Extensions

Clément Rambour

Plan

Cours précédents

- Estimation de la valeur par échantillonnage
- Deux grande familles d'approches :
  - MC : approximation de la valeur par moyenne des retour
  - TD : Minimisation de l'erreur de Bellman
- Contrôle : estimation de la fonction de valeur état-action
- Valeurs stockées dans la table Q

Cours précédents : Approches basées valeurs

- Approximation de la fonction de valeur par une fonction de paramètre $\boldsymbol{\theta}$:

$$Q(s,a) \simeq f_{\boldsymbol{\theta}}(s,a)$$

- Optimisation de $f_{\boldsymbol{\theta}}$ par descente de gradient
- Loss Meas Square Error
  - MC : par rapport au retour
  - TD : par rapport à la TD-target $\rightarrow$ Q-learning
- Deep Q Networks : $f_{\boldsymbol{\theta}} \rightarrow$ réseau de neurones

Cours précédents : Policy gradient

Motivation :

- Plutôt que de passer par une fonction décrivant le potentiel de chaque état/action de l'agent, pourquoi ne pas apprendre directement la politique ?
- Politique donnée par une distribution paramétrée par $\boldsymbol{\theta}$: $\pi_{\boldsymbol{\theta}}(s, a)$
- Recherche de la politique π_θ paramétrée par θ.
- Optimisation d'une fonction objectif : $J(\theta)$
- **REINFORCE** $J(\theta) = v_{\pi_\theta}(s)$:

$$\theta = \underset{\theta}{\operatorname{argmax}}\ v_{\pi_\theta}(s)$$

- **Actor Critique** Estimation de la valeur (Critique) et optimisation de la politique (Acteur)

optimisation de la valeur vs. optimisation de la politique

Avantages de l'optimisation de politique :

- Meilleure convergence
- Fonctionne en grande dimension ou si les actions sont continues
- Peut apprendre des politiques stochastiques

Désavantages :

- Converge vers un minimum local
- Grande variance

Aujourd'hui

1. Récapitulatif sur les approches Actor-Critique
2. Introduction aux approches basées modèles

Plan

Objectif

- Soit une trajectoire τ :

$$\tau = \{s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T\}$$

- On note $R(\tau) = \sum_{t=0}^{T} r_t$ la somme des récompense.
- $T(\tau) = G_0$ pour $\gamma = 1$
- La fonction objectif est donc :

$$V(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} r_t] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

item $P(\tau; \theta)$: probabilité de la trajectoire τ en suivant π_θ

- Notre **fonction objectif** est donc :

$$\underset{\theta}{\text{argmax}}\ V(s; \theta) = \underset{\theta}{\text{argmax}} \sum_{\tau} P(\tau; \theta) R(\tau)$$

Gradient de la politique

- Notre **fonction objectif** est donc :

$$\underset{\theta}{\operatorname{argmax}} \; V(s;\theta) = \underset{\theta}{\operatorname{argmax}} \; \sum_{\tau} P(\tau;\theta) R(\tau)$$

- On applique l'astuce vue précédemment pour calculer le gradient :

$$\nabla_\theta V(s;\theta) = \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)$$

- Espérance approchée par une moyenne sur un batch de m trajectoires :

$$\nabla_\theta V(s;\theta) \simeq \hat{g} = \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

- Pas besoin de la dynamique du modèle

Variance du gradient

- Problème : fonction objectif avec un potentiellement un **très grand nombre de paramètres**
- Conséquence : **grande variance** dans son estimation
- Comment réduire la variance dans l'estimation du gradient $\nabla_\theta V(s;\theta)$?

Variance du gradient

1. Structure temporelle → causalité :

$$\nabla_\theta V(\theta) = \mathbb{E}\big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t\big]$$

2. Bootstrapping → TD
   - Mise à jour de la politique par montée de gradient → Acteur
   - Fonction de valeur Q apprise via TD (par exemple) → Critique

$$\nabla_\theta V(\theta) \simeq \mathbb{E}\big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) Q(s_t, a_t)\big]$$

3. Calibration de la somme des récomenses → Utilisation de l'avantage $Q(s,a) - V(s)$:

$$\nabla_\theta V(\theta) = \mathbb{E}\big[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \underbrace{(Q(s_t, a_t) - V(s))}_{A(s_t,a_t)}\big]$$

4. L'erreur TD est un bon estimateur de l'avantage :

$$A(s_t, a_t) \simeq \delta_{\pi_\theta} = r + \gamma V_{\pi_\theta}(s') - V_{\pi_\theta}(s)$$

Off-line Actor Critic : schéma général

Offline Batch TD Actor-Critic
Jusqu'à convergence, répéter :

1. Échantillonner $\{s_t^{(i)}, a_t^{(i)}\}$ une trajectoire suivant $\pi_\theta(a|s)$
2. Calculer $y_t^{(i)} = r_{t+1}^{(i)} + \gamma V_{\pi_\theta}(s_{t+1}^{(i)}; w)$
3. $w \leftarrow w - \lambda \nabla_w \sum_i \mathcal{L}(V_{\pi_\theta}(s_t^{(i)}; w), y_t^{(i)})$
4. Évaluer $A_{\pi_\theta}(s_t^{(i)}, a_t^{(i)}) = r_{t+1}^{(i)} + \gamma V_{\pi_\theta}(s_{t+1}^{(i)}; w) - V_{\pi_\theta}(s_t^{(i)}; w)$
5. $\nabla_\theta J(\theta) \simeq \sum_i \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) A_{\pi_\theta}(s^{(i)}, a^{(i)})$
6. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Online Actor Critic : schéma général

Online TD Actor-Critic Jusqu'à convergence, répéter :

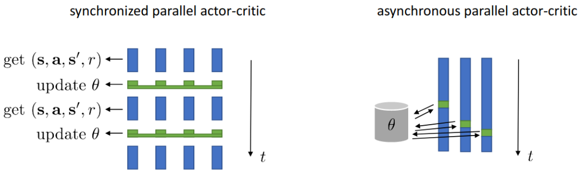
1. Prendre l'action $a_t \sim \pi_\theta(a_t|s_t)$ et observer la transition (s_t, a_t, s_{t+1})
2. Calculer $y_t = r_{t+1} + \gamma V_{\pi_\theta}(s_{t+1}; w)$
3. $w \leftarrow w - \lambda \nabla_w \mathcal{L}(V_{\pi_\theta}(s_t; w), y_t)$
4. Évaluer $A_{\pi_\theta}(s_t, a_t) = r_{t+1} + \gamma V_{\pi_\theta}(s_{t+1}; w) - V_{\pi_\theta}(s_t; w)$
5. $\nabla_\theta J(\theta) \simeq \nabla_\theta \log \pi_\theta(a_t|s_t) A_{\pi_\theta}(s, a)$
6. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Online Actor Critic : schéma général

Online AC

1. Prendre l'action $a \sim \pi_\theta(a|s)$
2. *Fit* $V_{\pi_\theta}(s_t^{(i)}; w)$ vers la TD target
3. Évaluer $A_{\pi_\theta}(s_t^{(i)}, a_t^{(i)}) = r_{t+1}^{(i)} + \gamma V_{\pi_\theta}(s_{t+1}^{(i)}; w) - V_{\pi_\theta}(s_t^{(i)}; w)$
4. $\nabla_\theta J(\theta) \simeq \sum_i \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) A_{\pi_\theta}(s^{(i)}, a^{(i)})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Problème : batch de taille 1 $\Rightarrow$ grande corrélation entre les transitions



Soft Actor Critic

Constat :

- Les algorithmes vus jusqu'à présents convergent souvent vers une politique déterministe
- Particulièrement vrai pour SARSA ou Q-learning
- Également avec policy gradient... donc aussi avec AC
- Lorsqu'un chemin fait monter la somme des récompenses celui-ci est trop vite privilégié
- Problème : politique déterministe pas forcément optimale

Intuition : Encourager le modèle à continuer à explorer

Principe

Comment favoriser l'exploration ?

- Value based algorithms : $\epsilon - greedy \rightarrow$ au final converge vers déterministe
- Policy gradient : variance du gradient $\rightarrow$ non souhaitable

Ajout d'un terme opposée à la confiance du réseau : **l'entropie**

$$\mathcal{H}(\pi(.|s)) = -\mathbb{E}[\log \pi(.|s)]$$

Soft RL

On cherche la politique qui maximise la somme des récompense et d'entropie maximale :

$$J(\theta) = \mathbb{E}\Big[\sum_{t=0}^{T-1} r_t\Big] + \lambda \mathcal{H}(\pi(.|s_t))$$

Soft RL

On cherche la politique qui maximise la somme des récompense et d'entropie maximale :

$$J(\theta) = \mathbb{E}\Big[\sum_{t=0}^{T-1} r_t\Big] + \lambda\mathcal{H}(\pi(.|s_t))$$

- Modification de la récurrence de Bellman par ajout d'un terme d'entropie Q :

$$Q(s,a) \leftarrow \mathbb{E}[R_{t+1} + Q(S_{t+1}, A_{t+1}|S_t = s, A_t = a) - \lambda \log \pi(A_{t+1}|S_{t+1})]$$

- Mise à jour de la politique par minimisation de la KL-divergence :

$$\pi \leftarrow \text{argmin } D_{KL}(\pi||\exp Q(s,.)/Z)$$

- cf. TP

Soft Actor Critic

- De nombreuses versions dans la littératures
- Dernière en date : pas d'utilisation de la fonction de valeur
- Peut être utilisée comme A2C vu précédement : avec Q **et** V, avec A...

Plan

Model based

Et si on **connaît** le modèle *ie.* les transitions entre états ?

- Connu dans de nombreux cas : jeux, systèmes physiques simples, simulations
- Utilisation d'un modèle : plus d'information sur les états suivants
- Possibilité de simuler l'environnement
- Combinaison avec les approches non basées modèles possible

$\rightarrow$ Connaître le modèle rend la vie plus simple

Écriture du problème basé modèle

Notation issues de communautés proches mais différentes : contrôle optimal, contrôle de trajectoire, planification

- c : Coût / rôle inverse de r la récompense
- f : fonction de transition

Formulation :

$$a_1, ..., a_T = \underset{a_1,\dots,a_T}{\operatorname{argmin}} \sum_{t=1}^{T} c(s_t, a_t) \text{ tel que } s_t = f(s_{t-1}, a_{t-1})$$

$$\Updownarrow$$

$$a_1, ..., a_T = \underset{a_1,\dots,a_T}{\operatorname{argmax}} \sum_{t=1}^{T} r(s_t, a_t) \text{ tel que } s_t = f(s_{t-1}, a_{t-1})$$

Random shooting methods

Optimisation de la fonction objectif J par tirage aléatoire

$$(a_1, ..., a_T) = \boldsymbol{A} = \underset{\boldsymbol{A}}{\operatorname{argmax}}\ J(\boldsymbol{A})$$

Deux étapes :

1. Tirer $\boldsymbol{A}_1, ..., \boldsymbol{A}_N \sim \mathcal{U}$
2. Choisir $\boldsymbol{A}_i = \underset{j}{\operatorname{argmax}}\ J(\boldsymbol{A}_j)$

Cross-Entropy Methods

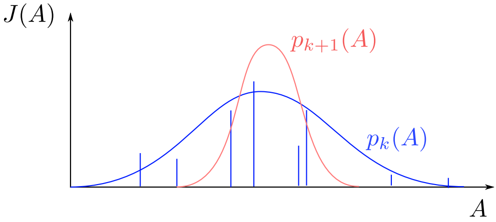
Restriction des échantillons aux actions les plus prometteuses

Initialisation $p(A) = \mathcal{U} ou \mathcal{N}$

repeat

1. Tirer $\boldsymbol{A}_1, ..., \boldsymbol{A}_N \sim p(A)$
2. Évaluer $J(A_1), ..., J(A_N)$
3. Prendre les M meilleures trajectoires
4. Raffiner $p(A)$ tel que $\boldsymbol{A}_1, ..., \boldsymbol{A}_M \sim p(A)$

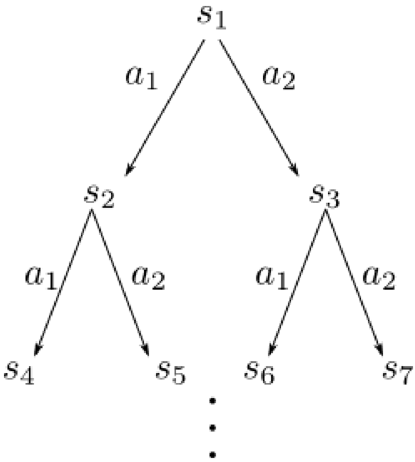
until **convergence**;



Monte Carlo Tree Search

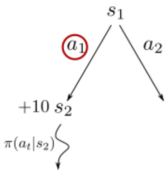
Dans le cas d'un environnement discret, on peut modéliser l'arbre de décision

- L'arbre est souvent trop grand pour être parcouru en entier...
- ... en particulier si l'environnement est stochastique
- On peut approcher la valeur des états par échantillonnage



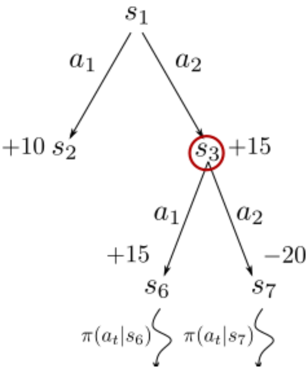
Monte Carlo Tree Search

Comment parcourir l'arbre ?
On préfère les noeuds non visités puis ceux de plus grande valeur



Monte Carlo Tree Search

Comment parcourir l'arbre ?
On préfère les noeuds non visités puis ceux de plus grande valeur



Monte Carlo Tree Search

generic MCTS sketch

1. find a leaf s_l using TreePolicy(s_1)
2. evaluate the leaf using DefaultPolicy(s_l)
3. update all values in tree between s_1 and s_l

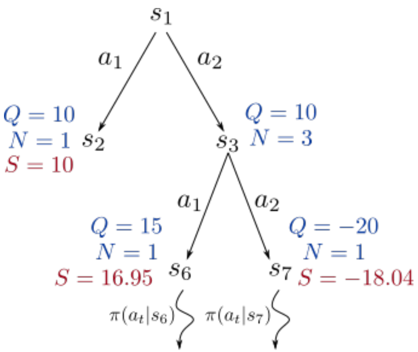
take best action from s_1

UCT TreePolicy(s_t)

if s_t not fully expanded, choose new a_t

else choose child with best Score(s_{t+1})

$$\text{Score}(s_t) = \frac{Q(s_t)}{N(s_t)} + 2C\sqrt{\frac{2\ln N(s_{t-1})}{N(s_t)}}$$



Optimisation de trajectoire

On cherche à résoudre :

$$a_1, ..., a_T = \underset{a_1,...,a_T}{\text{argmin}} \sum_{t=1}^{T} c(s_t, a_t) \text{ tel que } s_t = f(s_{t-1}, a_{t-1})$$

Cas linéaire :

- $f(s_t, a_t) = \boldsymbol{F}_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + \boldsymbol{f}_t$
- $c(s_t, a_t) = \frac{1}{2} \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T \boldsymbol{C}_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T \boldsymbol{c}_t$

Optimisation de trajectoire

Linear Quadratic Regulator

Backward Recursion

$(\boldsymbol{V}_T, \boldsymbol{v}_T) = (\boldsymbol{0}, \boldsymbol{0})$

for $t = T$ **to** 1 **do**

1. $(\boldsymbol{Q}_t, \boldsymbol{q}_t) = g(\boldsymbol{C}_t, \boldsymbol{F}_t, \boldsymbol{V}_t, \boldsymbol{c}_t, \boldsymbol{f}_t, \boldsymbol{v}_t)$
2. $J(\boldsymbol{s}_t, \boldsymbol{a}_t) = \frac{1}{2} \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T \boldsymbol{Q}_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T \boldsymbol{q}_t$
3. $a_t^* = \underset{a_t}{\text{argmax}}\ J(\boldsymbol{s}_t, \boldsymbol{a}_t) = \boldsymbol{K}_t \boldsymbol{s}_t + \boldsymbol{k}_t$
4. $(\boldsymbol{V}_t, \boldsymbol{v}_t) = h(\boldsymbol{C}_t, \boldsymbol{F}_t, \boldsymbol{V}_t, \boldsymbol{c}_t, \boldsymbol{f}_t, \boldsymbol{v}_t)$

end

Avec g et h deux fonctions linéaires

Forward Recursion

$(\boldsymbol{V}_T, \boldsymbol{v}_T) = (\boldsymbol{0}, \boldsymbol{0})$

for $t = 1$ **to** T **do**

1. $\boldsymbol{a}_t = \boldsymbol{K}_t \boldsymbol{s}_t + \boldsymbol{k}_t$
2. $\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t)$

end

Optimisation de trajectoire

- Dans le cas de transitions stochastiques, DQR est toujours optimal si la distribution est symétrique *eg* gaussienne.
- Cas non linéaire $\rightarrow$ On se ramène au cas linéaire en partant d'une séquence de référence $(s'_1, a'_1), ...(s'_T, a'_T)$:

$$\boldsymbol{C}_t = \nabla^2_{s_t, a_t} c(s'_t, a'_t)$$
$$\boldsymbol{c}_t = \nabla_{s_t, a_t} c(s'_t, a'_t)$$
$$\boldsymbol{F}_t = \nabla_{s_t, a_t} f(s'_t, a'_t)$$

- Dans le cas non-linéaire, on fait généralement quelques itérations seulement en suivant la récursion vers l'avant avant de replanifier.

Plan

Bibliographie

- Reinforcement Learning : an introduction, second edition, Richard S. Sutton and Adrew G. Barto
- Reinforcement Learning courses, David Silver, DeepMind (https ://www.davidsilver.uk/)
- A3C (Mnih et al. ICML 2016)
- T. P. Lillicrap et al., Continuous control with deep reinforcement learning
- Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, Colton. (2012). A Survey of Monte Carlo Tree Search Methods