

Планирование и приближённые методы в RL

Сергей Николенко

Академия MADE – Mail.Ru

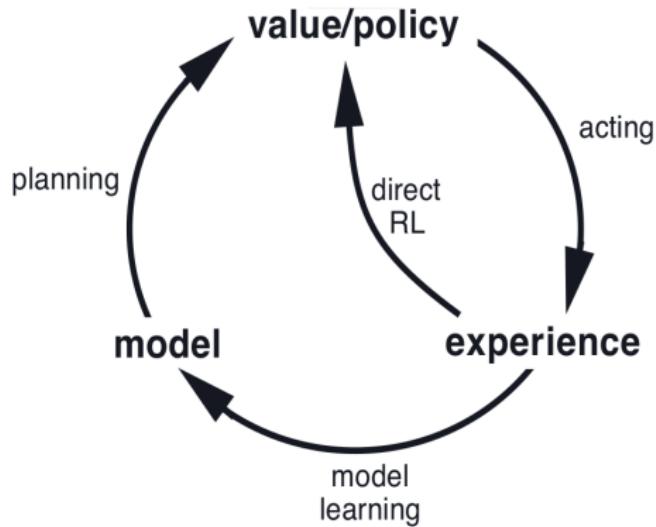
21 октября 2020 г.

Random facts:

- 21 октября в Великобритании — Apple Day; с 1990 года проводятся ярмарки с конкурсами вроде стрельбы из лука по яблокам или срезания самой длинной кожуры
- 21 октября 63 г. до н.э. избранный консулом Цицерон, получив сведения о намерениях Катилины совершить переворот, произнёс в сенате речь (то самое «Доколе же ты, Катилина, будешь злоупотреблять нашим терпением?») и планы Катилины сорвал
- 21 октября 1805 г. Пьер-Шарль Вильнёв, несмотря на возражения Антонио де Эсканью, выстроил свои корабли в линию; увидев эскадру Горацио Нельсона, Вильнёв приказал сделать поворот фордевинд, но не успел выстроиться в кильватерный строй, и англичане победили превосходящие силы противника; сам Нельсон погиб, его тело для сохранности поместили в бочку с ромом, и с тех пор выдаваемый на кораблях ром английские моряки называли «адмиральской кровью»
- 21 октября 1824 г. Джозеф Аспдин запатентовал портлендский цемент, 21 октября 1832 г. Павел Шиллинг в своей петербургской квартире впервые продемонстрировал изобретённый им электромагнитный телеграф, а 21 октября 1879 г. Томас Эдисон испытал свою первую лампу накаливания с угольной нитью

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:

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

- 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:

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

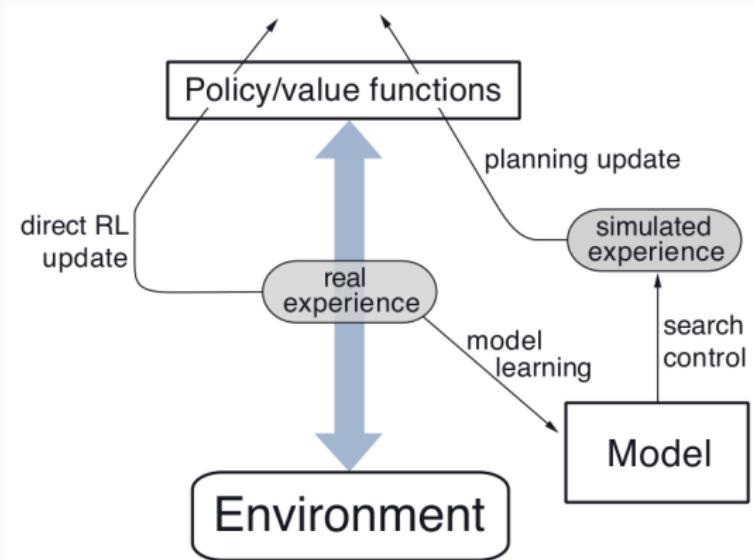
$$Q_\pi(s, a) = \mathbb{E}_\pi [G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]$$

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 , про наблюдать 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.

Планирование

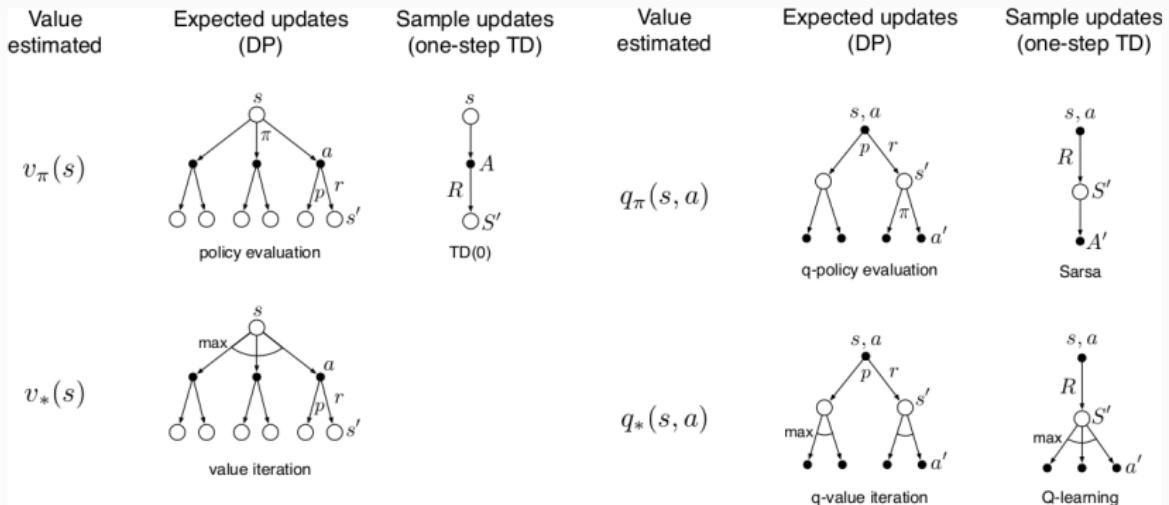
- Конечно, ещё лучше будет обновлять не случайно.

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 $(S, A) := \text{Head}(PQ)$, 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 оценками вероятностей попасть в S .

Expected vs. sample updates

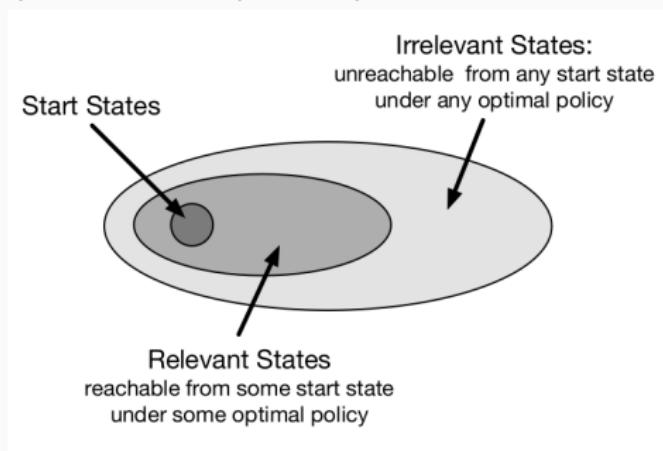
- Вот ещё один взгляд на то, чем мы занимались – expected updates vs. sample updates:



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

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.

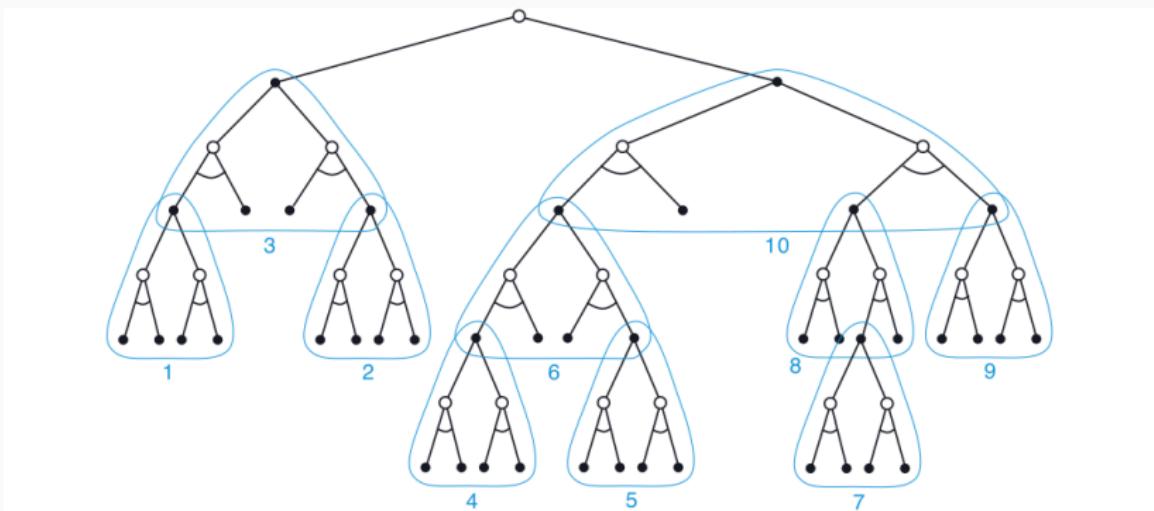


Планирование в текущем моменте

- 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

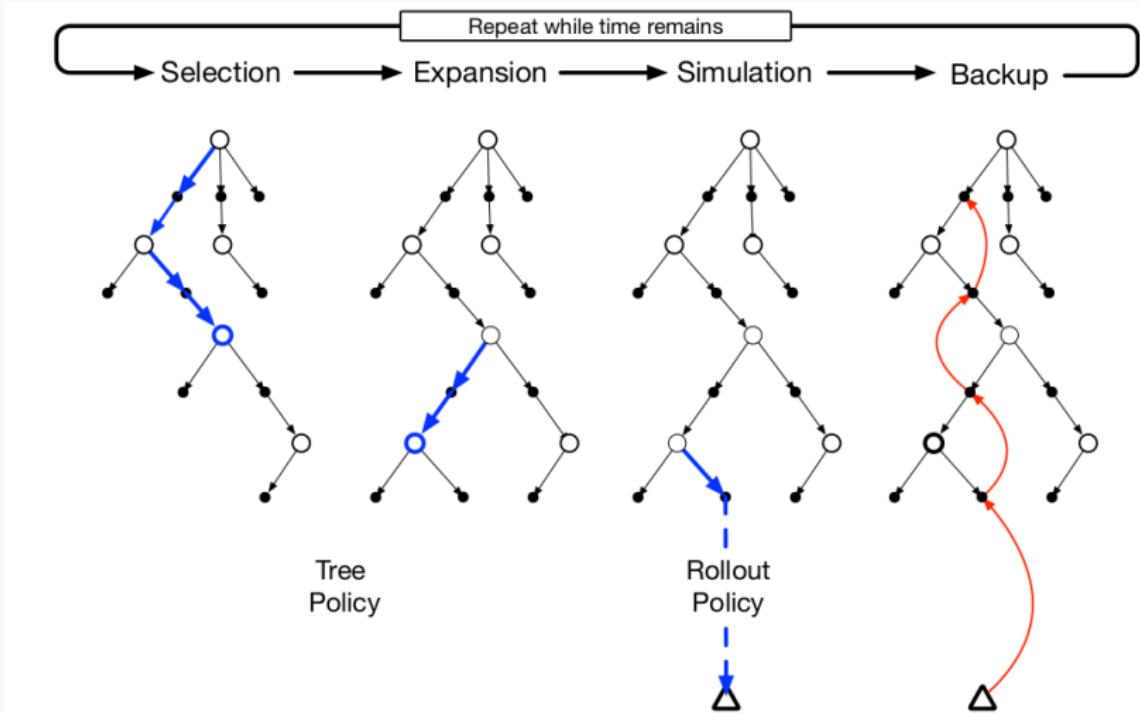
- Другой способ реализовать такое планирование: rollouts.
- Начиная с текущего состояния, симулируем несколько траекторий по текущей стратегии, а потом оцениваем действия, усредняя результаты этих симуляций.
- Когда оценки становятся достаточно точными, можно сделать ход, а потом опять строить rollouts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

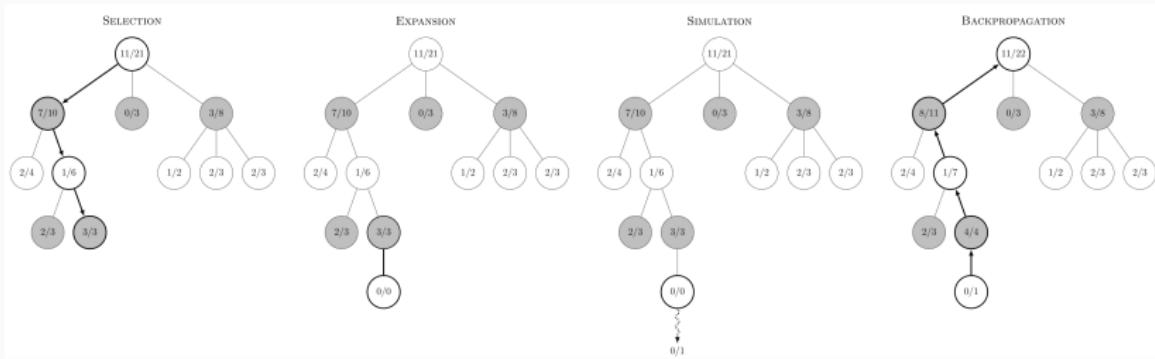
Планирование в текущем моменте

- Вот такая картинка получается:



Планирование в текущем моменте

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

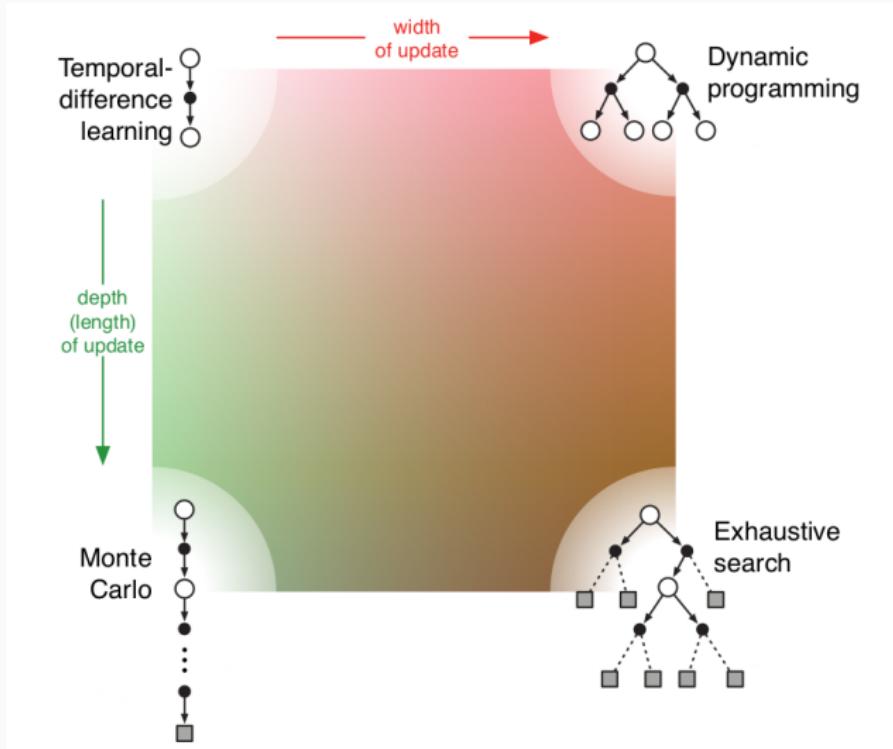


- Например, по методу UCT (upper confidence bounds applied to trees): выбираем для rollout действие, максимизирующее

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

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 $\hat{V}(s, w)$, которая будет приближать $V(s)$.
- Качество приближения оценим, например, как

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

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 $Q(s, a)$.

Градиентные методы в RL

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

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

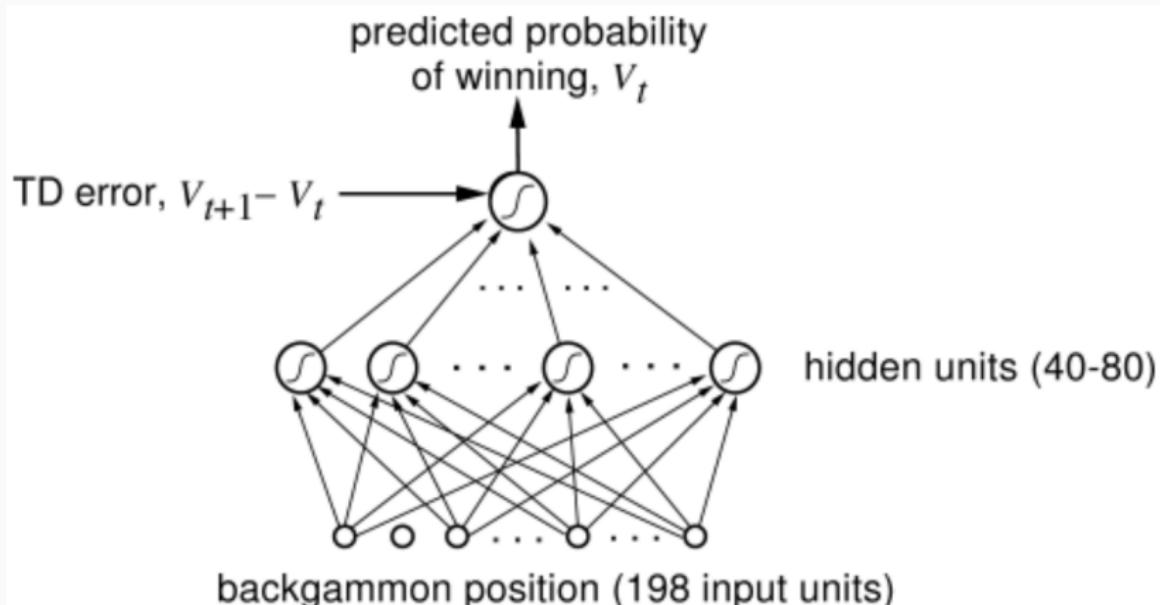
- semi-gradient off-policy TD(0):

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

- 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 :)



- К Deep RL мы и перейдём, но сначала ещё несколько небольших сюжетов.

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:

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

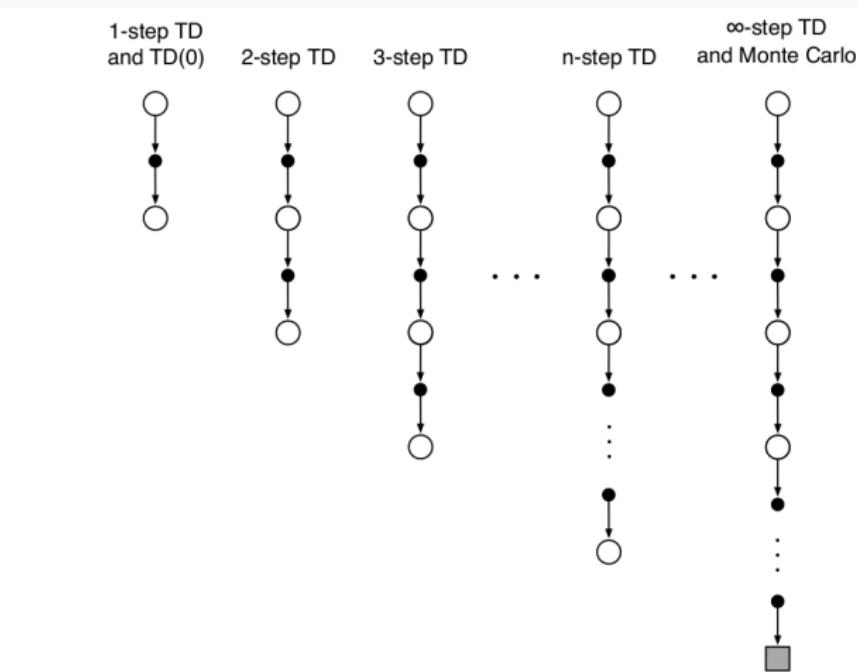
$$G_{t:t+3} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V_{t+2}(S_{t+3})$$

$\dots = \dots$

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

n -step алгоритмы

- На бесконечности, кстати говоря, это всё сливаются 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.

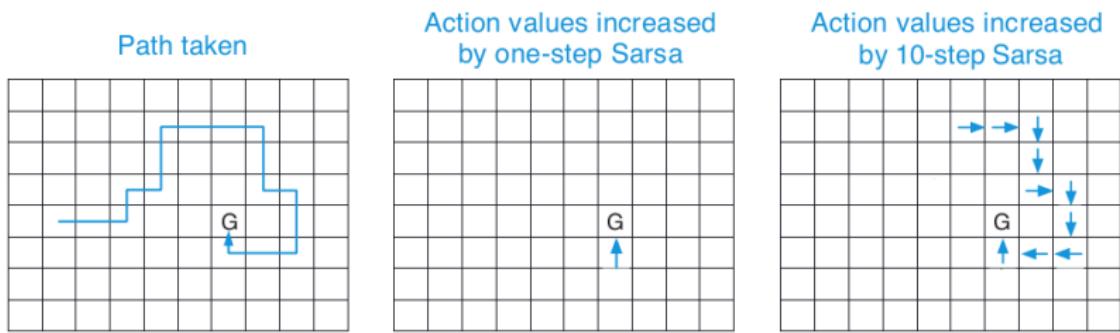
Алгоритм *n*-step Sarsa (on-policy TD control):

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

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

n-step алгоритмы

- 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:

$$G_{t:t+1} = R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_t(S_{t+1}, a),$$

$$G_{t:t+2} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a)$$

$$+ \gamma \pi(A_{t+1}|S_{t+1}) (R_{t+2} + \gamma \sum_a \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a)),$$

...

...

$$G_{t:t+n} = R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a)$$

$$+ \gamma \pi(A_{t+1}|S_{t+1}) G_{t+1:t+n}.$$

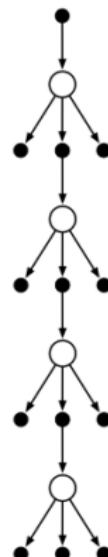
n -step алгоритмы

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

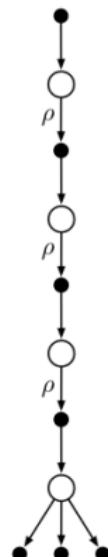
4-step
Sarsa



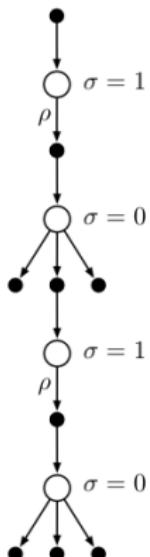
4-step
Tree backup



4-step
Expected Sarsa



4-step
 $Q(\sigma)$



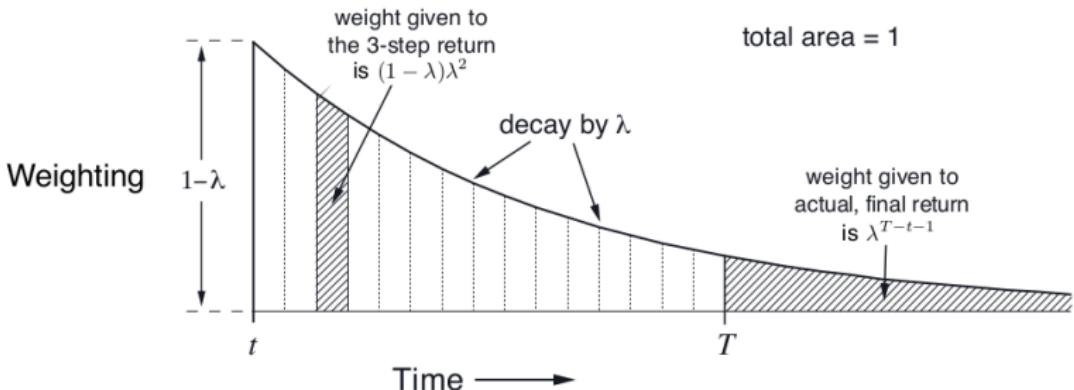
Eligibility traces

- И есть 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 $z_t \in \mathbb{R}^d$, который соответствует тому, насколько недавно в результате участвовали компоненты $w_t \in \mathbb{R}^d$;
 - тогда мы будем обучать этот компонент 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 :

$$w_{t+1} = w_t + \alpha \left(G_t^\lambda - \hat{V}(S_t, w_t) \right) \nabla_w \hat{V}(S_t, w_t).$$

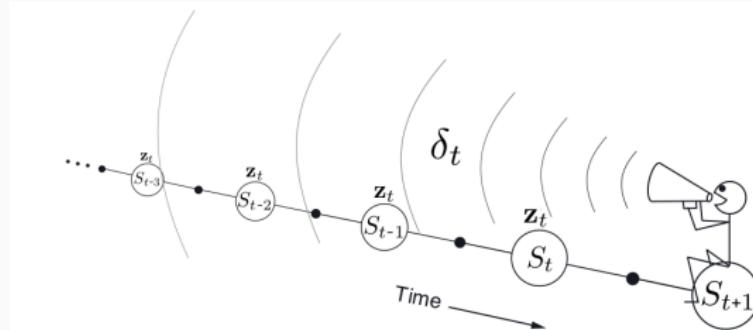


Eligibility traces

- Например, TD-обучение для оценки V_π .

Алгоритм Semi-gradient TD(λ):

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

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

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

- 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

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

- 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

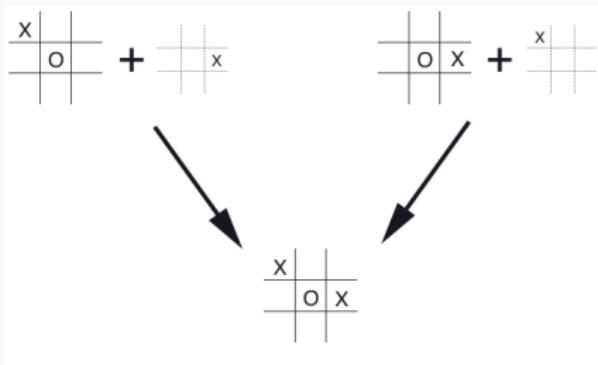
$$\bar{R} := \bar{R} + \beta \left(R_{t+1} - \bar{R}_t + \hat{Q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{Q}(S_t, A_t, \mathbf{w}) \right).$$

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.

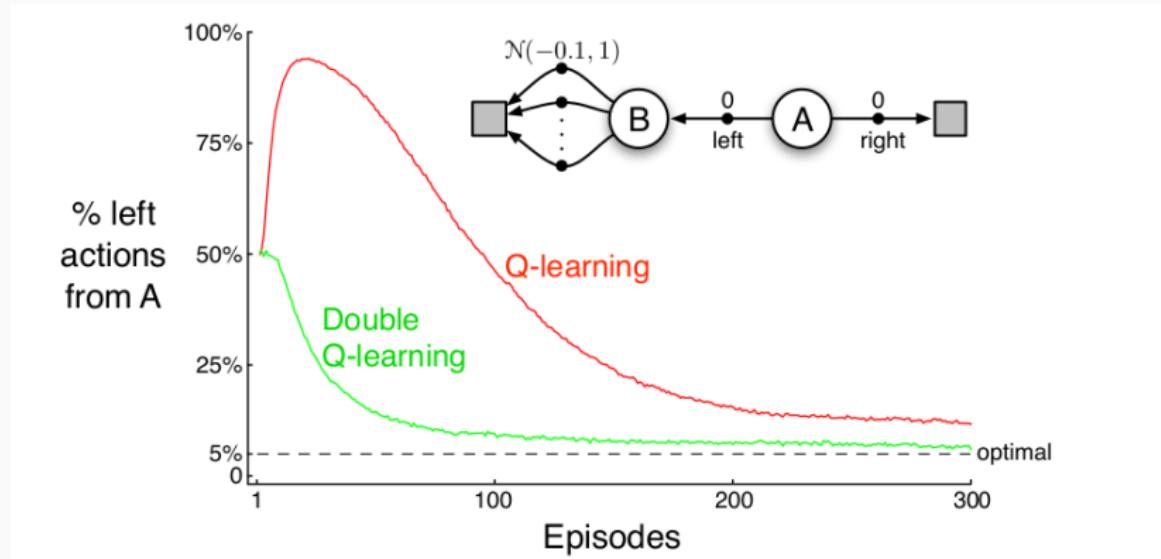
Afterstates

- 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

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

Deep Q-Learning

Deep Q-Learning

- Революция deep learning пришла в RL около 2013 года
- (Mnih et al., 2013): команда DeepMind взялась решать игры Atari:



- Идея, конечно, ровно та же: давайте приближать функцию Q при помощи нейронной сети.
- Но есть тонкости...

Deep Q-Learning

- Experience replay: давайте хранить опыт $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ в памяти (replay memory), а потом доставать оттуда мини-батчи для апдейта.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory \mathcal{D} to capacity N

Initialize action-value function Q with random weights

for episode = 1, M **do**

 Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

 With probability ϵ select a random action a_t

 otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$

 Execute action a_t in emulator and observe reward r_t and image x_{t+1}

 Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

 Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in \mathcal{D}

 Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from \mathcal{D}

 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

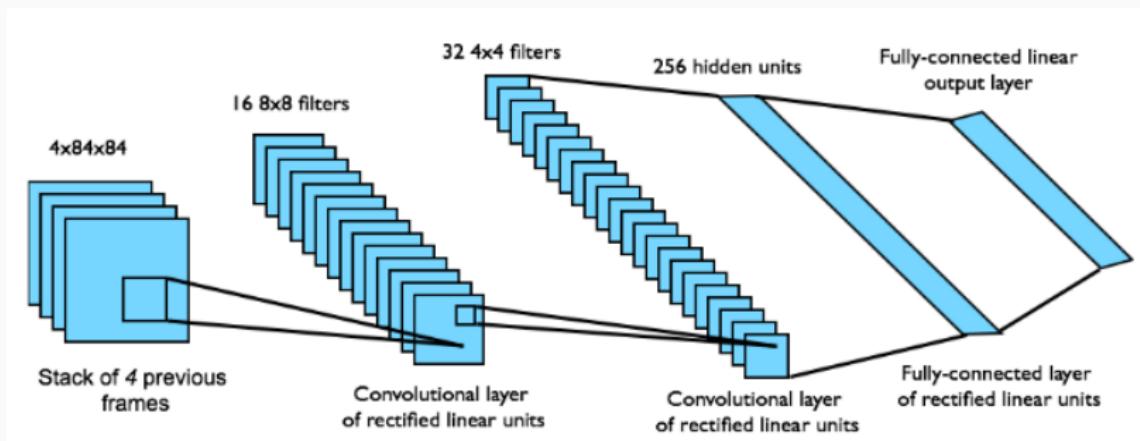
 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

end for

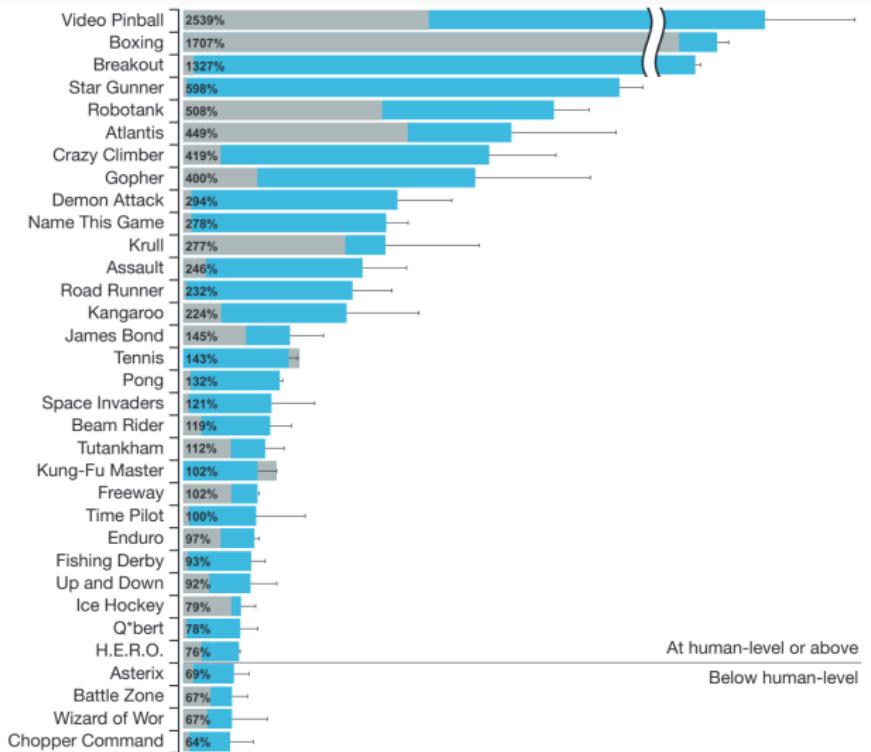
Deep Q-Learning

- А на вход даём просто картинки с экрана, сжатые до 84×84 пикселей (с 210×160), с небольшой историей в 4 кадра.
- Кроме того, можно пропускать кадры, брать, скажем, каждый четвёртый.
- Важно, что на выходе оценки всех действий, а не просто функция Q , иначе её пришлось бы считать много раз.



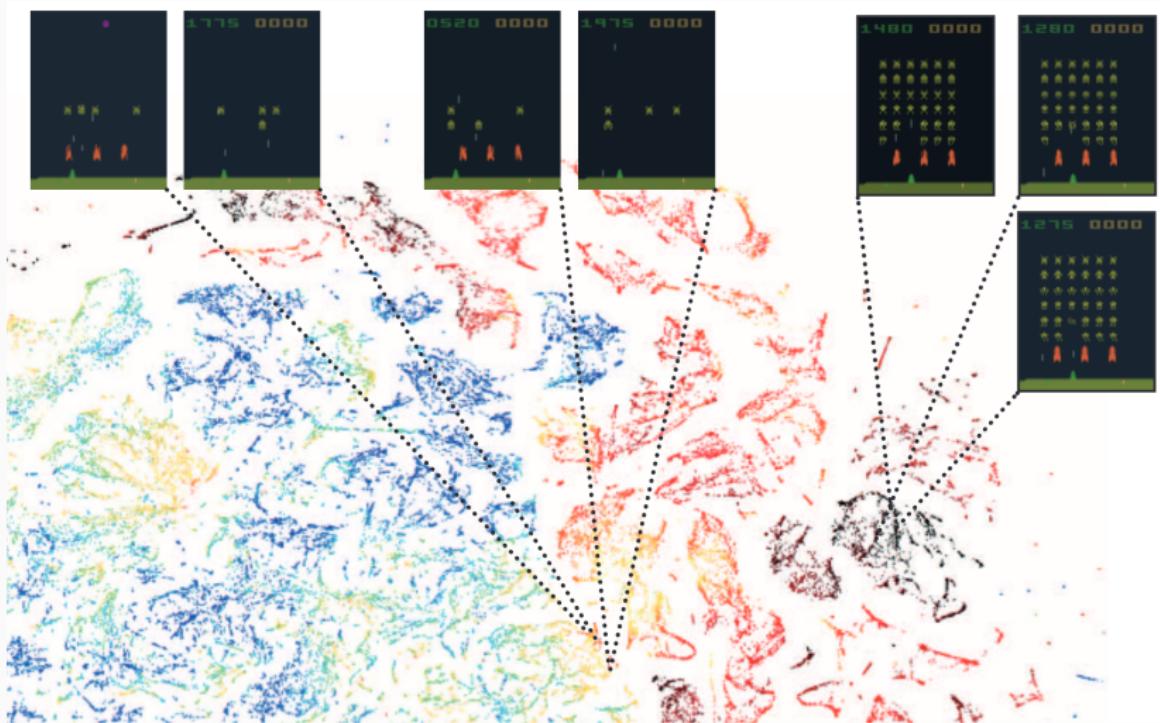
Deep Q-Learning

- Результаты (из статьи в Nature 2015):



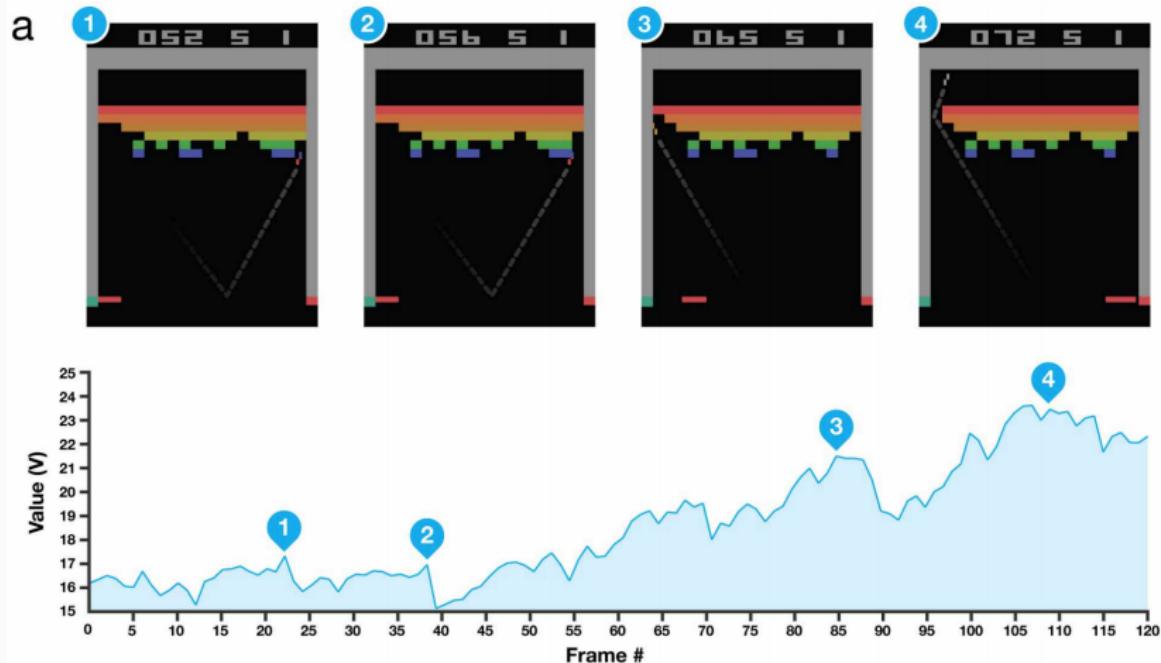
Deep Q-Learning

- Признаки на последнем слое дают разумные представления:



Deep Q-Learning

- А вот как меняется оценка состояния со временем:



- В DQN тоже разумно использовать Double DQN: давайте вместо $\max_a Q(S', a)$ брать

$$\gamma Q(S', \arg \max_a Q(S', a; w); w^-),$$

где w^- – это параметры другой сети (target network).

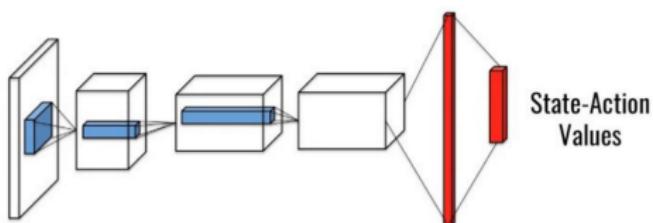
- Повтор по приоритетам: не просто experience replay, а приоритизированный тем, насколько для нас это было неожиданно; кладём в очередь по значениям ошибки

$$r + \gamma \max_a Q(s', a'; w^-) - Q(s, a; w).$$

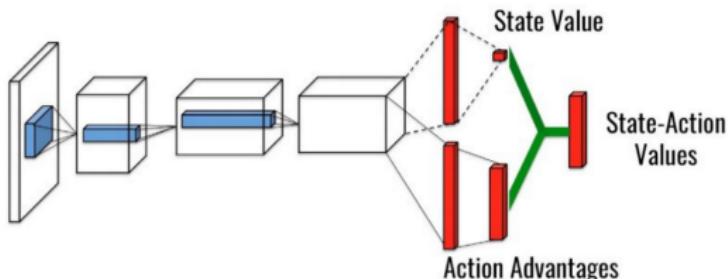
Deep Q-Learning

- Позже появился трюк с *dueling network* – разделим Q-сеть на два канала, value function $V(s; w)$ и зависящую от действия advantage function $A(s, a; w)$:

$$Q(s, a; w) = V(s; w) + A(s, a; w).$$



Standard
Q-network



Dueling
Q-network

Спасибо!

Спасибо за внимание!