

DQN: от Atari до AlphaZero и дальше

Сергей Николенко

Академия MADE – Mail.Ru

28 октября 2020 г.

Random facts:

- 28 октября – Международный день анимации; 28 октября 1892 г. художник и изобретатель Эмиль Рейно продемонстрировал свои «светящиеся пантомимы» в зале «Кабинета фантастики» музея Гревен
- 28 октября в Греции – День Охи; 28 октября 1940 года греческому правительству под угрозой объявления войны было предъявлено требование позволить итальянским войскам войти на территорию Греции и занять «стратегические позиции» (порты, аэроромы и т.д.); ответ был краток: «όχι», то есть «нет»
- 28 октября 1726 г. вышли в свет «Путешествия в некоторые отдалённые страны мира в четырёх частях: сочинение Лемюэля Гулливера, сначала хирурга, а затем капитана нескольких кораблей»
- 28 октября 1995 г. произошёл пожар в бакинском метро, крупнейший по числу жертв инцидент в истории всех метрополитенов мира (286 погибших)
- 28 октября 2007 г. в СПбГУ ИТМО завершилась первая Вики-конференция русской Википедии

Развитие DQN: методы и трюки

Наш план

- Наша главная цель сегодня – поговорить об AlphaGo, AlphaZero и MuZero.
- Но прежде чем мы туда двинемся, нужно обсудить и DQN вообще, те трюки, которые используются в этой части Deep RL.
- Фактически сегодня будет рекламная лекция о достижениях DeepMind, но что уж тут полделаешь...
- Пойдём по порядку; сначала даже давайте повторим одну важную мысль из предыдущей лекции.

Double DQN

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

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

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

Double DQN

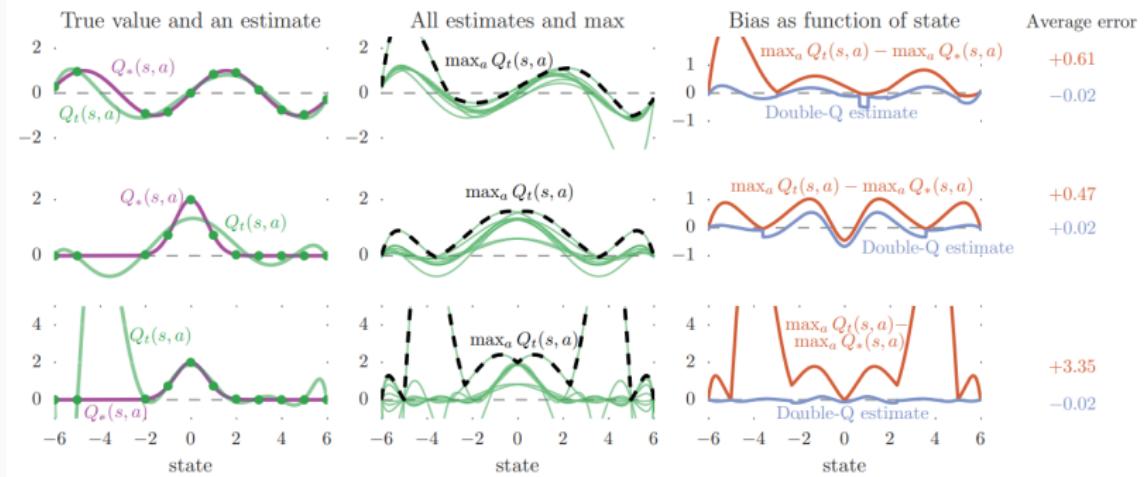
- Поэтому ван Хассельт предложил использовать Double Q-learning: давайте возьмём два разных независимых набора сэмплов для X_i , и будем использовать один из них в тот момент, когда делаем апдейт Q-обучения для другого.
- Тогда (при независимых оценках) всё будет сходиться.

Algorithm 1 Double Q-learning

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

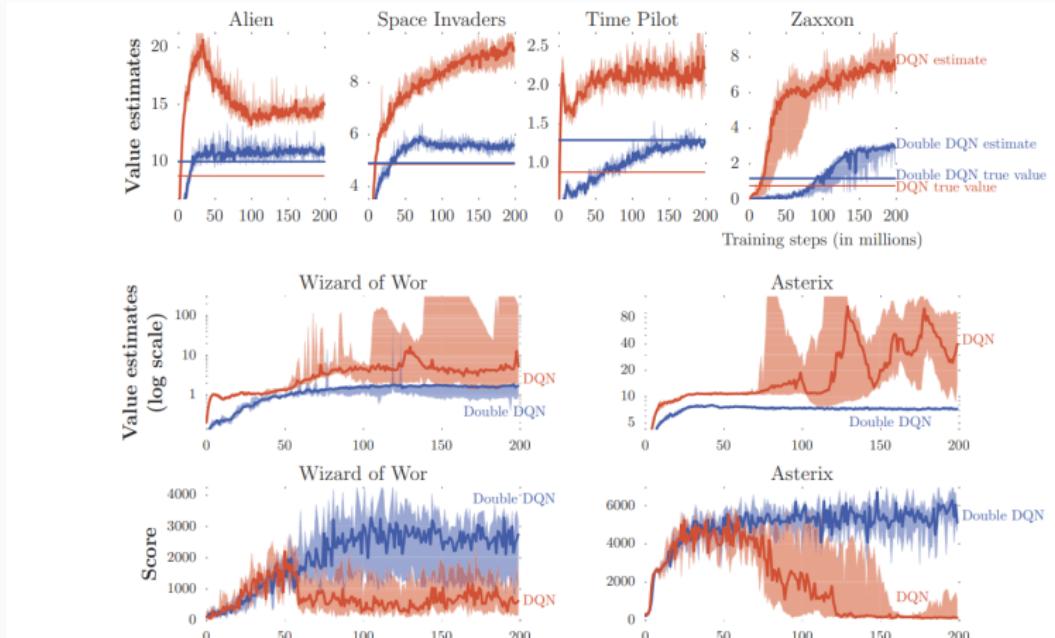
Double DQN

- (van Hasselt et al., 2016): в DQN тоже разумно использовать Double DQN
- Для приближённого Q-обучения с нейросетями тоже возникает ровно тот же эффект:



Double DQN

- Поэтому давайте вместо $\gamma \max_a Q(S', a)$ брать $\gamma Q(S', \arg \max_a Q(S', a; w); w^-)$, где w^- – это параметры другой сети (target network).
- И это действительно помогает:

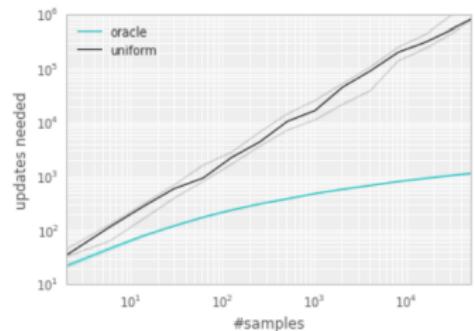
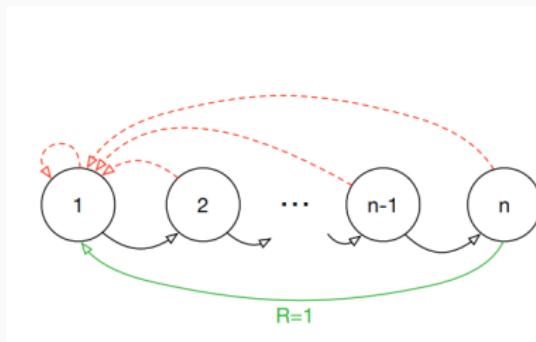


Prioritized experience replay

- Мы уже говорили о том, что просто обучать DQN по последовательным единицам опыта (s, a, r, s') — плохая идея по двум причинам:
 - слишком сильно скоррелированы соседние состояния;
 - слишком быстро мы будем забывать редкий, но важный опыт из прошлого.
- Поэтому нужно сохранять опыт в специальной памяти (replay memory), а потом брать из неё батчи для обучения.
- Но это ещё не всё...

Prioritized experience replay

- Давайте рассмотрим показательный пример:



- Чтобы случайно дойти до награды, нужно экспоненциально много шагов.
- Справа – сколько нужно шагов обучения в зависимости от объема памяти; что это за оракул такой?

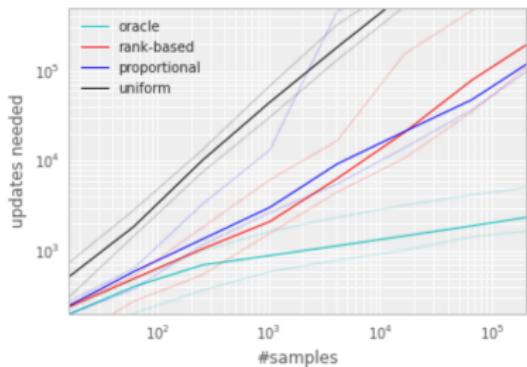
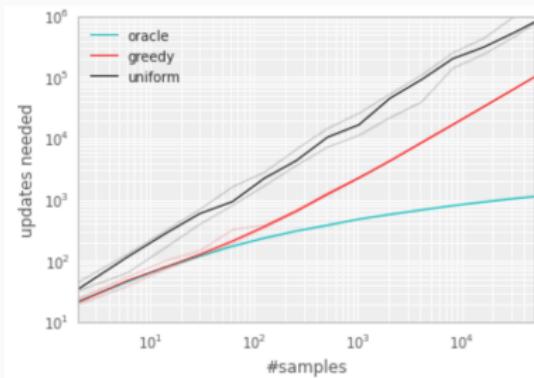
Prioritized experience replay

- Оракул выбирает для обучения переходы, которые максимально улучшают глобальную ошибку после апдейта; и получает экспоненциальный выигрыш.
- Конечно, на самом деле такого оракула у нас не будет, но можно пытаться приближаться к этому.
- *Повтор по приоритетам:* не просто experience replay, а приоритизированный тем, насколько для нас это было неожиданно; мы кладём новые единицы опыта в очередь по значениям ошибки

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

Prioritized experience replay

- Как обычно, просто жадно это делать не надо (слишком чувствительно к шуму, плюс никогда не вернёмся к переходам с поначалу низкой ошибкой).
- Лучше сэмплировать, или пропорционально ошибке, или пропорционально $\frac{1}{\text{rank}}$:



Prioritized experience replay

- Итого получается вот такой алгоритм (Schaul et al., 2016):

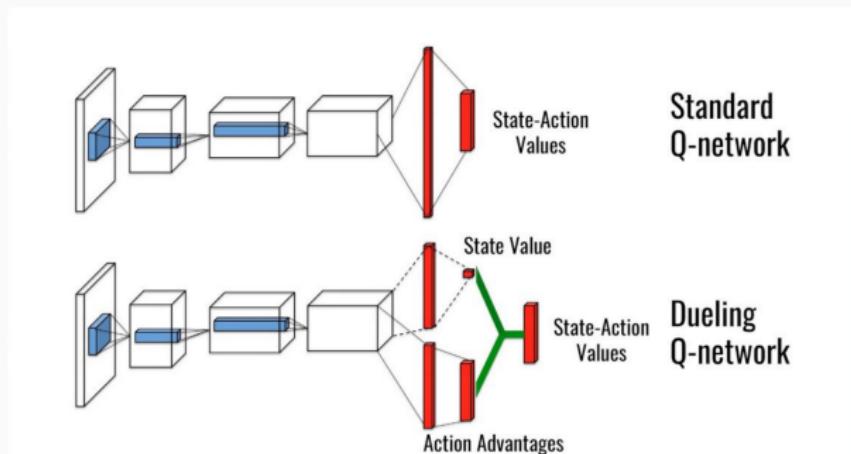
Algorithm 1 Double DQN with proportional prioritization

```
1: Input: minibatch  $k$ , step-size  $\eta$ , replay period  $K$  and size  $N$ , exponents  $\alpha$  and  $\beta$ , budget  $T$ .
2: Initialize replay memory  $\mathcal{H} = \emptyset$ ,  $\Delta = 0$ ,  $p_1 = 1$ 
3: Observe  $S_0$  and choose  $A_0 \sim \pi_\theta(S_0)$ 
4: for  $t = 1$  to  $T$  do
5:   Observe  $S_t, R_t, \gamma_t$ 
6:   Store transition  $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$  in  $\mathcal{H}$  with maximal priority  $p_t = \max_{i < t} p_i$ 
7:   if  $t \equiv 0 \pmod K$  then
8:     for  $j = 1$  to  $k$  do
9:       Sample transition  $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$ 
10:      Compute importance-sampling weight  $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$ 
11:      Compute TD-error  $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg \max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$ 
12:      Update transition priority  $p_j \leftarrow |\delta_j|$ 
13:      Accumulate weight-change  $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$ 
14:    end for
15:    Update weights  $\theta \leftarrow \theta + \eta \cdot \Delta$ , reset  $\Delta = 0$ 
16:    From time to time copy weights into target network  $\theta_{\text{target}} \leftarrow \theta$ 
17:  end if
18:  Choose action  $A_t \sim \pi_\theta(S_t)$ 
19: end for
```

Dueling networks

- Следующий трюк – *dueling networks* (Wang et al. 2016)
- Мы много говорили о том, как встроить сети в Q-обучение, но, может быть, сами сети тоже надо как-то подправить?
- Идея такая – разделим Q-сеть на два канала, value function $V(s; w)$ и зависящую от действия advantage function $A(s, a; w)$:

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



Dueling networks

- Формально мы хотели бы получить что-то вроде

$$Q(s, a | \theta, \alpha, \beta) = V(s | \theta, \beta) + A(s, a | \theta, \alpha)$$

- Но так не получится, V вовсе не обязательно будет обучаться, из Q это не следует.
- Можно заставить сделать так, чтобы advantage у лучшего действия был нулевой:

$$Q(s, a | \theta, \alpha, \beta) = V(s | \theta, \beta) + \left(A(s, a | \theta, \alpha) - \max_{a' \in \mathcal{A}} A(s, a' | \theta, \alpha) \right)$$

- А на практике ещё лучше получается с

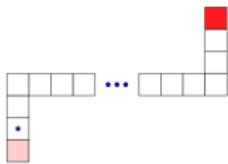
$$Q(s, a | \theta, \alpha, \beta) = V(s | \theta, \beta) + \left(A(s, a | \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s, a' | \theta, \alpha) \right)$$

Dueling networks

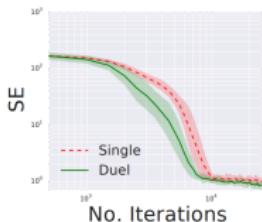
- Вот модельный пример, который показывает разницу; на картинке ошибка в policy evaluation по мере обучения:

$$SE = \sum_{s,a} (Q(s, a | \theta) - Q^\pi(s, a))^2$$

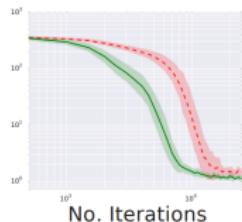
CORRIDOR ENVIRONMENT



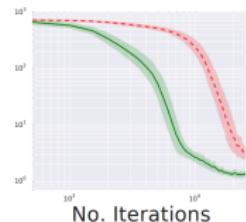
5 ACTIONS



10 ACTIONS

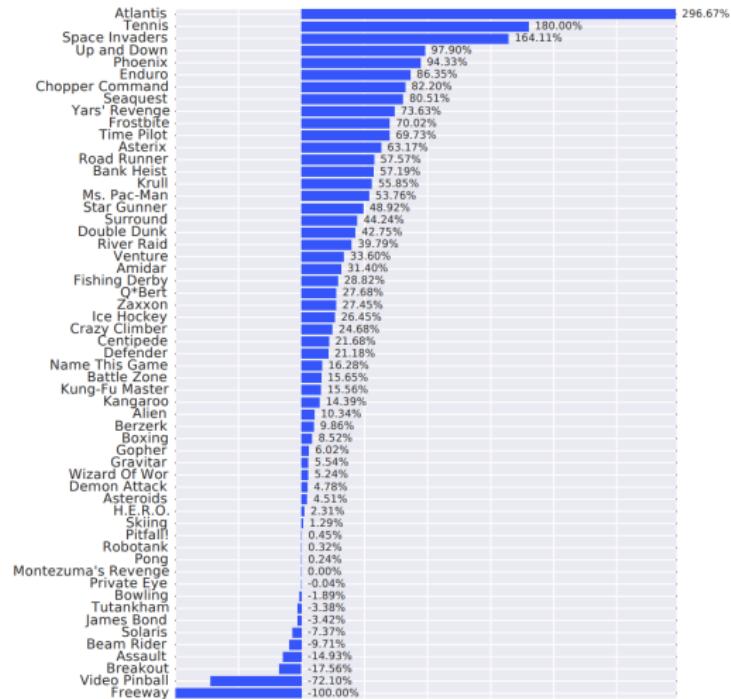


20 ACTIONS



Dueling networks

- Пример улучшений по сравнению с (van Hasselt et al., 2016)

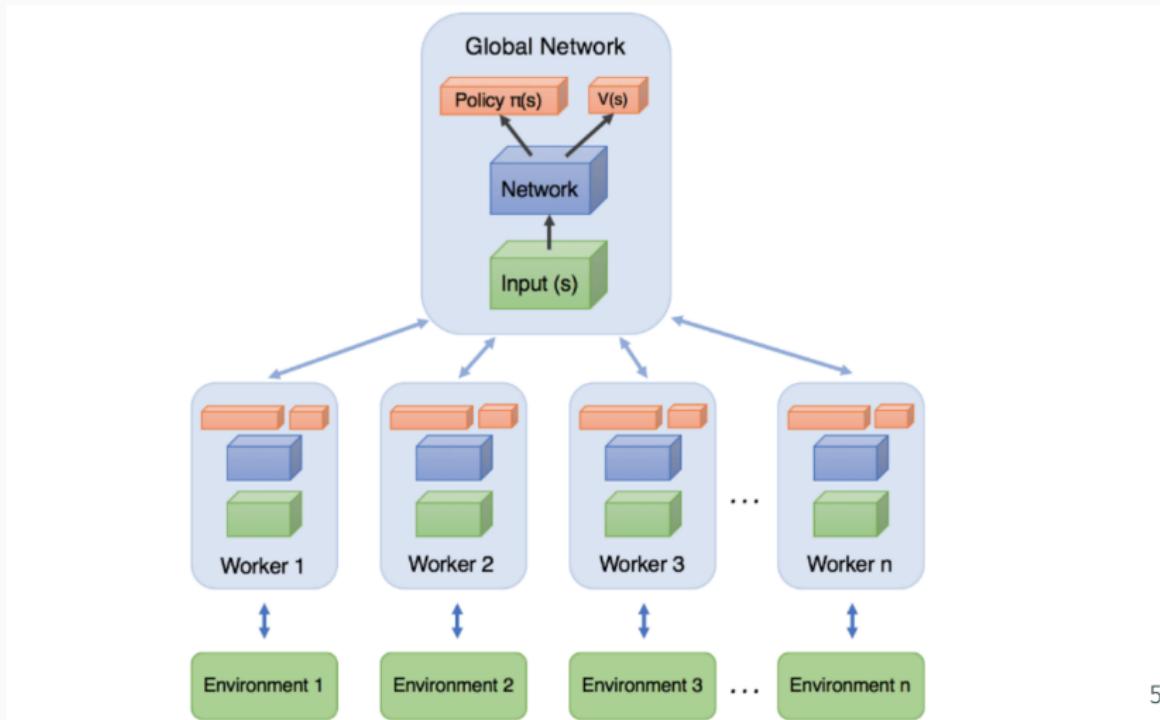


Как делать всё асинхронно

- Мы использовали experience replay, чтобы сэмплировать достаточно разнообразный опыт
- Но для этого нужно больше памяти и вычислений на каждом эпизоде
- Кроме того, это автоматически требует использовать только off-policy алгоритмы (мы обучаемся на какой-то предыдущей версии стратегии)

Как делать всё асинхронно

- (Mnih et al., 2016): вместо этого можно асинхронно запускать сразу много агентов, на разных копиях окружения



Как делать всё асинхронно

- Если формально:

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ , θ^- , and counter $T = 0$.
Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
 Take action a with ϵ -greedy policy based on $Q(s, a; \theta)$
 Receive new state s' and reward r
 $y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$
 Accumulate gradients wrt θ : $d\theta \leftarrow d\theta + \frac{\partial(y - Q(s, a; \theta))^2}{\partial \theta}$
 $s = s'$
 $T \leftarrow T + 1$ and $t \leftarrow t + 1$
 if $T \bmod I_{target} == 0$ **then**
 Update the target network $\theta^- \leftarrow \theta$
 end if
 if $t \bmod I_{AsyncUpdate} == 0$ or s is terminal **then**
 Perform asynchronous update of θ using $d\theta$.
 Clear gradients $d\theta \leftarrow 0$.
 end if
until $T > T_{max}$

Как делать всё асинхронно

- И так можно не только Q-обучение, но и on-policy алгоритмы обработать, например Sarsa
- Но особенно хорошо получилось с actor-critic алгоритмом, который мы ещё не обсуждали.
- Поговорим о нём в следующий раз, а пока просто замечу, что он поддерживает стратегию $\pi(a | s; \theta)$ (actor) и функцию $V(s | \theta_V)$ (critic).
- Обновления следуют из policy gradient (о нём как раз в следующий раз), но асинхронное обобщение то же самое: накапливаем градиенты и время от времени обновляем θ и θ_V .

Как делать всё асинхронно

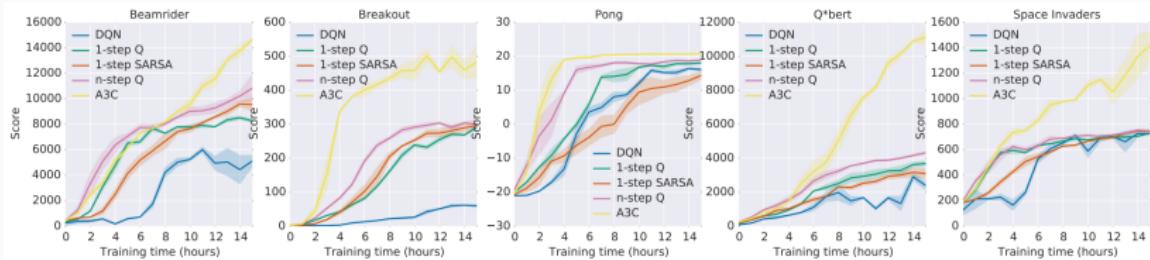
- Вот такой алгоритм:

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```
// Assume global shared parameter vectors  $\theta$  and  $\theta_v$  and global shared counter  $T = 0$ 
// Assume thread-specific parameter vectors  $\theta'$  and  $\theta'_v$ 
Initialize thread step counter  $t \leftarrow 1$ 
repeat
    Reset gradients:  $d\theta \leftarrow 0$  and  $d\theta_v \leftarrow 0$ .
    Synchronize thread-specific parameters  $\theta' = \theta$  and  $\theta'_v = \theta_v$ 
     $t_{start} = t$ 
    Get state  $s_t$ 
    repeat
        Perform  $a_t$  according to policy  $\pi(a_t|s_t; \theta')$ 
        Receive reward  $r_t$  and new state  $s_{t+1}$ 
         $t \leftarrow t + 1$ 
         $T \leftarrow T + 1$ 
    until terminal  $s_t$  or  $t - t_{start} == t_{max}$ 
     $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t // \text{Bootstrap from last state} \end{cases}$ 
    for  $i \in \{t - 1, \dots, t_{start}\}$  do
         $R \leftarrow r_i + \gamma R$ 
        Accumulate gradients wrt  $\theta'$ :  $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$ 
        Accumulate gradients wrt  $\theta'_v$ :  $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i; \theta'_v))^2 / \partial \theta'_v$ 
    end for
    Perform asynchronous update of  $\theta$  using  $d\theta$  and of  $\theta_v$  using  $d\theta_v$ .
until  $T > T_{max}$ 
```

Как делать всё асинхронно

- Результаты поражают:



Distributional Q-learning

- (Bellemare et al., 2017): а почему, собственно, мы всё время говорим только об обучении функции $V(s)$ или $Q(s, a)$?
- Что если обучать сразу *распределение* V или Q ?
- Про распределения тоже есть такое же уравнение Беллмана: случайная величина $Z(x, a)$, ожидание которой равно Q , распределена как

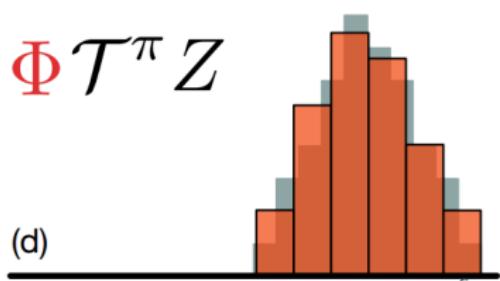
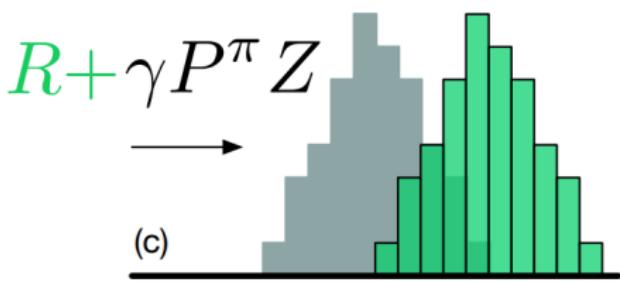
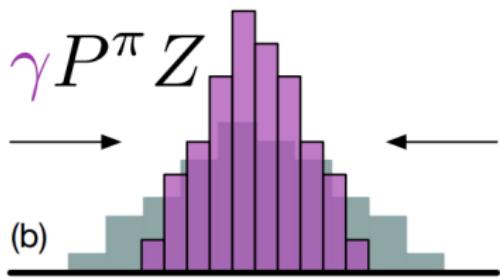
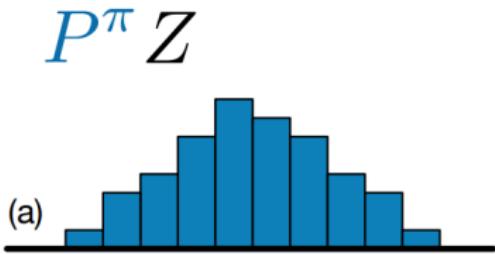
$$Z(x, a) = R(x, a) + \gamma Z(X', A'),$$

где (X', A') — случайные величины про следующее состояние.

- Это всё появилось очень давно, ещё в 1970е, но про это долго не вспоминали.

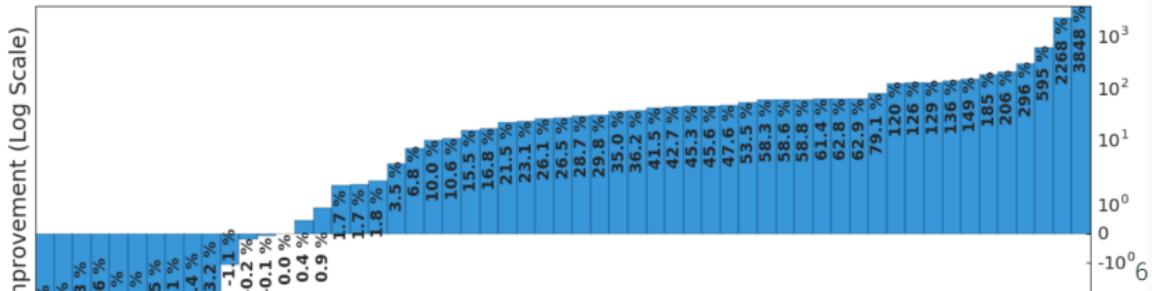
Distributional Q-learning

- Здесь всё непросто, подробно разбирать не будем. Смысл в том, что нужно определить оператор Беллмана, действующий на распределениях, и потом обучать параметры распределения:



Distributional Q-learning

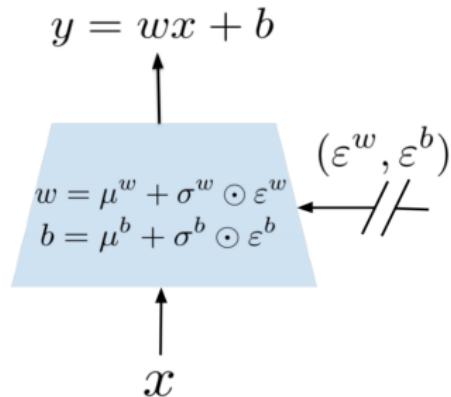
- Но в целом тоже всё становится лучше:



- И ещё одна интересная мысль:
 - нам часто нужно обеспечивать exploration;
 - в бандитах мы использовали какой-то вариант оптимизма при неопределённости;
 - но в DQN мы добавляли exploration пока что только в форме ϵ -мягких стратегий;
 - может быть, что-то другое можно придумать?

Noisy Networks

- (Fortunato et al., 2017): а давайте встроим exploration прямо в сеть!
- Т.е. рассмотрим слой нейронной сети, в котором веса задаются распределениями, а в реальности сэмплируются с шумом:

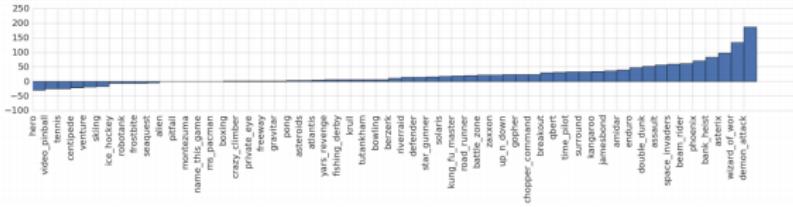


Noisy Networks

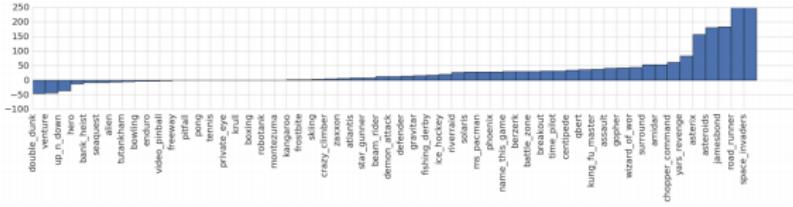
- Т.е. по сути это рандомизированная функция Q .
- Причём рандомизированная куда интереснее, чем просто ϵ -мягкая стратегия, с использованием сложных зависимостей внутри сети.
- Шум можно добавлять по-разному: можно просто гауссовский независимо к каждому весу, итого $n_{\text{in}}n_{\text{out}} + n_{\text{out}}$ для одного слоя.
- А можно в факторизованном виде: иметь n_{in} значений шума для входов и n_{out} для выходов, потом для соответствующего веса w_{ij} брать $f(\epsilon_i)f(\epsilon_j)$, где $f(x) = \text{sign}(x)\sqrt{|x|}$.

Noisy Networks

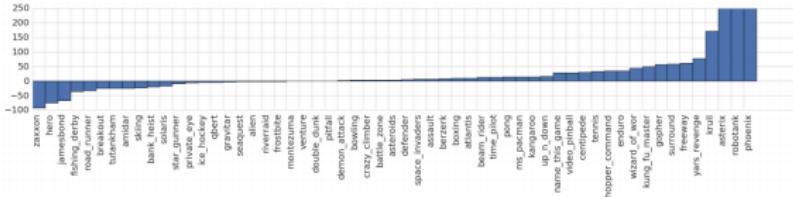
- Получается, опять же, лучше:



(a) Improvement in percentage of NoisyNet-DQN over DQN (Mnih et al., 2015)



(b) Improvement in percentage of NoisyNet-Dueling over Dueling (Wang et al., 2016)

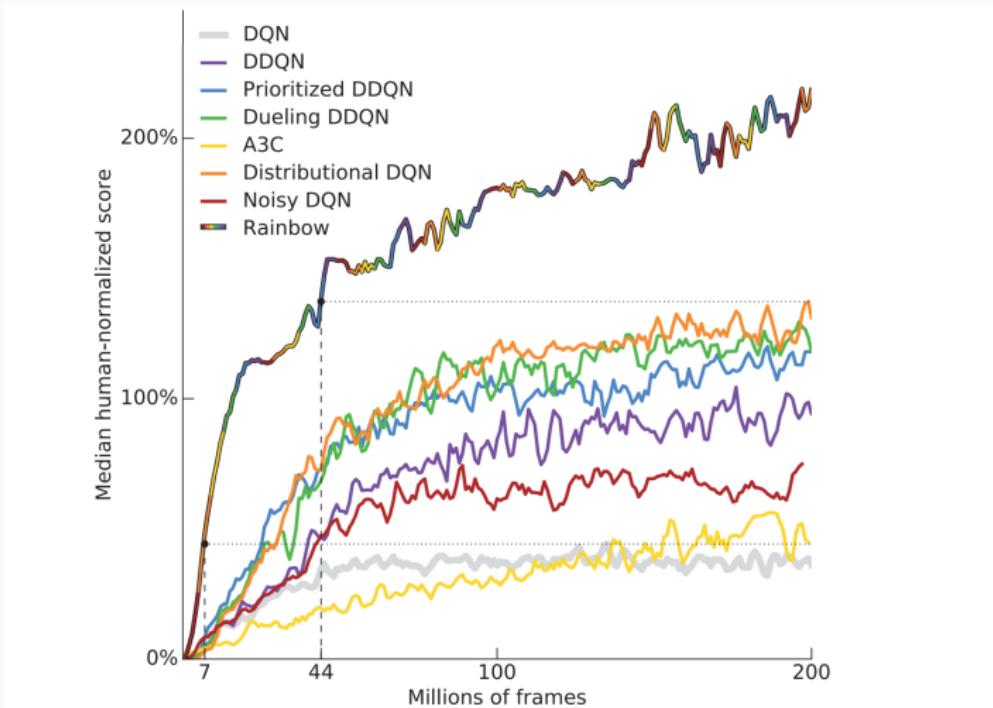


(c) Improvement in percentage of NoisyNet-A3C over A3C (Mnih et al., 2016)

- Rainbow (Hessel et al., 2018): а теперь давайте соберём всё вместе! А именно:
 - возьмём distributional loss, причём многошаговый вариант;
 - используем для оценки отдельную target network, как в Double DQN;
 - приоритеты для prioritized replay расставим соответственно KL-ошибке, которую оптимизируем;
 - в качестве архитектуры возьмём dueling network;
 - а все линейные слои заменим их зашумленными эквивалентами.

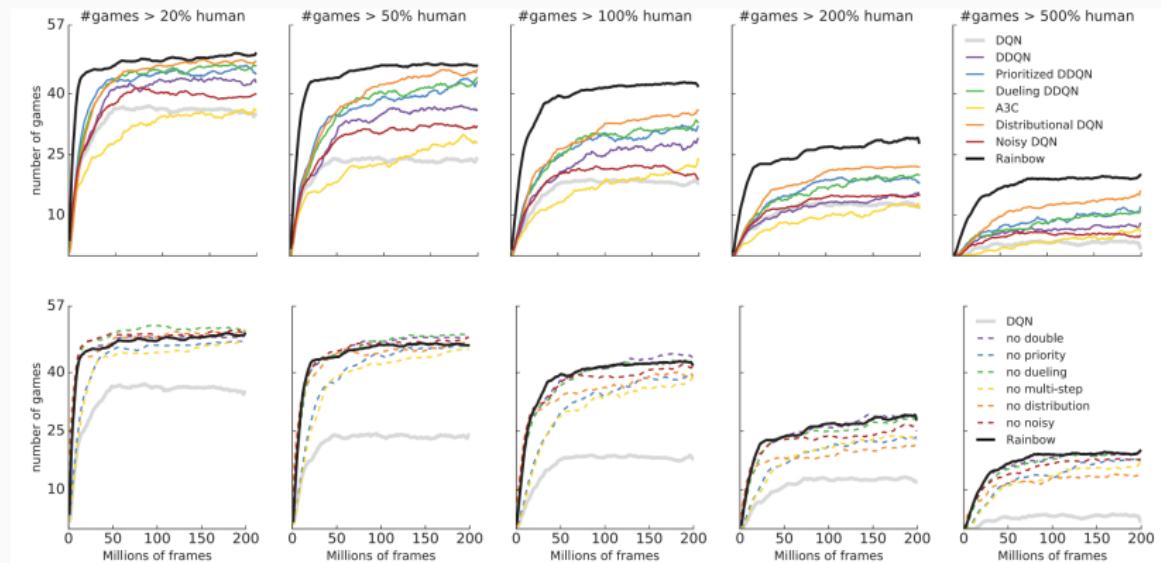
Собираем всё вместе

- И если всё правильно настроить, то получается круто:



Собираем всё вместе

- Везде преимущество:

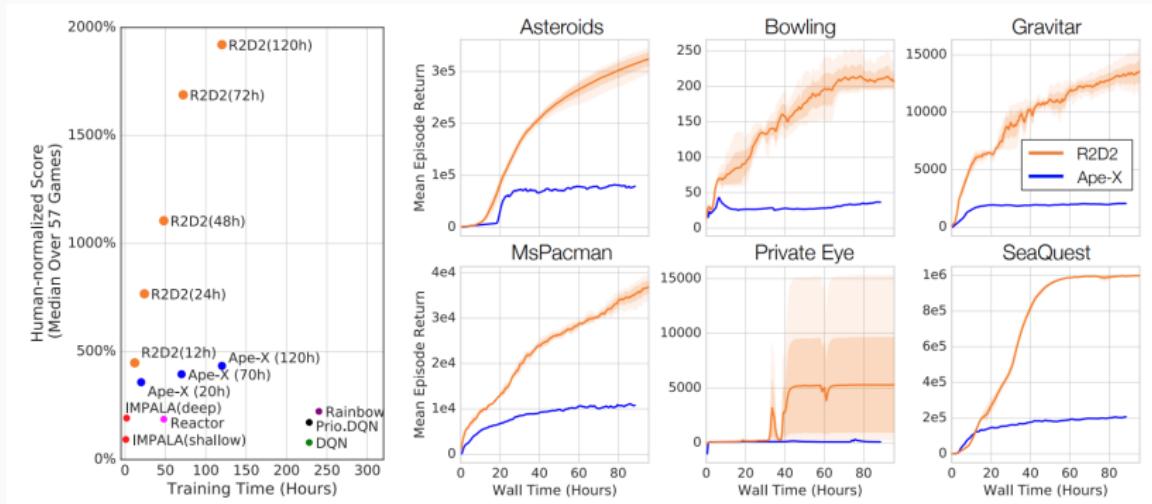


- R2D2 (Kapturowski et al., 2019): Recurrent Replay Distributed DQN:
 - используют prioritized distributed replay, Double DQN и dueling network;
 - но теперь есть ещё рекуррентная сеть, которая реализует как бы память о состоянии процесса;
 - формально это значит, что у нас теперь POMDP (partially observable MDP), иначе нет смысла в скрытом состоянии;
 - в памяти храним не отдельные (s, a, r, s') , а целые последовательности состояний, не выходящие за границы эпизодов.

- Как обучать рекуррентную сеть? Надо откуда-то взять скрытое состояние; два варианта:
 - можно хранить скрытое состояние в памяти и брать для replay;
 - а можно делать burn-in сети, чтобы она «вошла в тему».
- В целом получается лучше, если целые траектории переигрывать.

R2D2

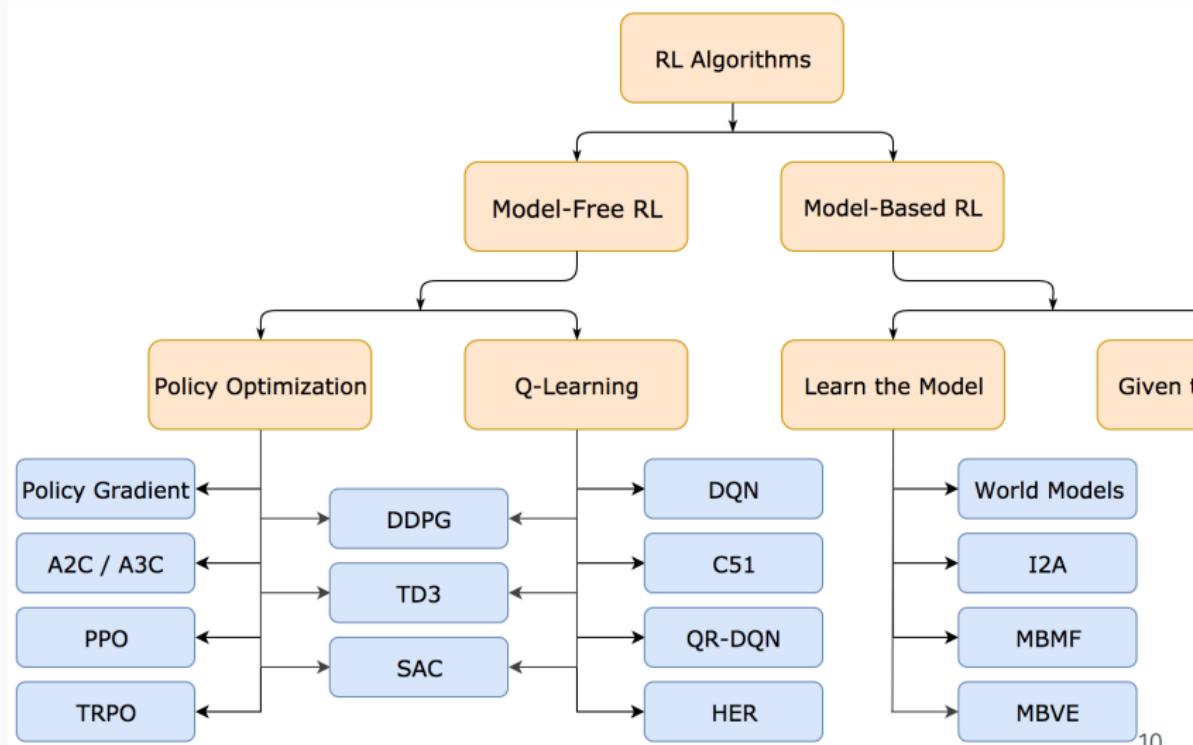
- На играх Atari метод R2D2 определял state of the art до самого MuZero:



- Это довольно любопытно, потому что, казалось бы, с несколькими кадрами на входе Atari почти полностью наблюдаема (для многих игр).
- Kapturowski et al. приходят к выводу, что LSTM не просто

Таксономия

- Это ещё далеко не всё, но нам пока что хватит...



AlphaGo

What happened

- 8-15 марта 2016 года прошёл исторический матч.
- AlphaGo (Google DeepMind) 4 : 1 Ли Седоль 9р (Корея).



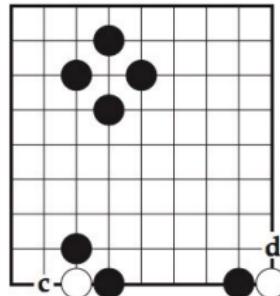
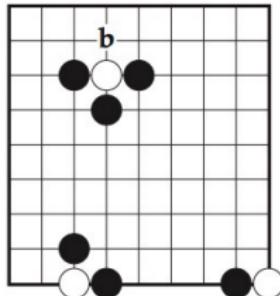
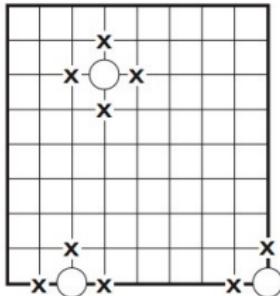
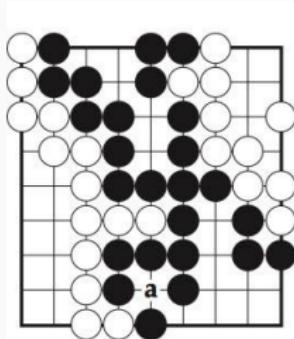
What happened

- 8-15 марта 2016 года прошёл исторический матч.
- AlphaGo (Google DeepMind) 4 : 1 Ли Седоль 9р (Корея).



- ...but why do we care?

Правила го

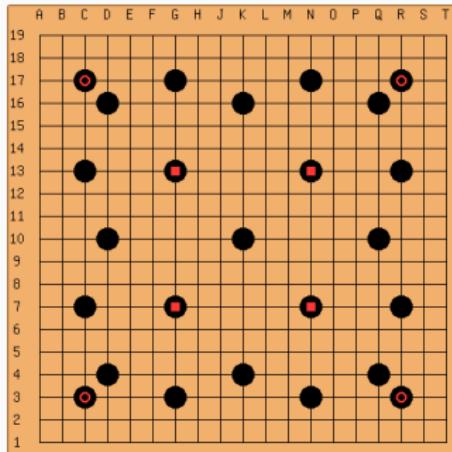


История компьютерного го

- 1968: Albert Zobrist – первая программа, влияние камней, Zobrist hashing.
- Середина 1980х: появились РС, интерес к этому.
- Ing Prize: 40M тайваньских долларов, если компьютер сможет до 2000 года обыграть профессионала 1 дана.
- Программы 1990х: Goliath (1989-1991), Go Intellect, Handtalk, Goemate.
- (David Fotland, 1993): Many Faces of Go, долгое время одна из ведущих программ.
- Они работали на pattern matching и базах знаний.
- И насколько же хорошо они играли?..

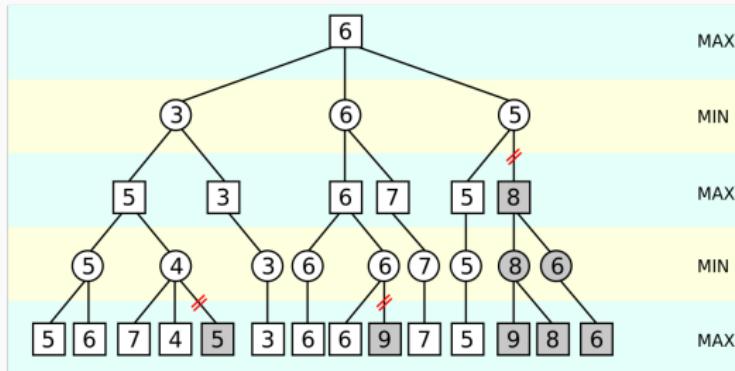
О силе игры

- ...да совсем никак.
- 1997: Janice Kim 1p победила HandTalk на 25 камнях форы.
- В том же году HandTalk выиграл часть Ing Prize, победив трёх 11-13-летних любителей 2-6d на 11 камнях форы.
- 2001: Many Faces выиграл у insei (1kyu pro) на 15 камнях.
- Это те годы, когда DeepBlue уже давно победил Каспарова.
- В чём же дело? Почему так сложно?



Почему го – это сложно

- В других играх (шахматы) отлично работает alpha-beta search: строим минимакс-дерево ходов, выкидываем ходы, которые заведомо хуже других, ищем в глубину.



- Почему не в го?

Почему го – это сложно

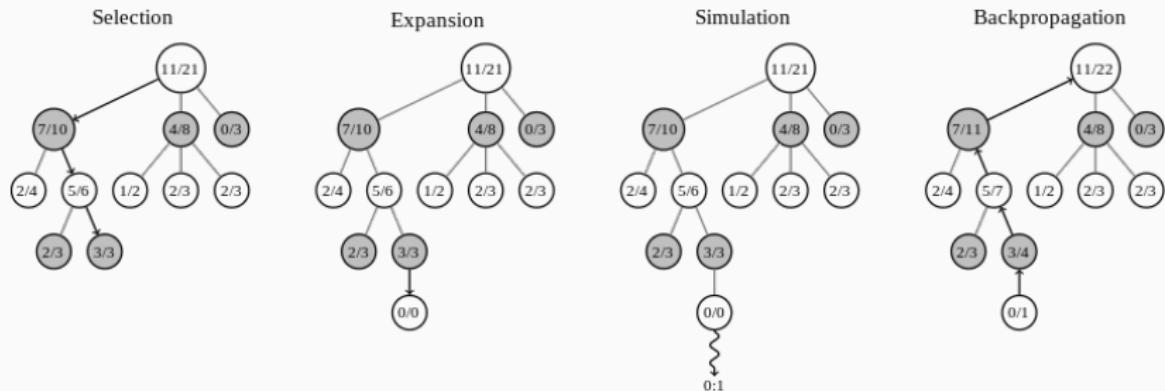
- Почему го сложнее, чем шахматы?
- Очень большая ширина дерева (branching factor):
 - в шахматах 30-40 возможных ходов, из них многие сразу приводят к потерям и их можно обрезать;
 - в го около 250 возможных ходов, и из них «вполне разумных» тоже около сотни.
- Оценка позиции:
 - в шахматах можно посчитать материал и какие-то простые эвристики, и если разрыв большой, скорее всего оценка очевидна;
 - в го тоже может упасть большая группа, но это редкость, обычно потери территориальные, труднооцениваемые;
 - очень сложная для автоматизации задача – даже просто оценить конечную позицию в го (когда люди уже согласились, кто выиграл).

Почему го – это сложно

- Тактика – life and death problems:
 - казалось бы, компьютеры считают варианты быстро;
 - но в го на решение локальных проблем влияют далеко расположенные камни и вся позиция; редко можно обрезать часть доски и сделать полный перебор;
 - а статус групп может измениться под влиянием глобальных эффектов и взаимодействовать с ними;
 - кроме того, человеку в го считать проще, чем в шахматах.
- Стратегия: нужно понимать взаимодействие камней на всей доске и очень тонко оценивать возникающую позицию, досчитать большинство веток до «явного преимущества», как в шахматах, невозможно.
- Ко-борьба: программы всегда отвратительно играли ко, бездарно тратили ко-угрозы, не замечали, что ко может возникнуть.

Monte-Carlo tree search

- 2006 (Remi Coulom, затем MoGo, затем все): революция компьютерного го – Monte-Carlo tree search (MCTS).
- Запускаем случайные симуляции с текущей позиции, смотрим, в каких ветках больше выигрышней, повторяем.



Monte-Carlo tree search

- Основная часть MCTS – формула, по которой выбирают узел для дальнейшего анализа:
 - это всё основано на тех же многоруких бандитах;
 - UCT (upper confidence bound UCB1 applied to trees) – выбираем узел с максимальным

$$\frac{w_i}{n_i} + c \sqrt{\frac{\ln t}{n_i}},$$

где w_i – число побед, n_i – число симуляций, c – параметр (часто $\sqrt{2}$), $t = \sum_i n_i$ – общее число симуляций.

Monte-Carlo tree search

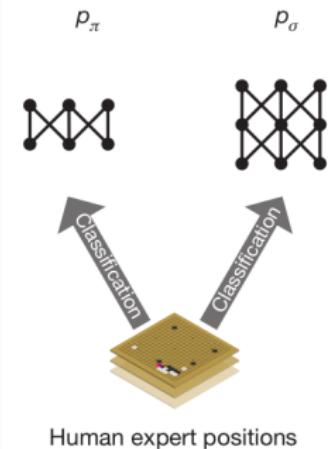
- И пошли более разумные результаты:
 - 2006: CrazyStone выиграл ICGA Computer Olympiad на доске 9×9 ;
 - 2007: CadiaPlayer – чемпион по General Game Playing;
 - 2008: MoGo достиг уровня дана на доске 9×9 ;
 - 2009: Fuego побеждает одного из топ-игроков на доске 9×9 ;
 - 2009: Zen достигает уровня 1d на сервере KGS на доске 19×19 ;
 - 2012: Zen выиграл 3:1 матч у 2d любителя на доске 19×19 ;
 - 2013: CrazyStone выиграл у Yoshio Ishida 9р на четырёх камнях;
 - около 2015-2016 CrazyStone и Zen, основанные на MCTS, достигали уровня 6-7d на KGS (но это совсем не то же, что профессиональные даны).
- MCTS работал и в других играх (Hex, Arimaa, MtG, Settlers).
- Но в 2016-м «внезапно» появилась AlphaGo. Что же она делает?

AlphaGo: история

- AlphaGo разработана в 2014-2015гг. в Google DeepMind: David Silver, Aja Huang и другие.
- В октябре 2015 года распределённая версия AlphaGo (1202 CPUs + 176 GPUs) победила Fan Hui 2р, чемпиона Европы, со счётом 5:0 (и 3:2 с укороченным контролем).
- Об этом рассказывает статья в *Nature* «Mastering the game of Go with deep neural networks and tree search», основной источник об AlphaGo.
- В 2016 – победа 4:1 над Lee Sedol 9р, одним из лучших профессионалов в мире.
- Что же изменилось в AlphaGo по сравнению с MCTS-программами?

AlphaGo: краткий обзор

- SL policy network предсказывает ходы:
 - 13-слойная свёрточная сеть выделяет признаки по всей доске;
 - обучается на профессиональных играх (30 млн позиций с KGS);
 - на выходе распределение ходов $p_\sigma(a | s)$;
 - получается 57.0% правильных предсказаний (очень много);
 - ещё обучаем быструю, но более плохую стратегию p_π (точность 24.2%, но оценка позиции за 2нс вместо 3мс).

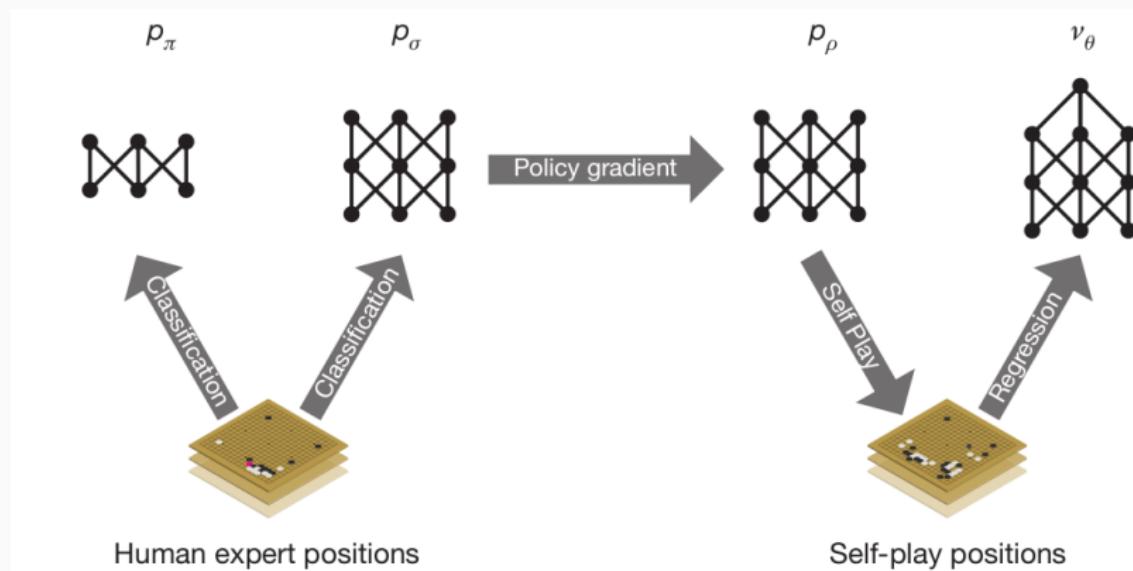


AlphaGo: краткий обзор

- SL policy network предсказывает ходы.
- RL policy network улучшает policy network за счёт обучения с подкреплением:
 - та же структура (13-слойная свёрточная сеть);
 - инициализируется с SL policy network;
 - затем играет сама с собой (с одной из предыдущих итераций обучения);
 - веса дообучаются так, чтобы максимизировать вероятность выигрыша;
 - RL policy network выигрывает 80% игр против SL, 85% игр против Pachi (одной из лучших MCTS-программ);
 - и это всё ещё без всякого счёта, просто глобальной оценкой позиции!

AlphaGo: краткий обзор

- SL policy network предсказывает ходы.
- RL policy network улучшает policy network за счёт обучения с подкреплением.

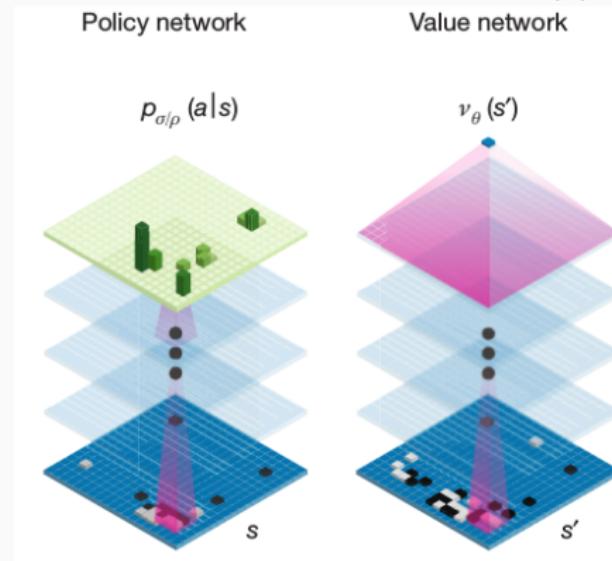


AlphaGo: краткий обзор

- SL policy network предсказывает ходы.
- RL policy network улучшает policy network за счёт обучения с подкреплением.
- Затем обучаем функцию оценки позиции $v_\theta(s)$:
 - предсказываем результат игры по позиции нейронной сетью;
 - структура сети похожа на policy networks, но теперь один выход;
 - если обучить на наборе профессиональных игр, будет просто overfitting, недостаточно данных;
 - поэтому обучаем на наборе self-play, порождённом RL policy network;
 - получается сеть для оценки позиции, которая работает очень быстро по сравнению с MCTS и даёт качество на уровне MCTS с RL policy (в 15K раз быстрее) и гораздо лучше, чем обычный MCTS.

AlphaGo: краткий обзор

- SL policy network предсказывает ходы.
- RL policy network улучшает policy network за счёт обучения с подкреплением.
- Затем обучаем функцию оценки позиции $v_\theta(s)$.

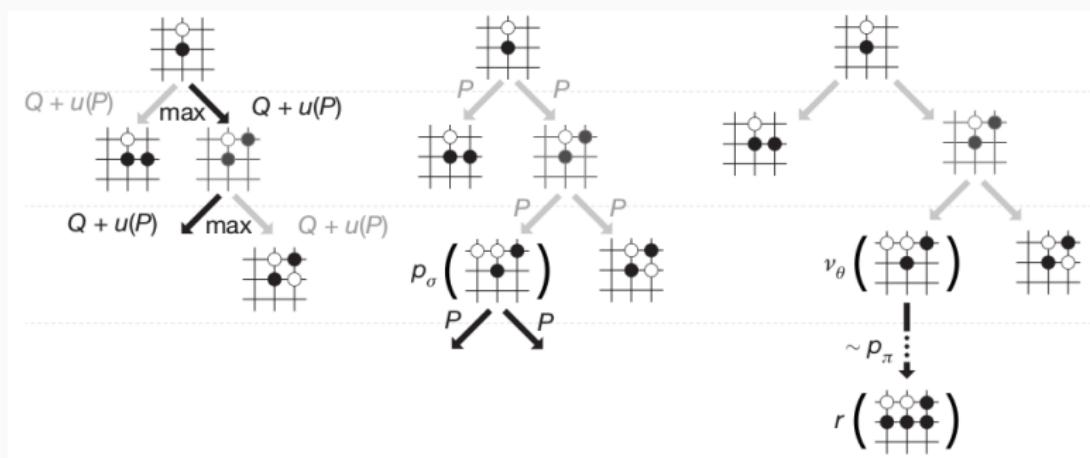


AlphaGo: краткий обзор

- SL policy network предсказывает ходы.
- RL policy network улучшает policy network за счёт обучения с подкреплением.
- Затем обучаем функцию оценки позиции $v_\theta(s)$.
- А теперь можно и деревья посчитать:
 - строим MCTS-подобное дерево;
 - априорные вероятности инициализируются через SL policy network как $p_\sigma(a | s)$;
 - в листе L считаем значение value network $v_\theta(s_L)$ и результат random rollout z_L при помощи стратегии p_π , объединяя результат;
 - любопытно, что SL policy работает лучше для построения дерева (видимо, более разнообразные варианты), но value function лучше строить на основе более сильной RL policy.

AlphaGo: краткий обзор

- SL policy network предсказывает ходы.
- RL policy network улучшает policy network за счёт обучения с подкреплением.
- Затем обучаем функцию оценки позиции $v_\theta(s)$.
- А теперь можно и деревья посчитать.



Матч с Ли Седолем

- Перед началом матча Ли Седоль предсказывал свою победу со счётом 5:0 или 4:1.
- Практически все эксперты были согласны с таким прогнозом.



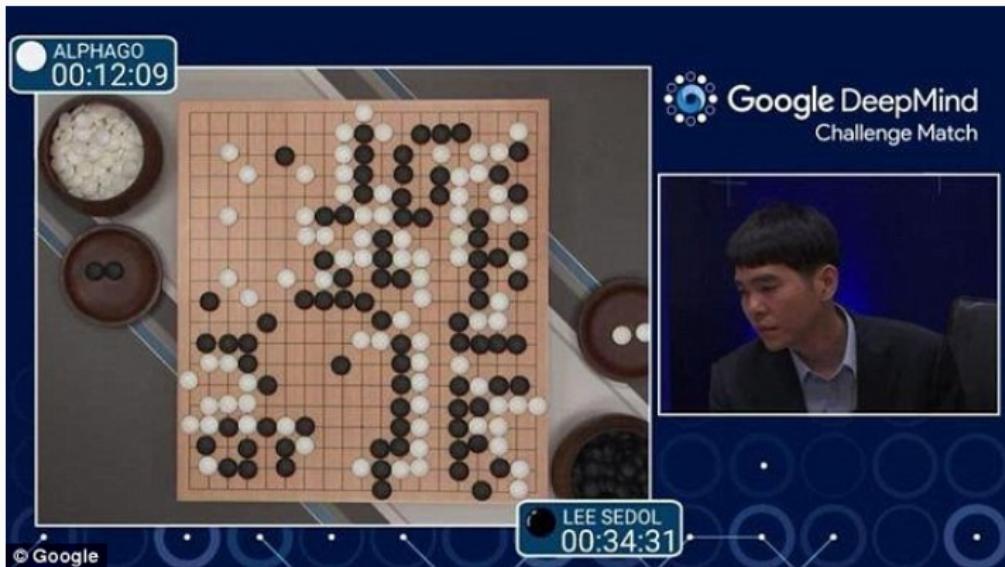
Матч с Ли Седолем

- Первая партия: Ли Седоль необычно разыграл фусеки (дебют), желая, видимо, свернуть с «накатанных» для AlphaGo путей (примерно так играли против компьютера в шахматы, когда это было возможно).



Матч с Ли Седолем

- Исход игры долго был неясен, AlphaGo, по мнению экспертов, делала ошибки, но около 120-130 хода белые (AlphaGo) получили преимущество и безупречно разыграли йосе (эндшпиль), без шансов для Ли Седоля.



Матч с Ли Седолем

- Во второй партии AlphaGo сделала ход, которого комментаторы совершенно не ожидали, и снова безупречно удержала небольшое преимущество в йосе.
- Ли Седоль после второй партии: «AlphaGo played a near perfect game».



Матч с Ли Седолем

- Исход третьей партии решился в первой части игры: AlphaGo блестяще провела сложную тактическую борьбу (слабое место других программ) и точными ходами захватила преимущество.
- Счёт стал 3:0, матч был проигран, казалось, что победа роботов будет абсолютной...



Матч с Ли Седолем

- ...но в четвёртой партии уже Ли Седоль сделал несколько великолепных ходов, успешно использовал недочёты в игре AlphaGo и выиграл!



Матч с Ли Седолем

- Пятая партия была отчасти экспериментальной, Ли Седоль попросил чёрный цвет (которым играть сложнее) и думал, что понимает, как обыграть AlphaGo... но, по своим собственным словам, пожадничал и проиграл.
- «The week with AlphaGo felt like a bamboo clapper».



AlphaGo Master

- В конце мая 2017 г. AlphaGo Master (новая версия) обыграла Ke Jie (#1 рейтинга), без шансов, 3:0...

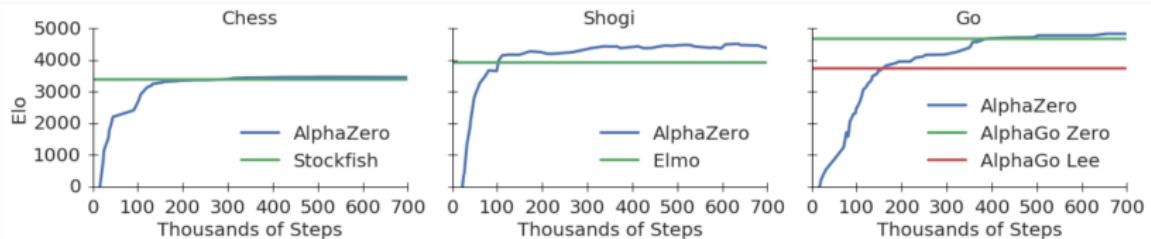


- ...и сейчас AlphaGo ушла из большого спорта, оставив небольшое, но очень важное наследие.

AlphaZero

AlphaGo Zero и AlphaZero

- Следующие супер-новости от DeepMind: AlphaGo Zero (Silver et al., 2017a) и особенно AlphaZero (Silver et al., 2017b).
- AlphaZero обучается без всяких данных, только играя сама с собой.
- И обыгрывает AlphaGo в го, Stockfish в шахматы, Elmo в сёги...



- Как это?

AlphaGo Zero и AlphaZero

- Обычно шахматные программы (тот же Stockfish) основаны на функции оценки позиции и, главное, альфа-бета поиске с отсечениями.
- Альфа-бета поиск обычно работает на основе минимакса. Но AlphaZero действует по-другому:
 - обучается с подкреплением исключительно на играх против самой себя;
 - оценивает позиции глубокой нейронной сетью (о признаках потом);
 - использует MCTS-поиск, оценивая всего 80К позиций в секунду для шахмат (у Stockfish 70М);
 - вместо минимакса усредняет оценку по поддереву; интуиция: ошибки в оценке позиции при этом усредняются и "взаимно сокращаются", а при минимаксе самая большая ошибка пропагируется в корень.

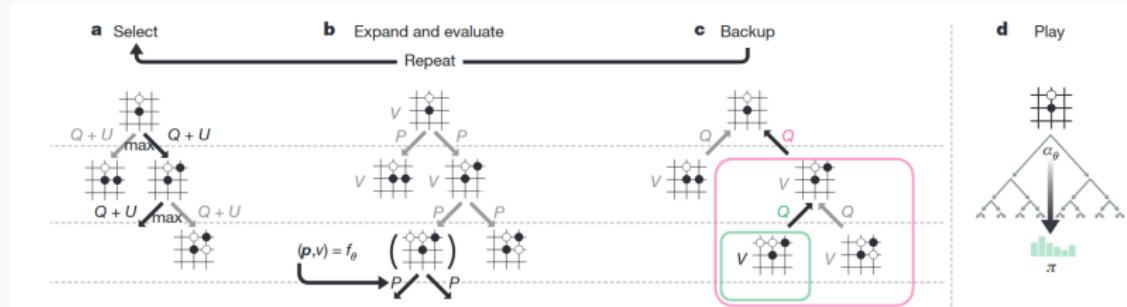
AlphaGo Zero и AlphaZero

- Признаки AlphaZero:

Feature	Go Planes	Chess Feature	Chess Planes	Shogi Feature	Shogi Planes
P1 stone	1	P1 piece	6	P1 piece	14
P2 stone	1	P2 piece	6	P2 piece	14
		Repetitions	2	Repetitions	3
				P1 prisoner count	7
				P2 prisoner count	7
Colour	1	Colour	1	Colour	1
		Total move count	1	Total move count	1
		P1 castling	2		
		P2 castling	2		
		No-progress count	1		
Total	17	Total	119	Total	362

AlphaGo Zero и AlphaZero

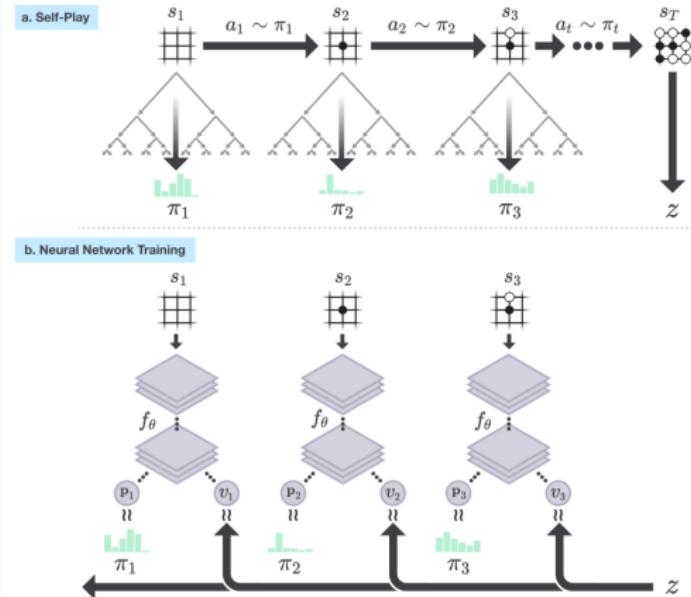
- Как мы делаем ход (картинка из AlphaGo Zero):



- Сразу MCTS при обучении:
 - оцениваем узлы по $Q + U$, где Q – среднее значение детей узла, U – добавка для оптимизма;
 - дойдя до листа, оцениваем через сеть, записываем v в узел, создаём детей по распределению из сети p , больше для этой вершины сеть не запускаем.
- После такого цикла выбираем ход, где больше N , или с дополнительным exploration при обучении.

AlphaGo Zero и AlphaZero

- Обучение с подкреплением (картина из AlphaGo Zero):



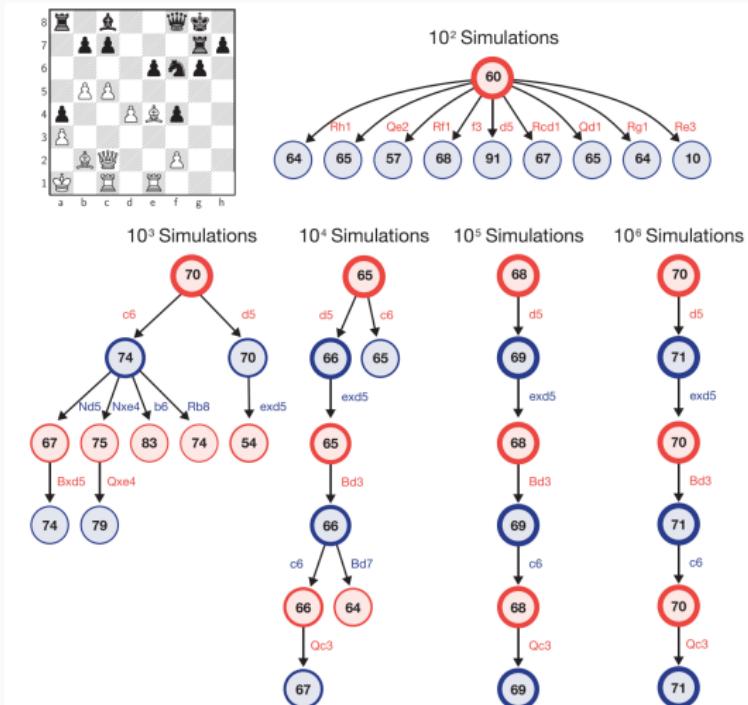
- Накапливаем опыт и минимизируем
$$L = (z - v)^2 + \pi^\top \log p + c\|\theta\|^2$$
, где p – вероятности сети, π – улучшенные через MCTS вероятности.

AlphaGo Zero и AlphaZero

- Более того, AlphaZero даже проще и прямолинейнее, чем AlphaGo Zero:
 - AlphaGo Zero использовала симметрию го для аугментаций, а AlphaZero этого не делает;
 - AlphaGo Zero поддерживала текущего лучшего противника из предыдущих итераций, периодически его заменяя, а AlphaZero просто использует всегда текущую сеть;
 - свёрточная сеть хорошо приспособлена для го, а для шахмат/сёги интуитивно не очень, но на это тоже забили.
- И тем не менее, всё получилось очень хорошо.

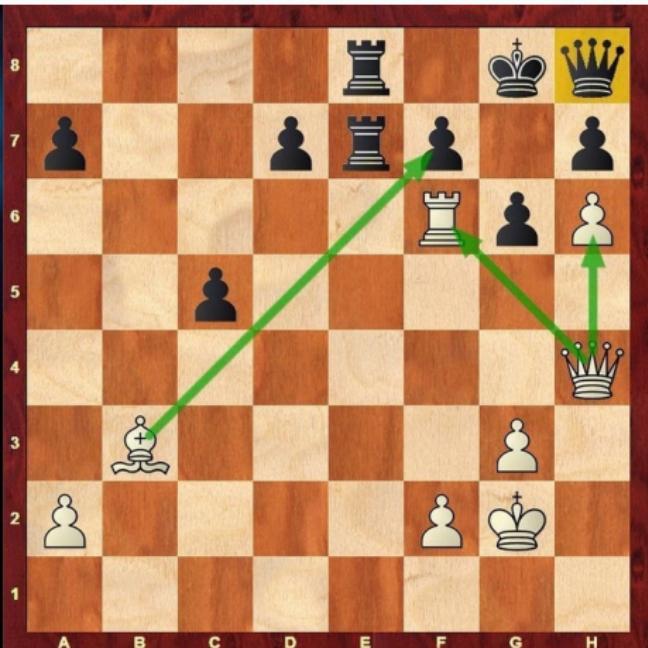
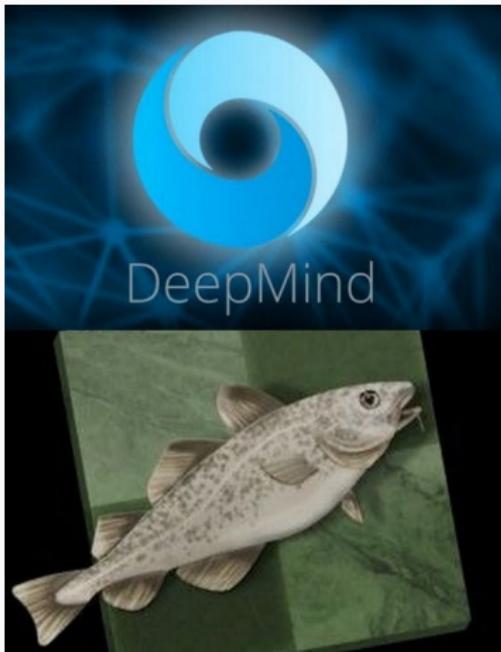
AlphaGo Zero и AlphaZero

- Пример раздумий MCTS:



AlphaGo Zero и AlphaZero

- И всё, да здравствует король!

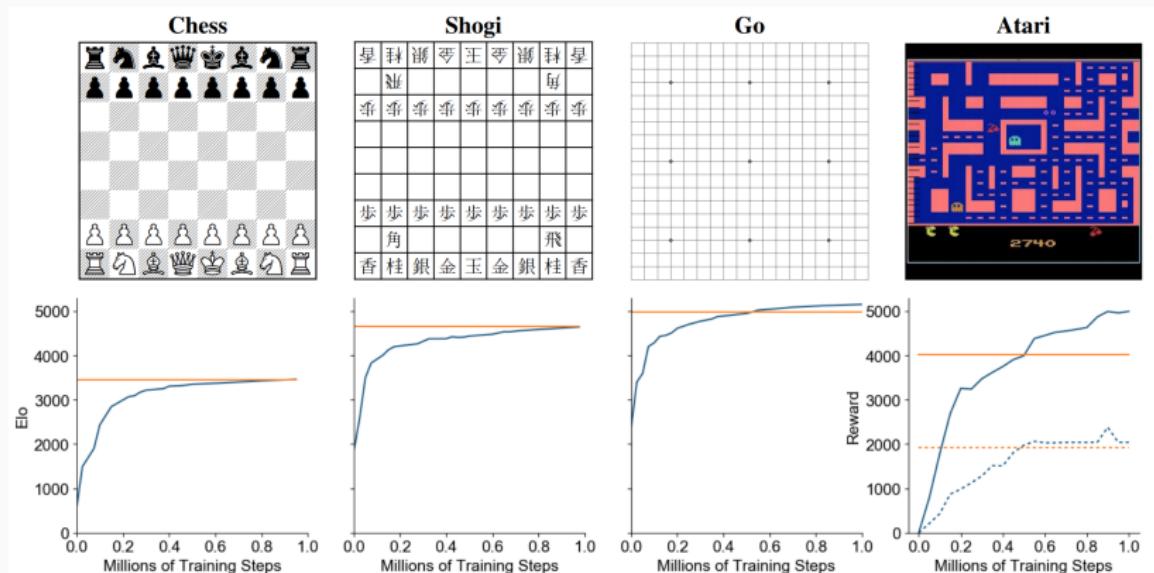


- Но и это ещё не вся история...

MuZero

MuZero

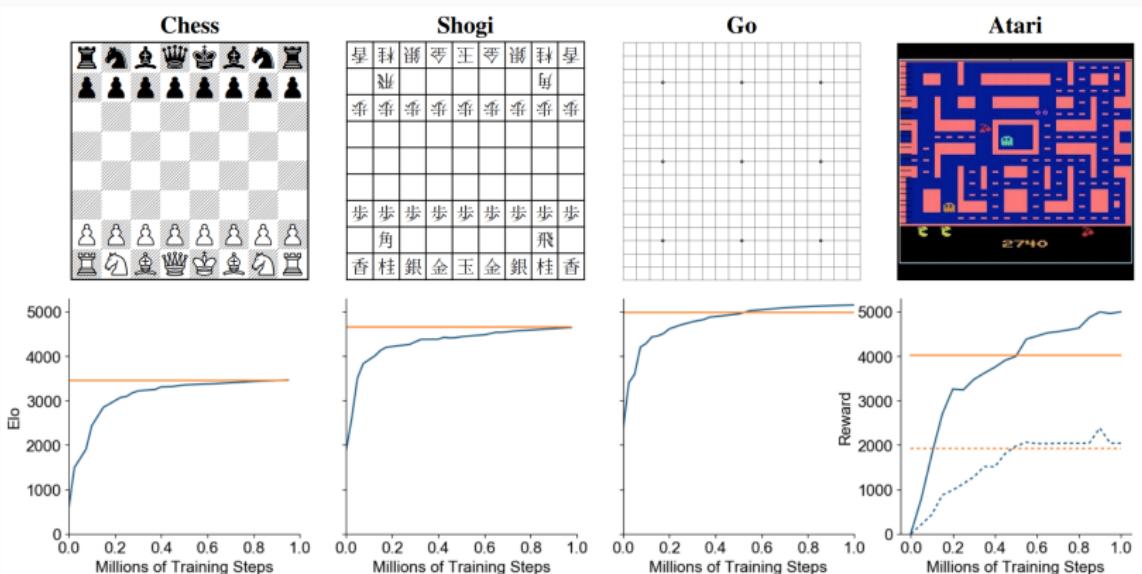
- В конце 2019 года появилась работа о MuZero, которая ещё лучше (Schrittwieser et al., 2019)!



- Как они смогли? Ключевое слово здесь...

MuZero

- ...планирование! Не зря мы его изучали.
- MuZero поддерживает модель динамики MDP, обучая её тоже по ходу дела:

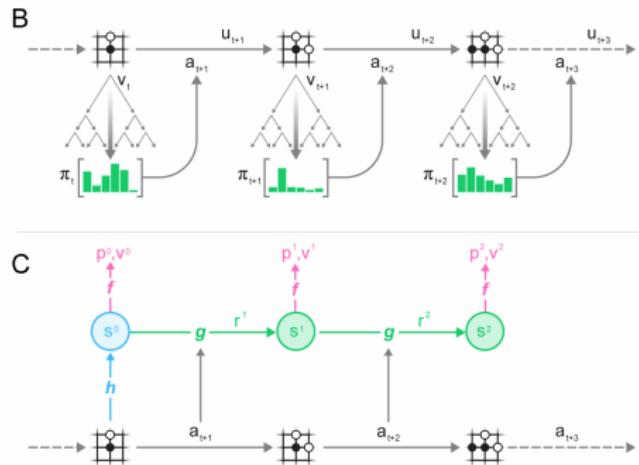
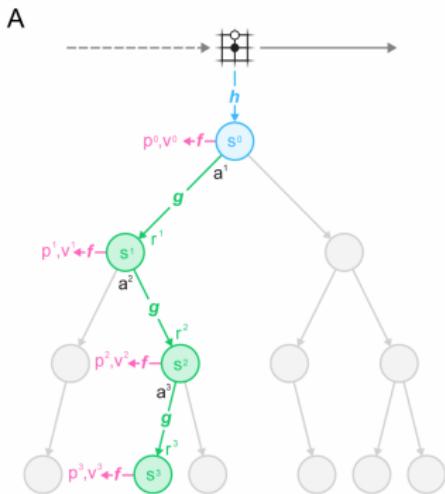


- Давайте посмотрим в подробностях...

- MuZero строит модель окружения по ходу обучения, заранее не зная даже правил игры (ну, кроме финальных наград).
- MuZero предсказывает на каждом шаге t три вещи для каждого будущего шага $k = 1, \dots, K$ на основе наблюдений o_1, \dots, o_t :
 - стратегию $p_t^k \approx \pi(a_{t+k+1} | o_1, \dots, o_t, a_{t+1}, \dots, a_{t+k})$;
 - функцию $v_t^k \approx \mathbb{E}[R_{t+k+1} + \gamma R_{t+k+2} + \dots | o_1, \dots, o_t, a_{t+1}, \dots, a_{t+k}]$;
 - награду $r_t^k \approx R_{t+k}$.

- Внутри это представлено тремя функциями:
 - функция динамики процесса: $r^k, s^k = g_\theta(s^{k-1}, a^k)$, но здесь s^k – это не состояние среды, а просто внутреннее состояние модели;
 - функция предсказания $p^k, v^k = f_\theta(s^k)$, примерно как в AlphaZero;
 - функция представления $s^0 = h_\theta(o_1, \dots, o_t)$.

- Вот так это выглядит:



- Когда есть такое представление, можно искать по гипотетическим будущим траекториям a^1, \dots, a^k .
- Для такого поиска, опять же, используется MCTS, слегка обобщённый с го/шахмат на окружения с, возможно, одним агентом и промежуточными наградами.
- Поиск выдаёт стратегию π_t и оценку позиции v_t , выбираем действие $a_{t+1} \sim \pi_t$.
- И дальше, почти как в AlphaZero, оптимизируем ошибку, но не в конец эпизода (его может не быть), а в bootstrap-оценку на n шагов вперёд:

$$z_t = u_{t+1} + \gamma u_{t+2} + \dots + \gamma^{n-1} u_{t+n} + \gamma^n v_{t+n},$$

где u_t — мгновенные награды.

- И обучаем функцию ошибки, составленную из ошибок всех трёх обучаемых функций:

$$L_t(\theta) = \sum_{k=0}^K \left(\ell^r(u_{t+k}, r_t^k) + \ell^v(z_{t+k}, v_t^k) + \ell^p(\pi_{t+k}, p_t^k) \right) + c\|\theta\|^2.$$

- Получается, что можем и в Atari играть, и роботов двигать, и всё одной моделью фактически:

Agent	Median	Mean	Env. Frames	Training Time	Training Steps
Ape-X [18]	434.1%	1695.6%	22.8B	5 days	8.64M
R2D2 [21]	1920.6%	4024.9%	37.5B	5 days	2.16M
<i>MuZero</i>	2041.1%	4999.2%	20.0B	12 hours	1M
IMPALA [9]	191.8%	957.6%	200M	–	–
Rainbow [17]	231.1%	–	200M	10 days	–
UNREAL ^a [19]	250% ^a	880% ^a	250M	–	–
LASER [36]	431%	–	200M	–	–
<i>MuZero Reanalyze</i>	731.1%	2168.9%	200M	12 hours	1M

Спасибо!

Спасибо за внимание!

