Обучение с подкреплением

Классическое обучение с подкреплением

JET BRAINS Research

академия больших данных mail.ru group made

HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

Про что этот курс?

Что будет на курсе:
- Немного классической теории
- Популярные современные алгоритмы
- Некоторые продвинутые алгоритмы RL, которые будут полезны на практике
- Рассказ о том, где и как применяется RL

Что можно посмотреть:
- Reinforcement Learning: An Introduction (Sutton & Barto)
- Spinning Up in Deep RL (OpenAI)
- Practical RL (Yandex)

Оценка:
- Будет 4 ДЗ
- За каждое ДЗ можно получить от 0 до 10 баллов
- 32+ баллов - отлично, 24+ баллов - хорошо, 16+ баллов - удовлетворительно

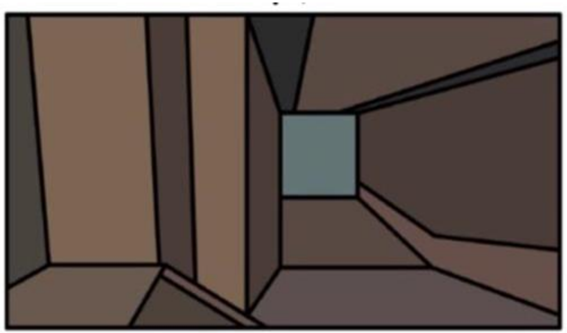
Задача обучения с учителем



Кошка ↔ Собака



Обучение с учителем



Состояние

Действие

Обучение с подкреплением

Обучение с подкреплением

Supervised Learning vs Reinforcement Learning

Обучение с учителем:
- Заранее известны ответы
- Есть фиксированный набор данных
- Результат зависит только от текущего решения

Обучение с подкреплением:
- Набора данных нет
- Известных ответов нет
- Распределение получаемых данных сильно зависит от агента
- Результат зависит от последовательности принятых решений

Простой случай: многорукие бандиты

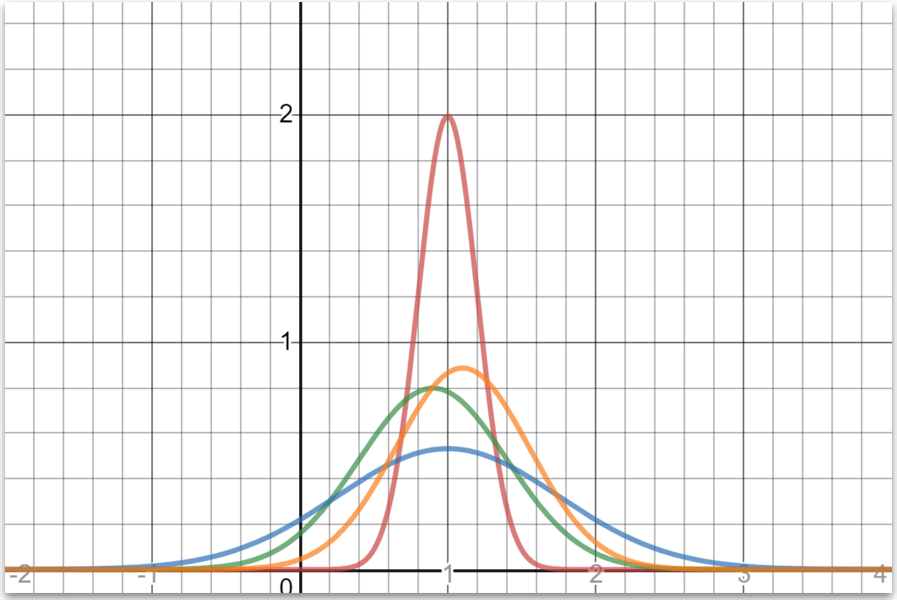
Постановка задачи:

- Эпизод длится ровно t шагов.
- На каждом шаге можно совершить одно из k действий и получить награду, которая распределена случайно и зависит только от действия.
- Хотим максимизировать суммарную награду за эпизод.

Простой случай: многорукие бандиты

Постановка задачи:

- Эпизод длится ровно t шагов.
- На каждом шаге можно совершить одно из k действий и получить награду, которая распределена случайно и зависит только от действия.
- Хотим максимизировать суммарную награду за эпизод.



Простой случай: многорукие бандиты

Определим value действия как математическое ожидание награды:

$$q(a) = \mathbb{E}_{r \sim p(r|a)} r$$

Если мы знаем value каждого действия, то можем просто выбрать то действие, у которого больше всего value.

Но value мы не знаем, и поэтому будем его приближать с помощью Q(a). Хотим, чтобы Q(a) было как можно ближе к q(a).

Наивный алгоритм

Что будет, если мы будем всегда выбирать лучшее действие в соответствии с приближением, а затем обновлять приближение на основе наблюдений?

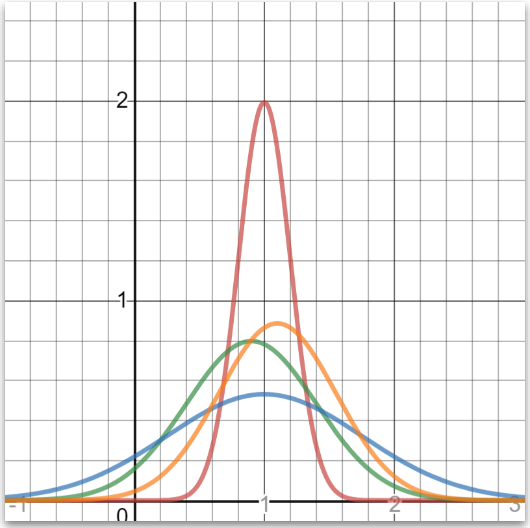
Алгоритм:

1. Инициализировать $Q(a) \leftarrow 0$
2. Инициализировать $N(a) \leftarrow 0$ // Счетчик количества применений действия
3. for t in range(T):
    1. a = argmax Q(a)
    2. r = step(a) // Совершаем действие и получаем награду
    3. N(a) += 1
    4. Q(a) = r / N(a) + (1 - 1 / N(a)) Q(a) // Рекуррентная формула среднего

Наивный алгоритм

Что будет, если мы будем всегда выбирать лучшее действие в соответствии с приближением, а затем обновлять приближение на основе наблюдений?

Время, *t*	Действие, *a*	Q(1)	Q(2)	Q(3)	Q(4)	Награда, *r*
1	3	0	0	0	0	1.2050
2	3	0	0	1.2050	0	0.1138
3	3	0	0	0.6594	0	1.5140
4	3	0	0	0.9442	0	1.3417



Что пошло не так?

Epsilon-greedy exploration

Когда мы выбираем действие с наибольшим value, мы используем уже полученные знания (**exploit**), однако чтобы получить эти знания, нам необходимо исследовать среду (**explore**).

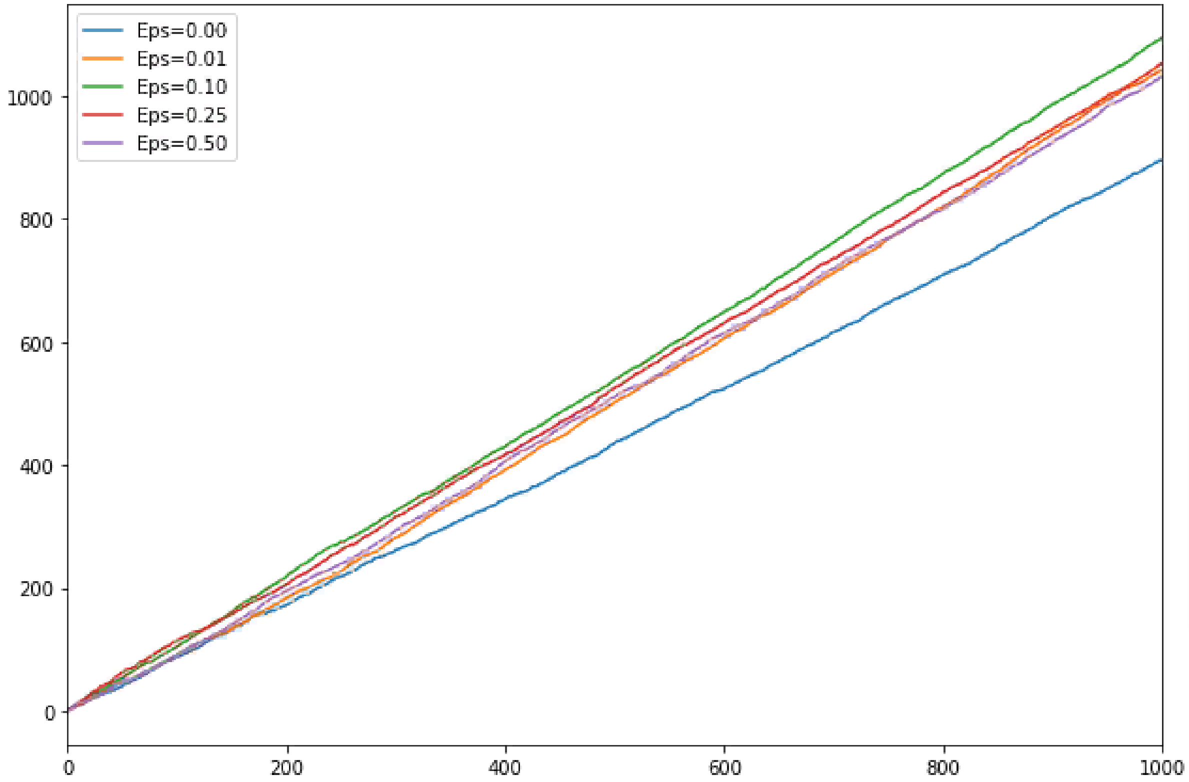
Как это делать?

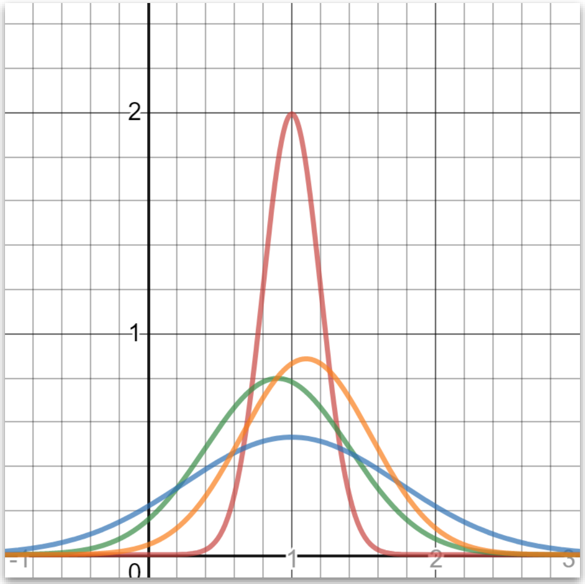
Простой ε-greedy алгоритм:

1. Инициализировать $Q(a) \leftarrow 0$
2. Инициализировать $N(a) \leftarrow 0$ // Счетчик количества применений действия
3. for t in range(T):
    1. a = argmax Q(a) с вероятностью 1 - ε, случайное действие с вероятностью ε
    2. r = step(a) // Совершаем действие и получаем награду
    3. c(a) += 1
    4. Q(a) = r / c(a) + (1 - 1 / c(a)) Q(a)   // Рекуррентная формула среднего

Epsilon-greedy exploration

Посмотрим на результат ε-greedy алгоритма:





Exploration via UCB sampling

Можно ли исследовать среду лучше, чем просто совершая случайные действия с определенной вероятностью?

Upper-confidence bound алгоритм предлагает повышать вероятность совершить действие, если мы совершаем его слишком редко:

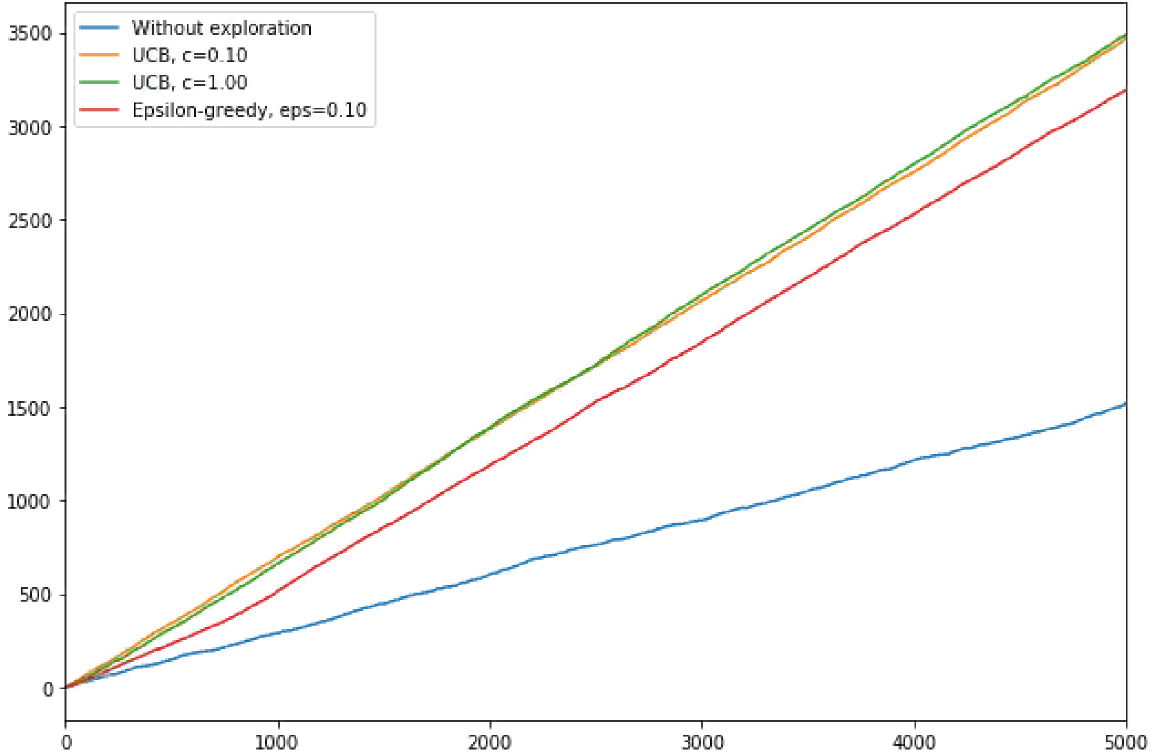
$$a = \arg\max Q(a) + c \cdot \sqrt{\frac{\ln t}{N(a)}}$$

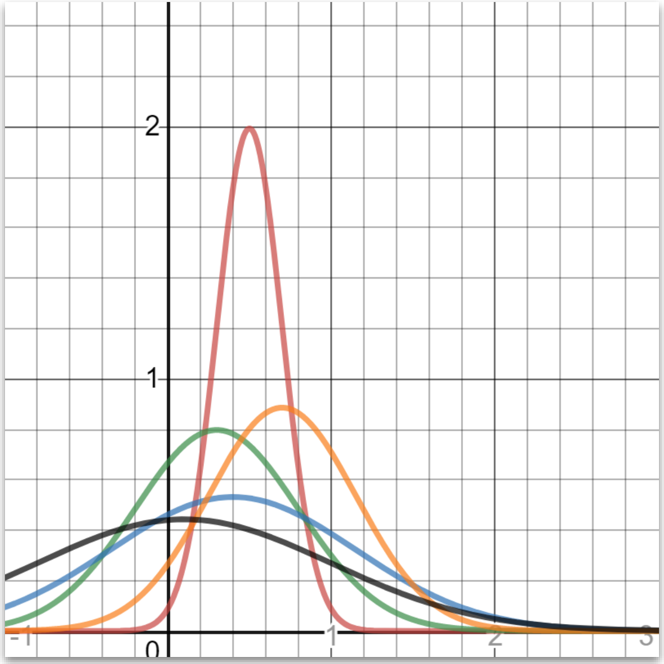
Свойства:

1) Совершим каждое действие хотя бы один раз (т.к. в начале N(a) = 0)
2) Со временем количество неоптимальных действий убывает
3) Можно регулировать то, насколько часто исследуем среду с помощью константы

Exploration via UCB sampling

Сравним два подхода:





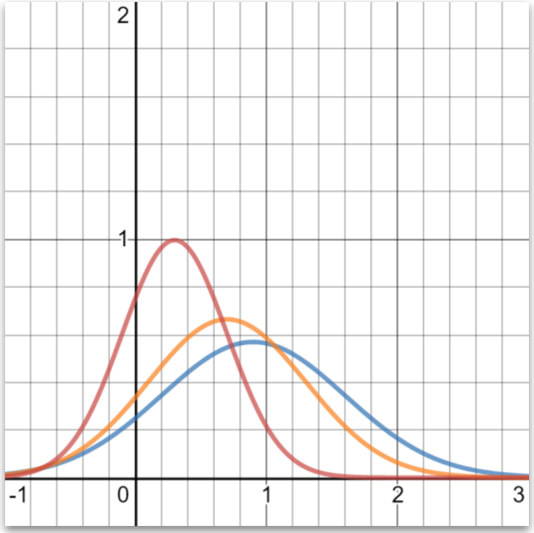
Contextual bandits

Предположим, что у распределение наград теперь зависит от некоторого состояния, которое известно в начале.

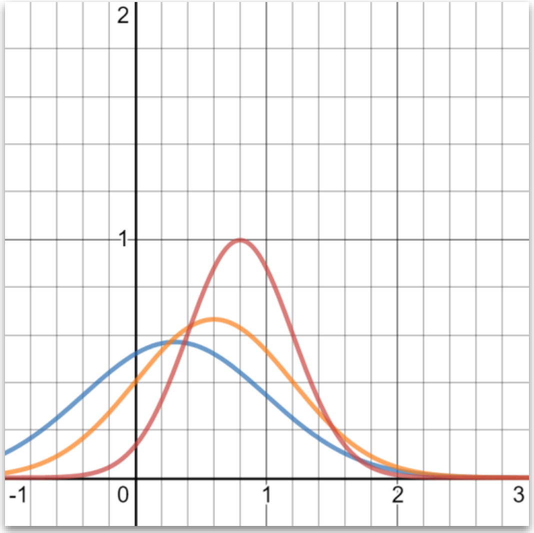
Старый подход работает плохо. Нужно учитывать **контекст**.

Можно просто приближать награду для каждого состояния независимо, но есть более сложные алгоритмы.

Состояние 1



Состояние 2



Применение контекстуальных бандитов

Artwork Personalization as Contextual Bandit

- **Environment**: Netflix homepage
- **Context**: Member, device, page, etc.
- **Learner**: Artwork selector for a show
- **Action**: Display specific image for show
- **Reward**: Member has positive engagement

Markov Decision Process

S - множество состояний
A - множество действий
T(s, a): S x A → S - функция перехода (может быть случайной величиной)
R(s, a, s'): S x A x S → R - функция награды (может быть случайной величиной)
D(s): S → {0, 1} - функция, которая определяет, закончился ли эпизод

Задача: максимизировать $\sum_{i=1}^{T} R(s_{i-1}, a_{i-1}, s_i)$

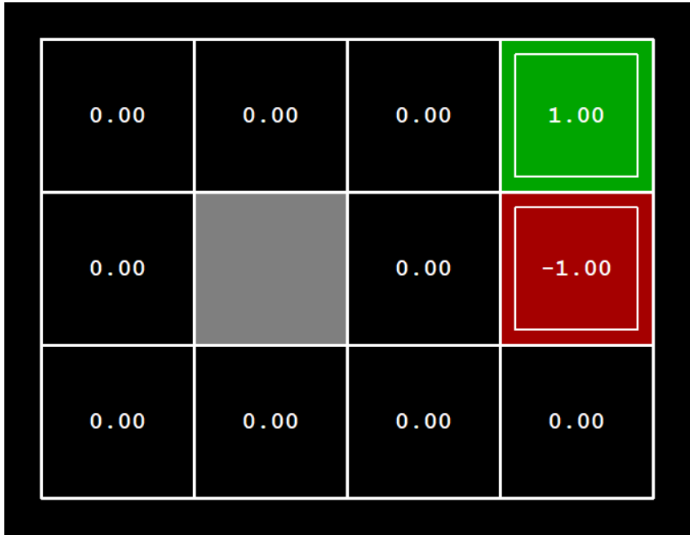
Марковский процесс принятия решений (MDP) - это набор (S, A, T, R, D). Чаще всего D присутствует неявно. В таком случае MDP определяется как (S, A, T, R)

Grid World

S: множество возможных положений на сетке
A: {идти вверх, идти налево, идти направо, идти вниз}
T(s, a): сместиться в нужную сторону если есть путь
R(s, a, s'): *+1* если в (4, 3), *-1* если в (4, 2), иначе *0*
D(s): 1 если совершаем любое действие в (4, 3) или в (4, 2), иначе 0

Игру начинаем в (1, 1)

Задача: максимизировать $\sum_{i=1}^{T} R(s_{i-1}, a_{i-1}, s_i)$



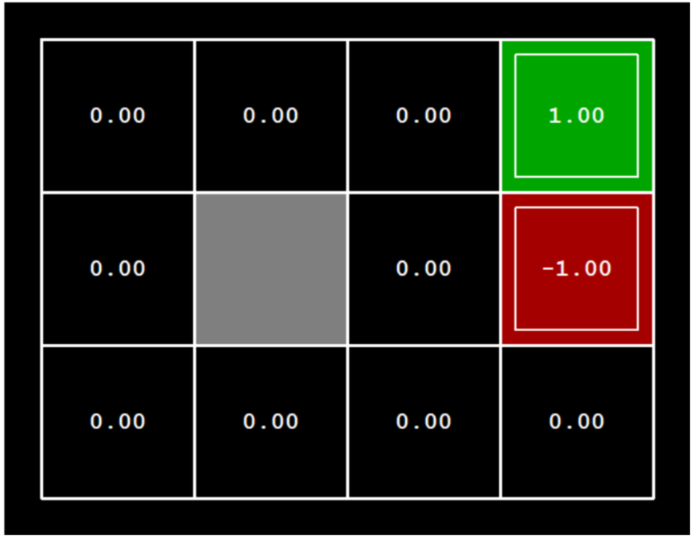
Markov Decision Process

S - множество состояний
A - множество действий
T(s, a): S x A → S - функция перехода (может быть случайной величиной)
R(s, a, s’): S x A x S → R - функция награды (может быть случайной величиной)
D(s): S → {0, 1} - функция, которая определяет, закончился ли эпизод

Задача: максимизировать $\sum_{i=1}^{T} R(s_{i-1}, a_{i-1}, s_i)$

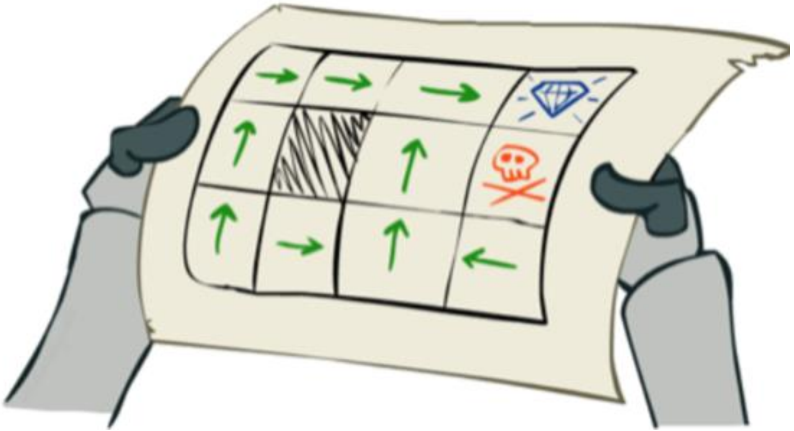
Проблема: можем решать бесконечно долго

Новая задача: $\sum_{i=0}^{T} \gamma^i R(s_i, a_i, s_{i+1})$, $0 < \gamma < 1$



Policy

π:



Политика/стратегия $\pi(s) : S \to A$ - правило, по которому агент принимает решения.

Оптимальная стратегия: $\pi^*(s) : \sum_{i=0}^{T} \gamma^i R(s_i, a_i, s_{i+1}) \underset{a}{\to} \max$

На самом деле хотим найти оптимальную политику!

Value Function

Policy Value Function определяет то, какую награду получит агент, который находится в заданном состоянии

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(s)}\mathbb{E}_{s'\sim T(s,a),r\sim R(s,a,s'))}[r + \gamma V_\pi(s')]$$

Действие политики

Ожидаемая суммарная награда

Реакция среды на действие

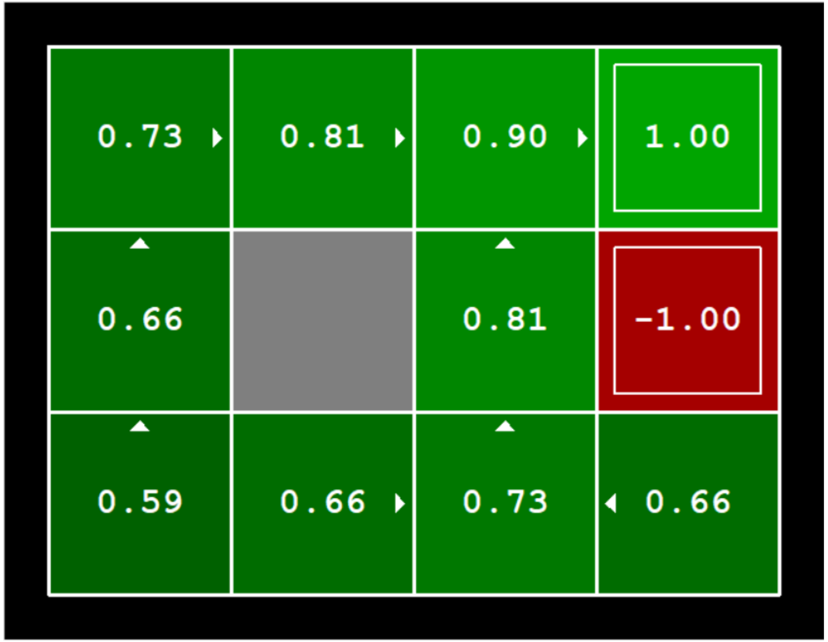
Value Function определяет то, насколько большую суммарную награду мы можем получить если находимся в определенном состоянии

$$V(s) = V_{\pi^*}(s) = \max_a \mathbb{E}_{s'\sim T(s,a),r\sim R(s,a,s')}[r + \gamma \cdot V_{\pi^*}(s')]$$

А оптимальную стратегию можно записать как:

$$\pi^*(s) = \arg\max_a \mathbb{E}_{s'\sim T(s,a),r\sim R(s,a,s')}[r + \gamma \cdot V_{\pi^*}(s')]$$

Value Function



Policy Evaluation

Дана политика π. Хотим посчитать value function для нее.

Алгоритм:

1. Инициализируем вектор V
2. Пока не сошлись:
    1. Для всех состояний s ∈ S:
        1. В стохастическом случае
           $$V_\pi(s) = \mathbb{E}_{a\sim\pi(s)}\mathbb{E}_{s'\sim T(s,a),r\sim R(s,a,s'))}[r + \gamma V_\pi(s')]$$
           В детерминированном случае
           $$V_\pi(s) = R(s,a,T(s,a)) + \gamma V_\pi(T(s,a)))$$

Policy Iteration

Алгоритм:

1. Пока политика улучшается:
    1. Получаем V с помощью алгоритма Policy Evaluation
    2. Для всех состояний s ∊ S:
        1. $\pi_{new}(s) = \arg\max_a \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')}[r + \gamma V_\pi(s')]$
        2. Если новое действие отличается от старого, то вернуться к шагу 1.1

Замечание 1:

В алгоритме Policy Evaluation можно инициализировать значение V ранее посчитанными значениями. Для многих сред это значительно ускорит алгоритм.

Замечание 2:

Полученная политика будет оптимальной.

Value Iteration

Алгоритм:

1. Инициализируем вектор значений функции **V**
2. В течении **T** шагов:
   1. Для каждого состояния среды **s**:

$$V(s) = \max_a \mathbb{E}_{s' \sim T(s,a), r \sim R(s,a,s')}[r + \gamma V_\pi(s')]$$

Замечание 1:

Политики в явном виде нигде нет. Ее можно получить с помощью argmax

Замечание 2:

Обычно мы не хотим считать мат. ожидание честно, поэтому давайте от него избавимся

Value Iteration

Алгоритм:

1. Инициализируем вектор значений функции **V**
2. В течении **T** шагов:
   1. Для каждого состояния среды **s**:

$$V(s) = (1 - \alpha) \cdot V(s) + \alpha \max_a [R(s, a, T(s, a))) + \gamma V_\pi(T(s, a)))]$$

Learning rate

Здесь используются семплы из распределения

Замечание 1:

Политики в явном виде нигде нет. Ее можно получить с помощью argmax

Замечание 2:

Теперь нет никакого мат. ожидания. В случае стохастической среды приближение может быть не точным, но алгоритм работает значительно быстрее.

Ограничения

Алгоритм:

1. Инициализируем вектор значений функции **V**
2. В течении **T** шагов:
   1. Для каждого состояния среды **s**:

$$V(s) = \underbrace{(1-\alpha)}_{\text{Learning rate}} \cdot V(s) + \underbrace{\alpha}_{} \max_a [\underbrace{R}_{}(s, a, \underbrace{T(s,a)}_{})) + \gamma V_\pi(\underbrace{T(s,a)}_{})))]$$

Learning rate

Здесь используются семплы из распределения

Ограничения:

1. Должны знать T и R
2. Должны уметь перебирать все состояния
3. Должны уметь перебирать все действия

Перерыв

Value Iteration

Алгоритм:

1. Инициализируем вектор значений функции **V**
2. В течении **T** шагов:
    1. Для каждого состояния среды **s**:

$$V(s) = (1-\alpha)\cdot V(s) + \alpha \max_a [R(s,a,T(s,a))) + \gamma V_\pi(T(s,a)))]$$

Ограничения:

1. Должны знать T и R
2. Должны уметь перебирать все состояния
3. Должны уметь перебирать все действия

First-visit Monte Carlo method

Дана политика π. Хотим посчитать value function для нее. Динамика среды не известна

Алгоритм:

1. Инициализируем вектор **V**
2. Инициализируем вектор массивов **G** пустыми массивами
3. В течении **N** итераций:
    1. Получаем эпизод $\boldsymbol{\tau}$ следуя политике $\boldsymbol{\pi}$
    2. Для каждого состояния **s** в $\boldsymbol{\tau}$:
        1. Добавить в **G[s]** суммарную дисконтированную награду для этого состояния, полученную начиная с первого посещения этого состояния
        2. **V[s] = mean(G[s])**

Value Function

Policy Value Function определяет то, какую награду получит агент, который находится в заданном состоянии

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(s)}\mathbb{E}_{s'\sim T(s,a),r\sim R(s,a,s')}[r+\gamma V_\pi(s')]$$

Действие политики

Ожидаемая суммарная награда

Реакция среды на действие

Value Function определяет то, насколько большую суммарную награду мы можем получить если находимся в определенном состоянии

$$V(s) = V_{\pi^*}(s) = \max_a \mathbb{E}_{s'\sim T(s,a),r\sim R(s,a,s')}[r+\gamma\cdot V_{\pi^*}(s')]$$

А оптимальную стратегию можно записать как:

$$\pi^*(s) = \arg\max_a \mathbb{E}_{s'\sim T(s,a),r\sim R(s,a,s')}[r+\gamma\cdot V_{\pi^*}(s')]$$

Quality Function

Policy Quality Function определяет то, какую награду получит агент, который находится в заданном состоянии при совершении определенного действия

$$Q_\pi(s,a) = \mathbb{E}_{s'\sim T(s,a), r\sim R(s,a,s')}[r + \gamma \mathbb{E}_{a'\sim\pi(s')} Q_\pi(s',a')]$$

Реакция среды на действие

Ожидаемая суммарная награда

Действие политики

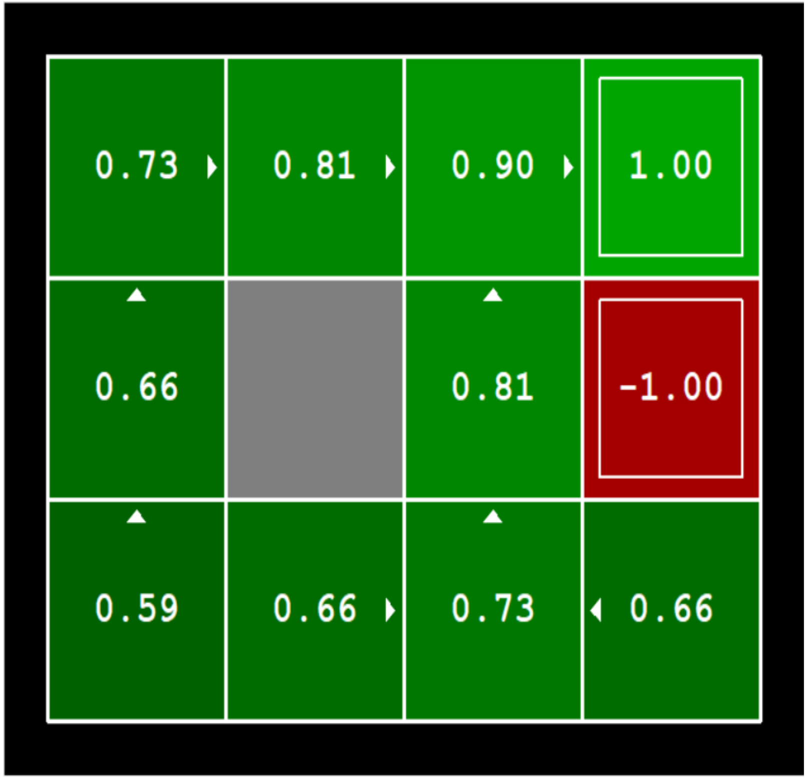
Quality Function определяет то, насколько большую суммарную награду мы можем получить если находимся в определенном состоянии совершив действие

$$Q(s,a) = Q_{\pi^*}(s,a)) = \mathbb{E}_{s'\sim T(s,a), r\sim R(s,a,s')}[r + \gamma \max_{a'} Q(s',a')]$$

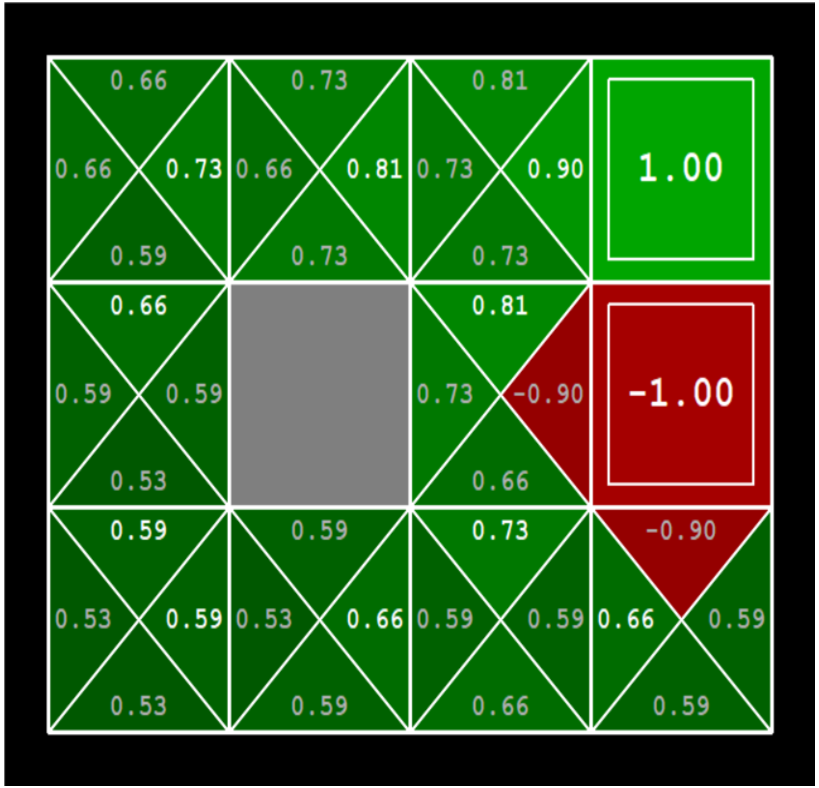
А оптимальную стратегию можно записать как:

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

Q-Function



0.73
0.81
0.90
1.00
0.66
0.81
-1.00
0.59
0.66
0.73
0.66



0.66
0.66
0.73
0.59
0.73
0.66
0.81
0.73
0.81
0.73
0.90
0.73
1.00
0.66
0.59
0.59
0.53
0.81
0.73
-0.90
0.66
-1.00
0.59
0.53
0.59
0.53
0.59
0.53
0.66
0.59
0.73
0.59
0.59
0.66
-0.90
0.66
0.59
0.59

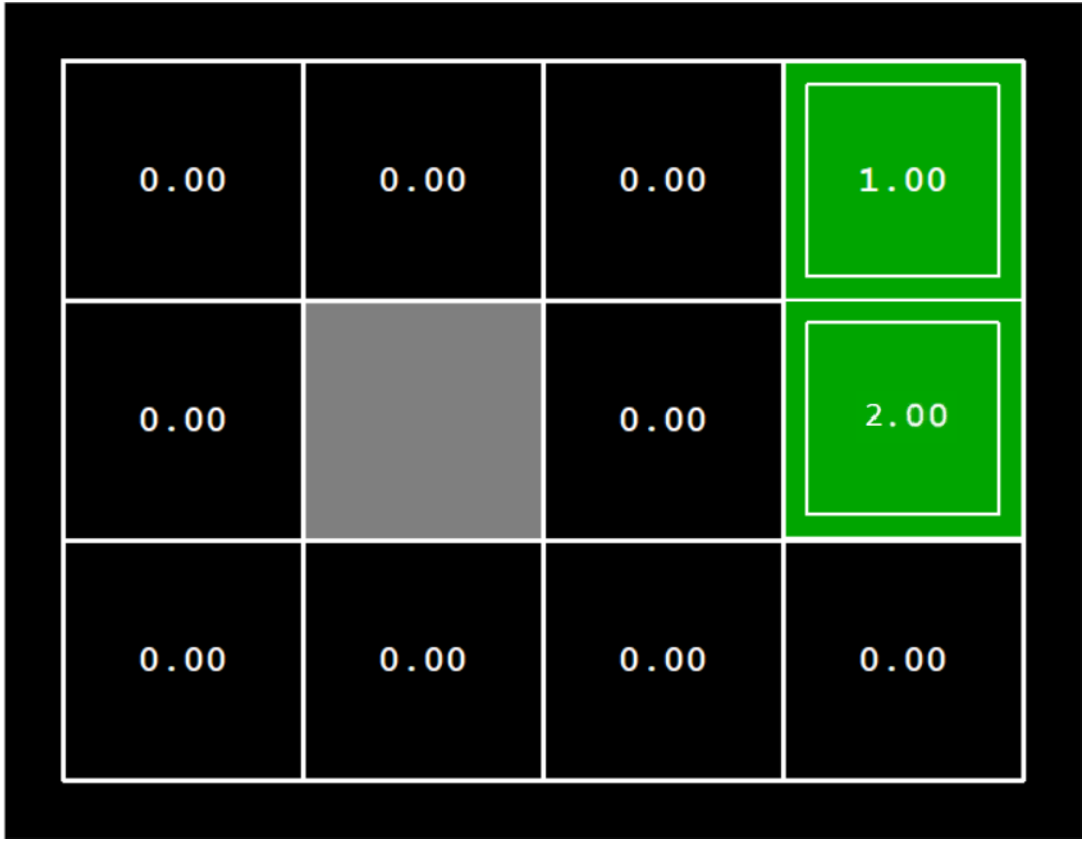
First-visit Monte Carlo method

Дана политика π. Хотим посчитать value function для нее. Динамика среды не известна

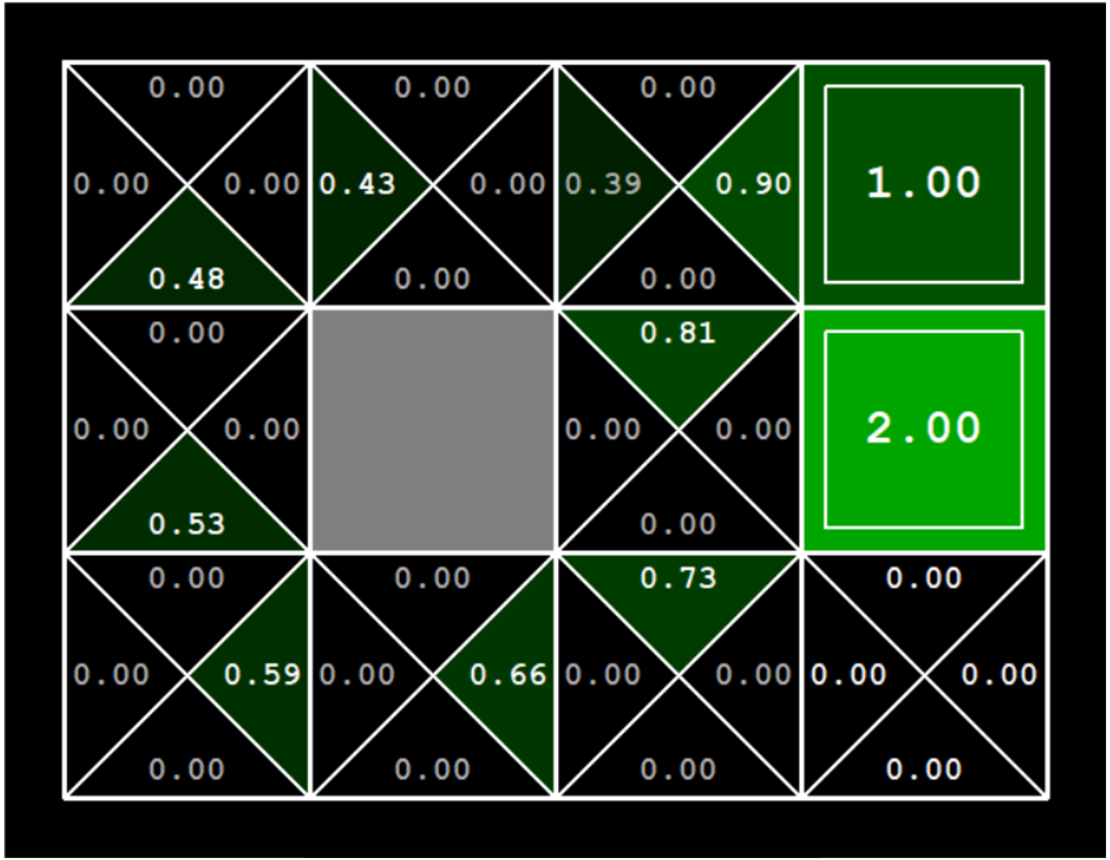
Алгоритм:

1. Инициализируем вектор **Q**
2. Инициализируем вектор массивов **G** пустыми массивами
3. В течении **N** итераций:
    1. Получаем эпизод $\boldsymbol{\tau}$ следуя политике $\boldsymbol{\pi}$
    2. Для каждого состояния **s** в $\boldsymbol{\tau}$:
        1. Добавить в **G[s, a]** суммарную дисконтированную награду для этого состояния, полученную начиная с первого посещения этого состояния
        2. **Q[s, a] = mean(G[s, a])**

Есть ли проблема?



Для большинства пар нет оценки



First-visit Monte Carlo method

Дана политика π. Хотим посчитать value function для нее. Динамика среды не известна

Алгоритм:

1. Инициализируем вектор **Q**
2. Инициализируем вектор массивов **G** пустыми массивами
3. В течении **N** итераций:
    1. Получаем эпизод $\boldsymbol{\tau}$ следуя политике $\boldsymbol{\pi}$
    2. Для каждой пары (**s, a**) в $\boldsymbol{\tau}$:
        1. Добавить в **G[s, a]** суммарную дисконтированную награду для этого состояния, полученную начиная с первого посещения этого состояния
        2. **Q[s, a] = mean(G[s, a])**

Предполагаем, что можем начать эпизод с любой парой **(s_0, a_0)** с вероятностью больше нуля, либо считаем, что политика имеет ненулевую вероятность достичь любой пары **(s, a)** (например, является epsilon-greedy)

Monte Carlo Control

Алгоритм:

1. Инициализируем вектор **Q**
2. Инициализируем вектор массивов **G** пустыми массивами
3. В течении **N** итераций:
    1. Получаем эпизод $\boldsymbol{\tau}$ следуя политике $\boldsymbol{\pi}$
    2. Для каждой пары (**s, a**) в $\boldsymbol{\tau}$:
        1. Добавить в **G[s, a]** суммарную дисконтированную награду для этого состояния, полученную начиная с первого посещения этого состояния
        2. **Q[s, a] = mean(G[s, a])**
    3. Для каждого состояния **s** в $\boldsymbol{\tau}$:
        1. Обновить “жадное” действие в политике $\boldsymbol{\pi}$

Замечание 1: Используем старые награды. Иногда так делать плохо

Замечание 2: Оценка Q-функции учитывает то, что наша политика epsilon-greedy

Q-learning

Алгоритм:

1. Инициализируем **Q(s, a)**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
   1. **a** := argmax **Q(s, .)**
   2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
      $$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \cdot \max_{a'} Q(s',a'))$$
   3. **s** := **s'** если не **d**, иначе реинициализируем среду

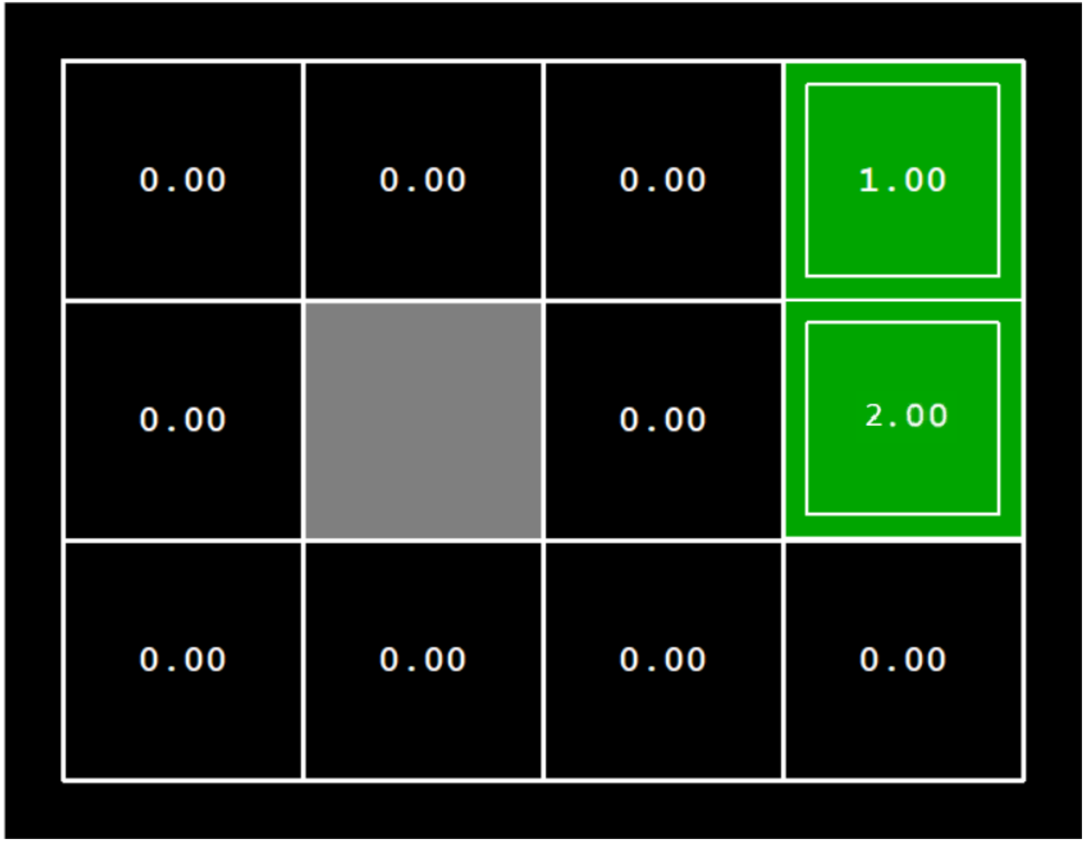
Рассмотрим 3.3 подробнее:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \cdot \max_{a'} Q(s',a'))$$

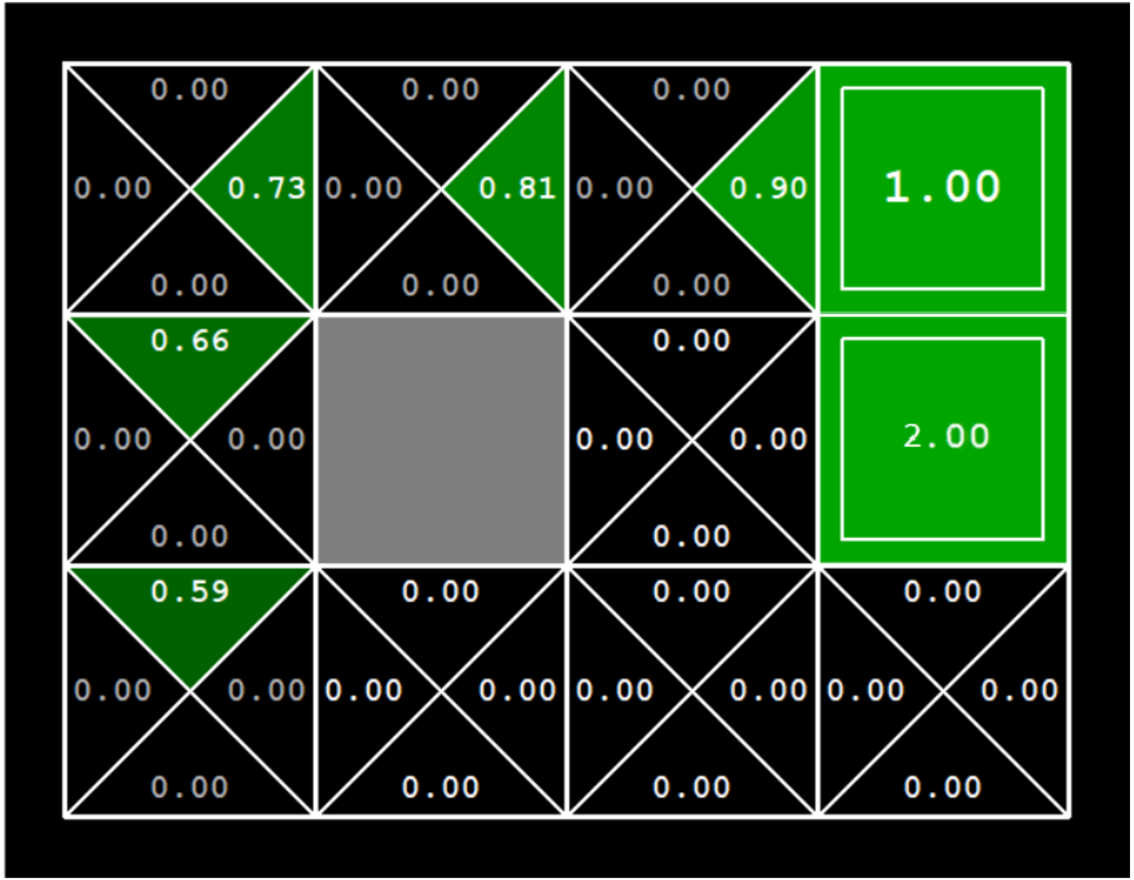
Learning rate

Получено из среды, знать R не нужно!

Есть ли в Q-learning проблема?



Снова не исследуем среду



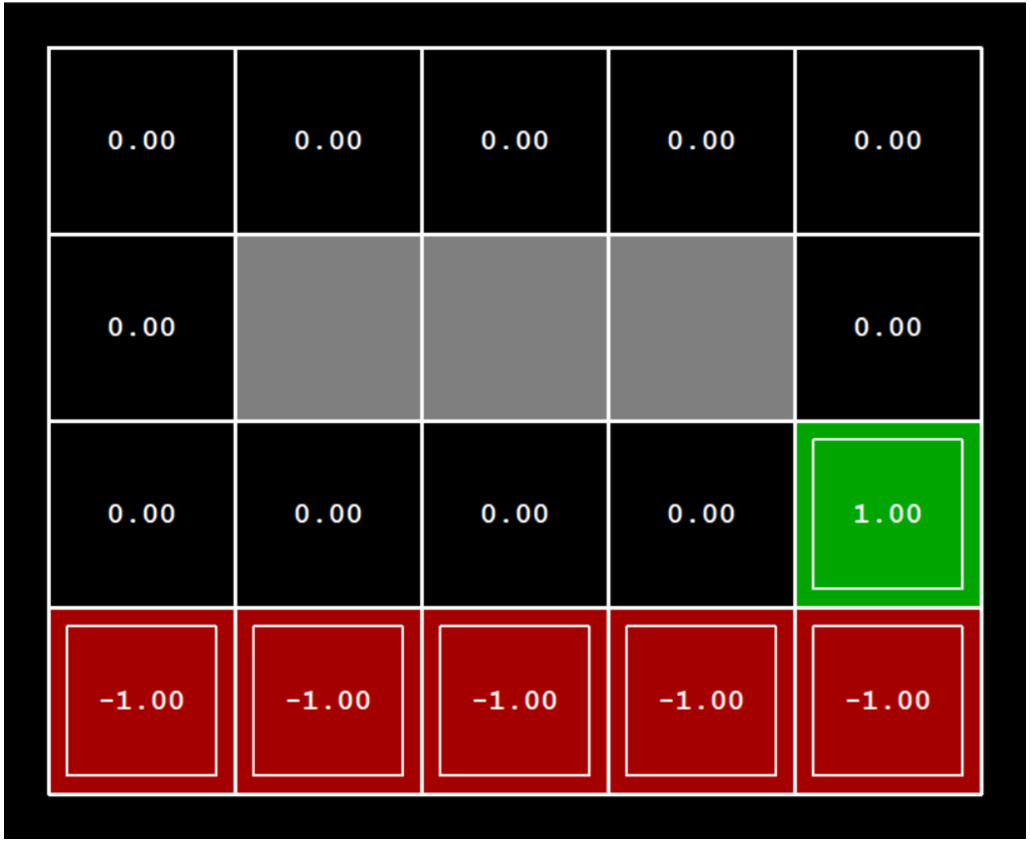
Epsilon-greedy Q-learning

Алгоритм:

1. Инициализируем **Q(s, a)**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
    1. **a** := argmax **Q(s, .)** с вероятностью 1-**eps**, иначе **a** := random(**A**)
    2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
       $$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \cdot \max_{a'} Q(s',a'))$$
    3. **s** := **s'** если не **d**, иначе реинициализируем среду

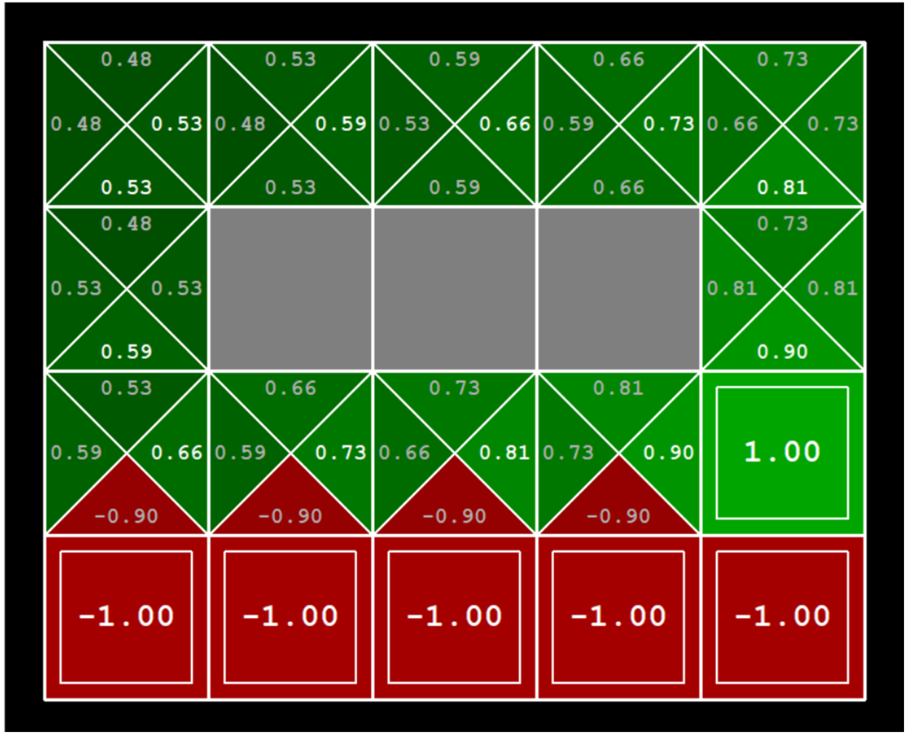
Замечание 1: Оценка Q не учитывает то, что политика epsilon-greedy

Cliff World

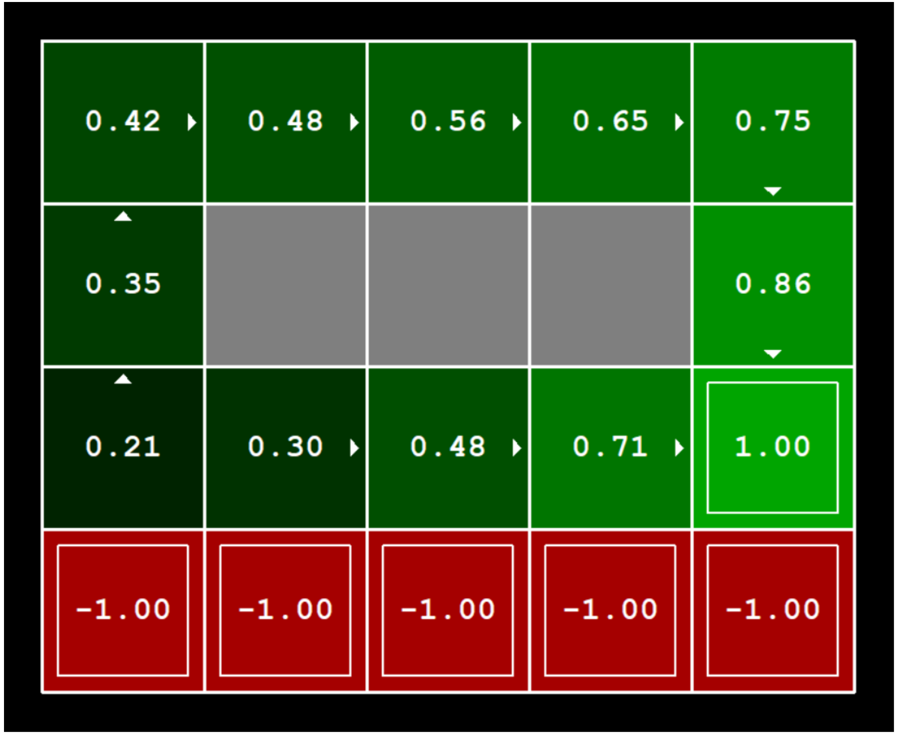

0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
1.00
-1.00
-1.00
-1.00
-1.00
-1.00

Cliff World

Что выучит Q-learning:



Оптимальная epsilon-greedy политика:



SARSA

Алгоритм:

1. Инициализируем **Q(s, a)**
2. Инициализируем среду, получаем начальное состояние **s**
3. В течении **T** шагов:
    1. **a** := argmax **Q(s, .)** с вероятностью 1-**eps**, иначе **a** := random(**A**)
    2. Совершаем действие **a**, получаем **s'**, **r**, **d** из среды
    3. $Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \cdot Q(s', a'))$, т.е. смотрим на реальное следующее действие
    4. **s** := **s'** если не **d**, иначе реинициализируем среду

On-policy VS Off-policy

On-policy
- При обучении агент оценивает реальную суммарную награду
- После обучения агент использует ту же политику, что и во время обучения
- Во время не использует дополнительных методов среды, весь exploration является частью агента

Примеры: **SARSA, MC Control**

$$r + \gamma \cdot Q(s', a')$$

Off-policy
- При обучении агент оценивает суммарную награду для оптимальной (жадной) политики
- После обучения агент использует жадную политику, убирая дополнительные exploration методы
- Во время обучения использует дополнительные методы исследования среды

Пример: **Q-learning**

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

Ограничения

1) ~~Должны знать T и R~~
2) Дискретное пространство состояний
3) Дискретное пространство действий

Пусть все оценки храним во float. Размер 4 байта. Пусть есть 16 Гб оперативной памяти. Сколько всего можем хранить пар состояние-действие?

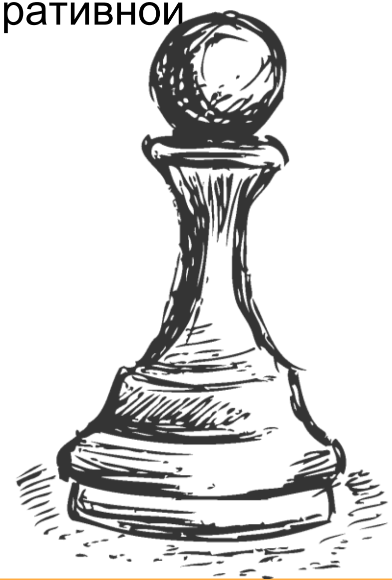
Ограничения

1) ~~Должны знать T и R~~
2) Дискретное пространство состояний
3) Дискретное пространство действий

Пусть все оценки храним во float. Размер 4 байта. Пусть есть 16 Гб оперативной памяти. Сколько всего можем хранить пар состояние-действие?

$$\frac{16 \cdot 2^{30}}{4} = 4 \cdot 2^{30} \approx 4 \cdot 10^9$$

Сколько всего возможных позиций в игре в шахматы?



Ограничения

1) ~~Должны знать T и R~~
2) Дискретное пространство состояний
3) Дискретное пространство действий

Пусть все оценки храним во float. Размер 4 байта. Пусть есть 16 Гб оперативной памяти. Сколько всего можем хранить пар состояние-действие?

$$\frac{16 \cdot 2^{30}}{4} = 4 \cdot 2^{30} \approx 4 \cdot 10^9$$

Сколько всего возможных позиций в игре в шахматы?

Оценка сверху (число Шеннона): 10^{43}

После первых 10 ходов (точное число): $69352859712417 \approx 69 \cdot 10^{12}$