REINFORCEMENT LEARNING PROJECT

Cas 1 : carte à reconstituer

Cas 2 : carte connue

Cas 3 : sélection des publicités

**Concept de bases :**

**Agent et son environnement :**

L’agent agit sur son environnement de plusieurs façons, il peut soit avoir connaissance de son environnement a l’avance soit le découvrir en ce basant sur son historique.

L’agent effectue ses choix suivant certain principe :

* Déterministe
* Stochastique ou distributionnelle
* Choix aléatoire e-greedy
* Choix via Softmax Boltzmann

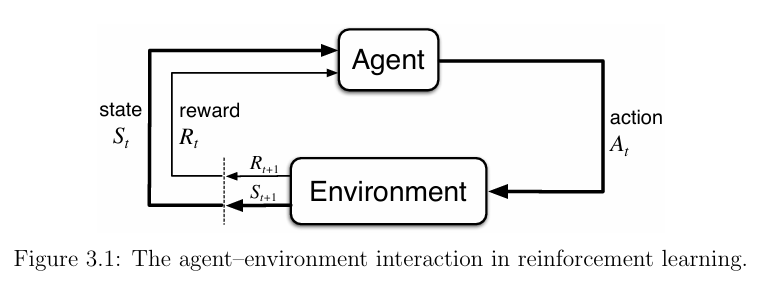
[

- Direct (argmax)

- Aléatoire (tirage uniforme ou pondéré)

- Mixte (ε-greedy, softmax)

]



**Equation de Bellman :** La valeur d’une action est égale à sa récompense immédiate plus la meilleure valeur qu’on peut espérer ensuite.

**Policy :** définit la manière dont un agent choisis ses actions en fonction de l’état courants elle peut être déterministe ou stochastique. Elle définit le cerveau de l’agent, son comportement (la manière d’agir dans chaque situation), son style d’exploration, et ça capacité d’adaptation (aléatoire, agressive…), (évolue au cour du temps, constant…).

Pour l’évaluer il faut établir une fonction de valeurs. Elle va venir évaluer la qualité globale du comportement d’un agent.

* Fonction d’état : Elle va mesurer **la valeur attendu** d’un état ‘S’, en suivant sa Policy pi a cet partir de cet état
* Fonction d’action : Elle mesure la valeur attendue de **prendre l’action a** dans l’état s, puis de suivre la Policy.

La fonction de valeur permet de :

* **Comparer plusieurs policies** : celle qui donne les plus hauts V^\pi(s) est meilleure.
* **Identifier les états rentables** : où l’agent a le plus à gagner.
* **Guider l’amélioration** : en ajustant la Policy pour augmenter V^\pi(s).

**Outils :**

Gestionnaire de Map

**Agent**

Gestionnaire de reward

Gestionnaire de policy

algorithm used

n-armed bandit (stationary and non-stationary)

Finite Markov decision process

###

Déplacement

verifications for end or not

observation

choose policy

agent state

**Q-learning : établis pour les cas statiques**

1. **Stockage implicite dans la Q-table**

* La Q-table **n’enregistre pas les trajectoires**, mais elle **encode la qualité des actions** dans chaque état.
* Quand une action mène à une récompense élevée, sa Q-valeur augmente.
* Quand une action mène à une impasse ou une pénalité, sa Q-valeur diminue.

Résultat : **les bonnes décisions sont renforcées**, les mauvaises sont oubliées.

2. **Apprentissage par répétition**

* L’agent explore l’environnement sur plusieurs épisodes.
* À chaque interaction, il met à jour ses Q-valeurs.
* Les valeurs se stabilisent progressivement c’est la **convergence**.

Une fois les Q-valeurs stables, l’agent **n’a plus besoin d’explorer** : il suit les meilleures actions à chaque étape.

##map

generate map

clean map

Check map quality

Refresh map

# type de choix (sélection concrète)

Déterministe

Stochastique ou distributionnelle

Choix aléatoire e-greedy

Choix via Softmax Boltzmann

[

- Direct (argmax)

- Aléatoire (tirage uniforme ou pondéré)

- Mixte (ε-greedy, softmax)

]

#type de decision (evaluation)

Q-learning - Table Q mise à jour par Bellman - Grilles, environnements discrets

DQN - Réseau de neurones pour approximer Q - Environnements visuels ou complexes

Monte Carlo - Moyenne des récompenses observées -Sans modèle, exploration libre

TD(λ) - Estimation par bootstrapping - Environnements continus

**Concept :** un agent qui **définit sa personnalité à partir de ses choix** est une idée **philosophiquement puissante.**

Il va falloir mettre en place un arbres complets d’objectif précis qui vont permettre ensuit de reconstituer la personnalité de l’agent (étudier la formation de la personnalité humaine en détails).

**Exemple :**

Premièrement on choisit le prochain déplacement grâce à l’algorithme qui est affecter à max\_feature, ensuite on récupère la valeur a la position courante, et on calcule la nouvelle valeur q lie à l’état courant, et on l’affecte a q. l’algorithme lui en fonction de celui choisis va effectuer le choix du prochain déplacement et le transmettre à la fonction de calcul de Q-learning.

ON va partir de (0, 0) et suivre **position par position** comment la Q-table se remplit à travers un épisode. On va simuler un agent qui se déplace dans une grille 3×3 pour atteindre un but en (2, 2) avec une récompense de +1. Toutes les autres cases donnent 0.

**⚙️ Paramètres**

* Grille : 3×3
* Actions: ["up", "down", "left", "right"] → indexées 0 à 3
* Alpha = 0.5, Gamma = 0.9
* Q-table initialisée à zéro

**🎯 Épisode simulé : chemin (0, 0) → (0, 1) → (0, 2) → (1, 2) → (2, 2)**

**🔹 Étape 1 : (0, 0) → action "right" (index 3) → (0, 1)**

* reward = 0
* max\_future\_q = max(Q[(0, 1)]) = 0 (car jamais visité)
* current\_q = Q[(0, 0)][3] = 0
* Mise à jour :
* new\_q = 0.5 \* (0 + 0.9 \* 0) = 0
* Q[(0, 0)][3] = 0

**🔹 Étape 2 : (0, 1) → action "right" (index 3) → (0, 2)**

* reward = 0
* max\_future\_q = max(Q[(0, 2)]) = 0
* Mise à jour :
* Q[(0, 1)][3] = 0

**🔹 Étape 3 : (0, 2) → action "down" (index 1) → (1, 2)**

* reward = 0
* max\_future\_q = max(Q[(1, 2)]) = 0
* Mise à jour :
* Q[(0, 2)][1] = 0

**🔹 Étape 4 : (1, 2) → action "down" (index 1) → (2, 2) (but)**

* reward = 1
* max\_future\_q = max(Q[(2, 2)]) = 0
* Mise à jour :
* new\_q = 0.5 \* (1 + 0.9 \* 0) = 0.5
* Q[(1, 2)][1] = 0.5

**🔁 Propagation en arrière**

Maintenant que (1, 2) a une Q-valeur de 0.5, on peut rejouer les étapes précédentes pour propager cette information :

**🔹 Mise à jour de (0, 2) → action "down" (index 1)**

* reward = 0
* max\_future\_q = max(Q[(1, 2)]) = 0.5
* Mise à jour :
* new\_q = 0.5 \* (0 + 0.9 \* 0.5) = 0.225
* Q[(0, 2)][1] = 0.225

**🔹 Mise à jour de (0, 1) → action "right" (index 3)**

* max\_future\_q = max(Q[(0, 2)]) = 0.225
* Mise à jour :
* new\_q = 0.5 \* (0 + 0.9 \* 0.225) = 0.10125
* Q[(0, 1)][3] = 0.10125

**🔹 Mise à jour de (0, 0) → action "right" (index 3)**

* max\_future\_q = max(Q[(0, 1)]) = 0.10125
* Mise à jour :
* new\_q = 0.5 \* (0 + 0.9 \* 0.10125) = 0.0455625
* Q[(0, 0)][3] = 0.0455625

**📊 Q-table après un épisode**

| **Position** | **Q-values [up, down, left, right]** |
| --- | --- |
| (0, 0) | [0.0, 0.0, 0.0, 0.0456] |
| (0, 1) | [0.0, 0.0, 0.0, 0.1013] |
| (0, 2) | [0.0, 0.225, 0.0, 0.0] |
| (1, 2) | [0.0, 0.5, 0.0, 0.0] |
| (2, 2) | [0.0, 0.0, 0.0, 0.0] |