

ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ II

Сергей Николенко

СПбГУ — Санкт-Петербург

16 мая 2024 г.

Random facts:



- 16 мая 1816 г. — «день рождения индоевропеистики»; этим днём датировано предисловие работы Франца Боппа *Über das Conjugationssystem der Sanskritsprache in Vergleichung mit jenem der griechischen, lateinischen, persischen und germanischen Sprache* («О системе спряжений санскрита в сравнении с таковым в греческом, латинском, персидском, и германском языках»)
- 16 мая 1836 г. Эдгар Аллан По женился на своей 13-летней кузине Вирджинии Клемм
- 16 мая 1896 г. в петербургском саду «Аквариум» состоялся первый в России киносеанс
- 16 мая 1916 г. было подписано тайное соглашение Сайкса — Пико, в котором правительства Великобритании, Франции, Российской империи и позднее Италии разграничивали интересы на Ближнем Востоке
- 16 мая 1924 г. вышел первый номер журнала «Мурзилка», а 16 мая 1929 г. прошла первая церемония вручения «Оскаров» (тогда они так ещё не назывались)
- 16 мая 1970 г. британский хит-парад на три недели возглавила песня «Back Home» в исполнении сборной Англии по футболу; впрочем, в Мексике англичане проиграли уже в четвертьфинале

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} ;
 - $V(S_t) := V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$.

- Здесь по сути методы Монте-Карло и TD-обучение расходятся в том, как строить оценку для $V(s)$:
 - MC-методы оценивают $V(s) = \mathbb{E}_{\pi} [G_t \mid S_t = s]$, собирая статистику из G_t ;
 - а TD-методы оценивают на один шаг вперёд как $V(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma V_{\pi}(S_{t+1}) \mid S_t = s]$.
- 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.

- 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 :

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

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

- 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):

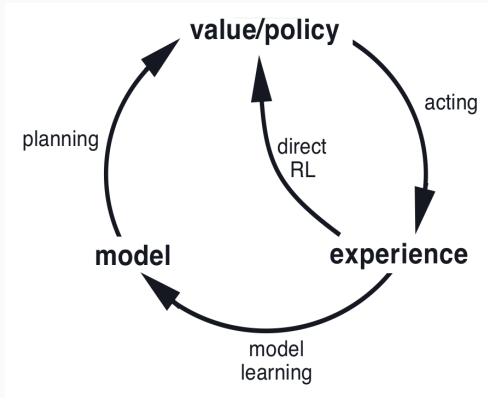
- инициализировать случайно $Q(s, a)$;
- повторять до сходимости:
 - инициализировать S_0
 - для каждого шага в эпизоде $t = 0, \dots, T$:
 - выбрать A_t по стратегии, полученной из Q (например, по ϵ -жадной стратегии);
 - сделать действие A_t , получить награду R_{t+1} , перейти в состояние S_{t+1}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

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

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

- награды за каждый эпизод, начиная со времени t :

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

- функцию значения для состояний и пар состояние-действие:

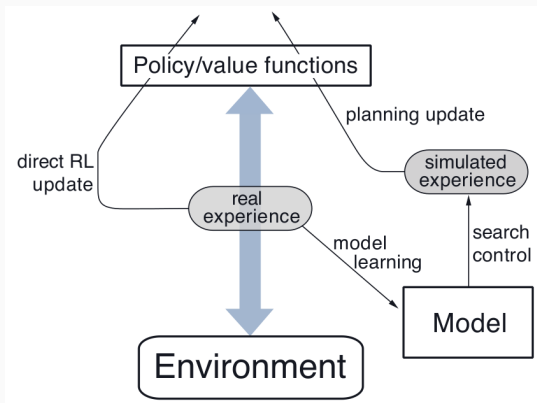
$$V_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right],$$

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

и их оптимальные варианты V_*, Q_* ;

- 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' ;
 - $Q(S, A) := Q(S, A) + \alpha (R + \gamma \max_a Q(S', a) - Q(S, A))$;
 - 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 $M(S, A)$;
 - $Q(S, A) := Q(S, A) + \alpha (R + \gamma \max_a Q(S', a) - Q(S, A))$.
- Dyna-Q случайно обновляет k значений $Q(s, a)$ по имеющейся модели; k можно выбирать из соображений имеющихся ресурсов.

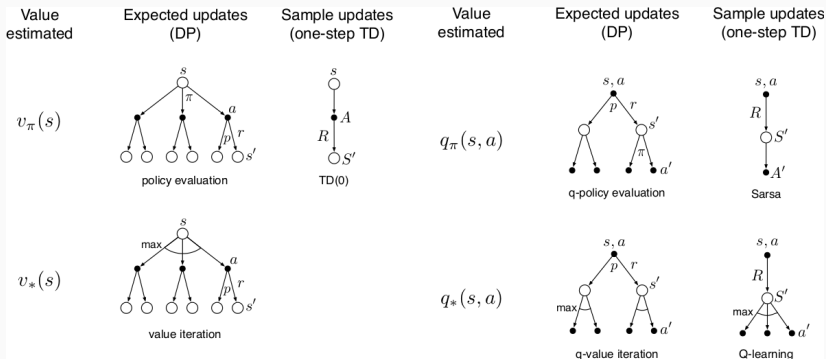
- Конечно, ещё лучше будет обновлять не случайно.

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 $(S, A) := \text{Head}(PQ)$, породить R, S' по $M(S, A)$;
 - $Q(S, A) := Q(S, A) + \alpha (R + \gamma \max_a Q(S', a) - Q(S, A))$.
 - 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 оценки вероятностей попасть в S .

EXPECTED VS. SAMPLE UPDATES

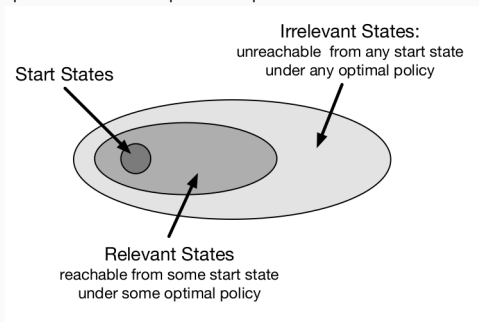
- Ещё один взгляд – expected updates vs. sample updates:



- 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

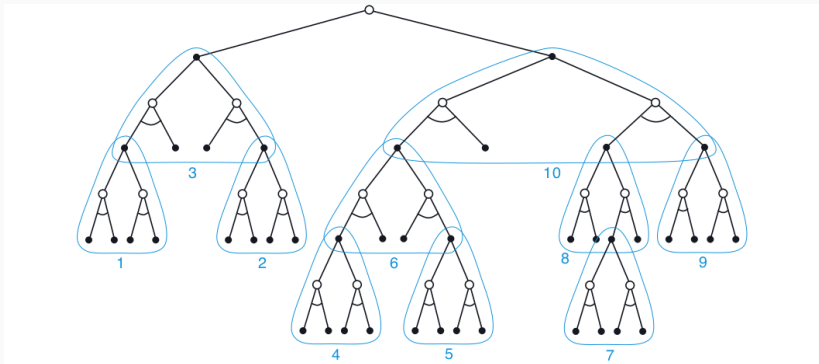
- Как лучше распределить апдейты (т.е. фактически имеющийся вычислительный ресурс)?
- Trajectory sampling: лучше сэмплировать сразу траекториями по реальной стратегии. Так мы не будем тратить время на недостижимые или очень-редко-достижимые состояния.
- Например, RTDP (real-time dynamic programming):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



- 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. :)



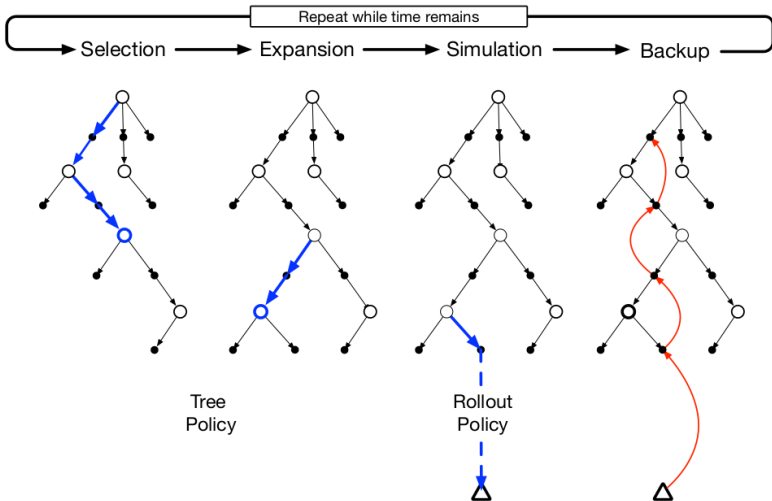
- Heuristic search: делаем поиск в ширину на несколько ходов вперёд, выбирая наилучшее действие (кстати, почему это называется «минимакс», если тут только max?)

- 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 нарда, и получалось очень хорошо, прямо неожиданно хорошо.
- Но ещё лучше совместить одно и другое и получить...

- ...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.

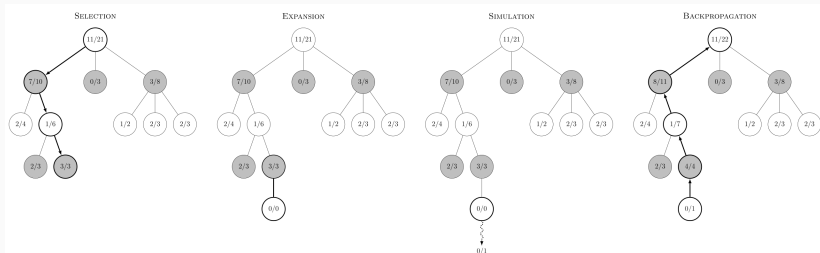
ПЛАНИРОВАНИЕ В ТЕКУЩЕМ МОМЕНТЕ

- Вот такая картинка получается:



ПЛАНИРОВАНИЕ В ТЕКУЩЕМ МОМЕНТЕ

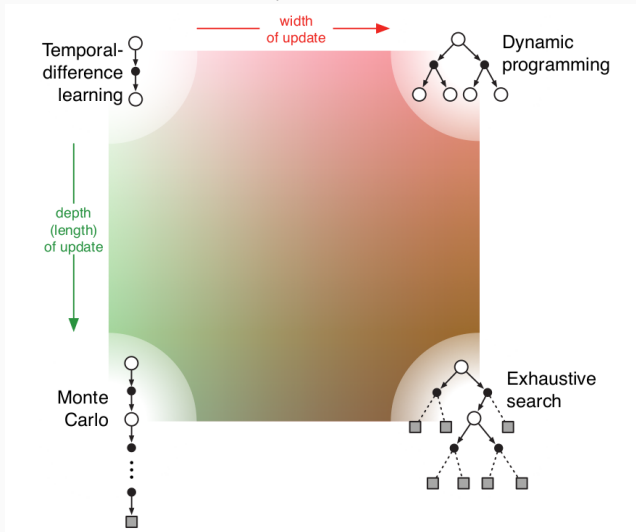
- Выбирать листья и действия нужно по статистике побед/поражений:



- 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:



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?..

- ...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).

- 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 $Q(s, a)$.

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

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

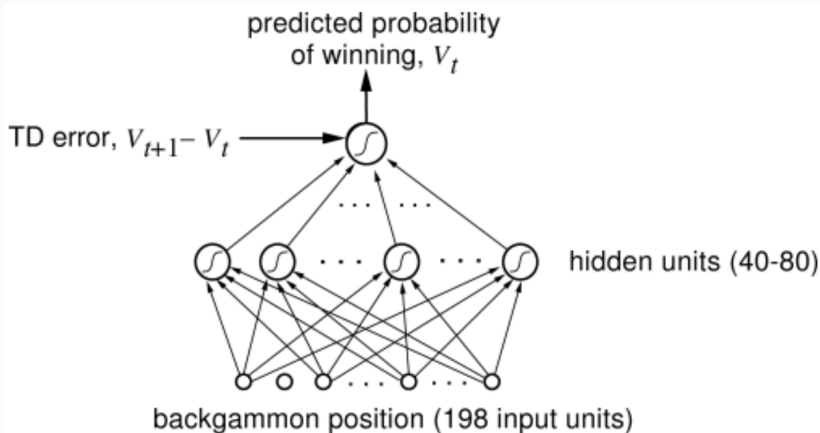
- semi-gradient off-policy TD(0):

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

- 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

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

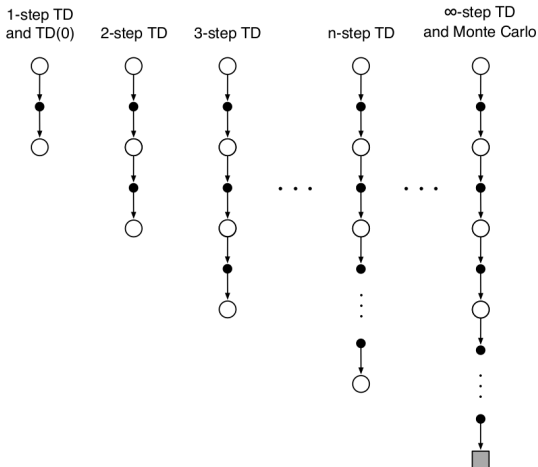
$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+1})$$

$$G_{t:t+3} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V_{t+2}(S_{t+3})$$

$$\dots = \dots$$

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

- 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:



- И теперь появляются n -step варианты всех алгоритмов.

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

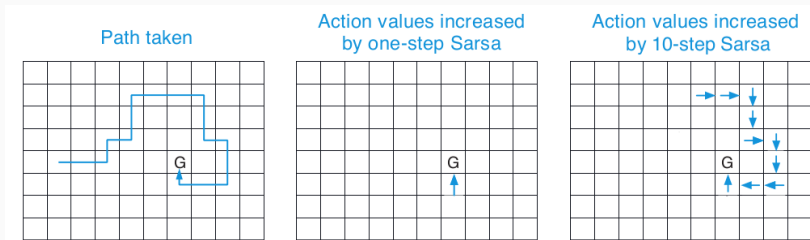
инициализировать случайно $Q(s, a)$; повторять до сходимости:

- 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 $G := G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$;
 - $Q(S_\tau, A_\tau) := Q(S_\tau, A_\tau) + \alpha (G - Q(S_\tau, A_\tau))$
 - 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:

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

- 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:

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

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

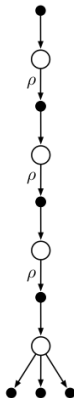
4-step
Sarsa



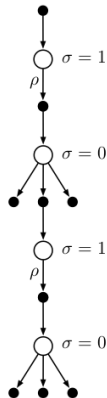
4-step
Tree backup



4-step
Expected Sarsa



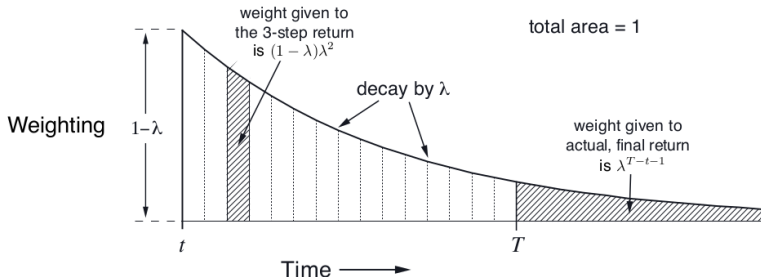
4-step
 $Q(\sigma)$



- 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 обнулится, когда мы получили ненулевую TD-ошибку.
- Параметр λ в $TD(\lambda)$ отвечает как раз за затухание этого следа.

- 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 :

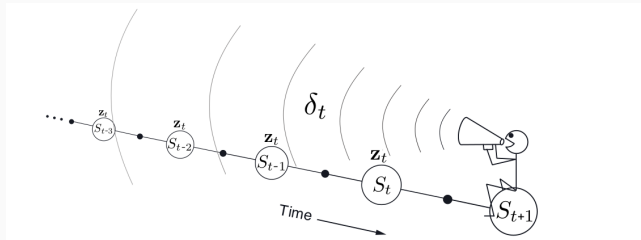
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left(G_t^\lambda - \hat{V}(S_t, \mathbf{w}_t) \right) \nabla_{\mathbf{w}} \hat{V}(S_t, \mathbf{w}_t).$$



- 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' ;
 - $\mathbf{z} := \gamma \lambda \mathbf{z} + \nabla_{\mathbf{w}} \hat{V}(S, \mathbf{w})$;
 - $\mathbf{w} := \mathbf{w} + \alpha (R + \gamma \hat{V}(S', \mathbf{w}) - \hat{V}(S, \mathbf{w})) \mathbf{z}$; и переходим $S := S'$.
- Обновляем по eligibility traces в прошлом:



- 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)...

- 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:

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

где веса $\mu_{\pi}(s) = \lim_{h \rightarrow \infty} p(S_t = s \mid \pi)$.

- 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

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

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

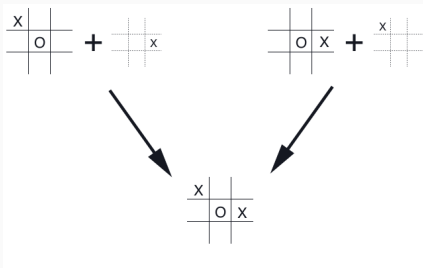
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left(R_{t+1} - \bar{R}_t + \hat{Q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{Q}(S_t, A_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{Q}(S_t, A_t, \mathbf{w})$$

где $\bar{R}_t$ — это оценка средней награды во время t , т.е. её можно просто обновлять как

$$\bar{R} := \bar{R} + \beta \left(R_{t+1} - \bar{R}_t + \hat{Q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{Q}(S_t, A_t, \mathbf{w}) \right).$$

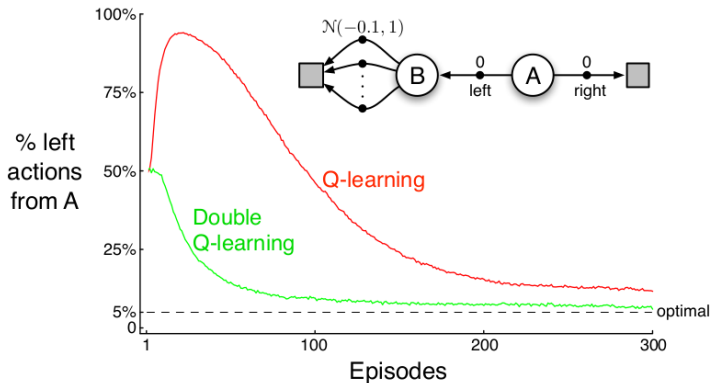
- Здесь самое интересное, что на самом деле между discounted reward и average reward разницы в среднем нет:
 - рассмотрим бесконечную последовательность вознаграждений;
 - тогда одно и то же вознаграждение R_t один раз появится в момент $t - 1$ без γ , один раз в момент $t - 2$ с коэффициентом γ , потом 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.

- 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.



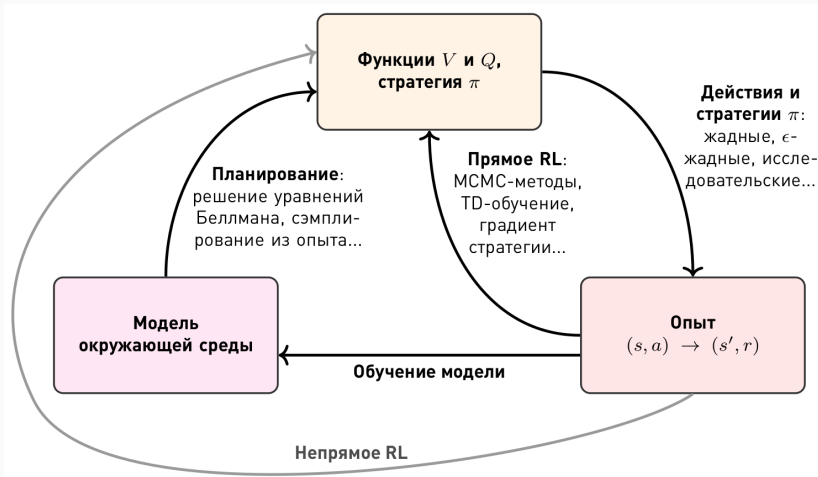
- Чтобы это поправить, можно использовать двойное обучение:
 - поддерживаем две стратегии Q_1 и Q_2 ;
 - каждый раз бросаем монетку, выбираем, какую брать стратегию для обновления, а какую для цели:

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

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

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

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

- награды за каждый эпизод, начиная со времени t :

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

- функцию значения для состояний и пар состояние-действие:

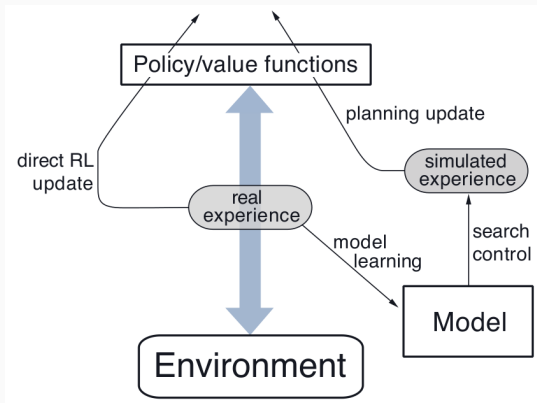
$$V_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right],$$

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

и их оптимальные варианты V_*, Q_* ;

- 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' ;
 - $Q(S, A) := Q(S, A) + \alpha (R + \gamma \max_a Q(S', a) - Q(S, A))$;
 - 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 $M(S, A)$;
 - $Q(S, A) := Q(S, A) + \alpha (R + \gamma \max_a Q(S', a) - Q(S, A))$.
- Dyna-Q случайно обновляет k значений $Q(s, a)$ по имеющейся модели; k можно выбирать из соображений имеющихся ресурсов.

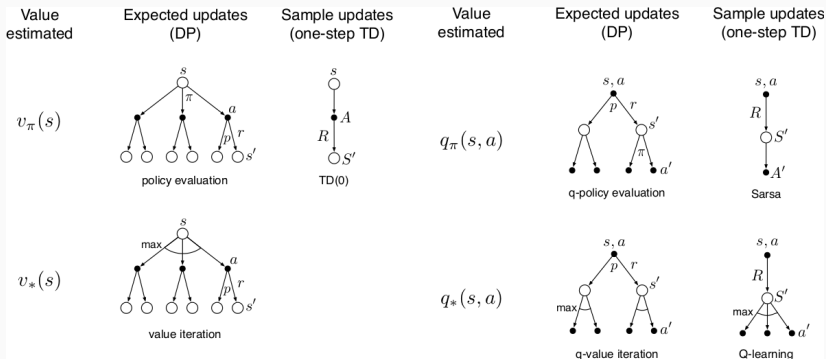
- Конечно, ещё лучше будет обновлять не случайно.

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 $(S, A) := \text{Head}(PQ)$, породить R, S' по $M(S, A)$;
 - $Q(S, A) := Q(S, A) + \alpha (R + \gamma \max_a Q(S', a) - Q(S, A))$.
 - 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 оценки вероятностей попасть в S .

EXPECTED VS. SAMPLE UPDATES

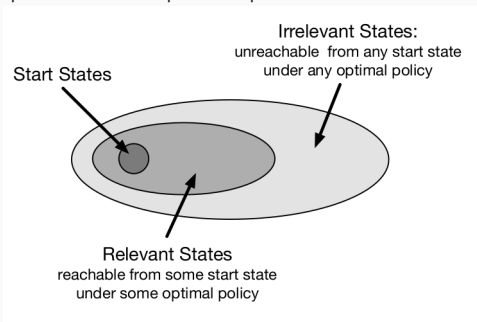
- Ещё один взгляд – expected updates vs. sample updates:



- 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

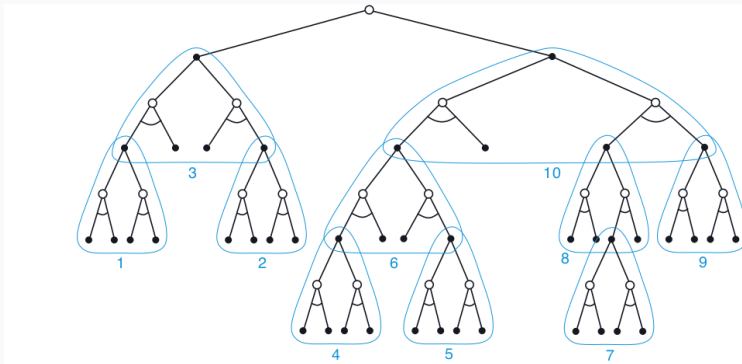
- Как лучше распределить апдейты (т.е. фактически имеющийся вычислительный ресурс)?
- Trajectory sampling: лучше сэмплировать сразу траекториями по реальной стратегии. Так мы не будем тратить время на недостижимые или очень-редко-достижимые состояния.
- Например, RTDP (real-time dynamic programming):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



- 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. :)



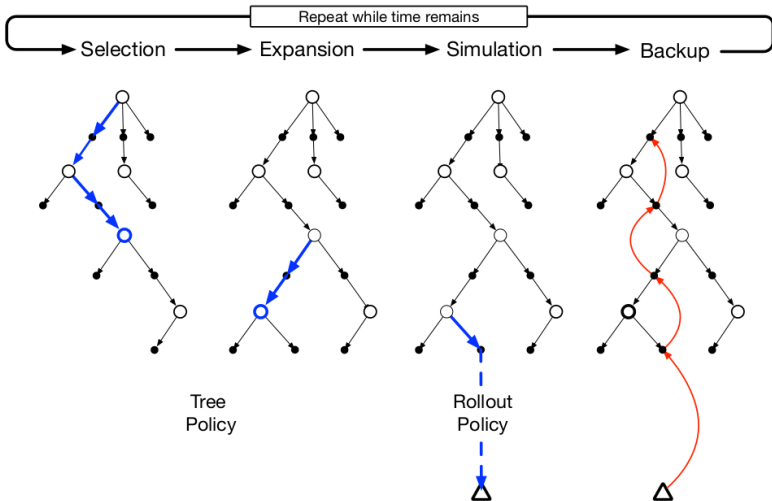
- Heuristic search: делаем поиск в ширину на несколько ходов вперёд, выбирая наилучшее действие (кстати, почему это называется «минимакс», если тут только max?)

- 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 нарда, и получалось очень хорошо, прямо неожиданно хорошо.
- Но ещё лучше совместить одно и другое и получить...

- ...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.

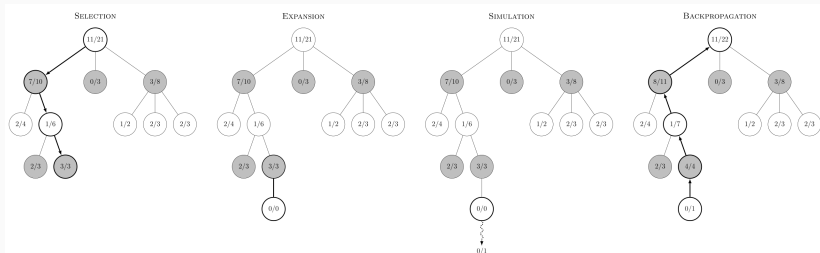
ПЛАНИРОВАНИЕ В ТЕКУЩЕМ МОМЕНТЕ

- Вот такая картинка получается:



ПЛАНИРОВАНИЕ В ТЕКУЩЕМ МОМЕНТЕ

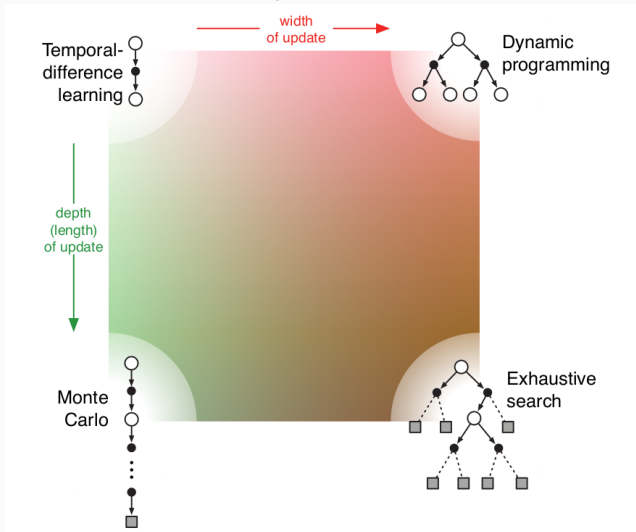
- Выбирать листья и действия нужно по статистике побед/поражений:



- 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:



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?

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

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

- 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 .

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

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

- 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 theorem:

$$\begin{aligned}\nabla V_{\pi}(s) &= \nabla \left(\sum_a \pi(a|s) Q_{\pi}(s, a) \right) = \\ &= \dots = \\ &= \sum_{x \in S} \sum_{k=0}^{\infty} p(\text{перейти из } s \text{ в } x \text{ за } k \text{ шагов по } \pi) \sum_a \nabla \pi(a|x) Q_{\pi}(x, a),\end{aligned}$$

то есть

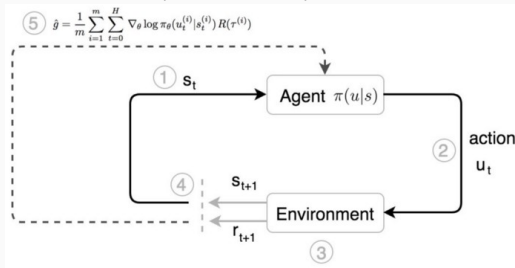
$$\nabla J(\theta) \propto \nabla V_{\pi}(s) = \sum_s \mu(s) \sum_a \nabla \pi(a|s) Q_{\pi}(s, a),$$

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

- Иначе говоря,

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

и теперь можно и алгоритмы построить.



- Можно all-actions алгоритм:

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

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

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

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

- И алгоритм такой:

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

- 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 $\pi(a|s, \theta)$

Algorithm parameter: step size $\alpha > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

 Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \theta)$

 Loop for each step of the episode $t = 0, 1, \dots, T-1$:

$$\begin{aligned} G &\leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \\ \theta &\leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t | S_t, \theta) \end{aligned} \tag{G_t}$$

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

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

и это не повлияет ни на что, мы вычитаем

$$b(s) \nabla \sum_a \pi(a|s, \theta) = b(s) \nabla 1 = 0.$$

- 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 $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Algorithm parameters: step sizes $\alpha^{\theta} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

Generate an episode $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \theta)$

Loop for each step of the episode $t = 0, 1, \dots, T-1$:

$$G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \quad (G_t)$$

$$\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t, \mathbf{w})$$

$$\theta \leftarrow \theta + \alpha^{\theta} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \theta)$$

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

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

- Алгоритм получается такой:

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

Input: a differentiable policy parameterization $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Parameters: step sizes $\alpha^\theta > 0$, $\alpha^\mathbf{w} > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ 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):

$A \sim \pi(\cdot|S, \theta)$

 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 $\hat{v}(S', \mathbf{w}) \doteq 0$)

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^\mathbf{w} \delta \nabla \hat{v}(S, \mathbf{w})$

$\theta \leftarrow \theta + \alpha^\theta I \delta \nabla \ln \pi(A|S, \theta)$

$I \leftarrow \gamma I$

$S \leftarrow S'$

- 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 $\pi(a|s, \theta)$

Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$

Parameters: trace-decay rates $\lambda^{\theta} \in [0, 1]$, $\lambda^{\mathbf{w}} \in [0, 1]$; step sizes $\alpha^{\theta} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ 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)

$\mathbf{z}^{\theta} \leftarrow \mathbf{0}$ (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):

$A \sim \pi(\cdot|S, \theta)$

 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 $\hat{v}(S', \mathbf{w}) \doteq 0$)

$\mathbf{z}^{\mathbf{w}} \leftarrow \gamma \lambda^{\mathbf{w}} \mathbf{z}^{\mathbf{w}} + \nabla \hat{v}(S, \mathbf{w})$

$\mathbf{z}^{\theta} \leftarrow \gamma \lambda^{\theta} \mathbf{z}^{\theta} + I \nabla \ln \pi(A|S, \theta)$

$\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \mathbf{z}^{\mathbf{w}}$

$\theta \leftarrow \theta + \alpha^{\theta} \delta \mathbf{z}^{\theta}$

$I \leftarrow \gamma I$

$S \leftarrow S'$

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

$$\begin{aligned}\nabla J(\theta) &\propto \nabla_{\theta} \sum_s \mu_{\pi}(s) \sum_a \pi(a|s) Q_{\pi}(s, a) = \\ &= \sum_s \mu_{\pi}(s) \sum_a [\nabla_{\theta} \pi(a|s) Q_{\pi}(s, a) + \pi(a|s) \nabla_{\theta} Q_{\pi}(s, a)],\end{aligned}$$

- Теперь первое слагаемое можно оценить как обычно:

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

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

- 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'

$$\delta \leftarrow r + \gamma(s') \mathbf{v}^T \mathbf{x}_{s'} - \mathbf{v}^T \mathbf{x}_s$$

$$\rho \leftarrow \pi_{\mathbf{u}}(a|s) / b(a|s)$$

 Update the critic (GTD(λ) algorithm):

$$\mathbf{e}_v \leftarrow \rho (\mathbf{x}_s + \gamma(s) \lambda \mathbf{e}_v)$$

$$\mathbf{v} \leftarrow \mathbf{v} + \alpha_v [\delta \mathbf{e}_v - \gamma(s') (1 - \lambda) (\mathbf{w}^T \mathbf{e}_v) \mathbf{x}_s]$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_w [\delta \mathbf{e}_v - (\mathbf{w}^T \mathbf{x}_s) \mathbf{x}_s]$$

 Update the actor:

$$\mathbf{e}_u \leftarrow \rho \left[\frac{\nabla_{\mathbf{u}} \pi_{\mathbf{u}}(a|s)}{\pi_{\mathbf{u}}(a|s)} + \gamma(s) \lambda \mathbf{e}_u \right]$$

$$\mathbf{u} \leftarrow \mathbf{u} + \alpha_u \delta \mathbf{e}_u$$

$$s \leftarrow s'$$

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

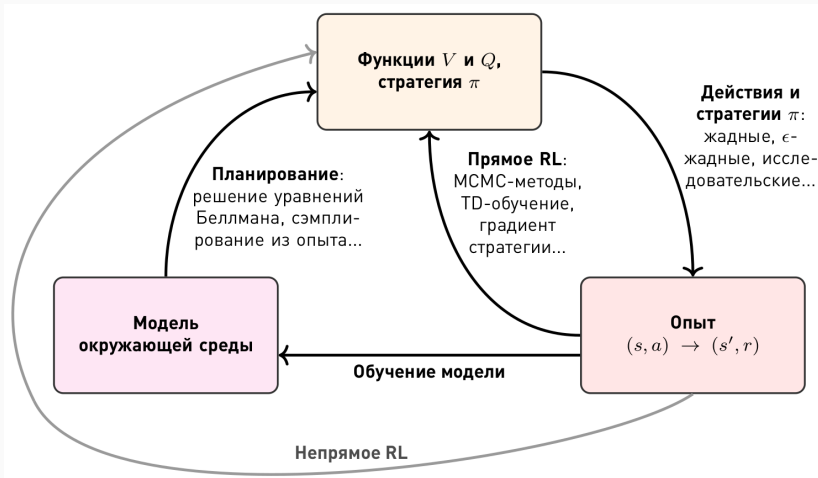
$$\pi(a|s, \theta) = \frac{1}{\sigma(s, \theta)\sqrt{2\pi}} e^{-\frac{1}{2\sigma(s, \theta)^2} (a - \mu(s, \theta))^2}.$$

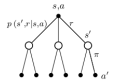
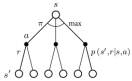
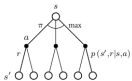
- Если, например,

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

то можно подсчитать градиенты:

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



Оценка стратегии π	Функция	Обновление в ожидании	Обновление по выборке
	$V_{\pi}(S_t) :=$	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) $V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$ 
Управление, поиск π_*	$Q_{\pi}(S_t, A_t) :=$	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 $Q(S_t, A_t) + \alpha (R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$ $Q(S_t, A_t) + \alpha (R_{t+1} + \gamma \sum_a \pi(a S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t))$  
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!

Спасибо за внимание!

