Q-learning

Оля Агапова, 181 группа

Cliff Walking

1. Нам надо на другую сторону пропасти.
2. Падать в пропасть нельзя.
3. Мы спешим.

Зададим каждой клетке “вознаграждение” (цену)

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	2.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0

В чем смысл такой постановки задачи?

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	2.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	1.0

Задача построена так, чтобы ее решением было *максимизировать сумму всех будущих вознаграждений*

Ок. Как это сделать?

Q-learning алгоритм:

1. Создаем таблицу размера n*m, где n - количество возможных положений в нашем клетчатом мире, а m - количество возможных действий (например, пойти вверх-вниз-влево-вправо)

Ок. Как это сделать?

Q-learning алгоритм:

1. Создаем таблицу размера n*m, где n - количество возможных положений в нашем клетчатом мире, а m - количество возможных действий (например, пойти вверх-вниз-влево-вправо)
2. В таблице будем хранить Q-values (скалярные числа)

Что такое Q-values?

Каждое Q-value -- это оценка суммы будущих вознаграждений (как раз то, что мы хотели бы максимизировать),

при условии, что мы находимся в *этой* клетке и делаем *это* действие.

Обновление Q-values

Заполненная таблица выглядит так:

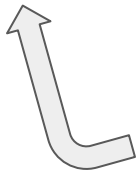
	0	1	2	3	4	5	6	7	8	9	10	11
0	U: -6.76 D: -6.73 R: -6.75 L: -6.71	U: -6.70 D: -6.74 R: -6.60 L: -6.62	U: -6.42 D: -6.53 R: -6.34 L: -6.34	U: -6.14 D: -6.09 R: -6.06 L: -6.12	U: -5.82 D: -5.77 R: -5.74 L: -5.78	U: -5.51 D: -5.37 R: -5.36 L: -5.53	U: -5.12 D: -4.97 R: -4.94 L: -5.32	U: -4.58 D: -4.49 R: -4.49 L: -4.69	U: -4.01 D: -4.02 R: -3.94 L: -4.28	U: -3.57 D: -3.37 R: -3.36 L: -3.70	U: -2.65 D: -2.69 R: -2.65 L: -3.01	U: -2.26 D: -1.90 R: -2.07 L: -2.15
1	U: -6.81 D: -6.96 R: -6.89 L: -6.89	U: -6.80 D: -6.71 R: -6.70 L: -6.75	U: -6.51 D: -6.43 R: -6.45 L: -6.67	U: -6.17 D: -6.08 R: -6.09 L: -6.39	U: -5.76 D: -5.68 R: -5.68 L: -5.98	U: -5.55 D: -5.21 R: -5.21 L: -5.63	U: -4.73 D: -4.68 R: -4.68 L: -4.92	U: -4.37 D: -4.09 R: -4.09 L: -4.23	U: -3.92 D: -3.44 R: -3.44 L: -3.98	U: -3.80 D: -2.71 R: -2.71 L: -2.93	U: -2.84 D: -1.90 R: -1.90 L: -3.24	U: -1.72 D: -1.00 R: -1.43 L: -2.27
2	U: -7.06 D: -7.40 R: -6.86 L: -7.14	U: -6.96 D: -99.95 R: -6.51 L: -7.15	U: -6.71 D: -93.75 R: -6.13 L: -6.86	U: -6.37 D: -96.88 R: -5.70 L: -6.16	U: -6.09 D: -99.61 R: -5.22 L: -6.12	U: -5.60 D: -99.22 R: -4.69 L: -5.68	U: -5.07 D: -99.22 R: -4.10 L: -5.18	U: -4.60 D: -99.22 R: -3.44 L: -4.07	U: -4.07 D: -96.88 R: -2.71 L: -3.98	U: -3.34 D: -98.44 R: -1.90 L: -3.35	U: -2.64 D: -98.44 R: -1.00 L: -2.63	U: -1.72 D: 0.00 R: -1.00 L: -1.81
3	U: -7.18 D: -7.46 R: -99.22 L: -7.45	U: 0.00 D: 0.00 R: 0.00 L: 0.00	U: 0.00 D: 0.00 R: 0.00 L: 0.00	U: 0.00 D: 0.00 R: 0.00 L: 0.00	U: 0.00 D: 0.00 R: 0.00 L: 0.00	U: 0.00 D: 0.00 R: 0.00 L: 0.00	U: 0.00 D: 0.00 R: 0.00 L: 0.00	U: 0.00 D: 0.00 R: 0.00 L: 0.00	U: 0.00 D: 0.00 R: 0.00 L: 0.00	U: 0.00 D: 0.00 R: 0.00 L: 0.00	U: 0.00 D: 0.00 R: 0.00 L: 0.00	U: 0.00 D: 0.00 R: 0.00 L: 0.00

Обновление Q-values

1. Посмотрим на соседнюю от цели клетку, запишем в ней максимальную возможную сумму (соотв. нужному действию), потому что знаем, что до нее 1 шаг
2. Все соседние клетки с этой первой соседней тоже нас устраивают, поэтому и им поставим максимальную возможную сумму (по направлению к первой соседней клетке)
3. Так рано или поздно можем изучить и заполнить всю таблицу

Обновление Q-values

1. Посмотрим на соседнюю от цели клетку, запишем в ней максимальную возможную сумму (соотв. нужному действию), потому что знаем, что до нее 1 шаг
2. Все соседние клетки с этой первой соседней тоже нас устраивают, поэтому и им поставим максимальную возможную сумму (по направлению к первой соседней клетке)
3. Так рано или поздно можем изучить и заполнить всю таблицу



Это и делает Q-алгоритм!

Но как?

Уравнение Беллмана

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

Если без коэффициента:
Для любой пары *(положение на клетке **s**; действие **a**)* Q-value будет равно сумме вознаграждения за действие ***a*** из клетки ***s***
+
максимально возможной сумме будущих после этого вознаграждений.

Но как?

Уравнение Беллмана

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

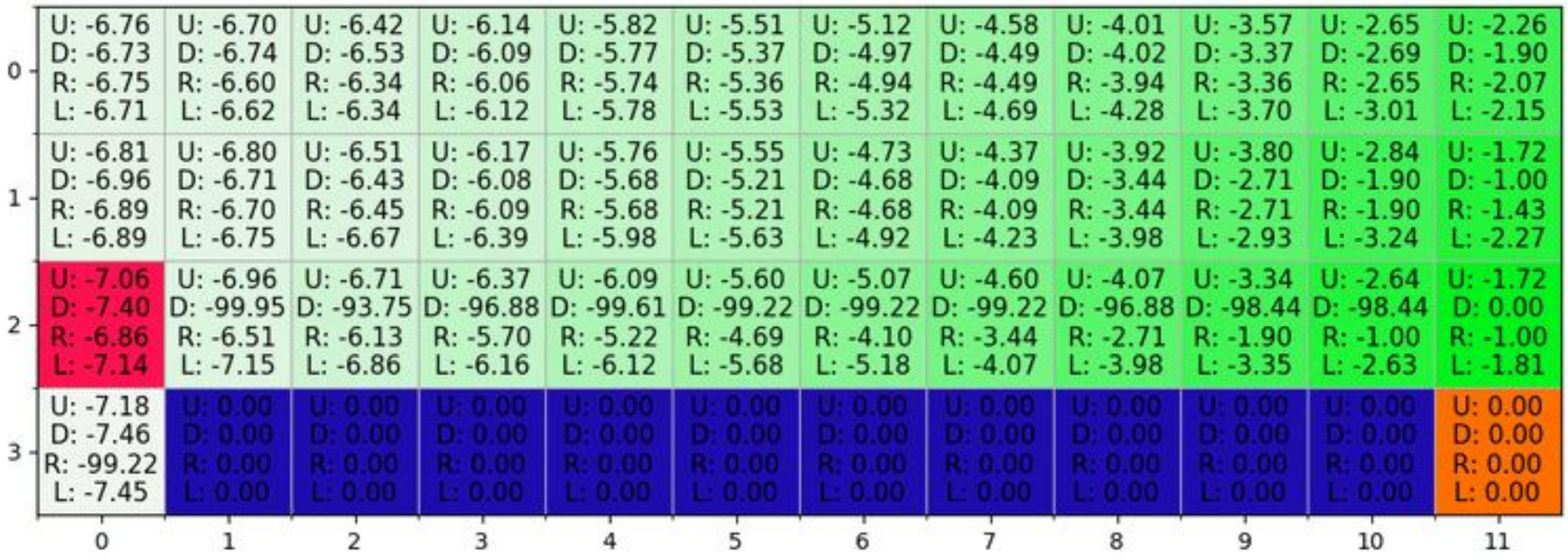
Если без коэффициента:
Для любой пары *(положение на клетке **s**; действие **a**)* Q-value будет равно
сумме вознаграждения за действие ***a*** из клетки ***s***
+
максимально возможной сумме будущих после этого вознаграждений.

Коэффициент от 0 до 1 позволяет нам регулировать важность этой “будущей суммы”

$$Q(s[2,11],\ \text{down}\ ) = r + \gamma \max_{a'} Q(s[3,11], a') = 0 + 0.9 \times 0 = 0$$

$$Q(s[2,10],\ \text{right}\ ) = r + \gamma \max_{a'} Q(s[2,11], a') = (-1) + 0.9 \times 0 = -1$$

$$Q(s[2,9],\ \text{right}\ ) = r + \gamma \max_{a'} Q(s[2,10], a') = (-1) + 0.9 \times (-1) = -1.9$$



Имплементация

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

Имплементация

```
# Initialize Q arbitrarily, in this case a table full of zeros
q_values = np.zeros((num_states, num_actions))

# Iterate over 500 episodes
for _ in range(500):
    state = env.reset()
    done = False

    # While episode is not over
    while not done:
        # Choose action
        action = egreedy_policy(q_values, state, epsilon=0.1)
        # Do the action
        next_state, reward, done = env.step(action)
        # Update q_values
        td_target = reward + gamma * np.max(q_values[next_state])
        td_error = td_target - q_values[state][action]
        q_values[state][action] += learning_rate * td_error
        # Update state
        state = next_state
```

```
def egreedy_policy(q_values, state, epsilon=0.1):
    # Get a random number from a uniform distribution between 0 and 1,
    # if the number is lower than epsilon choose a random action
    if np.random.random() < epsilon:
        return np.random.choice(4)
    # Else choose the action with the highest value
    else:
        return np.argmax(q_values[state])
```

On-policy / off-policy

Что такое “*policy*”?

Это “политика”, “стратегия”, согласно которой мы двигаемся по полю. Она может, например, быть жадной -- это значит, что мы движемся по полю только в направлении максимальной суммы вознаграждений, и никуда больше.

По сути она задает вероятность того, что из конкретной клетки мы сделаем конкретное движение

On-policy / off-policy

Что такое “*policy*”?

Это “политика”, “стратегия”, согласно которой мы двигаемся по полю. Она может, например, быть жадной -- это значит, что мы движемся по полю только в направлении максимальной суммы вознаграждений, и никуда больше.

Что мы вообще хотим в процессе обучения?

1. Подобрать функцию $Q(s,a)$, которая предсказывает сумму будущих вознаграждений
2. Подобрать политику π (actually, $\pi(a|s)$), которая нас приведет к максимальному вознаграждению.

On-policy and off-policy имеет отношение только к первой задаче.

В чем разница?

В *on-policy learning*, функция $Q(s,a)$ обучается из действий, которые выбираются согласно какой-то политике $\boldsymbol{\pi(a|s)}$.

В *off-policy learning*, она $Q(s,a)$ обучается из действий, необязательно согласующихся с политикой (например, рандомными).

Почему Q-learning - off-policy?

Потому что Q-values обновляются с помощью Q-value следующего положения s' и самого выгодного (жадного) действия a'.

Иными словами, алгоритм оценивает сумму будущих вознаграждений для пары (положение, действие) в предположении, что у нас жадная политика, но сам он необязательно ей следует (ведь робот может двигаться еще и исследуя территорию).

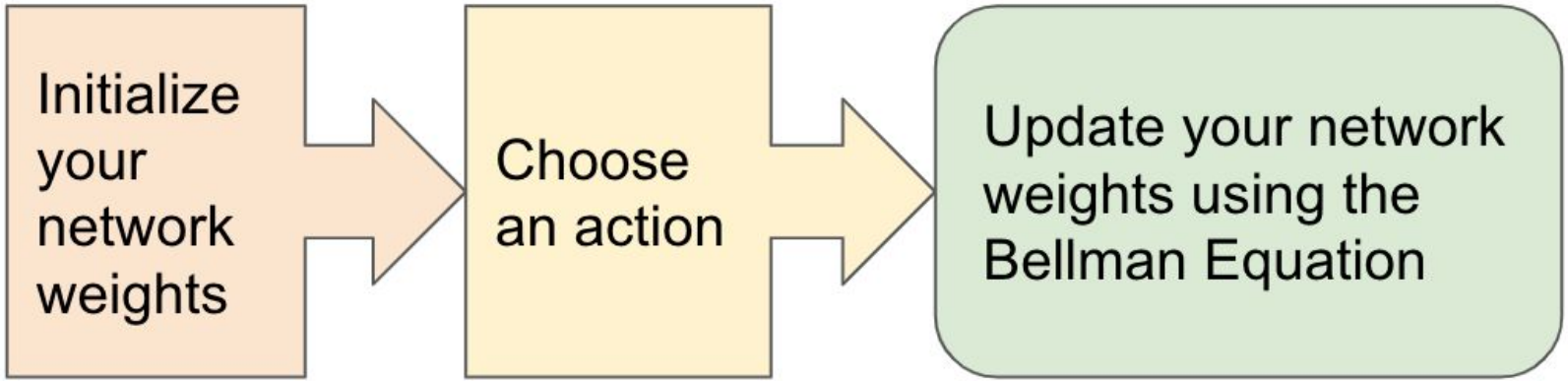
Пример on-policy - SARSA

Это алгоритм, который похож на Q, но обновляет Q-values с помощью Q-value следующего положения и действия, которое подчиняется текущей политике (необязательно жадной).

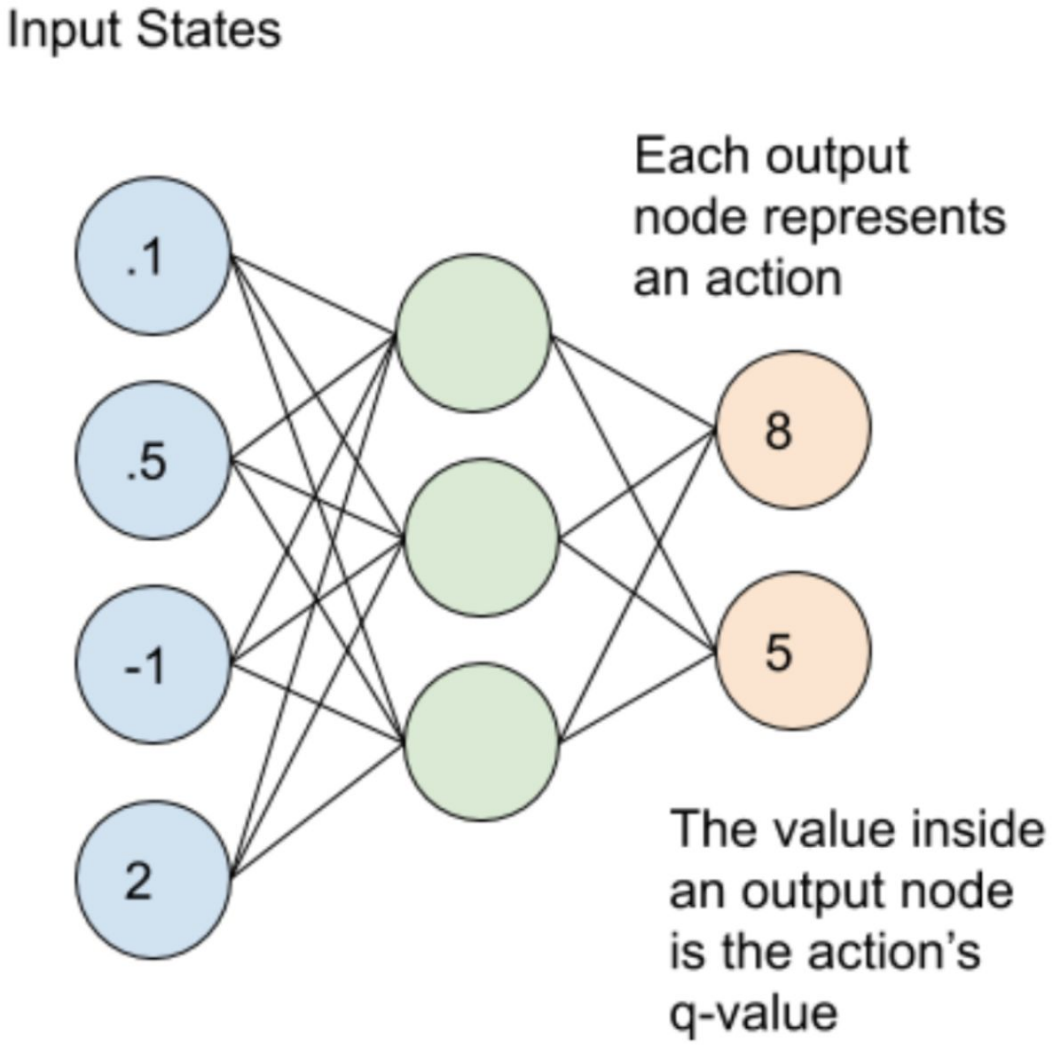
Представим Q-learning как нейронную сеть

Обычный Q-Learning: таблица, в которой каждой паре (положение, действие) ставится в соответствие Q-value

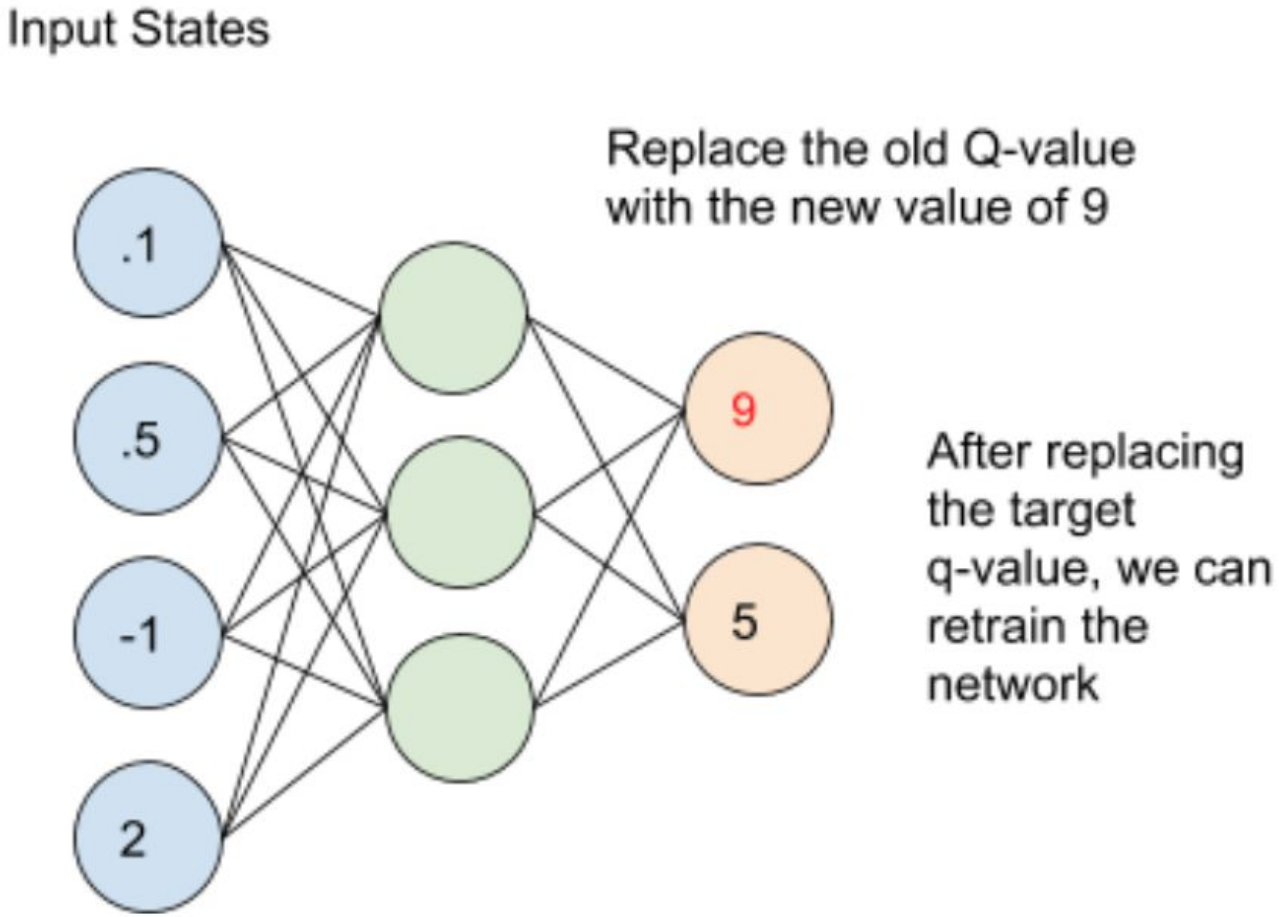
Deep Q-Learning: нейронная сеть, которая ставит входящим положениям в соответствие пары (действие, Q-value)



- Нейронных сети 2 -- главная и целевая
- Раз в N шагов веса из главной копируются в целевую (для стабильности)
- Входные ноды - это положения, выходные - действие и предсказание его Q-value



- Обновление происходит с помощью того же уравнения Беллмана



Experience Replay

- Сохраняем информацию о всех состояниях в формате (положение, действие, награда, следующее положение)
- Через каждые несколько шагов можно брать батчи из этих данных и учиться на них, а не на предыдущем действии (это повышает устойчивость обучения)

Список литературы

- Understanding Q-Learning, the Cliff Walking problem
- Deep Q-Learning Tutorial: minDQN
- Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, 2018
- Reinforcement Learning (RL for the intimates)