ОБОБЩЕНИЯ TD И ПЛАНИРОВАНИЕ

Сергей Николенко

Академия MADE — Mail.Ru

02 октября 2021 г.

Random facts:

- 2 октября 1187 г. Саладин после недолгой осады захватил Иерусалим, что послужило поводом к началу Третьего крестового похода
- 2 октября 1552 г. Иван Грозный вошёл в Казань и присоединил Казанское ханство к России
- 2 октября 1607 г. голландский очковый мастер Иоанн Липперсгей продемонстрировал в Гааге прототип оптического телескопа; Галилей направил его в небо только в 1609 году
- 2 октября 1836 г. Чарльз Дарвин вернулся из длившегося почти пять лет кругосветного путешествия на корабле «Бигль»
- 2 октября 1928 г. святой Хосемария Эскрива де Балагер создал Прелатуру Святого Креста Opus Dei, позже прославленную Дэном Брауном

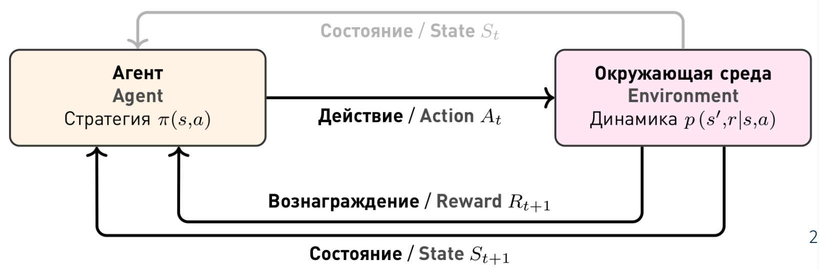
Обучение с подкреплением

- В прошлый раз мы ввели основные понятия динамики марковских процессов принятия решений:
    - собственно динамику процесса:

$$p\left(s', r \mid s, a\right) = p\left(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a \,|;\right)$$

    - награды за каждый эпизод, начиная со времени t:

$$G_t = R_{t+1} + \gamma G_{t+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1};$$



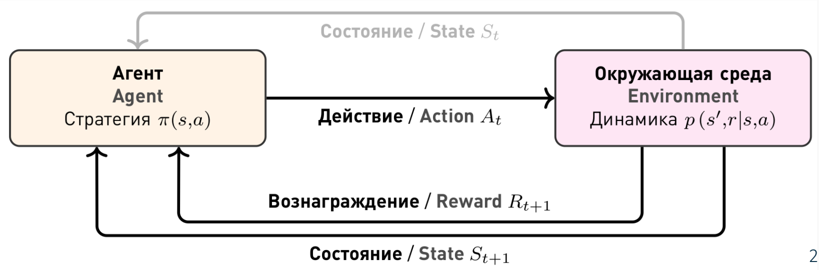
Обучение с подкреплением

- Определили функции значений V и Q

$$V_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right],$$

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

- Выписали уравнения Беллмана и научились их решать.



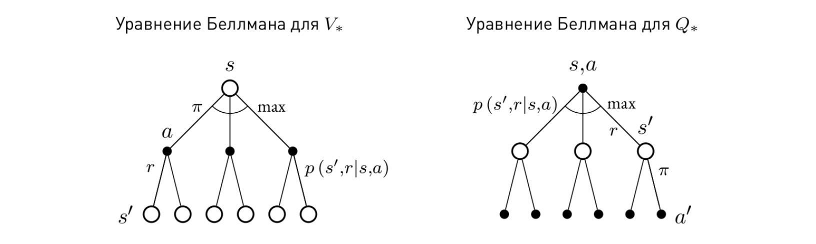
Основные задачи

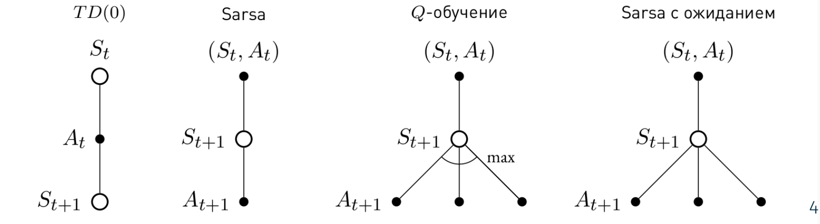
- Теоретически всё готово, но у нас много проблем:
    - уравнения знаем, но пока не знаем, как их решать, то есть как найти V^π для данного π?
    - разных стратегий очень, очень много — как найти оптимальную стратегию поведения агента в данной модели и соответствующие V^*?
    - но уравнений тоже не 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...

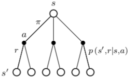
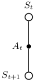
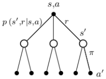
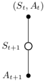
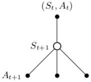
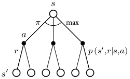
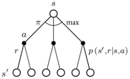
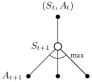
Диаграммы обхода

- Вспомним, о чём мы говорили, и заодно введём новый вид картинок: *диаграммы обхода* (backup diagrams)

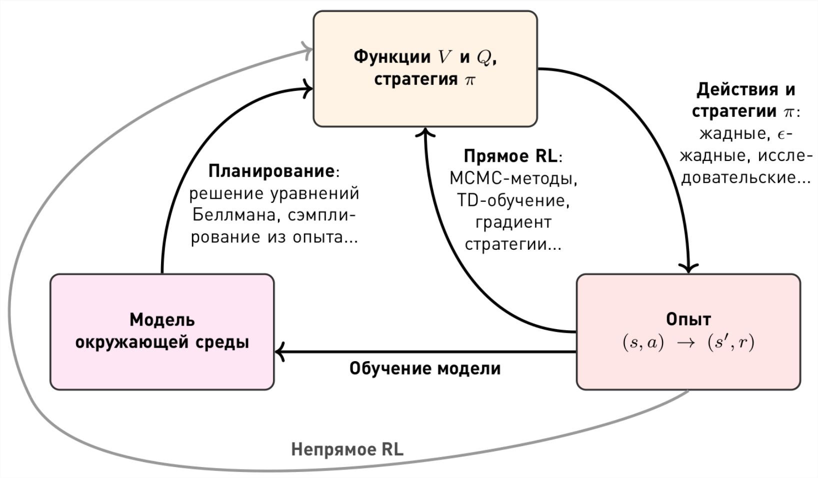




Таблица

	Функция	Обновление в ожидании	Обновление по выборке
Оценка стратегии π	$V_\pi(S_t) :=$	**Уравнение Беллмана** $\sum_{a\in\mathcal{A}} \pi(a\|S_t) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s',r\|s,a)\,(r+\gamma V_\pi(s'))$ 	**TD**(0) $V(S_t)+\alpha\,(R_{t+1}+\gamma V(S_{t+1})-V(S_t))$ 
	$Q_\pi(S_t,A_t) :=$	**Уравнение Беллмана** $\sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s',r\|S_t,A_t)\,(r+\gamma \sum_{a'\in\mathcal{A}} \pi(a'\|s')\,Q_\pi(s',a'))$ 	**Sarsa, Sarsa с ожиданием** $Q(S_t,A_t)+\alpha\,(R_{t+1}+\gamma Q(S_{t+1},A_{t+1})-Q(S_t,A_t))$ $Q(S_t,A_t)+\alpha\left(R_{t+1}+\gamma\sum_a \pi(a\|S_{t+1})\,Q(S_{t+1},a)-Q(S_t,A_t)\right)$ 
Управление, поиск π_*	$V_*(S_t) :=$	**Уравнение Беллмана** $\max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s',r\|S_t,a)\,(r+\gamma V_*(s'))$ 	
	$Q_*(S_t,A_t) :=$	**Уравнение Беллмана** $\sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s',r\|S_t,A_t)\left(r+\gamma \max_{a'} Q_*(s',a')\right)$ 	**Q-обучение** $Q(S_t,A_t)+\alpha\,(R_{t+1}+\gamma\max_a Q(S_{t+1},a)-Q(S_t,A_t))$ 

Обучение с подкреплением



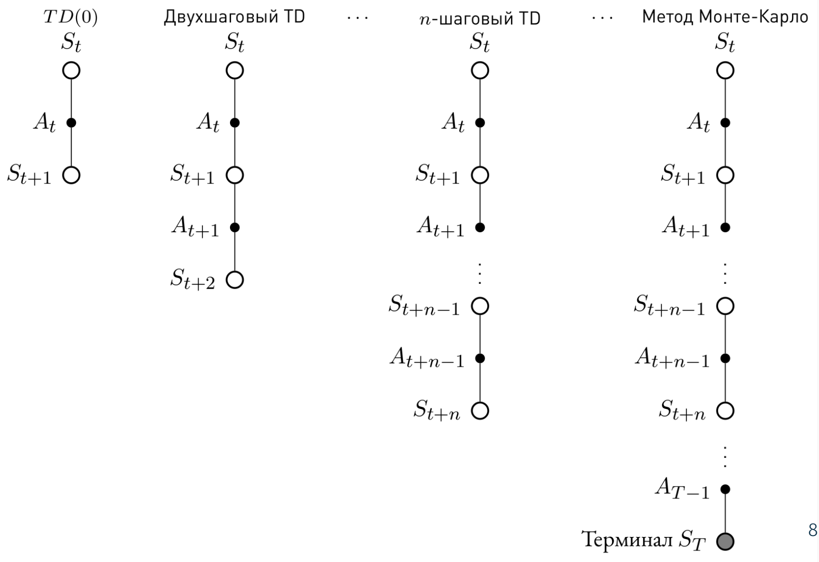
Обобщения и дополнительные замечания

- Мы говорили о TD-алгоритмах, которые оценивают $G_t = R_{t+1} + \gamma V_t(S_{t+1})$.
- Но можно же и дальше развернуть:

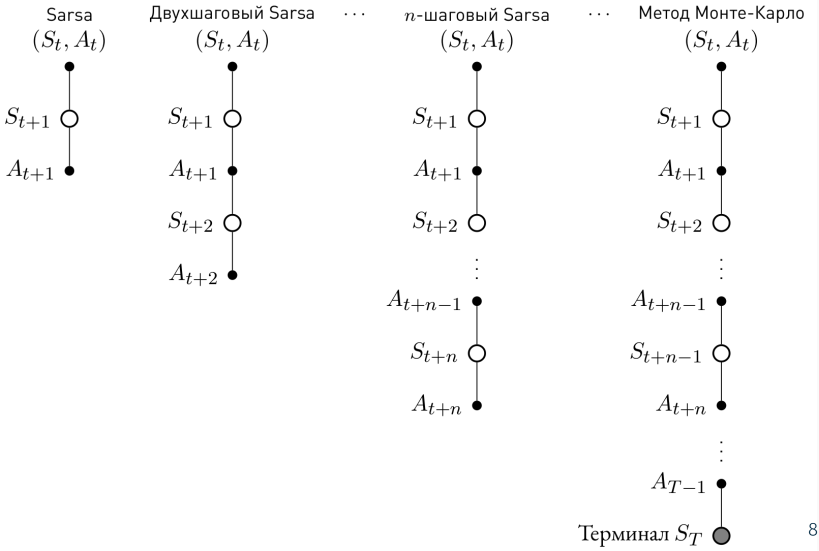
$$
\begin{aligned}
G_{t:t+2} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+1}) \\
G_{t:t+3} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V_{t+2}(S_{t+3}) \\
\ldots &= \ldots \\
G_{t:t+n} &= R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})
\end{aligned}
$$

- На бесконечности это сливается с методами Монте-Карло, потому что там мы обновляем на основе награды за весь эпизод, от конца к началу

- Обобщение TD-обучения: n-шаговые методы



- Обобщение TD-управления: n-шаговые методы



- И теперь появляются n-шаговые варианты всех алгоритмов.

Алгоритм n-step Sarsa (on-policy TD control):

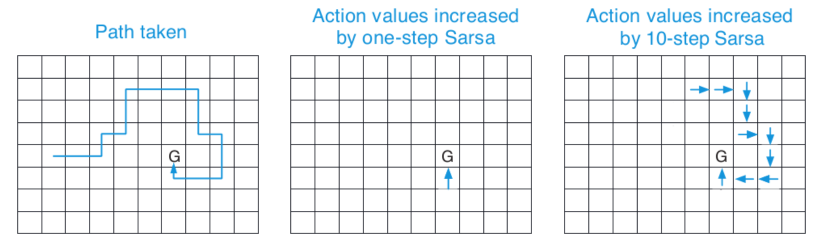
инициализировать случайно $Q(s,a)$; повторять до сходимости:

- инициализировать S_0, выбрать A_0 по стратегии, полученной из Q (например, по ϵ-жадной стратегии); $T := \infty$;
- для каждого шага в эпизоде $t = 0, \dots, T$:
    - если $t < T$, то:
        - сделать действие A_t, получить R_{t+1}, S_{t+1};
        - если это терминальное состояние, то $T := t+1$, а если нет, выбрать A_{t+1} в состоянии S_{t+1} по стратегии π;
    - $\tau := t - n + 1$ (мы будем обновлять оценку на шаге τ)
    - если $\tau \geq 0$, то обновляем:
        - $G := \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
        - если $\tau + n < T$, то $G := G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$;
        - $Q(S_\tau, A_\tau) := Q(S_\tau, A_\tau) + \alpha \left(G - Q(S_\tau, A_\tau)\right)$
        - если обучаем π, то поменять π на ϵ-жадную по Q

- Та же Sarsa, но дальше заглядывает.

n-шаговые алгоритмы

- Смысл здесь в том, что n-step алгоритмы обновляют сразу много значений по одной и той же награде, даже если всё остальное пока что нулевое:



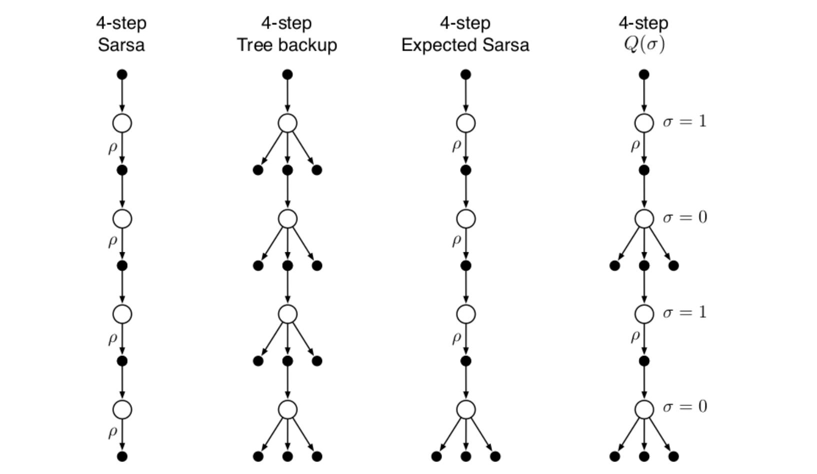
- Всё то же самое можно переписать в терминах просто изменения TD-ошибки:

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n,T)-1} \gamma^{k-t} \left(R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k) \right)$$

- Если хочется сделать off-policy, то можно сделать, конечно, с importance sampling, как мы обсуждали.
- А можно сделать поиск по дереву: идём обратно по дереву от (S_t, A_t) к концу эпизода; в каждый момент у нас есть одно действие, которое реально произошло, а для других берём bootstrap-оценки:

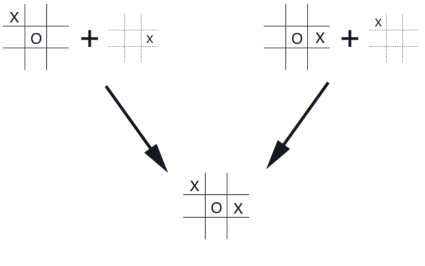
$$
\begin{array}{rclll}
G_{t:t+1} & = & R_{t+1} & +\gamma \sum_a \pi\left(a|S_{t+1}\right) Q_t(S_{t+1}, a), & \\
G_{t:t+2} & = & R_{t+1} & +\gamma \sum_{a \neq A_{t+1}} \pi\left(a|S_{t+1}\right) Q_{t+1}(S_{t+1}, a) & \\
 & & & +\gamma\pi\left(A_{t+1}|S_{t+1}\right) \left(R_{t+2} + \gamma \sum_a \pi\left(a|S_{t+2}\right) Q_{t+1}(S_{t+2}, a)\right), & \\
\ldots & & \ldots & & \\
G_{t:t+n} & = & R_{t+1} & +\gamma \sum_{a \neq A_{t+1}} \pi\left(a|S_{t+1}\right) Q_{t+1}(S_{t+1}, a) & \\
 & & & +\gamma\pi\left(A_{t+1}|S_{t+1}\right) G_{t+1:t+n}. &
\end{array}
$$

- А можно пытаться нечто среднее сделать, выбирая то сэмплы, как в Sarsa, то апдейт по дереву, как выше:



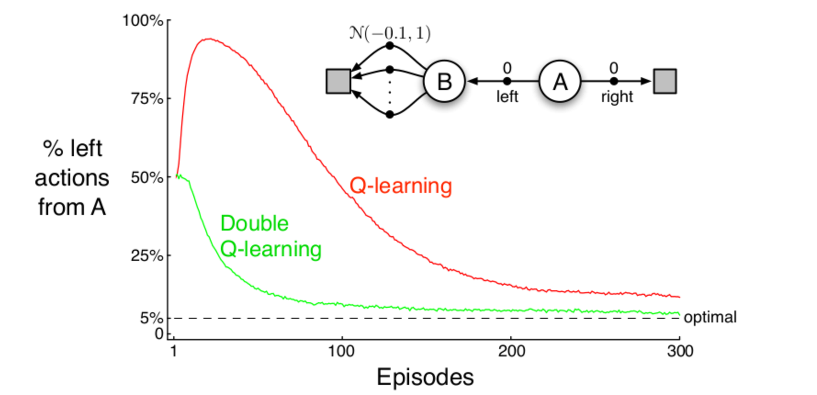
Afterstates

- Часто обучают не по состояниям $V(s)$ и не по парам $Q(s, a)$, а по т.н. *afterstates* – состояниям после действия.
- Это хорошо, когда действия имеют немедленный эффект, а случайный процесс происходит уже потом.
- Крестики-нолики – многие пары приводят к одной и той же позиции.



Двойное обучение

- В TD-обучении есть проблема: наша целевая стратегия – это жадная стратегия по текущей Q, т.е. мы используем максимум по оценкам, чтобы оценить максимальное значение, а это вносит смещение в оценки.



Двойное обучение

- Чтобы это поправить, можно использовать двойное обучение:
  - поддерживаем две стратегии Q_1 и Q_2;
  - каждый раз бросаем монетку, выбираем, какую брать стратегию для обновления, а какую для цели:

$$Q_1(S, A) := Q_1(S, A) + \alpha \left(R + \gamma \max_a Q_2(S', a) - Q_1(S, A) \right)$$

или

$$Q_2(S, A) := Q_2(S, A) + \alpha \left(R + \gamma \max_a Q_1(S', a) - Q_2(S, A) \right).$$

- (van Hasselt, 2010): у Q-обучения основное уравнение апдейта

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha_t \left(R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q(S_t, A_t) \right)$$

- И по умолчанию Q-обучение использует максимум из имеющихся оценок $Q_t(S_{t+1}, a)$.
- Т.е. получается, что мы оцениваем максимум из нескольких ожиданий, $\max_i \mathbb{E}\left[X_i\right]$, через максимум из оценок каждого их них.
- А на самом деле это будет несмещённая оценка для ожидания максимума $\mathbb{E}\left[\max_i X_i\right]$!

- Поэтому ван Хассельт предложил использовать Double Q-learning: давайте возьмём два разных независимых набора сэмплов для X_i, и будем использовать один из них в тот момент, когда делаем апдейт Q-обучения для другого.
- Тогда (при независимых оценках) всё будет сходиться.

Algorithm 1 Double Q-learning

```
1:  Initialize Q^A, Q^B, s
2:  repeat
3:      Choose a, based on Q^A(s,·) and Q^B(s,·), observe r, s'
4:      Choose (e.g. random) either UPDATE(A) or UPDATE(B)
5:      if UPDATE(A) then
6:          Define a* = arg max_a Q^A(s', a)
7:          Q^A(s,a) ← Q^A(s,a) + α(s,a)(r + γQ^B(s', a*) − Q^A(s,a))
8:      else if UPDATE(B) then
9:          Define b* = arg max_a Q^B(s', a)
10:         Q^B(s,a) ← Q^B(s,a) + α(s,a)(r + γQ^A(s', b*) − Q^B(s,a))
11:     end if
12:     s ← s'
13: until end
```

Планирование

Планирование

- В прошлый раз мы ввели основные понятия динамики марковских процессов принятия решений:
    - собственно динамику процесса:

$$p\left(s', r \mid s, a\right) = p\left(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right);$$

    - награды за каждый эпизод, начиная со времени t:

$$G_t = R_{t+1} + \gamma G_{t+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1};$$

    - функцию значения для состояний и пар состояние-действие:

$$V_\pi(s) = \mathbb{E}_\pi\left[G_t \mid S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right],$$

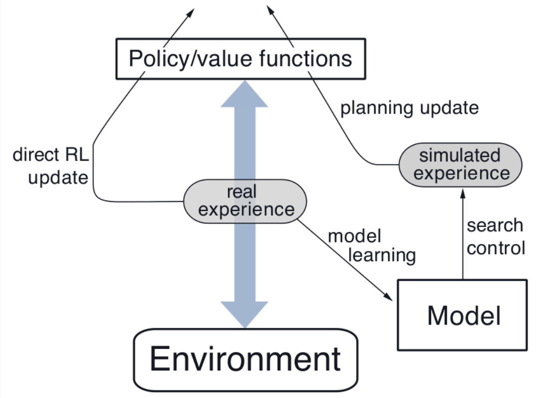
$$Q_\pi(s, a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

    и их оптимальные варианты V_*, Q_*;

Планирование

- Мы выписывали уравнения Беллмана на V и Q и научились их решать.
- Кроме того, методами Монте-Карло мы умеем обучать модель окружающей среды.
- А TD-обучением мы можем обучать одновременно и модель, и оптимальную стратегию.
- *Планирование* (planning) — это то, как использовать модель, чтобы лучше обучать V и Q.
- Как нам это сделать?..

- Базовая идея: давайте сэмплировать новый опыт из модели окружающей среды, использовать симуляции.
- Пример – архитектура Dyna:



- В простейшем случае можно делать ещё несколько обновлений, скажем, Q-обучения, исходя из модели.

Алгоритм Dyna-Q:

- инициализировать случайно π, $Q(s, a)$, $M(s, a)$ (модель);
- повторять (цикл внутри эпизода, эпизоды тоже повторять):
  - для состояния S выбрать A по ϵ-жадной стратегии из Q;
  - сделать действие A, пронаблюдать R и S';
  - $Q(S, A) := Q(S, A) + \alpha\left(R + \gamma \max_a Q(S', a) - Q(S, A)\right)$;
  - добавить R, S' в $M(S, A)$;
  - повторить k раз:
    - (S, A) — случайная пара, которую уже наблюдали раньше (т.е. для неё есть $M(S, A)$);
    - породить R, S' по $M(S, A)$;
    - $Q(S, A) := Q(S, A) + \alpha\left(R + \gamma \max_a Q(S', a) - Q(S, A)\right)$.

- Dyna-Q случайно обновляет k значений $Q(s, a)$ по имеющейся модели; k можно выбирать из соображений имеющихся ресурсов.

- Конечно, ещё лучше будет обновлять не случайно.

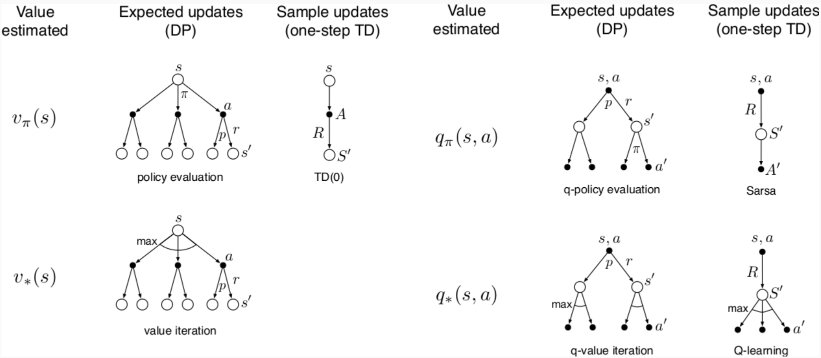
Алгоритм обхода по приоритетам (prioritized sweeping):

- инициализировать π, $Q(s, a)$, $M(s, a)$ (модель), PQ (очередь);
- повторять (цикл внутри эпизода, эпизоды тоже повторять):
  - для состояния S выбрать A по ϵ-жадной стратегии из Q;
  - сделать A, пронаблюдать R и S', добавить R, S' в $M(S, A)$;
  - $P := |R + \gamma \max_a Q(S', a) - Q(S, A)|$; если $P > \theta$ (порог), добавить (S, A) в PQ с приоритетом P;
  - повторить k раз, пока $PQ \neq \emptyset$:
    - взять $(S, A) := \mathrm{Head}(PQ)$, породить R, S' по $M(S, A)$;
    - $Q(S, A) := Q(S, A) + \alpha \left(R + \gamma \max_a Q(S', a) - Q(S, A)\right)$.
    - для каждой пары $(\tilde{S}, \tilde{A})$, из которой модель попадает в S:
      * $\tilde{R}$ – предсказанная моделью награда для $(\tilde{S}, \tilde{A}, S)$;
      * $P := |\tilde{R} + \gamma \max_a Q(S, a) - Q(\tilde{S}, \tilde{A})|$ (приоритет);
      * если $P > \theta$, то добавить $(\tilde{S}, \tilde{A})$ в PQ с приоритетом P.

- Это для детерминированного окружения, для случайного можно взвесить оценками вероятностей попасть в S.

Expected vs. sample updates

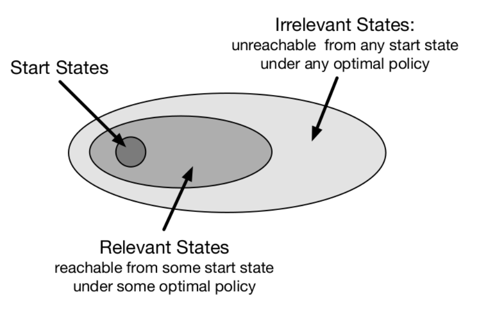
- Вот ещё один взгляд на то, чем мы занимались – expected updates vs. sample updates:



- Expected update проходит по всем следующим состояниям.
- Sample update выбирает одно (случайно или из опыта).
- Expected, конечно, лучше, но и дороже, в целом sample даже выгоднее может получиться.

Expected vs. sample updates

- Как лучше распределить апдейты (т.е. фактически имеющийся вычислительный ресурс)?
- Trajectory sampling: лучше сэмплировать сразу траекториями по реальной стратегии. Так мы не будем тратить время на недостижимые или очень-редко-достижимые состояния.
- Например, RTDP (real-time dynamic programming): обновляем по уравнениям Беллмана, но только состояния, которые реально встречались в траекториях.

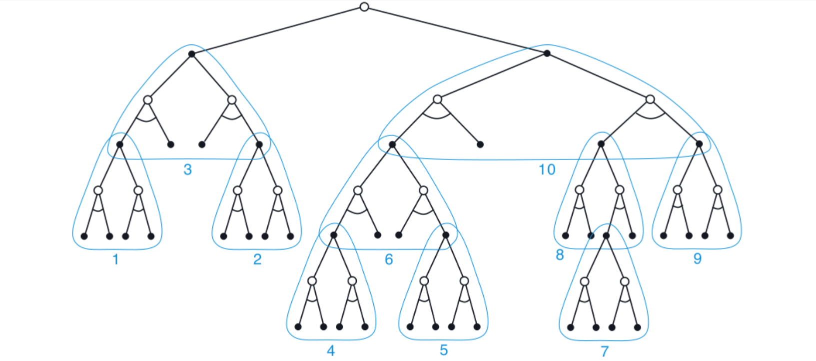


Планирование в текущем моменте

- А можно использовать планирование прямо при выборе действия, т.е. фактически как часть стратегии (decision-time planning).
- Простейшее такое планирование мы знаем: когда мы обучали $V(s)$, а не $Q(s, a)$, потом для получения стратегии π по функции V нужно было перебрать возможные следующие состояния и выбрать лучшее.
- Как это обобщить и улучшить?..

Планирование в текущем моменте

- Правильно, это поиск на несколько «ходов» в глубину! Точнее говоря, поиск получается в ширину. :)



- Heuristic search: делаем поиск в ширину на несколько ходов вперёд, выбирая наилучшее действие (кстати, почему это называется «минимакс», если тут только $\max$?)

Планирование в текущем моменте

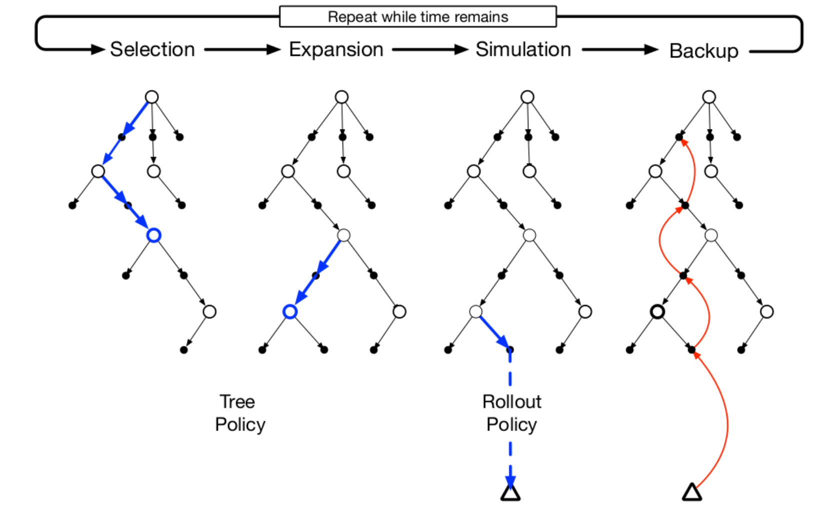
- Другой способ реализовать такое планирование: rollouts.
- Начиная с текущего состояния, симулируем несколько траекторий по текущей стратегии, а потом оцениваем действия, усредняя результаты этих симуляций.
- Когда оценки становятся достаточно точными, можно сделать ход, а потом опять строить rollouts из следующего состояния.
- Это пробовал ещё Тезауро для нард, и получалось очень хорошо, прямо неожиданно хорошо.
- Но ещё лучше совместить одно и другое и получить...

Планирование в текущем моменте

- ...Monte Carlo tree search (MCTS)! Это один из ключевых компонентов, например, в прогрессе алгоритмов для го между 2005 (слабый любитель) до 2015 (6 дан), а потом к AlphaGo, а потом и к AlphaZero — там везде есть MCTS.
- Мы строим дерево, в котором оцениваем листья через rollouts и выбираем, когда раскрыть лист дальше, т.е. итеративно:
    - выбираем лист дерева, действие в нём и следующее за ним состояние;
    - раскрываем всевозможные действия в этом состоянии;
    - делаем rollouts исходя из этих действий, используя их результаты для обновления оценок действий на пути к корню дерева.

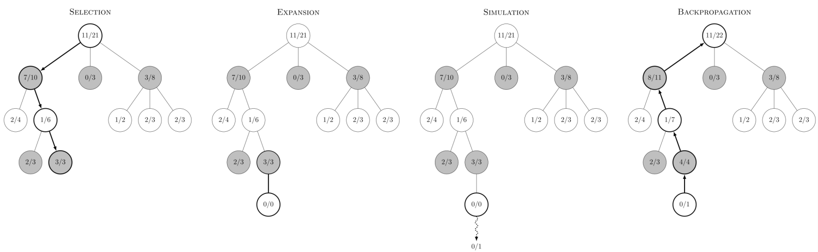
Планирование в текущем моменте

- Вот такая картинка получается:



Планирование в текущем моменте

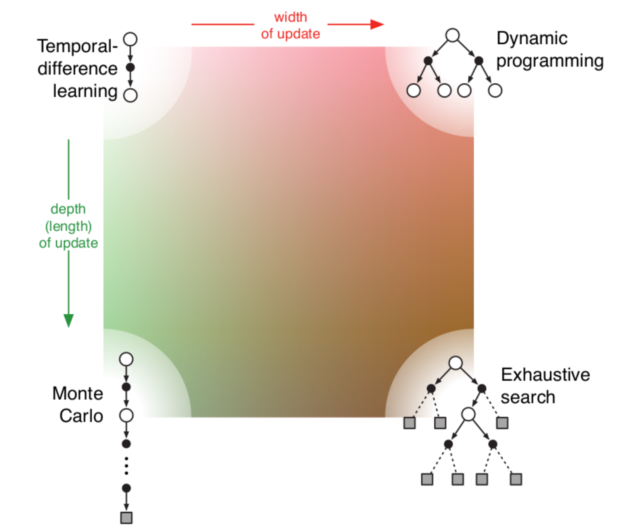
- Выбирать листья и действия нужно по статистике побед/поражений:



- Например, по методу UCT (upper confidence bounds applied to trees): выбираем для rollout действие, максимизирующее

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}.$$

- И ещё один взгляд-иллюстрация:



Спасибо!

Спасибо за внимание!