """ Définit la position de la souris """
86          self.__positionSouris = case
87
88      #
        ================================================================================
89      #                                    affichage
90      #
        ================================================================================
91      def affichageGrille(self):
92          """ Renvoie un affichage dans la console de la grille
93          o : objectif;   s : souris;
94          # : mur;        '': case vide;
95          e : eau;        x : décharge
96          """
97          grilleAffichage = np.reshape(np.array(['']*self.__dim**2), (self.
        __dim,self.__dim)) # crée une grille de NxN remplie de ''
98
99          # Souris
100         grilleAffichage[self.__positionSouris] = 's'
101
102         # Fromage
103         grilleAffichage[self.__fromage] = 'o'
104
105         # Murs
106         for mur in self.__murs:
107             grilleAffichage[mur] = '#'
108
109         # Eau
110         for eau in self.__gouttesEau:
111             grilleAffichage[eau] = '~'
112
113         # Décharge
114         for decharge in self.__decharges:
115             grilleAffichage[decharge] = 'x'
116
117         # Affichage
```

```
118         print("INFO: grille - \n{}".format(grilleAffichage))
119         print("-"*40)
120 
121     #
    ================================================================================
122     #                              Fonctions utiles au jeu
123     #
    ================================================================================
124     def casesVoisinesDisponibles(self, case):
125         """ Contrôle quelles cases voisines peuvent être visitées par la
    souris """
126         i, j = case
127         voisins = []
128 
129         for v in [(i,j+1), (i,j-1), (i+1,j), (i-1,j)] : # nord/sud/est/
    ouest
130             # On vérifie que la case est dans la grille et n'est pas un mur
131             a, b = v
132             if a >= 0 and a < self.__dim and b >= 0 and b < self.__dim and
    v not in self.__murs:
133                 voisins.append(v)
134 
135         return voisins
136         # TODO: ajouter l'environnement : ie dé-privilégier les cases avec
    un environnement négatif (https://amunategui.github.io/reinforcement-
    learning/index.html)
137 
138     def fromTuple2caseNumber(self, case):
139         """ Transforme la position de la case (i, j) en son numéro """
140         i,j = case
141         return i*self.__dim+j
142 
143     def doSomething(self, currentState):
144         """ Fait faire quelque chose à la souris au hasard """
145         actionsPossibles = self.casesVoisinesDisponibles(currentState)
146         return actionsPossibles[np.random.randint(len(actionsPossibles))] #
     parmi les cases dispo, en choisir une random
147 
148     def train(self, iterations):
149         """ Entraîne le programme à jouer pendant n iterations"""
150         tempsInit = time.time()
151         for i in range(iterations):
152             rdm = np.random.randint(0, self.__dim**2) # une case au hasard
153             currentState = (rdm//self.__dim, rdm%self.__dim)
154             action = self.doSomething(currentState)
155             self.updateProbabilityMatrix(currentState, action, gamma=self.
    __gamma, alpha=self.__alpha)
156 
157         # Normalisons la matrice de probabilités "entraînée"
158         print("INFO: Trained Q matrix ({}s) -".format(time.time()-tempsInit
    ))
159         print(self.__probabilityMatrix/np.max(self.__probabilityMatrix)
    *100)
160         print('-'*40)
161 
```

```
162     def play(self, initialState):
163         """ Fait jouer le programme """
164         if type(initialState) == tuple :
165             initialState = self.fromTuple2caseNumber(initialState)
166         steps = [initialState]
167         current_state = initialState
168         print("INFO: Initial state - {}".format(initialState))
169         print("INFO: Fromage - {}".format(self.fromTuple2caseNumber(self.
    __fromage)))
170 
171         while current_state != self.fromTuple2caseNumber(self.__fromage) :
172             # prochaine étape
173             next_step_index = np.where(self.__probabilityMatrix[
    current_state,] == np.max(self.__probabilityMatrix[current_state,]))[1]
174 
175             if next_step_index.shape[0] > 1: # s'il y en a plusieurs (max
    atteint plusieurs fois)
176                 next_step_index = int(np.random.choice(next_step_index,
    size=1)) # on en choisit 1 au hasard
177             else:
178                 next_step_index = int(next_step_index)
179 
180             steps.append(next_step_index)
181             current_state = next_step_index
182             self.setPositionSouris((current_state//self.__dim,
    current_state%self.__dim))
183 
184         # Affiche le chemin selectionné
185         print("INFO: Selected path - {}".format(steps))
186         print("-"*40)
187 
188     #
    ================================================================================

189     #                Fonctions utiles à la matrice de récompenses
190     #
    ================================================================================

191     def updateRewardMatrix(self):
192         # reset
193         self.__rewardMatrix = np.matrix(np.ones((self.__dim**2, self.__dim
    **2))) # matrice de n²*n² car on peut (potentiellement) aller de chaque
    case à chaque case,
194                                         # ie n² états possibles vers n²
     autre états possibles
195         self.__rewardMatrix *= -1 # matrice de -1 partout
196 
197             # Decharges
198         for decharge in self.__decharges :
199             antecedents = self.casesVoisinesDisponibles(decharge)
200             # print("DEBUG: antécédents decharges - {}".format(antecedents)
    )
201             for v in antecedents :
202                 old, new = self.fromTuple2caseNumber(v), self.
    fromTuple2caseNumber(decharge)
203                 self.__rewardMatrix[old, new] = RWD_DECHARGE
204 
```

```
205                 # Gouttes d'eau
206         for goutte in self.__gouttesEau :
207             antecedents = self.casesVoisinesDisponibles(goutte)
208             # print("DEBUG: antécédents goutte - {}".format(antecedents))
209             for v in antecedents :
210                 old, new = self.fromTuple2caseNumber(v), self.
    fromTuple2caseNumber(goutte)
211                 self.__rewardMatrix[old, new] = RWD_EAU
212 
213                 # Fromage
214         antecedents = self.casesVoisinesDisponibles(self.__fromage)
215         # print("DEBUG: antécédents fromage - {}".format(antecedents))
216         for v in antecedents :
217             old, new = self.fromTuple2caseNumber(v), self.
    fromTuple2caseNumber(self.__fromage)
218             self.__rewardMatrix[old, new] = RWD_FROMAGE
219 
220         print("INFO: Reward Matrix - {}".format(self.__rewardMatrix))
221         print("-"*40)
222 
223 
224 
225     #
    ================================================================================

226     #                 Fonctions utiles à la matrice de proba (Q-matrix)
227     #
    ================================================================================

228     def updateProbabilityMatrix(self, currentState, action, gamma=0.8,
    alpha=0.1):
229         """ Met à jour la matrice de probabilités """
230         # tuple -> case number
231         if type(currentState) == tuple:
232             currentState = self.fromTuple2caseNumber(currentState)
233         if type(action) == tuple:
234             action = self.fromTuple2caseNumber(action)
235 
236         # Calculons l'estimation de la prochaine valeur optimale
237         maxIndex = np.where(self.__probabilityMatrix[action,] == np.max(
    self.__probabilityMatrix[action,]))[1]
238 
239         if maxIndex.shape[0] > 1: # s'il y a plus d'1 indice max
240             maxIndex = int(np.random.choice(maxIndex, size=1)) # on en
    prend 1 au hasard
241         else:
242             maxIndex = int(maxIndex)
243         maxValue = self.__probabilityMatrix[action, maxIndex]
244 
245         # Q learning formula
246         # self.__probabilityMatrix[currentState, action] += self.
    __rewardMatrix[currentState, action] + gamma * maxValue
247 
248         # Q learning formula (wikipedia)
249         self.__probabilityMatrix[currentState, action] = (1-alpha)*self.
    __probabilityMatrix[currentState, action] + alpha*(self.__rewardMatrix[
    currentState, action] + gamma*maxValue)
```

```
250
251         # DEBUG: print("Qmatrix = {}".format(self.__probabilityMatrix))
```

Listing 2.1 – Classe Grille

2.2 ... et le main.py

```
1  # -*- coding: utf-8 -*-
2  import numpy as np
3
4  from Grille import *
5
6  # ================ BASIC TEST ================
7  # On place des murs et de l'eau à des positions fixes.
8  # Le fromage est dans la 0-ième case de la grille (en haut à gauche)
9  # La souris est initialement dans le coin bas droit (en case 15)
10
11
12 # Initialisation
13 N = 4
14
15 fromage = (0, 0)
16 murs = [(1, 1), (1, 2), (2, 1)]
17 # decharges = [(3, 1)]
18 decharges = []
19 gouttes_eau = [(3, 1)]
20 # gouttesEau = []
21 g = Grille(N)
22 g.addMurs(murs)
23 g.addDecharges(decharges)
24 g.addGouttesEau(gouttes_eau)
25 g.addFromage(fromage)
26
27 # Entraînement
28 g.train(10000) # 10 000 itérations
29
30 # Test
31 # rdm = np.random.randint(N**2)
32 # position_initiale = (rdm//N, rdm%N)
33 position_initiale = (3, 3)
34 g.setPositionSouris(position_initiale)
35 g.affichageGrille()
36 g.play(position_initiale)
37
38 # ================ RANDOM TEST ================
39 # On choisit une grille de taille 20x20
40 # Nombre d'éléments de chaque sorte : entre N/2 et 80% de N (pour une
       taille 4,on a donc entre 2 et 3 murs/décharges/eau)
41 # Tous les éléments sont placés random
42 # Entraînement sur 10 000 essais
43 # Souris placée au hasard (pas sur un mur mais potentiellement sur une eau/
       décharge)
44
45
46 N = 20
```

```
47 nb_elements = int(0.8*N) # 80% de N
48 nb_murs = np.random.randint(N//2, nb_elements)
49 nb_decharges = np.random.randint(N//2, nb_elements)
50 nb_gouttes_eau = np.random.randint(N//2, nb_elements)
51 casesDisposPourAjoutEnvironnement = [(i, j) for i, j in np.ndindex((N, N))]
       # pour éviter qu'un même case ait plusieurs éléements d'environnement
52 fromage = casesDisposPourAjoutEnvironnement.pop(np.random.randint(len(
       casesDisposPourAjoutEnvironnement)))
53 murs = []
54 decharges = []
55 gouttes_eau = []
56 for i in range(nb_murs):
57     murs.append(casesDisposPourAjoutEnvironnement.pop(np.random.randint(len
       (casesDisposPourAjoutEnvironnement))))
58 for i in range(nb_decharges):
59     decharges.append(casesDisposPourAjoutEnvironnement.pop(np.random.
       randint(len(casesDisposPourAjoutEnvironnement))))
60 for i in range(nb_gouttes_eau):
61     gouttes_eau.append(casesDisposPourAjoutEnvironnement.pop(np.random.
       randint(len(casesDisposPourAjoutEnvironnement))))
62 g = Grille(N)
63 g.addMurs(murs)
64 g.addDecharges(decharges)
65 g.addGouttesEau(gouttes_eau)
66 g.addFromage(fromage)
67
68 # Entraînement
69 g.train(10000) # 10 000 itérations
70
71 # Test
72 rdm = np.random.randint(N**2)
73 while (rdm//N, rdm%N) in murs : # vérifions que position_initiale n'est pas
       un mur
74     rdm = np.random.randint(N**2)
75 position_initiale = (rdm//N, rdm%N)
76 g.setPositionSouris(position_initiale)
77 g.affichageGrille()
78 g.play(position_initiale)
```

Listing 2.2 – main.py

Chapitre 3

Analyse des résultats

3.1 Complexité

Nous essayons d'analyser la complexité temporelle de cet algorithme. Pour cela, changeons la taille de la grille et mesurons le temps d'entraînement :

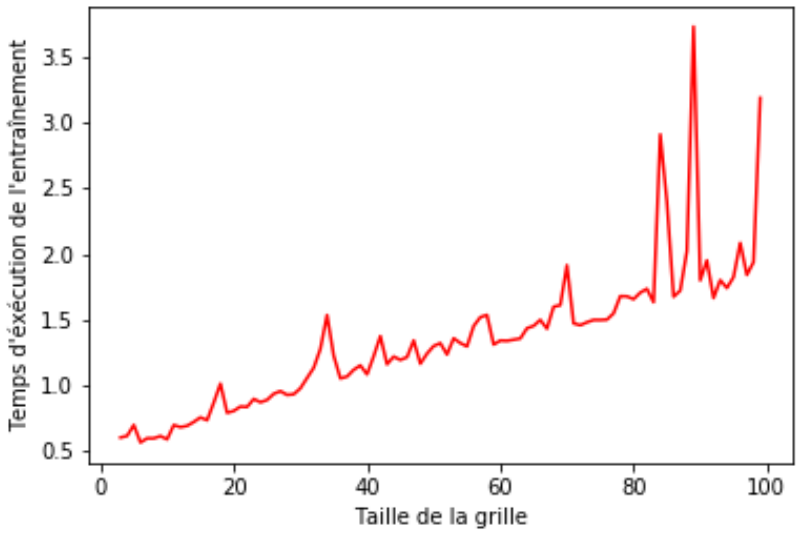

FIGURE 3.1 – Complexité temporelle

On constate que tant que la grille n'est pas trop grande, le temps d'éxécution augmente de manière linéaire avec la taille mais dès que cette taille devient trop importante le temps d'éxécution explose.

Conclusion

Le Q learning est une méthode puissante de machine learning et peut être améliorée en Deep Q learning lorsque l'on utilise un réseau de neurones. Les applications sont multiples en robotique, médecine, l'analyse de texte, en finances ...

En revanche, puisque cet algorithme se base sur un système de récompense, il faut pouvoir les déterminer précisemment. Il faut donc pour cela que le problème ait une liste des possibilités finies et connues à l'avance pour pouvoir déterminer récompenses à assigner à chaque action. On peut quand même imaginer un système on l'on définit la récompense lorsque l'ordi choisit une action, mais l'algorithme doit être modifié à cet effet.