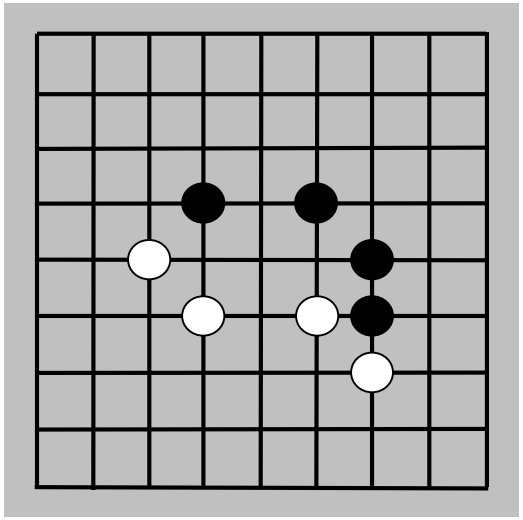
Outline

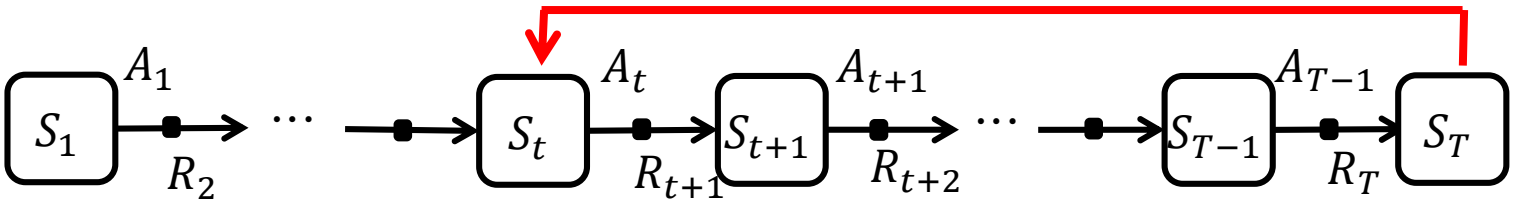
- Monte-Carlo Tree Search Review
- AlphaGo
- AlphaZero
- MuZero
  - Deterministic, Stochastic
- Gumble MuZero/AlphaZero
- EfficientZero

Monte-Carlo Tree Search Review

Monte Carlo Learning in Go

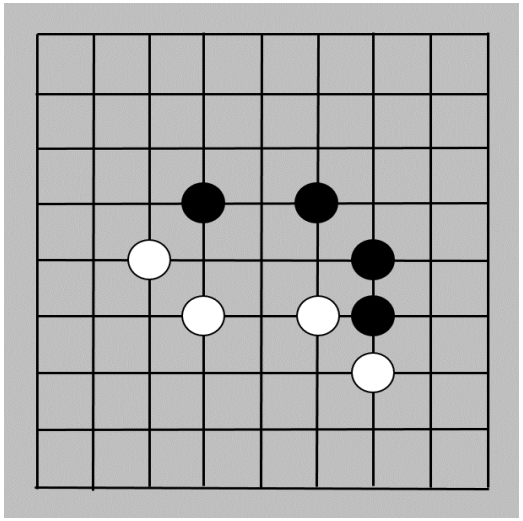
- How to evaluate this position?
  - Why not play randomly until the game end?

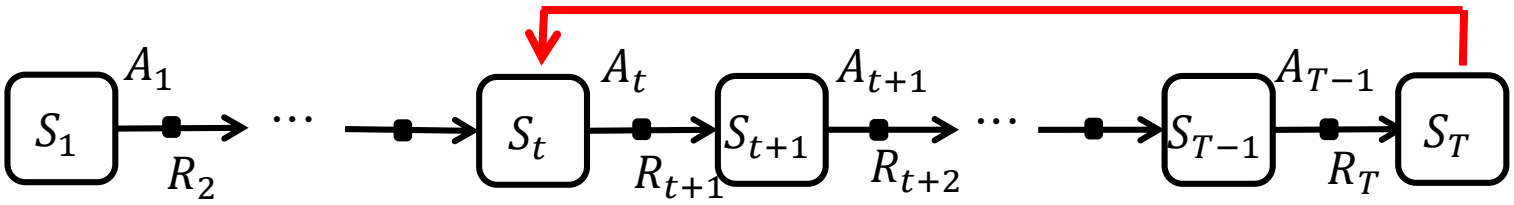




Monte Carlo Learning in Go

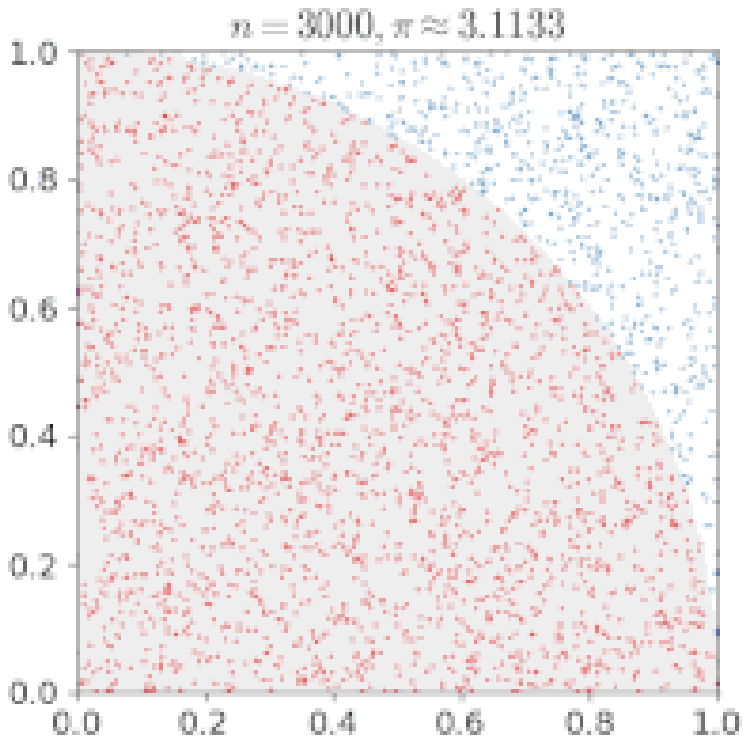
- How to evaluate this position?
  - Why not play randomly until the game end?





Monte Carlo Learning

- Calculate values based on stochastics
  - Example: calculate π.



Multi-Armed Bandit Problem (吃角子老虎問題)

- Assume that you have infinite plays
  - How to choose the one with the maximal average return?

Exploration vs. Exploitation

- Example for the exploration vs exploitation dilemma
  - **Exploration**: is a long-term process, with a risky, uncertain outcome.
  - **Exploitation**: by contrast is short-term, with immediate, relatively certain benefits

Deterministic Policy: UCB1

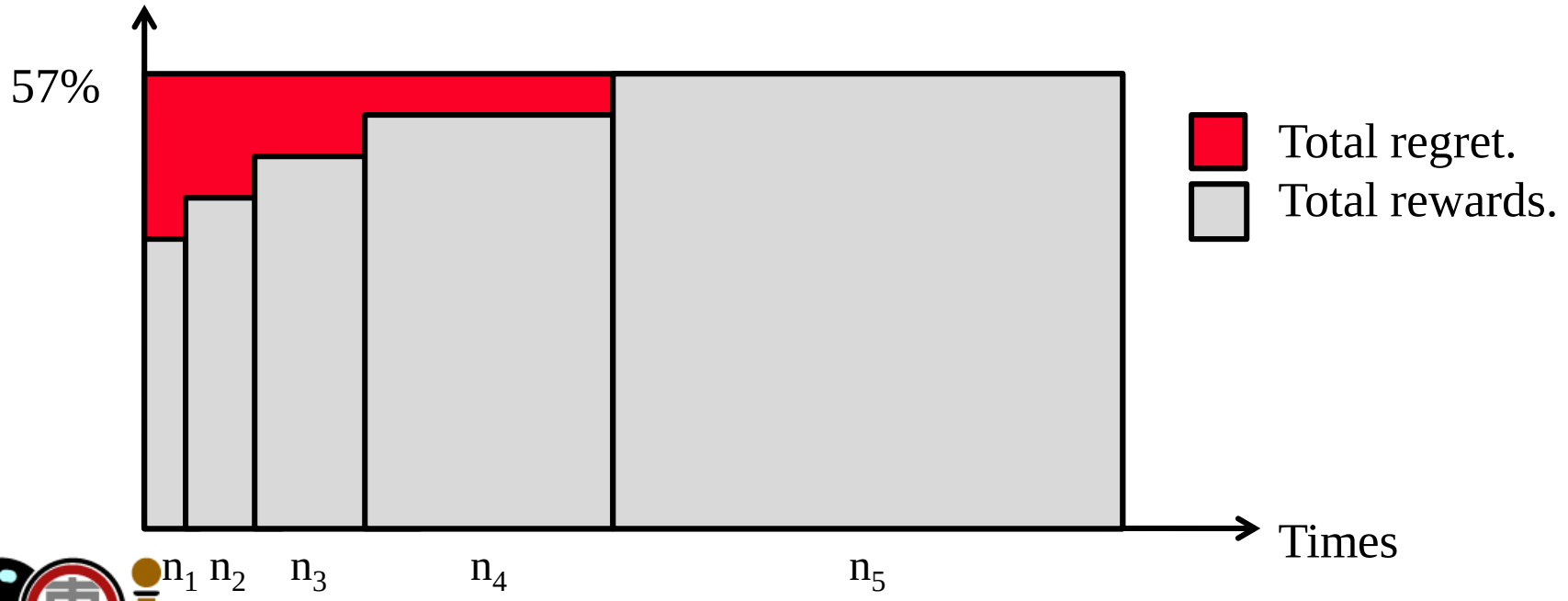
- UCB: Upper Confidence Bounds. [Auer *et al.*, 2002]
- Initialization: Play each machine once.
- Loop:
  - Play machine i that maximizes, (Exploitation + Exploration)

$$X_i + \sqrt{\frac{2 \log n}{n_i}}$$

  - where
    - $n = \sum_{i=1}^{k} n_i$ is the total number of playing trials.
    - n_i is the number of playing trials on machine i.
    - X_i is the (average) win rate on machine i.
- Key:
  - Ensure optimal machine is played exponentially more often than any other machine.

Cumulative Regret

- Assume Machines M_1, M_2, M_3, M_4, M_5
  - Win rates: 37%, 42%, 47%, 52%, 57%
  - Trial numbers: n_1, n_2, n_3, n_4, n_5.



Monte Carlo Tree Search

- A kind of planning
- A kind of Reinforcement learning

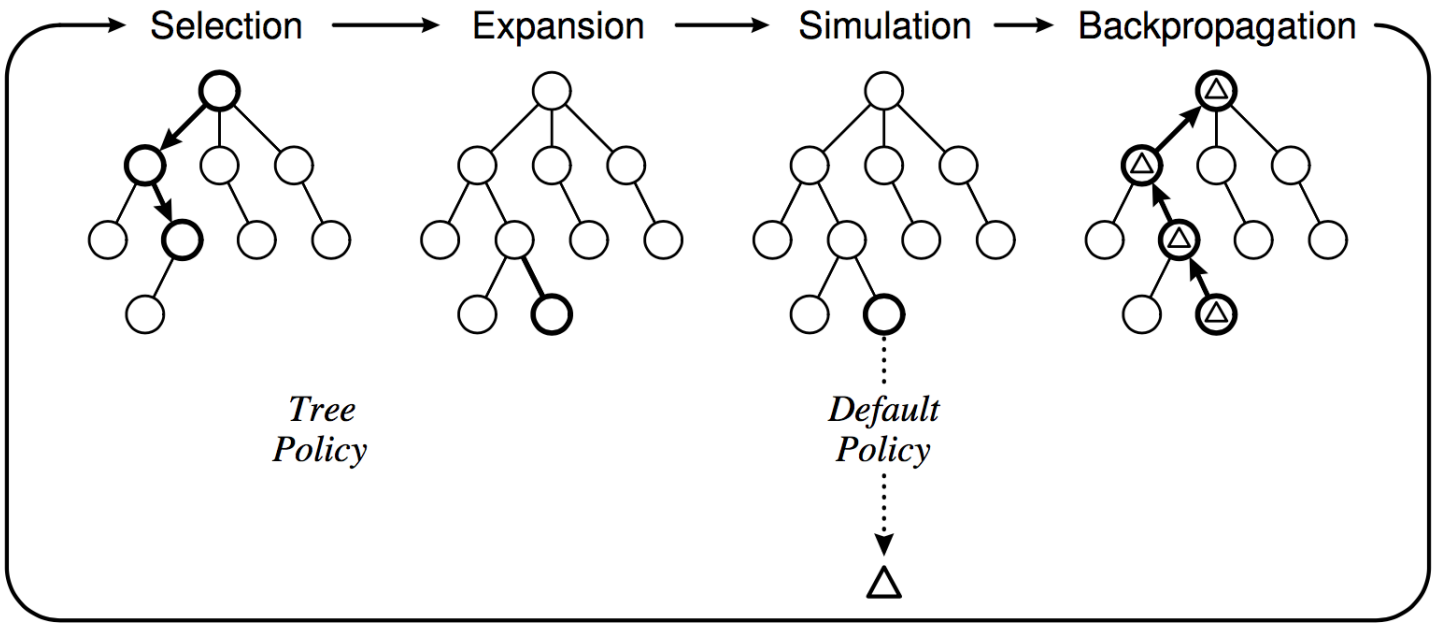
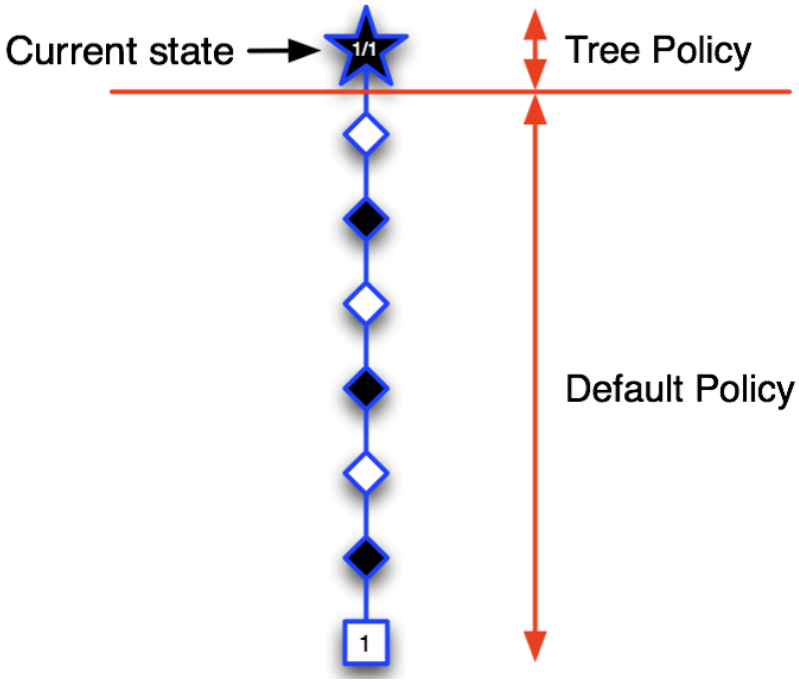
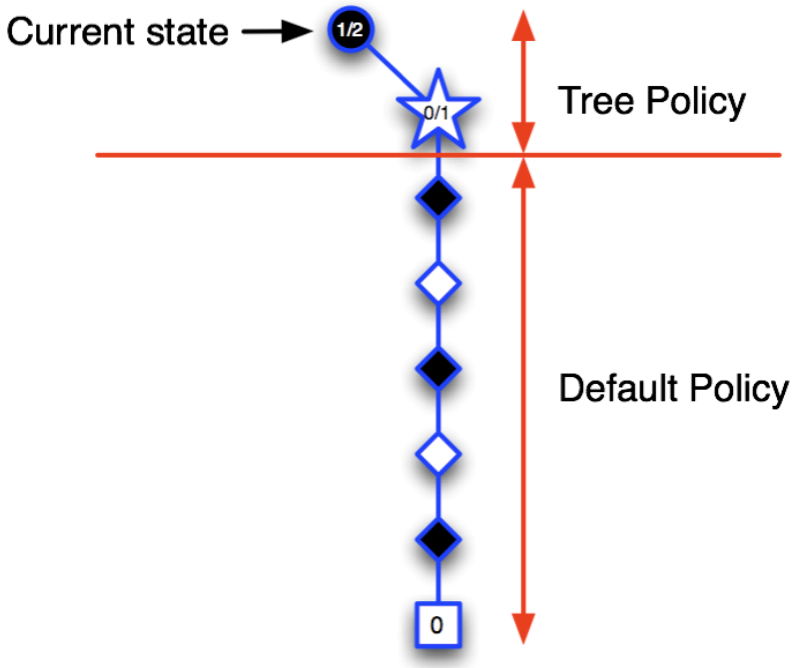
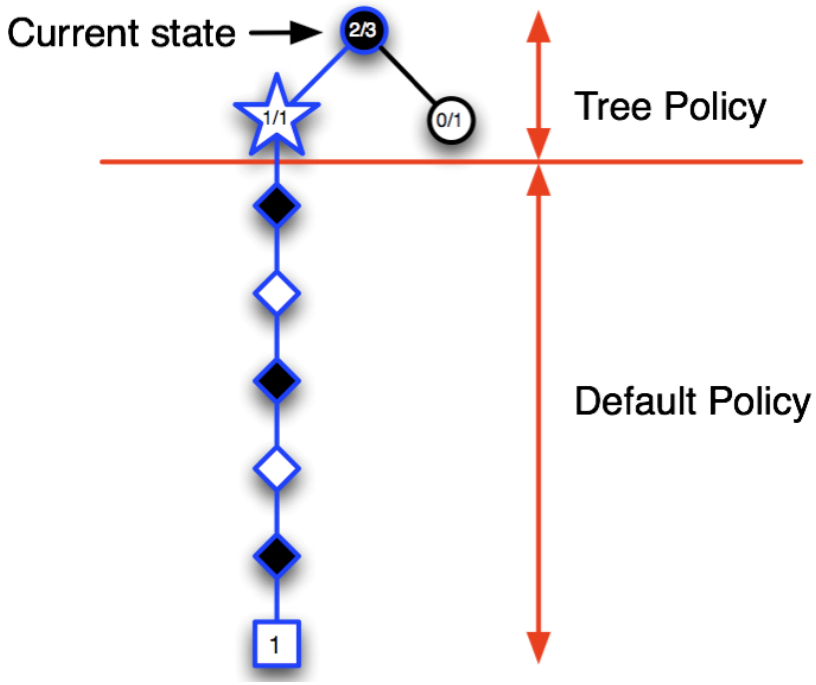


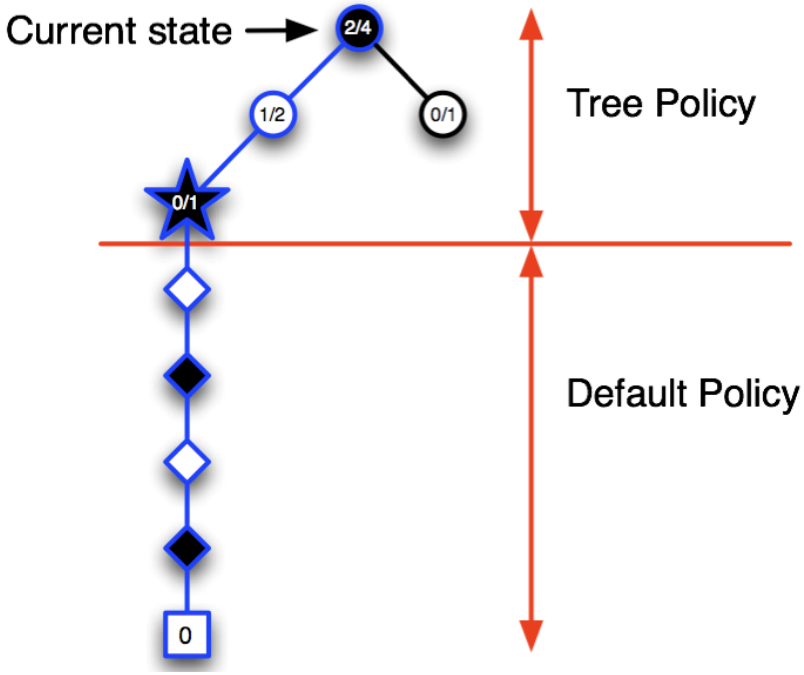
Fig. 2. One iteration of the general MCTS approach.

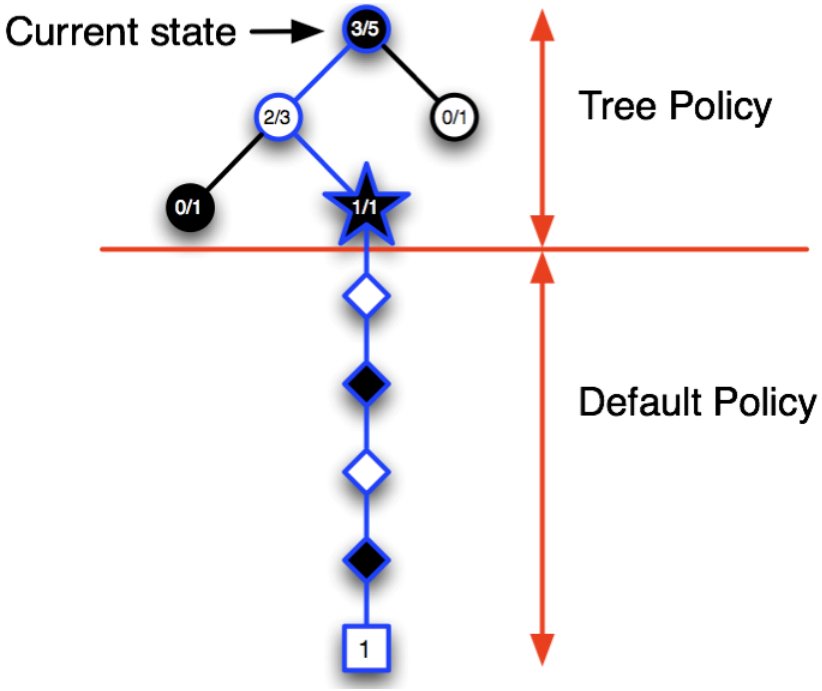
Image source: Browne et al., A Survey of Monte Carlo Tree Search Methods


Current state
1/1
Tree Policy
Default Policy
1

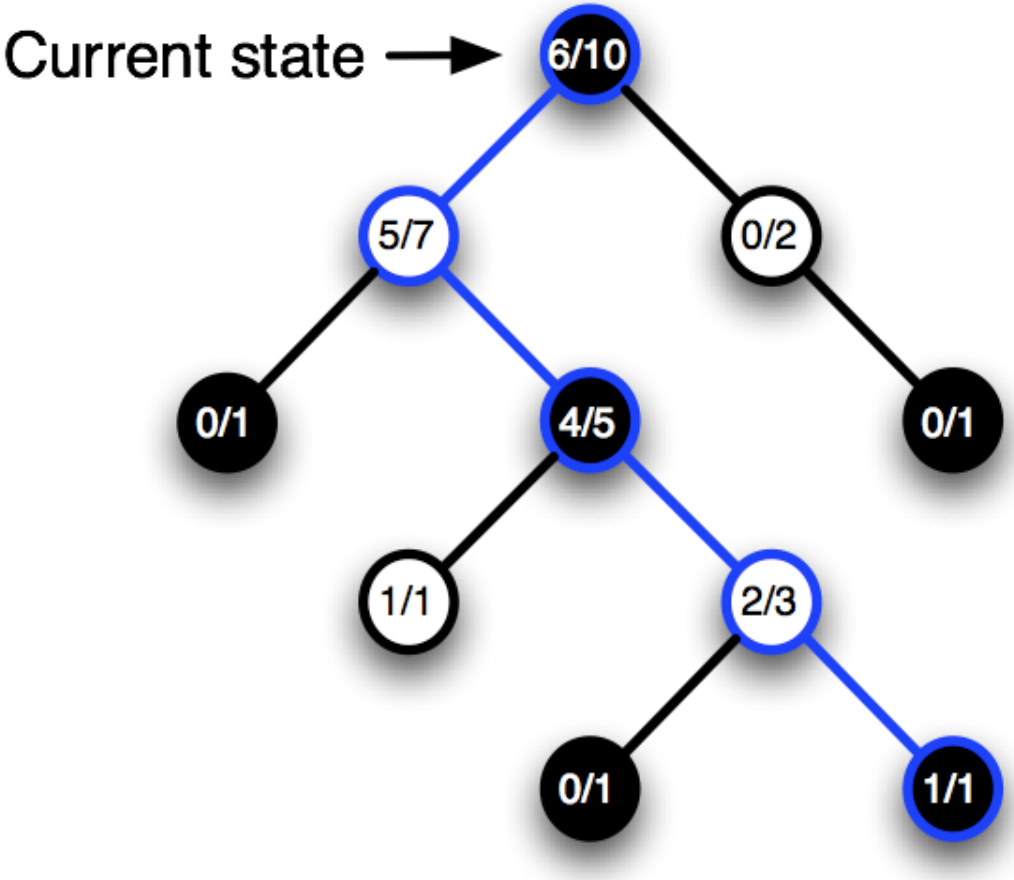

Current state
1/2
0/1
Tree Policy
Default Policy
0


Current state
2/3
1/1
0/1
Tree Policy
Default Policy
1

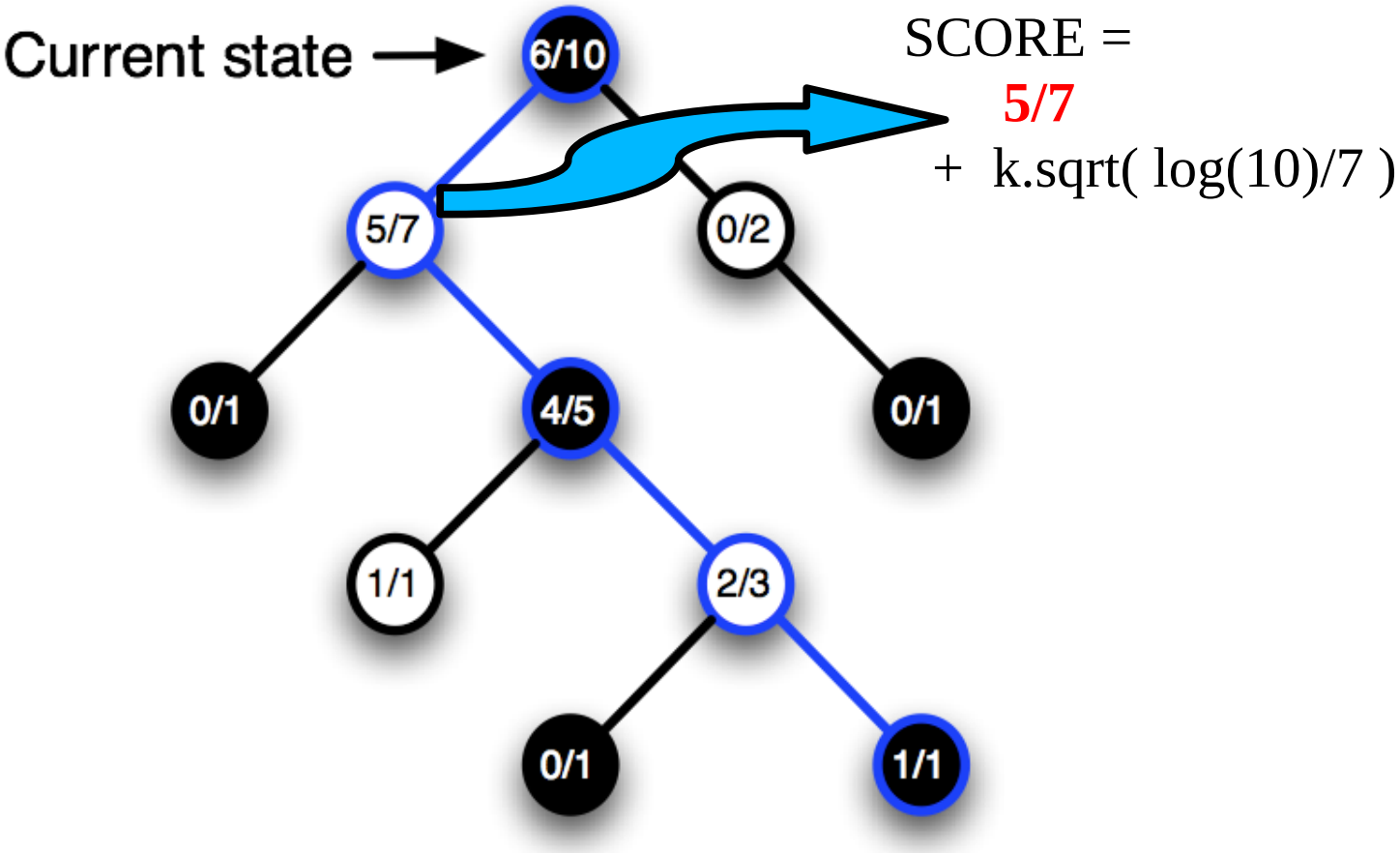

Current state
2/4
1/2
0/1
0/1
0
Tree Policy
Default Policy


Current state
3/5
2/3
0/1
0/1
1/1
1
Tree Policy
Default Policy

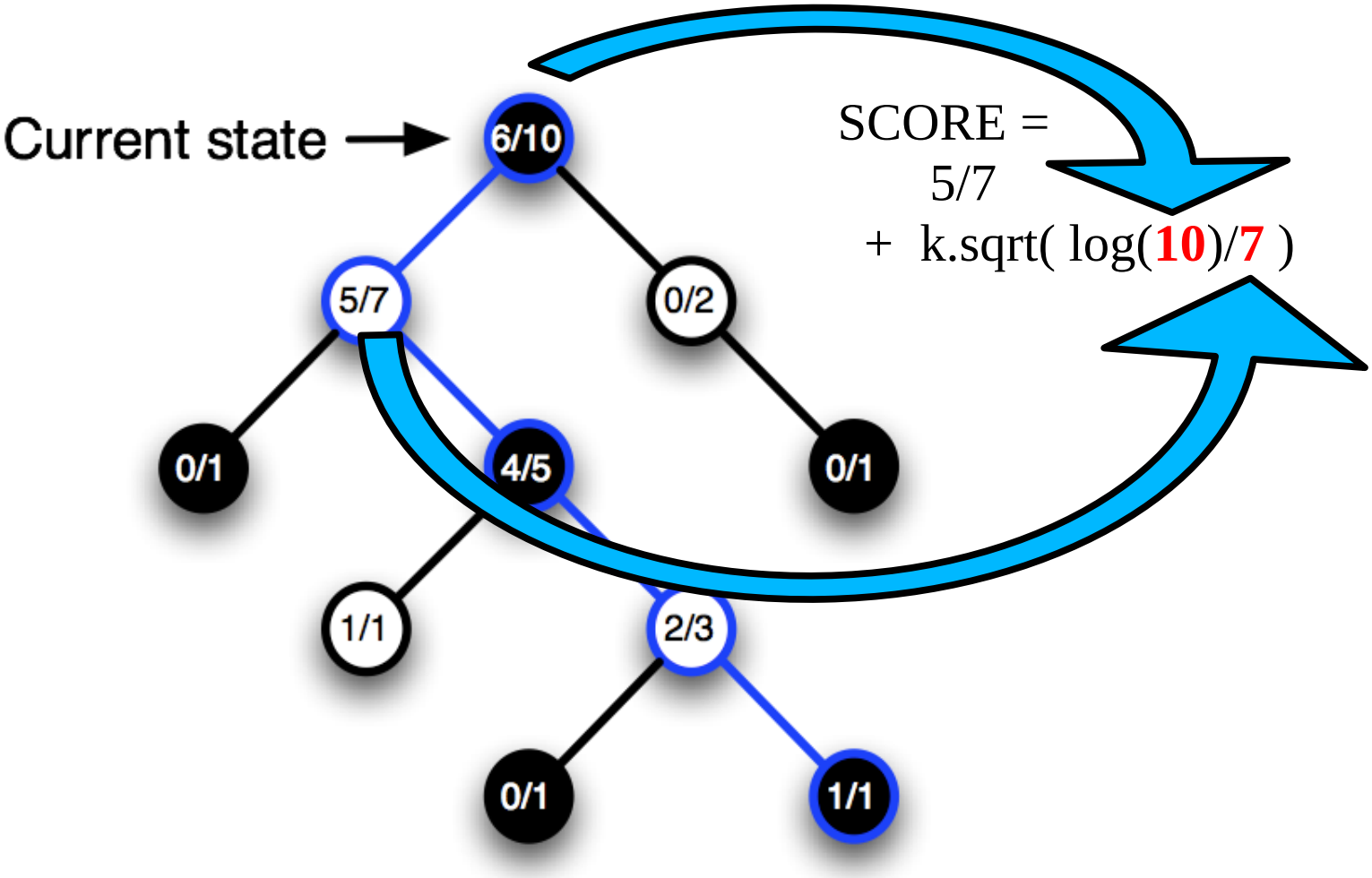
Exploitation and Exploration



Exploitation and Exploration



Exploitation and Exploration



AlphaGo

AlphaGo

- Use DCNN to learn experts' moves
  - (學習高手的著手策略)
- Use Monte-Carlo Tree Search (MCTS) for search to avoid pitfalls (避開陷阱)
  - MCTS is a kind of RL that do planning.
- Use DCNN to train "reinforcement learning (RL) network"

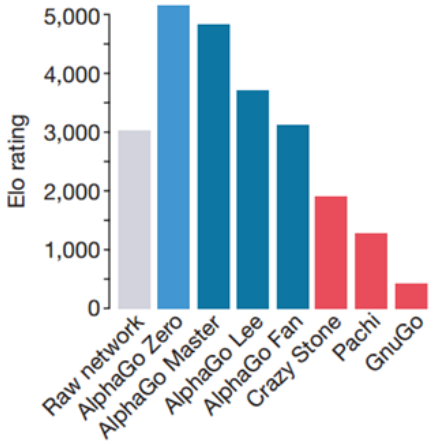
- Use DCNN to train "value network" (價值網路)
  - Learn the values of game positions (學習盤勢之優劣)

AlphaGo

- Use DCNN to learn experts' moves → DL
  - (學習高手的著手策略)
- Use Monte-Carlo Tree Search (MCTS) for search to avoid pitfalls (避開陷阱) → RL
  - MCTS is a kind of RL that do planning.
- Use DCNN to train "reinforcement learning (RL) network" → DRL
- Use DCNN to train "value network" (價值網路) → DL
  - Learn the values of game positions (學習盤勢之優劣)

Summary of the Different AlphaGo Versions

- Two nature papers:
  - Mastering the game of Go with deep neural networks and tree search
  - Mastering the game of Go without human knowledge

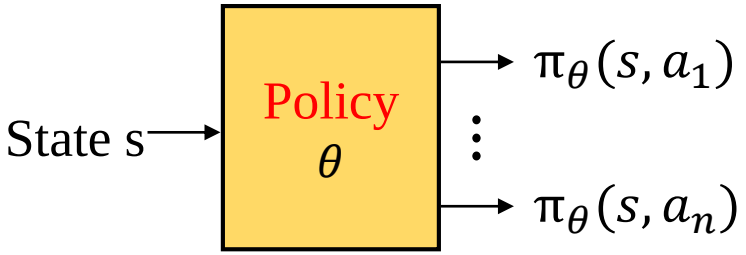


	AlphaGo Fan	AlphaGo Lee	AlphaGo Master	AlphaGo Zero
Elo rating	3,144	3,739	4,858	5,185
Tournament	5-0 Fan Hui October 2015	4-1 Lee Sedol March 2016	60-0 top players January 2017	89-11 AlphaGo Master
Machine	176 GPUs	48 TPUs	-	4 TPUs
Main Technique	First paper	First paper	Second paper	Second paper
Network	~ 12 layers x 192 filters	~ 12 layers x 256 filters	~ 40 layers x 256 filters	~ 80 layers x 256 filters
Human Knowledge	Yes	Yes	Yes	No

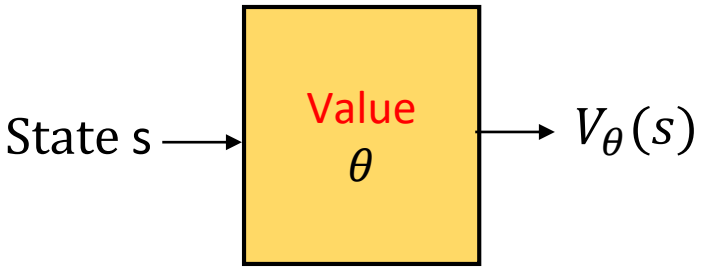
Policy Network and Value Network

- Policy Network
  - Use stochastic policy gradient ascent to maximize the likelihood of the human move a selected in state s

    $\Delta\theta = \alpha\nabla_\theta \log \pi_\theta(s_t, a_t) \cdot z$
    - θ: network parameter.
    - α: learning rate
    - z: the value of the episode
      - win/loss (1/-1) of the game

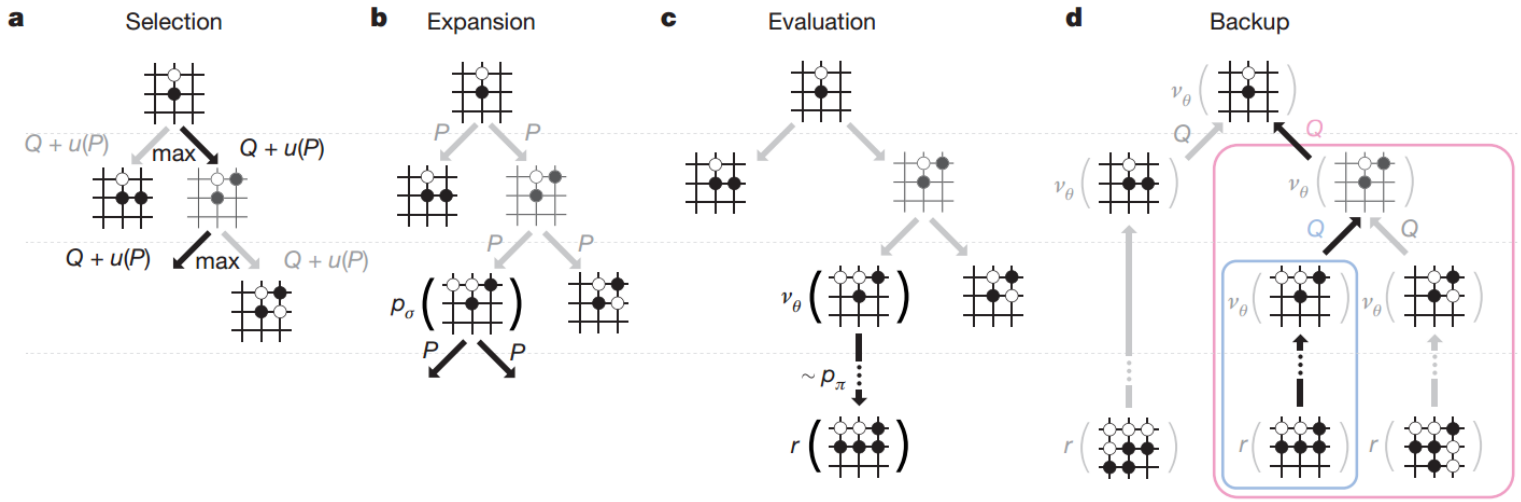


- Value Network
  - Goal: estimate a value (or a win rate) for any given game position
    - $v^*(s)$: perfect play optimal value function
    - $v^\rho(s)$: estimate value function by RL policy P_ρ
    - train a value network $v_\theta(s)$ with weights θ to approximate $v^\rho(s)$
      - $v_\theta(s) \approx v^\rho(s) \approx v^*(s)$
  - $\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$
    - z is outcome



Combine with MCTS

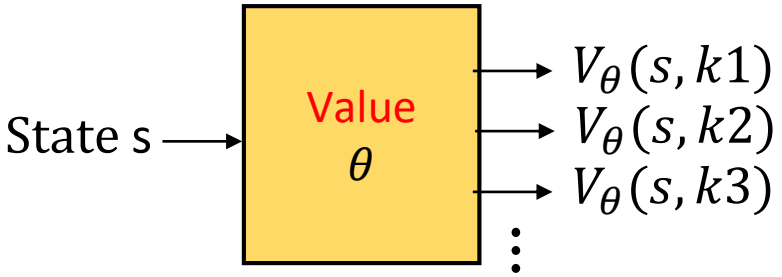
- Ppolicy network used for selection and expansion
- Mixed fast rollout policy and value network for evaluation



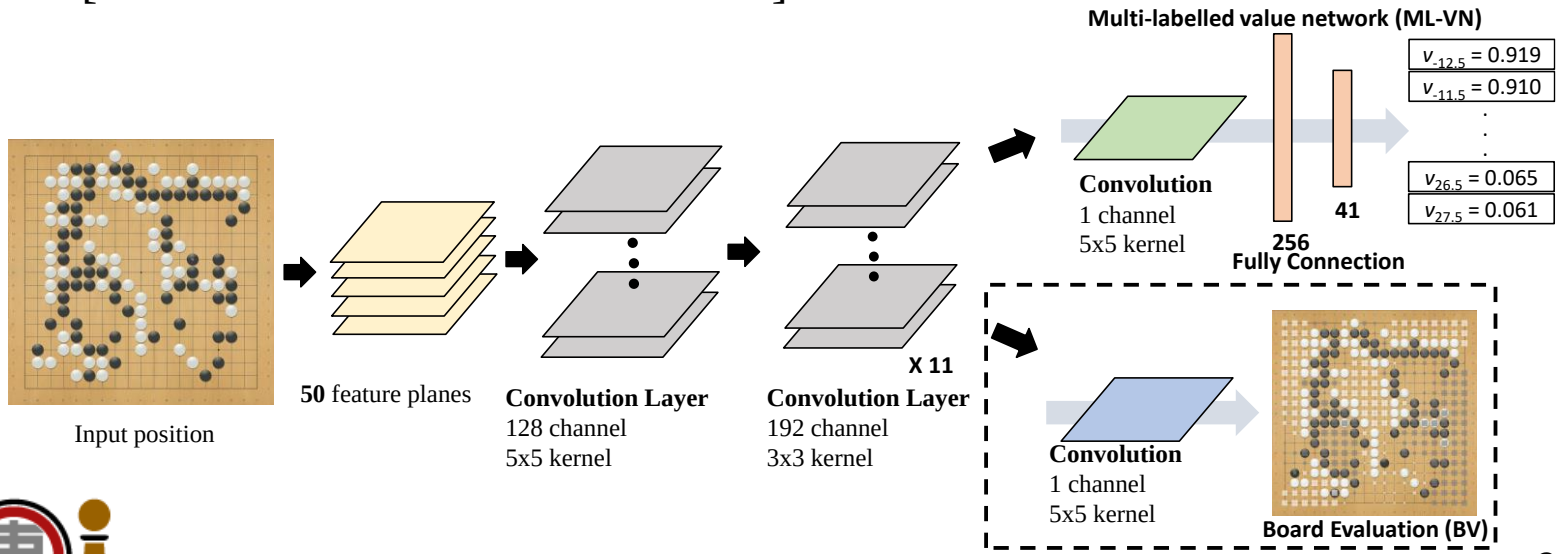
Appendix:
Multi-labelled (ML) Value Network

Multi-labelled (ML) Value Network

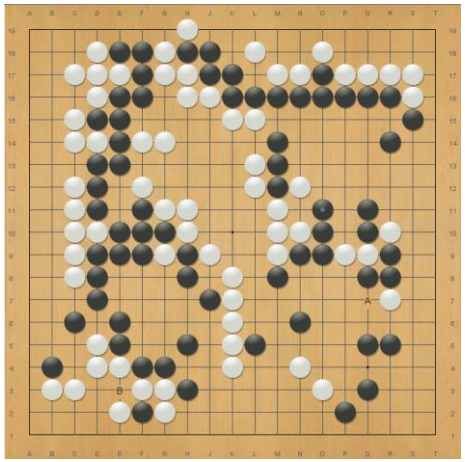
- Includes all value outputs v_k for k-komi games (貼k目)
  - Ideally, $v_{-361.5}$ to $v_{361.5}$
  - Only consider $v_{-12.5}$ to $v_{27.5}$ for simplify



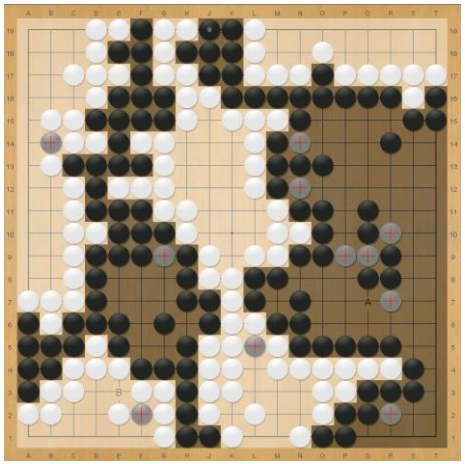
[IEEE Transactions on Games 2018]



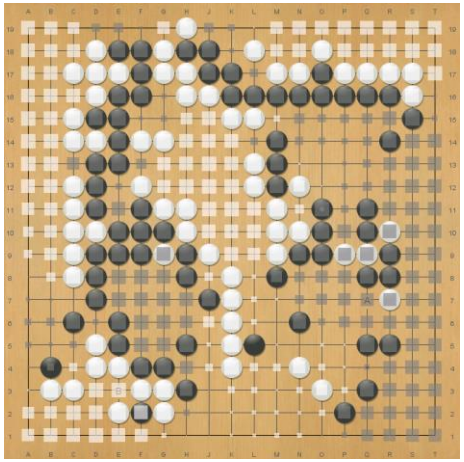
ML Value Network with Board Evaluation



Current positions
(Training data)



Final positions



Output of Board Evaluation

Territory Difference is about 3

k (komi)	Label on v_k	v_k (win rate)
0.5	1	0.678897
1.5	1	0.599618
2.5	1	0.599108
3.5	-1	0.512413
4.5	-1	0.511263
5.5	-1	0.423886
6.5	-1	0.423626
7.5	-1	0.339738
8.5	-1	0.339353

ML Value Networks for Handicap Games

- For handicap games (讓子棋)
  - Ours based on ML value network clearly outperforms most of other methods.

(Currently used in many Go programs)

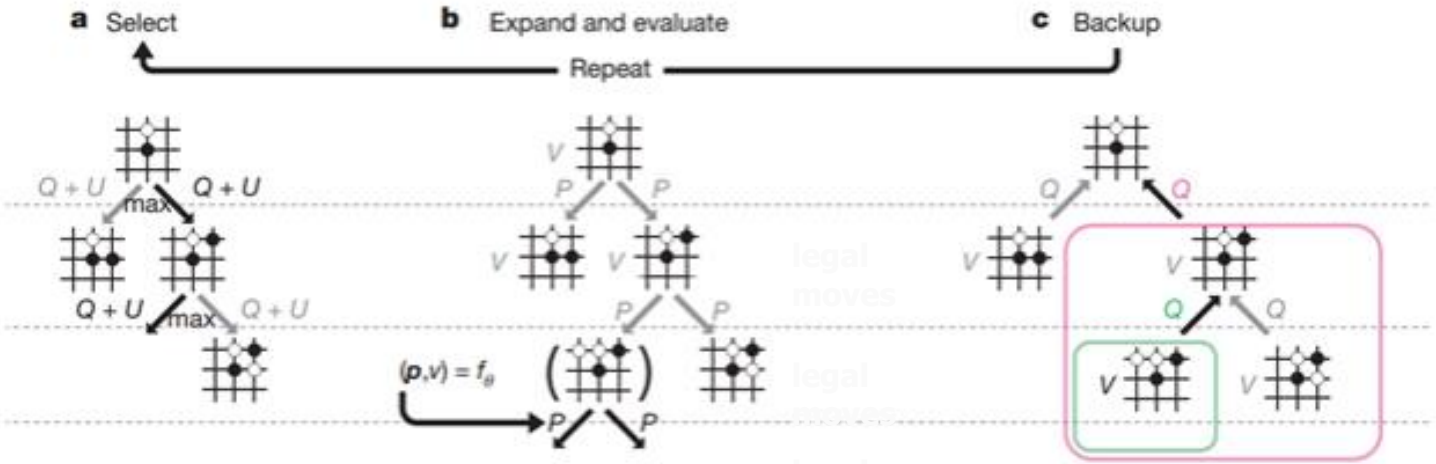
Method	Even Game	H1	H2	H3	H4
No dynamic komi	80.80%	74.00%	50.00%	30.40%	10.80%
SS-R	80.00%	76.00%	58.00%	38.00%	22.00%
SS-B	78.00%	77.60%	54.00%	31.60%	20.40%
SS-M	79.60%	74.40%	49.20%	34.00%	12.80%
VS-M	82.00%	71.60%	46.80%	30.40%	9.20%
ML-DK	**83.20%**	**80.00%**	**57.20%**	**41.60%**	**18.40%**

AlphaZero

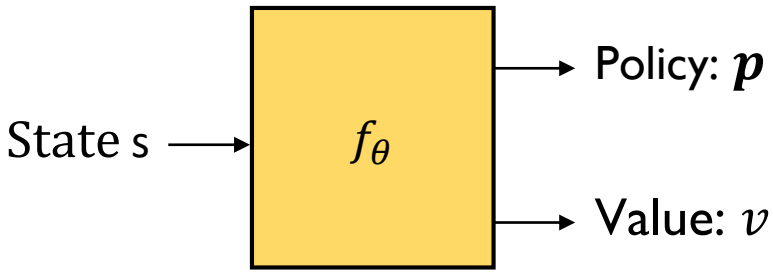
AlphaZero

[1] David Silver, et al., 2017. Mastering the game of Go without human knowledge. Nature 550(7676):354.

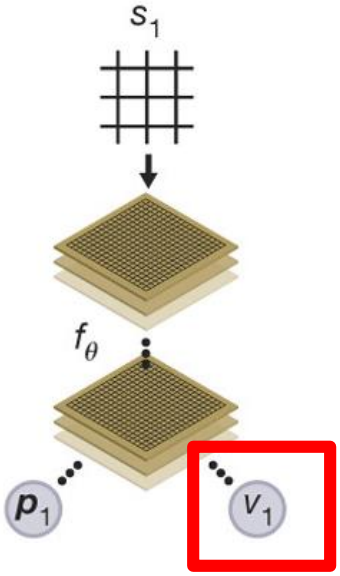
- Uses a deep neural network as an evaluation function during MCTS.



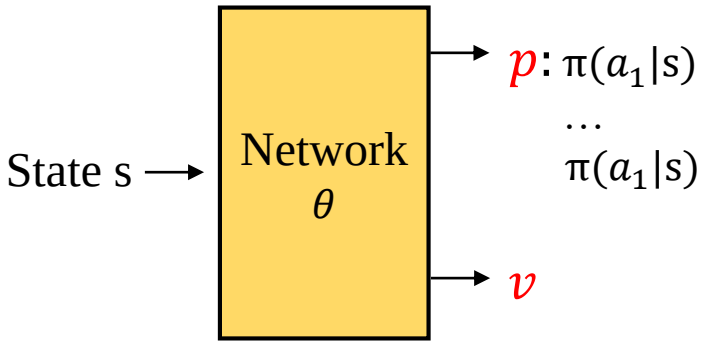
$$(\boldsymbol{p}, v) = f_\theta(s)$$



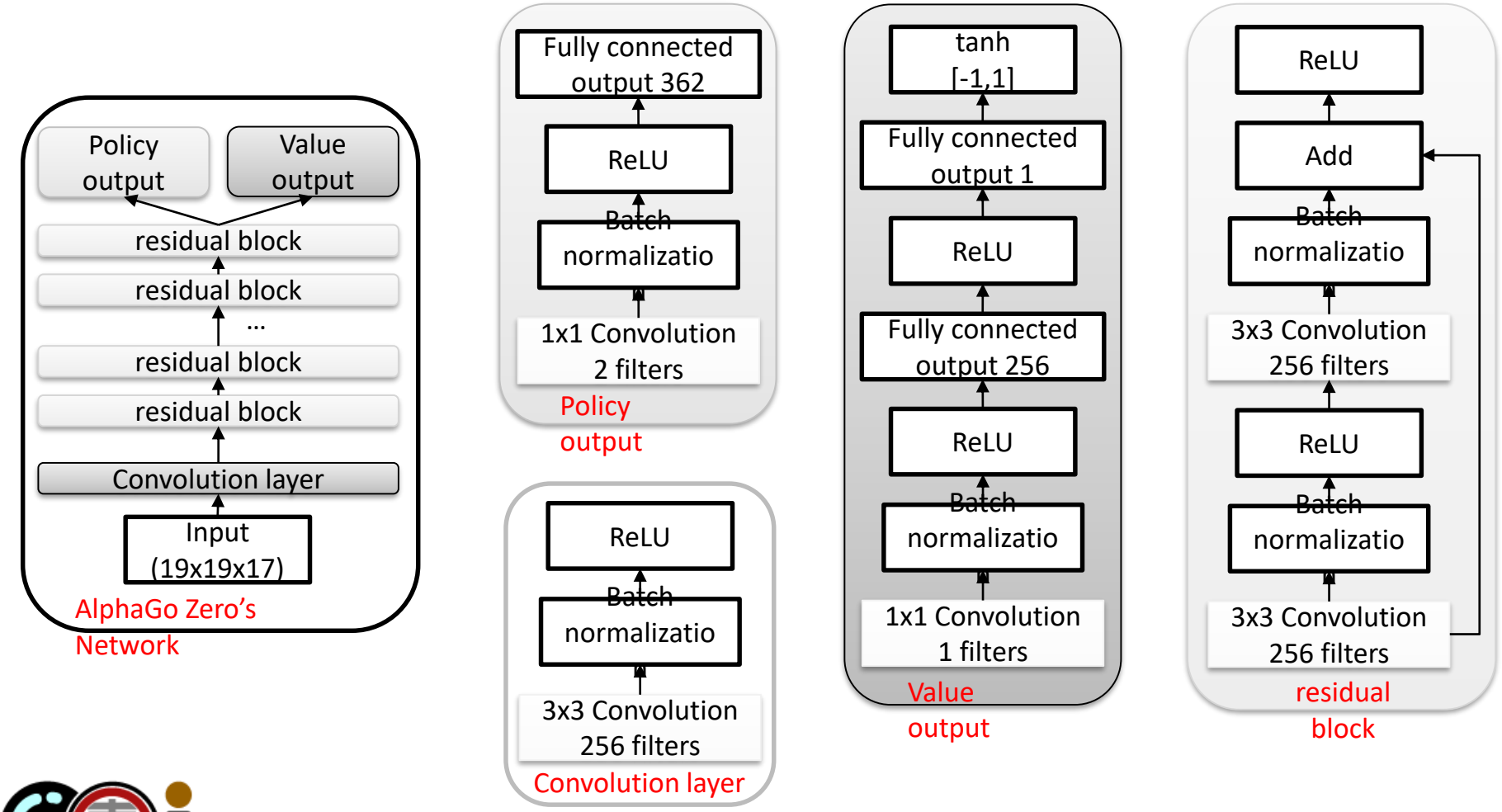
AlphaGo Zero



- Deep neural network $f_\theta(s) = (p, v)$
  - θ: network parameters
  - s: board representation of the position and its history
  - p: vector of move probabilities
  - v: scalar evaluation
- Algorithm
  - Self-Play
  - Optimization
  - Evaluator



Networks


Policy output
Value output
residual block
residual block
...
residual block
residual block
Convolution layer
Input (19x19x17)
AlphaGo Zero's Network
Fully connected output 362
ReLU
Batch normalizatio
1x1 Convolution 2 filters
Policy output
ReLU
Batch normalizatio
3x3 Convolution 256 filters
Convolution layer
tanh [-1,1]
Fully connected output 1
ReLU
Fully connected output 256
ReLU
Batch normalizatio
1x1 Convolution 1 filters
Value output
ReLU
Add
Batch normalizatio
3x3 Convolution 256 filters
ReLU
Batch normalizatio
3x3 Convolution 256 filters
residual block

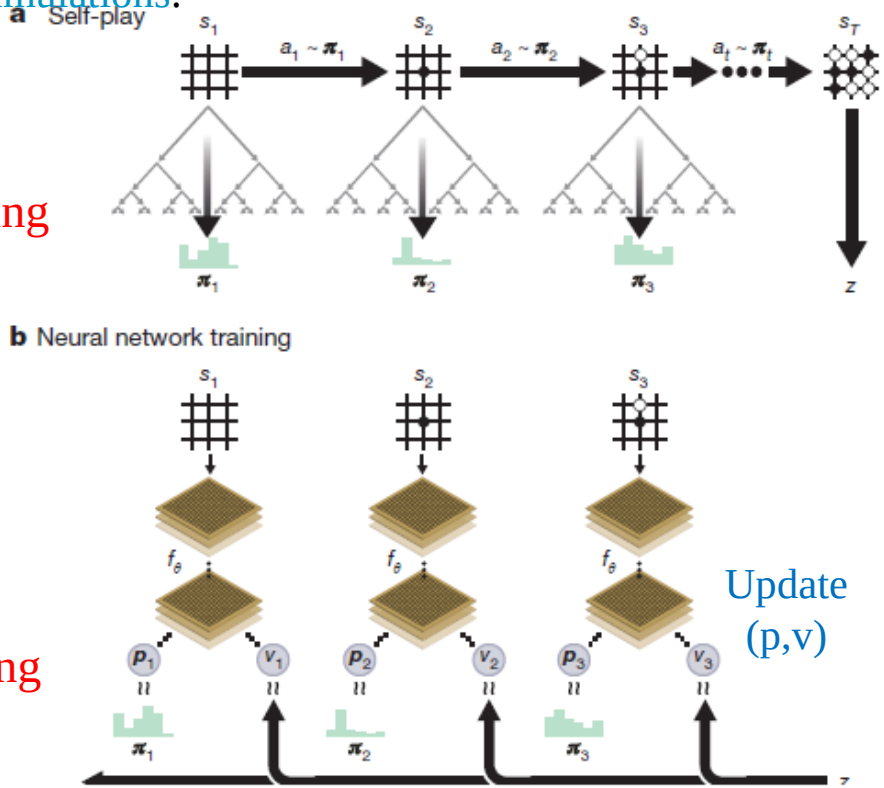
AlphaZero

A general approach

- Combine "value/policy networks" → DRL
- Use self-play to generate higher quality games by using MCTS → RL
  - For each MCTS move, it requires 800 simulations.

Like a tutor

Learning improves searching



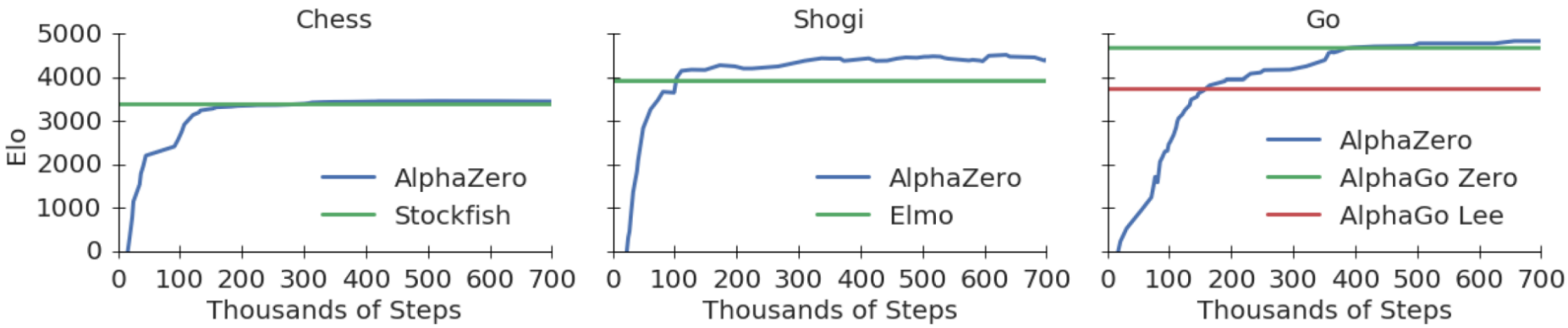
Learn from Zero Knowledge!!!

Like a student

Searching improves learning

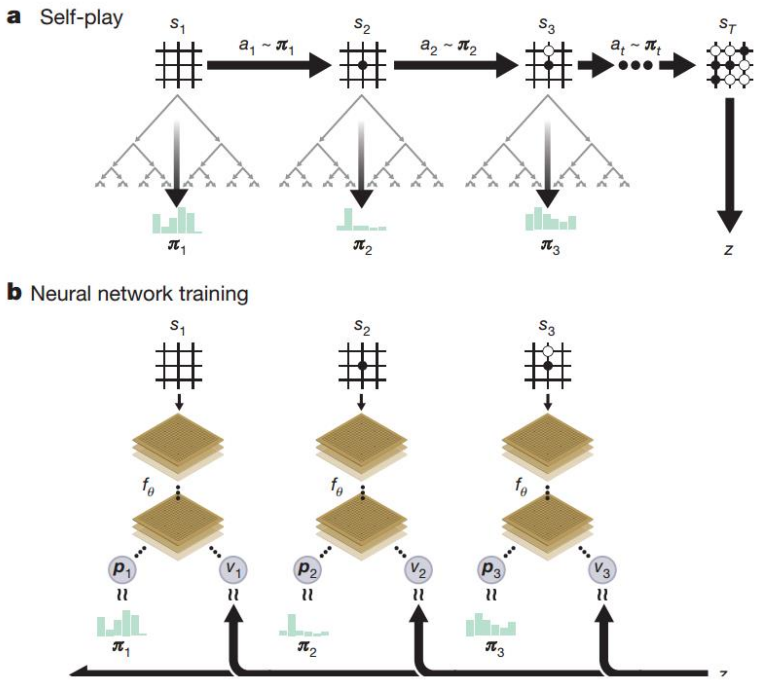
AlphaZero Performances

- State-of-the-art for many games, e.g., Go, Chess, Shogi.



AlphaGo Zero's Training Step

- Self-Play
- Optimization
- Evaluator

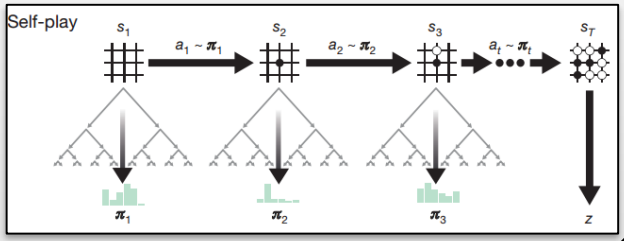


Self-Play Game Collections

Self-Play step2

Self-Play

1. Ask a new f_{θ_*} from evaluator
2. Play *25,000* games of self-play



Self-Play step1

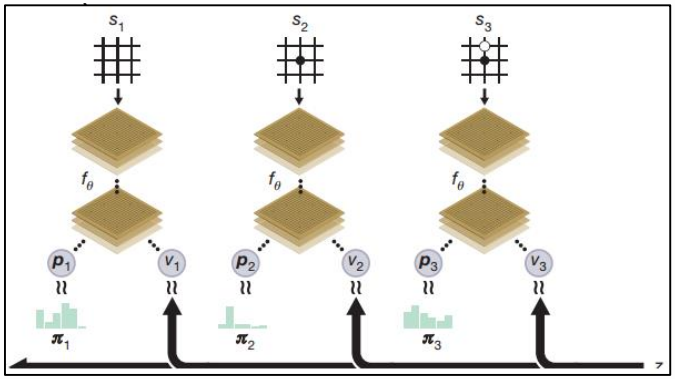
Evaluator

1. Evaluate playing strength between f_{θ_*} and f_{θ_i} (*400* games)
2. Update best network $f_{\theta_*} = f_{\theta_i}$ if f_{θ_i} *wins* $> 55\%$ to f_{θ_*}

Optimization step1

Optimization

1. Sample a mini-batch size of *2048* positions from recent *500,000* games
2. Optimize f_θ (Produces f_{θ_i} every *1,000* training



Optimization step2

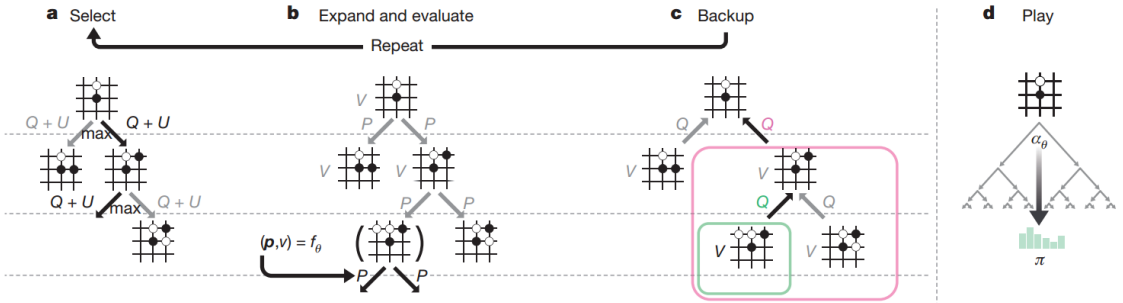
Network f_θ checkpoint Collections

Evaluator step1

Pipeline Training of AlphaGo Zero

Self-Play

- In each position s, an MCTS search is executed, guided by f_θ
  - MCTS search outputs probabilities π of playing each move
- Once the search is complete, search probabilities π are returned, proportional to $N^{1/\tau}$
  - N is the visit count of each move from the root state
  - τ is a parameter controlling temperature
    - $\tau = 1$ in first 30 moves, $\tau \to 0$ for the remainder of the game



Optimization

- When the game terminates at step T, the game is then scored to give a final reward of $r_T \in \{-1, +1\}$
  - both pass, resign, exceeds a maximum length
- Parameters θ are adjusted by gradient descent on a loss function l that sums over the mean squared error and cross-entropy losses
  - $(p, v) = f_\theta(s)\ and\ l = (z - v)^2 - \pi^T log p + c\|\theta\|^2$
  - c: L2 weight regularization (prevent overfitting)
- Produces a new checkpoint every n training steps

Evaluator

- Goal: To ensure always generate the best quality data
- Evaluate playing strength between f_{θ_*} and f_{θ_i}
  - $f_{\theta_*} = \begin{cases} f_{\theta_i}, & if\ f_{\theta_i}\ wins > 55\%\ to\ f_{\theta_*} \\ f_{\theta_*}, & otherwise \end{cases}$
  - f_{θ_*}: current bet network
  - f_{θ_i}: network checkpoint by optimization
- The new f_{θ_*} will use in self-play generation

Experiment settings

- Common settings:
  - batch size: 2048 positions
  - MCTS search: 1600 simulations/move (approximately 0.4s/move)
  - optimization: 64 GPU workers and 19 CPU parameter servers
- AlphaGo Master
  - based on the algorithm and architecture presented in this paper
  - supervised learning from human data
  - using same handcrafted features and rollout as AlphaGo Lee

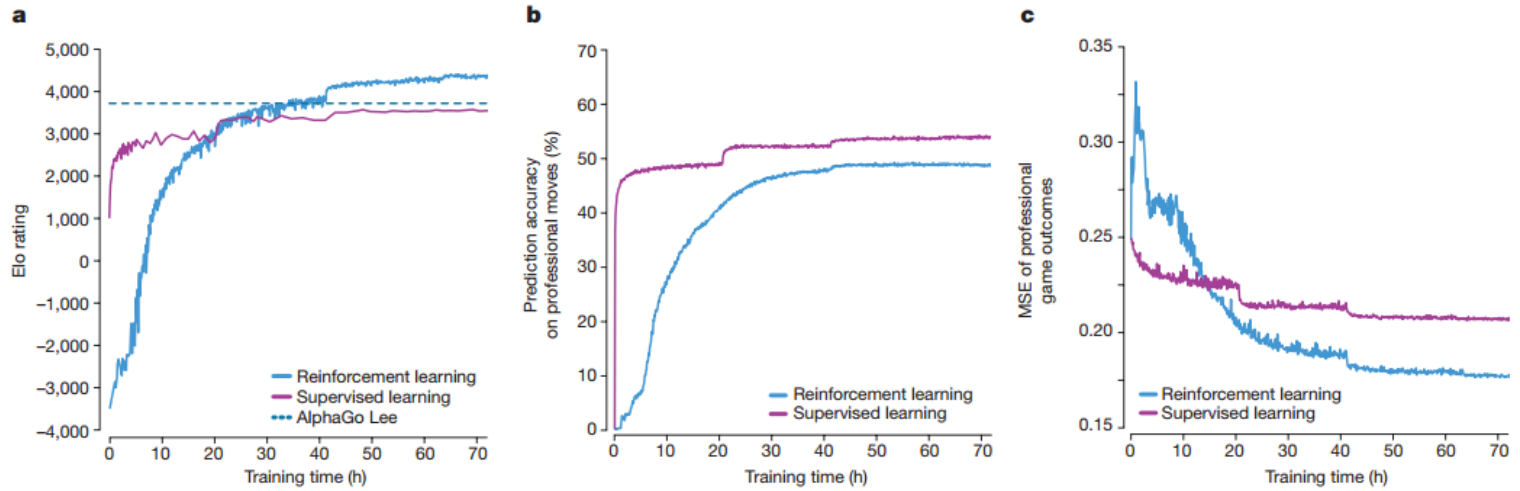
AlphaGo Zero

- Starting from random initial weights, without using rollout and human knowledge
- Input features $S_t = [X_t, Y_t, X_{t-1}, Y_{t-1}, \dots, X_{t-7}, Y_{t-7}, C]$
  - X_t indicating the presence of current player's stone (8 features planes)
    - $X_t^i = 1$ if intersection i contains a stone of the player's color at time-step t
  - Y_t indicating the presence of opponent player's stone (8 features
  - _

AlphaGo Zero	20 Block	40 Block
Total Self-play games	4.9 million	29 million
Total update iterations	0.7 million	3.1 million
Training days	~ 3 days	~ 40 days

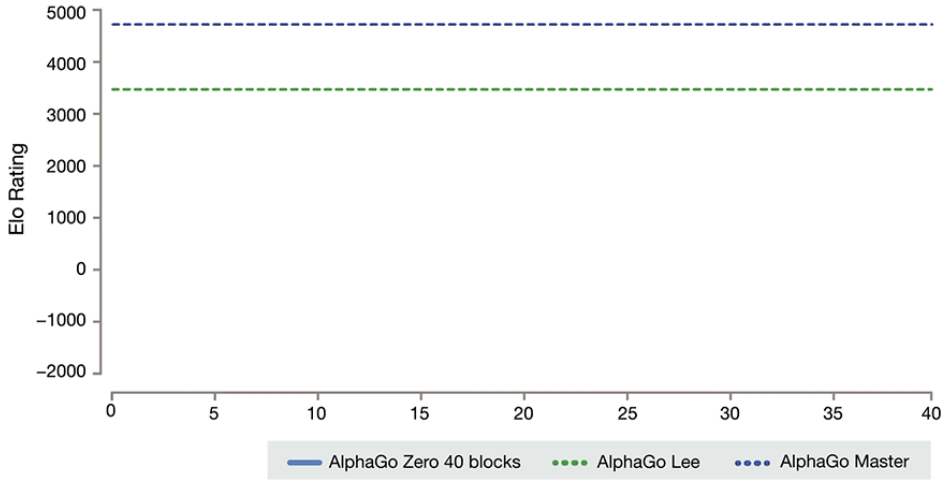
Empirical Evaluation of AlphaGo Zero (20 Block)

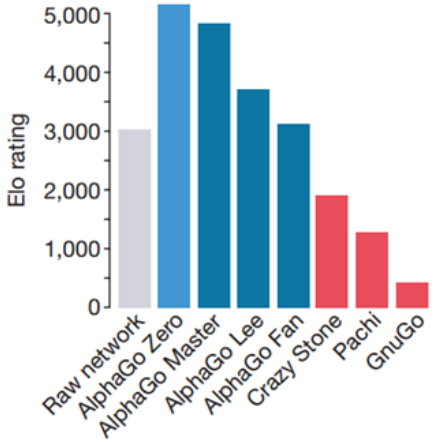
- Training approximately 3 days, 4.9 million games of self-play were generated



Performance of AlphaGo Zero (40 block)

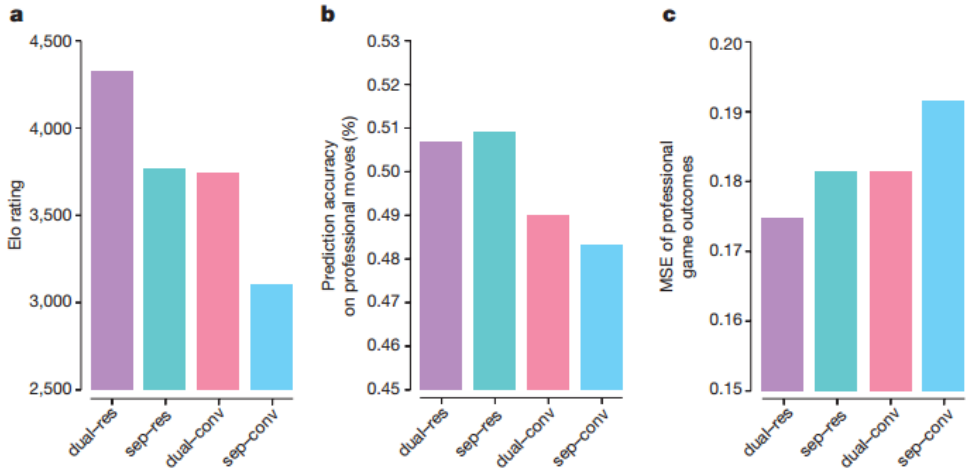
- Training approximately 40 days, 29 million games of self-play were generated





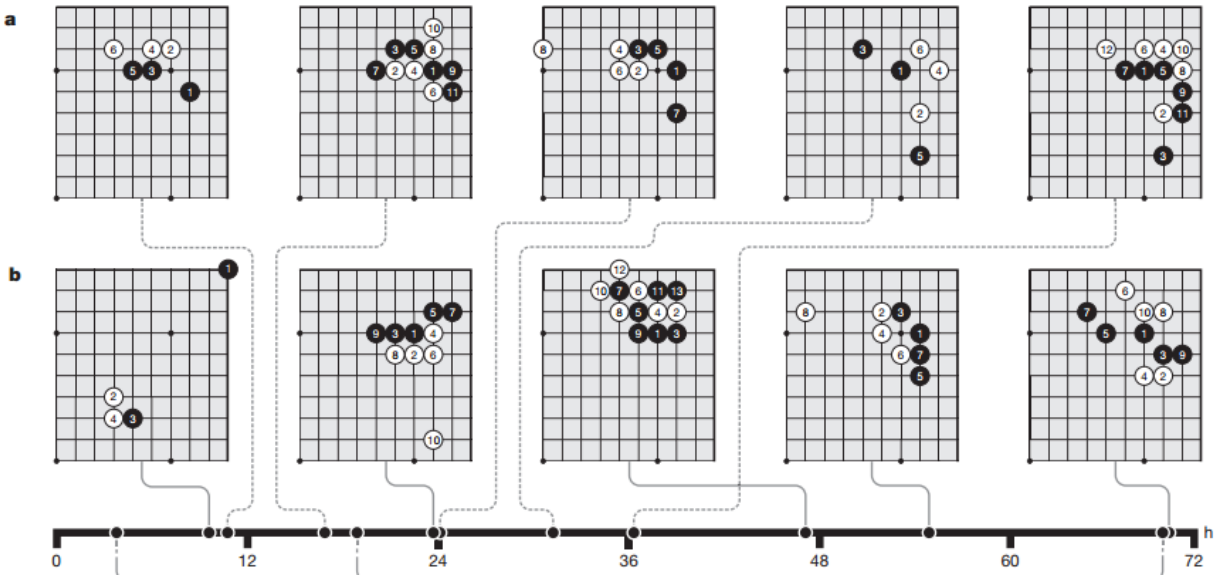
Comparison of Neural Network Architectures

- Improve 600 Elo by combining policy and value together into a single network
  - slightly reduced the move prediction accuracy
  - reduced the value error

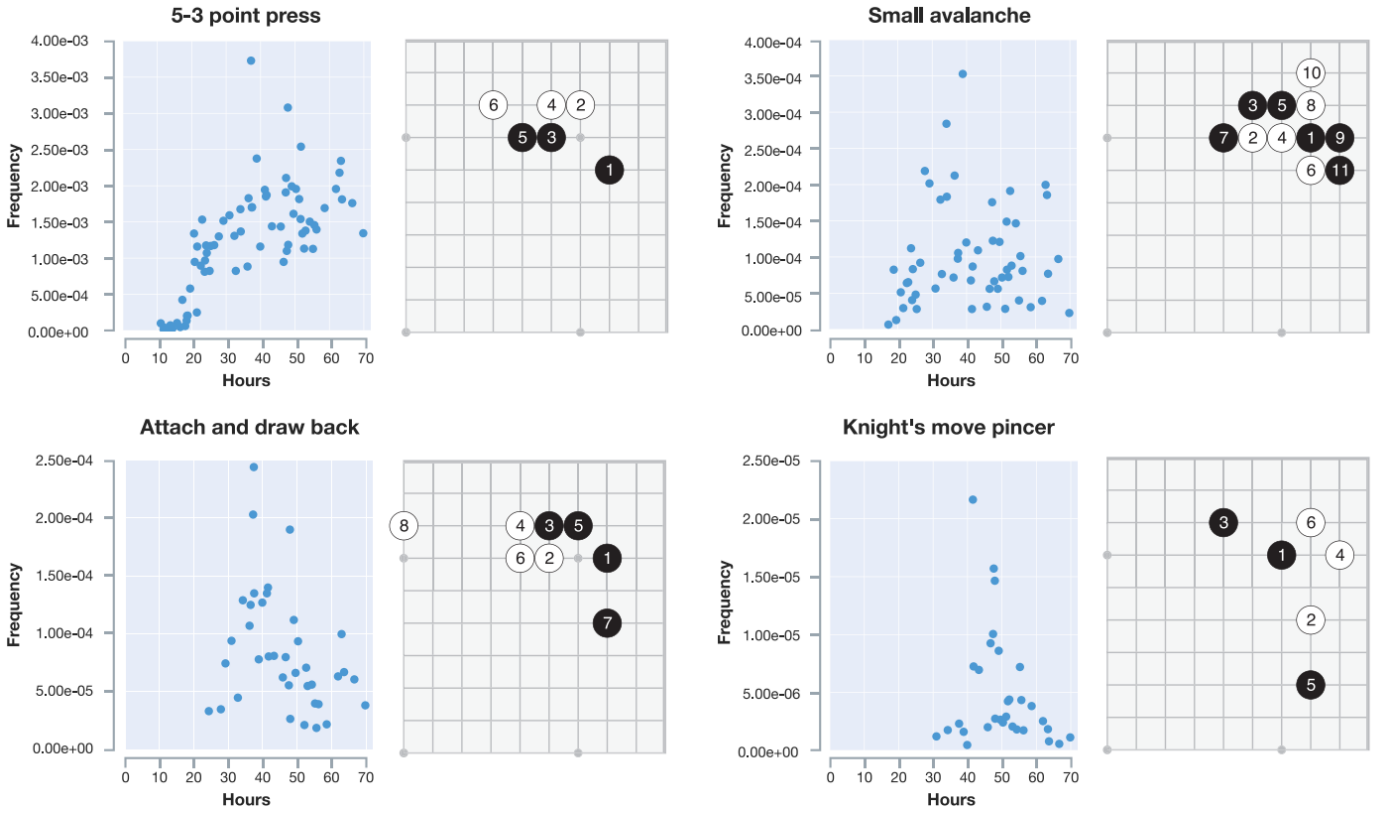


Go Knowledge Learned by AlphaGo Zero

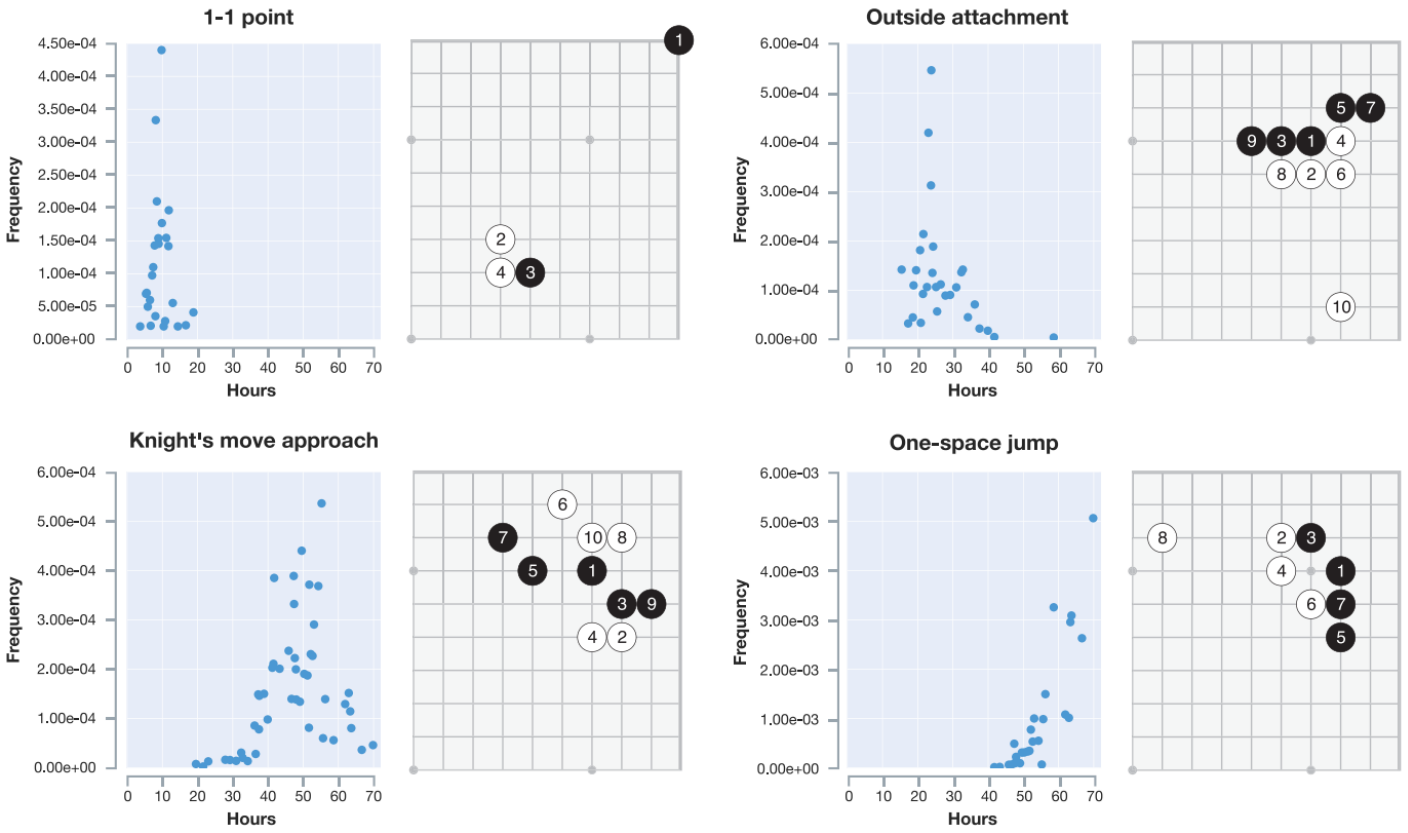
- *shicho* (ladder, 征子) were only understood by AlphaGo Zero much later in training



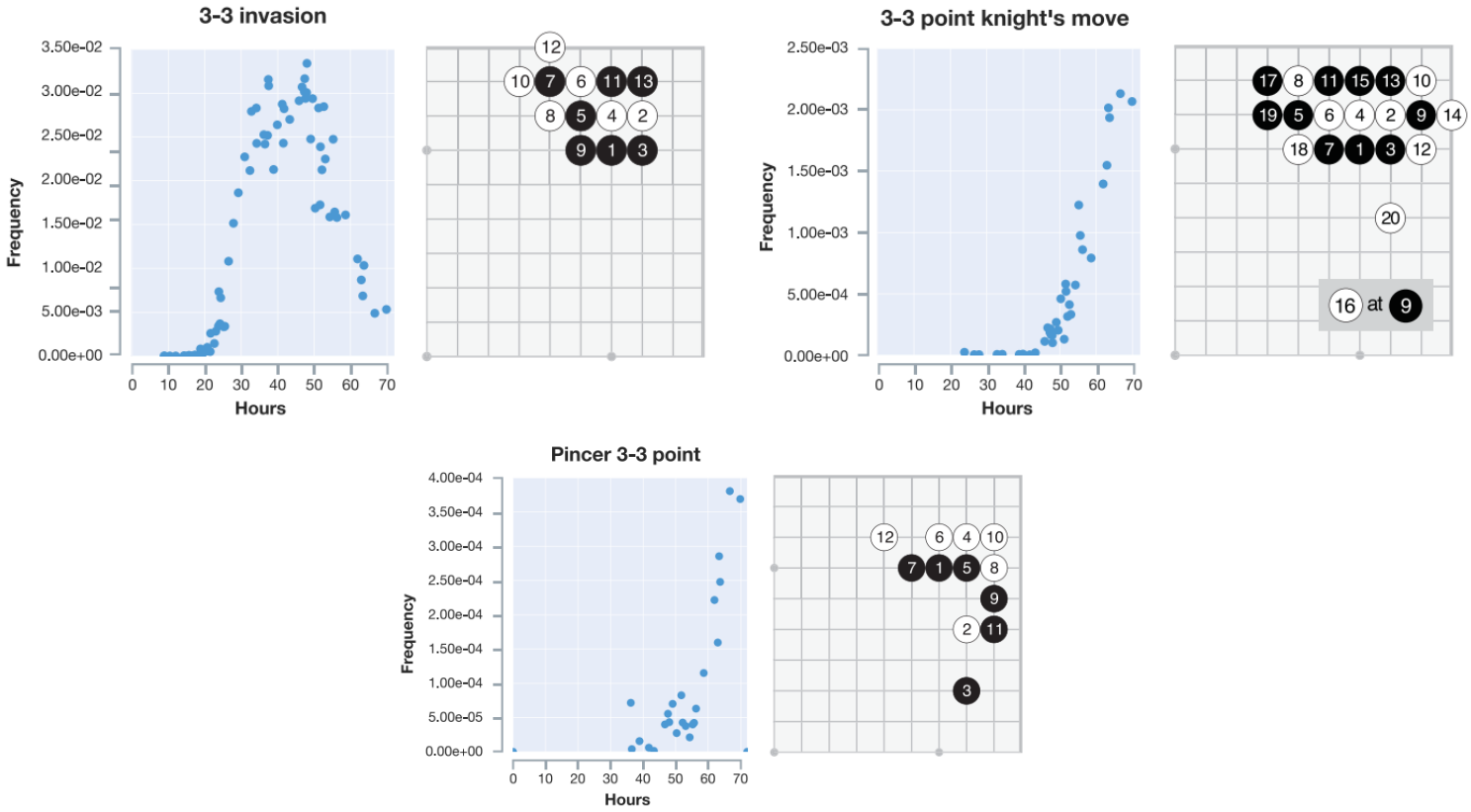
Frequency of *Joseki* (定式) during training



Frequency of *Joseki* (定式) during training



Frequency of *Joseki* (定式) during training

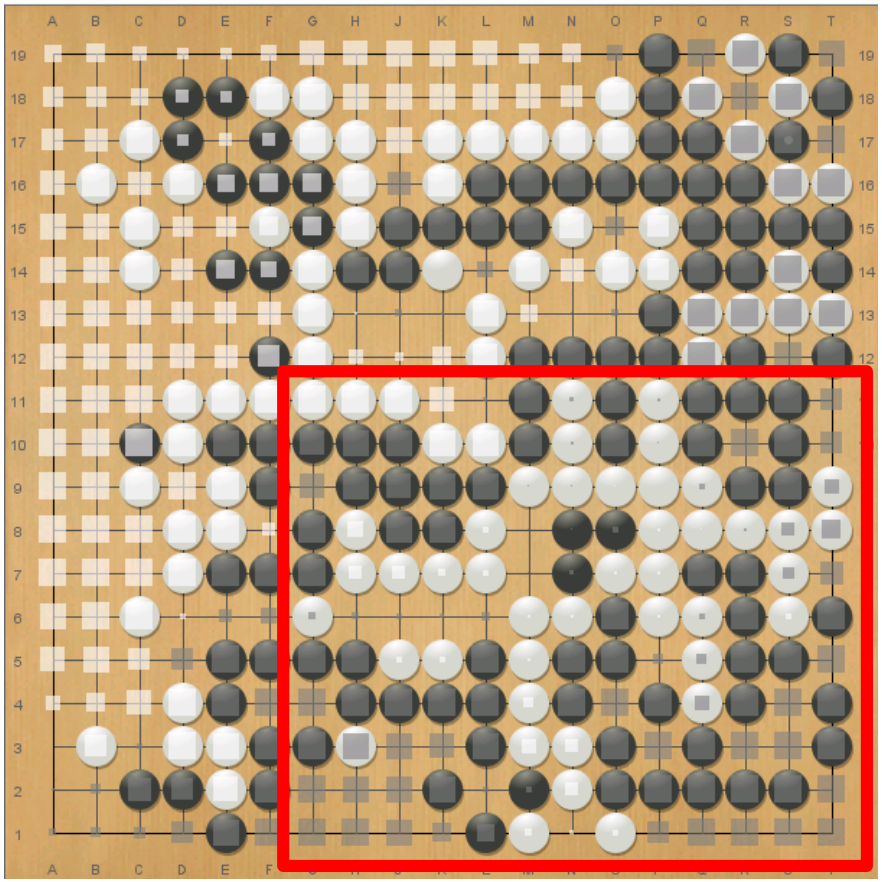


Any Weakness of AlphaZero?

- Known problems:
  - Usually weak for long groups
  - Still wrong for some ladder problems.
  - Wrong in most circular problems. (Found recently)
- Solutions: by Transformers?

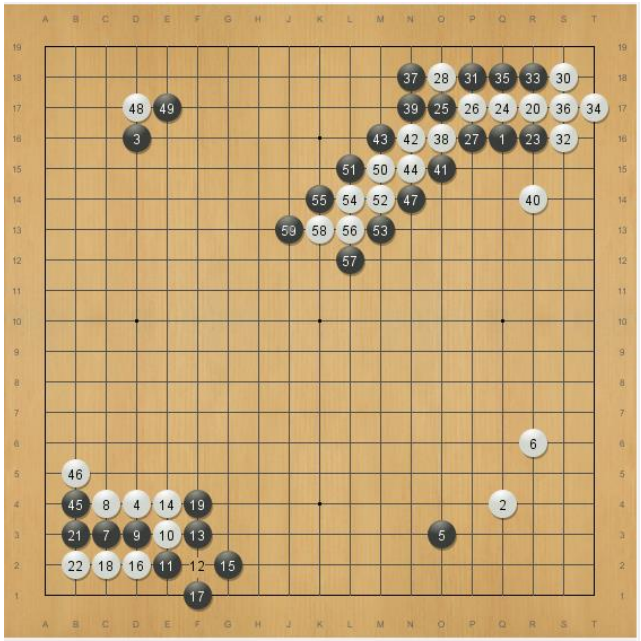
Long Group Problem:
[Flyhigher]vs[絕藝] 2017/05/18

- Lower-middle white group is not alive.
  - But, the top program fineart (絕藝) could not read it.

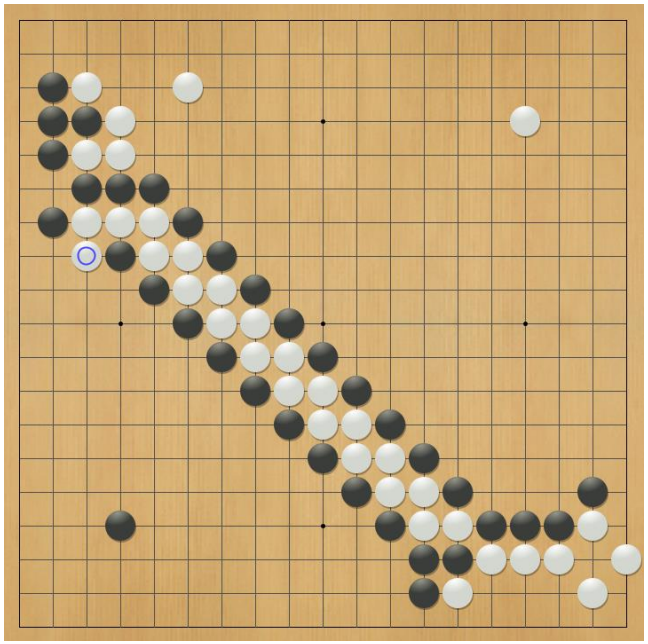


Missing Ladders (看錯征子問題)

ELF OpenGo

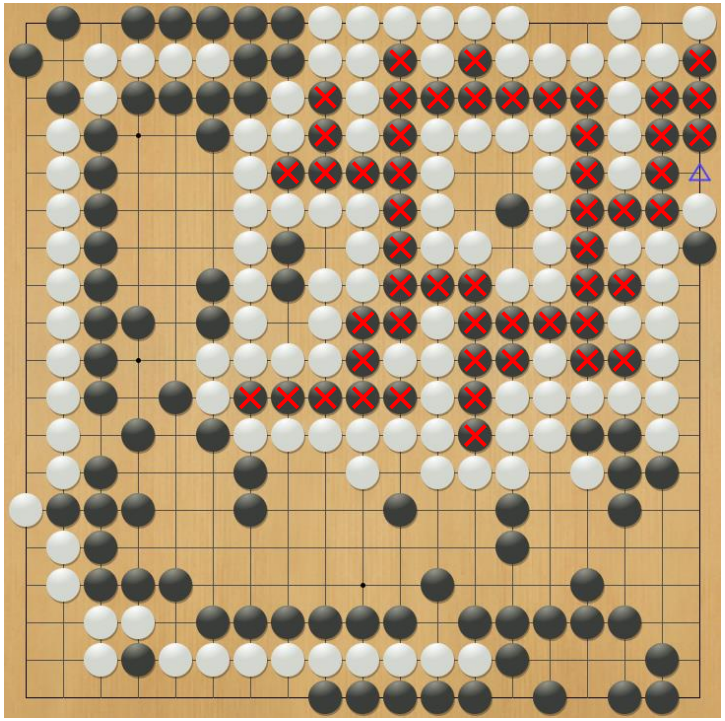


KataGo



Circular Pattern

- AlphaZero:
  - The marked black is alive.
  - This is terribly wrong!!!



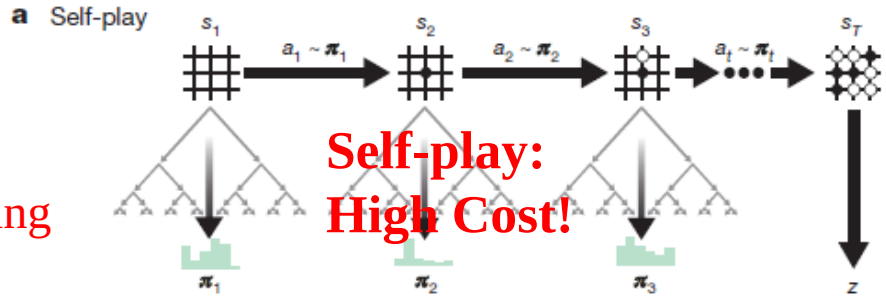
Appendix: AlphaZero/PBT

AlphaZero Based on PBT

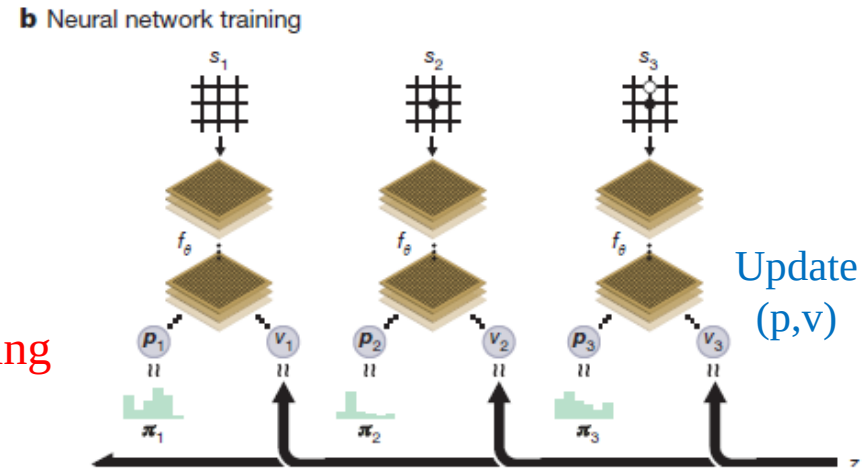
- State-of-the-art for many games, e.g., Go, Chess, Shogi.
- But, self-play (MCTS) requires a huge amount of computations.
  - **Question: how can we train a stronger AlphaZero program with less resources?**

Like a tutor

Learning improves searching



Self-play: High Cost!



Update (p,v)

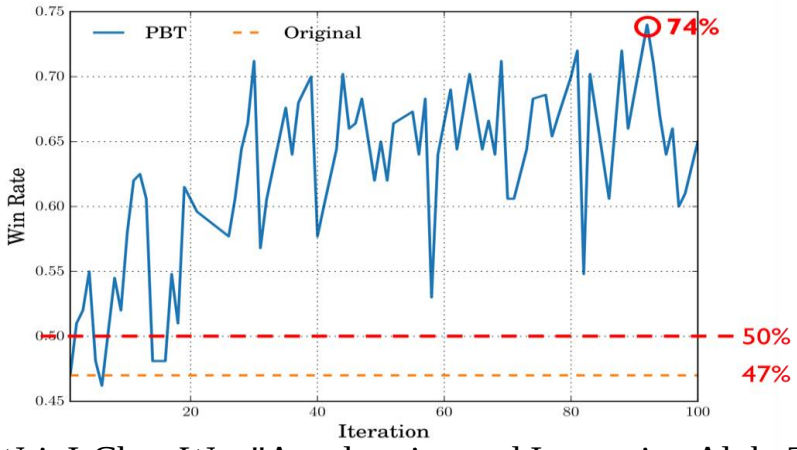
Learn from Zero Knowledge!!!

Like a student

Searching improves learning

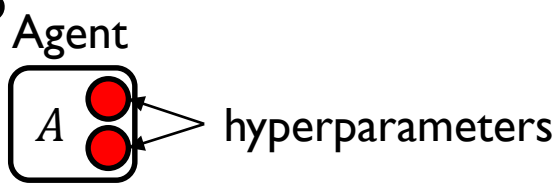
Use PBT to Improve Strength

- We propose to apply PBT (Population Based Training) to tune the hyperparameters. (Present in [**AAAI 2020**])
- **Achieve the state-of-the-art**: CGI obtained **a winrate of 74% against ELF OpenGo (developed by Facebook)**
  - **ELF OpenGo was the state-of-the-art open-sourced AlphaZero**
- In addition, **save a huge amount of computing resource**

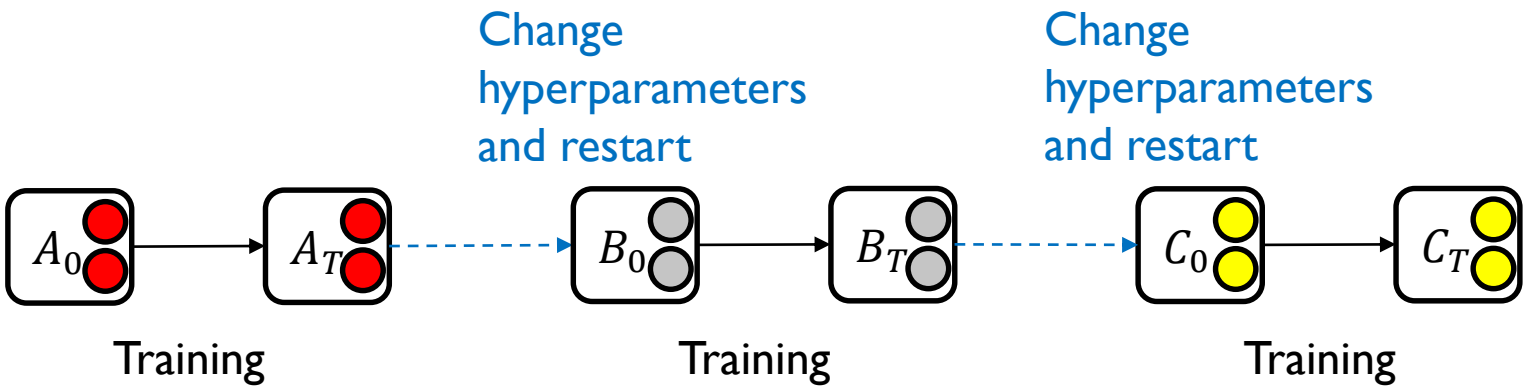


[2] Ti-Rong Wu, Tinghan Wei, I-Chen Wu, "Accelerating and Improving AlphaZero Using Population Based Training", the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20), February 2020.

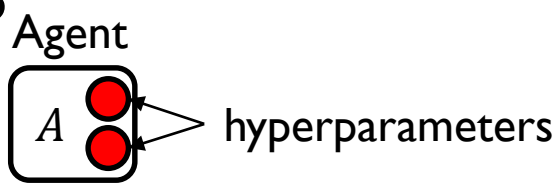
Traditional Training



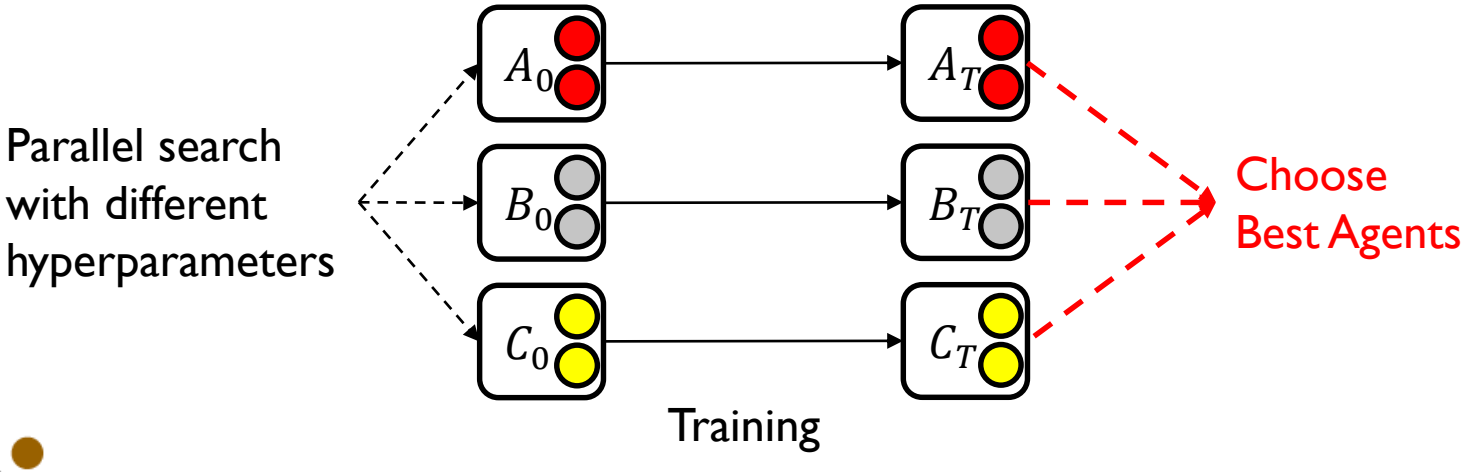
- Sequential Optimization
  - Train multiple runs to find the optimal hyperparameters
    - The hyperparameters are adjusted based on previous runs



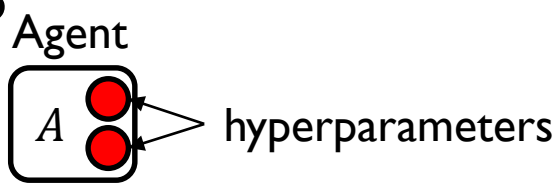
Traditional Training



- Sequential Optimization
  - Train multiple runs to find the optimal hyperparameters
    - The hyperparameters are adjusted based on previous runs
- Parallel Search
  - Train multiple models in parallel with different hyperparameters
  - Only requires for one training run

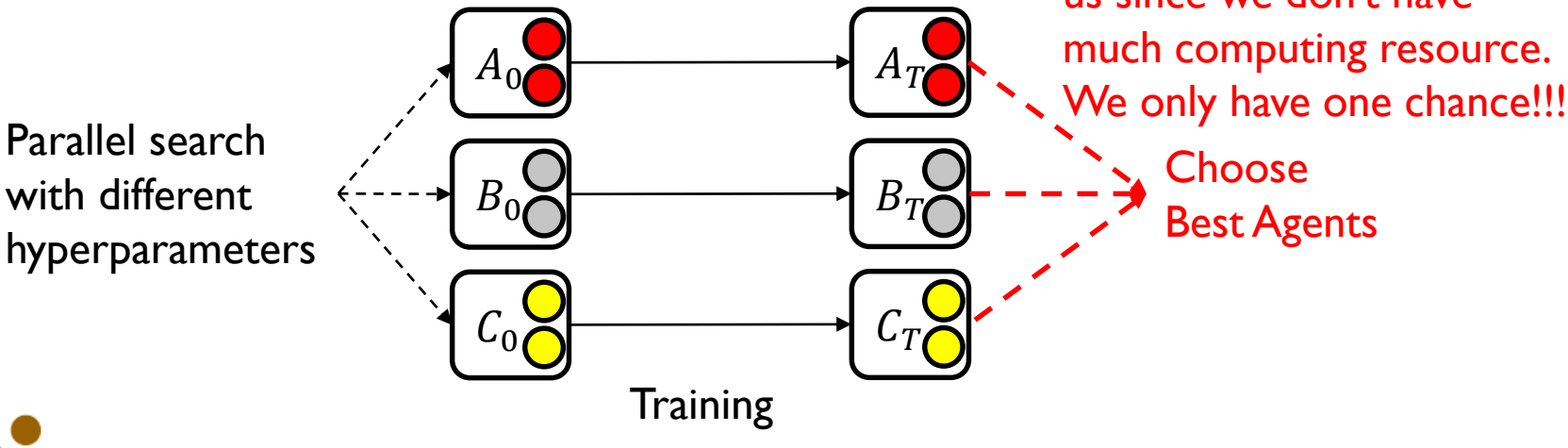


Traditional Training



- Sequential Optimization
  - Train multiple runs to find the optimal hyperparameters
    - The hyperparameters are adjusted based on previous runs
- Parallel Search
  - Train multiple models in parallel with different hyperparameters
  - Only requires for one training run

But, these won't work for us since we don't have much computing resource. We only have one chance!!!



Population Based Training

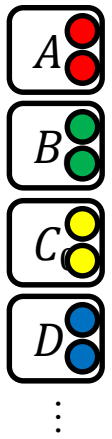
- A kind of evolutionary algorithm
  - Include a population of agents
- In each iteration of training
  - the hyperparameters are changed slightly,
  - while we keep the best performing agents
- PBT leads to more stable training and better final performance

[PBT] Jaderberg, Max, et al. **"Population based training of neural networks."** arXiv preprint arXiv:1711.09846 (2017).

Our Method

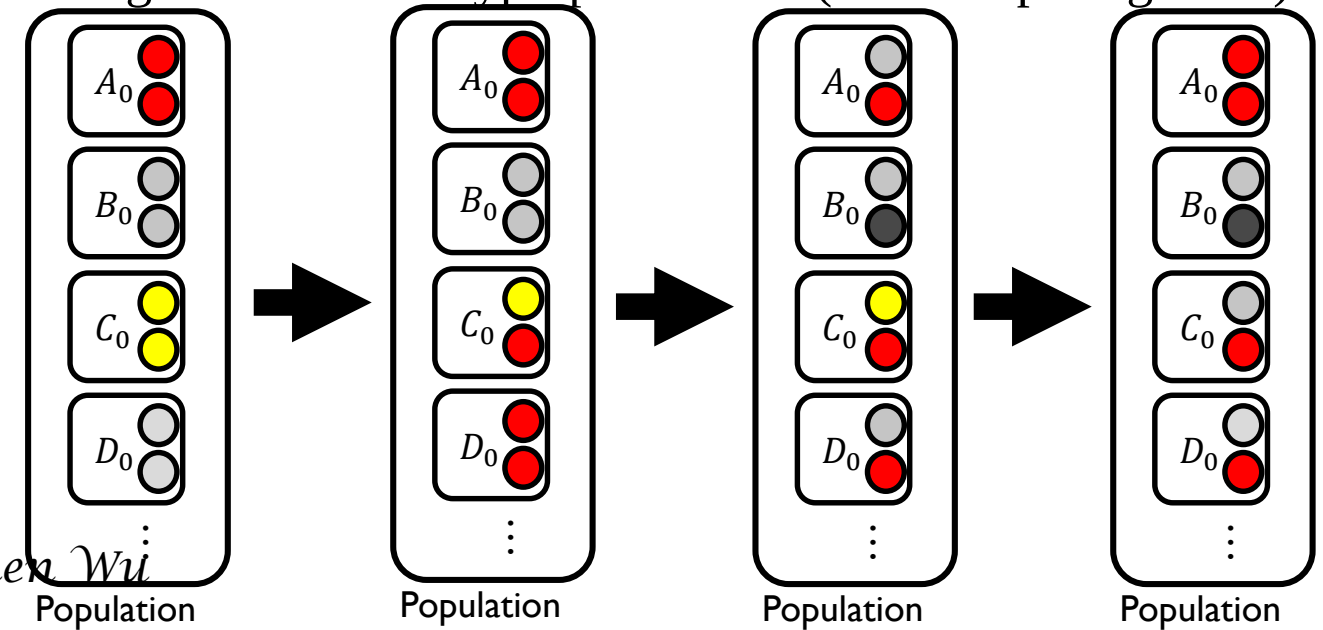
- Incorporate PBT into the AlphaZero algorithm.
  - Use a population of P agents instead of using a single agent.
    - $P = 16$ in our experiments.
  - Tune two hyperparameters online.
    - Value loss ratio: x
      - $L = xL_{value} + L_{policy} + L2$
    - Learning rate: r
    - …(actually there can be more)

Population of P agents



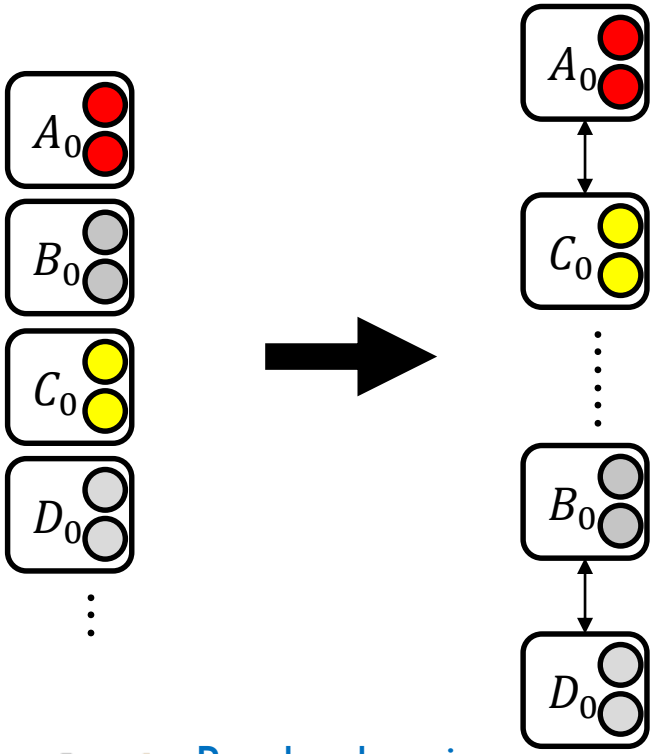
Our Method

- Incorporate PBT into the AlphaZero algorithm
  - Use a population of 16 agents instead of a single agent
- Each iteration:
  - Maintain the same number of self-play games
    - **Most time consuming part in AlphaZero.**
  - Train 16 agents and tune hyperparameters (16x computing costs)



Self-play

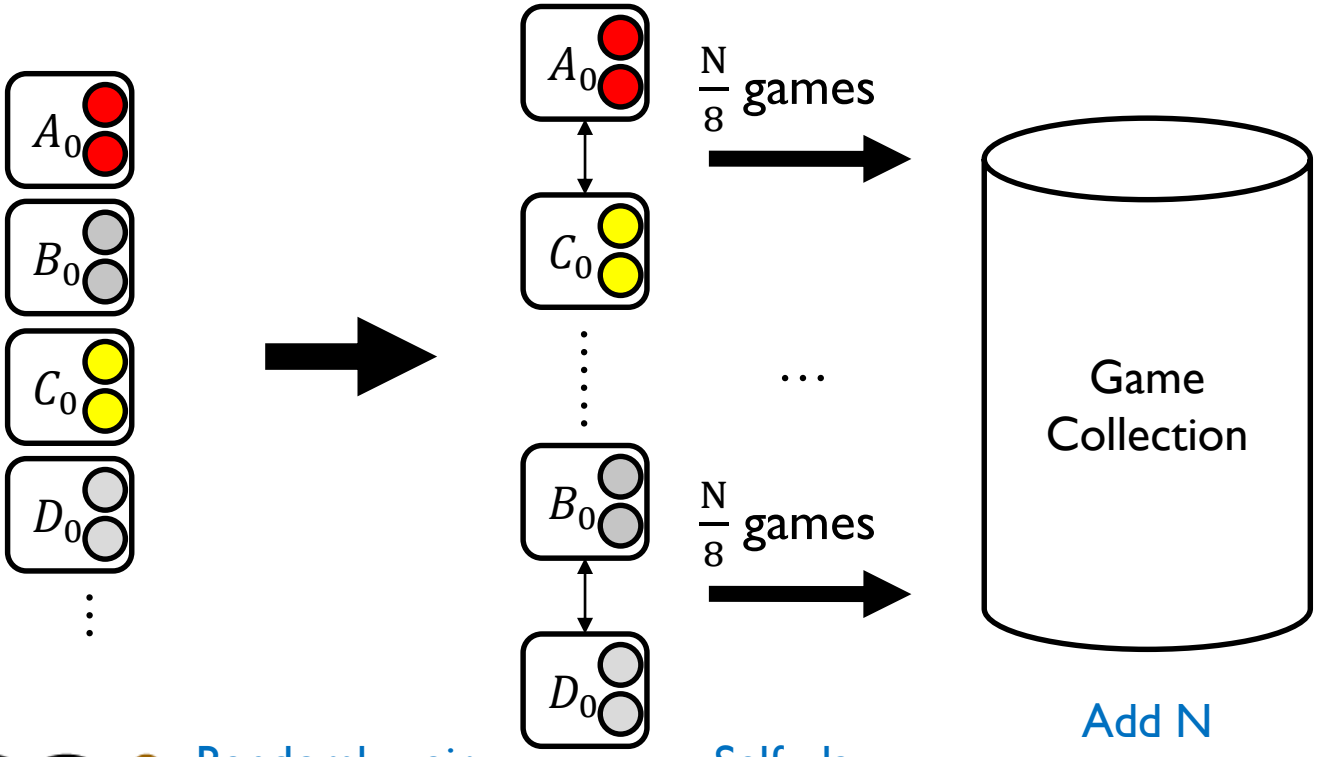
- The 16 agents are randomly paired for self-play
  - Alternating between Black and White



Randomly pair

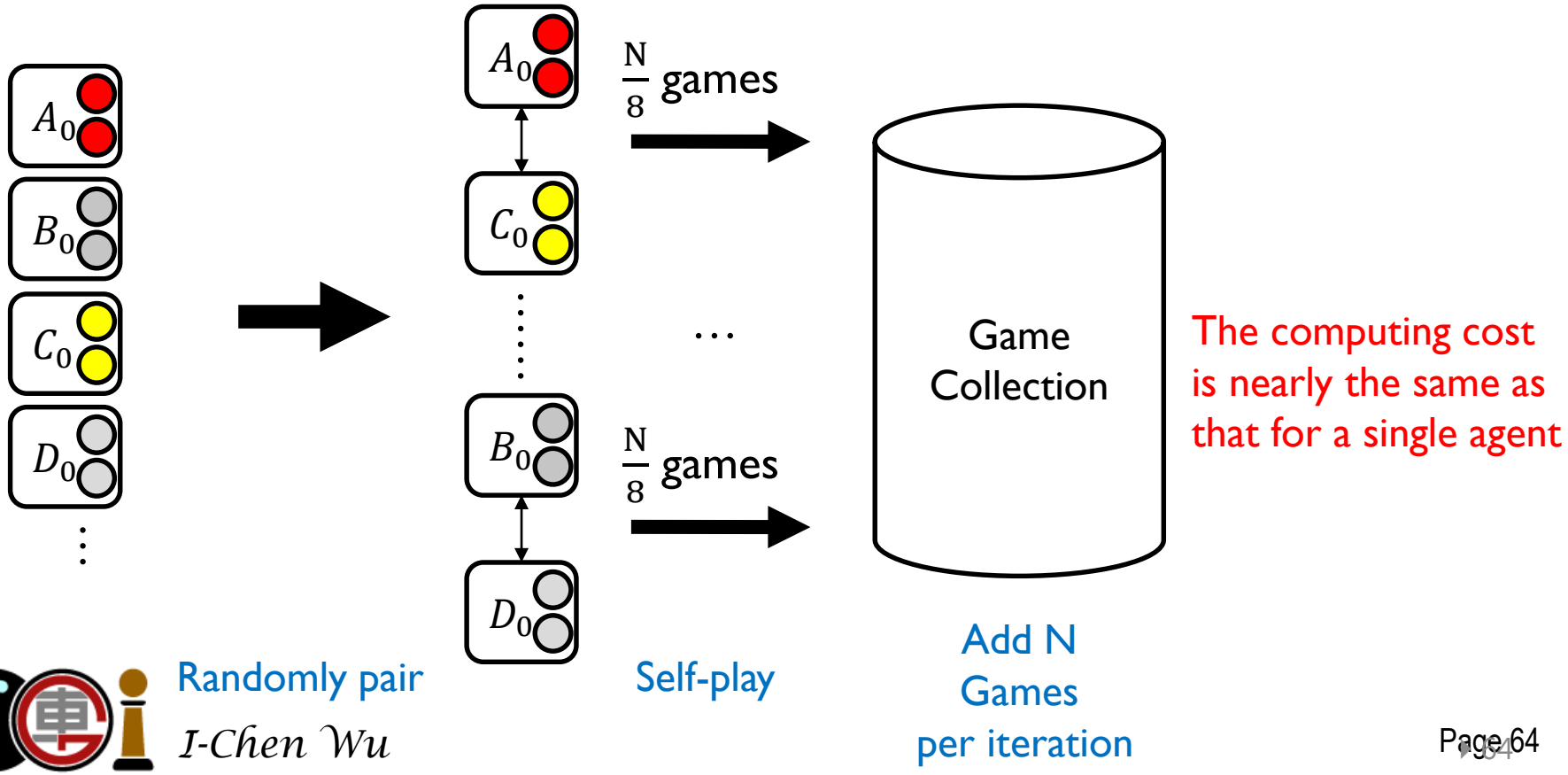
Self-play

- The 16 agents are randomly paired for self-play
  - Alternating between Black and White



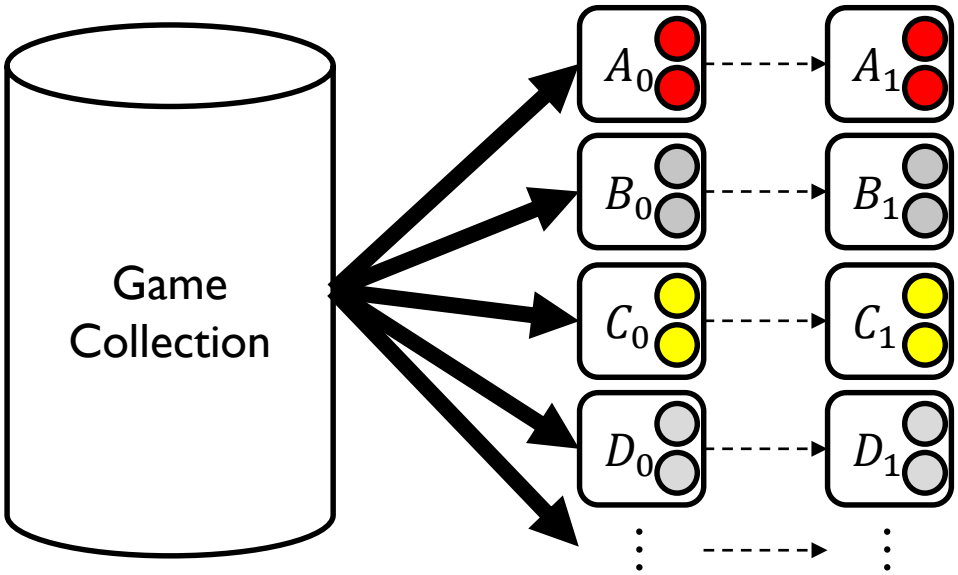
Self-play

- The 16 agents are randomly paired for self-play
  - Alternating between Black and White



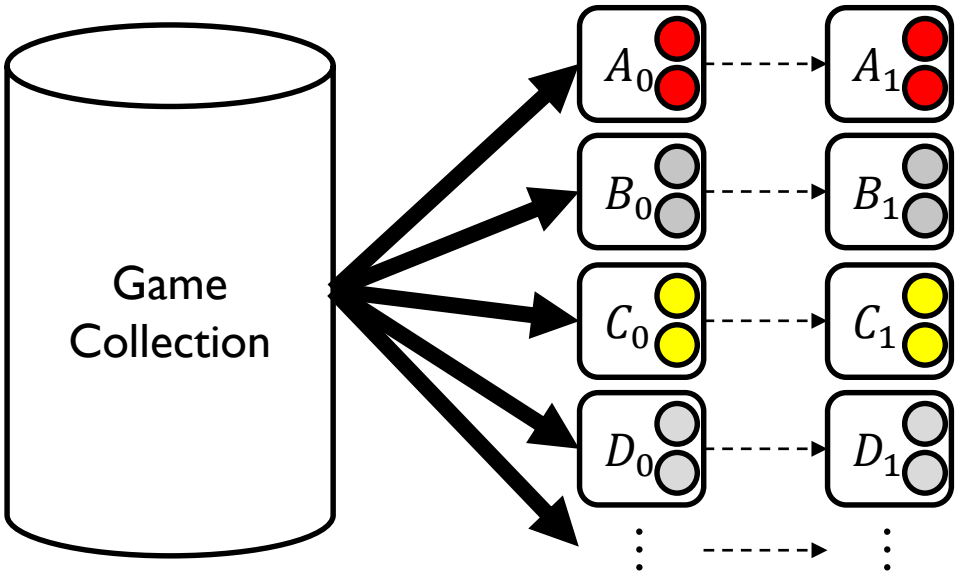
Optimization

- The 16 agents are optimized individually from the same game collection



Optimization

- The 16 agents are optimized individually from the same game collection

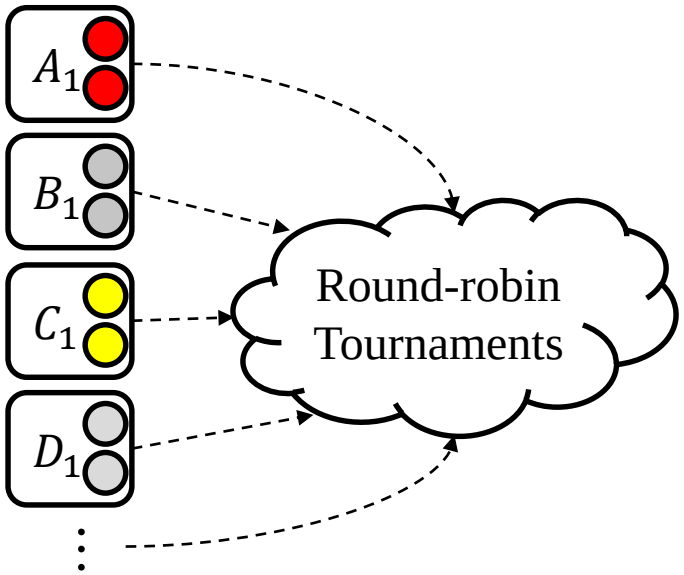


The computing cost is 16 times more than a single agent

However, the cost for optimization is still much smaller than that for self-play

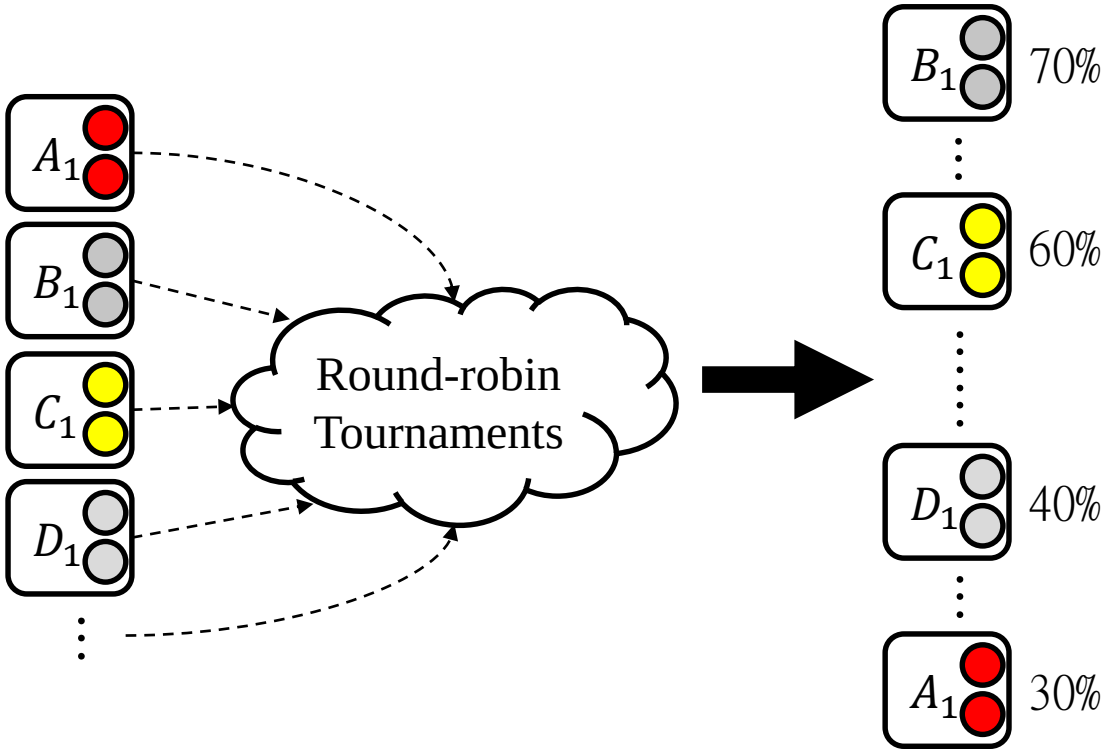
Evaluation

- The 16 agents compete in round-robin tournaments during evaluation



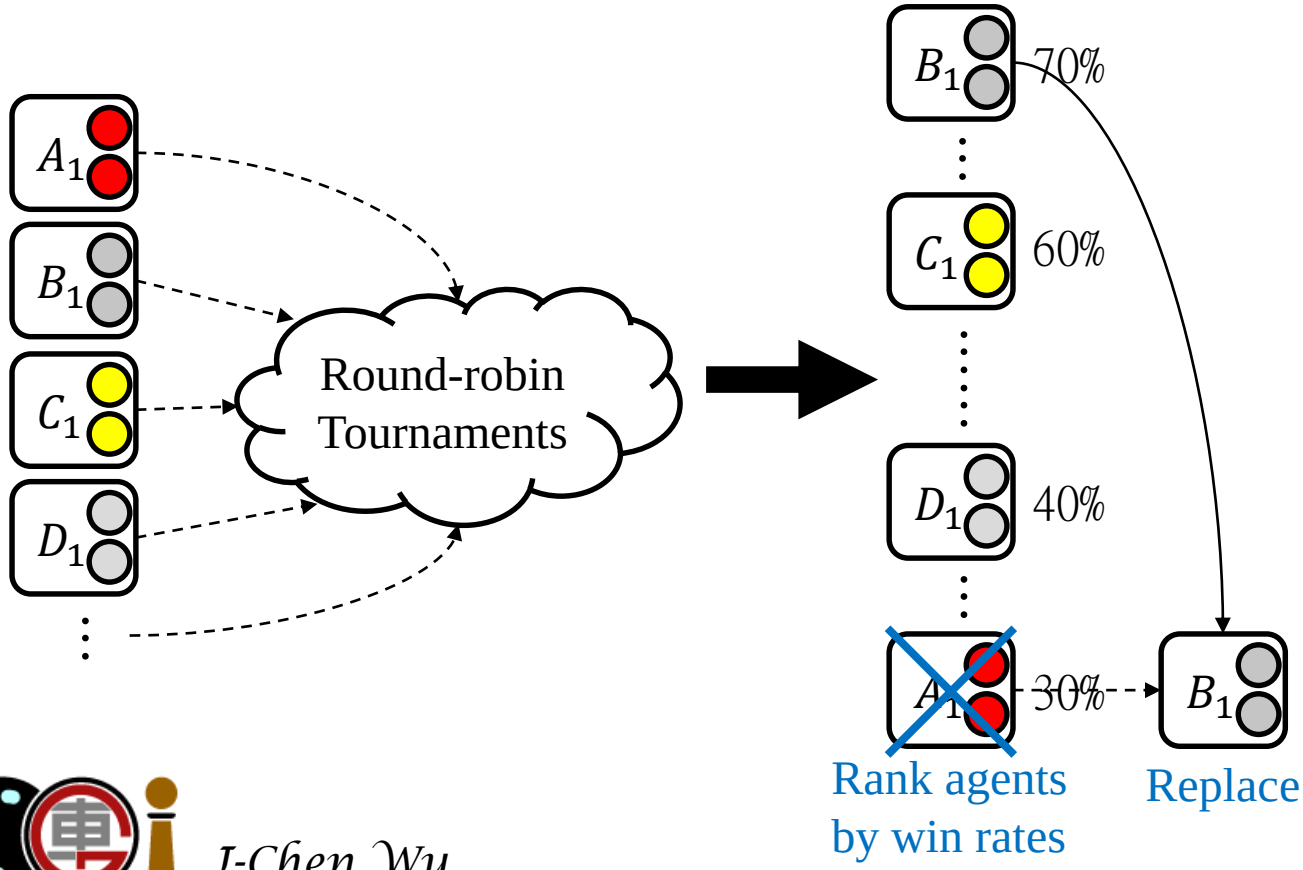
Evaluation

- The 16 agents compete in round-robin tournaments during evaluation
- Rank agents:
  - Find the agents that have higher win rates against all other agents



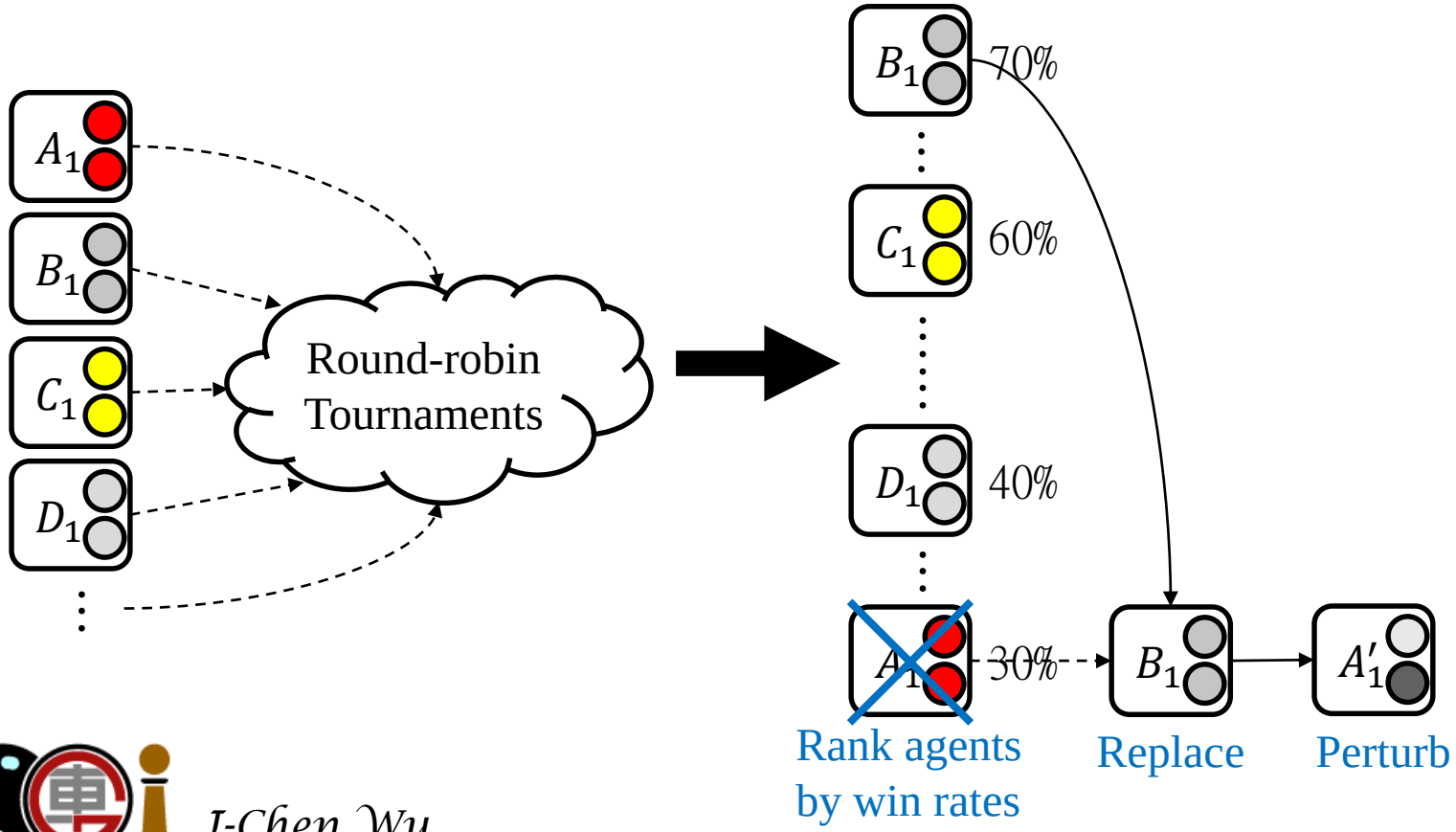
PBT Method

- PBT exploitation: *truncation*
  - Replace some bottom agents by some top agents in the population (about 1/5 agents are replaced in our experiments)



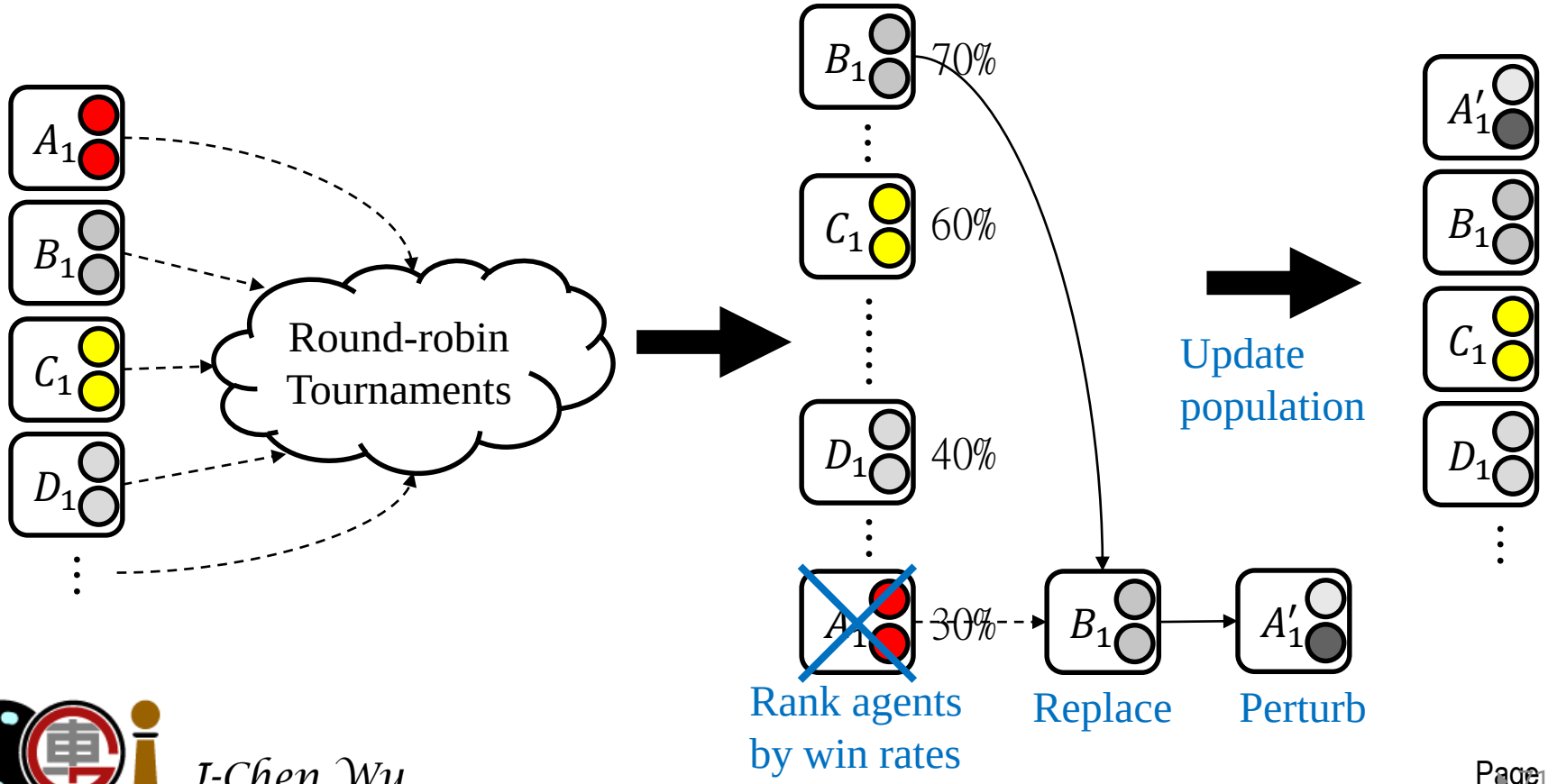
PBT Method

- PBT exploration: *perturbation*
  - The replaced agents randomly multiply their hyperparameters by a factor (0.8 or 1.2 in our experiments)



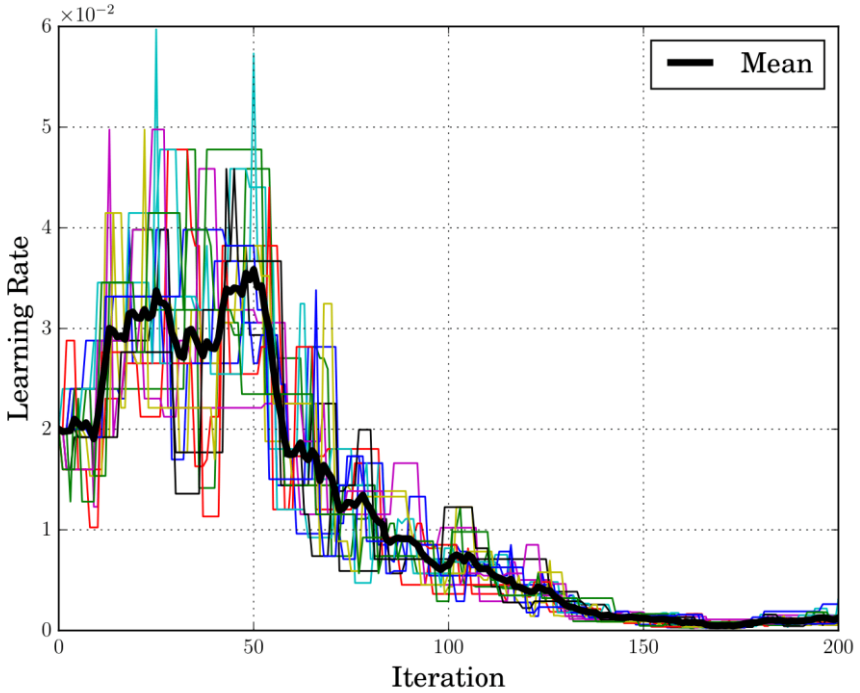
PBT Method

- PBT exploration: *perturbation*
  - The replaced agents randomly multiply their hyperparameters by a factor (0.8 or 1.2 in our experiments)



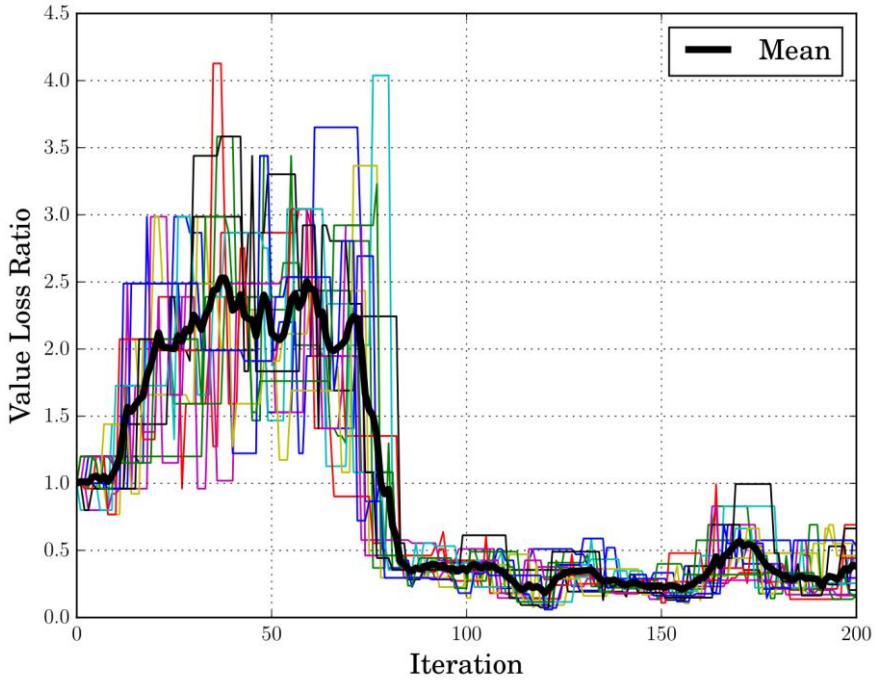
Hyperparameters for 9x9 Go – PBT

The learning rate curve during training.



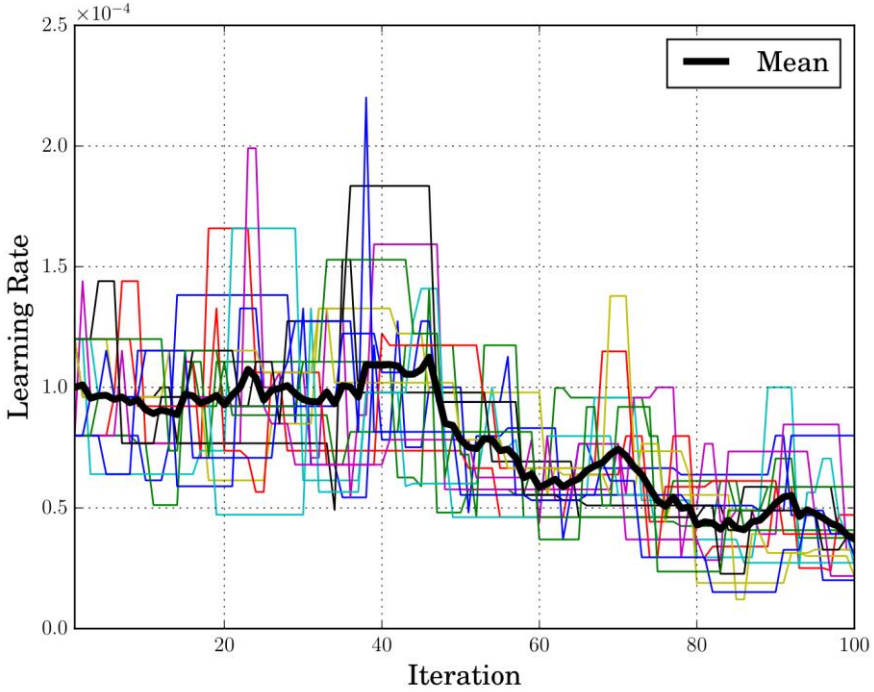
Hyperparameters for 9x9 Go – PBT

The value loss ratio curve during training.



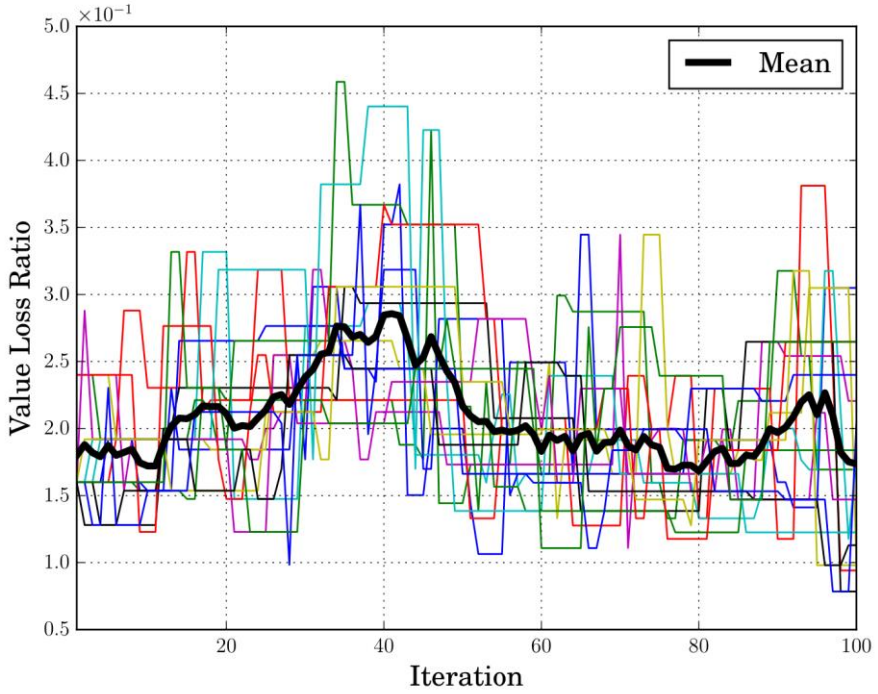
Hyperparameters for 19x19 Go – PBT

The learning rate curve during training.



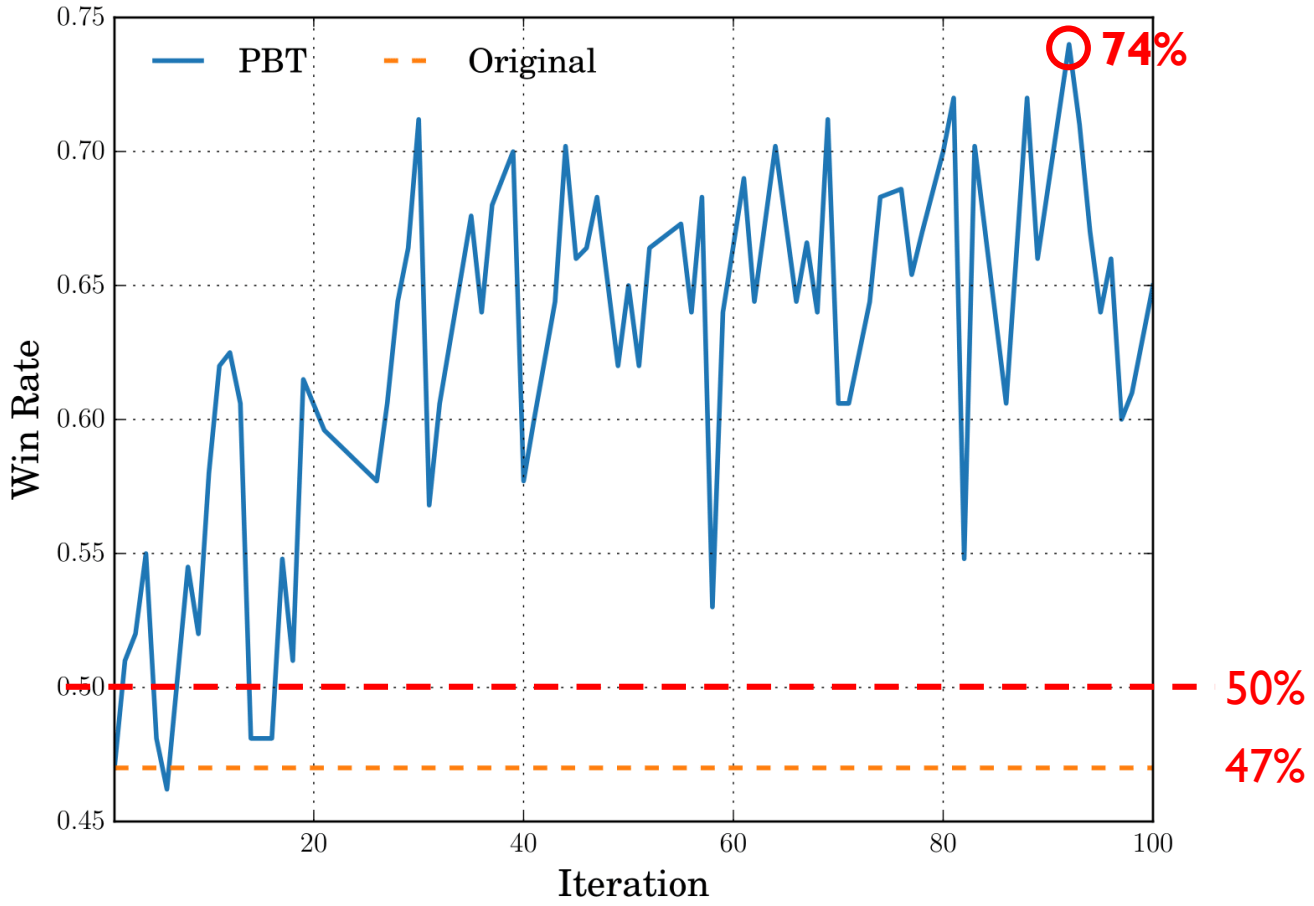
Hyperparameters for 19x19 Go – PBT

The value loss ratio curve during training.



PBT Results

Achieve 74% win rate against ELF OpenGo v2 after 100 iterations, while saving up to at least 10 times computing resources.



MuZero

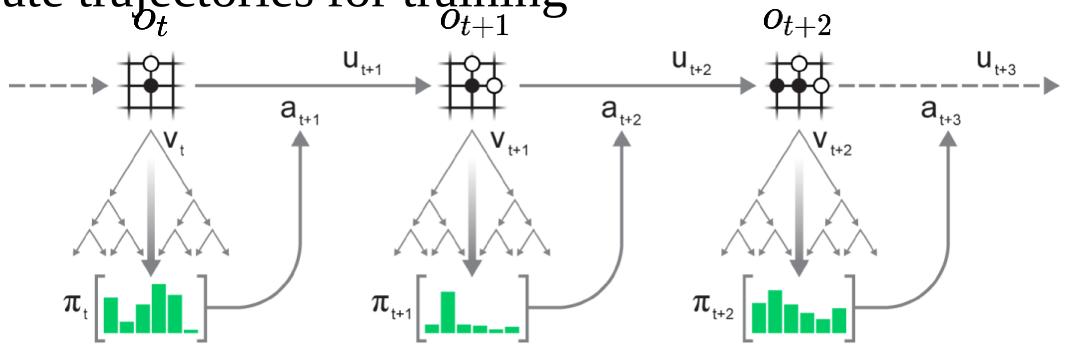
(Deterministic Environment)

MuZero

- *Silver said: "Model-based RL has failed (so far) in Atari"*
- The training of MuZero algorithm consists of two parts:
  - ⊄ Self-play: Generate trajectories for training

Interact with environments →

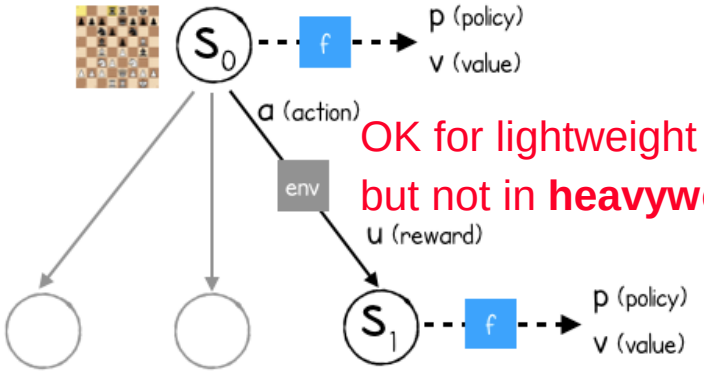
MCTS **without** simulators →
(in the case of **heavy-weight** environment)



  - ⊄ Neural network training:
    Learn the self-play value, MCTS policy, and **dynamics**

[muzero] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. Nature, 588(7839):604–609, 2020.

AlphaZero

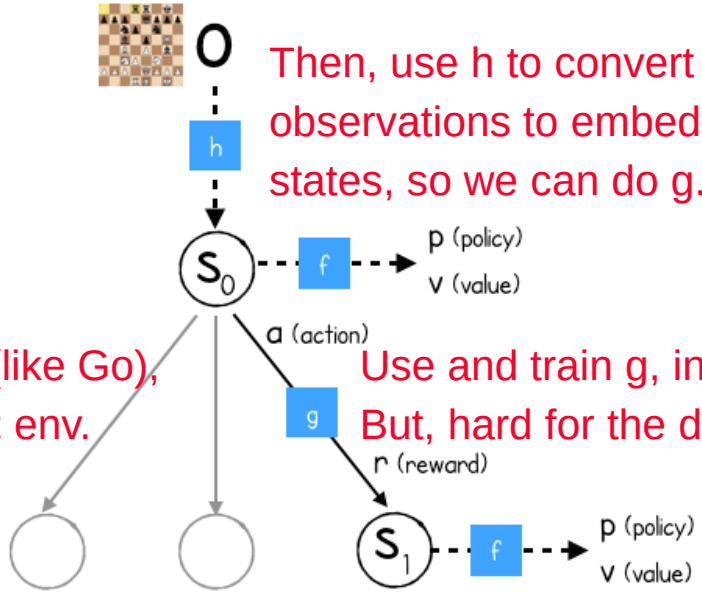


OK for lightweight env (like Go), but not in **heavyweight** env.

AlphaZero has 1 network

		from	to
prediction	f:	s	p, v

MuZero



Then, use h to convert observations to embedding states, so we can do g.

Use and train g, instead. But, hard for the dynamics

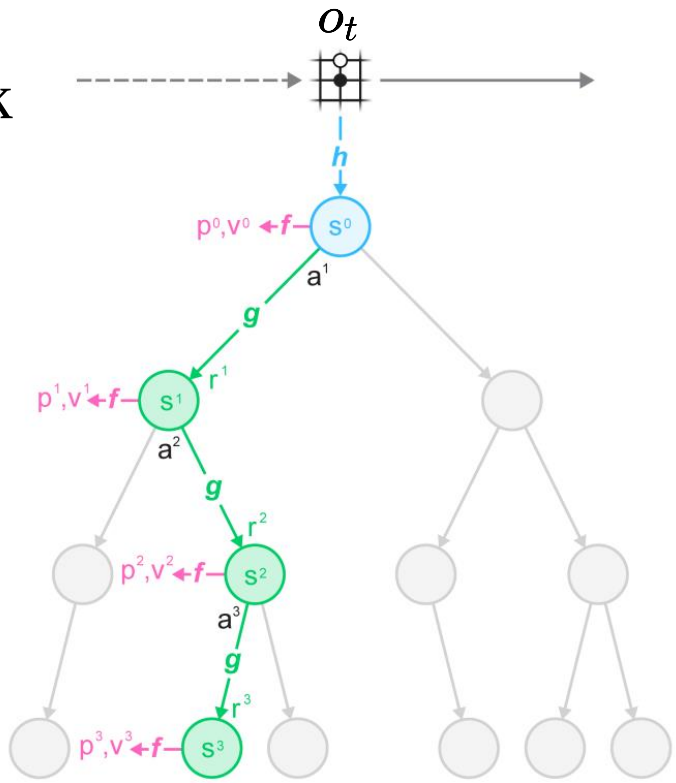
MuZero has 3 networks

		from	to
prediction	f:	s	p, v
dynamics	g:	s, a	r, s
representation	h:	o	s

Note: only deal with deterministic actions, and Atari is deterministic!!!

MuZero: MCTS Planning

- h: representation network
  $s^0 = h_\theta(o_1, \ldots, o_t)$
- g: dynamics network
  $r^k, s^k = g_\theta(s^{k-1}, a^k)$
- f: prediction network
  $p^k, v^k = f_\theta(s^k)$

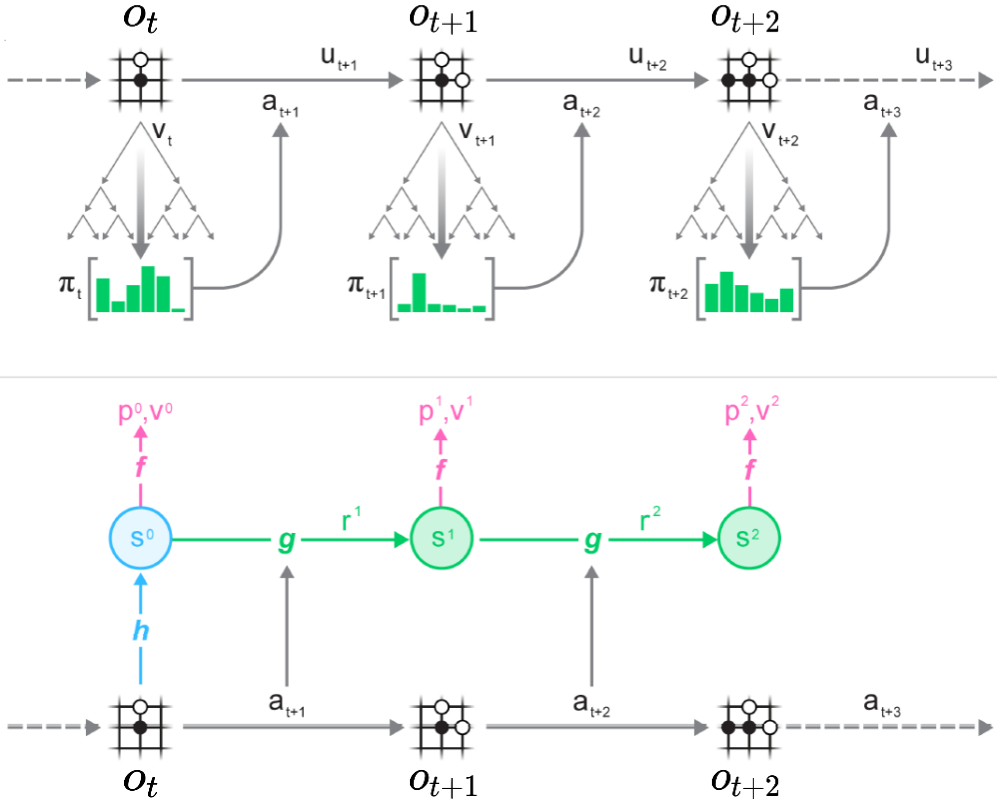


o_t : observation
s^k : hidden state
a^k : action
p^k : policy
v^k : value
r^k : reward

- the dynamics function is represented deterministically
- s^k is an internal state **without** semantics of env. state.

MuZero: Training

- Search policy
  - Let $\boldsymbol{p_t^k}$ **predict** $\boldsymbol{\pi_{t+k}}$
    - Cross-entropy
- Value (either final reward or k-step return)
  - Let $\boldsymbol{v_t^k}$ **predict** $\boldsymbol{z_{t+k}}$
  - For board games, $\boldsymbol{z_{t+k}}$ is final 0/1.
    - Mean squared error (like AZ)
  - For Atari, $\boldsymbol{z_{t+k}}$ is k-step target.
    - Cross-entropy
- Observed reward
  - Let $\boldsymbol{r_t^k}$ **predict** $\boldsymbol{u_{t+k}}$
    - Like Value
- Loss function:

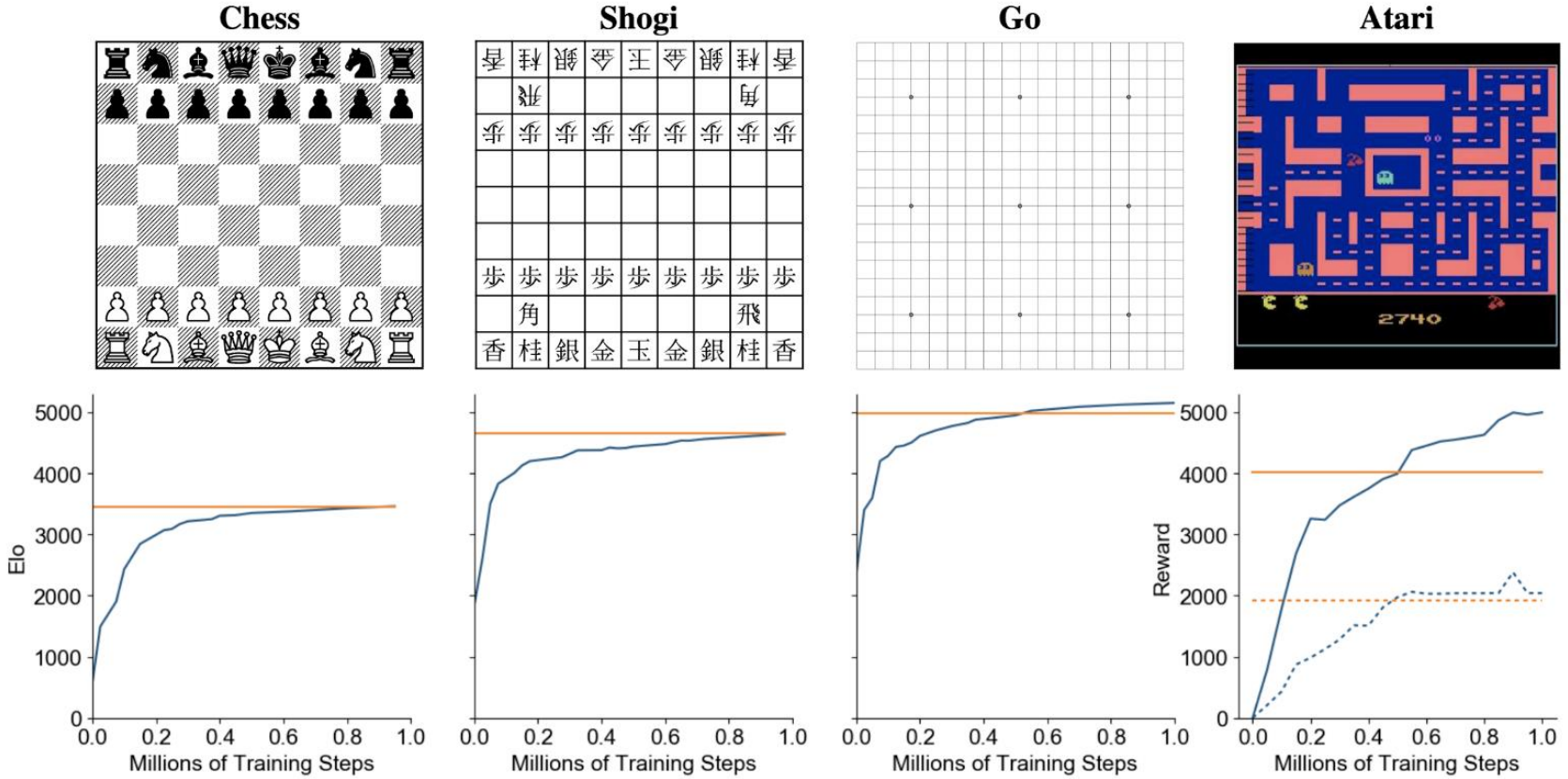


$$l_t(\theta) = \sum_{k=0}^{K} l^r\left(u_{t+k}, r_t^k\right) + l^v\left(z_{t+k}, v_t^k\right) + l^p\left(\pi_{t+k}, \boldsymbol{p}_t^k\right) + c\left|\left|\theta\right|\right|^2$$

Reward loss | Value loss | Policy loss | L2 regularization

I-Chen Wu

Experiment Results



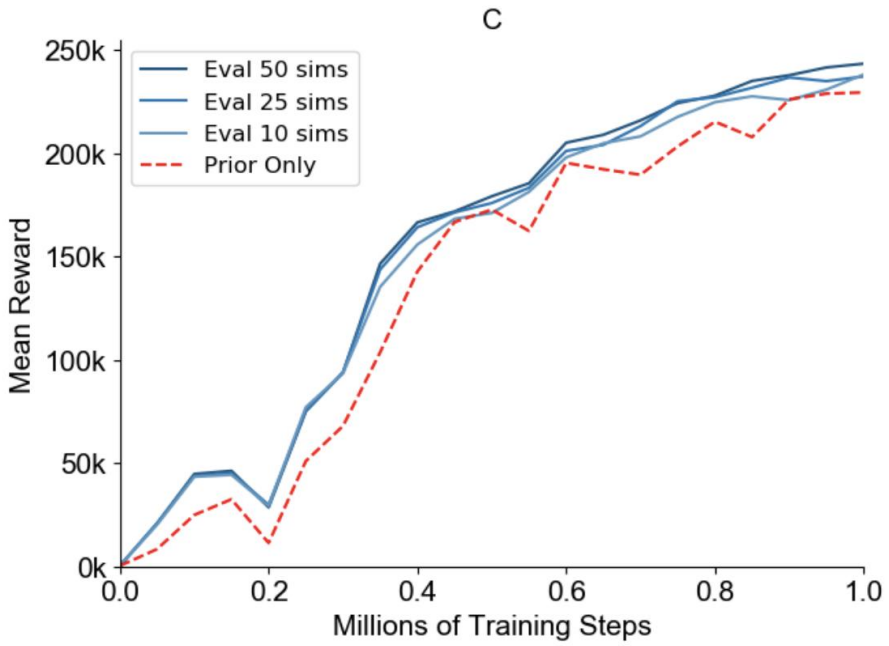
Orange: AlphaZero

Orange: R2D2

- Full: mean score
- Dashed: median score

Policy Improvement: Ms. Pacman

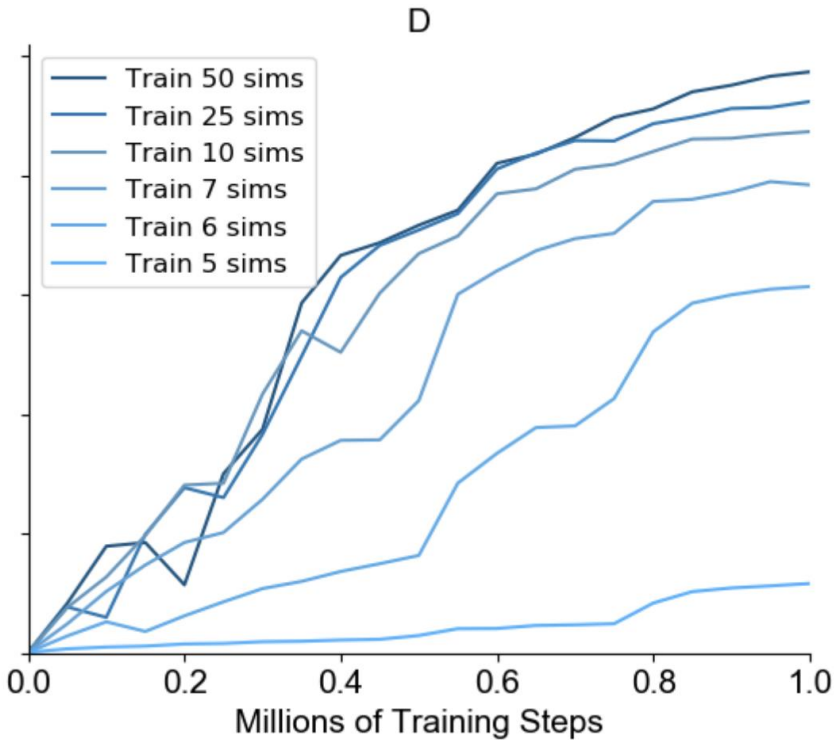
- Training with 50 simulations
- Evaluation with different sims.



Trained with Different Simulations: Ms. Pacman

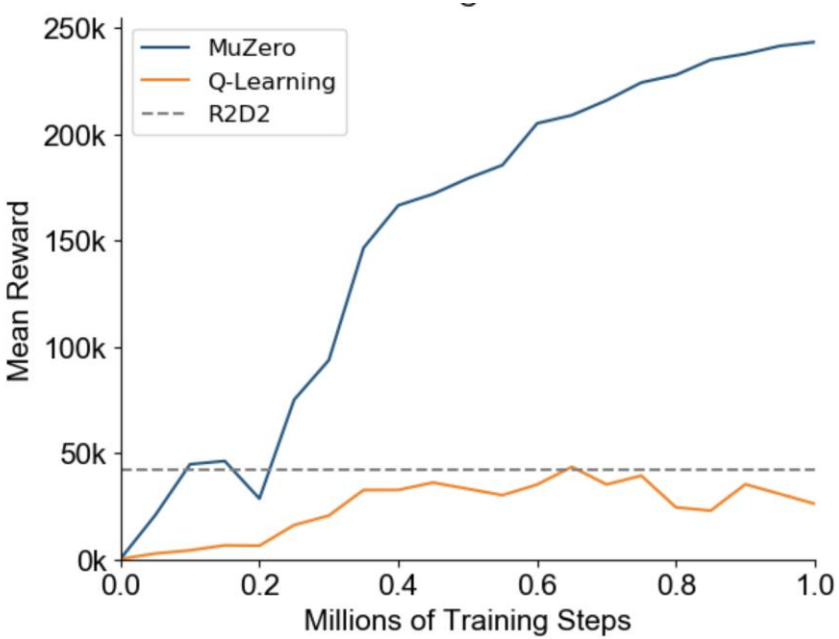
All evaluated with 50 simulations.
(Ms. Pacman has 8 possible actions.)

- Even with only 6 simulations number of actions – MuZero and improved rapidly.

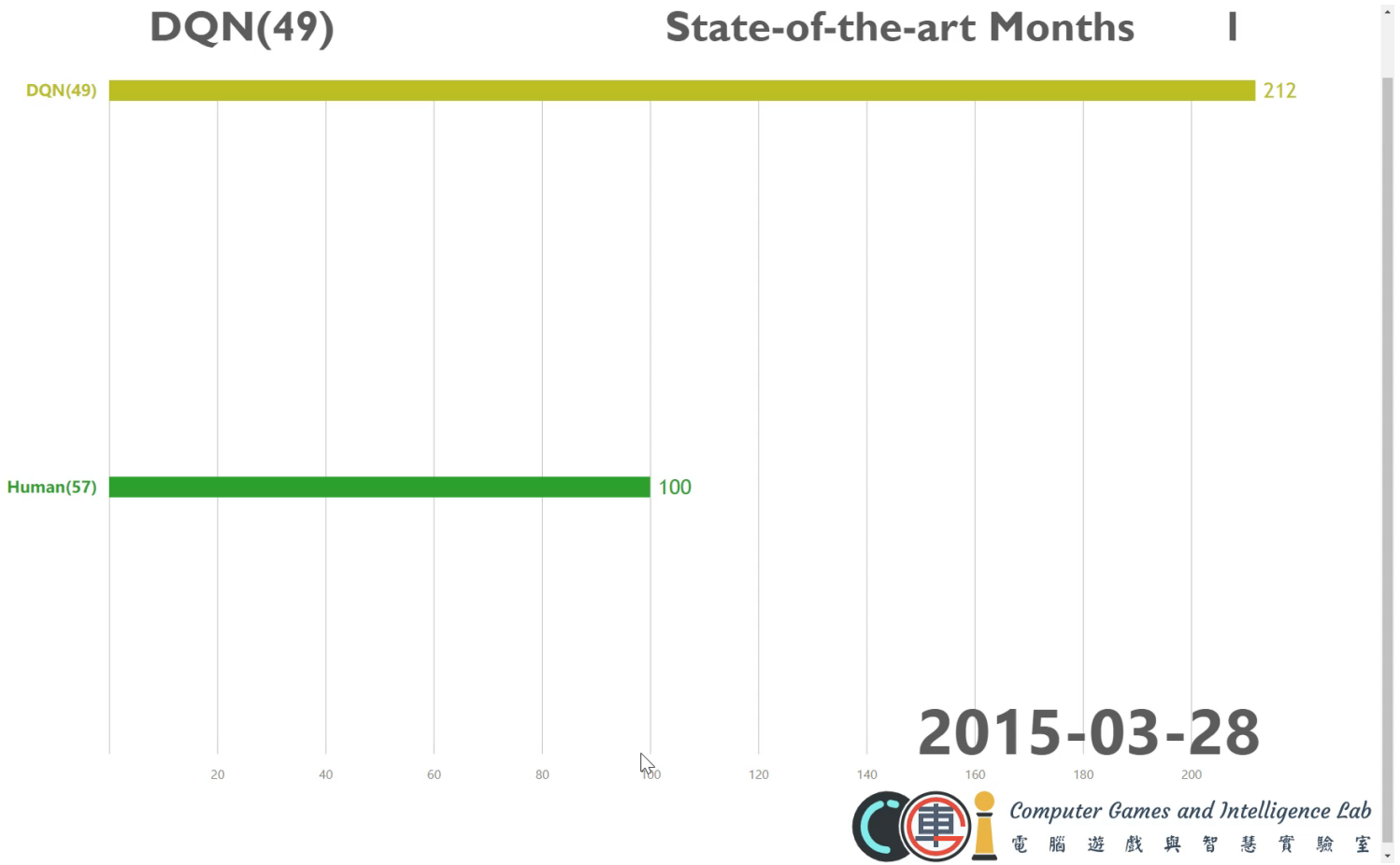


Comparison to Model-Free Algorithms

- Q-learning (modified from MuZero)
  - Replace with one Q-function
  - Replace with Q-learning objective
  - Train and evaluate without search
- Conjecture:
  - the search-based policy improvement step of MuZero provides a stronger learning signal than the high bias, high variance targets used by Q-learning.

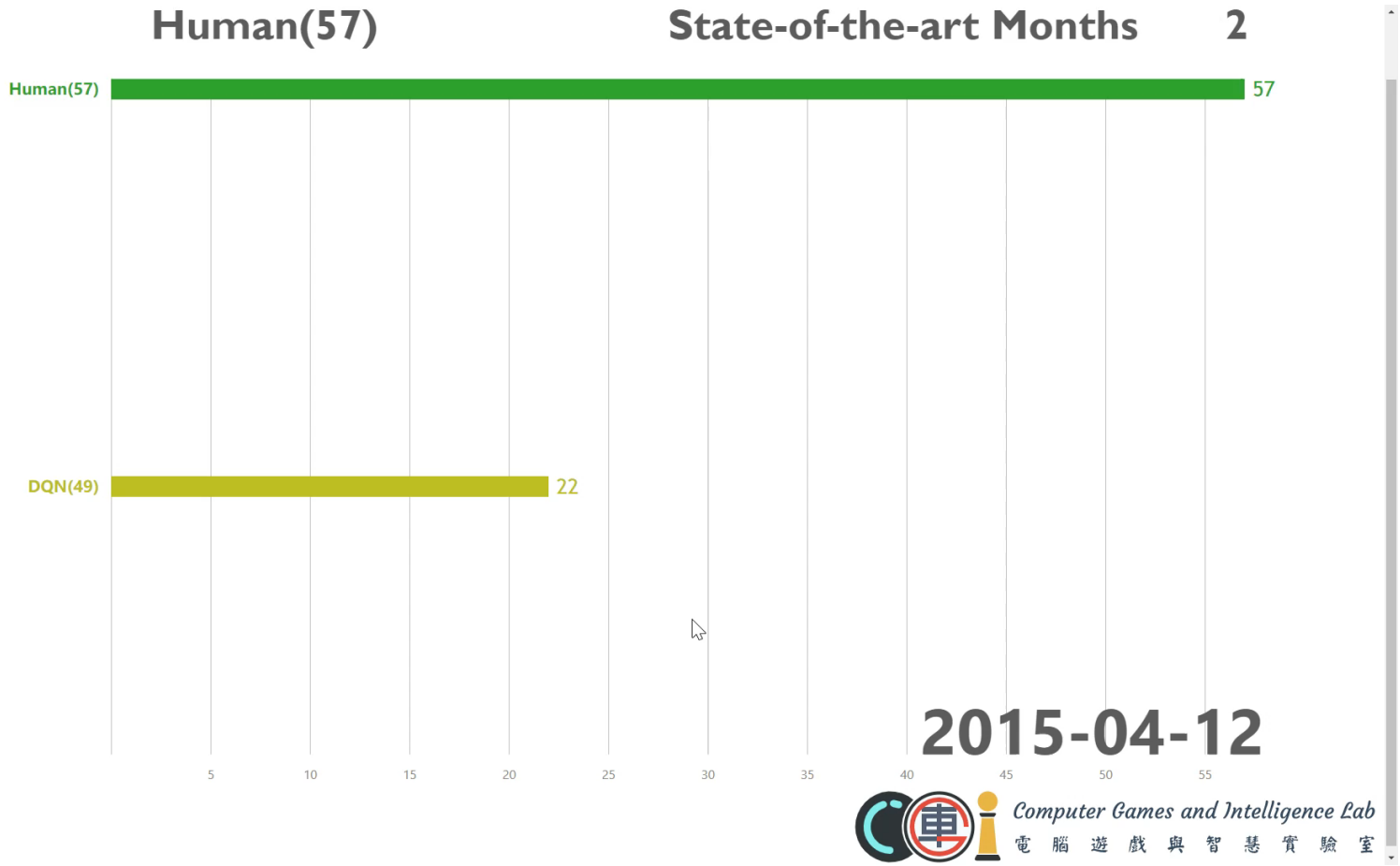


Bar Chart Race Animation: Mean Scores for Atari Games



Acknowledgement:
Thanks to C.C. Lin and G.H. Ho for making this animation

of Atari Games of Super Human



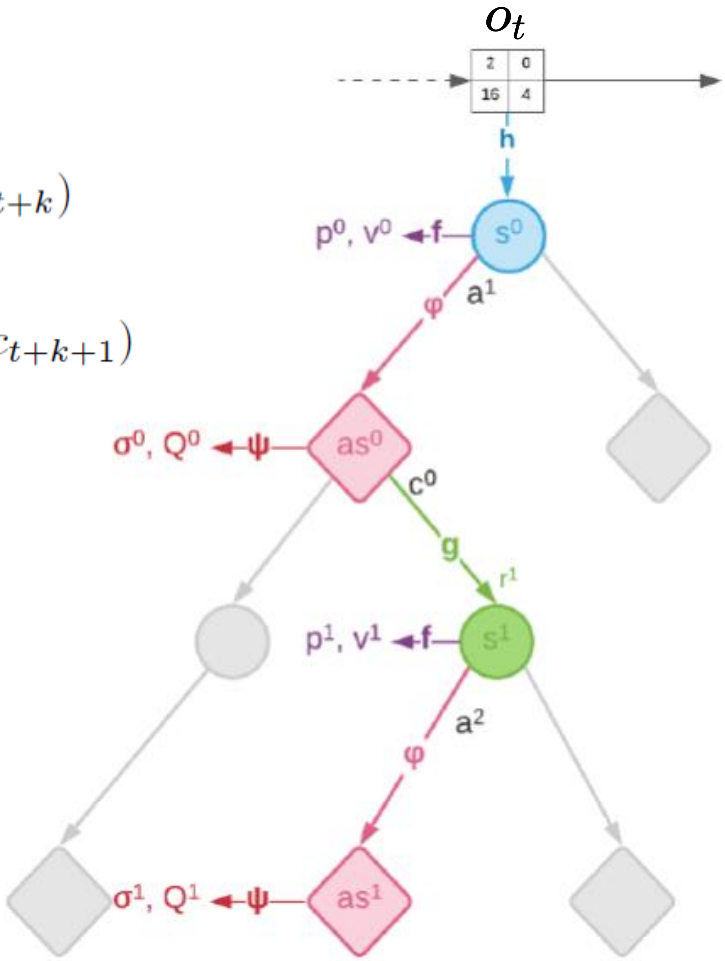
MuZero

(Stochastic Environment)

Antonoglou, Ioannis, et al. "Planning in stochastic environments with a learned model." *International Conference on Learning Representations*. 2021

Model

$$\begin{aligned}
\textit{Representation} \quad & s_t^0 = h(o_{\leq t}) \\
\textit{Prediction} \quad & p_t^k, v_t^k = f(s_t^k) \\
\textit{Afterstate Dynamics} \quad & as_t^k = \phi(s_t^k, a_{t+k}) \\
\textit{Afterstate Prediction} \quad & \sigma_t^k, Q_t^k = \psi(as_t^k) \\
\textit{Dynamics} \quad & s_t^{k+1}, r_t^{k+1} = g(as_t^k, c_{t+k+1})
\end{aligned}$$

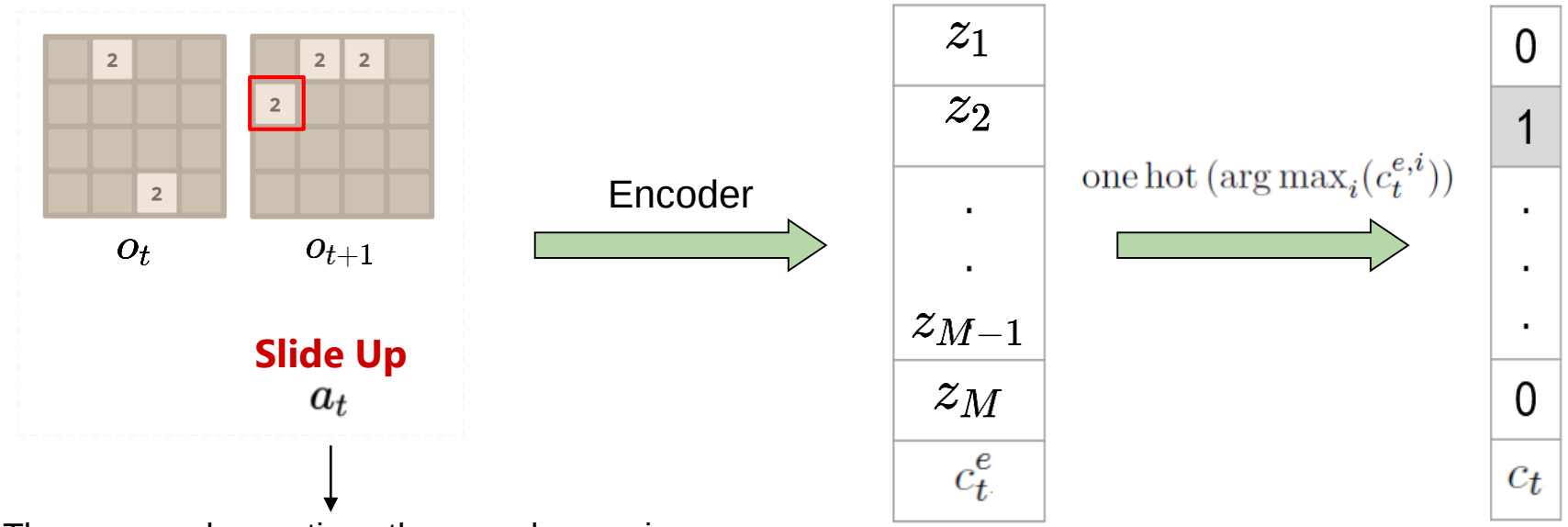


- What if stochastic environment like 2048?
- Representation:
  - Need afterstates like 2048, $\boldsymbol{as_t^k}$, and $\boldsymbol{s_t^0 = h(o_{\leq t})}$.
- Dynamics become:
  - Afterstate dynamics: An action leads to an afterstate.
  - Dynamics: An afterstate leads to next state.
    - Chance outcome: $\boldsymbol{c_t^k}$ or $\boldsymbol{c_{t+k}}$
    - Distribution over chance outcomes: $\boldsymbol{\sigma_t^k}$
- Prediction becomes:
  - Afterstate Prediction: $\boldsymbol{p_t^k, Q_t^k = g(as_t^k)}$
  - Prediction: $\boldsymbol{p_t^k, v_t^k = f(s_t^k)}$

Chance Outcome

- The encoder is trained end-to-end with MuZero model
  - Based on so-called VQ-VAE

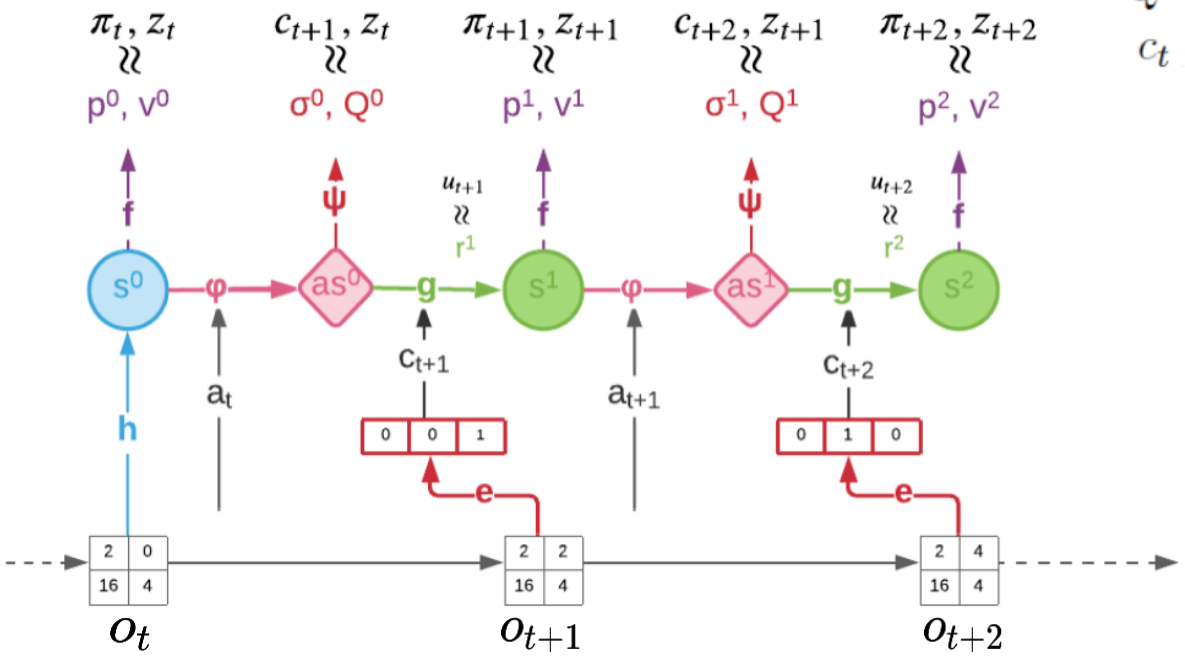
Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, NeurIPS 2017.



The paper only mentions the encoder receives observations as inputs, but I think actions are also needed.

chance outcome

Model Training



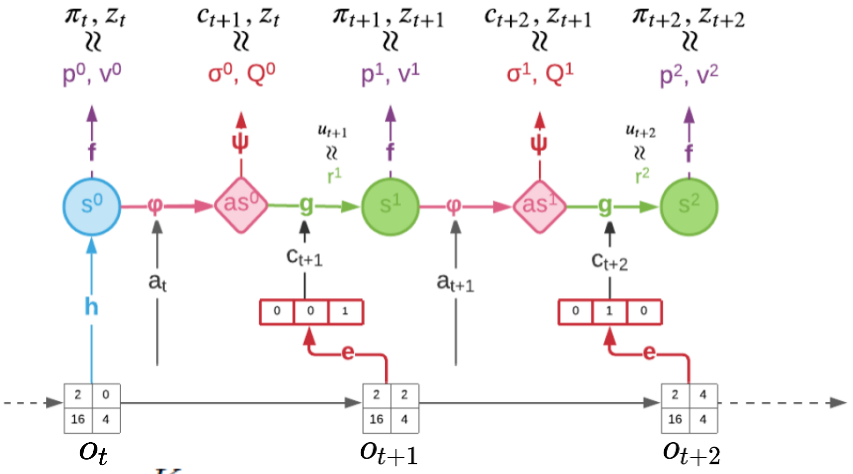
as^k: latent afterstate

σ^k: distribution over chance outcomes

Q^k: afterstate value

c_t : chance outcome

Loss Function



$$L^{total} = \overset{(1)}{L^{MuZero}} + \overset{(2)}{L^{chance}}$$

(1) $$L^{MuZero} = \sum_{k=0}^{K} l^p(\pi_{t+k}, p_t^k) + \sum_{k=0}^{K} l^v(z_{t+k}, v_t^k) + \sum_{k=1}^{K} l^r(u_{t+k}, r_t^k)$$

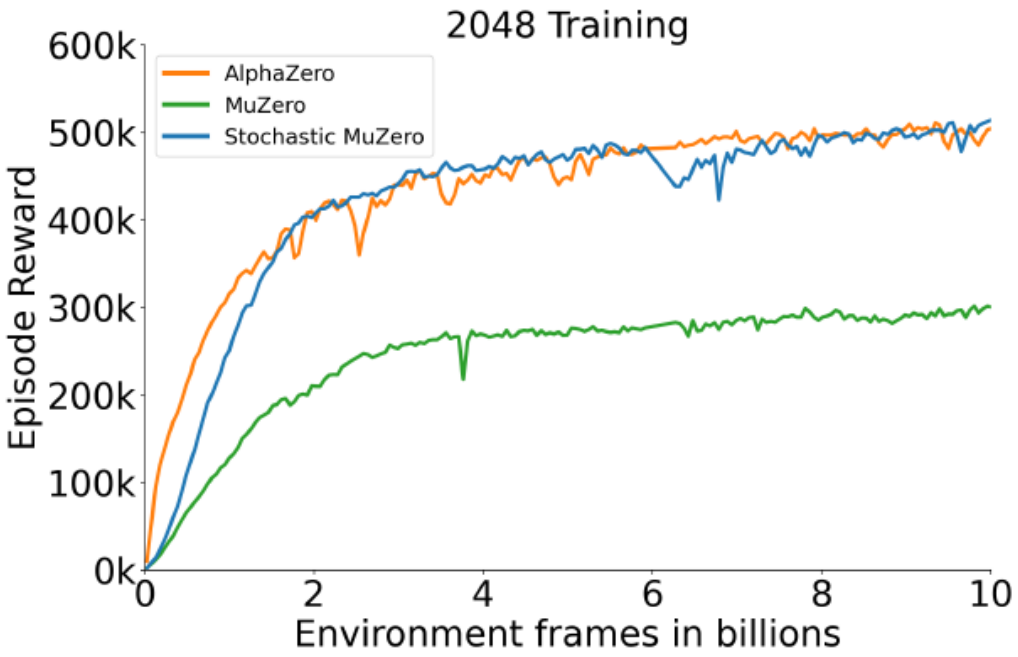
policy loss　　state value loss　　reward loss

(2) $$L_w^{chance} = \sum_{k=0}^{K-1} l^Q(z_{t+k}, Q_t^k) + \sum_{k=0}^{K-1} l^\sigma(\sigma_t^k, c_{t+k+1}) + \beta \underbrace{\sum_{k=0}^{K-1} \left\| c_{t+k+1} - c_{t+k+1}^e \right\|^2}_{\text{VQ-VAE commitment cost}}$$

One hot　　Output of encoder

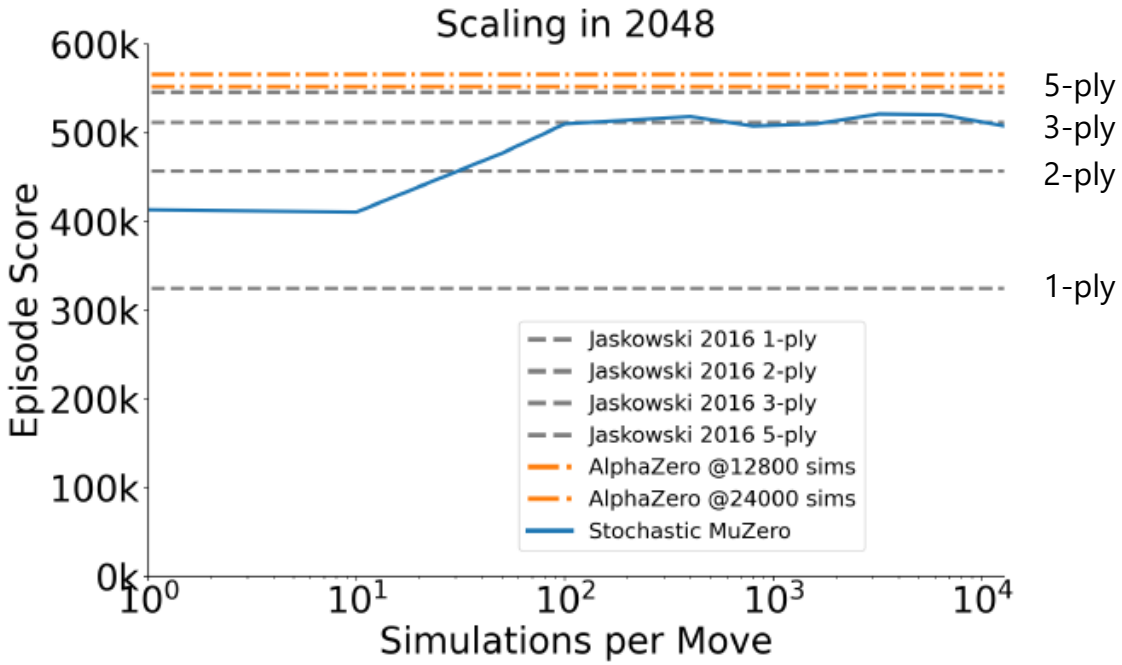
2048

- Match the performance of AlphaZero



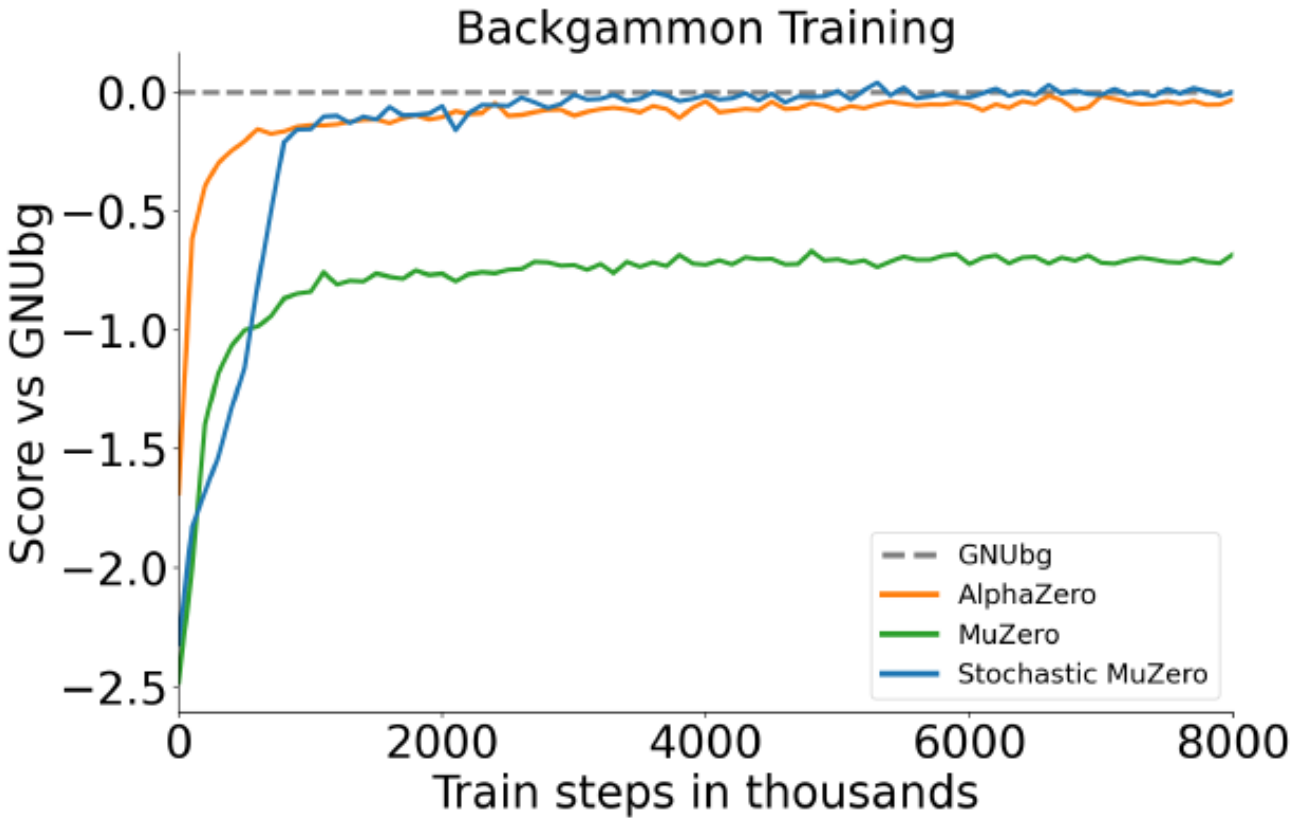
2048

- Scale well during evaluation to intermediate levels of search



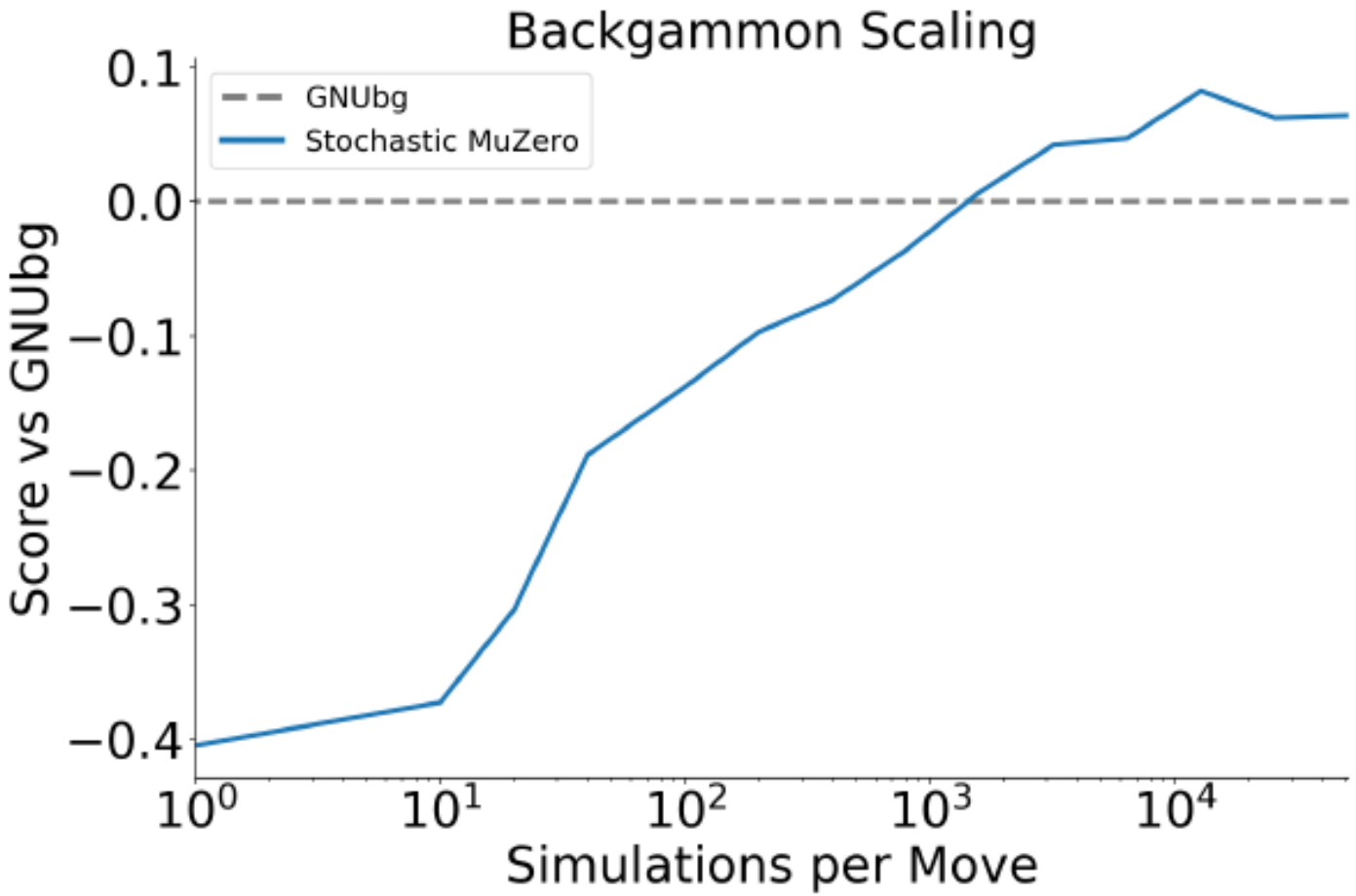
Backgammon

- Match the performance of AlphaZero



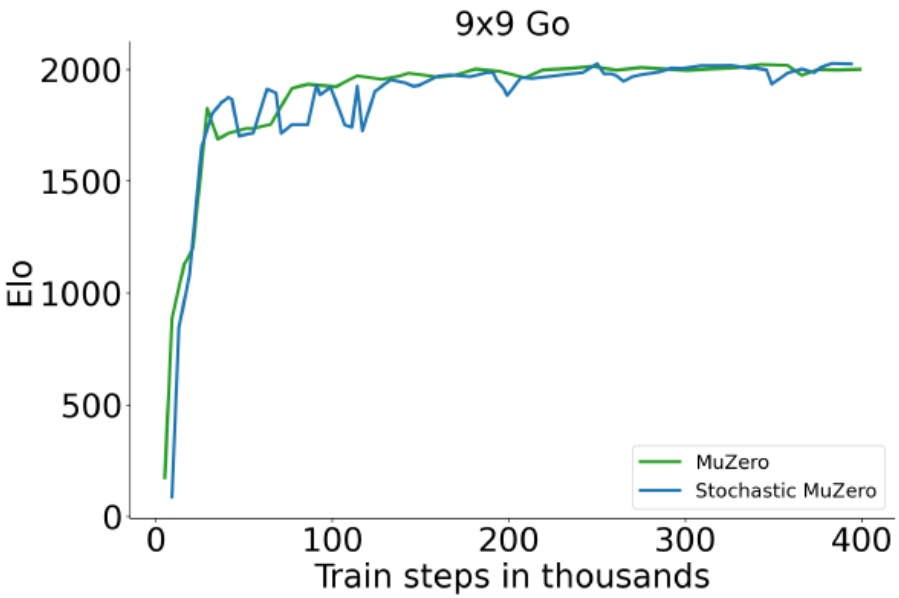
Backgammon

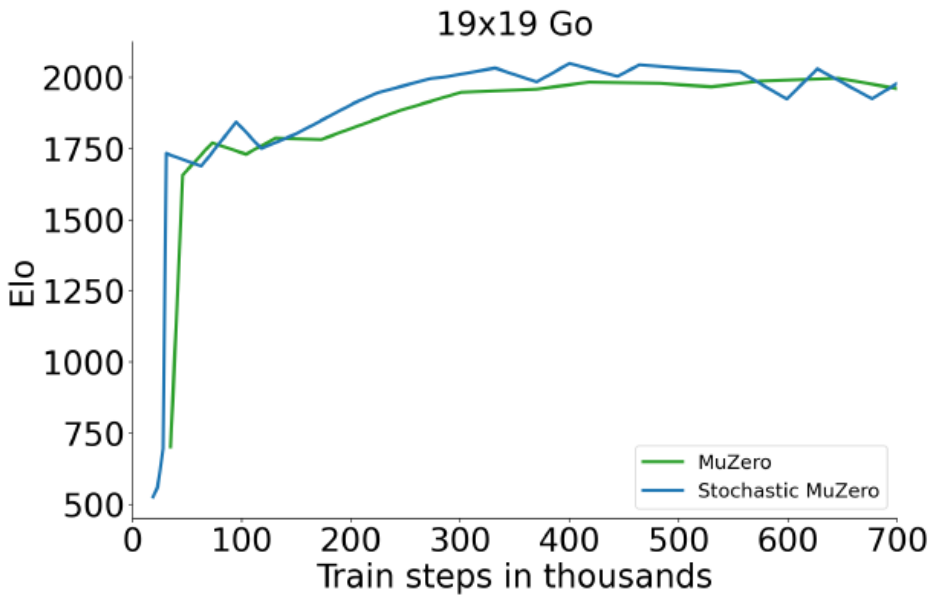
- Scale well to large searches



Go

- Maintain the performance in deterministic environments





Gumbel Muzero

Danihelka, Ivo, et al. "Policy improvement by planning with Gumbel." *International Conference on Learning Representations*. 2021.

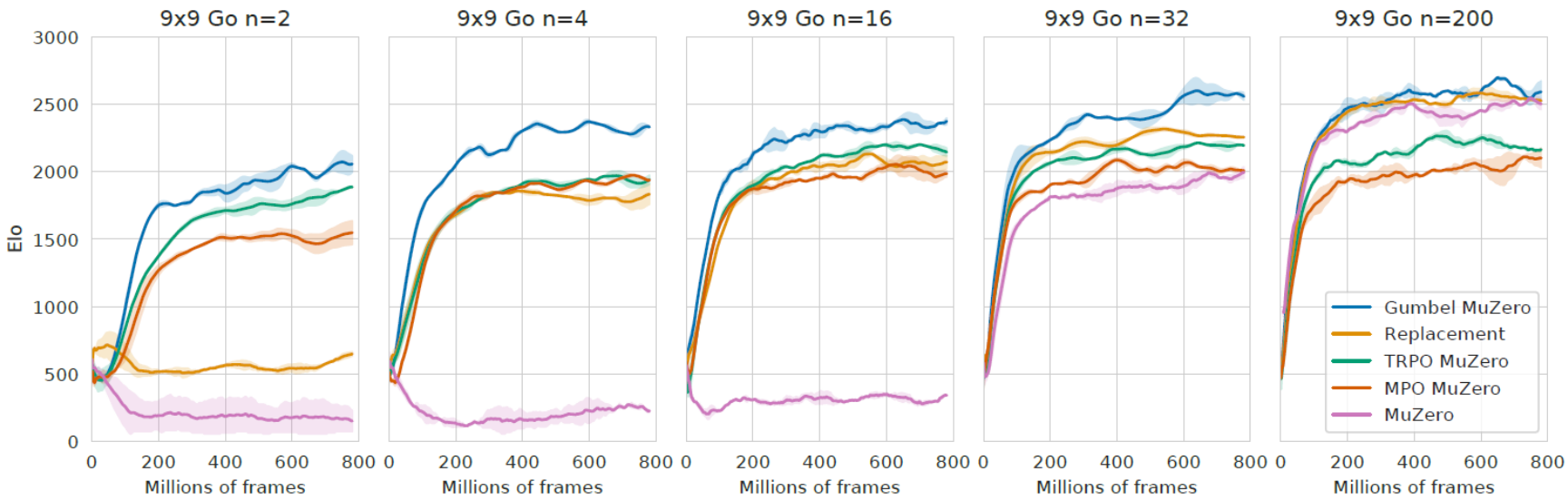
Policy Improvement by Planning with Gumbel

- Motivation
  - Neither AlphaZero nor MuZero guarantees a policy improvement, especially when using small numbers of simulations.
- Method
  - Redesign AlphaZero/MuZero tree search by planning with Gumbel-Top-k trick.
  - Instead of insufficient simulation count, we limit the search space and find the best within a smaller scope.
- Note
  - This method is useful for both AlphaZero and Muzero.

[gumbel] Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy Improvement by Planning with Gumbel. ICLR 2022.

Performance with Gumbel

- Gumbel MuZero works well even with 2 simulation counts when training



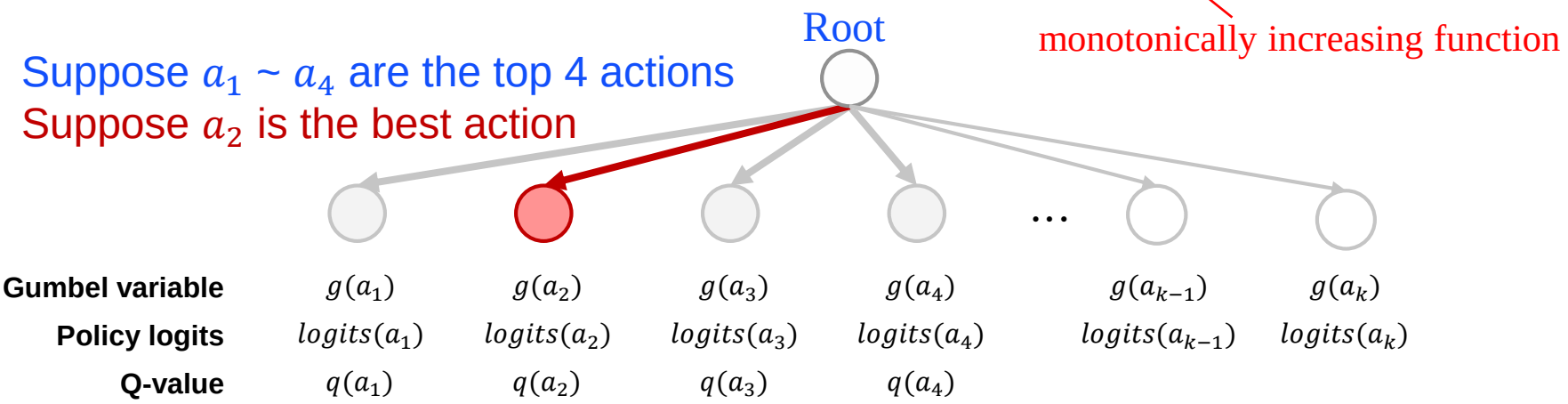
Computation Costs with Gumbel

- The speedup from a smaller number of simulations on 9x9 Go

	Training step speedup
MuZero $n = 200$	1.0
Full Gumbel MuZero $n = 200$	1.0
Gumbel MuZero $n = 200$	1.0
Gumbel MuZero $n = 32$	5.9
Gumbel MuZero $n = 16$	11.3
Gumbel MuZero $n = 8$	16.2
Gumbel MuZero $n = 4$	24.3

Planning with Gumbel

- Modif the MCTS selection with Gumbel at root.
- Example: $\boldsymbol{k}$ actions, $\boldsymbol{n = 4}$ simulations, $\boldsymbol{m = 4}$ sampled actions.
  - Select the action with the highest $\boldsymbol{g(a) + logits(a) + \sigma(q(a))}$.



[1] Danihelka, Ivo, et al. "Policy improvement by planning with Gumbel." International Conference on Learning Representations. 2021.

Gumbel MuZero – Sequential Halving

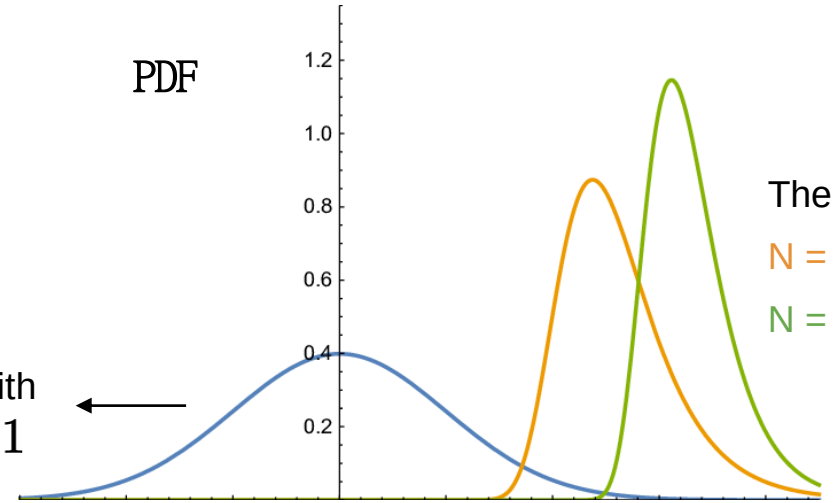
- Use **Sequential Halving**[1] when there are more simulations ($\boldsymbol{n > m}$).
- Example: $\boldsymbol{k}$ actions, $\boldsymbol{n = 600}$ simulations, $\boldsymbol{m = 8}$ sampled actions
  - Divide to **3** phases, each phase **200** simulations.
  - Allocate **200** simulations to considered actions **equally** at each phase.
  - Reject **one half** of considered actions after each phase.

Action	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	…	a_{k-1}	a_k
Visits in 1st phase	25	25	25	25	25	25	25	25	0	…	0	0
Visits in 2nd phase	50	50	50	50	0	0	0	0	0	…	0	0
Visits in 3rd phase	100	100	0	0	0	0	0	0	0	…	0	0
Visits in total	175	175	75	75	25	25	25	25	0	…	0	0

[1] Karnin, Zohar, Tomer Koren, and Oren Somekh. "Almost optimal exploration in multi-armed bandits." International Conference on Machine Learning. PMLR, 2013.

Appendix: Planning with Gumbel

- Gumbel distribution
  - used to model the distribution of the maximum (or the minimum) of a number of samples of various distributions



The corresponding Gumbel distributions for

N = 100 samples

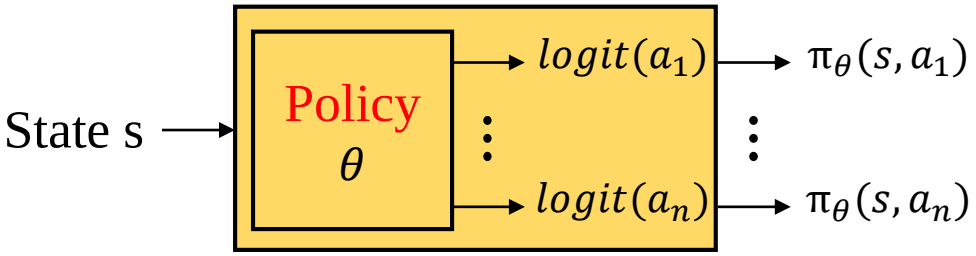
N = 1000 samples

Normal distribution with $\mu = 0$ and $\sigma^2 = 1$

Gumbel-Max Trick

$$\text{logits}(a) \xrightarrow{\text{softmax}} \pi$$

- Let π be a categorical distribution with logits $\in R^k$
- Let $logits(a)$ be the logit of the action a output by **policy network**
  - In practice,



- **Theroem: Sampling from the distribution** π is equivalent to **generating a vector of *k* (independent) Gumbel variables and then taking *argmax***:

$$(g \in \mathbb{R}^k) \sim \text{Gumbel}(0)$$

$$A = \arg\max_a (g(a) + \text{logits}(a))$$

Gumbel-Max Trick

- An example

```
policy_logits:
[ 0.8 -0.1  0.5  0.7  0.2]

policy from softmax(policy_logits):
[0.2777 0.1129 0.2057 0.2513 0.1524]

proportion of 1000000 samples by argmax{ policy_logits + g }:
[0.2778 0.1128 0.2055 0.2513 0.1526]
```

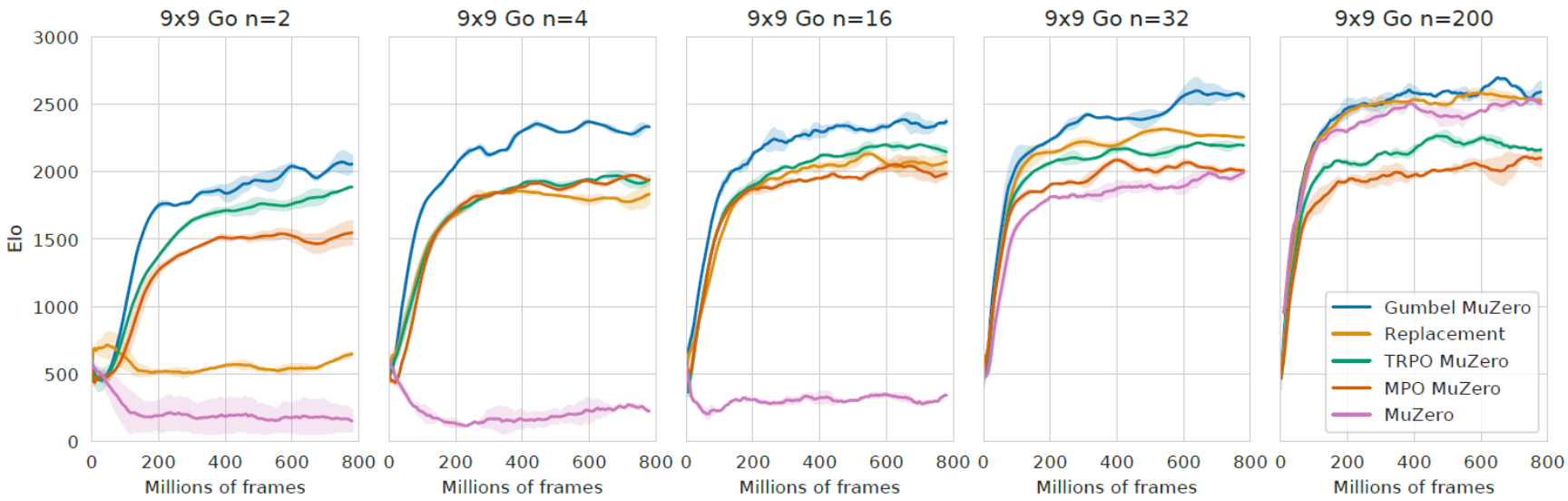
Gumbel-Top-k Trick

- The Gumbel-Max trick can be generalized to sampling *n* actions **without replacement**, by taking *n* top actions:

$$
\begin{aligned}
(g \in \mathbb{R}^k) &\sim \text{Gumbel}(0) \\
A_1 &= \arg\max_a (g(a) + \text{logits}(a)) \\
&\vdots \\
A_n &= \arg\max_{a \notin \{A_1, \ldots, A_{n-1}\}} (g(a) + \text{logits}(a))
\end{aligned}
$$

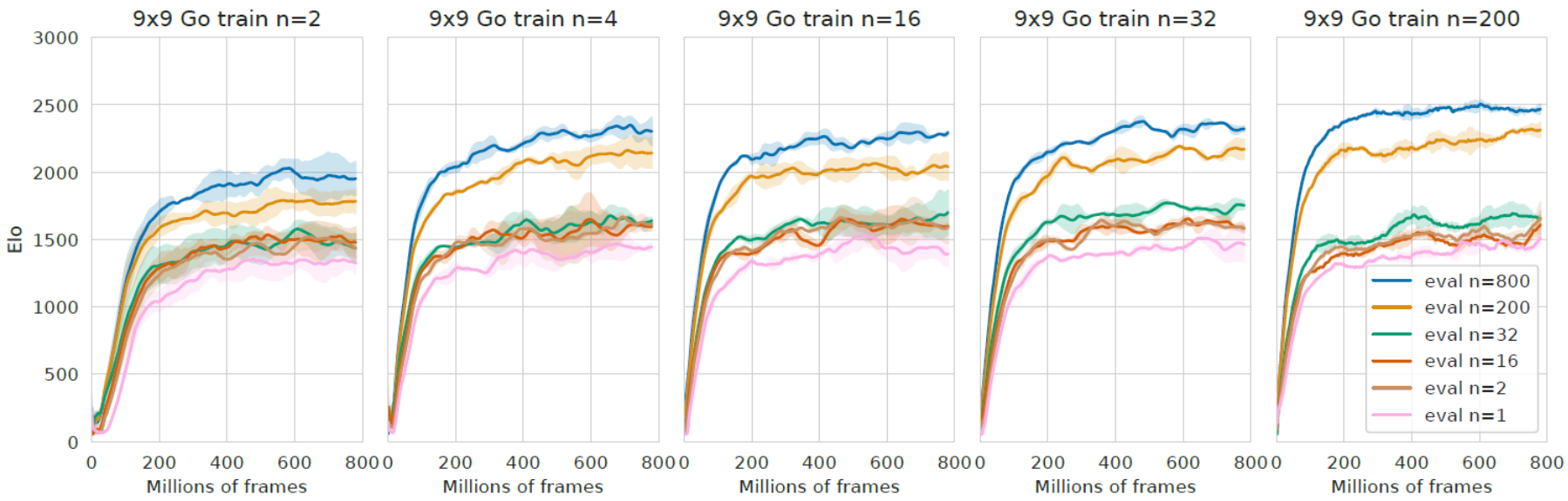
9x9 Go

- Different numbers of simulations during training



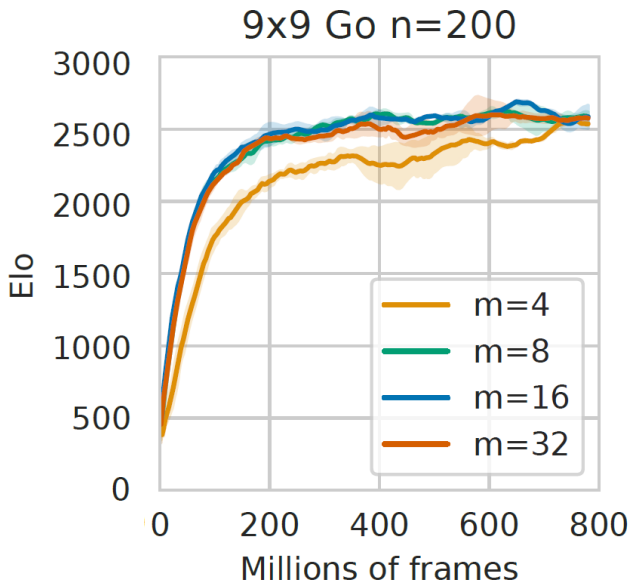
9x9 Go

- Different numbers of simulations during evaluation



9x9 Go

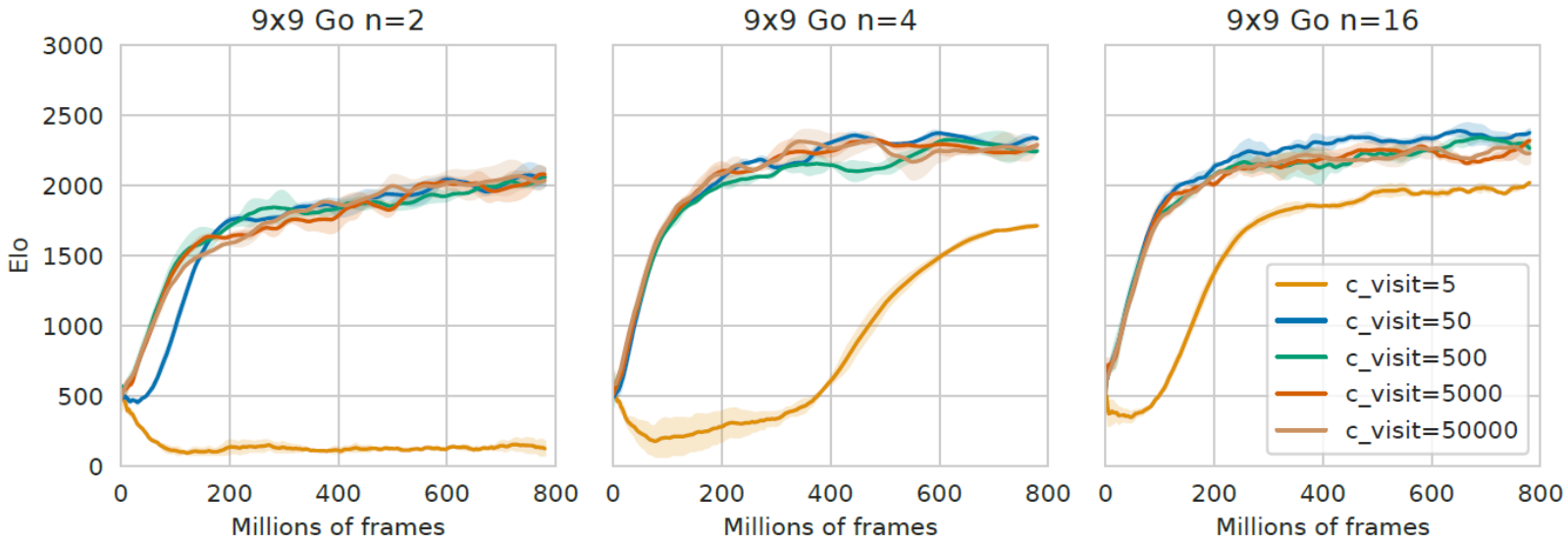
- Different numbers of sampled actions



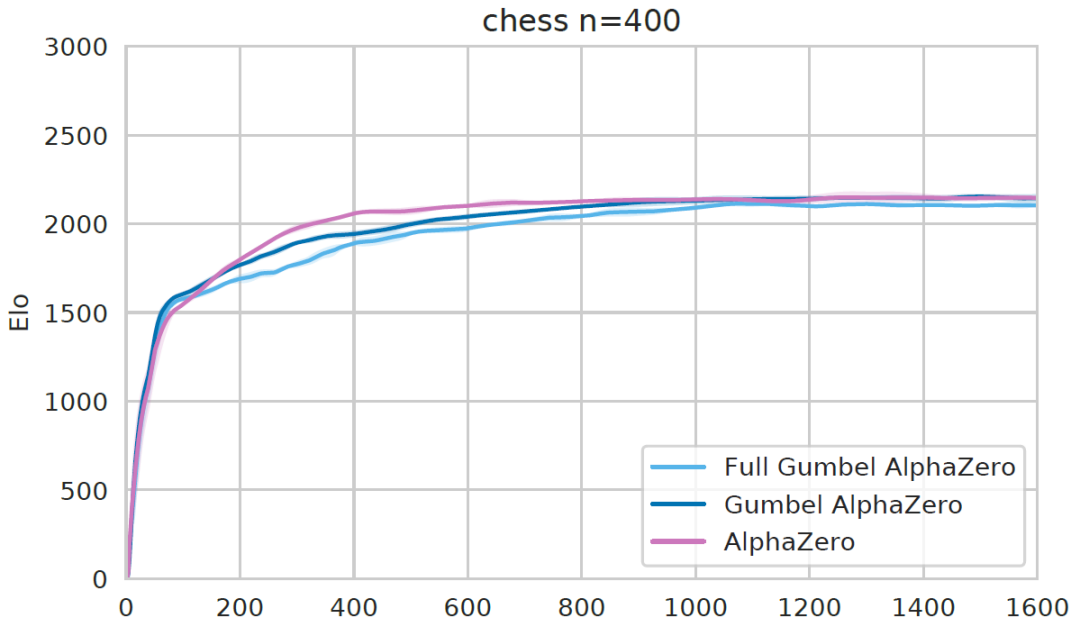
9x9 Go

$$\sigma(\hat{q}(a)) = (c_{\text{visit}} + \max_b N(b))c_{\text{scale}}\hat{q}(a)$$

- Different Q-value scaling by c_{visit}

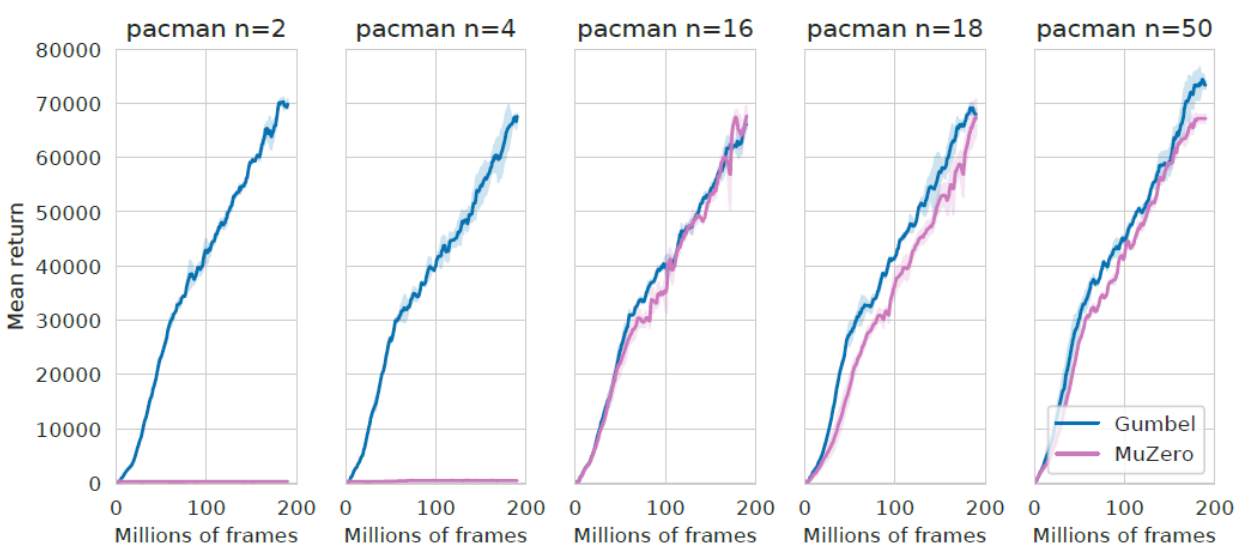


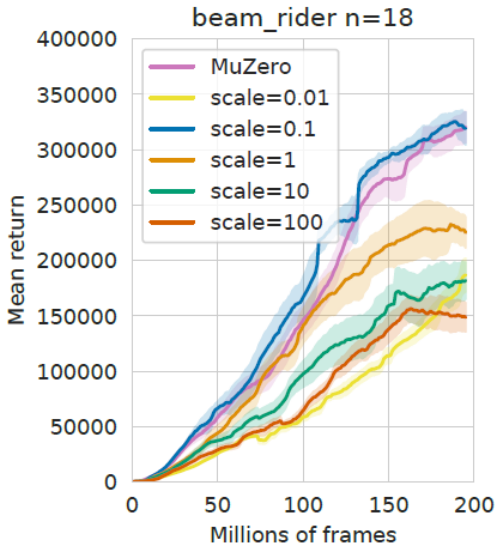
Chess


chess n=400
Elo
3000
2500
2000
1500
1000
500
0
0
200
400
600
800
1000
1200
1400
1600
Full Gumbel AlphaZero
Gumbel AlphaZero
AlphaZero

Atari

- Game: **ms_pacman** and **beam_rider**





Speedup

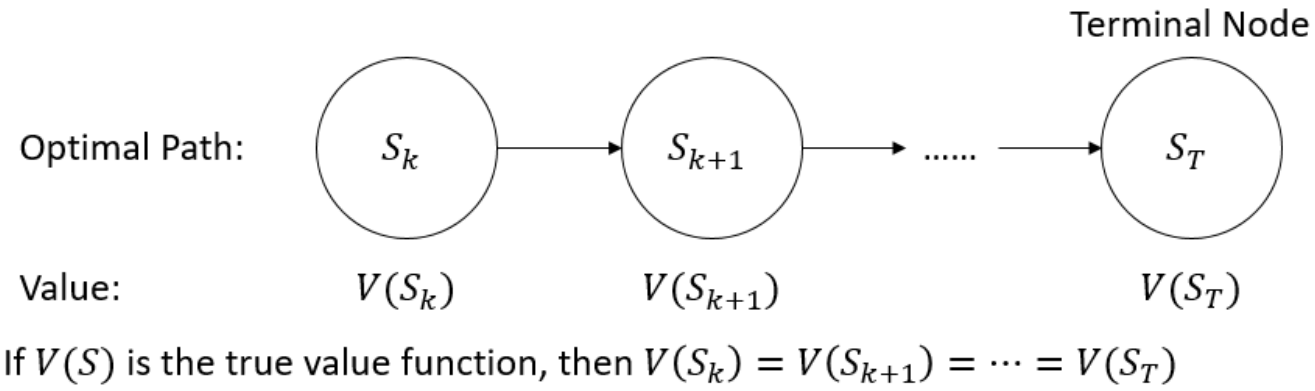
- The speedup from a smaller number of simulations on 9x9 Go

	Training step speedup
MuZero $n = 200$	1.0
Full Gumbel MuZero $n = 200$	1.0
Gumbel MuZero $n = 200$	1.0
Gumbel MuZero $n = 32$	5.9
Gumbel MuZero $n = 16$	11.3
Gumbel MuZero $n = 8$	16.2
Gumbel MuZero $n = 4$	24.3

Appendix:
Path Consistency AlphaZero (PCZero)

PCZero

- Path Consistency, i.e., ideally, values on one optimal path should be identical



If $V(S)$ is the true value function, then $V(S_k) = V(S_{k+1}) = \cdots = V(S_T)$

- As the result, our value estimate for those states should be similar.
- AlphaZero has PC (Path Consistency) phenomenon.

PCZero

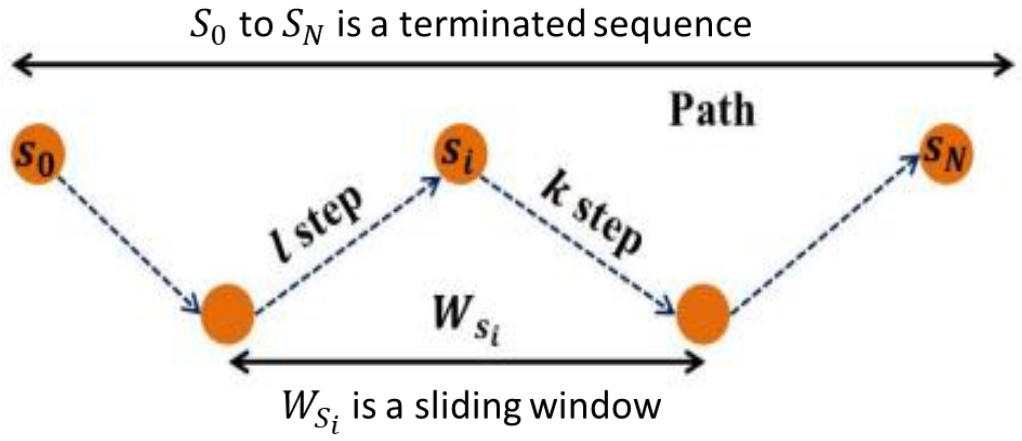
- In this paper, they propose a regularization term for PC on training loss.

$$L(\theta) = L_{RL}(\theta) + \lambda L_{PC}(\theta)$$

  - $L_{RL}(\theta)$: the RL loss, the same as AlphaZero loss target.
  - $L_{PC}(\theta)$: the PC loss, described in next slides.
- Performance result:
  - PCZero obtains 94.1% winning rate against MoHex2.0, higher than 84.3% by AlphaZero.

Method - Path Consistency

- The idea of implementing PC loss is to make adjacent state have similar value estimate.
- The goal is to make the value estimate of states within window to be similar.

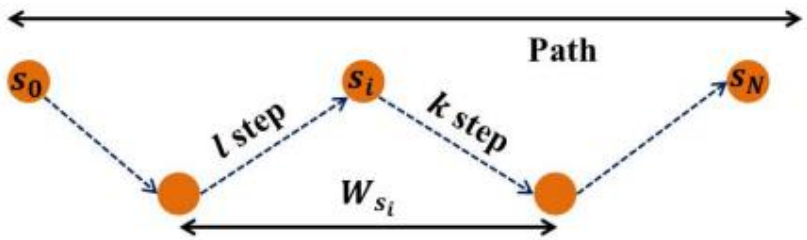


Method - Path Consistency

- Then, they calculate the PC loss for each state in sequence as following:

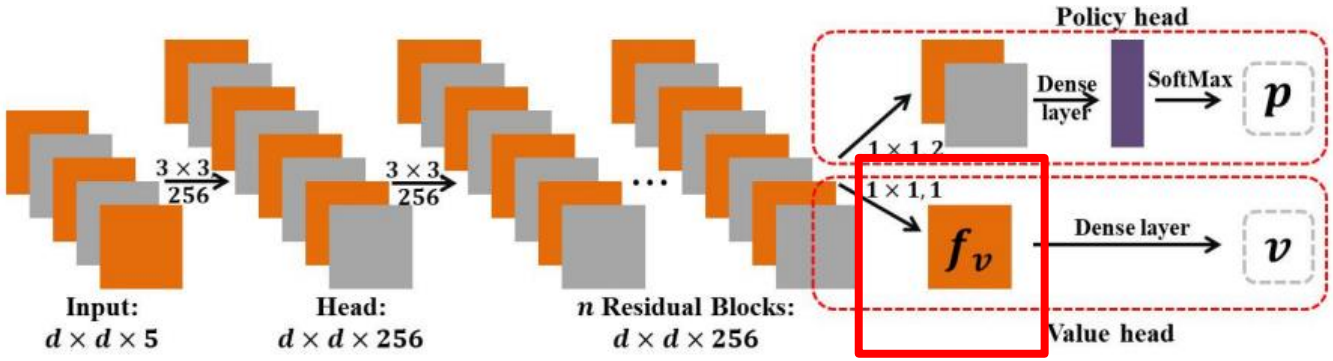
$$L_{PC}(S_i) = \left(V_\theta(S_i) - \frac{1}{(l+k+1)} \sum_{j=i-l}^{i+k} V_\theta(S_j) \right)^2$$

  - $V_\theta(S_i)$ is the state value predicted by neural network.



Method - Feature Consistency

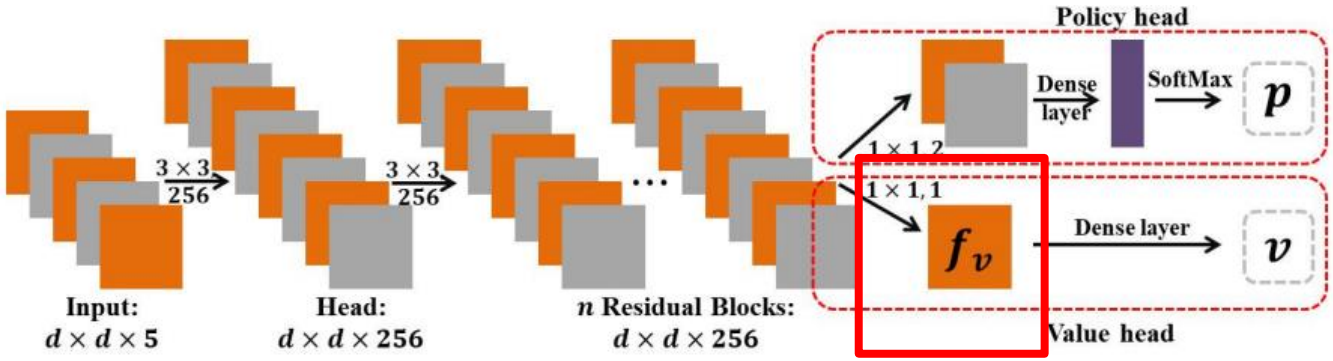
- In addition to PC, this paper also proposed Feature Consistency.
- That is, the feature map before dense layer in value head should be consistency.



feature map size: $d \times d$
d is the board size

Method - Feature Consistency

- If the feature map is similar, the output of dense layer will also be similar.
- As the result, Feature Consistency is a tighter constraint of PC.
  - Feature Consistency can be a supplementary of PC.



feature map size: $d \times d$
d is the board size

Method - Feature Consistency

- There is another regularization term to represent Feature Consistency.
  - The definition is very similar to PC (L_{PC}).

$$L_{PC}^{f}(S_i) = \left\| f_v(S_i) - \frac{1}{(l+k+1)} \sum_{j=i-l}^{k+i} f_v(S_j) \right\|^2$$

- Both PC and FC (Feature Consistency) can be added to loss target:

$$L_2 = -\pi^T \log p + (z-v)^2 + \lambda L_{PC} + \beta L_{PC}^{f} + c\|\theta\|^2,$$

PCZero Results - Online Training

- When AlphaZero has been well trained, it also has low PC loss.
- Therefore, PC loss guides the learning process towards low PC loss more efficiency.

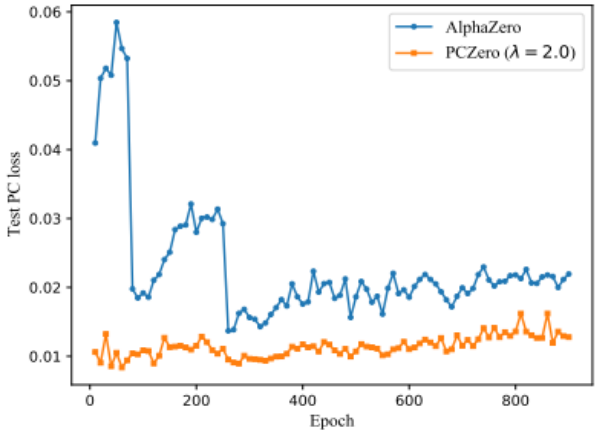


Figure 5. Test PC loss on expert dataset for online learning.

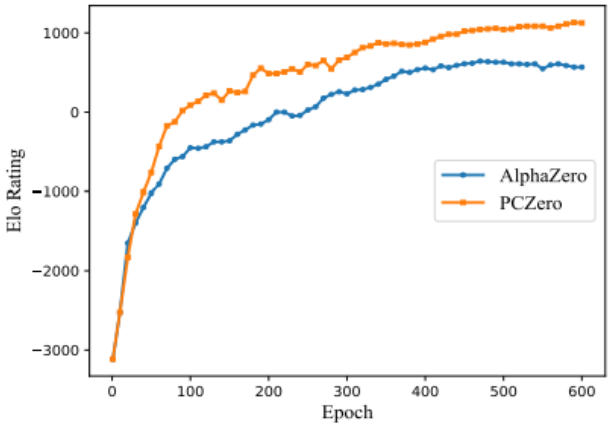


Figure 6. Elo ratings for 8×8 Hex without data sharing.

PCZero Results - Online Training

- In tournament experiment, the game they used in this experiment is Hex.
- All the agents are fight with MoHex 2.0 (baseline).

Table 1. Winning rate of models against MoHex 2.0 (10s) in 13 × 13 Hex, trained with $0.9M$ selfplay games.

MODEL	AS BLACK	AS WHITE	OVERALL
MOHEX-CNN	78.6%	61.2%	69.9%
MOHEX-3HNN	/	/	82.4%
ALPHAZERO	89.3%	79.3%	84.3%
PCZERO ($\beta = 0$)	**96.4%**	90.5%	93.5%
PCZERO	94.7%	**93.5%**	**94.1%**

PCZero without FC

PCZero with FC

PCZero Loss: $L_2 = -\pi^T \log p + (z - v)^2 + \lambda L_{PC} + \beta L^f_{PC} + c\|\theta\|^2,$

PCZero Results - Offline Training

- The offline training experiment, which compare with AlphaZero.
  - Greedy player: choose actions with maximum policy probability.
  - MCTS player: choose actions with maximum simulation count in MCTS.
- They use the expert dataset to train those model.

Table 3. Winning rate of offline PCZero against offline AlphaZero at $\lambda = 2.0, \beta = 0.0$.

GAME	GREEDY PLAYER	MCTS PLAYER	
HEX (8×8)	51.6%	58.6%	1600 simulations
HEX (9×9)	53.1%	59.9%	
HEX (13×13)	52.1%	61.5%	
OTHELLO	50.5%	80.5%	800 simulations
GOMOKU	56.8%	64.0%	

PCZero Loss: $L_2 = -\pi^T \log p + (z - v)^2 + \lambda L_{PC} + \beta L^f_{PC} + c\|\theta\|^2,$

PCZero Results - Offline Training

- The table below shows the effect of training with / without FC loss:

Table 5. Winning rate of different offline PCZero against offline AlphaZero for 13 × 13 Hex.

λ	β	GREEDY PLAYER	MCTS PLAYER
2.0	0.0	52.1%	61.5%
0.0	1.0	50.6%	50.3%
2.0	1.0	**53.3%**	**70.7%**

PCZero Loss: $L_2 = -\pi^T \log p + (z-v)^2 + \lambda L_{PC} + \beta L^f_{PC} + c\|\theta\|^2,$

Appendix: EfficientZero

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. Advances in Neural Information Processing Systems (NeurIPS-21), 34:25476–25488, 2021.

EfficientZero Introduction

- EfficientZero is built on top of MuZero Reanalyze algorithm.
- The sample efficiency
  - Problem of Muzero: The sample efficiency of MuZero is actually not very high, often requiring millions of game frames to train an Atari game AI.
  - Solution: For Atari, EfficientZero only needs 400k frames (2 hours of real-time gameplay) to surpass human level and performs comparably to DQN trained with 200 million frames.

EfficientZero Introduction

- Through their ablations, they confirm the following three issues which pose challenges to MuZero in data-limited settings and make improvements.

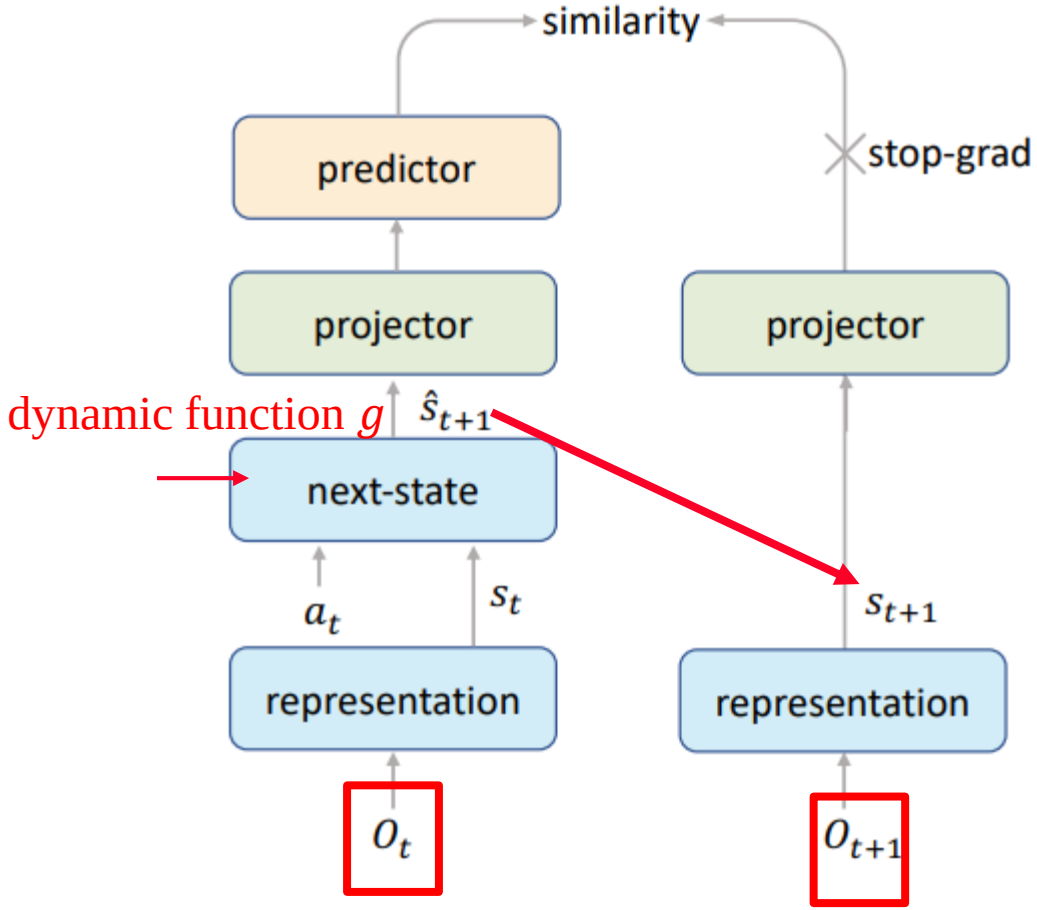
Disadvantage of MuZero on Limited Data	**Improvements to MuZero**
Lack of supervision on environment model	Self-Supervised Consistency Loss
State aliasing problem	Prediction of the Value Prefix
Off-policy issues of multi-step value	Off-Policy Correction

Lack of supervision on environment model

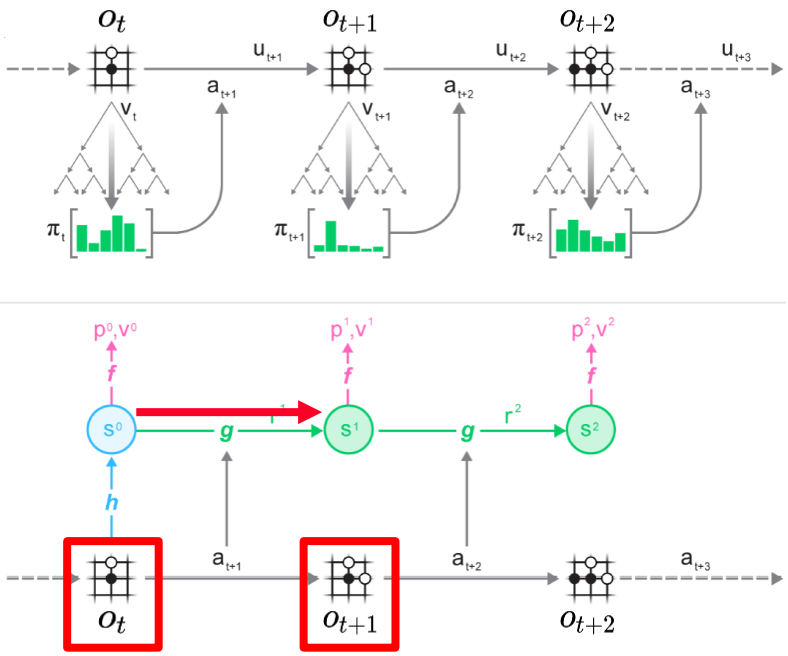
- In MuZero, the environment dynamics is only trained through the reward, value and policy functions, which cannot provide sufficient training signals.
- The problem is more severe when the reward is sparse or the bootstrapped value is not accurate.

Improvements - Self-Supervised Consistency Loss

Use the method: SimSiam



Recall Muzero:



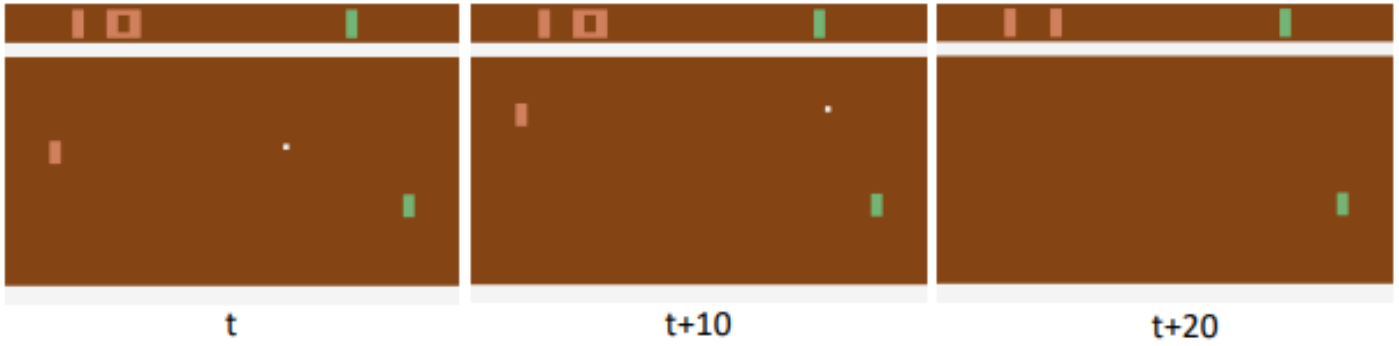
State Aliasing Problem

- Predicting the reward from an aliased state is a hard problem.

- If we only see the first observation along with future actions, it is very hard both for an agent and a human to predict at which exact future timestep the player would lose a point.

State Aliasing Problem

- Predicting the reward from an aliased state is a hard problem.

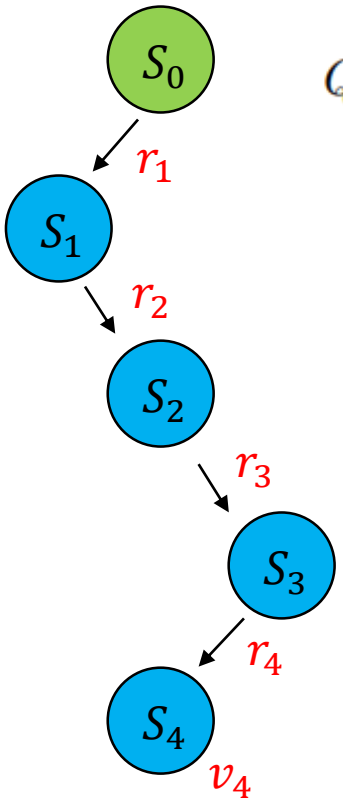


- It is easy to predict the agent will miss the ball after a sufficient number of timesteps if he does not move.
- Predict over a longer horizon gets a more confident prediction.

Improvements - Value Prefix

$$a^k = \arg\max_a \left[Q(s,a) + P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \left(c_1 + \log\left(\frac{\sum_b N(s,b) + c_2 + 1}{c_2} \right) \right) \right]$$

MuZero:



$$Q(s_t, a) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k v_{t+k}$$

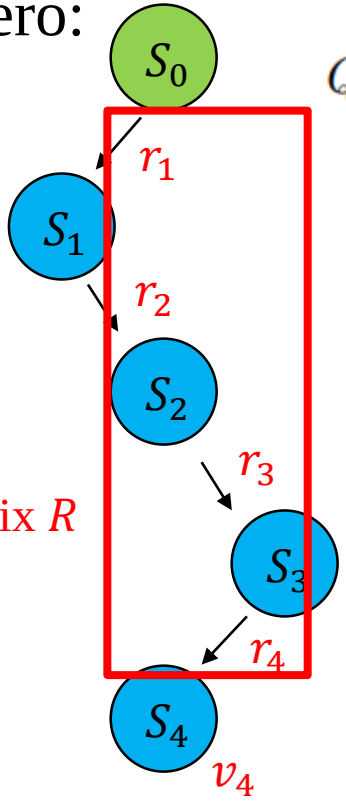
During backprop:

$$Q(S_0, a) = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \gamma^4 v_4$$

Improvements - Value Prefix

$$a^k = \arg\max_a \left[Q(s,a) + P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \left(c_1 + \log\left(\frac{\sum_b N(s,b) + c_2 + 1}{c_2} \right) \right) \right]$$

EfficientZero:



$$Q(s_t, a) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k v_{t+k}$$

Value Prefix

Predict
a value prefix R

During backprop:

$$Q(S_0, a) = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \gamma^4 v_4$$

Value Prefix $= R = f(S_t, \hat{S}_{t+1}, \ldots, \hat{S}_{t+k-1})$

f is a LSTM that takes in a variable number of inputs and outputs a scalar.

Off-policy Issues of Multi-step Value

- In MuZero, the value target is computed by sampling a trajectory from the replay buffer and computing:

$$z_t = \sum_{i=0}^{k-1} \gamma^i u_{t+i} + \gamma^k v_{t+k}$$

- The trajectory is rolled out using an older policy, and thus the value target is no longer accurate.

Improvements - Off-Policy Correction

MuZero: $z_t = \sum_{i=0}^{k-1} \gamma^i u_{t+i} + \gamma^k v_{t+k}$

EfficientZero: $z_t = \sum_{i=0}^{l-1} \gamma^i u_{t+i} + \boxed{\gamma^l \nu_{t+l}^{\text{MCTS}}}$

- The root value of the MCTS tree expanded from s_{t+l} with the **current** policy (latest network parameter).
  - Reanalyze
- $l \leq k$ and l should be smaller if the trajectory is older.

EfficientZero Experiments

Atari 100k benchmark:
Contains 26 Atari games

Game	Random	Human	SimPLe	OTRainbow	CURL	DrQ	SPR	MuZero	Ours
Alien	227.8	7127.7	616.9	**824.7**	558.2	771.2	801.5	530.0	808.5
Amidar	5.8	1719.5	88.0	82.8	142.1	102.8	**176.3**	38.8	148.6
Assault	222.4	742.0	527.2	351.9	600.6	452.4	571.0	500.1	**1263.1**
Asterix	210.0	8503.3	1128.3	628.5	734.5	603.5	977.8	1734.0	**25557.8**
Bank Heist	14.2	753.1	34.2	182.1	131.6	168.9	**380.9**	192.5	351.0
BattleZone	2360.0	37187.5	5184.4	4060.6	14870.0	12954.0	**16651.0**	7687.5	13871.2
Boxing	0.1	12.1	9.1	2.5	1.2	6.0	35.8	15.1	**52.7**
Breakout	1.7	30.5	16.4	9.8	4.9	16.1	17.1	48.0	**414.1**
ChopperCmd	811.0	7387.8	1246.9	1033.3	1058.5	780.3	974.8	**1350.0**	1117.3
Crazy Climber	10780.5	35829.4	62583.6	21327.8	12146.5	20516.5	42923.6	56937.0	**83940.2**
Demon Attack	152.1	1971.0	208.1	711.8	817.6	1113.4	545.2	3527.0	**13003.9**
Freeway	0.0	29.6	20.3	25.0	**26.7**	9.8	24.4	21.8	21.8
Frostbite	65.2	4334.7	254.7	231.6	1181.3	331.1	**1821.5**	255.0	296.3
Gopher	257.6	2412.5	771.0	778.0	669.3	636.3	715.2	1256.0	**3260.3**
Hero	1027.0	30826.4	2656.6	6458.8	6279.3	3736.3	7019.2	3095.0	**9315.9**
Jamesbond	29.0	302.8	125.3	112.3	471.0	236.0	365.4	87.5	**517.0**
Kangaroo	52.0	3035.0	323.1	605.4	872.5	940.6	**3276.4**	62.5	724.1
Krull	1598.0	2665.5	4539.9	3277.9	4229.6	4018.1	3688.9	4890.8	**5663.3**
Kung Fu Master	258.5	22736.3	17257.2	5722.2	14307.8	9111.0	13192.7	18813.0	**30944.8**
Ms Pacman	307.3	6951.6	**1480.0**	941.9	1465.5	960.5	1313.2	1265.6	1281.2
Pong	-20.7	14.6	12.8	1.3	-16.5	-8.5	-5.9	-6.7	**20.1**
Private Eye	24.9	69571.3	58.3	100.0	**218.4**	-13.6	124.0	56.3	96.7
Qbert	163.9	13455.0	1288.8	509.3	1042.4	854.4	669.1	3952.0	**13781.9**
Road Runner	11.5	7845.0	5640.6	2696.7	5661.0	8895.1	14220.5	2500.0	**17751.3**
Seaquest	68.4	42054.7	683.3	286.9	384.5	301.2	583.1	208.0	**1100.2**
Up N Down	533.4	11693.2	3350.3	2847.6	2955.2	3180.8	**28138.5**	2896.9	17264.2
Normed Mean	0.000	1.000	0.443	0.264	0.381	0.357	0.704	0.562	**1.943**
Normed Median	0.000	1.000	0.144	0.204	0.175	0.268	0.415	0.227	**1.090**

EfficientZero Experiments

DMControl 100k benchmark:

Task	CURL	Dreamer	MuZero	SAC-AE	Pixel SAC	State SAC	EfficientZero
Cartpole, Swingup	582± 146	326±27	218.5 ± 122	311±11	419±40	835±22	**813±19**
Reacher, Easy	538± 233	314±155	493 ± 145	274±14	145±30	746±25	**952±34**
Ball in cup, Catch	769± 43	246 ± 174	542 ± 270	391± 82	312± 63	746±91	**942±17**