Mastering the Game of Go with Deep Neural Networks and Tree Search

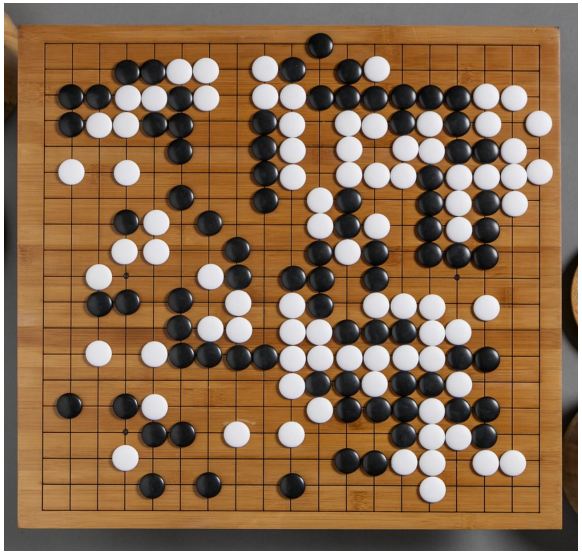
AlphaGo

В 2014 году DeepMind решила создать программу способную обыграть человека в Го

Го - стратегическая настольная игра, в которой ИИ не мог победить человека

Go

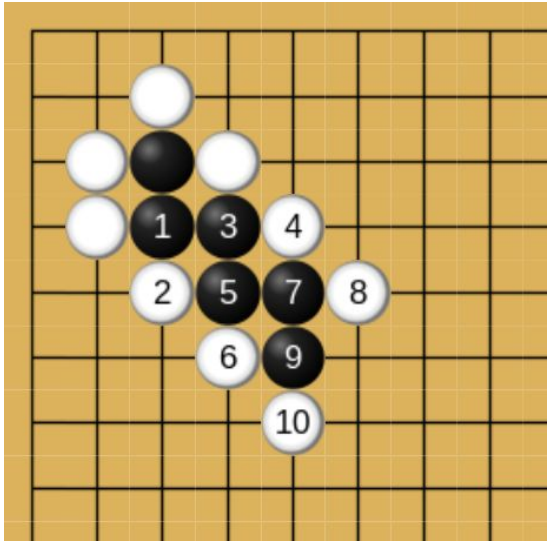
- Два игрока
- Поле 19x19
- Цель игры - захват территории
- При окружении группы чужих камней они снимаются с доски
- Нет ничьих



Go

- При окружении группы чужих камней они снимаются с доски

  Если белые поставят камень правее камня 9, то все камни черных будут сняты с доски.



В чем сложность игры для ИИ

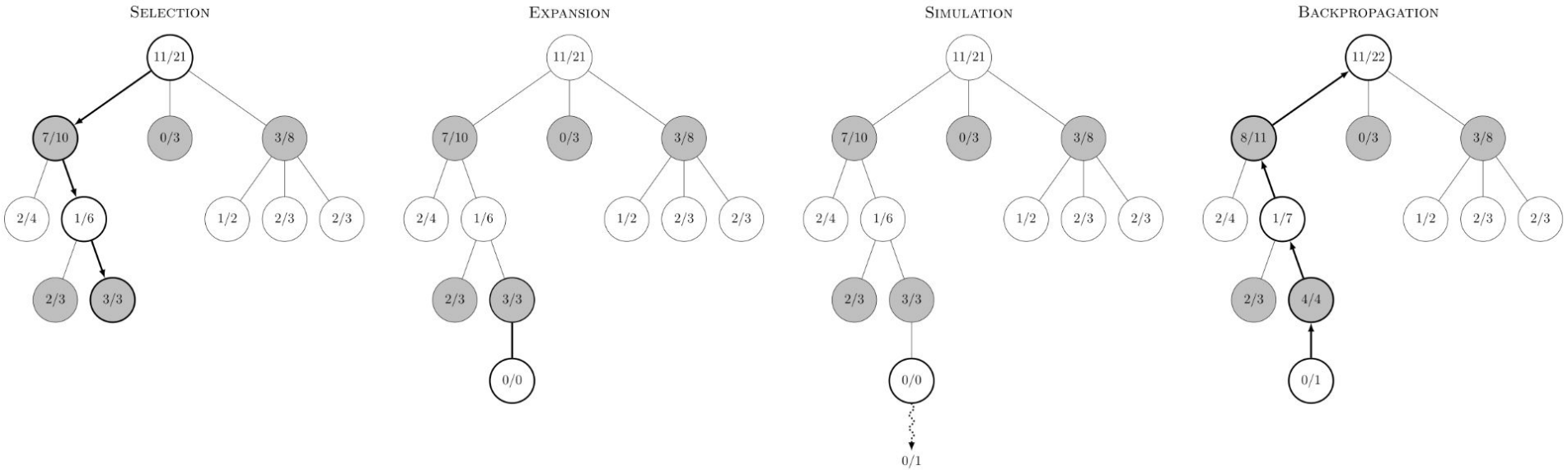
Высокая вычислительная сложность

В среднем примерно 250 возможных действий каждый ход, и 150 ходов за игру против 35 действий и 80 ходов для шахмат.

Конкуренты

Лучшие программы до AlphaGo играли на уровне 5-го любительского дана (из 9)
С 2006 года доминировали программы на основе MTCS (Monte-Carlo Tree Search)

Monte-Carlo Tree Search



UCT

Exploration vs Exploitation
Exploration - хотим чаще ходить в вершины в которых мы были мало раз
Exploitation - хотим чаще ходить в вершины с высоким винрейтом

w_i - число побед в вершине
n_i - число проходов через вершину
N_i - общее число проходов

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

Повторение RL

Policy функция - отображение множества состояний в действия.

- Deterministic policy: A = π(S)
- Stochastic policy: π(A|S) = p(A|S)

Value функция - ожидаемая награда в состоянии s, в нашем случае - вероятность победы.

Value и Policy можно использовать для улучшения MCTS.

AlphaGo

Learning from humans

- Supervised learning
- Хотим построить policy функцию предсказывающую ходы профессиональных игроков
- 160k реплеев игр (6-9 проф. дан), разбитые на 30m ходов
- 13 layer CNN на расположении фишек на доске + доп фичи
- Negative log likelihood loss

$$\Delta\sigma = \frac{\alpha}{m} \sum_{k=1}^{m} \frac{\partial \log p_\sigma(a^k|s^k)}{\partial\sigma}$$

Learning from humans

- 57% accuracy, 55.7% без доп фич
- Также обучили маленькую модель с 24.2%, она в 1000 раз быстрее

Learning from selfplay

- Получившаяся политика играет против одной из своих старых версий, и обучается с помощью Policy Gradient

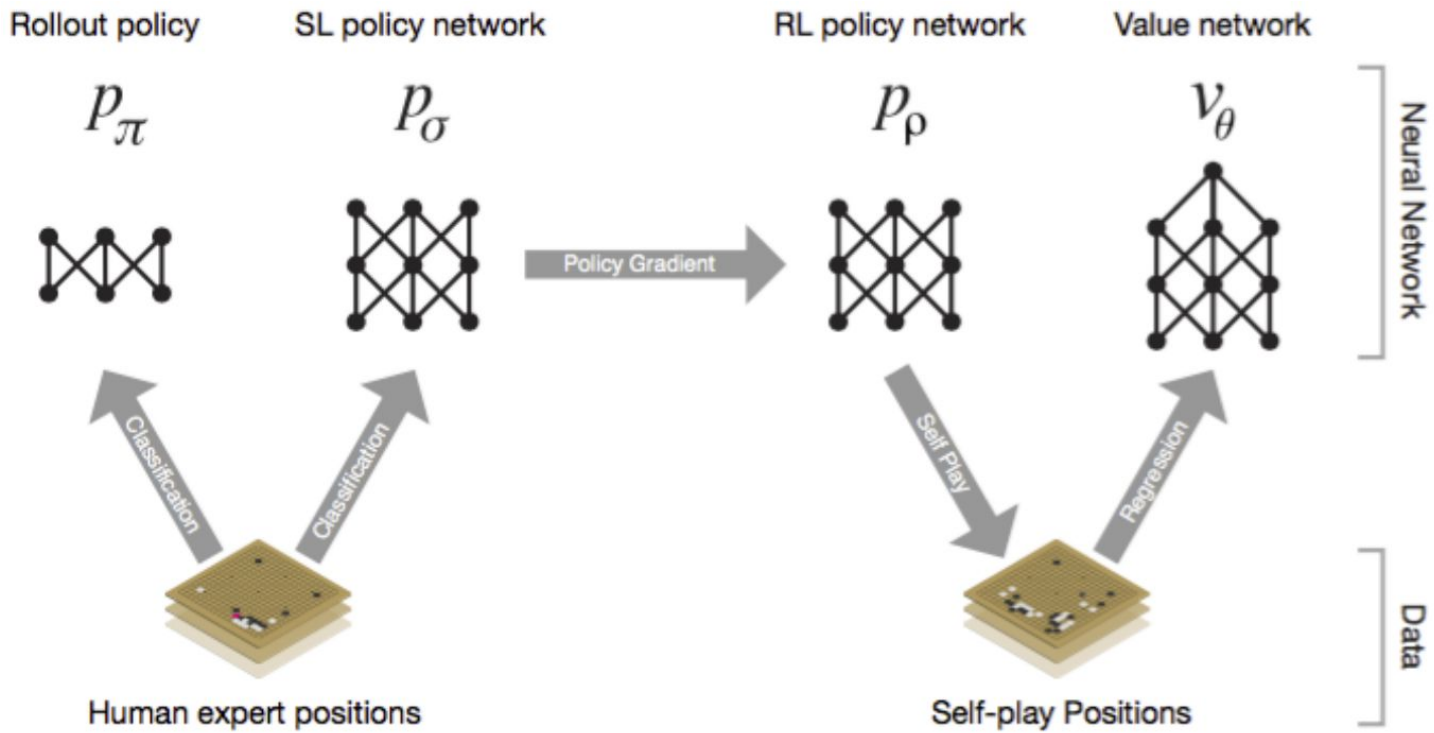
$$\Delta\rho = \frac{\alpha}{n}\sum_{i=1}^{n}\sum_{t=1}^{T^i}\frac{\partial \log p_\rho(a_t^i|s_t^i)}{\partial\rho}\left(z_t^i - v(s_t^i)\right)$$

Value network

- С помощью RL policy генерируем 50m игр, из каждой игры берем один ход
- Учим value функцию по состоянию предсказывать победителя
- Используем такую же CNN как и для SL policy, меняем только последние 2 слоя
- Используем MSE loss

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

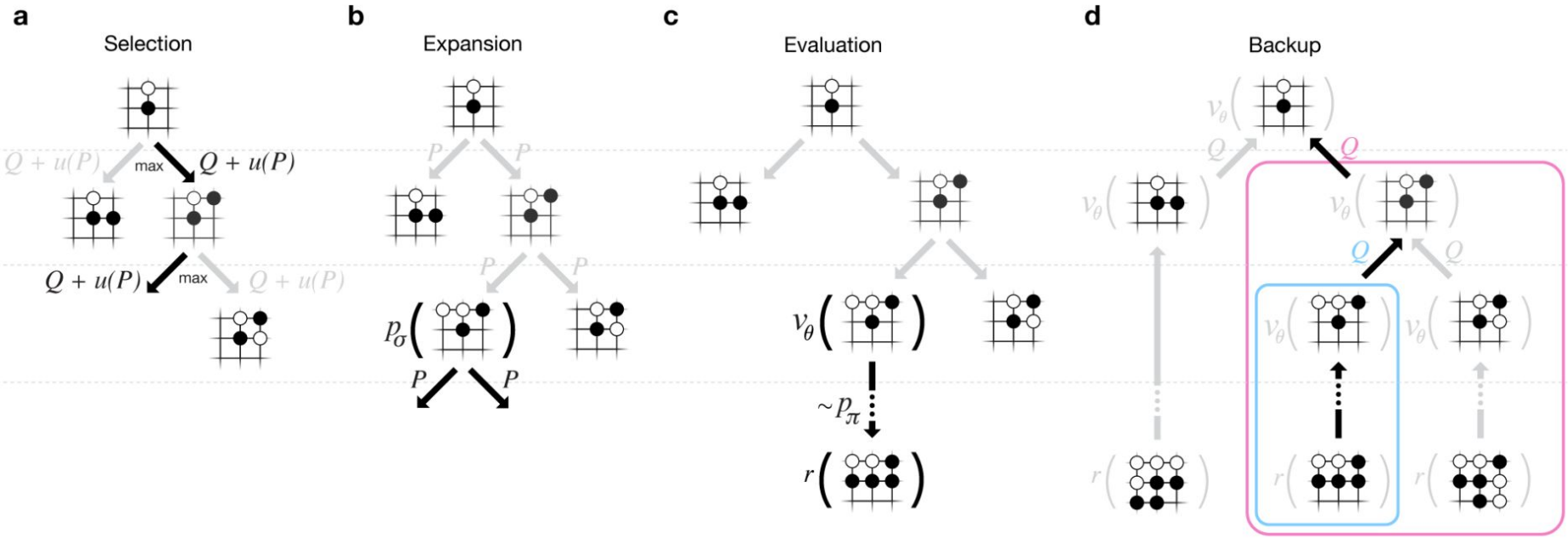
AlphaGo


Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy Gradient
Neural Network
Classification
Classification
Self Play
Regression
Data
Human expert positions
Self-play Positions

AlphaGo

RL policy выигрывает 80% игр у SL policy и 85% у лучшей open source программы Pachi

AlphaGo MCTS



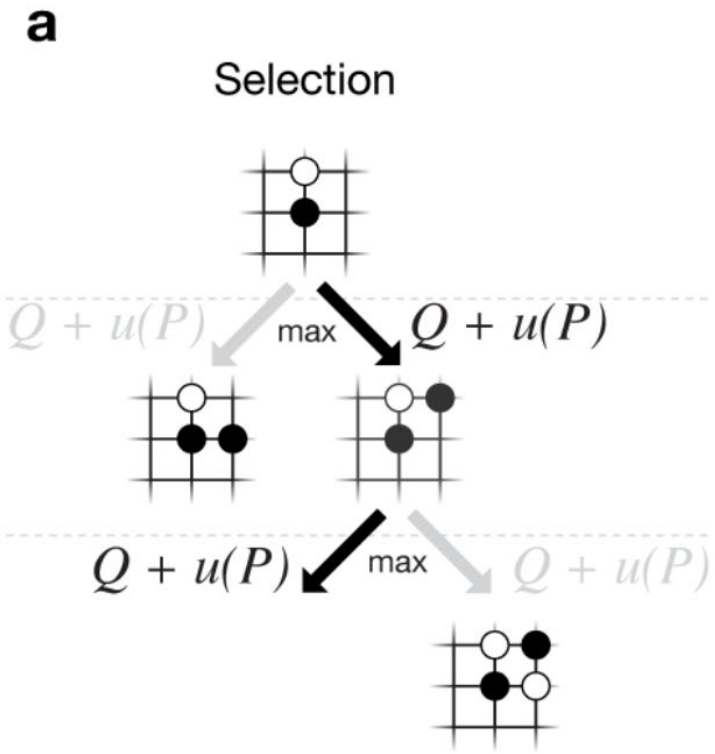
AlphaGo MCTS Selection

Похоже на UTC
У всех ребер есть вес из двух слагаемых отвечающих за exploitation и exploration
Q - оценка винрейта хода получаемая с помощью MCTS

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

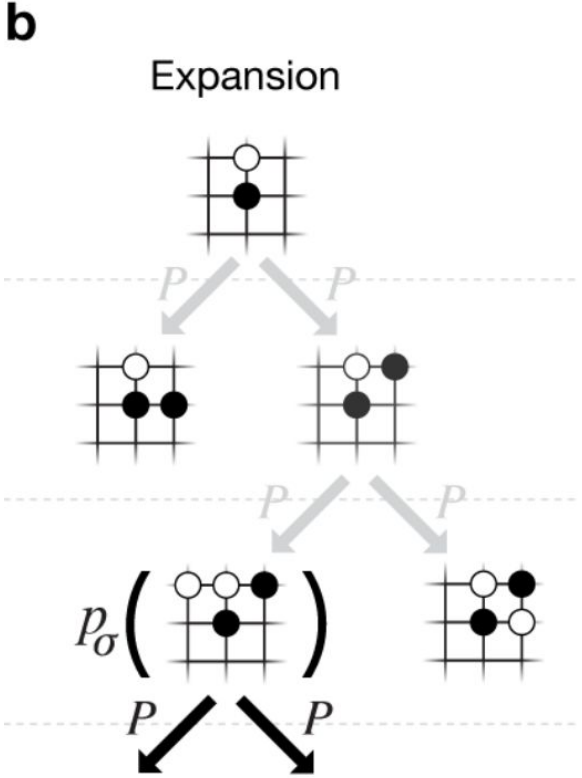
$$a_t = \underset{a}{\operatorname{argmax}} \left(Q(s_t,a) + u(s_t,a) \right)$$

Здесь P - SL policy



AlphaGo MCTS Expansion

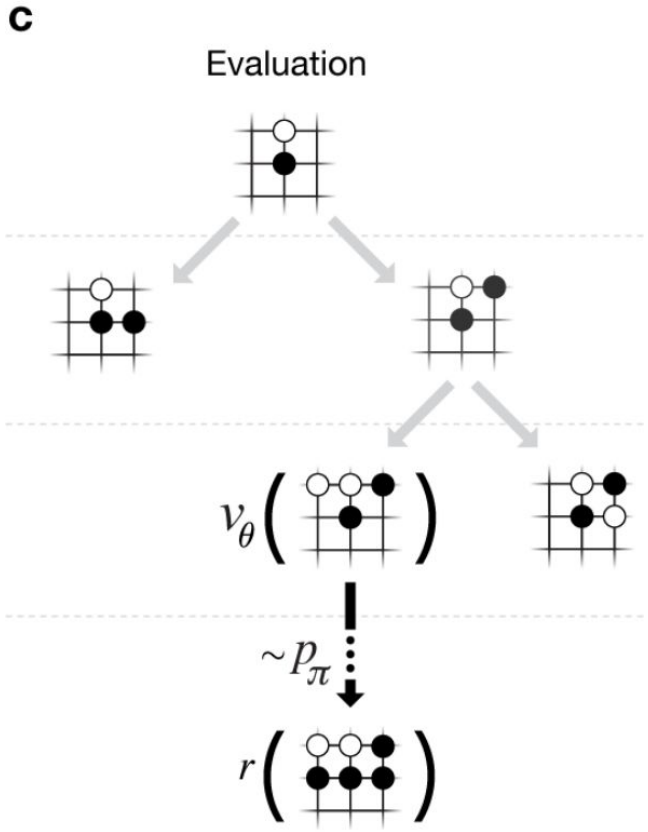
Прошлый шаг заканчивается когда попадаем в вершину в которой еще не были
Считаем и сохраняем значение SL policy в новой вершине



AlphaGo MCTS Evaluation

Проходим по дереву до конца используя вероятности из rollout policy (маленькая быстрая нейронка)
Считаем value вершины как комбинацию RL value функции в этой вершине + результат прохода до конца z.

$$V(s_L) = (1 - \lambda)v_\theta(s_L) + \lambda z_L$$

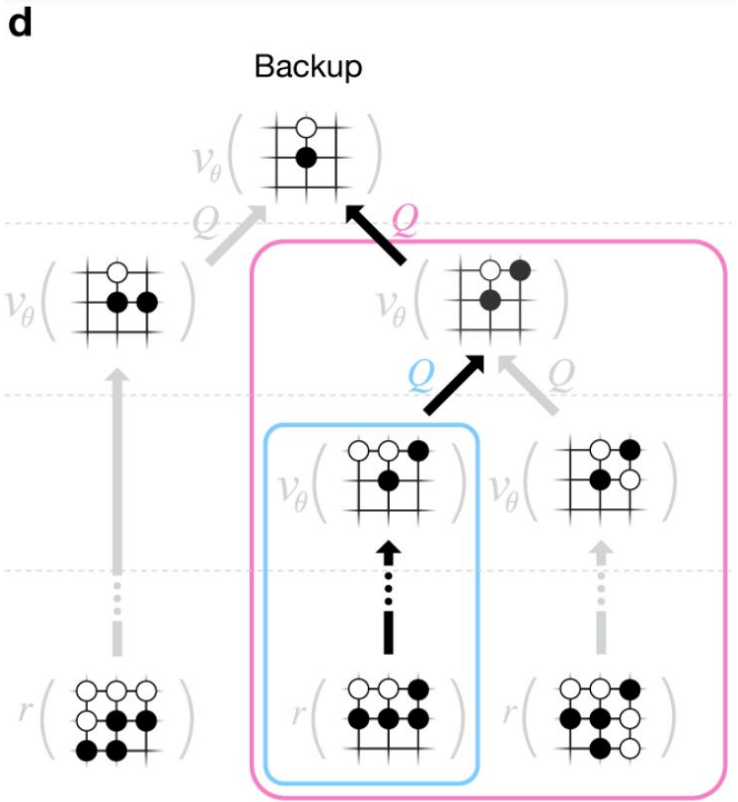


AlphaGo MCTS Backup

Из листа поднимаемся обратно, обновляя Q и число проходов по вершинам N.

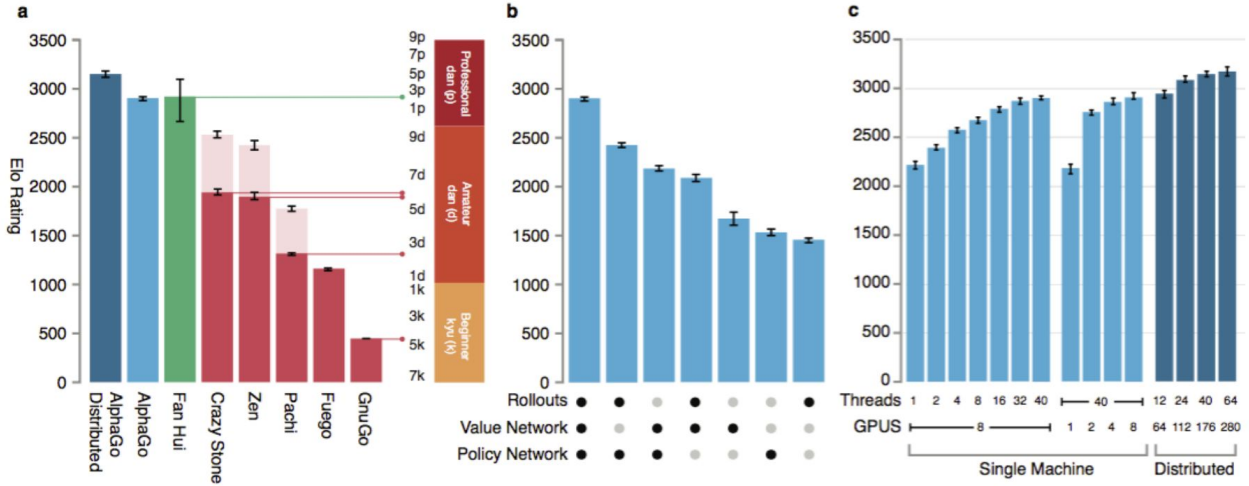
$$N(s,a) = \sum_{i=1}^{n} \mathbf{1}(s,a,i)$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} \mathbf{1}(s,a,i) V(s_L^i)$$



AlphaGo

Вычисления распараллеливаются на 40 потоков
2 версии: AlphaGo с 48 CPU, 8 GPU и AlphaGo distributed (впоследствии назвали AlphaGo Fan) с 1202 CPU, 176 GPU
Выиграла 5-0 у чемпиона Европы Fan Hui (2 проф. дан),
но в играх с более жестким контролем времени сыграла 3-2



AlphaGo Lee

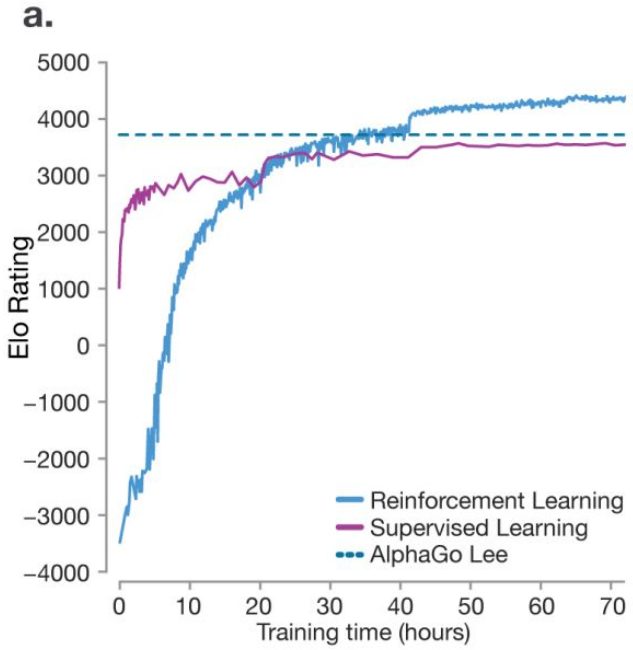
Улучшенная (после выхода статьи) версия, обыграла 4-1 одного из лучших игроков мира Ли Седоля (9 проф. дан, 2 по рейтингу эло)
Использовала нейронки побольше, вместо GPU использовала 48 TPU.

AlphaGo Master

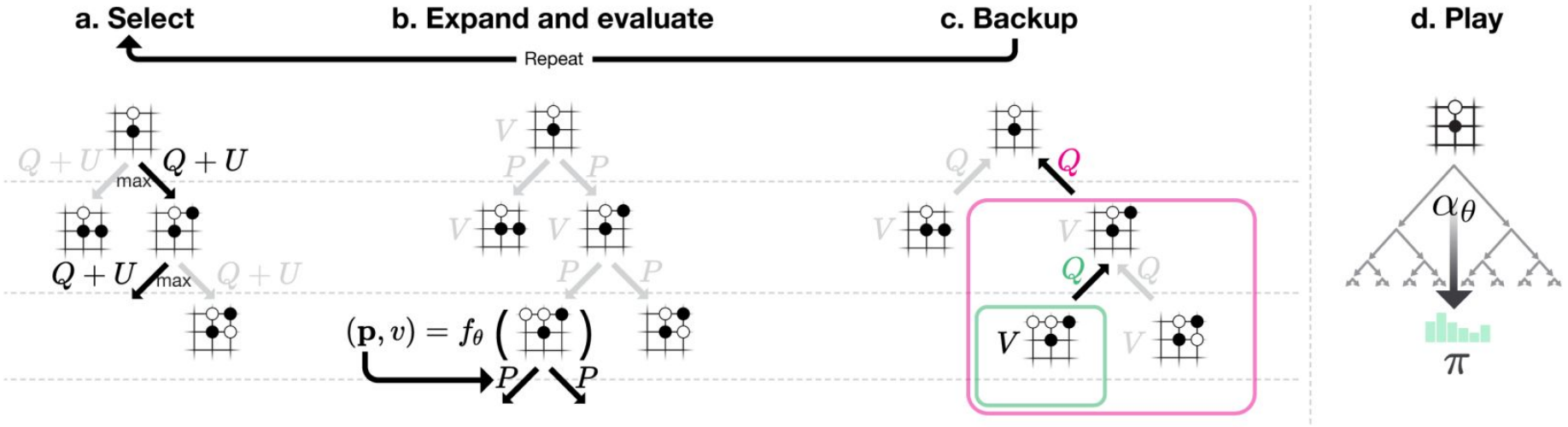
Версия обыгравшая 60-0 лучших игроков мира в онлайн матчах. Использовала улучшенный алгоритм MCTS по сравнению с прошлыми версиями

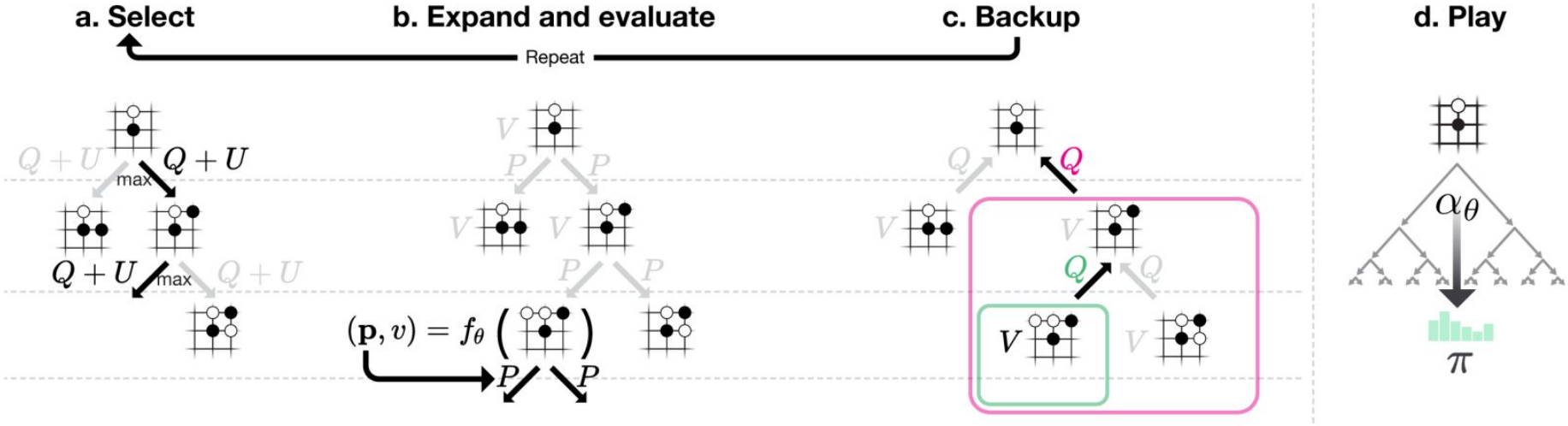
AlphaGo Zero

Использует улучшенный MCTS из Master версии, не использует дополнительные фичи, обучается с помощью self play с нуля не используя записи игр, использует только одну нейронку.
Для игры использует всего 4 TPU.

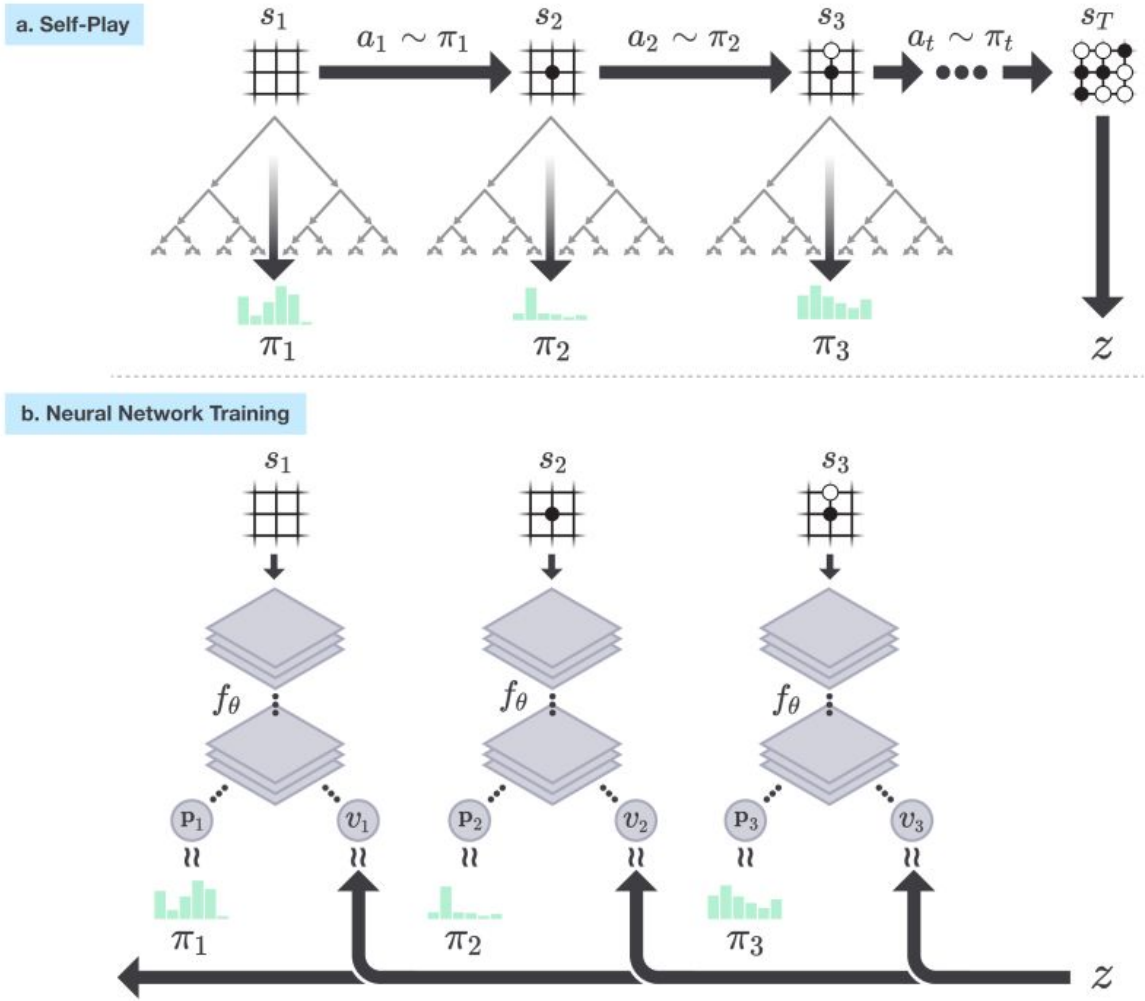


Улучшенный MTCS





Ключевые отличия: в новых вершинах не играем до конца а считаем value исключительно на основе выхода нейросети
Для получения оптимального хода используется только частота прохода по вершинам, Q не используется


a. Self-Play
s_1
$a_1 \sim \pi_1$
s_2
$a_2 \sim \pi_2$
s_3
$a_t \sim \pi_t$
s_T
π_1
π_2
π_3
z
b. Neural Network Training
s_1
s_2
s_3
f_θ
f_θ
f_θ
p_1
v_1
p_2
v_2
p_3
v_3
π_1
π_2
π_3
z

Доп фичи

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).

Источники

1. Silver, D., Huang, A., Maddison, C. *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 484–489 (2016)
2. Silver, D., Schrittwieser, J., Simonyan, K. *et al.* Mastering the game of Go without human knowledge. *Nature* 550, 354–359 (2017).