

## AlphaGo

## IBM Deep Blue



Figure 1: First defeat of a world chess champion by a machine in 1997

Deep Blue was handcrafted by programmers & chess grandmasters

AlphaGo *learnt* from human moves & self play

AlphaGo evaluated fewer positions

- **width** - policy network select states more intelligently
- **depth** - value function evaluate states more precisely

## Why Go?

Long held as the most challenging classic game for artificial intelligence

- massive search space
- more legal positions than atoms in universe
- difficult to evaluate positions & moves
- sparse & delayed reward

Difficult to evaluate positions

- chess you can evaluate positions by summing the value of all the pieces
- go - it's just stones on the board, equal numbers each side

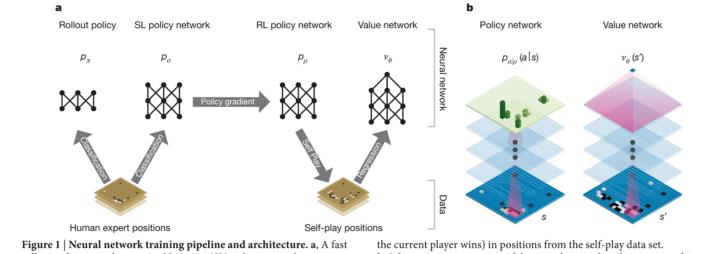
## Components of the AlphaGo agent

Three policy networks  $\pi(s)$

- fast rollout policy network – linear function
- supervised learning policy – 13 layer convolutional NN
- reinforcement learning policy – 13 layer convolutional NN

One value function  $V(s)$  - convolutional neural network

Combined together using Monte Carlo tree search



**Figure 1 | Neural network training pipeline and architecture.** **a**, A fast rollout policy  $p_a$  and supervised learning (SL) policy network  $p_d$  are trained to predict human expert moves in a data set of positions. A reinforcement learning (RL) policy network  $p_r$  is initialized to the SL policy network, and is then improved by policy gradient learning to maximize the the outcome (that is, winning more games) against previous versions of the policy network. Finally, a value network  $v_d$  is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set. **b**, Schematic representation of the neural network architecture used in AlphaGo. The policy network takes a representation of the board position  $s$  as its input, passes it through many convolutional layers with parameters  $\sigma$  (SL policy network) or  $\rho$  (RL policy network), and outputs a probability distribution  $p_{clip}(a|s)$  or  $p_r(a|s)$  over legal moves  $a$ , represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters  $\theta$ , but outputs a scalar value  $v_d(s')$  that predicts the expected outcome in position  $s'$ .

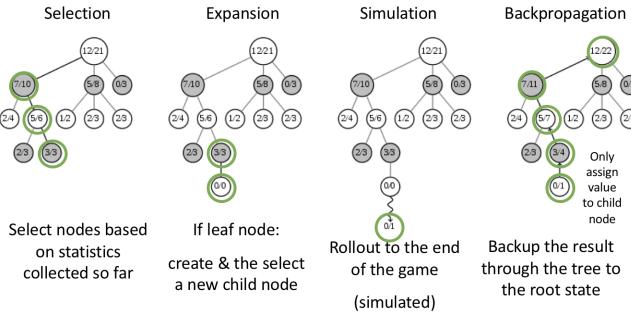
Figure 2: Learning of AlphaGo

## Monte Carlo Tree Search

Value & policy networks combined using MCTS

Basic idea = analyse most promising next moves

Planning algorithm - simulated (not actual experience) - roll out to end of game (a simulated Monte Carlo return)



[https://en.wikipedia.org/wiki/Monte\\_Carlo\\_tree\\_search](https://en.wikipedia.org/wiki/Monte_Carlo_tree_search)

Figure 3: fig

## AlphaGo, in context – Andrej Karpathy

### Convenient properties of Go

- fully deterministic
- fully observed
- discrete action space
- access to perfect simulator
- relatively short episodes
- evaluation is clear
- huge datasets of human play

DeepMind take advantage of properties of Go that will not be available in real world applications of reinforcement learning.

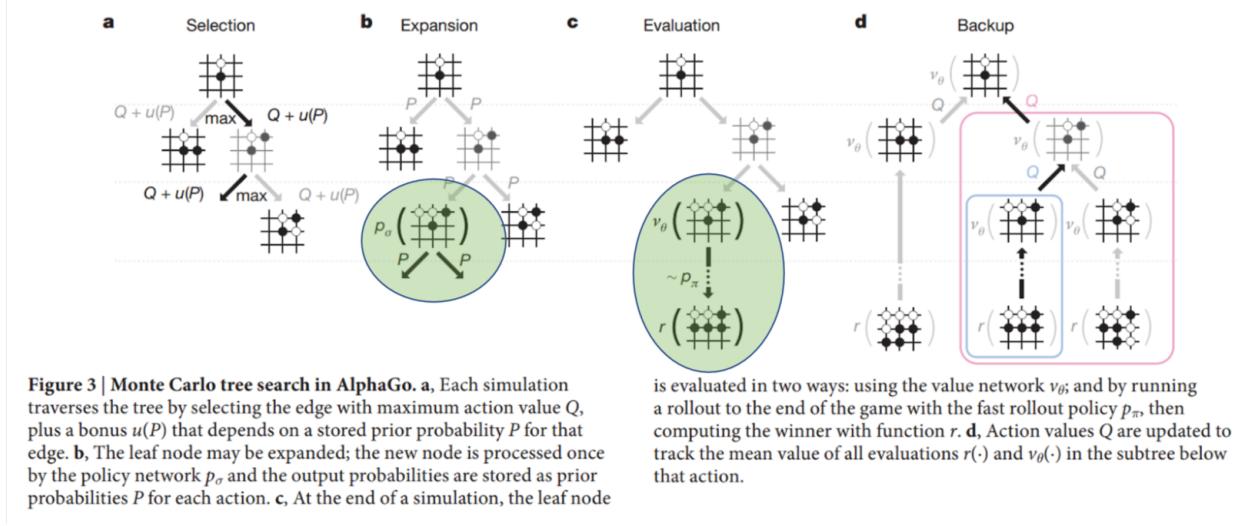


Figure 4: MCTS in AlphaGo

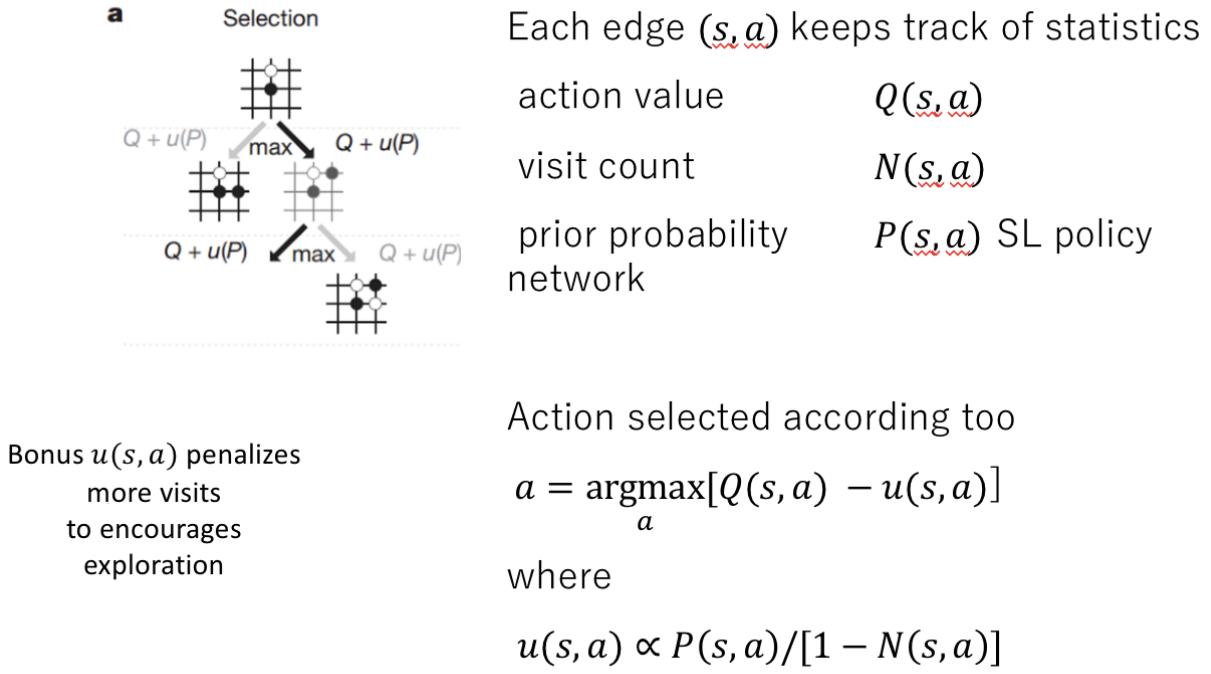
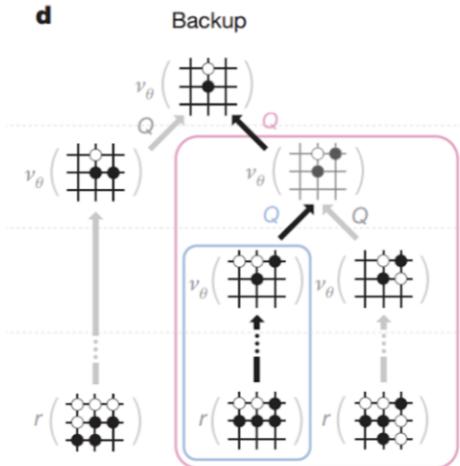


Figure 5: fig



After we finish our rollout – we calculate a state value for our leaf node  $s_L$

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

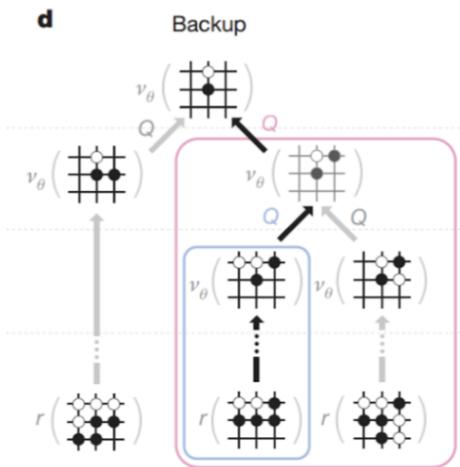
$\lambda$  mixing parameter

$v_\theta$  value network estimate

$z$  simulated result of rollout

We are combining the value network with the MCTS rollout

Figure 6: fig



Then use our combined estimate  $V(s_L)$  to update

action value  $Q(s, a)$

visit count  $N(s, a)$

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

We are averaging across all visits in the simulation

After all simulations finished - select the most visited action from the root state

Figure 7: fig

# ARTICLE

doi:10.1038/nature24270

## Mastering the game of Go without human knowledge

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

<sup>1</sup>DeepMind, 5 New Street Square, London EC4A 3TW, UK.

\*These authors contributed equally to this work.

354 | NATURE | VOL 550 | 19 OCTOBER 2017

Figure 8: fig

Key ideas

- simpler
- search
- adversarial
- machine knowledge only

Training time & performance

- AG Lee trained over several months
- AG Zero beat AG Lee 100-0 after 72 hours of training

Computational efficiency

- AG Lee = distributed w/ 48 TPU
- AG Zero = single machine w/ 4 TPU

### AlphaGo Zero innovations

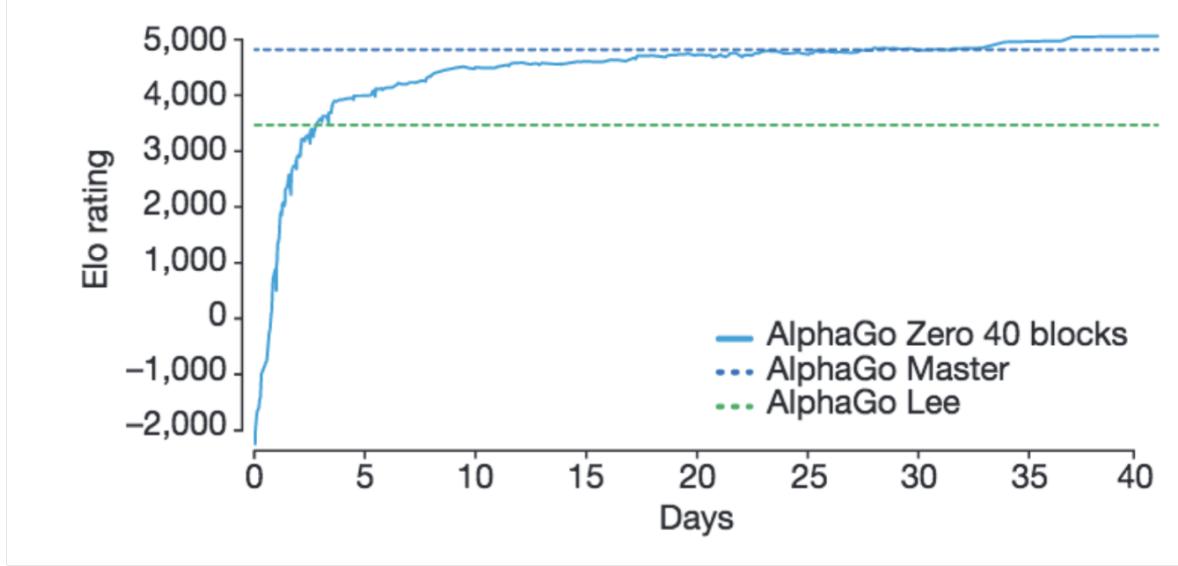
Learns using only self play

- no learning from human expert games
- no feature engineering
- learn purely from board positions

Single neural network - combine the policy & value networks

MCTS only during acting (not during learning)

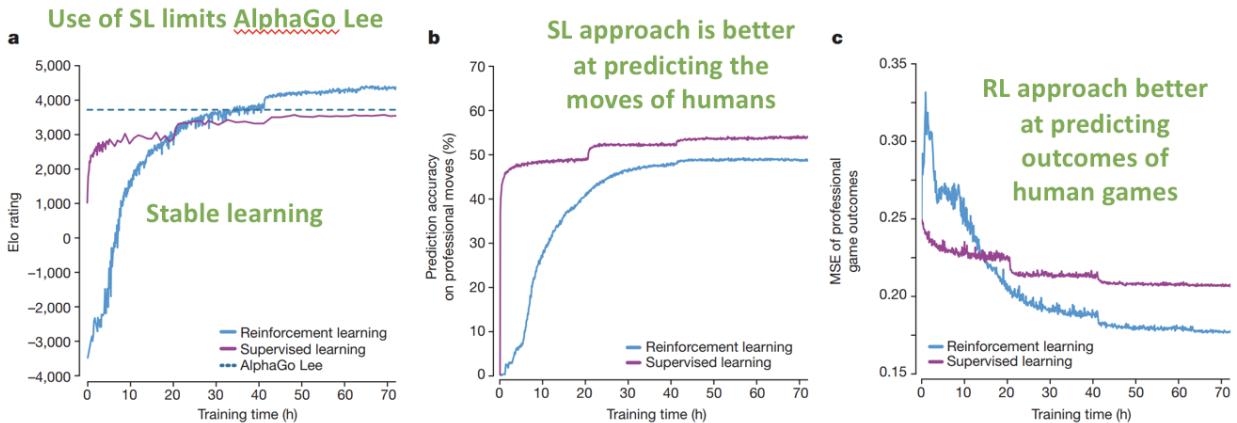
Use of residual networks



**Figure 6 | Performance of AlphaGo Zero.** **a**, Learning curve for AlphaGo Zero using a larger 40-block residual network over 40 days. The plot shows the performance of each player  $\alpha_{\theta_i}$  from each iteration  $i$  of our reinforcement learning algorithm. Elo ratings were computed from evaluation games between different players, using 0.4 s per search (see Methods)

Figure 9: fig

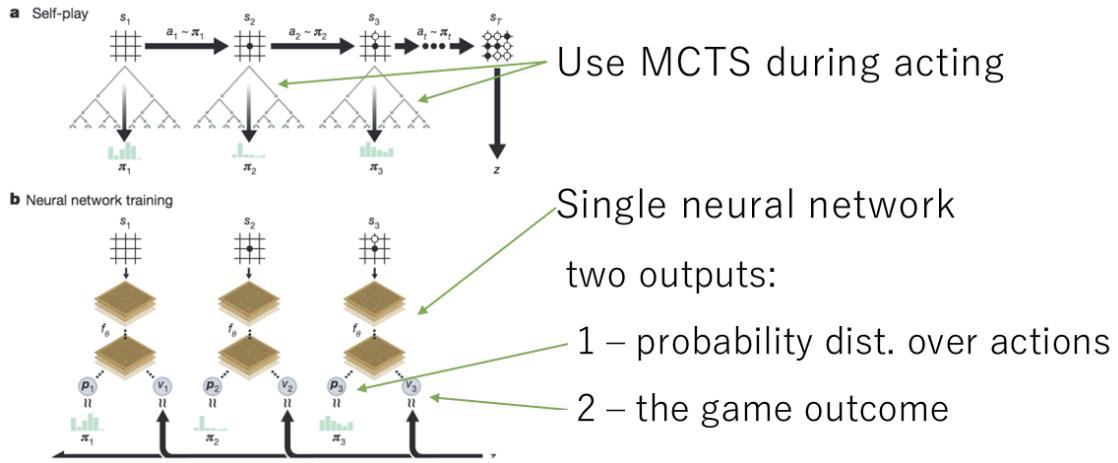
RL surpasses SL after around 1 day



**Figure 3 | Empirical evaluation of AlphaGo Zero.** **a**, Performance of self-play reinforcement learning. The plot shows the performance of each MCTS player  $\alpha_{\theta_i}$  from each iteration  $i$  of reinforcement learning in AlphaGo Zero. Elo ratings were computed from evaluation games between different players, using 0.4 s of thinking time per move (see Methods). For comparison, a similar player trained by supervised learning from human data, using the KGS dataset, is also shown. **b**, Prediction accuracy on human professional moves. The plot shows the accuracy of the neural network  $f_{\theta_i}$ , at each iteration of self-play  $i$ , in predicting human professional moves from the GoKifu dataset. The accuracy measures the

percentage of positions in which the neural network assigns the highest probability to the human move. The accuracy of a neural network trained by supervised learning is also shown. **c**, Mean-squared error (MSE) of human professional game outcomes. The plot shows the MSE of the neural network  $f_{\theta_i}$  at each iteration of self-play  $i$ , in predicting the outcome of human professional games from the GoKifu dataset. The MSE is between the actual outcome  $z \in \{-1, +1\}$  and the neural network value  $v$ , scaled by a factor of  $\frac{1}{4}$  to the range of 0–1. The MSE of a neural network trained by supervised learning is also shown.

Figure 10: fig



**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  $\alpha_\theta$  is executed (see Fig. 2) using the latest neural network  $f_\theta$ . 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  $\theta$ , 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  $\theta$  are updated to maximize the similarity of the policy vector  $p_t$  to the search probabilities  $\pi_t$ , 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**.

Model is trained to predict the probabilities as generated by MCTS during acting

Figure 11: Acting and learning

## Search in AlphaGo Zero

### Policy evaluation

Policy is evaluated through self play

This creates high quality training signals - the game result

### Policy improvement

MCTS is used during acting to create the improved policy

The improved policy generated during acting becomes the target policy during training

[Keynote David Silver NIPS 2017 Deep Reinforcement Learning Symposium AlphaZero](#)

## DeepMind AlphaGo AMA

## AMA: We are David Silver and Julian Schrittwieser from DeepMind's AlphaGo team. Ask us anything. (self.MachineLearning)

submitted 3 days ago \* (last edited 1 day ago) by [David\\_Silver](#)  - announcement

this post was submitted on 17 Oct 2017

**245 points** (97% upvoted)

shortlink: <https://redd.it/76xjb5>

Figure 12: fig

↳ [·] [David\\_Silver](#)  (S) 9 points 1 day ago  
Creating a system that can learn entirely from self-play has been an open problem in reinforcement learning. Our initial attempts, as for many similar algorithms reported in the literature, were quite unstable. We tried many experiments - but ultimately the AlphaGo Zero algorithm was the most effective, and appears to have cracked this particular issue.

[permalink](#) [source](#) [embed](#) [save](#) [save-RES](#) [parent](#) [report](#) [give gold](#) [reply](#) [hide child comments](#)

↳ [·] [David\\_Silver](#)  (S) 3 points 1 day ago  
In some sense, training from self-play is already somewhat adversarial: each iteration is attempting to find the "anti-strategy" against the previous version.

[permalink](#) [source](#) [embed](#) [save](#) [save-RES](#) [parent](#) [report](#) [give gold](#) [reply](#)

↳ [·] [David\\_Silver](#)  (S) 13 points 1 day ago  
Actually we never guided AlphaGo to address specific weaknesses - rather we always focused on principled machine learning algorithms that learned for themselves to correct their own weaknesses.  
Of course it is infeasible to achieve optimal play - so there will always be weaknesses. In practice, it was important to use the right kind of exploration to ensure training did not get stuck in local optima - but we never used human nudges.

[permalink](#) [source](#) [embed](#) [save](#) [save-RES](#) [parent](#) [report](#) [give gold](#) [reply](#)

Figure 13: fig