Обучение с подкреплением: определения и примеры

План

- Обучение с подкреплением: определения и примеры
- Постановка задачи: вознаграждение, состояние, обозрение
- Схема RL-агента: стратегия, полезность, модель
- Подзадачи в обучении с подкреплением
- Общая информация по курсу

ИИ и обучение с подкреплением

ИИ - это наука и технология создания интеллектуальных машин, особенно интеллектуальных компьютерных программ (**Джон Маккарти**)

ИИ - это наука об «интеллектуальных агентах», т.е. о некотором устройстве или программе, которая воспринимает свою среду и выполняет действия, которые максимизируют ее шансы на успех при достижении какой-то цели (**Стюарт Рассел, Питер Норвиг**)

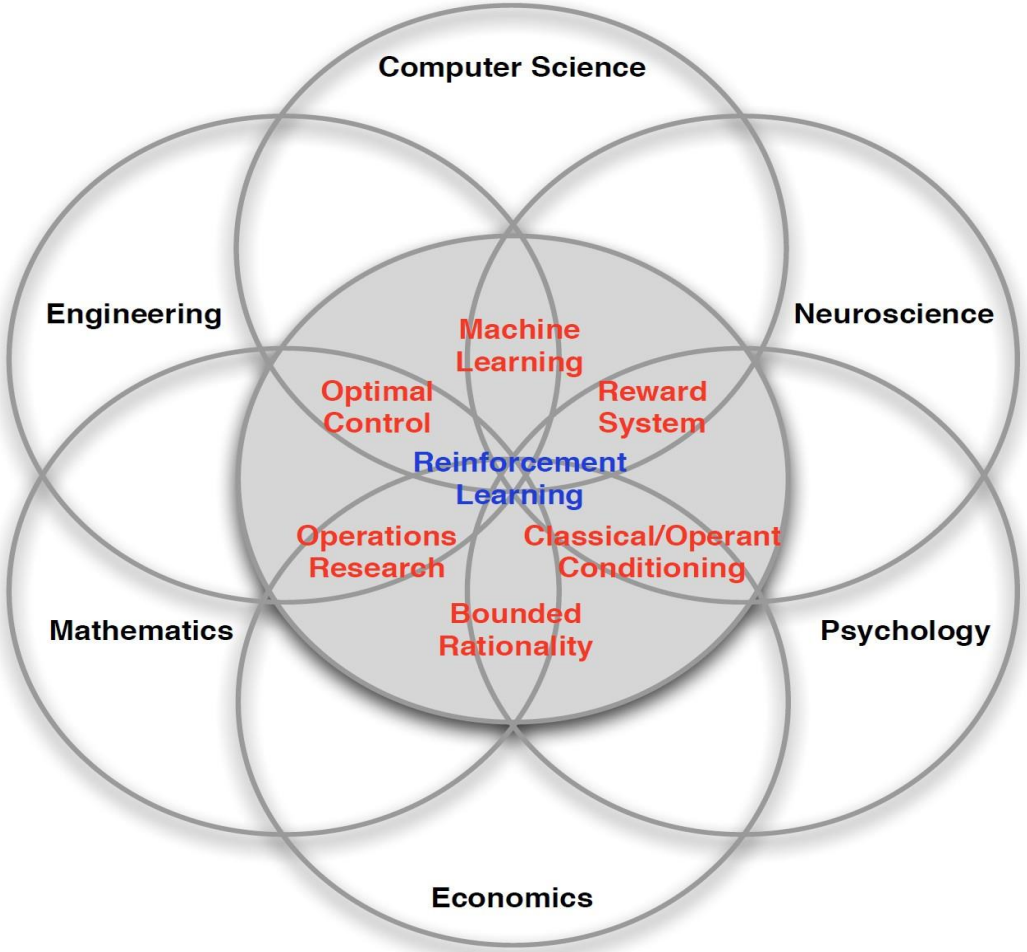
ИИ - это научное направление, в рамках которого ставятся и решаются задачи аппаратного или программного моделирования тех видов человеческой деятельности, которые традиционно считаются интеллектуальными (**Д.А. Поспелов)**

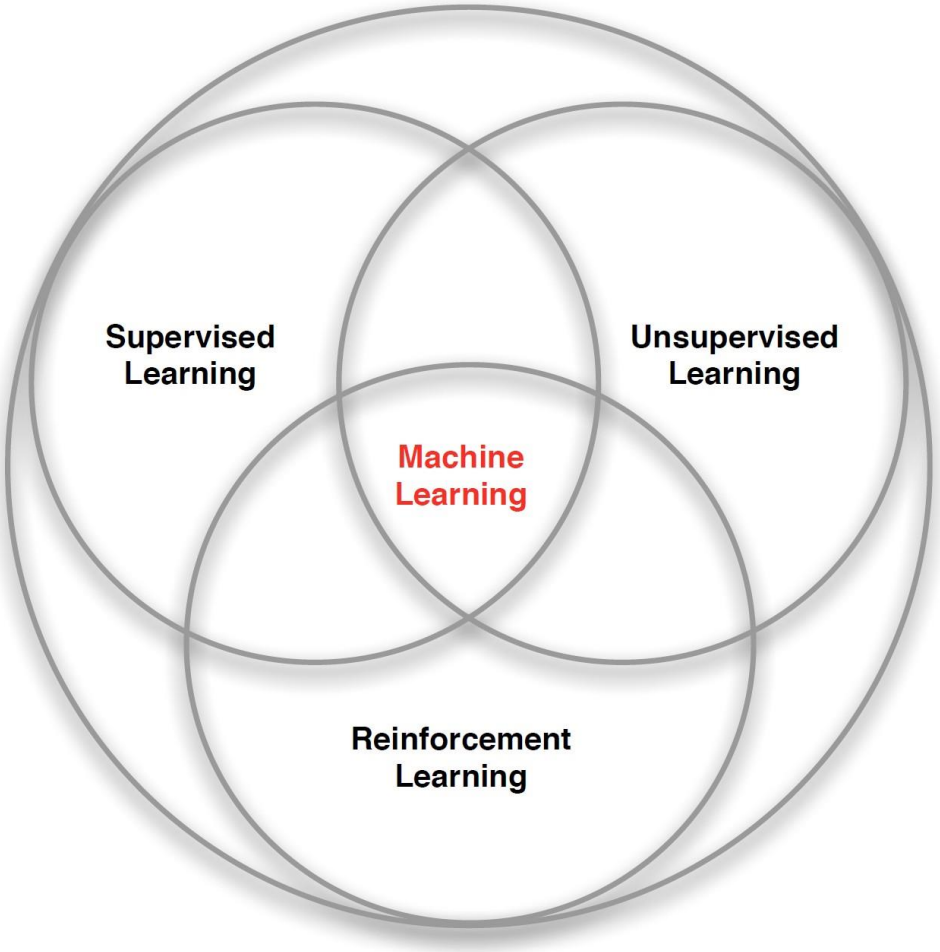
Методы искусственного интеллекта применяются к тем задачам, алгоритм решения которых заранее неизвестен. Этот алгоритм является одним из результатов работы методов ИИ (**Г.С. Осипов**)

Обучение с подкреплением

Обучение интеллектуального агента хорошей последовательности принятия решения в условиях неполной информации

Место обучения с подкреплением





Обучение с подкреплением: определения и примеры

Отделение среды (**environment**) и агенты (**agent**) – источник и акцептор данных в явном виде присутствуют в постановке задачи.

Нет учителя, т.е. ошибка не задается явно, а косвенно передается через вознаграждение (**reward**).

Обратная связь (**feedback**) от среды может поступать с задержкой (**delayed**).

Параметр времени имеет особое значение – последовательные (**sequential**) данные.

Действия агента влияют на поступающие в дальнейшем данные.

Примеры обучения с подкреплением

AlphaGo – игра в Go лучше человека

Игра в игры Atari, Starcraft, Dota, MineRL – иногда лучше человека

Управление движением мобильного и человекоподобного робота

Управление энергетической станцией

Управление инвестиционным пакетом

Автоматизация «холодных звонков»

Реализация сложных движений вертолета

Примеры: автономный вертолет

Видео полета вертолета, обученного по демонстрациям

Autonomous Helicopter Aerobatics through Apprenticeship Learning, Pieter Abbeel, Adam Coates, and Andrew Y. Ng. IJRR, 2010, http://heli.stanford.edu/

Примеры: автономный вертолет

Обучение игре в аркадные игры по изображениям

Playing Atari with Deep Reinforcement Learning, OpenAI, 2018, http://arxiv.org/abs/1808.00177

Примеры: автономный вертолет

Видео работы обученного манипулятора со многими степенями свободы

Learning Dexterous In-Hand Manipulation, OpenAI, 2018, http://arxiv.org/abs/1808.00177

Примеры: автономный вертолет

Видео обучения в мультиагентной среде

Emergent complexity via multi-agent competition, Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch, ICML 2018, http://arxiv.org/abs/1710.03748

Примеры: автономный вертолет

Видео обучения коллективным действиям

Emergent tool use from multi-agent autocurricula, B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, 2019, http://arxiv.org/abs/1909.07528

Примеры: автономный вертолет

Видео с достижением цели наземным роботом

Rapid Exploration for Open-World Navigation with Latent Goal Models,D. Shah, B. Eysenbach, N. Rhinehart, S. Levine, 2021, https://arxiv.org/abs/2104.05859

Постановка задачи: вознаграждение, состояние, наблюдаемость

Обучение с подкреплением: определения и примеры

Вознаграждение (reward) r_t – скалярный сигнал обратной связи. Показывает насколько агент был успешен на шаге t.

Задача агента – максимизировать суммарное вознаграждение.

В обучении с подкреплением принята следующая гипотеза:

Все цели могут быть описаны через максимизацию суммарного вознаграждения.

Примеры вознаграждений

AlphaGo: $+r$ за победу в партии, $-r$ за проигрыш.

Atari: $+r$ за увеличение счета, $-r$ за уменьшение счета.

Робот: $+r$ за прямой шаг, $-r$ за падение.

Энергетическая станция: $+r$ за генерацию энергии, $-r$ за превышение ограничений безопасности.

Инвестиционный пакет: $+r$ за каждый заработанный рубль, $-r$ за каждый потерянный рубль.

Автоматизация «холодных звонков»: $+r$ за привлечение клиента, $-r$ плата за звонок.

Вертолет: $+r$ за следование намеченной траектории, $-r$ за падение.

Последовательное принятие решений

Цель – выбор действий, максимизирующих суммарное будущее вознаграждение.

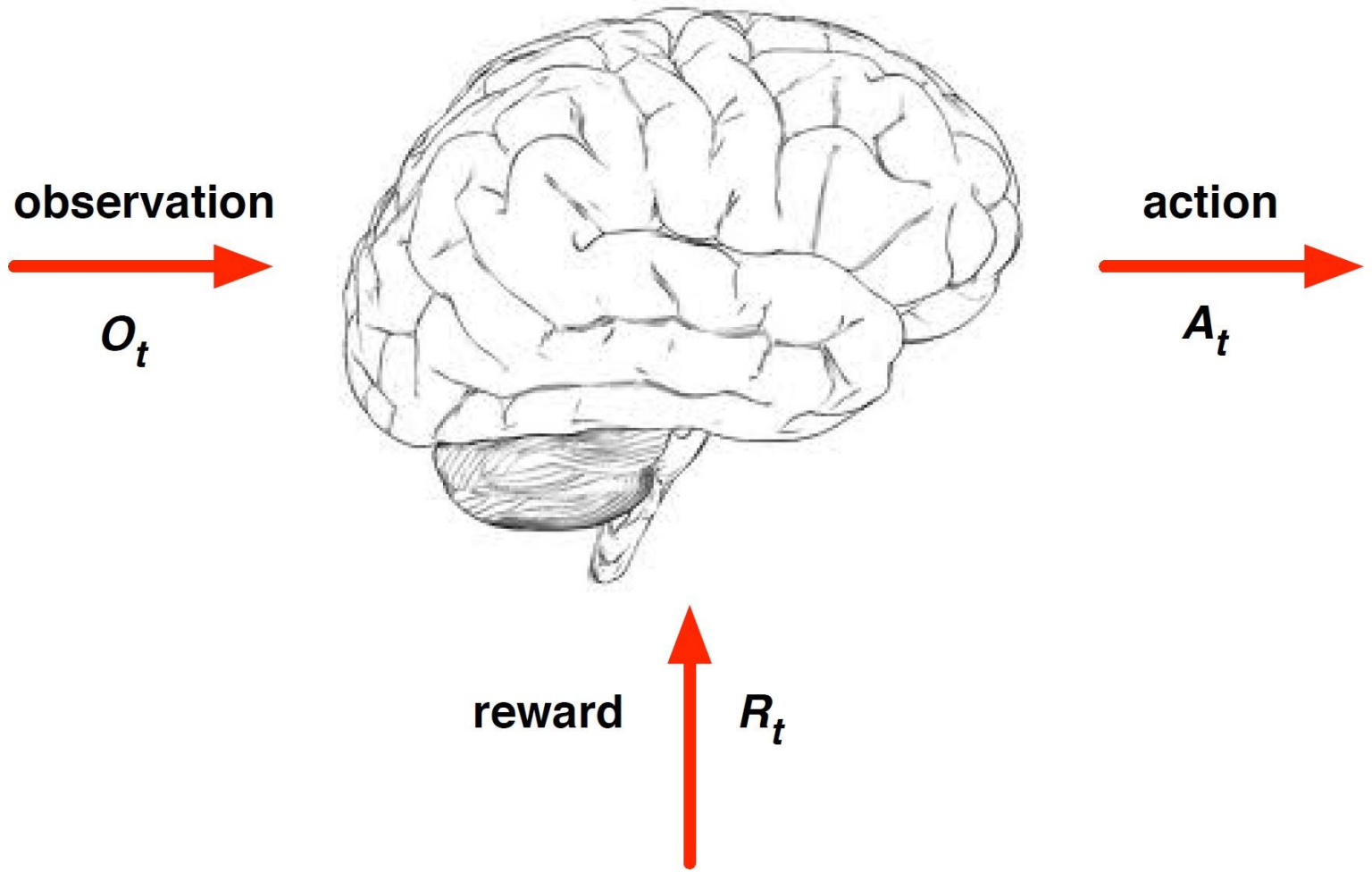
Действия могут иметь долговременные (long term) последствия. Вознаграждения могут быть отложенными.

Иногда выгоднее пренебречь немедленным вознаграждением для получения большего долгосрочного вознаграждения.

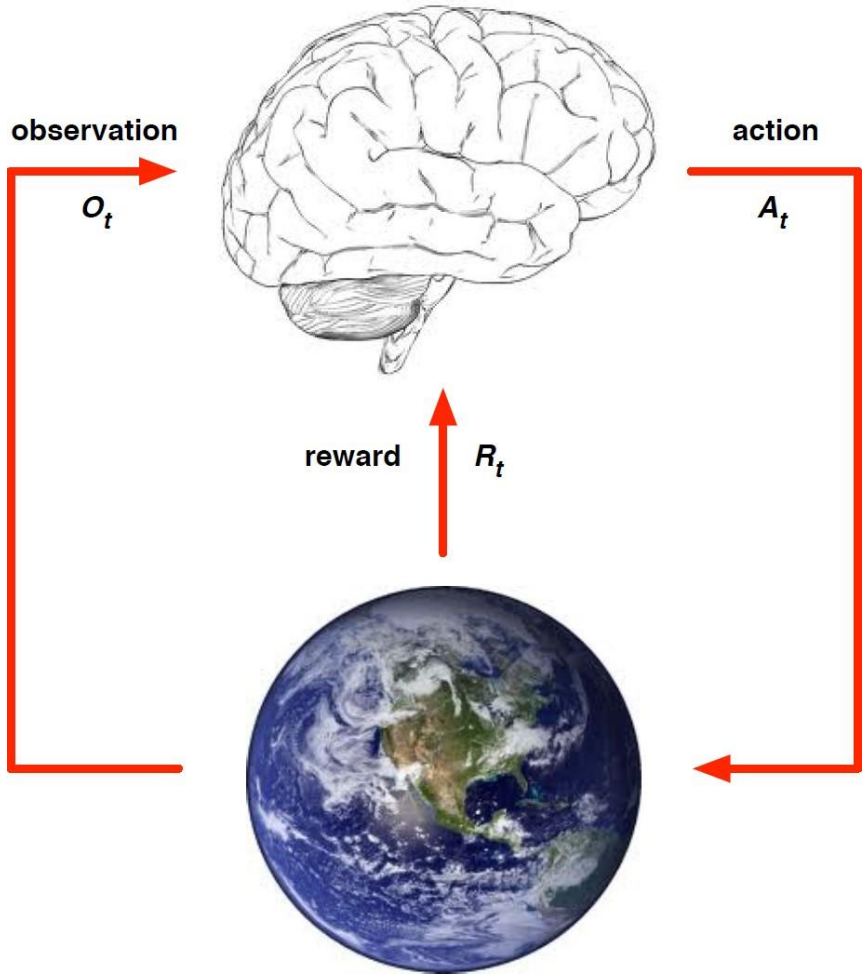
Примеры:

- Инвестирование – последствия решений могут проявиться через несколько месяцев.
- Заправка вертолета может предотвратить его падение в следующие несколько часов.
- Блокирование хода противника может помочь выиграть спустя много ходов.

Взаимодействие агенты и среды



Взаимодействие агенты и среды



В каждый момент времени t агент:

- воспринимает наблюдение o_t,
- выполняет действие a_t,
- получает скалярное вознаграждение r_{t+1}.

В каждый момент времени t среда:

- реагирует на действие a_t,
- генерирует следующее наблюдение o_{t+1},
- генерирует скалярное вознаграждение r_{t+1}.

Переход к шагу $t + 1$.

История и состояние

История (взаимодействия) это последовательность наблюдений, действий и вознаграждений:

$$H_t = \langle o_1, r_1, a_1, \ldots, a_{t-1}, o_t, r_t \rangle$$

Например: все наблюдаемые переменные к моменту времени t
или сенсомоторный поток робота.

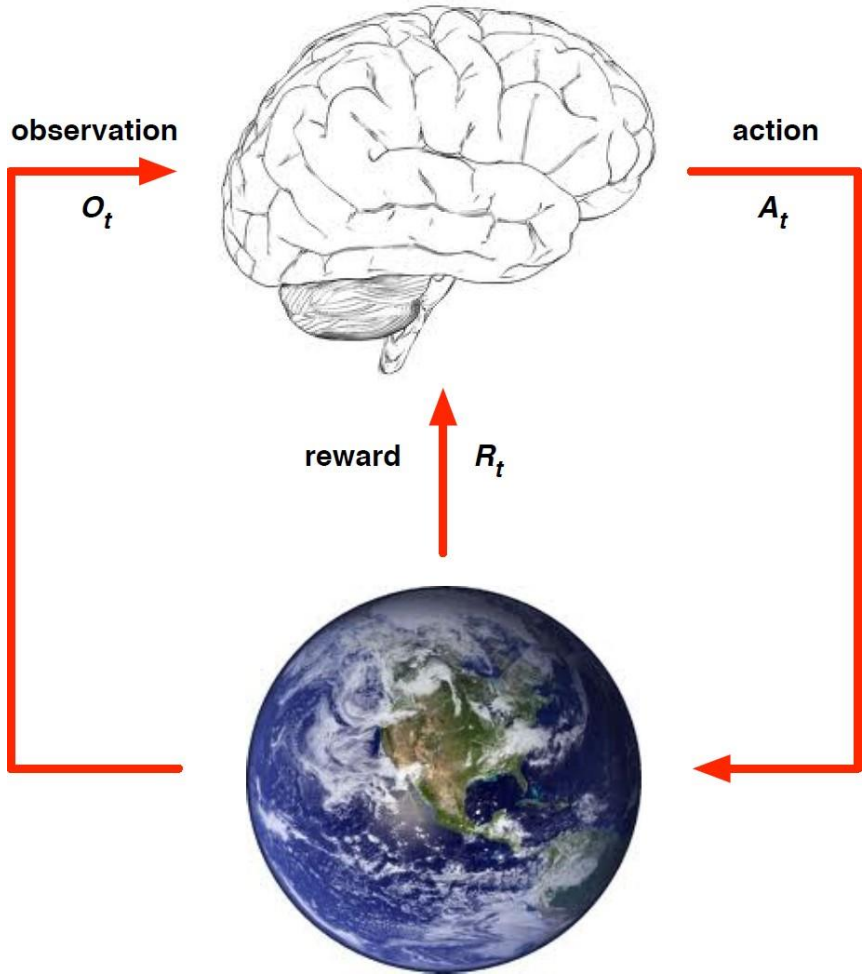
То, что случится далее зависит от истории:

- агент выберет действие,
- среда сгенерирует наблюдение и вознаграждение.

Состояние (state) – это информация используемая для определения того, что произойдет далее:

$$s_t = f(H_t)$$

Взаимодействие агенты и среды



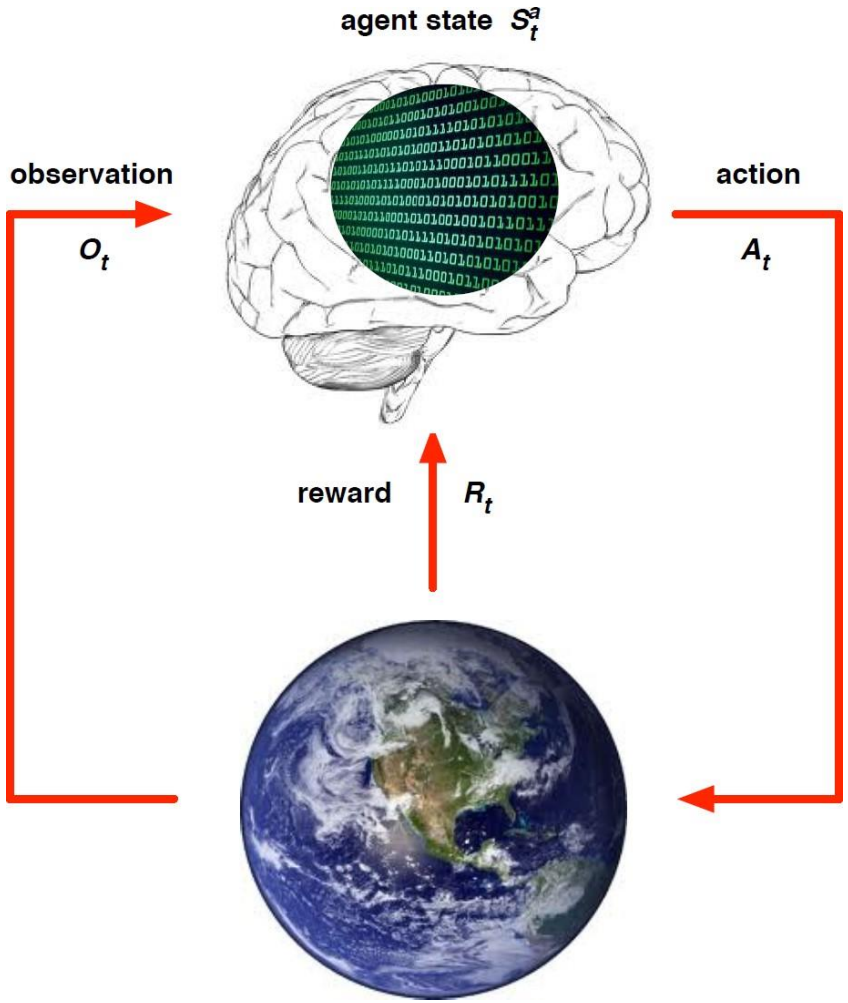
Состояние среды *s(e,t)* – это внутреннее представление информации в среде.

Пример: любые данные, которые используются для выдачи следующего наблюдения и вознаграждения.

Состояние среды обычно не наблюдается напрямую агентом.

Даже если состояние *s(e,t)* доступно, оно может содержать некорректную информацию.

Взаимодействие агенты и среды



Состояние агента *s(a,t)* – это внутреннее представление информации агентом.

Пример: любые данные, которые используются агентом для выдачи следующего действия.

Состояние агента обычно является функцией от истории:

$$s(a,t) = f(H_t)$$

Марковское состояние

Марковское (или *информационное*) состояние содержит всю необходимую информацию, которая может иметься в истории.

Definition: Состояние s_t называется марковским, если и только если

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_1, \ldots, s_t]$$

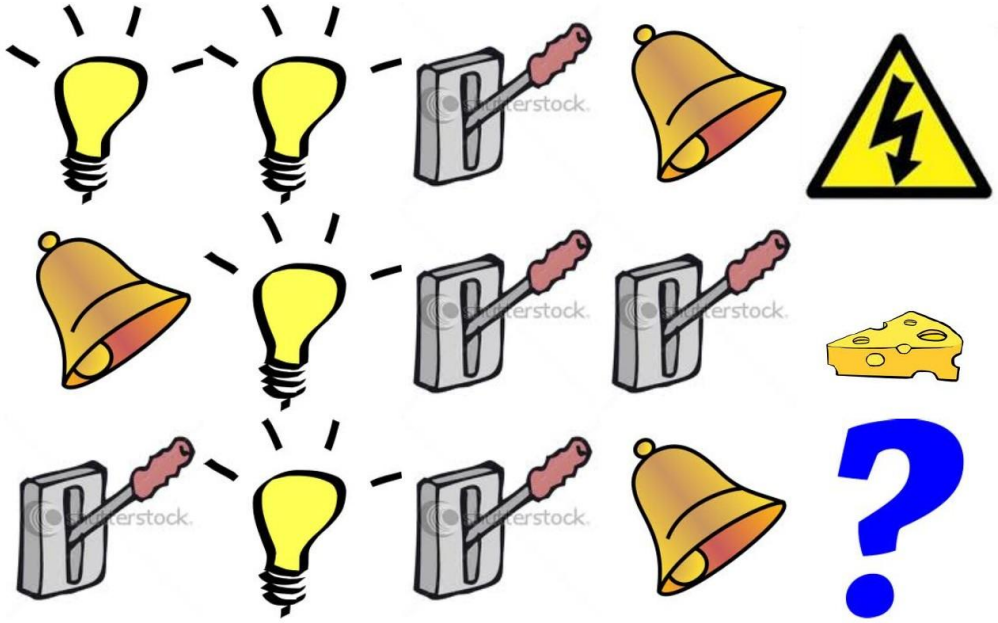
Иными словами, будущее не зависит от прошлого и определяется только настоящим:

$$H_{1:t} \rightarrow s_t \rightarrow H_{t+1:\infty}$$

Если известно текущее состояние, история может быть отброшена.
Текущее состояние содержит всю необходимую статистику о будущем.

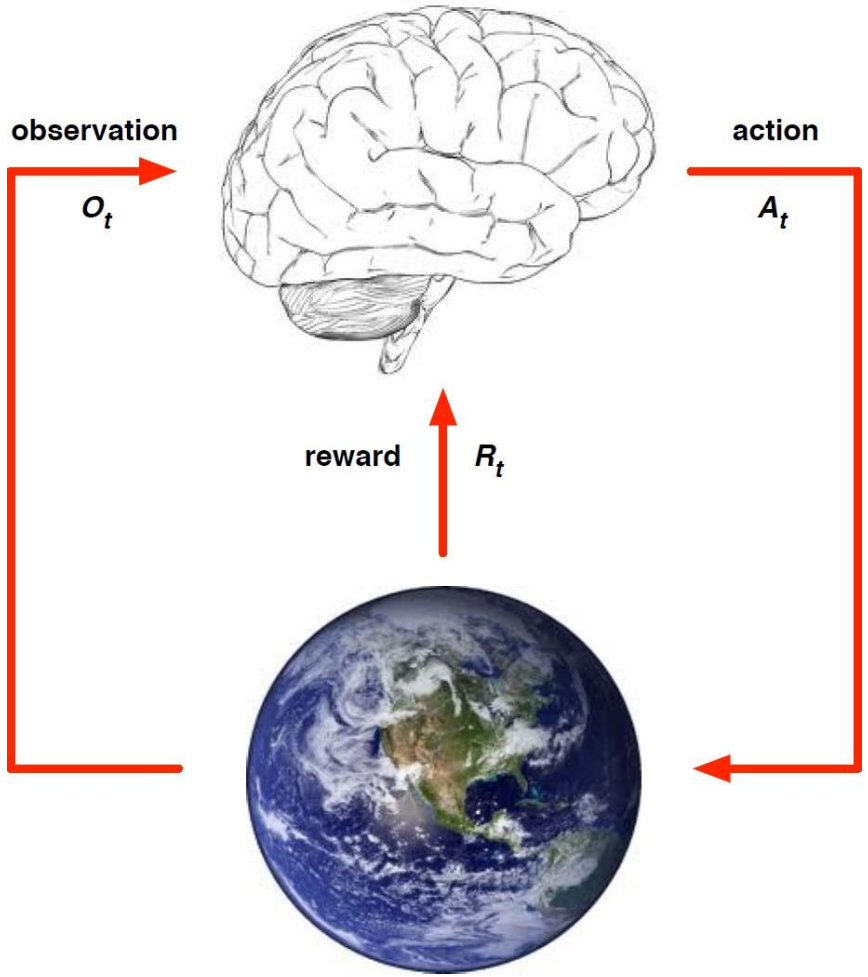
Предполагаем, что состояние среды *s(e,t)* является марковским.

Взаимодействие агенты и среды



s(a,t) = последние три события в последовательности,
s(a,t) = количество появлений света, звонка и рычага,
s(a,t) = вся последовательность,

Полностью наблюдаемые среды



Полная наблюдаемость: агенту напрямую доступно состояние среды:

$o_t = s(a,t) = s(e,t)$

Состояние агента = состояние среды = информационное состояние.

Формально такой случай взаимодействия называется *марковским процессом принятия решений* (МПР, Markov decision process, MDP).

Частично наблюдаемые среды

Частичная наблюдаемость, агент наблюдает среду опосредованно:

- робот через видеокамеру не может определить своего точного местоположения,
- торговый агент может наблюдать только текущие цены,
- игроку в покер доступны только открытые всем карты.

В этом случае состояние агента != состояние среды.

Формально – это *частично наблюдаемый марковский процесс принятия решений* (partially observable Markov decision process, POMDP).

Агент должен сформировать свое собственное представление $s(a,t)$:

- полная история: $s(a,t) = H_t$,
- *представление* о состоянии среды: $s(a,t) = (P[s(e,t) = s^1], \ldots, P[s(e,t) = s^n])$,
- рекуррентная нейронная сеть: $s(a,t) = \sigma(s(a,t-1) w_s + o_t w_o)$.

Схема RL-агента: стратегия, полезность, модель

Строение RL агента

Определение любого RL-агент обычно включает одну или несколько следующих составляющих:

Стратегия (policy): функция поведения агента.

Функция полезности (value function): оценка того, насколько полезно состояние и/или действие.

Модель (model): представление агента о среде.

Стратегия

Стратегия – это функция поведения агента. Обычно это отображение состояния в действие.

Примеры:

Детерминированная стратегия: $a = \pi(s)$. Стохастическая стратегия: $\pi(a|s) = \mathrm{P}[a_t = a|s_t = s]$.

Функция полезности

Функция полезности – это предсказание будущего вознаграждения и используется для оценки состояний.

Другими словами, по функции полезности происходит отбор действий:

$$V^{\pi} = \mathrm{E}_{\pi}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s]$$

Модель

Модель служит для предсказания того, что произойдет в среде в следующий момент времени.

Примеры:

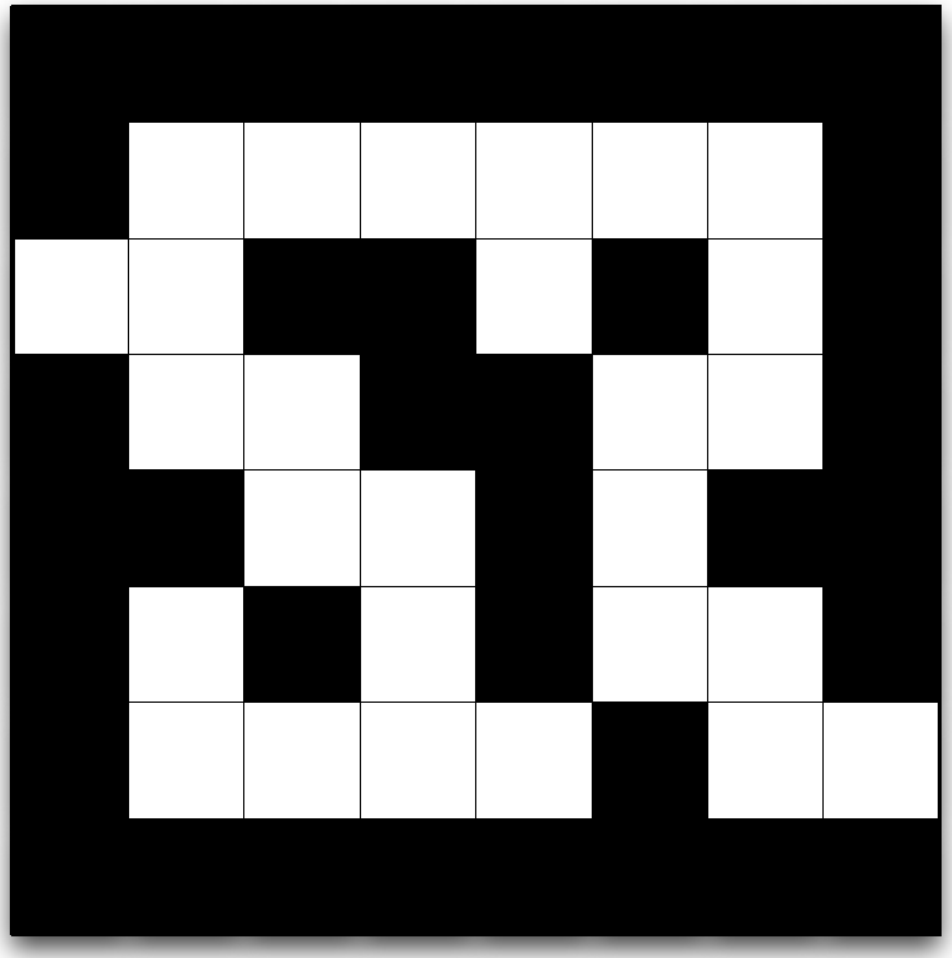
модель переходов P предсказывает следующее состояние:

$$\mathrm{P}(\mathrm{s},\mathrm{a},\mathrm{s}') = \mathrm{P}[s_{t+1} = s' \mid s_t = s, a_t = a]$$

модель вознаграждений R предсказывает следующее вознаграждение:

$$\mathrm{R}(\mathrm{s},\mathrm{a}) = \mathrm{E}[r_t \mid s_t = s, a_t = a].$$

Лабиринт

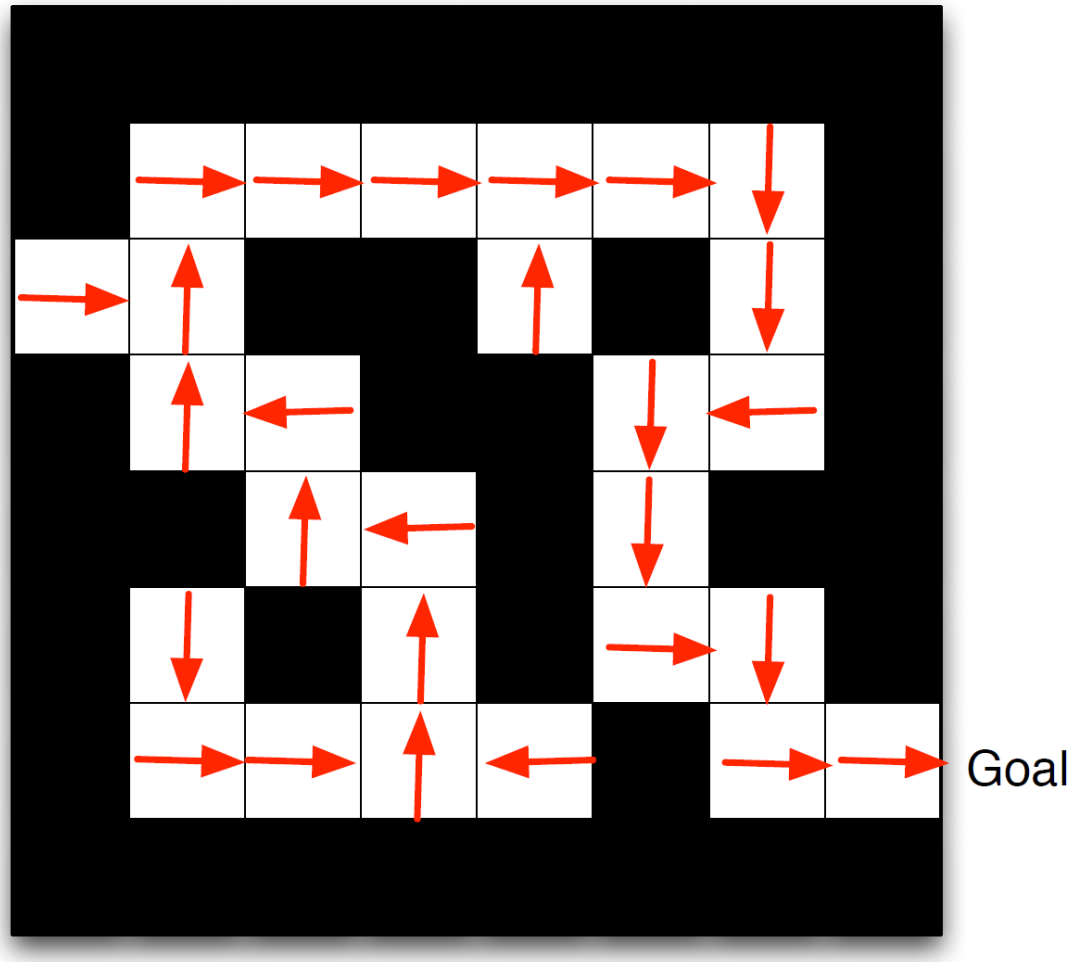


Вознаграждения: -1 за каждый шаг.

Действия: $\uparrow, \leftarrow, \rightarrow, \downarrow$.

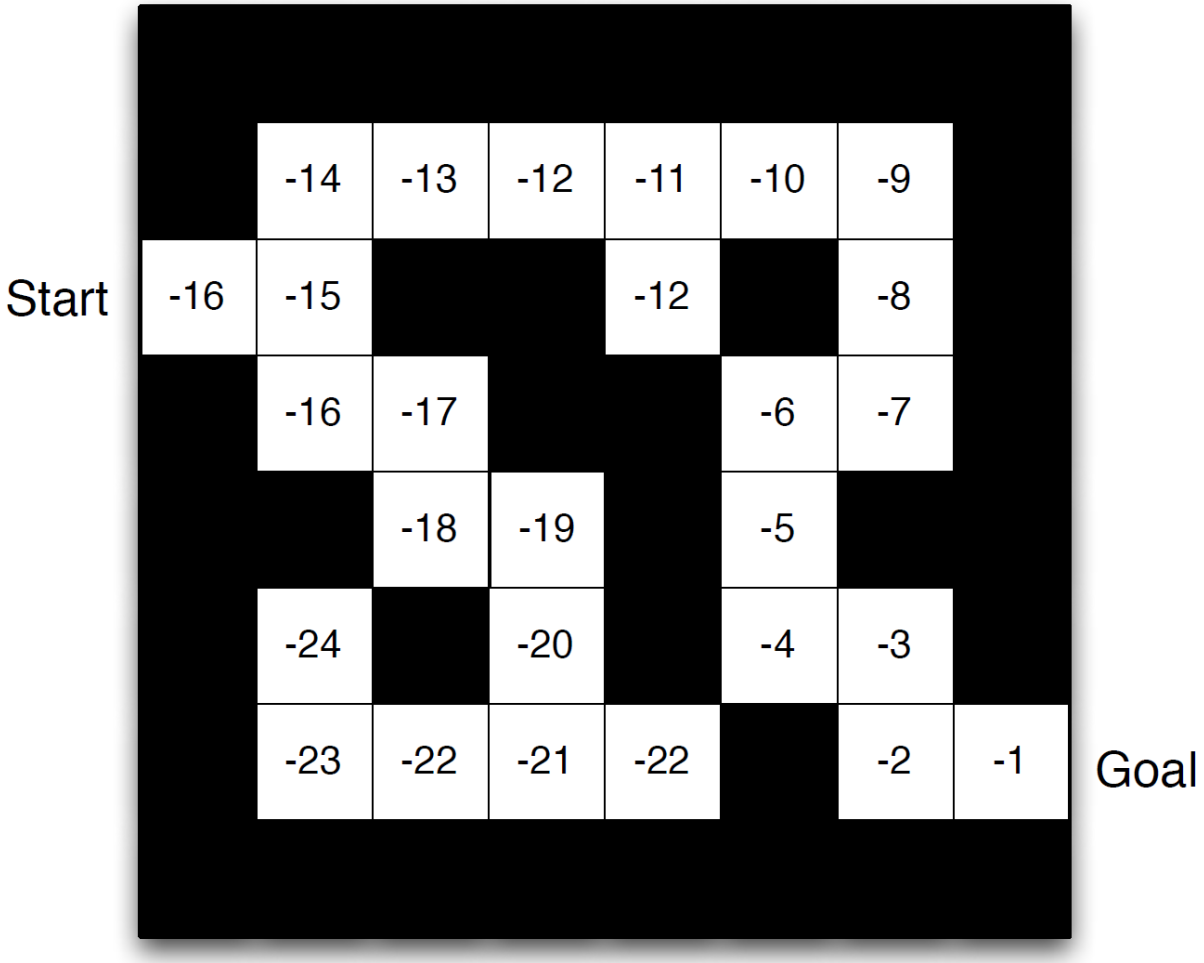
Состояния: местоположение агента.

Лабиринт: стратегия



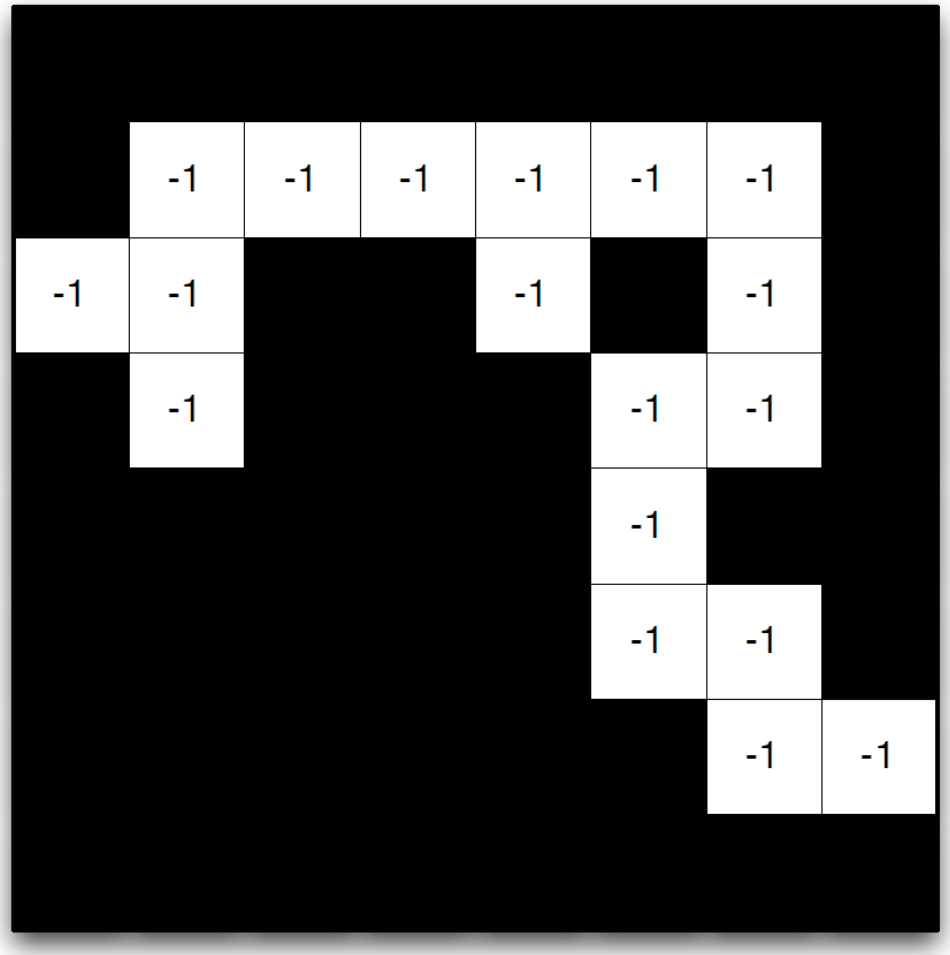
Стрелки отображают стратегию $\pi(s)$ для каждого состояния s.

Лабиринт: функция полезности



Числа отображают полезность $V^{\pi}(s)$ для каждого состояния s.

Лабиринт: модель



Агент может обладать внутренней моделью среды.

Динамика: как действия меняют среду.

Вознаграждения: какое вознаграждение может быть получено в каждом состоянии.

Модель может быть неточной.

Клетки отображают модель переходов $P(s,a,s')$.

Числа представляют следующее вознаграждение $R(s,a)$ для каждого состояния s.

Типизация RL агентов

Оценивающие функцию полезности (value based):
- стратегия не представлена явно,
- функция полезности.

Оценивающие стратегию (policy based):
- стратегия,
- функция полезности не вычисляется явно.

Актор-критик (actor-critic):
- стратегия,
- функция полезности.

Типизация RL агентов

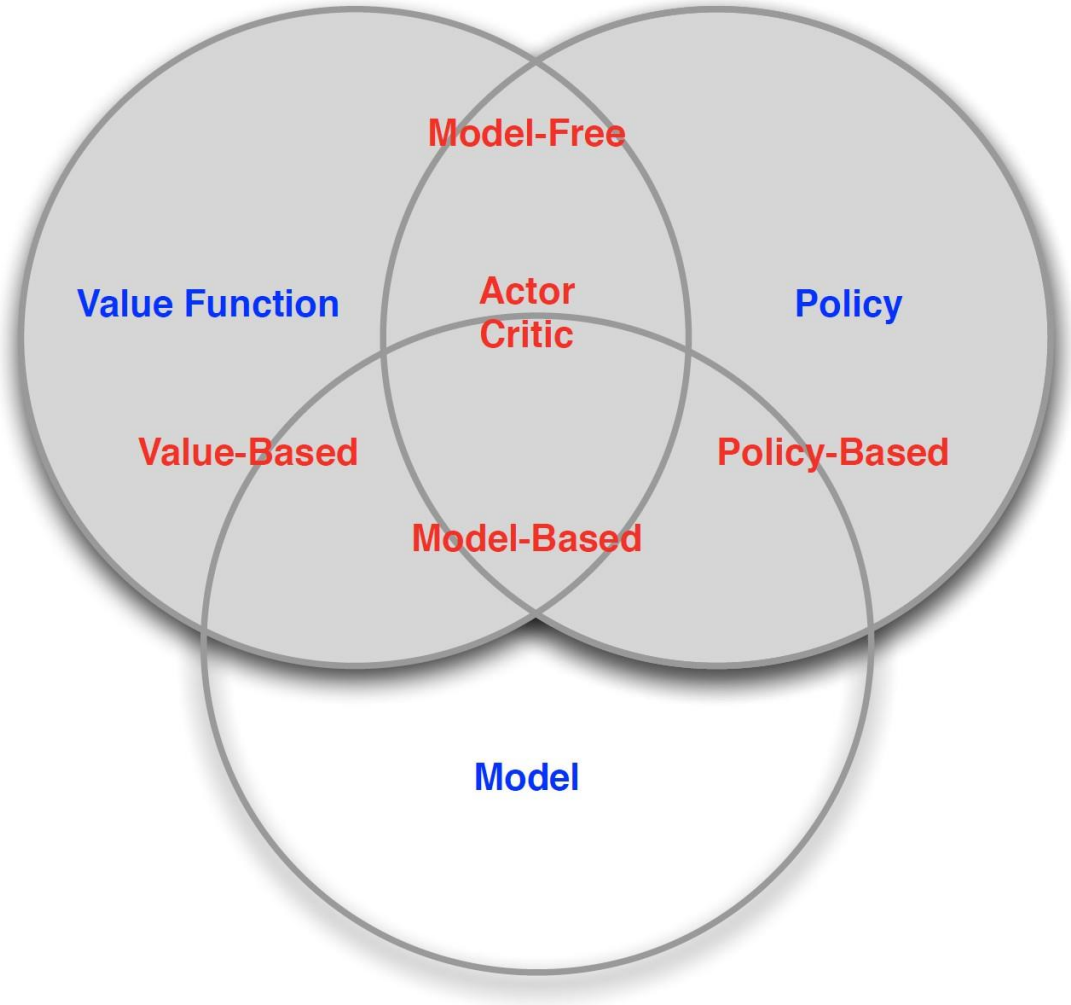
Безмодельные (model free):

- стратегия и/или функция полезности,
- нет модели.

Основанные на модели (model based):

- стратегия и/или функция полезности,
- строят модель.

Таксономия RL агентов



Подзадачи в обучении с подкреплением

Планирование и обучение

В теории последовательного принятия решений существует две основных постановки задачи.

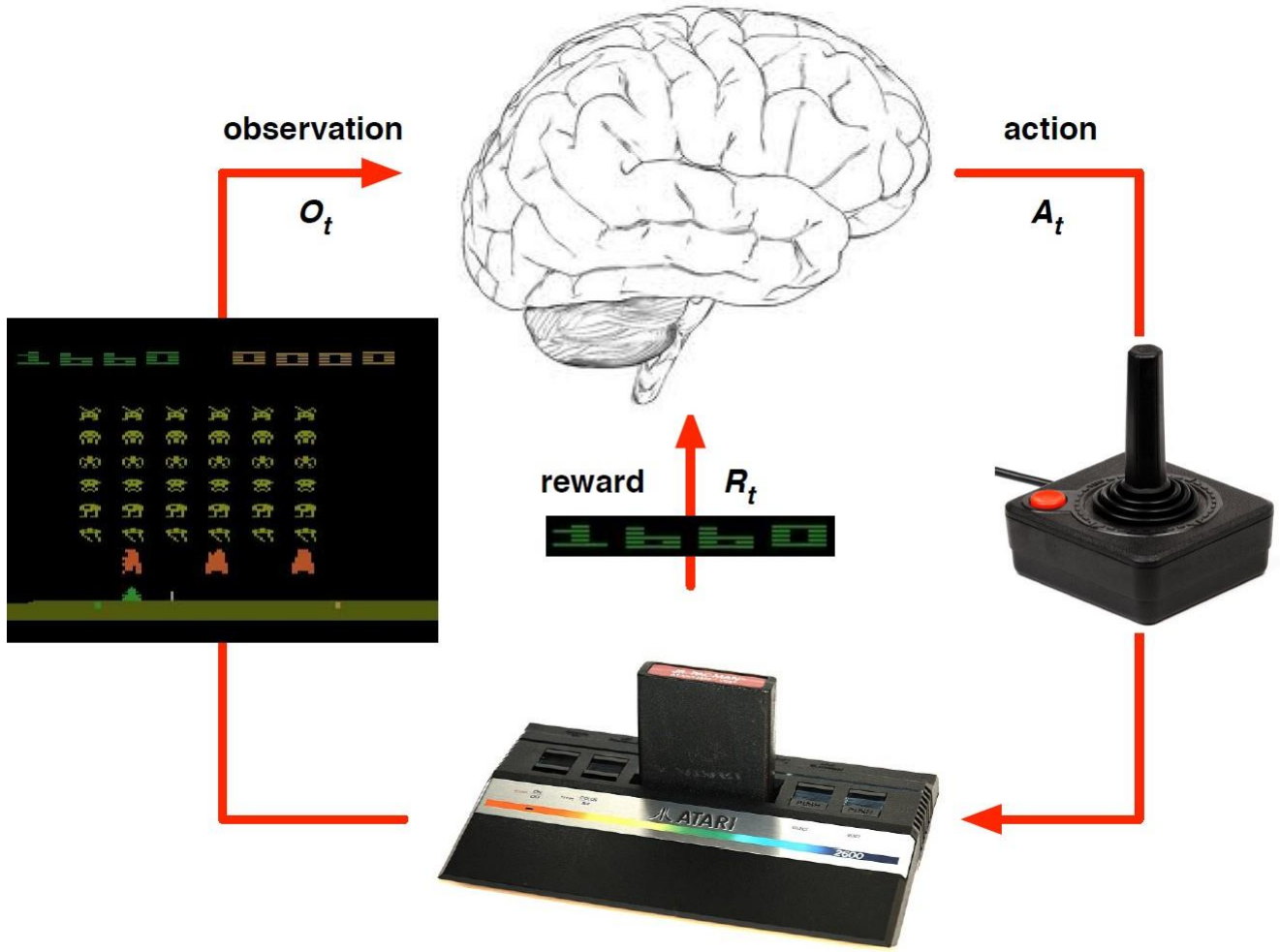
Обучение с подкреплением:

- среда изначально неизвестна,
- агент взаимодействует со средой,
- агент оптимизирует свою стратегию.

Планирование поведения:

- модель среды известна,
- агент производит вычисления, используя свою модель без взаимодействия со средой,
- агент оптимизирует стратегию,
- взаимосвязано с рассуждениями, интроспекцией, поиском.

Пример Atari: обучение



Правила игры неизвестны.

Обучение напрямую в игре.

Подача действия на джойстик, восприятие пикселей изображения.

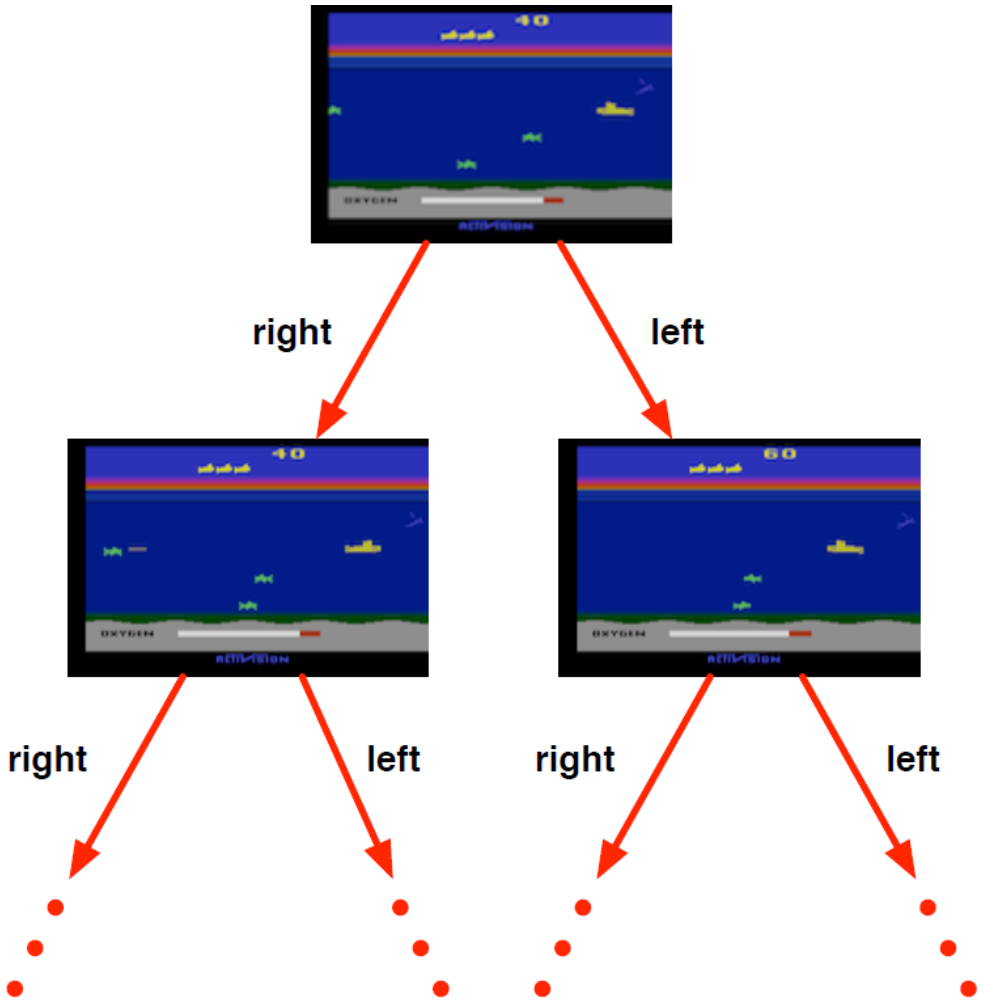
Пример Atari: планирование

Правила игры известны.

Можно запросить эмулятор – идеальная модель внутри агента.

Если я выполню действие *a* в состоянии *s*, каким будет следующее состояние или каким будет счет?

Планируем заранее для поиска оптимальной стратегии, например поиском по дереву (tree searh).



Исследование и применение

Обучение с подкреплением похоже на обучение методом проб и ошибок.

Агент должен найти оптимальную стратегию.

Он основывается на опыте, полученном при взаимодействии со средой.

При этом нельзя он не должен терять много вознаграждений в этом процессе.

Исследование и применение: примеры

Исследование (exploration) – это процесс поиска новой информации о среде.

Применение (explotation) – это процесс использования найденной информации для максимизации вознаграждения.

Обычно важно как исследовать (среду), так и применять (знания).

Предсказание и управление

Выбор ресторана:

- исследование – попробовать новый ресторан,
- применение – пойти в любимый ресторан.

Баннерная онлайн-реклама:

- исследование – показать новое объявление,
- применение – показывать наиболее привлекательное объявление.

Добыча нефти:

- исследование – пробурить новую скважину,
- применение – бурить в наилучшем известном месте.

Игры:

- исследование – сыграть экспериментальный ход,
- применение – сыграть ход, который кажется наилучшим.

Клеточный мир: предсказание

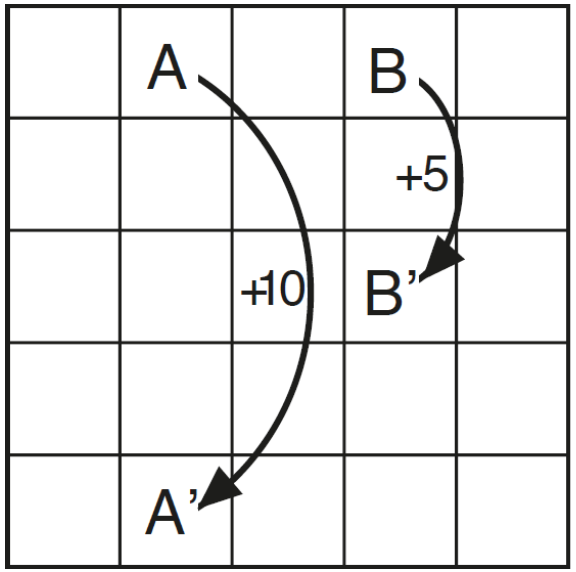
Оценка (prediction, evaluation) – вычисление будущего:

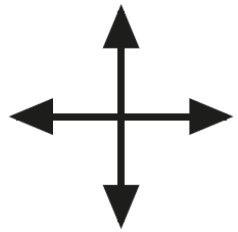
- стратегия известна (фиксирована).

Управление (control) – оптимизация будущего:

- поиск лучшей стратегии.

Клеточный мир: управление



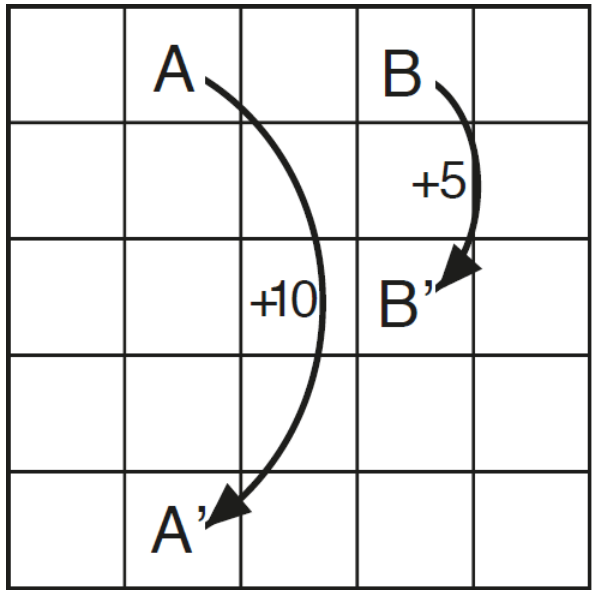


Actions

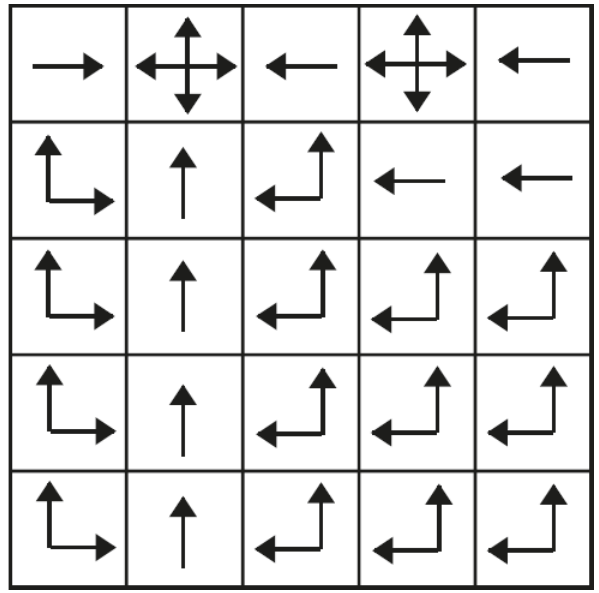
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Функция полезности для случайной равномерной стратегии.

Итог



22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7



Оптимальные функции полезности V^* и оптимальная стратегия π^*.

Взаимодействие агенты и среды

Обучение с подкреплением (RL) – особый раздел машинного обучения, в котором важно последовательное поступление данных их среды, меняющейся в следствие действий агента

В постановке задачи RL особую роль играет способ задания вознаграждения и состояния, доступного агенту

Все RL-агенты имеют общие компоненты: стратегию, аппроксиматор полезности и, возможно, модель среды

В обучении с подкреплением можно выделить подзадачи, связанные с долгосрочным планированием, исследование среды и обновлением стратегии

Обучение с подкреплением: определения и примеры

План

- Марковское свойство и марковские процессы
- Уравнение Беллмана для value функции
- Value функция
- Оптимальная value и qvalue функции
- Алгоритм итерации по qvalue функции
- Алгоритм итерации по value функции

Марковское свойство и марковский процесс

Марковский процесс принятия решений

Марковский процесс принятия решений (МППР, MDP) моделирует взаимодействие агенты и среды.

Предполагаем, что среда *полностью наблюдаема* (fully observable).

Текущее состояние полностью характеризует весь процесс взаимодействия.

Почти все задачи обучения с подкреплением (RL) могут быть сведены к задаче с MDP:

- Оптимальное управление – MDP с непрерывным множеством состояний и действий.
- Частично наблюдаемые среды (partially observable) могут быть сведены к MDP.
- Игровые автоматы – пример MDP с одним состоянием.

Марковское свойство

Будущее не зависит от прошлого и определяется только настоящим.

Definition

Состояние s_t называется марковским если и только если

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_1, \ldots, s_t]$$

Текущее состояние содержит всю необходимую информацию из истории взаимодействия.

Если есть текущее состояние, история может далее не учитываться.

Состояние содержит достаточно статистики для определения будущего.

Матрица переходов

Вероятность перехода для марковского состояния s в следующее состояние s' определяется как

$$P_{ss'} = P[s_{t+1} = s' | s_t = s]$$

Матрица переходов P определяет вероятности переходов между всеми возможными состояниями:

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

Каждая строчка матрицы в сумме дает 1.

Марковский процесс

Марковский процесс (Markov process, Markov chain) – это случайный процесс без памяти, т.е. последовательность случайных состояний $s_1, s_2, \ldots$ с марковским свойством.

Definition

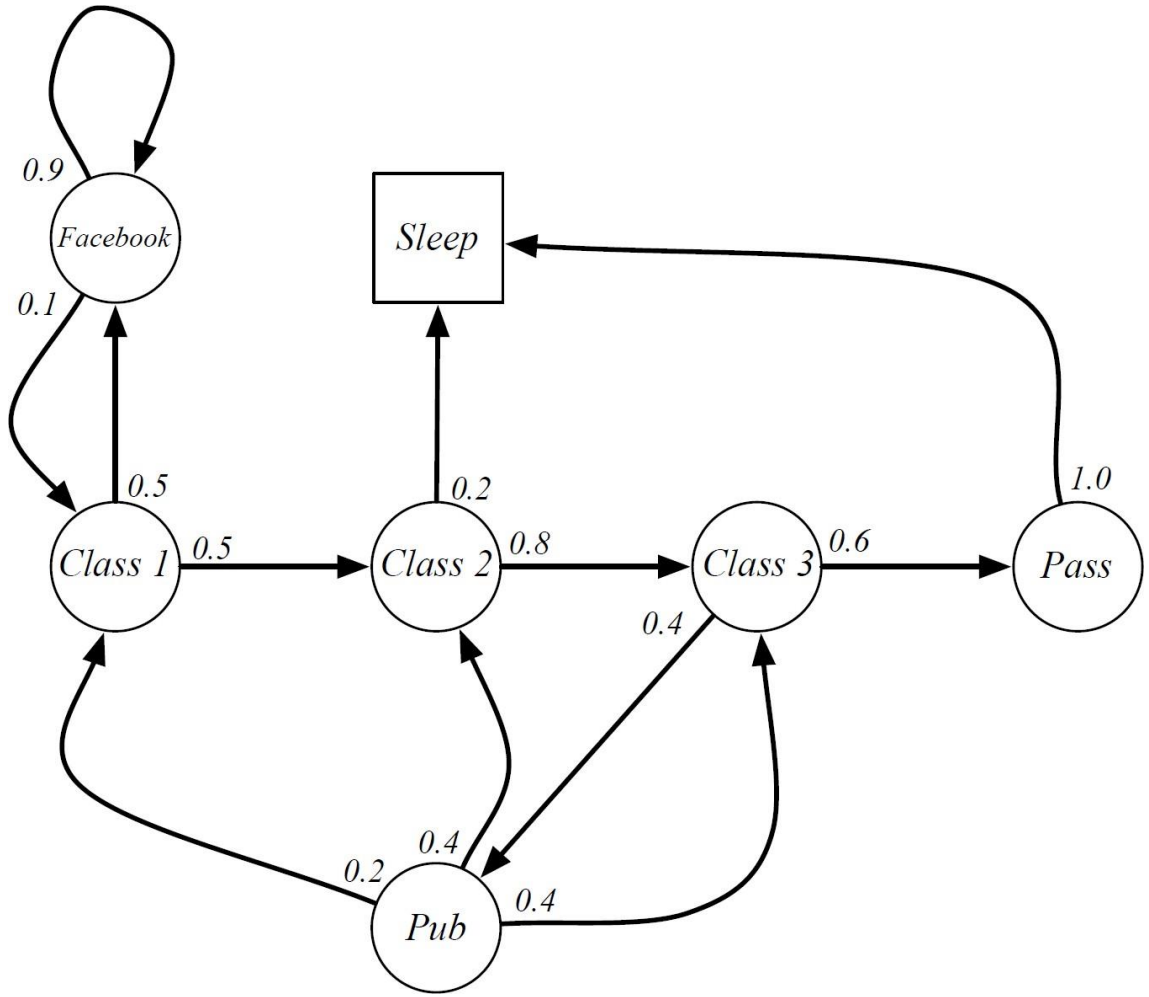
Марковский процесс (или марковская цепь) – это двойка $\langle S, \mathrm{P} \rangle$, где

S – (конечное) множество состояний,

P – матрица переходов

$$\mathrm{P}_{ss'} = \mathrm{P}[s_{t+1} = s' | s_t = s].$$

Пример: студенческая марковская цепь



Пример: эпизоды студенческой марковской цепи

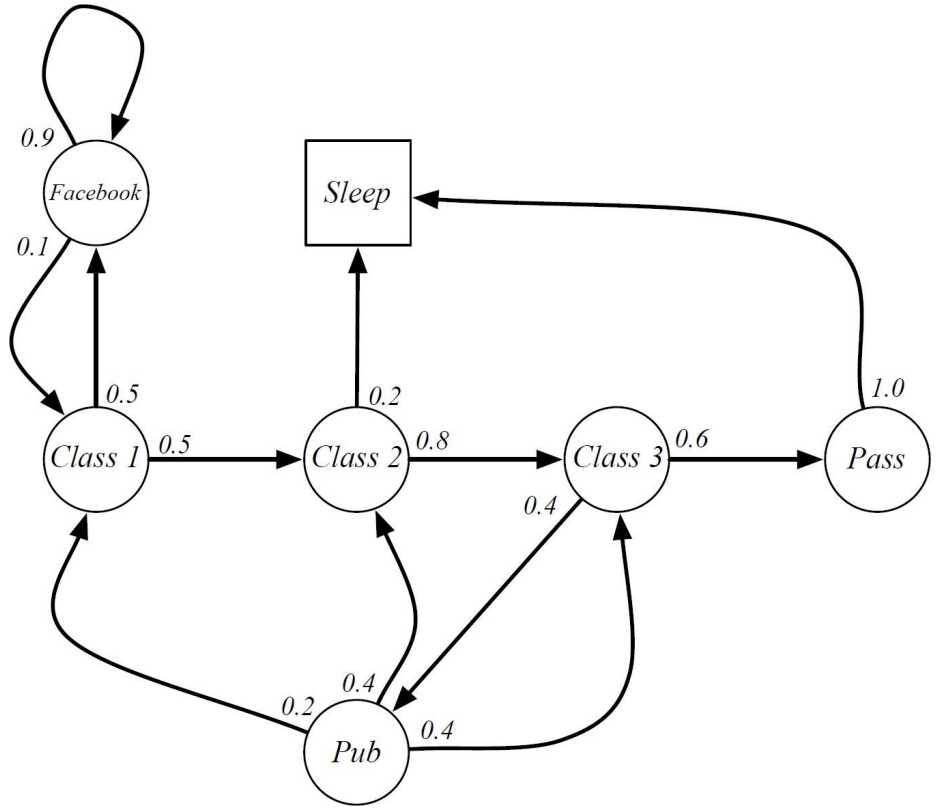
Выборка эпизодов для студенческой марковской цепи, начинающихся с
$s_1 = C1$:

C1 C2 C3 Pass Sleep

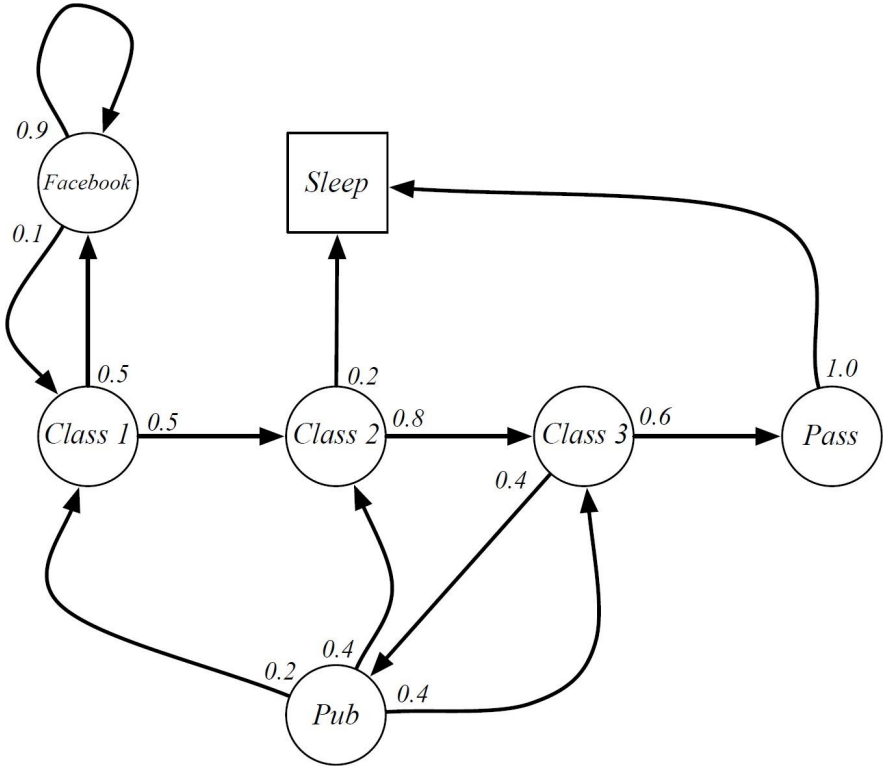
C1 FB FB C1 C2 Sleep

C1 C2 C3 Pub C2 C3 Pass Sleep

C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep



Пример: переходы студенческой марковской цепи



$$\mathcal{P} = \begin{array}{c} \\ C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{array} \begin{array}{c} \begin{array}{ccccccc} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{array} \\ \begin{bmatrix} & 0.5 & & & 0.5 & & \\ & & 0.8 & & & & 0.2 \\ & & & 0.6 & 0.4 & & \\ & & & & & & 1.0 \\ 0.2 & 0.4 & 0.4 & & & & \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1.0 \end{bmatrix} \end{array}$$

Марковский процесс вознаграждения

Марковский процесс вознаграждения – это марковская цепь с дополнительными значениями переходов.

Definition

Марковский процесс вознаграждения (MRP) – это тройка $\langle S, \mathrm{P}, \mathrm{R}, \gamma \rangle$, где:

S – конечное множество состояний,

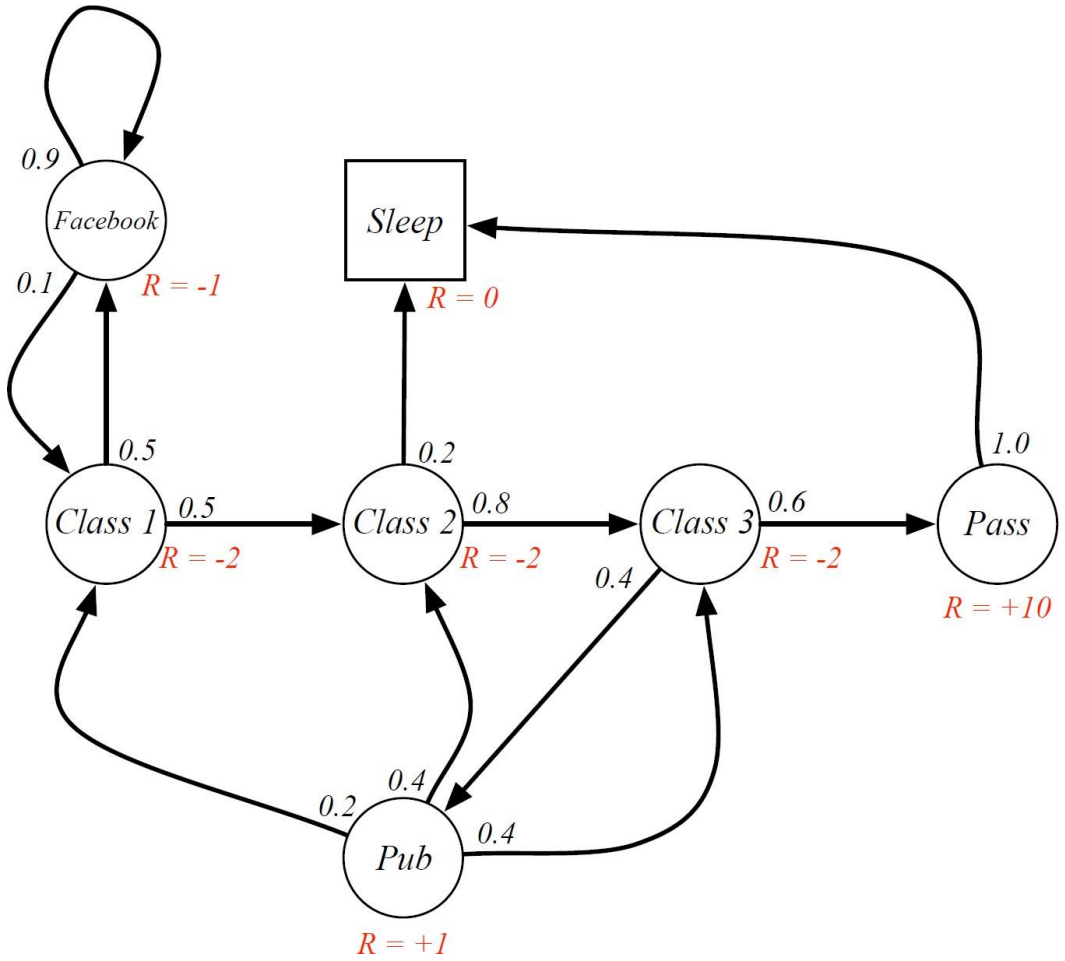
P – матрица вероятностей переходов:

$$\mathrm{P}_{SS'} = \mathrm{P}[s_{t+1} = s' | s_t = s],$$

R – функция вознаграждения $\mathrm{R}_S = \mathrm{E}[r_{t+1} | s_t = s]$,

$\gamma \in [0, 1]$ – дисконтирующий множитель.

Пример: студенческий MRP



Суммарное вознаграждение

Definition

Суммарное вознаграждение (*отдача*, return) – сумма дисконтированных вознаграждений с момента времени t:

$$G_t = r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Дисконтирующий множитель $\gamma \in [0, 1]$ – оценка значений будущих вознаграждений.
Значение получаемого вознаграждения после $k + 1$ шагов – $\gamma^k r$. Моментальное вознаграждение важнее отложенных будущих:

- $\gamma \sim 0$ – близорукий агент,
- $\gamma \sim 1$ – дальнозоркий агент.

Дисконтирование

Большинство задач включают дисконтирующий множитель.

Математическое удобство.

Избегаем бесконечных отдач в марковских процессах с циклами (бесконечный горизонт).

Отражение неопределенности будущего.

Во многих задачах немедленное вознаграждение важнее отложенного.

Биологически/психологически правдоподобно.

Это более общий случай (всегда можно установить $\gamma = 1$).

Функция полезности

Функция полезности $V(s)$ дает долгосрочную оценку отдачи, начиная с состояния s.

Definition

Функция полезности $V(s)$ марковского процесса вознаграждения – это ожидаемая отдача, получаемая начиная с состояния s:

$$V(s) = \mathrm{E}[G_t | s_t = s].$$

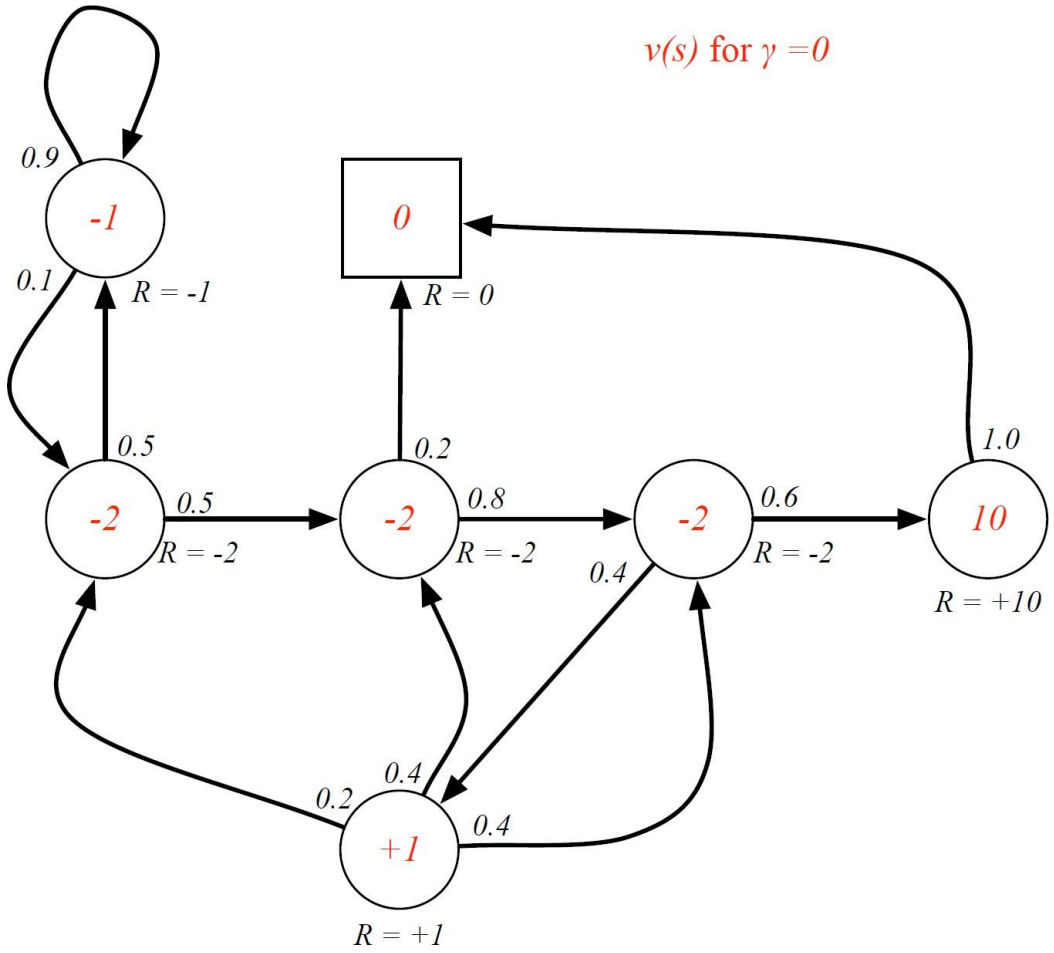
Пример: отдача в StudMRP

Выборка отдач для студенческого MRP, начиная с состояния $s_1 = C1$ с $\gamma = 1/2$:

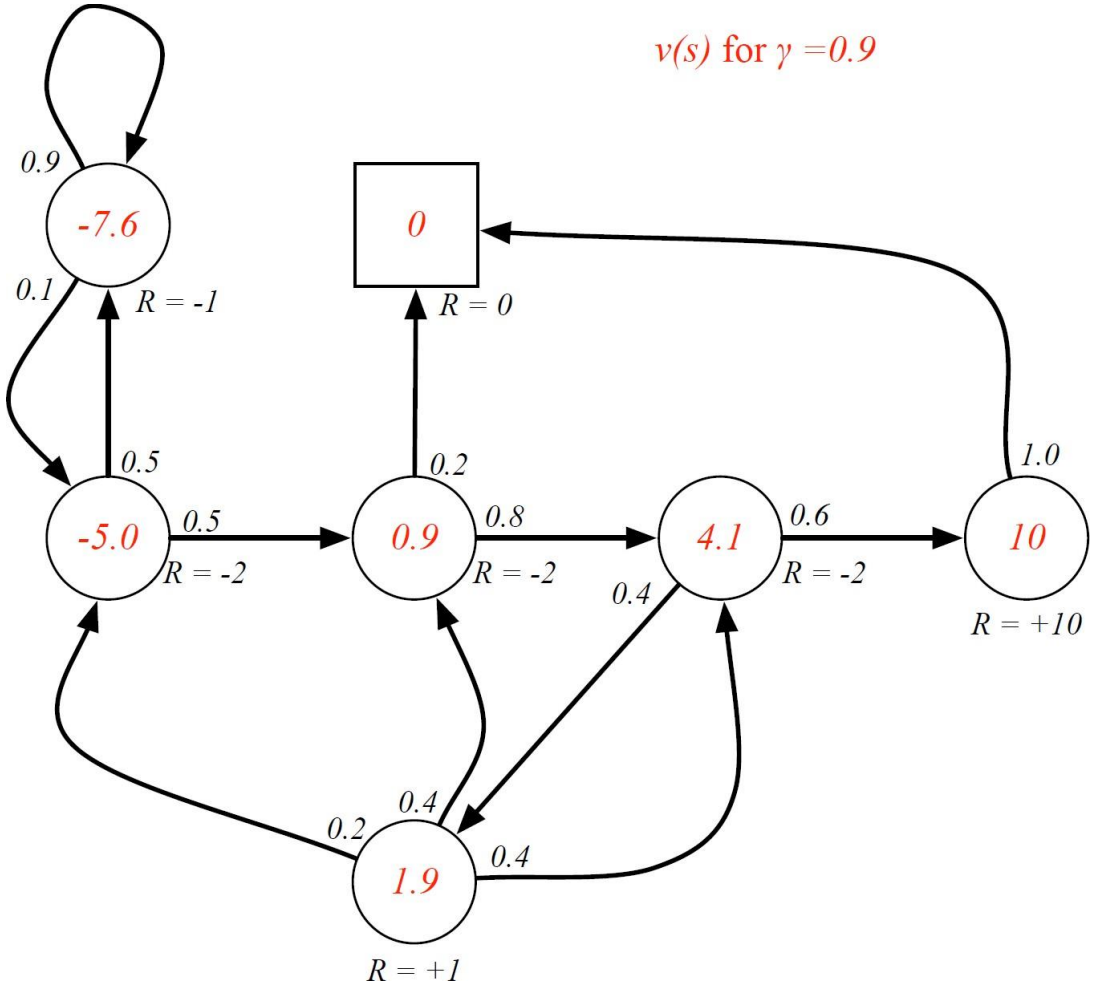
$$G_1 = r_2 + \gamma r_3 + \cdots + \gamma^{T-2} r_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	=	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	=	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	=	-3.41
C1 FB FB C1 C2 C3 Pub C1 ... FB FB FB C1 C2 C3 Pub C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	=	-3.20

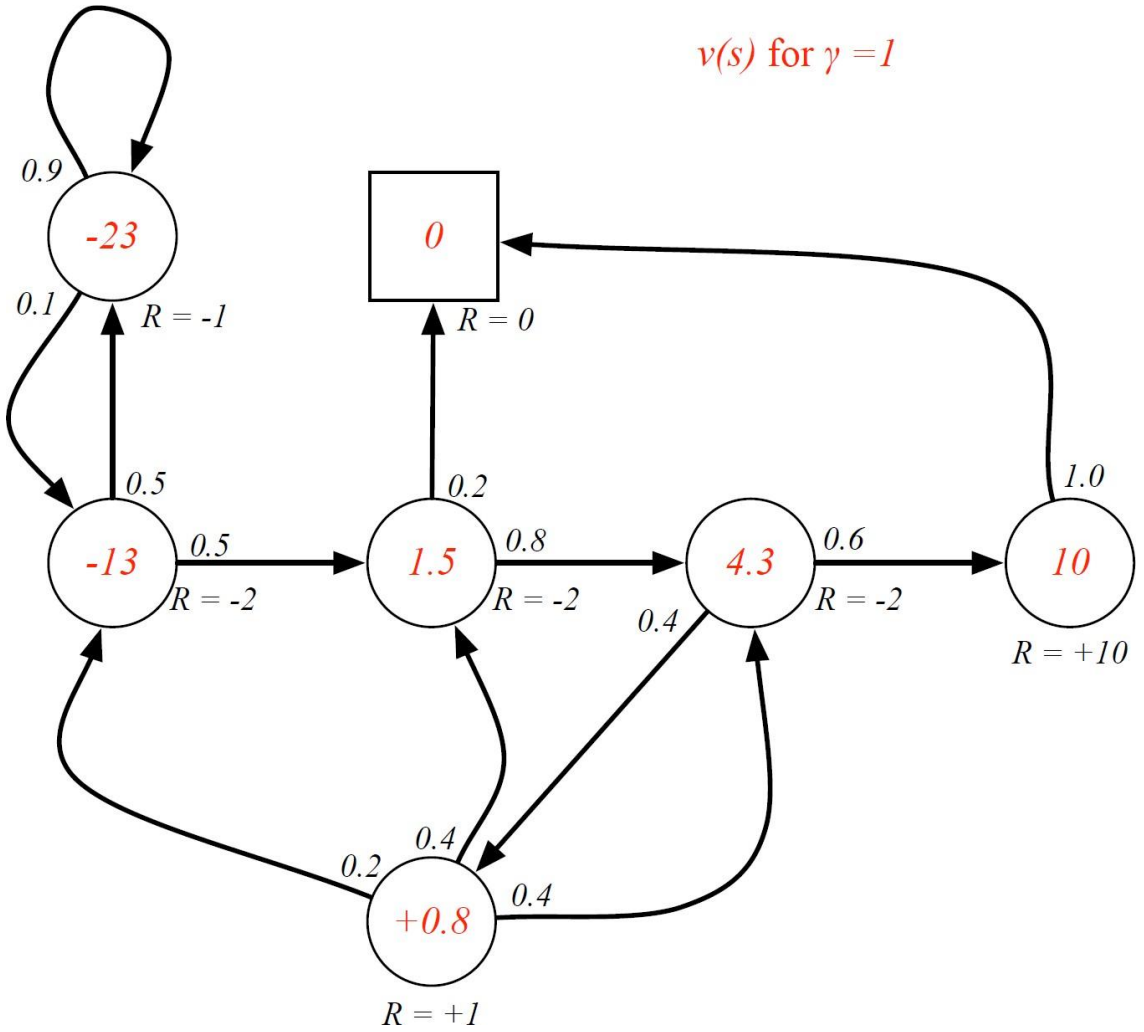
Пример: полезность состояний для StudMRP



Пример: полезность состояний для StudMRP



Пример: полезность состояний для StudMRP



Уравнение Беллмана для полезности

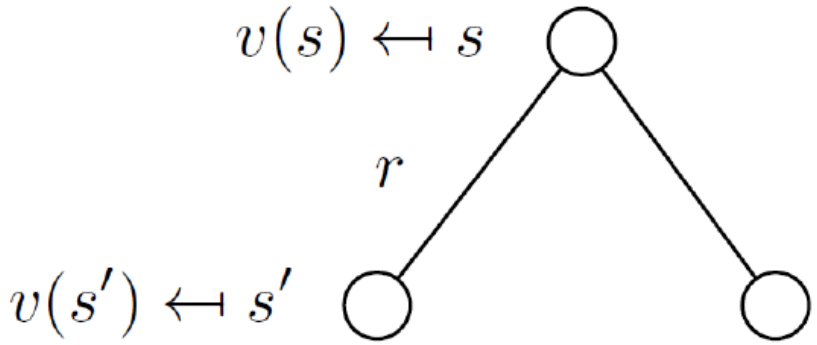
Уравнение Беллмана для MRP

Функцию полезности можно представить в виде двух слагаемых: немедленное вознаграждение r_{t+1}, дисконтированное значение следующего состояния $\gamma V(s_{t+1})$:

$$
\begin{aligned}
V(s) &= \mathrm{E}[G_t | s_t = s] = \\
&\mathrm{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | s_t = s] = \\
&\mathrm{E}[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \ldots) | s_t = s] = \\
&\mathrm{E}[r_{t+1} + \gamma G_{t+1} | s_t = s] = \\
&\mathrm{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s]
\end{aligned}
$$

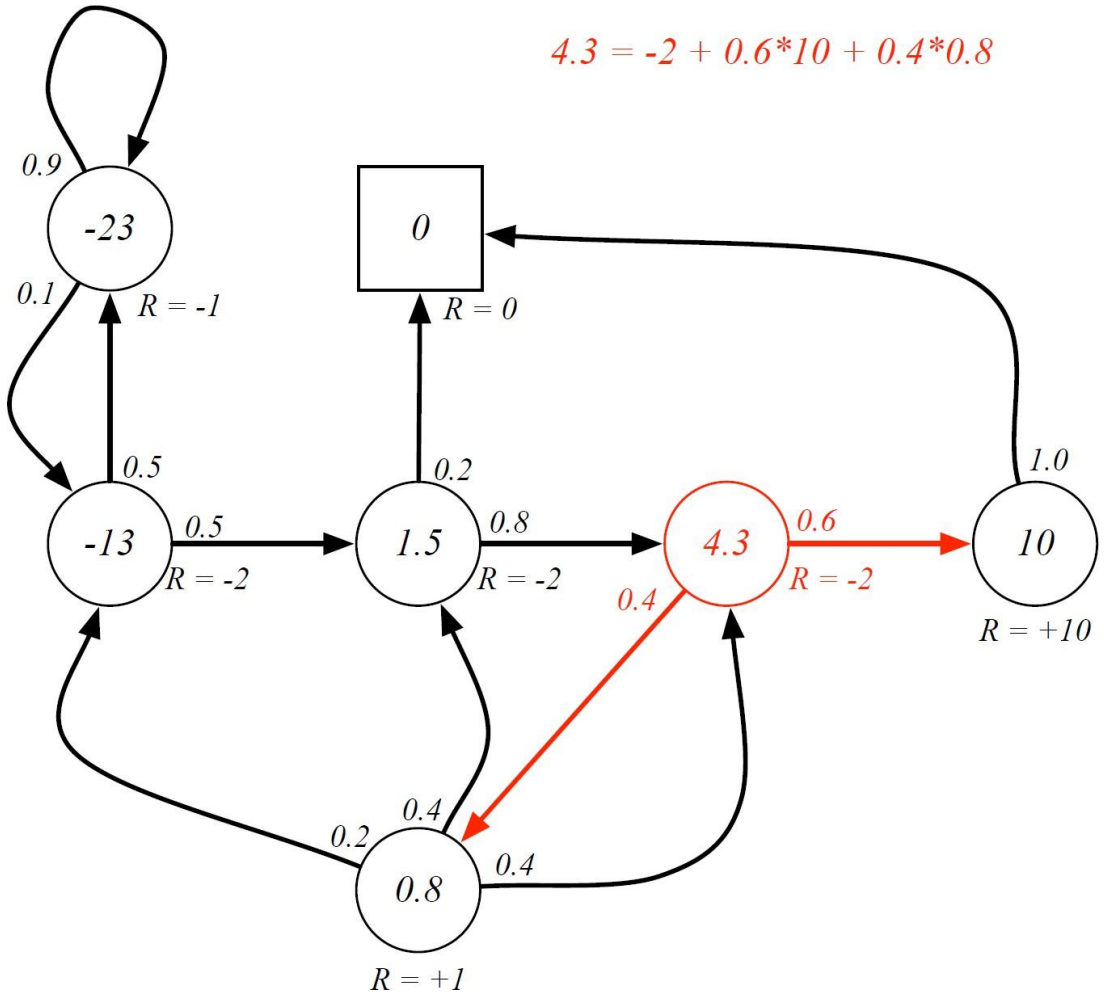
Уравнение Беллмана для MRP

$$V(s) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1}) | s_t = s]$$



$$V(s) = \mathcal{R}_s + \gamma \sum_{s' \in S} \mathcal{P}_{ss'} V(s')$$

Пример: уравнение Беллмана для StudMRP



Уравнение Беллмана в матричной форме

Мы можем записать уравнение Беллмана в матричной форме:

$$V = \mathcal{R} + \gamma \mathcal{P} V$$

где V – это вектор-колонка с одной компонентной на состояние:

$$\begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}(1) \\ \vdots \\ \mathcal{R}(n) \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix}$$

Решение уравнения Беллмана

Уравнение Беллмана – это линейное уравнение (система линейных уравнений).

Аналитическое решение:

$$V = \mathrm{R} + \gamma \mathrm{P} V$$

$$(I - \gamma \mathrm{P}) V = \mathrm{R}$$

$$V = (I - \gamma \mathrm{P})^{-1} \mathrm{R}$$

Вычислительная сложность $O(n^3)$, где n – число состояний. Аналитическое решение возможно только для небольших задач.

Но есть много итерационных приближенных методов для больших задач:

- динамическое программирование,
- Монте-Карло подход,
- метод временных различий.

Марковский процесс принятия решений

Марковский процесс принятия решений – это марковский процесс вознаграждения для действий. Он описывает взаимодействие со средой с марковскими состояниями.

Definition

Марковский процесс принятия решений – это кортеж $\langle S, A, \mathrm{P}, \mathrm{R}, \gamma \rangle$, где

S – конечное множество состояний,

A – конечное множество действий,

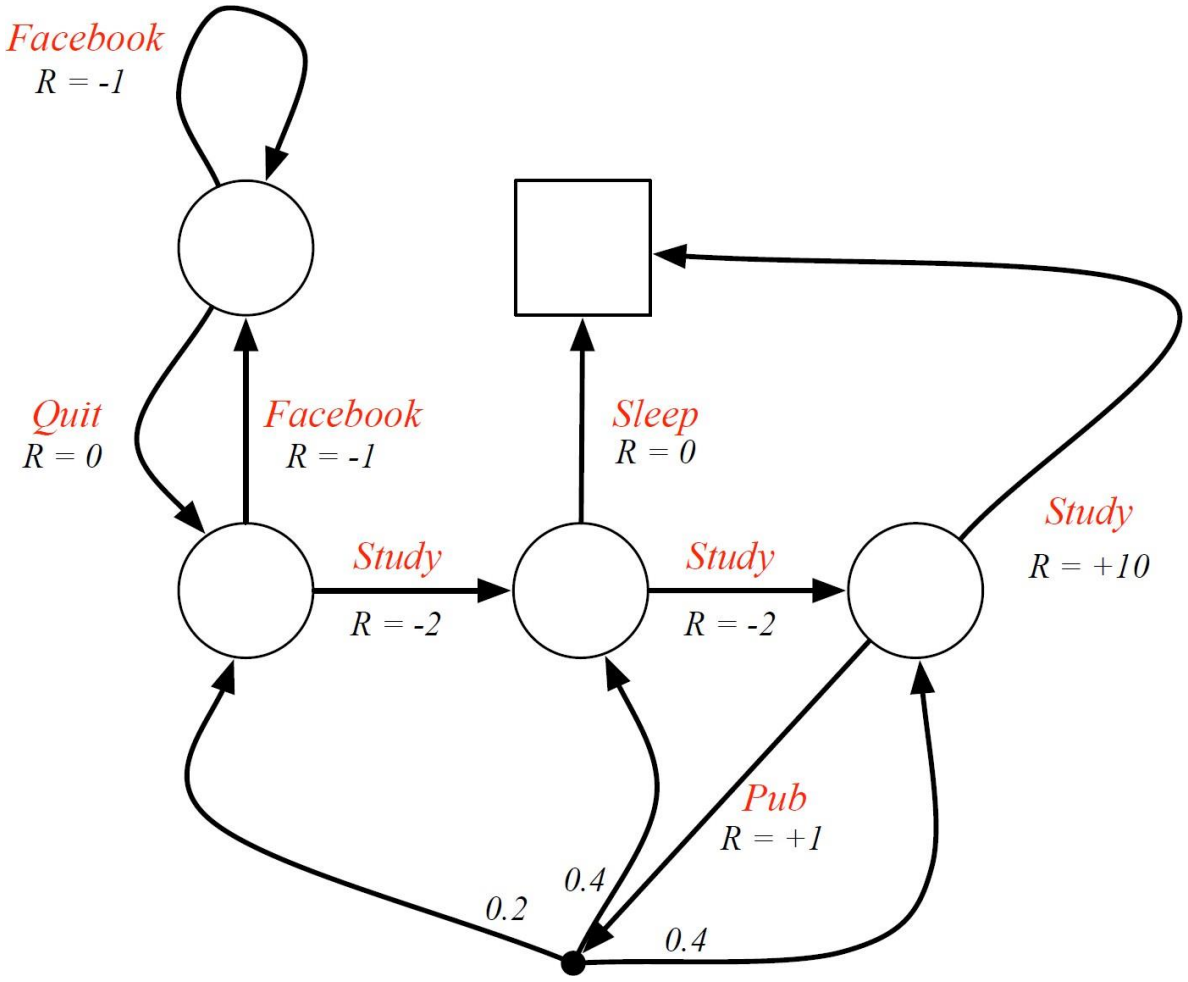
P – матрица вероятностей переходов:

$$\mathrm{R}(\mathrm{a}, \mathrm{s}, \mathrm{s}') = \mathrm{P}[s_{t+1} = \mathrm{s}' | s_t = s, a_t = \mathrm{a}]$$

R – функция вознаграждения $\mathrm{R}(\mathrm{a}, \mathrm{s}) = \mathrm{E}[r_{t+1} | s_t = s, a_t = \mathrm{a}]$

$\gamma \in [0, 1]$ – дисконтирующий множитель.

Пример: студенческий MDP



Стратегии

Definition

Стратегией π будем называть вероятностное распределение на множестве действий при текущем состоянии s:

$$\pi(a|s) = \mathrm{P}[a_t = a|s_t = s]$$

Стратегия полностью определяет поведение агента. В MDP стратегии зависят от текущего состояния. Стратегии стационарны (не зависят от времени):

$$a_t \sim \pi(\cdot|s_t), \forall t > 0.$$

Связь MDP, MRP и марковских процессов

Пусть дан MDP $\mathrm{M} = \langle S, A, \mathrm{P}, \mathrm{R}, \gamma \rangle$ и стратегия π.

Последовательность состояний $s_1, s_2, \ldots$ – марковский процесс $\langle S, \mathrm{P} \rangle$

Последовательность состояний и вознаграждений $s_1, r_1, s_2, \ldots$ – марковский процесс вознаграждений $\langle S, \mathrm{P}^{\pi}, \mathrm{R}^{\pi}, \gamma \rangle$, где

$$\mathcal{P}^{\pi}_{ss'} = \sum_{a \in A} \pi(a|s) \mathcal{P}^{a}_{ss'}$$

$$\mathcal{R}^{\pi}_{s} = \sum_{a \in A} \pi(a|s) \mathcal{R}^{a}_{s}$$

Value функция

Функция полезности

Definition

Функцией полезности состояний MDP $V^{\pi}(s)$ будем называть математическое ожидание отдачи, полученной, начиная с состояния s, при выполнении стратегии π:

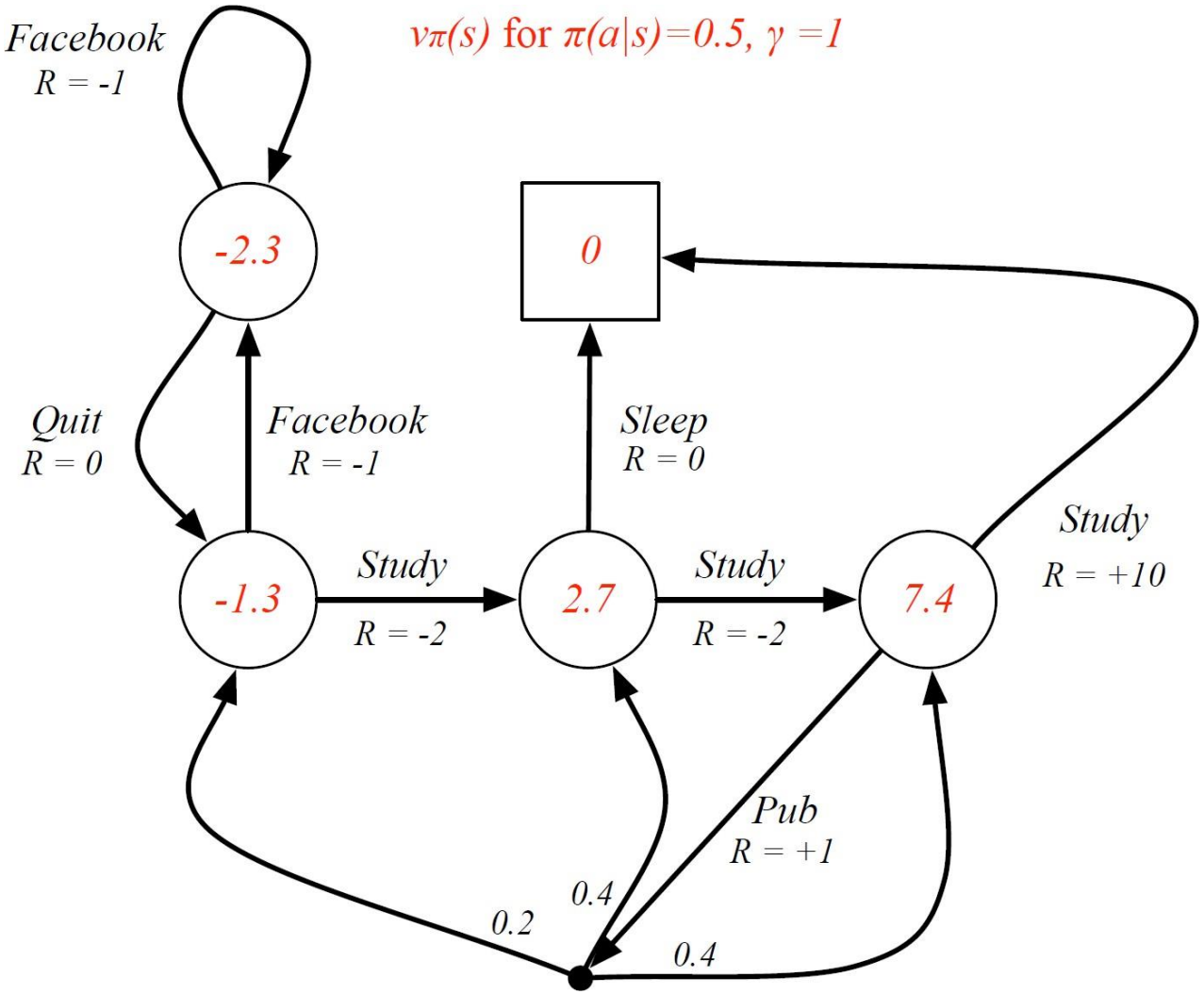
$$V^{\pi}(s) = \mathrm{E}_{\pi}[G_t | s_t = s]$$

Definition

Функцией полезности действий MDP $Q^{\pi}(s, a)$ будем называть математическое ожидание отдачи, полученной, начиная с состояния s и выбранного действия a, при выполнении стратегии π:

$$Q^{\pi}(s, a) = \mathrm{E}_{\pi}[G_t | s_t = s, a_t = a]$$

Пример: функция полезности для StudMDP



Уравнение Беллмана для MDP

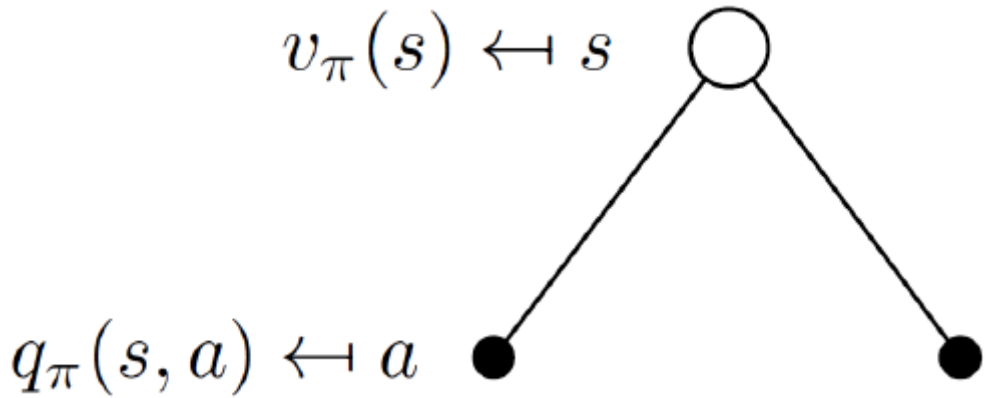
Функция полезности состояний может быть разложена на немедленное вознаграждение и дисконтированную полезность следующего состояния:

$$V^{\pi}(s) = \mathrm{E}_{\pi}[r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s]$$

Для функции полезности действий аналогично:

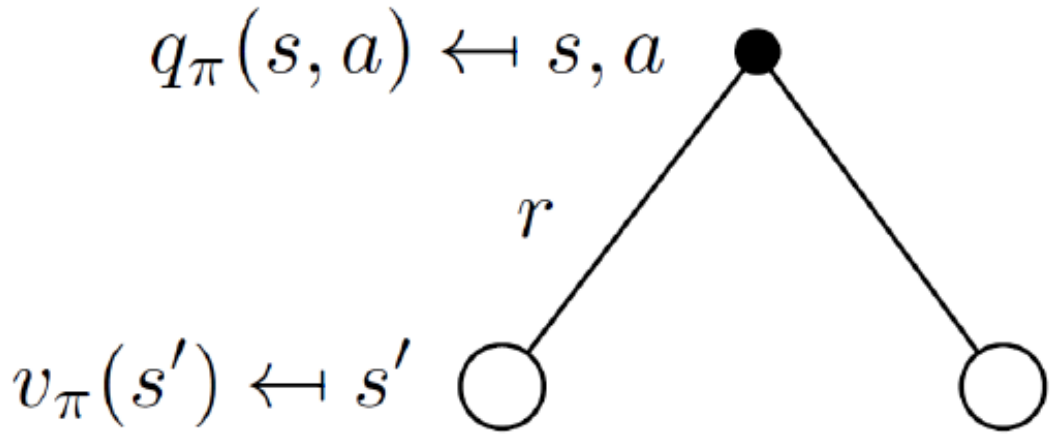
$$Q^{\pi}(s, a) = \mathrm{E}_{\pi}[r_{t+1} + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) | s_t = s, a_t = a]$$

Уравнение Беллмана для V^{π}



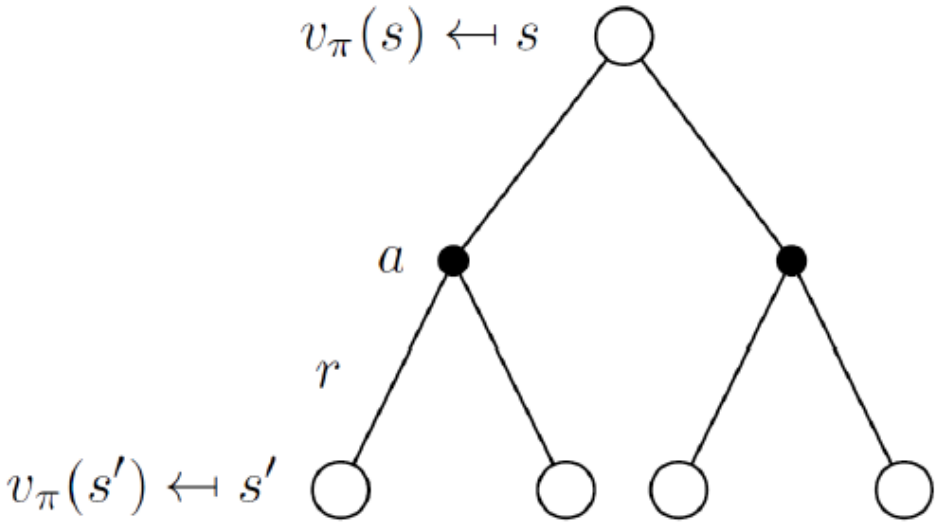
$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) Q^{\pi}(s,a)$$

Уравнение Беллмана для Q^{π}



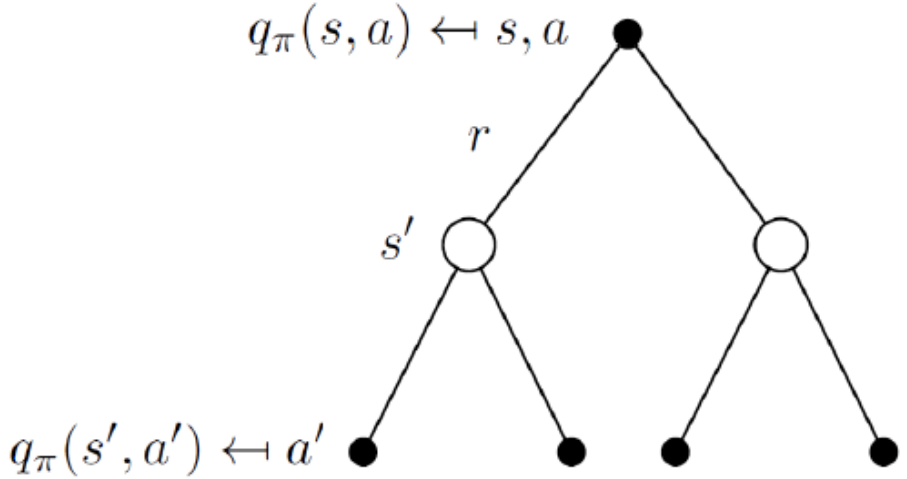
$$Q^{\pi}(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V^{\pi}(s')$$

Уравнение Беллмана для V^{π}



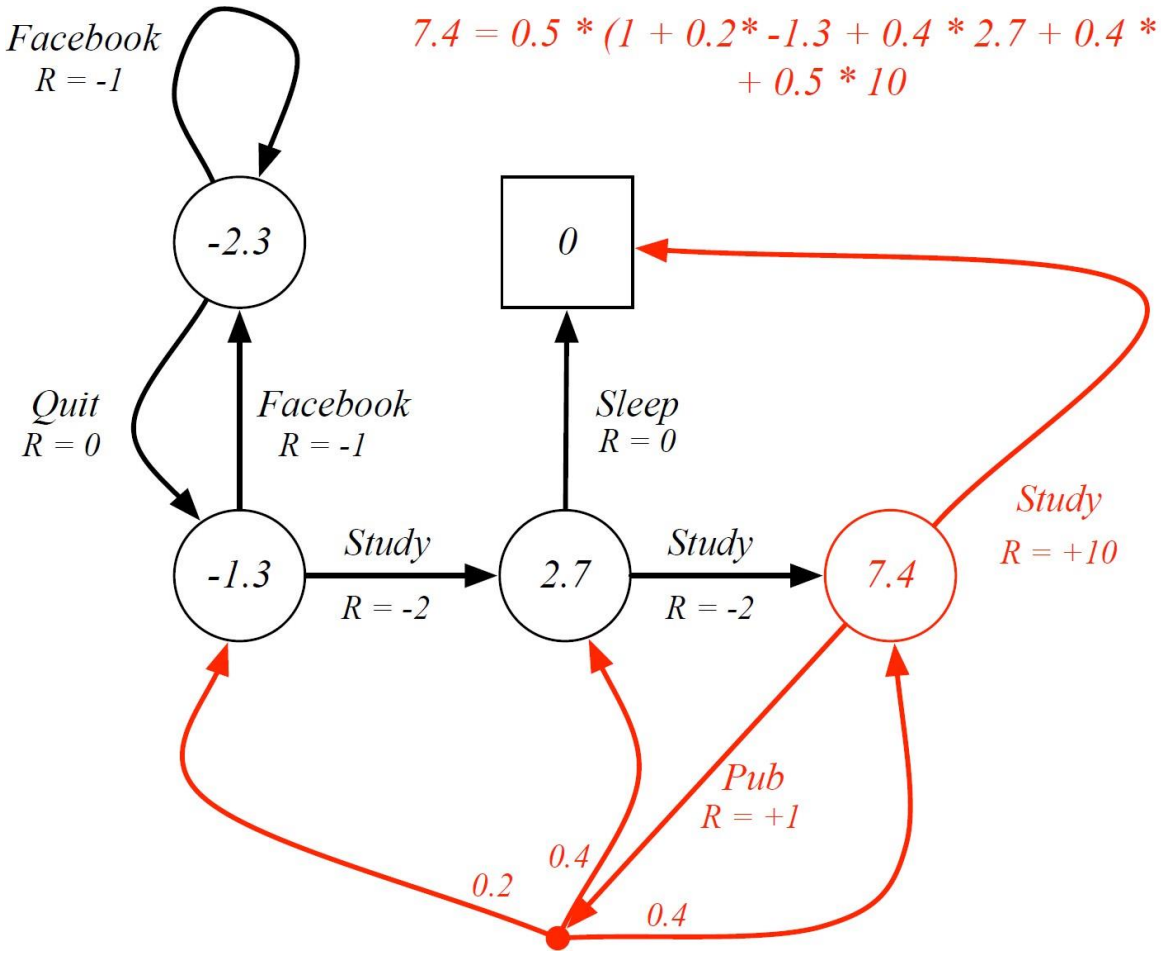
$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s)\left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V^{\pi}(s')\right)$$

Уравнение Беллмана для Q^{π}



$$Q^{\pi}(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \sum_{a' \in A} \pi(a'|s') Q^{\pi}(s',a')$$

Пример: уравнение Беллмана для StudMDP



Матричная форма уравнения Беллмана для MDP

Уравнение Беллмана с матожиданиями может быть кратко записано по аналогии с MRP:

$$V^{\pi} = \mathrm{R}^{\pi} + \gamma \mathrm{P}^{\pi} V^{\pi}$$

Аналитическое решение:

$$V^{\pi} = (I - \gamma \mathrm{P}^{\pi})^{-1} \mathrm{R}^{\pi}$$

Оптимальная value и qvalue функции

Оптимальная функция полезности

Definition

Оптимальная функция полезности состояний $V^*(s)$ – это максимальное значение функции полезности по всем стратегиям:

$$V^*(s) = \max_{\pi} V^{\pi}(s).$$

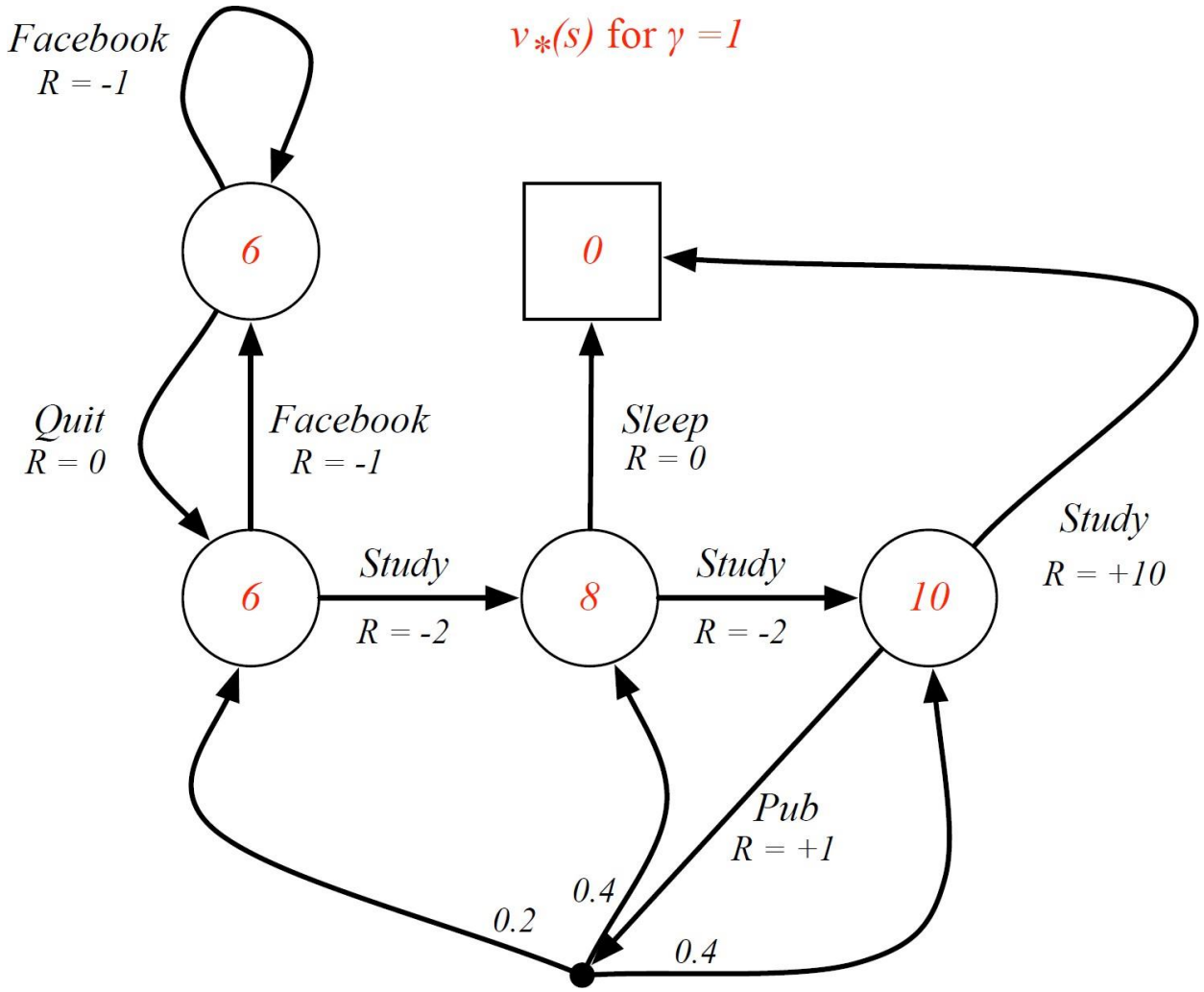
Оптимальная функция полезности действий $Q^*(s, a)$ – это максимальное значение функции полезности по всем стратегиям:

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a).$$

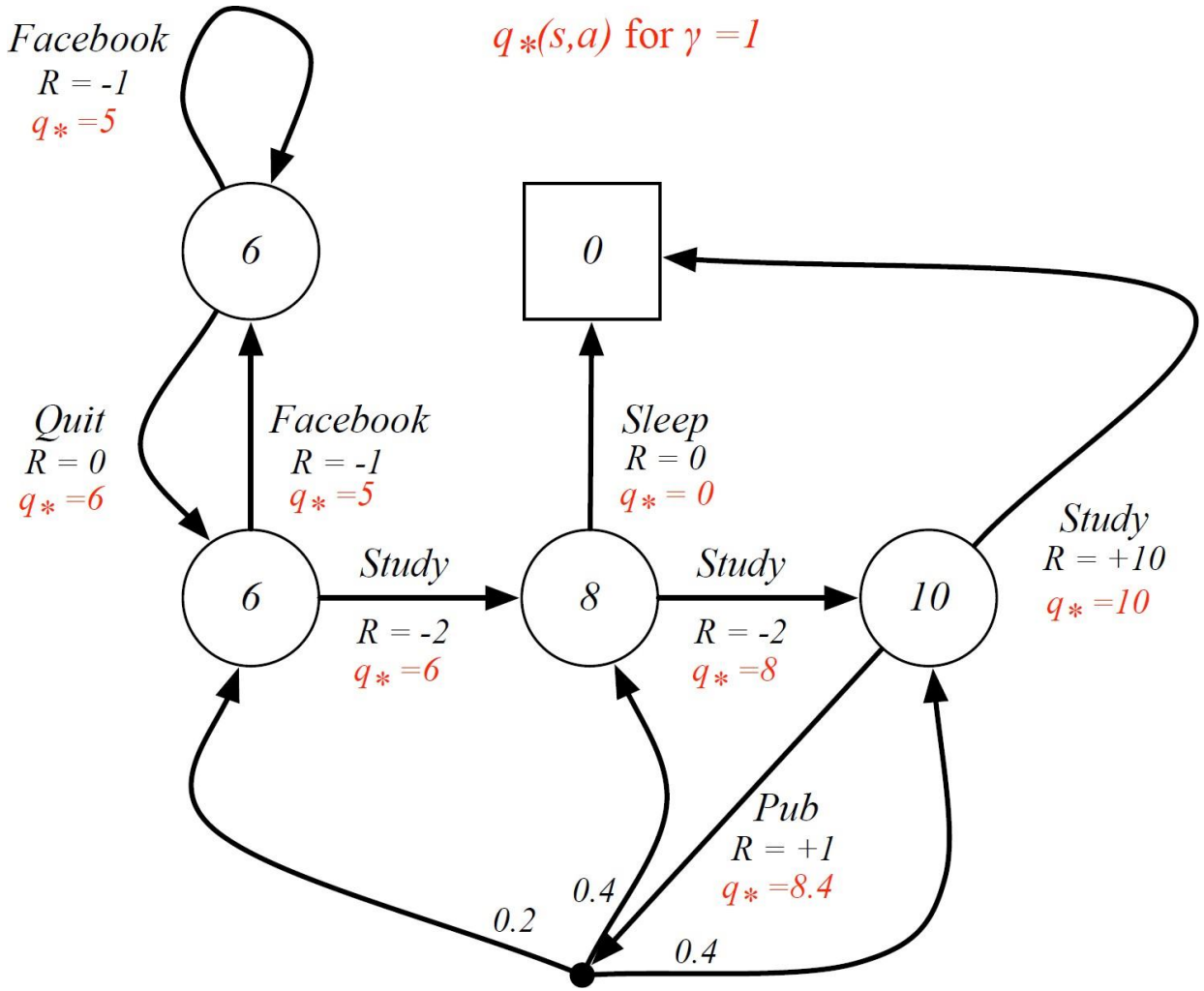
Оптимальные стратегии характеризуют лучшее поведение в MDP.

Говорят, что задача MDP решена, когда найдена оптимальная функция полезности.

Пример: V ∗ для StudMDP



Пример: Q* для StudMDP



Оптимальная стратегия

Определим частичный порядок на множестве стратегий:

$\pi \geq \pi'$, если $V^{\pi}(s) \geq V^{\pi'}(s), \forall s$

Theorem

Для любого марковского процесса принятия решений:

существует оптимальная стратегия π^*, *которая лучше или эквивалентна всем другим стратегиям:* $\pi^* \geq \pi, \forall \pi$

все оптимальные стратегии удовлетворяют оптимальной функции полезности состояний: $V^{\pi^*}(s) = V^*(s)$,

все оптимальные стратегии удовлетворяют оптимальной функции полезности действий: $Q^{\pi^*}(s, a) = Q^*(s, a)$.

Поиск оптимальной стратегии

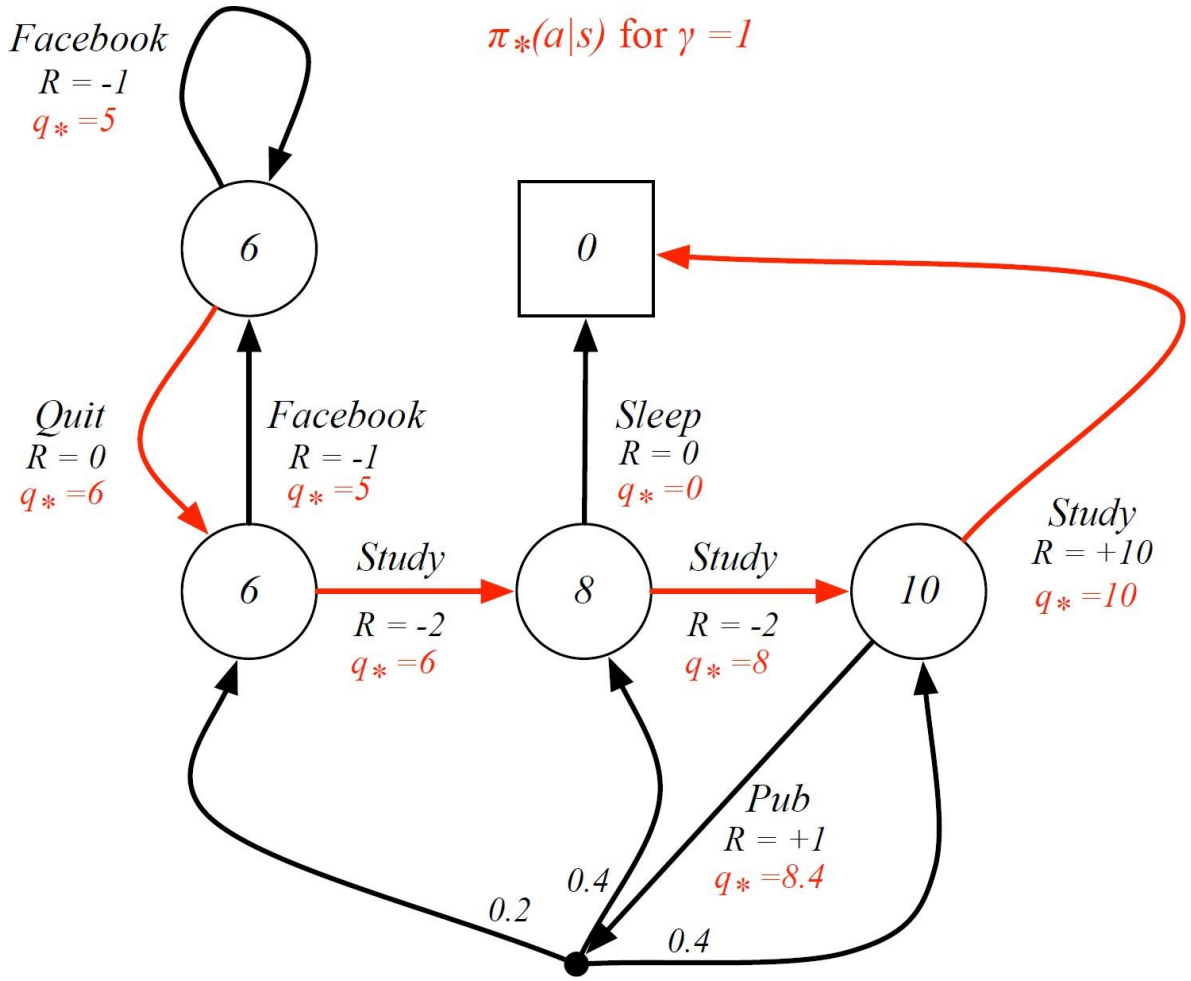
Оптимальная стратегия π^* может быть найдена максимизацией функции полезности действий $Q^{\pi^*}(s, a)$:

$$\pi^*(a|s) = \begin{cases} 1, & \text{если } a = \arg\max_{a \in A} Q^*(s, a) \\ 0, & \text{иначе.} \end{cases}$$

Для любого MDP существует оптимальная детерминированная стратегия.

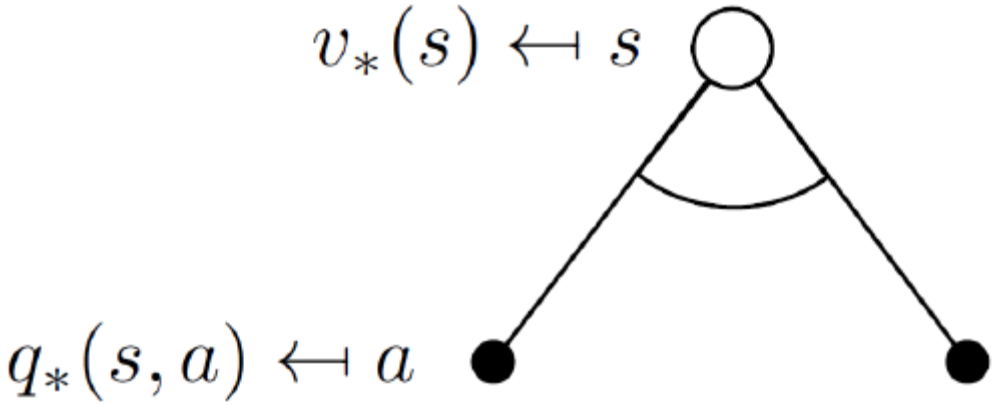
Если известна $Q^*(s, a)$, то мы одновременно получаем и оптимальную стратегию.

Пример: оптимальная стратегия для StudMDP



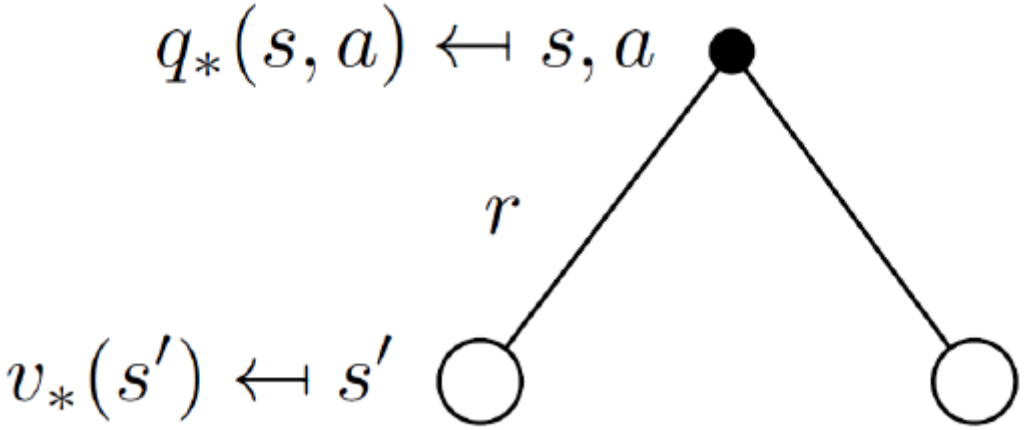
Уравнение Беллмана для V*

Оптимальные значения функции полезности связаны уравнением оптимальности Беллмана:



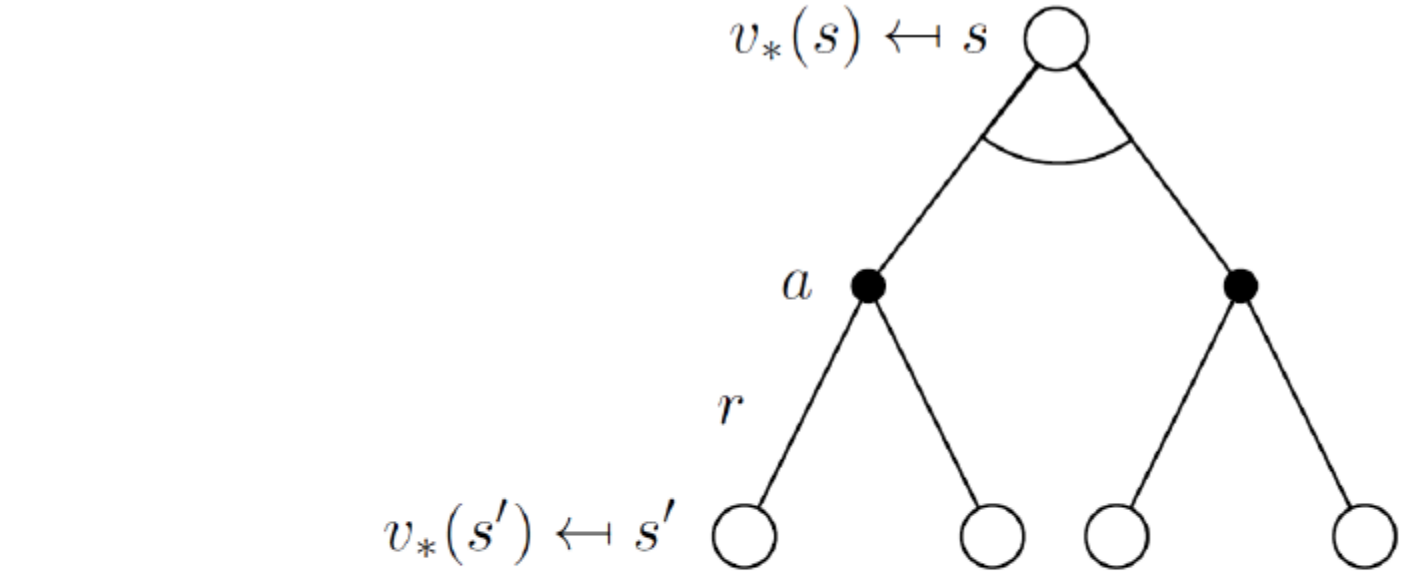
$$V^*(s) = \max_{a \in A} Q^*(s,a)$$

Уравнение Беллмана для Q*



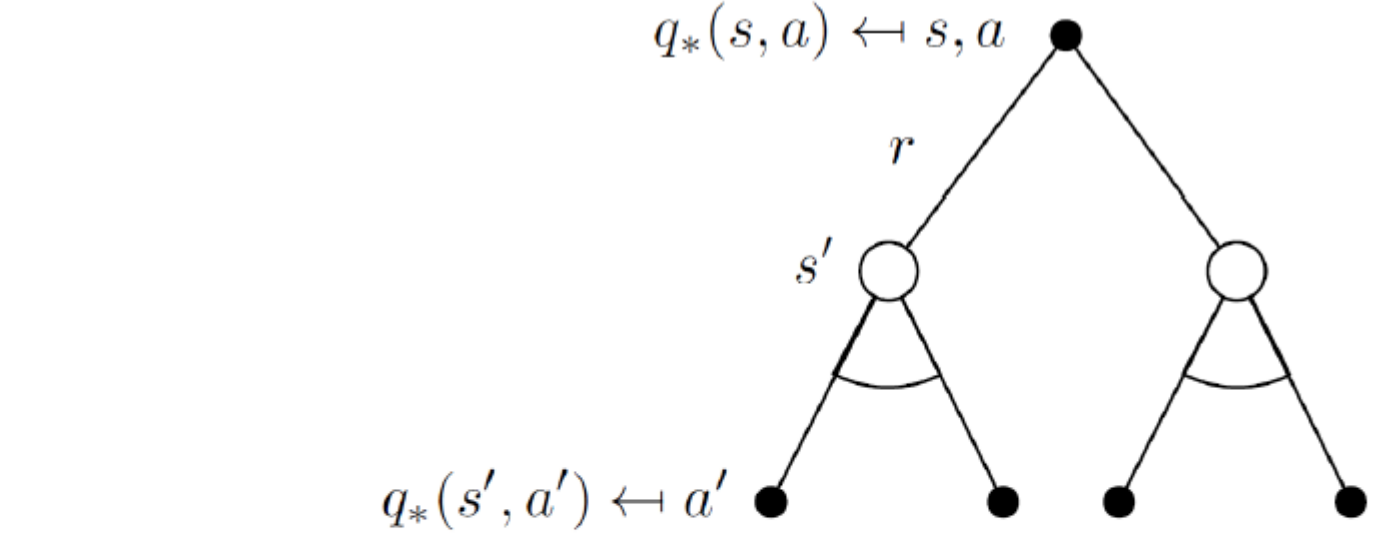
$$Q^*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V^*(s')$$

Уравнение Беллмана для V*



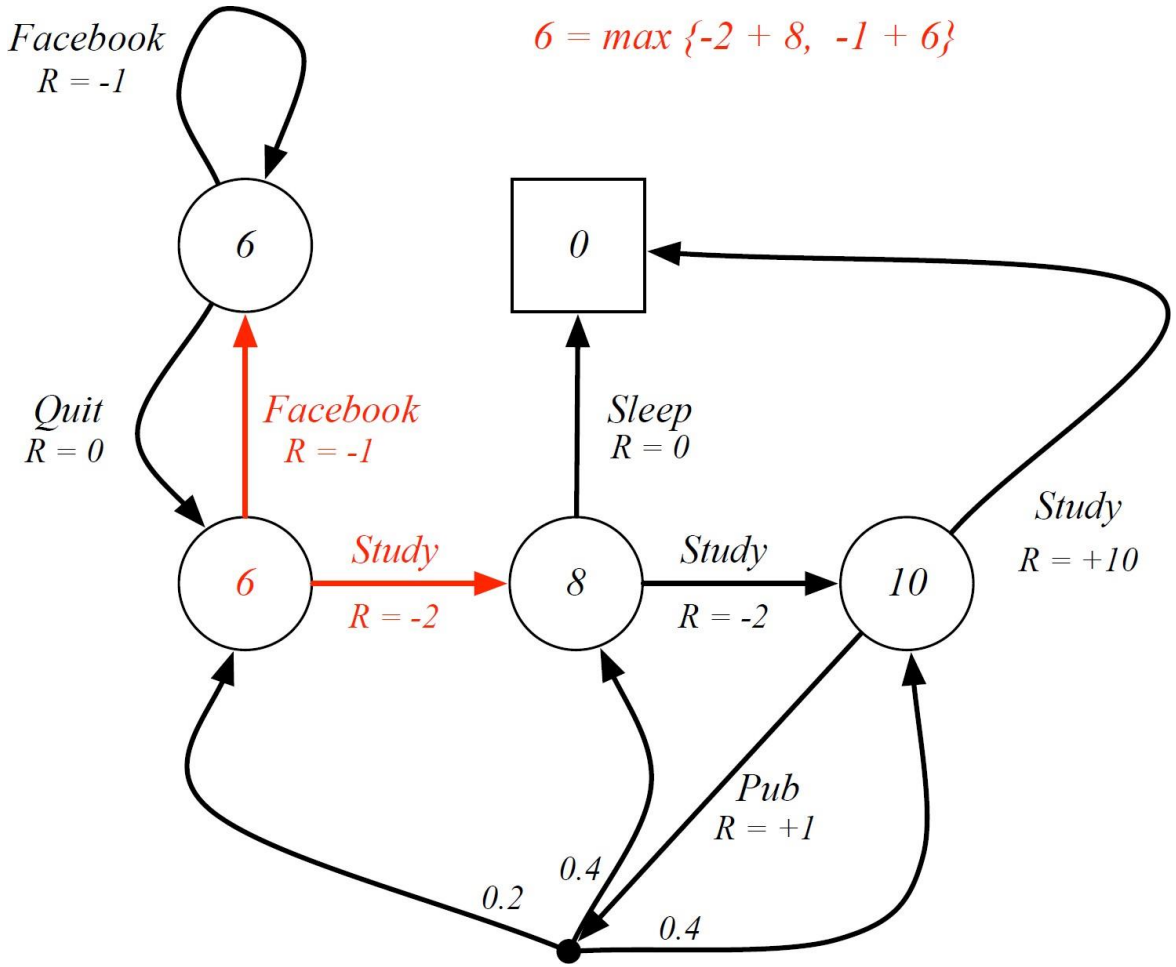
$$V^*(s) = \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V^*(s') \right)$$

Уравнение Беллмана для Q*



$$Q^*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \max_{a' \in A} Q^*(s',a')$$

Пример: уравнение оптимальности для StudMDP



Решение уравнения оптимальности Беллмана

Уравнение оптимальности Беллмана нелинейно. Аналитического решения в общем виде не существует. Много итерационных методов:

- итерации по полезностям,
- итерации по стратегии,
- Q-обучение,
- SARSA.

Алгоритм итерации по qvalue функции

Динамическое программирование

Динамическое программирование (dynamic programming, DP) – очень общий способ решения задач, обладающих двумя свойствами:

оптимальной структурой:

- применим принцип оптимальности,
- оптимальное решение может быть декомпозировано в подзадачи;

перекрывающиеся подзадачи:

- подзадачи повторяется многократно,
- решения могут сохранены и переиспользованы.

Марковский процесс принятия решений удовлетворяет обоим свойствам:

уравнение Беллмана задает рекурсивную структуру подзадач, функция полезности сохраняет и переиспользует решения.

Планирование с помощью DP

DP предполагает использование всей информации о MDP.

В реальных системах используется на этапе планирования.

Для задачи оценки:

$$\langle S, A, \mathcal{P}, \mathcal{R}, \gamma \rangle \rightarrow V^{\pi}$$

или

$$\langle S, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle \rightarrow V^{\pi}$$

Для задачи управления:

$$\langle S, A, \mathcal{P}, \mathcal{R}, \gamma \rangle \rightarrow \langle V^{*}, \pi^{*} \rangle$$

Итерационная оценка стратегии

Задача: оценить текущую стратегию π.

Решение: итеративное применение уравнения Беллмана в обратном направлении:

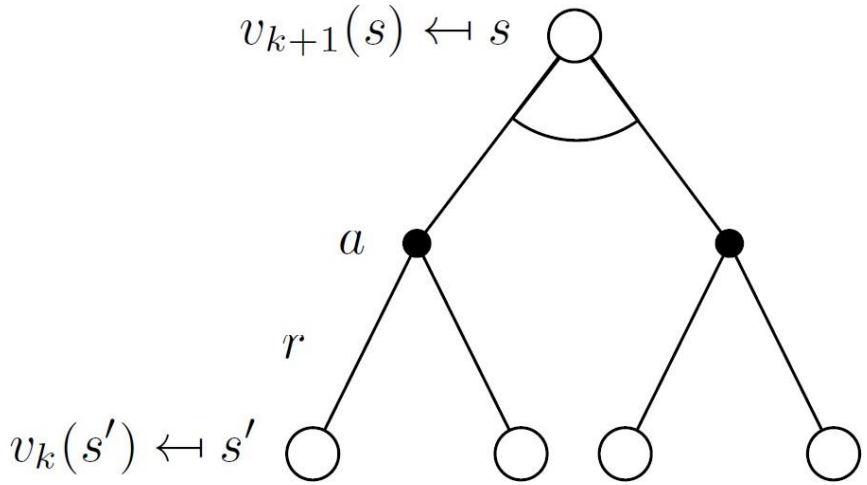
$$V^1 \rightarrow V^2 \rightarrow \cdots \rightarrow V^\pi$$

Использование *синхронных* шагов:

- для каждой итерации k + 1:
- для каждого состояния $s \in S$:
- обновить $V^{k+1}(s)$ по $V^k(s')$, где s' – следующее состояние после s.

Можно использовать *асинхронные* шаги. Сходится к истинным значениям V^π .

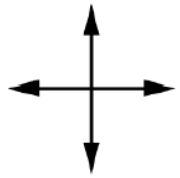
Итерационная оценка стратегии



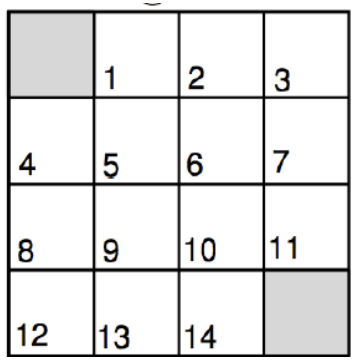
$$V^{k+1}(s) = \sum_{a \in A} \pi(a|s)\left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V^k(s')\right)$$

$$V^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi V^k$$

Пример: случайная стратегия в клеточном мире



actions



$r = -1$
on all transitions

Эпизодический MDP без дисконтирования ($\gamma = 1$).

Нетерминальные состояния 1, 2, . . . , 14.

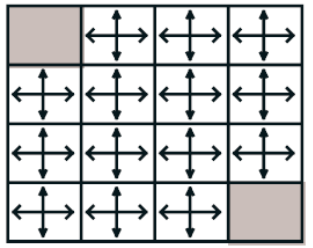
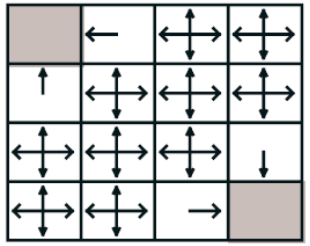
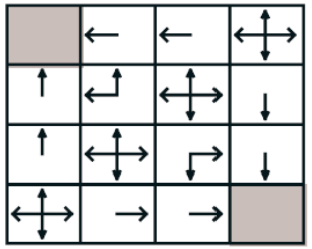
Одно терминальное состояние (серые квадраты).

Действия, ведущие за пределы карты не меняют состояния.

Агент следует случайной стратегии:

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(w|\cdot) = \pi(s|\cdot) = 0.25.$$

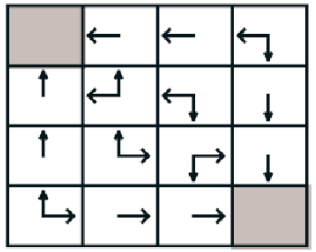
Пример: итерационная оценка стратегии для GW

	V^k для случайной стратегии	Жадная стратегия по V^k
$k = 0$	0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0	 *Случайная стратегия*
$k = 1$	0.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 0.0	
$k = 2$	0.0 -1.7 -2.0 -2.0 -1.7 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -1.7 -2.0 -2.0 -1.7 0.0	

Пример: итерационная оценка стратегии для клеточного мира

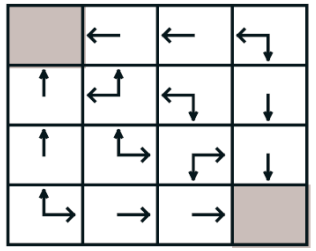
$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



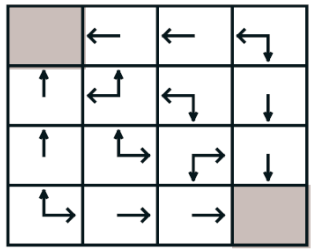
$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Улучшение стратегии

Дана стратегия π:

- *Оценить* (evaluate) стратегию π:

$$V^{\pi}(s) = \mathrm{E}[r_{t+1} + \gamma r_{t+2} + \ldots | s_t = s].$$

- *Улучшить* (improve) стратегию, выбирая действия жадно (greedy), но в соответствии с V^{π}:

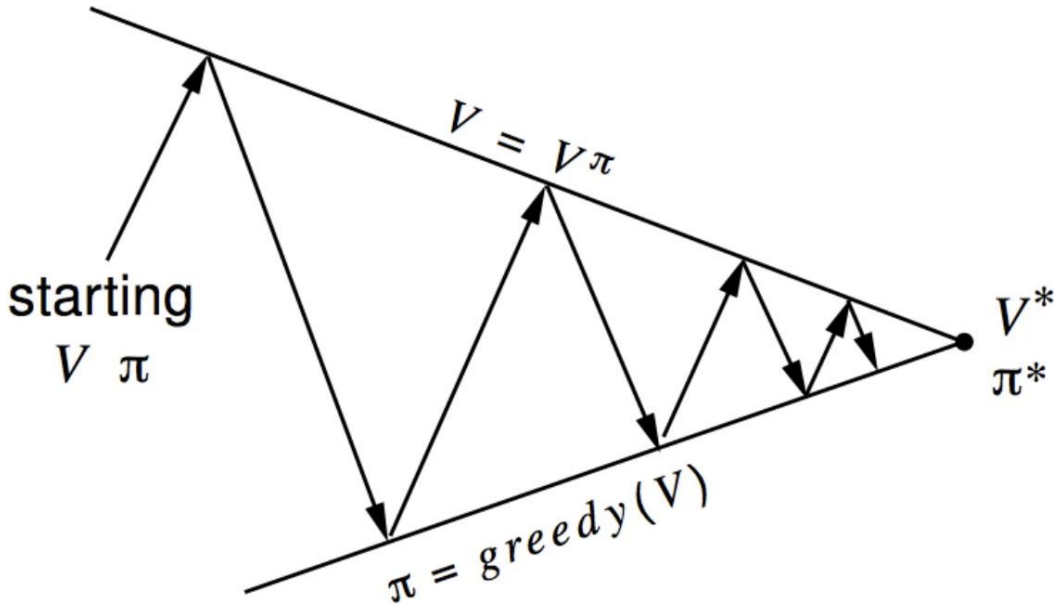
$$\pi' = greedy(V^{\pi})$$

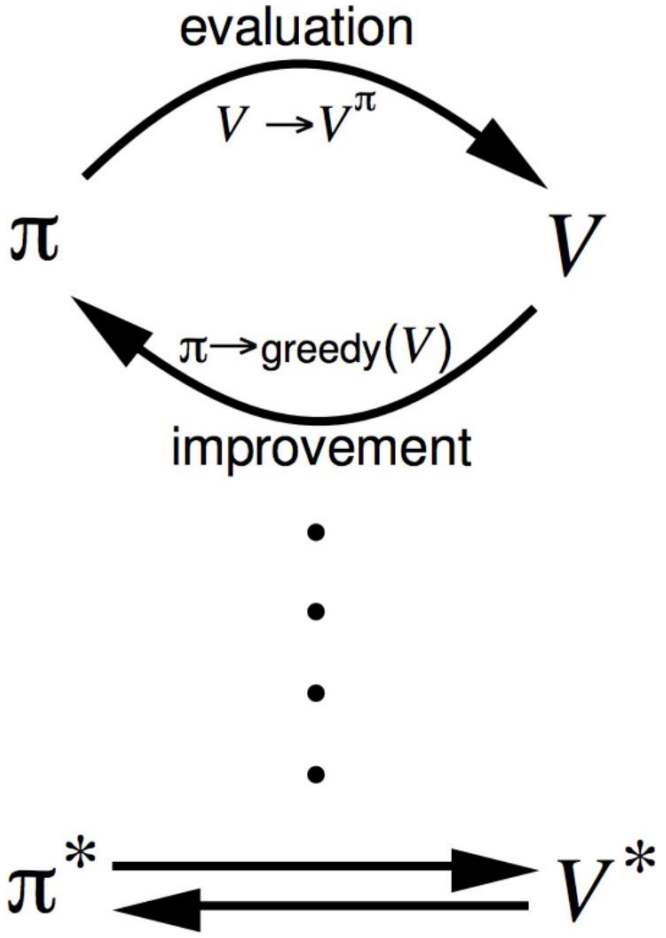
В клеточном мире улучшенная стратегия оказалась оптимальной: $\pi^{j} = \pi^{*}$.

В общем случае нужно больше итераций.

Однако этот процесс *итерации по стратегии* всегда сходится к оптимальной стратегии π^{*}.

Итерация по стратегиям (Policy Iteration - PI)





Оценка стратегии – вычисление V^{π}
Итеративная оценка стратегии

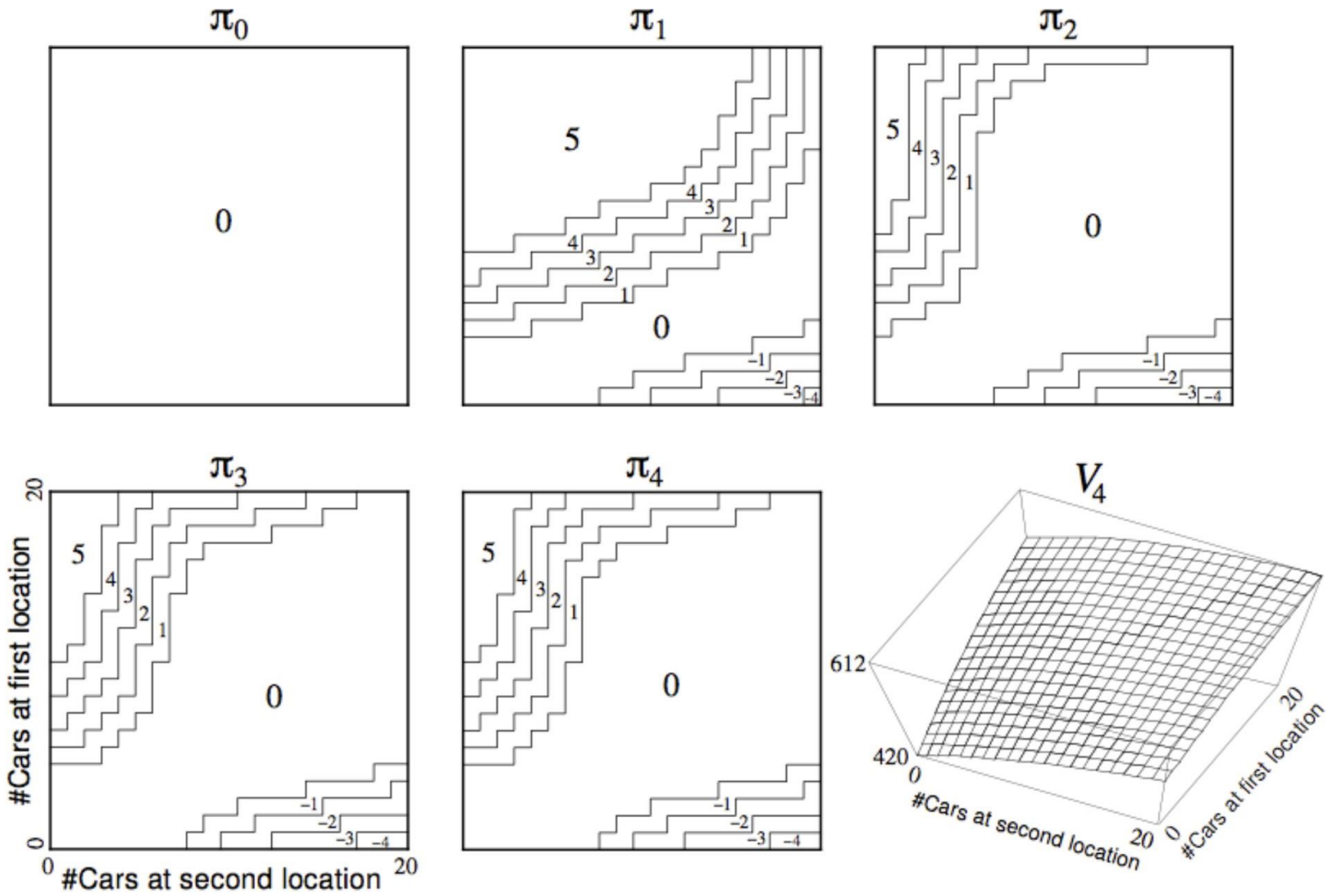
Улучшение стратегии – генерация $\pi^j \geq \pi$
Жадное обновление стратегии

Пример: аренда машин

Состояния: две локации, максимально по 20 машин в каждой. *Действия*: перегнать до 5 машин в течение ночи между локациями.
Вознаграждение: 10$ за каждую арендованную машину.
Переходы: машины возвращаются и арендуются случайно:

- пуассоновское распределение: n запросов/возвратов с вероятностями $\frac{\lambda^n}{n!}e^{-\lambda}$
- 1 локация: среднее кол-во запросов 3, среднее кол-во возвратов 3,
- 2 локация: среднее кол-во запросов 4, среднее кол-во возвратов 2.

Пример: итерация по стратегиям для аренды машин



Улучшение стратегии

Рассмотрим детерминированную стратегию $a = \pi(s)$. Мы можем улучшить стратегию, действуя жадно:

$$\pi'(s) = \arg\max_{a \in A} Q^{\pi}(s, a).$$

Это улучшает полезность от любого состояния s на один шаг

$$Q^{\pi}(s, \pi'(s)) = \max_{a \in A} Q^{\pi}(s, a) \geq Q^{\pi}(s, \pi(s)) = V^{\pi}(s).$$

Таким образом мы улучшаем функцию полезности: $V^{\pi'}(s) \geq V^{\pi}(s)$.

Улучшение стратегии

Когда процесс улучшения останавливается, мы получаем

$$Q^{\pi}(s, \pi^{j}(s)) = \max_{a \in A} Q^{\pi}(s, a) = Q^{\pi}(s, \pi(s)) = V^{\pi}(s).$$

Тогда уравнение оптимальности Беллмана будет удовлетворено:

$$V^{\pi}(s) = \max_{a \in A} Q^{\pi}(s, a)$$

А это означает, что $V^{\pi}(s) = V^{*}(s)$ для всех $s \in S$.

Стратегия π будет являться оптимальной.

Алгоритм итерации по полезностям

Принцип оптимальности

Любая оптимальная стратегия может быть разделена на две части:

оптимальный первый шаг a^*,

следование оптимальной стратегии, начиная со следующего состояния s'.

Теорема (Принцип оптимальности)

Стратегия $\pi(a|s)$ *достигает оптимальной оценки состояния* s
$V^{\pi}(s) = V^*(s)$, *если и только если для любого* s', *достижимого из* s, π
достигает оптимальной оценки состояния s': $V^{\pi}(s') = V^*(s')$.

Детерминированные итерации по полезностям

Пусть мы знаем решение для подзадачи $V^*(s')$, тогда мы можем найти решение за один шаг:

$$V^*(s) \leftarrow \max_{a \in A} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V^*(s') \right)$$

Идея итераций по полезностям – применять эти обновления рекурсивно.

Интуиция: начать с конечных вознаграждений и двигаться назад.

Пример: кратчайший путь

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Итерация по полезностям

Задача: найти оптимальную стратегию π^*.

Решение: итеративное применение **уравнения оптимальности Беллмана** в обратном направлении:

$$V^1 \to V^2 \to \cdots \to V^*$$

Использование *синхронных* шагов:

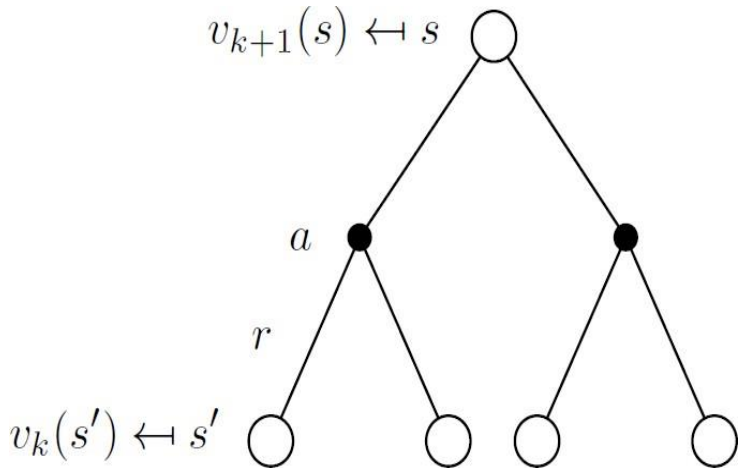
- для каждой итерации k + 1:
- для каждого состояния s ∈ S
- обновить $V^{k+1}(s)$ по $V^k(s')$, где s' – следующее состояние после s.

Сходится к истинным значениям V^*.

В отличие от итерации по стратегиям мы не получаем стратегию в явном виде.

Промежуточные значения полезностей могут не соответствовать ни одной стратегии.

Итерация по полезностям



$$V^{k+1}(s) = \max_{a \in A} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a V^k(s') \right)$$

$$V^{k+1} = \max_{a \in A} \mathcal{R}^a + \gamma \mathcal{P}^a V^k$$

$V^{k+1} = \mathrm{B} V^k$ – опреатор Беллмана (Bellman backup operator)

Синхронные алгоритмы DP

Задача	Уравнение Беллмана	Алгоритм
Оценка	для матожиданий	итеративная оценка стратегии
Управление	для матожиданий + жадное улучшение стратегии	итерация по стратегиям
Управление	оптимальное	итерация по полезностям

При использовании функции полезности $V^{\pi}(s)$ или $V^{*}(s)$ сложность итерации равна $O(mn^2)$, где m – количество действий, n – количество состояний.

При использовании функции полезности $Q^{\pi}(s, a)$ или $Q^{*}(s, a)$
сложность итерации равна $O(m^2n^2)$.

Итог

Марковский процесс принятия решений – формальное описание процесса взаимодействия агенты и среды

Ключевое понятие динамического программирование – уравнение Беллмана

Основная задача – поиск оптимальной функции стратегии или стратегии

Аналитически решить уравнение оптимальности Беллмана невозможно – используем приближенные методы

Базовые алгоритмы при известной функции переходов: итерации по стратегиям и полезностям

Обучение с подкреплением: определения и примеры

План

- Метод Монте-Карло
- Метод временных различий
- Обобщенный подход
- Безмодельное управление
- Обучение по чужому опыту
- Q-обучение

Метод Монте-Карло
(Monte-Carlo)

Подход к обучению Монте-Карло

- Value-based
- Обучение по полным эпизодам (нет бутстрэп выборок)
- Обучение после непосредственного взаимодействия со средой
- Модель переходов и функция вознаграждения неизвестны (не используются), т.е. model-free
- Может быть применен только к эпизодическим процессам (не обязательно MDP)
- Подход Монте-Карло использует идею: полезность = средняя отдача (скор (score) - средняя дисконтированная кумулятивная награда)

Монте-Карло оценка стратегии

Цель: улучшать V^{π} по траекториям (состоящим из полных эпизодов) полученным стратегией π:

$$s_1, a_1, r_1, \ldots s_k \sim \pi$$

Отдача (*total return*) – это дисконтированная кумулятивная награда:

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

Обычно value функция перевычисляется через матожидание:

$$V^{\pi}(s) = \mathrm{E}_{\pi}[R_t \mid s_t = s].$$

А в Монте-Карло через *эмпирическое среднее*

Монте-Карло оценка стратегии с **первым** посещением

value функцию оцениваем как среднее арифметическое отдач: $V(s) = Sum(s)/N(s)$

для этого для каждого эпизода если в нем встретилось состояние s накапливаем:
$N(s) \leftarrow N(s) + 1$ в скольких эпизодах встречалось это состояние
$Sum(s) \leftarrow Sum(s) + R_t$ сумма отдач всех траекторий отсчитываемых с **первого** появления этого состояния

По закону больших чисел:

$$V(s) \xrightarrow{N(s)\to\infty} V^{\pi}(s)$$

Несмещенная, состоятельная оценка с высокой дисперсией

**Состоятельность – сходимость к ней при большой выборке.*
**Смещенность (bies) – текущее матожидание оценки отличается от истинного значения величины.*

Монте-Карло оценка стратегии с **каждым** посещением

value функцию оцениваем как среднее арифметическое отдач: $V(s) = Sum(s)/N(s)$

для этого для каждого эпизода если в нем встретилось состояние s накапливаем:
$N(s) \leftarrow N(s) + 1$ в скольких эпизодах встречалось это состояние
$Sum(s) \leftarrow Sum(s) + R_t$ сумма отдач всех траекторий отсчитываемых с **каждого** появления этого состояния

По закону больших чисел:

$$V(s) \xrightarrow{N(s)\to\infty} V^{\pi}(s)$$

Смещенная, состоятельная оценка с более низкой дисперсией

**Состоятельность – сходимость к ней при большой выборке.*
**Смещенность (bies) – текущее матожидание оценки отличается от истинного значения величины.*

Пример: блек-джек

200 состояний:

- текущая сумма (12-21),
- дилер показывает карту (максимально – 10 очков),
- есть ли особая комбинация (да-нет)

Действия: **stick** - не получать карты (завершить игру) и **twist** - взять новую карту (без замены)

Вознаграждения за stick:

- +1, если сумма карт больше, чем у дилера,
- 0, если сумма карт та же, чем у дилера,
- -1, если сумма карт меньше, чем у дилера

Вознаграждения за twist:

- -1, если сумма карт больше 21,
- 0 иначе.

Переходы: автоматически twist, если сумма карт меньше 12

Среднее приращений

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j$$

$$= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right)$$

$$= \frac{1}{k}\left(x_k + (k-1)\mu_{k-1}\right)$$

$$= \mu_{k-1} + \frac{1}{k}\left(x_k - \mu_{k-1}\right)$$

Монте-Карло для приращений

Обновим $V(s)$ с приращением (incrementally) для траектории

$$s_1, a_1, r_1, \ldots, s_T .$$

Для каждого состояния s_t (будем говорить что у него отдача R_t) выполним:

$$N(s_t) \leftarrow N(s_t) + 1,$$

$$V(s_t) \leftarrow V(s_t) + (R_t - V(s_t))/N(s_t)$$

В нестационарных задачах может быть полезно отслеживать текущее среднее:

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t - V(s_t))$$

Метод временных различий (temporal difference)

Обучение на основе временных различий

- Обучение может происходить до окончания эпизода взаимодействия со средой
- Используем безмодельный подход (model-free): модель переходов МППР и функция вознаграждения не известны
- Будем рассматривать неполные эпизоды, используя бутстреп (bootstrappig, раскрутка) для получения информации об оставшихся будущих шагах
- Идея метода временных различий (temporal-difference, ВР): приближать значение полезности на основе предыдущего приближения

МК vs. ВР

- Цель: построить V^{π} интерактивно (online) по эпизодам взаимодействия по стратегии π
- Монте-Карло с каждым посещением для приращений: обновляем $V(s_t)$ на основе текущей отдачи R_t

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t - V(s_t))$$

- Самый простой подход временных различий: TD(0):
  - обновляем на основе *ожидаемой* отдачи $r_{t+1} + \gamma V(s_{t+1})$

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)),$$

  - $r_{t+1} + \gamma V(s_{t+1})$ называется *TD показателем*,
  - $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ называется *TD ошибкой*

Преимущества и недостатки МК и ВР

МК обладает высокой дисперсией и нулевым смещением:

- может обучаться только на полных эпизодах,
- работает только в эпизодических окружениях (с терминальными состояниями),

хорошие показатели сходимости (даже без аппроксимации),
не очень сильно зависит от начального приближения, очень прост для понимания и использования,

ВР имеет низкую дисперсию, ненулевое смещение:

может обучаться интерактивно на каждом шаге,
работает и для бесконечных (без терминального состояния) окружений
обычно более эффективен, чем МК,

- TD(0) сходится к $V^{\pi}(s)$ (но не всегда при использовании аппроксимации),
- более чувствителен к начальному приближению

Смещенность и дисперсия

Отдача $R_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-1} r_T$ является несмещенной оценкой для $V^\pi(s)$

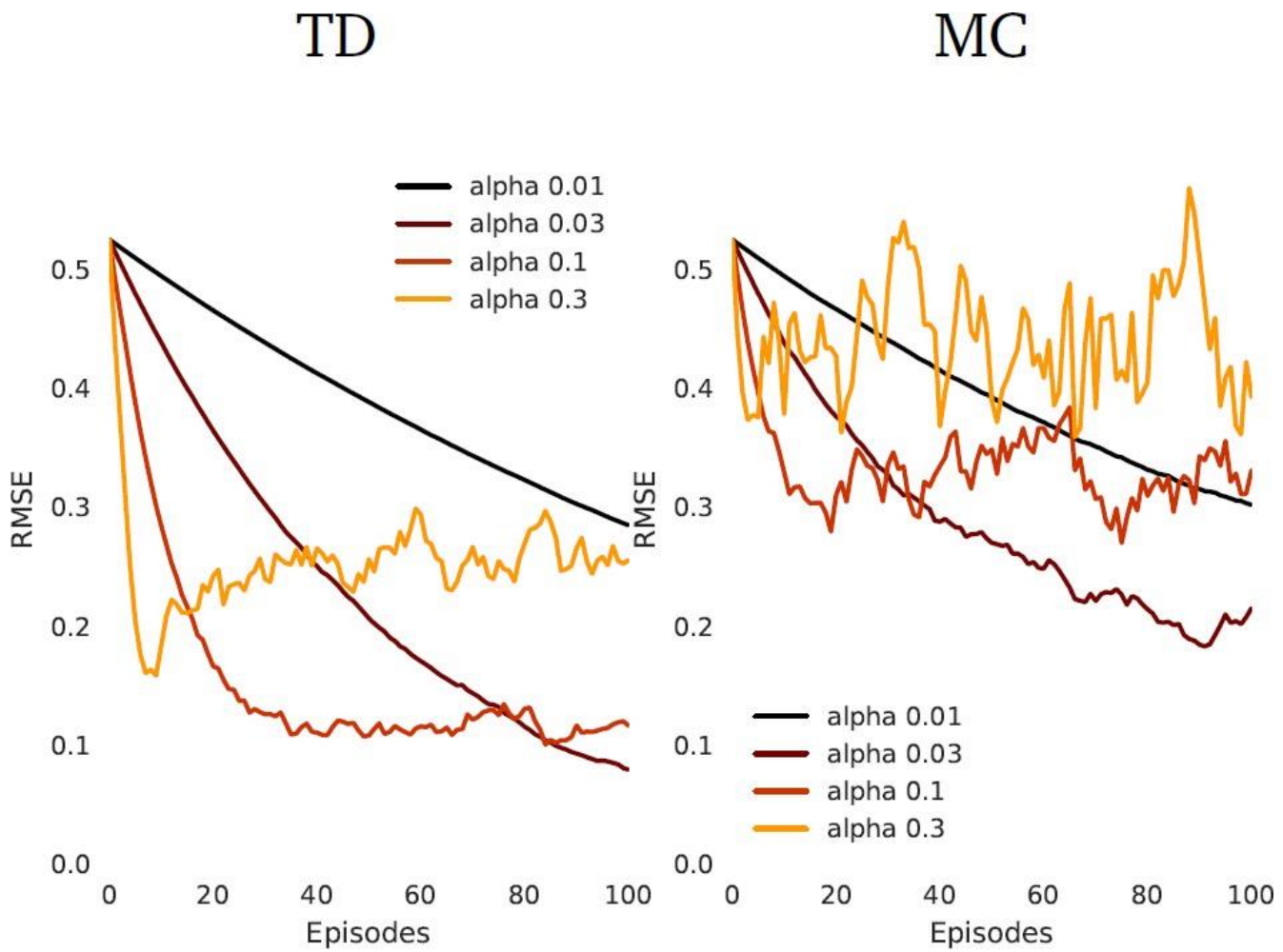
Истинный TD показатель $r_{t+1} + \gamma V^\pi(s_{t+1})$ является несмещенной оценкой для $V^\pi(s)$

TD показатель $r_{t+1} + \gamma V(s_{t+1})$ является смещенной оценкой для $V^\pi(s)$

TD показатель имеет меньшую дисперсию, чем отдача:

- отдача зависит от *большого* количества случайных действий, переходов, вознаграждений,
- показатель зависит только от *одного* случайного действия, перехода и вознаграждения

Пример: МК vs. ВР



Пакетные МК и ВР

МК и ВР сходятся к $V(s) \rightarrow V^{\pi}(s)$, если опыт $\rightarrow \infty$

А если мы применим пакетный (batch) подход для конечного опыта?

$$s_1^1, a_1^1, r_1^1, \ldots, s_{T_1}^1$$

$$\vdots$$

$$s_1^K, a_1^K, r_1^K, \ldots, s_{T_K}^K$$

Например, многократно выбирать эпизод k ∈ [1, K]

Применять MC или TD(0) к эпизоду k

Пример: AB

Два состояния A,B; нет дисконтирования, 8 эпизодов опыта:

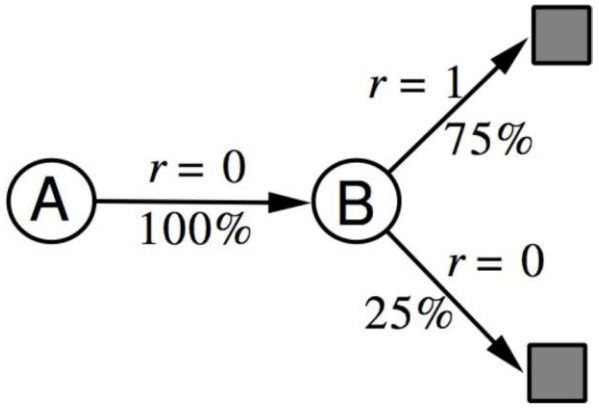
A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

Какие значения $V(A)$ и $V(B)$?

Пример: AB

Два состояния A,B; нет дисконтирования, 8 эпизодов опыта:

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



Какие значения $V(A)$ и $V(B)$?

Эквивалентность

МК сходится к решению с минимальной среднеквадратичной ошибкой

- Наилучшее приближение к наблюдаемой отдаче:

$$\sum_{k=1}^{K}\sum_{t=1}^{T_k}(R_t^k - V(s_t^k))^2$$

Для AB примера $V(A) = 0$.

TD(0) сходится к решению максимально правдоподобной марковской модели

- Решение для МППР $\langle S, A, \hat{\mathrm{P}}, \hat{\mathrm{R}}, \gamma\rangle$, который лучше всего удовлетворяет данным

$$\hat{\mathcal{P}}_{ss'}^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}1(s_t^k, a_t^k, s_{t+1}^k = s, a, s'),$$

- Для AB примера $V(A) = 0.75$

$$\hat{\mathcal{R}}_s^a = \frac{1}{N(s,a)}\sum_{k=1}^{K}\sum_{t=1}^{T_k}1(s_t^k, a_t^k = s, a)r_t^k$$

Преимущества и недостатки МК и ВР

ВР использует марковское свойство:

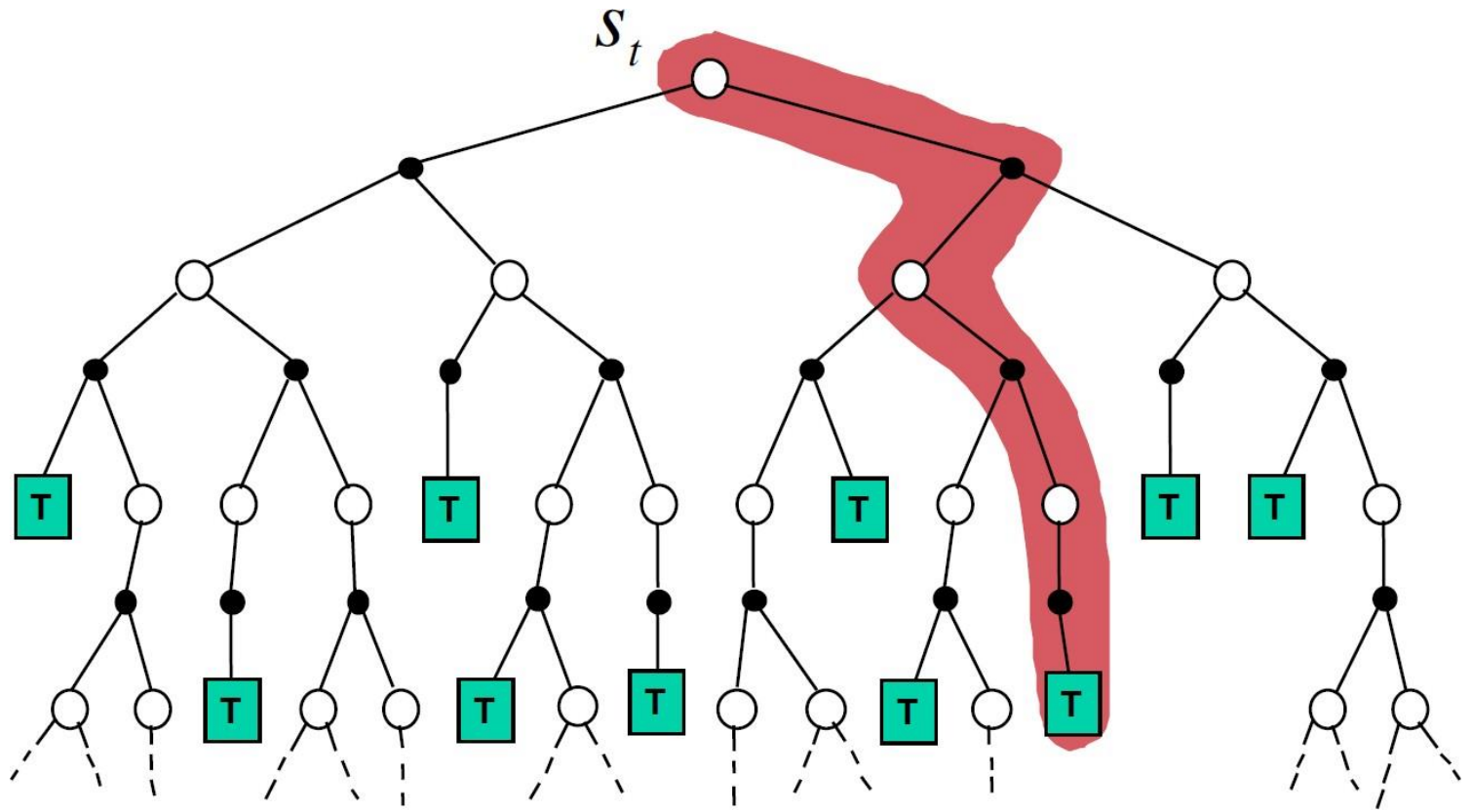
- обычно более эффективен в марковских окружениях

МК не использует марковское свойство:

- обычно более эффективен в немарковских окружениях

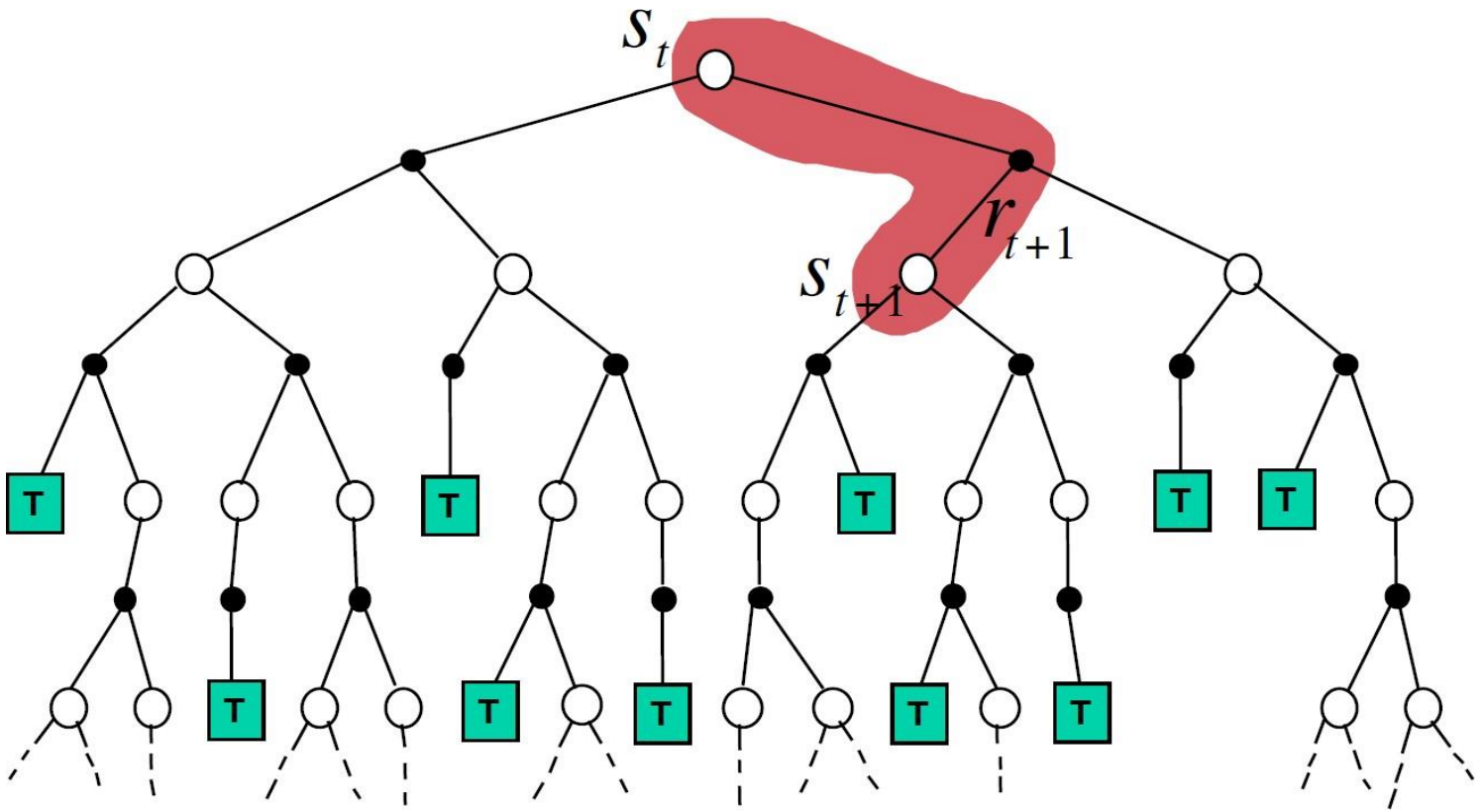
Монте-Карло обновление

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t - V(s_t))$$



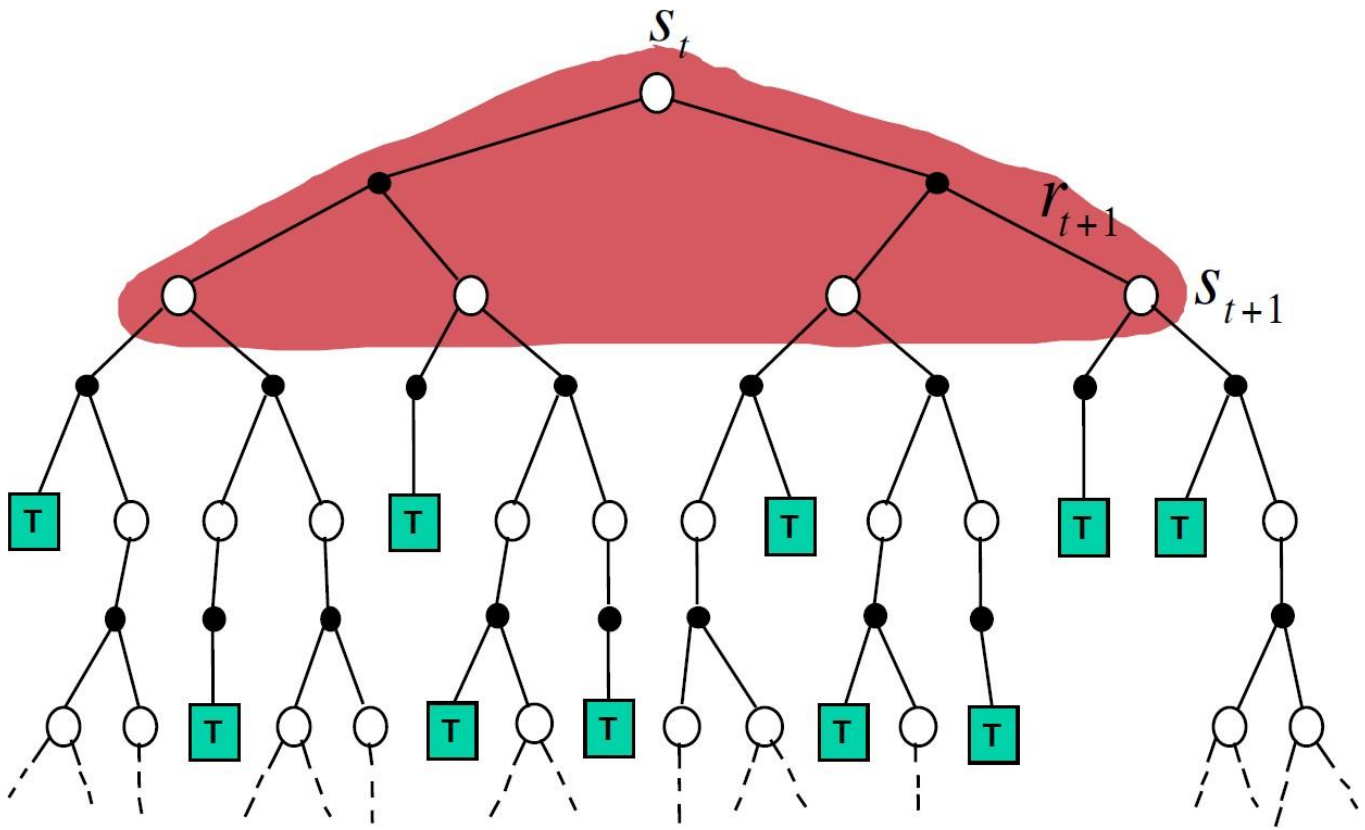
Обновление временных различий

$$V(s_t) \leftarrow V(s_t) + \alpha(r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

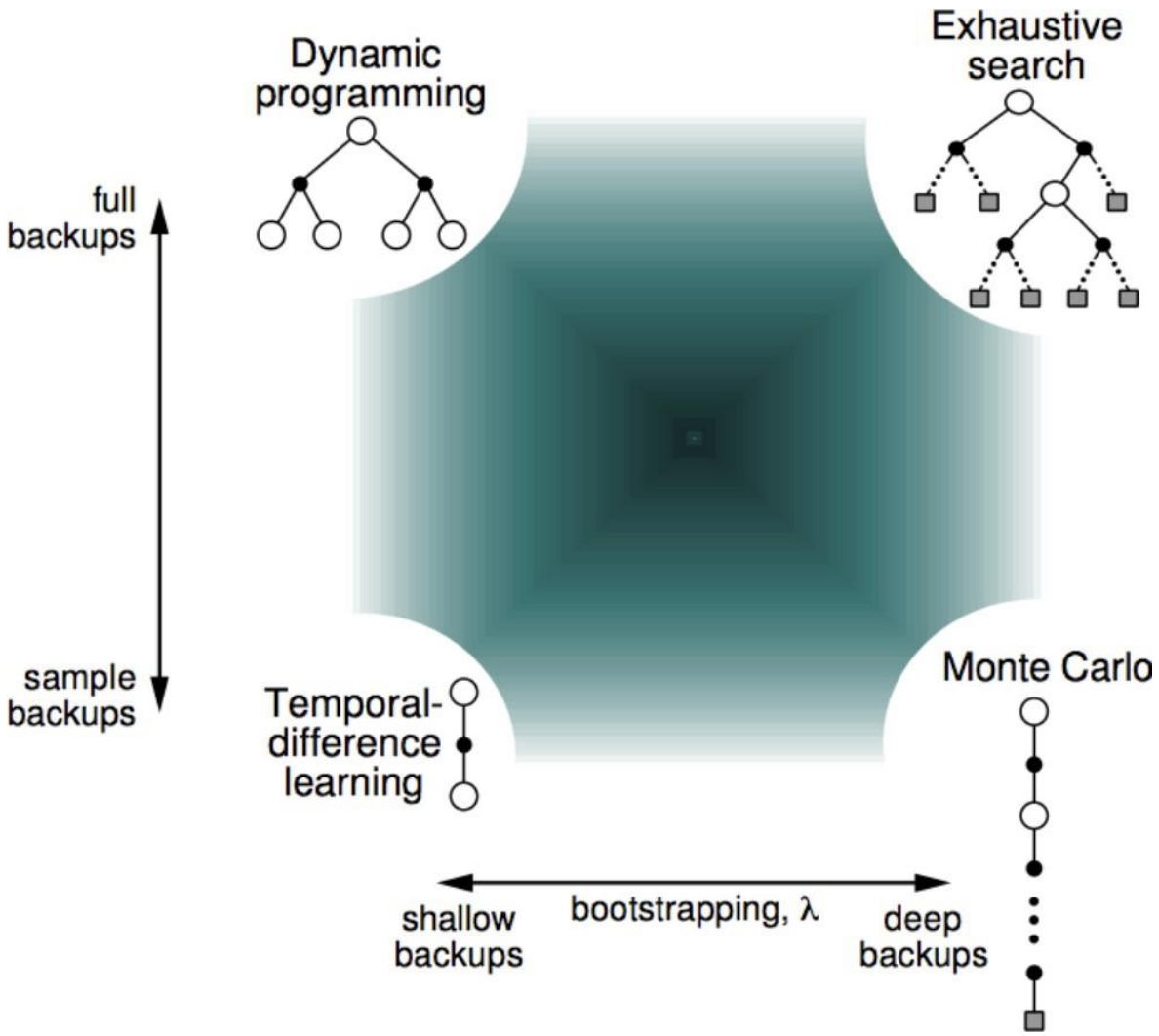


Обновление динамического программирования

$$V(s_t) \leftarrow \mathrm{E}_\pi[r_{t+1} + \gamma V(s_{t+1})]$$



Обобщенный подход к обучению с подкреплением



Безмодельное управление

Безмодельное обучение с подкреплением по актуальному и отложенному опыту

Безмодельное предсказание (model-free prediction): оценка функции полезности по неизвестному МППР

Безмодельное управление (model-free control): оптимизация функции полезности по неизвестному МППР

Обучение **по актуальному опыту** (on-policy):

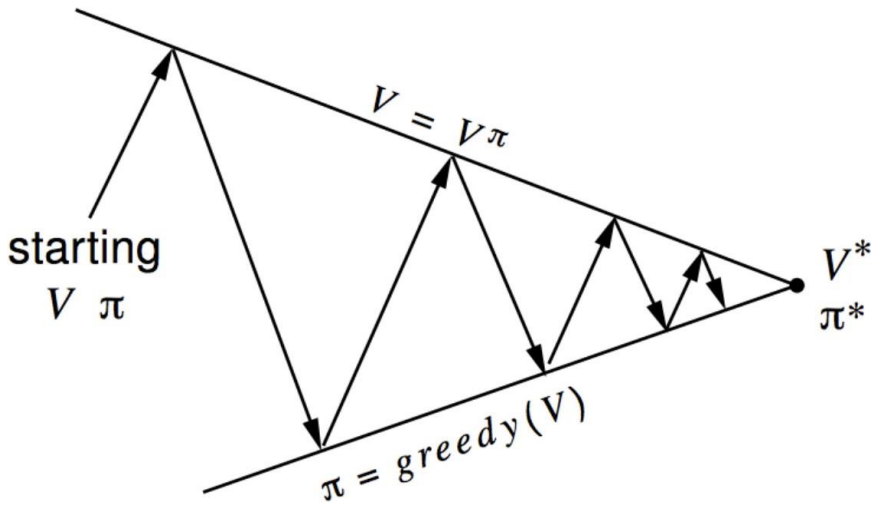
- “обучение по ходу дела”,
- обучение стратегии π используя опыт, полученный на основе π

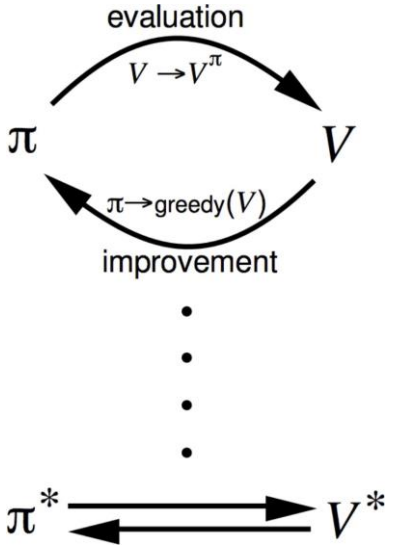
Обучение **по отложенному опыту** (off-policy):

- “обучение на чужих ошибках”,
- обучение стратегии π используя опыт, полученный на основе другой стратегии μ

Обобщенные итерации по стратегиям

Оценка стратегии – вычисление V^{π}
Итеративная оценка стратегии **Улучшение стратегии** – генерация $\pi' \geq \pi$
Жадное обновление стратегии





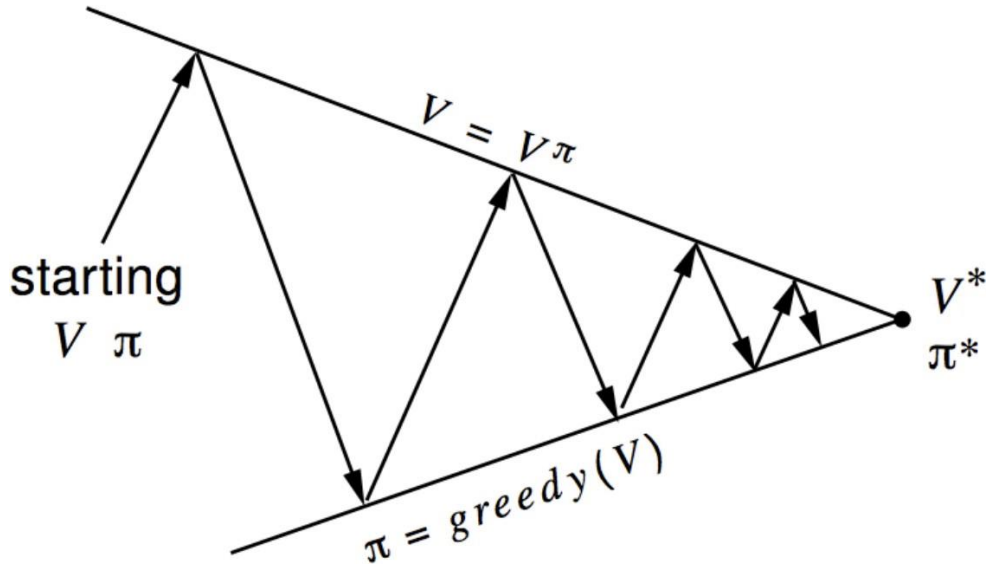
Обобщенные итерации по стратегиям с Монте-Карло оценкой

Оценка стратегии

Монте-Карло оценка стратегии по $V = V^{\pi}$?

Улучшение стратегии

Жадное обновление стратегии?



Безмодельная итерация по стратегиям с функцией полезности действий

Жадное улучшение стратегии по $V(s)$ требует знания модели МППР:

$$\pi'(s) = \arg\max_{a \in A} \left(\mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s') \right)$$

Жадное обновление стратегии по $Q(s, a)$ не требует знания модели:

$$\pi'(s) = \arg\max_{a \in A} Q(s, a)$$

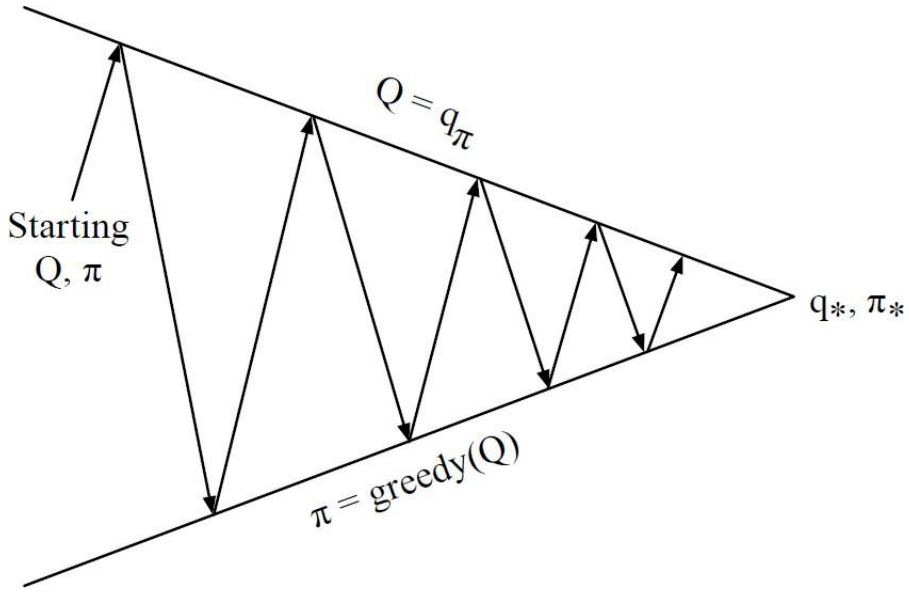
Обобщенные итерации по стратегиям с функцией полезности действий

Оценка стратегии

Монте-Карло оценка $Q = Q^{\pi}$

Улучшение стратегии

Жадное обновление стратегии?



Пример жадного выбора действий

"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

Перед вами две двери.

Вы открываете левую дверь и получаете вознаграждение 0 $V(left) = 0$.

Вы открываете правую дверь и получаете вознаграждение +1 $V(right) = 1$.

Вы открываете правую дверь и получаете вознаграждение +2 $V(right) = 2$.

Вы открываете правую дверь и получаете вознаграждение +2 $V(right) = 2$.

Вы уверены, что выбирали лучшую дверь?

ϵ-жадное исследование

Простейшая идея, обеспечивающее постоянное исследование среды.

Все действия выбираются с ненулевой вероятностью. С вероятностью ϵ – 1выбираем действие жадно.

С вероятностью ϵ выбираем действие случайно.

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{если } a^* = \arg\max_{a \in A} Q(s,a) \\ \epsilon/m, & \text{иначе.} \end{cases}$$

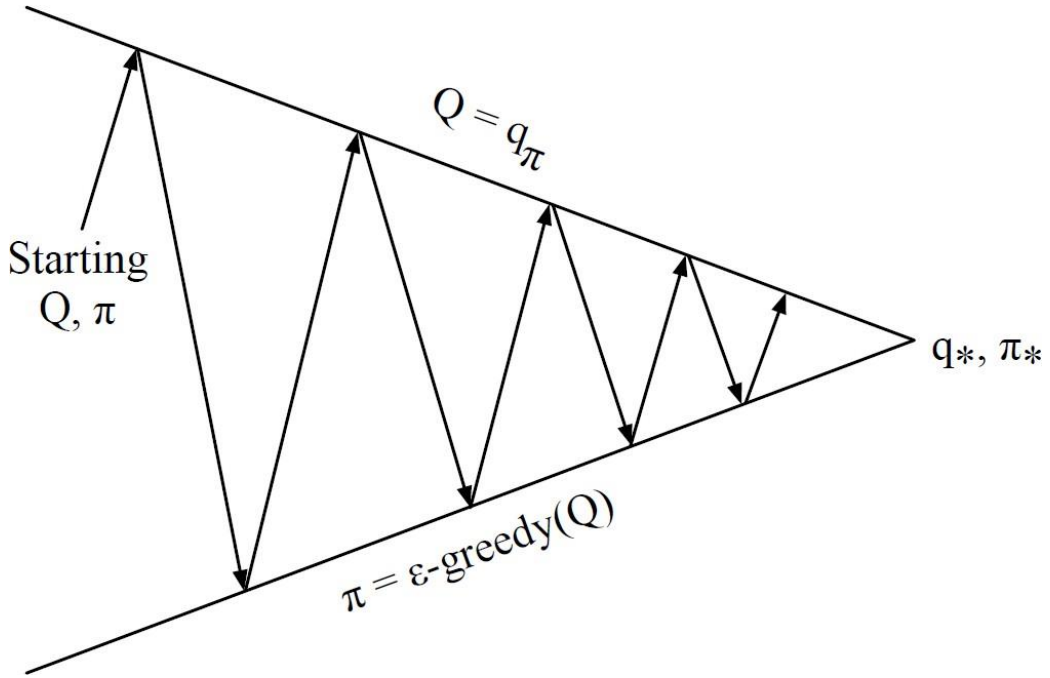
Монте-Карло итерация итерация по стратегиям

Оценка стратегии

Монте-Карло оценка стратегии $Q = Q^{\pi}$

Улучшение стратегии

ϵ-жадное улучшение стратегии



Монте-Карло управление

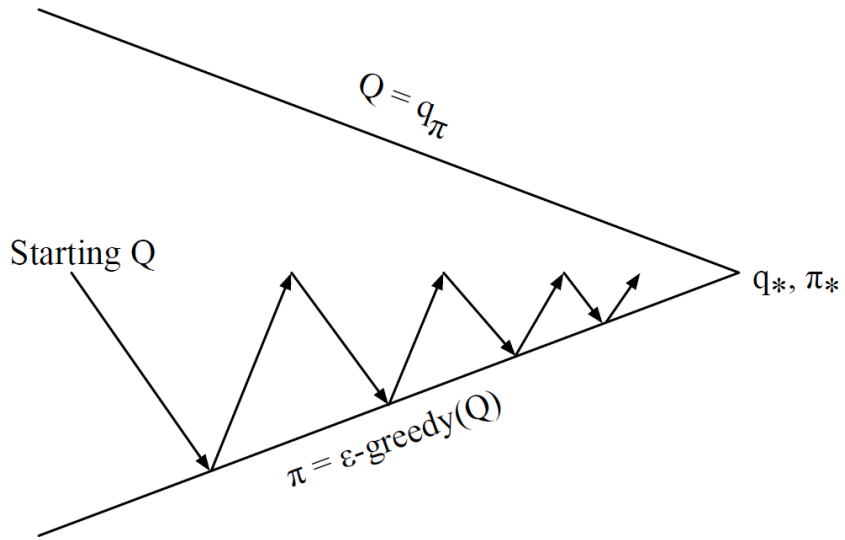
Для каждого эпизода:

Оценка стратегии

Монте-Карло оценка стратегии $Q \approx Q^{\pi}$

Улучшение стратегии

ϵ-жадное обновление стратегии



Алгоритм Монте-Карло управления

Выбираем k-ый эпизод $(s_1, a_1, r_2, \ldots, s_T) \sim \pi$
Для каждой пары состояния и действия в эпизоде выполняем

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1,$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N(s_t, a_t)}(R_t - Q(s_t, a_t))$$

Улучшаем стратегию на основе новых значений полезности:

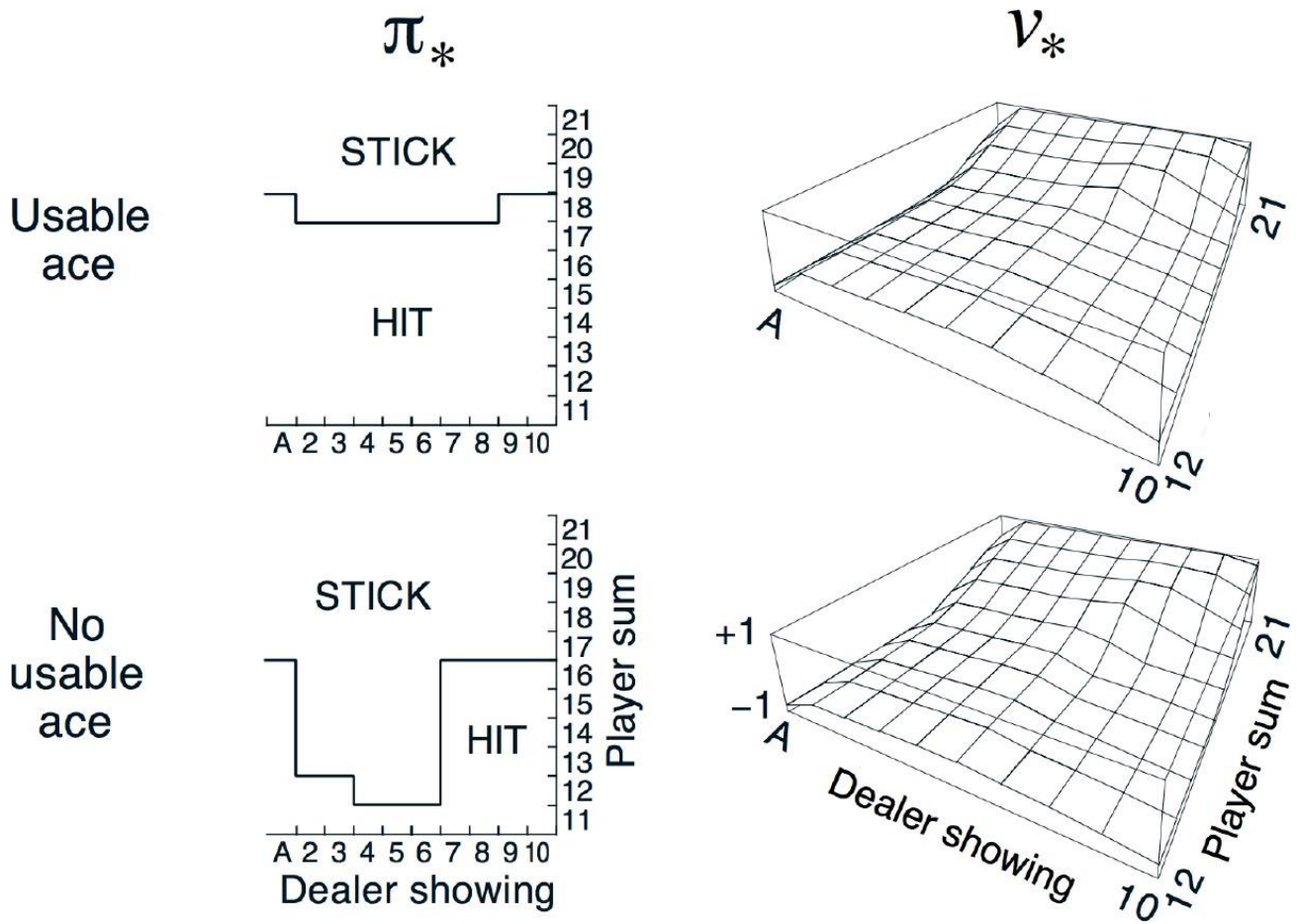
$$\epsilon \leftarrow 1/k,$$

$$\pi \leftarrow \epsilon\text{-жадная}(Q)$$

Этот алгоритм сходится к оптимальному решению

$$Q(s, a) \rightarrow Q^*(s, a)$$

Пример: МК и блек-джек



МК vs. ВР управление

TD имеет несколько преимуществ относительно МК:

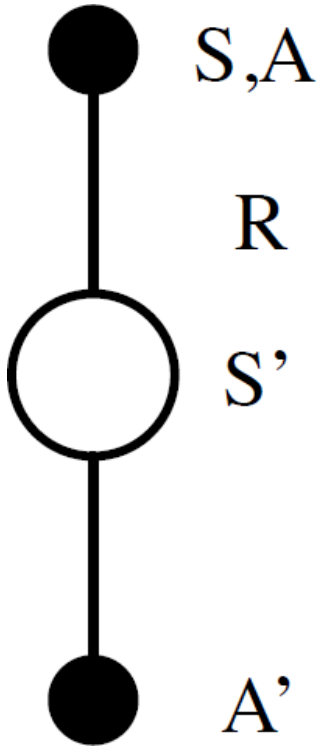
- меньше дисперсия,
- интерактивное,
- неполные последовательности

Естественная идея: использовать ВР вместо МК в цикле управления:

- применить TD к $Q(s, a)$,
- использовать ϵ-жадное улучшение,
- обновлять на каждом шаге

Обновление qvalue функции по SARSA

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$



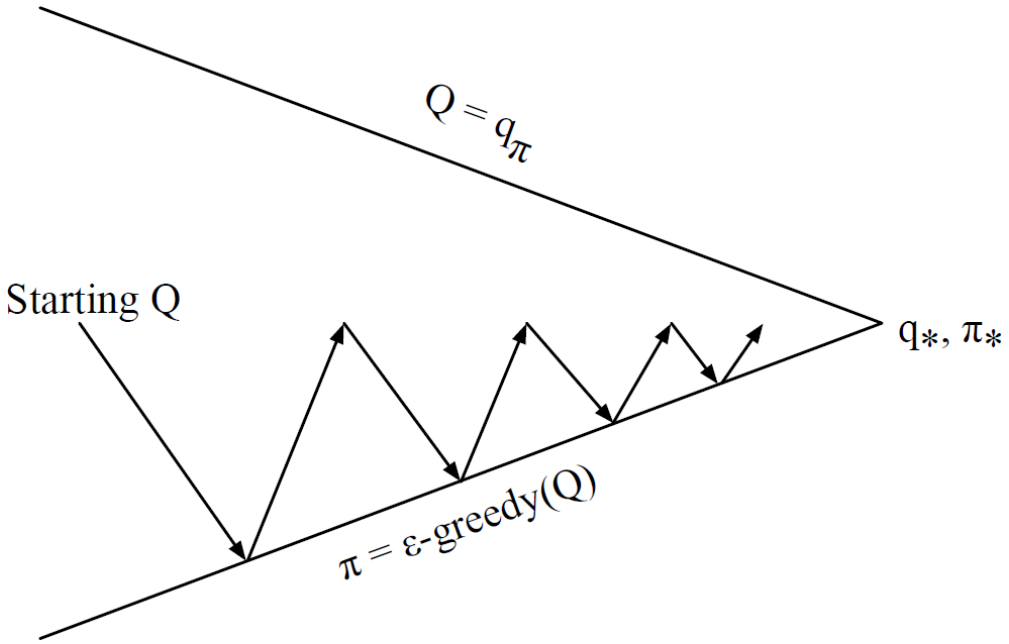
Управление по актуальному опыту с SARSA

Для каждого временного шага: Оценка стратегии

SARSA $Q \approx Q^{\pi}$

Улучшение стратегии

ϵ-жадное обновление стратегии



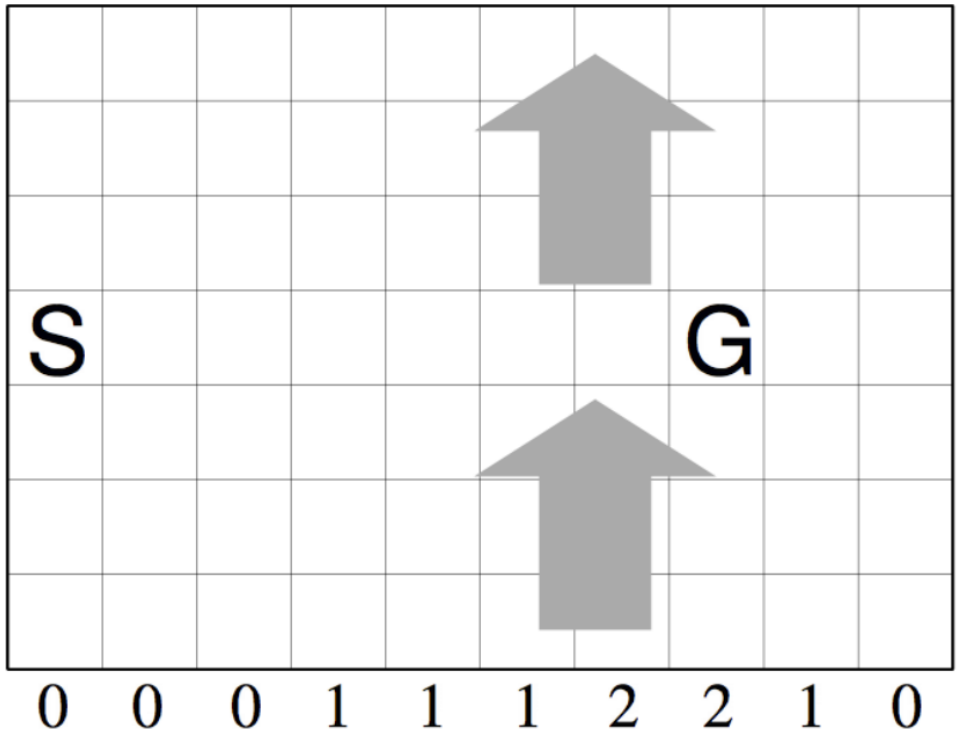
SARSA алгоритма для управления по актуальному опыту

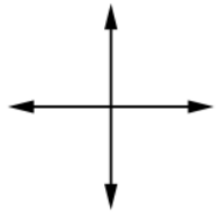
Algorithm 1 SARSA с ϵ-жадной стратегией

1: $MDP =< S, A, \gamma >$ ▷ задача МППР
2: $Q \leftarrow [0, \dots]$; ▷ таблица значений полезности действий
3: **for all** эпизодов **do**
4: s - начальное состояние;
5: $a \leftarrow \epsilon$-жадная $\pi(s)$
6: **for all** шагов эпизода **do**
7: $r, s' \leftarrow$ применение a;
8: $a' \leftarrow \epsilon$-жадная $\pi(s')$;
9: $Q[s, a] \leftarrow Q(s, a) + \alpha(r + \gamma Q[s', a'] - Q[s, a])$;
10: $s \leftarrow s', a \leftarrow a'$;

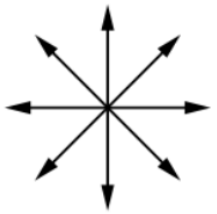
Пример: ветреный клеточный мир

Вознаграждение -1 за каждый шаг Без дисконтирования



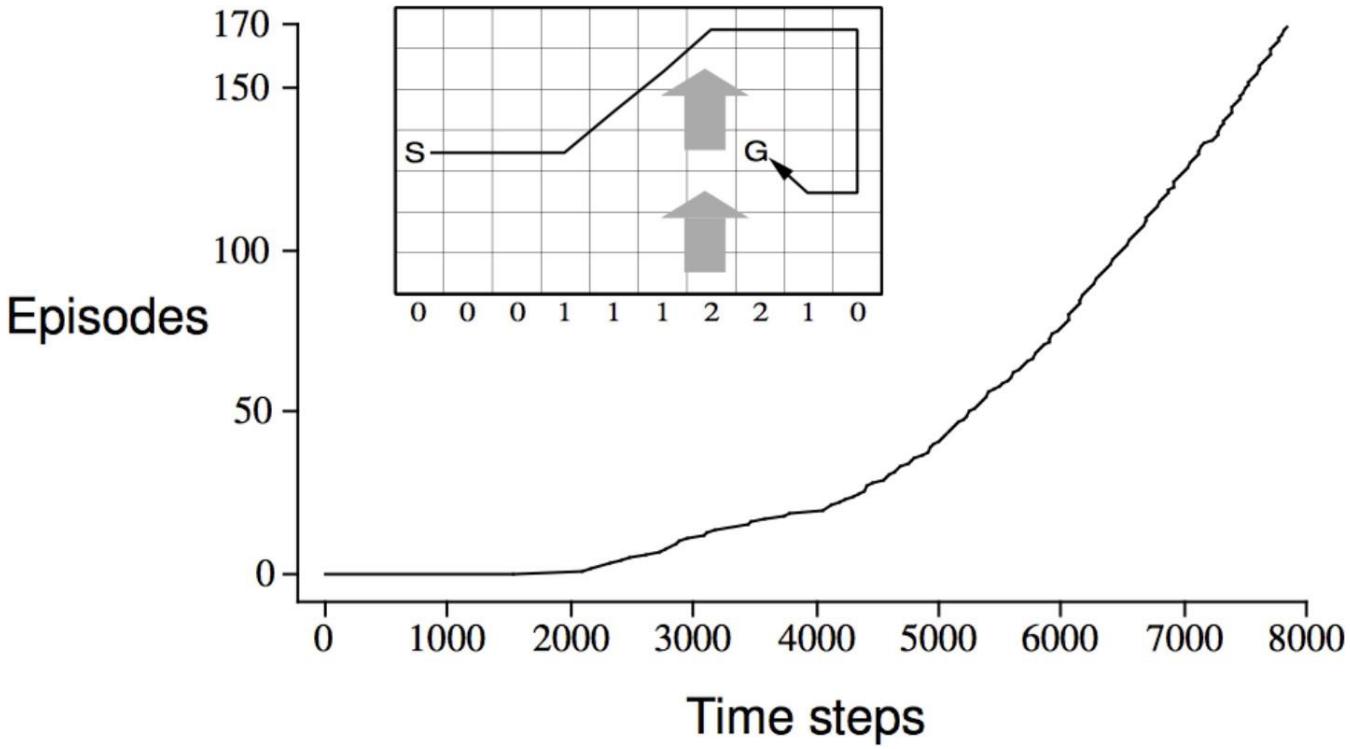


standard moves



king's moves

SARSA в ветреном клеточном мире



Обучение используя чужой опыт

Обучение по отложенному опыту

Строим целевую стратегию $\pi(a|s)$ для вычисления $V^{\pi}(s)$ или $Q^{\pi}(s, a)$
по следующей поведенческой стратегии $\mu(a|s)$:

$$(s_t, a_t, r_t, \ldots) \sim \mu$$

В чем разница?

- Обучение по наблюдениям за человеком или другими агентами Переиспользование опыта, полученного по старым стратегиям $\pi_1, \pi_2, \ldots, \pi_{t-1}$.
- Конструирование *оптимальной* стратегии, следуя *поисковой* стратегии
- Конструирование *нескольких* стратегий, следуя *одной*

Выборка по значимости

Оценка матожидания другого распределения (importance sampling):

$$\begin{aligned}\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\ &= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]\end{aligned}$$

Выборка по значимости для Монте-Карло по отложенному опыту

Используем отдачи, полученные по μ, для вычисления π

Взвешиваем отдачу R_t в соответствии со сходством стратегий

Считаем поправки для выборки значимости по всему эпизоду:

$$R_t^{\pi/\mu} = \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} \frac{\pi(a_{t+1}|s_{t+1})}{\mu(a_{t+1}|s_{t+1})} \cdots \frac{\pi(a_T|s_T)}{\mu(a_T|s_T)} R_t$$

Обновляем стратегию на основе скорректированной отдачи:

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t^{\pi/\mu} - V(s_t))$$

Не применяем, если μ – нулевая, а π – ненулевая

Выборка по значимости может существенно увеличить дисперсию

Выборка по значимости для ВР по отложенному опыту

Используем TD показатели, полученные по μ, для вычисления π

Взвешиваем TD показатель $R + \gamma V(s^J)$ в соответствии с выборкой по значимости

Необходима только одна правка по значимости

$$V(s_t) \leftarrow V(s_t) + \alpha \left(\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} (r_{t+1} + \gamma V(s_{t+1})) - V(s_t) \right)$$

Намного более низкая дисперсия по сравнению с Монте-Карло

Стратегии должны быть схожи только на одном шаге итерации

Алгоритм q-обучения

Q-обучение

Теперь рассмотрим обучение по отложенному опыту для функции действий $Q(s_t, a_t)$

Для Q-значения выборка по значимости не требуется! Действие a_t выбирается по поведенческой стратегии $\mu(\cdot|s_t)$

Но для оценки полезности в следующем состоянии мы возьмем альтернативное действие по целевой стратегии $a' \sim \pi(\cdot|s_t)$

Тогда обновление $Q(s_t, a_t)$ в соответствии с полезностью альтернативного действия будет выглядеть так:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a') - Q(s_t, a_t))$$

Управление по отложенному опыту с Q-обучением

Будем *улучшать* как целевую, так и поведенческую стратегии Целевая стратегия π улучшается жадно с учетом $Q(s, a)$:

$$\pi(s_{t+1}) = \arg\max_{a'} Q(s_{t+1}, a')$$

Поведенческая стратегия μ, например, ϵ-жадно, также с учетом $Q(s, a)$

Показатель Q-обучения упрощается:

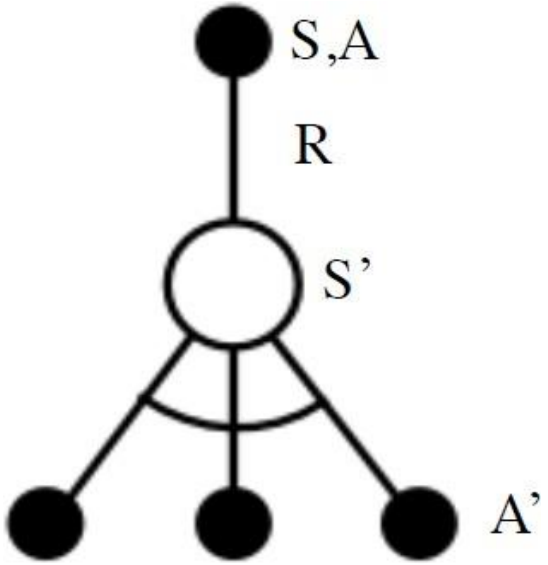
$$r_{t+1} + \gamma Q(s_{t+1}, a')$$

$$= r_{t+1} + \gamma Q(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'))$$

$$= rt+1 + \max_{a'} \gamma Q(st+1, a').$$

Алгоритм управления с Q-обучением (SARSAMAX)

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

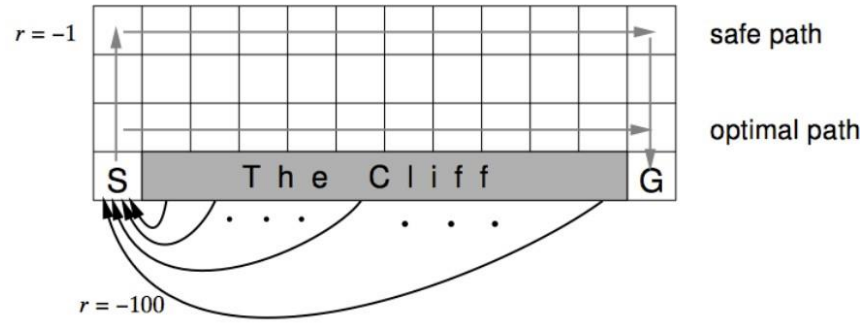


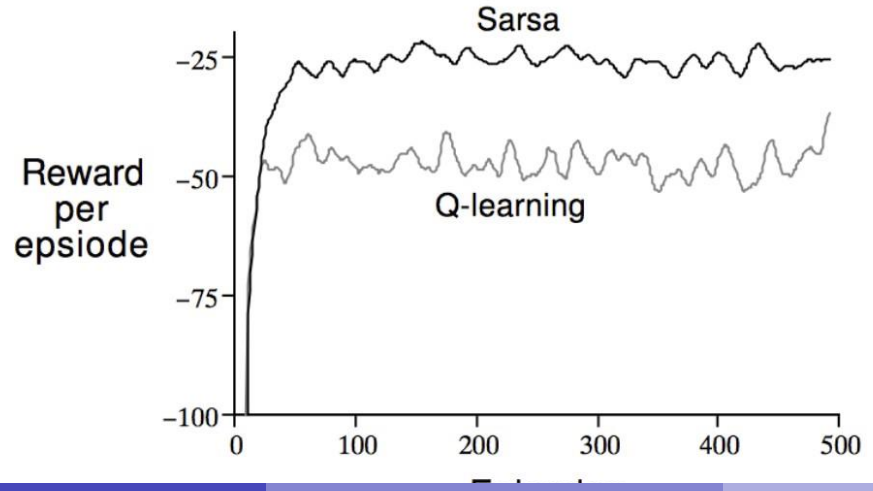
Алгоритм Q-обучения для управления по отложенному опыту

Algorithm 2 Q-обучение с ϵ-жадной стратегией

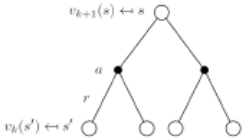
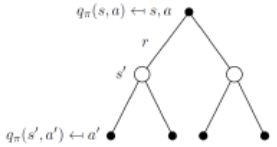
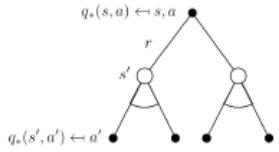
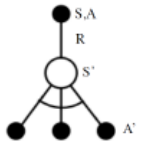
1: $MDP =< S, A, \gamma >$ ▷ задача МППР
2: $Q \leftarrow [0, \dots]$; ▷ таблица значений полезности действий
3: **for all** эпизодов **do**
4: s - начальное состояние;
5: **for all** шагов эпизода **do**
6: $a \leftarrow \epsilon$-жадная $\pi^Q(s)$
7: $r, s' \leftarrow$ применение a;
8: $a' \leftarrow \epsilon$-жадная $\pi(s')$;
9: $Q[s, a] \leftarrow Q(s, a) + \alpha(r + \gamma \max_a Q[s', a] - Q[s, a])$;
10: $s \leftarrow s', a \leftarrow a'$;

Пример: случайные блуждания над обрывом





Взаимосвязь между ДП и ВР

	Полный просмотр ДП	**Выборочный просмотр ВР**
Уравнение Беллмана для матожиданий $V^{\pi}(s,a)$	 Итерационная оценка стратегии	 TD-обучение
Уравнение Беллмана для матожидания $Q^{\pi}(s,a)$	 Итерация по Q-стратегиям	 SARSA
Уравнение оптимальности Беллмана $Q^{*}(s,a)$	 Итерация по Q-значениям	 Q-обучение

Взаимосвязь между ДП и ВР

Полный просмотр ДП	**Выборочный просмотр ВР**
Итерационная оценка стратегии	TD-обучение
$V(s) \leftarrow \mathbb{E}[r + \gamma V(s')\|s]$	$V(s) \xleftarrow{\alpha} r + \gamma V(s')$
Итерация по Q-стратегиям	SARSA
$Q(s,a) \leftarrow \mathbb{E}[r + \gamma Q(s',a')\|s,a]$	$Q(s,a) \xleftarrow{\alpha} r + \gamma Q(s',a')$
Итерация по Q-значениям	Q-обучение
$Q(s,a) \leftarrow \mathbb{E}\left[r + \gamma \max_{a' \in A} Q(s',a')\|s,a\right]$	$Q(s,a) \xleftarrow{\alpha} r + \gamma \max_{a' \in A} Q(s',a')$

$x \xleftarrow{\alpha} y$ означает $x \leftarrow x + \alpha(y - x)$.

Итог

В задаче безмодельного предсказания эффективности стратегии используются метод Монте-Карло и метод временных различий

В задаче безмодельного управления (поиска стратегии) применяется обобщенный метод итерации по стратегиям

Один из наиболее часто используемых методов – Q-обучение является методом обучения по отложенному опыту

Обучение с подкреплением: определения и примеры

План

- Введение
- Инкрементные методы
- Пакетные методы (Batches methods)
- Глубокое обучение с подкреплением (Deep RL)
- Алгоритм Rainbow

Введение

Пространства большой размерности в RL

Обучение с подкреплением может быть использовано для задач большой размерности, например:

нарды: 10^{20} состояний,

игра Го: 10^{170} состояний,

вертолет: непрерывное множество состояний

Как можно масштабировать безмодельные методы *предсказания* и *управления*, которые мы рассматривали ранее?

Аппроксимация функции полезности

Мы рассматривали функцию полезности в виде таблицы (lookup table):

- для каждого s состояния была запись $V(s)$,
- или для каждой пары состояние-действие s, a была запись $Q(S, a)$

Проблема с большими MDP:

- слишком много состояний и/или действий, чтобы хранить их в памяти,
- слишком медленное обучение функции полезности для каждого состояния в отдельности

Решение для больших MDP:

- Оценка функции полезности с помощью *аппроксимации функции*:

$$\hat{V}(s, \mathrm{w}) \approx V^{\pi}(s),$$

$$\hat{Q}(s, a, \mathrm{w}) \approx Q^{\pi}(s, a).$$

- Обобщение наблюдаемых состояний на ненаблюдаемые
- Обновление параметров w с использованием MC или TD обучения

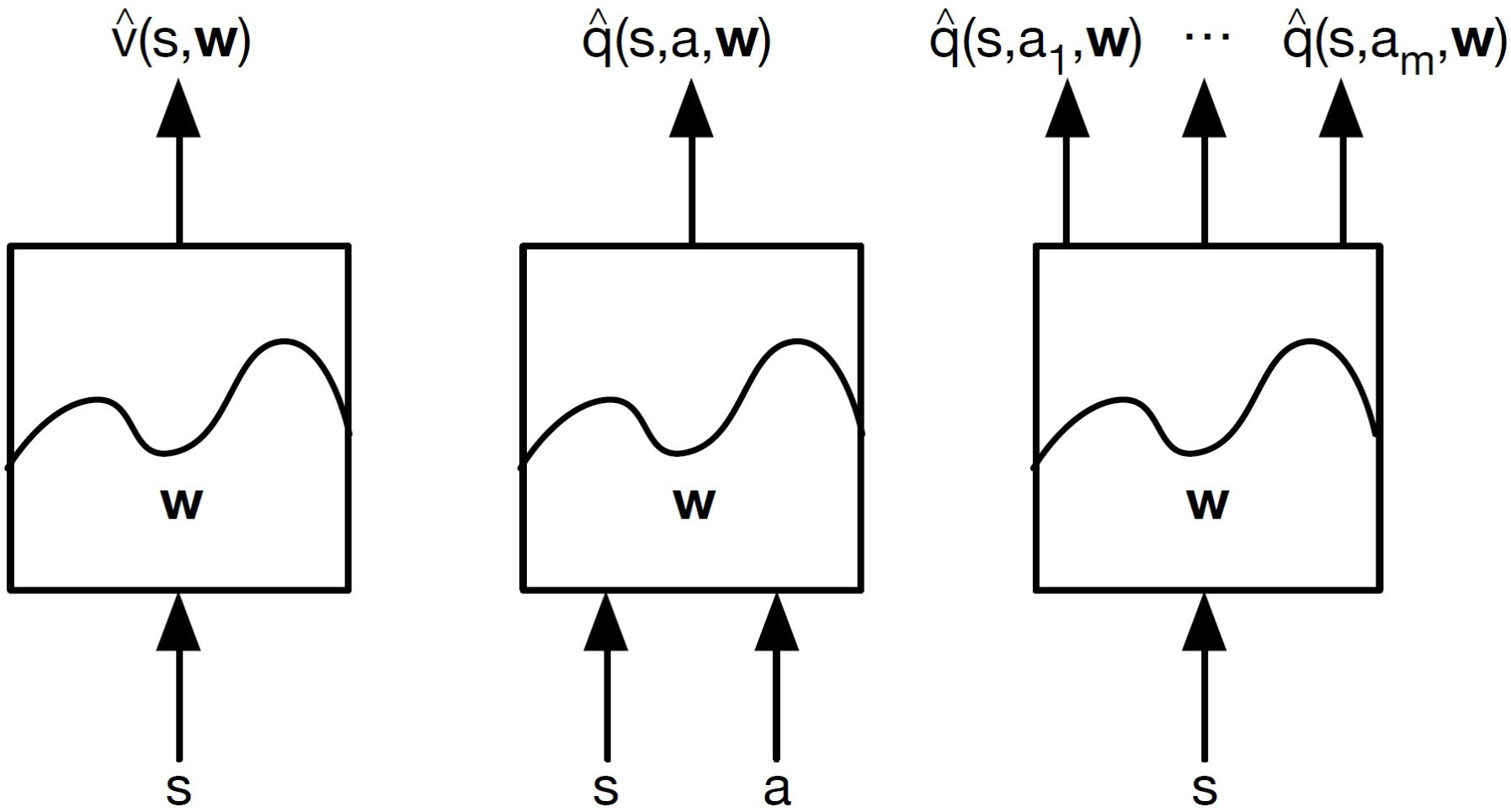
Преимущества обобщения

Сокращение используемой памяти для $(T, R)/V/Q/\pi$

Сокращение вычислительных затрат при работе с $(T, R)/V/Q/\pi$

Сокращение количества необходимых данных (опыта) для поиска оптимальных $(T, R)/V/Q/\pi$

Виды аппроксимации функции полезности



Аппроксиматоры

Линейная комбинация признаков

Нейронные сети

Деревья принятия решений

Ближайшие соседи

Фурье/вейвлет разложения

. . .

Аппроксиматоры

Будем рассматривать **дифференцируемые** аппроксиматоры:

Линейная комбинация признаков Нейронные сети

Деревья принятия решений Ближайшие соседи Фурье/вейвлет разложения

. . .

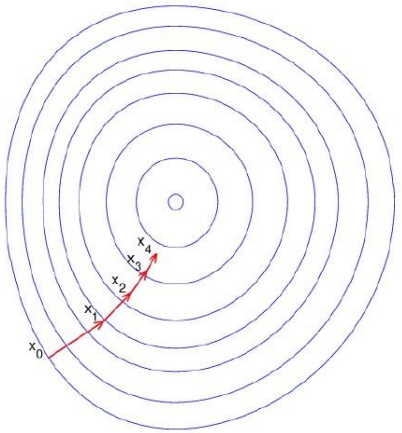
Нам нужны методы обучения, подходящие для анализа
нестационарных и не размеченных данных

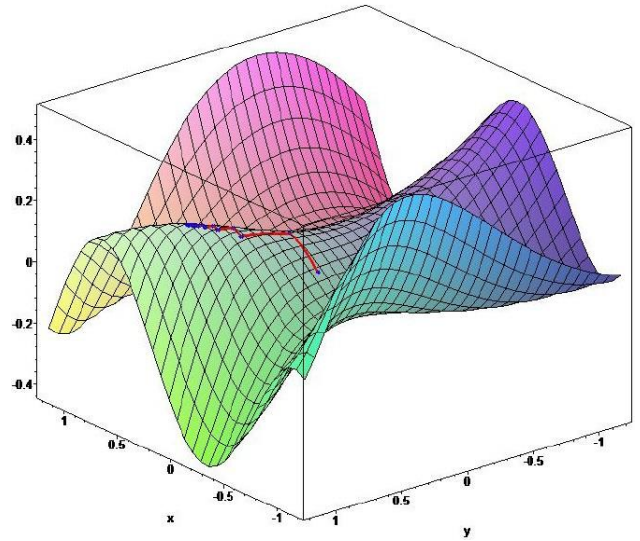
Градиентный спуск

Пусть $J(\mathrm{w})$ – дифференцируемая функция вектора параметров w

Запишем градиент функции $J(\mathrm{w})$:

$$\nabla_{\mathrm{w}} J(\mathrm{w}) = \begin{pmatrix} \frac{\partial J(\mathrm{w})}{\partial \mathrm{w}_1} \\ \vdots \\ \frac{\partial J(\mathrm{w})}{\partial \mathrm{w}_n} \end{pmatrix}$$





с

Будем искать локальный минимум функции $J(\mathrm{w})$

Будем обновлять w в направлении антиградиента

(α – параметр длины шага) (gradient descent):

$$\Delta \mathrm{w} = -\frac{1}{2}\alpha \nabla_{\mathrm{w}} J(\mathrm{w})$$

Стохастический градиентный спуск

Цель: найти вектор параметров w, минимизируя среднеквадратичную ошибку в предположении, что мы знаем истинные значения $V^{\pi}(s)$:

$$J(\mathrm{w}) = \mathbb{E}_{\pi}[(V^{\pi}(s) - \hat{V}(s, \mathrm{w}))^2]$$

Градиентный спуск позволяет найти локальный минимум:

$$\Delta \mathrm{w} = -\frac{1}{2}\alpha \nabla_{\mathrm{w}} J(\mathrm{w}) = \alpha \mathbb{E}_{\pi}[(V^{\pi}(s) - \hat{V}(s, \mathrm{w}))\nabla_{\mathrm{w}} \hat{V}(s, \mathrm{w})]$$

Стохастический градиентный спуск (stochastic gradient descent) производит спуск по подвыборке:

$$\Delta \mathrm{w} = \alpha(V^{\pi}(s) - \hat{V}(s, \mathrm{w}))\nabla_{\mathrm{w}} \hat{V}(s, \mathrm{w})$$

Матожидание обновления равно полному градиентному обновлению

Вектор признаков

Представим состояние в виде вектора признаков:

$$x(s) = \begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

Примеры:

- Расстояние от робота до ключевых точек местности Тренды на бирже
- Сильные и слабые конфигурации в шахматах

Линейная аппроксимация функции полезности

Представим функцию полезности в виде линейной комбинации признаков:

$$\hat{V}(s, \mathrm{w}) = \mathrm{x}(s)^\top \mathrm{w} = \sum_{j=1}^{n} \mathrm{x}_j(s)\mathrm{w}_j$$

Целевая функция квадратична по w:

$$\Delta \mathrm{w} = \alpha(V^\pi(s) - \mathrm{x}(s)^\top \mathrm{w})\mathrm{x}(s)$$

Стохастический градиентный спуск сходится к глобальному оптимуму.
Правило обновления:

$$\nabla_{\mathrm{w}} \hat{V}(s, \mathrm{w}) = \mathrm{x}(s)$$

$$J(\mathrm{w}) = \mathbb{E}_\pi[(V^\pi(s) - \mathrm{x}(s)^\top \mathrm{w})^2]$$

Обновление = длина шага × ошибка предсказания × значение признака.

Табличные признаки

Табличное представление функции полезности – частный случай линейного аппроксиматора:

$$x^{table}(s) = \begin{pmatrix} 1(s = s_1) \\ \vdots \\ 1(s = s_n) \end{pmatrix}$$

Вектор параметров представляет собой значения полезности каждого состояния:

$$\hat{V}(s, w) = \begin{pmatrix} 1(s = s_1) \\ \vdots \\ 1(s = s_n) \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}$$

Инкрементные алгоритмы

Инкрементные алгоритмы предсказания

Мы предполагали, что истинные значения функции полезности нам доступны

Однако в нашей задаче нет учителя, только вознаграждения На практике мы заменяем учителя оценкой функции $V^{\pi}(s)$:

- для MC оценка – это отдача R_t :

$$\Delta w = \alpha(R_t - \hat{V}(s_t, w))\nabla_w \hat{V}(s_t, w),$$

- для TD(0) оценка – это показатель $r_{t+1} + \gamma \hat{V}(s_{t+1}, w)$:

$$\Delta w = \alpha(r_{t+1} + \gamma \hat{V}(s_{t+1}, w) - \hat{V}(s_t, w))\nabla_w \hat{V}(s_t, w)$$

Монте-Карло с аппроксимацией функции полезности

Отдача R_t – несмещенная, зашумленная подвыборка истинного значения $V^{\pi}(s_t)$

Можем применить обучение с учителем для “обучающей выборки”:

$$\langle s_1, R_1 \rangle, \langle s_2, R_2 \rangle, \ldots, \langle s_T, R_T \rangle$$

Пример: использование линейной Монте-Карло оценки стратегии

$$\Delta \mathrm{w} = \alpha(R_t - \hat{V}(s_t, \mathrm{w}))\nabla_{\mathrm{w}} \hat{V}(s_t, \mathrm{w}) = \alpha(R_t - \hat{V}(s_t, \mathrm{w}))\mathrm{x}(s_t)$$

Монте-Карло оценка сходится к локальному оптимуму

Это верно и при использовании нелинейных аппроксиматоров

TD-обучение с аппроксимацией функции полезности

TD показатель $r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathrm{w})$ – смещенная подвыборка истинного значения $V^{\pi}(s_t)$

Можем применить обучение с учителем для “обучающей выборки”:

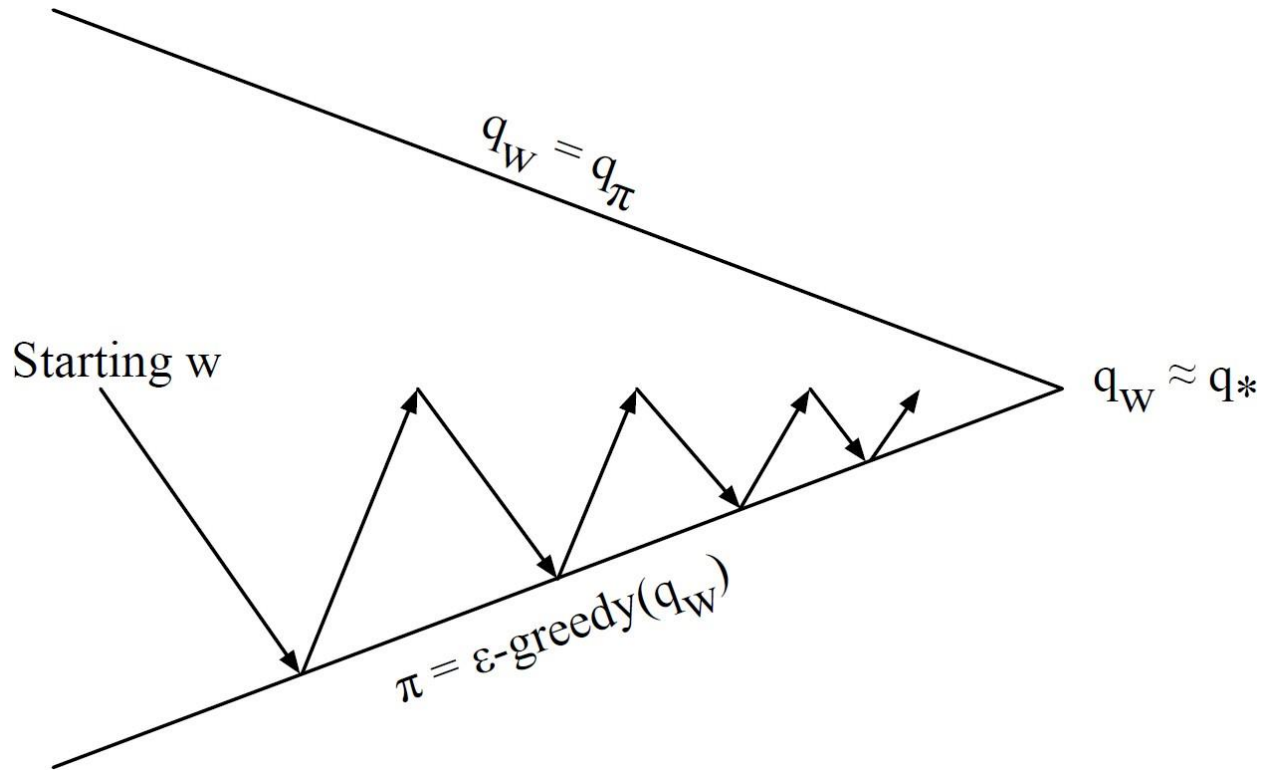
$$\langle s_1, r_2 + \gamma \hat{V}(s_2, \mathrm{w}) \rangle, \langle s_2, r_3 + \gamma \hat{V}(s_3, \mathrm{w}) \rangle, \ldots, \langle s_{T-1}, r_T \rangle$$

Пример: использование линейного TD(0)

$$\Delta \mathrm{w} = \alpha(r + \gamma \hat{V}(s', \mathrm{w}) - \hat{V}(s_t, \mathrm{w})) \nabla_{\mathrm{w}} \hat{V}(s_t, \mathrm{w}) = \alpha \delta \mathrm{x}(s)$$

Линейное TD(0)-обучение сходится к глобальному оптимуму

Управление с аппроксимацией функции полезности



Оценка стратегии: приближенная оценка стратегии $\hat{Q}(\cdot,\cdot,w) \approx Q^\pi$.
Улучшение стратегии: ϵ-жадное улучшение стратегии.

Аппроксимация функции полезности действия

Аппроксимация функции полезности действия:

$$\hat{Q}(s, a, \mathrm{w}) \approx Q^{\pi}(s, a).$$

Минимизация среднеквадратичной ошибки:

$$J(\mathrm{w}) = \mathrm{E}_{\pi}[(Q^{\pi}(s, a) - \hat{Q}(s, a, \mathrm{w}))^2].$$

Использование стохастического градиентного спуска для поиска локального минимума:

$$-\frac{1}{2}\nabla_{\mathrm{w}} J(\mathrm{w}) = (Q^{\pi}(s, a) - \hat{Q}(s, a, \mathrm{w}))\nabla_{\mathrm{w}} \hat{Q}^{\pi}(s, a, \mathrm{w}).$$

$$\Delta \mathrm{w} = \alpha(Q^{\pi}(s, a) - \hat{Q}(s, a, \mathrm{w}))\nabla_{\mathrm{w}} \hat{Q}^{\pi}(s, a, \mathrm{w}).$$

Линейная аппроксимация функции полезности действия

Будем представлять и состояние и действие в виде вектора признаков:

$$\mathrm{x}(s,a) = \begin{pmatrix} \mathrm{x}_1(s,a) \\ \vdots \\ \mathrm{x}_n(s,a) \end{pmatrix}$$

Представим функцию полезности действия в виде линейной комбинации признаков:

$$\hat{Q}(s,a,\mathrm{w}) = \mathrm{x}(s,a)^\top \mathrm{w} = \sum_{j=1}^{n} \mathrm{x}_j(s,a)\mathrm{w}_j$$

Обновление стохастического градиентного спуска:

$$\nabla_{\mathrm{w}} \hat{Q}(s,a,\mathrm{w}) = \mathrm{x}(s,a),$$

$$\Delta \mathrm{w} = \alpha(Q^\pi(s,a) - \hat{Q}(s,a,\mathrm{w}))\mathrm{x}(s,a)$$

Инкрементальный алгоритмы управления

Как и в предсказании, мы должны чем-то заменить оценку для $Q^{\pi}(s, a)$:

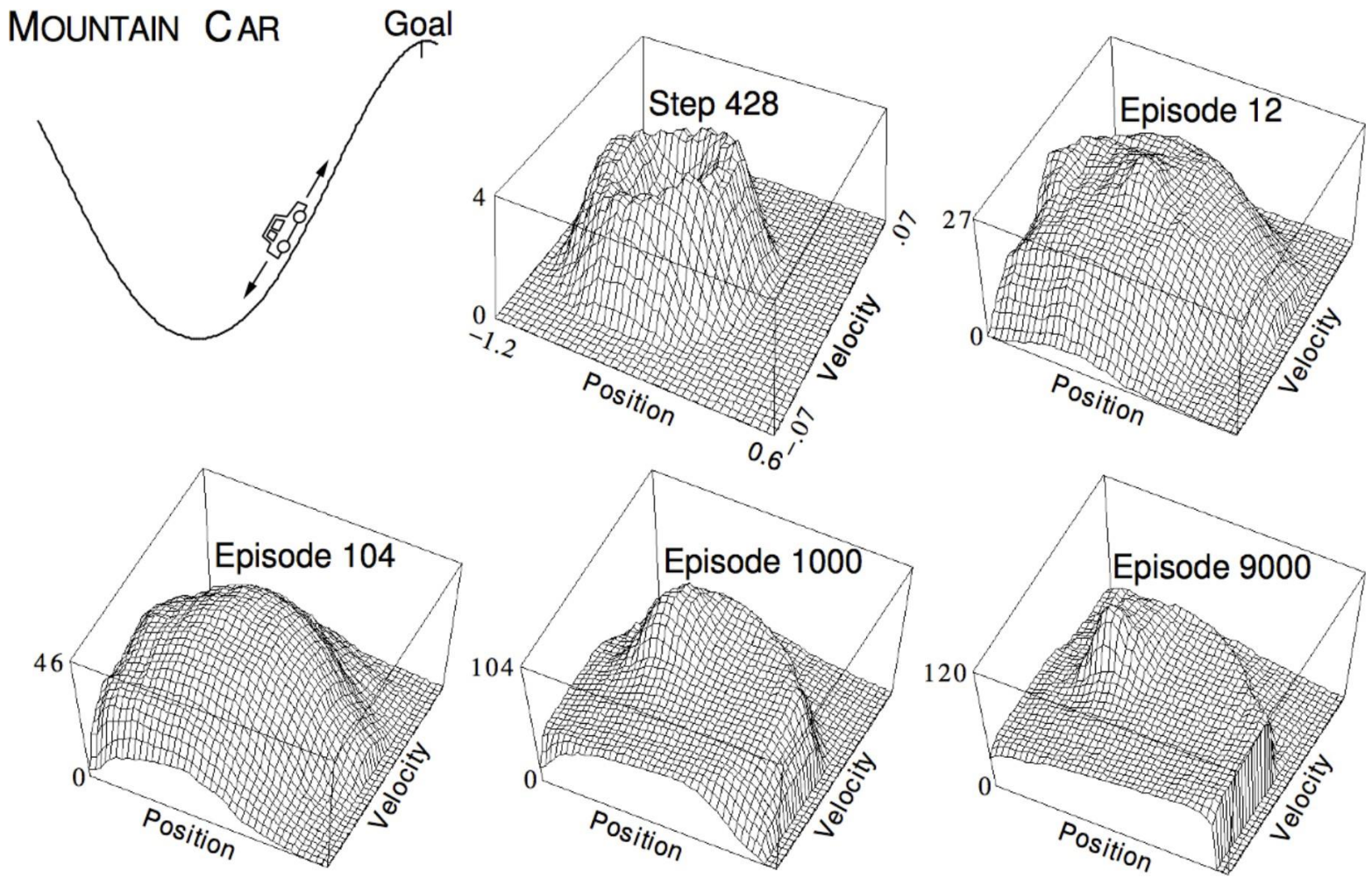
- для MC оценка – это отдача R_t :

$$\Delta \mathrm{w} = \alpha(R_t - \hat{Q}(s_t, a_t, \mathrm{w}))\nabla_{\mathrm{w}}\hat{Q}(s_t, a_t, \mathrm{w})$$

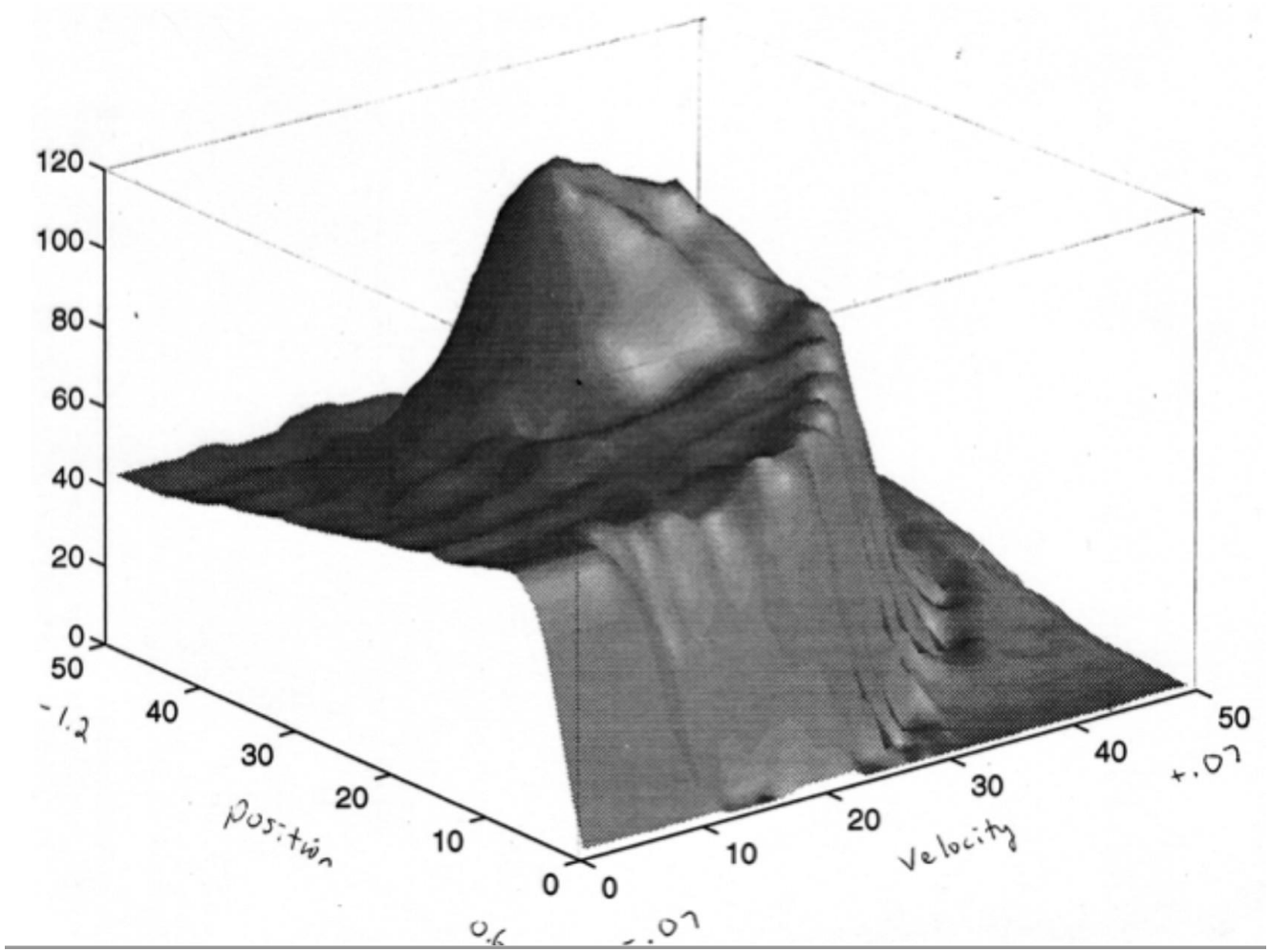
- для TD(0) оценка – это показатель $r_{t+1} + \gamma\hat{Q}(s_{t+1}, a_{t+1}, \mathrm{w})$:

$$\Delta \mathrm{w} = \alpha(r_{t+1} + \gamma\hat{Q}(s_{t+1}, a_{t+1}, \mathrm{w}) - \hat{Q}(s_t, a_t, \mathrm{w}))\nabla_{\mathrm{w}}\hat{Q}(s_t, a_t, \mathrm{w})$$

Линейная SARSA для задачи “Внедорожник”



Линейная SARSA с радиальными базисными функциями для задачи “Внедорожник”



Сходимость алгоритмов управления

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
Gradient Q-learning	✓	✓	✗

(C) – остается в некоторой окрестности оптимального значения функции полезности.

Пакетные методы (Batches methods)

Пакетное обучение с подкреплением

Градиентный спуск прост и эффективен в теории

Однако он не столь эффективен на подвыборках

Пакетные методы ставят своей целью найти наиболее подходящую функцию полезности

В качестве тренировочных данных используется опыт агента

Предсказание методом наименьших квадратов

Пусть имеется аппроксимация функции полезности
$\hat{V}(s, w) \approx V^{\pi}(s)$

Опыт агент представляет собой множество пар состояние-действие:

$$\mathcal{D} = \{\langle s_1, V_1^{\pi}\rangle, \langle s_2, V_2^{\pi}\rangle, \ldots, \langle s_T, V_T^{\pi}\rangle\}$$

Вопрос: какое значение вектора параметров w даст наиболее подходящее значений функции $\hat{V}(s, w)$?

Алгоритмы **метода наименьших квадратов** (LSM) позволяют найти значение вектора параметров w, минимизирующее сумму квадратов ошибок между $\hat{V}(s, w)$ и целевым значением V_t^{π}:

$$LS(w) = \sum_{t=1}^{T}(V_t^{\pi} - \hat{V}(s_t, w))^2 = \mathbb{E}_{\mathcal{D}}[(V^{\pi} - \hat{V}(s, w))^2]$$

Стохастический градиентный спуск с повторением опыта

Пусть есть опыт агента в виде множества пар состояние-действие:

$$\mathcal{D} = \{\langle s_1, V_1^\pi \rangle, \langle s_2, V_2^\pi \rangle, \ldots, \langle s_T, V_T^\pi \rangle\}$$

Будем повторять:

Делаемы выборку состояния и действия из опыта:

$$\langle s, V^\pi \rangle \sim \mathcal{D}$$

Применяем обновление стохастического градиентного спуска:

$$\Delta \mathrm{w} = \alpha(V^\pi - \hat{V}(s, \mathrm{w}))\nabla_{\mathrm{w}} \hat{V}(s, \mathrm{w})$$

Этот процесс сходится к решению LSM:

$$\mathrm{w}^\pi = \arg\min_{\mathrm{w}} LS(\mathrm{w})$$

Линейное предсказание LSM

Повторение опыта позволяет найти решение LSM

Может понадобиться много итераций

Применим *линейную* аппроксимацию функции $\hat{V}(s, w) = x(s)^{T}w$

Можем найти решение LSM аналитически

Линейное предсказание LSM

В точке минимума матожидание обновления должно быть равно 0:

$$\mathbb{E}_{\mathcal{D}}[\Delta \mathrm{w}] = 0$$

$$\alpha \sum_{t=1}^{T} \mathrm{x}(s_t)(V_t^{\pi} - \mathrm{x}(s_t)^{\top}\mathrm{w}) = 0$$

$$\sum_{t=1}^{T} \mathrm{x}(s_t) V_t^{\pi} = \sum_{t=1}^{T} \mathrm{x}(s_t)\mathrm{x}(s_t)^{\top}\mathrm{w}$$

$$\mathrm{w} = \left(\sum_{t=1}^{T} \mathrm{x}(s_t)\mathrm{x}(s_t)^{\top}\right)^{-1} \sum_{t=1}^{T} \mathrm{x}(s_t) V_t^{\pi}$$

Для N признаков сложность прямого решения равна $O(N^3)$

Можно найти решение итерационным методом Шермана-Моррисона за $O(N^2)$

Алгоритмы линейного предсказание LSM

Нам не известны истинные значения V_t^π

На практике “тренировочные данные” должны давать зашумленную или смещенную выборку V_t^π :

LSMC Монте-Карло метод наименьших квадратов использует отдачу $V_t^\pi \approx R_t$

LSTD Метод наименьших квадратов с временной разностью использует TD-показатель

В каждом случае возможно прямое решение для фиксированной точки MC/TD

$$V_t^\pi \approx r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathrm{w})$$

Алгоритмы линейного предсказание LSM

LSMC

$$0 = \sum_{t=1}^{T} \alpha(R_t - \hat{V}(s_t, \mathrm{w}))\mathrm{x}(s_t)$$

$$\mathrm{w} = \left(\sum_{t=1}^{T} \mathrm{x}(s_t)\mathrm{x}(s_t)^T \right)^{-1} \sum_{t=1}^{T} \mathrm{x}(s_t) R_t$$

LSTD

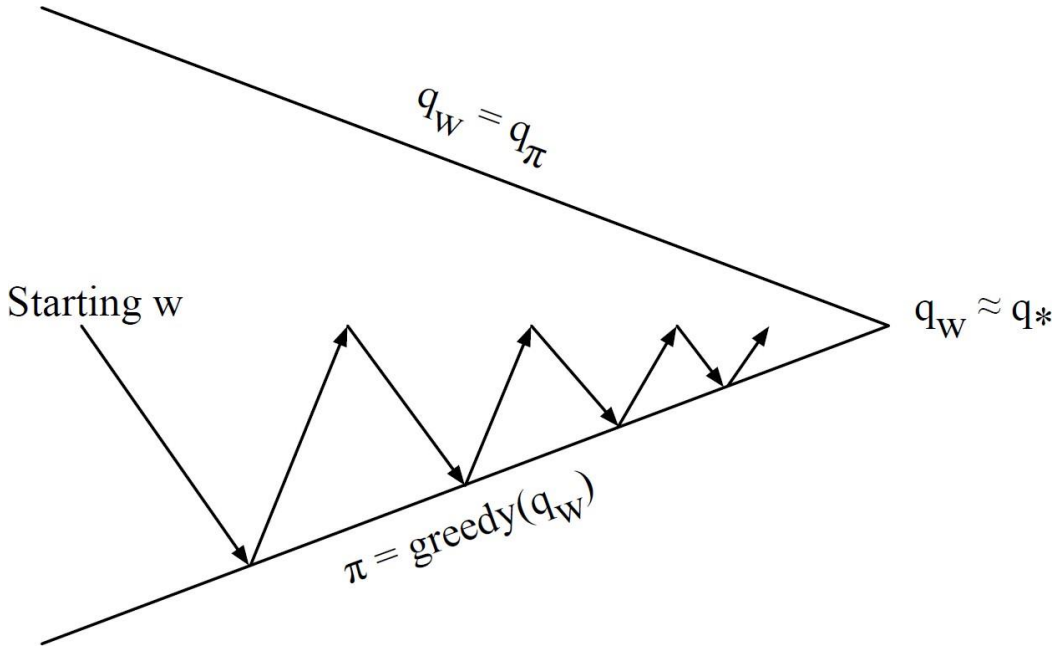
$$0 = \sum_{t=1}^{T} \alpha(r_{t+1} + \gamma \hat{V}(s_{t+1}, \mathrm{w}) - \hat{V}(s_t, \mathrm{w}))\mathrm{x}(s_t)$$

$$\mathrm{w} = \left(\sum_{t=1}^{T} \mathrm{x}(s_t)(\mathrm{x}(s_t) - \gamma \mathrm{x}(s_{t+1}))^T \right)^{-1} \sum_{t=1}^{T} \mathrm{x}(s_t) r_{t+1}$$

Сходимость алгоритмов линейного предсказание LS

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✓	✗
	LSTD	✓	✓	-
Off-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✗	✗
	LSTD	✓	✓	-

LSM итерация по стратегиям



Оценка стратегии: с помощью Q-обучения методом наименьших квадратов
Улучшение стратегии: жадное улучшение стратегии

Аппроксимация функции полезности действия с помощью LSM

Будем аппроксимировать функцию полезности действия $Q^{\pi}(s, a)$ линейной комбинацией признаков $\mathrm{x}(s, a)$:

$$\hat{Q}(s, a, \mathrm{w}) = \mathrm{x}(s, a)^{T}\mathrm{w} \approx Q^{\pi}(s, a)$$

Минимизируем квадрат ошибки между $\hat{Q}(s, a, \mathrm{w})$ и $Q^{\pi}(s, a)$ по опыту, полученному с помощью π и состоящего из следующих пар:

$$\mathcal{D} = \{\langle (s_1, a_1), V_1^{\pi}\rangle, \langle (s_2, a_2), V_2^{\pi}\rangle, \ldots, \langle (s_T, a_T), V_T^{\pi}\rangle\}$$

Управление LSM

Для оценки стратегии мы используем весь опыт

В случае управления нам необходимо улучшать стратегию В этом случае опыт генерируется многими стратегиями

Поэтому для вычисления $Q^{\pi}(s, a)$ мы должны использовать алгоритмы с неявной стратегией (off-policy)

Будем использовать ту же идею, что и в Q-обучении:

- используем опыт, полученный при помощи старой стратегии:

$$s_t, a_t, r_{t+1}, s_{t+1} \sim \pi_{old},$$

- рассматриваем альтернативное следующее действие $a' = \pi_{new}(s_{t+1})$,
- обновляем $\hat{Q}(s, a, w)$ по полезности альтернативного действия $r_{t+1} + \gamma \hat{Q}(s_{t+1}, a, w)$

Q-обучение LSM

Рассмотрим следующее обновление линейным Q-обучением:

$$\delta = r_{t+1} + \gamma \hat{Q}(s_{t+1}, \pi(s_{t+1}), \mathrm{w}) - \hat{Q}(s_t, a_t, \mathrm{w})$$

$$\Delta \mathrm{w} = \alpha \delta \mathrm{x}(s_t, a_t)$$

LSTDQ алгоритм – решением ищем по нулевому обновлению:

$$0 = \sum_{t=1}^{T} \alpha(r_{t+1} + \gamma \hat{Q}(s_{t+1}, \pi(s_{t+1}), \mathrm{w}) - \hat{Q}(s_t, a_t, \mathrm{w}))\mathrm{x}(s_t, a_t),$$

$$\mathrm{w} = \left(\sum_{t=1}^{T} \mathrm{x}(s_t, a_t)(\mathrm{x}(s_t, a_t) - \gamma \mathrm{x}(s_{t+1}, \pi(s_{t+1}))) \right)^{-1} \sum_{t=1}^{T} \mathrm{x}(s_t, a_t) r_{t+1}$$

Алгоритм итераций по стратегиям с LSM

Ниже дан псевдокод, который использует вызов функции LSTDQ
Здесь постоянно пересчитывается опыт $\mathbb{D}$ с различными стратегиями

Algorithm 1 LSPI

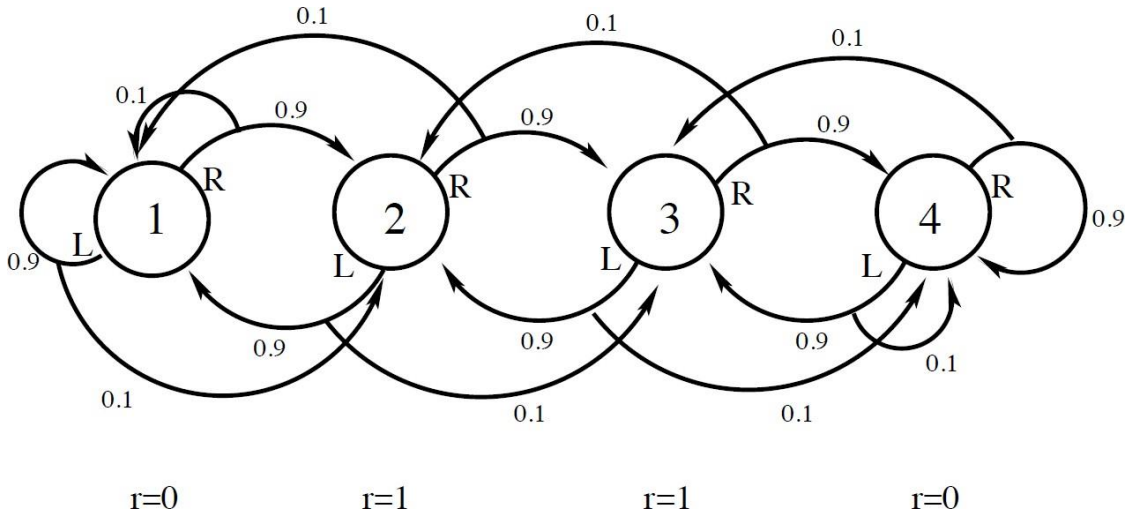
function $\text{LSPI-TD}(\mathcal{D}, \pi_0)$
  $\pi' \leftarrow \pi_0$
  repeat
    $\pi \leftarrow \pi'$
    $Q \leftarrow LSTDQ(\pi, \mathcal{D})$
    for all $s \in S$ **do**
      $\pi'(s) \leftarrow \arg\max_{a \in A} Q(s, a)$
  until $\pi \approx \pi'$
  return π

Сходимость алгоритмов управления

Algorithm	Table Lookup	Linear	Non-Linear
Monte-Carlo Control	✓	(✓)	✗
Sarsa	✓	(✓)	✗
Q-learning	✓	✗	✗
LSPI	✓	(✓)	-

(C) – остается в некоторой окрестности оптимального значения функции полезности.

Пример: блуждания по цепи



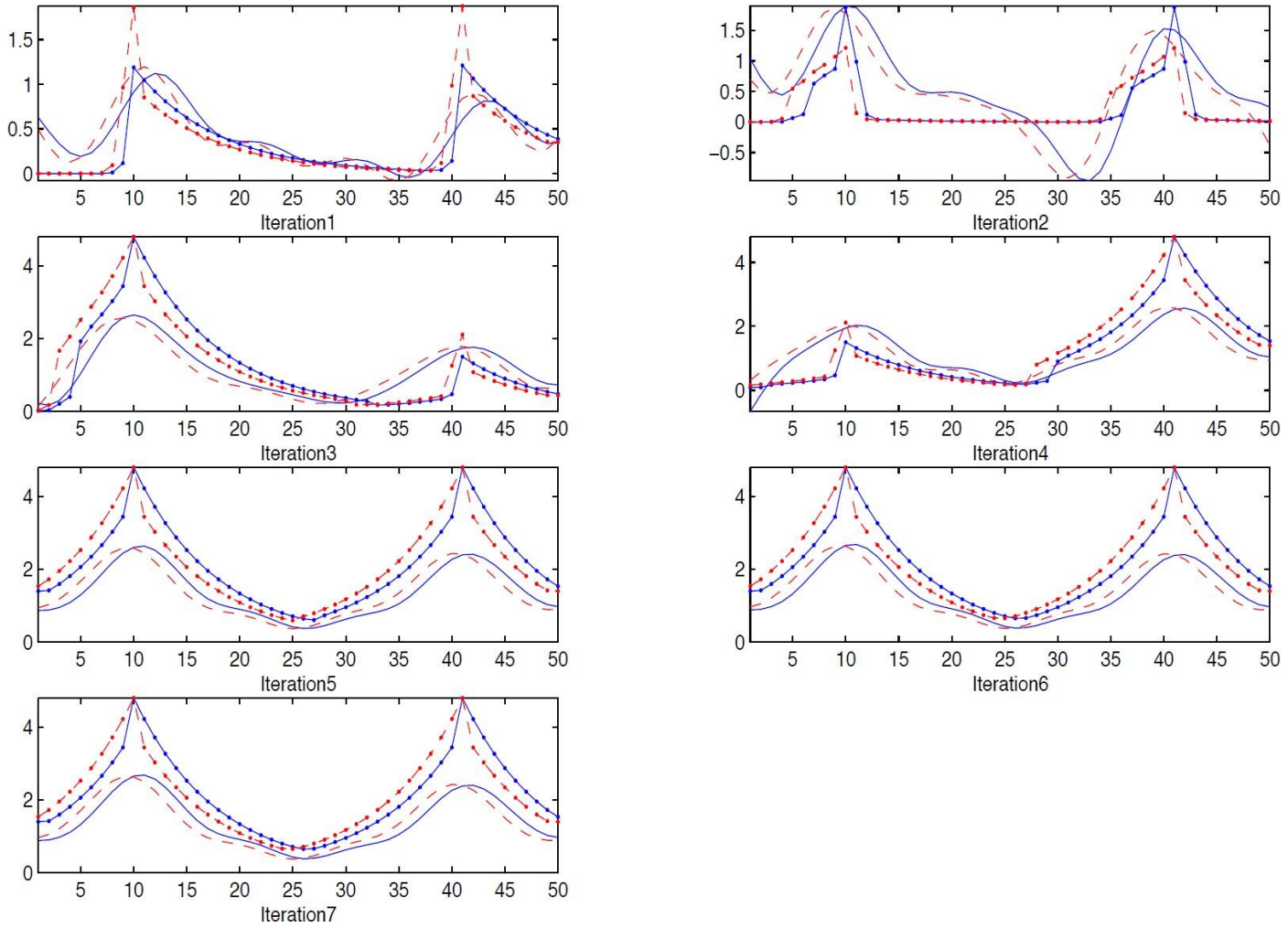
Рассмотрим версию задачи с 50 состояниями Вознаграждение +1 в состояниях 10 и 41, 0 в остальных

Оптимальная стратегия: R (1-9), L(20-25), R (26-41), L (42, 50)

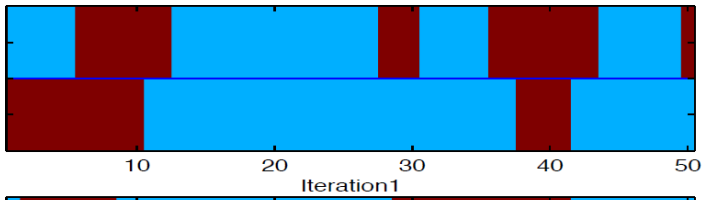
Признаки: 10 равномерно расставленные гауссианы ($\sigma = 4$) для каждого действия

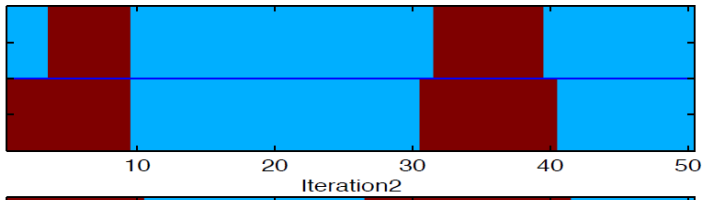
Опыт: 10 000 шагов случайных блужданий

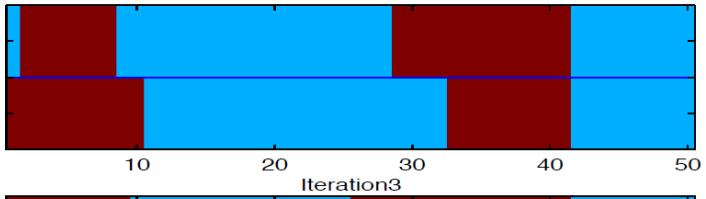
LSPI для блужданий по цепи: функция полезности действий

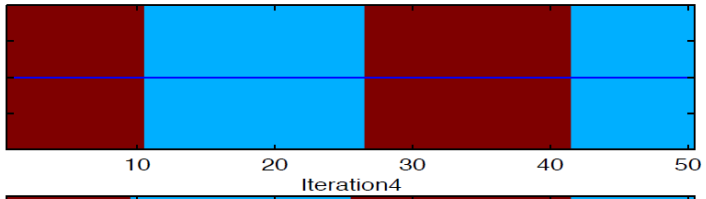


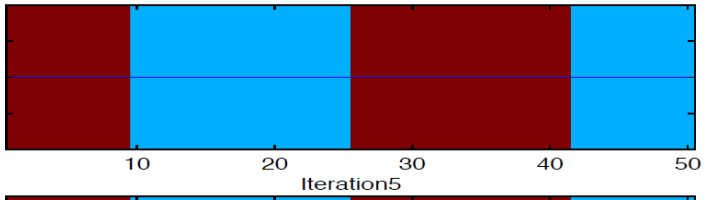
LSPI для блужданий по цепи: стратегия

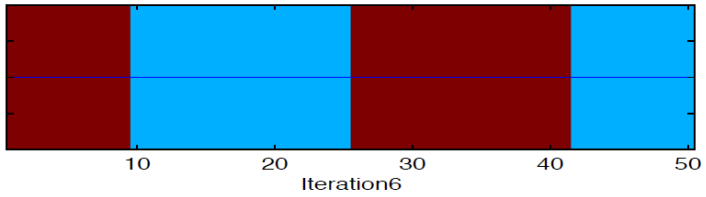


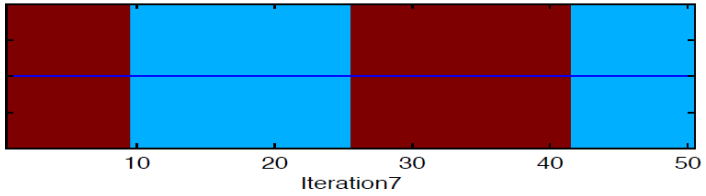












Аппроксимация глубокими нейронными сетями

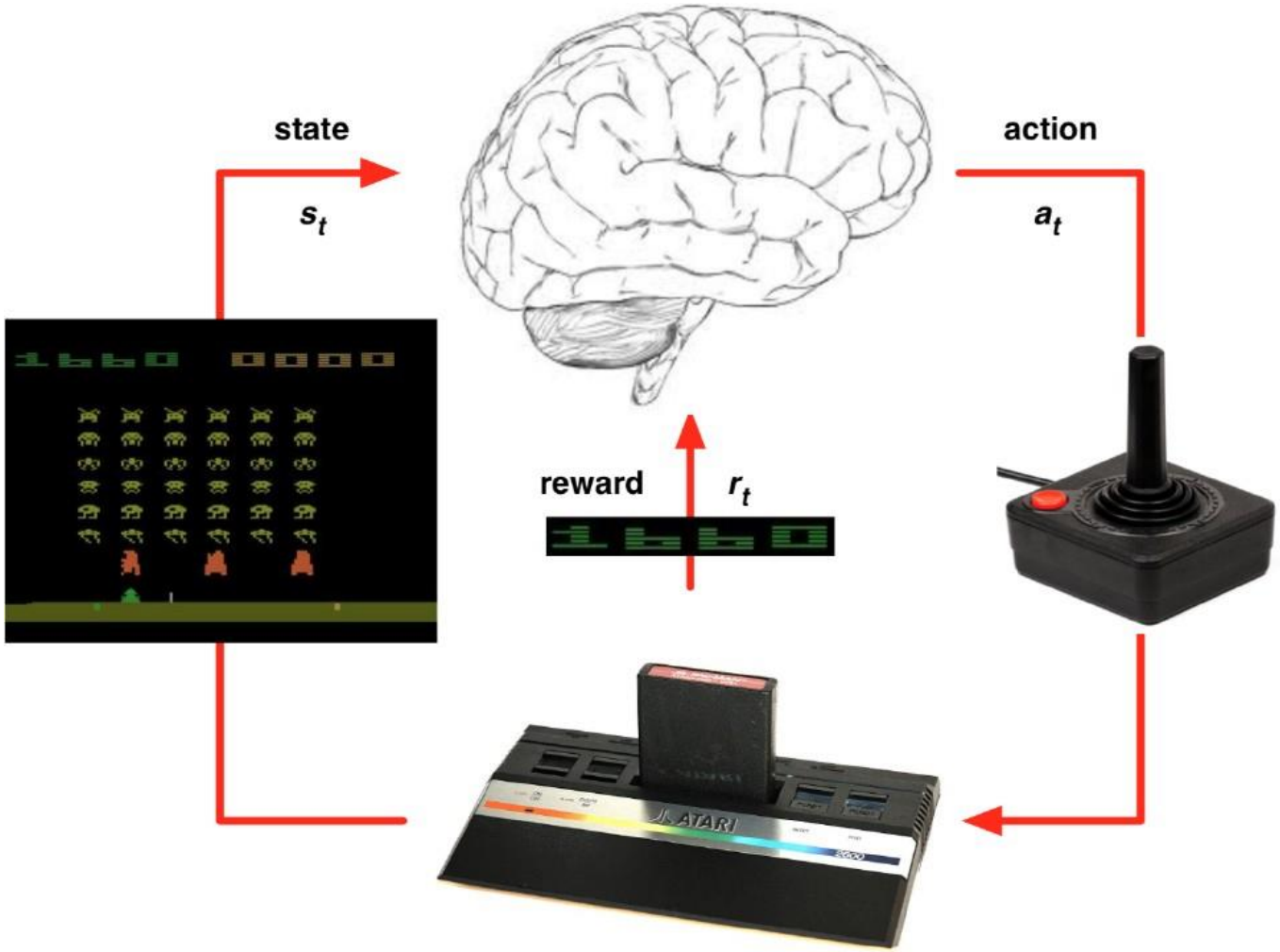
Композиция множества функций $y = h_n(h_{n-1} \ldots h_1(\mathrm{x}))$.

Комбинация линейного и нелинейного преобразования: $h_n = wh_{n-1}$ и $h_n = f(h_{n-1})$.

Используем цепное правило обратного распространения для вычисления градиента.

Для обучения необходимо задание функции потерь L. Используются распределенные представления вместо локальных.

Нейросетевая аппроксимация в Atari



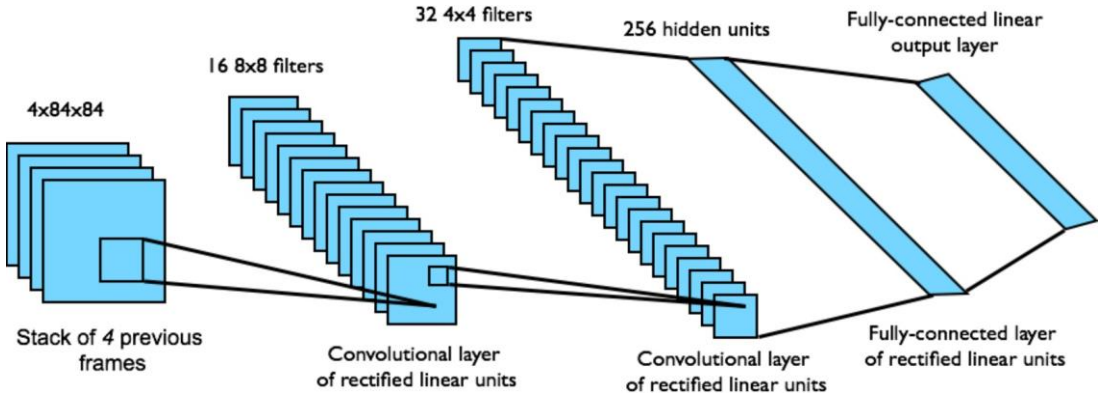
DQN в играх Atari

Сквозное (end-to-end) обучение значений $Q(s, a)$ по пикселям s

Входное состояние – 4 склеенных изображения 4 последних фреймов

Выход – значения $Q(s, a)$ для 18 позиций джойстика/кнопки

Вознаграждение – изменение в счете за этот шаг



Проблемы сходимости Q-обучения с аппроксимацией:

- Высокая корреляция между примерами в выборке
- Нестационарные целевых значений

Сохранение опыта в глубоких Q-сетях (DQN)

DQN использует **память прецедентов** (replay buffer) и **фиксированные** Q-показатели

- Выбираем действие a_t в соответствии с ϵ-жадной стратегией
- Сохраняем наблюдаемый переход (s_t, a_t, r_{t+1}, s_{t+1}) в память переигровок $\mathbb{D}$
- Выбираем случайный мини-пакет (mini-batch) переходов (s, a, r, s^J) из памяти $\mathbb{D}$
- Используем один из вариантов стохастического градиентного спуска:

$$\Delta w = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a', w) - \hat{Q}(s, a, w))\nabla_w \hat{Q}(s, a, w)$$

Фиксирование Q-показателя

Зафиксируем значение весов используемых для вычисления Q-показателя на несколько шагов обновления

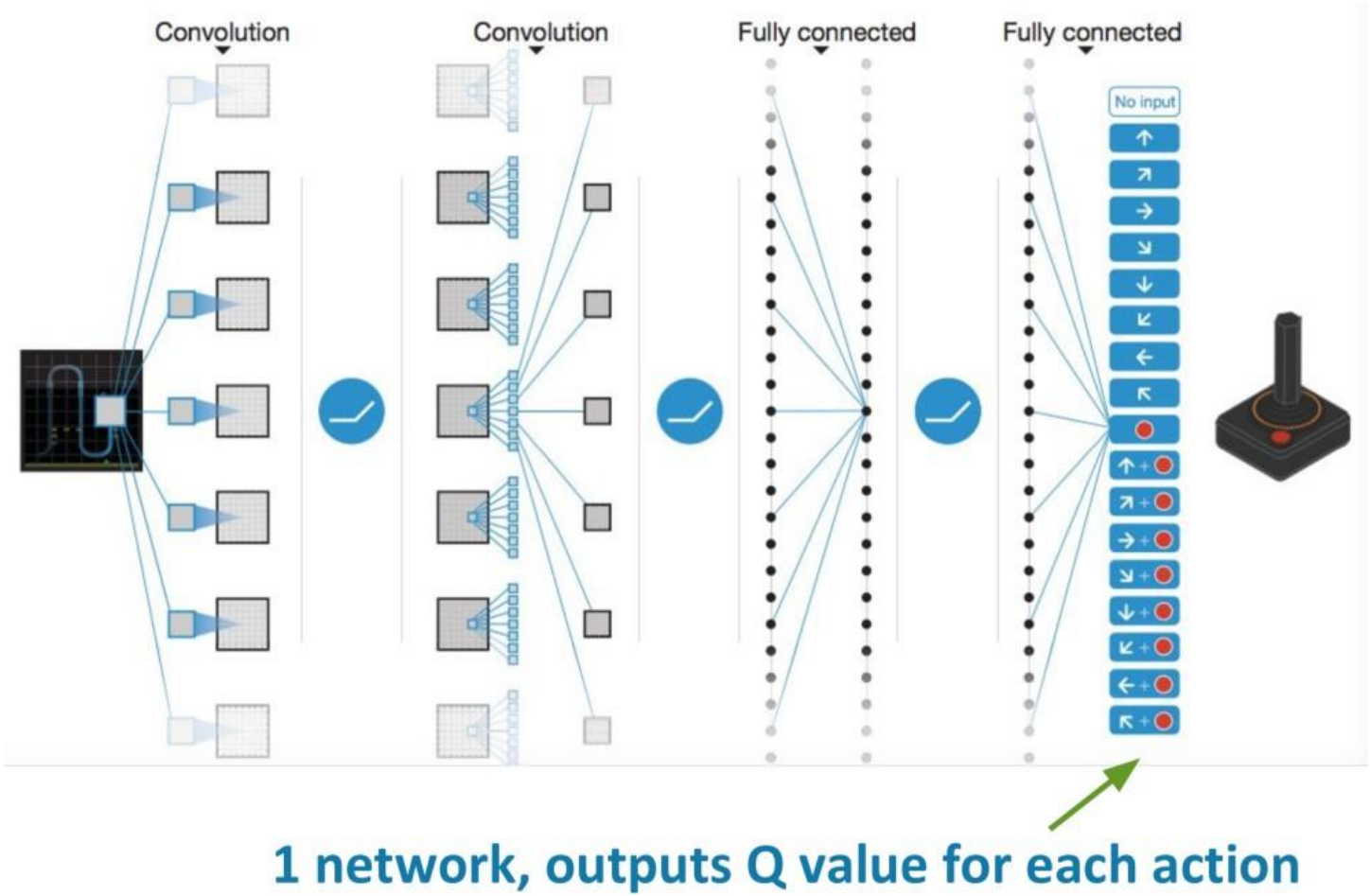
Используем отдельное множество параметров w^-, которое затем обновляется

Вычисляем показатели Q-обучения на основе предыдущего фиксированного значения параметров w^-

Оптимизируем ошибку между Q-сетью и показателями Q-обучения:

$$\mathcal{L}_i = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w_i^-) - Q(s, a, w_i) \right)^2 \right]$$

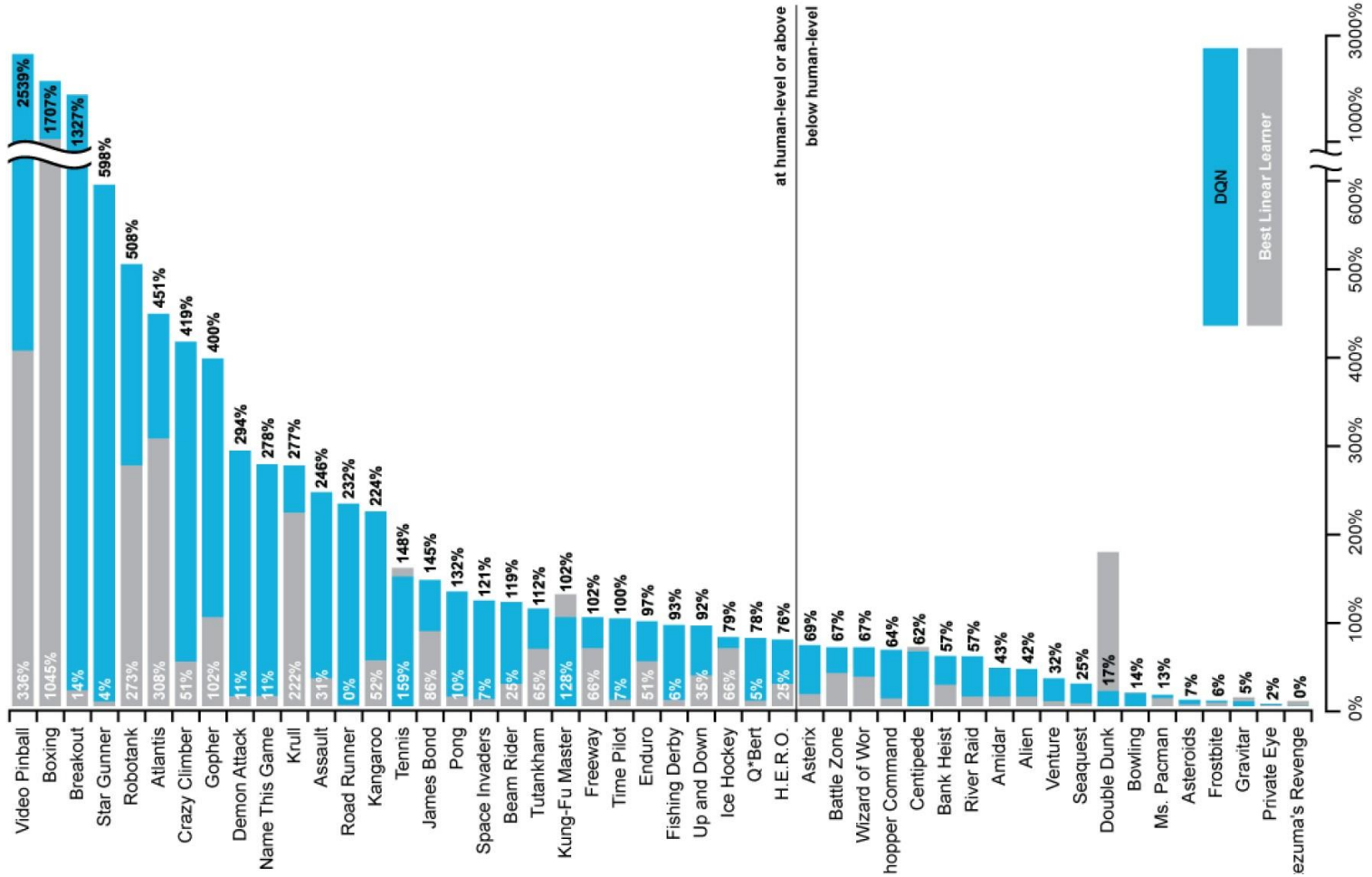
Глубокая Q-сеть



DQN: результаты обучения

Видео игры в Brakeout с образование туннеля

Результаты DQN в играх Atari



Результаты DQN в играх Atari

	Replay Fixed-Q	Replay Q-learning	No replay Fixed-Q	No replay Q-learning
Breakout	316.81	240.73	10.16	3.17
Enduro	1006.3	831.25	141.89	29.1
River Raid	7446.62	4102.81	2867.66	1453.02
Seaquest	2894.4	822.55	1003	275.81
Space Invaders	1088.94	826.33	373.22	301.99

Развитие DQN

Вслед за успешным применением DQN было предложено много различных улучшений:

Двойной DQN - Double DQN (AAAI 2016)
http://arxiv.org/abs/1509.06461

Приоритизированная память прецедентов - Prioritized Replay (ICLR 2016)
https://arxiv.org/abs/1511.05952

Дуэльный DQN - Dueling DQN (ICML 2016)
https://arxiv.org/abs/1511.06581

Радуга - Rainbow (AAAI 2018)
http://arxiv.org/abs/1710.02298

Двойное Q-обучение

Initialize $Q_1(s, a)$ and $Q_2(s, a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$
loop
  Select a_t using ϵ-greedy $\pi(s) = \arg\max_a Q_1(s_t, a) + Q_2(s_t, a)$
  Observe (r_t, s_{t+1})
  if (with 0.5 probability True) **then**

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha(r_t + Q_1(s_{t+1}, \arg\max_{a'} Q_2(s_{t+1}, a')) - Q_1(s_t, a_t))$$

  else

$$Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha(r_t + Q_2(s_{t+1}, \arg\max_{a'} Q_1(s_{t+1}, a')) - Q_2(s_t, a_t))$$

  end if
  $t = t + 1$
end loop

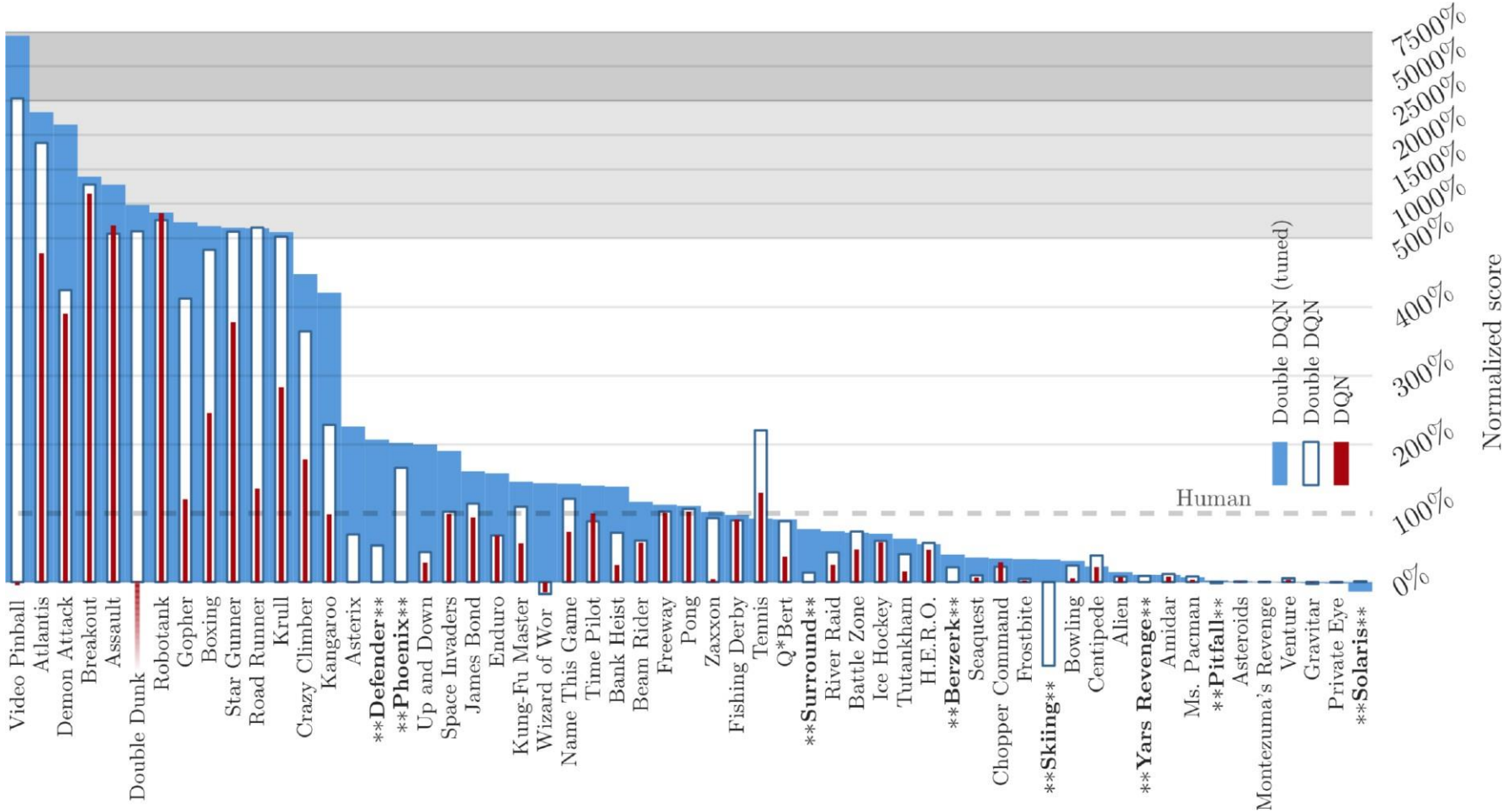
Двойной DQN

Текущая Q-сеть с параметрами w используется для выбора действий

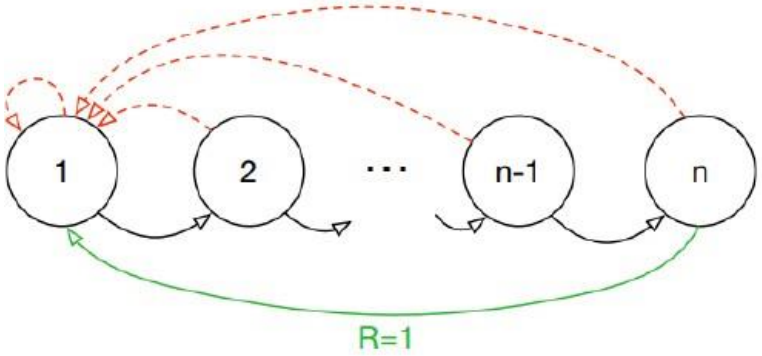
Вторая Q-сеть с параметрами w^- используется для оценки действий

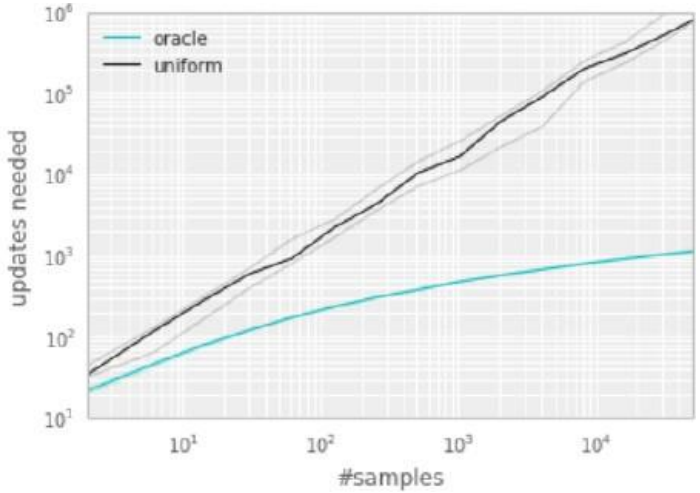
$$\Delta w = \alpha\Big(r_{t+1} + \gamma \overbrace{\hat{Q}\big(s_{t+1}, \underbrace{\arg\max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}, w)}_{\text{выбор действия}}, w^-\big)}^{\text{оценка действия}} - \hat{Q}(s_t, a_t, w)\Big)$$

Двойной DQN на играх Atari



Порядок выбора в базе прецедентов





Теоретический пример: пусть оракул, который подсказывает в каком порядке выбирать прецеденты

При удачном порядке нужно в экспоненциальное количество раз меньшее количество шагов до сходимости

Приоритизированная память прецедентов

i-ый прецедент из памяти (s_i, a_i, r_i, s_{i+1})

Выбираем прецеденты для обновления весов по приоритету, пропорциональному ошибке DQN

$$p_i = \left| r + \gamma \max_{a'} Q(s_{i+1}, a', \mathrm{w}^-) - Q(s_i, a_i, \mathrm{w}) \right|$$

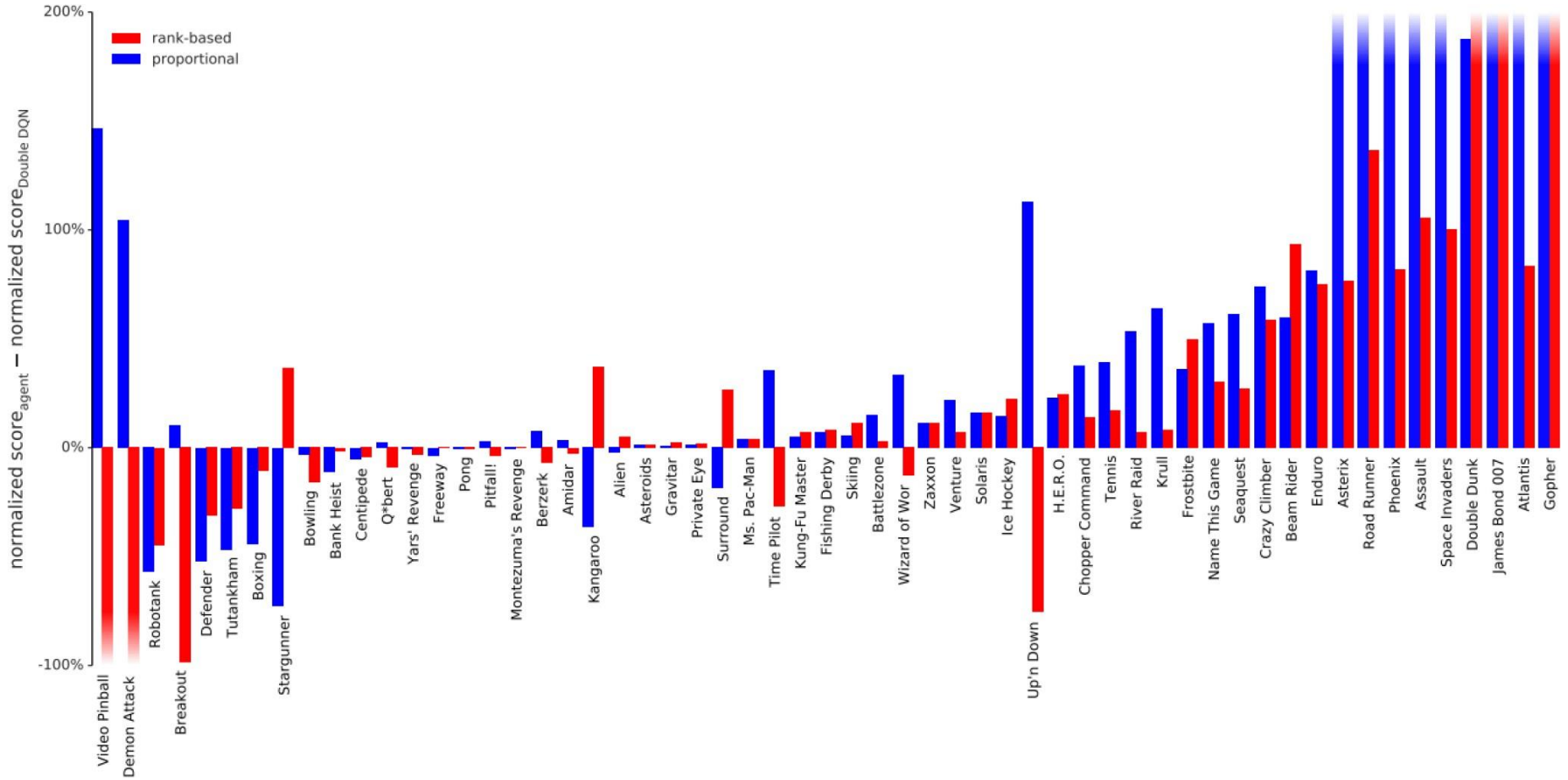
Обновляем приоритеты после каждого обновления весов

Для новых прецедентов приоритет равен 0

Стохастическая приоритизация

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

PRB vs DDQN


rank-based
proportional
200%
100%
0%
-100%
normalized score$_{agent}$ − normalized score$_{Double\ DQN}$
Video Pinball
Demon Attack
Robotank
Breakout
Defender
Tutankham
Boxing
Stargunner
Bowling
Bank Heist
Centipede
Q*bert
Yars' Revenge
Freeway
Pong
Pitfall!
Montezuma's Revenge
Berzerk
Amidar
Kangaroo
Alien
Asteroids
Gravitar
Private Eye
Surround
Ms. Pac-Man
Time Pilot
Kung-Fu Master
Fishing Derby
Skiing
Battlezone
Wizard of Wor
Zaxxon
Venture
Solaris
Ice Hockey
Up'n Down
H.E.R.O.
Chopper Command
Tennis
River Raid
Krull
Frostbite
Name This Game
Seaquest
Crazy Climber
Beam Rider
Enduro
Asterix
Road Runner
Phoenix
Assault
Space Invaders
Double Dunk
James Bond 007
Atlantis
Gopher

Функция преимущества

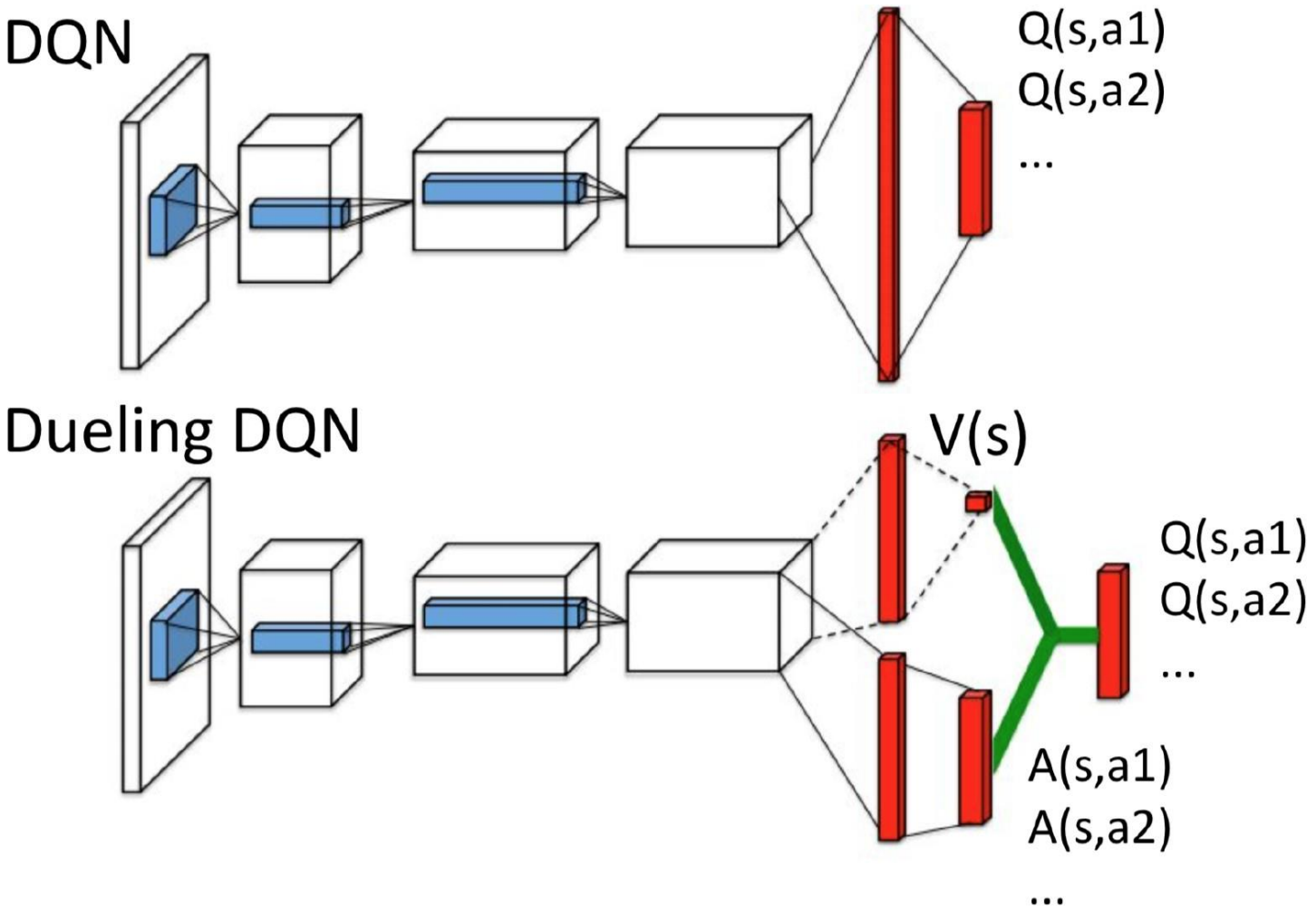
Идея: признаки, используемые при определении полезности состояния, могут не совпадать с признаками, определяющими успешность действия

Пример: счет играет роль в определении полезности состояния, но не нужен для выбора действия

Функция преимущества (advantage function):

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

Дуэльный DQN



Дуэльный DQN

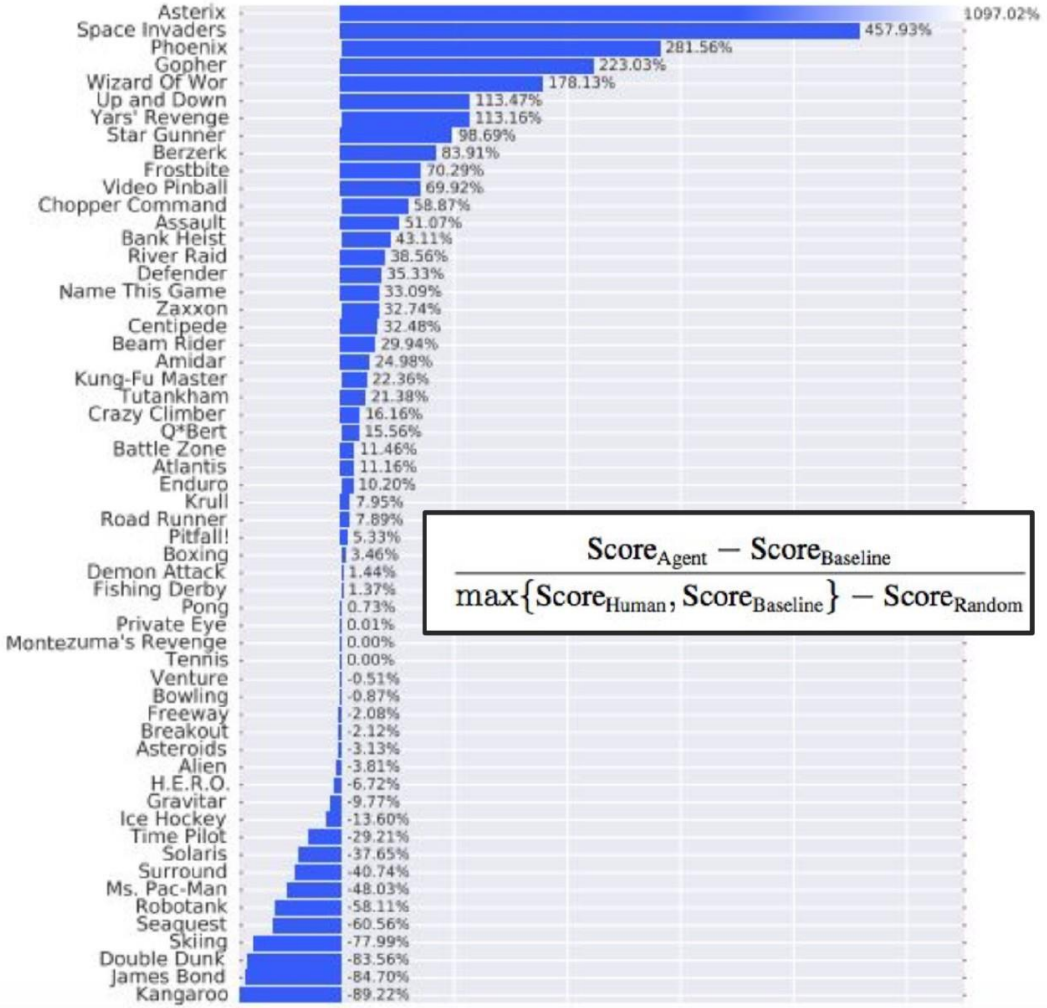
Назначаем $A(s, a) = 0$ для выбранного действия a

$$\hat{Q}(s, a, \mathrm{w}) = \hat{V}(s, \mathrm{w}) + \left(\hat{A}(s, a, \mathrm{w}) - \max_{a'} \hat{A}(s', a', \mathrm{w}) \right)$$

Используем среднее в качестве базового значения

$$\hat{Q}(s, a, \mathrm{w}) = \hat{V}(s, \mathrm{w}) + \left(\hat{A}(s, a, \mathrm{w}) - \frac{1}{|A|} \sum_{a'} \hat{A}(s', a', \mathrm{w}) \right)$$

Дуэльный DQN на Atari



Новые улучшения DQN

DQN и другие улучшения показали, что удается успешно сочетать глубокое обучение и обучение с подкреплением

Кроме DDQN, PRB, Dueling предлагались и другие улучшения: многошаговое обучение (multi-step learning), обучение по распределениям (distributional RL), шумовые слои (noisy net)

Оказывается, что многие улучшения могут быть применены одновременно ⇒ алгоритм «радуги» (Rainbow)

Многошаговое обучение

Вместо жадного (одношагового) подхода, будем использовать более «дальнюю» оценку полезности, которая будет получена после *n* шагов:

$$R_t^{(n)} = \sum_{k=1}^{n-1} \gamma^{(k)} r_{t+k+1}$$

В этом случае функция потерь будет выглядеть так:

$$\mathcal{L}_i = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(R_t^{(n)} + \gamma^{(n)} \max_{a'} Q(s', a', \mathrm{w}_i^-) - Q(s, a, \mathrm{w}_i) \right)^2 \right]$$

Такое многошаговое обучение ускоряет процесс обучения в том случае, когда является изменяемым гиперпараметром

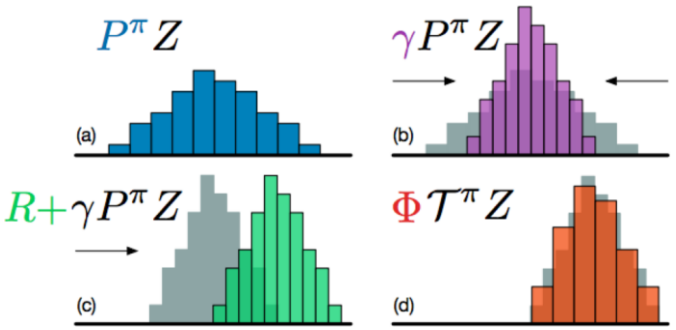
Обучение по распределениям

Вместо оценки ожидаемой отдачи $Q(s, a)$ будем оценивать **распределений отдач** с матожиданием $Q(s, a)$
Пусть z – носитель распределения вероятностей полезности (распределение по категориям):

$$z^i = V_{min} + (i-1)\frac{V_{max} - V_{min}}{N-1}, \; i \in \{1, \ldots, N\}$$

Наша цель - настроить параметры сети w, задающие распределение $d_t = (z, p(s_t, a_t, w))$, где $p(s_t, a_t, w)$ – вероятностная мера для каждого атома i, так, чтобы оно было близко к целевому распределению

В данном случае минимизируется KL-дивергенция $d'_t = (r_{t+1} + \gamma z, p(s_{t+1}, a^*_{t+1}, w'))$



Шумовые слои

Идея – заменить линейный слой шумовым линейным слоем, в котором комбинируются детерминированные и случайные параметры:

$$y = (b + Wx) + (b_{noisy} \odot \epsilon^b + (W_{noisy} \odot \epsilon^W)x)$$

После некоторого обучения, сеть приобретет способность игнорировать шумовой поток, но в различной степени в зависимости от состояний среды

Таким образом мы реализуем условное исследование среды

Это замена ϵ-жадному алгоритмы: мы привносим шум в параметрическое пространство для поддержания необходимого уровня исследования среды

Алгоритм Rainbow

Собираем все улучшения вместе:

Используем обучения по распределениям для оценки распределения отдачи вместо ожидаемой отдачи

Заменяем одношаговую функцию потерь по распределениям на многошаговую:

$$d_t'^{(n)} = (R_{t+1}^{(n)} + \gamma^{(n)} z, p(s_{t+n}, a_{t+n}^*, w'))$$

Комбинируем эту функцию потерь со стандартной функцией потерь двойного Q-обучения

Используем идею приоритезированной памяти прецедентов для выборки переходов с учетом приоритета по функции потерь

$$D_{RL}(\Phi_z d_t^{(n)} \| d_t)$$

Возвращаемые распределения используются в дульной архитектуре сети

$$p^i(s, a, w) \propto \exp\left(\hat{V}^i(s, w_1) + \left(\hat{A}^i(s, a, w_2) - \frac{1}{|A|}\sum_{a'} \hat{A}^i(s', a', w_2)\right)\right)$$

Привносим шум заменой линейных слоев на шумовые

Rainbow: производительность и значимость