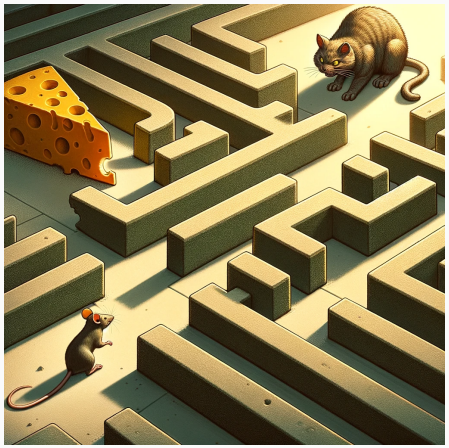
ВВЕДЕНИЕ В RL И МНОГОРУКИЕ БАНДИТЫ

Сергей Николенко

СПбГУ — Санкт-Петербург

25 апреля 2024 г.



Random facts:

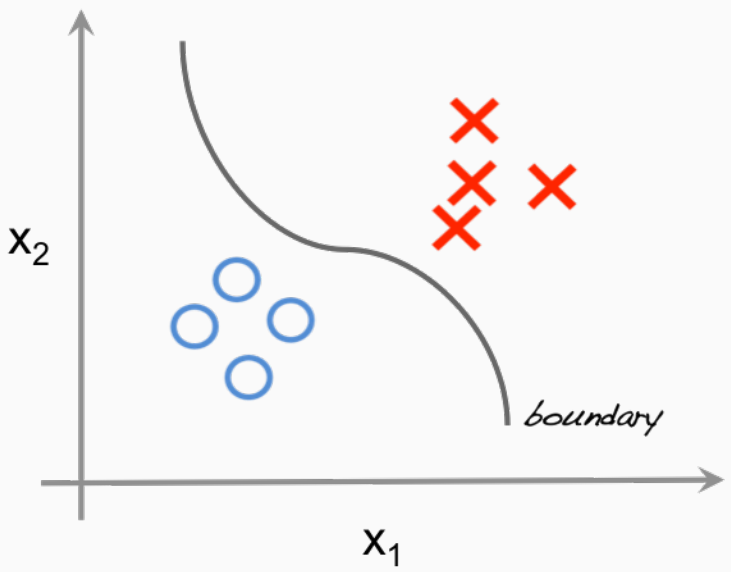
- 25 апреля — День ДНК; именно 25 апреля 1953 г. в *Nature* вышли три статьи Джеймса Уотсона, Фрэнсиса Крика, Мориса Уилкинса, Розалинд Франклин и их коллег, а 25 апреля 2003 г. объявили о завершении проекта «Геном человека»
- 25 апреля 1792 г. разбойник Николя Пеллетье первым испробовал на себе гильотину
- 25 апреля 1826 г. в Англии был запатентован первый автомобиль с двигателем внутреннего сгорания, а 25 апреля 1901 г. в штате Нью-Йорк были впервые в мире введены автомобильные номера
- 25 апреля 1945 г. советские и американские войска встретились на Эльбе
- 25 апреля 1960 г. субмарина *USS Triton* завершила подводное кругосветное плавание
- 25 апреля 1983 г. Юрий Андропов пригласил в Советский Союз Саманту Смит
- 25 апреля 1993 г. на референдуме в России большинство поддержали политику Бориса Ельцина, а 25 апреля 2007 г. прошли его похороны

Обучение с подкреплением

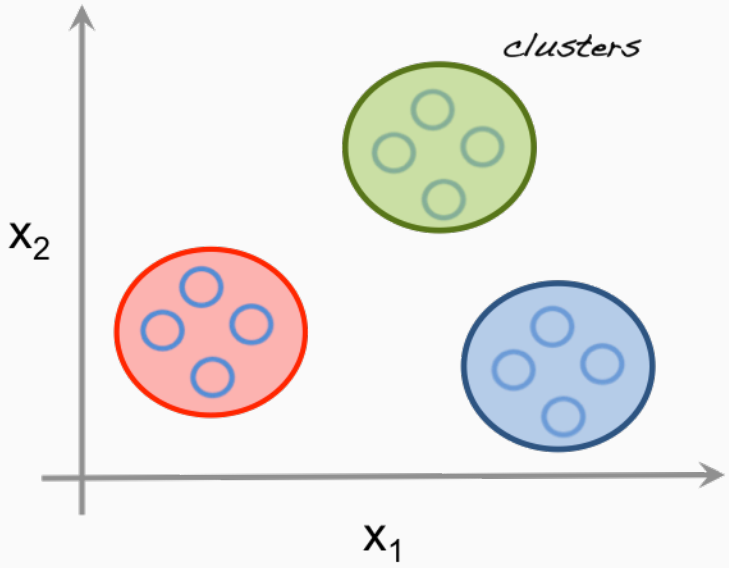
Постановка задачи

- В машинном обучении задача обычно ставится так:
  - или есть набор «правильных ответов», и нужно его продолжить на всё пространство (supervised learning),
  - или есть набор тестовых примеров без дополнительной информации, и нужно понять его структуру (unsupervised learning).

Supervised learning



Unsupervised learning

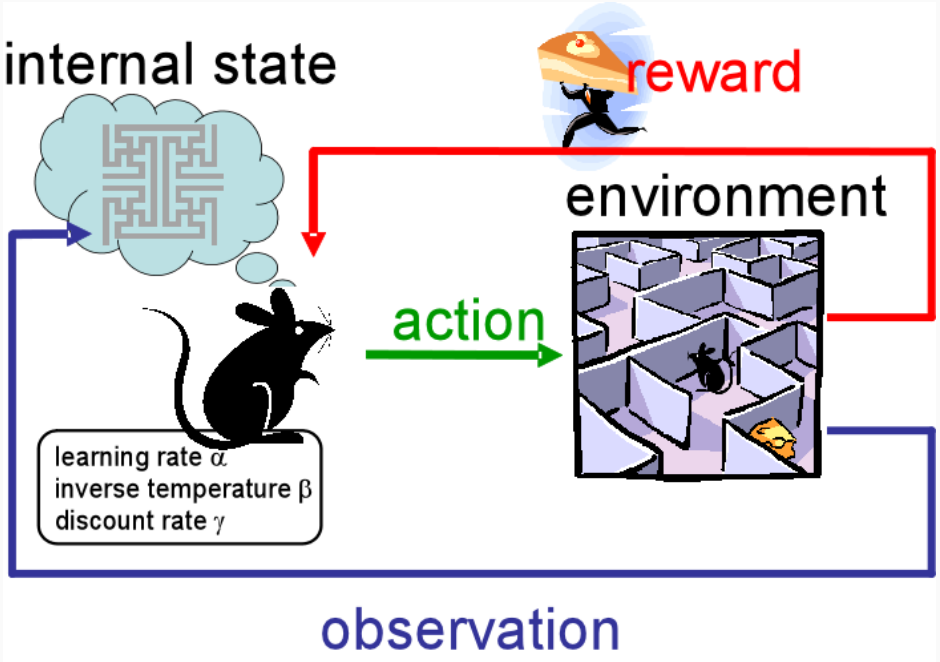


Постановка задачи

- Но как работает обучение в реальной жизни?
- Как ребёнок учится ходить?
- Мы далеко не всегда знаем набор правильных ответов, мы просто делаем то или иное действие и получаем результат.

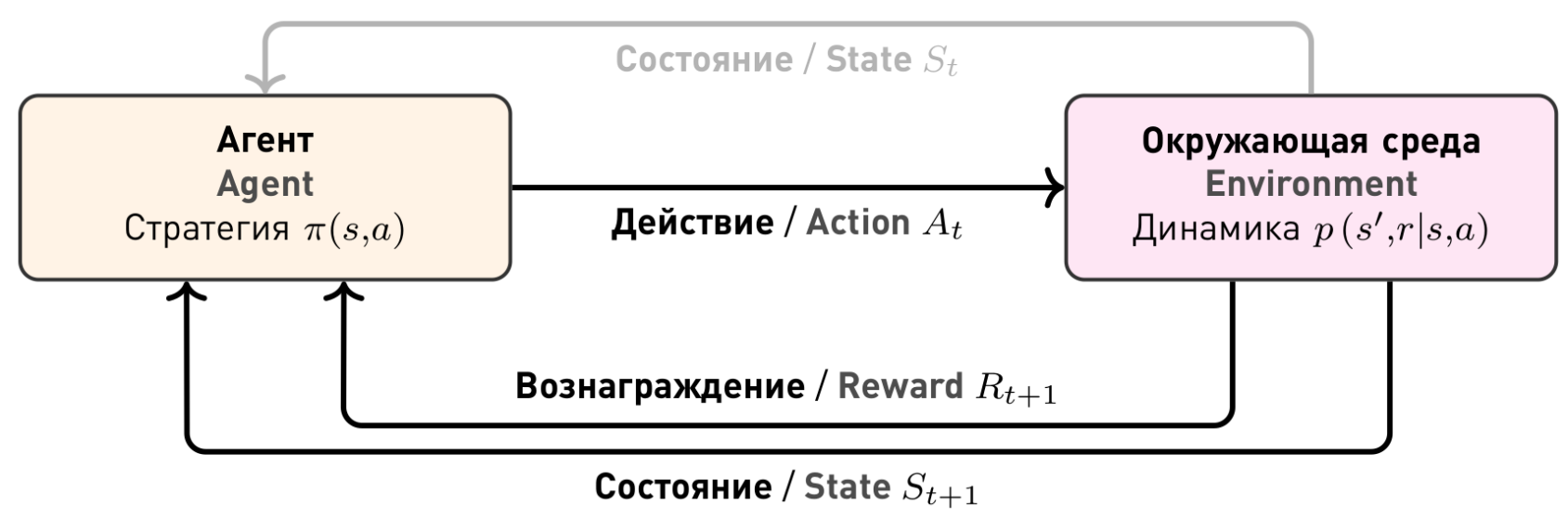
Постановка задачи

- Отсюда и обучение с подкреплением (reinforcement learning).
- Агент взаимодействует с окружающей средой, предпринимая действия; окружающая среда его поощряет за эти действия, а агент продолжает их предпринимать.



Постановка задачи -- формально

- На каждом шаге агент может находиться в состоянии $s \in S$.
- На каждом шаге агент выбирает из имеющегося набора действий некоторое действие $a \in A$.
- Окружающая среда сообщает агенту, какую награду r он за это получил и в каком состоянии s' после этого оказался.



- (Sutton, Barto, 1998)

Пример

- Диалог:

Среда: Агент, ты в состоянии 1; есть 5 возможных действий.
Агент: Делаю действие 2.
Среда: Даю тебе 2 единицы за это. Попал в состояние 5, есть 2 возможных действия.
Агент: Делаю действие 1.
Среда: Даю тебе за это -5 единиц. Попал в состояние 1, есть 5 возможных действий.
Агент: Делаю действие 4.
Среда: Даю тебе 14 единиц за это. Попал в состояние 3, есть 3 возможных действия...

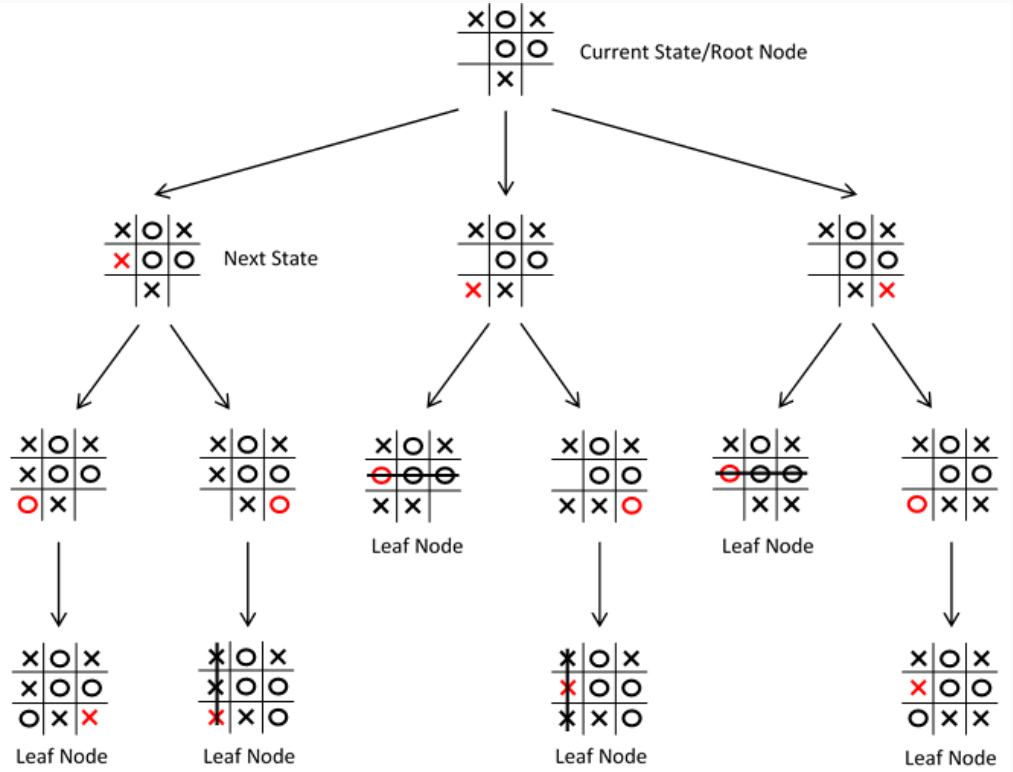
- В этом примере агент успел вернуться в состояние 1 и исследовать ранее не пробовавшуюся опцию 4 (получив за это существенную награду).

Exploitation vs. exploration

- Каждый алгоритм должен и изучать окружающую среду, и пользоваться своими знаниями, чтобы максимизировать прибыль.
- Вопрос — как достичь оптимального соотношения? Та или иная стратегия может быть хороша, но вдруг она не оптимальная?
- Этот вопрос всегда присутствует в обучении с подкреплением.

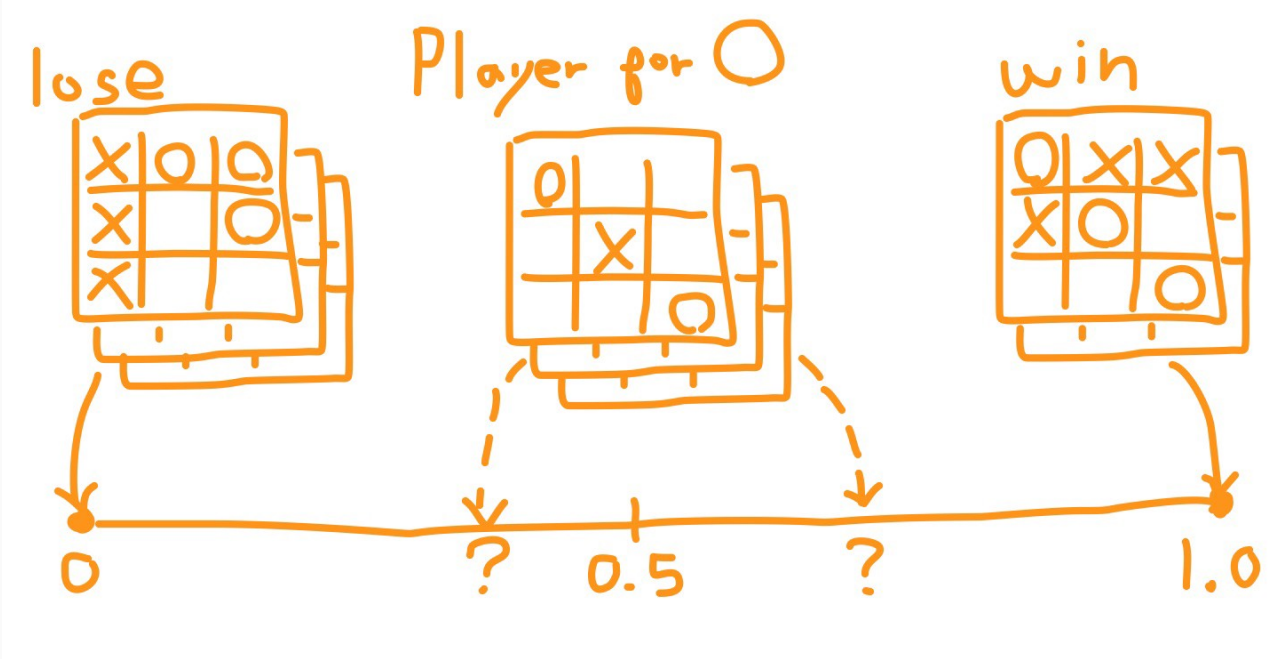
Пример

- Пример: крестики-нолики. Как научить машину играть и *выигрывать* в крестики-нолики?
- Можно, конечно, дерево построить, но это не масштабируется.



Пример

- Состояния – позиции на доске.
- Для каждого состояния введём функцию $V(s)$ (value function).
- Подкрепление приходит только в самом конце, когда мы выиграли или проиграли; как его распространить на промежуточные позиции?



Пример

- Небольшой трейлер того, что будет дальше: можно пропагировать оценку позиции обратно.
- Если мы попадали из s в s', апдейтим

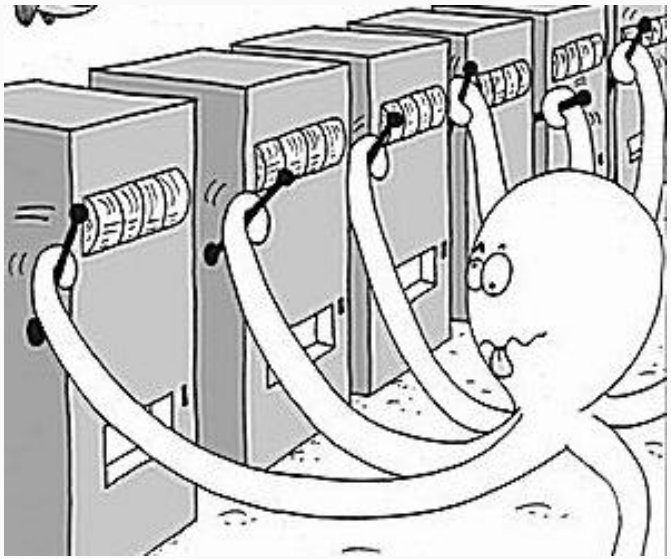
$$V(s) := V(s) + \alpha \left[V(s') - V(s) \right].$$

- Это называется TD-обучение (temporal difference learning), оно очень хорошо работает на практике и лежит в основе и AlphaGo, и много чего ещё.
- Но это будет ещё не сегодня..

Многорукие бандиты

Агенты с одним состоянием

- Формально всё то же самое, но $|S| = 1$, т.е. состояние агента не меняется. У него фиксированный набор действий A и возможность выбора из этого набора действий.
- Модель: агент в комнате с несколькими игровыми автоматами. У каждого автомата своё ожидание выигрыша.
- Нужно заработать побольше: exploration vs. exploitation.



Жадный алгоритм

- *Жадный алгоритм*: всегда выбирать стратегию, максимизирующую прибыль; прибыль можно оценить как среднее вознаграждение, полученное от этого действия:

$$Q_t(a) = \frac{r_1 + r_2 + ... + r_{k_a}}{k_a}.$$

- Что не так с таким алгоритмом?

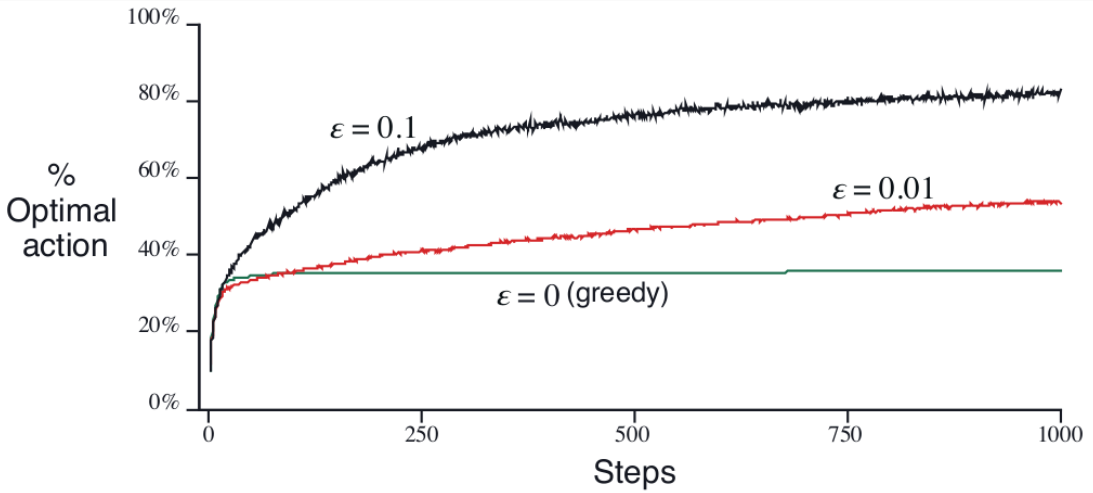
Жадный алгоритм

- Оптимум легко проглядеть, если на начальной выборке не повезёт (что вполне возможно).
- Поэтому полезная эвристика — *оптимизм при неопределённости*.

- То есть выбирать жадно, но при этом прибыль оценивать оптимистично, и нужны серьёзные свидетельства, чтобы отклонить стратегию.

Случайные стратегии

- *ϵ-жадная стратегия* (ϵ-greedy): выбрать действие с наилучшей ожидаемой прибылью с вероятностью $1 - \epsilon$, а с вероятностью ϵ выбрать случайное действие.



- Обычно начинают с больших ϵ, затем уменьшают.
- Алгоритм не отличает хорошую альтернативу от бесполезной, но всё равно разумный, про него много чего доказать можно.

Интервальные оценки

- Естественный способ применить оптимистично–жадный метод – доверительные интервалы.
- Для каждого действия мы храним статистику n и w, а потом вычисляем доверительный интервал для вероятности успеха (с границей $1 - \alpha$) и для выбора стратегии используем *верхнюю границу* этого интервала.
- Например, для испытаний Бернулли (монетка) с вероятностью $.95$ среднее лежит в интервале

$$\left(\bar{x} - 1.96\frac{s}{\sqrt{n}}, \bar{x} + 1.96\frac{s}{\sqrt{n}}\right),$$

где 1.96 берётся из распределения Стьюдента, n — количество испытаний, $s = \sqrt{\frac{\sum(x-\bar{x})^2}{n-1}}$.

Правило инкрементального обновления

- Теперь – о среднем и $Q_t(a)$.
- Как пересчитывать $Q_t(a) = \frac{r_1 + \ldots + r_{k_a}}{k_a}$ при поступлении новой информации?
- Довольно просто:

$$Q_{k+1} = \frac{1}{k+1} \sum_{i=1}^{k+1} r_i = \frac{1}{k+1} \left[r_{k+1} + \sum_{i=1}^{k} r_i \right] =$$
$$= \frac{1}{k+1} \left(r_{k+1} + kQ_k \right) = Q_k + \frac{1}{k+1} \left(r_{k+1} - Q_k \right).$$

Правило инкрементального обновления

- Это частный случай общего правила – сдвигаем оценку так, чтобы уменьшалась ошибка:

  $$\text{НоваяОценка} := \text{СтараяОценка} + \text{Шаг}\,[\text{Цель} - \text{СтараяОценка}]\,.$$

- Заметим, что шаг у среднего непостоянный, $\alpha_k(a) = \frac{1}{k_a}$:

  $$Q_{k+1} = Q_k + \frac{1}{k+1}\left(r_{k+1} - Q_k\right).$$

- Изменяя последовательность шагов, можно добиться других эффектов.

Нестационарная задача

- Часто бывает, что выплаты из разных бандитов на самом деле нестационарны, т.е. меняются со временем.
- В такой ситуации имеет смысл давать большие веса недавней информации и маленькие веса – давней.
- Пример: у правила апдейта

$$Q_{k+1} = Q_k + \alpha \left[r_{k+1} - Q_k \right]$$

с постоянным α фактически веса затухают экспоненциально:

$$Q_k = Q_{k-1} + \alpha \left[r_k - Q_{k-1} \right] = \alpha r_k + (1-\alpha) Q_{k-1} =$$

$$= \alpha r_k + (1-\alpha)\alpha r_{k-1} + (1-\alpha)^2 Q_{k-2} = (1-\alpha)^k Q_0 + \sum_{i=1}^{k} \alpha (1-\alpha)^{k-i} r_i.$$

Нестационарная задача

- Такое правило апдейта не обязательно сходится, но это и хорошо – мы хотим следовать за целью.
- Общий результат – правило апдейта сходится, если последовательность весов удовлетворяет

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty \quad \text{и} \quad \sum_{k=1}^{\infty} \alpha_k^2(a) < \infty.$$

- Например, для $\alpha_k(a) = \frac{1}{k_a}$ явно сходится.

Стратегии, минимизирующие regret

Динамическое программирование

- Предположим, что агент действует на протяжении h шагов.
- Используем байесовский подход для определения оптимальной стратегии.
- Начинаем со случайных параметров $\{p_i\}$, например, равномерно распределённых, и вычисляем отображение из *belief states* (состояния после нескольких раундов обучения) в действия.
- Состояние выражается как $\mathcal{S} = \{n_1, w_1, \dots, n_k, w_k\}$, где каждого бандита i запустили n_i раз и получили w_i единичек (считаем, что результат бинарный).

Динамическое программирование

- $V^*(\mathcal{S})$ — ожидаемый оставшийся выигрыш.
- Рекурсивно: если $\sum_{i=1}^{k} n_i = h$, то больше нечего делать, и $V^*(\mathcal{S}) = 0$.
- Если знаем V^* для всех состояний, когда осталось t запусков, сможем пересчитать и для $t + 1$:

$$V^*(n_1, w_1, \dots, n_k, w_k) = \\ = \max_i \left(\rho_i(1 + V^*(\dots, n_i + 1, w_i + 1, \dots)) + \right. \\ \left. (1 - \rho_i)V^*(\dots, n_i + 1, w_i, \dots)\right),$$

где ρ_i — апостериорные вероятности того, что действие i оправдается (если изначально p_i равномерно распределены, то $\rho_i = \frac{w_i+1}{n_i+2}$).

Байесовский подход к многоруким бандитам

- А теперь давайте посмотрим на многоруких бандитов в общем вероятностном виде.
- Для простоты – бинарный случай, выплата либо 1, либо 0.

Байесовский подход к многоруким бандитам

- Пусть во время t у нас состояние $\mathbf{s}_t = (s_{1t}, \dots, s_{Kt})$ для K ручек, и мы хотим дёрнуть такую ручку, чтобы максимизировать общее ожидаемое число успехов.
- Есть функция вознаграждения $R_i(\mathbf{s}_t, \mathbf{s}_{t+1})$ – награда за дёргание ручки i (a_i), которое переводит состояние $\mathbf{s}_t$ в $\mathbf{s}_{t+1}$.
- Есть вероятность перехода $p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, a_i\right)$.
- И мы хотим обучить стратегию $\pi(\mathbf{s}_t)$, которая возвращает, какую ручку дёргать.

Байесовский подход к многоруким бандитам

- Тогда value function в самом общем виде до горизонта T:

$$V_T(\pi, \mathbf{s}_0) = \mathrm{E}\left[R_{\pi(\mathbf{s}_0)}(\mathbf{s}_0, \mathbf{s}_1) + V_{T-1}(\pi, \mathbf{s}_1)\right] =$$
$$= \int p\left(\mathbf{s}_1 \mid \mathbf{s}_0, \pi(\mathbf{s}_0)\right)\left[R_{\pi(\mathbf{s}_0)}(\mathbf{s}_0, \mathbf{s}_1) + V_{T-1}(\pi, \mathbf{s}_1)\right] d\mathbf{s}_1.$$

- Если всё известно, и T невелико, то можно, опять же, динамическим программированием.
- Но даже подсчитать отдачу от фиксированной стратегии может быть очень дорого, не говоря уж об оптимизации.

Байесовский подход к многоруким бандитам

- Если T большой/неограниченный, логично рассмотреть

$$R = R(0) + \gamma R(1) + \gamma^2 R(2) + \dots, \quad 0 < \gamma < 1.$$

- Теорема Гиттинса (1979): задачу поиска оптимальной стратегии

$$\pi(\mathbf{s}_t) = \arg\max_\pi V(\pi, \mathbf{s}_t = (s_{1t}, \dots, s_{Kt}))$$

можно факторизовать и свести к

$$\pi(\mathbf{s}_t) = \arg\max_i \gamma(s_{it}).$$

- $\gamma(s_{it})$ – *индекс Гиттинса.*

Байесовский подход к многоруким бандитам

- Это очень крутой результат, который очень сильно сокращает задачу:

Professor P. Whittle (Statistical Laboratory, Cambridge): We should recognize the magnitude of Dr Gittins' achievement. He has taken a classic and difficult problem, that of the multi-armed bandit, and essentially solved it by reducing it to the case of comparison of a single arm with a standard arm. In this paper he brings a number of further insights. Giving words their everyday rather than their technical usage, I would say that my admiration for this piece of work is unbounded, meaning, of course, very great.

Despite the fact that Dr Gittins proved his basic results some seven years ago, the magnitude of his advance has not been generally recognized and I hope that one result of tonight's meeting will be that the strength of his contribution, its nature and its significance will be apparent to all.

As I said, the problem is a classic one; it was formulated during the war, and efforts to solve it so sapped the energies and minds of Allied analysts that the suggestion was made that the problem be dropped over Germany, as the ultimate instrument of intellectual sabotage. In the event, it seems to have landed on Cardiff Arms Park. And there is justice now, for if a Welsh Rugby pack scrumming down is not a multi-armed bandit, then what is?

- Но, к сожалению, индекс Гиттинса подсчитать тоже очень вычислительно трудно; поэтому не будем в это углубляться.

Оценки на regret

- Другой вариант – давайте рассчитаем приоритет каждой ручке i так, чтобы непосредственно regret ограничить.
- [Auer et al., 2002]: стратегия UCB1. Учитывает неопределённость, «оставшуюся» в той или иной ручке, старается ограничить regret. Если мы из t экспериментов n_j раз дёрнули за j-ю ручку и получили среднюю награду $\bar{x}_i$, алгоритм UCB1 присваивает ей приоритет

$$\text{Priority}_i = \bar{x}_i + a(j,t) = \bar{x}_i + c\sqrt{\frac{\log t}{n_j}}.$$

Дёргать дальше надо за ручку с наивысшим приоритетом.

Оценки на regret

- При таком подходе для $c = \sqrt{2}$ можно доказать, что субоптимальные ручки будут дёргать $O(\log T)$ раз, и regret будет $O(\sqrt{KT \log T})$.
- Если хватит времени, давайте оценку докажем...
- Но можно и ещё лучше (но доказательства будут ещё сложнее):
  - UCB2 — дёргаем за ручки по несколько раз, более сложная форма добавки;
  - UCB-Tuned — заменим $\sqrt{\frac{2 \log t}{n_j}}$ на

$$\sqrt{\frac{\log t}{n_j} \min\left(\frac{1}{4}, V_j(n_j)\right)}, \quad \text{где}$$

$$V_j(n_j) = \left(\frac{1}{n_j} \sum_{\tau=1}^{n_j} r_\tau^2\right) - \left(\frac{1}{n_j} \sum_{\tau=1}^{n_j} r_\tau\right)^2 + \sqrt{\frac{2 \log t}{n_j}}.$$

Сэмплирование по Томпсону

- И ещё один вариант: сэмплирование по Томпсону (Thompson sampling).
- Как добавить байесовской мудрости в наших многоруких бандитов?
- Идея: давайте не присваивать какой-то приоритет, а *сэмплировать ожидания наград из их апостериорных распределений* и выбирать максимальную.
- Любопытно, что тут есть два термина:
  - Thompson sampling — сэмплируем ожидания из апостериорных распределений и берём максимум;
  - probability matching — выбираем действие с вероятностями, с которыми это действие будет оптимальным согласно текущим апостериорным распределениям; это встречается где-то самостоятельно, но в целом эквивалентно.

Конкурентные бандиты

Adversarial bandits

- Конкурентные бандиты (adversarial bandits) — обобщение многоруких бандитов, в котором награды выбираются не из фиксированных распределений, а выбираются противником (adversary). Т.е. процесс из трёх шагов:
  - противник выбирает распределения наград;
  - агент выбирает ручку, не зная, какие распределения выбрал противник;
  - награда выбирается из соответствующего распределения.
- Здесь есть информационная асимметрия между агентом и противником; без ограничений противник всегда сможет победить любую стратегию, конечно, он выбирает награды $r_j(t)$ произвольно.

Adversarial bandits

- Но сделать что-то можно!
- Можно даже доказать оценки на regret; под regret здесь понимается разница между доходом глобально наилучшей ручки

$$G_{\max}(T) = \max_j \sum_{t=1}^{T} r_j(t)$$

  и ожиданием нашего дохода.
- А идея алгоритма будет в том, чтобы
  - сэмплировать действие с вероятностями $p_1(t), \dots, p_K(t)$ из распределения, которое будет смесью равномерного и экспоненциальных весов от текущей оценки накопленной награды;
  - делать действие, получить награду $r_{i_t}(t)$, но ожидаемую награду считать как $\hat{r}_{i_t}(t) = r_{i_t}(t)/p_i(t)$; тогда $\mathbb{E}\left[\hat{r}_{i_t}(t) \mid i_1, \dots, i_{t-1}\right] = r_{i_t}(t)$.

Adversarial bandits

- Алгоритм Exp3 (Exponential-weight algorithm for Exploration and Exploitation):
  - для данного $\gamma \in [0, 1]$, инициализируем $w_i(1) = 1$, $i = 1, \dots, K$;
  - на каждом раунде t:
    - считаем вероятности для каждого i:

$$p_i(t) = (1 - \gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K};$$

    - берём следующее действие i_t случайно с вероятностью $p_i(t)$, получаем награду $r_{i_t}(t)$;
    - обновляем вес этого действия (остальные не меняются):

$$w_{i_t}(t+1) = w_{i_t}(t) e^{\frac{\gamma}{K}\frac{r_{i_t}(t)}{p_i(t)}}.$$

- Для такого алгоритма можно доказать adversarial оценку $2.63\sqrt{G_{\max} K \log K}$ для $\gamma = \min\left(1, \frac{K \log K}{(e-1)g}\right)$, где $G_{\max}$ – верхняя оценка на максимальный доход ручки $\max_j \sum_{t=1}^{T} r_j(t)$.

Контекстуальные бандиты

Контекстуальные бандиты

- А что если жизнь устроена сложнее? Пусть мы каждую ручку дёргаем в некотором *контексте*; например, даём рекомендации пользователям, и при этом о пользователях что-то знаем.
- Т.е. на каждом шаге наблюдаем контекст $x_t \in C$, потом выбираем $I_t \in 1, \dots, K$, получаем $r_t \sim p(r_t \mid I_t, x_t)$.
- Тогда мы уже должны не одну ручку выбрать, а обучить стратегию $\pi : C \to 1, \dots, K$.

Контекстуальные бандиты

- Наивный подход: давайте запустим Exp3 для каждого контекста по отдельности.
- Это не так уж плохо, если контекстов мало.
- Но получается дополнительный сомножитель $\sqrt{|C|}$ в оценке на regret, а это нехорошо, вдруг их очень много.

Контекстуальные бандиты

- Другая идея – давайте вместо этого запускать Exp3 *по стратегиям* (пусть их мало); на каждом раунде t:
  - получаем совет $\xi_{k,t}$ от каждой стратегии («эксперта») $k \in \Pi$ в виде распределения вероятностей на ручках;
  - берём следующее действие I_t случайно с распределением $p_t = \mathbb{E}_{k \sim q_t}\left[\xi_{k,t}\right]$, получаем награду $x_{I_t,t}$;
  - вычисляем ожидаемую награду для каждого i $\tilde{x}_{i,t}$ и ожидаемую награду для каждой стратегии

$$\tilde{y}_{k,t} = \mathbb{E}_{i \sim \xi_{k,t}}\left[\tilde{x}\right]_{i,t} = \sum_{i=1}^{K} \xi_{k,t}(i)\tilde{x}_{i,t}.$$

  - обновляем распределение на стратегиях

$$q_{j,t+1} \propto e^{-\eta \sum_{s=1}^{t} \tilde{y}_{k,s}}.$$

- Это алгоритм Exp4 (Exponential-weight algorithm for Exploration and Exploitation with Experts)
- Всё понятно?..

Контекстуальные бандиты

- ...но откуда возьмётся $\tilde{x}_{i,t}$?
- Очень важный трюк в reinforcement learning: off-policy estimation.
- Пусть мы хотим оценить качество стратегии π, но играли мы по другой стратегии p. Что делать?
- Нам нужно оценить

$$R(x, \pi(x)) = \frac{1}{n} \sum_{s=1}^{n} \mathbb{E}_{r \sim p(r|x_s, \pi(x_s))} [r] .$$

- Это было бы легко, если бы мы могли оценить для каждой ручки a

$$R(x, a) = \mathbb{E}_{r \sim p(r|a,x)} [r] .$$

Контекстуальные бандиты

- Оказывается, это можно сделать по данным другой стратегии. Рассмотрим

$$\hat{R}(x_s, a) = r_s \frac{[a_s = a]}{p(a_x \mid x_s)}.$$

- Тут надо только предполагать, что $p(a_x \mid x_s)$ положительно, т.е. стратегия p покрывает стратегию π.
- Тогда

$$\mathbb{E}\left[\hat{R}(x_s, a) \mid x_s, a\right] = \mathbb{E}_{r_s}\left[\mathbb{E}_{a_s \sim p(x_s)}\left[r_s \frac{[a_s = a]}{p(a_x \mid x_s)}\right]\right] =$$
$$= \mathbb{E}_{r_s}\left[r_s \frac{p(\pi(x_s) = a_s)}{p(a_s \mid x_s)}\right] = R(x_s, a),$$

  у которого $a_s \sim p$.
- Т.е. надо просто перевзвесить вознаграждения.

Контекстуальные бандиты

- Итого:

Algorithm 2 Exp4 Algorithm (Exponential weights algorithm for Exploration and Exploitation with Experts)

Input: Set of K arms, set of experts Π.
Parameter: real number η
Let q_1 be the uniform distribution over the experts (policies), $\{1, \cdots, |\Pi|\}$.
For each round $t = 1, \cdots, T$:

- **Receive** expert advice $\xi_{k,t}$ for each expert $k \in \Pi$, where each $\xi_{k,t}$ is a probability distribution over arms.
- **Draw** an arm I_t from the probability distribution $p_t = (p_{1,t}, \cdots, p_{K,t})$, where $p_t = \mathbb{E}_{k \sim q_t} \xi_{k,t}$.
- **Observe** loss $\ell_{I_t,t}$. For each arm i, compute $\tilde{\ell}_{i,t}$, using the Inverse Propensity Score trick in Theorem 1 to obtain an unbiased estimator for the loss of arm i:

$$\tilde{\ell}_{i,t} = \frac{\ell_{i,t}}{p_{i,t}} \mathbb{1}_{I_t = i} \qquad i = 1, \cdots, K$$

- **Compute** the estimated loss for each expert, by taking the expected loss over the expert's predictions.

$$\tilde{y}_{k,t} = \mathbb{E}_{i \sim \xi_{k,t}} \tilde{\ell}_{i,t} = \sum_{i=1}^{K} \xi_{k,t}(i) \tilde{\ell}_{i,t} \qquad k = 1, \cdots, |\Pi|$$

- **Compute** the new probability distribution over the experts $q_{t+1} = (q_{1,t+1}, \cdots, q_{N,t+1})$, where

$$q_{j,t+1} = \frac{\exp(-\eta \sum_{s=1}^{t} \tilde{y}_{k,s})}{\sum_{k=1}^{|\Pi|} \exp(-\eta \sum_{s=1}^{t} \tilde{y}_{k,s})}$$

Контекстуальные бандиты

- А ещё, конечно, в контекстуальном бандите можно сделать модель какую-нибудь. Например, LinUCB предполагает, что ожидаемая награда – это линейная функция:

$$r_{a,t} = \mathbf{x}_{a,t}^\top \theta_a^* + \epsilon_t,$$

где θ_a^* — вектор коэффициентов, который нужно оценивать, а $\mathbf{x}_{a,t}$ — вектор признаков контекста.

- И тогда по LinUCB надо выбирать $I_t = \arg\max_a u_{a,t}$, где

$$u_{a,t} = \max_{\theta_a \in C_{a,t-1}} \mathbf{x}_{a,t}^\top \theta_a,$$

а $C_{a,t-1}$ — это доверительное множество, аналог доверительного интервала, которое мы в обычном UCB оценивали.

Контекстуальные бандиты

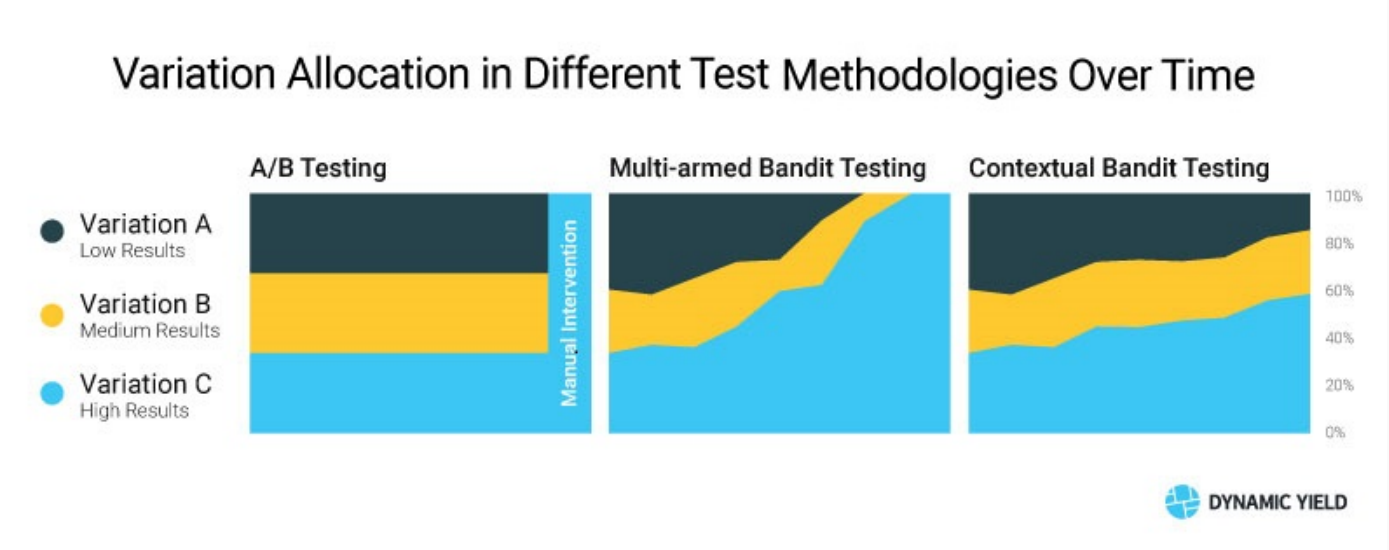
- Итого:

Algorithm 3 LinUCB with Contextual Bandits

Input: $R \in \mathbb{R}^+$, regularization parameter λ
for $t = 1, 2, \ldots, T$ **do**
  Observe feature vectors of all arms $a \in \mathcal{A}_t$: $\mathbf{x}_{a,t} \in \mathbb{R}^d$
  for all $a \in \mathcal{A}_t$ **do**
    if a is new **then**
      $\mathbf{A}_a \leftarrow \lambda \mathbf{I}_d$ (d-dimensional identity matrix)
      $\mathbf{b}_a \leftarrow \mathbf{0}_{d \times 1}$ (d-dimensional zero vector)
    end if
    $\hat{\theta}_a \leftarrow \mathbf{A}_a^{-1} \mathbf{b}_a$
    $$C_{a,t} \leftarrow \left\{ \theta_a^* \in \mathbb{R}^d : \left\| \hat{\theta}_{a,t} - \theta_a^* \right\|_{\mathbf{A}_a} \leq R \sqrt{2 \log \left(\frac{\det(\mathbf{A}_a)^{1/2} \det(\lambda I)^{-1/2}}{\delta} \right)} + \lambda^{1/2} S \right\}$$
    $p_{a,t} \leftarrow \arg\max_{\hat{\theta}_a \in C_{a,t}} \mathbf{x}_{a,t}^T \hat{\theta}_a$
  end for
  Choose arm $a_t = \arg\max_{a \in \mathcal{A}_t} p_{a,t}$ with ties broken arbitrarily, and observe payoff r_t
  $\mathbf{A}_{a_t} \leftarrow \mathbf{A}_{a_t} + \mathbf{x}_{a_t,t} \mathbf{x}_{a_t,t}^T$
  $\mathbf{b}_{a_t} \leftarrow \mathbf{b}_{a_t} + r_t \mathbf{x}_{a_t,t}$
end for

Зачем нужны многорукие бандиты

- Многорукие бандиты используются, например, для A/B тестирования.



- Можно и для оптимизации гиперпараметров в тех же нейронных сетях (или где угодно).
- Контекстуальные – для рекомендаций, для выбора из нескольких вариантов и т.д.
- Но для нас сейчас это первый шаг, упрощённая постановка...

Агенты с несколькими состояниями

И спросила кроха

- Вернёмся теперь к задаче с несколькими состояниями.
- Вознаграждения (rewards) – на каждом шаге: r_t, r_{t+1}, ...
- Но что такое «хорошо» in the long run? Как оценивать поведение алгоритма в приведённом выше сеттинге?
- Если есть естественное конечное число шагов (партия), то это *эпизодическая задача* (episodic task), и логично суммировать вознаграждение по эпизоду (до терминального состояния).
- А что с продолжающимися задачами?

Разные модели

- Ваши версии?

Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие h шагов: $E\left[\sum_{t=0}^{h} r_t\right]$.

Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие h шагов: $E\left[\sum_{t=0}^{h} r_t\right]$.
- Модель *бесконечного горизонта*: хотелось бы учесть все возможные шаги в будущем, т.к. время жизни агента может быть не определено. Но при этом чем раньше получим прибыль, тем лучше. Как это учесть?

Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие h шагов: $E\left[\sum_{t=0}^{h} r_t\right]$.
- Модель *бесконечного горизонта*: хотелось бы учесть все возможные шаги в будущем, т.к. время жизни агента может быть не определено. Но при этом чем раньше получим прибыль, тем лучше. Как это учесть?
- $$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right],$$

  где γ — некоторая константа (discount factor).

Разные модели

- Модель *конечного горизонта*: агент обращает внимание только на следующие h шагов: $E\left[\sum_{t=0}^{h} r_t\right]$.
- Модель *бесконечного горизонта*: хотелось бы учесть все возможные шаги в будущем, т.к. время жизни агента может быть не определено. Но при этом чем раньше получим прибыль, тем лучше. Как это учесть?
- $$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right],$$

  где γ — некоторая константа (discount factor).
- Модель *среднего вознаграждения* (average-reward model):

$$\lim_{h\to\infty} E\left[\frac{1}{h}\sum_{t=0}^{h} r_t\right].$$

Что мы будем использовать

- Все модели разные, приводят к разным результатам.
- Обычно используется модель бесконечного горизонта с некоторым фиксированным discount factor. Её и мы будем использовать.
- Кроме того, её можно обобщить на эпизодические задачи: достаточно просто положить $\gamma = 1$ и добавить одно лишнее состояние с вознаграждением 0, которое будет замкнуто на себя. Так что отныне навсегда

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?

Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?
- Рано или поздно сходится к оптимальному. Это часто можно доказать, но может быть слишком медленно и/или со слишком большими потерями по дороге.

Как оценивать качество обучения?

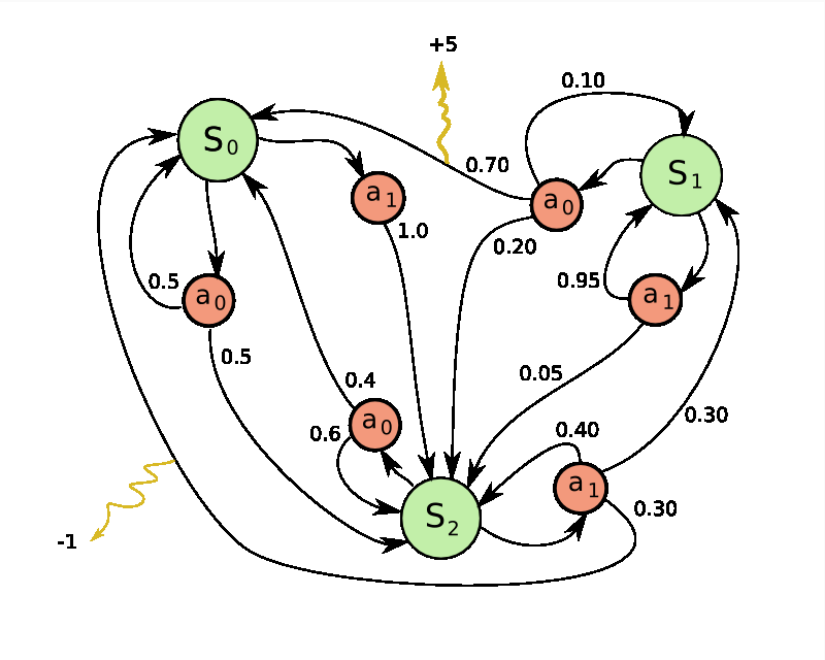
- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?
- Рано или поздно сходится к оптимальному. Это часто можно доказать, но может быть слишком медленно и/или со слишком большими потерями по дороге.
- Сходится с большой скоростью. Два подхода:
  - Скорость сходимости к какой-то фиксированной доле оптимальности. Какой?
  - Насколько хорошо себя ведёт алгоритм после фиксированного времени. Какого?

Как оценивать качество обучения?

- Предыдущие модели могут оценивать готовый обучившийся алгоритм. Как оценивать качество обучения?
- Рано или поздно сходится к оптимальному. Это часто можно доказать, но может быть слишком медленно и/или со слишком большими потерями по дороге.
- Сходится с большой скоростью. Два подхода:
  - Скорость сходимости к какой-то фиксированной доле оптимальности. Какой?
  - Насколько хорошо себя ведёт алгоритм после фиксированного времени. Какого?
- Минимизировать цену (regret), т.е. уменьшение общей суммы выигрыша по сравнению с оптимальной стратегией с самого начала. Это очень хорошая мера, но результаты о ней получить очень сложно.

Марковские процессы

- *Марковский процесс принятия решений* (Markov decision process) состоит из:
  - множества состояний S; множества действий A;
  - функции поощрения $R : S \times A \to \mathbb{R}$; ожидаемое вознаграждение при переходе из s в s' после a – $R^a_{ss'}$;
  - функции перехода между состояниями $p^a_{ss'} : S \times A \to \Pi(S)$, где $\Pi(S)$ — множество распределений вероятностей над S. Вероятность попасть из s в s' после a равна $P^a_{ss'}$.
- Модель *марковская*: переходы не зависят от истории.



Подкрепление и значение состояния

- Главный момент – разница между *reward function* (непосредственное подкрепление) и *value function* (общее подкрепление, ожидаемое, если начать с этого состояния).

- Суть многих методов обучения с подкреплением – в том, чтобы оценивать и оптимизировать value function.

Подкрепление и значение состояния

- Для марковских процессов можно формально определить:

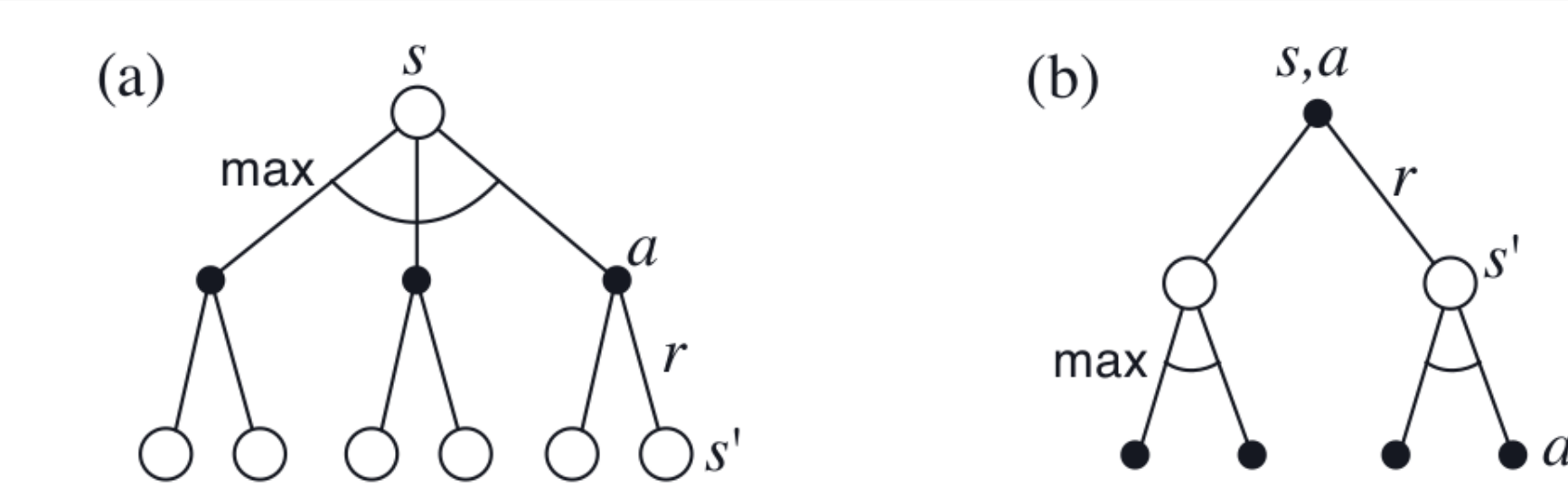
$$V^{\pi}(s) = \mathrm{E}_{\pi}\left[R_t \mid s_t = s\right] = \mathrm{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right].$$

- Можно более детализированно – общее подкрепление, ожидаемое, если начать с состояния s и действия a:

$$Q^{\pi}(s,a) = \mathrm{E}_{\pi}\left[R_t \mid s_t = s, a_t = a\right] = \\ = \mathrm{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right].$$

Подкрепление и значение состояния

- Функции V и Q – это как раз то, что нам нужно оценить; если бы мы их знали, можно было бы просто выбирать то a, которое максимизирует $Q(s, a)$.



Подкрепление и значение состояния

- Для известной стратегии π V^π удовлетворяют уравнениям Беллмана:

$$V^\pi(s) = \mathrm{E}_\pi\left[R_t \mid s_t = s\right] = \mathrm{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right] =$$

$$= \mathrm{E}_\pi\left[r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} \mid s_t = s\right] =$$

$$= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}\left(R^a_{ss'} + \gamma\mathrm{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} \mid s_{t+1} = s'\right]\right) =$$

$$= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}\left(R^a_{ss'} + \gamma V^\pi(s')\right).$$

Основные задачи

- Теоретически всё готово, но у нас много проблем:
  - уравнения знаем, но пока не знаем, как их решать, то есть как найти V^π для данного π?
  - разных стратегий очень, очень много — как найти оптимальную стратегию поведения агента в данной модели и соответствующие V^*?
  - но уравнений тоже не знаем — в реальности обычно P и R не даны, их тоже нужно обучить; как?
  - более того, их обычно даже записать не получится, слишком уж много состояний в любой реальной задаче... что делать?

- Давайте есть слона по частям...

Оптимальные значения состояний

- *Оптимальное значение состояния* — ожидаемая суммарная прибыль, которую получит агент, если начнёт с этого состояния и будет следовать оптимальной стратегии:

$$V^*(s) = \max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right].$$

- Эту функцию можно определить как решение уравнений

$$V^*(s) = \max_{a} \sum_{s' \in S} P^a_{ss'} \left(R^a_{ss'} + \gamma V^*(s')\right),$$

  а затем выбрать оптимальную стратегию

$$\pi^*(s) = \arg\max_a \sum_{s' \in S} P^a_{ss'} \left(R^a_{ss'} + \gamma V^*(s')\right).$$

- Как решать уравнения?

Policy evaluation

- Чтобы посчитать value functions для данной стратегии π, можно просто итеративно пересчитывать по уравнениям Беллмана:

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

  пока не сойдётся.

- Соответственно, для оптимального надо решать уравнения с максимумами:

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

Итеративное решение (по значениям)

- Можно то же самое по Q: пока не сойдётся,

$$Q(s,a) := \sum_{s' \in S} P^a_{ss'} \left(R^a_{ss'} + \gamma \sum_{a'} \pi(s,a') Q(s,a') \right).$$

- А потом просто $V(s) := \max_a Q(s,a)$.
- Оптимальное $Q^*(s,a)$ тоже нехитро:

$$Q^*(s,a) := \sum_{s' \in S} P^a_{ss'} \left(R^a_{ss'} + \gamma \max_{a'} Q^*(s,a') \right).$$

Другой вариант

- Пересчёт в предыдущем алгоритме использует информацию от всех состояний–предшественников. Можно сделать другой вариант:

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

- Он работает, если каждая пара (s,a) встречается бесконечное число раз, s' выбирают из распределения $P^a_{ss'}$, а r сэмплируют со средним $R(s,a)$ и ограниченной дисперсией.
- Но ведь на самом деле нам не V и не Q нужно, а оптимальная стратегия...

Улучшение стратегий

- Мы ищем V^π, чтобы улучшить π. Как улучшить π?
- Естественная идея: давайте жадно выбирать a в s как $\arg\max_a Q^\pi(s, a)$ после вычисления Q^π.
- Policy improvement theorem: для π и π', если для всех s

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s),$$

  то π' не хуже π, т.е. $\forall s\ V^{\pi'}(s) \geq V^\pi(s)$.
- Как доказать?

Улучшение стратегий

- Просто будем разворачивать V^{π}:

$$\begin{aligned} V^{\pi} \leq & Q^{\pi}(s, \pi'(s)) = \mathrm{E}_{\pi'}\left[r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_{t+1} = s\right] \\ \leq & \mathrm{E}_{\pi'}\left[r_{t+1} + \gamma Q^{\pi}(s_{t+1}), \pi'(s_{t+1})) \mid s_{t+1} = s\right] \\ \leq & \ldots \leq \\ \leq & \mathrm{E}_{\pi'}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s_{t+1} = s\right] = V^{\pi'}(s). \end{aligned}$$

Итеративное решение (по стратегиям)

- Ищем оптимальную стратегию итеративным алгоритмом.
- `PolicyIteration` – инициализировать π, потом, пока $\pi \neq \pi'$, повторять:
  - вычислить значения состояний для стратегии π, решив систему линейных уравнений

$$V^{\pi}(s) := \sum_{a} \pi(s,a) \sum_{s' \in S} P^{a}_{ss'} \left(R^{a}_{ss'} + \gamma V^{\pi}(s') \right);$$

  - улучшить стратегию на каждом состоянии:

$$\pi'(s) := \arg\max_{a} Q^{\pi}(s,a) = \arg\max_{a} P^{a}_{ss'} \left(R^{a}_{ss'} + \gamma V^{\pi}(s') \right);$$

- Почему оно сходится?

Итеративное решение (по стратегиям)

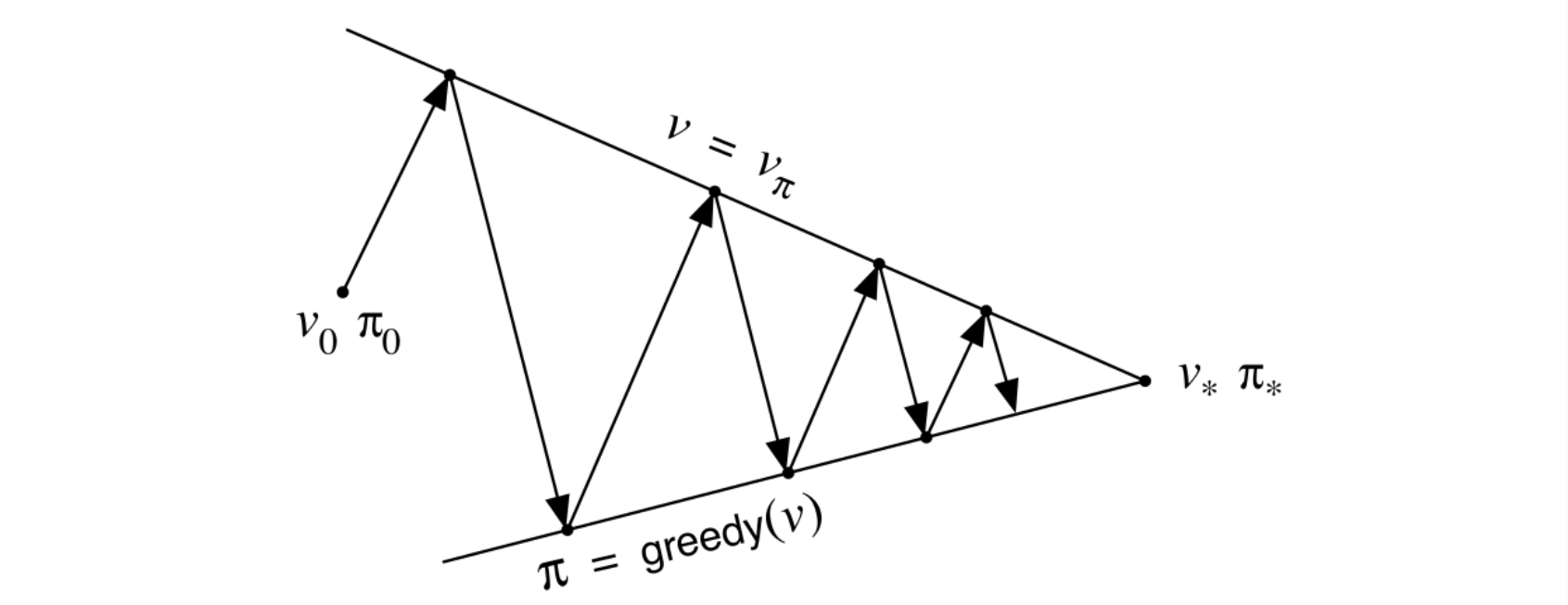
- Сходится, т.к. на каждом шаге строго улучшаем целевую функцию, а всего существует конечное число ($|A|^{|S|}$) стратегий.
- Но, конечно, это медленно, надо V^π пересчитывать; проще делать на каждой итерации ровно один шаг пересчёта V^π, а потом сразу выбирать жадную стратегию:

$$V_{k+1}(s) := \max_a \sum_{s' \in S} P^a_{ss'} \left(R^a_{ss'} + \gamma V_k(s') \right).$$

- Это называется value iteration.

Итеративное решение (по стратегиям)

- Есть другие похожие методы – их всех объединяет подход, основанный по сути на чём-то вроде EM-алгоритма с динамическим программированием.



- Это может быть достаточно эффективно даже для больших задач (с трюками, позволяющими не всё пространство исследовать).

Основные задачи

- Теоретически всё готово, но у нас много проблем:
  - уравнения знаем, но пока не знаем, как их решать, то есть как найти V^π для данного π?
  - разных стратегий очень, очень много — как найти оптимальную стратегию поведения агента в данной модели и соответствующие V^*?
  - но уравнений тоже не знаем — в реальности обычно P и R не даны, их тоже нужно обучить; как?
  - более того, их обычно даже записать не получится, слишком уж много состояний в любой реальной задаче... что делать?

- Давайте есть слона по частям...

Спасибо за внимание!