Косарев Евгений Александрович. 201 гр.

Reinforcement Learning

Введение:

Для начала, нужно ввести термины, которыми мы будем оперировать в ходе обсуждения темы обучения с подкреплением. В ней нам понадобятся такие понятие, как агент (agent), который взаимодействует с окружающей средой (environment) и предпринимает действия (actions). Согласно некоторым правилам, среда выдает награду на действие агента (reward). Таким образом, задача состоит в том, чтобы поставить в соответствие множество состояний среды действия агента.

С математической точки зрения, среда формулируется, как марковский процесс принятия решений с конечным множеством состояний. Запишем это как следующий кортеж $\big(S, A, P.(°,°), R.(°,°)\big)$, где

- S – множество состояний
- A – множество действий, где A_s – множество действий, доступных из состояние s
- $P_a(s, s`) = \mathbb{P}(s_{t+1} = s` | s_t = s, a_t = a)$ – то есть вероятность, что действие a приведет из состояния s в момент времени t в состояние s` в момент времени t+1
- $R_a(s, s`)$- вознаграждение при переходе от s к s` с вероятностью $P_a(s, s`)$

Таким образом, получая на каждом шаге t вознаграждение $r_t = R_a(s, s`)$, наш алгоритм должен выбрать такую стратегию, которая максимизирует величину $R = \sum_t r_t$, если существуют конечные состояние. Иначе вводится дисконтирующий множитель $\gamma \epsilon [0,1]$, что $R = \sum_t \gamma^t r_t$. С помощью него можно регулировать ориентирование модели на долгосрочный или краткосрочный выигрыш от стратегии.

Задача о многоруком бандите:

Пусть у нас есть автомат, который дергает за N ручек. В общем случае, мы не знаем распределение наград за действия, но пусть оно не меняется со временем. Возьмем выборочное математическое ожидание для оценки

каждой из случайных величин. Тогда реализуем жадный eps-алгоритм, стратегия обучения будет следующей:

1. Введем параметр $\varepsilon \in (0,1)$
2. Заведем массивы
   2.1. $N_a = 0,\ 1 \le a \le N$, где N_a-число выбора действий a
   2.2. $M_a = 0,\ 1 \le a \le N$, где M_a – текущая оценка математического ожидания награды от действия a
3. На каждом шаге t
   3.1. С помощью любого пакета сгенерируем значение случайной величины θ, равномерно распределенной на отрезке (0,1)
   3.2. Если $\theta \in (0, \varepsilon)$, то выберем действие a_t из набора действий A случайно.
   Иначе выбираем $a_t = argmax(M_a, 1 \le a \le N)$
   3.3. Выполняем a_t и получаем награду r_t
   3.4. Обновляем оценку мат. ожидания для действия r_t:

$$N_{a_t} += 1$$

$$Q_a += \frac{1}{N_{a_t}}(r_t - Q_a)$$

Рассмотрим 10-рукий автомат. Выберем 10 чисел распределением $\mathrm{N}_{0,1}$ и мат ожиданием $\mathrm{E}_a,\ 1 \le a \le 10$. Тогда каждой ручке поставим в соответствие нормальное распределение с математическим ожиданием из E_a и дисперсией 1. Это будут 10 случайных величин, которые определяют награду.

Каждый эксперимент будет состоять из 1000 шагов. При этом мы заранее сформируем данные (т.е. для каждого шага запомним награду за каждую ручку), чтобы можно было повторять эксперименты, меняя стратегию. Проведем серию из 2000 экспериментов. Для каждого шага будем считать среднюю награду по всем 2000 экспериментам.

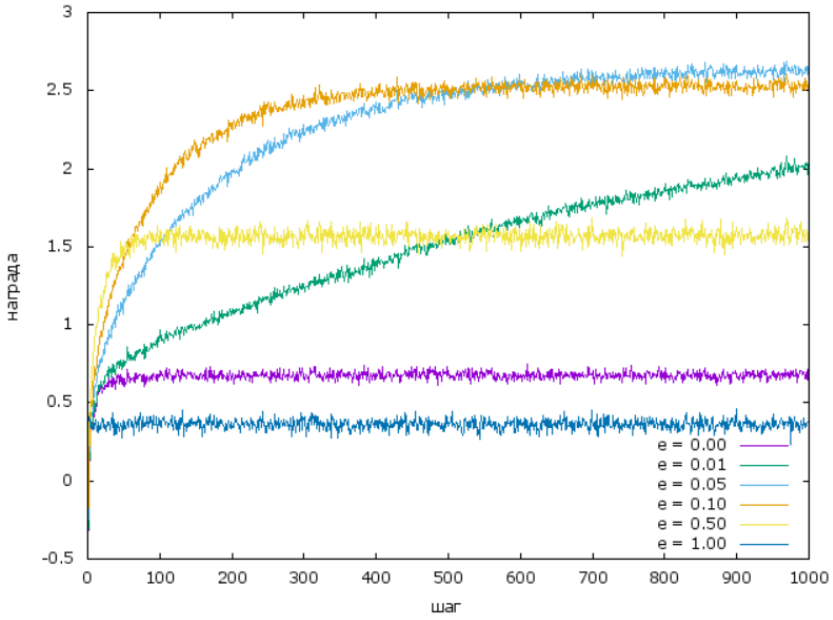
Допустим, в качестве средних для наград выбрали некоторые величины (сгенерированы случайно, как и писалось выше, из нормального распределения с математическим ожиданием нуль и единичной дисперсией):

Ручка	Ср. награда
1	0.089668

Ручка	Ср. награда
6	-0.666402

2	-0.757752
3	0.497168
4	0.811979
5	-0.367975

7	-0.757752
8	0.497168
9	0.811979
10	-0.367975



Таким образом, худшая стратегия выбрать $\varepsilon = 1$

Достаточно хорошо себя показывает выбор $\varepsilon = 0.05$

Ссылка на код работы:
https://github.com/EvgenijGod/Reinforcement_learning_mmp

Q-learning:

Для знакомства с алгоритмом, рассмотрим следующую задачу. Пусть на квадратном поле из клеток есть n шпионов, расставленных случайным образом, и агент. В моей реализации 0 – пустая клетка, 1 – агент, 2 – шпион. Так же каждый объект может двигаться в произвольном направлении на 1 клетку (и по диагонали), агент же может вообще не двигаться. Шпионы двигаются неким образом, при этом критерием остановки симуляции является то, что агент был пойман (те после очередного хода агент оказался в одной клетке с шпионом). Цель агента – как можно дольше бегать от шпионов, цель последних – поймать агента.

Для ознакомления со средой можно позволить всем ходить произвольно, кроме агента, он будет стоять неподвижным. Для поля 5x5 и 3-х шпионов

агент живет от 30 до 100 ходов в результате моих экспериментов. Добавим алгоритм q-learning.

Сначала рассмотрим алгоритм с математической точки зрения, а позже придадим этим рассуждениям «физический смысл».

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_{t+k} + \cdots$$

$$R_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} + \cdots$$

И чем выше $\gamma \in [0,1]$, тем более дальновидный наш агент. Введем две функции:

Функция ценности состояния s при стратегии π:

$$V^{\pi}(s) = E_{\pi}(R_t | s_t = s) = E_{\pi}(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s)$$

Функция ценности действия a в состоянии s при стратегии π:

$$Q^{\pi}(s,a) = E_{\pi}(R_t | s_t = s, a_t = a) = E_{\pi}(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a)$$

Пусть теперь о среде известно все, а также

$\mathcal{P}_{ss`}^{a} = p(s_{t+1} = s | s_t = s, a_t = a)$ – вероятность перехода от s к s` через a.

$\mathcal{R}_{ss`}^{a} = E(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s`)$ – ожидаемая награда

Тогда получим, что

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a) \\ &= E_{\pi}(r_{t+1} | s_t = s, a_t = a) + \gamma E_{\pi}(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a) \\ &= \sum_{s`} \mathcal{R}_{ss`}^{a} \mathcal{P}_{ss`}^{a} + \sum_{s`} E_{\pi}(\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s, a_t = a) \mathcal{P}_{ss`}^{a} \end{aligned}$$

Таким образом, мы получаем уравнение Беллмана для функции ценности:

$$Q^{\pi}(s,a) = \sum_{s`} \mathcal{P}_{ss`}^{a} \left(\mathcal{R}_{ss`}^{a} + \gamma Q^{\pi}(s`, a) \right)$$

Введем отношение порядка в стратегиях, те

$$\pi \leq \pi` \leftrightarrow \forall s \in S \; V^{\pi}(s) \leq V^{\pi`}(s)$$

Оптимальной стратегией π^* назовем ту, что

$$\forall \pi: \pi^* \not\leq \pi$$

Отсюда, с помощью уравнения Беллмана, можно доказать, что для финитного марковского процесса оптимальные V и Q единственны.

Распишем, как должны выглядеть оптимальные функции по Беллману (запишем уравнения оптимальности Беллмана):

$$\begin{cases} V(s,a) = max_a \sum_{s` \in S} \mathcal{P}_{ss`}^{a} \left(\mathcal{R}_{ss`}^{a} + \gamma V(s`) \right) \\ Q(s,a) = \sum_{s` \in S} \mathcal{P}_{ss`}^{a} \left(\mathcal{R}_{ss`}^{a} + \gamma \cdot max_a Q(s`,a) \right) \end{cases}$$

Для решения системы воспользуемся следующим утверждением:

Если π_1, π_2 такие, что $\forall s \in S$ выполнено, что $V^{\pi_1} \leq Q^{\pi_1}(s, \pi_2(s))$, тогда $\pi_1 \leq \pi_2$ и выполнено $\forall s \in S \; V^{\pi_1}(s) \leq V^{\pi_2}(s)$.

Таким образом, если при выборе действия a в состоянии s следовать стратегии π, то

$$Q^{\pi}(s,a) = \sum_{s`} \mathcal{P}_{ss`}^{a} \left(\mathcal{R}_{ss`}^{a} + \gamma V^{\pi}(s`) \right)$$

И стратегию можно обновлять следующим образом:

$$\pi(s) = argmax_a \sum_{s`} \mathcal{P}_{ss`}^{a} \left(\mathcal{R}_{ss`}^{a} + \gamma V^{\pi}(s`) \right)$$

Таким образом, можно комбинировать следующие шаги в произвольном порядке в силу сходимости метода к оптимальному

$$\begin{cases} \pi(s) = argmax_a \sum_{s`} \mathcal{P}_{ss`}^{a} \left(\mathcal{R}_{ss`}^{a} + \gamma V^{\pi}(s`) \right) \\ V(s) = \sum_{s` \in S} \mathcal{P}_{ss`}^{\pi(s)} \left(\mathcal{R}_{ss`}^{\pi(s)} + \gamma \cdot V(s`) \right) \end{cases}$$

Так как в реальности $\mathcal{P}_{ss`}^{a}$ и $\mathcal{R}_{ss`}^{a}$ неизвестны, то оценим их через следующее отношение:

$$V^{\pi}(s) = E_{\pi}(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s) = E_{\pi}(r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s) = \\ = E_{\pi}(r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s)$$

Аналогично, с помощью оценки экспоненциальным скользящим средним (F(t) – прогноз на момент времени t, S(t)- было ранее $F(t+1) = \alpha * S(t) + (1-\alpha) * F(t)$) получим, что

$$V(s_t) = V(s_t) + \alpha_t \cdot \left(r_{t+1} + \gamma \cdot V(s_{t+1}) - V(s_t)\right)$$

$$\begin{aligned} Q^{\pi}(s,a) &= E_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a\right) = \\ &= E_{\pi}(r_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | s_t = s, a_t = a) \approx \\ &\approx Q^{\pi}(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot max_a Q^{\pi}(s_{t+1}, a) - Q^{\pi}(s_t, a_t)\right) \end{aligned}$$

В итоге выведено следующее уравнение для итерационного процесса:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)$$

А теперь про физический смысл:

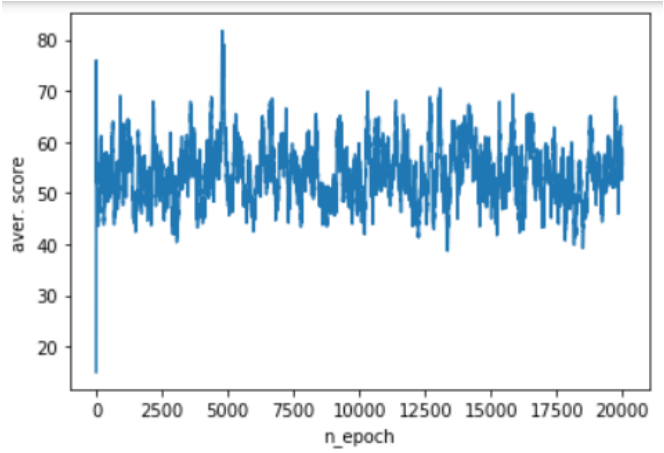
- Пусть $Q(s, a)$ – функция, отражающая ценность каждого возможного действия a агента для текущего состояния s, в котором сейчас находится симуляция.
- Пусть $\mathrm{max}_a Q(s_{t+1}, a)$ – число, отражающее максимальную ожидаемую награду на последующих ходах, а r_t-награда, получаемая агентом за этот ход.
- Как описывалось ранее, нужно ввести дисконтирующий множитель, отражающий отношение ценности выгоды ближайшей и будущей (чем он больше, тем агент больше задумывается о ценности будущих действий).
- Таким образом, можно сделать следующую оценку функции Q:
$$Q(s_t, a_t) \approx r_t + \gamma \cdot max_a Q(s_{t+1}, a)$$
- Ошибка предсказания будет равна:
$$\Delta q = r_t + \gamma \cdot max_a Q(s_{t+1}, a) \; - \; Q(s_t, a_t)$$
- Введем коэффициент α , характеризующий скорость обучения агента, те финальная формула итерационного расчета функции Q будет выглядеть так:
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)$$

В реализованной задаче рассмотрим несколько ситуаций. Будут приведены графики, иллюстрирующие пары {очередная игра, количество ходов прежде, чем быть пойманным}. Здесь и далее обсчитывается поле 5x5 и n=1 (число

шпионов) из-за ограничений по производительности ноутбука. Теперь агент может двигаться.

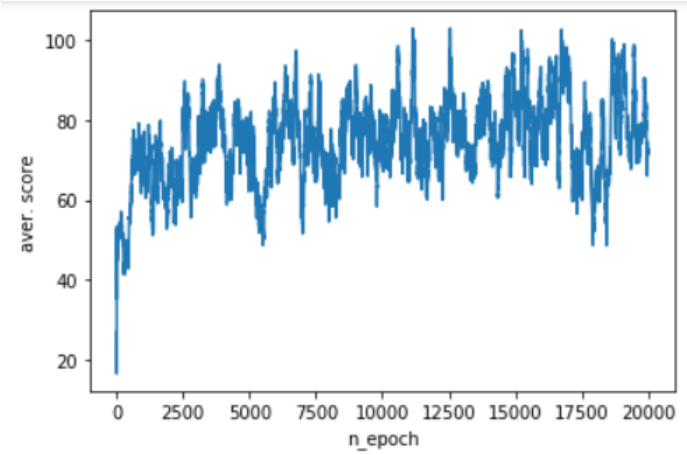
https://github.com/EvgenijGod/Reinforcement_learning_mmp – код работы

1. Пусть агенту не поступает информации об окружающей среде. Тогда получим, что в среднем агент живет 50 ходов. Можно считать, что его очки находятся в диапазоне [40, 65]



Время работы – 2 минуты.

2. Теперь добавим метрики евклидового расстояния до ближайшего из противников и до кромки поля.

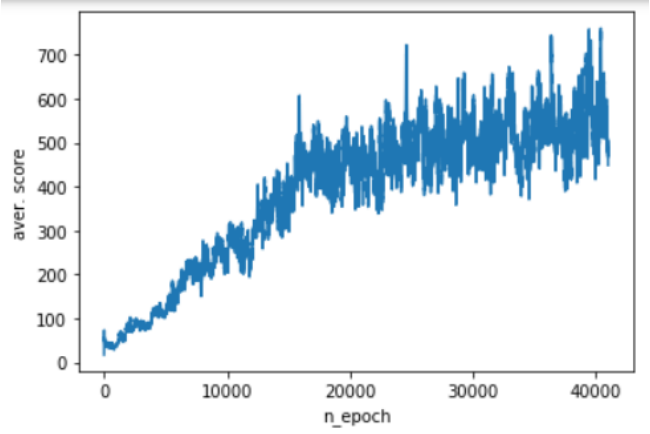


Тогда заметим улучшения в работе алгоритма. Модель обучилась лучше и получает в среднем 80 очков.

Можно считать, что очки агента находятся в диапазоне [55, 100].

Однако этой информации мало, так как агент не имеет информации о том, в какую сторону двигаться от шпиона. Время работы – 15 минут.

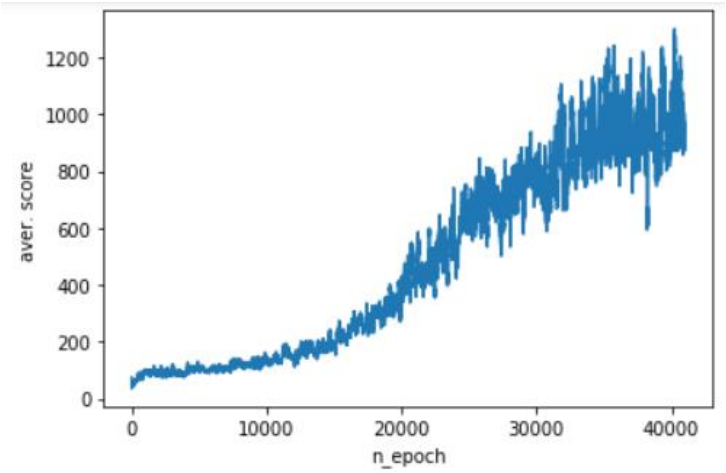
3. Добавим агенту понимание о том, с какой стороны к нему приближается ближайший шпион.



Выживаемость по очкам выросла в 6 раз. Агент теперь выживает в среднем 500 ходов и очки находятся в диапазоне [400, 650].

Время работы – 35 минут.

4. Ну и ради интереса добавим в качестве признаков всю информацию о среде (всю матрицу 5x5). В реальных задачах это делать бессмысленно, так как смысл обучения с подкреплением в том, чтобы не держать в памяти модели всю информацию о среде.



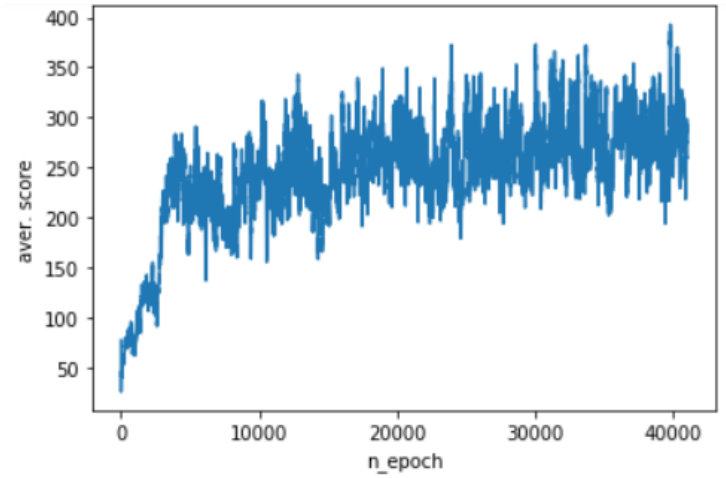
Время работы – 45 минут.

SARSA:

Отличается от Q-learning тем, что вместо жадной стратегии, используется действие, выбранное на основе текущей политики действий, те:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \cdot \left(r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right)$$

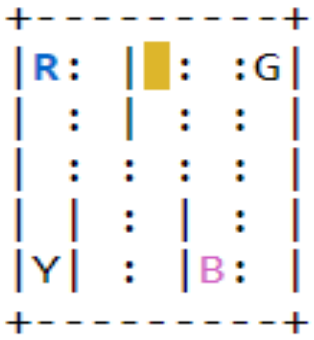
В рамках предыдущей задачи пункта 2 получим:



Результат вышел лучше, чем в Q-learning и, очевидно, алгоритм сработал быстрее (не нужно было искать максимум).

Время работы – 20 минут.

Так же работу алгоритма можно продемонстрировать на примере работы со средой Taxi-v3.



Наша задача – подхватить пассажира в одной точке и высадить его в другой.

Итак, опишем задачу подробнее. Такси – единственный автомобиль на данной парковке. Парковку можно разбить в виде сетки 5x5, где получаем 25 возможных расположений такси. Эти 25 значений –элемент пространства состояний.

В среде есть 4 точки, в которых допускается высадка пассажиров: это: R, G, Y, B или [(0,0), (0,4), (4,0), (4,3)] в координатах (по горизонтали; по вертикали), если интерпретировать среду в декартовых координатах. Если также учесть еще одно (1) состояние пассажира: внутри такси, то можно взять все комбинации локаций пассажиров и их мест назначения, чтобы подсчитать общее количество состояний в нашей среде для обучения такси: имеем четыре (4) места назначения и пять (4+1) локаций пассажиров.

Итак, в нашей среде для такси насчитывается 5×5×5×4=500 возможных состояний. Агент имеет дело с одним из 500 состояний и предпринимает действие.
Это множество action space(A): совокупность всех действий, которые наш агент может предпринять в заданном состоянии.

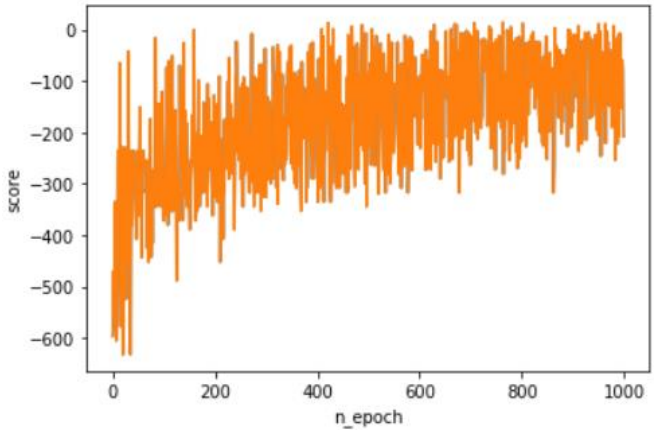
Как понятно из иллюстрации выше, такси не может совершать определенные действия в некоторых ситуациях (мешают стены). В коде, описывающем среду, мы просто назначим штраф -1 за каждое попадание в стену. Таким образом, подобные штрафы будут накапливаться, поэтому такси попытается не врезаться в стены. Ещё получаем +20 очков за успешную высадку пассажира и теряем 1 очко за каждый шаг, затраченный на его перемещение.

Также предусмотрен штраф 10 очков за каждую непредусмотренную посадку и высадку пассажира. В итоге цель – минимизировать штраф.

Реализована eps-greedy политика, то есть чем сильнее колебания у решения, тем сильнее должны влиять новые знания от обучения.

Код работы: https://github.com/EvgenijGod/Reinforcement_learning_mmp

Результат: Время работы – 25 минут.



$$\alpha = 0.1, \gamma = 0.9, \varepsilon = 0.9$$

Различия Q-learning и SARSA:

SARSA изучает значения действий по сравнению с политикой, которую она следует, в то время как вне политическое Q-Learning делает это относительно жадной политики. К результату они сходятся, но с разной скоростью.

Q-Learning имеет тенденцию к сближению немного медленнее, в силу перебора на каждом шаге по всем действиям, но обладает способностью продолжать обучение при изменении политики. Q-learning изучает оптимальную политику действий.

SARSA работает быстрее, так как фактические следует одной и той же политике. SARSA - почти оптимальную. Для изучения оптимальной – нужно использовать greedy-action, которое влечет ввод дополнительного гиперпараметра.

С физической точки зрения, SARSA всегда старается учесть штрафы одной стратегии, то есть происходит учет штрафов за исследование действий. Q-

learning же игнорирует это и каждый раз меняется стратегию. Таким образом, есть шанс, что Q-learning может получить большие штрафы за свои действия, а SARSA действует аккуратнее. В реальном мире это очень важно, например для обучения дорогостоящих роботов. Не хотелось бы, чтобы робот прежде, чем научился что-то делать, разбил себя в процессе обучения в силу смены стратегии.