AlphaGo Zero

Антон Урусов

15 января 2018 г.

План

- Алгоритм AlphaGo Zero
- Отличия от предыдущих версий
- Сравнение с предыдущими версиями по качеству

Основа алгоритма

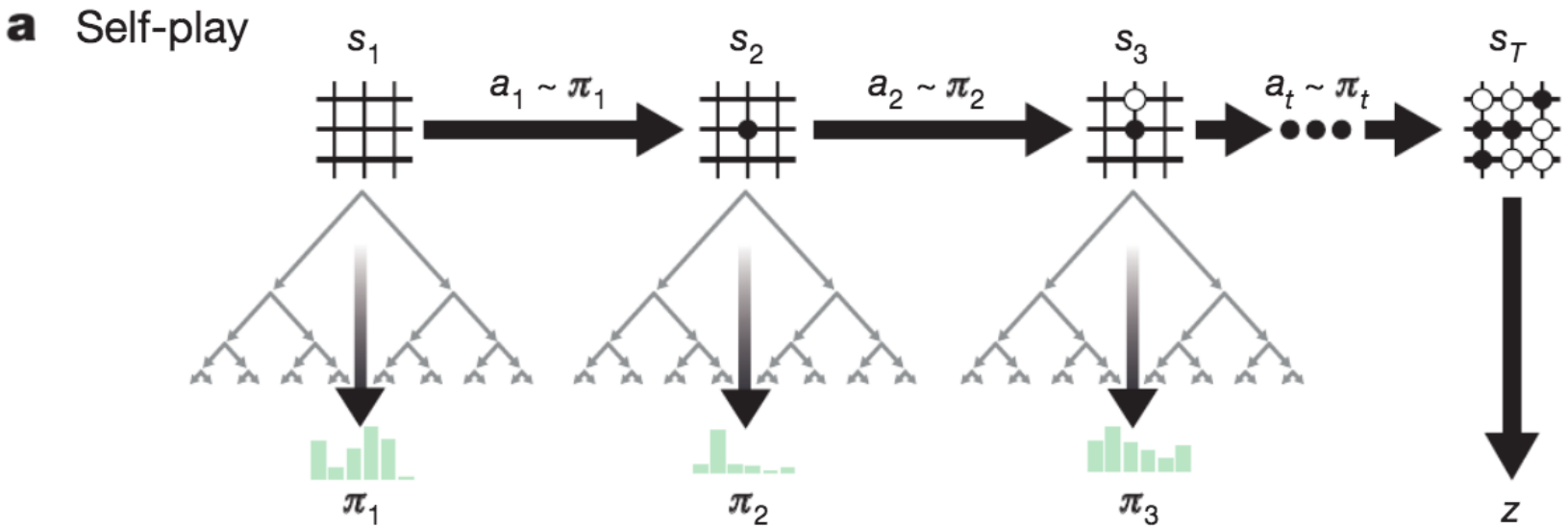
В основе алгоритма лежит глубокая нейросеть f_θ, которая принимает состояние доски как input, и возвращает (p, v), где p - вероятности ходов из данного состояния, включая пропуск хода, а v - вероятность выигрыша из этой позиции.

Общая схема алгоритма

Общая схема обучения следующая:

1. Играем несколько игр с собой
2. Обновляем веса модели и возвращаемся к шагу 1.

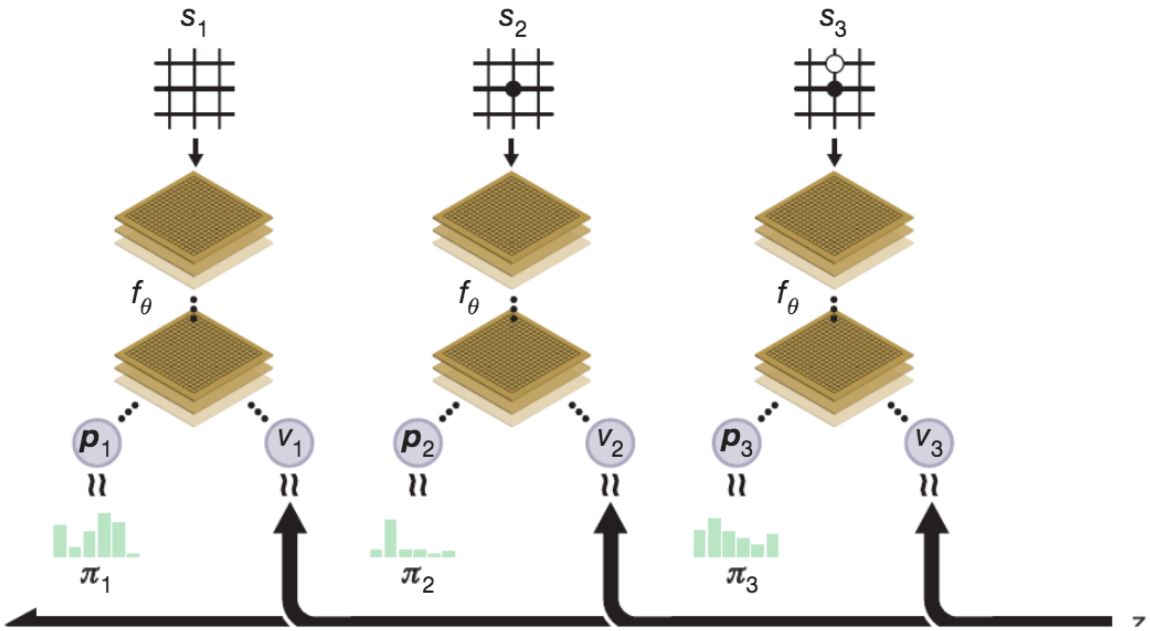
Общая схема игры следующая:



Обновление весов

Обновление весов происходит следующим образом: мы из игр с собой запоминаем для каждого пройденного состояния s_i более точные оценки на вероятности ходов π_i, а также можем легко определить, кто выиграл для этого состояния ($\pm z$). Эти значения используем как обучающую выборку для сети, и обновляем модель градиентным спуском. Оптимизируемый функционал:

$$(z - v)^2 - \pi^T \log p + c\|\theta\|^2 \tag{1}$$



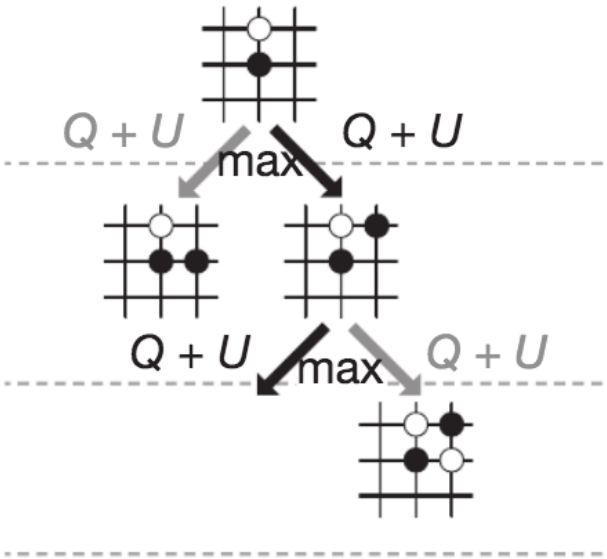
Игра с собой: MCTS

На каждом шаге мы хотим более точно оценить вероятности ходов с помощью Monte-Carlo tree search. Для каждой вершины s текущего дерева и для всех действий a храним следующие значения:

- $P(s, a)$ - априорная вероятность каждого хода
- $Q(s, a)$ - ценность каждого действия в состоянии
- $N(s, a)$ - сколько раз из данной вершины мы выбрали действие a.

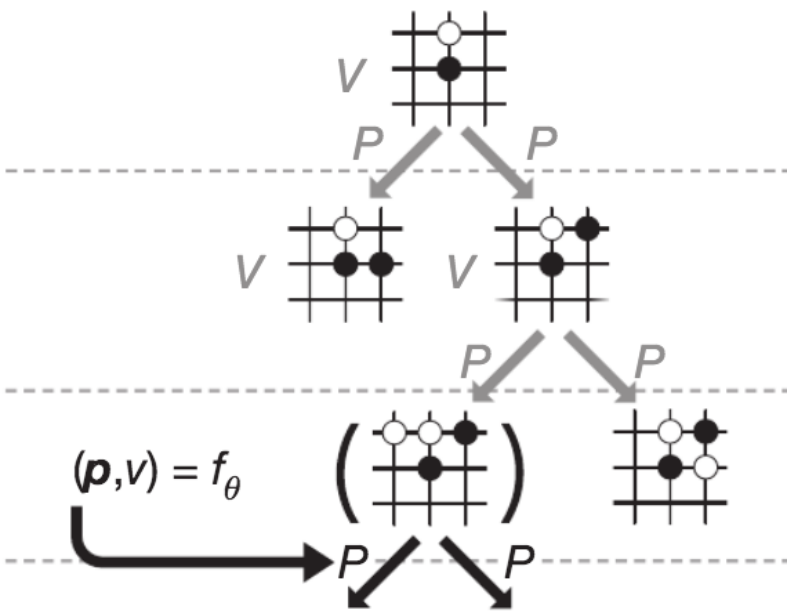
Далее итеративно обновляем дерево путем запуска симуляций, о которых подробнее ниже.

Select



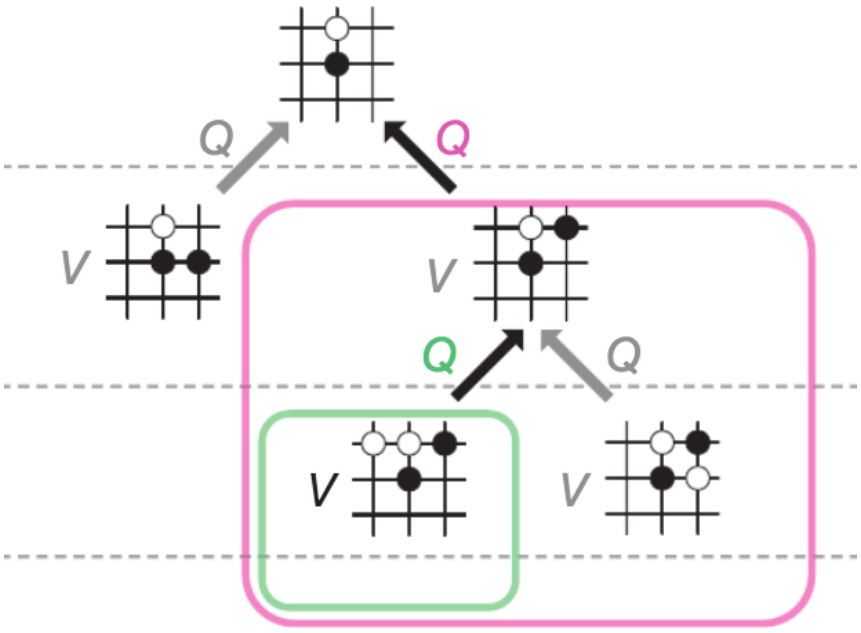
В
каждой симуляции начинаем
в корневом состоянии
s_0. Оттуда идем вниз по
дереву, выбирая действие как
$\arg\max_a(Q(s,a) + U(s,a))$,
где $Q(s,a)$ - текущая
оценка ценности действия, а
$U(s,a) = CP(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$.
Так действуем,
пока не дойдем до листа.

Expand and evaluate



В листе инициализируем $P(s, a)$ нейросетью, а остальные значения - нулями, а также получаем из нейросети значение v, оценивающее ценность действия.

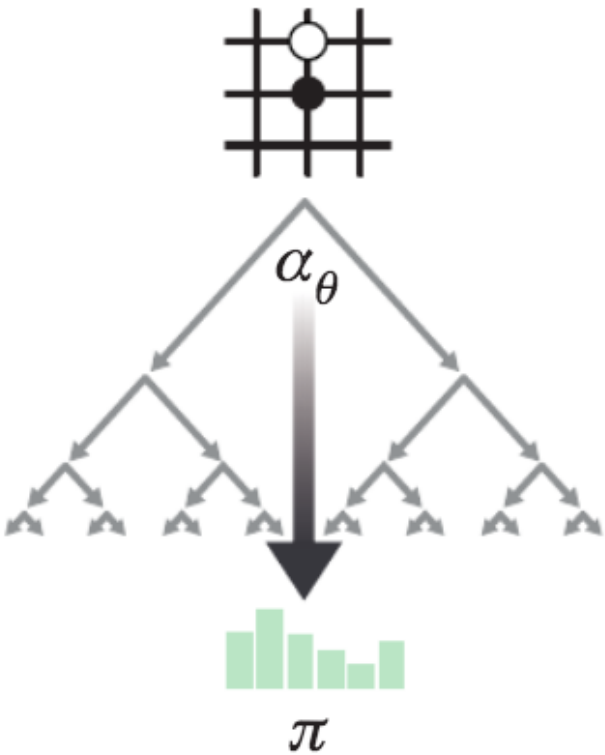
Backup



Далее обновляем значения N, Q для всех состояний, пройденных на этой стимуляции: $N(s_t, a_t) += 1$, $W(s_t, a_t) = W(s_t, a_t)$, где $W(s, t) = Q(s, t) * N(s, t)$. Таким образом, в $Q(s, t)$ у нас будет лежать среднее из всех v, полученных в потомках.

Play

После какого-то количества стимуляций, получаем $\pi(a)$ пропорционально $N(s_0, a)^{1/\tau}$, и делаем ход из этого распределения.



Ключевые отличия от предыдущих версий

- Обучается только с помощью self-play, то есть не использует никакой информации от human experts
- Использует только позиции черных и белых камней в качестве входных признаков
- Одна нейросеть, а не две
- Использует более простой алгоритм поиска в дереве Монте-Карло.

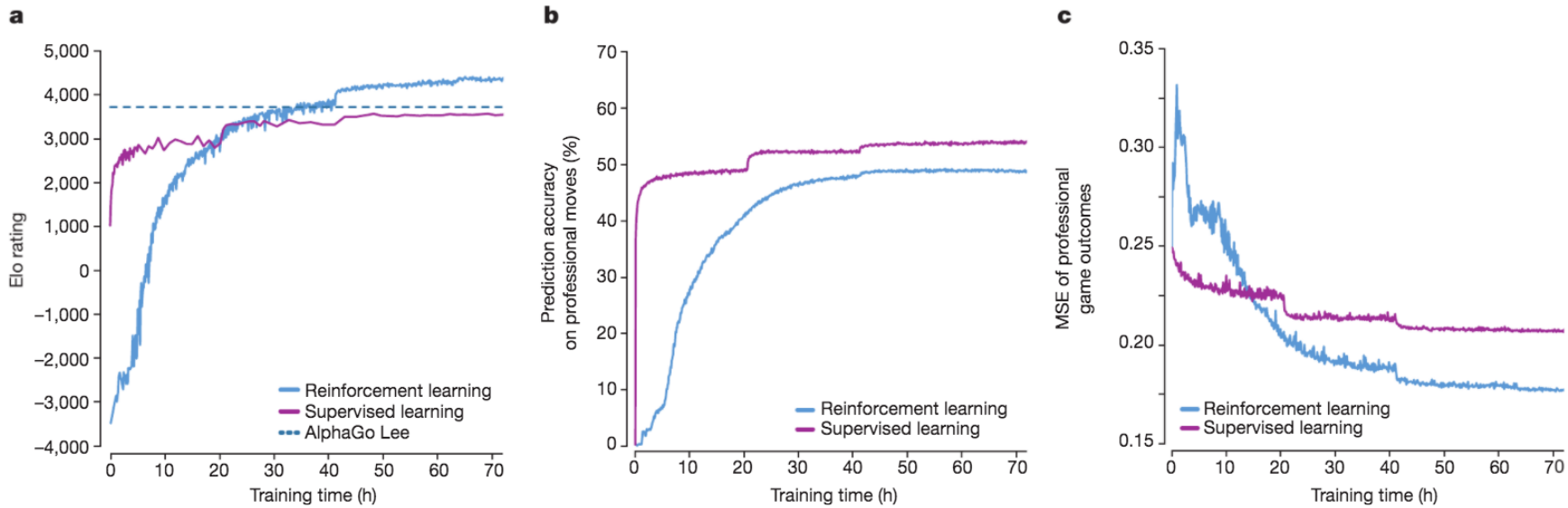
Методы для оценивания результатов

- Для оценивания возьмем полученный алгоритм α_θ, а также несколько предыдущих версий AlphaGo, проведем турнир и посчитаем ELO-рейтинг нашего алгоритма. ELO-рейтинг остальных участников берем из соответствующих статей.
- Кроме того, мы будем сравнивать качество самой обученной сети AlphaGo Zero с сетью такой же архитектуры, но обученной по датасету. Сравнивать будем по двум метрикам:
  - точность предсказания хода
  - MSE предсказания исхода партии.

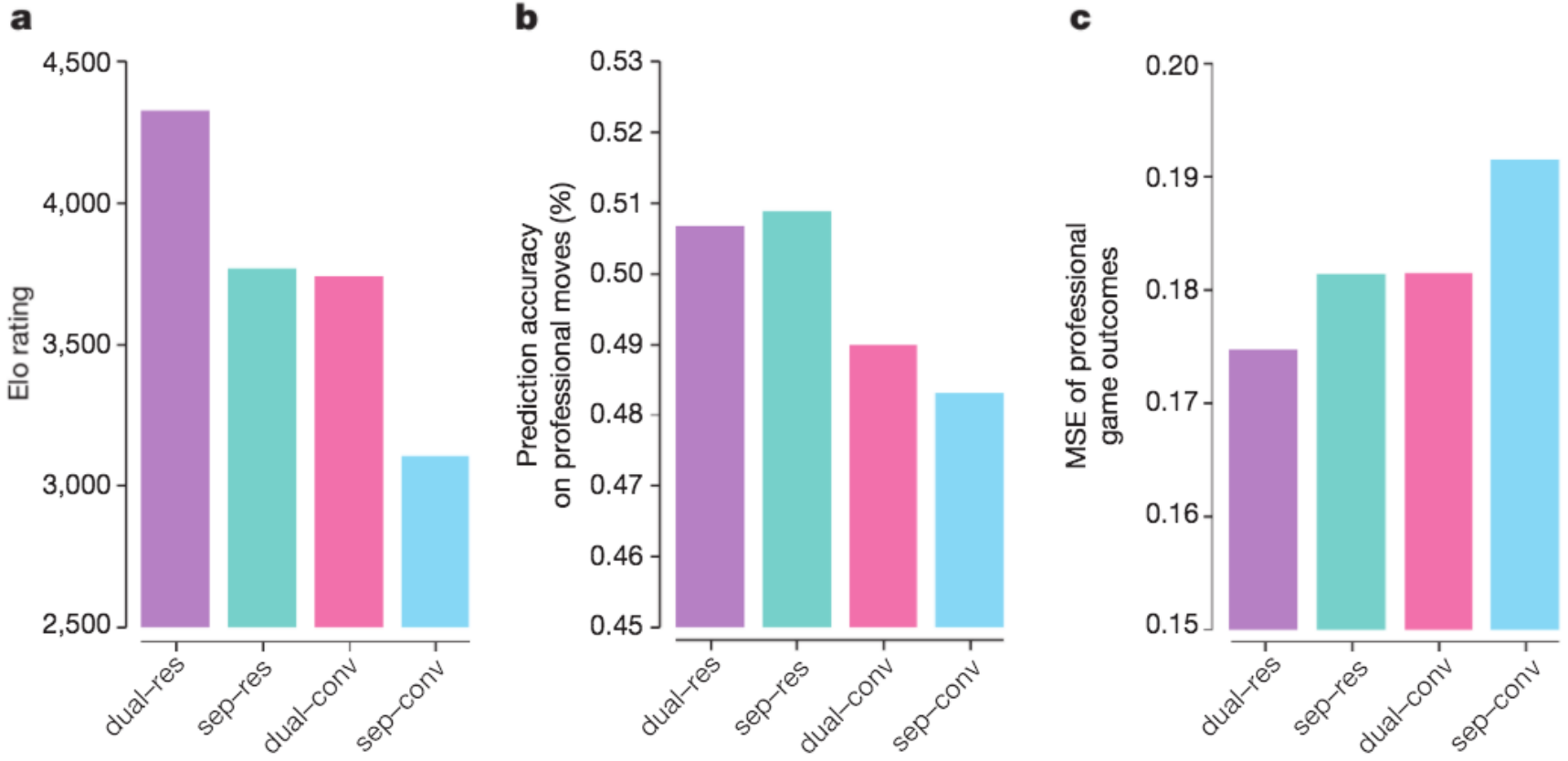
Resources for learning

- AlphaGo Zero использовала 4 TPU на одной машине
- AlphaGo Lee использовала 48 TPU на нескольких машинах и обучалась несколько месяцев.

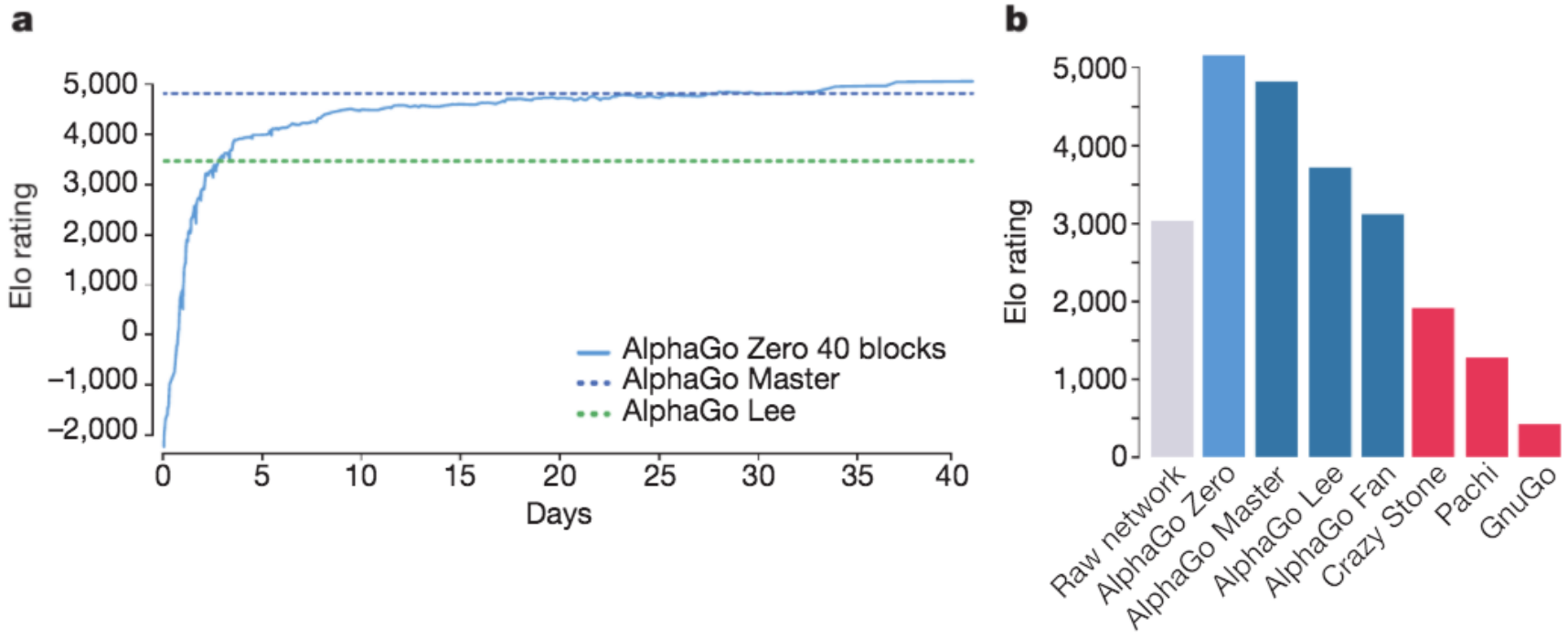
Experiments: performance



Experiments: разные архитектуры



Experiments: final performance



References

Mastering the game of Go without human knowledge (https://deepmind.com/research/publications/mastering-game-go-without-human-knowledge/)