



ТЕХНОСФЕРА

Лекция 14 Reinforcement learning pt.2

Храбров Кузьма

7 мая 2018 г.

План лекции

Напоминания (MDP, policy and q-function)

Q-learning

DQN

Многорукие бандиты

Monte Carlo Tree Search

AlphaGo

Markov decision process

Definition

MRP это кортеж (S, A, R, P, γ) , где

- ▶ S - состояния (дискретное пространство)
- ▶ A - действия (дискретное пространство)
- ▶ R - функция rewards, $R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- ▶ P - матрица переходов (transition matrix)
 $P_{ss'}^a = Pr(S_{t+1} = s' | S_t = s, A_t = a)$
- ▶ γ - discount factor

Definition (Policy)

$\pi(a|s) = Pr(A_t = a | S_t = s)$ - стратегия, т.е. то как мы выбираем действия оказавшись в состоянии s .

Definition (Value function)

$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$ - ценность состояния.

Definition (Q-function)

$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$ - ценность действия в состоянии s .

Bellman Equations

Definition (Policy)

$\pi(a|s) = Pr(A_t = a|S_t = s)$ - стратегия, т.е. то как мы выбираем действия оказавшись в состоянии s .

Definition (Value function)

$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$ - ценность состояния.

Definition (Q-function)

$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$ - ценность действия в состоянии s .

Bellman equations

Bellman equation для value-function

$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s]$$

Bellman equation для q-function

$$q_{\pi}(s, a) = \mathbb{E}_{\pi} [R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

Оптимальные value-functions

Оптимальная value-function - максимальное принимаемое значение

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) = Q^{\pi^*}(s, a)$$

Как только мы знаем Q^* мы можем выбрать оптимальную стратегию

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$$

Оптимальное значение - максимум по всем принимаемым решениям:

$$\begin{aligned} Q^*(s, a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \dots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

Соответствующее уравнение Беллмана:

$$Q^*(s, a) = \mathbb{E}_{s'} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]$$

Q-learning

Будем решать уравнение Беллмана:

$$Q^*(s, a) = \mathbb{E}_{s'} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right]$$

методом конечных приращений, то есть будем повторять:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a'))$$

где α - learning rate. Обычно берут $\alpha = 0.9$.

Упражнение: покажите, что таким образом минимизируется квадрат разности.

Подходы к обучению с подкреплением

1. Value-based RL

Оцениваем оптимальную Q -функцию $Q^*(s, a)$.

Максимальное значение принимаемое при любой стратегии.

2. Policy-based RL

Ищем оптимальную стратегию π^* . Стратегия обеспечивающая максимальное будущее вознаграждение.

3. Model-based RL

Строим и используем модель внешней среды.

Deep Q-learning

В каждом подходе к RL можно применить нейронные сети.

Разберем подробно Value-based случай.

Будем аппроксимировать $Q(s, a) = Q(s, a, w)$ с помощью нейронной сети с весами w . Тогда для решения соответствующего уравнения Беллмана можно минимизировать функцию потерь MSE

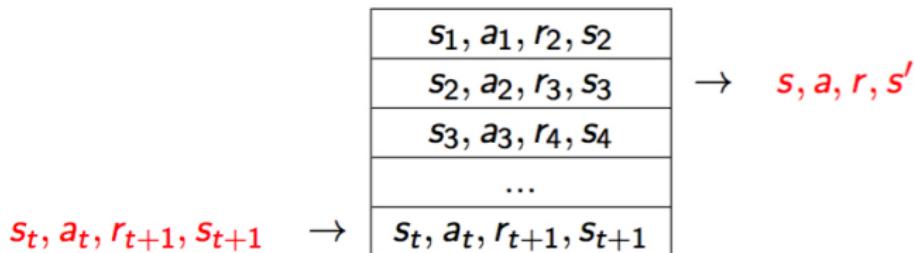
$$l = \left(r + \gamma \max_a Q(s', a', w) - Q(s, a, w) \right)^2$$

Проблемы:

1. Корреляции между входами
2. Нестационарные целевые переменные

DQN-2

Чтобы убрать корреляции в данных: используем опыт агента
(Replay memory)

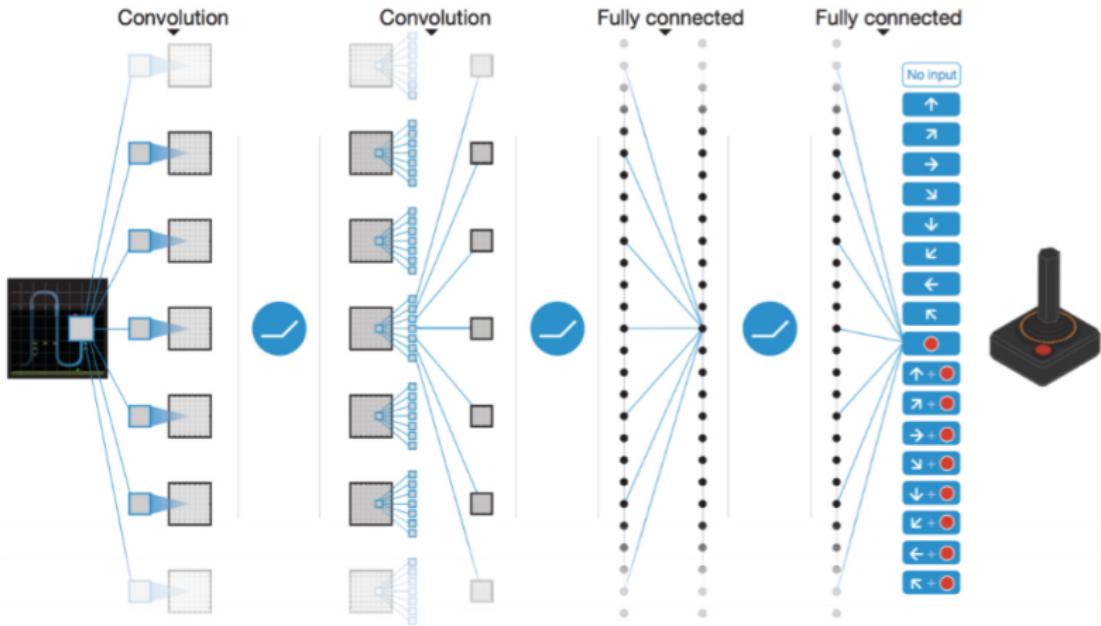


Сэмплируем опыт из данных и обновляем

$$l = \left(r + \gamma \max_a Q(s', a', w^-) - Q(s, a, w) \right)^2$$

Причем w^- оставляем фиксированными, чтобы убрать нестационарность.

ATARI



ATARI=2

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

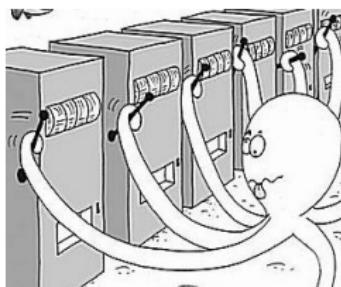
end for

end for

ATARI-improvements

1. Double DQN
2. Prioritised replay
3. Duelling network

Многорукие бандиты



Рассмотрим задачу обучения с подкреплением, в которой состояние среды/агента от действия к действию не изменяется. Иначе говоря, у агента есть некоторый набор возможных действий, агент выбирает одно из них, получает за это некоторое вознаграждение (которое является случайной величиной), а затем снова может выбирать из тех же действий.

Многорукие бандиты

Решение задачи:

Algorithm 1 – UCB1

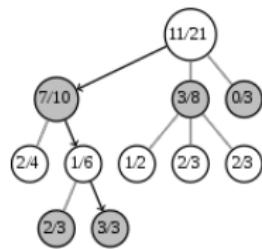
At time t play the arm j that maximizes
 $\bar{X}_{j,n_j} + \sqrt{\frac{2 \ln(t)}{n_j}}$, where n_j denotes $T_j(t - 1)$.

Algorithm 2 – HUCB1

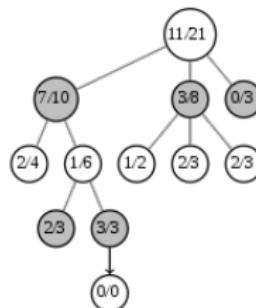
At time t play the arm j that maximizes
 $\bar{X}_{j,n_j}^h + \sqrt{\frac{2 \ln(H_j + t)}{n_j + H_j}}$, where n_j denotes $T_j(t - 1)$.

Monte Carlo Tree Search (MCTS)

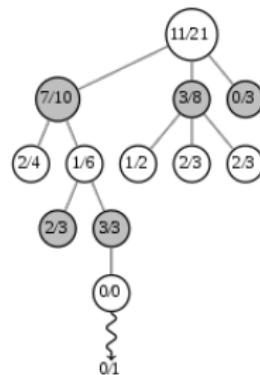
Selection



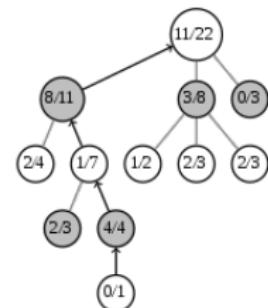
Expansion



Simulation



Backpropagation



Monte Carlo Tree Search (MCTS)

1. Selection (Выбор)

Начиная с корня дерева выбираем двигаемся по ветвям, выбирая путь согласно ($UCB1$) пока не доберемся до листа L .

2. Expansion (Расширение)

Если L - не терминальное состояние, делаем еще одно действие добавляя узел C .

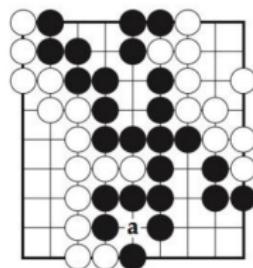
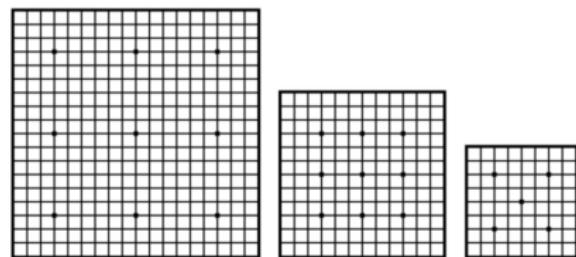
3. Simulation (Симуляция)

Запускаем симуляцию из C до окончания игры.

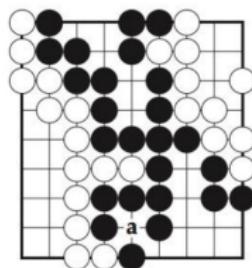
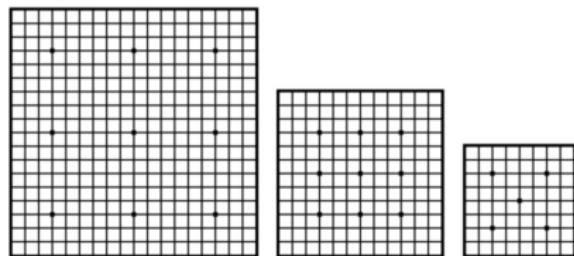
4. Backpropagation

Обновляем информацию во всех узлах дерева.

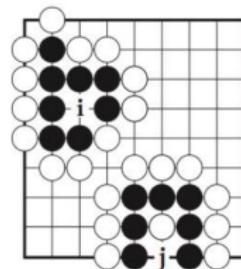
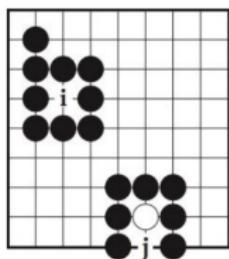
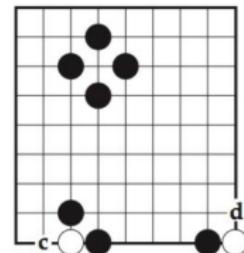
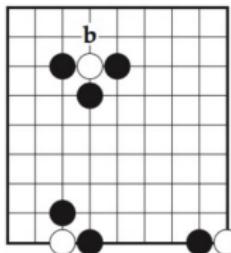
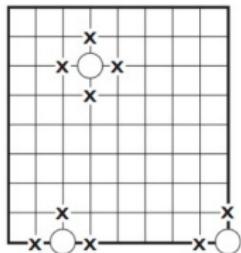
Игра Го



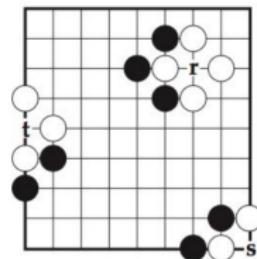
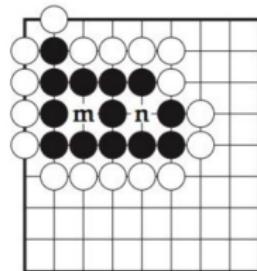
Игра ГО



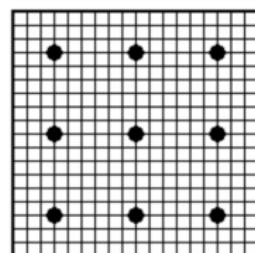
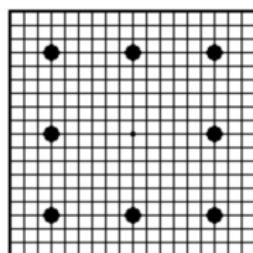
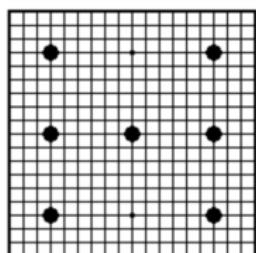
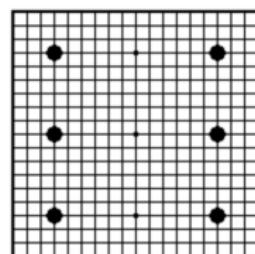
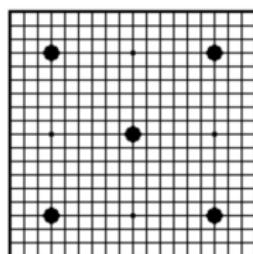
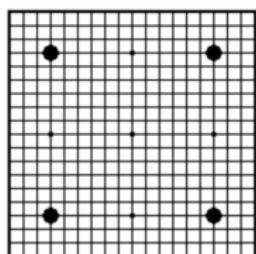
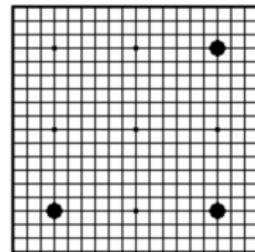
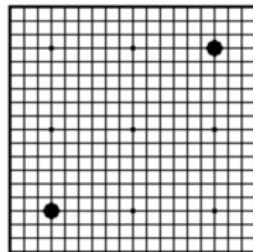
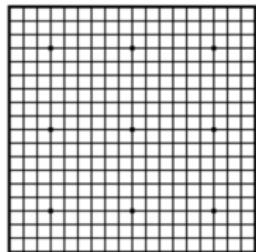
Игра ГО



Игра ГО

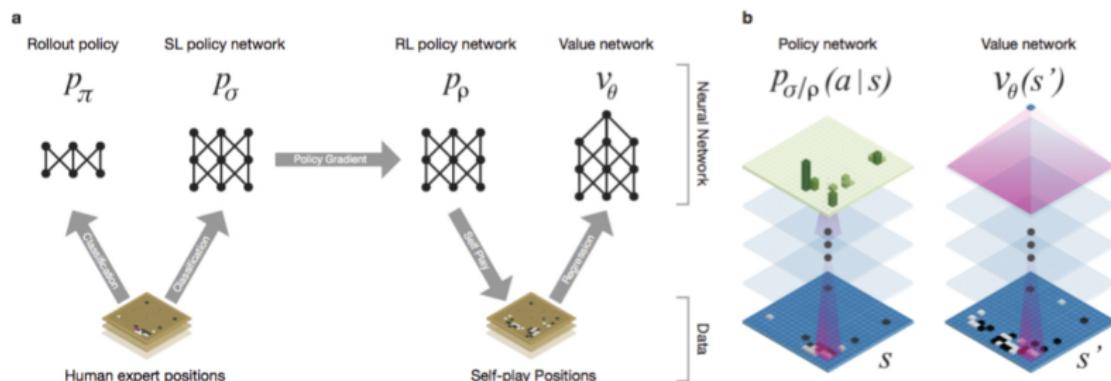


Игра ГО



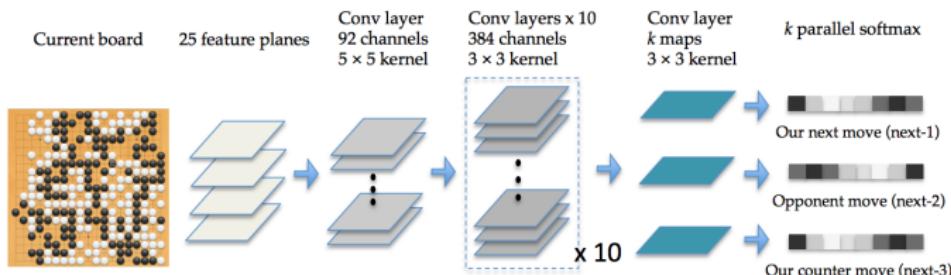
AlphaGo

Ключевая идея алгоритма: будем приближать value-function и policy с помощью двух различных нейросетей, а играть и получать новую информацию с помощью MCTS. Плюс обучим классификаторы шагов на известных партиях экспертов.



AlphaGo. SL

Большая сеть p_σ обучена на позициях из KGS data set (29.4 миллиона позиций из 160000 игр сыгранных игроками от бго до 9го дана). Состояние системы - положение камней на доске, действие - ход игрока. Вход в сеть - onehot-encoding состояния. Маленькие сети p_π и p_τ (Rollout Policy и Tree Policy) - линейный классификатор, обученный на локальных состояниях (3×3) игровой доски вокруг следующего и предыдущего действий + множества других признаков.



AlphaGo. SL

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Extended Data Table 2: **Input features for neural networks.** Feature planes used by the policy network (all but last feature) and value network (all features).

Feature	# of patterns	Description
Response	1	Whether move matches one or more response features
Save atari	1	Move saves stone(s) from capture
Neighbour	8	Move is 8-connected to previous move
Nakade	8192	Move matches a <i>nakade</i> pattern at captured stone
Response pattern	32207	Move matches 12-point diamond pattern near previous move
Non-response pattern	69338	Move matches 3 × 3 pattern around move
Self-atari	1	Move allows stones to be captured
Last move distance	34	Manhattan distance to previous two moves
Non-response pattern	32207	Move matches 12-point diamond pattern centred around move

Extended Data Table 4: **Input features for rollout and tree policy.** Features used by the rollout policy (first set) and tree policy (first and second set). Patterns are based on stone colour (black/white/empty) and liberties ($1, 2, \geq 3$) at each intersection of the pattern.

AlphaGo. RL Policy and Value Networks

1. Тренируем сеть p_ρ аппроксимировать оптимальную policy, инициализировав веса с помощью p_σ .

Каждый батч обучения: n игр со старыми версиями policy $p_{p_}$.

$$\Delta \rho = \frac{\alpha}{n} \sum_{i=1}^n \sum_{t=1}^{T^i} \frac{\partial \log p_\rho(a_t^i | s_t^i)}{\partial \rho} (z_t^i - v(s_t^i))$$

2. Тренируем сеть $v_\theta(s)$ аппроксимировать value-function.

Обучающая выборка - случайный набор позиций из "игр с собой".

$$\Delta \theta = \frac{\alpha}{m} \sum_{k=1}^m (z^k - v_\theta(s^k)) \frac{\partial v_\theta(s^k)}{\partial \theta}$$

AlphaGo. MCTS

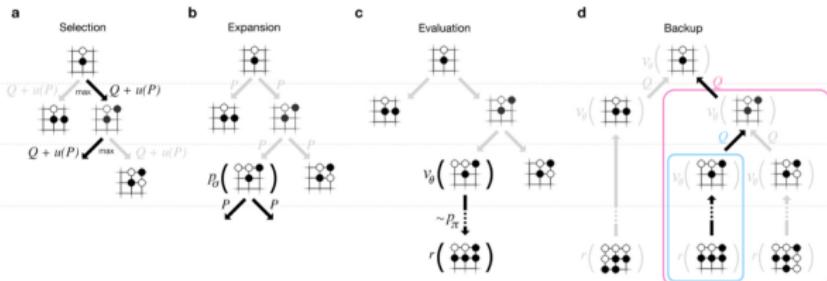


Figure 3: **Monte-Carlo tree search in AlphaGo.** **a** Each simulation traverses the tree by selecting the edge with maximum action-value Q , plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b** The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c** At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ ; and by running a rollout to the end of the game with the fast rollout policy p_π , then computing the winner with function r . **d** Action-values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

AlphaGo. MCTS

В каждом ребре храним:

$$\{P(s, a), N_v(s, a), N_r(s, a), W_v(s, a), W_r(s, a), Q(s, a)\}$$

1. Selection (Выбор)

Начиная с корня дерева выбираем двигаемся по ветвям, выбирая путь $a_t = \operatorname{argmax}_a(Q(s_t, a) + u(s_t, a))$

$$u(s, a) = c_{\text{puct}} P(s, a) \frac{\sqrt{\sum_b N_r(s, b)}}{1 + N_r(s, a)}$$

2. Evaluation

Добавляем лист L в очередь для вычисления $v_\theta(s_L)$, в случае если это значение еще не известно. Запускаем "rollout" с помощью p_π .

3. Expansion

Если $N_r(s, a) > n_{thr}$ добавляем новый узел с помощью p_τ .

4. Backpropagation

Обновляем информацию во всех узлах дерева.

$$Q(s, a) = (1 - \lambda) \frac{W_v(s, a)}{N_v(s, a)} + \lambda \frac{W_r(s, a)}{N_r(s, a)}$$

AlphaGo. Замечания

1. Можно использовать симметрии доски (D_4) для аугментации.
2. AlphaGo сдается, если $\max_a Q(s, a) < -0.8$ (вероятность выиграть < 10%).

AlphaGo Zero. Without human knowledge

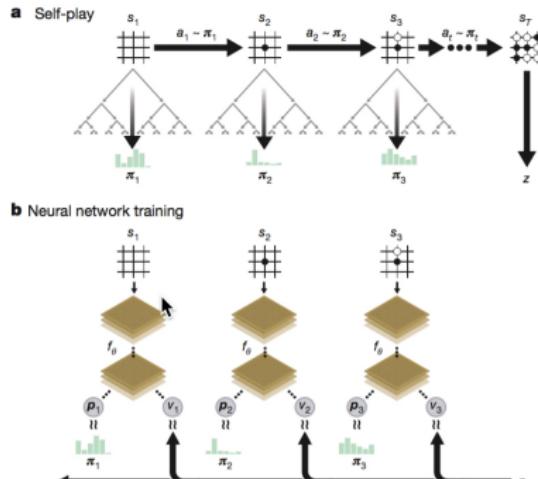


Figure 1 | Self-play reinforcement learning in AlphaGo Zero. **a**, The program plays a game s_1, \dots, s_T against itself. In each position s_t , an MCTS α_t is executed (see Fig. 2) using the latest neural network f_θ . Moves are selected according to the search probabilities computed by the MCTS, $a_t \sim \pi_t$. The terminal position s_T is scored according to the rules of the game to compute the game winner z . **b**, Neural network training in AlphaGo Zero. The neural network takes the raw board position s_t as its input, passes it through many convolutional layers with parameters θ , and outputs both a vector p_t , representing a probability distribution over moves, and a scalar value v_t , representing the probability of the current player winning in position s_t . The neural network parameters θ are updated to maximize the similarity of the policy vector p_t to the search probabilities π_t , and to minimize the error between the predicted winner v_t and the game winner z (see equation (1)). The new parameters are used in the next iteration of self-play as in **a**.

AlphaGo. Without human knowledge

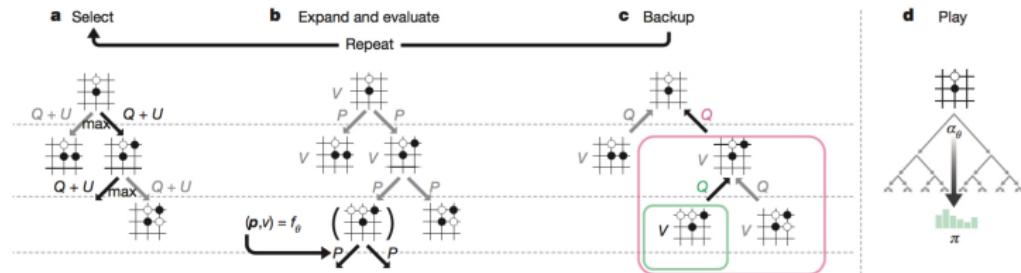


Figure 2 | MCTS in AlphaGo Zero. **a**, Each simulation traverses the tree by selecting the edge with maximum action value Q , plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed). **b**, The leaf node is expanded and the associated position s is evaluated by the neural network ($P(s, \cdot), V(s) = f_\theta(s)$); the vector of P values are stored in

the outgoing edges from s . **c**, Action value Q is updated to track the mean of all evaluations V in the subtree below that action. **d**, Once the search is complete, search probabilities π are returned, proportional to $N^{1/\tau}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.

Вопросы

