ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ

Сергей Николенко

СПбГУ — Санкт-Петербург

02 мая 2024 г.

Random facts:

- 2 мая 1563 г. Иван Фёдоров начал работу над «Апостолом», первой датированной русской печатной книгой, а 2 мая 1611 г. была впервые опубликована санкционированная в Англии версия Библии, «Библия короля Якова» (KJV, King James Version)
- 2 мая 1922 г. сочетались браком Сергей Есенин и Айседора Дункан
- 2 мая 1949 г. прошел первый в СССР телерепортаж футбольного матча со стадиона «Динамо»
- 2 мая 1964 г. британский хит-парад впервые возглавили The Beatles с песней «From Me to You», начав добрую и очень долгую традицию
- 2 мая 1999 г. на Эвересте нашли тело английского альпиниста Джорджа Ли Мэллори; в 1924 году он (вместе с Эндрю Ирвайном) пропал во время метели; местоположение тела Мэллори говорит, что с большой вероятностью их восхождение увенчалось успехом
- 2 мая 2012 г. пастельная версия «Крика» была продана на аукционе за $120 миллионов

Методы Монте-Карло в RL

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...

Метод Монте-Карло

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

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

  - 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_*;

Метод Монте-Карло

- В прошлый раз мы выписали уравнения Беллмана на V и Q и научились их решать.
- Теперь будем обучать одновременно и модель, и оптимальную стратегию; вознаграждения и переходы не даны.
- Начнём со стохастических алгоритмов (метода Монте-Карло); но начнём опять с простой задачи.
- Как обучить вознаграждения $V^\pi(s)$, ожидаемые от состояния s в эпизодической задаче?

Метод Монте-Карло

- Да очень просто: будем накапливать данные и усреднять.

Алгоритм Monte Carlo estimation:

- инициализировать случайно π и $V(s)$, пустые списки $\mathrm{Ret}(s)$;
- повторять до сходимости:
  - сгенерировать эпизод $S_0, A_0, R_1, S_1, A_1, \dots, S_T$ по стратегии π;
  - $G := 0$
  - для каждого $t = T-1, T-2, \dots, 0$:
    - $G := \gamma G + R_{t+1}$
    - если надо, то добавить G в $\mathrm{Ret}(S_t)$ и обновить $V(S_t) := \mathrm{Avg}(\mathrm{Ret}(S_t))$.

- «Если надо» скрывает тонкую разницу между first-visit и every-visit Monte Carlo.
- На выходе этот алгоритм выдаст V_π для данной π, которой порождаются эпизоды.

Метод Монте-Карло

- Но вообще без модели гораздо удобнее оценивать Q_π. Сразу можно и стратегию обновлять, тот же policy iteration.

Алгоритм Monte Carlo control with exploring starts:

- инициализировать случайно π и $Q(s)$, пустые списки $\mathrm{Ret}(s)$;
- повторять до сходимости:
  - выбрать S_0, A_0 случайно так, чтобы $\forall\ (s,a) \quad p(s,a) > 0$;
  - сгенерировать эпизод $S_0, A_0, R_1, S_1, A_1, \dots, S_T$ по стратегии π;
  - $G := 0$
  - для каждого $t = T-1, T-2, \dots, 0$:
    - $G := \gamma G + R_{t+1}$
    - если надо, то добавить G в $\mathrm{Ret}(S_t, A_t)$ и обновить:

$$Q(S_t, A_t) := \mathrm{Avg}(\mathrm{Ret}(S_t, A_t)),$$
$$\pi(S_t) := \arg\max_a Q(S_t, a).$$

- Этот алгоритм выдаст π_* и соответствующую ей функцию Q_*.
- Здесь важно предположение exploring starts, без него мы не исследуем все действия.

Метод Монте-Карло

- А что делать, если оно не выполняется? Придётся исследовать самостоятельно.

Алгоритм on-policy Monte Carlo control с мягкими стратегиями:

- инициализировать ϵ-мягкую π и $Q(s)$, пустые $\mathrm{Ret}(s)$;
- повторять до сходимости:
  - выбрать S_0, A_0 случайно так, чтобы $\forall\ (s,a) \quad p(s,a) > 0$;
  - сгенерировать эпизод $S_0, A_0, R_1, S_1, A_1, \ldots, S_T$ по стратегии π;
  - $G := 0$
  - для каждого $t = T-1, T-2, \ldots, 0$:
    - $G := \gamma G + R_{t+1}$
    - если надо, то добавить G в $\mathrm{Ret}(S_t, A_t)$ и обновить:

$$Q(S_t, A_t) := \mathrm{Avg}(\mathrm{Ret}(S_t, A_t)),$$

$$a_* := \arg\max_a Q(S_t, a),$$

$$\pi(a \mid S_t) := \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(S_t)|}, & \text{если } a = a_*, \\ \frac{\epsilon}{|A(S_t)|}, & \text{если } a \neq a_*. \end{cases}$$

- Этот алгоритм умеет искать оптимальную *мягкую* π.

Метод Монте-Карло

- На самом деле для ϵ-мягких стратегий тоже верен аналог policy improvement теоремы, и для ϵ-мягких стратегий метод policy iteration тоже вполне работает.
- Но это on-policy алгоритм, он найдёт мягкую стратегию, а в реальности шахматист, который играет как Магнус Карлсен 90% ходов, а 10% ходов делает случайно, вряд ли продвинется сильно дальше третьего разряда.
- Но ведь и исследовать тоже нужно! Хорошо было бы научиться исследовать по одной стратегии, а оценивать другую...

Метод Монте-Карло

- ...и такой трюк действительно можно сделать!
- Вспомним сэмплирование со значимостями (importance sampling): если мы умеем брать сэмплы по распределению $q(\mathbf{x})$, а оценивать хотим ожидание по распределению $p(\mathbf{x})$, то можно сделать так:

$$\mathbb{E}_{p(\mathbf{x})}\left[f(\mathbf{x})\right] = \int f(\mathbf{x})p(\mathbf{x})\mathrm{d}\mathbf{x} = \int f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})\mathrm{d}\mathbf{x} = \mathbb{E}_{q(\mathbf{x})}\left[f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}\right].$$

- А у нас в качестве p и q выступают распределения на траекториях:

$$\begin{aligned} p\left(\mathrm{Traj}|\pi\right) &= p\left(A_t, S_{t+1}, A_{t+1}, \dots, S_{T-1}, A_{T-1}, S_T|\pi\right) = \\ &= \pi\left(A_t|S_t\right) p\left(S_{t+1}|S_t, A_t\right) \dots \pi\left(A_{T-1}|S_{T-1}\right) p\left(S_T|S_{T-1}, A_{T-1}\right) = \\ &= \prod_{k=t}^{T-1} \pi\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right). \end{aligned}$$

Метод Монте-Карло

- И получается, что для двух стратегий π, b (от слова behaviour) определить веса

$$\rho_{t:T-1}^{\pi,b} = \frac{p\left(\text{Traj}|\pi\right)}{p\left(\text{Traj}|b\right)} = \frac{\prod_{k=t}^{T-1} \pi\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right)}{\prod_{k=t}^{T-1} b(A_k \mid S_k) p\left(S_{k+1}|S_k, A_k\right)} = \\ = \frac{\prod_{k=t}^{T-1} \pi\left(A_k|S_k\right)}{\prod_{k=t}^{T-1} b(A_k \mid S_k)},$$

  то неизвестные вероятности сократятся и останется, что когда мы порождаем эпизоды по b, нужно просто усреднять не G_t, а $\rho_{t:T-1}^{\pi,b} G_t$, и будут получаться оценки π!
- Единственное условие – покрытие (coverage): должно быть верно, что если $\pi\left(a|s\right) > 0$, то и $b(a \mid s) > 0$.

Метод Монте-Карло

- Ещё есть тонкая разница между обычным importance sampling и взвешенным (weighted importance sampling):
  - в обычном мы берём оценку среднего через сэмплы

$$V(s) = \frac{1}{N} \sum_{t=1}^{N} \rho_{t:T-1}^{\pi,b} G_t,$$

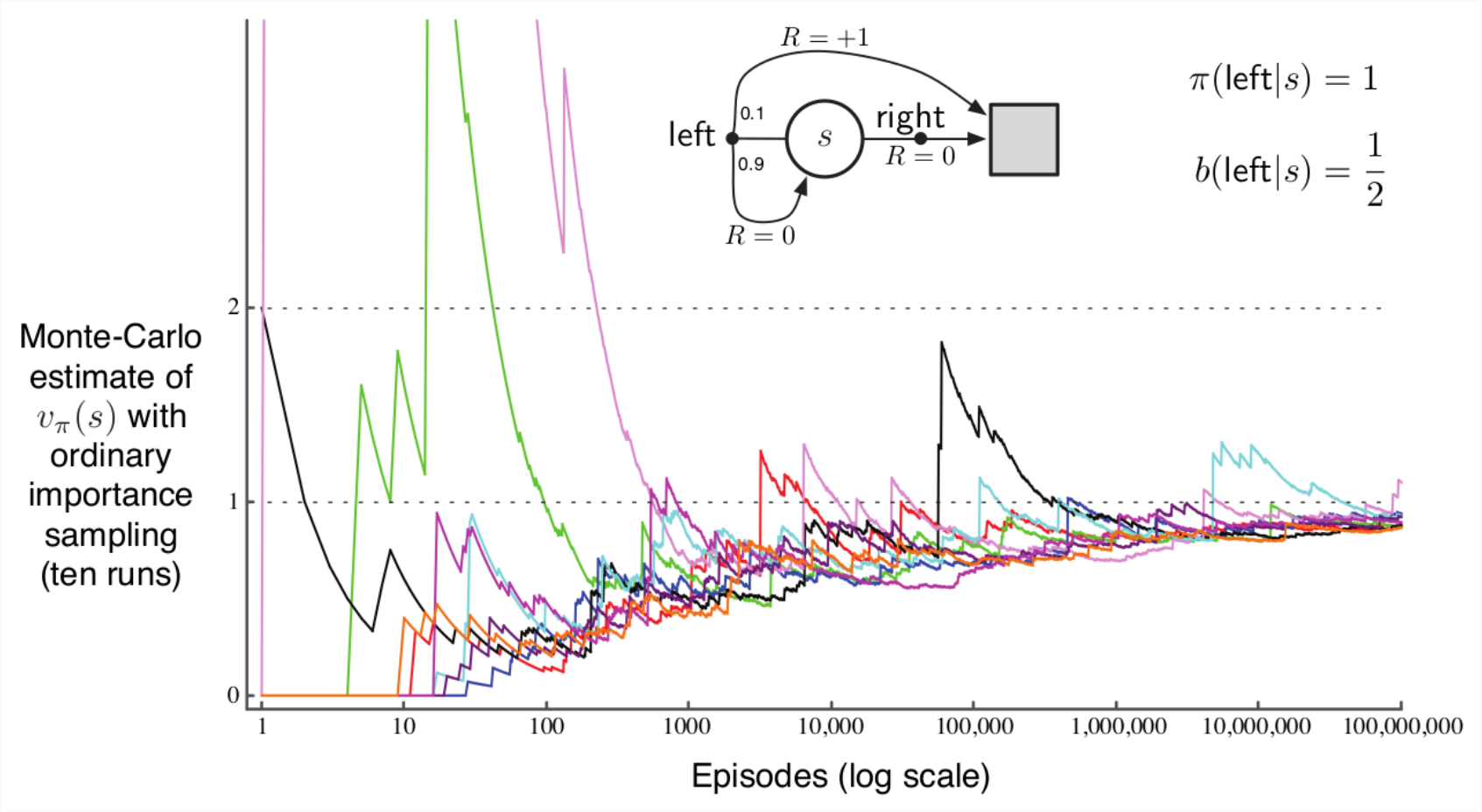
  - а во взвешенном ещё нормируем суммой весов

$$V(s) = \frac{\sum_{t=1}^{N} \rho_{t:T-1}^{\pi,b} G_t}{\sum_{t=1}^{N} \rho_{t:T-1}^{\pi,b}}.$$

- Вторая оценка смещённая, но сходится куда надо и у неё нормальная человеческая дисперсия.

Метод Монте-Карло

- А у первого варианта дисперсия очень большая, и может быть даже бесконечная! Пример:



Метод Монте-Карло

- Итого вот какой алгоритм получается

Алгоритм on-policy Monte Carlo control с мягкими стратегиями:

- инициализировать ϵ-мягкую b, π, $Q(s)$, пустые $\mathrm{Ret}(s)$ и $c(s)$;
- повторять до сходимости:
  - сгенерировать эпизод $S_0, A_0, R_1, S_1, A_1, \dots, S_T$ по мягкой стратегии b;
  - $G := 0$, $W := 1$
  - для каждого $t = T-1, T-2, \dots, 0$:
    - $G := \gamma G + R_{t+1}$
    - $c(S_t, A_t) := c(S_t, A_t) + W$
    - $Q(S_t, A_t) := Q(S_t, A_t) + \frac{W}{c(S_t, A_t)} (G - Q(S_t, A_t))$
    - $\pi(a \mid S_t) := \arg\max_a Q(S_t, a)$
    - если $A_t \neq \pi(S_t)$, то перейти к следующему эпизоду
    - $W := \frac{W}{b(A_t \mid S_t)}$.
- А если убрать $\arg\max$, то получится просто алгоритм оценки данной стратегии π.

TD-обучение

TD-обучение

- Общий принцип TD-обучения: давайте обучать оценки состояний на основе обученных нами ранее оценок для последующих состояний.
- $TD(0)$-обучение: инициализировать $V(s)$ и π произвольно, затем на каждом эпизоде обучения:
  - инициализировать s;
  - для каждого шага t в эпизоде:
    - выбрать A_t в состоянии S_t по стратегии π;
    - сделать A_t, пронаблюдать результат R_{t+1} и следующее состояние S_{t+1};
    - $V(S_t) := V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$.

TD-обучение

- Здесь по сути методы Монте-Карло и TD-обучение расходятся в том, как строить оценку для $V(s)$:
  - MC-методы оценивают $V(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$, собирая статистику из G_t;
  - а TD-методы оценивают на один шаг вперёд как $V(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s \right]$.
- Смысл TD-обучения в том, чтобы использовать уже обученные закономерности для поиска более глубоких закономерностей.
- В результате обучение получится целенаправленным, обучается гораздо быстрее, чем другие стратегии.

TD-обучение

- Здесь тоже есть on-policy и off-policy варианты

Алгоритм 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);
  - для каждого шага в эпизоде $t = 0, \dots, T$:
    - сделать действие A_t, получить награду R_{t+1}, перейти в состояние S_{t+1};
    - выбрать A_{t+1} по стратегии, полученной из Q (например, по ϵ-жадной стратегии);
    - обновить Q:

$$Q(S_t, A_t) := Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right)$$

- Этот алгоритм умеет искать оптимальную *мягкую* π (т.е. опять нужно самому исследовать)

TD-обучение

- А off-policy вариант называется Q-обучение; он ещё проще, и это была очень мощная идея, которая до сих пор определяет многое в RL (Watkins, 1989)

Алгоритм Q-learning (off-policy TD control):

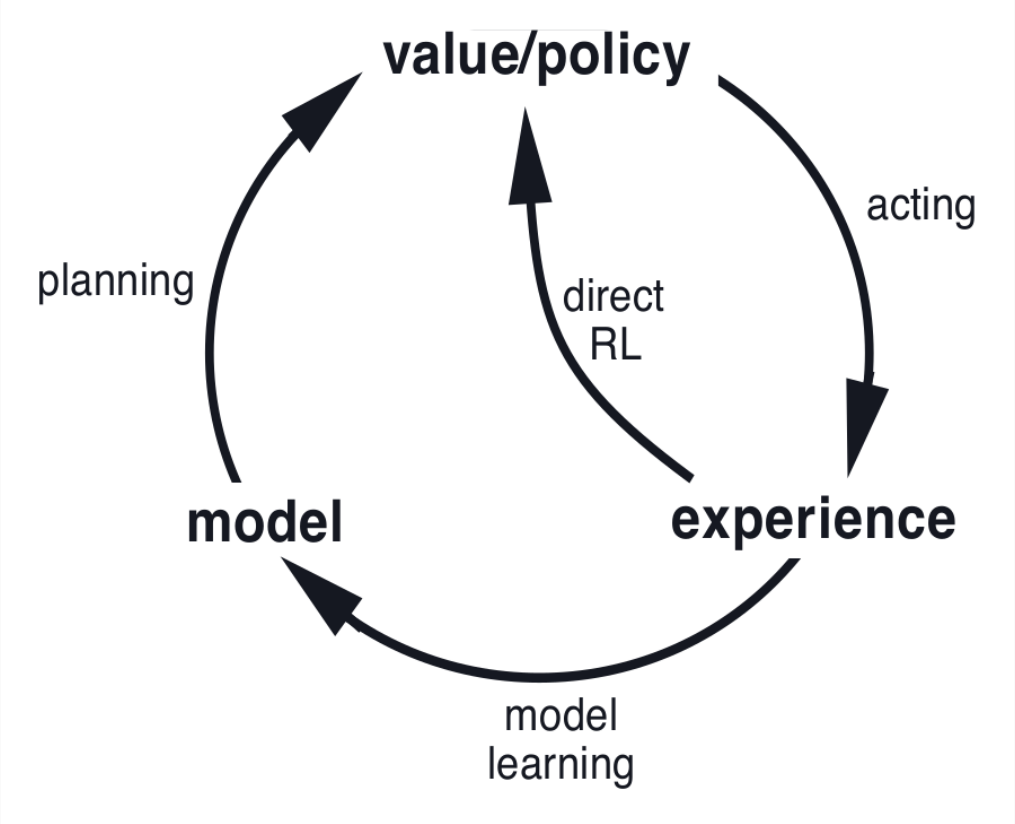
- инициализировать случайно $Q(s, a)$;
- повторять до сходимости:
  - инициализировать S_0
  - для каждого шага в эпизоде $t = 0, \dots, T$:
    - выбрать A_t по стратегии, полученной из Q (например, по ϵ-жадной стратегии);
    - сделать действие A_t, получить награду R_{t+1}, перейти в состояние S_{t+1};
    - обновить Q:

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

- Этот алгоритм умеет искать оптимальную *жёсткую* π_*, делая ходы по мягкой стратегии

Давайте подытожим

- Давайте подытожим всё то, о чём мы говорили до сих пор:



- Мы много говорили о direct RL, говорили об обучении модели окружающей среды, но планирование пока не особенно использовали, а это важная и логичная мысль...
- Давайте исправляться!

Планирование

Планирование

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

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

  - 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:

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

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

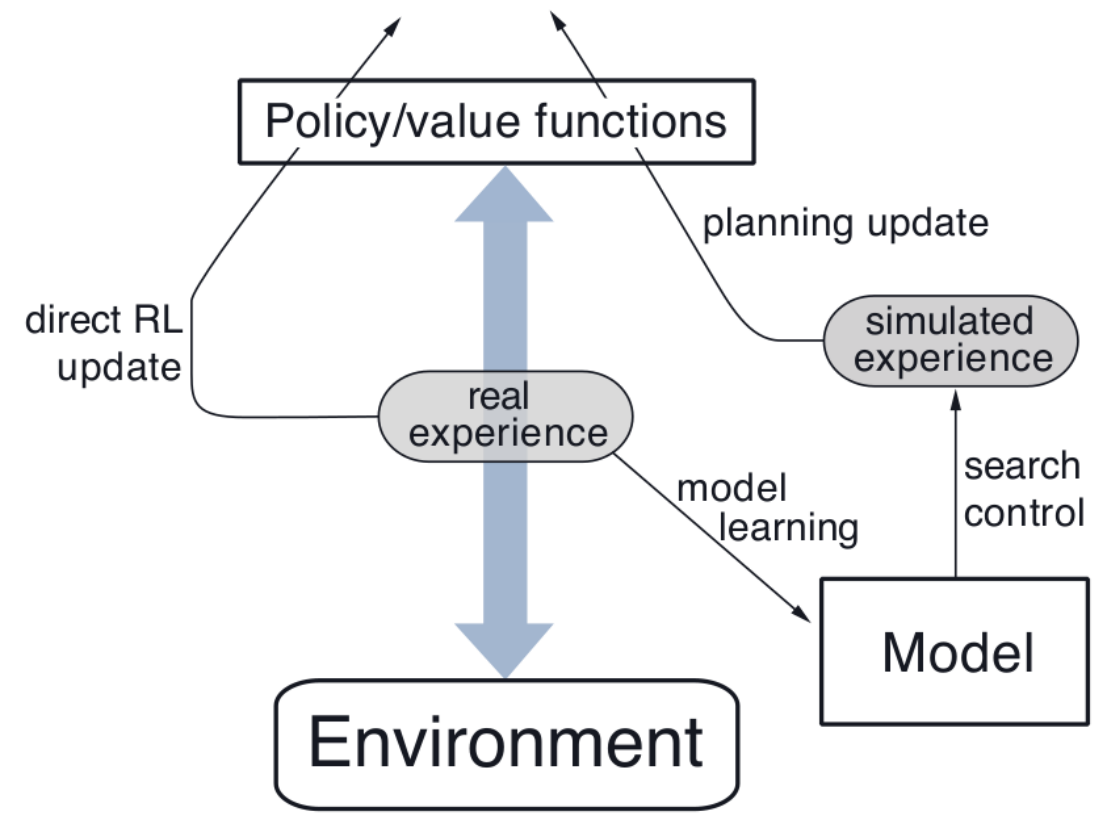
  и их оптимальные варианты V_*, Q_*;

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?..

Планирование

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



Планирование

- 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.

Планирование

- Конечно, ещё лучше будет обновлять не случайно.

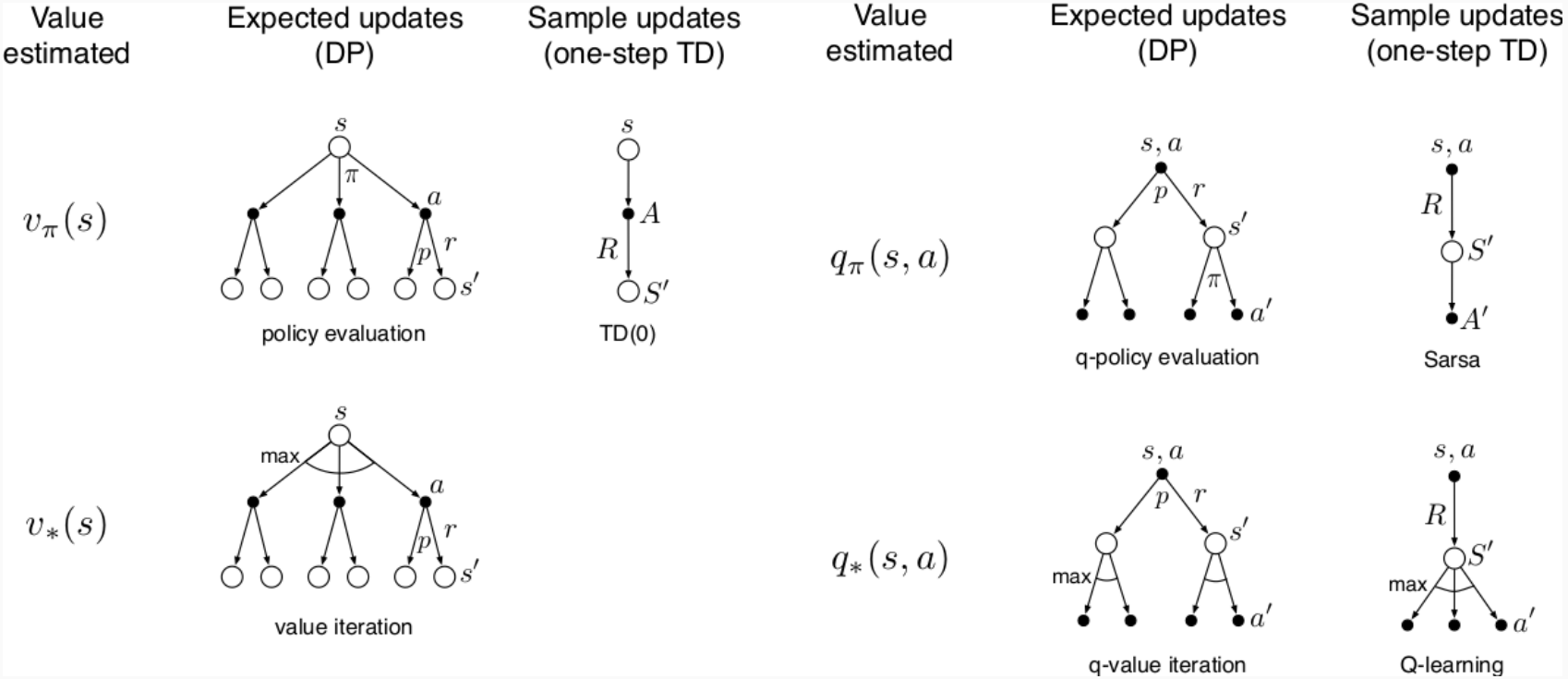
Алгоритм обхода по приоритетам (prioritized sweeping):

- 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

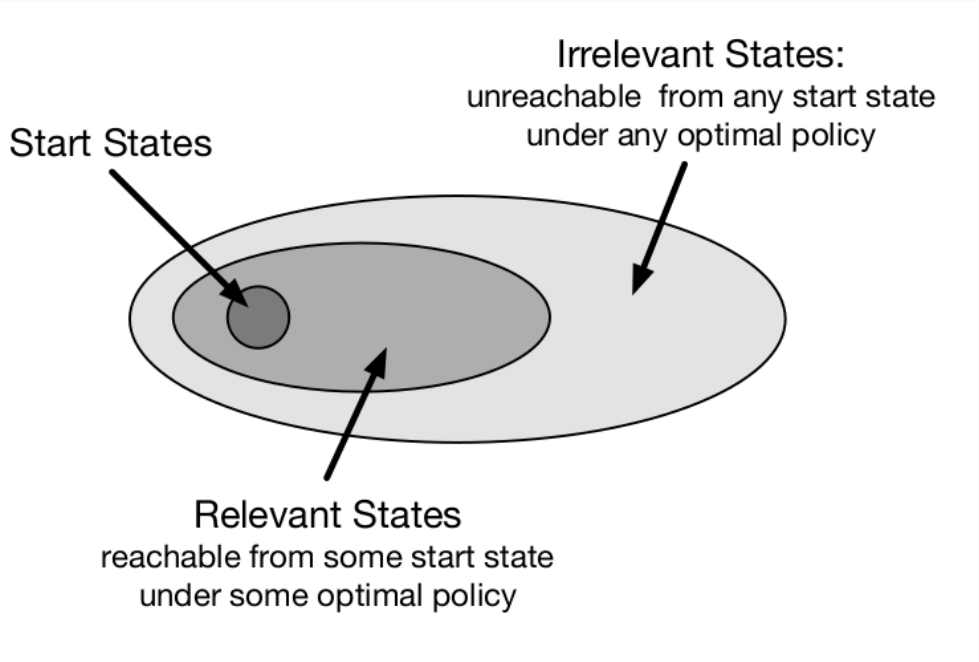
- Ещё один взгляд – expected updates vs. sample updates:



- Expected update проходит по всем возможным 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.

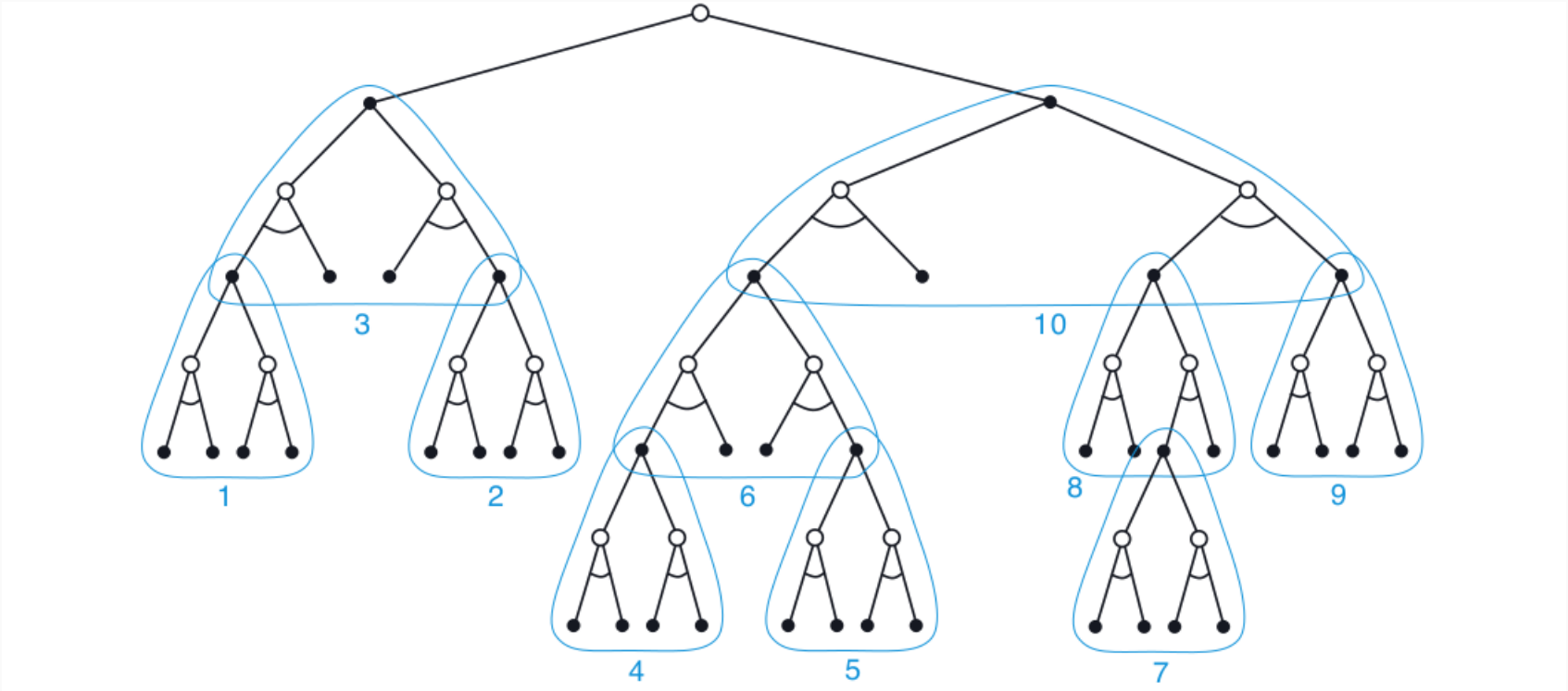


Планирование в текущем моменте

- 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

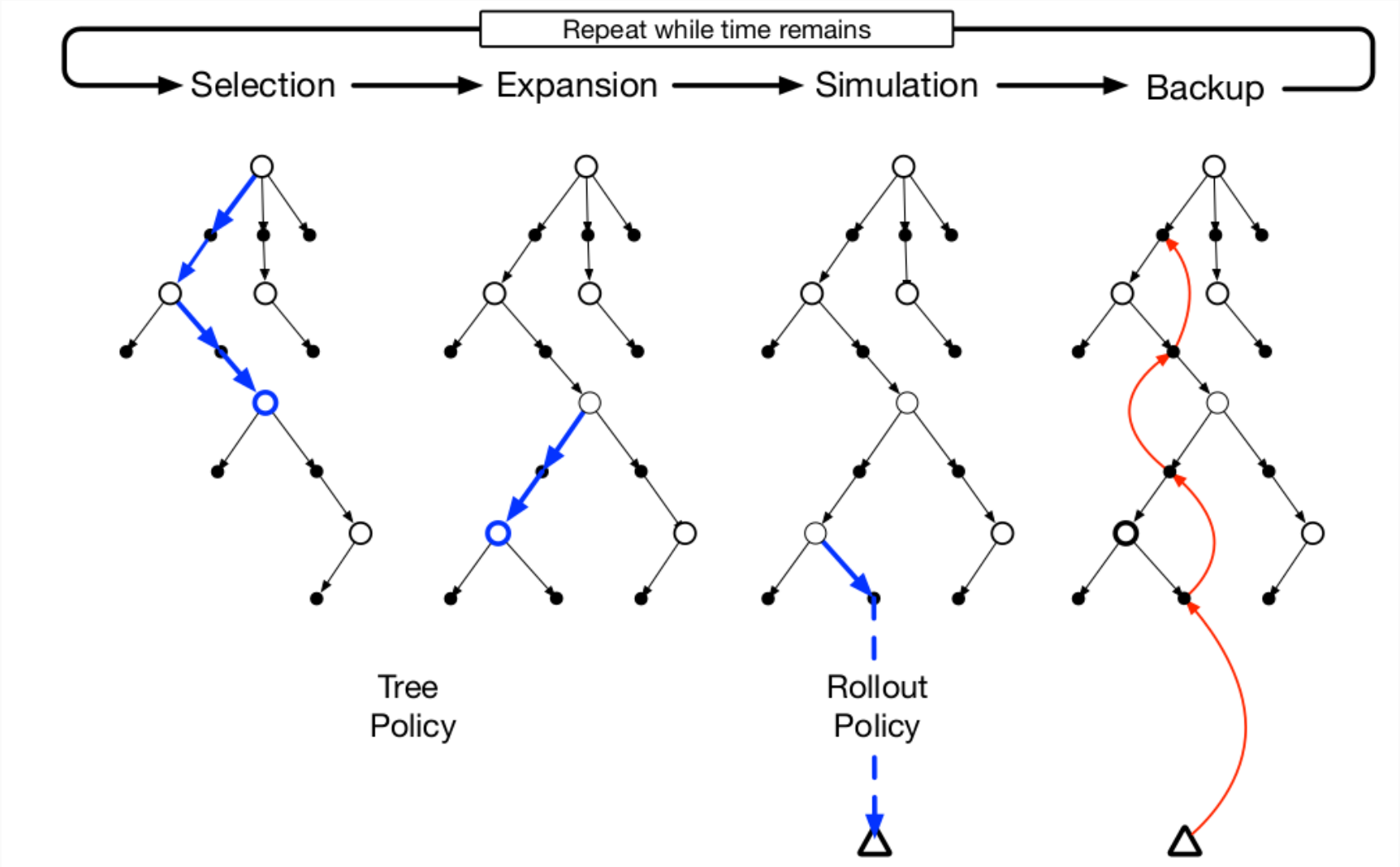
- Другой способ реализовать такое планирование: rollouts.
- Начиная с текущего состояния, симулируем несколько траекторий по текущей стратегии, а потом оцениваем действия, усредняя результаты этих симуляций.
- Когда оценки становятся достаточно точными, можно сделать ход, а потом опять строить rollouts 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.

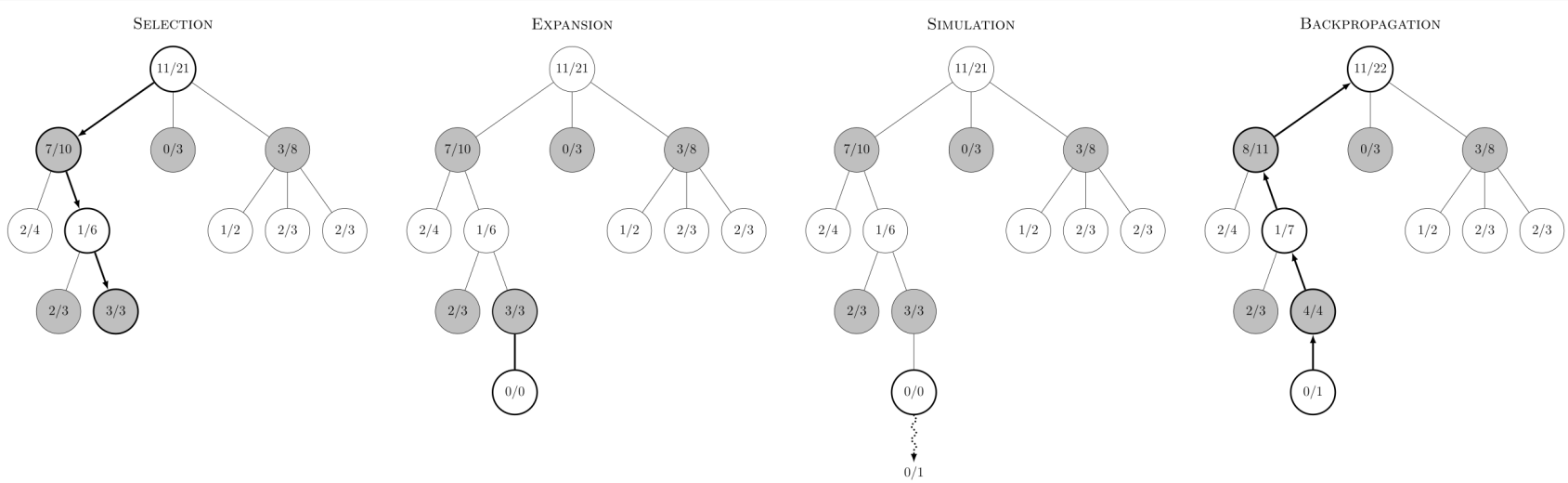
Планирование в текущем моменте

- Вот такая картинка получается:



Планирование в текущем моменте

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

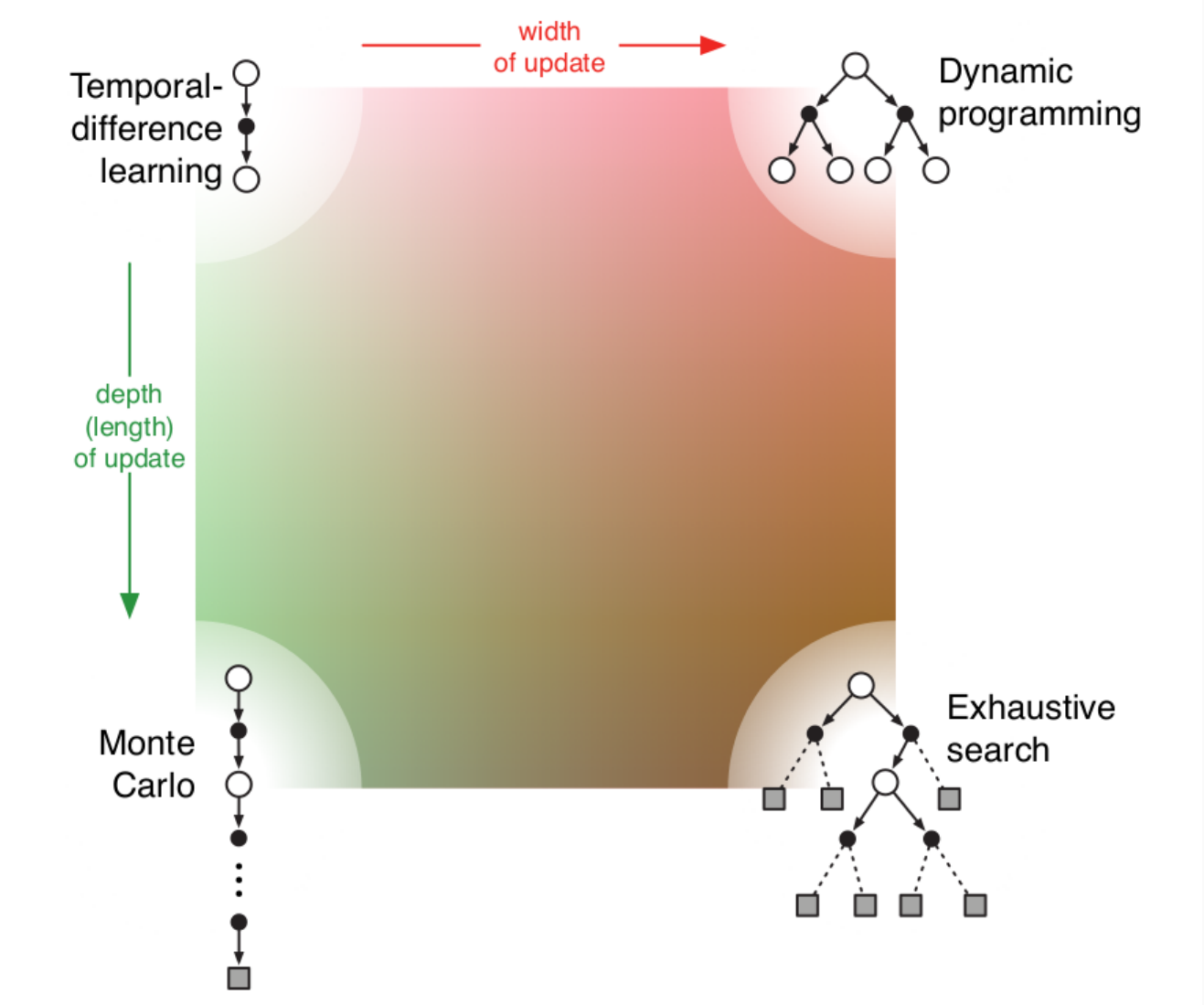


- Например, по методу UCT (upper confidence bounds applied to trees): выбираем для rollout действие, максимизирующее

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

Наши методы

- И ещё один взгляд-иллюстрация:



Приближённые методы в RL

Основные задачи

- Вспомним ключевые проблемы:
  - уравнения знаем, но пока не знаем, как их решать...
  - разных стратегий очень много — как найти оптимальную...
  - но уравнений тоже не знаем — обычно P и R не даны...
  - более того, их обычно даже записать не получится, слишком уж много состояний в любой реальной задаче... что делать?

- Вроде всё решили, кроме последней проблемы. То есть мы можем посетить небольшую часть (s, a), и надо как-то обобщить эту информацию, построить функцию, которая продолжала бы имеющуюся информацию о V или Q на другие пары (s, a).
- Как же это сделать?..

Градиентные методы в RL

- ...ну конечно, всё машинное обучение этому посвящено!
- Давайте заведём какую-нибудь модель $\hat{V}(s, \mathbf{w})$, которая будет приближать $V(s)$.
- Качество приближения оценим, например, как

$$\widetilde{VE}(\mathbf{w}) = \sum_{s \in S} \mu(s) \left(V(s) - \hat{V}(s, \mathbf{w}) \right)^2,$$

где $\mu(s)$ – веса (например равномерные, или отражающие долю времени, проведённого нами в том или ином состоянии).

Градиентные методы в RL

- Теперь можно по этой модели строить, например, SGD:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left(V(S_t) - \hat{V}(S_t, \mathbf{w}_t) \right) \nabla_{\mathbf{w}} \hat{V}(S_t, \mathbf{w}_t).$$

- Мы, конечно, не знаем $V(S_t)$, так что вместо него подставим
  - либо текущий сэмпл G_t, получая Gradient MC:

$$\mathbf{w} := \mathbf{w} + \alpha \left(G_t - \hat{V}(S_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{V}(S_t, \mathbf{w});$$

  - либо TD-оценку, получая Semi-gradient TD(0):

$$\mathbf{w} := \mathbf{w} + \alpha \left(R_{t+1} + \gamma \hat{V}(S_{t+1}, \mathbf{w}) - \hat{V}(S_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{V}(S_t, \mathbf{w}).$$

- То же самое можно сделать и с $Q(s, a)$.

Градиентные методы в RL

- И управление тоже можно сделать:
  - episodic semi-gradient Sarsa:

$$\mathbf{w} := \mathbf{w} + \alpha \left(R_{t+1} + \gamma \hat{Q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{Q}(S_t, A_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{Q}(S_t, A_t, \mathbf{w});$$

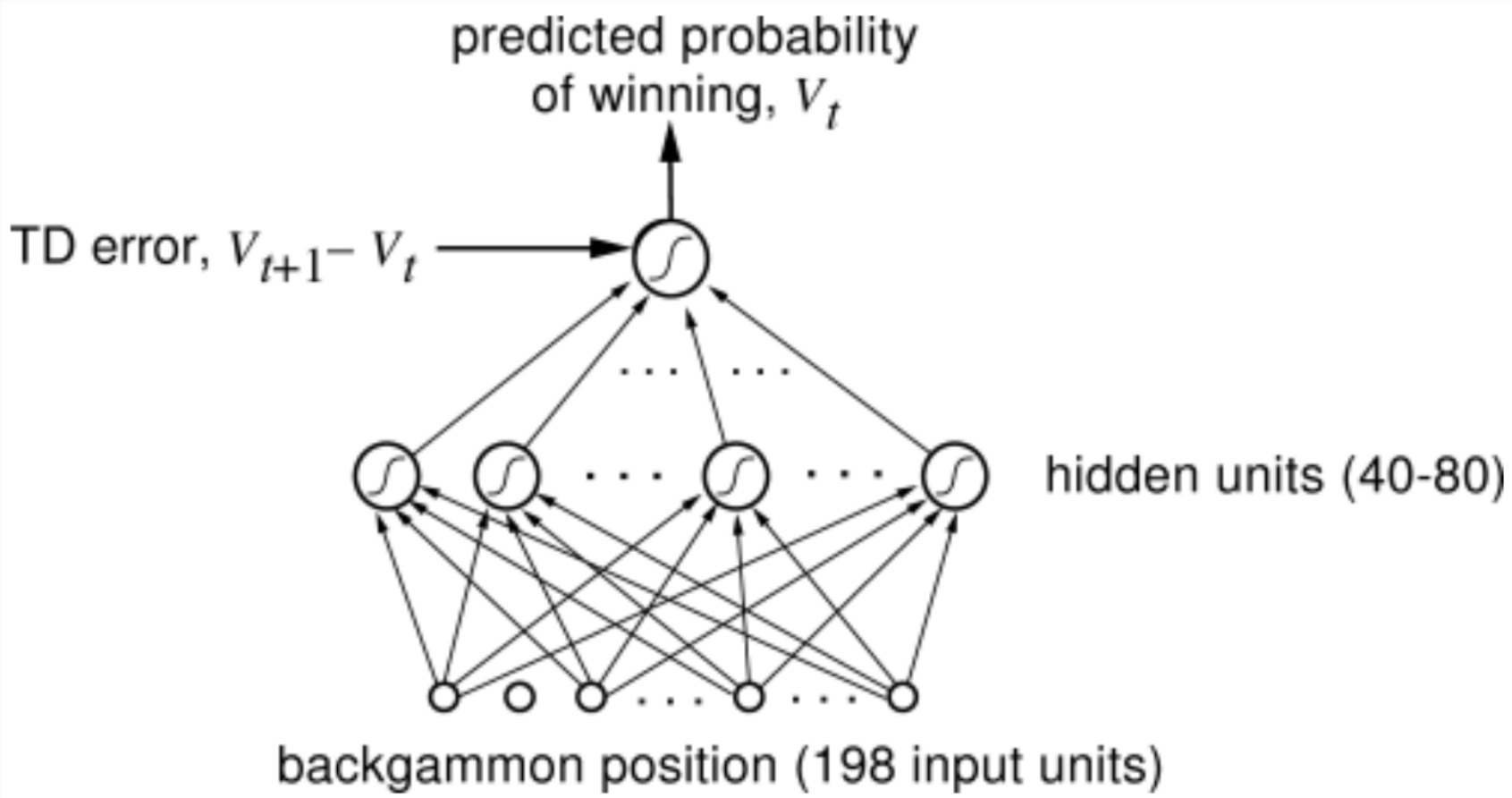
  - semi-gradient off-policy TD(0):

$$\mathbf{w} := \mathbf{w} + \alpha \frac{\pi(A_t|S_t)}{b(A_t \mid S_t)} \left(R_{t+1} + \gamma \hat{V}(S_{t+1}, \mathbf{w}) - \hat{V}(S_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{V}(S_t, \mathbf{w}).$$

- Весь Deep RL по сути является одним из этих методов, просто в качестве $\hat{Q}$ и $\hat{V}$ мы берём глубокие нейронные сети.

Градиентные методы в RL

- Первый супер-успешный пример – TD-Gammon, хотя это ещё Shallow RL :)



- А сейчас всё уже совсем по-другому, но об этом в следующем семестре...

Дополнительные замечания и расширения

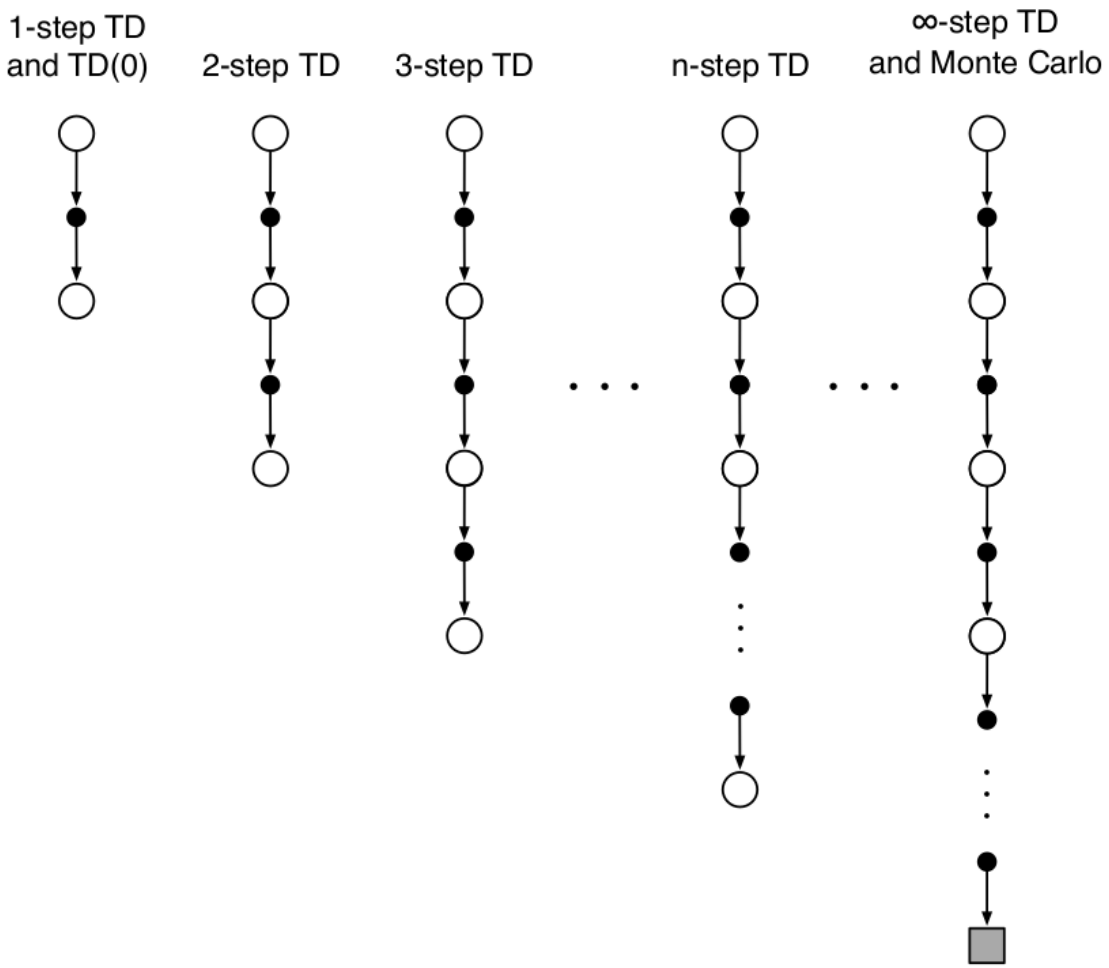
n-step алгоритмы

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

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

n-step алгоритмы

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



n-STEP АЛГОРИТМЫ

- И теперь появляются n-step варианты всех алгоритмов.

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

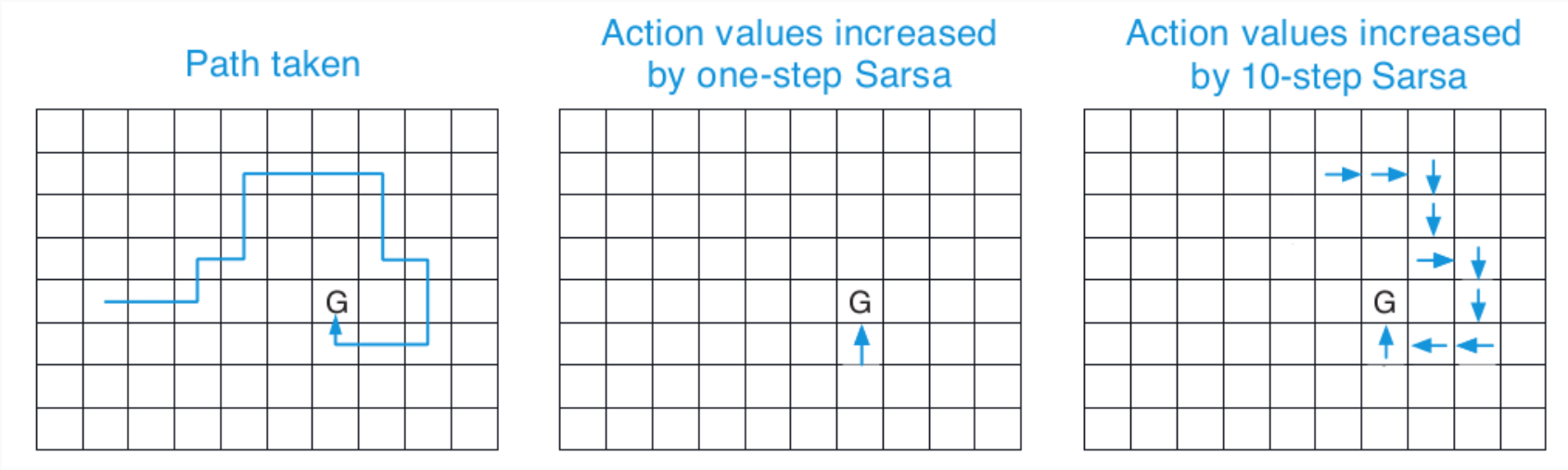
инициализировать случайно $Q(s, a)$; повторять до сходимости:

- 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-step 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:

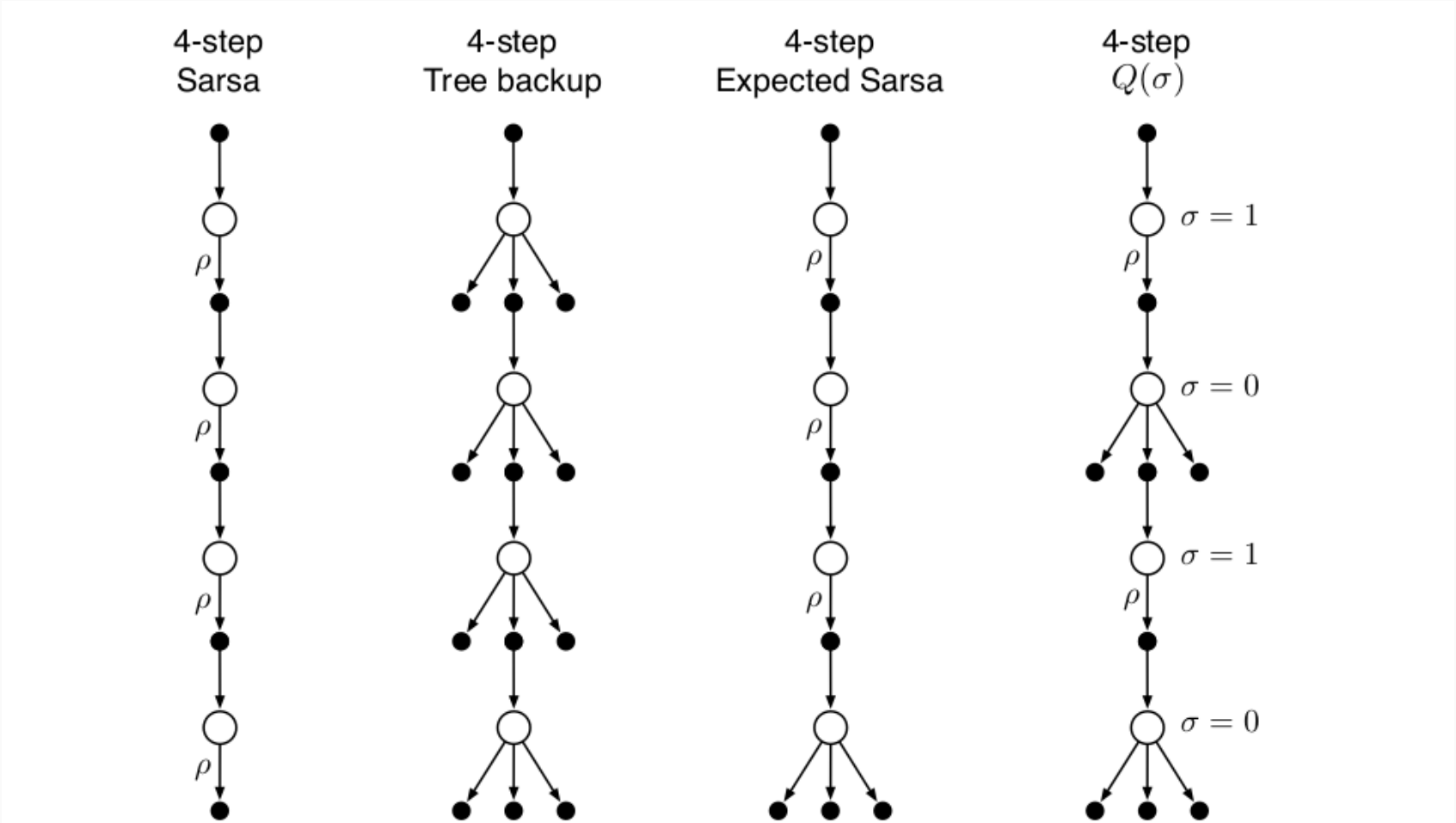
$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n,T)-1} \gamma^{k-t} \left(R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}\right.$$

- 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:

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

n-step алгоритмы

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



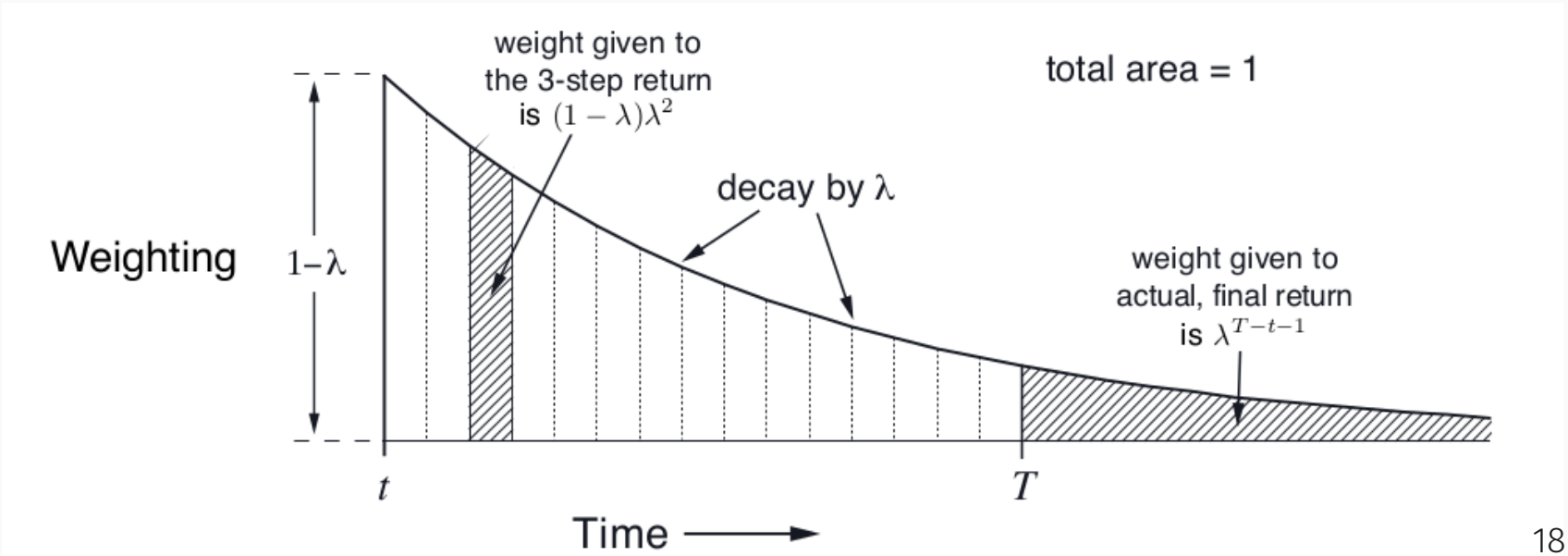
Eligibility traces

- И ещё одно расширение, которое тоже заполняет пространство между методами Монте-Карло и TD-обучением: мы называли базовый алгоритм TD(0), но что там может быть вместо нуля?
- Eligibility trace:
  - давайте введём вектор $\mathbf{z}_t \in \mathbb{R}^d$, который соответствует тому, насколько недавно в результате участвовали компоненты $\mathbf{w}_t \in \mathbb{R}^d$;
  - тогда мы будем обучать этот компонент $\mathbf{w}_t$, если z ещё не обнулился, когда мы получили ненулевую TD-ошибку.
- Параметр λ в $TD(\lambda)$ отвечает как раз за затухание этого следа.

Eligibility traces

- Если формально:
  - мы говорили только что о $G_{t:t+n}$; но можно и усреднить несколько, например $\frac{1}{2}G_{t:t+1} + \frac{1}{4}G_{t:t+2} + \frac{1}{4}G_{t:t+3}$;
  - давайте определим экспоненциально затухающее среднее $G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$;
  - тогда можно все алгоритмы так же определить, но с целью G_t^λ:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left(G_t^\lambda - \hat{V}(S_t, \mathbf{w}_t)\right) \nabla_{\mathbf{w}} \hat{V}(S_t, \mathbf{w}_t).$$



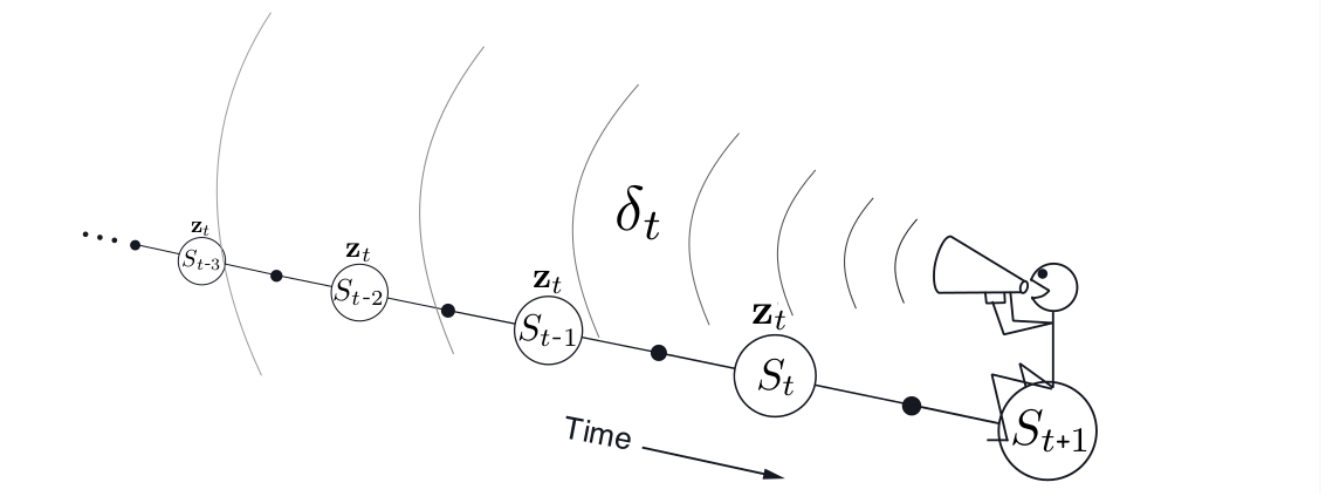
Eligibility traces

- Например, TD-обучение для оценки V_π.

Алгоритм Semi-gradient $TD(\lambda)$:

- инициализировать $\hat{V}(s, \mathbf{w})$, веса $\mathbf{w}$;
- повторять по эпизодам:
  - инициализировать S, $\mathbf{z} := 0$;
  - для каждого шага в эпизоде $t = 0, \dots, T$:
    - выбрать A по стратегии π, сделать действие A, получить R, S';
    - $\mathbf{z} := \gamma\lambda\mathbf{z} + \nabla_{\mathbf{w}}\hat{V}(S, \mathbf{w})$;
    - $\mathbf{w} := \mathbf{w} + \alpha\left(R + \gamma\hat{V}(S', \mathbf{w}) - \hat{V}(S, \mathbf{w})\right)\mathbf{z}$; и переходим $S := S'$.

- Обновляем по eligibility traces в прошлом:



Eligibility traces

- TD(λ) обобщает то, что было раньше:
  - для $\lambda = 0$ получим обычный value gradient $\mathbf{z} = \nabla_{\mathbf{w}} \hat{V}(S, \mathbf{w})$;
  - для $\lambda = 1$ более ранние состояния умножаются только на γ, и получается в точности метод Монте-Карло;
  - кстати, TD(1) – это более удобный и правильный метод реализовывать Монте-Карло, чем те, что было раньше.
- Есть ещё разные варианты, например dutch traces, но про них давайте не будем, см. (Sutton, Barto)...

Средняя награда как целевая функция

- Мы всё время формулировали цель как $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.
- Но можно и по-другому, просто усреднить по времени:

$$r(\pi) = \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}\left[R_t \mid \pi\right] = \sum_s \mu_\pi(s) \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) r,$$

  где веса $\mu_\pi(s) = \lim_{h\to\infty} p\left(S_t = s|\pi\right)$.
- Чтобы это вообще сходилось, нужно, чтобы процесс был *эргодическим*, но давайте не будем в это углубляться.

Средняя награда как целевая функция

- Теперь награды определяются через разницу со средней:

$$G_t = R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + \dots .$$

- Можно уравнения Беллмана построить, например

$$Q_*(s,a) = \sum_{r,s'} p\left(s', r | s, a\right) \left(r - \max_{\pi} r)\pi + \max_{a'} Q_*(s', a') \right) .$$

- Можно взять градиент и сделать, например, версию semi-gradient Sarsa:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left(R_{t+1} - \bar{R}_t + \hat{Q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{Q}(S_t, A_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{Q}(S_t, A$$

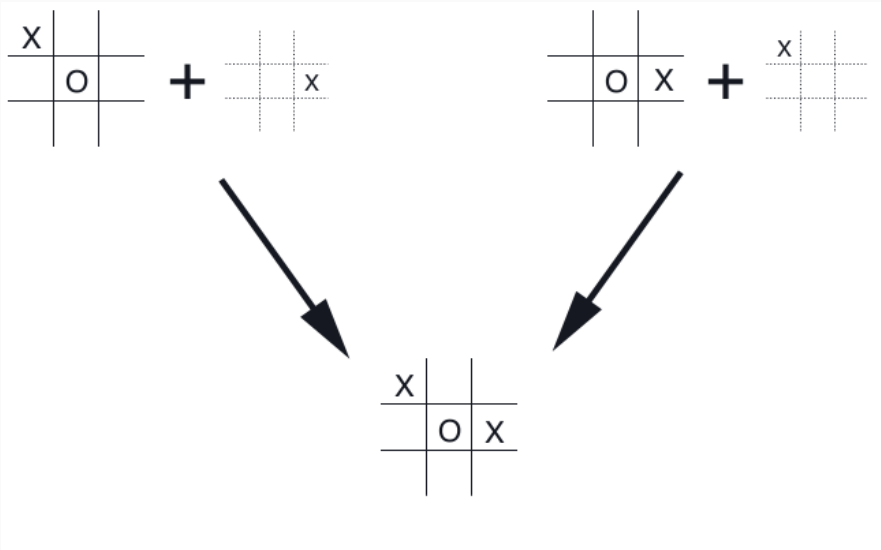
где $\bar{R}_t$ — это оценка средней награды во время t, т.е. её можно просто обновлять как
$\bar{R} := \bar{R} + \beta \left(R_{t+1} - \bar{R}_t + \hat{Q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{Q}(S_t, A_t, \mathbf{w}) \right)$.

Средняя награда как целевая функция

- Здесь самое интересное, что на самом деле между discounted reward и average reward разницы в среднем нет:
  - рассмотрим бесконечную последовательность вознаграждений;
  - тогда одно и то же вознаграждение R_t один раз появится в момент $t-1$ без γ, один раз в момент $t-2$ с коэффициентом γ, потом γ^2 и так далее;
  - и если всё усреднить, получится, что средняя награда — это просто $\frac{r(\pi)}{(1-\gamma)}$, где $r(\pi)$ — награда с дисконтом.
- Это вполне формальное рассуждение.

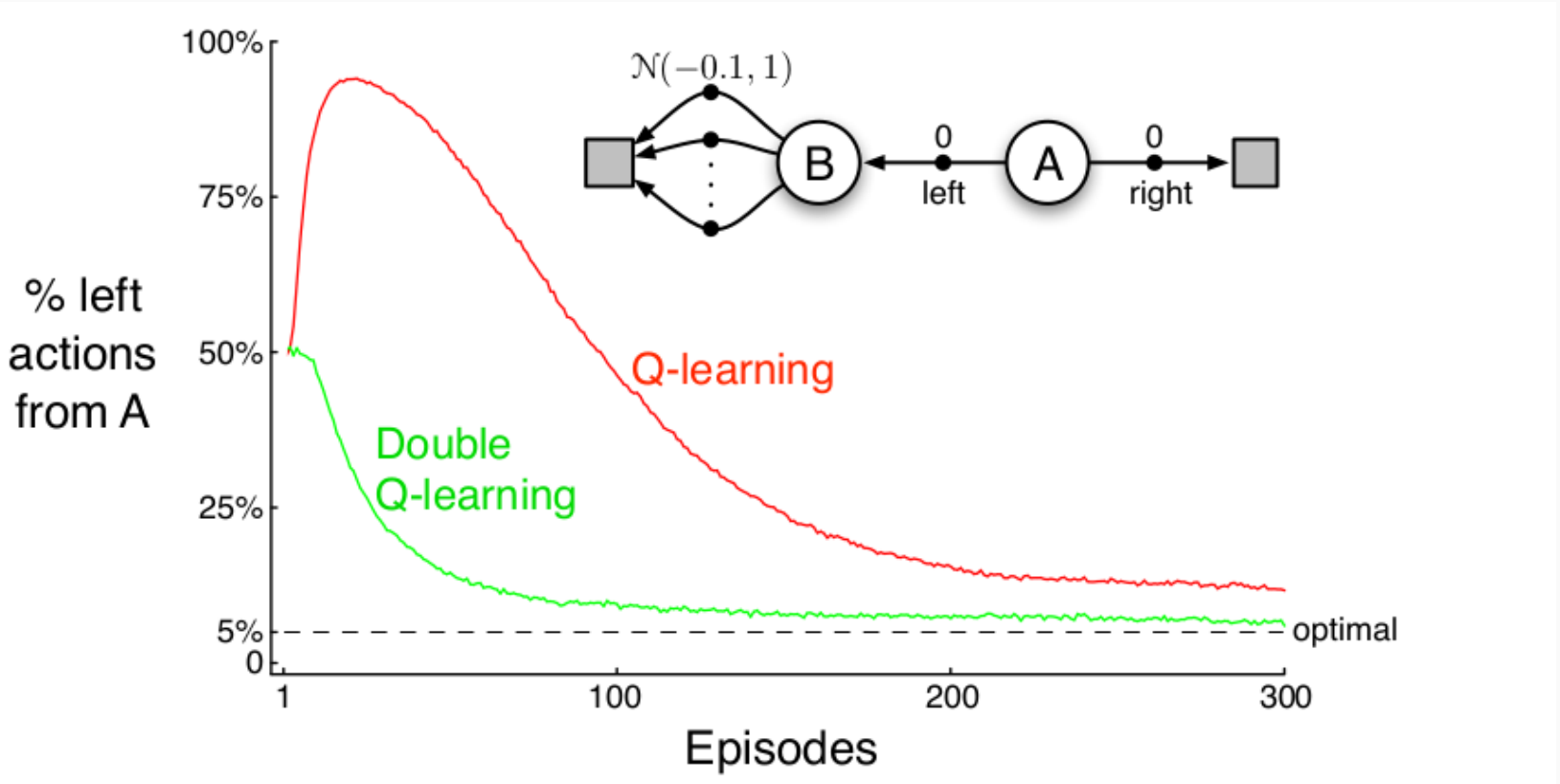
Afterstates

- 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

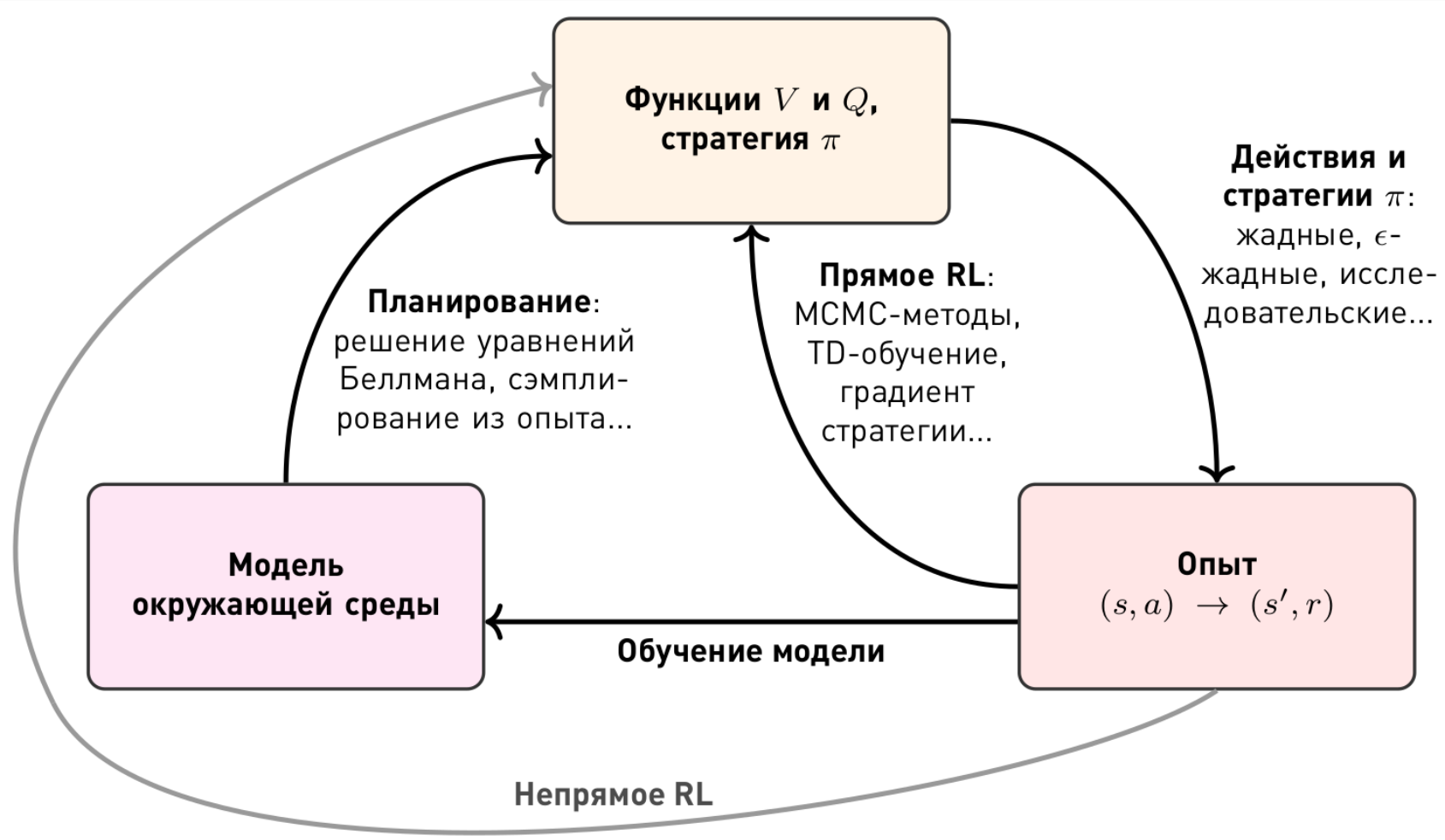
- Чтобы это поправить, можно использовать двойное обучение:
  - поддерживаем две стратегии Q_1 и Q_2;
  - каждый раз бросаем монетку, выбираем, какую брать стратегию для обновления, а какую для цели:

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

или

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

Обучение с подкреплением



Планирование

Планирование

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

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

  - 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:

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

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

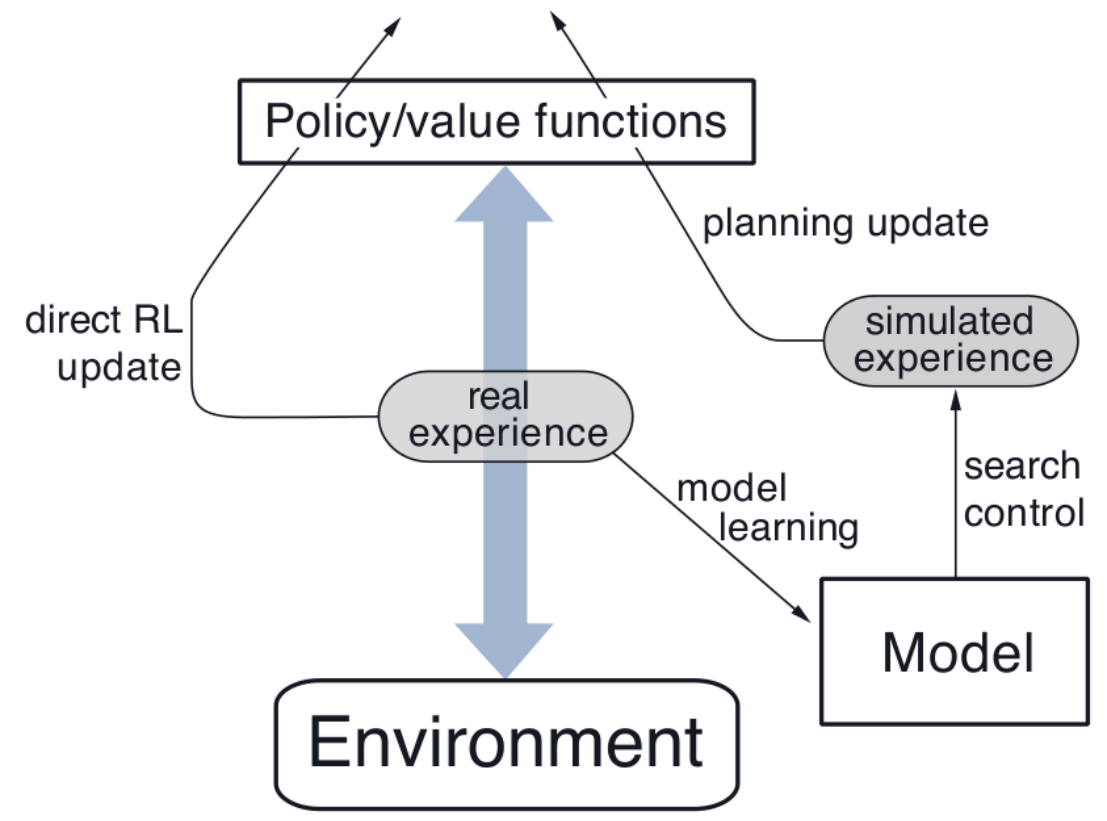
  и их оптимальные варианты V_*, Q_*;

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?..

Планирование

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



Планирование

- 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.

Планирование

- Конечно, ещё лучше будет обновлять не случайно.

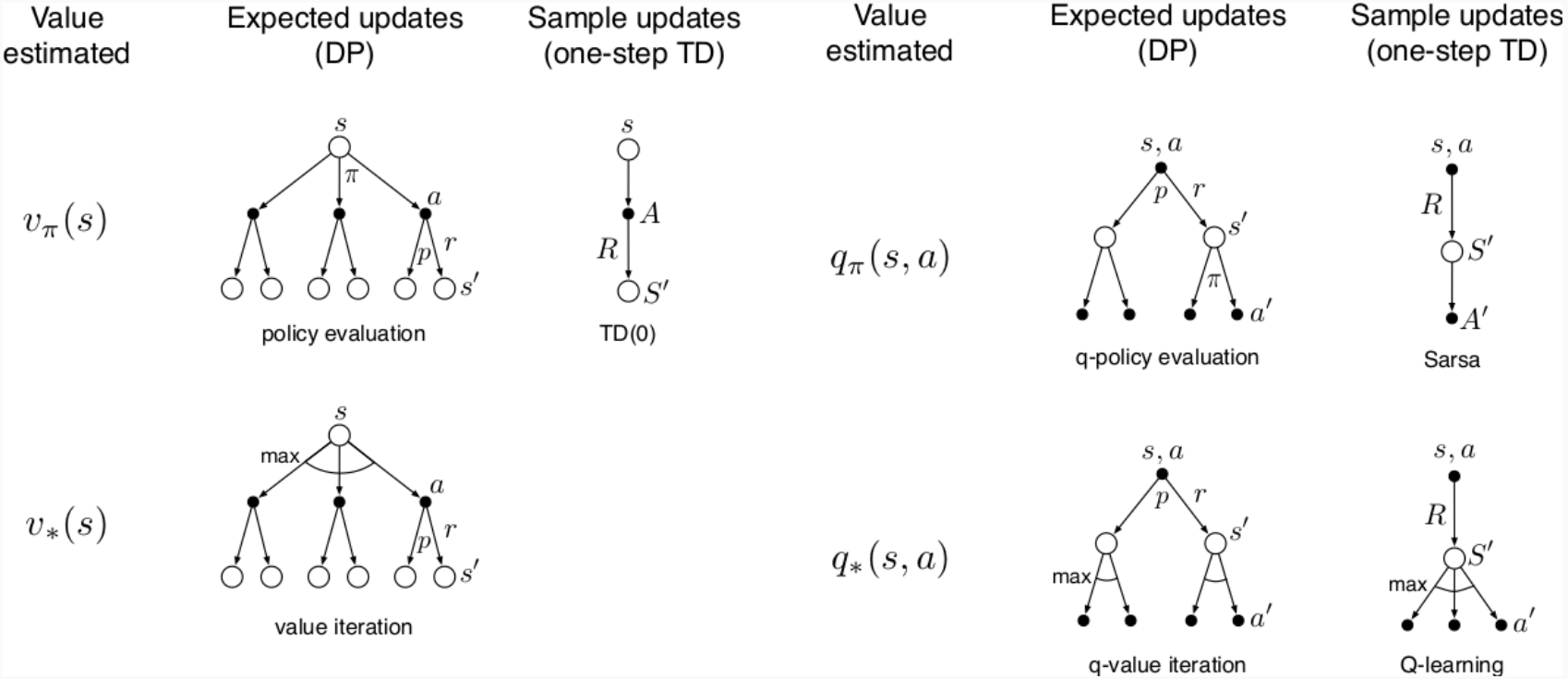
Алгоритм обхода по приоритетам (prioritized sweeping):

- 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

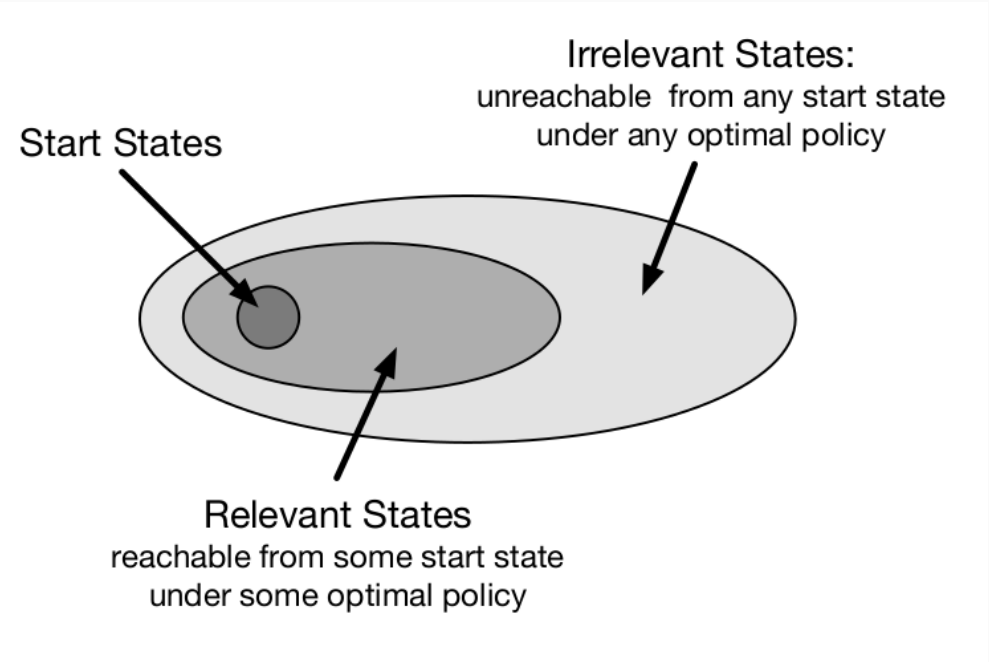
- Ещё один взгляд – expected updates vs. sample updates:



- Expected update проходит по всем возможным 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.

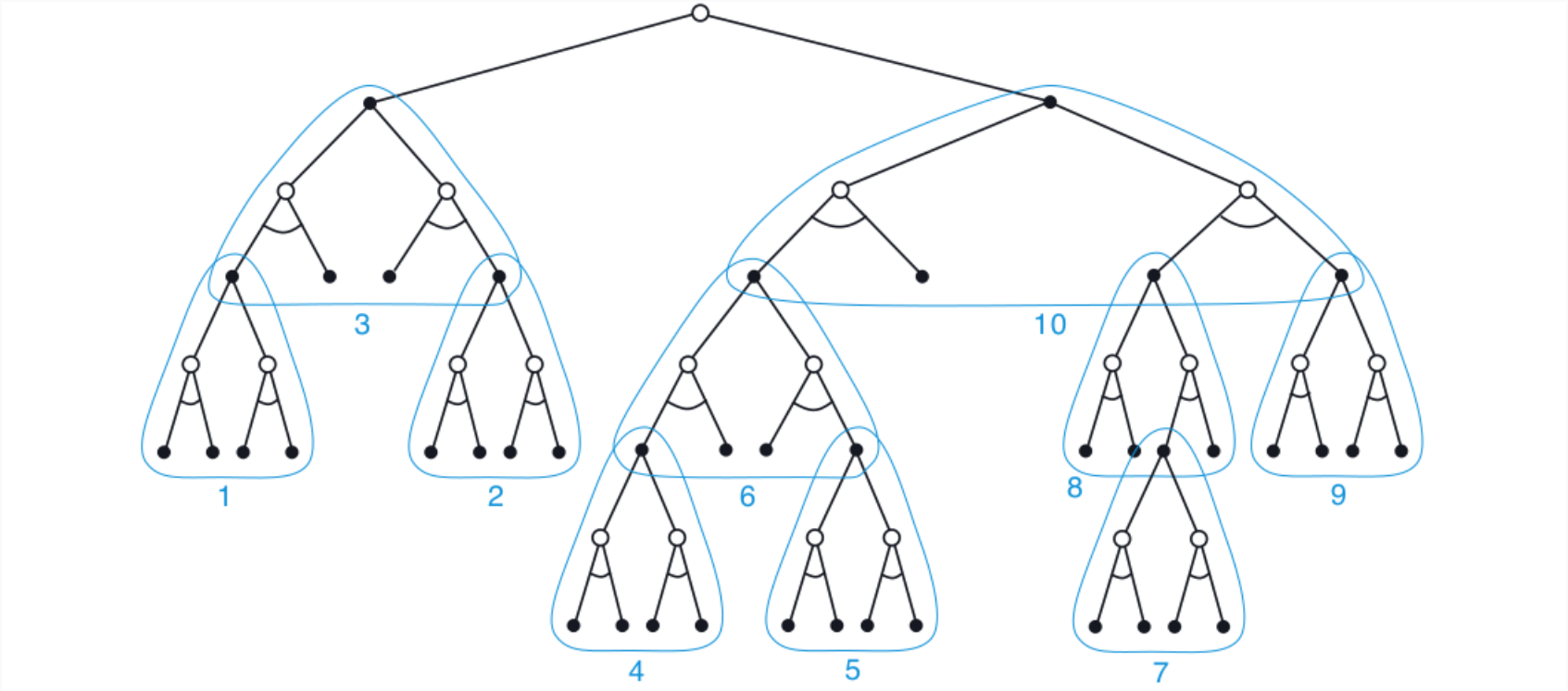


Планирование в текущем моменте

- 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

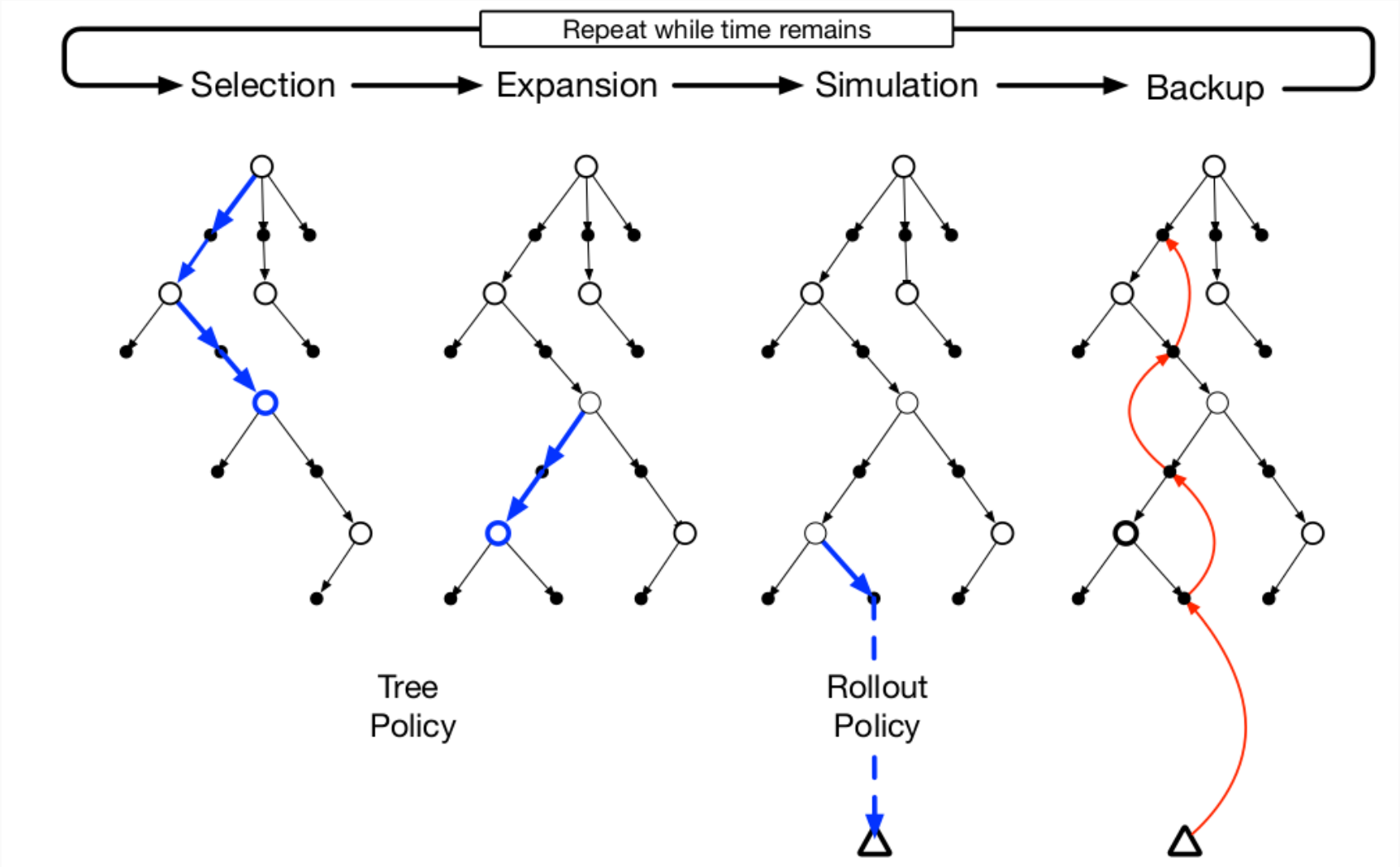
- Другой способ реализовать такое планирование: rollouts.
- Начиная с текущего состояния, симулируем несколько траекторий по текущей стратегии, а потом оцениваем действия, усредняя результаты этих симуляций.
- Когда оценки становятся достаточно точными, можно сделать ход, а потом опять строить rollouts 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.

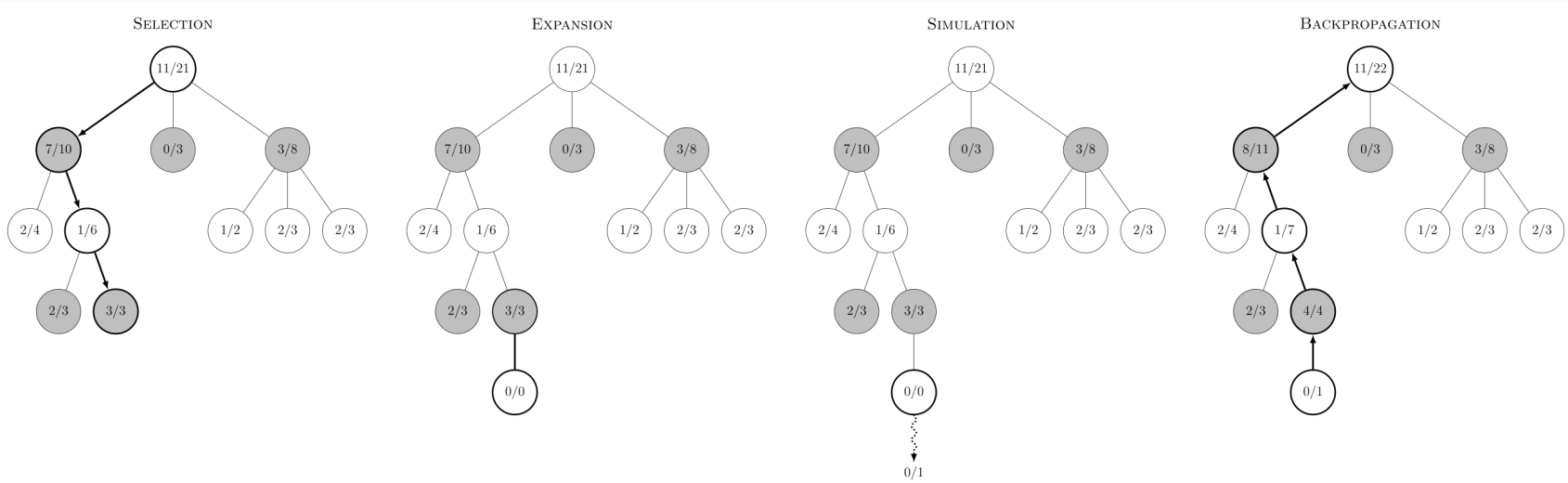
Планирование в текущем моменте

- Вот такая картинка получается:



Планирование в текущем моменте

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

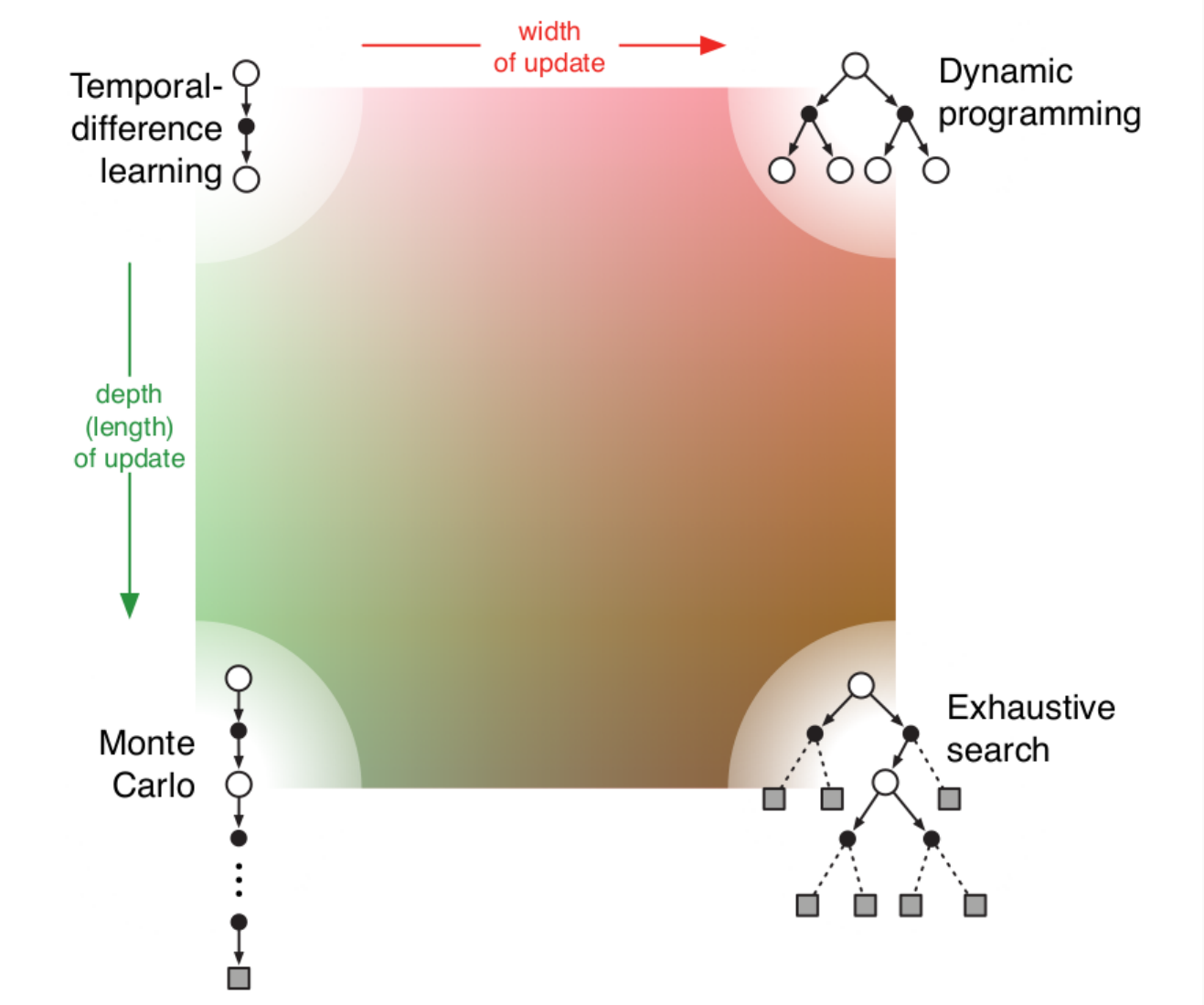


- Например, по методу UCT (upper confidence bounds applied to trees): выбираем для rollout действие, максимизирующее

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

Наши методы

- И ещё один взгляд-иллюстрация:



Policy gradient

Policy gradient

- Мы до сих пор говорили про action-value методы: берём марковский процесс принятия решений и ищем V и/или Q.
- Но это не единственный подход!
- Что если мы будем оптимизировать стратегии напрямую?

$$\pi\left(a|s,\theta\right)=p\left(A_t=a|S_t=s,\theta_t=\theta\right)$$

- Может быть, есть ещё и $\hat{V}(s,\mathbf{w})$, это отдельно.

Policy gradient

- Тогда можно попробовать сформулировать цель $J(\theta)$ и просто двигаться как

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta).$$

- Такие методы называются policy gradient методами.
- А если ещё и $\hat{V}(s, \mathbf{w})$, то actor-critic методами: критик – это $\hat{V}$, которая «критикует» стратегию π.

Policy gradient

- Если действий не слишком много, можно искать $h(s, a, \theta)$ (хоть линейно, хоть нейросетью) и строить

$$\pi\left(a|s, \theta\right) = \mathrm{softmax}\left(h\right).$$

- Преимущества:
  - можно сходиться к детерминированной стратегии без явной стратегии уменьшения ϵ;
  - а можно строить стохастическую стратегию, которую через $Q(s, a)$ вообще непонятно как найти;
  - оптимизировать π может быть проще, чем Q.
- Но как же это сделать?

Policy gradient

- Policy gradient theorem:

$$\nabla V_\pi(s) = \nabla \left(\sum_a \pi(a|s) Q_\pi(s,a) \right) =$$

$$= \ldots =$$

$$= \sum_{x \in S} \sum_{k=0}^{\infty} p(\text{перейти из } s \text{ в } x \text{ за } k \text{ шагов по } \pi) \sum_a \nabla \pi(a|x) Q_\pi(x,a),$$

то есть

$$\nabla J(\theta) \propto \nabla V_\pi(s) = \sum_s \mu(s) \sum_a \nabla \pi(a|s) Q_\pi(s,a),$$

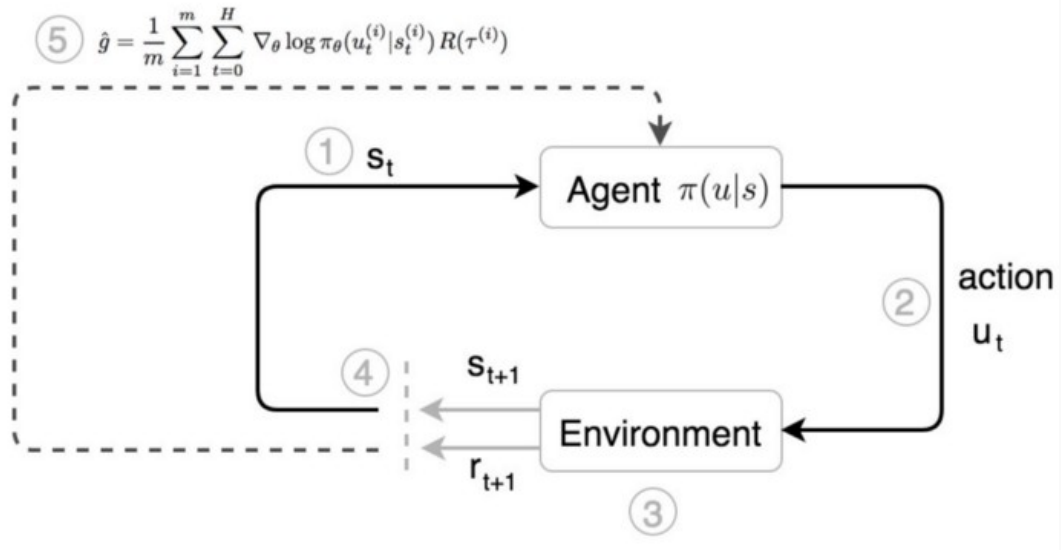
где $\mu(s)$ – ожидаемая доля времени, проведённого в состоянии s.

Policy gradient

- Иначе говоря,

$$\nabla J(\theta) \propto \mathbb{E}_{\pi}\left[\sum_{a} \nabla \pi\left(a|s\right) Q_{\pi}(s,a)\right],$$

и теперь можно и алгоритмы построить.



- Можно all-actions алгоритм:

$$\theta_{t+1} = \theta_t + \alpha \sum_{a} \hat{Q}(S_t, a, \mathbf{w}) \nabla_\theta \pi\left(a|S_t, \theta\right).$$

Policy gradient

- Но лучше уж сразу REINFORCE (Williams, 1992):

$$\nabla J(\theta) \propto \mathbb{E}_\pi \left[\sum_a \nabla \pi \left(A_t | S_t, \theta \right) Q_\pi(S_t, a) \right] =$$

$$= \mathbb{E}_\pi \left[Q_\pi(S_t, A_t) \frac{\nabla \pi \left(A_t | S_t, \theta \right)}{\pi \left(A_t | S_t, \theta \right)} \right] = \mathbb{E}_\pi \left[G_t \frac{\nabla \pi \left(A_t | S_t, \theta \right)}{\pi \left(A_t | S_t, \theta \right)} \right],$$

потому что $Q_\pi(S_t, A_t) = \mathbb{E}_\pi \left[G_t \mid S_t, A_t \right]$.

- И алгоритм такой:

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla \log \pi \left(A_t | S_t, \theta \right).$$

- Т.е. двигаем туда, где максимальный return, всё логично.

- G_t можно оценить по методу Монте-Карло:

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

- Но это ещё не всё. Ещё можно добавить baseline:

$$\nabla J(\theta) \propto \nabla V_\pi(s) = \sum_s \mu(s) \sum_a \left(Q_\pi(s,a) - b(s)\right) \nabla \pi\left(a|s,\theta\right),$$

  и это не повлияет ни на что, мы вычитаем $b(s)\nabla \sum_a \pi\left(a|s,\theta\right) = b(s)\nabla 1 = 0$.
- Это не меняет ожидание, но может сильно изменить дисперсию! В том числе в хорошую сторону.
- Естественный baseline – это оценка состояния $\hat{V}(S_t, \mathbf{w})$.

- Его нужно будет оценить, как мы обычно оцениваем $\hat{V}$, то есть теперь два разных градиента получается:

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

- А чтобы получился полноценный actor-critic, нужно через $\hat{V}$ оценить и результаты действия тоже:

$$\theta_{t+1} = \theta_t + \alpha \left(G_{t:t+1} - \hat{V}(S_t, \mathbf{w}) \right) \nabla \log \pi \left(A_t | S_t, \theta \right) =$$
$$= \theta_t + \alpha \left(R_{t+1} + \gamma \hat{V}(S_{t+1}, \mathbf{w}) - \hat{V}(S_t, \mathbf{w}) \right) \nabla \log \pi \left(A_t | S_t, \theta \right) =$$
$$= \theta_t + \alpha \delta_t \frac{\nabla \pi \left(A_t | S_t, \theta \right)}{\pi \left(A_t | S_t, \theta \right)}.$$

- Алгоритм получается такой:

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Policy gradient

- Можно с eligibility traces, как мы раньше обсуждали:

Actor–Critic with Eligibility Traces (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1]$, $\lambda^{\mathbf{w}} \in [0,1]$; step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \mathbf{0}$ (d'-component eligibility trace vector)
  $\mathbf{z}^{\mathbf{w}} \leftarrow \mathbf{0}$ (d-component eligibility trace vector)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{z}^{\mathbf{w}} \leftarrow \gamma\lambda^{\mathbf{w}}\mathbf{z}^{\mathbf{w}} + \nabla\hat{v}(S,\mathbf{w})$
    $\mathbf{z}^{\boldsymbol{\theta}} \leftarrow \gamma\lambda^{\boldsymbol{\theta}}\mathbf{z}^{\boldsymbol{\theta}} + I\nabla \ln\pi(A|S,\boldsymbol{\theta})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\mathbf{z}^{\mathbf{w}}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta\mathbf{z}^{\boldsymbol{\theta}}$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

- А можно попробовать сделать off-policy вариант; для другой стратегии β:

$$\nabla J(\theta) \propto \nabla_\theta \sum_s \mu_\pi(s) \sum_a \pi\left(a|s\right) Q_\pi(s,a) =$$

$$= \sum_s \mu_\pi(s) \sum_a \left[\nabla_\theta \pi\left(a|s\right) Q_\pi(s,a) + \pi\left(a|s\right) \nabla_\theta Q_\pi(s,a)\right],$$

- Теперь первое слагаемое можно оценить как обычно:

$$\sum_s \mu_\pi(s)[\ldots] = \mathbb{E}_\pi\left[\ldots\right] = \mathbb{E}_\beta\left[\frac{\pi\left(a|s\right)}{\beta(a \mid s)}\ldots\right], \text{ то есть}$$

$$\nabla J(\theta) \propto \mathbb{E}_\beta\left[\frac{\pi\left(a|s\right)}{\beta(a \mid s)} Q_\pi(s,a) \nabla_\theta \ln \pi\left(a|s\right)\right].$$

Policy gradient

- На первый взгляд кажется, что всё равно всё пропало, потому что $\nabla_\theta Q_\pi(s, a)$ мы никак не оценим.
- Но оказывается (Degris et al., 2012), что его можно просто выбросить, и полученная аппроксимация всё равно неплохая и минимум у неё там же.
- Так что в алгоритме просто добавятся importance weights:

Algorithm 1 The Off-PAC algorithm

Initialize the vectors $\mathbf{e}_v$, $\mathbf{e}_u$, and $\mathbf{w}$ to zero
Initialize the vectors $\mathbf{v}$ and $\mathbf{u}$ arbitrarily
Initialize the state s
For each step:
  Choose an action, a, according to $b(\cdot|s)$
  Observe resultant reward, r, and next state, s'
  $\delta \leftarrow r + \gamma(s')\mathbf{v}^\mathsf{T}\mathbf{x}_{s'} - \mathbf{v}^\mathsf{T}\mathbf{x}_s$
  $\rho \leftarrow \pi_\mathbf{u}(a|s)/b(a|s)$
  Update the critic (GTD(λ) algorithm):
    $\mathbf{e}_v \leftarrow \rho\left(\mathbf{x}_s + \gamma(s)\lambda\mathbf{e}_v\right)$
    $\mathbf{v} \leftarrow \mathbf{v} + \alpha_v\left[\delta\mathbf{e}_v - \gamma(s')(1-\lambda)(\mathbf{w}^\mathsf{T}\mathbf{e}_v)\mathbf{x}_s\right]$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha_w\left[\delta\mathbf{e}_v - (\mathbf{w}^\mathsf{T}\mathbf{x}_s)\mathbf{x}_s\right]$
  Update the actor:
    $\mathbf{e}_u \leftarrow \rho\left[\frac{\nabla_\mathbf{u}\pi_\mathbf{u}(a|s)}{\pi_\mathbf{u}(a|s)} + \gamma(s)\lambda\mathbf{e}_u\right]$
    $\mathbf{u} \leftarrow \mathbf{u} + \alpha_u\delta\mathbf{e}_u$
  $s \leftarrow s'$

Policy gradient

- В policy gradient можно рассмотреть даже, например, непрерывные действия.
- Например, параметризуем действие $a \in \mathbb{R}$ через гауссиан:

$$\pi\left(a|s,\theta\right) = \frac{1}{\sigma(s,\theta)\sqrt{2\pi}} e^{-\frac{1}{2\sigma(s,\theta)}\left(a-\mu(s,\theta)\right)^2}.$$

- Если, например,

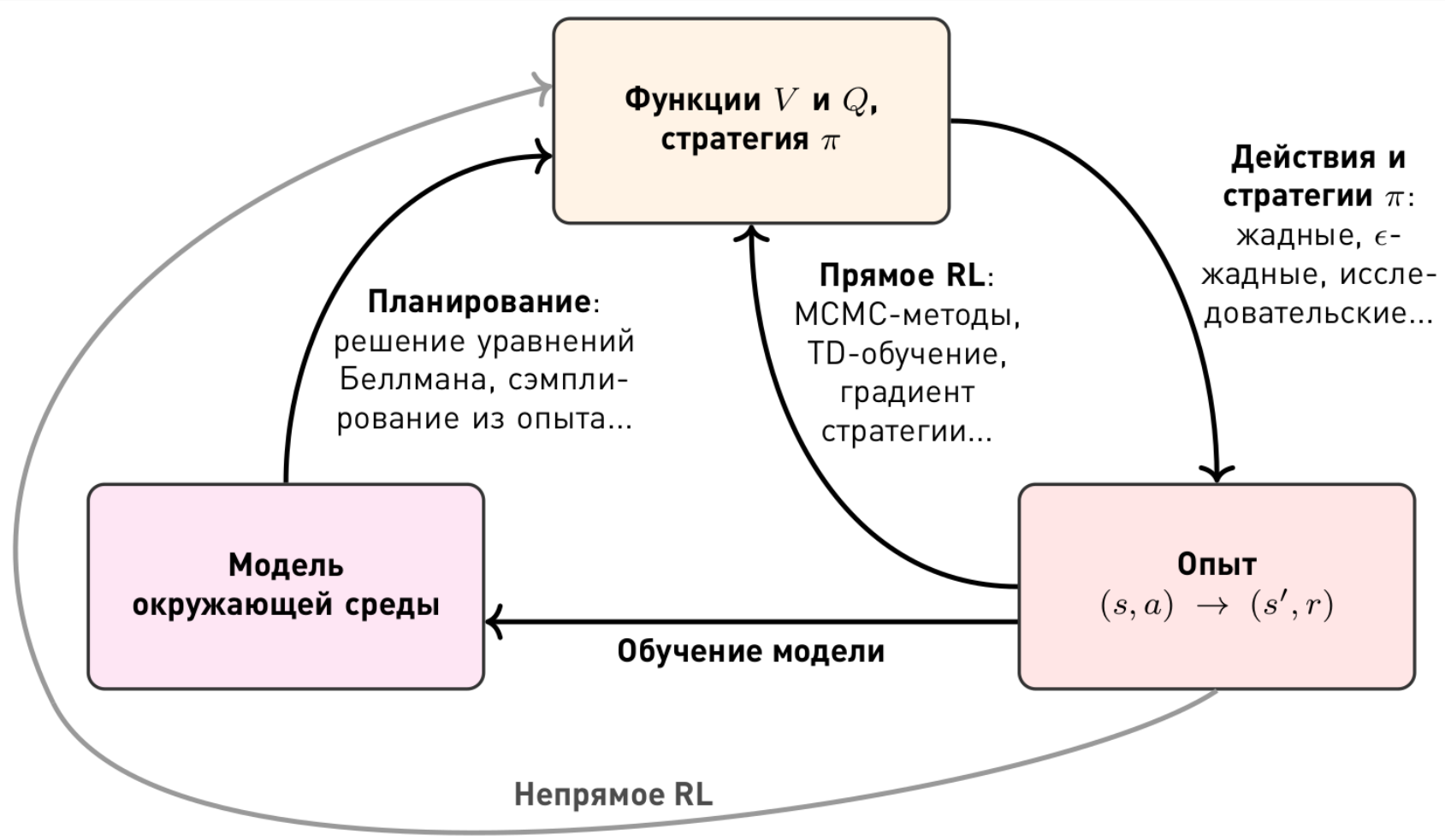
$$\mu(s,\theta) = \theta_\mu^\top \mathbf{x}_\mu(s), \qquad \sigma(s,\theta) = e^{\theta_\sigma^\top \mathbf{x}_\sigma(s)},$$

то можно подсчитать градиенты:

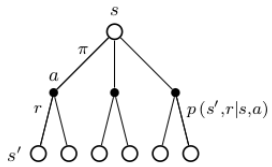
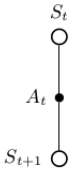
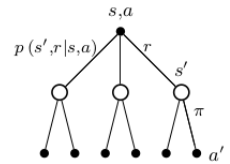
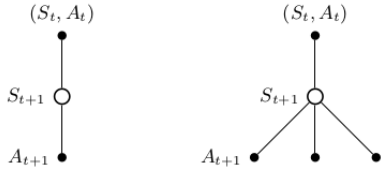
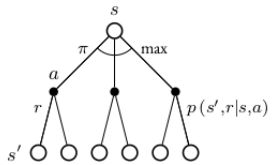
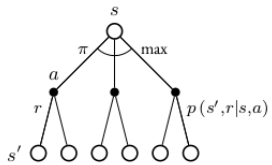
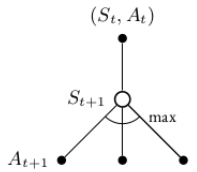
$$\nabla \ln \pi\left(a|s,\theta_\mu\right) = \frac{1}{\sigma(s,\theta)^2}\left(a-\mu(s,\theta)\right)\mathbf{x}_\mu(s),$$

$$\nabla \ln \pi\left(a|s,\theta_\sigma\right) = \left(\frac{1}{\sigma(s,\theta)^2}\left(a-\sigma(s,\theta)\right)^2 - 1\right)\mathbf{x}_\sigma(s).$$

Обучение с подкреплением



Обзор методов

	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** $Q(S_t,A_t) + \alpha\,(R_{t+1} + \gamma Q(S_{t+1},A_{t+1}) - Q(S_t,A_t))$ $Q(S_t,A_t) + \alpha\,\big(R_{t+1} + \gamma \sum_a \pi(a\|S_{t+1})\,Q(S_{t+1},a) - Q(S_t,A_t)\big)$ 
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** $Q(S_t,A_t) + \alpha\,(R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t))$ 

Спасибо за внимание!