

Model-free RL : Q-learning

Ghazi Bel Mufti

ESSAI - 2023

Table of Contents

- 1 Model-free reinforcement learning
- 2 Monte-Carlo
- 3 Temporal difference (TD) learning
- 4 Exploitation/Exploration
- 5 SARSA
- 6 Q-learning
- 7 Deep Q-Learning
- 8 Application à la navigation en conduite autonome

- 1 Model-free reinforcement learning
- 2 Monte-Carlo
- 3 Temporal difference (TD) learning
- 4 Exploitation/Exploration
- 5 SARSA
- 6 Q-learning
- 7 Deep Q-Learning
- 8 Application à la navigation en conduite autonome

Model-free reinforcement learning

- Les problèmes d'apprentissage par renforcement avec des actions discrètes peuvent souvent être modélisés à l'aide des PDM, mais l'agent n'a initialement aucune idée des probabilités de transitions (i.e. les $T(s, a, a')$) et ne sait pas quelles sont les récompenses (i.e. les $R(s, a, s')$).
- Il doit tester au moins une fois chaque état et chaque transition pour connaître les récompenses.
- Il doit le faire à plusieurs reprises s'il veut avoir une estimation raisonnable des probabilités de transition.
- Le Q-learning est l'un des principaux model-free, qui apprend la Q-function directement de l'expérience, sans nécessiter l'accès à un modèle : l'agent connaît initialement uniquement les états et les actions possibles et rien de plus.
- Avant d'aborder le Q-learning, nous commençons par présenter le Monte Carlo et le temporal difference learning.

- 1 Model-free reinforcement learning
- 2 Monte-Carlo
- 3 Temporal difference (TD) learning
- 4 Exploitation/Exploration
- 5 SARSA
- 6 Q-learning
- 7 Deep Q-Learning
- 8 Application à la navigation en conduite autonome

Monte-Carlo learning I

- Monte Carlo approaches require that the RL task is episodic, meaning that the task has a defined start and terminates after a finite number of actions, resulting in a total cumulative reward at the end of the episode : **Games are good examples of episodic RL tasks.**
- In Monte Carlo learning, the total cumulative reward at the end of the task is used to estimate either the value function V or the quality function Q by dividing the final reward equally among all of the intermediate states or state-action pairs, respectively.
- Consider the case of Monte Carlo learning of the value function. Given a new episode consisting of n steps, the cumulative discounted reward R_Σ is computed :

$$R_\Sigma = \sum_{t=1}^n \gamma^t r_t$$

and used to update the value function at every state s_t visited in this episode :

$$V(s_t)_{new} \leftarrow V(s_t)_{old} + \frac{1}{n} [R_\Sigma - V(s_t)_{old}], \forall t \in [1, \dots, n]$$

Monte-Carlo learning II

- Similarly, in the case of Monte Carlo learning of the Q function, the discounted reward R_Σ is used to update the Q function at every state-action pair (s_t, a_t) visited in this episode :

$$Q(s_t, a_t)_{new} \leftarrow Q(s_t, a_t)_{old} + \frac{1}{n}[R_\Sigma - Q(s_t, a_t)_{old}], \forall t \in [1, \dots, n]$$

- It is also possible to discount past experiences by introducing a learning rate $\alpha \in [0, 1]$ and using this to update the Q function :

$$Q(s_t, a_t)_{new} \leftarrow Q(s_t, a_t)_{old} + \alpha[R_\Sigma - Q(s_t, a_t)_{old}], \forall t \in [1, \dots, n]$$

- 1 Model-free reinforcement learning
- 2 Monte-Carlo
- 3 Temporal difference (TD) learning
- 4 Exploitation/Exploration
- 5 SARSA
- 6 Q-learning
- 7 Deep Q-Learning
- 8 Application à la navigation en conduite autonome

Temporal difference (TD) learning I

- In contrast to Monte Carlo learning, TD learning is not restricted to episodic tasks, but instead learns continuously by bootstrapping based on current estimates of the value function V or quality function Q ,
- To understand TD learning, it is helpful to begin with the simplest algorithm : TD(0).
- **TD(0) : 1-step look ahead**
 - ▶ In TD(0), the estimate of the one-step-ahead future reward is used to update the current value function.
 - ▶ Given a control trajectory generated through an optimal policy π , the value function at state s_t is given by

$$V(s_t) = \mathbb{E}(r_t + \gamma V(s_{t+1}))$$

- ▶ As $r_t + \gamma V(s_{t+1})$ is an unbiased estimator for $V(s_t)$, we update the value function based on the value function one step in the future :

$$V(s_t)_{new} \leftarrow V(s_t)_{old} + \alpha[r_t + \gamma V(s_{t+1})_{old} - V(s_t)_{old}]$$

Temporal difference (TD) learning II

$$V(s_t)_{new} \leftarrow V(s_t)_{old} + \alpha[r_t + \gamma V(s_{t+1})_{old} - V(s_t)_{old}]$$

- ★ where $r_t + \gamma V(s_{t+1})_{old}$ is the **TD estimate of R_{Σ}** ,
- ★ and $r_t + \gamma V(s_{t+1})_{old} - V(s_t)_{old}$ is the **TD error**.

- **TD(1)** uses a TD target estimate based on two steps into the future :

$$r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$$

- **TD(n)** converges to the Monte Carlo learning approach.

- 1 Model-free reinforcement learning
- 2 Monte-Carlo
- 3 Temporal difference (TD) learning
- 4 Exploitation/Exploration**
- 5 SARSA
- 6 Q-learning
- 7 Deep Q-Learning
- 8 Application à la navigation en conduite autonome

Choix des actions : Exploitation/Exploration

- Un agent apprenant est sujet au compromis entre :
 - ▶ *l'exploitation* : refaire des actions, dont il sait qu'elles vont lui donner de bonnes récompenses,
 - ▶ *l'exploration* : essayer de nouvelles actions, pour apprendre de nouvelles choses.
- Sélection d'action ϵ – *greedy* où $\epsilon \in [0, 1]$:
 - (a) tirer uniformément un nombre p au hasard dans l'intervalle $[0, 1]$:
 - (b) si $p < \epsilon$, tirer une action a_t dans A (*exploration*).
 - (c) si $p \geq \epsilon$, agir de façon gourmande (*greedy*), en choisissant l'action a_t dans A ayant la valeur Q la plus élevée (*exploitation*).

- 1 Model-free reinforcement learning
- 2 Monte-Carlo
- 3 Temporal difference (TD) learning
- 4 Exploitation/Exploration
- 5 SARSA**
- 6 Q-learning
- 7 Deep Q-Learning
- 8 Application à la navigation en conduite autonome

SARSA : State–Action–Reward–State–Action learning I

- SARSA is a popular TD algorithm that is used to learn the Q function **on-policy**.
- The Q update equation in SARSA is nearly identical to the V update equation in TD(0)

$$Q(s_t, a_t)_{new} \leftarrow Q(s_t, a_t)_{old} + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)_{old}]$$

- SARSA is considered on-policy because it uses the action a_{t+1} based on the actual policy $\pi : a_{t+1} = \pi(s_{t+1})$.

SARSA : State–Action–Reward–State–Action learning II

SARSA ALGORITHM

- Initialise the Q -table

Say $Q(s, a) = 0$ for all (s, a) pairs.

Repeat for N episodes

- $a = \epsilon$ -greedy at S_0

Repeat for k steps

- At any state S

- 1 Execute a , get reward $r \rightarrow$ state S'

- 2 Next action, $a' = \epsilon$ -greedy at S'

- 3 Get $Q(S', a')$ from the table.

- 4 Update $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$

- 5 Set $a \leftarrow a'$, $S \leftarrow S'$

- 1 Model-free reinforcement learning
- 2 Monte-Carlo
- 3 Temporal difference (TD) learning
- 4 Exploitation/Exploration
- 5 SARSA
- 6 Q-learning**
- 7 Deep Q-Learning
- 8 Application à la navigation en conduite autonome

Q-learning I

- Q-learning is one of the most central approaches in model-free RL.
- Q-learning is essentially an **off-policy** TD learning scheme for the Q function.
- In Q-learning, the Q update equation is :

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha [r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

- Notice that the only difference between Q-learning and SARSA is that SARSA uses $Q(s_{t+1}, a_{t+1})$ for the TD target, while Q-learning uses $\max_a Q(s_{t+1}, a)$ for the TD target.
- In contrast to SARSA, Q-learning is off-policy because it uses the optimal a for the update based on the current estimate for Q, while taking a different action a_{t+1} based on a different behavior policy.

Q-learning II

Algorithm 1 Q-learning

Input: espace (S, A) , ϵ , α , γ

Output: fonction valeur Q , actions émises

initialiser Q arbitrairement

boucle

initialiser l'état initial s de l'épisode

répéter

$a \leftarrow \pi_Q^\epsilon(s)$ (ϵ -greedy)

émettre a ; observer r et s'

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

$s \leftarrow s'$

jusqu'à s terminal

fin boucle

Q-learning III

Q - Learning.

- Initialise the Q-table

Say $Q(s,a) = 0$ for all (s,a) pairs.

Repeat for N episodes

Repeat for k steps

- At any state s

1 $a = \epsilon$ -greedy at s

2 Execute a , get reward $r \rightarrow$ state s'

3 Get $\max_{a_i} Q(s', a_i)$ from the table.

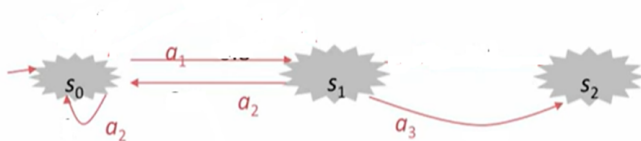
4 Update $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma \max_{a_i} Q(s', a_i) - Q(s,a)]$

5 $s \leftarrow s'$

...

Exemple 1 I

- On considère le Processus de Décision Markovien (PDM) suivant :



- L'environnement est constitué de 3 états s_0 , s_1 et s_2 , où s_0 est l'état initial et s_2 est l'état final.
- Etant dans l'état s_0 , l'agent a deux actions possibles : a_1 qui l'amène à l'état s_1 et a_2 qui l'amène à l'état s_0 .
- La fonction de récompense associée à ce PDM est donnée par : $R(s_0) = -0.1$, $R(s_1) = -0.1$ et $R(s_2) = 1$.
- On voudrait utiliser l'algorithme du Q-learning pour déterminer l'action à choisir par l'agent lorsqu'il se trouve dans un état donné. On suppose que $\alpha = 0.5$ et $\gamma = 0.5$ et que, au démarrage, l'agent se trouve à l'état s_0 et choisit l'action a_2 .
- Déterminer, en utilisant l'algorithme du Q-learning, l'action suivante qui sera choisie par l'agent.

Exemple 1 II

- Rappelons que la formule de mise à jour de la Q-table :

$$Q(s, a) = Q(s, a) + \alpha(R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a))$$



- Observations: $(s_0)_{-0.1}$

On ne fait rien (on a besoin d'un triplet (s, a, s'))

- Action à prendre $\pi(s_0) = \operatorname{argmax}\{Q(s_0, a_1), Q(s_0, a_2)\}$
 $= \operatorname{argmax}\{0, 0\}$
 $= a_2$ (arbitraire, ça aurait aussi pu être a_1)

Exemple 1 III

- Rappelons que la formule de mise à jour de la Q-table :

$$Q(s, a) = Q(s, a) + \alpha(R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

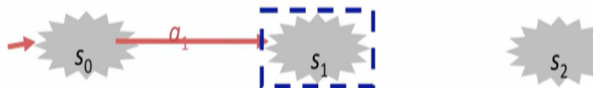


- Observations: $(s_0)_{-0.1} \xrightarrow{a_2} (s_0)_{-0.1}$

$$\begin{aligned} Q(s_0, a_2) &\leftarrow Q(s_0, a_2) + \alpha (R(s_0) + \gamma \max\{Q(s_0, a_1), Q(s_0, a_2)\} - Q(s_0, a_2)) \\ &= 0 + 0.5 (-0.1 + 0.5 \max\{0, 0\} - 0) \\ &= -0.05 \end{aligned}$$

- Action à prendre $\pi(s_0) = \operatorname{argmax}\{Q(s_0, a_1), Q(s_0, a_2)\}$
 $= \operatorname{argmax}\{0, -0.05\}$
 $= a_1$ (changement de politique!)

Exemple 1 IV

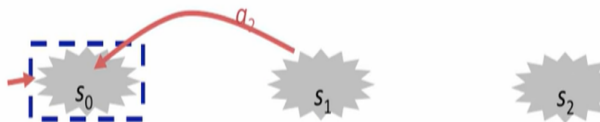


- Observations: $(s_0) \xrightarrow{-0.1, a_2} (s_0) \xrightarrow{-0.1, a_1} (s_1) \xrightarrow{-0.1}$

$$\begin{aligned} Q(s_0, a_1) &\leftarrow Q(s_0, a_1) + \alpha (R(s_0) + \gamma \max\{ Q(s_1, a_2), Q(s_1, a_3) \} - Q(s_0, a_1)) \\ &= 0 + 0.5 (-0.1 + 0.5 \max\{ 0, 0 \} - 0) \\ &= -0.05 \end{aligned}$$

- Action à prendre $\pi(s_1) = \operatorname{argmax}\{ Q(s_1, a_2), Q(s_1, a_3) \}$
 $= \operatorname{argmax}\{ 0, 0 \}$
 $= a_2$ (arbitraire, ça aurait aussi pu être a_3)

Exemple 1 V

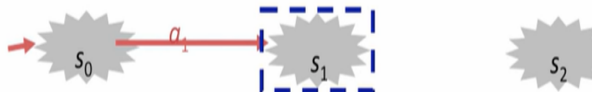


- Observations: $(s_0) \xrightarrow{-0.1, a_2} (s_0) \xrightarrow{-0.1, a_1} (s_1) \xrightarrow{-0.1, a_2} (s_0)$

$$\begin{aligned} Q(s_1, a_2) &\leftarrow Q(s_1, a_2) + \alpha (R(s_1) + \gamma \max\{Q(s_0, a_1), Q(s_0, a_2)\} - Q(s_1, a_2)) \\ &= 0 + 0.5 (-0.1 + 0.5 \max\{-0.05, -0.05\} + 0) \\ &= -0.0625 \end{aligned}$$

- Action à prendre $\pi(s_0) = \operatorname{argmax}\{Q(s_0, a_1), Q(s_0, a_2)\}$
 $= \operatorname{argmax}\{-0.05, -0.05\}$
 $= a_1$ (arbitraire, ça aurait aussi pu être a_2)

Exemple 1 VI



- Observations: $(s_0) \xrightarrow{a_2}_{-0.1} (s_0) \xrightarrow{a_1}_{-0.1} (s_1) \xrightarrow{a_2}_{-0.1} (s_0) \xrightarrow{a_1}_{-0.1} (s_1)$

$$Q(s_0, a_1) \leftarrow Q(s_0, a_1) + \alpha (R(s_0) + \gamma \max\{Q(s_1, a_2), Q(s_1, a_3)\} - Q(s_0, a_1))$$

$$= -0.05 + 0.5 (-0.1 + 0.5 \max\{-0.0625, 0\} + 0.05)$$

$$= -0.075$$
- Action à prendre $\pi(s_1) = \operatorname{argmax}\{Q(s_1, a_2), Q(s_1, a_3)\}$

$$= \operatorname{argmax}\{-0.0625, 0\}$$

$$= a_3 \quad \text{(changement de politique!)}$$

Exemple 1 VII



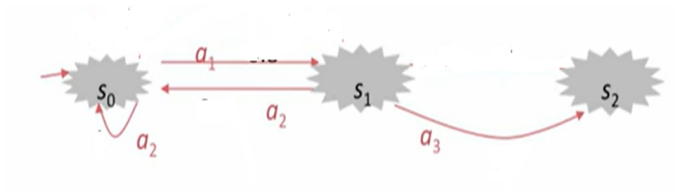
- Observations: $(s_0) \xrightarrow{-0.1, a_2} (s_0) \xrightarrow{-0.1, a_1} (s_1) \xrightarrow{-0.1, a_2} (s_0) \xrightarrow{-0.1, a_1} (s_1) \xrightarrow{-0.1, a_3} (s_2) \quad 1$

État terminal: $Q(s_2, \text{None}) = 1$

$$\begin{aligned} Q(s_1, a_3) &\leftarrow Q(s_1, a_3) + \alpha (R(s_1) + \gamma \max\{Q(s_2, \text{None})\} - Q(s_1, a_3)) \\ &= 0 + 0.5 (-0.1 + 0.5 \max\{1\} + 0) \\ &= 0.2 \end{aligned}$$

- On recommence un nouvel essai...




Exemple 1 VIII



Etats	Actions	
s_0	$Q(s_0, a_1) = -0,075$	$Q(s_0, a_2) = -0,05$
s_1	$Q(s_1, a_2) = -0,062$	$Q(s_1, a_3) = 0,20$

Exemple2 I

- Un environnement simple composé d'une grille carrée à 9 cases numérotées de 1 à 9 dont chacune représente un état.
- L'agent, représenté sur la grille par l'icône de la voiture, se trouve initialement à l'état 7.
- L'agent peut se déplacer vers le haut, le bas, la gauche ou la droite.
- Les récompenses instantanées sont définies selon l'état de l'agent : -1 si état 5, 1 si état 3 et 0 partout ailleurs.

1	2	3  +1
4	5  -1	6
7 	8	9

Exemple2 II

1 Initialisation de la Q -table

- ▶ Nombre d'époques = 100
- ▶ Learning rate $\alpha = 0.1$
- ▶ Paramètre de rabais $\gamma = 0.9$

État	$a = 0(\uparrow)$	$a = 1(\downarrow)$	$a = 2(\leftarrow)$	$a = 3(\rightarrow)$
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0

Exemple2 III

$$Q(s_t, a_t)_{new} \leftarrow Q(s_t, a_t)_{old} + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)_{old}]$$

② La fonction `take_action` détermine l'action à adopter :

- ▶ Si $\epsilon > 0$ l'action est choisie selon la stratégie $\epsilon - greedy$...
- ▶ Sinon (i.e. $\epsilon == 0$) l'action choisie est celle réalisant le *max* de la fonction Q pour l'état dans lequel se trouve l'agent.

```
def take_action(st, Q, eps):  
    # Take an action  
    if random.uniform(0, 1) < eps:  
        action = randint(0, 3)  
    else: # Or greedy action  
        action = np.argmax(Q[st])  
    return action
```

Exemple2 IV

$$Q(s_t, a_t)_{new} \leftarrow Q(s_t, a_t)_{old} + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)_{old}]$$

3 Mise à jour de la Q-table

1. Déterminer a_t selon la stratégie $\epsilon - greedy$ et en déduire s_{t+1} et r à l'aide de la fonction `env_step`
2. Déterminer a_{t+1} comme l'action réalisant le *max* de la fonction Q pour l'état s_{t+1}
3. Mettre à jour la Q-table : calcul de $Q(s_t, a_t)_{new}$

```
while not env.is_finished():  
  
    at = take_action(st, Q, 0.4)  
  
    stp1, r = env.step(at)  
  
    atp1 = take_action(stp1, Q, 0.0)  
  
    Q[st][at] = Q[st][at] + 0.1*(r + 0.9*Q[stp1][atp1] - Q[st][at])  
  
    st = stp1
```

Exemple2 V

4 Q-table avec $\epsilon = 0.1$

```
1 [0.2677130454492659, 0.12110277767834766, 0.2705860048296834, 0.899769557818689]
2 [0.30351044155604734, -0.09039852293751256, 0.30738858546192843, 0.9999734386011123]
3 [0, 0, 0, 0]
4 [0.8086239836436035, 0.06949662245198857, 0.1844721569688091, -0.1825757655402866]
5 [0.7227973059729618, 0.04196239267462237, 0.07060218337601808, 0.001420317959587902]
6 [0, 0, 0, 0]
7 [0.4262570334192201, 0.03535177348216714, 0.014786944522645666, 0]
8 [-0.09982101592, 0.0002892270284383321, 0.04695190982210117, 0]
9 [0, 0, 0, 0]
```

5 Q-table avec $\epsilon = 0.4$

```
1 [0.5077926289431602, 0.327045029402648, 0.5166111900534934, 0.8996087946614753]
2 [0.540336174155807, -0.36853537974311984, 0.5579775246750276, 0.9999314403867587]
3 [0, 0, 0, 0]
4 [0.8076806238306969, 0.4258918220131581, 0.48916225424449944, -0.4447172032599095]
5 [0.7513728124027056, 0.21479897891295316, 0.3773557877396743, 0.24773033123304058]
6 [0.6125795110000001, 0, -0.09643744798273, 0.030951000000000006]
7 [0.7000735017708594, 0.11140683868992345, 0.2721689177416005, 0.06338101703237808]
8 [-0.27660186671095865, 0.028699685395430358, 0.3999206033207965, 2.3864555448108373e-05]
9 [0.024390000000000005, 0, 0.00017993117202938855, 0]
```


- 1 Model-free reinforcement learning
- 2 Monte-Carlo
- 3 Temporal difference (TD) learning
- 4 Exploitation/Exploration
- 5 SARSA
- 6 Q-learning
- 7 Deep Q-Learning**
- 8 Application à la navigation en conduite autonome

Deep Q-Learning

- Le Q-learning s'adapte mal aux PDM de grande, voire moyenne, taille, avec un nombre élevé d'états.
- Par exemple, lorsque l'environnement d'apprentissage par renforcement est un simulateur de conduite autonome où un état est défini par les images fournies par la caméra frontale du véhicule : deux images différentes représentent deux états différents.

Deep Q-Learning

- Le Q-learning s'adapte mal aux PDM de grande, voire moyenne, taille, avec un nombre élevé d'états.
- Par exemple, lorsque l'environnement d'apprentissage par renforcement est un simulateur de conduite autonome où un état est défini par les images fournies par la caméra frontale du véhicule : deux images différentes représentent deux états différents.

Mnih et al. [2013] propose l'utilisation des réseaux de neurones afin d'avoir les meilleures estimations de la Q -valeur donnée par l'équation d'optimalité de Bellman.

- Un réseau de neurones utilisé pour estimer la Q valeur est appelé *Deep Q-network (DQN)*.
- L'utilisation d'un *DQN* pour l'approximation de la valeur Q s'appelle *Deep Q-Learning*.

Deep Q-Learning

Pseudo algorithme

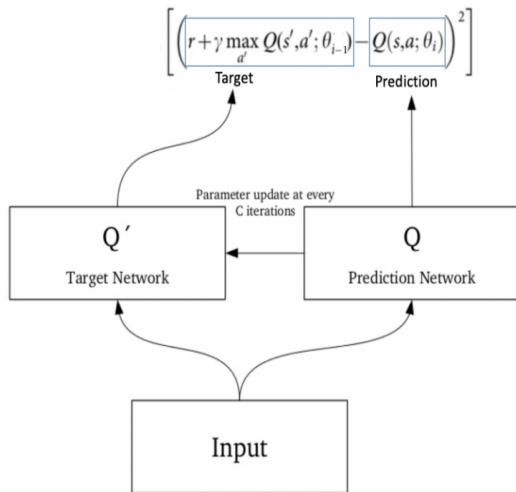
- 1 Transmettre l'état au *DQN*
- 2 Récupérer les valeurs de Q correspondant à chaque action $Q^\pi(s_{t+1}, a), a \in A$
- 3 Choisir l'action ($\epsilon - greedy$)
- 4 Calculer la valeur Q cible :

$$Q_{cible} = r_t + \gamma \max_a Q^\pi(s_{t+1}, a)$$

- 5 Exécuter une itération d'entraînement du modèle à l'aide de tout algorithme de descente de gradient en minimisant l'erreur entre la valeur Q de l'action choisie à l'étape (3.) et la Q_{cible}

Architecture du réseau de DQN

- In deep Q-learning, we estimate TD-target y_i and $Q(s, a)$ separately by two different neural networks, often called the **Target-network** and **Q-networks**.
- The parameters θ_{i-1} (weights, biases) belong to the target-network, while θ_i belong to the Q-network.
- Given the current state s , the **Q-network** calculates the action-values ($Q(s, a)$). At the same time the **Target-network** uses the next state s' to calculate $Q(s', a)$ for the TD target.



Experience Replay And Replay Memory

- ▶ With deep Q-networks, we often utilize this technique called experience replay during training.
- ▶ With experience replay, we store the agent's experiences at each time step in a data set called the replay memory.
- ▶ We represent the agent's experience at time t as e_t .
- ▶ At time t , the agent's experience e_t is defined as this tuple :

$$e_t = (s_t, a_t, r_t, s_{t+1})$$



Experience Replay And Replay Memory II

Gaining experiences

- ① Initialize replay memory capacity.
 - ② Initialize the network with random weights.
 - ③ For each episode :
 - Initialize the starting state.
 - For each time step :
 1. Select an action : Via exploration or exploitation
 2. Execute selected action in an emulator.
 3. Observe reward and next state.
 4. Store experience in replay memory D .
- All of the agent's experiences at each time step over all episodes played by the agent are stored in the replay memory.
 - In practice, we'll usually see the replay memory set to some finite size limit, N , and therefore, it will only store the last N experiences.
 - This replay memory data set is what we'll randomly sample the minibatch from to train the network.
 - **Why Use Experience Replay ?** A key reason for using replay memory is to break the correlation between consecutive samples.

Liens utiles pour le DRL

[1] Keras RL

- ▶ Build Agent with Keras-RL, N. Renote. (voir le lien github pour le script)
`https://www.youtube.com/watch?v=c05g5qLrLS0&ab_channel=NicholasRenotte`
- ▶ Build your first Reinforcement learning agent in Keras [Tutorial]
`https://hub.packtpub.com/build-reinforcement-learning-agent-in-keras-tutorial/`

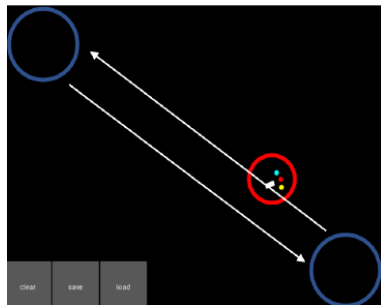
[2] Stable Baselines

- ▶ Stable Baselines for Reinforcement Learning Tutorial, N. Renote. (voir le lien github pour le script)
`https://www.youtube.com/watch?v=nRHjymV2PX8&t=816s&ab_channel=NicholasRenotte`
- ▶ Reinforcement Learning with Stable Baselines 3 - Introduction (P.1) `https://www.youtube.com/watch?v=XbWhJdQgi7E&t=1025s&ab_channel=sentdex`
- ▶ Saving and Loading Models - Stable Baselines 3 Tutorial (P.2) `https://www.youtube.com/watch?v=dLP-2Y6yu70&t=501s&ab_channel=sentdex`

- 1 Model-free reinforcement learning
- 2 Monte-Carlo
- 3 Temporal difference (TD) learning
- 4 Exploitation/Exploration
- 5 SARSA
- 6 Q-learning
- 7 Deep Q-Learning
- 8 Application à la navigation en conduite autonome

Application à la navigation en conduite autonome I

- Nous avons utilisé le package Kivy sous Python pour créer un environnement de véhicule autonome.
- La mission du véhicule est de faire des allers-retours du coin supérieur gauche au coin inférieur droit de la carte sans aucune collision, en choisissant à chaque état l'action qui lui permet de maximiser sa récompense.



Application à la navigation en conduite autonome II

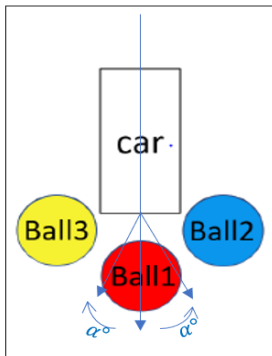
- L'utilisateur peut dessiner du "sable" en utilisant la souris.
- Une collision a lieu quand la voiture touche le sable.
- Après chaque voyage qu'elle effectue, le nombre de pas que la voiture a effectué pour arriver à sa destination est compté : il faudrait qu'il soit inférieur à celui du voyage précédent.



Application à la navigation en conduite autonome III

Etats

- *Capteur1* (Rouge)
- *Capteur2* (Bleu)
- *Capteur3* (Jaune)
- *Orientation*
- – *Orientation*



- Chaque capteur détecte le nombre de pixels de sable dans un carré de surface 400 pixels (20*20) dont le centre coïncide avec celui du capteur.
- Les variables *orientation* et –*orientation* mesurent en degrés l'orientation de la voiture.

Application à la navigation en conduite autonome IV

Actions

- Tourner de 20° dans le sens des aiguilles d'une montre.
- Tourner de 20° dans l'autre sens.
- Ne pas tourner.

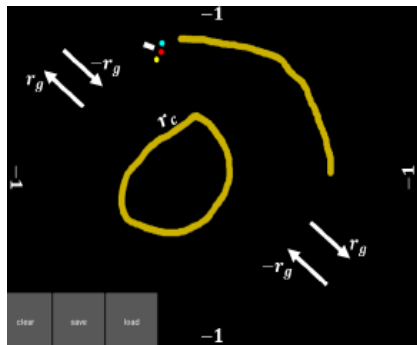
Récompenses

● Récompenses instantanées

- Se rapprocher de l'objectif : r_g
- S'éloigner de l'objectif : $-r_g$
- Toucher le sable : r_c
- Trop se rapprocher des bords de la carte : -1

● Récompense d'optimisation du chemin à l'arrivée

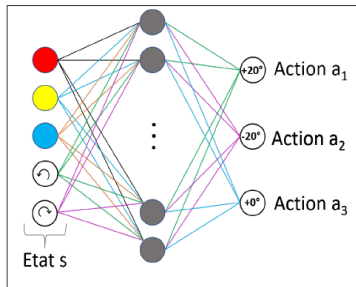
$$r = [\text{Nb pas}]_{\text{act}} - [\text{Nb pas}]_{\text{préc}}$$



Application à la navigation en conduite autonome V

DQN

- **Une couche d'entrée (input layer)** : 5 nœuds dont chacun correspond à une variable d'état (*Capteur1*, *Capteur2*, *Capteur3*, *Orientation* et $-Orientation$)
- **Une couche cachée (hidden layer)** : 30 nœuds.
- **Une couche de sortie (output layer)** : 3 nœuds dont chacun correspond à une action.



Bibliographie

- [1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Daan Antonoglou, Ioannis Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.
- [2] Aurélien Géron and Hervé Soutard. Deep learning avec TensorFlow : mise en oeuvre et cas concrets. O'Reilly Media, 2017.
- [3] Richard Sutton and Andrew Barto. Reinforcement learning : an introduction. MIT Press, 2018.
- [4] Thibault Neveu. aihub, 2018.
- [5] Jerry Qu. I created a virtual self driving car with deep Q-networks, 2018.
- [6] Le code source et une description détaillée de l'application sur :

GitHub :

<https://github.com/abdouboutiti/Voiture-Autonome-Virtuelle-VAV-App-Deep-Q-Learning>