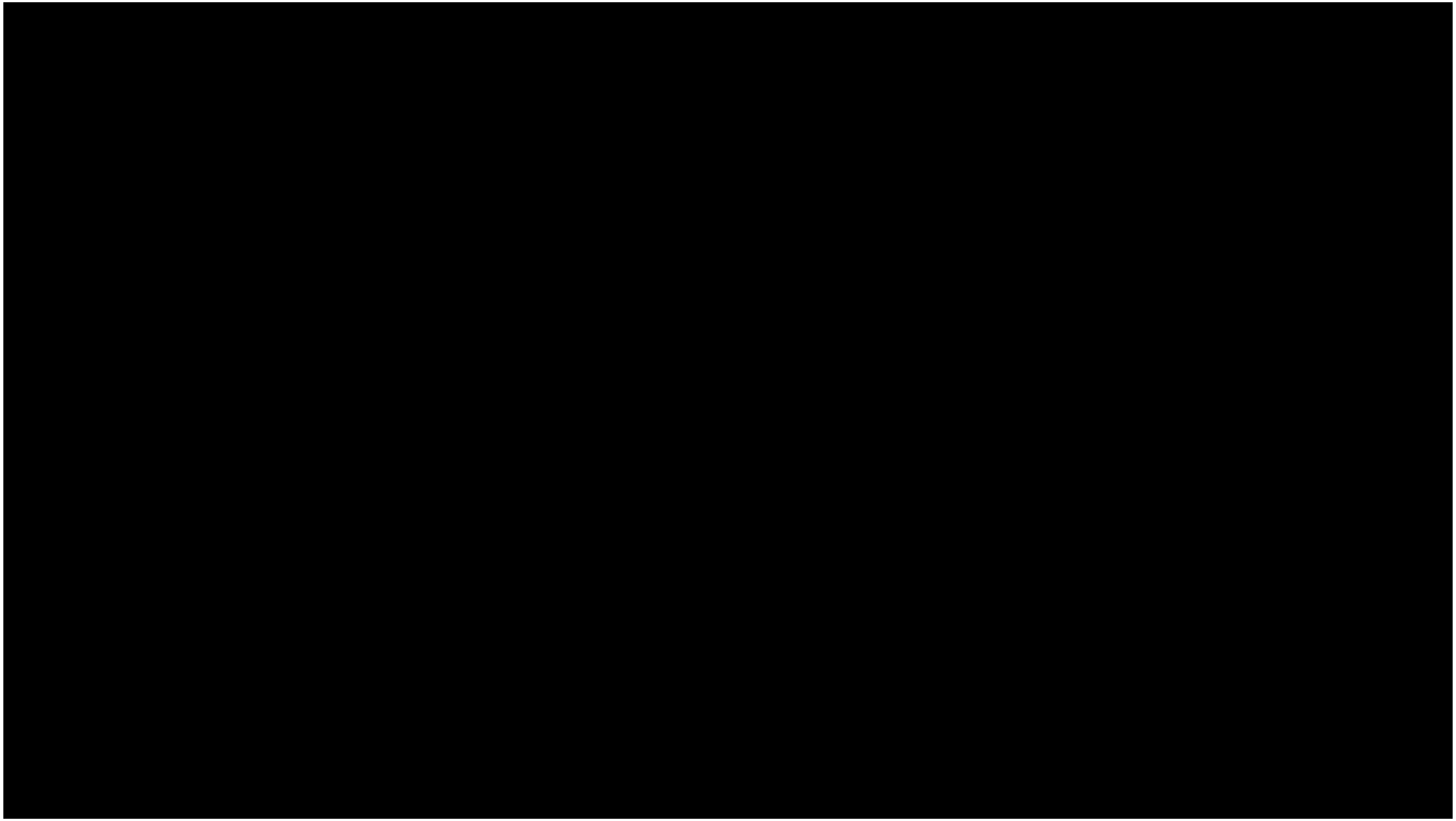
Master
Reinforcement Learning 2022
Lecture 6:
Two-Agent Self-Play

Aske Plaat

Different Approaches

- Model-free
    - Value-based [2,3]
    - Policy-based [4]
- Model-based
    - Learned [5]
    - Perfect; Two-Agent [6]
- Multi-agent [7]
- Hierarchical Reinforcement Learning (Sub-goals) [8]
- Meta Learning [9]

Motivation



Overview

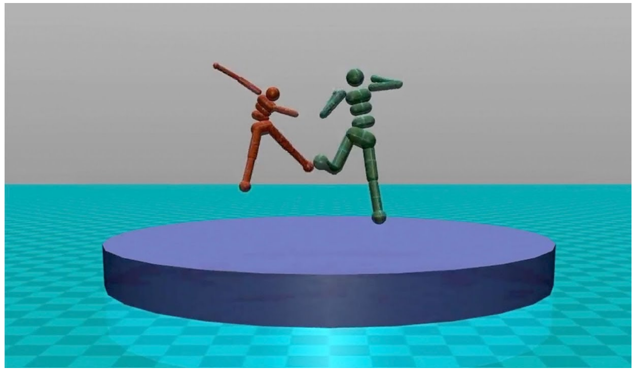
- MCTS: a well-known RL planner
- What if Internal Transition Function is Perfect & your Environment is yourself?
- AlphaZero
- Curriculum Learning

What if Internal Transition function is Perfect?

- Previous chapter showed that accuracy of model is important.
  What if we have a perfect transition function?
  What if we can also use it to learn?
- Then World Champions get beaten:
  - Backgammon
  - Go
  - Chess
  - Shogi
- Today is about Best Case, when everything fits together and works

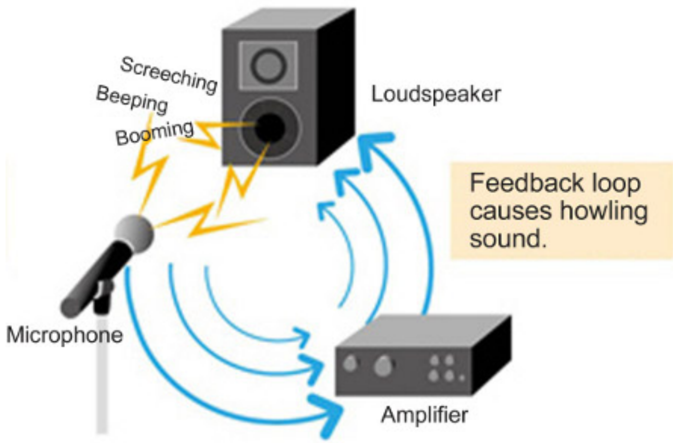
Self-Play is old

- Most two-agent board game-playing programs choose (versions of) themselves as opponent for simulation or learning.
- Minimax (1949) is Self-Play
- Samuel's checkers players (1950-1960) used self-play outcomes for modifying evaluation weigths.
- TD-Gammon (1992) used Self-Play learning
- However, Self-Play is potentially unstable due to feedback and deadly triad
- It is overcome in AlphaGo in different ways



Surprising Self-Play

- Find high quality examples to train RL on using RL. RL at three levels



- RL suffers from feedback, feedback creates instability

- What methods have been used to overcome feedback?

AlphaZero: Three Levels of Self Play

1. **Move**-level self play (minimax, MCTS)

2. **Example**-level self play (learning, Actor Critic)

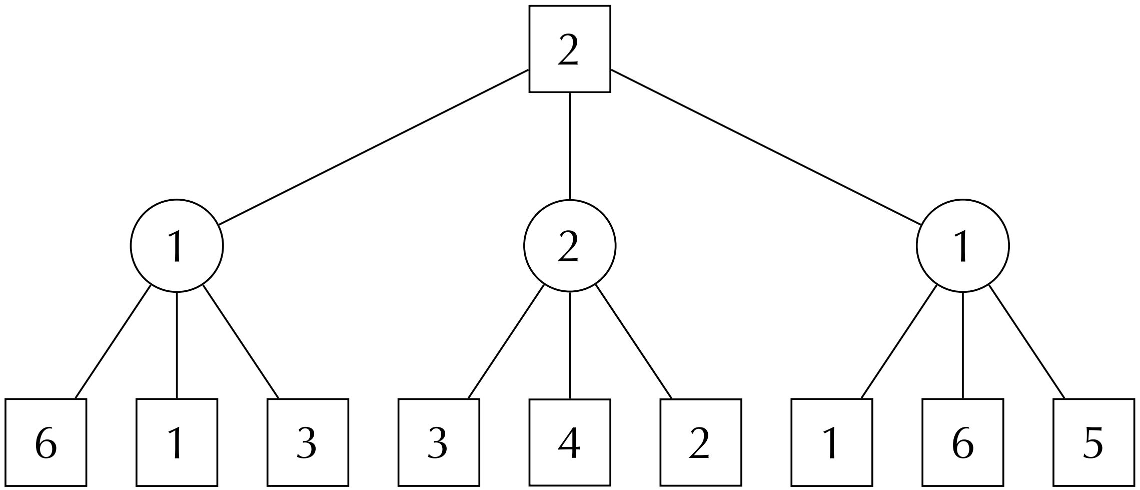
3. **Game**-level self play (curriculum, self transcending player)

1. Move-level self play

Minimax

- Assume you play best move, and opponent has your knowledge

Minimax



- Two-agent zero sum: my win is your loss
- Max/min/max/min/max/min
- Max: Square
- Min: Circle
- b=branching factor, d=search depth

Minimax

```
INF = 99999

def eval(n):
    if n['type'] == 'LEAF':
        return n['value']
    else:
        error("Calling eval not on LEAF")

def minimax(n):
    if n['type'] == 'LEAF':
        return eval(n)
    elif n['type'] == 'MAX':
        g = -INF
        for c in n['children']:
            g = max(g, minimax(c))
    elif n['type'] == 'MIN':
        g = INF
        for c in n['children']:
            g = min(g, minimax(c))
    else:
        error("Wrong node type")
    return g

print("Minimax value: ", minimax(root))
```

Listing 4.2: Minimax code

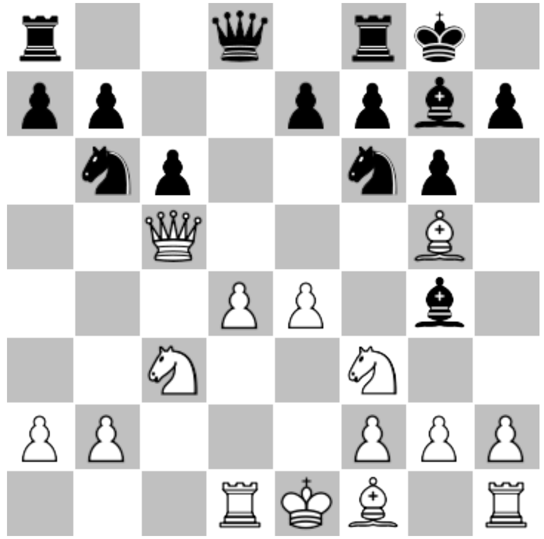
State Space Complexity

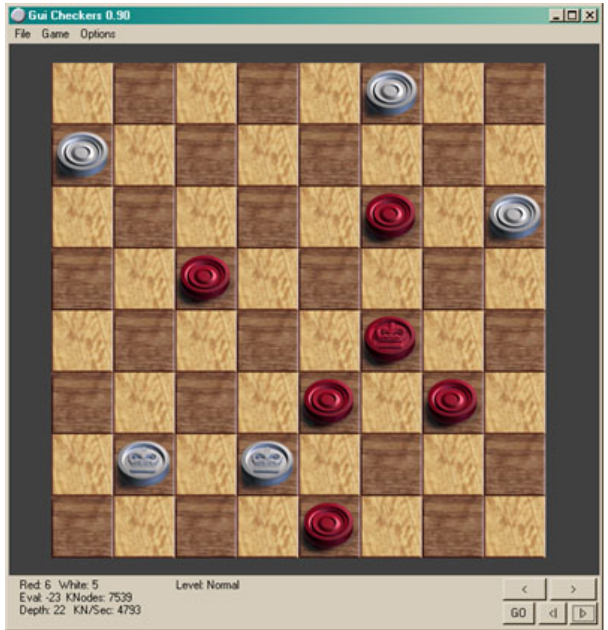
Name	board	state space	zero sum	information	turn
Chess	8×8	10^{47}	zero sum	perfect	turn
Checkers	8×8	10^{18}	zero sum	perfect	turn
Othello	8×8	10^{28}	zero sum	perfect	turn
Backgammon	24	10^{20}	zero sum	chance	turn
Go	19×19	10^{170}	zero sum	perfect	turn
Shogi	9×9	10^{71}	zero sum	perfect	turn
Poker	card	10^{161}	non-zero	imperfect	turn
StarCraft	real time strategy	10^{1685}	non-zero	imperfect	simultaneous

- 10^47… 1 ns/position -> 10^38 s -> 10^30 earth-year 10^21 times the age of the known universe/position

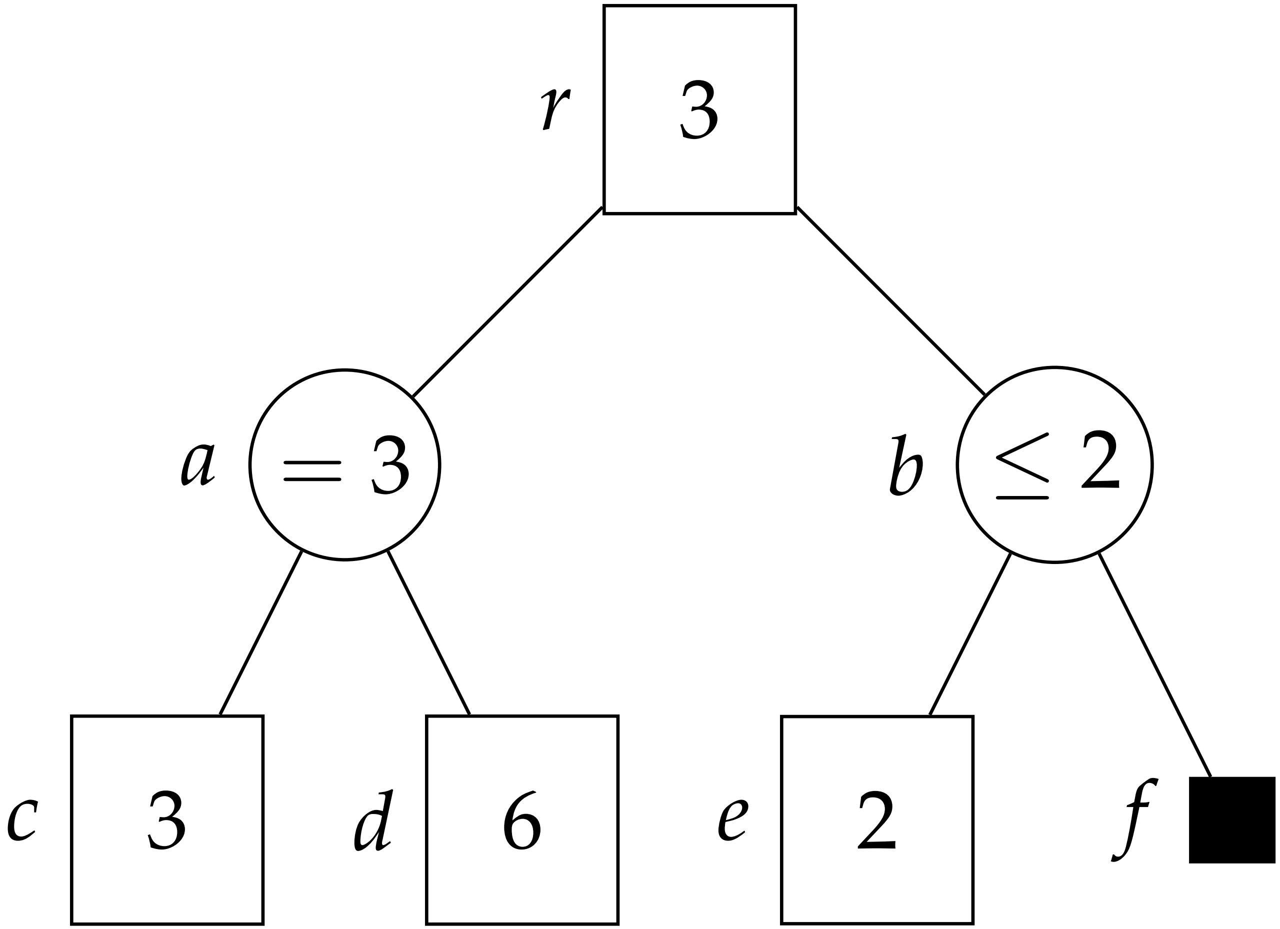
Heuristics

- Material (pawns, bishops, knights, …)
- Mobility (# actions)
- Center control
- King Safety
- …





Alpha-Beta



- A cutoff is an action (e) that is so strong for my opponent that I will not play b (because a is better) and hence we can stop searching b

After Chess?

Go!

Go

Name	board	state space	zero sum	information	turn
Chess	8×8	10^{47}	zero sum	perfect	turn
Checkers	8×8	10^{18}	zero sum	perfect	turn
Othello	8×8	10^{28}	zero sum	perfect	turn
Backgammon	24	10^{20}	zero sum	chance	turn
Go	19×19	10^{170}	zero sum	perfect	turn
Shogi	9×9	10^{71}	zero sum	perfect	turn
Poker	card	10^{161}	non-zero	imperfect	turn
StarCraft	real time strategy	10^{1685}	non-zero	imperfect	simultaneous

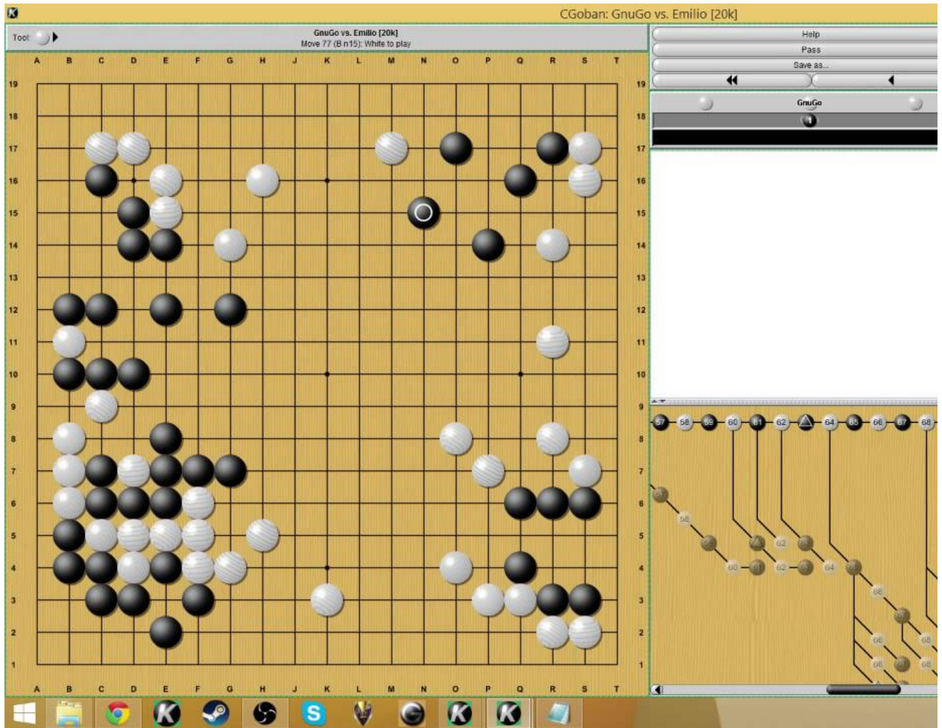
- Worldwide most popular combinatorial game of strategy
- Much more complex than Chess

Heuristic Planning

- Successful in games with tactical play where efficient heuristics can be found
  - Pieces move, and material is a good indicator of the score
- In Go, board is large, pieces do not move, material is typically balanced, and “influence” turned out to be difficult to program efficiently

Traditional Go program

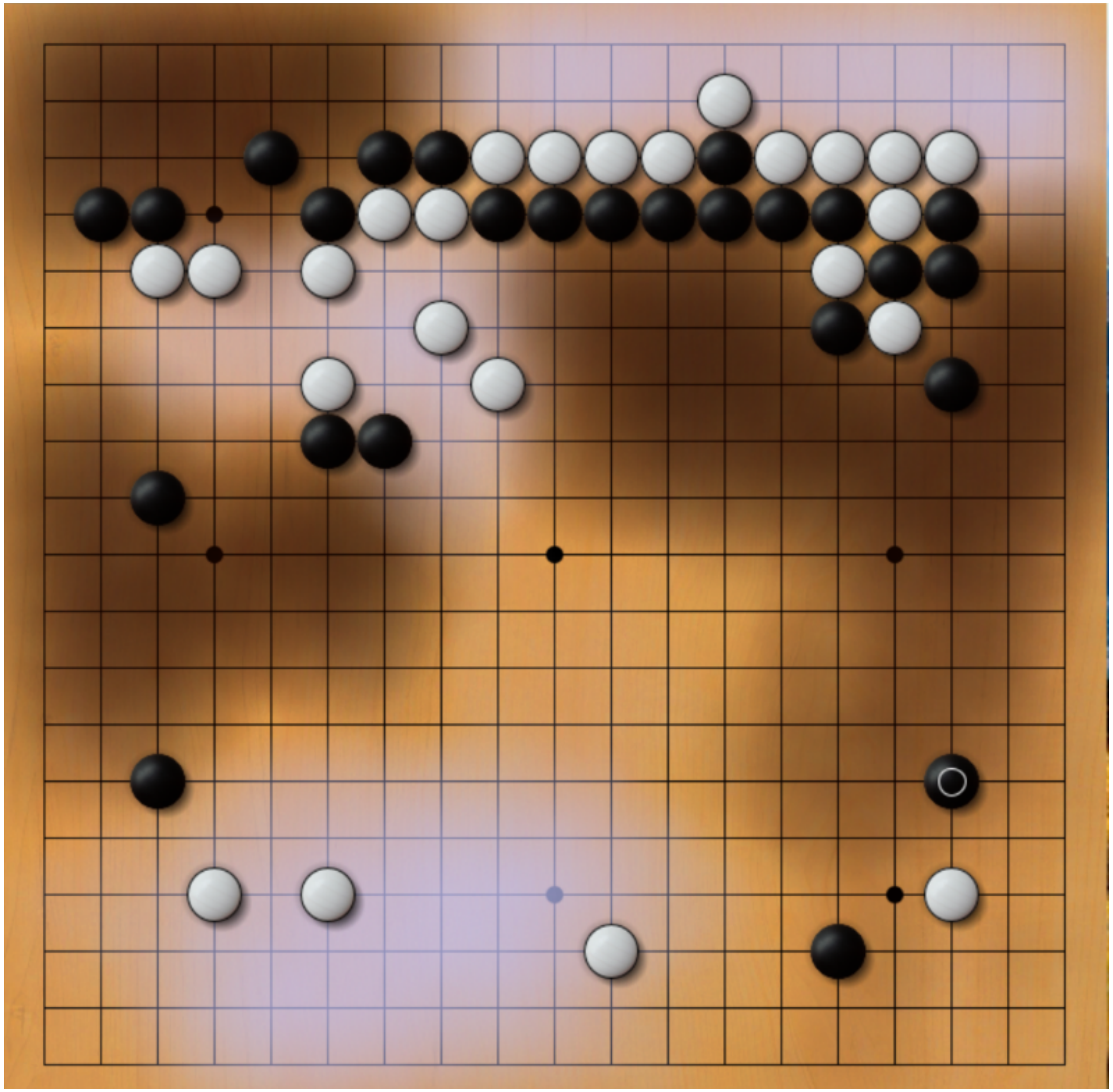
- Minimax
- Forward Pruning: only try “sensible” actions:
  connect, defend, territory jump
  Like a knowledge-based expert system
- Influence calculation for scoring
- Weak amateur level (10 kyu)
- Years of no real progress
- Then: MCTS



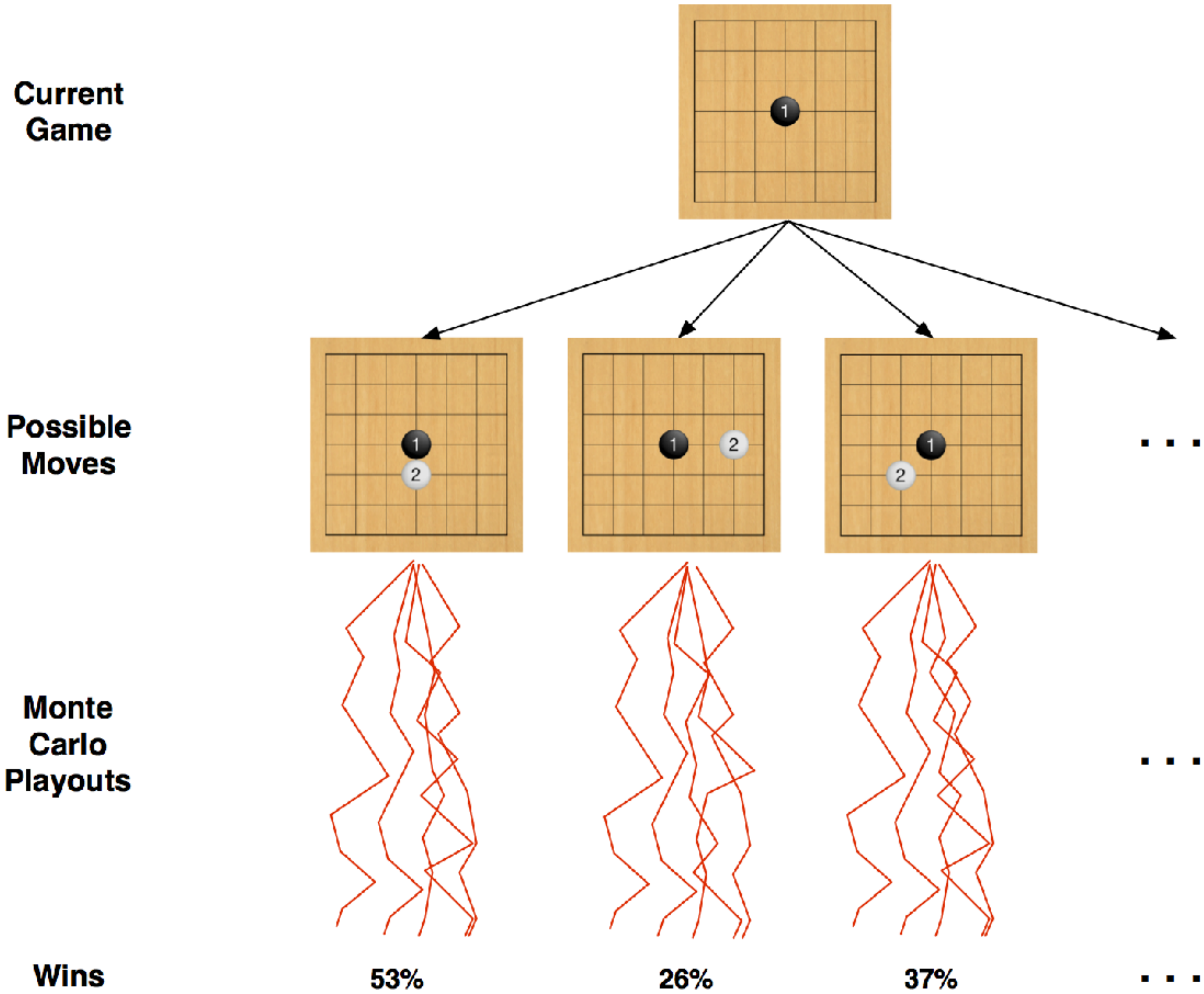
MCTS

Adaptive Sampling

- No Efficient Heuristics for influence
- Rigid Search does not work well in large, flat state space



Monte Carlo playouts



Monte Carlo Playouts

- Chess: b=10. Full width
- Go: b=200. Forward pruning
- Playout: Not search **Tree b^d** but search **Path d**

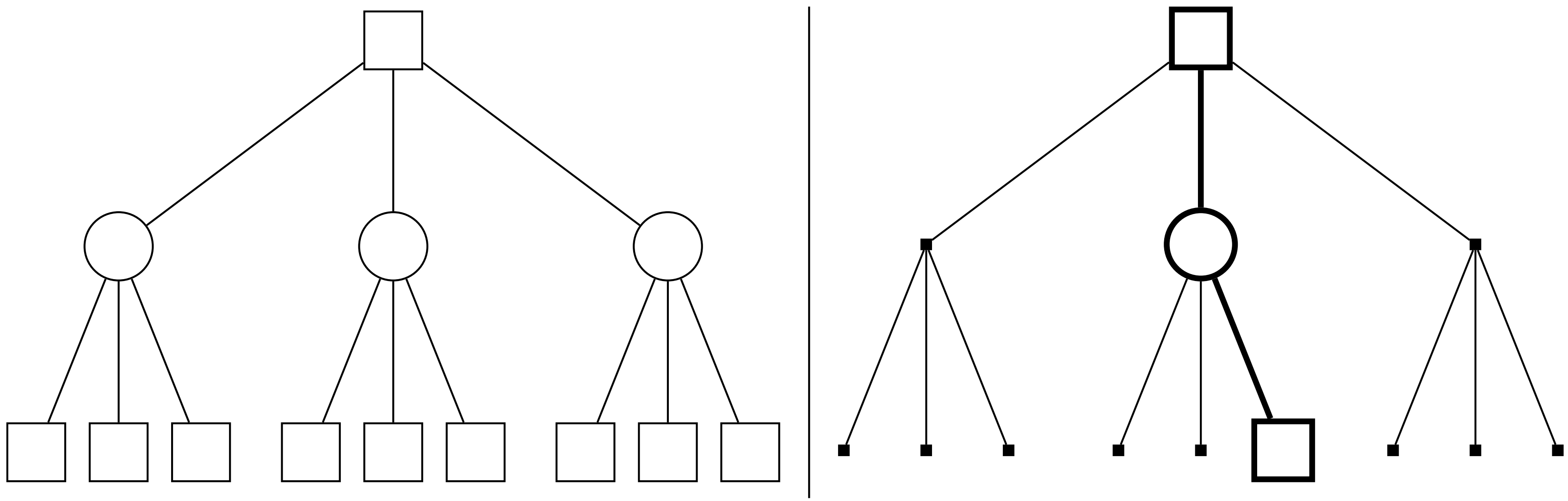


Figure 4.1: Searching a Tree vs a Path

Monte Carlo vs Minimax

- Minimax: Best of all actions
- Monte Carlo: Average of random playouts

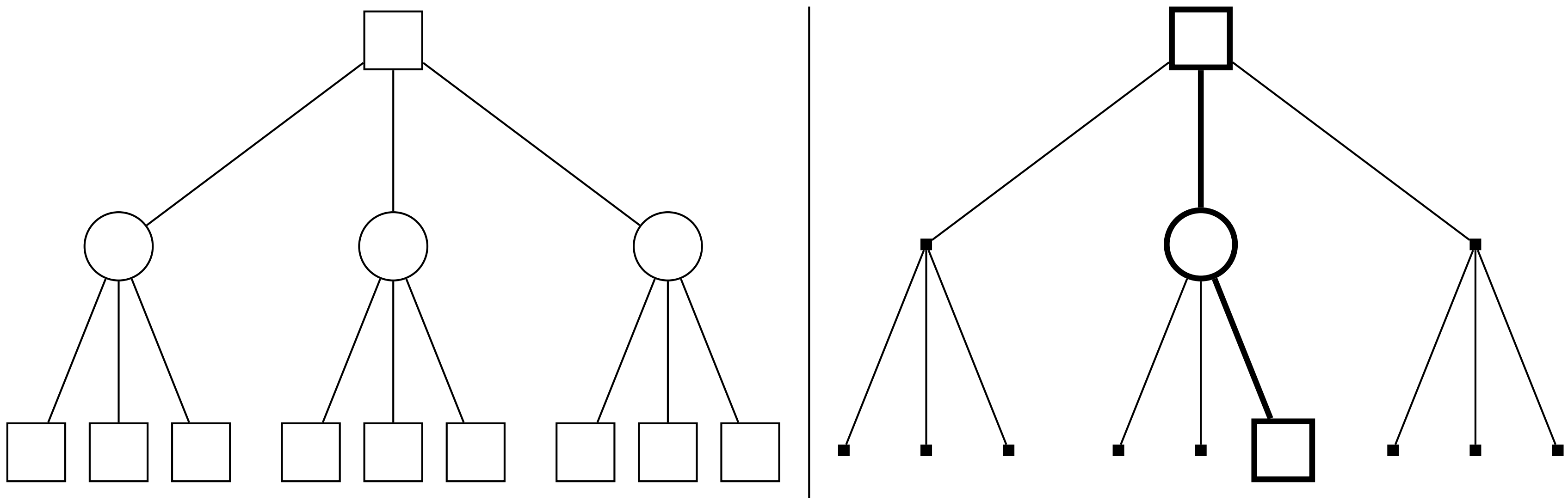


Figure 4.1: Searching a Tree vs a Path

Monte Carlo Tree Search

- Brügmann 1993 tried all playouts from the root (**flat**) Results were better than random, but **not great** -> MC

- Coulom 2006 (after work of others) tried it **recursively**, in a tree. This did give **good results** -> MCTS

- Kocsis & Szepesvari 2006 suggested the UCT selection rule to balance exploration and exploitation.
  (Based on extensive work in multi-armed bandit theory)

Four Operations

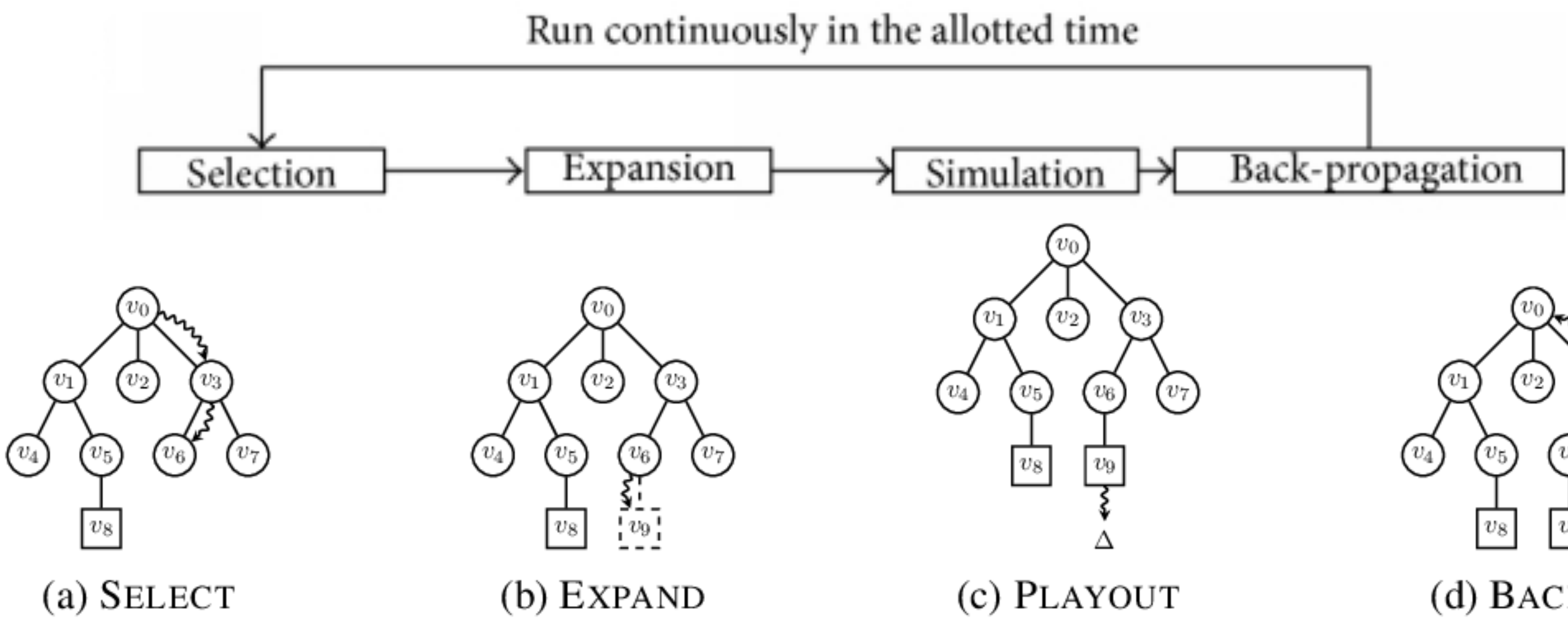


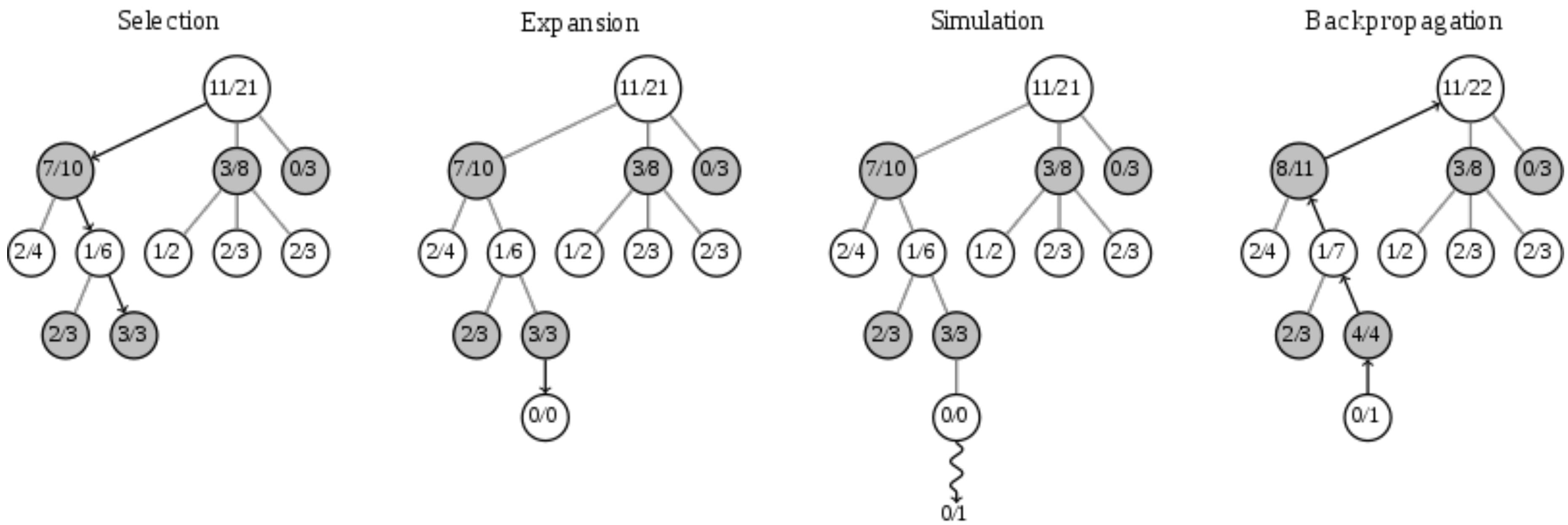
Figure 1: One iteration of MCTS.

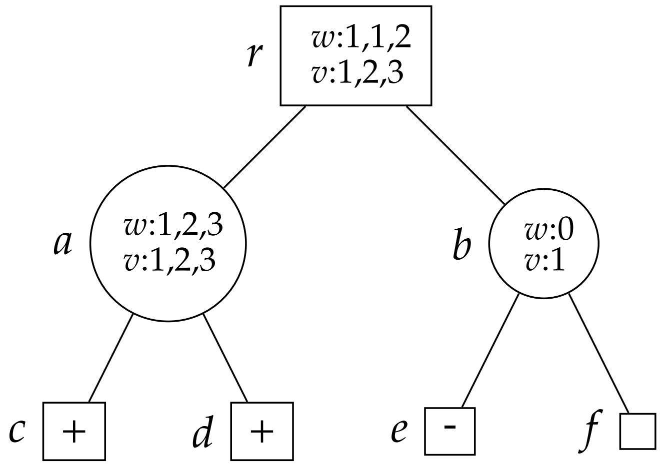
UCT Selection formula

$$\mathrm{UCT}(j) = \frac{w_j}{n_j} + C_p \sqrt{\frac{\ln n}{n_j}}$$

- UCT_j = $\mathrm{winrate}_j$ + Cp * $\mathrm{newness}_j$
- Select the child j with the highest UCT value
- Winrate is for exploitation of what is known to be good
- Newness is for exploration of lesser-searched subtrees
- Larger Cp means more exploration
- **U**pper **C**onfidence bounds applied to **T**rees

Example

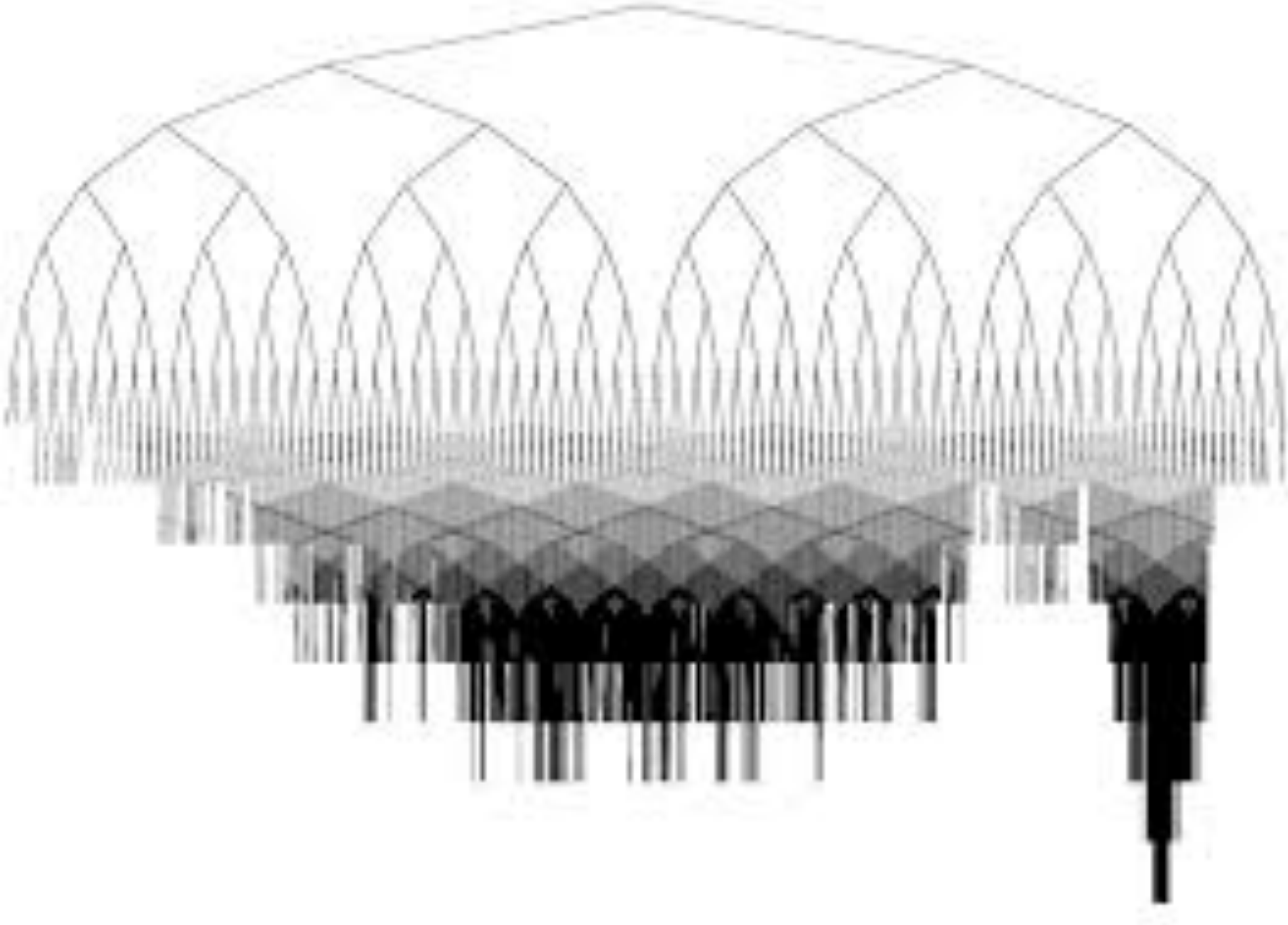

Selection
Expansion
Simulation
Backpropagation
11/21
7/10
3/8
0/3
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
0/1
11/22
8/11
1/7
4/4



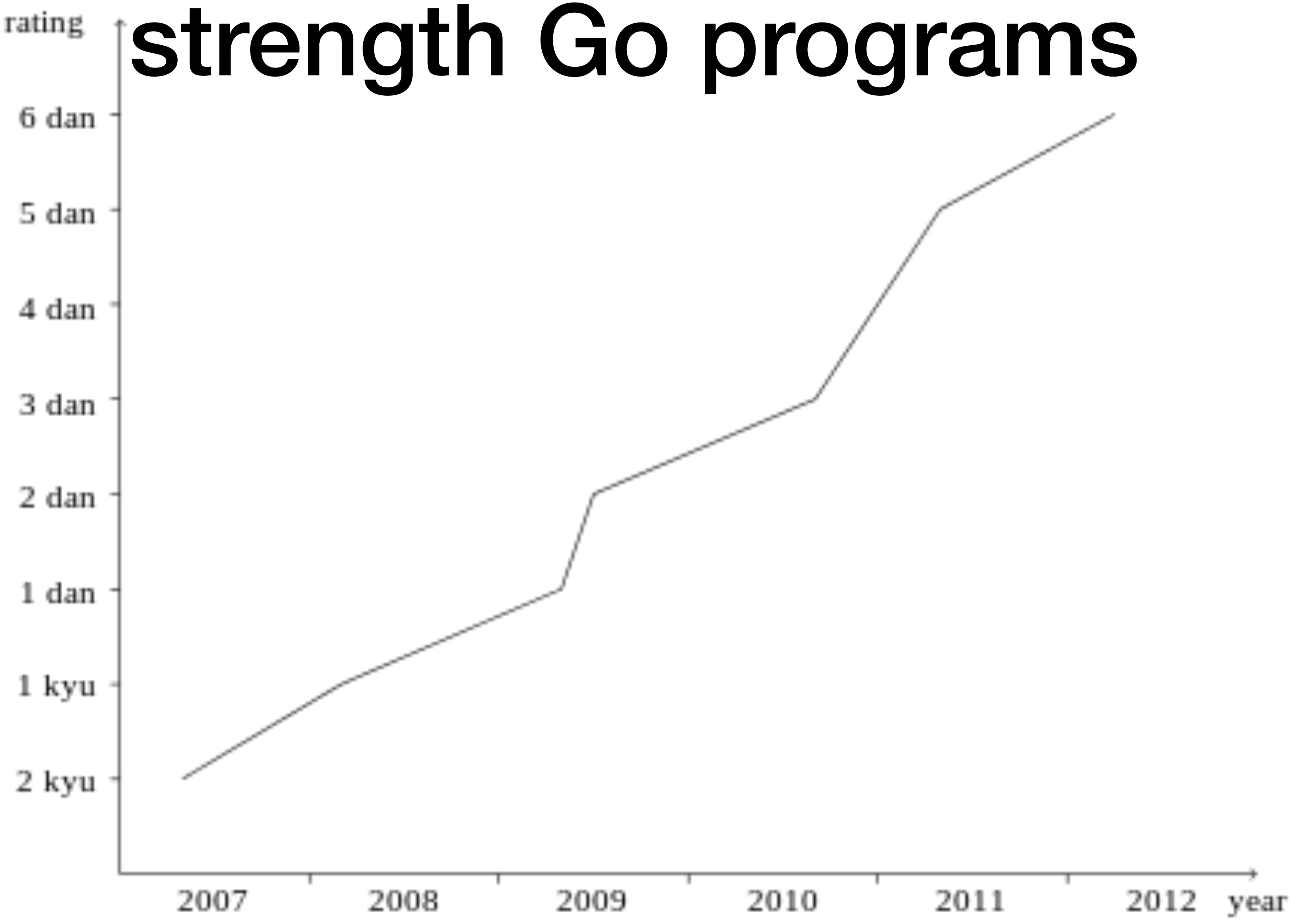
Example

- 1
- Select r
- expand a, playout a -> d=+, update a&r win++, visit++
- 2
- Select r
- expand b, playout b -> e=-, update b&r, visit++
- 3
- select UCT(a) 1/1+1*sqrt(ln 1/1) = 1, UCT(b) 0/1+1*sqrt(ln 1/1)=0
- Playout a -> c=+, update a&r, win++, visit++
- 4
- Select r
- select UCT(a) 2/2+1*sqrt(ln 2/2) = 1.588, UCT(b) 0/1+1*sqrt(ln 2/1)=0.832
- expand c, playout c=+, update c&a&r, win++, visit++

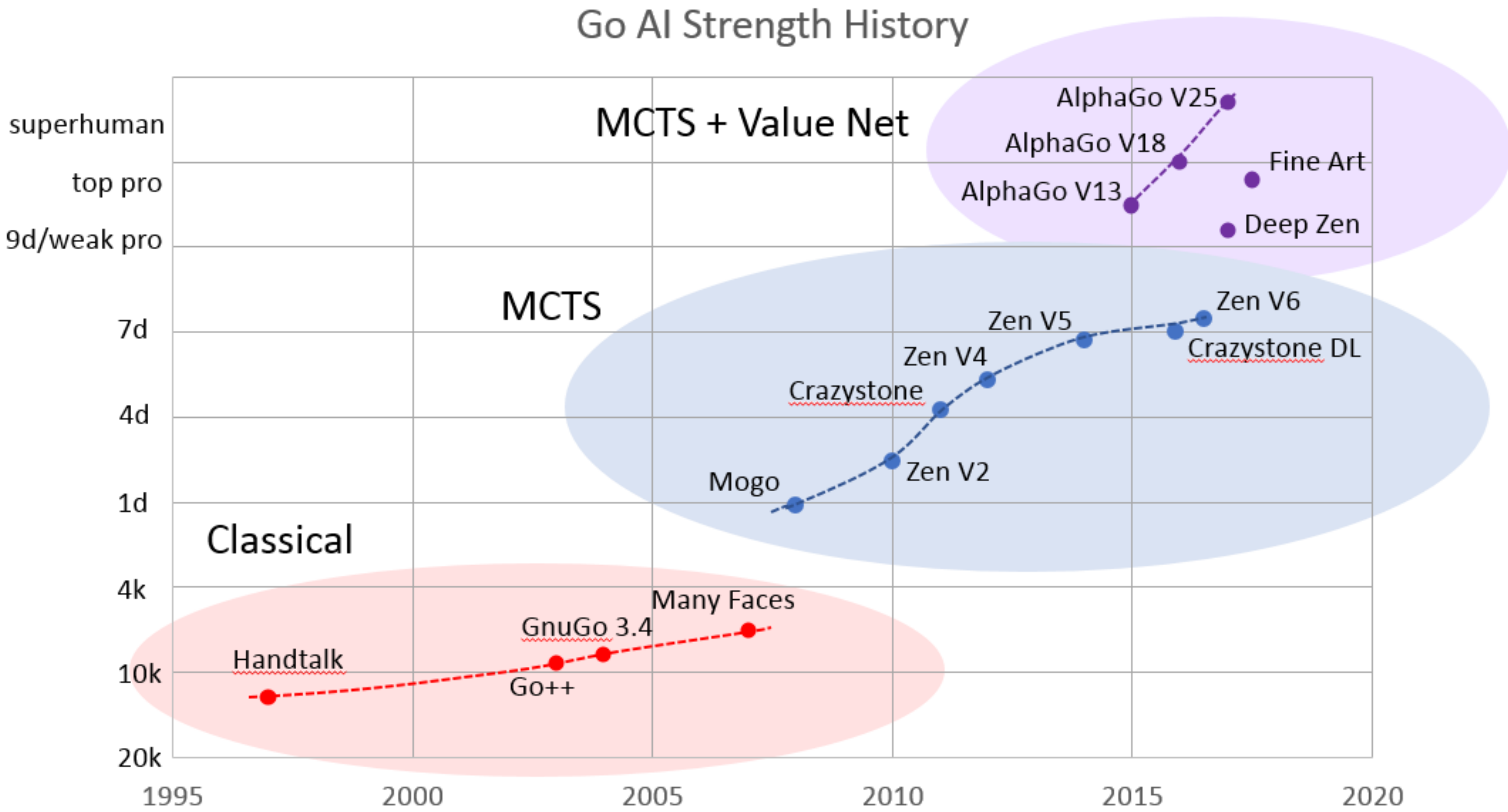
Tree Shape



Impact MCTS playing strength Go programs



MCTS Go



Go AI Strength History
superhuman
top pro
9d/weak pro
7d
4d
1d
4k
10k
20k
1995
2000
2005
2010
2015
2020
MCTS + Value Net
AlphaGo V25
AlphaGo V18
AlphaGo V13
Fine Art
Deep Zen
MCTS
Zen V6
Zen V5
Crazystone DL
Zen V4
Crazystone
Zen V2
Mogo
Classical
Many Faces
GnuGo 3.4
Handtalk
Go++

Questions?

2. Example-level self play

AlphaGo 1 2 3

1. AlphaGo: The Champion

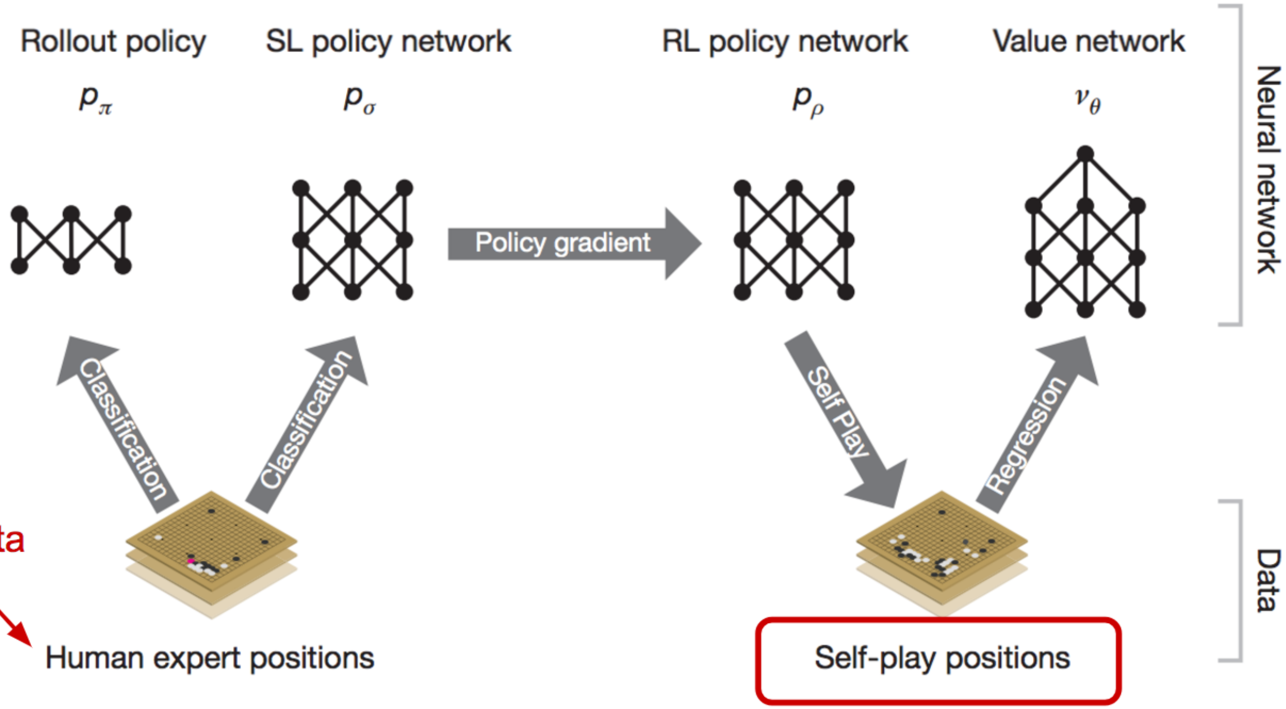
2. AlphaGo Zero: Tabula Rasa
   The Self-Learner

3. AlphaZero: Three games: Chess, Shogi, Go.
   The Generalist

AlphaGo Structure

- 4 nets
    - fast rollout policy
    - slow sl policy
    - slow rl policy
    - value net

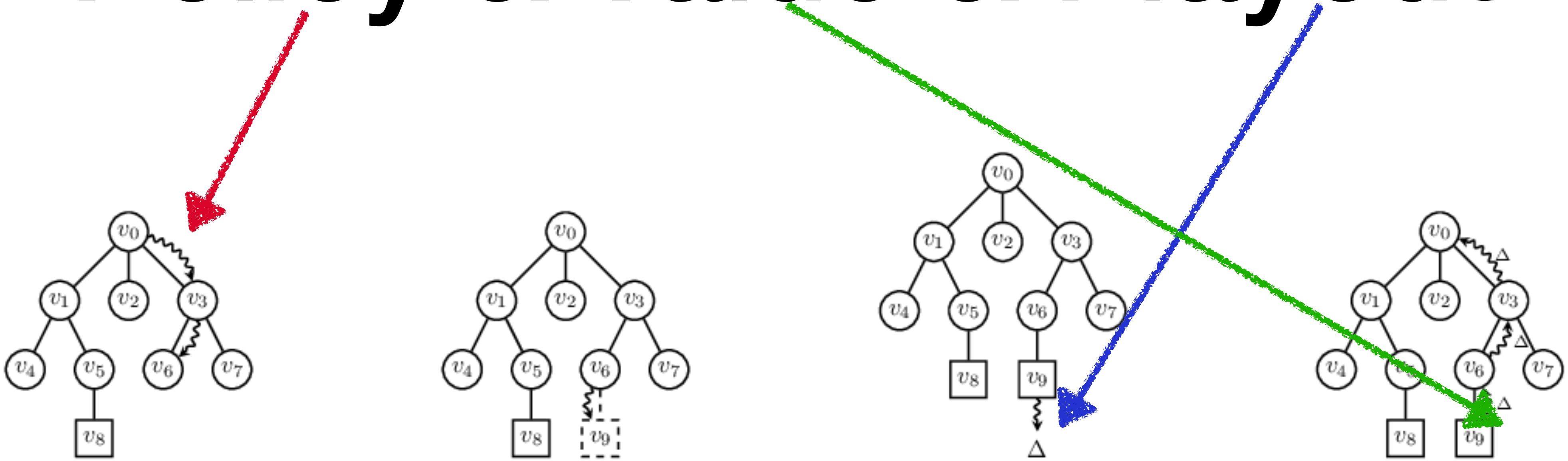
Traditional way to collect training data



Generating a lot of additional training data by self-play.

- 3 learning methods
    - supervised small patterns fast rollout policy
    - supervised database grandmaster games
    - reinforcement from database de-correlated self-play games

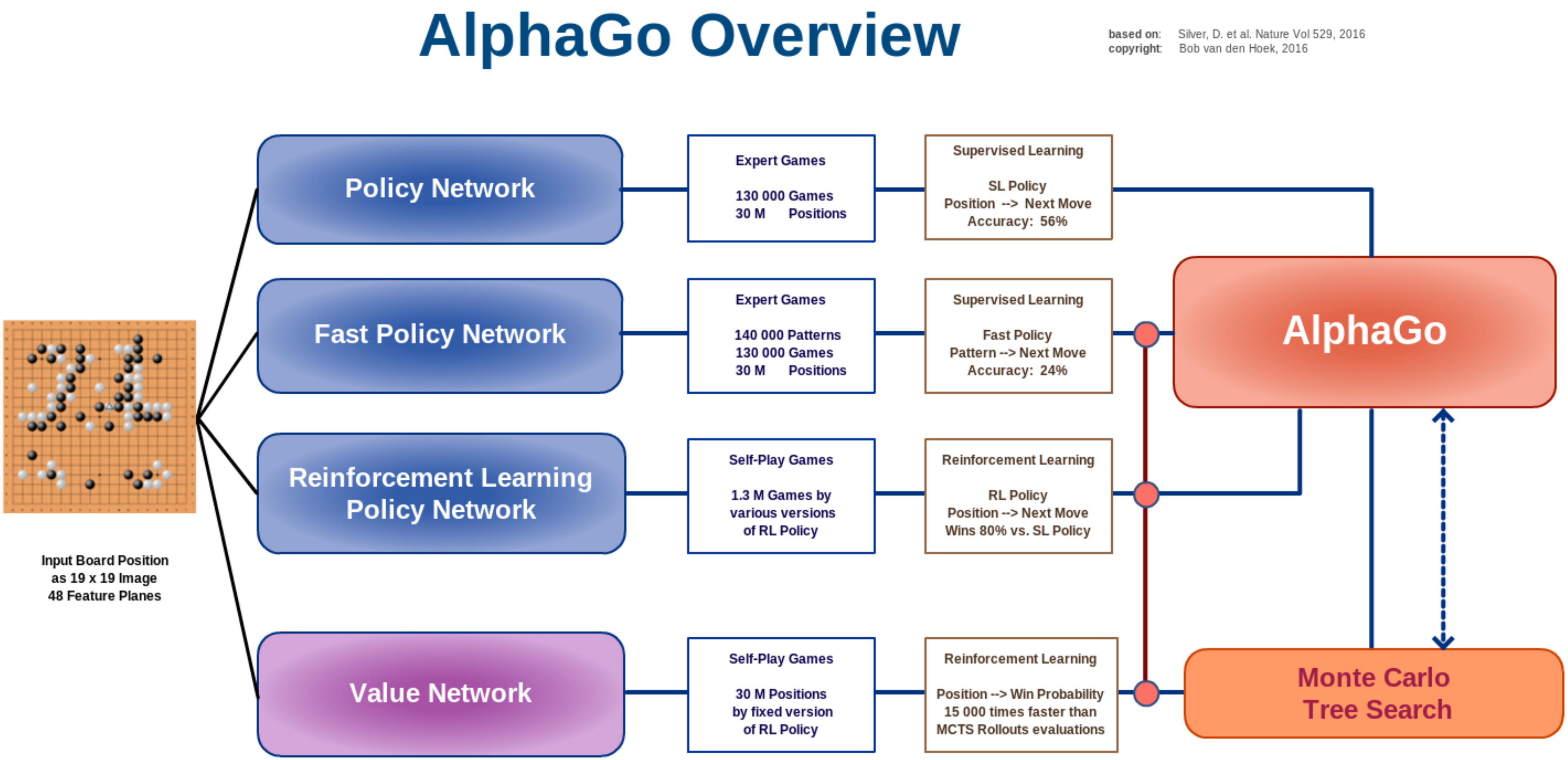
Policy & Value & Playout



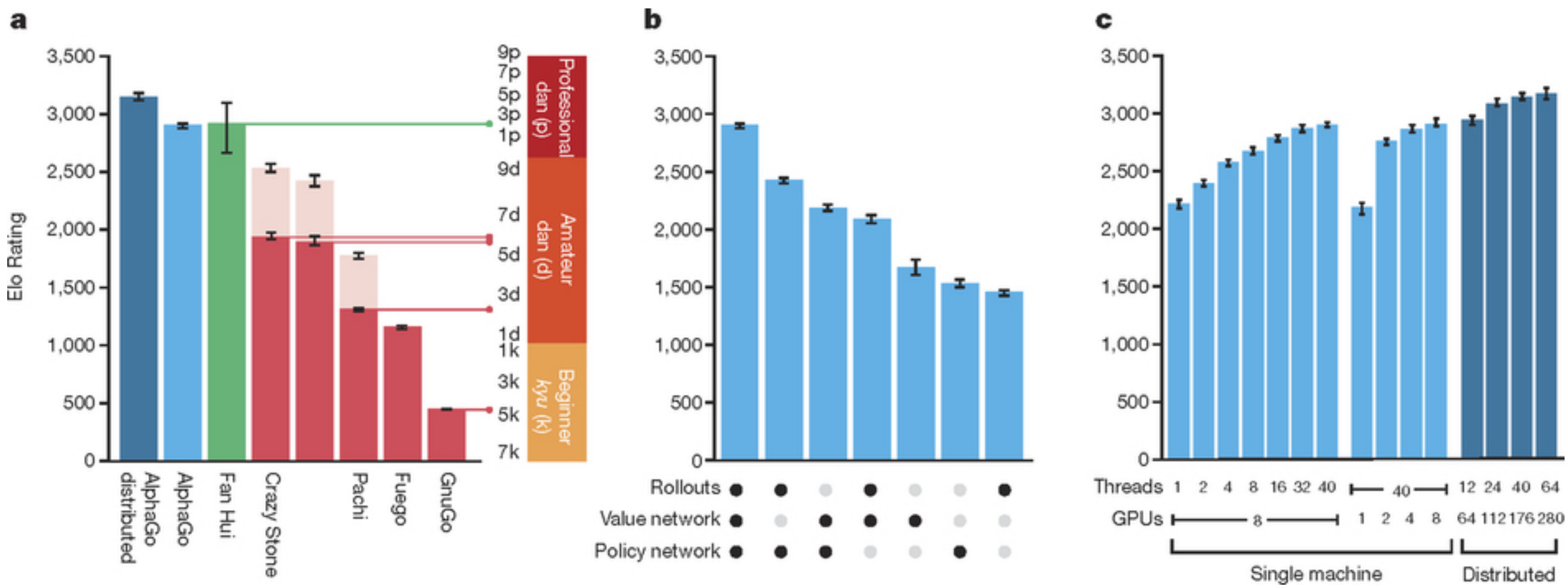
(a) SELECT (b) EXPAND (c) PLAYOUT (d) BACKUP

Figure 1: One iteration of MCTS.

AlphaGo Structure



AlphaGo Performance



AlphaGo Matches

- 2015 Fan Hui London
- 2016 Lee Sedol Seoul
- 2017 Ke Jie Wuzhen

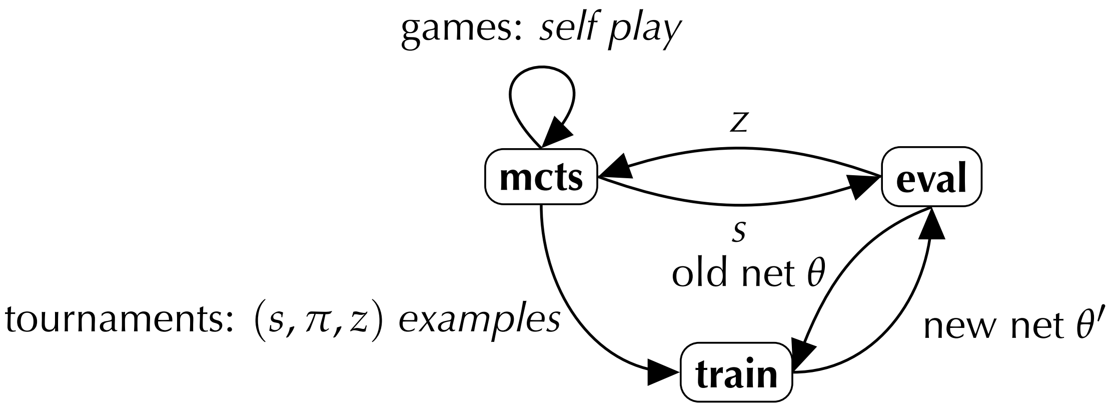

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO
CONSERVATION
SONGBIRDS À LA CARTE
RESEARCH ETHICS
SAFEGUARD TRANSPARENCY
POPULAR SCIENCE
WHEN GENES GOT 'SELFISH'
ARTICLE
doi:10.1038/nature16961
Mastering the game of Go with deep neural networks and tree search
ARTICLE
doi:10.1038/nature24270
Mastering the game of Go without human knowledge

AlphaGo Zero

- Faster
  days, not weeks
- Better
  Higher Elo
- Elegant
  1 network

Self-Play Loop

- Generate a sequence of own training examples

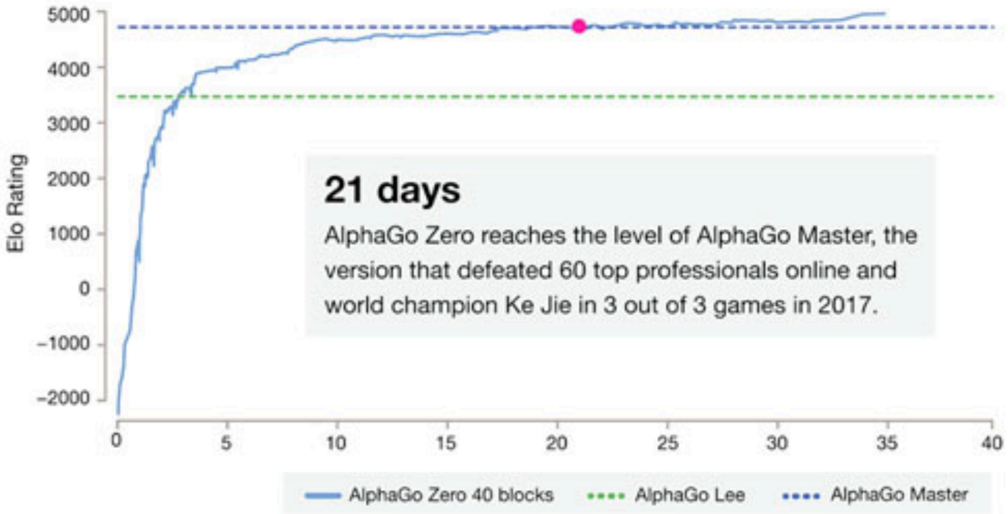


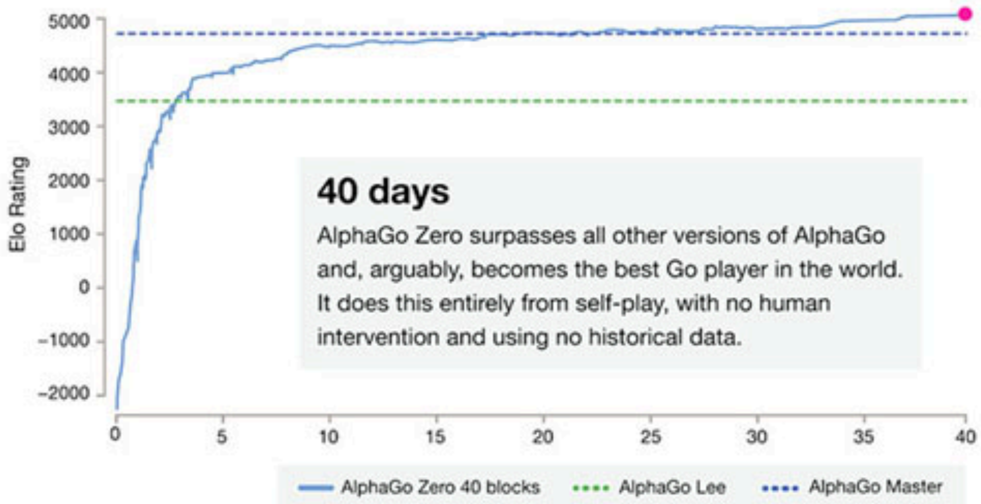

```
for it in range (1, max_iterations): # do a curric. of self-play tourn.
    for game in range(1, max_games): # play a tourn. of games; then train
        trim(triples) # if buffer full: replace old entries
        while not game_over(): # generate the moves of one game
            game_pairs += mcts(eval(net)) # move is a (state,action) pair
        triples += add(games_pairs, game_outcome()) # add win/lose to buf
    net = train(net, triples) # retrain with (state,action,outc) triples
```

AlphaGo Zero Overview

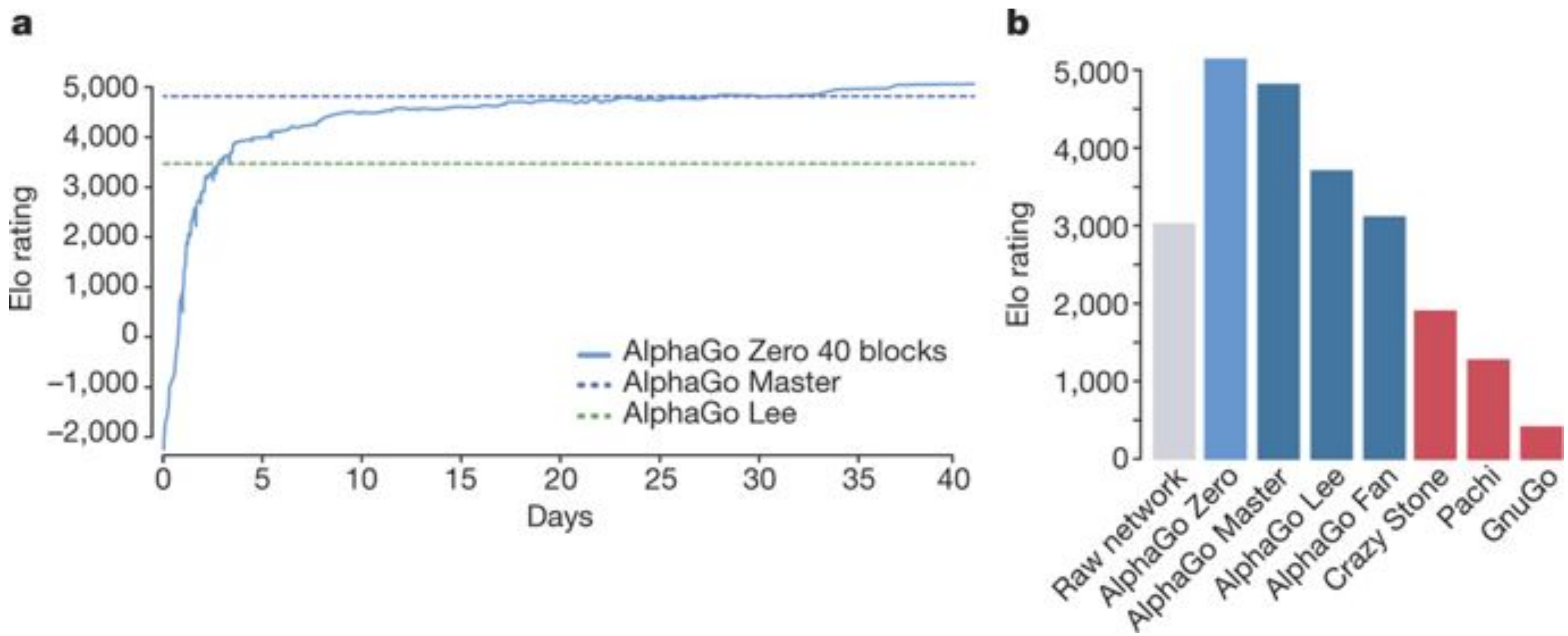
- Zero-knowledge
- One net (double-headed)
- One learning method: Self-Play
- Tabula Rasa: Only the rules & input/output layers, zero heuristics, zero grandmaster games
- Curriculum learning

AlphaGo Zero Performance

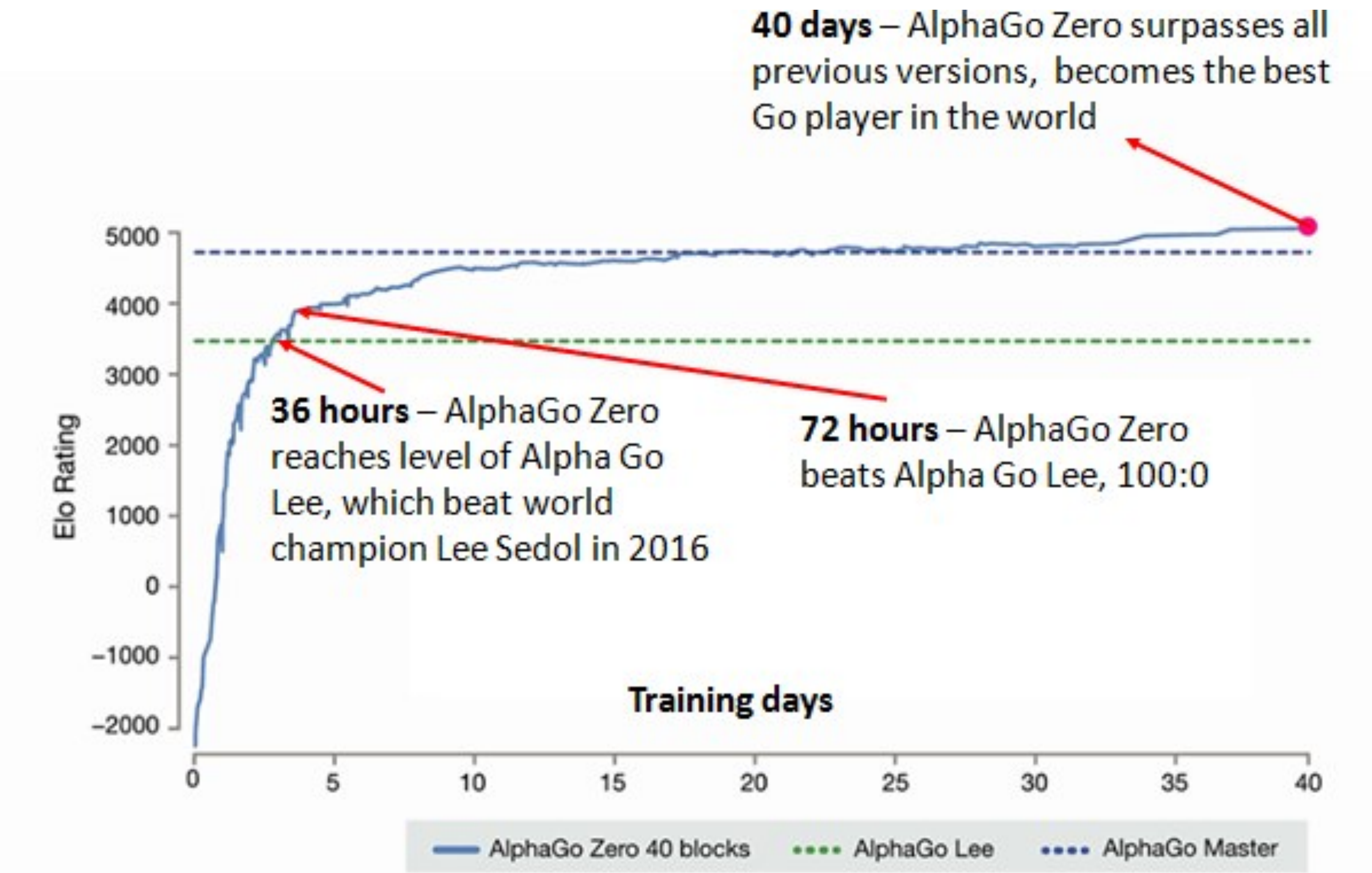




AlphaGo Zero Performance



AlphaGo Zero Performance

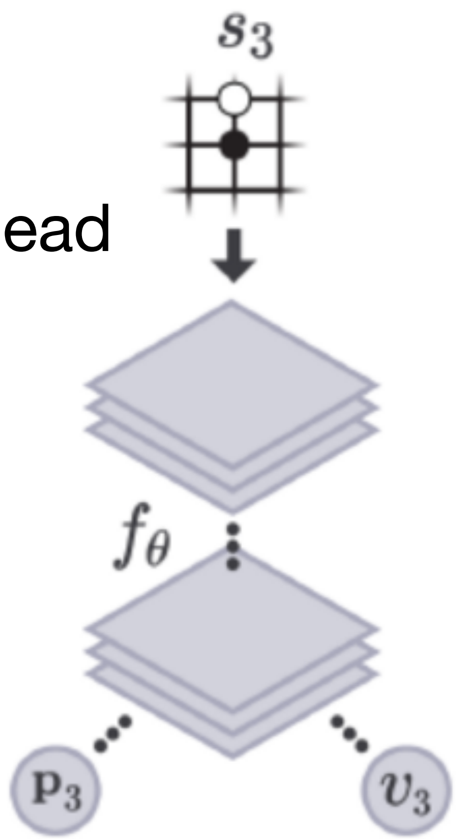


AlphaGo Zero Structure

- 1 net: ResNet with policy head and value head
  Combined loss-function

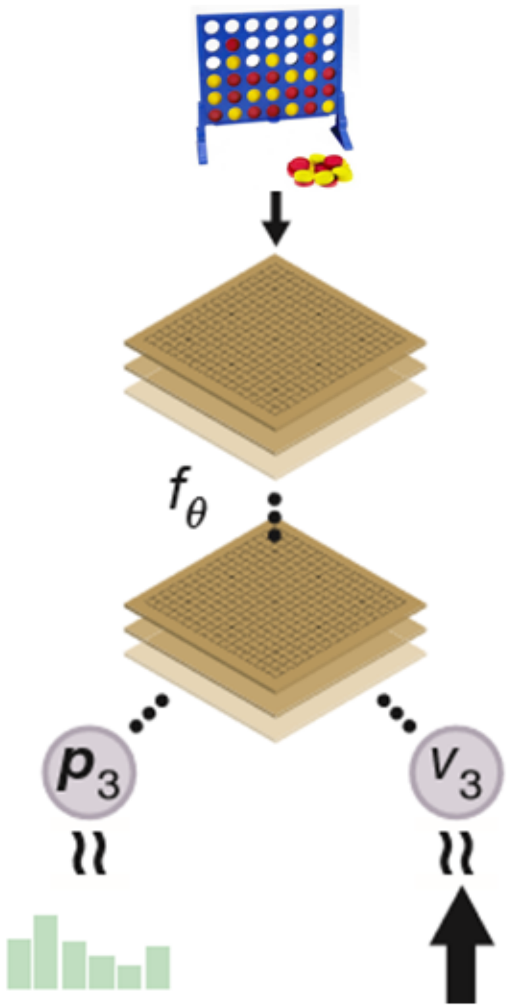
- 1 learning: RL Self-Play

- Tabula Rasa



AlphaGo Zero Structure

Input: Board state (encoded)



Outputs: *P* – Policy, *v* - value

<u>One convolution block</u>

128 filters (3X3 kernel, stride 1) + Batch norm + relu

<u>19 Res blocks</u>

Each block has 128 filters (3X3 kernel, stride 1) + Batch norm + relu + 128 filters (3X3 kernel, stride 1) + Batch norm + residual connection + relu

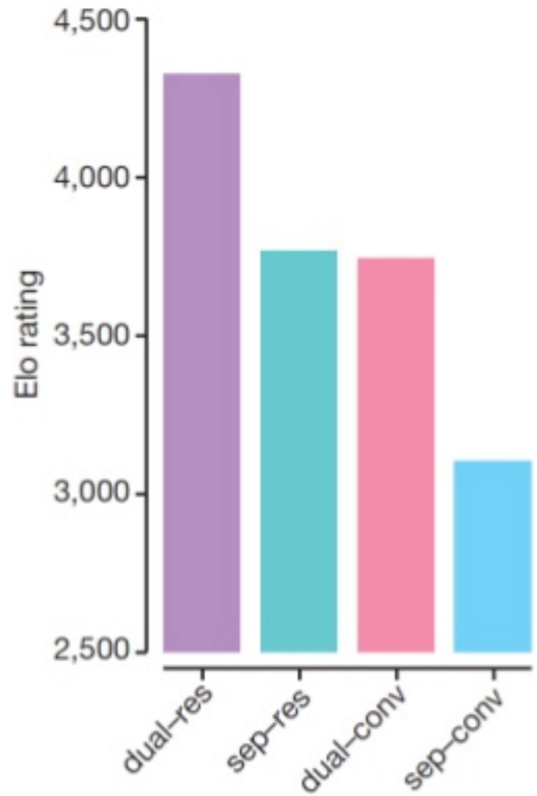
<u>One Output Block</u>

Policy: convo of 32 filters (1X1 kernel, stride 1) + batch norm + relu + linear + softmax

Value: convo of 3 filters (1X1 kernel, stride 1) + batch norm + relu + linear + relu + linear + tanh

AlphaGo Zero Networks

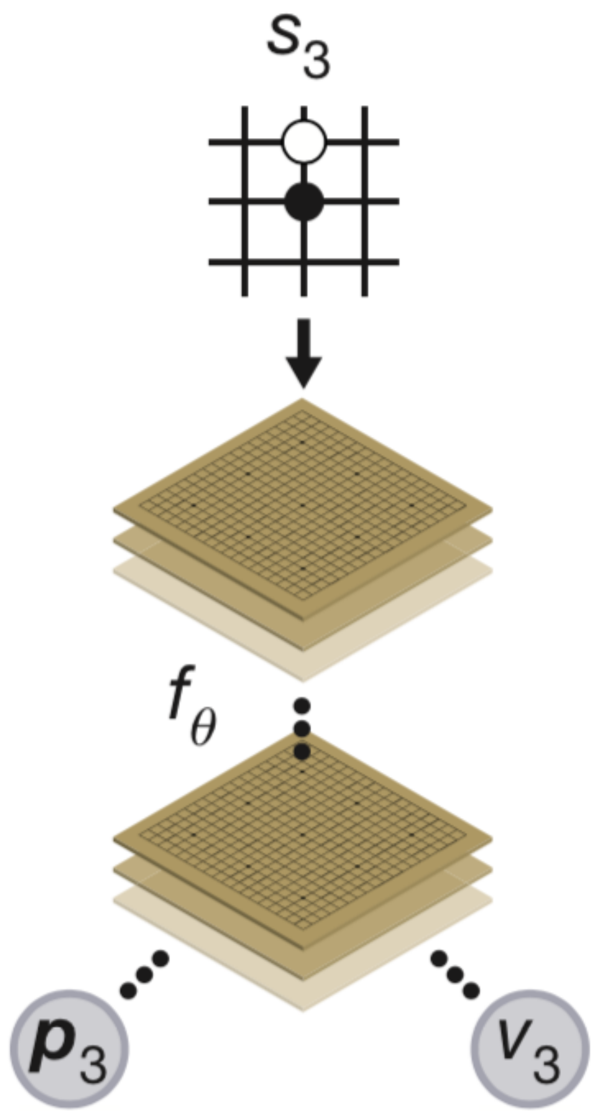
AG0: Comparison of Various Neural Network Architectures



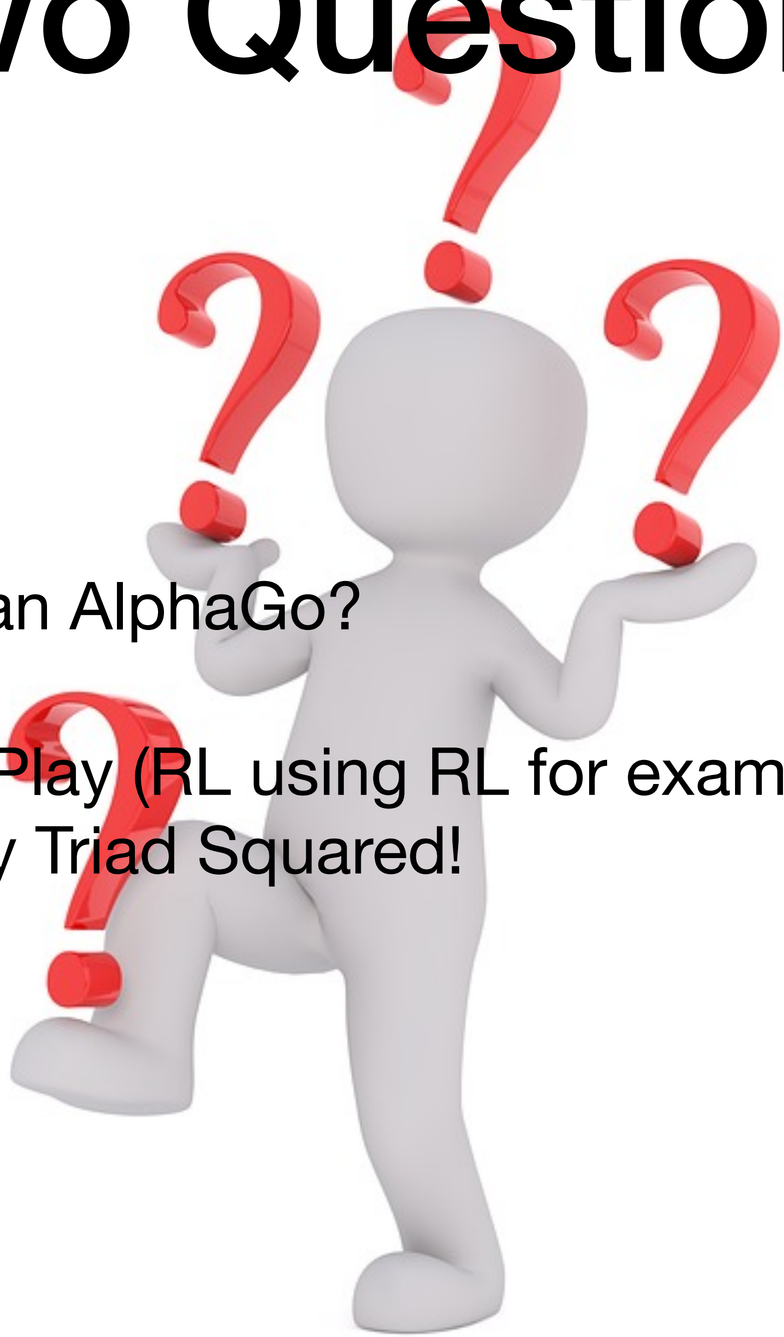
[Silver et al. 2017b]

AlphaGo Zero

- One net
- No Random Playout
- No Games database



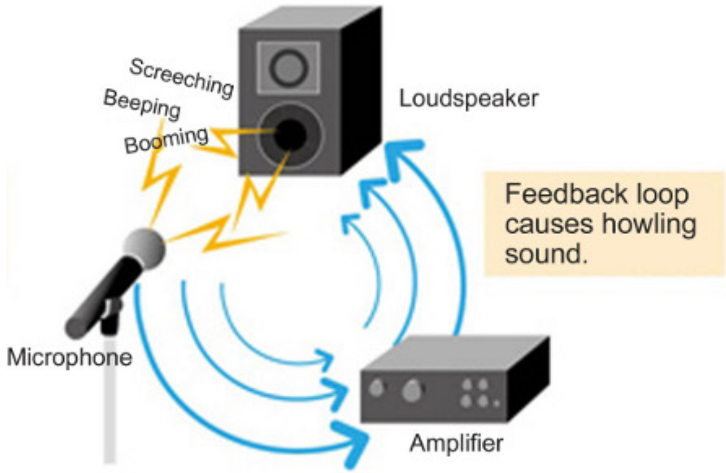
Two Questions



- Why Faster than AlphaGo?
- How can Self-Play (RL using RL for examples) ever be Stable? Deadly Triad Squared!

AlphaGo Zero

- Stable
  - Extra Exploration
  - De-correlation



- How?
  - MCTS & Noise & Exploration & Replay Buffer & Many games
  - AlphaGo Zero’s nets are not optimized against themselves, but against MCTS-improved versions of themselves

AlphaGo Zero

- How Faster?
    - Curriculum learning

3. Game-level self play

Curriculum Learning

- AlphaGo Zero learns better than AlphaGo
- AlphaGo Zero learns faster than AlphaGo. Why?
- Curriculum learning: start with easy examples
- Many small steps are faster than one large step

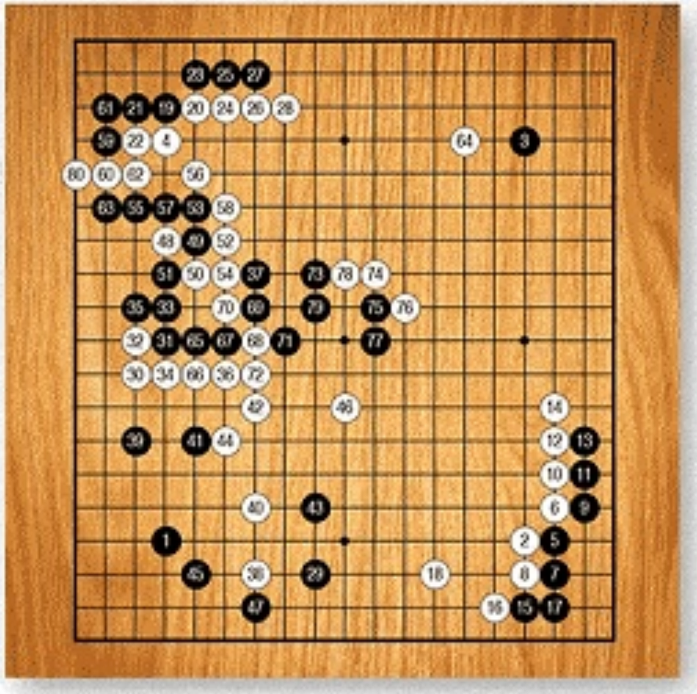
Learning to Play

Captured Stones

3 hours

AlphaGo Zero plays like a human beginner, forgoing long term strategy to focus on greedily capturing as many stones as possible.

Learning to Play



19 hours

AlphaGo Zero has learnt the fundamentals of more advanced Go strategies such as life-and-death, influence and territory.

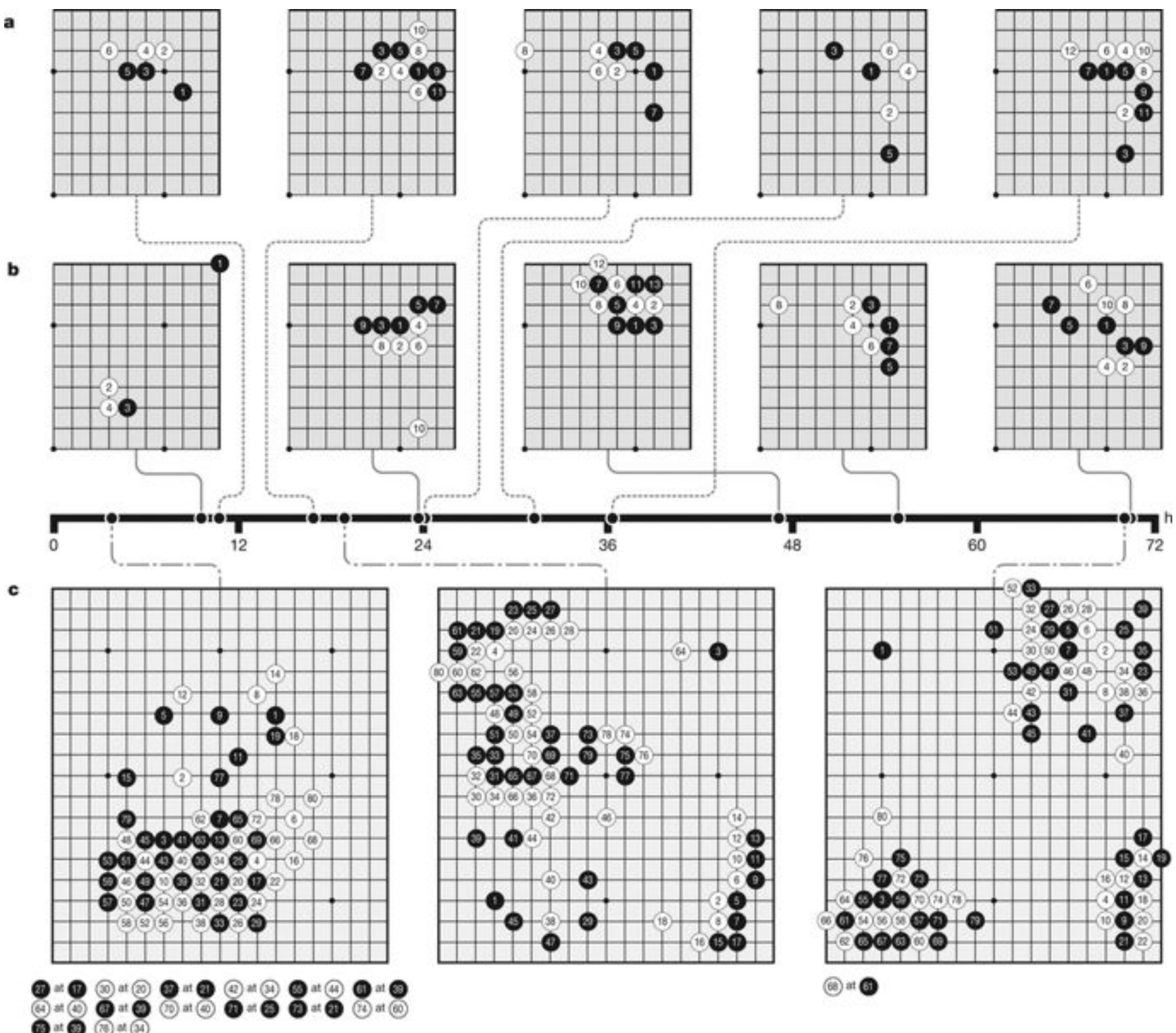
Learning to Play

Captured Stones

70 hours

AlphaGo Zero plays at super-human level. The game is disciplined and involves multiple challenges across the board.

Curriculum

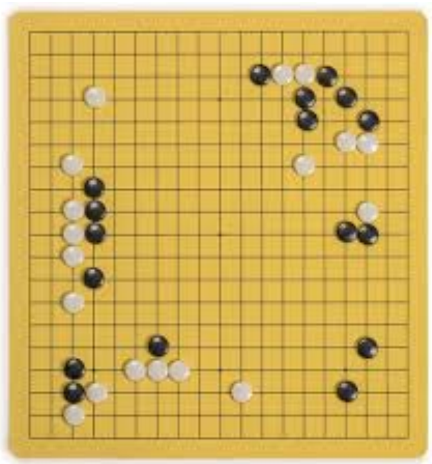


AlphaZero General


How research funders profit from hidden investments p. 1100
New books for budding scientists p. 1104
Drug leads for malaria pp. 1122 & 1129
Science
$15
7 DECEMBER 2018
AAAS
A DIGITAL
PRODIGY
AlphaZero teaches itself chess, shogi, and Go
pp. 1087, 1118, & 1140

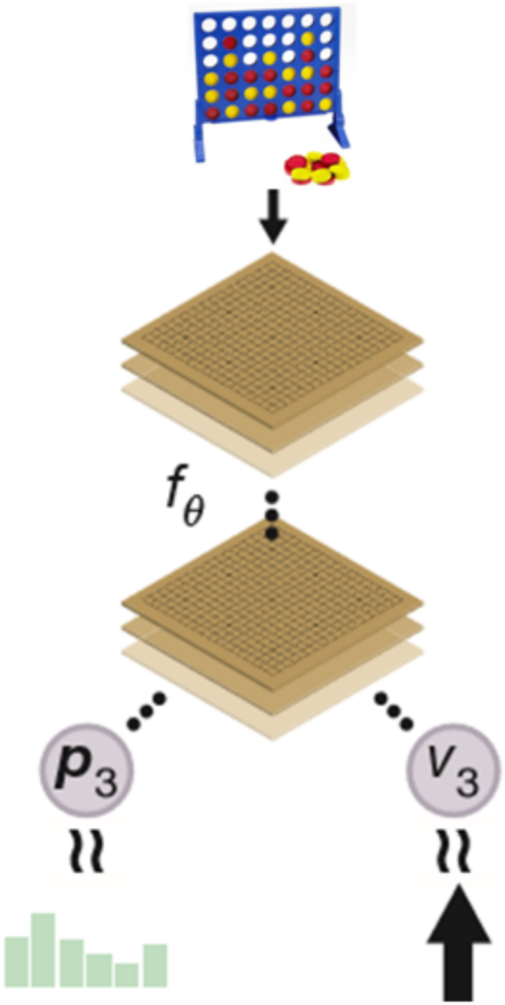
AlphaZero Overview

- Same net, same search, same tabula rasa self-play
- Different Input/Output layers
- Go, Chess, Shogi



AlphaZero Structure

Input: Board state (encoded)



Outputs: *P* – Policy, *v* - value

One convolution block

128 filters (3X3 kernel, stride 1) + Batch norm + relu

19 Res blocks

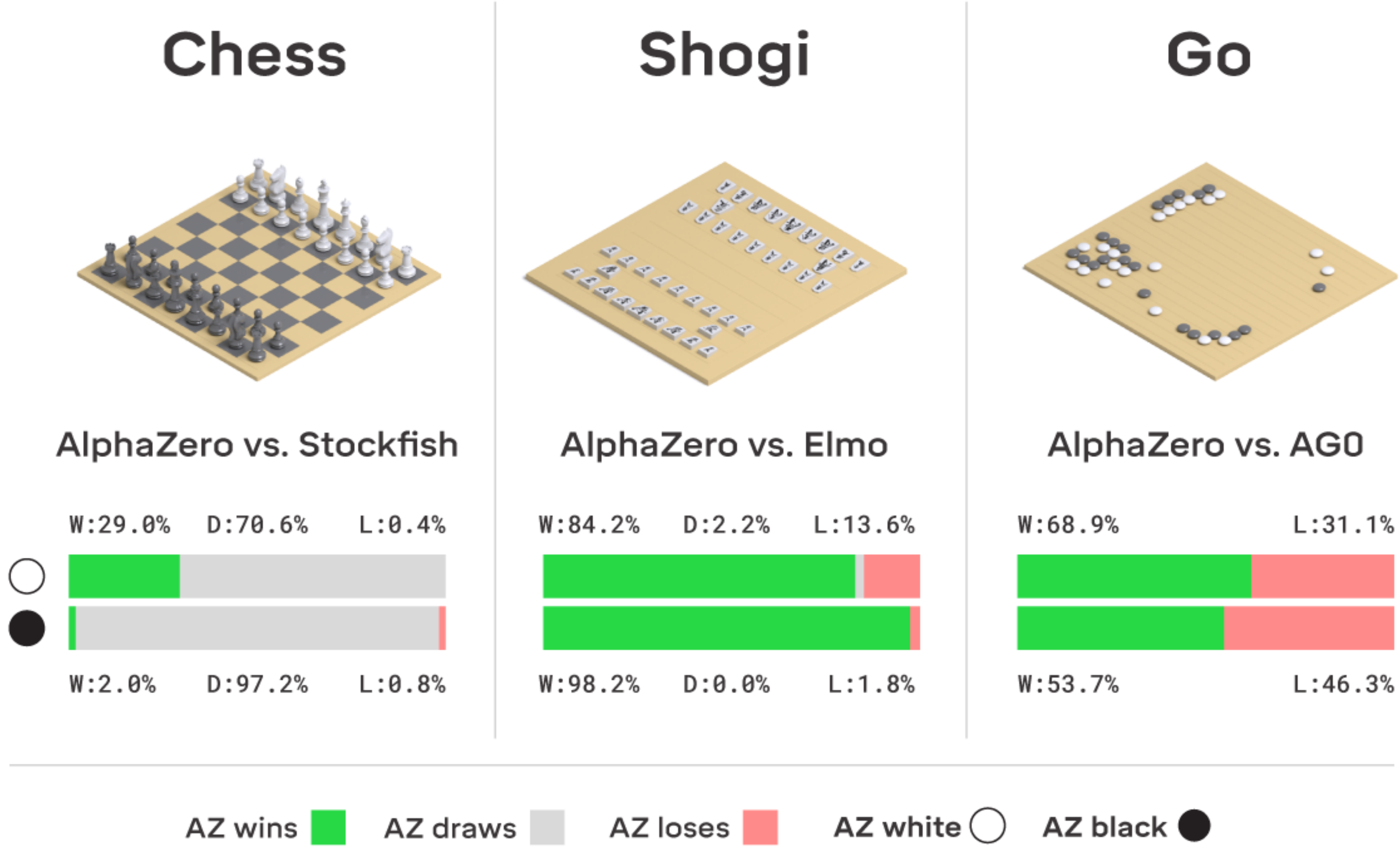
Each block has 128 filters (3X3 kernel, stride 1) + Batch norm + relu + 128 filters (3X3 kernel, stride 1) + Batch norm + residual connection + relu

One Output Block

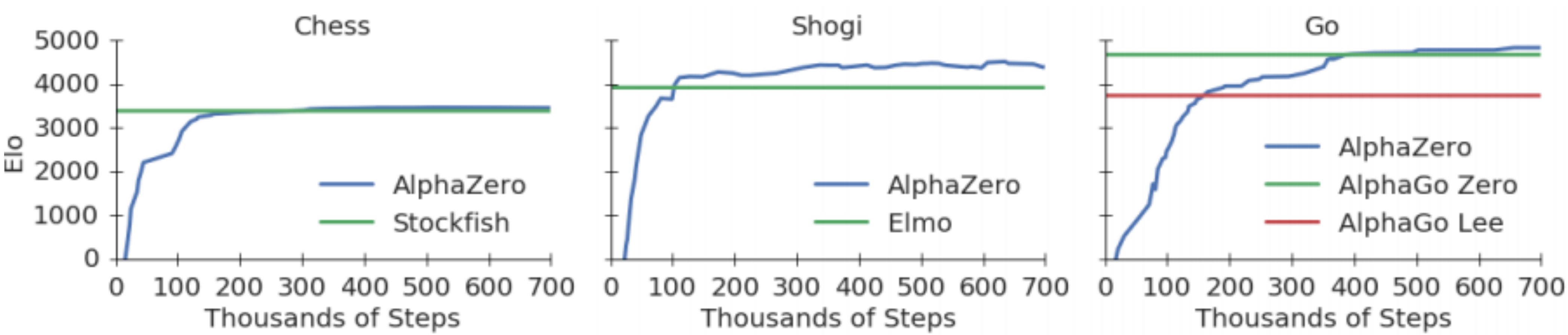
Policy: convo of 32 filters (1X1 kernel, stride 1) + batch norm + relu + linear + softmax

Value: convo of 3 filters (1X1 kernel, stride 1) + batch norm + relu + linear + relu + linear + tanh

AlphaZero Performance



AlphaZero Performance



AlphaZero Conclusions

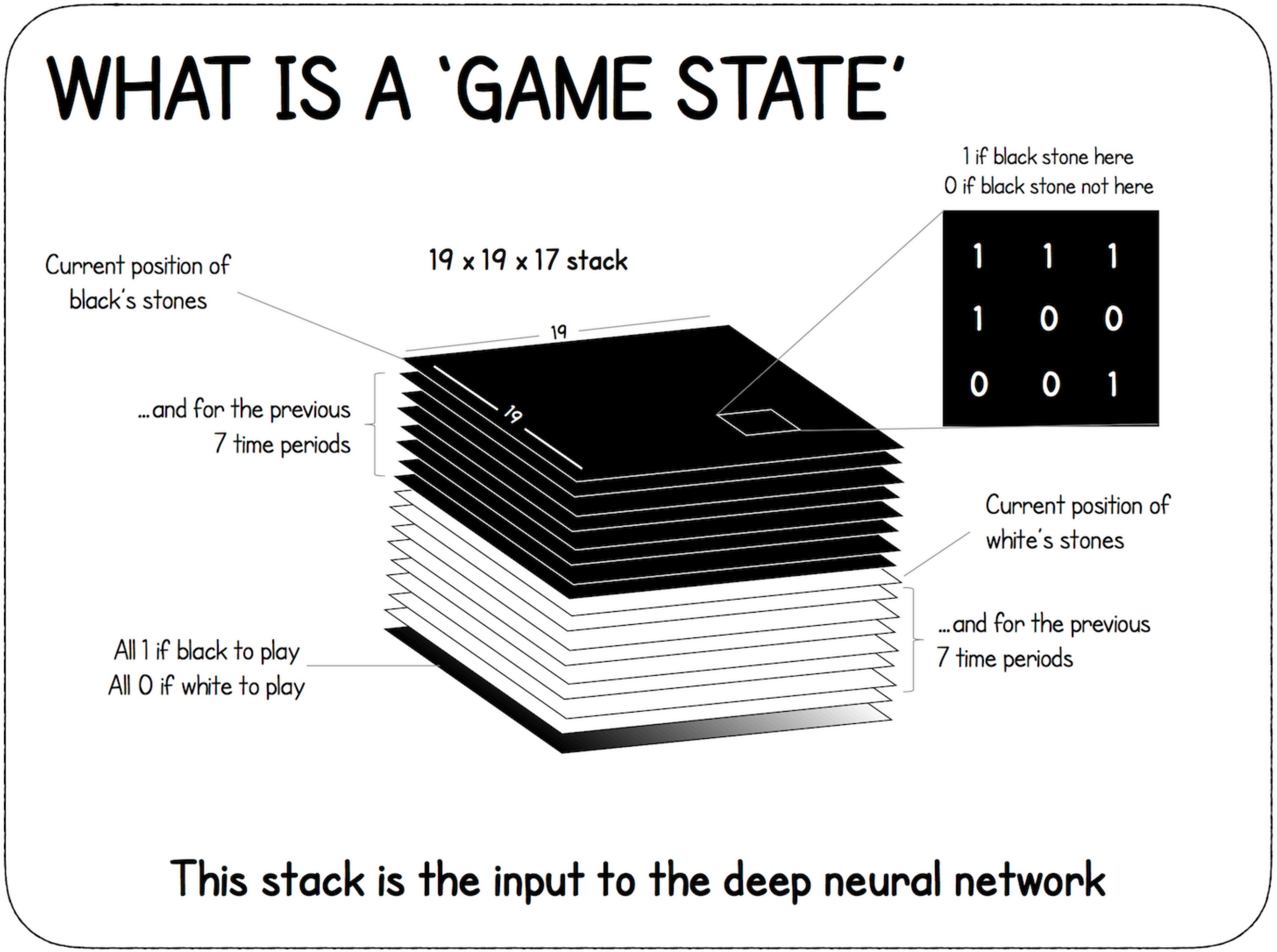
- First time learning, neural nets, and MCTS work in Chess
- Decades of heuristic planning research are surpassed
- Three Games share a general essence, since same architecture works (except I/O)
- Not same net. Net trained for Chess does not work for Shogi
- First architecture achieving very high performance in three games

Curriculum and other learning

Curriculum Learning & Friends

- **Learning** is Generalization from example to example
- **Curriculum** learning easy to hard concepts
- **Multi-task** learning two tasks at the same time
- **Transfer** learning from problem to problem

Game State


WHAT IS A 'GAME STATE'
1 if black stone here
0 if black stone not here
19 x 19 x 17 stack
Current position of black's stones
19
19
1 1 1
1 0 0
0 0 1
...and for the previous 7 time periods
Current position of white's stones
...and for the previous 7 time periods
All 1 if black to play
All 0 if white to play
This stack is the input to the deep neural network

ALPHAGO ZERO CHEAT SHEET

The training pipeline for AlphaGo Zero consists of three stages, executed in parallel

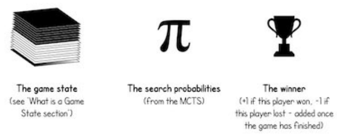
SELF PLAY

Create a 'training set'

The best current player plays 25,000 games against itself

See MCTS section to understand how AlphaGo Zero selects each move

At each move, the following information is stored



RETRAIN NETWORK

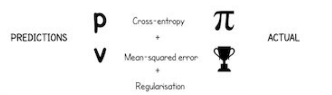
Optimise the network weights

A TRAINING LOOP

Sample a mini-batch of 2048 positions from the last 500,000 games

Retrain the current neural network on these positions
– The game states are the input (see 'Deep Neural Network Architecture')

Loss function
Compares predictions from the neural network with the search probabilities and actual winner



After every 1,000 training loops, evaluate the network

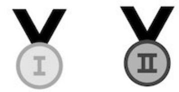
EVALUATE NETWORK

Test to see if the new network is stronger

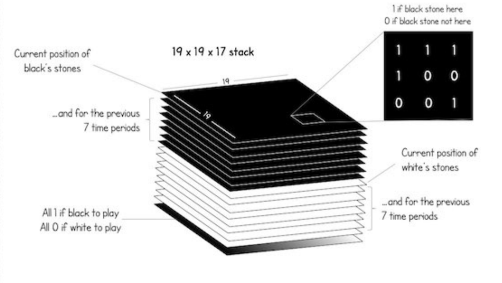
Play 400 games between the latest neural network and the current best neural network

Both players use MCTS to select their moves, with their respective neural networks to evaluate leaf nodes

Latest player must win 55% of games to be declared the new best player



WHAT IS A 'GAME STATE'



This stack is the input to the deep neural network

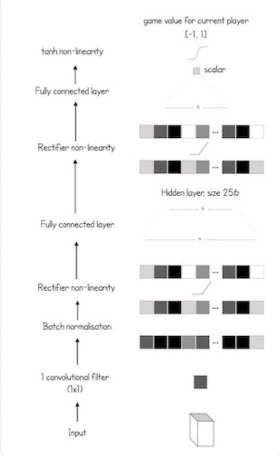
THE DEEP NEURAL NETWORK ARCHITECTURE

How AlphaGo Zero assesses new positions

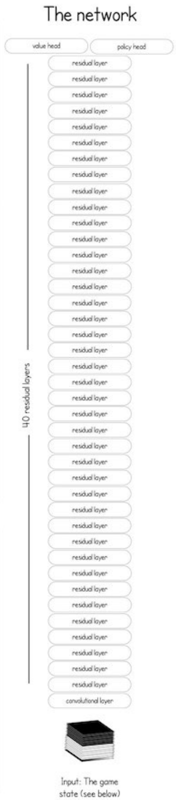
The network learns 'tabula rasa' (from a blank slate)

At no point is the network trained using human knowledge or expert moves

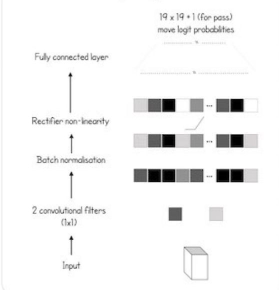
The value head



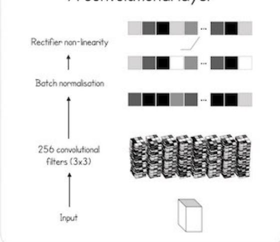
The network



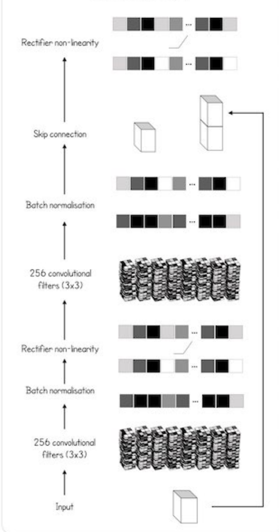
The policy head



A convolutional layer

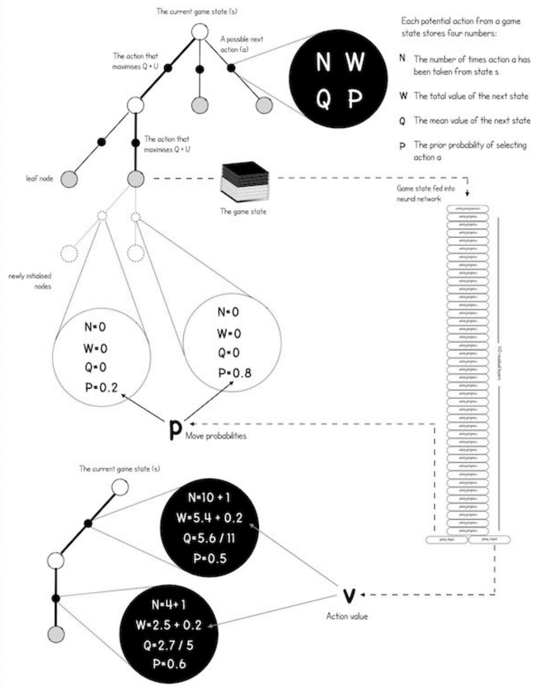


A residual layer



MONTE CARLO TREE SEARCH (MCTS)

How AlphaGo Zero chooses its next move



First, run the following simulation 1,600 times...

Start at the root node of the tree (the current game state)

1. Choose the action that maximises...

$$Q + U$$

The mean value of the next state

A function of P and N that increases if an action hasn't been explored much, relative to the other actions, or if the prior probability of the action is high

Early on in the simulation, U dominates (more exploration), but later, Q is more important (less exploration)

2. Continue until a leaf node is reached

The game state of the leaf node is passed into the neural network, which outputs predictions about two things:

p Move probabilities

v Value of the state (for the current player)

The move probabilities p are attached to the new feasible actions from the leaf node

3. Backup previous edges

Each edge that was traversed to get to the leaf node is updated as follows:

$$N \rightarrow N + 1$$
$$W \rightarrow W + v$$
$$Q = W / N$$

...then select a move

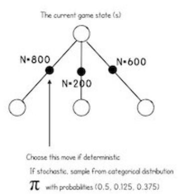
After 1,600 simulations, the move can either be chosen:

Deterministically (for competitive play)
Choose the action from the current state with greatest **N**

Stochastically (for exploratory play)
Choose the action from the current state from the distribution

$$\pi \sim N^{1/\tau}$$

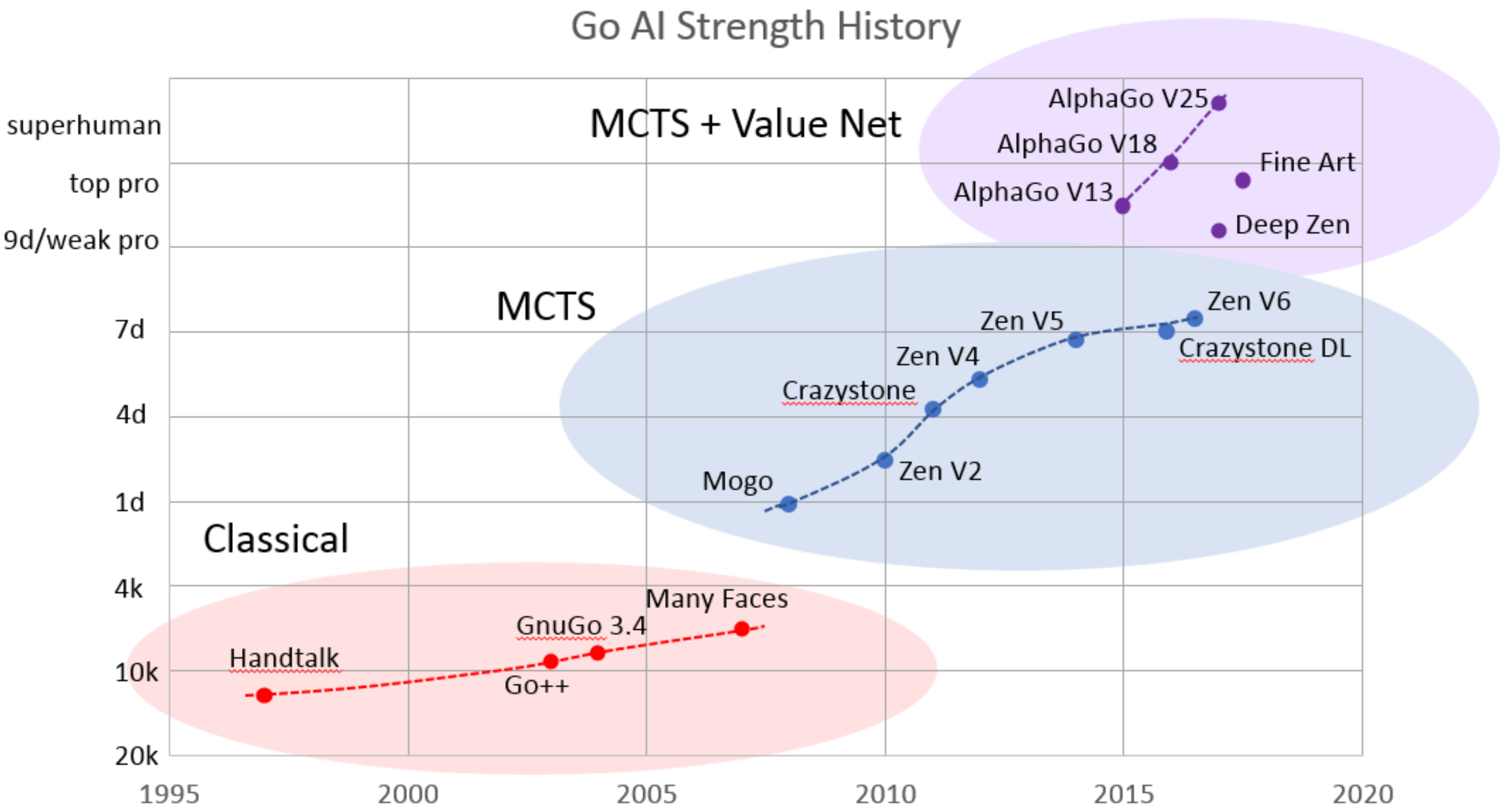
where τ is a temperature parameter, controlling exploration



Other points

- The sub-tree from the chosen move is retained for calculating subsequent moves
- The rest of the tree is discarded

AlphaGo Performance



Open Source AlphaZero Reimplementations

- Leela
- ELF Facebook
- AlphaZero General Stanford
- PhoenixGo Tencent
- PolyGames Facebook