ELF OpenGo: An Analysis and Open Reimplementation of AlphaZero

Yuandong Tian, Jerry Ma*, Qucheng Gong*, Shubho Sengupta*,
Zhuoyuan Chen, James Pinkerton, Larry Zitnick

Presented by Yuandong Tian

*Equal Contributions

DeepMind

AlphaGo, 2016

ARTICLE

doi:10.1038/nature16961

Mastering the game of Go with deep neural networks and tree search

David Silver[1]*, Aja Huang[1]*, Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s, under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game's breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35$, $d \approx 80$)[1] and especially Go ($b \approx 250$, $d \approx 150$)[1], exhaustive search is infeasible[2,3], but the effective search space can be reduced by two general principles. First, the depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s. This approach has led to superhuman performance in chess[4], checkers[5] and othello[6], but it was believed to be intractable in Go due to the complexity of the game[7]. Second, the breadth of the search may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s. For example, Monte Carlo rollouts[8] search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p. Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance in backgammon[8] and Scrabble[9], and weak amateur level play in Go[10].

Monte Carlo tree search (MCTS)[11,12] uses Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant values become more accurate. The policy used to select actions during search is also improved over time, by selecting children with higher values. Asymptotically, this policy converges to optimal play, and the evaluations converge to the optimal value function[12]. The strongest current Go programs are based on MCTS, enhanced by policies that are trained to predict human expert moves[13]. These policies are used to narrow the search to a beam of high-probability actions, and to sample actions during rollouts. This approach has achieved strong amateur play[13–15]. However, prior work has been limited to shallow policies[13–15] or value functions[16] based on a linear combination of input features.

Recently, deep convolutional neural networks have achieved unprecedented performance in visual domains: for example, image classification[17], face recognition[18], and playing Atari games[19]. They use many layers of neurons, each arranged in overlapping tiles, to construct increasingly abstract, localized representations of an image[20]. We employ a similar architecture for the game of Go. We pass in the board position as a 19×19 image and use convolutional layers to construct a representation of the position. We use these neural networks to reduce the effective depth and breadth of the search tree: evaluating positions using a value network, and sampling actions using a policy network.

We train the neural networks using a pipeline consisting of several stages of machine learning (Fig. 1). We begin by training a supervised learning (SL) policy network p_σ directly from expert human moves. This provides fast, efficient learning updates with immediate feedback and high-quality gradients. Similar to prior work[13,15], we also train a fast policy p_π that can rapidly sample actions during rollouts. Next, we train a reinforcement learning (RL) policy network p_ρ that improves the SL policy network by optimizing the final outcome of games of self-play. This adjusts the policy towards the correct goal of winning games, rather than maximizing predictive accuracy. Finally, we train a value network v_θ that predicts the winner of games played by the RL policy network against itself. Our program AlphaGo efficiently combines the policy and value networks with MCTS.

Supervised learning of policy networks

For the first stage of the training pipeline, we build on prior work on predicting expert moves in the game of Go using supervised learning[13,21–24]. The SL policy network $p_\sigma(a|s)$ alternates between convolutional layers with weights σ, and rectifier nonlinearities. A final softmax layer outputs a probability distribution over all legal moves a. The input s to the policy network is a simple representation of the board state (see Extended Data Table 2). The policy network is trained on randomly

[1]Google DeepMind, 5 New Street Square, London EC4A 3TW, UK. [2]Google, 1600 Amphitheatre Parkway, Mountain View, California 94043, USA.
*These authors contributed equally to this work.

© 2016 Macmillan Publishers Limited. All rights reserved

AlphaGo Zero, 2017

No human knowledge

ARTICLE

doi:10.1038/nature24270

Mastering the game of Go without human knowledge

David Silver[1*], Julian Schrittwieser[1*], Karen Simonyan[1*], Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

Much progress towards artificial intelligence has been made using supervised learning systems that are trained to replicate the decisions of human experts[1–4]. However, expert data sets are often expensive, unreliable or simply unavailable. Even when reliable data sets are available, they may impose a ceiling on the performance of systems trained in this manner[5]. By contrast, reinforcement learning systems are trained from their own experience, in principle allowing them to exceed human capabilities, and to operate in domains where human expertise is lacking. Recently, there has been rapid progress towards this goal, using deep neural networks trained by reinforcement learning. These systems have outperformed humans in computer games, such as Atari[6,7] and 3D virtual environments[8–10]. However, the most challenging domains in terms of human intellect—such as the game of Go, widely viewed as a grand challenge for artificial intelligence[11]—require a precise and sophisticated lookahead in vast search spaces. Fully general methods have not previously achieved human-level performance in these domains.

AlphaGo was the first program to achieve superhuman performance in Go. The published version[12], which we refer to as AlphaGo Fan, defeated the European champion Fan Hui in October 2015. AlphaGo Fan used two deep neural networks: a policy network that outputs move probabilities and a value network that outputs a position evaluation. The policy network was trained initially by supervised learning to accurately predict human expert moves, and was subsequently refined by policy-gradient reinforcement learning. The value network was trained to predict the winner of games played by the policy network against itself. Once trained, these networks were combined with a Monte Carlo tree search (MCTS)[13–15] to provide a lookahead search, using the policy network to narrow down the search to high-probability moves, and using the value network (in conjunction with Monte Carlo rollouts using a fast rollout policy) to evaluate positions in the tree. A subsequent version, which we refer to as AlphaGo Lee, used a similar approach (see Methods), and defeated Lee Sedol, the winner of 18 international titles, in March 2016.

Our program, AlphaGo Zero, differs from AlphaGo Fan and AlphaGo Lee[12] in several important aspects. First and foremost, it is trained solely by self-play reinforcement learning, starting from random play, without any supervision or use of human data. Second, it uses only the black and white stones from the board as input features. Third, it uses a single neural network, rather than separate policy and value networks. Finally, it uses a simpler tree search that relies upon this single neural network to evaluate positions and sample moves, without performing any Monte Carlo rollouts. To achieve these results, we introduce a new reinforcement learning algorithm that incorporates lookahead search inside the training loop, resulting in rapid improvement and precise and stable learning. Further technical differences in the search algorithm, training procedure and network architecture are described in Methods.

Reinforcement learning in AlphaGo Zero

Our new method uses a deep neural network f_θ with parameters θ. This neural network takes as an input the raw board representation s of the position and its history, and outputs both move probabilities and a value, $(\boldsymbol{p}, v) = f_\theta(s)$. The vector of move probabilities $\boldsymbol{p}$ represents the probability of selecting each move a (including pass), $p_a = \Pr(a|s)$. The value v is a scalar evaluation, estimating the probability of the current player winning from position s. This neural network combines the roles of both policy network and value network[12] into a single architecture. The neural network consists of many residual blocks[4] of convolutional layers[16,17] with batch normalization[18] and rectifier nonlinearities[19] (see Methods).

The neural network in AlphaGo Zero is trained from games of self-play by a novel reinforcement learning algorithm. In each position s, an MCTS search is executed, guided by the neural network f_θ. The MCTS search outputs probabilities π of playing each move. These search probabilities usually select much stronger moves than the raw move probabilities $\boldsymbol{p}$ of the neural network $f_\theta(s)$; MCTS may therefore be viewed as a powerful policy improvement operator[20,21]. Self-play with search—using the improved MCTS-based policy to select each move, then using the game winner z as a sample of the value—may be viewed as a powerful policy evaluation operator. The main idea of our reinforcement learning algorithm is to use these search operators

[1]DeepMind, 5 New Street Square, London EC4A 3TW, UK.
*These authors contributed equally to this work.

© 2017 Macmillan Publishers Limited, part of Springer Nature. All rights reserved.

A general reinforcement learning algorithm that masters chess, shogi and Go through self-play

David Silver,[1,2*] Thomas Hubert,[1*] Julian Schrittwieser,[1*]
Ioannis Antonoglou,[1,2] Matthew Lai,[1] Arthur Guez,[1] Marc Lanctot,[1]
Laurent Sifre,[1] Dharshan Kumaran,[1,2] Thore Graepel,[1,2]
Timothy Lillicrap,[1] Karen Simonyan,[1] Demis Hassabis[1]

[1]DeepMind, 6 Pancras Square, London N1C 4AG.
[2]University College London, Gower Street, London WC1E 6BT.
*These authors contributed equally to this work.

Abstract

The game of chess is the longest-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. By contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go by reinforcement learning from self-play. In this paper, we generalize this approach into a single AlphaZero algorithm that can achieve superhuman performance in many challenging games. Starting from random play and given no domain knowledge except the game rules, AlphaZero convincingly defeated a world champion program in the games of chess and shogi (Japanese chess) as well as Go.

The study of computer chess is as old as computer science itself. Charles Babbage, Alan Turing, Claude Shannon, and John von Neumann devised hardware, algorithms and theory to analyse and play the game of chess. Chess subsequently became a grand challenge task for a generation of artificial intelligence researchers, culminating in high-performance computer chess programs that play at a super-human level (*1, 2*). However, these systems are highly tuned to their domain, and cannot be generalized to other games without substantial human effort, whereas general game-playing systems (*3, 4*) remain comparatively weak.

A long-standing ambition of artificial intelligence has been to create programs that can instead learn for themselves from first principles (*5, 6*). Recently, the AlphaGo Zero algorithm achieved superhuman performance in the game of Go, by representing Go knowledge using deep convolutional neural networks (*7, 8*), trained solely by reinforcement learning from games

1

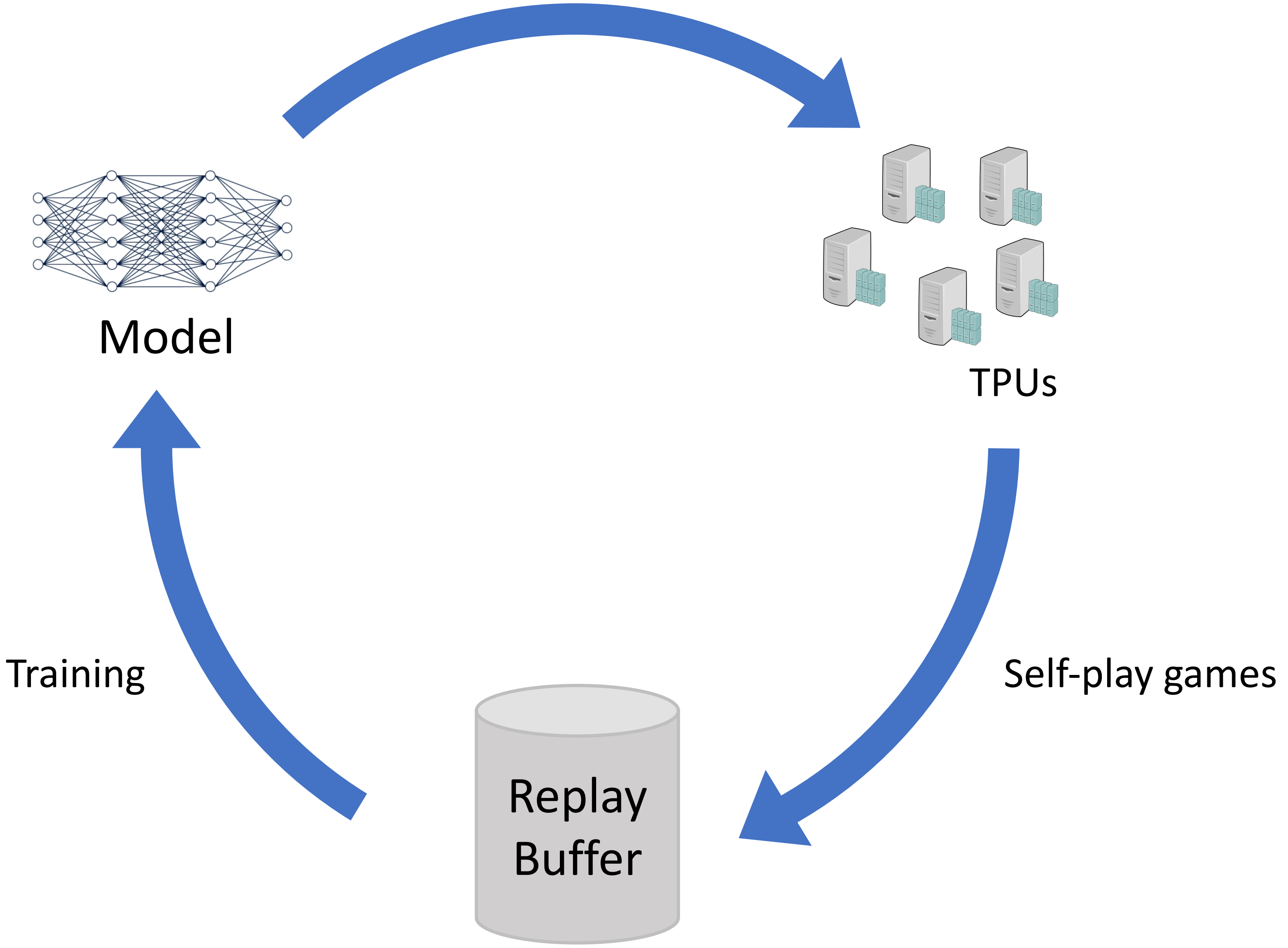
DeepMind

AlphaZero, 2018

Generalization to other games

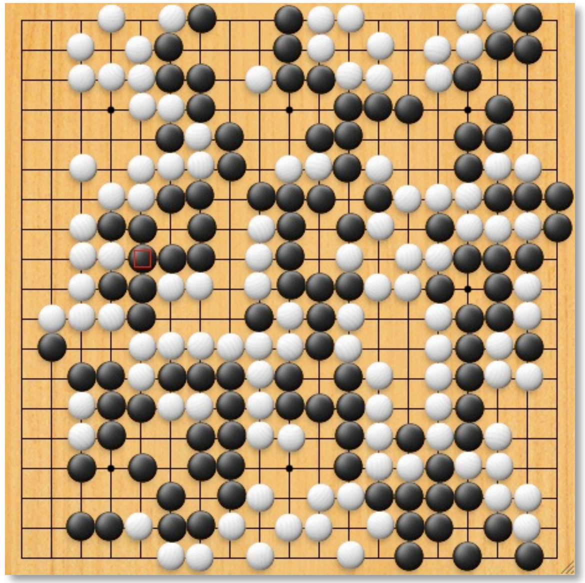
AlphaZero

Learning without human knowledge



Silver et al., A general reinforcement learning algorithm that masters chess, shogi and Go through self-play, *Science 2018*

Model



s

Residual Network
20 or 40 blocks

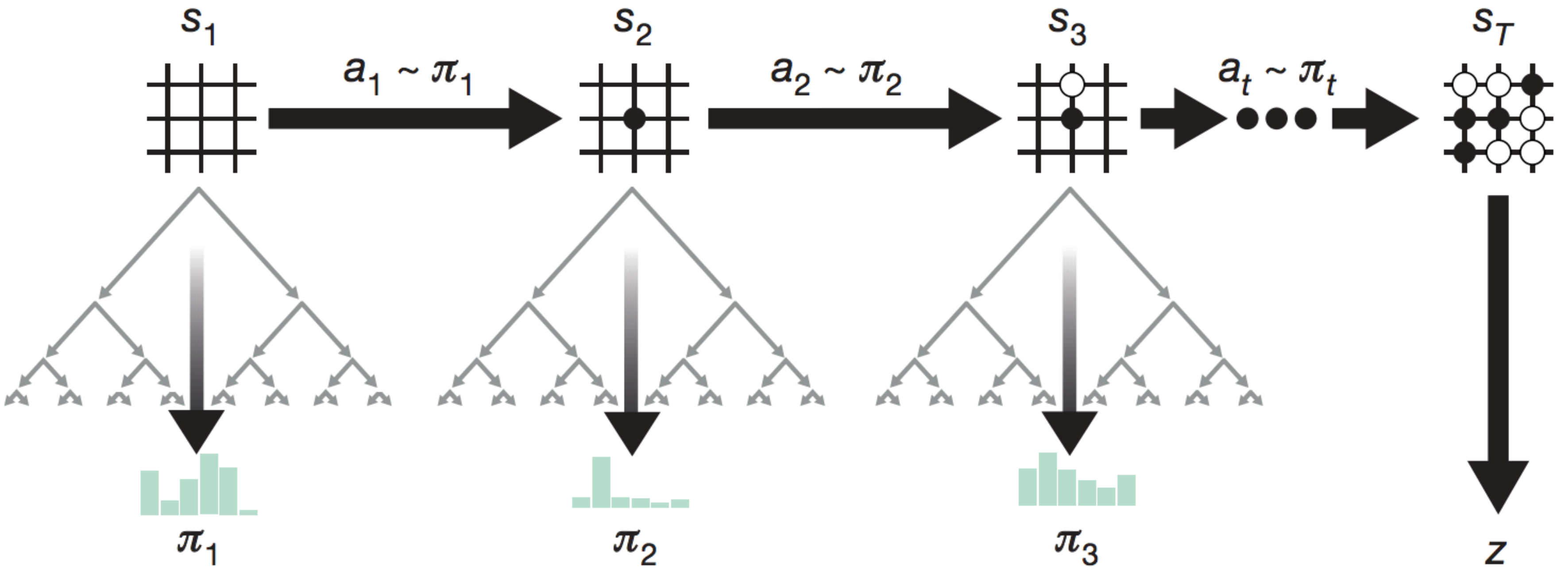
$\mathbf{p}_\theta(s)$ Policy

$V_\theta(s)$ Value

Silver et al., A general reinforcement learning algorithm that masters chess, shogi and Go through self-play, *Science 2018*

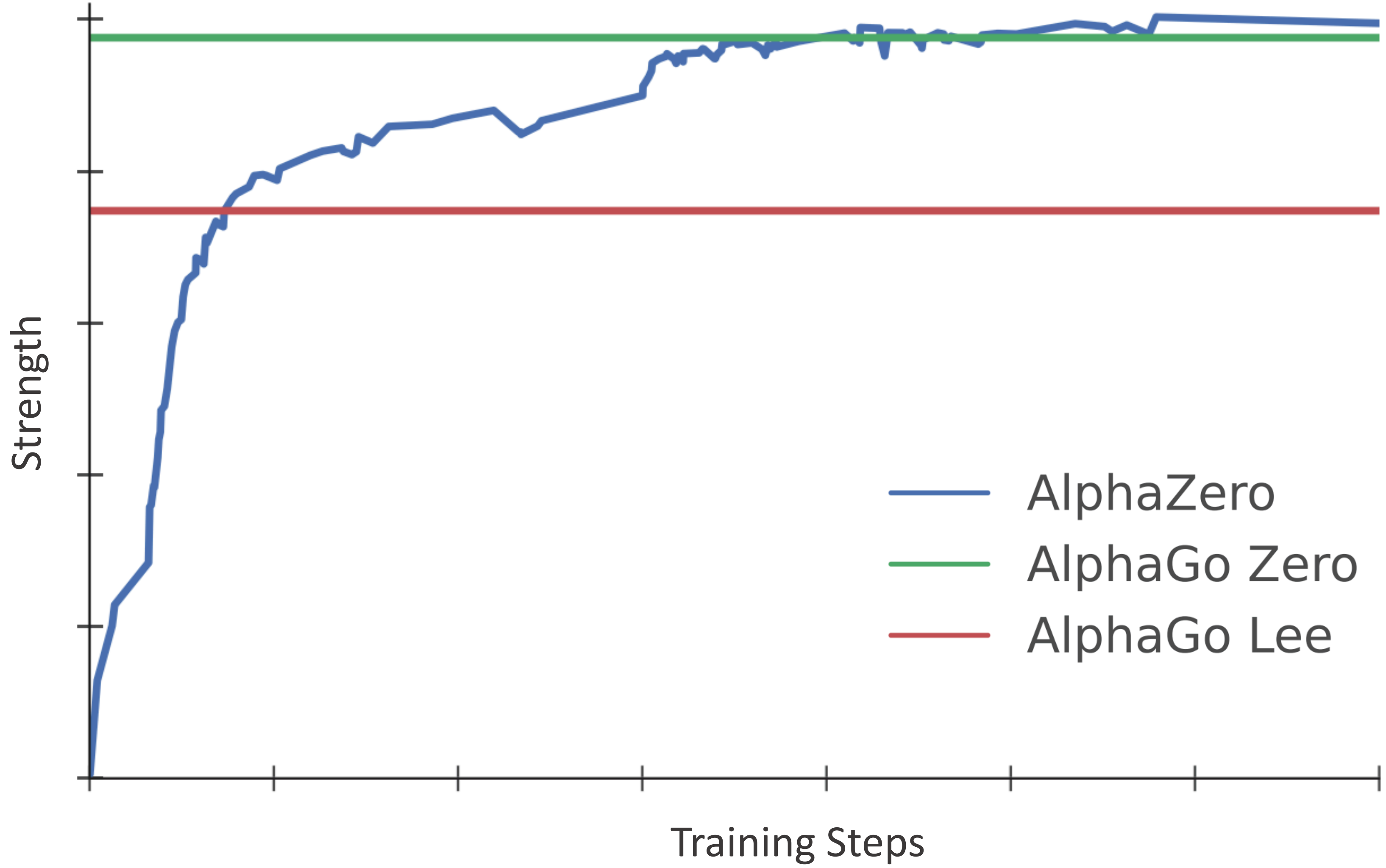
Generating Self-play Games

Monte Carlo Tree Search with most recent model



Silver et al., A general reinforcement learning algorithm that masters chess, shogi and Go through self-play, *Science 2018*

AlphaZero Strength



Silver et al., A general reinforcement learning algorithm that masters chess, shogi and Go through self-play, *Science 2018*

Self-play games

AlphaGo Zero trained from 4.9 million self-play games!

~150,000 hours* of GPU time!

*Facebook estimates

ELF OpenGo

Reproduce AlphaZero

Gain a deeper understanding

Enable the community

ELF OpenGo

Reproduce AlphaZero

Gain a deeper understanding

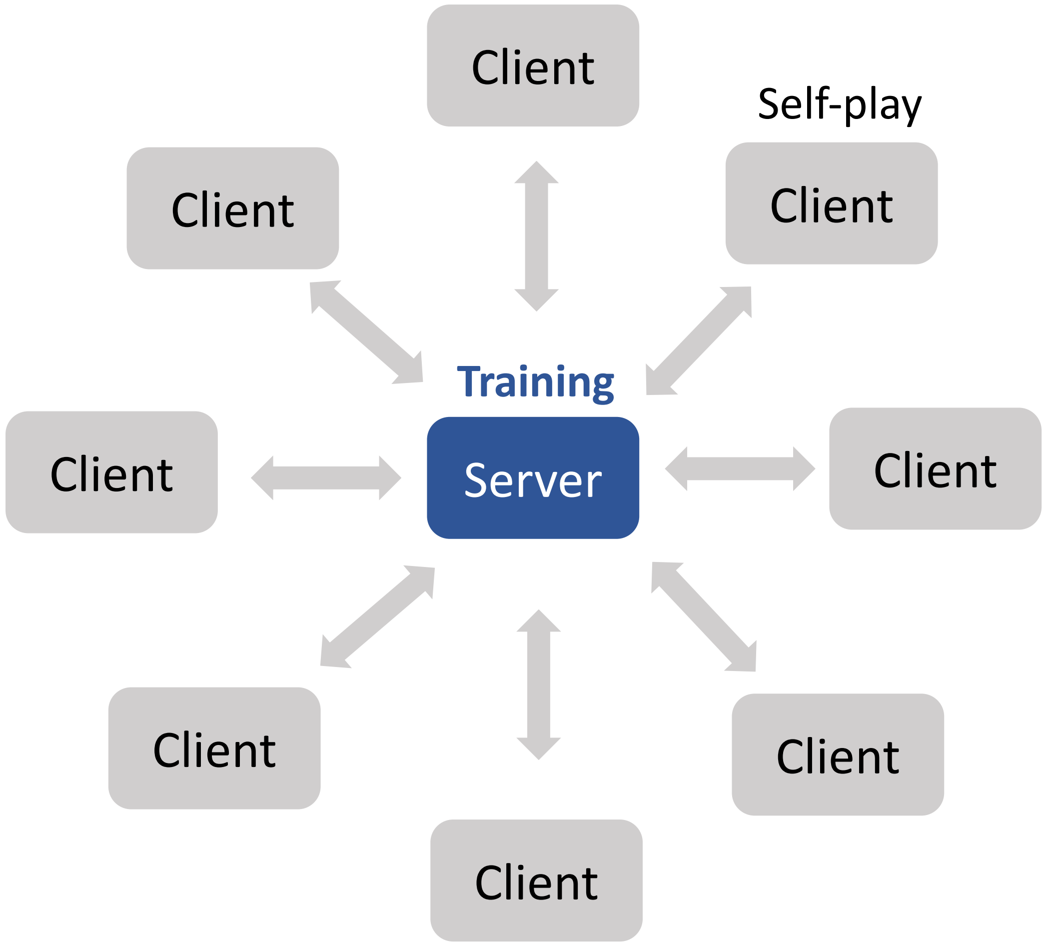
Enable the community

ELF OpenGo

- 20 block ResNet model
- 2,000 GPUs, 2 weeks
- 20 million self-play games

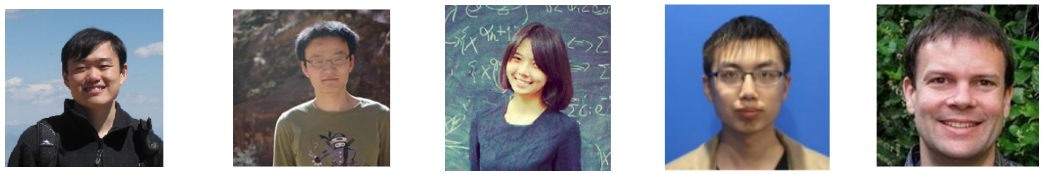
	AlphaGo Zero	AlphaZero	*ELF OpenGo*
c_{puct}	?	?	1.5
MCTS virtual loss constant	?	?	1.0
MCTS rollouts (self-play)	1,600	800	1,600
Replay buffer size	500,000	?	500,000
Training minibatch size	2048	4096	2048
Self-play hardware	?	5,000 TPUs	2,000 GPUs
Training hardware	64 GPUs	64 TPUs	8 GPUs

ELF Distributed System



Server
- Receives self-play games
- Trains and broadcasts models

Client
- Receives model updates
- Performs self-play



Yuandong Tian Qucheng Gong Wenling Shang Yuxin Wu Larry Zitnick

[Y. Tian et al, ELF: An Extensive, Lightweight and Flexible Research Platform for Real-time Strategy Games, NeurIPS 2017]

ELF OpenGo Timeline

2017

Nov

AlphaZero
paper released

ELF OpenGo Timeline

2017 | 2018

Nov — AlphaZero paper released

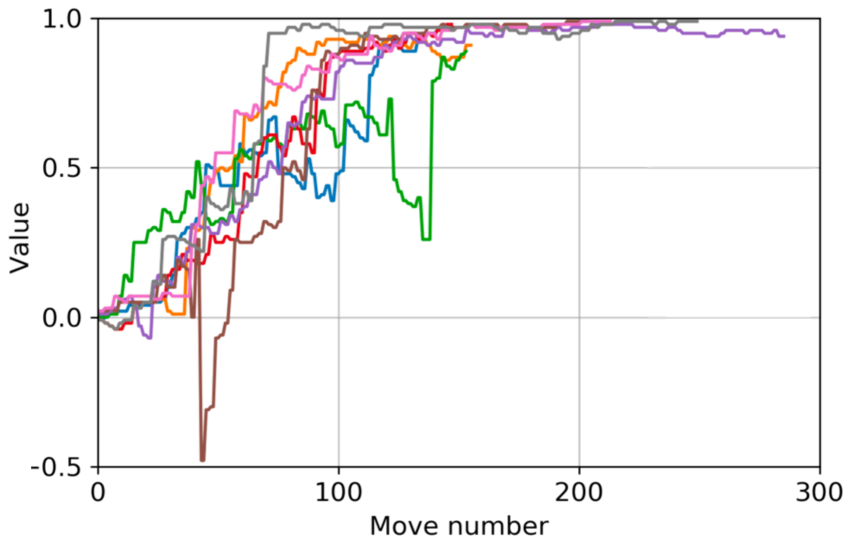
Jan — **ELF OpenGo Starts**

Feb — Amateur level

Apr — Prototype model wins against professional players

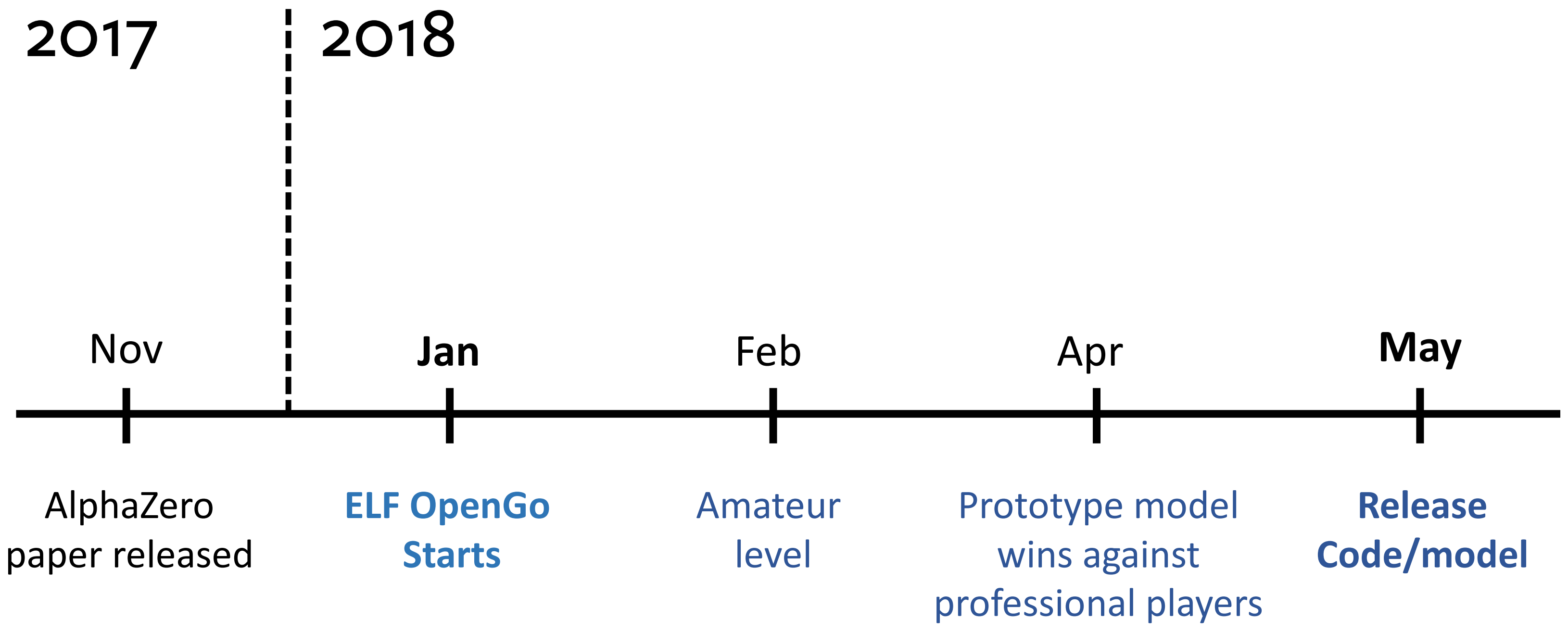
20-0 win rate

#3 Kim Ji-seok
#5 Shin Jin-seo
#23 Park Yeonghun
#30 Choi Cheolhan

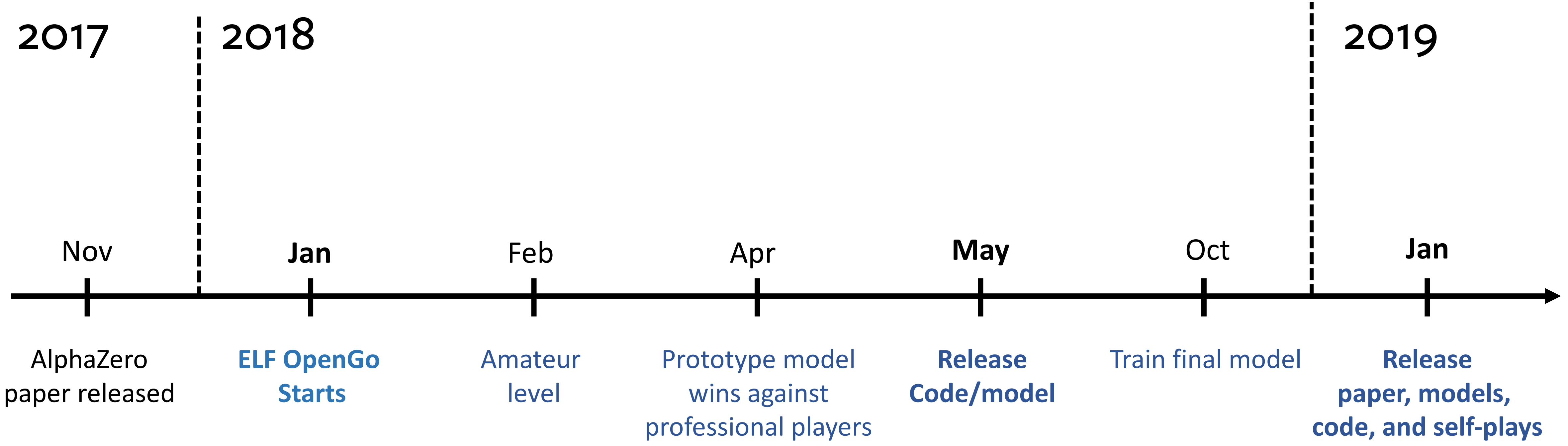


- Single V100 GPU
- 80k rollouts
- 50 seconds
- Unlimited thinking time for human players

ELF OpenGo Timeline



ELF OpenGo Timeline



facebook Artificial Intelligence

ELF OpenGo

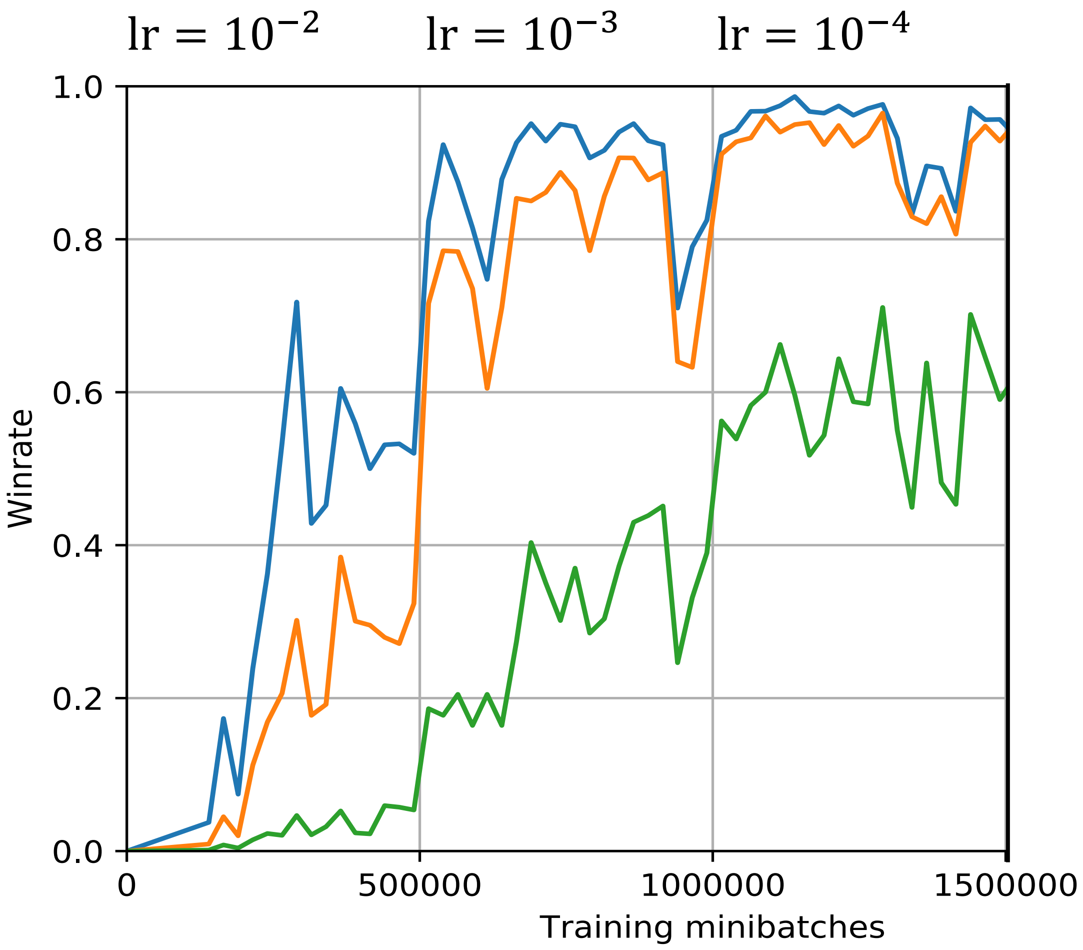
Reproduce AlphaZero

Gain a deeper understanding

Enable the community

High-variance in training

High-variance in training



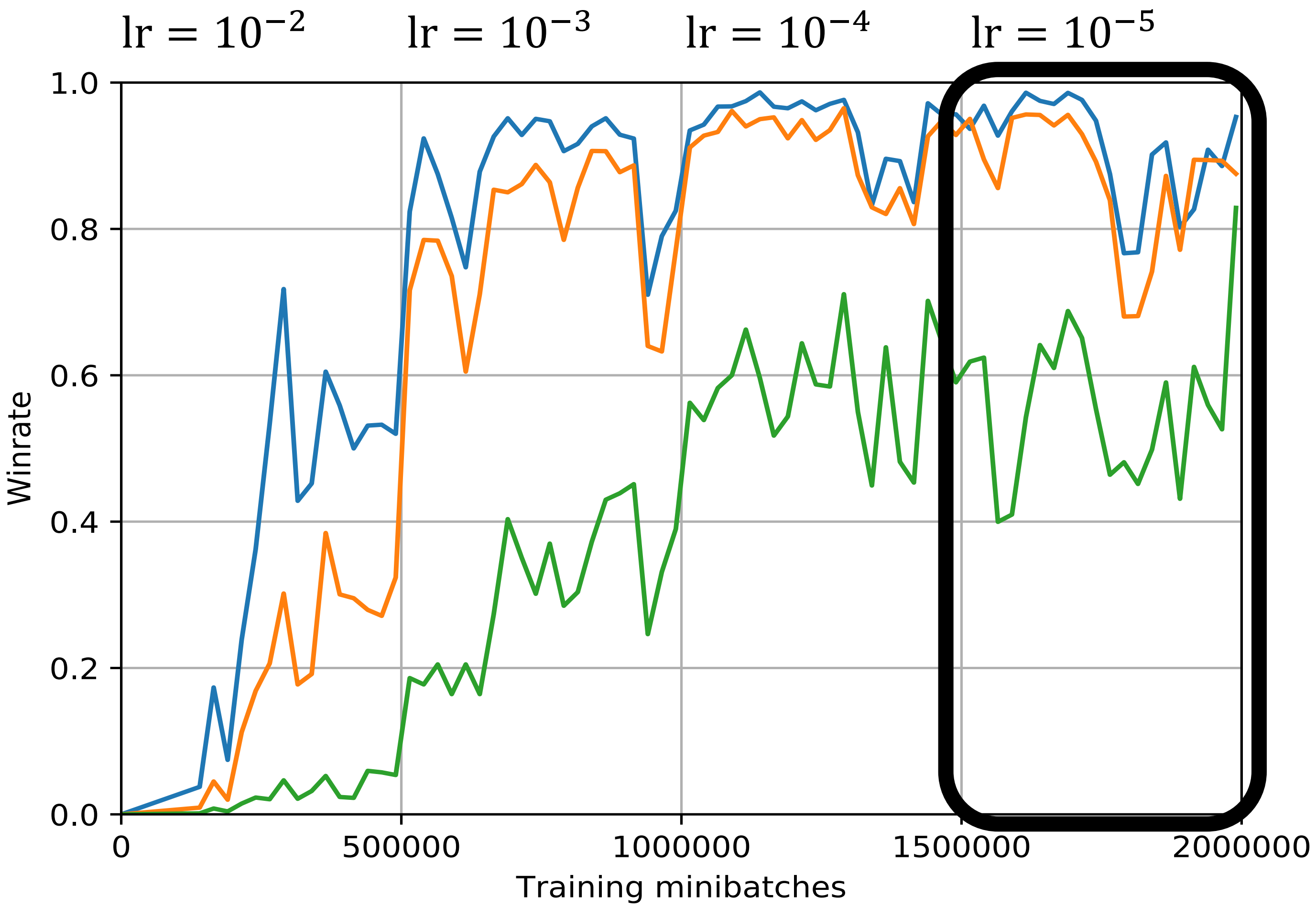
Strong amateur level

Professional level

Superhuman level
(won vs. professional players)

Learning rate dropped every
500k mini-batches (10^{-2}, 10^{-3}, 10^{-4})

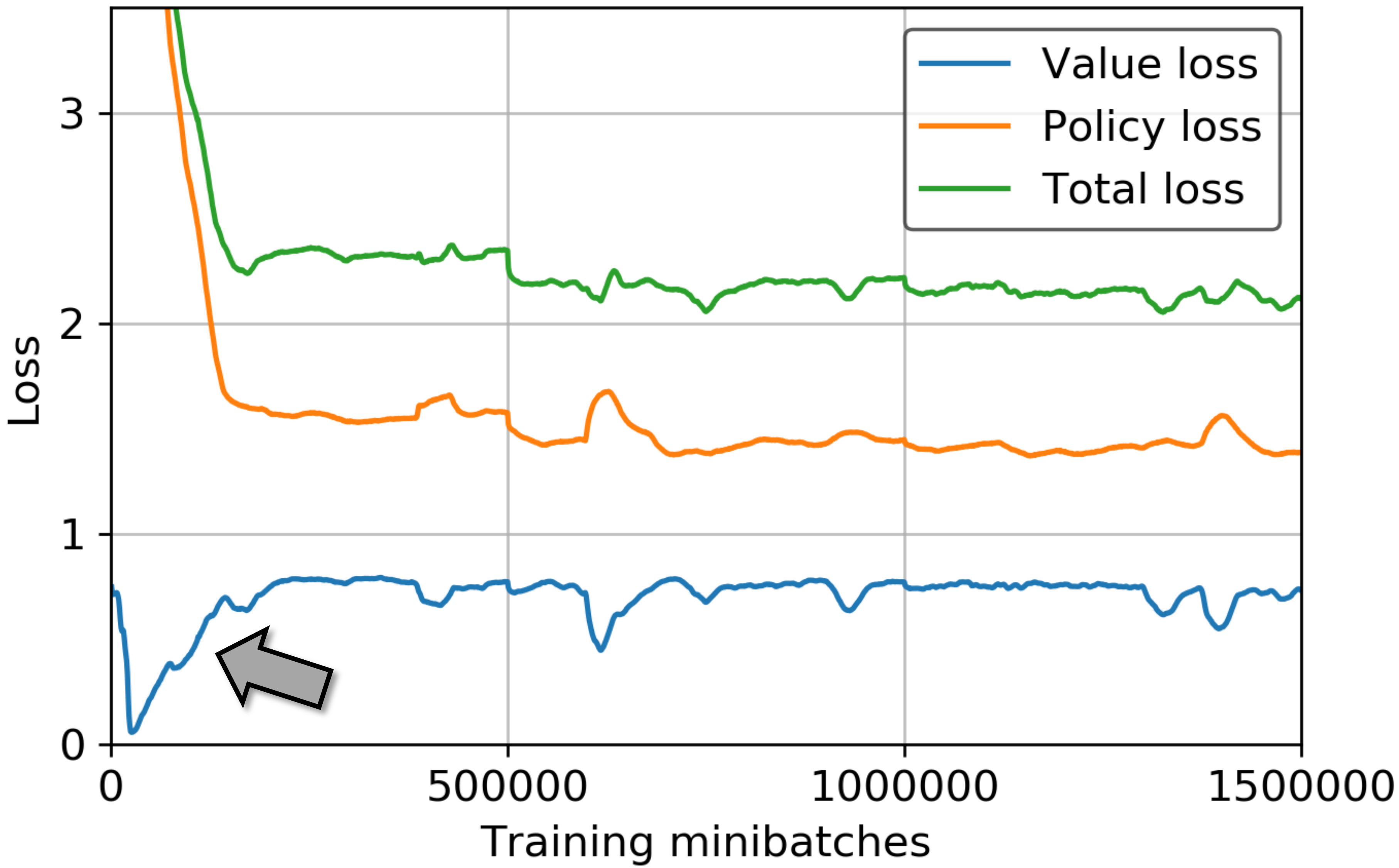
A bit unstable with learning rate 10^{-5}



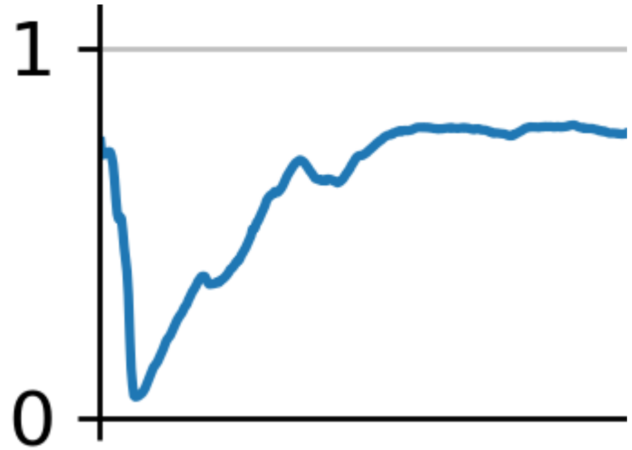
Once at capacity, new models become similar to each other?

Replay buffer becomes uniform and models start to overfit?

Overfitting issues



Overfitting issues



Dip of the value function

Overestimate white winrate

Black resigns prematurely

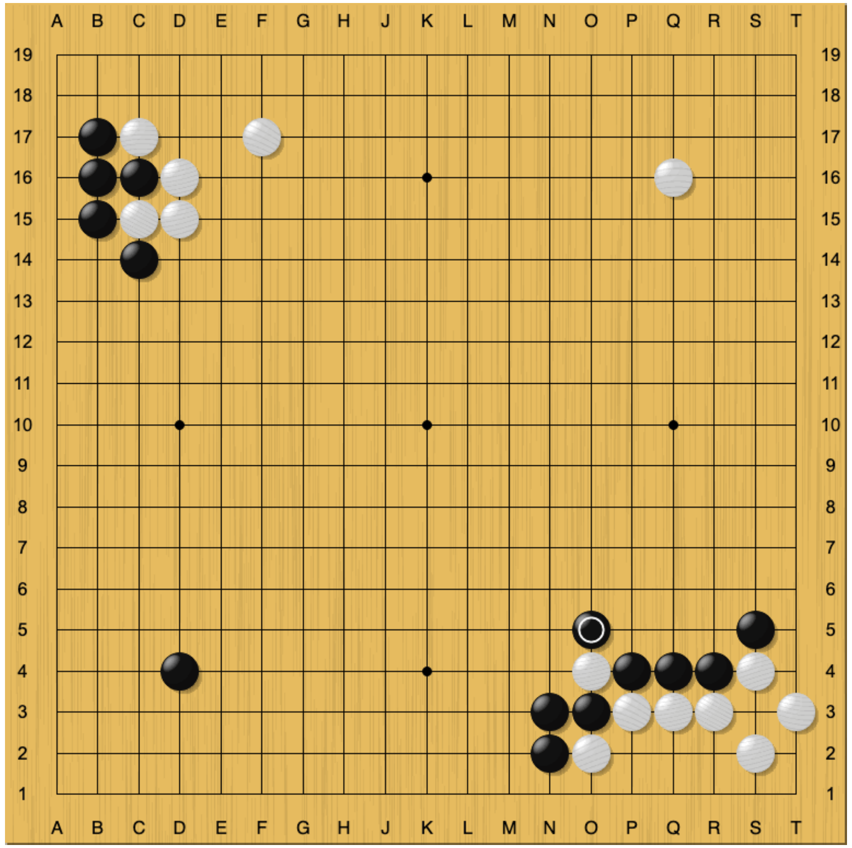
Black loses many games

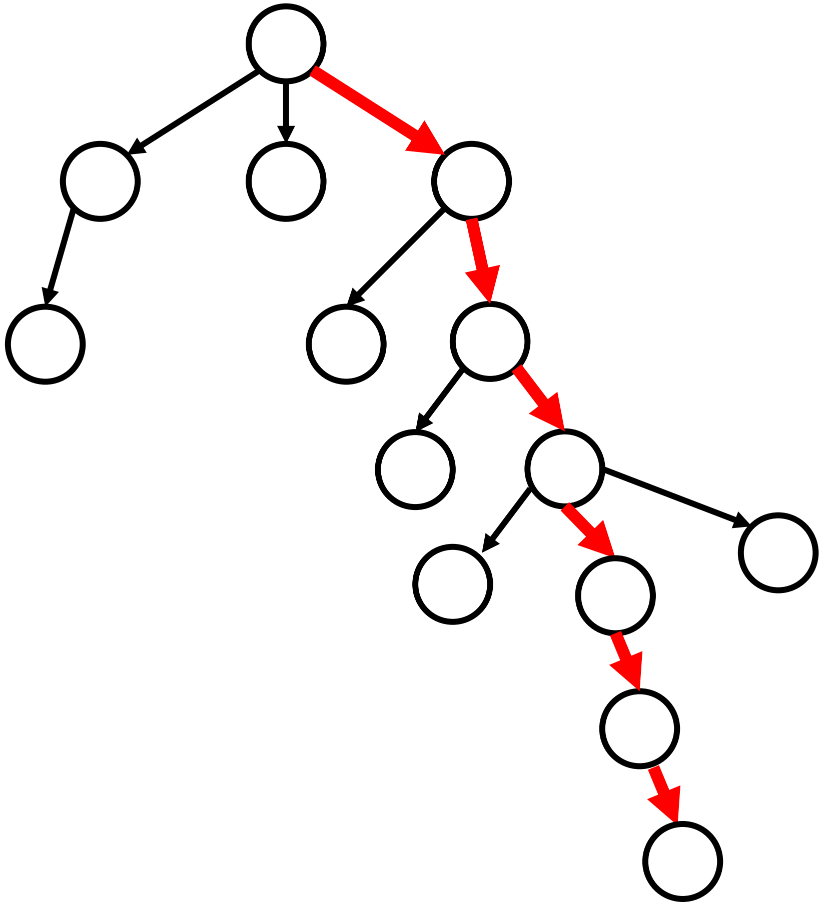
Imbalanced replay buffer

Balanced replay buffer is the key

Learning ladder moves

Learning ladder moves



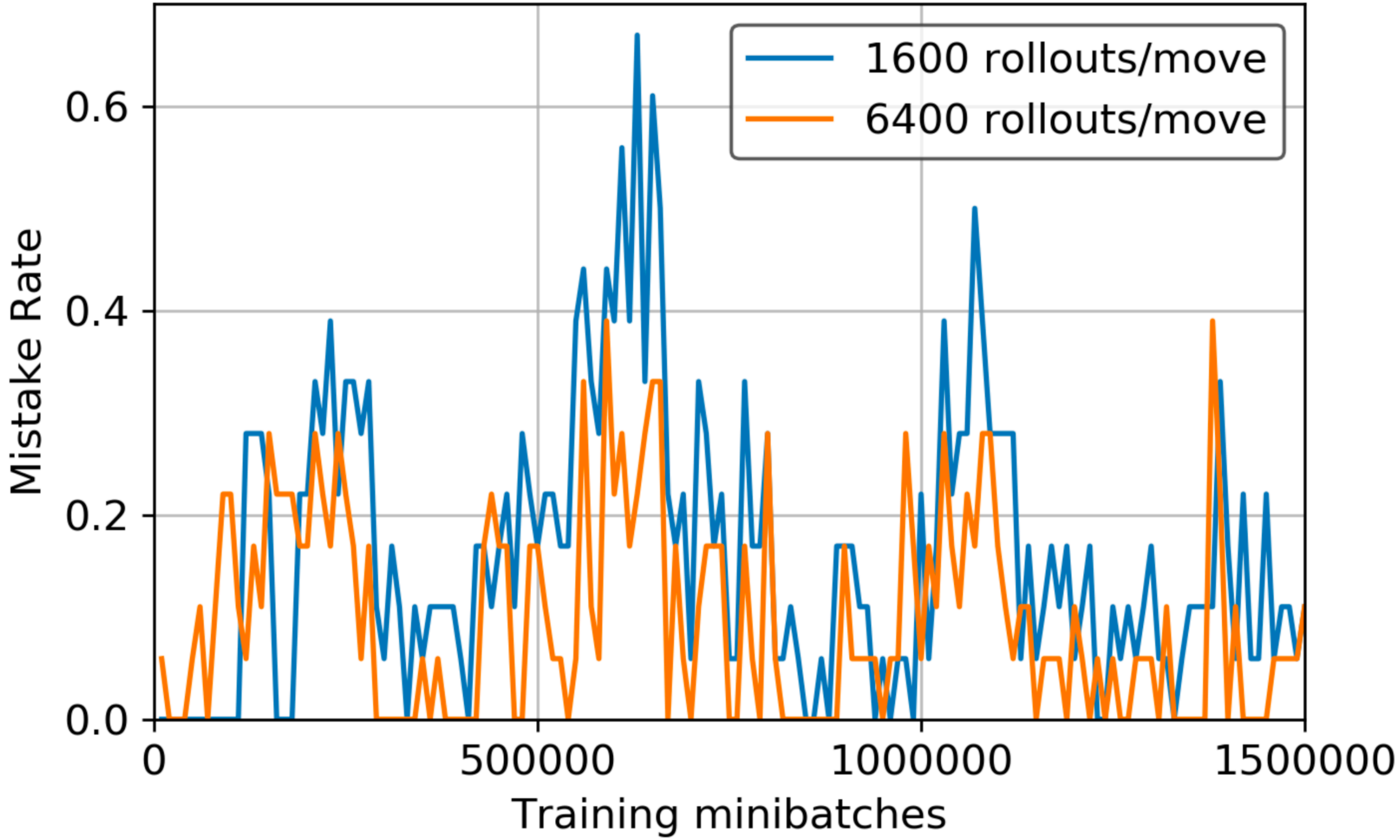


When there is only one long path that is correct, value propagation is inefficient.

Are ladders played correctly?

Ladder unit test

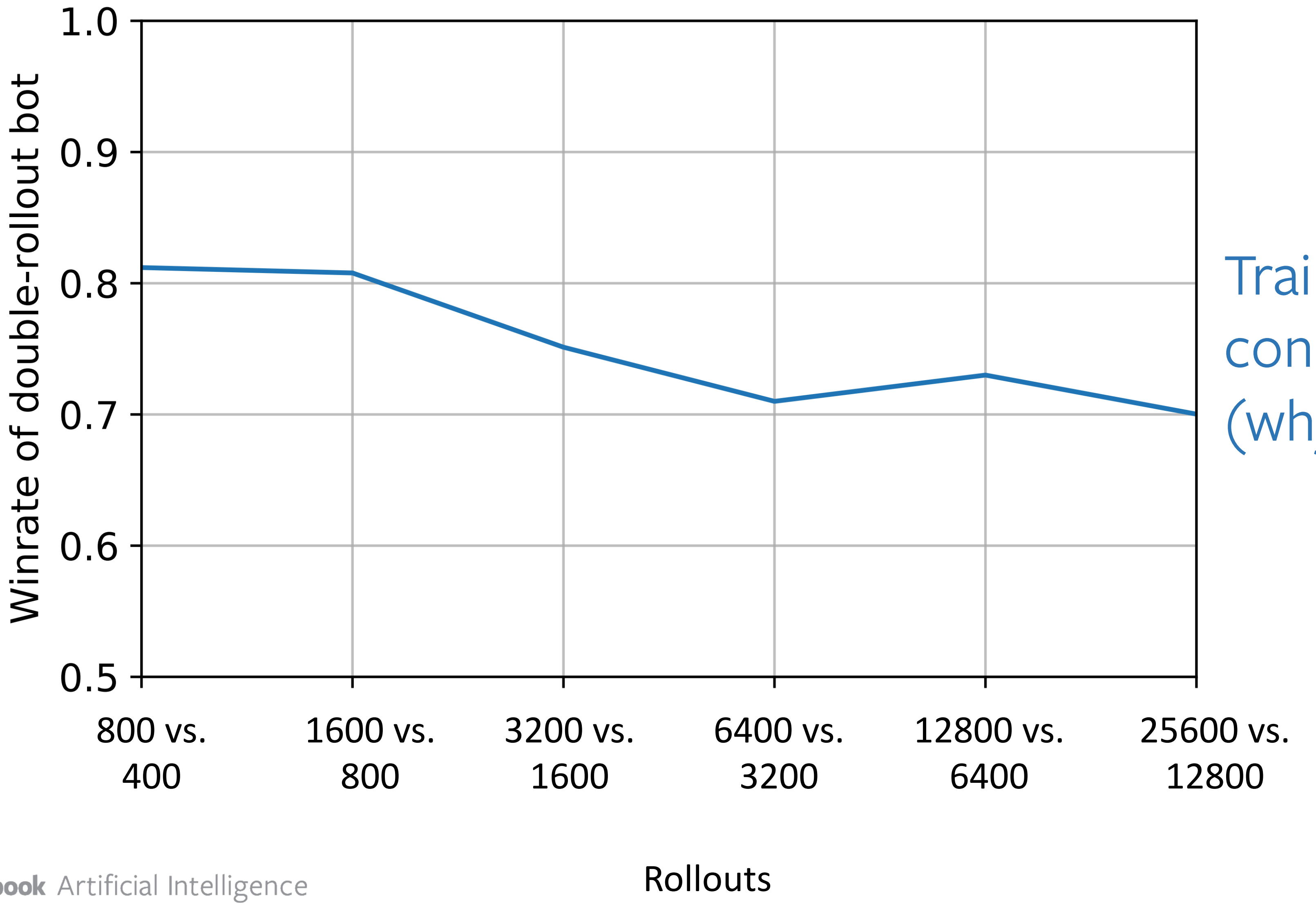
- Eval 100 ladder moves



Why is the model still strong?

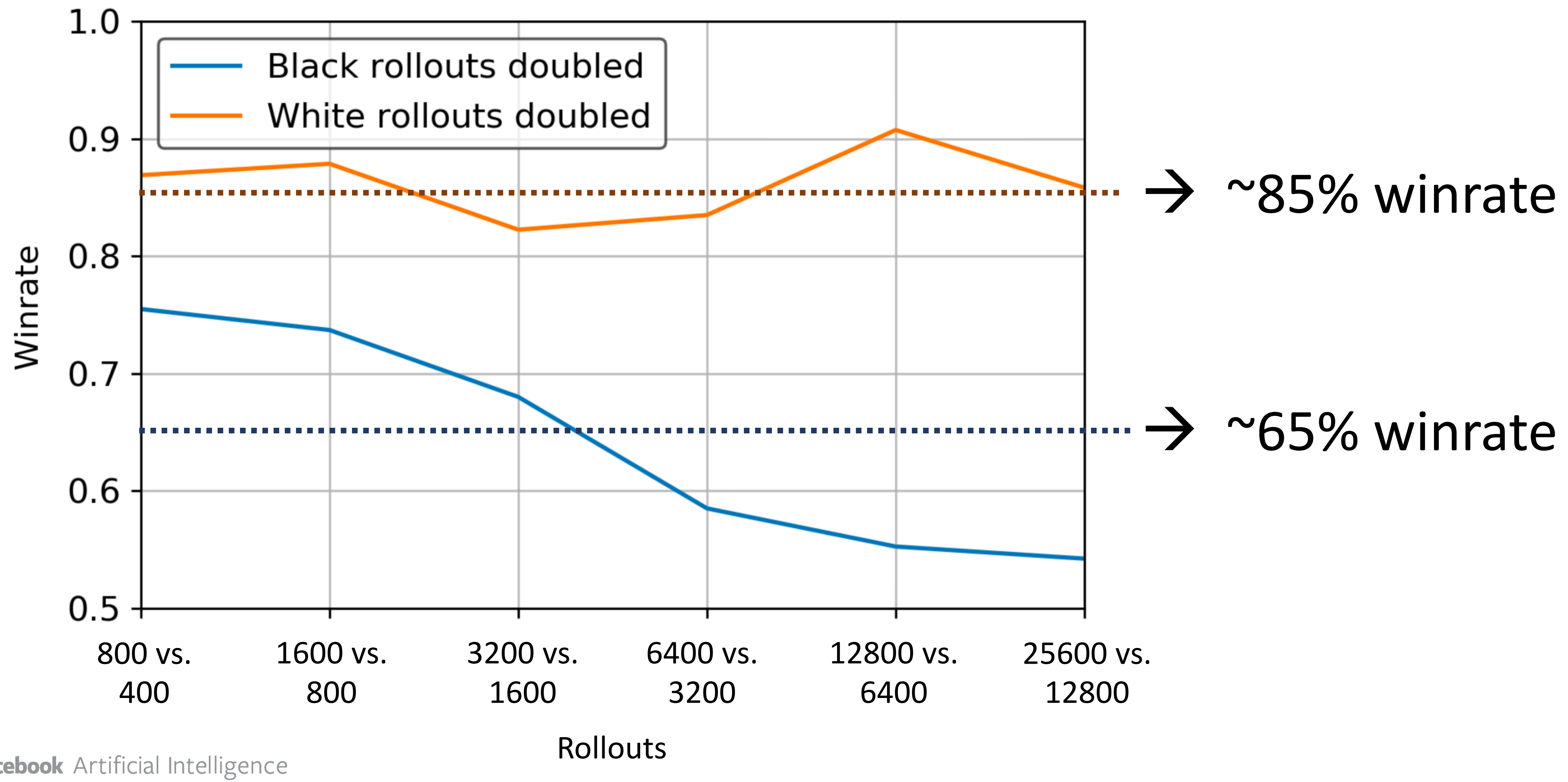
It plays alternative moves to avoid these situations.

How important is MCTS?



Training is almost always constrained by model capacity (why 40 blocks > 20 blocks)

Black versus White



ELF OpenGo

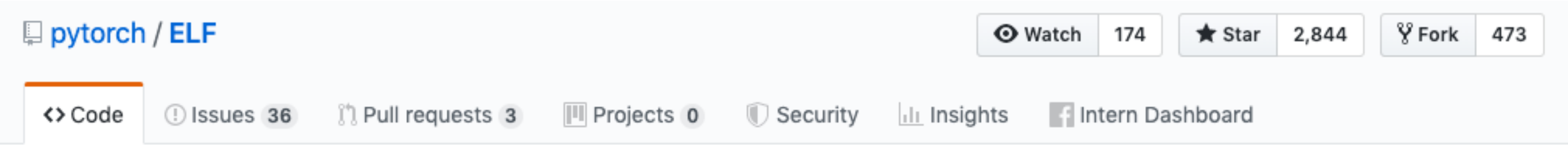
Reproduce AlphaZero

Gain a deeper understanding

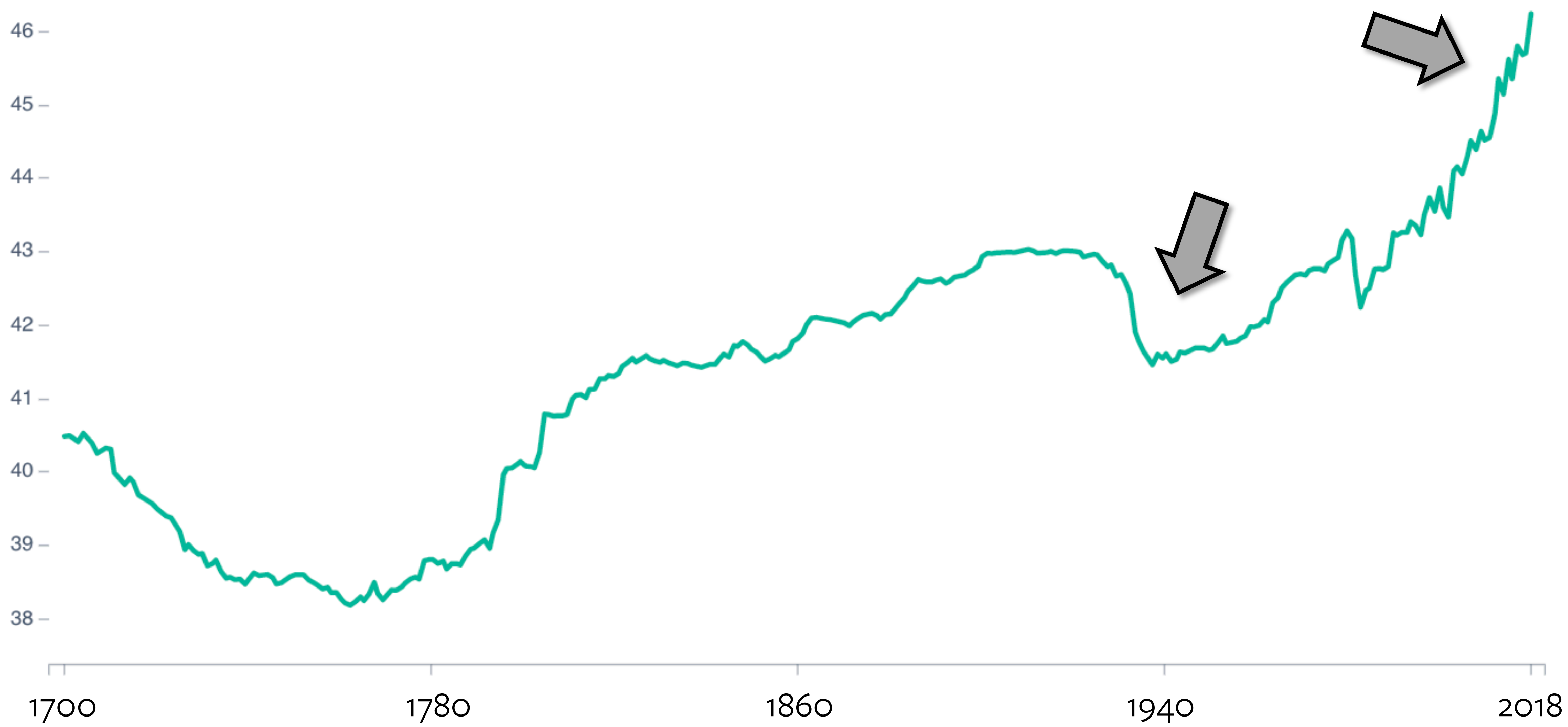
Enable the community

Open-source

- Code available on Github
- Final models
  - Enables benchmarking
- 20 million self-play games
- Public binary for playing Go



% Matching Moves 1700 - 2018



Go community

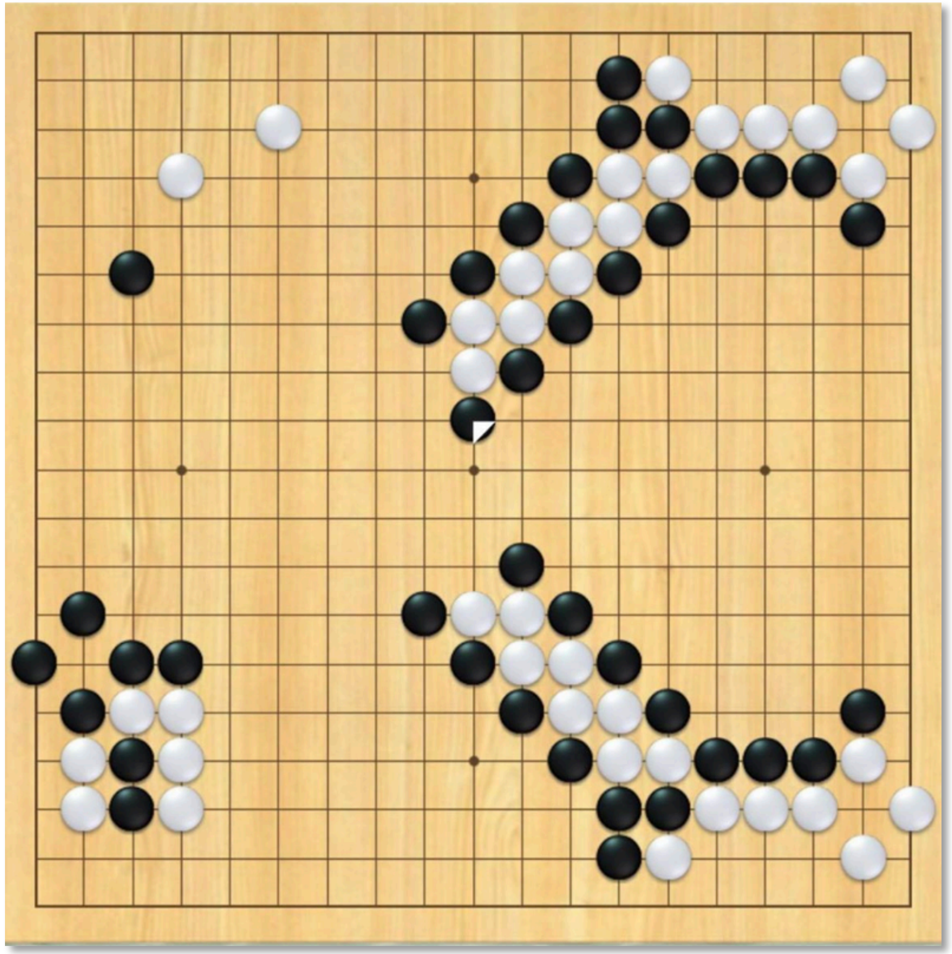
- Pre-trained weights used in Go AI competition
  - Directly used by Raynz, Golois
  - Helped to train LeeloZero, Baduki, Yike

- Judge for opening tournament at the Opening Master Championship by the Korean Go Association

- Used for pair tournament at the US Go congress by the American Go association.

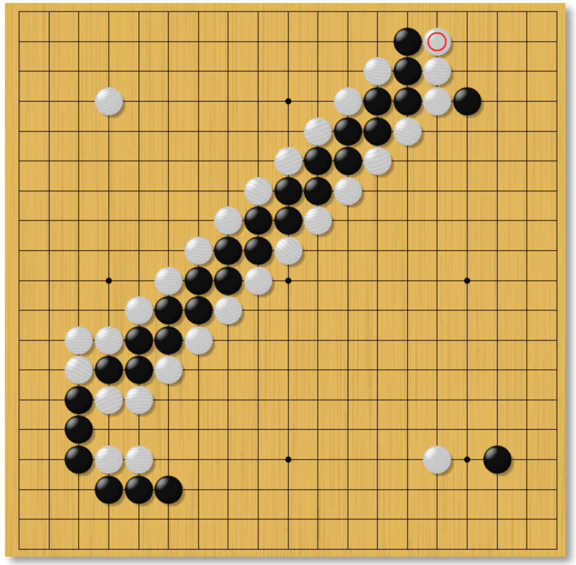
Can a human beat ELF OpenGo?

Yes!



OpenGo: White

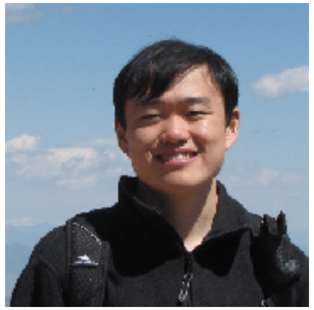
Significant questions remain...



- Stability
  - Can we reduce the variance of the algorithm?
- Sample-efficiency
  - Can we train with fewer self-play games while covering rare events?
- Adversarial robustness
  - Can we reduce the exploitable weaknesses of the bot?
  - Can we learn moves requiring significant look ahead?

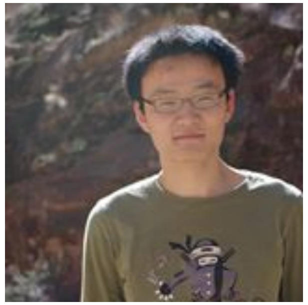
Thanks!

Poster: Pacific 31



Yuandong Tian

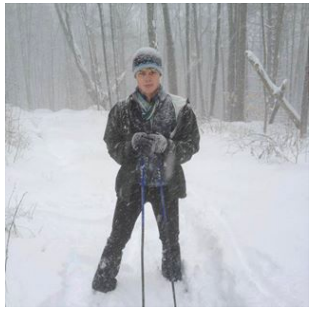
Jerry Ma*



Qucheng Gong*

Shubho Sengupta*

Zhuoyuan Chen



James Pinkerton

Larry Zitnick