Q-Learning

Julien Desvergnes

Table des matières

1 Introduction

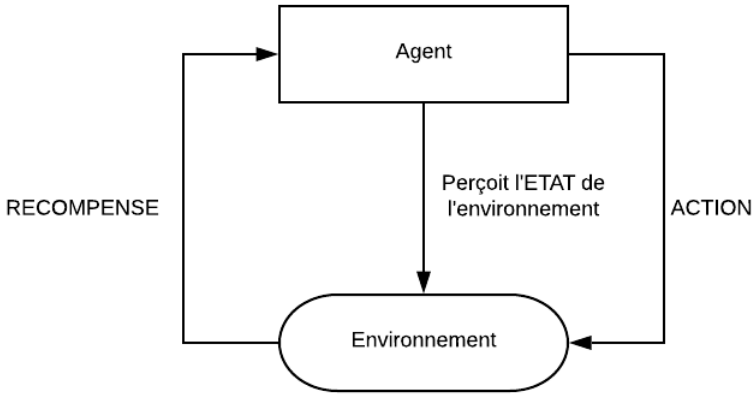

Figure 1 – Modèle du système

Dans ce système, on distingue plusieurs éléments, l'**agent** effectue une **action** sur l'**environnement** pour modifier son **état**. Suite à une action et une modification d'état, l'environnement envoie alors une **récompense** à l'agent. L'agent est alors capable de percevoir le nouvel état de l'environnement, et de renvoyer une action.
L'objectif de l'apprentissage profond par renforcement est d'apprendre par l'expérience une stratégie comportementale (politique) en fonction des échecs ou succès constatés.

2 Quelques définitions

2.1 Chaîne de Markov

Soit X, l'espace d'états de l'environnement. On appelle processus une séquence d'états. Soit la chaîne d'état $c = (x_0,\mathrm{x}_1, ...,\mathrm{x}_n)$.

On dit que la propriété de Markov est vérifiée si $\mathbb{P}(x_{n+1}|c) = \mathbb{P}(x_{n+1}|x_n)$

Autrement dit, le futur ne dépend que de l'état courant.

2.2 Processus de décision markovien

On définit un processus de décision markovien par le quadruplet (X, A, p, r) avec :

- X : l'espace d'états,
- A : l'espace d'actions,
- p : la fonction de probabilité de transition. Pour une transition d'un état x_1 vers un état x_2 sur l'action a, elle s'écrit $p(x_2|x_1, a)$.
- r : la fonction de récompense. Pour une transition d'un état x_1 vers un état x_2 sur l'action a, elle peut s'écrit $r(x_1, a, x_2)$. La fonction récompense étant issue du modèle, elle peut ne pas dépendre de l'action ou de l'état de départ.

Dans toute la suite on suppose que r est telle que définie ci-dessus

2.3 Règle de décision et politique

On appelle **règle de décision**, une fonction π_t (où t représente l'instant), qui détermine une loi d'action en chaque état de X. On a alors $\pi_t(x) =$ action choisie en x.

On appelle **politique**, la séquence de règles de décision à appliquer au cours du temps. On note $\pi = (\pi_0, \pi_1, \pi_2, ...)$. Dans le cas où on veut une règle de décision constante, on parle de politique markovienne : $\pi = (\pi, \pi, \pi, ...)$.

Propriété : Soit π une politique markovienne alors le processus $(x_t)_{t\geq 0}$ qui suit la politique π est une chaîne de Markov.

3 V fonction, V_π fonction et V-valeur

3.1 La V fonction

La fonction V est une fonction qui attribue à chaque état de X, ce que l'agent peut espérer de mieux **en moyenne** s'il part de cet état.

La fonction valeur doit tenir compte de la récompense **moyenne** immédiate à partir de l'état courant et de la **valeur moyenne** de l'état suivant **si** l'action optimale est choisie.

On peut donc écrire :
$\forall x \in X, V(x) = max_{a\in A}(\sum_{x'\in X}[r(x,a,x') + V(x^{'})]p(x,a,x^{'}))$
$\Leftrightarrow \forall x \in X, V(x) = max_{a\in A}(\sum_{x'\in X} r(x,a,x')p(x,a,x^{'}) + \sum_{x'\in X} V(x^{'})p(x,a,x^{'}))$
$\Leftrightarrow \forall x \in X, V(x) = max_{a\in A}(\bar{r}(x,a) + \sum_{x'\in X} V(x^{'})p(x,a,x^{'}))$
$\Leftrightarrow \forall x \in X, V(x) = max_{a\in A}(\bar{r}(x,a) + E[V(x^{'})|x,a]))$

Supposons que V est définie pour tous les états de X. Je peux donc à chaque instant choisir la meilleure action ! Il s'agit de $a = argmax_a(V(x))$, l'action qui maximise la valeur de la V fonction.

Remarque : en pratique on ne calcule jamais la V fonction (sauf dans des cas très simples i.e. diagramme d'état non cyclique de petite dimension). En effet, la présence de cycle rend les équations des V-valeurs non linéaires et une trop grande quantité d'états rendrait les calculs trop longs.

3.2 Une V fonction associée à une politique : V_π fonction

Soit la politique markovienne π, on peut définir V_π telle que :

$V_\pi(x) = \sum_{x'\in X}[r(x,\pi(x),x') + \gamma V_\pi(x^{'})]p(x,\pi(x),x^{'}))$
$\Leftrightarrow \mathrm{V}_\pi(x) = \bar{r}(x,\pi(x)) + \gamma E[V_\pi(x^{'})|x,\pi(x)))$
$\Leftrightarrow \mathrm{V}_\pi(x) = \bar{r}(x,\pi(x)) + \gamma \sum_{x'\in X} V_\pi(x^{'})p(x,\pi(x),x^{'})$

On constate alors deux choses :

— on a inséré un coefficient γ : sa valeur permet de définir l'importance que l'on donne au futur. Quand $\gamma = 0$ nous sommes face à un agent "pessimiste" qui ne cherche qu'à optimiser son gain immédiat. À l'opposé si $\gamma \to 1$, l'agent est "optimiste" puisqu'il tient de plus en plus sérieusement compte du futur lointain.
— L'équation de V_π vérifie une relation de récurrence appelée équation de Bellman.

Remarque : on peut également définir une V_π fonction sur une politique quelconque $\pi = (\pi_1, \pi_2, ...)$. On note alors $V_\pi(x) = E[\sum_{n=0}^{\infty} \gamma^t r(x_t, \pi_t(x_t)|x_0 = x, \pi]$ avec $r(x_t, \pi_t(x_t)$ la récompense obtenue depuis x_t en appliquant l'action $\pi_t(x_t)$

3.3 V^* : la fonction valeur optimale

Si on note pour $x \in X$ fixé, $V^*(x) = max_\pi(V_\pi(x))$ alors on a :

$V^*(x) = max_\pi[E[\sum_{n=0}^{\infty} \gamma^t r(x_t, \pi_t(x_t)|x_0 = x, \pi]]$
$\Leftrightarrow V^*(x) = max_{(a,\pi\prime)}[\bar{r}(x,a) + \gamma \sum_{x' \in X} V_{\pi\prime}(x^{'})p(x,a,x^{'})]$
$\Leftrightarrow V^*(x) = max_a[\bar{r}(x,a) + \gamma \sum_{x' \in X} max_{\pi'}(V_{\pi\prime}(x^{'}))p(x,a,x^{'})]$
$\Leftrightarrow V^*(x) = max_a[\bar{r}(x,a) + \gamma \sum_{x' \in X} V^*(x^{'})p(x,a,x^{'})]$

On constate que V^* vérifie l'équation de programmation dynamique !

3.4 Équation de Bellman : des propriétés intéressantes

Si on note OB l'opérateur de programmation dynamique :

$OB(f(x)) = max_{a \in A}[\bar{r}(x,a) + \gamma \sum_{x' \in X} f(x^{'})p(x,a,x^{'})])$

On a :
— V^* est l'unique point fixe de OB,
— Toute politique $\pi^*(x) = argmax_a[\bar{r}(x,a) + \gamma \sum_{x' \in X} f(x^{'})p(x,a,x^{'})]$ est **optimale** et **markovienne**,
— $\forall V_0 \in R^n, \forall \pi$ markovienne, on a $lim_{k \to \infty}(OB)^k V_0 = V^*$
Ces résultats donnent une méthode itérative (algorithme d'itération de la valeur).

3.5 Mise en place de l'algorithme d'itération de la valeur

Algorithm 1 Itération de la Valeur

Result: V^*
$V_0 \in R^n$ initialisé au hasard
$V = V_0$
while *critère d'arrêt non respecté* **do**
  $x \in X$ choisis au hasard
  $V(x) = OB(V(x))$
end

Cet algorithme couplé à celui d'itération des politiques permet de déterminer la politique optimale.

4 Q^π fonction et Q-valeur

4.1 Pourquoi des Q fonctions ?

Les algorithmes précédant ont un défaut majeur, ils nécessitent la connaissance des probabilités de transitions ! Or, dans beaucoup de systèmes, ces probabilités sont inconnues. Il faut donc trouver un moyen de passer outre le manque de connaissances. De plus la V fonction ne permet pas de stocker à la fois les états et les **actions.**

4.2 Définition de la Q_π fonction

On définie la Q_π comme suit :

$\forall \pi, \forall x \in X, \forall a \in A, Q_\pi(x,a) = \bar{r}(x,a) + \gamma \sum_{x' \in X} V_\pi(x')p(x,a,x')$

On définit également :

$\forall x \in X, \forall a \in A, Q^*(x,a) = max_\pi Q_\pi(x,a)$

4.3 Lien entre Q et V

On remarque que :
— $V_\pi(x) = Q_\pi(x, \pi(x))$,
— $Q^*(x,a) = \bar{r}(x,a) + \gamma \sum_{x' \in X} V^*(x')p(x,a,x')$
— $V^*(x) = Q^*(x, \pi^*(x)) = max_{a \in A} Q^*(x,a)$
— $\pi^*(x) = argmax_a Q^*(x,a)$

4.4 Equation de Bellman et programmation dynamique

Il se trouve que Q_π vérifie l'équation de Bellman et Q^* vérifie l'équation de la programmation dynamique !

On a :

$Q_\pi(x,a) = \bar{r}(x,a) + \gamma \sum_{x' \in X} Q_\pi(x', \pi(x'))p(x,a,x')$

$Q^*(x,a) = \bar{r}(x,a) + \gamma \sum_{x' \in X} max_{b \in A} Q^*(x',b)p(x,a,x')$

On peut donc comme précédemment définir un opérateur qui vérifie à nouveau les bonnes propriétés de convergence. On peut donc appliquer l'algorithme d'itération de la valeur de la même manière sur V et sur Q !

5 En pratique ? Algorithme de Q-learning

Algorithm 2 Itération de la Valeur

Result: V^*
$Q_0 \in R^{n*m}$ initialisé au hasard
$Q = Q_0$
while *critère d'arrêt non respecté* **do**
  $x \in X$ choisi au hasard
  $a \in A$ choisie au hasard
  $(x^{'}, r) = simulation(x, a)$
  $d = r + \gamma max_{b \in A} Q(x^{'}, b) - Q(x, a)$
  $Q(x, a) = Q(x, a) + \alpha d$
end

α est appelé le **taux d'apprentissage**. Il détermine à quel point on oublie vite les précédentes valeurs de $Q(x, a)$

6 Passage au réseau profond

Dans le cas où le nombre d'actions possibles et le nombre d'états sont assez petits, il est envisageable d'écrire l'algorithme du Q-learning et espérant une convergence pas trop lente.

En revanche, imaginons un jeu de déplacement sur une grille 10*10. Supposons qu'une case de la grille peut être **occupée par le joueur**, **occupée par un caillou** ou **libre**. Supposons également que les actions possibles sont **Haut**, **Bas**, **Gauche** et **Droite**.

Le nombre d'éléments de la Q-table est la produit du nombre d'état et du nombre d'actions, or

— nombre d'actions : 4,
— nombre d'états : 3^{100}, soit environ 10^{47} états

Il est inenvisageable de stocker ces valeurs dans un tableau. L'idée est la suivante, on ne va pas calculer explicitement les valeurs de la Q-table, on va utiliser un réseau de neurones.

Dans l'idée on envoie l'état du jeu en entrée du réseau et on récupère les Q-valeurs associées à chaque action possible.

6.1 Que devient l'algorithme ?

Algorithm 3 Apprentissage profond de la Q fonction

Result: Q^*
Initialisation des paramètres du réseau de neurones
Initialisation de la mémoire d'expérience D, de capacité C
for i = *1..NbSimulations* **do**
    Initialisation de l'état de départ de la partie
    while *simulationContinue* **do**
        a = choisir une action au hasard
        simuler l'action **a** et observer la récompense **r** et l'état suivant $s^{'}$
        ajouter l'expérience $(s, a, r, s^{'})$ à D
        sélectionner un mini-batch d'expériences dans D
        calculer $recompense_j = rj$ si la partie est finie ou $rj+\gamma \max_a Q(s^{'}, a, \theta)$ si elle continue
        Mettre à jour les paramètres du réseau de neurones via une rétropropagation de gradient de $(recompense_j - Q(s, a, \theta))^2$
    end
end

Une fois le réseau entraîné, il suffit de faire passer à chaque tour de simulation l'état courant dans le réseau pour obtenir les poids pour chaque action et enfin effectuer l'action qui maximise les poids.