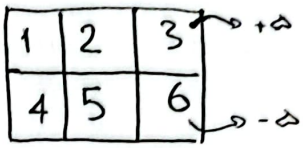
AI - A6 810101465

Markov Decision Process

Q1. $S = 1(1,1), 2(1,2), 3(1,3), 4(2,1), 5(2,2), 6(2,3)$



$$V_{t+1}(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_t(s')\right]$$

$V_0(s) = 0$ for each $s \in S$ if s in not in Final states

از آنجایی که reward رسیدن به هر خانه‌ای جز 3 و 6 صفر است، اگر همچنین value خانه‌های همسایه صفر باشند پس value خانه کنونی نیز صفر می‌شود. برای مثال می‌دانیم V_1 و V_2 و V_4 در زمان صفر، صفر هستند. می‌توان گفت در حالتی که در $s=1$ باشیم و به هر جهتی برویم، در یکی از خانه‌های 1 یا 2 یا 4 خواهیم رفت که value هم آن‌ها و reward نیز صفر است. پس $V_{t=1}(s=1)$ هم صفر خواهد بود. با این استدلال یک به یک خانه ها را بررسی می‌کنیم:

$V_1(1)$ و $V_1(4)$ حالت گفته شده را دارند.

$V_1(2) = \max\{0.8(0+0.9\times 0)+0.1(0+0.9\times 0),\ 0.8(0+0.9\times 0)+0.1(0+0.9\times 0)\} = V_{1,4}$ (right, up & down)

$Q_1(2)$: $V_2(4)$ در جهت left صفر است اما در سه جهت دیگر نیست.

$V_1(5) = -V_{1,4}$ ← $V_1(5)$: شرایط کاملاً مشابه $V_1(2)$ است با این تفاوت که به جای 5+ در 5- شاید بیافتد

$V_1(3)$ و $V_1(6)$ ثابت هستند (مانند value در $t=0$)

$V_2(1)$: فقط آن $Q_2(1)$ ای صفر است که agent به سمت left برود. max حالت وقتی رخ می‌دهد که به right برود (با احتمال بیشتری در خانه 2 می‌افتد تا وقتی که به up یا down برود)

$V_2(1) = 0.8(0+0.9\times V_{1,4}) = 5,4V2$

$V_2(4) = -5,4V2$ $V_2(4)$: شرایط کاملا مشابه $V_2(1)$ اما قرینه آن. در نتیجه

$V_2(2)$: دنبال max هستیم. واضحاً حرکت به پایین (مقدار منفی) و چپ (مقدار صفر) ما را به هدف نمی‌رساند. دو حرکت up و right را بررسی می‌کنیم

$V_2(2) = \max\{\underbrace{0.8(0+0.9\times V_{1,4})+0.1(5+0.9\times 5)}_{4,422},\ \underbrace{0.8(5+0.9\times 5)+0.1(0.9\times V_{1,4})-0.1(0.9\times V_{1,4})}_{V_{1,4}}\} = V_{1,4}$ و

$V_2(5) = -V_{1,4}$ ← $V_2(5)$: شرایط دقیقاً مثبا $V_2(2)$ اما قرینه آن است

$V_2(3)$ و $V_2(6)$ ثابت هستند (مانند value در $t=0$)

پس جدول نهایی به این صورت است

(1,1)	(1,2)	(1,3)
(2,1)	(2,2)	(2,3)

S	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)
V_0	0	0	5	0	0	-5
V_1	0	+V,14	5	0	-V,14	-5
V_2	+5,4V2	+V,14	5	-5,4V2	-V,14	-5

AE-A6 810101465

Markov Decision Process

۵٫۴۷۲	۷٫۶	۵
-۵٫۴۷۲	-۷٫۶	-۵

Q2.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_T^*(s')\right] \longrightarrow T=2$$

π^*_1: با حرکت به down مقدار منفی دریافت خواهیم کرد. همچنین حرکت به left مطمئنا نتیجه کمتری نسبت به right یا up خواهد داشت. دو حالت right و up را می‌سنجیم:

$$\pi^*(1) = \operatorname{argmax}\{\underbrace{(0.8+0.1)(0+0.9\times 5.472)+0.1(0+0.9\times 7.6)}_{5.116}, \underbrace{(0.1-0.1)(0.9\times 5.472)+0.8\times 0.9\times 7.6}_{5.472}\} = \text{right}\ (\rightarrow)$$

π^*_{24}: شرایط کاملا مشابه π^*_1 اما هزینه آن است. پس با حرکت به پایین یا راست مقدار ۵٫۱۱۶- و ۵٫۴۷۲- را خواهد گرفت. با حرکت به چپ هم مطمئنا مقداری منفی خواهد گرفت. اما حرکت به بالا منجر به Value مثبتی می‌شود

$$(\uparrow)\ \text{up} \Rightarrow 0.8\times 5.472\times 0.9 + 0.1\times 0.9(-5.472) + 0.1\times 0.9\times(-7.6) = 2.764 > 0$$

π^*_2: واضحاً حرکت به پایین حامل منفی دارد، سه جهت دیگر را بررسی می‌کنیم

up: $0.8\times 0.9\times 7.6 + 0.1\times 0.9\times 5.472 + 0.1\times 0.9\times 5$

right: $0.8\times 0.9\times 5 + 0$

left: $0.8\times 0.9\times 5.472 + 0$

$\Rightarrow$ argmax: up $(\uparrow)$

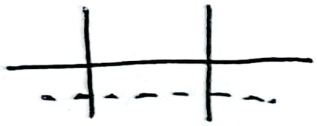
π^*_5: حرکت به left، right و down منجر به مقادیر منفی خواهند شد و تنها حرکت به up حامل مثبت دارد. پس با argmax گرفتن، $(\uparrow)$ up انتخاب می‌شود.

برای خانه‌های 3 و 6 نیاز به محاسبه π^* نیست. در آخر نتیجه بدین شکل می‌شود:

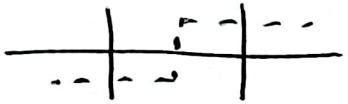
→	↑	+۵
↑	↑	-۵

Q3. روشی است که با شبیه‌سازی بازی روی state space تخمینی از transition function به دست می‌آوریم. هر چه تعداد simulation ها بیشتر باشد، نتیجه به دست آمده به نتیجه واقعی نزدیک‌تر خواهد بود اما هیچوقت نمی‌توان با اطمینان کامل گفت که به مقدار دقیق transition ها رسیدیم.

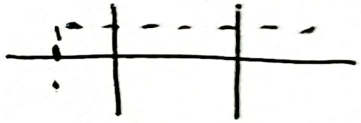
(1,1) → (1,2) → (1,3)



(1,1) → (1,2) → (2,2) → (2,3)



(1,1) → (2,1) → (2,2) → (2,3)



با مشاهده سه episode بالا، به تخمین مقادیر T می‌پردازیم

تخمین برای خانه (1,1)

$$((1,1), \text{right}, (1,2)) = \frac{2}{3} \qquad ((1,1), \text{right}, (2,1)) = \frac{1}{3}$$

تخمین برای خانه (2,2)

$$((2,2), \text{right}, (2,3)) = \frac{2}{2} = 1$$

از آنجایی که fixed policy داریم و در همه simulation ها به راست می‌رویم، مقدار تابع T را فقط به ازای این action می‌توان تخمین زد

AI-A6 810101465

Markov Decision Process

Q4. learning rate = 0.1 = α

Updating $V(s)$: TD-learning

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\left[\text{new sample} - V^{\pi}(s)\right] = (1-\alpha)V^{\pi}(s) + \alpha(\text{new sample})$$

فرض کنید تخمین ما تا الان $\bar{x}_n$ بوده و مشاهده جدید x_{n+1} است ←

$$\bar{x}_n = (1-\alpha)\bar{x}_{n-1} + \alpha x_n$$

$$\bar{x}_n = (1-\alpha)\left[(1-\alpha)\bar{x}_{n-2} + \alpha x_{n-1}\right] + \alpha x_n = (1-\alpha)^2\left[(1-\alpha)\bar{x}_{n-3} + \alpha x_{n-2}\right] + (1-\alpha)\alpha x_{n-1} + \alpha x_n$$

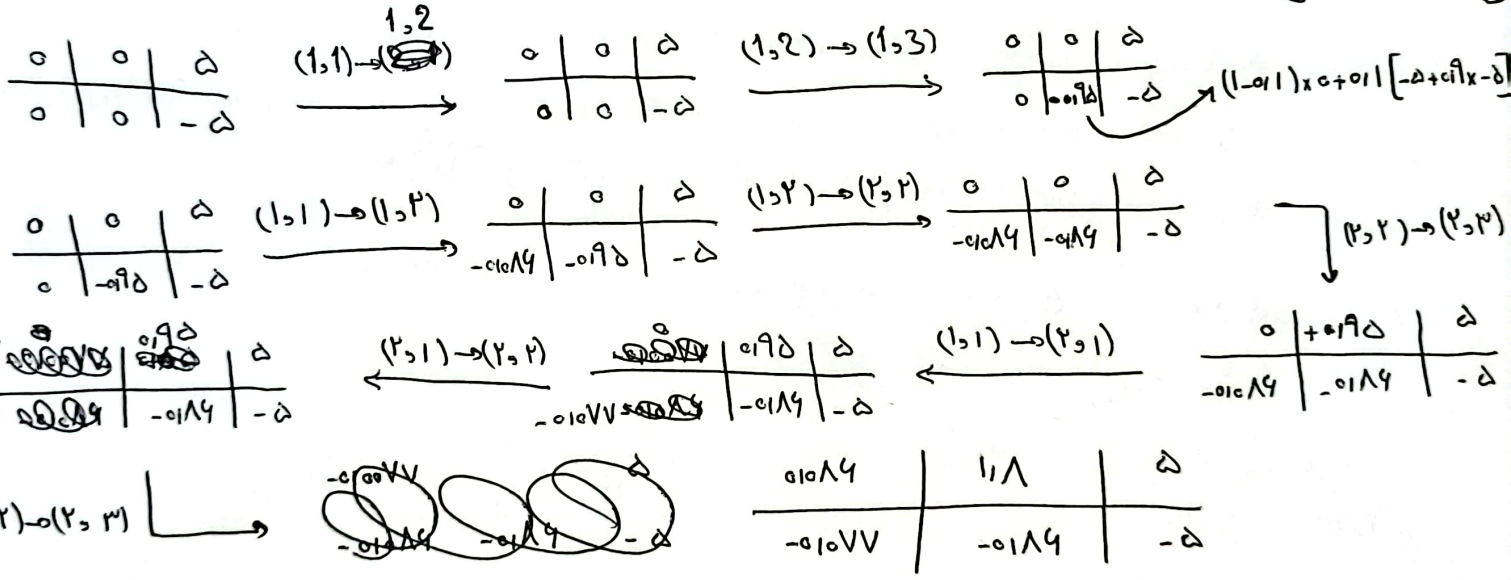
$$= (1-\alpha)^3\left[(1-\alpha)\bar{x}_{n-4} + \alpha x_{n-3}\right] + \left[(1-\alpha)^2 x_{n-2} + (1-\alpha)x_{n-1} + x_n\right]\alpha \rightarrow \cdots$$

$$\bar{x}_n = \left[\sum_{i=0}^{n-1} (1-\alpha)^i x_{n-i}\right] \times \alpha$$

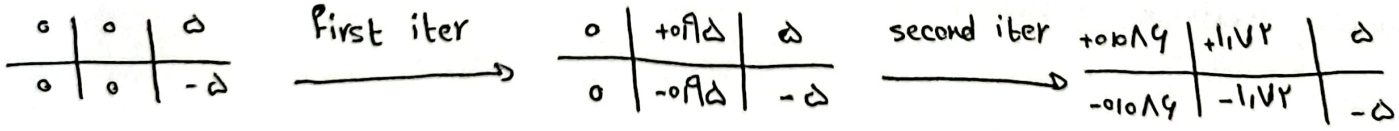
با گذر زمان می‌توان α را کم کرد زیرا تخمین دقیق‌تر شده است.

$V(s) \leftarrow (1-\alpha)V(s) + \alpha\left[R + \gamma V(s')\right]$ از همان نتایج Simulation ها در بخش قبل استفاده می‌کنیم (I)

جدول اولیه



(II) یکبار دیگر Value ها را تخمین می‌زنیم (با روش value iter)



$$V_{t+1}(s) = (1-\alpha)V_t(s) + \sum_{s'} \alpha\left[R + \gamma V_t(s')\right]$$

AI-AG 810101465

Deep Q-Network

Q1.

Deep Q-Network is a deep reinforcement learning algorithm that combines Q-learning algorithm with deep neural networks. DQN is a model-free algorithm. Q-learning is used for estimating value function, and DNN is used for estimating Q function. That's because directly calculating Q values is not efficient when working with large datasets.

1) init: initialize the NN with random weights.

2) action selection: using a greedy policy or other exploration strategies, choose an action.

3) perform action: execute the chosen action and recieve results (next state and reward).

4) store result: store the result tuple in the experience replay memory.

5) sample a batch: randomly take a sample of transition from memory.

6) update Q-Network's for each sampled transition calculate the Temporal Difference error. using the error & update weights of NN by applying GD algorithm (use a loss function)

7) repeat steps 2-6 until convergence or a desired performance is reached

Applications of DQN:

- game playing
- robot control
- autonomous driving
- resourse management in cloud computing
- trading strategies in finance