Recent Advances in Deep Reinforcement Learning

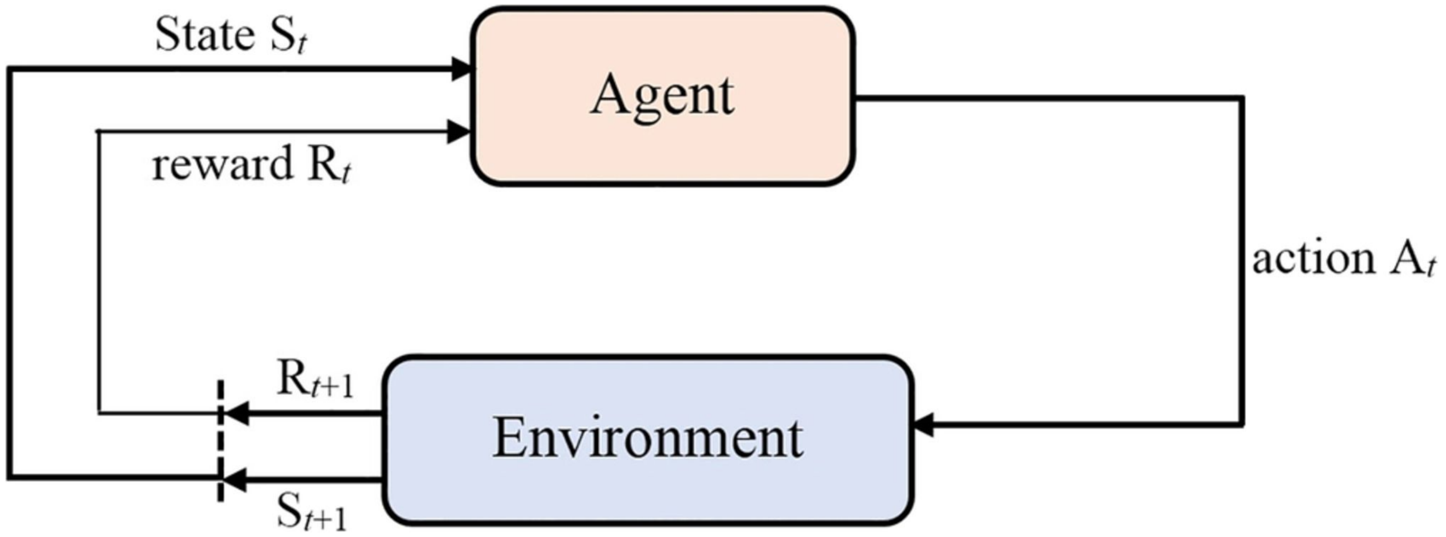
Arlindo Oliveira
Alexandre Borges

Outline

1. Reinforcement Learning
2. Deep Reinforcement Learning
3. Tree Search
4. AlphaGo to MuZero
5. One way of improving MuZero

Reinforcement Learning

- Sequential decision problem where an agent interacts with an environment



How do we solve these problems?

We can do this by learning one or more of following:

1. A value function V(s) or Q(s,a) that evaluates how good a state is

How do we solve these problems?

We can do this by learning one or more of following:

1. A value function $V(s)$ or $Q(s,a)$ that evaluates how good a state is
2. A policy $\pi(s)$ that is a mapping from a state to a action

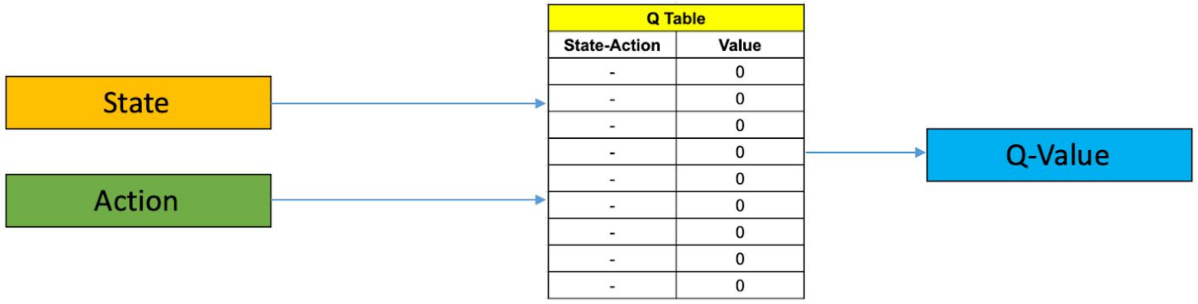
How do we solve these problems?

We can do this by learning one or more of following:

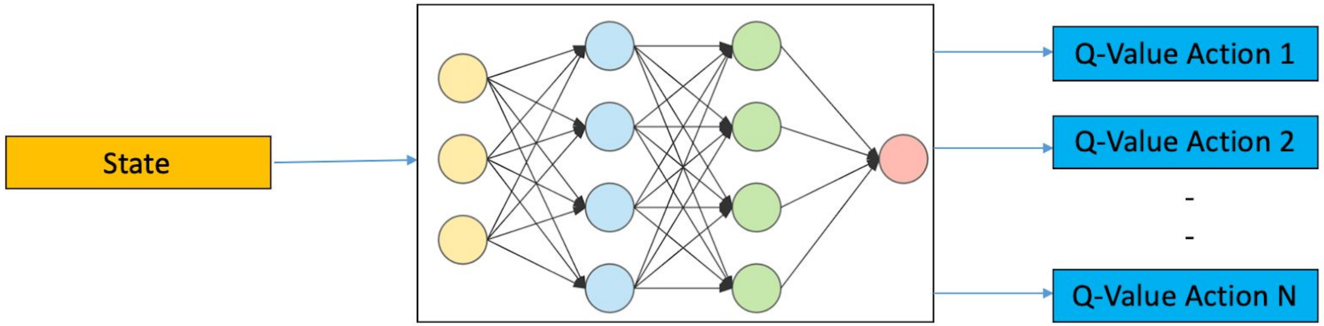
1. A value function $V(s)$ or $Q(s,a)$ that evaluates how good a state is
2. A policy $\pi(s)$ that is a mapping from a state to a action
3. A model of the environment then use planning algorithms

Deep Reinforcement Learning

We can use a neural networks for these learning objectives



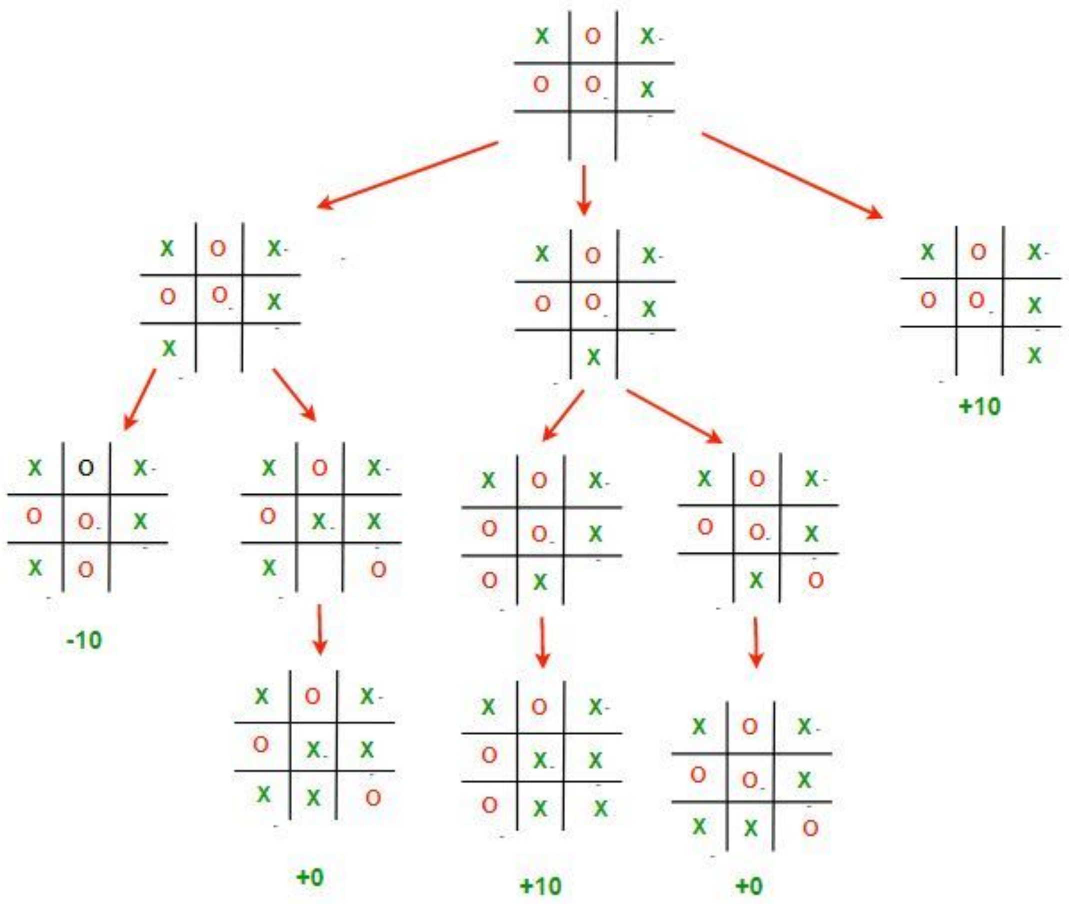
Q Learning



Deep Q Learning

Board Games and Tree Search

Tree search
- to explore the state space
- obtain a value for a state

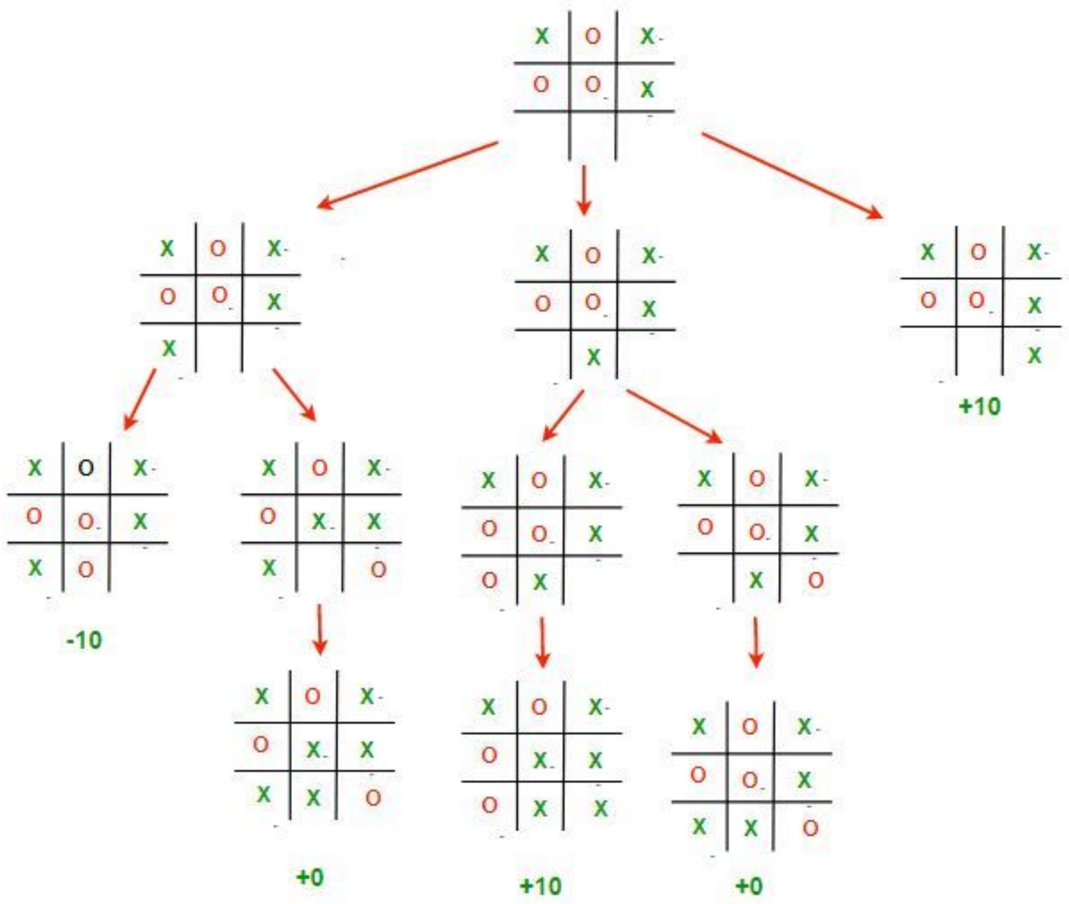


Board Games and Tree Search

Tree search

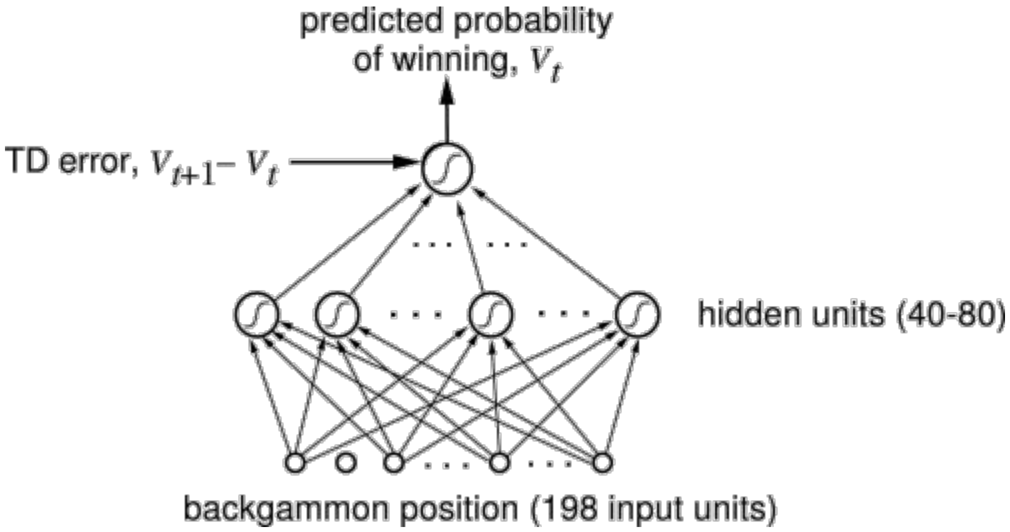
- to explore the state space
- obtain a value for a state

However, the computation cost is to high for more complex games

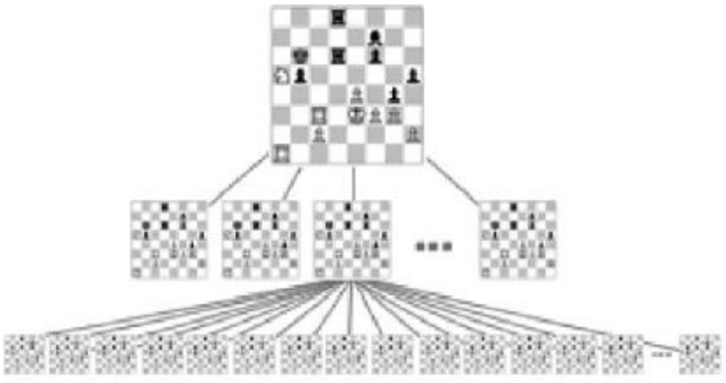


Reducing state space

TD-Gammon:
- Super Human performance in Gammon
- Combined tree search with neural networks



DeepBlue:
- Beat the word chess champion
- Combining tree search guided by heuristics

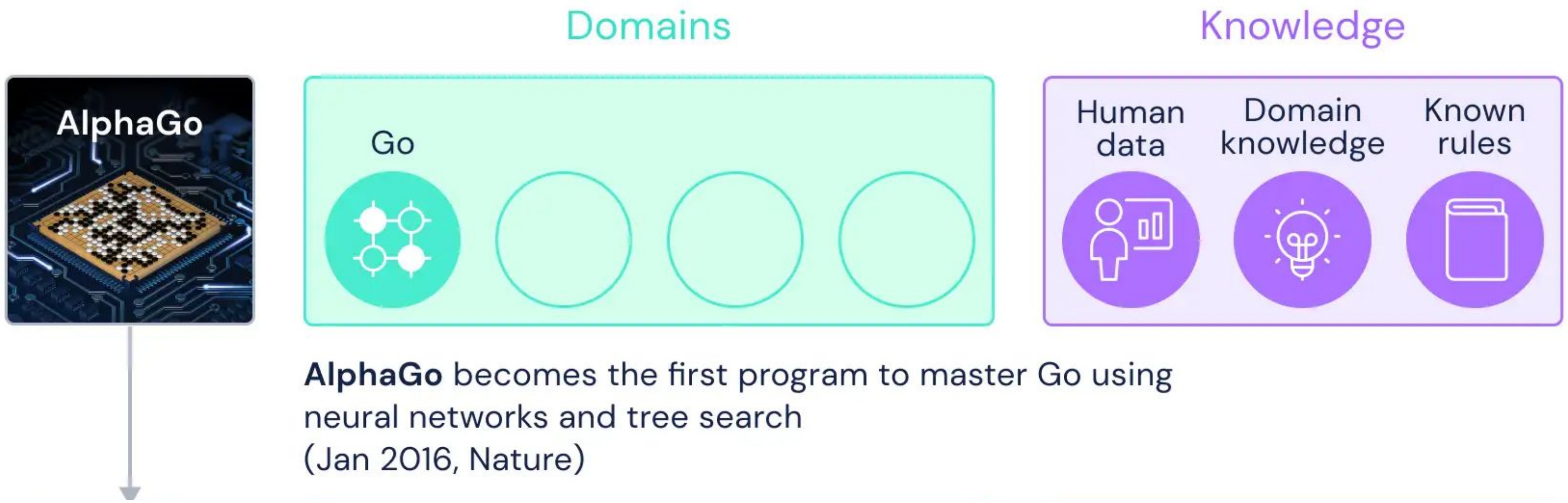


Go

- Game with long matches and high branching factor
- Search space state is huge

- None of the methods mentioned above are able to obtain super human performance

AlphaGo

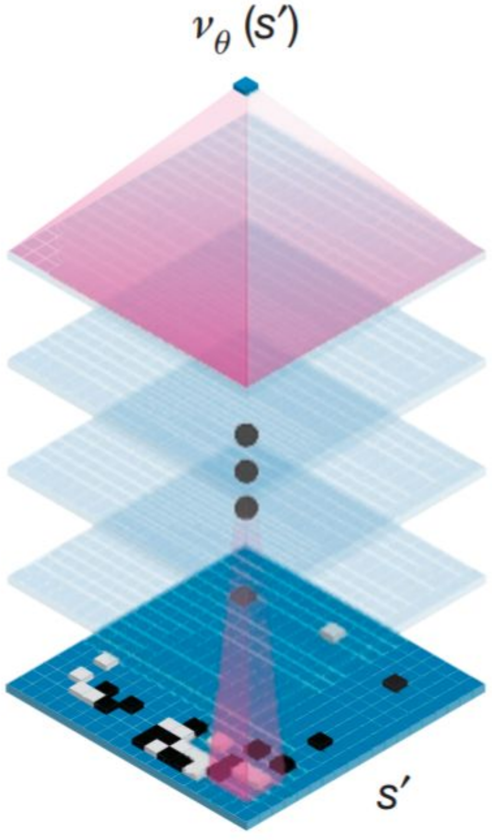


AlphaGo becomes the first program to master Go using neural networks and tree search
(Jan 2016, Nature)

Networks

AlphaGo uses two convolutional networks that they combine with tree search:

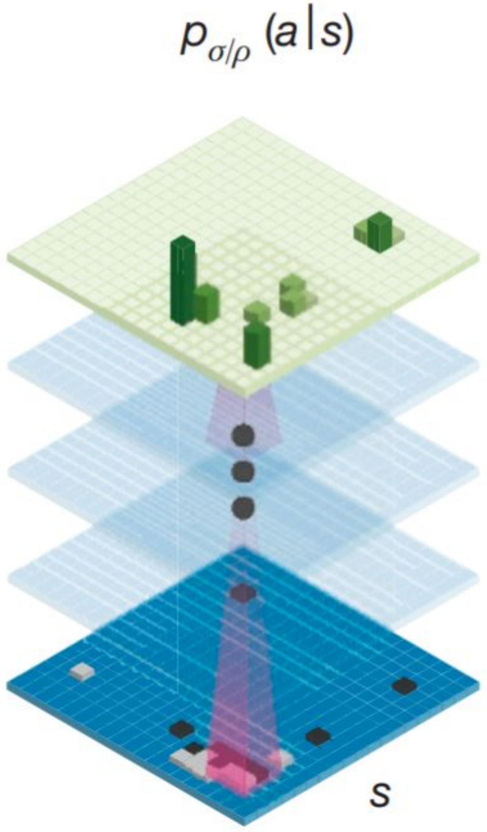
1. **Value Network**
    a. Reduces Depth
    b. Outputs a value in]-1,1[



Networks

AlphaGo uses two convolutional networks that they combine with tree search:

1. **Value Network**
   a. Reduces Depth
   b. Outputs a value in]-1,1[
2. **Policy Network**
   a. Reduces Breadth
   b. Outputs a probability over all possible actions



Training

Training in AlphaGo can be split into three phases:

1. Supervised learning of policy networks
    a. Using data from professional games

Training

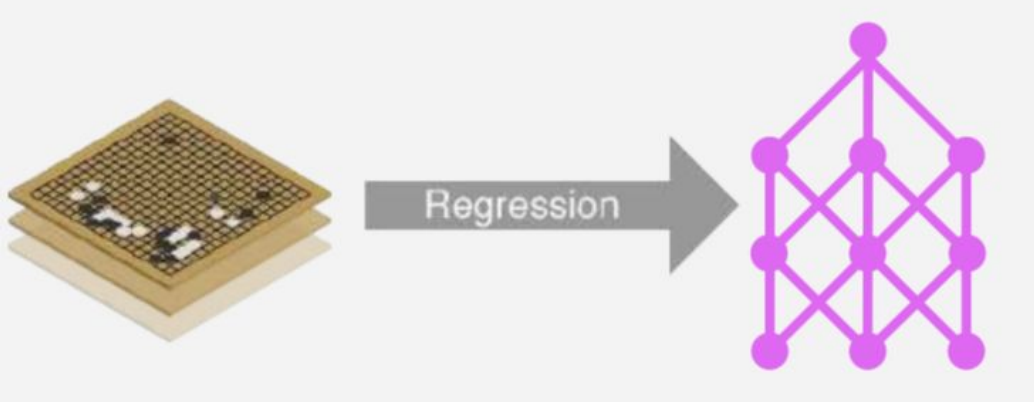
Training in AlphaGo can be split into three phases:

1. Supervised learning of policy networks
2. Reinforcement learning of policy networks
    a. Policy Gradient

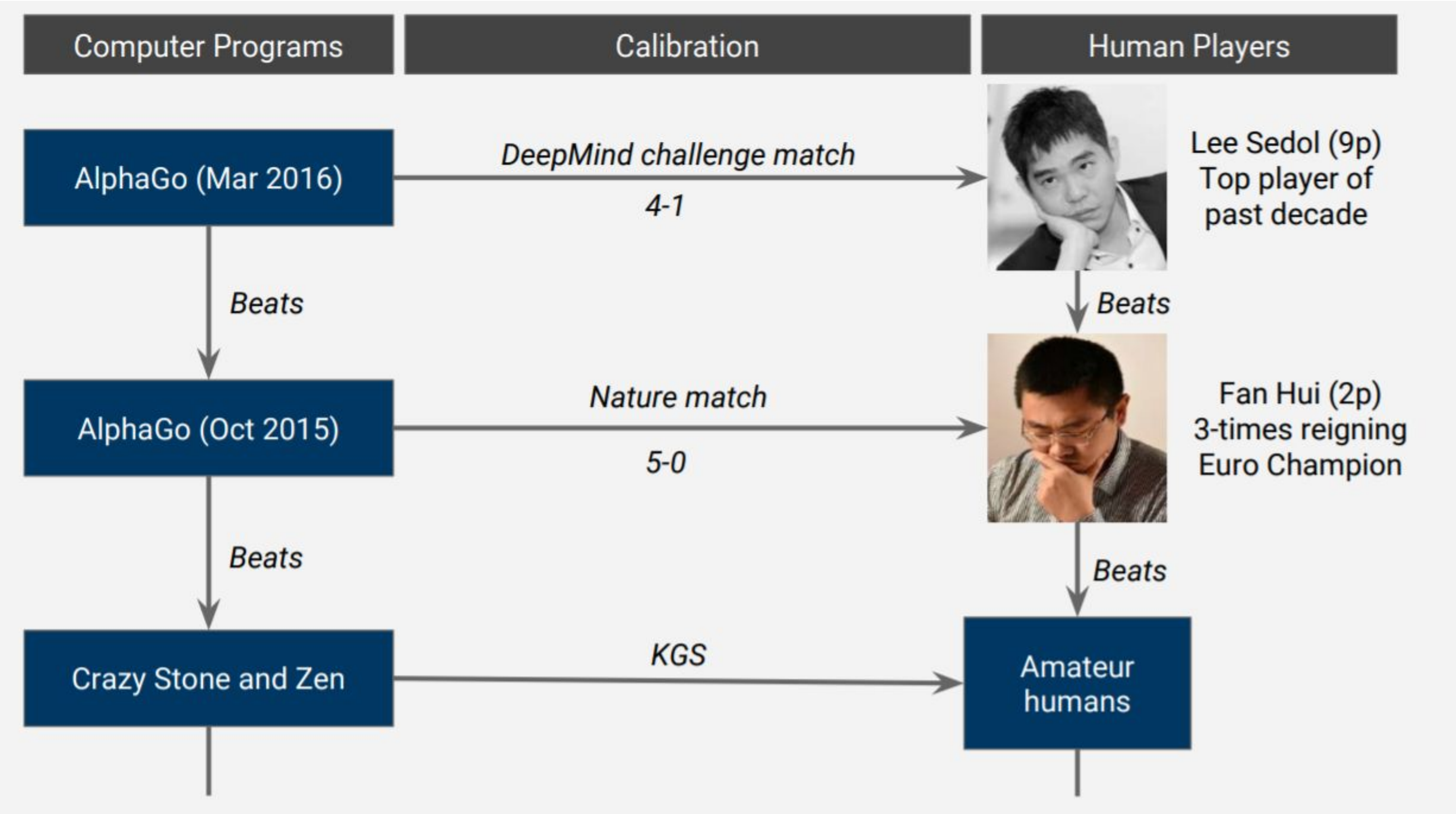
Training

Training in AlphaGo can be split into three phases:

1. Supervised learning of policy networks
2. Reinforcement learning of policy networks
3. Reinforcement learning of value networks

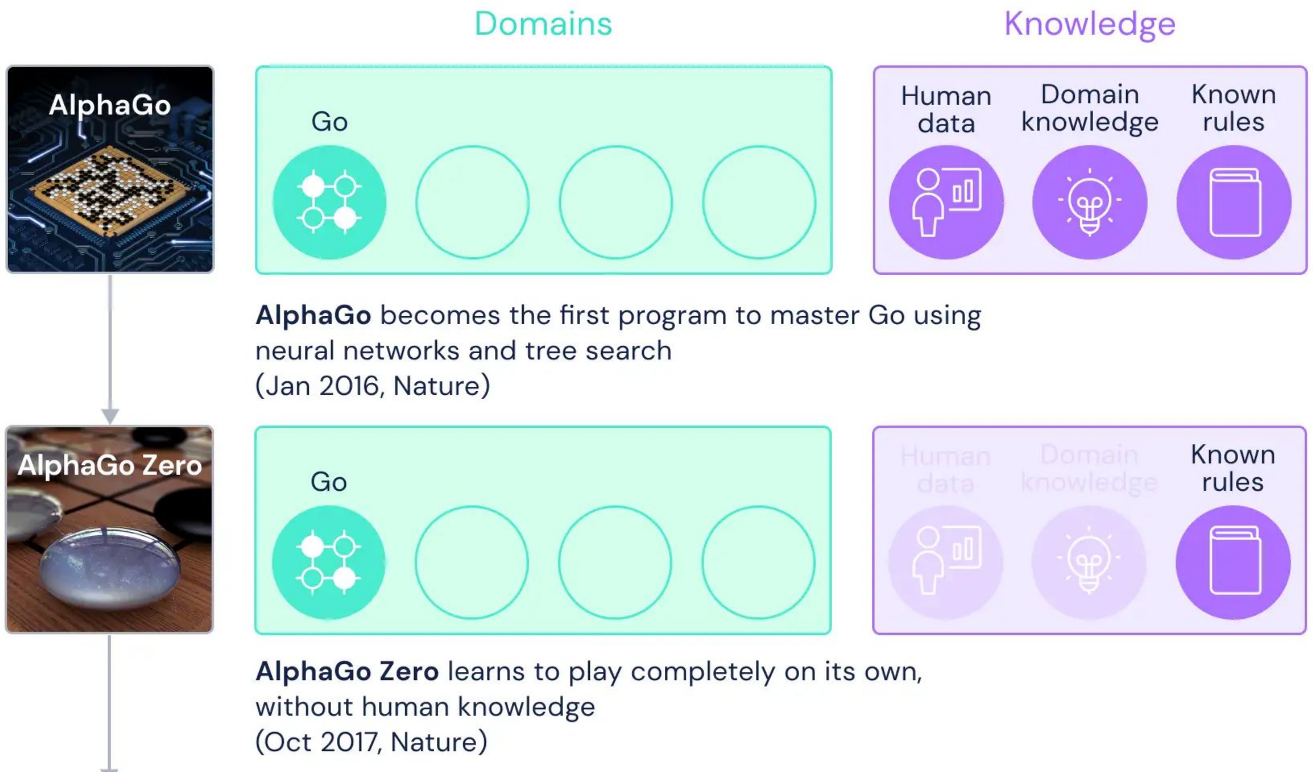


Results



Computer Programs
Calibration
Human Players
AlphaGo (Mar 2016)
DeepMind challenge match
4-1
Lee Sedol (9p)
Top player of
past decade
Beats
Beats
AlphaGo (Oct 2015)
Nature match
5-0
Fan Hui (2p)
3-times reigning
Euro Champion
Beats
Beats
Crazy Stone and Zen
KGS
Amateur
humans

AlphaGoZero



Differences between AlphaGo and AlphaGoZero

AlphaGo

1. Two separate networks
   a. Policy Network
   b. Value network

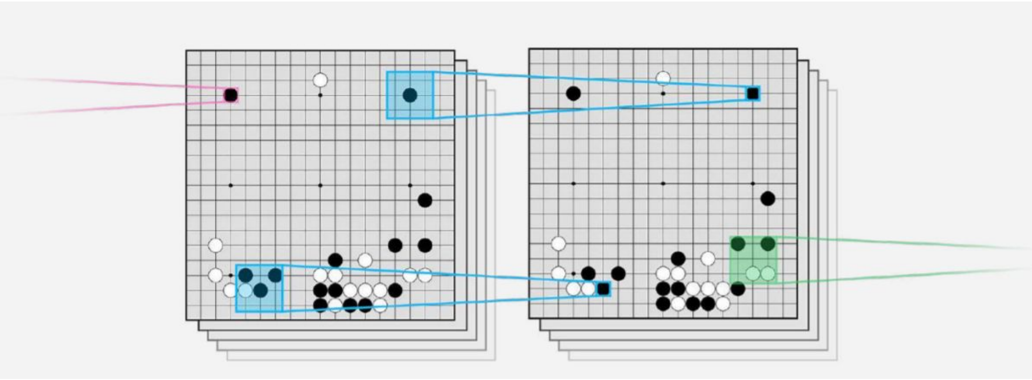
AlphaGoZero

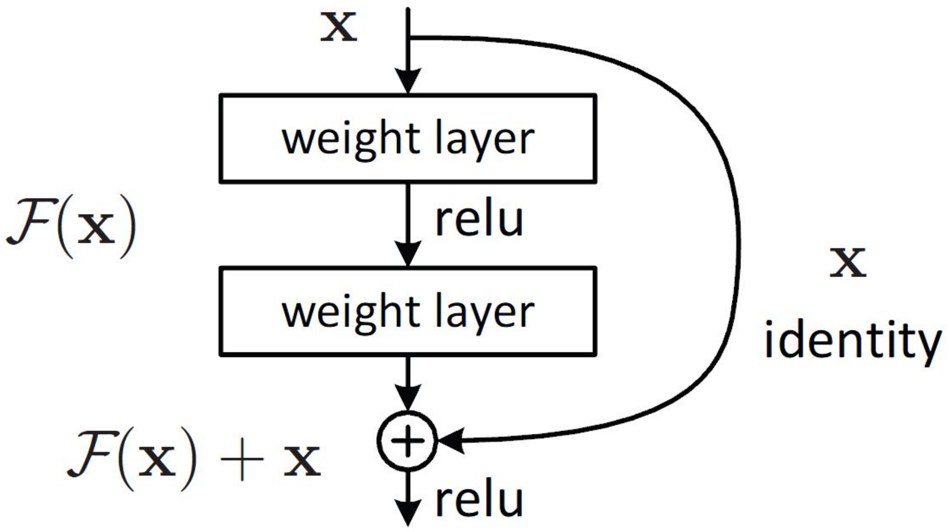
1. Only one network with two heads
   a. Policy Head
   b. Value Head

Differences between AlphaGo and AlphaGoZero

AlphaGo
1. Two separate networks
2. Uses convolutional networks

AlphaGoZero
1. Only one network with two heads
2. Uses convolutional residual networks





Differences between AlphaGo and AlphaGoZero

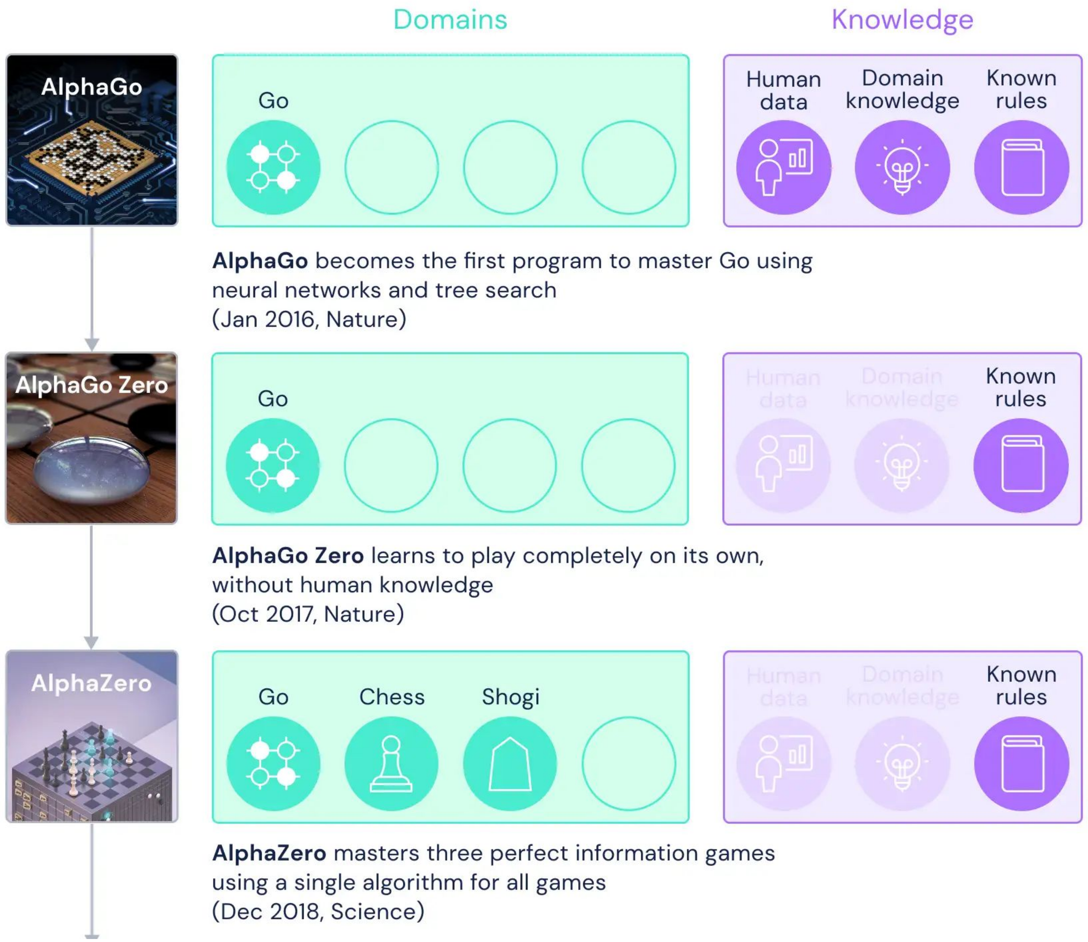
AlphaGo

1. Two separate networks
2. Uses convolutional networks
3. Supervised and reinforcement learning

AlphaGoZero

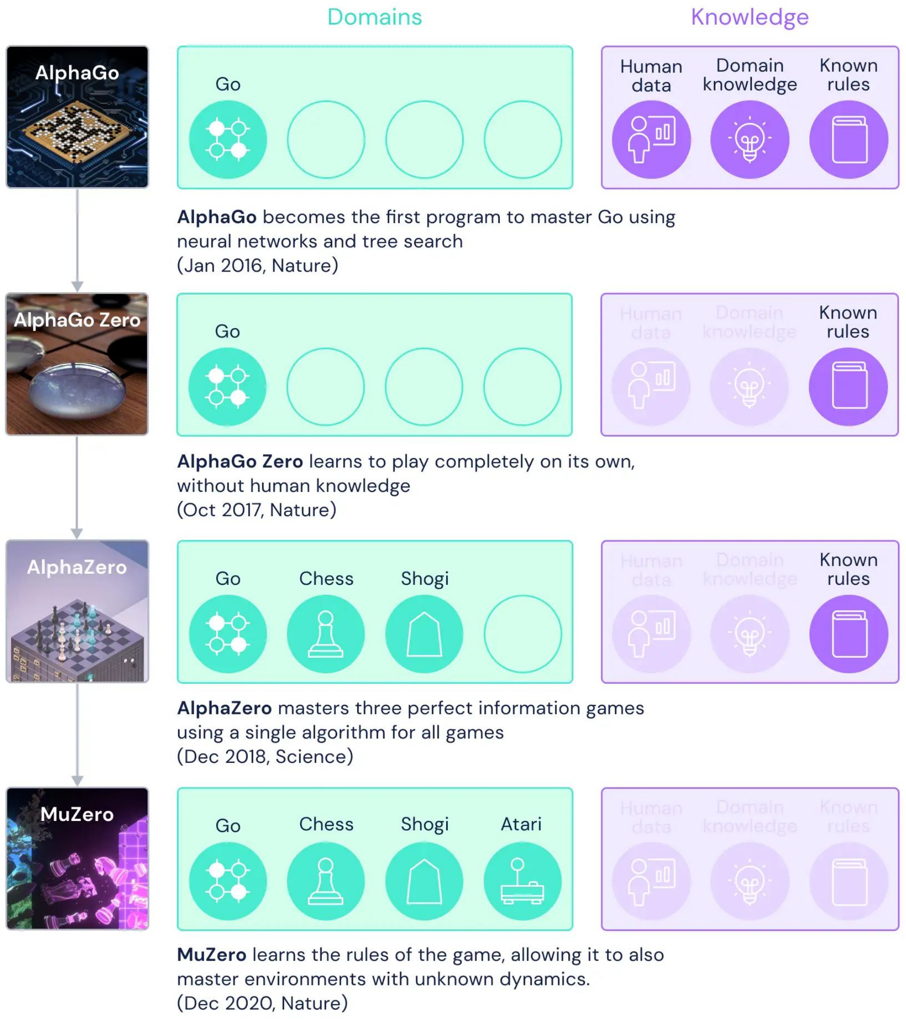
1. Only one network with two heads
2. Uses convolutional residual networks
3. Only Reinforcement learning through self-play
    a. No human knowledge used

AlphaZero



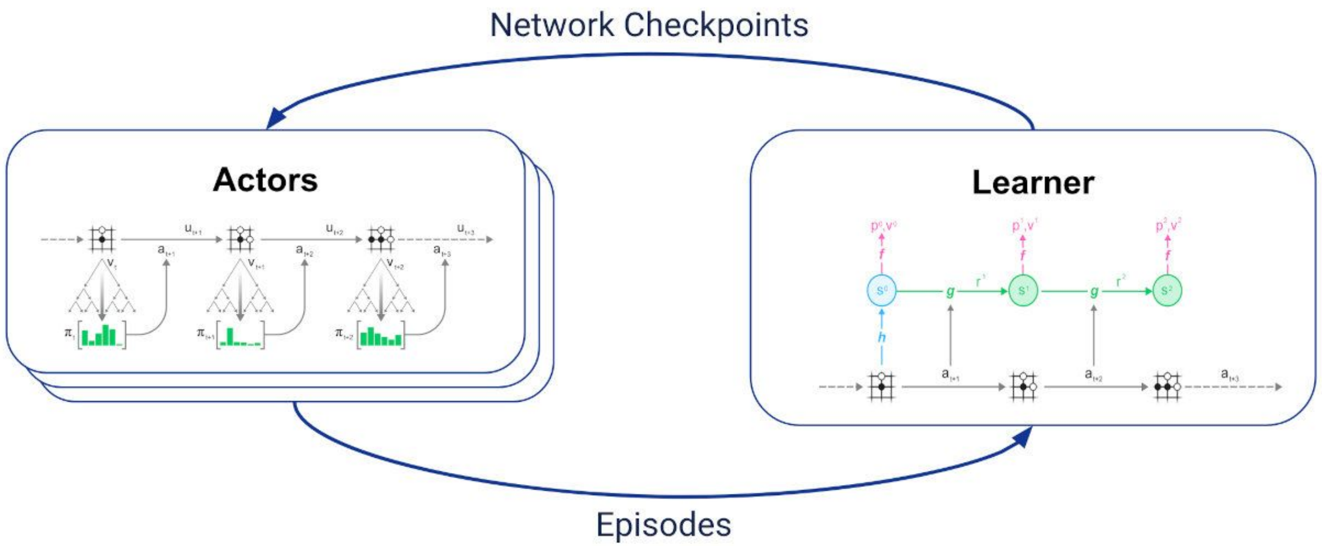
Domains
Knowledge
AlphaGo
Go
Human data
Domain knowledge
Known rules
AlphaGo becomes the first program to master Go using neural networks and tree search
(Jan 2016, Nature)
AlphaGo Zero
Go
Human data
Domain knowledge
Known rules
AlphaGo Zero learns to play completely on its own, without human knowledge
(Oct 2017, Nature)
AlphaZero
Go
Chess
Shogi
Human data
Domain knowledge
Known rules
AlphaZero masters three perfect information games using a single algorithm for all games
(Dec 2018, Science)

MuZero


Domains
Knowledge
AlphaGo
Go
Human data
Domain knowledge
Known rules
AlphaGo becomes the first program to master Go using neural networks and tree search (Jan 2016, Nature)
AlphaGo Zero
Go
Human data
Domain knowledge
Known rules
AlphaGo Zero learns to play completely on its own, without human knowledge (Oct 2017, Nature)
AlphaZero
Go
Chess
Shogi
Human data
Domain knowledge
Known rules
AlphaZero masters three perfect information games using a single algorithm for all games (Dec 2018, Science)
MuZero
Go
Chess
Shogi
Atari
Human data
Domain knowledge
Known rules
MuZero learns the rules of the game, allowing it to also master environments with unknown dynamics. (Dec 2020, Nature)

MuZero

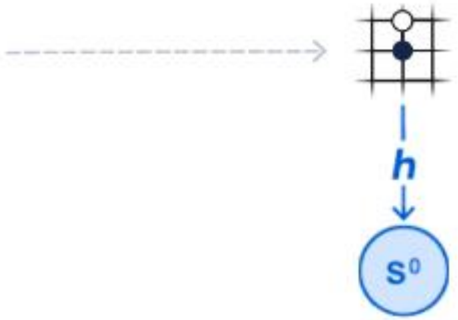
- Learns a model of the environment



A model of the environment

- Network composed of three functions:
  a. **Representation Function:**

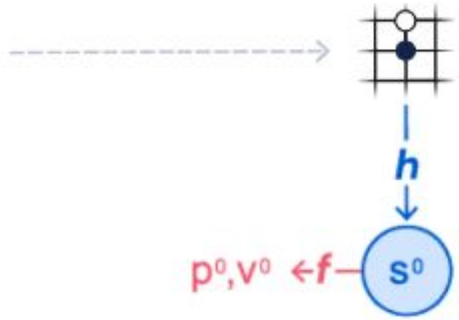
$$h_\theta(o_1, \ldots, o_t) = s^t$$



A model of the environment

- Network composed of three functions:
  a. **Representation Function:**
  b. **Prediction Function:**

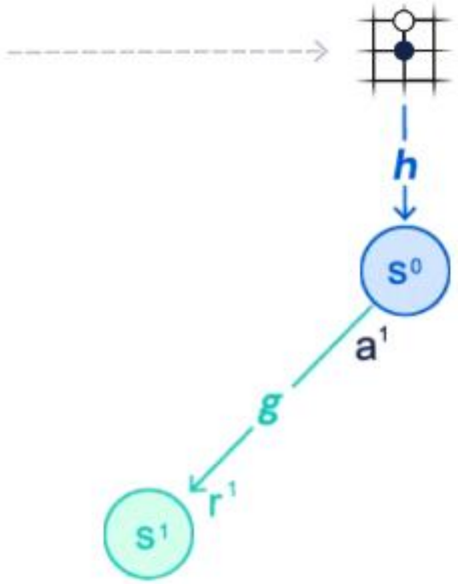
$$f_\theta(s^k) = p^k, v^k$$



A model of the environment

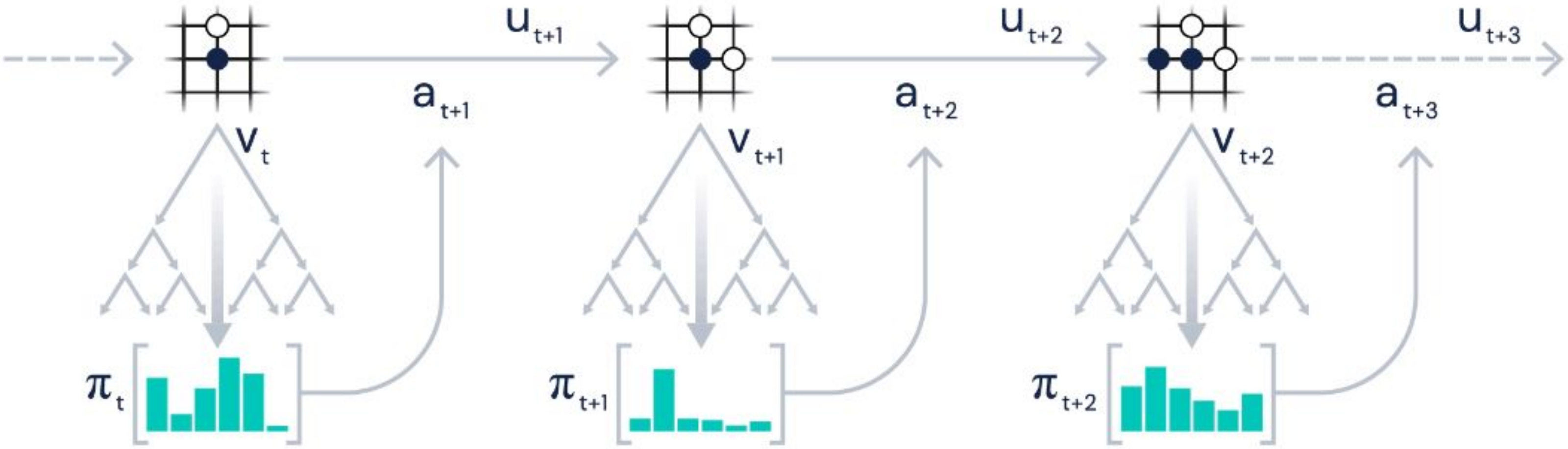
- Network composed of three functions:
  a. **Representation Function:**
  b. **Prediction Function:**
  c. **Dynamics Function:**

$$g_\theta(s^{k-1}, a^k) = r^k, s^k$$



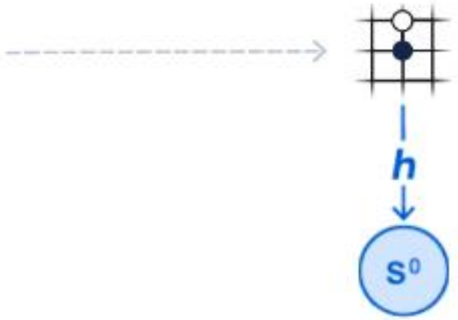
MuZero

- Learns a model of the environment
- Combines the model with MCTS to play



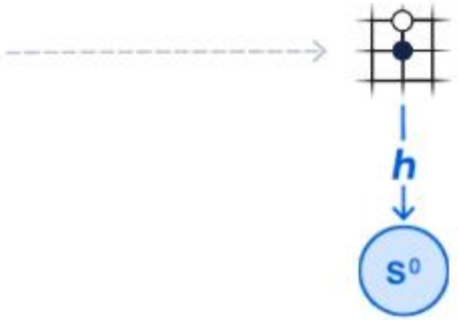
Monte Carlo Tree Search

- Get the hidden state for the current observation



Monte Carlo Tree Search

- We perform several simulations, for each simulation we:



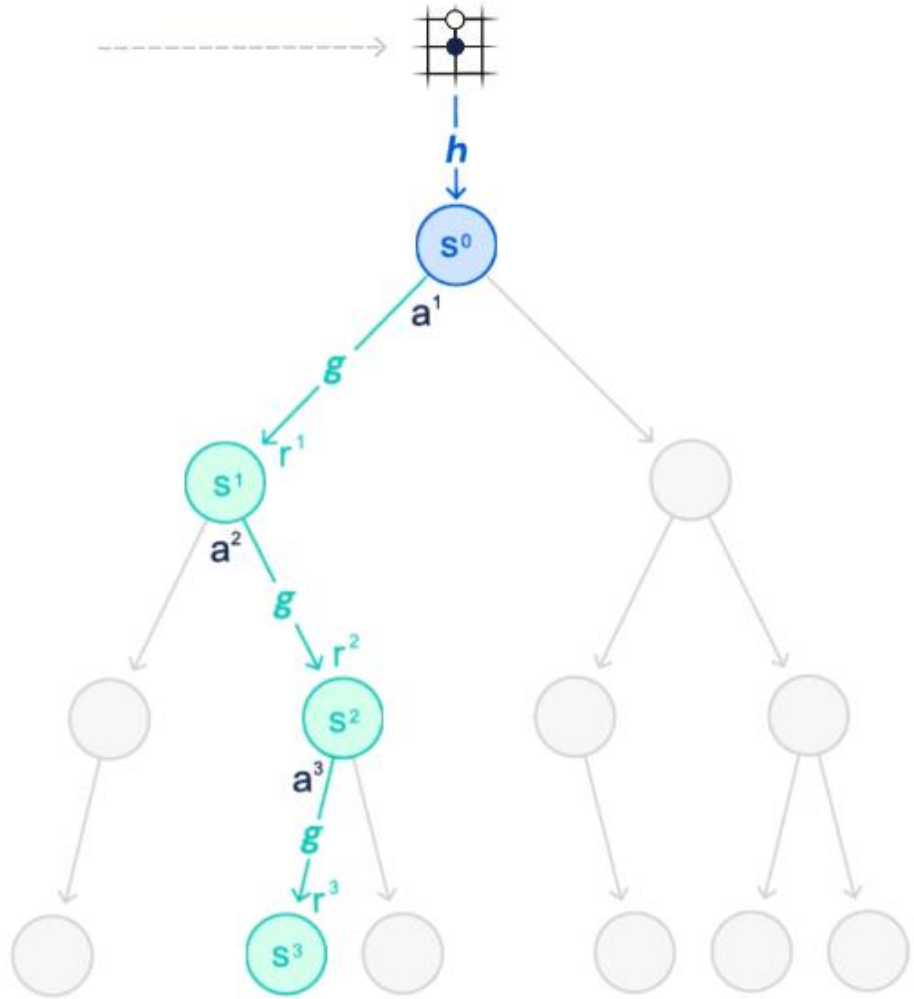
Monte Carlo Tree Search

- We perform several simulations, for each simulation we:

    Select: action according to tree statistics, until we reach a leaf node

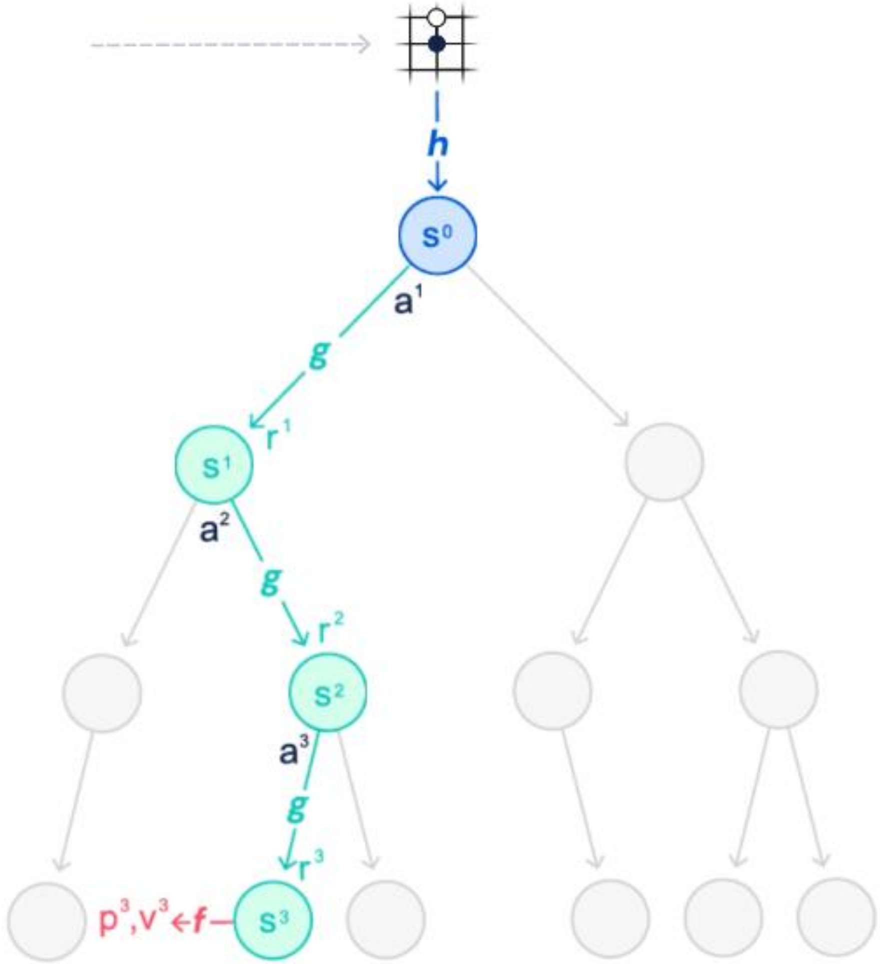
$$a^k = argmax_{a'}\left(Q(s,a') + U(s,a')\right)$$

$$U(s,a') = P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a')}\left(c_1 + log\left(\frac{\sum_b N(s,b) + c_2 + 1}{c_2}\right)\right)$$



Monte Carlo Tree Search

- We perform several simulations, for each simulation we:
  - **Select:** action according to tree statistics, until we reach a leaf node
  - **Expand:** send state to the prediction function for evaluation



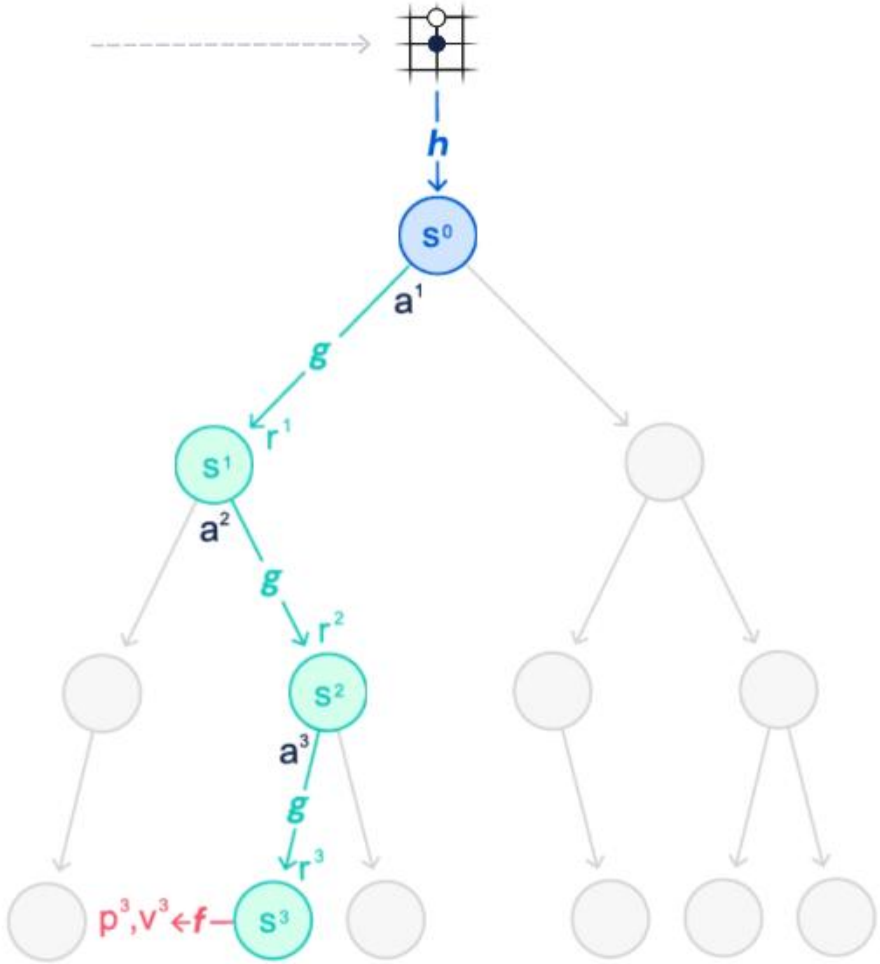
Monte Carlo Tree Search

- We perform several simulations, for each simulation we:
  - **Select:** action according to tree statistics, until we reach a leaf node
  - **Expand:** send state to the prediction function for evaluation
  - **Backup:** update the tree statistics

$$G^k = \left(\sum_{\tau=0}^{l-1-k} \gamma^{\tau} r_{k+1+\tau}\right) + \gamma^{l-k} v^l$$

$$Q(s^{k-1}, a^k) = \frac{N(s^{k-1}, a^k) \cdot Q(s^{k-1}, a^k) + G^k}{N(s^{k-1}, a^k) + 1}$$

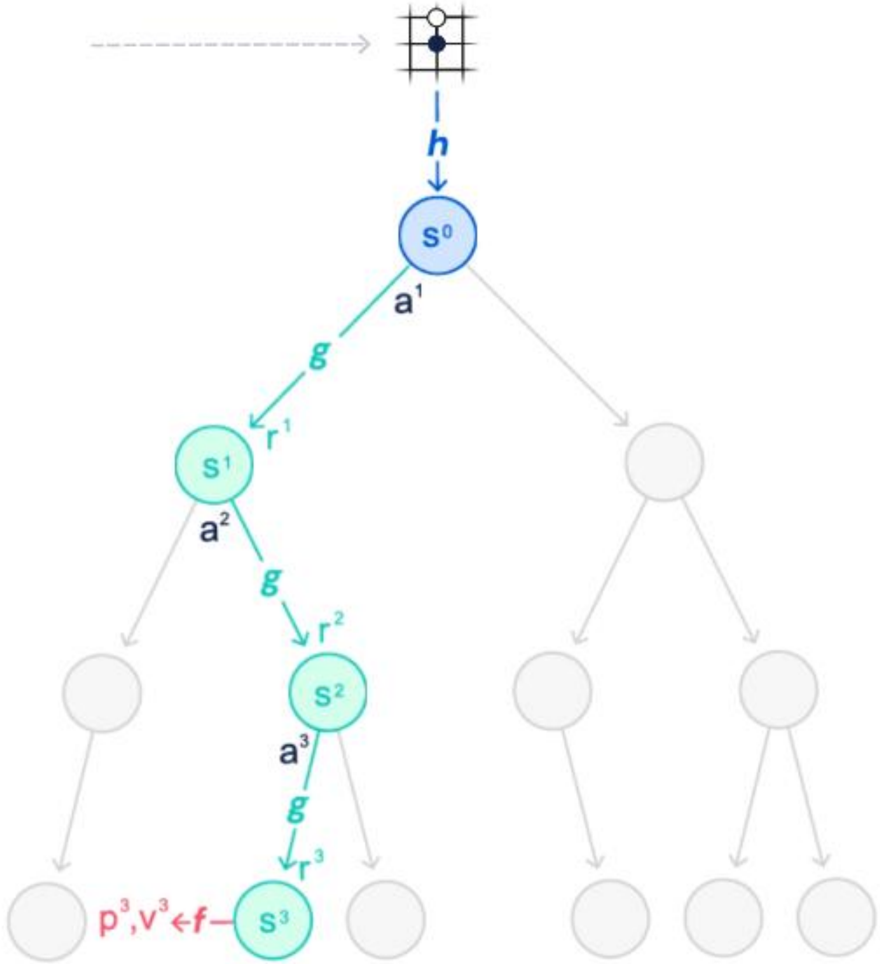
$$N(s^{k-1}, a^k) = N(s^{k-1}, a^k) + 1$$



Monte Carlo Tree Search

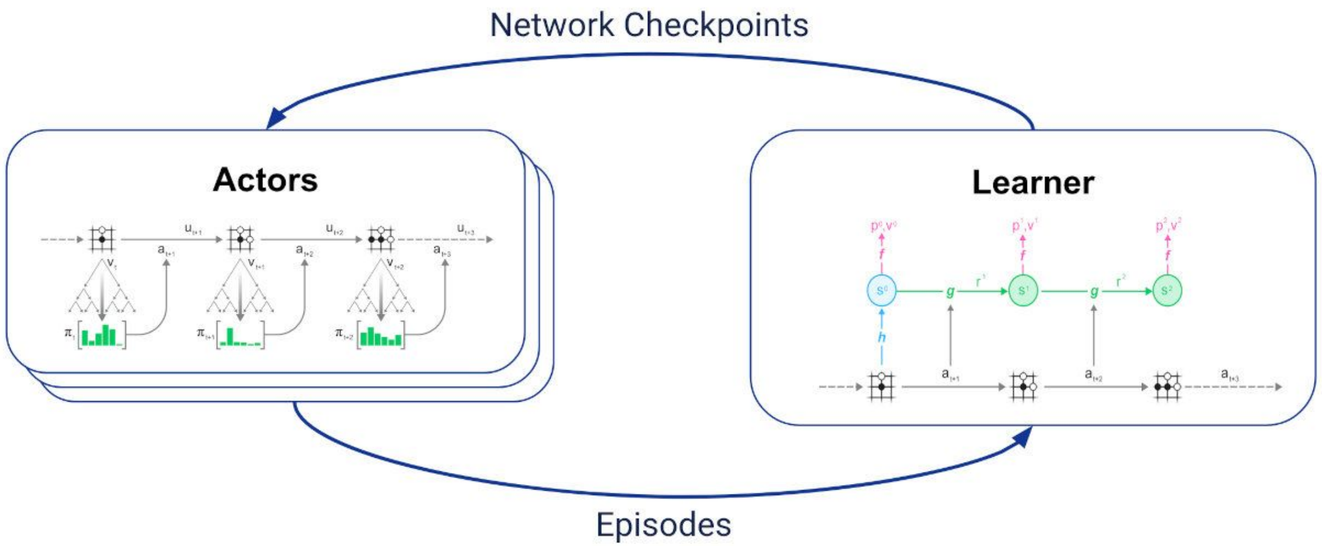
- From each state we are able to obtain the:
    1. policy
    2. value

$$\pi_{MCTS}(s,a) = \frac{N(s,a)^{1/\tau}}{\sum_b N(s,b)^{1/\tau}}$$



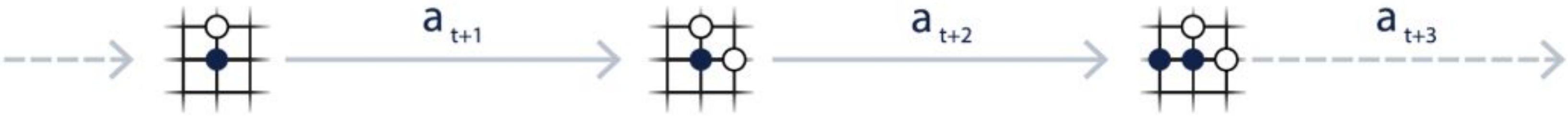
MuZero

- Learns a model of the environment
- Uses MCTS to pick the next action
- Trained by self-play



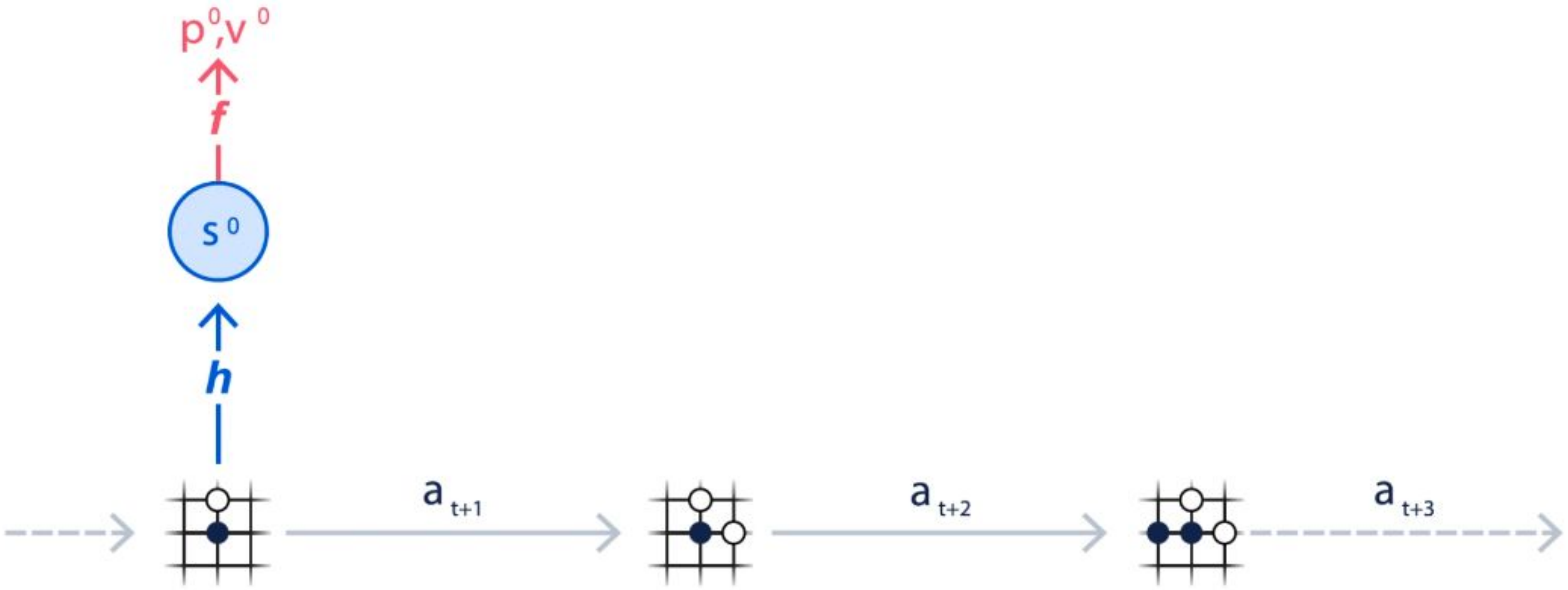
Training

We sample trajectories from the buffer



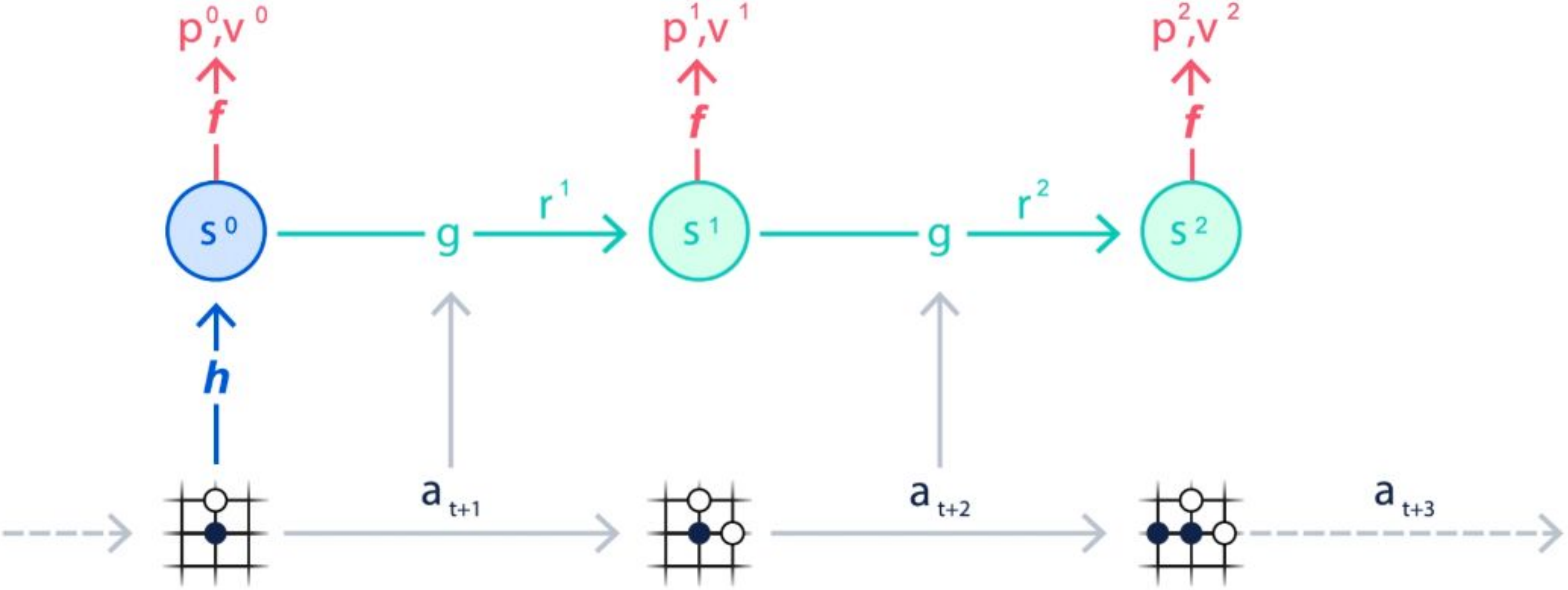
Training

We sample trajectories from the buffer
For each trajectory, we unroll the model for K steps



Training

We sample trajectories from the buffer
For each trajectory, we unroll the model for K steps



Training

After unrolling, we have a list of:

rewards:

$$r_t^0, \ldots, r_t^k$$

values:

$$v_t^0, \ldots, v_t^k$$

policies:

$$p_t^0, \ldots, p_t^k$$

Training

After unrolling, we have a list of:

rewards:

$$r_t^0, \ldots, r_t^k$$

values:

$$v_t^0, \ldots, v_t^k$$

policies:

$$p_t^0, \ldots, p_t^k$$

The trajectory has the following values:

rewards from the environment:

$$u_t, \ldots, u_{t+k}$$

values:

$$z_t, \ldots, z_{t+k}$$

policies from MCTS:

$$\pi_t, \ldots, \pi_{t+k}$$

$$z_{t+k} = u_{t+1} + \gamma u_{t+2} + .. + \gamma^{n-1} u_{t+n} + \gamma^n v_{t+n}$$

Training

After unrolling, we have a list of:

rewards:

$$r_t^0, \ldots, r_t^k$$

values:

$$v_t^0, \ldots, v_t^k$$

policies:

$$p_t^0, \ldots, p_t^k$$

The trajectory has the following values:

rewards from the environment:

$$u_t, \ldots, u_{t+k}$$

values:

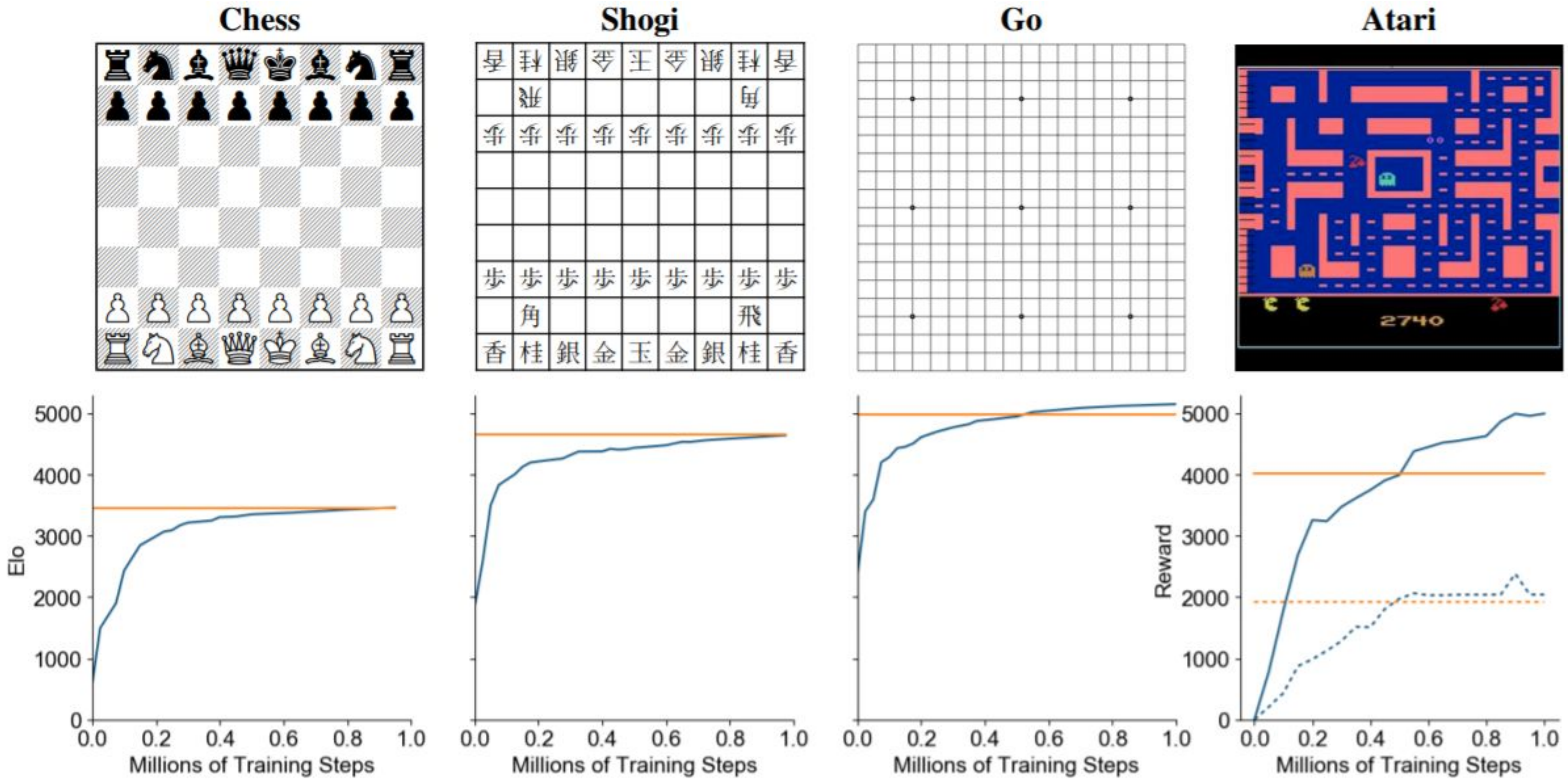
$$z_t, \ldots, z_{t+k}$$

policies from MCTS:

$$\pi_t, \ldots, \pi_{t+k}$$

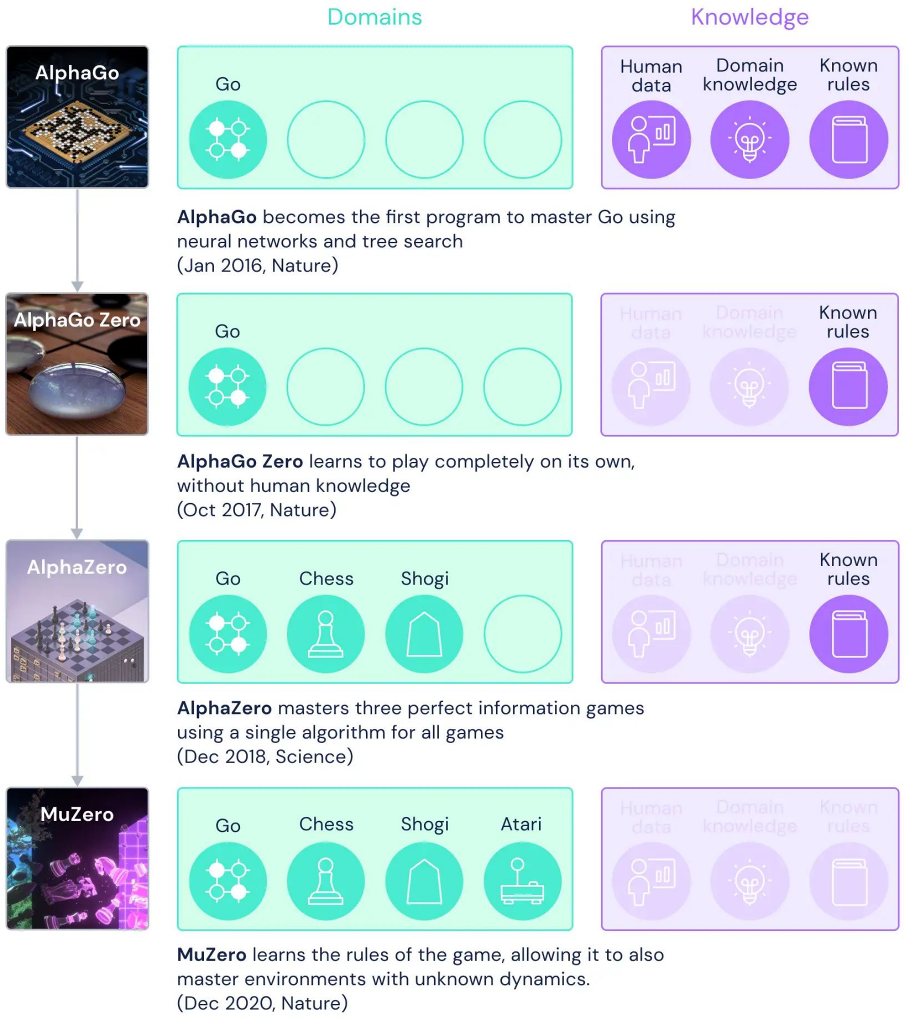
$$l_t(\theta) = \sum_{k=0}^{k} \left[l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\pi_{t+k}, p_t^k) \right]$$

Results



The blue line is MuZero. In Board Games, the orange line is AlphaZero.
In Atari, mean is the full line and median the dashed. Orange line is R2D2, the previous model based state of the art.

Improving MuZero



AlphaZero:

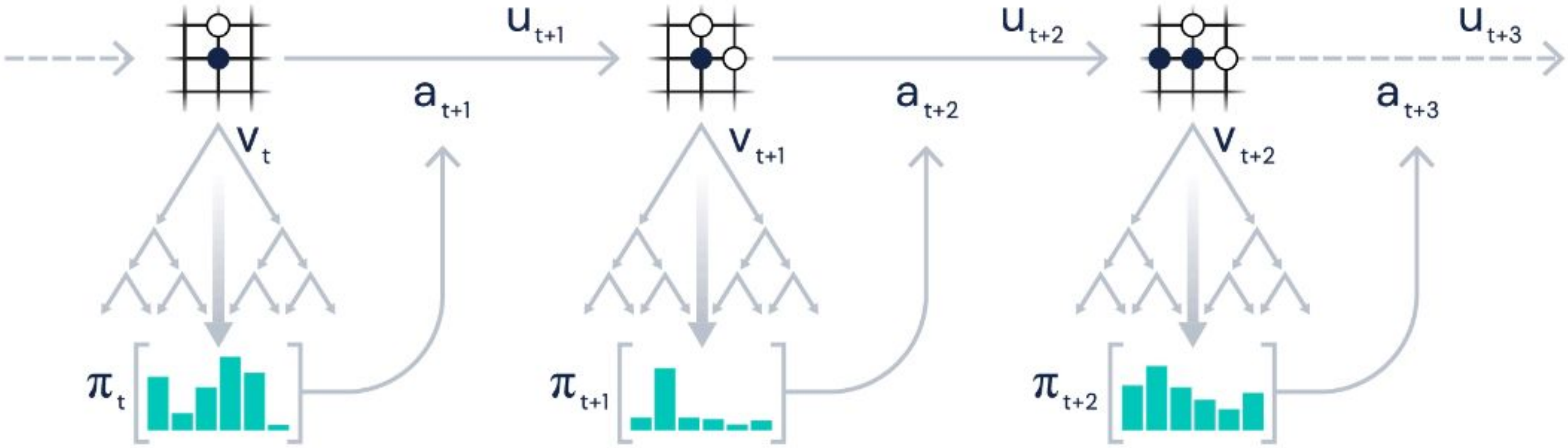
- 5064 TPUs
- 3 days

MuZero:

- 1016 TPUs - Board games
- 40 TPUs - Atari Games

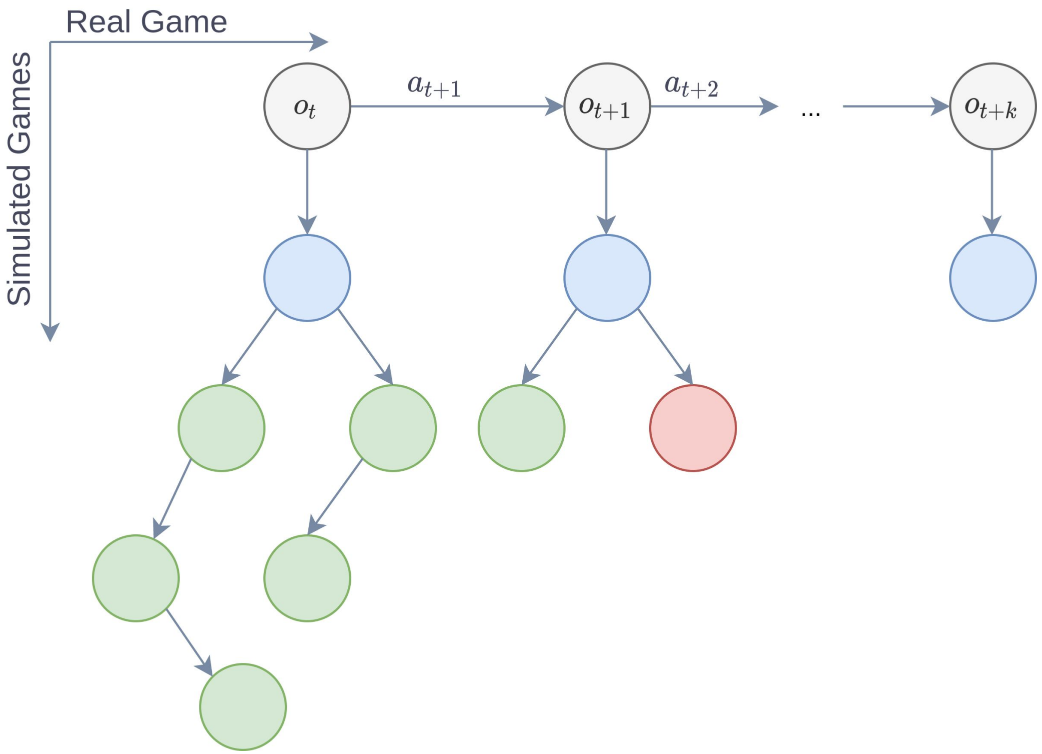
Improving MuZero

- Build a tree using Monte Carlo Tree Search (MCTS) when choosing a move
- This tree is composed of several possible future move trajectories
- We pick the action with the most promising trajectory
- Data from the tree is not used for training



Improving MuZero

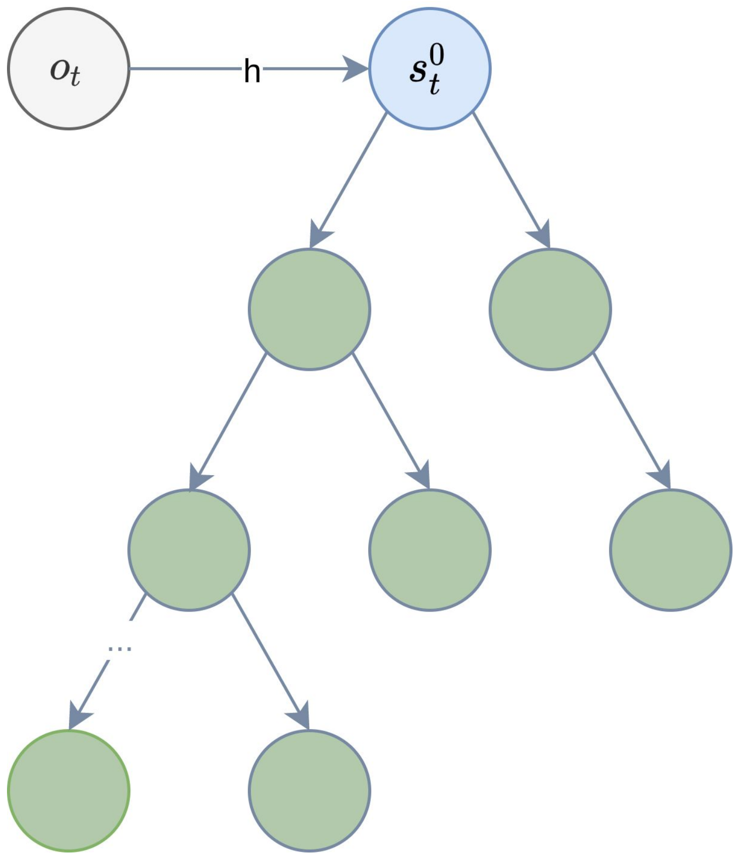
- **A0GB [5]** proposed a way of obtaining data from tree in AlphaZero.
- This data can be considered off-policy.
    a. The policy used to collected data is different than the one used for training.



How do we use simulated trajectories?

The trajectory has the following values:

observations: $o_0, \ldots, o_t$



How do we use simulated trajectories?

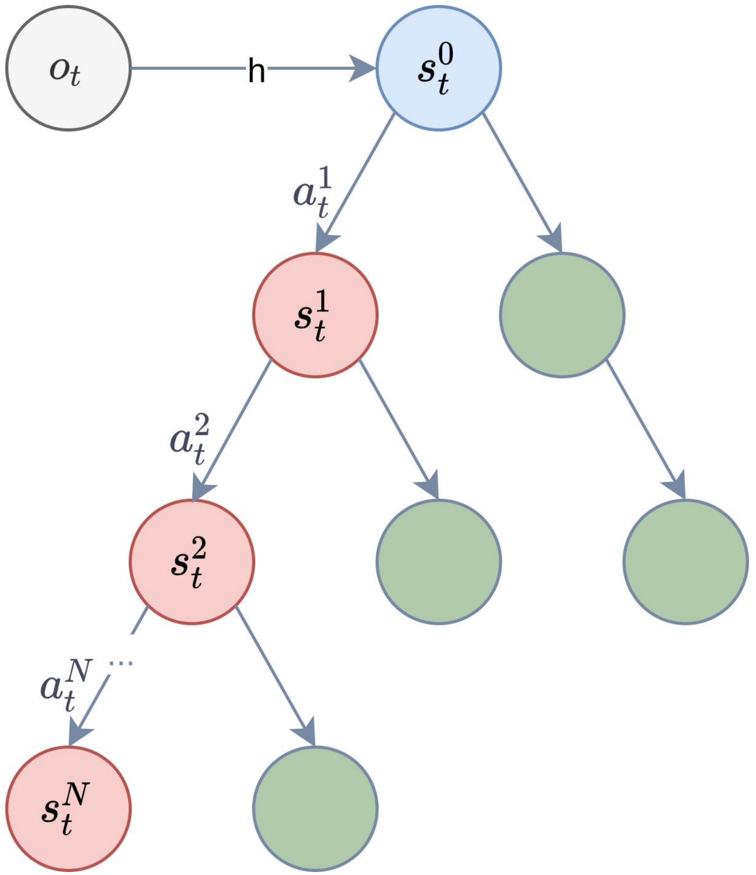
The trajectory has the following values:

observations: $o_0, \ldots, o_t$

Pick the path with the highest visit count:

actions: $a_t^0, \ldots, a_t^N$

policies from MCTS: $\pi_t^0, \ldots, \pi_t^N$



How do we use simulated trajectories?

The trajectory has the following values:

observations: $o_0, \ldots, o_t$

Pick the path with the highest visit count:

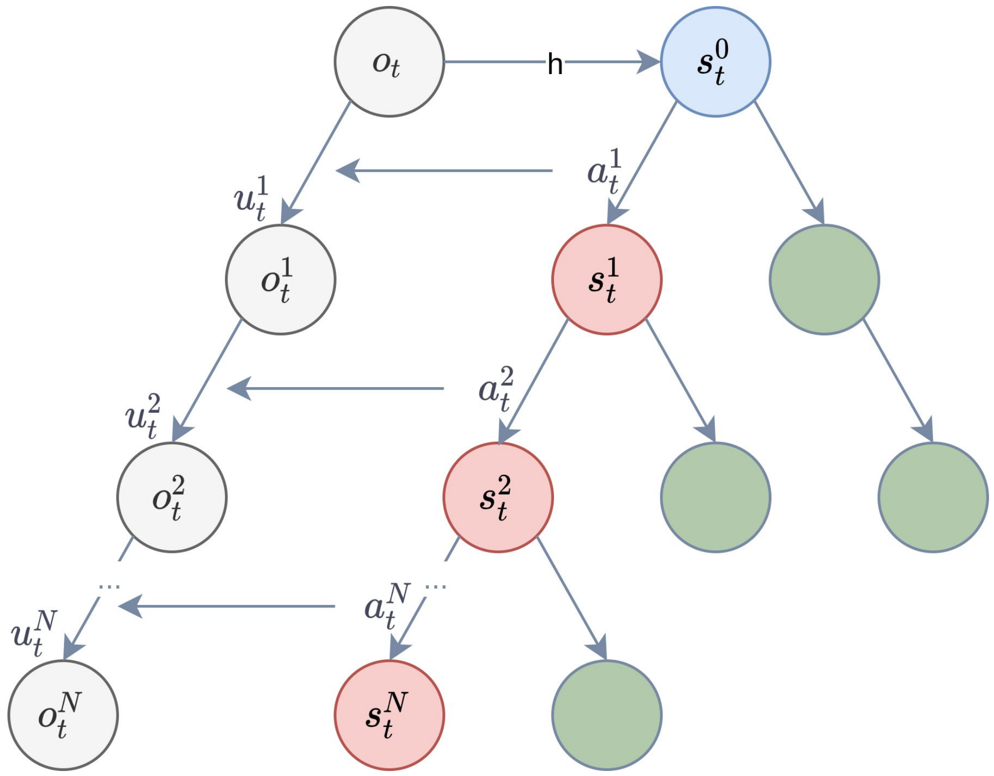
actions: $a_t^0, \ldots, a_t^N$

policies from MCTS: $\pi_t^0, \ldots, \pi_t^N$

Apply actions to the environment:

rewards: $u_t^0, \ldots, u_t^N$

values: $z_t^0, \ldots, z_t^N$



Combining Off and On-policy Targets

$$l_t(\theta) = \sum_{k=0}^{k} \left[l^r\left(u_{t+k}, r_t^k\right) + l^v\left(z_{t+k}, v_t^k\right) + l^p\left(\pi_{t+k}, p_t^k\right) \right]$$

Combining Off and On-policy Targets

$$l_t(\theta) = \sum_{k=0}^{k} \left[l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\pi_{t+k}, p_t^k) \right]$$

Combining Off and On-policy Targets

$$l_t(\theta) = \sum_{k=0}^{k} \left[l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\pi_{t+k}, p_t^k) \right]$$

$$l_t(\theta) = l_{value} + l_{policy}$$

$$l_{value} = \sum_{k=0}^{k} \left[l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) \right]$$

$$l_{policy} = \sum_{k=0}^{k} l^p(\pi_{t+k}, p_t^k)$$

Combining Off and On-policy Targets

$$l_t^{combined}(\theta) = \alpha l_{value}^{real} + \beta l_{policy}^{real} + \gamma l_{value}^{simulated} + \delta l_{policy}^{simulated}$$

MuZero:

$$\alpha = \beta = 1$$
$$\gamma = \delta = 0$$

Off-Policy MuZero:

$$\alpha = \beta = 0$$
$$\gamma = \delta = 1$$

Combining Off and On-policy Targets

$$l_t^{combined}(\theta) = \alpha l_{value}^{real} + \beta l_{policy}^{real} + \gamma l_{value}^{simulated} + \delta l_{policy}^{simulated}$$

Scaling:

Same loss magnitude regardless of parameters used

$$\alpha' = \frac{\alpha}{\alpha + \gamma} \qquad \gamma' = \frac{\gamma}{\alpha + \gamma}$$

$$\beta' = \frac{\beta}{\beta + \delta} \qquad \delta' = \frac{\delta}{\beta + \delta}$$

Results

We tested on three environments

- Cartpole
- TicTacToe
- Simplified MiniGrid

α	β	γ	δ	
1	1	0	0	MuZero
0	0	1	1	M0OFF
0	1	1	0	M0GB
1	1	1	0	M0OFFV
1	1	1	1	M0ALL

$\alpha : real\, value\ \beta : real\, policy\ \gamma : simulated\, value\ \delta : simulated\, policy$

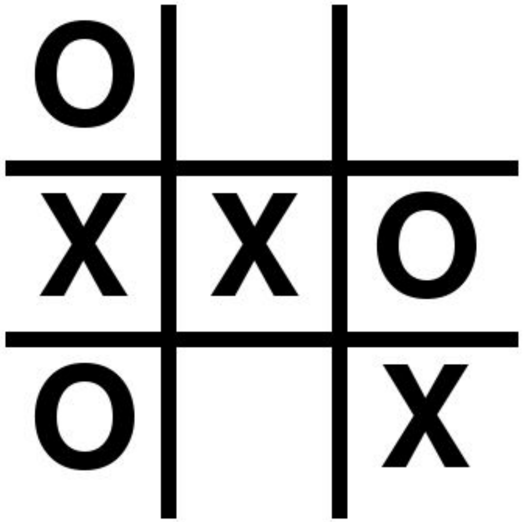
TicTacToe

Characteristics:

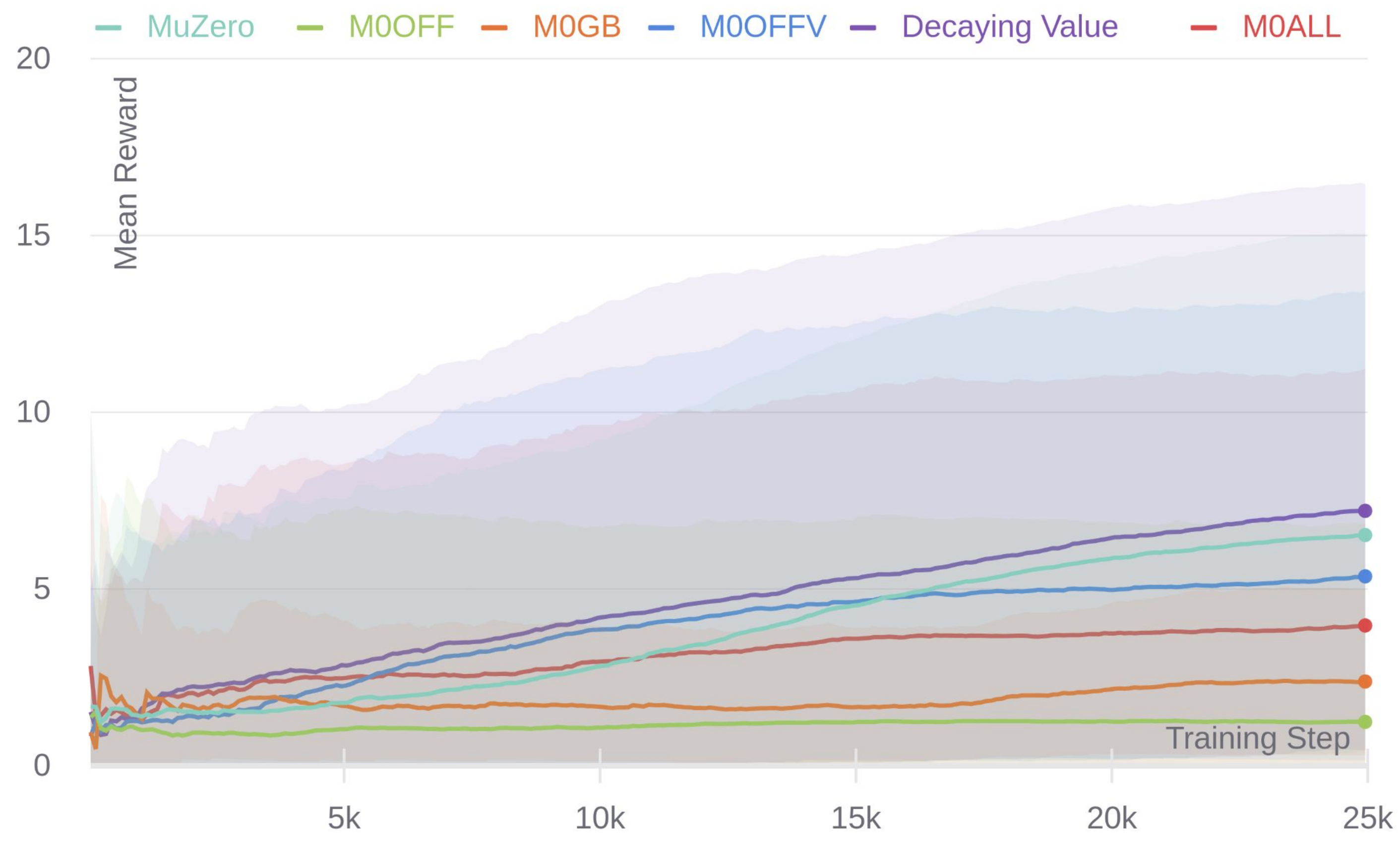
- Sparse Rewards
- 20 reward if wins, 0 draw, -20 loss

Parameters:

- 10 runs for each parameter set
- 25000 steps
- 9 look-head
- 3 unroll size
- 25 simulations

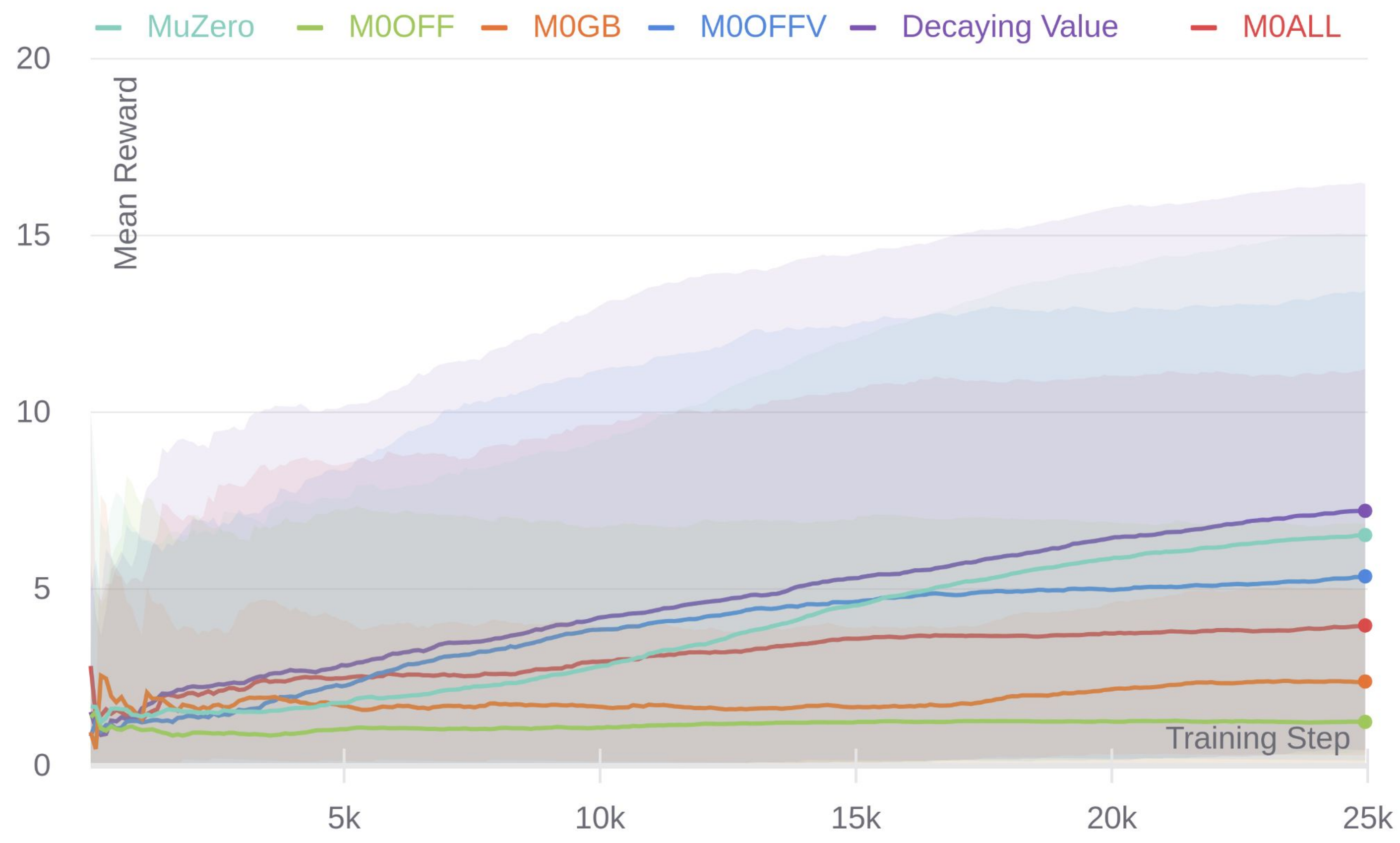


TicTacToe



MuZero	6.51 ± 4.95
M0OFF	1.22 ± 2.31
M0GB	2.54 ± 2.80
M0OFFV	5.44 ± 4.52
M0ALL	3.89 ± 3.96
Decaying Value	**7.23** ± 5.49

TicTacToe



Off-policy value target γ:

- Faster convergence
- Deteriorates towards the end

Off-policy policy target δ:

- Not useful at all

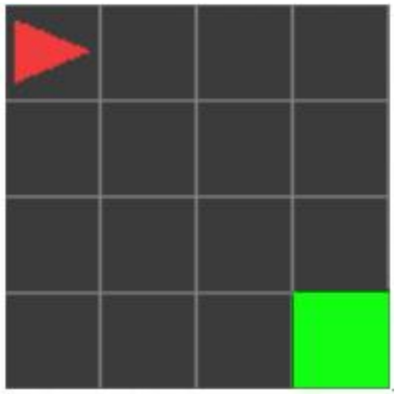
MiniGrid (N x N)

Characteristics:

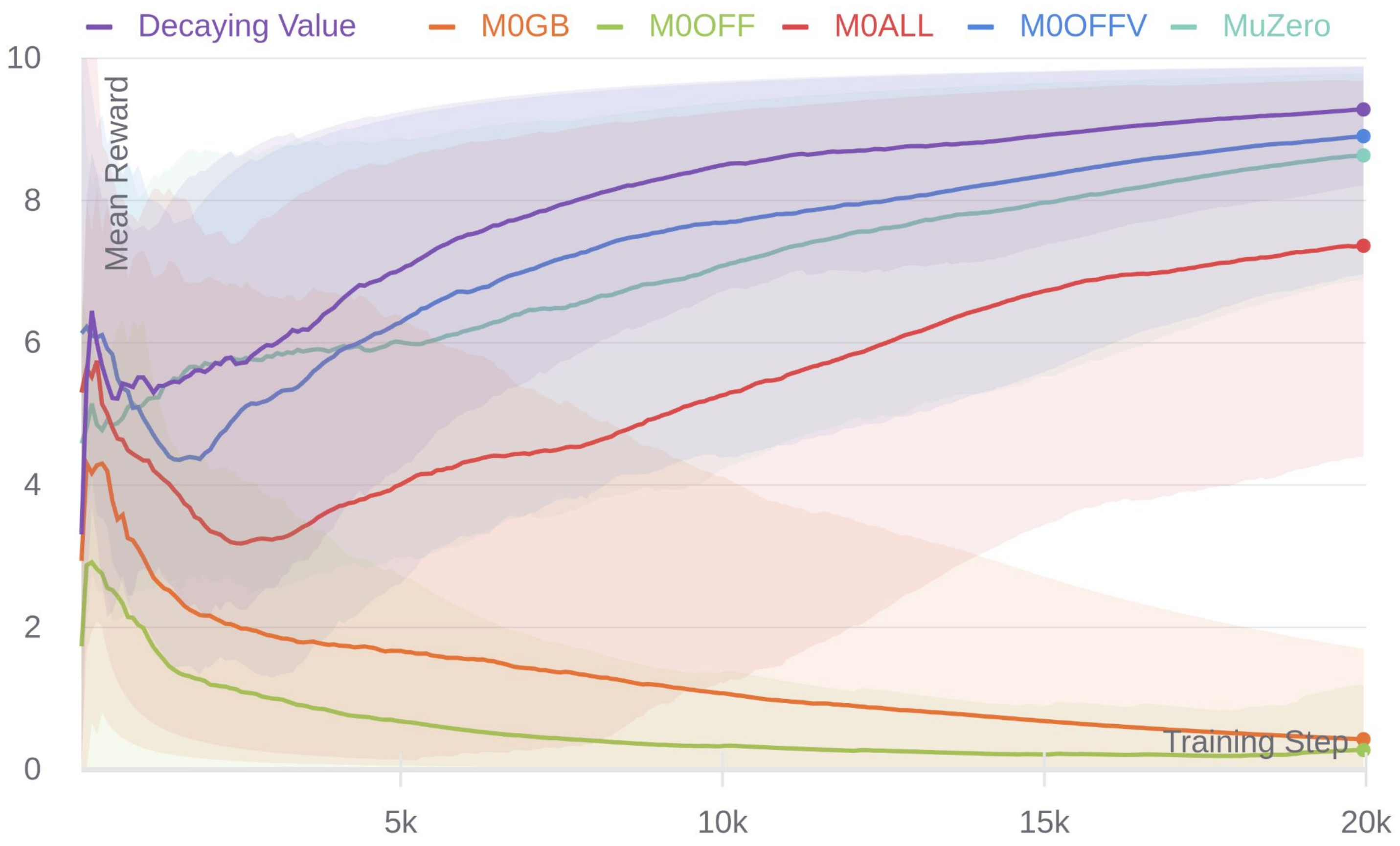
- Sparse Rewards
- Reach the other corner
- Ends when N + N steps have passed

Parameters:

- 6 runs for each parameter set
- Grid sizes of 3,4,5,6 tested
- 15000 steps and 20000 steps
- 7 look-head
- 7 unroll size
- 5 simulations



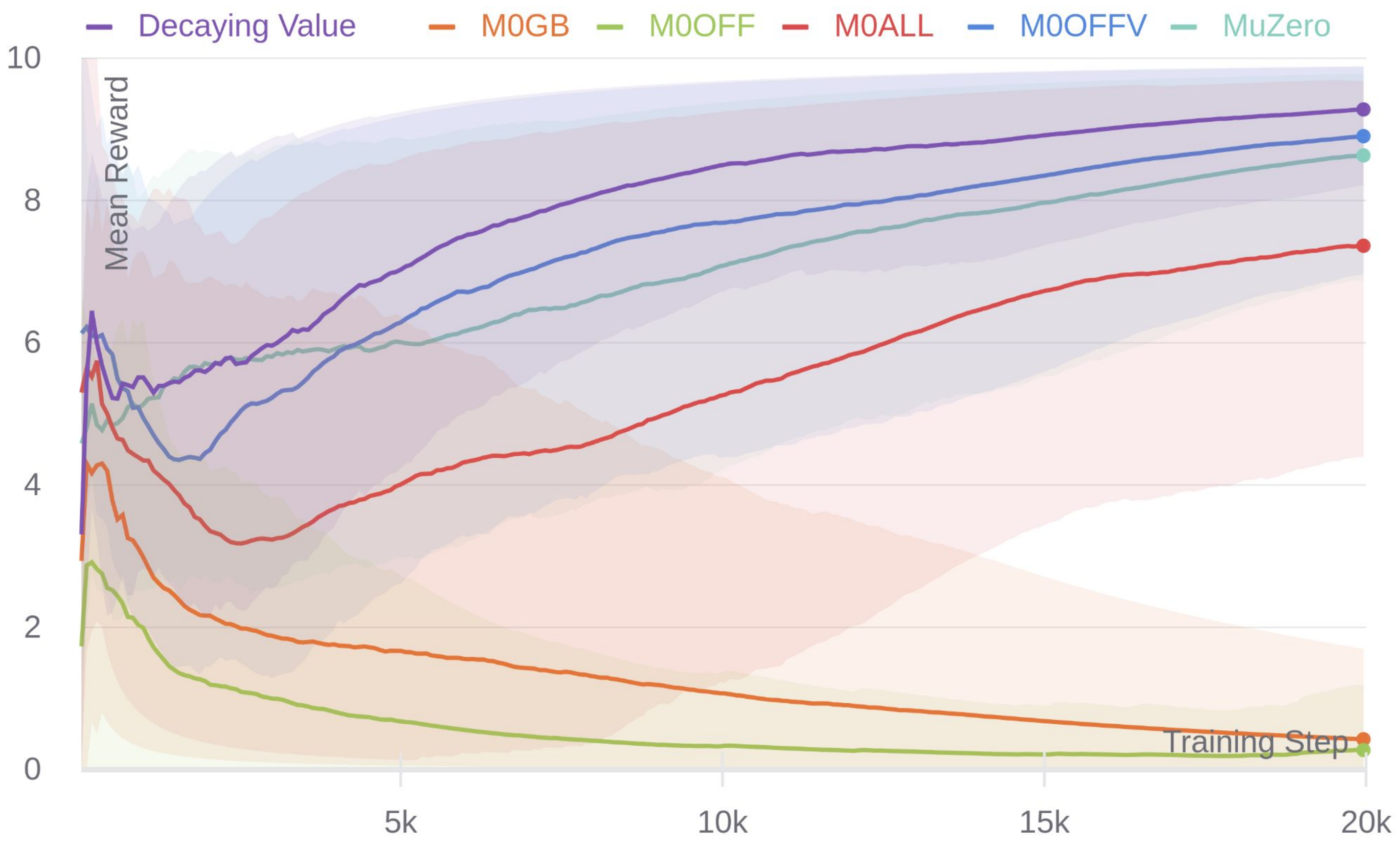
MiniGrid



MuZero	8.73 ± 1.20
M0OFF	0.26 ± 0.52
M0GB	0.44 ± 0.76
M0OFFV	8.88 ± 1.33
M0ALL	7.40 ± 2.34
Decaying Value	**9.18** ± 0.97

α : *real value* β : *real policy* γ : *simulated value* δ : *simulated policy*

MiniGrid



Off-policy value target γ:

- Faster convergence
- Higher end rewards

Off-policy policy target δ:

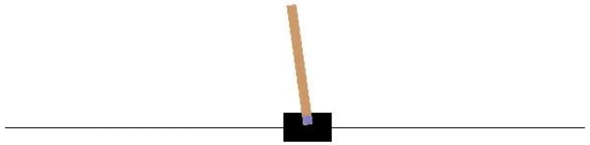
- Not useful at all

α : *real value* β : *real policy* γ : *simulated value* δ : *simulated policy*

Cartpole

Characteristics:

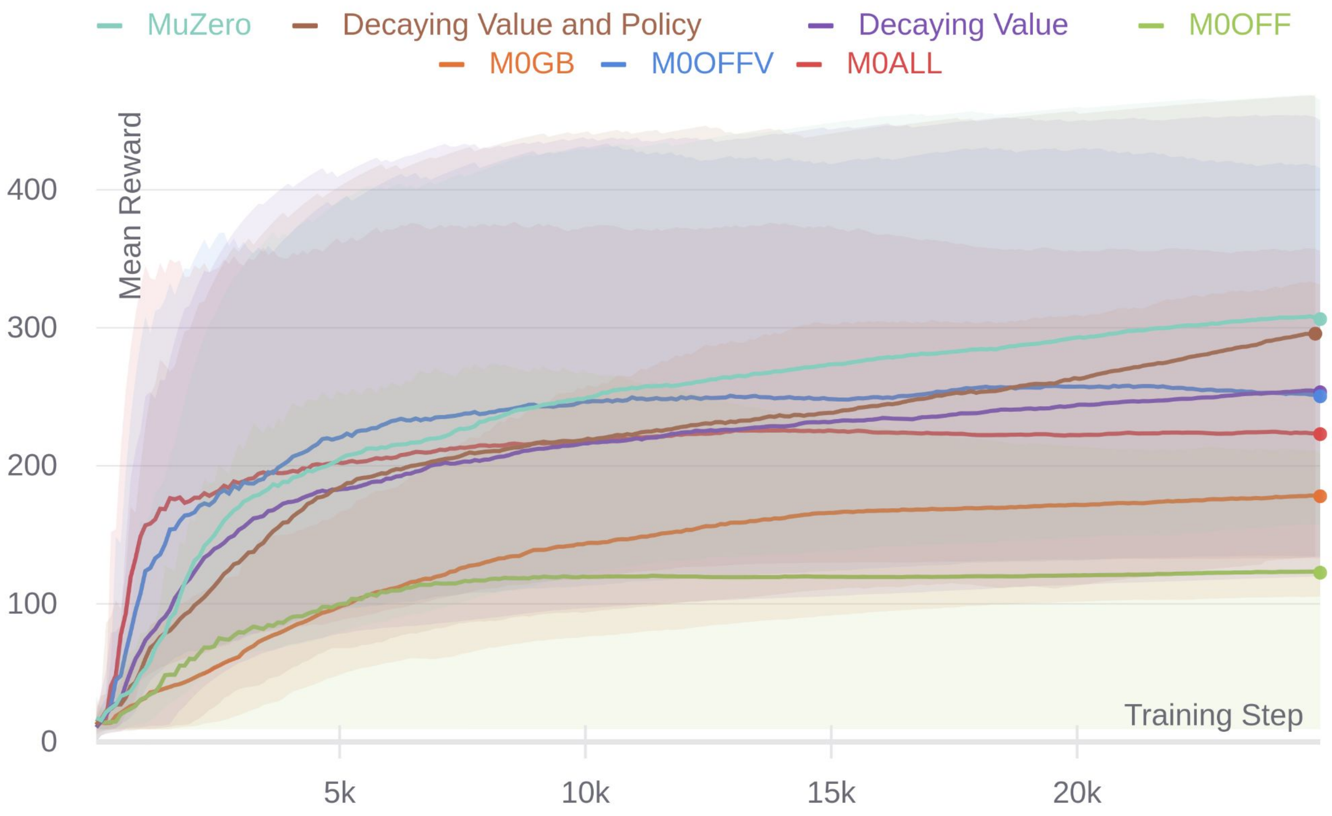
- Intermediate Rewards
- Keep the pole upright
- Ends when pole falls or 500 steps have passed
- Solved if able to keep pole upright for 195 steps



Parameters:

- 10 runs for each parameter set
- 25000 steps
- 50 look-ahead length
- 10 unroll size
- 50 simulations

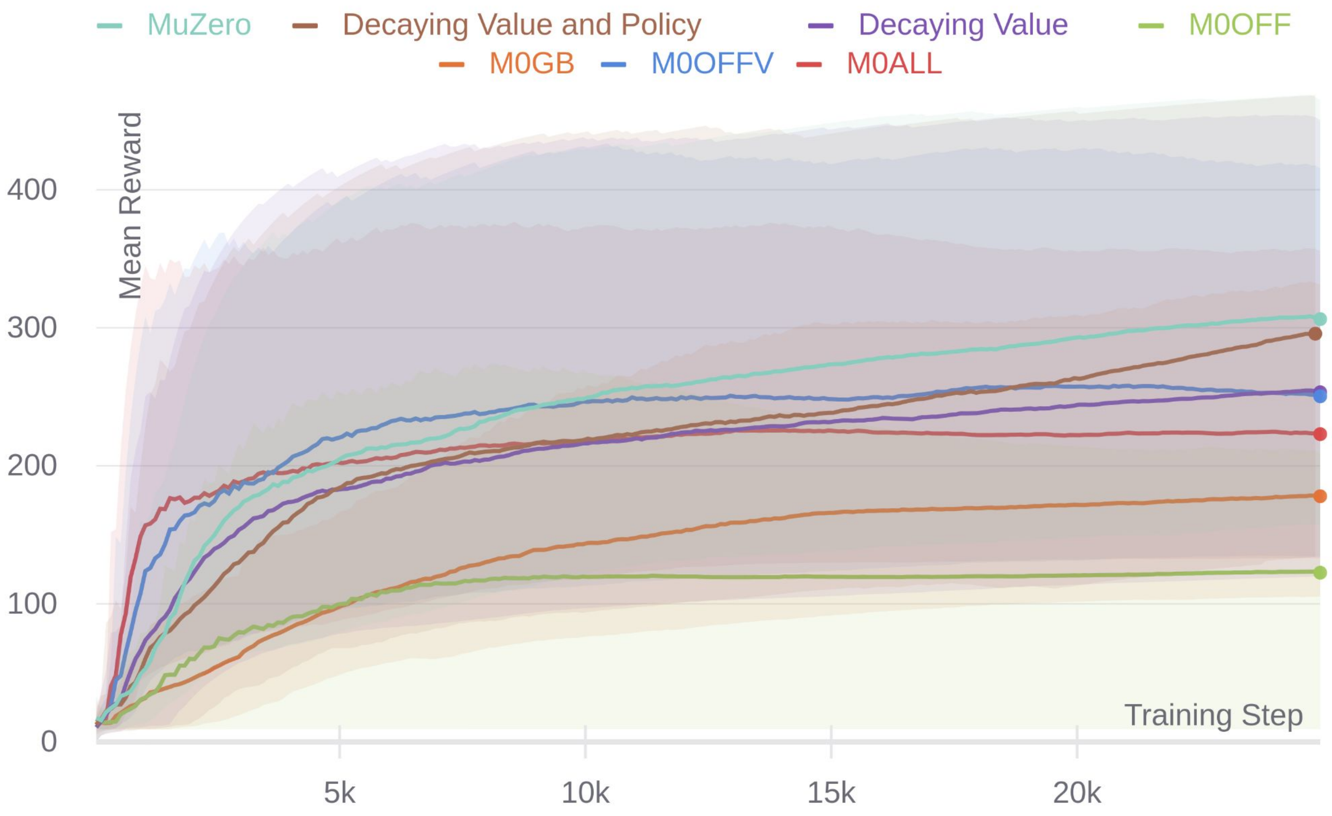
Cartpole



MuZero	**306** ± 136
M0OFF	123 ± 74
M0GB	178 ± 92
M0OFFV	250 ± 131
M0ALL	223 ± 93
Decaying Value	253 ± 121
Decaying Value and Policy	295 ± 143

α : *real value* β : *real policy* γ : *simulated value* δ : *simulated policy*

Cartpole



Off-policy value target γ:

- Faster convergence
- Deteriorates towards the end

Off-policy policy target δ:

- Useful in the beginning
- Runs that use this quickly stagnate

α : *real value* β : *real policy* γ : *simulated value* δ : *simulated policy*

Comparing environments

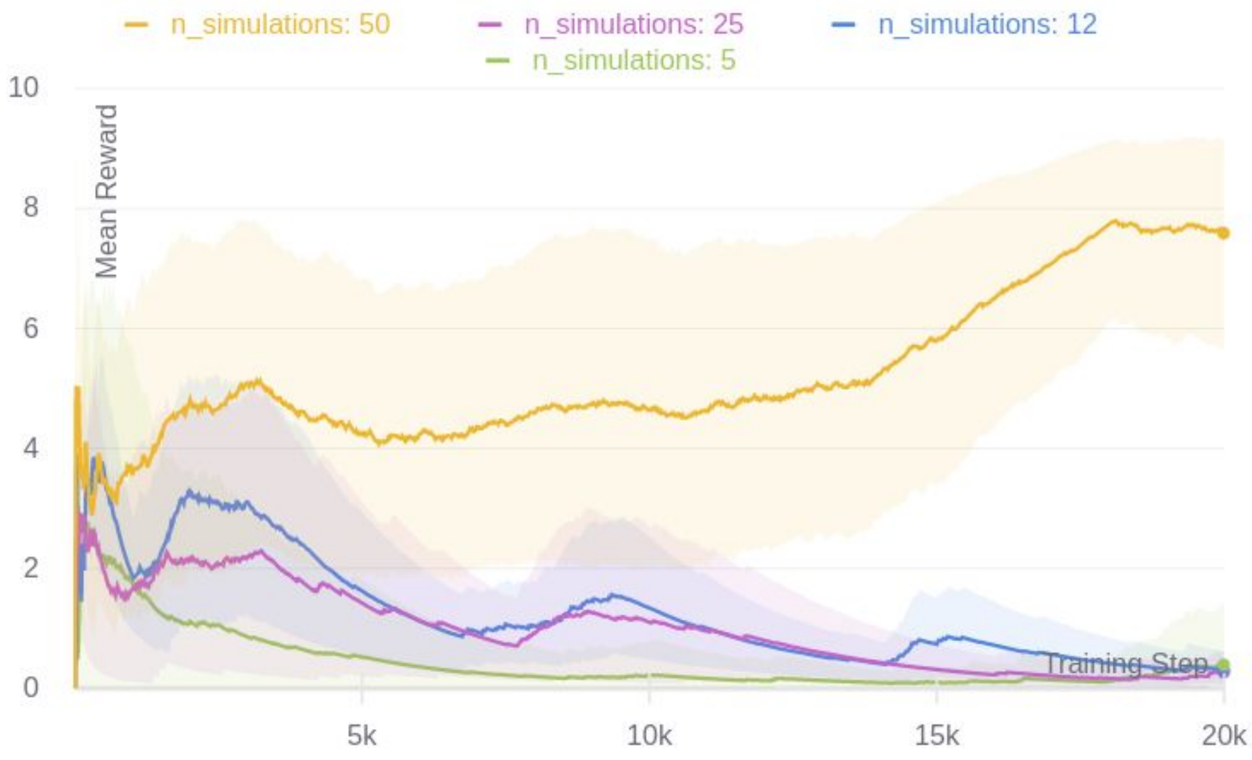
Off-policy policy target γ:

- Not useful in environments with sparse rewards

Comparing environments

Off-policy policy target γ:

- Not useful in environments with sparse rewards
- Dependent on the number of simulations.



M0OFF MiniGrid 6x6 Runs with varying number of simulations

Comparing environments

Off-policy value target $\boldsymbol{\delta}$:

- Improves convergence speed
- Relies less on the number of simulations

Comparing environments

Off-policy value target $\boldsymbol{\delta}$:

- Improves convergence speed
- Relies less on the number of simulations
- Seems to have more impact in environments with sparse rewards

Future Work

- Recent work [6] proposed a way to obtain policy from MCTS that isn't as dependent on the number of simulations.

Future Work

- Recent work [6] proposed a way to obtain policy from MCTS that isn't as dependent on the number of simulations.
- GradNorm [7] balances the losses of networks that are optimizing for several tasks in real time based on the gradients of those losses.

$$l_t^{combined}(\theta) = \alpha l_{value}^{real} + \beta l_{policy}^{real} + \gamma l_{value}^{simulated} + \delta l_{policy}^{simulated}$$

Conclusions

- The results show that these off-policy targets are useful for training and can speed up convergence

Conclusions

- The results show that these off-policy targets are useful for training and can speed up convergence
- Off policy value target:
  - seems to provide the most value, specially, in environments with sparse rewards
  - less sensitive to the number of simulations performed.

Conclusions

- The results show that these off-policy targets are useful for training and can speed up convergence
- Off policy value target:
  - seems to provide the most value, specially, in environments with sparse rewards
  - less sensitive to the number of simulations performed.
- In Cartpole, we were able at best get similar results to MuZero.
- In Minigrid and TicTacToe, were able to get better results than MuZero

References

- [1]- Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016).
- [2]- Silver, D., Schrittwieser, J., Simonyan, K. et al. Mastering the game of Go without human knowledge. Nature 550, 354–359 (2017).
- [3]- Silver, D., Schrittwieser, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 2018, 1140-1144
- [4]- Schrittwieser, J., Antonoglou, I., Hubert, T. et al. Mastering Atari, Go, chess and shogi by planning with a learned model. Nature 588, 604–609 (2020).
- [5]- D. Willemsen, H. Baier, and M. Kaisers, "Value targets in off-policy alphazero: a new greedy backup," 2020.
- [6]- J.-B. Grill, F. Altché, Y. Tang, T. Hubert, M. Valko, I. Antonoglou, and R. Munos, "Monte-carlo tree search as regularized policy optimization," 2020, pp. 3769–3778.
- [7]- Z. Chen, V. Badrinarayanan, C. Y. Lee, and A. Rabinovich, "Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks," vol. 2, 2018.
- [8] Alexandre Borges, Arlindo Oliveira, "Combining Off and On-Policy Training in Model-Based Reinforcement Learning"

Thank You!
Questions?