# Mastering the game of Go without human knowledge

Michael Wollowski

Summary of

Silver at al. *Mastering the game of Go without human knowledge*
https://www.nature.com/articles/nature24270

## Introduction

- AlphaGoZero is its own teacher: a neural network is trained to predict its own move selections and the winner of games.
- The training happens solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules.
- AlphaGoZero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

## Introduction

- Until recently, supervised learning systems were trained to replicate the decisions of human experts.
- 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.

## Introduction

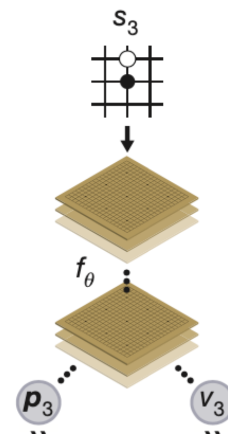
- By contrast, reinforcement learning systems are trained from their own experience.
- Recently, there has been rapid progress towards this goal, using deep neural networks trained by reinforcement learning.
- The game of Go was widely viewed as a grand challenge for artificial intelligence.
- It requires a precise and sophisticated look-ahead in vast search spaces.

# Basic Characteristics

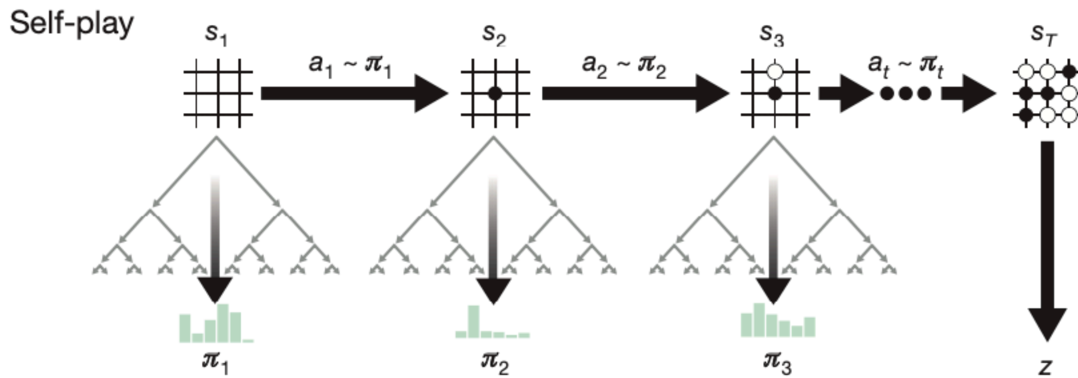
- AlphaGoZero differs from AlphaGo Fan and AlphaGo Lee in several important aspects.
  1. It is trained solely by self-play reinforcement learning, starting from random play, without any supervision or use of human data.
  2. It uses only the black and white stones from the board as input features.
  3. It uses a simpler tree search, without performing any Monte Carlo rollouts.

# Overall Setup of AlphaGoZero

- It uses a deep neural network.
- The network outputs both, move probabilities and a value.
- The vector of move probabilities ***p*** represents the probability of selecting each move.
- The value *v* is a scalar evaluation, estimating the probability of the current player winning from position *s*.
- It only uses its deep neural network to evaluate leaf nodes and to select moves.

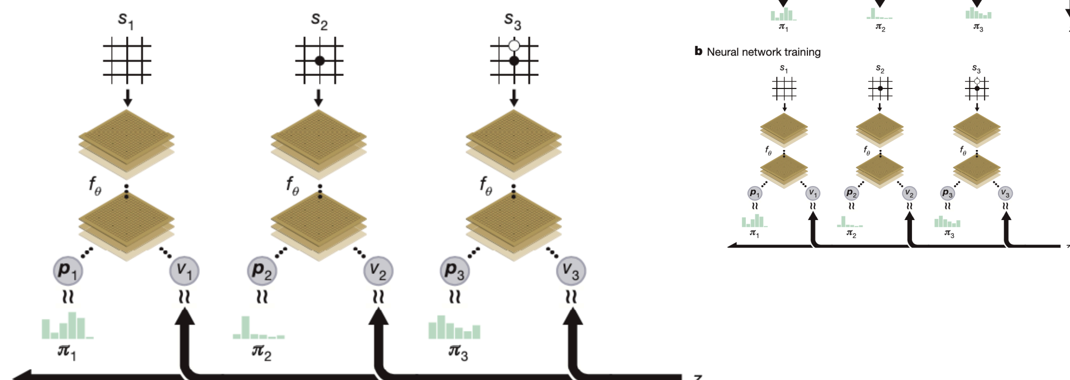


# Self-Play




- The program plays a game, $s_1$, … $s_T$ against itself.
- In each position $s_t$, a MCTS is executed, resulting in a probabilities $\boldsymbol{\pi}$ of playing each move in $s_t$.
- The final position $s_T$ is scored to determine the winner, *z*.

# NN Training

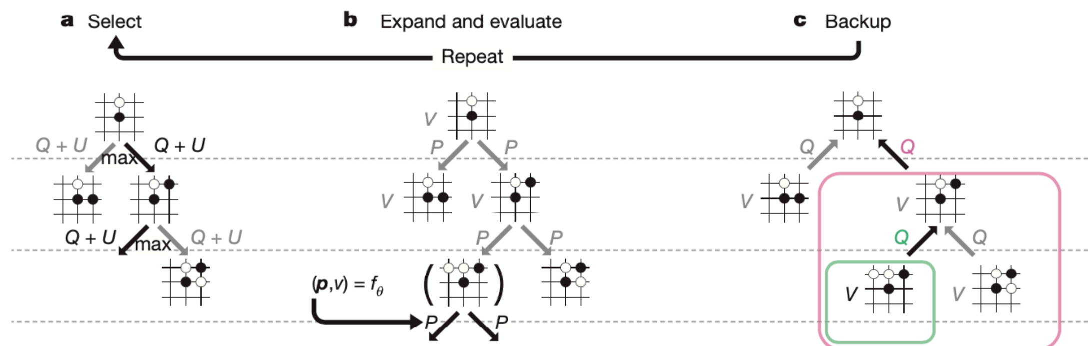



- The neural network takes the raw board position $s_t$ as its input and passes it through many convolutional layers.
- It outputs a vector $\boldsymbol{p}_t$, representing a probability distribution over moves, and
- a scalar value $v_t$, representing the probability of the current player winning in position $s_t$.

# NN Training

- The neural network parameters are updated:
  - to maximize the similarity of the probability vector $\boldsymbol{p}_t$ to the search probabilities $\boldsymbol{\pi}_t$, and
  - to minimize the error between the predicted winner $v_t$ and the game winner $z$.
- The new parameters are used in the next iteration of self-play.

# MCTS in AlphaGoZero

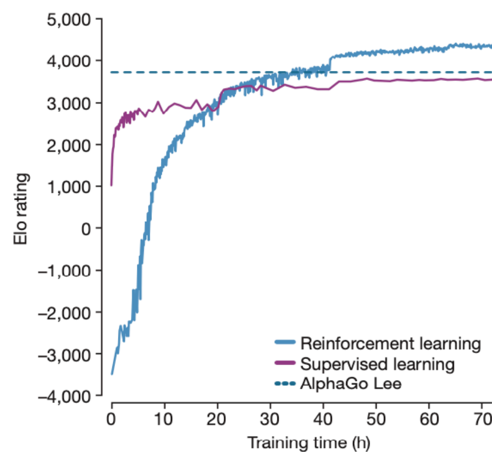



- MCTS uses the neural network to guide its simulations.
- Each edge ($s$, $a$), i.e. (*situation*, *action*) in the search tree stores a prior probability $P(s, a)$, a visit count $N(s, a)$, and an action value $Q(s, a)$.

# Evaluation

- Training started from completely random behavior.
- It continued without human intervention for approximately three days.
- 4.9 million games of self-play were generated
- Using 1,600 simulations for each MCTS, corresponds to approximately 0.4 s thinking time per move.

# Evaluation: Elo Rating



## Knowledge Learned by AlphaGoZero

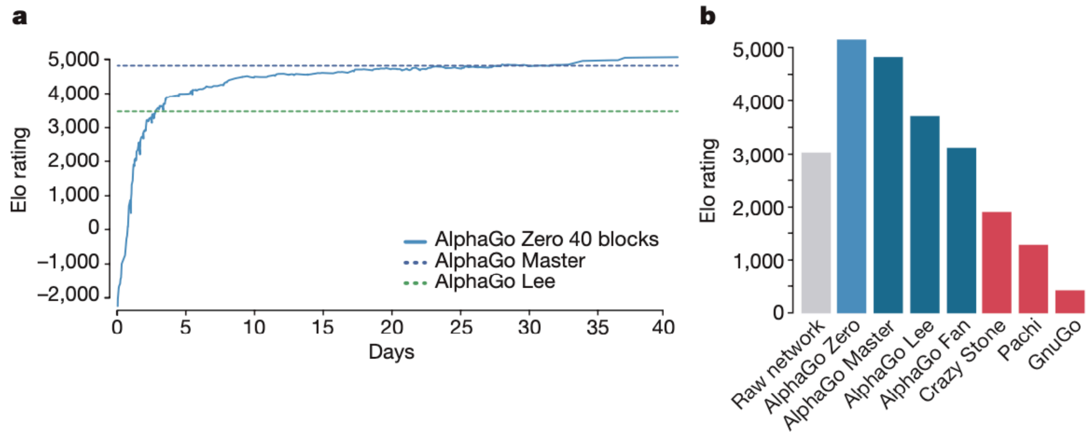
- AlphaGoZero discovered a remarkable level of Go knowledge during its self-play training process.
- This included fundamental elements of human Go knowledge
- As well as nonstandard strategies beyond the scope of traditional Go knowledge.

## Knowledge Learned by AlphaGoZero

It rapidly progressed from entirely random moves towards a sophisticated understanding of Go concepts, including:

- *fuseki* (opening),
- *tesuji* (tactics),
- life and death, *ko* (repeated board situations),
- *yose* (endgame),
- capturing races, *sente* (initiative),
- shape, influence and territory, all discovered from first principles.
- *shicho* ('ladder' capture sequences that may span the whole board)—one of the first elements of Go knowledge learned by humans—were only understood much later in training.

# Performance of AlphaGoZero




# Go Knowledge provided to AlphaGoZero

- The player is provided with the set of legal moves in each position.
- Games terminate when both players pass or after 19 × 19 × 2 = 722 moves.
- AlphaGoZero uses Tromp–Taylor scoring during MCTS simulations and self-1play training.

## NN Architecture

- The input to the neural network is a 19 × 19 × 17 image stack comprising 17 binary feature planes.
- The algorithm was started with random initial parameters for the neural network.
- Eight feature planes, $X_t$, consist of binary values indicating the presence of the current player's stones ($X^i_t$=1 if intersection *i* contains a stone of the player's color at time step *t*; 0 if the intersection is empty, contains an opponent stone, or if $t < 0$).
- Eight feature planes, $Y_t$, represent the corresponding features for the opponent's stones.
- The final feature plane, *C*, represents the color to play.
- These planes are concatenated together to give input features

  $s_t = [X_t, Y_t, X_{t-1}, Y_{t-1},..., X_{t-7}, Y_{t-7}, C]$.
- History features $X_t$, $Y_t$ are necessary, because Go is not fully observable solely from the current stones, as repetitions are forbidden; similarly, the color feature *C* is necessary, because the *komi* is not observable.

## NN Architecture

The input features $s_t$ are processed by a residual tower that consists of a single convolutional block followed by either 19 or 39 residual blocks.

The convolutional block applies the following modules:
(1) A convolution of 256 filters of kernel size 3 × 3 with stride 1
(2) Batch normalization
(3) A rectifier nonlinearity

## NN Architecture

Each residual block applies the following modules sequentially to its input:

(1) A convolution of 256 filters of kernel size 3 × 3 with stride 1
(2) Batch normalization
(3) A rectifier nonlinearity
(4) A convolution of 256 filters of kernel size 3 × 3 with stride 1
(5) Batch normalization
(6) A skip connection that adds the input to the block
(7) A rectifier nonlinearity

## NN Architecture

The output of the residual tower is passed into two separate 'heads' for computing the policy and value.

The policy head applies the following modules:

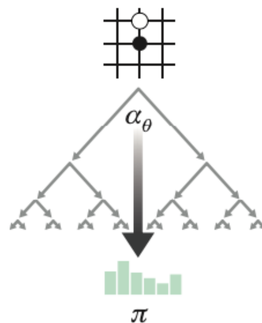
(1) A convolution of 2 filters of kernel size 1 × 1 with stride 1
(2) Batch normalization
(3) A rectifier nonlinearity
(4) A fully connected linear layer that outputs a vector of size $19^2+1=362$, corresponding to logit probabilities for all intersections and the pass move

# NN Architecture

The value head applies the following modules:
(1) A convolution of 1 filter of kernel size 1 × 1 with stride 1
(2) Batch normalization
(3) A rectifier nonlinearity
(4) A fully connected linear layer to a hidden layer of size 256
(5) A rectifier nonlinearity
(6) A fully connected linear layer to a scalar
(7) A tanh nonlinearity outputting a scalar in the range [−1, 1]

The overall network depth, in the 20 or 40 block network, is 39 or 79 parameterized layers, respectively, for the residual tower, plus an additional 2 layers for the policy head and 3 layers for the value head.

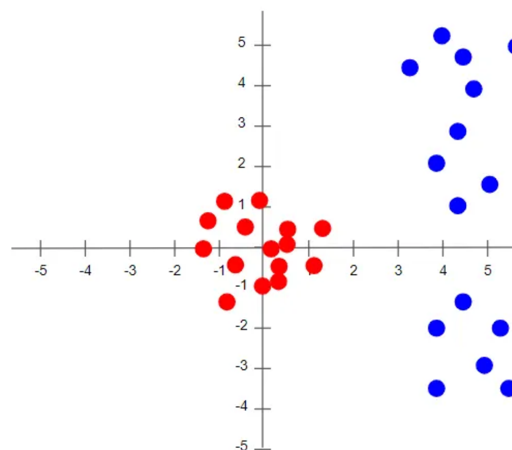
# Play



# Residual Blocks

- In traditional neural networks, each layer feeds into the next layer.
- In a network with residual blocks, each layer feeds into the next layer and directly into the layers about 2–3 hops away.

# Batch Normalization




Source: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739