

Обучение с подкреплением (Reinforcement Learning, RL)

RL - набор методов, позволяющих агенту (интеллектуальной системе) вырабатывать оптимальную стратегию при его взаимодействии со средой (внешним миром).

В каждый (обычно дискретный) момент времени t агент может оказать на среду некоторое **воздействие** (action) a_t . В ответ он получает информацию – **наблюдение** (observation) o_{t+1} , полностью или частично характеризующее новое состояние среды (state) s_{t+1} и числовую характеристику успешности действий, называемую **наградой** (reward) r_{t+1}

$$o_0 \xrightarrow{a_0} \{o_1, r_1\} \xrightarrow{a_1} \{o_2, r_2\} \xrightarrow{a_2} \dots$$

Обучение с подкреплением (Reinforcement Learning, RL)

Важное отличие обучения с подкреплением от стандартного обучения с учителем - это отсутствие обучающих данных вида ,

$$\{(s, a), (s', a'), \dots\}$$

где a - "известные учителю" оптимальные действия, которые необходимо выполнить, когда среда находится в данном состоянии.

Агент должен решить задачу, даже если человек не знает как её решать (не знает какие действия оптимальны).

Обучение с подкреплением (Reinforcement Learning, RL)

На основе полученной к моменту времени t информации

$\{\dots; o_{t-2}, a_{t-2}; o_{t-1}, a_{t-1}; o_t\}$ (траектории)

агент должен выбрать очередное действие a_t так, чтобы максимизировать накопленное вознаграждение $R = r_1 + r_2 + \dots$ за достаточно большой промежуток времени.

Среда (environment) может быть -

детерминированной или стохастической;

с полным или неполным описанием;

стационарной или нестационарной.

В среде могут быть терминальные состояния, при достижении которых "игровой" эпизод оканчивается.

Обучение с подкреплением (Reinforcement Learning, RL)

Возможные **действия** a_t могут принадлежать конечному множеству целых чисел (0 - шаг влево или 1 - шаг вправо) или быть вектором вещественных чисел: {скорость шага, угол поворота}.

Многие методы способны работать только с дискретными действиями или только с непрерывными действиями.

В типичных задачах RL **вознаграждение** r_{t+1} часто не связано непосредственно с действием a_t . Отличное от нуля вознаграждение может поступать редко, запаздывать и возникать только после серии негативных значений (долго шел, но затем много получил). Иногда вознаграждение приходит только в конце (например, при игре в го). Вознаграждение также может быть всегда отрицательным.

Обучение с подкреплением (Reinforcement Learning, RL)

Обычно вместо **наблюдения** за средой o_t , говорят о её **состоянии** s_t .

Следует помнить, что в общем случае эти величины могут не совпадать. Для среды с полным описанием $s_t = o_t$.

При неполном описании нужно учитывать историю наблюдений и действий. Но даже при учёте всей траектории $s_0, a_0, s_1, a_1, \dots$

состояние **среды** может быть вероятностно или вообще оказаться "вещью в себе".

Обучение с подкреплением (Reinforcement Learning, RL)

В процессе обучения агент может формировать модель среды, которая предсказывает вероятность следующего наблюдения на основании траектории:

$$P(s_{t+1}, r_{t+1} \mid \dots, s_t, a_t)$$

Если награда определяется только состоянием, то модель это

$$P(s_{t+1}, \mid \dots, s_t, a_t) \quad r_t = r(s_t)$$

Если текущее состояние среды s_t содержит всю информацию, необходимую для принятия решения, то такая среда называется **марковской**.

Формально, это означает, что условная вероятность перехода в новое состояние зависит только от текущего состояния и действия:

$$P(s_{t+1}, r_{t+1} \mid \dots, s_{t-1}, a_{t-1}, s_t, a_t) = P(s_{t+1}, r_{t+1} \mid s_t, a_t)$$

Описания процесса

Пространство состояний СМО

$M / M / 1$



Число означает количество заявок в системе
 λ - интенсивность поступления заявок
 μ - параметр показательного распределенного времени обслуживания в ОА

марковский процесс

Обучение с подкреплением (Reinforcement Learning, RL)

Выбор оптимальной стратегии в такой среде называется **марковским процессом принятия решения** (Markov decision process, MDP).

Если наблюдение не даёт всей информации о состоянии (неполное описание, частичная наблюдаемость), то решения в такой среде называется *частично наблюдаемым* марковским процессом принятия решения (partially observable Markov decision process, POMDP).

Обучение с подкреплением (Reinforcement Learning, RL)

Алгоритм, по которому агент выбирает действие, принято называть его **политикой**.

Это может быть обычная (детерминированная) функция

$$a_t = \pi(s_t) \quad (\text{policy function})$$

где в качестве аргумента выступает текущее состояние S_t .

В S_t могут также включаться предшествующие действия и награды.

Для изучения среды агент должен постоянно *экспериментировать*. Даже в стационарной среде с полным описанием, эксперименты нужны, чтобы *не застрять* в некоторой области пространства состояний, т.к. возможно, есть более выгодные состояния среды.

Поэтому в общем случае **политика** - это условная вероятность $p(a/s)$ выбора агентом действия a в состоянии s

Обучение с подкреплением (Reinforcement Learning, RL)

Детерминированную функцию политики можно превратить в вероятностную. Например, с вероятностью p выбирается оптимальное (к данному моменту обучения) действие $a = \pi(s)$, а с вероятностью $1-p$ - равновероятно любое другое состояние.

Если множество действий дискретно, можно, например, аппроксимировать $p(a/s)$ нейронной сетью с числом выходов равных числу возможных действий на которых будет (после softmax) распределение вероятностей соответствующего решения.

Граф состояний системы



Многие задачи обучения с подкреплением сводятся к марковским процессам. Формально, это означает, что условная вероятность перехода в новое состояние зависит только от текущего состояния, выполненного в нём действия и не зависит от предыдущих состояний и действий.

$$P(s_{t+1} r_{t+1} | \dots s_{t-1}, a_{t-1}, s_t, a_t) = P(s_{t+1} r_{t+1} | s_t, a_t)$$

В этом случае будущее определяется настоящим и не требует знания прошлого. В общем случае состояние может зависеть от нескольких предыдущих наблюдений.

Выбор оптимальной стратегии (policy function) в такой марковской модели называется марковским процессом принятия решения (Markov decision process MDP).

Модель марковских переходов

Рассмотрим сначала модель, в которой нет действий.

Среда в дискретные моменты времени из состояния s случайно переходит в очередное состояние s' с вероятностями $P_s = P(s' | s)$

При попадании в состояние с номером s' начисляется фиксированное вознаграждение r' , зависящее от s' .

В результате получается временная последовательность
(нижний индекс - момент времени):

$$s_0 \mapsto s'_1, r'_1 \mapsto s''_2, r''_2 \mapsto \dots, \quad s_t \in \mathbb{S}, \quad r_t \in \mathbb{R},$$

где \mathbb{S} - множество целых чисел (номера состояний), а \mathbb{R} - множество вещественных чисел (награды).

Нас интересует суммарное будущее дисконтированное вознаграждение, полученное после момента времени t

$$R_{t+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = r_{t+1} + \gamma R_{t+2}$$

Ценностью состояния $V(s)$ называется суммарное дисконтированное вознаграждение, при старте из состояния s в последующих состояниях, усреднённое по всем возможным траекториям:

$$V(s) = \langle R_{t+1} | s_t = s \rangle$$

Запишем будущее накопленное дисконтированное вознаграждение в виде

$$R_{t'} = r_{t'} + \gamma R_{t''} \quad \text{где } t'=t+1, t''=t+2$$

Усредним его выражение по траекториям, стартующим из состояния $s_t = s$

$$V(s) = \langle r_{t+1} + \gamma R_{t+2} \mid s_t = s \rangle$$

Пусть среда из состояния с номером i может перейти в состояния $1, 2, \dots, n$.

При переходе в j -е состояние агент получит вознаграждение r_j (за попадание в него) плюс дисконтированное будущее вознаграждение равное V_j за всю последующую историю (ценность j -го состояния).

Для получения V_i все эти награды следует усреднить.

Для этого умножим их на вероятности переходов $P_{ij} = P(s_{t'} = j \mid s_t = i)$ и просуммируем по j :

$$V_i = \sum_j P_{ij} \cdot (r_j + \gamma V_j)$$

Эта формула называется уравнением Беллмана

ФУНКЦИЯ ПОЛИТИКИ

Типичная для обучения с подкреплением (RL) постановка задачи

Пусть в состоянии с номером i агент с вероятностью $\pi_{i\alpha} = \pi(\alpha|i)$ совершает действие с номером α (латинские индексы используются для номеров состояний, а греческие - для действий).

В результате действия агента среда переходит в состояние под номером j с вероятностью $P_{i\alpha j} = P(j|i, \alpha)$ получая в нём награду $r_j = r(s_j)$.

$$V_i = \sum_j \tilde{P}_{ij} (r_j + \gamma V_j), \quad \tilde{P}_{ij} = \sum_{\alpha} \pi_{i\alpha} P_{i\alpha j}$$

Ценности состояний по-прежнему удовлетворяют уравнению Беллмана, где P_{ij} вероятности перехода из i в j равны произведению вероятности сделать в состоянии i действие α на вероятность перейти после этого в состояние j с суммой по всем возможным действиям.

ФУНКЦИЯ ПОЛИТИКИ

Найдём уравнение для Q-функции полезности действия a , совершённого в состоянии s :

$$Q(s, a) = \langle R_{t+1} \mid s_t = a, a_t = a \rangle$$

Для дискретного случая это будет матрица $Q_{i\alpha}$.

Зная её, можно вычислить полезность состояния V (при совершении любого действия с вероятностями $\pi_{i\alpha}$):

$$V_i = \sum_{\alpha} \pi_{i\alpha} Q_{i\alpha}$$

Можно также записать обратную связь. Ценность действий $Q_{i\alpha}$ в состоянии i равна среднему вознаграждению при непосредственном переходе в состояние j плюс дисконтированные будущие вознаграждения состояний, в которые среда попадает при таком переходе:

$$Q_{i\alpha} = \sum_j P_{i\alpha j} (r_j + \gamma V_j)$$

полезности состояний V и действий Q зависят от выбранной агентом политики (матрицы вероятностей $\pi_{i\alpha}$). Изменение политики меняет и эти функции.

ФУНКЦИЯ ПОЛИТИКИ

Разные политики $\pi(a/s)$ приводят к различным Q и V функциям. Из всех возможных политик оптимальной будет та, для которой значения ценности состояний и действий максимальны:

$$V^*(s) = \arg \max_{\pi} V^{\pi}(s), \quad Q^*(s, a) = \arg \max_{\pi} Q^{\pi}(s, a)$$

В свою очередь, если известна оптимальная Q^* -функция несложно записать оптимальную детерминированную политику:

$$a = \pi^*(s) = \arg \max_a Q^*(s, a)$$

В процессе обучения (пока Q известна плохо) такую детерминированную политику можно рандомизировать, выбрав параметр ϵ и равномерно распределённое случайное число p

$$a = \begin{cases} \arg \max_a Q(s, a) & p > \epsilon \\ \text{any} & \text{else} \end{cases}$$

ФУНКЦИЯ ПОЛИТИКИ

Если $\epsilon=0$ то всегда выбирается оптимальное действие. При $\epsilon=0.5$ в половине случаев будет выбираться случайное действие. По мере обучения ϵ уменьшается. Это позволяет на начальных этапах агенту больше "экспериментировать".

Вместо ϵ -стратегии действие можно также выбирать с вероятностями пропорциональными их полезности.

Для этого используется функция softmax по индексу α :

$$\pi_{i\alpha} = \text{soft}(Q_{i\alpha} - \bar{Q}_{i\alpha}), \quad \text{soft}(x_\alpha) = \frac{e^{x_\alpha}}{\sum_\beta e^{x_\beta}}$$

где среднее $Q_{i\alpha}$ (также по индексу α) вычитается, чтобы не произошло переполнения при вычислении экспонент. На Python это выглядит следующим образом:

```
from scipy.special import softmax # импортируем функцию softmax

Q = np.random.uniform( size=(num_states, num_actions) )
aQ = Q.mean(axis=1, keepdims=True) # средние значения (num_states, 1)
pi = softmax(Q - aQ, axis=1) # условные вероятности pi(a|s)
s = 1 # номер состояния
a = np.random.choice(np.arange(num_actions), p=pi[s])
```

Максимизация Q-функции

Пусть агент выбирает действие, соответствующее его максимальной полезности $a = \operatorname{argmax}_a Q(s, a)$ и всегда ей следует. Тогда:

$$V_i = \sum_{\alpha} \pi_{i\alpha} Q_{i\alpha} = \max_{\alpha} Q_{i\alpha}$$

так как все вероятности политики равны нулю, кроме того действия, для которого $Q_{i\alpha}$ максимальна. Поэтому, уравнение Беллмана $Q_{i\alpha} = \sum_j P_{i\alpha j} (r_j + \gamma V_j)$ для полезности действий в оптимальной стратегии приобретает вид:

$$Q_{i\alpha} = \sum_j P_{i\alpha j} (r_j + \gamma \max_{\beta} Q_{j\beta})$$

Даже, если модель среды $(P_{i\alpha j}, r_j)$ известна, непосредственно это уравнение решить нельзя. Однако можно воспользоваться итерационным методом:

$$Q_{i\alpha}^{\text{new}} = Q_{i\alpha} + \lambda \left[\sum_j P_{i\alpha j} (r_j + \gamma \max_{\beta} Q_{j\beta}) - Q_{i\alpha} \right]$$

где λ - гиперпараметр (скорость обучения). Смысл этого соотношения такой: если $Q_{i\alpha}$ меньше, чем требуемое по уравнению Беллмана значение, то её необходимо немного увеличить, а если больше - то уменьшить. Параметр λ регулирует степень изменения $Q_{i\alpha}$ на каждой итерации.

Метод Q-обучения (Q-learning)

Если модель среды неизвестна, используют Монте-Карло моделирование, в процессе которого подправляют значения $Q_i \alpha$.

При этом, для исследования пространства состояний применяется ϵ -жадная стратегия с убывающим ϵ :

$$a = \pi_\epsilon(s) = \text{any if } p < \epsilon \text{ else } \arg \max_a Q(s_t, a)$$

Тогда на каждом шаге моделирования производится вычисление:

$$a_t = \pi_\epsilon(s_t) \mapsto s_{t+1}, r_{t+1}$$

$$Q^{\text{new}}(s_t, a_t) = Q(s_t, a_t) + \lambda [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Выражение в квадратных скобках называется **temporal difference** (TD) и обычно обозначается символом δ .

Если $\delta > 0$, то переход $s, a \mapsto s', r'$ является позитивно неожиданным, а если $\delta < 0$ - то негативно неожиданным.

Такой итерационный метод называется Q-learning. Это типичный пример онлайн обучения, когда обучение происходит на каждой новой полученной порции данных.

Метод Q-обучения (Q-learning)

Параметр жадности ϵ начнем со значения eps1 , экспоненциально уменьшая его до значения eps2 в течении decays эпох, после чего положим $\epsilon = 0$

```
Q = np.zeros( (num_states, num_actions) ) # начальные значения 0

eps1, eps2, decays = 1, 0.001, 50000 # параметры epsilon-распада
epsilon = eps1
decay = math.exp(math.log(eps2/eps1)/decays)

def policy(s): # epsilon-жадная политика
    if np.random.random() < epsilon: # случайно любое действие
        return np.random.randint(n_actions)
    return np.argmax(Q[s]) # иначе лучшее

def run_episode(ticks=1000):
    s0 = env.reset() # начальное состояние среды
    for _ in range(ticks):
        a0 = policy(s0) # выбираем действие
        s1, r1, done = env.step(a) # новое состояние и награда

        Q[s0, a0] += lm * ( r1 + gamma * np.max(Q[s1]) - Q[s0, a0] )

    if done: # терминальное состояние
        return

    s0 = s # продолжаем с нового состояния
```

Итерационный поиск политики

Если модель среды $P(s|a)$ известна, можно воспользоваться простым **итерационным алгоритмом вычисления политики** (policy iteration algorithm). В этом алгоритме сначала задаётся некоторая детерминированная политика $\pi(a/s)$ (она обычно инициализируется случайными значениями).

На её основе вычисляется V -функция. Затем, с её помощью строится уточнённая (улучшенная) политика и вычисления повторяются.

Выполняются два повторяющихся этапа:

1. **Оценка политики** (policy evaluation). Итерационно вычисляем для всех состояний, пока с некоторой точностью функция ценности не перестанет изменяться:

$$V_{k+1}(s) = \sum_{s', r'} P(s', r' | s, a) [r' + \gamma V_k(s')], \quad a = \pi(s)$$

В принципе, можно использовать два массива для k -ой итерации V_k и для следующей V_{k+1} и затем менять их местами. Однако хорошо работает и обновление по месту, при котором для некоторых состояний уже используется улучшенное значение ценности.

2. **Улучшение политики** (improving the policy), при котором вычисляется политика на основании жадной стратегии:

$$\pi(s) = \arg \max_a Q(s, a) = \arg \max_a \sum_{s', r'} P(s', r' | s, a) [r' + \gamma V_k(s')]$$