MuZero, AlphaZero Policy Target, Gumbel-Softmax, GumbelZero

Milan Straka

May 6, 2024

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

MuZero

The MuZero algorithm extends the AlphaZero by a **trained model**, alleviating the requirement for a known MDP dynamics. It is evaluated both on board games and on the Atari domain.

At each time-step t, for each of $1 \le k \le K$ steps, a model μ_θ with parameters θ, conditioned on past observations $o_1, \ldots, o_t$ and future actions $a_{t+1}, \ldots, a_{t+k}$, predicts three future quantities:

- the policy $\boldsymbol{p}_t^k \approx \pi\big(a_{t+k+1} \big| o_1, \ldots, o_t, a_{t+1}, \ldots, a_{t+k}\big)$,
- the value function $v_t^k \approx \mathbb{E}\big[u_{t+k+1} + \gamma u_{t+k+2} + \ldots \big| o_1, \ldots, o_t, a_{t+1}, \ldots, a_{t+k}\big]$,
- the immediate reward $r_t^k \approx u_{t+k}$,

where u_i are the observed rewards and π is the behaviour policy.

MuZero

At each time-step t (omitted from now on for simplicity), the model is composed of three components: a **representation** function, a **dynamics** function, and a **prediction** function.

- The dynamics function, $(r^k, \boldsymbol{s}^k) \leftarrow g_\theta(\boldsymbol{s}^{k-1}, a^k)$, simulates the MDP dynamics and predicts an immediate reward r^k and an internal state $\boldsymbol{s}^k$. The internal state has no explicit semantics, its only goal is to accurately predict rewards, values, and policies.
- The prediction function, $(\boldsymbol{p}^k, v^k) \leftarrow f_\theta(\boldsymbol{s}^k)$, computes the policy and the value function, similarly as in AlphaZero.
- The representation function, $\boldsymbol{s}^0 \leftarrow h_\theta(o_1, \ldots, o_t)$, generates an internal state encoding the past observations.

MuZero

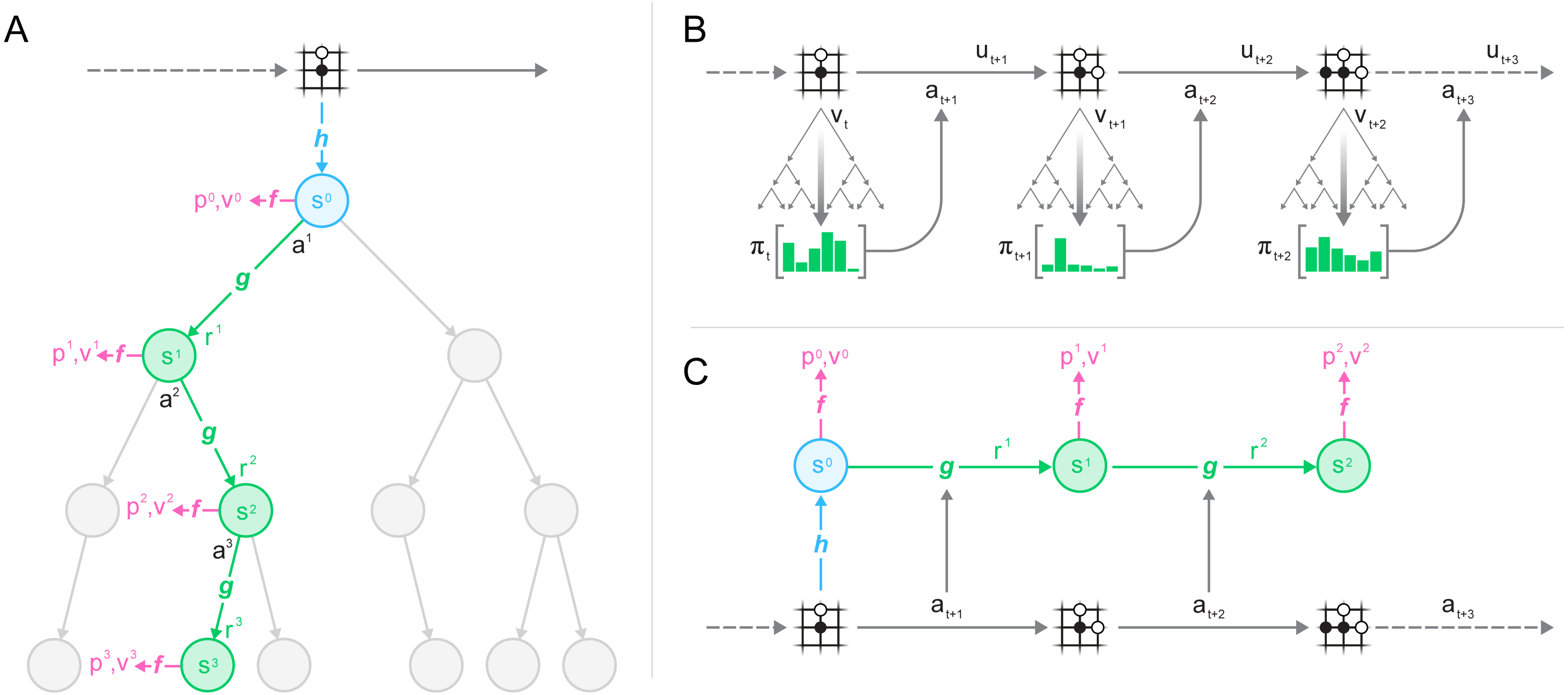


Figure 1 of "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

MuZero – MCTS

The MCTS algorithm is very similar to the one used in AlphaZero, only the trained model is used. It produces a policy $\boldsymbol\pi_t$ and a value estimate ν_t.

- All actions, including the invalid ones, are allowed at any time, except at the root, where the invalid actions (available from the current state) are disallowed.
- No states are considered terminal during the search.
- During the backup phase, we consider a general discounted bootstrapped return

$$G_k = \sum\nolimits_{t=0}^{l-k-1} \gamma^t r_{k+1+t} + \gamma^{l-k} v_l.$$

- Furthermore, the expected return is generally unbounded. Therefore, MuZero normalize the Q-value estimates to $[0, 1]$ range by using the minimum and maximum the values observed in the search tree until now:

$$\bar Q(s, a) = \frac{Q(s, a) - \min_{s',a' \in \mathrm{Tree}} Q(s', a')}{\max_{s',a' \in \mathrm{Tree}} Q(s', a') - \min_{s',a' \in \mathrm{Tree}} Q(s', a')}.$$

MuZero – Action Selection

To select a move, we employ a MCTS algorithm and then sample an action from the obtained policy, $a_{t+1} \sim \boldsymbol{\pi}_t$.

For games, the same strategy of sampling the actions a_t as in AlphaZero is used. In the Atari domain, the actions are sampled according to the visit counts for the whole episode, but with a given temperature T:

$$\pi(a|s) = \frac{N(s,a)^{1/T}}{\sum_b N(s,b)^{1/T}},$$

where T is decayed during training – for first 500k steps it is 1, for the next 250k steps it is 0.5 and for the last 250k steps it is 0.25.

While for the board games 800 simulations are used during MCTS, only 50 are used for Atari.

In case of Atari, the replay buffer consists of 125k sequences of 200 actions.

MuZero – Training

During training, we utilize a sequence of K moves. We estimate the return using bootstrapping as $z_t = u_{t+1} + \gamma u_{t+2} + \ldots + \gamma^{n-1} u_{t+n} + \gamma^n \nu_{t+n}$. The values $K = 5$ and $n = 10$ are used in the paper, with batch size 2048 for the board games and 1024 for Atari.

The loss is then composed of the following components:

$$\mathcal{L}_t(\theta) = \sum_{k=0}^{K} \mathcal{L}^r(u_{t+k}, r_t^k) + \mathcal{L}^v(z_{t+k}, v_t^k) + \mathcal{L}^p(\boldsymbol{\pi}_{t+k}, \boldsymbol{p}_t^k) + c\|\theta\|^2.$$

Note that in Atari, rewards are scaled by $\operatorname{sign}(x)\big(\sqrt{|x|+1} - 1\big) + \varepsilon x$ for $\varepsilon = 10^{-3}$, and authors utilize a cross-entropy loss with 601 categories for values $-300, \ldots, 300$, which they claim to be more stable (this can be considered distributional RL).

Furthermore, in Atari the discount factor $\gamma = 0.997$ is used, and the replay buffer elements are sampled according to prioritized replay with priority $\propto |\nu - z|^\alpha$, and importance sampling with exponent β is used to account for changing the sampling distribution ($\alpha = \beta = 1$ is used).

MuZero

Model

$$\left.\begin{aligned} \boldsymbol{s}^0 &= h_\theta(o_1, \ldots, o_t) \\ r^k, \boldsymbol{s}^k &= g_\theta(\boldsymbol{s}^{k-1}, a^k) \\ \boldsymbol{p}^k, v^k &= f_\theta(\boldsymbol{s}^k) \end{aligned}\right\} \quad \boldsymbol{p}^k, v^k, r^k = \mu_\theta(o_1, \ldots, o_t, a^1, \ldots, a^k)$$

Search

$$\nu_t, \boldsymbol{\pi}_t = MCTS(\boldsymbol{s}_t^0, \mu_\theta)$$

$$a_t \sim \boldsymbol{\pi}_t$$

$$\begin{aligned}
&\text{Learning Rule}\\
\boldsymbol{p}_t^k, v_t^k, r_t^k &= \mu_\theta(o_1, \ldots, o_t, a_{t+1}, ..., a_{t+k})\\
z_t &= \begin{cases} u_T & \text{for games}\\ u_{t+1} + \gamma u_{t+2} + ... + \gamma^{n-1} u_{t+n} + \gamma^n \nu_{t+n} & \text{for general MDPs}\end{cases}\\
\mathcal{L}_t(\theta) &= \sum_{k=0}^{K} \mathcal{L}^r(u_{t+k}, r_t^k) + \mathcal{L}^v(z_{t+k}, v_t^k) + \mathcal{L}^p(\boldsymbol{\pi}_{t+k}, \boldsymbol{p}_t^k) + c\|\theta\|^2\\
&\text{Losses}\\
\mathcal{L}^r(u, r) &= \begin{cases} 0 & \text{for games}\\ -\boldsymbol{\varphi}(u)^T \log \boldsymbol{\varphi}(r) & \text{for general MDPs}\end{cases}\\
\mathcal{L}^v(z, q) &= \begin{cases} (z-q)^2 & \text{for games}\\ -\boldsymbol{\varphi}(z)^T \log \boldsymbol{\varphi}(q) & \text{for general MDPs}\end{cases}\\
\mathcal{L}^p(\boldsymbol{\pi}, p) &= -\boldsymbol{\pi}^T \log \boldsymbol{p}
\end{aligned}$$

MuZero – Evaluation

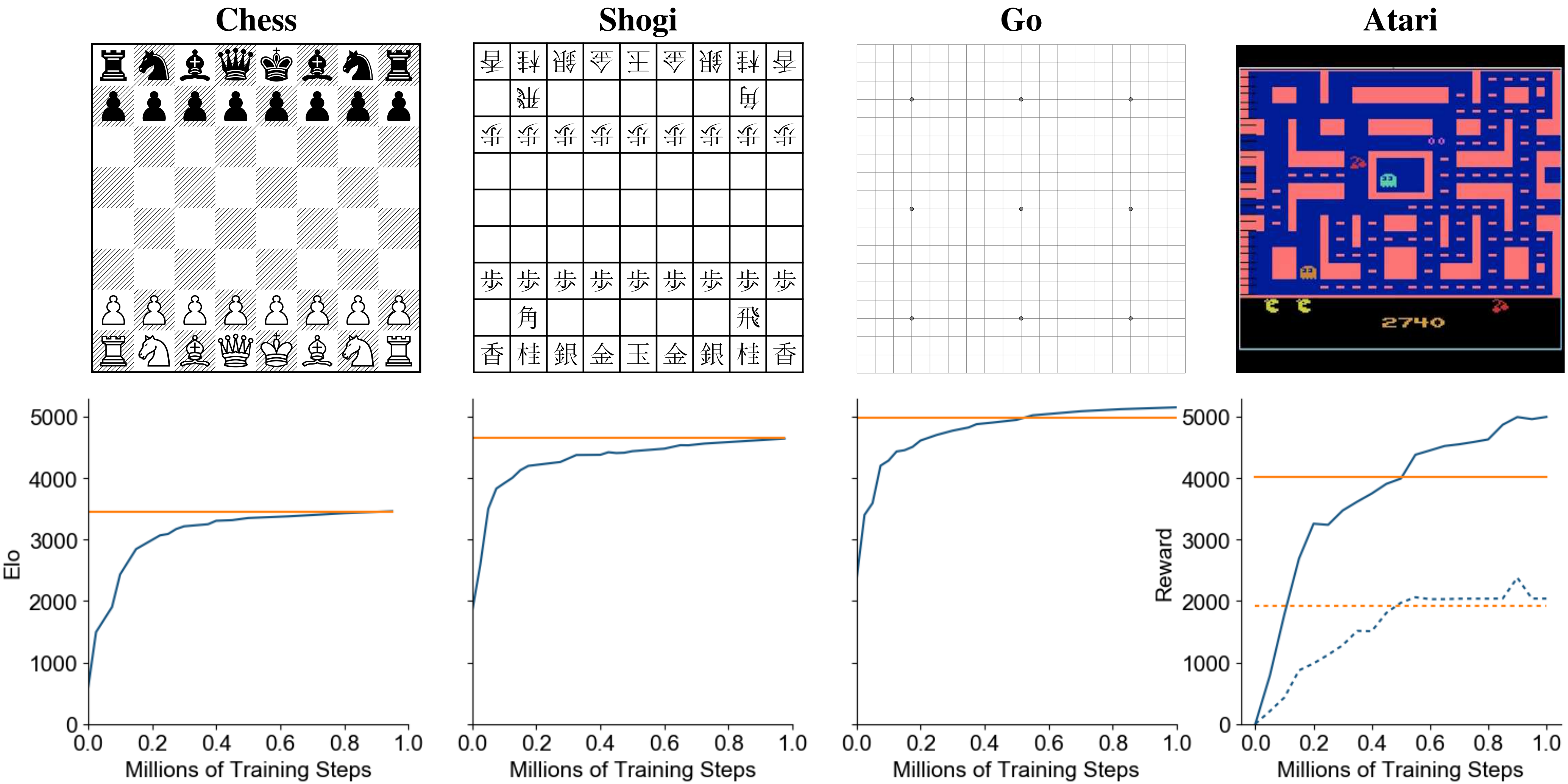
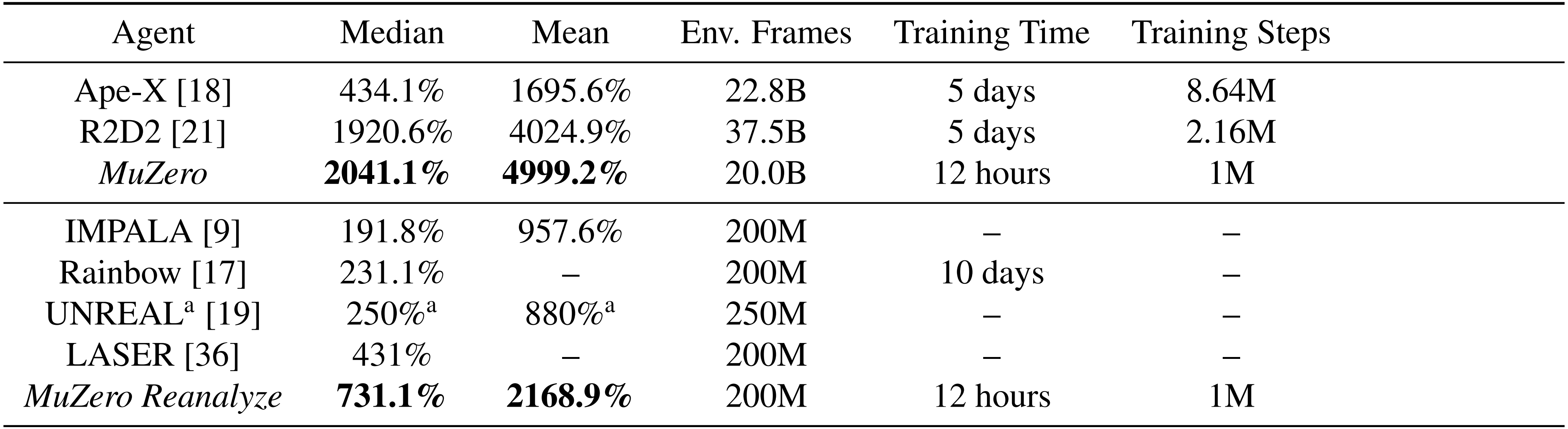


Figure 2 of "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

MuZero – Atari Results



Agent	Median	Mean	Env. Frames	Training Time	Training Steps
Ape-X [18]	434.1%	1695.6%	22.8B	5 days	8.64M
R2D2 [21]	1920.6%	4024.9%	37.5B	5 days	2.16M
MuZero	**2041.1%**	**4999.2%**	20.0B	12 hours	1M
IMPALA [9]	191.8%	957.6%	200M	–	–
Rainbow [17]	231.1%	–	200M	10 days	–
UNREAL[a] [19]	250%[a]	880%[a]	250M	–	–
LASER [36]	431%	–	200M	–	–
MuZero Reanalyze	**731.1%**	**2168.9%**	200M	12 hours	1M

Table 1 of "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

MuZero Reanalyze is optimized for greater sample efficiency. It revisits past trajectories by re-running the MCTS using the network with the latest parameters, notably

- using the fresh policy as target in 80% of the training updates, and
- always using the fresh $v^k \leftarrow f_\theta(s^k)$ in the bootstrapped target z_t.

Some hyperparameters were changed too – 2.0 samples were drawn per state instead of 0.1, the value loss was weighted down to 0.25, and the n-step return was reduced to $n = 5$.

MuZero – Planning Ablations

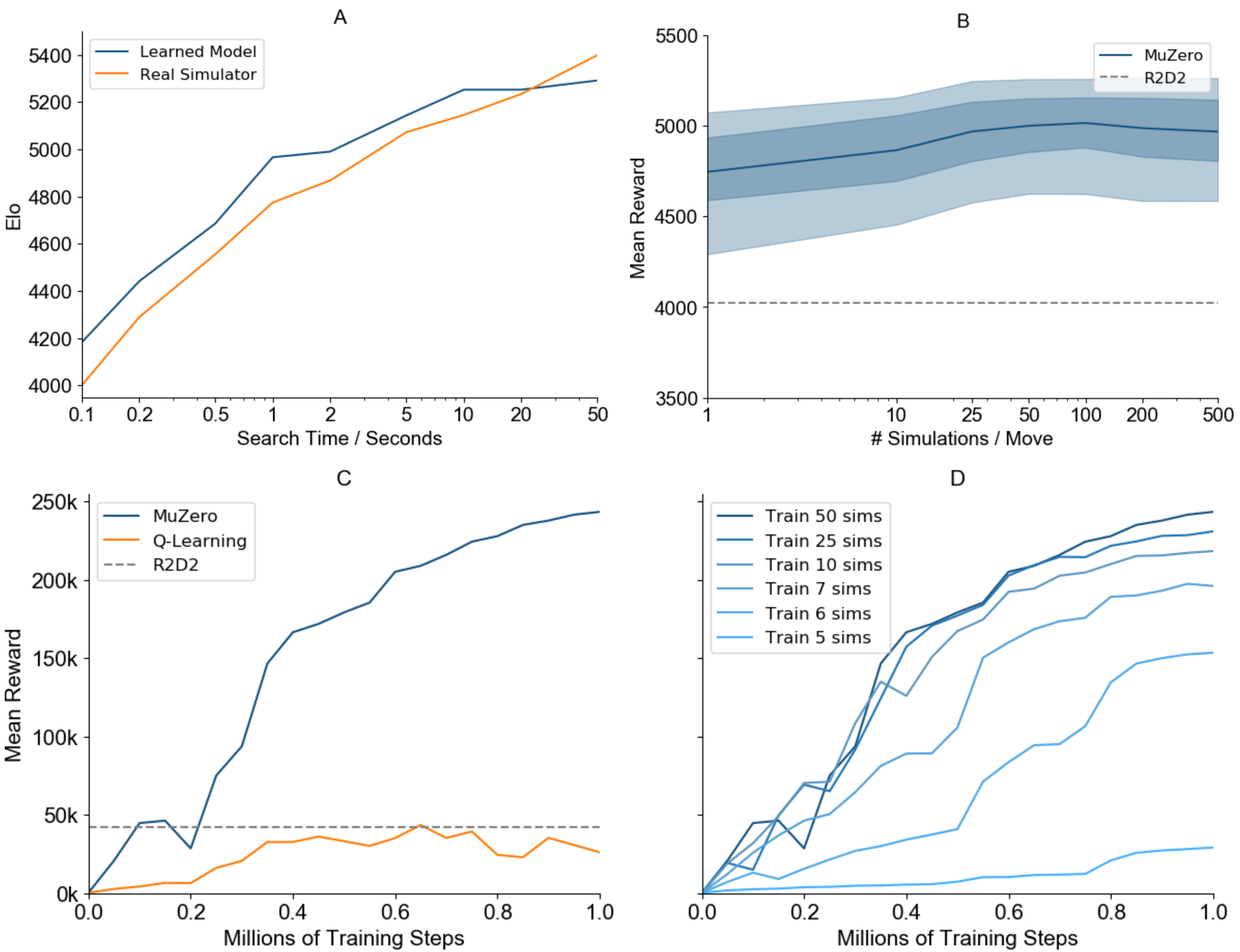


Figure 3 of "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

MuZero – Planning Ablations

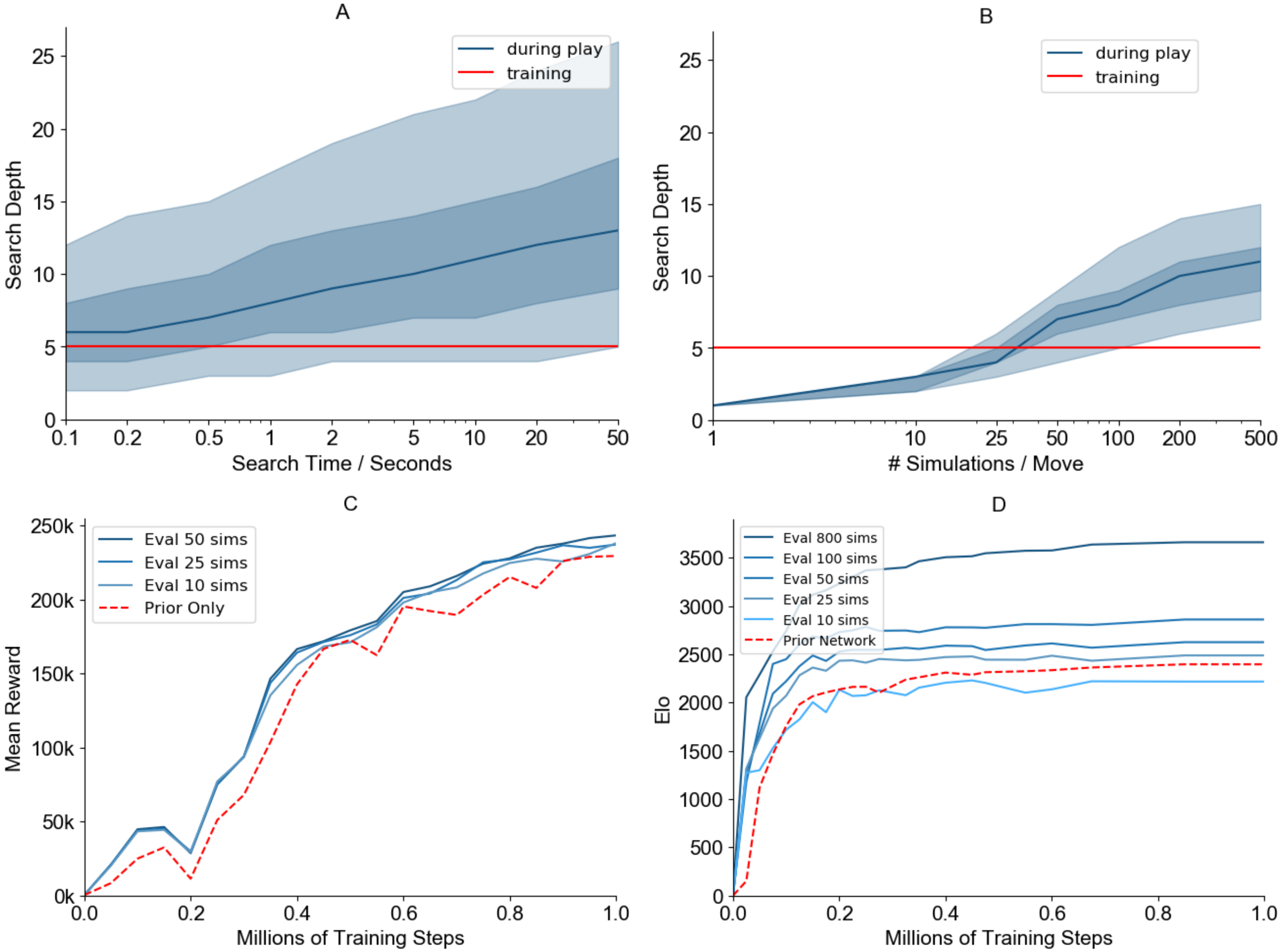


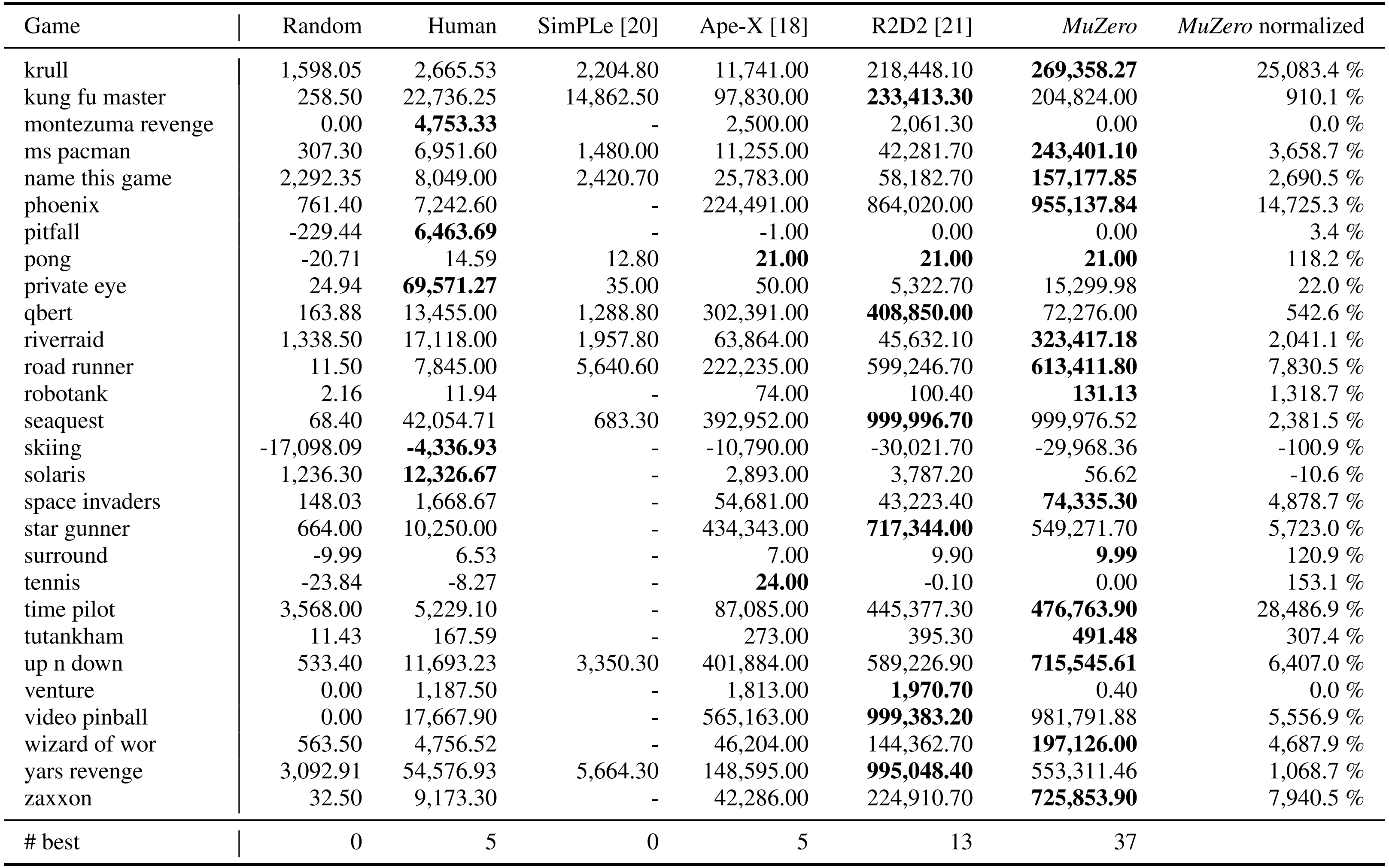
Figure S3 of "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

MuZero – Detailed Atari Results

Game	Random	Human	SimPLe [20]	Ape-X [18]	R2D2 [21]	*MuZero*	*MuZero* normalized
alien	227.75	7,127.80	616.90	40,805.00	229,496.90	**741,812.63**	10,747.5 %
amidar	5.77	1,719.53	74.30	8,659.00	**29,321.40**	28,634.39	1,670.5 %
assault	222.39	742.00	527.20	24,559.00	108,197.00	**143,972.03**	27,664.9 %
asterix	210.00	8,503.33	1,128.30	313,305.00	**999,153.30**	998,425.00	12,036.4 %
asteroids	719.10	47,388.67	793.60	155,495.00	357,867.70	**678,558.64**	1,452.4 %
atlantis	12,850.00	29,028.13	20,992.50	944,498.00	1,620,764.00	**1,674,767.20**	10,272.6 %
bank heist	14.20	753.13	34.20	1,716.00	**24,235.90**	1,278.98	171.2 %
battle zone	2,360.00	37,187.50	4,031.20	98,895.00	751,880.00	**848,623.00**	2,429.9 %
beam rider	363.88	16,926.53	621.60	63,305.00	188,257.40	**454,993.53**	2,744.9 %
berzerk	123.65	2,630.42	-	57,197.00	53,318.70	**85,932.60**	3,423.1 %
bowling	23.11	160.73	30.00	18.00	219.50	**260.13**	172.2 %
boxing	0.05	12.06	7.80	**100.00**	98.50	**100.00**	832.2 %
breakout	1.72	30.47	16.40	801.00	837.70	**864.00**	2,999.2 %
centipede	2,090.87	12,017.04	-	12,974.00	599,140.30	**1,159,049.27**	11,655.6 %
chopper command	811.00	7,387.80	979.40	721,851.00	986,652.00	**991,039.70**	15,056.4 %
crazy climber	10,780.50	35,829.41	62,583.60	320,426.00	366,690.70	**458,315.40**	1,786.6 %
defender	2,874.50	18,688.89	-	411,944.00	665,792.00	**839,642.95**	5,291.2 %
demon attack	152.07	1,971.00	208.10	133,086.00	140,002.30	**143,964.26**	7,906.4 %
double dunk	-18.55	-16.40	-	**24.00**	23.70	23.94	1,976.3 %
enduro	0.00	860.53	-	2,177.00	2,372.70	**2,382.44**	276.9 %
fishing derby	-91.71	-38.80	-90.70	44.00	85.80	**91.16**	345.6 %
freeway	0.01	29.60	16.70	**34.00**	32.50	33.03	111.6 %
frostbite	65.20	4,334.67	236.90	9,329.00	315,456.40	**631,378.53**	14,786.7 %
gopher	257.60	2,412.50	596.80	120,501.00	124,776.30	**130,345.58**	6,036.8 %
gravitar	173.00	3,351.43	173.40	1,599.00	**15,680.70**	6,682.70	204.8 %
hero	1,026.97	30,826.38	2,656.60	31,656.00	39,537.10	**49,244.11**	161.8 %
ice hockey	-11.15	0.88	-11.60	33.00	**79.30**	67.04	650.0 %
jamesbond	29.00	302.80	100.50	21,323.00	25,354.00	**41,063.25**	14,986.9 %
kangaroo	52.00	3,035.00	51.20	1,416.00	14,130.70	**16,763.60**	560.2 %
# best	0	5	0	5	13	37	

Table S1 of "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

MuZero – Detailed Atari Results



Game	Random	Human	SimPLe [20]	Ape-X [18]	R2D2 [21]	*MuZero*	*MuZero* normalized
krull	1,598.05	2,665.53	2,204.80	11,741.00	218,448.10	**269,358.27**	25,083.4 %
kung fu master	258.50	22,736.25	14,862.50	97,830.00	**233,413.30**	204,824.00	910.1 %
montezuma revenge	0.00	**4,753.33**	-	2,500.00	2,061.30	0.00	0.0 %
ms pacman	307.30	6,951.60	1,480.00	11,255.00	42,281.70	**243,401.10**	3,658.7 %
name this game	2,292.35	8,049.00	2,420.70	25,783.00	58,182.70	**157,177.85**	2,690.5 %
phoenix	761.40	7,242.60	-	224,491.00	864,020.00	**955,137.84**	14,725.3 %
pitfall	-229.44	**6,463.69**	-	-1.00	0.00	0.00	3.4 %
pong	-20.71	14.59	12.80	**21.00**	**21.00**	**21.00**	118.2 %
private eye	24.94	**69,571.27**	35.00	50.00	5,322.70	15,299.98	22.0 %
qbert	163.88	13,455.00	1,288.80	302,391.00	**408,850.00**	72,276.00	542.6 %
riverraid	1,338.50	17,118.00	1,957.80	63,864.00	45,632.10	**323,417.18**	2,041.1 %
road runner	11.50	7,845.00	5,640.60	222,235.00	599,246.70	**613,411.80**	7,830.5 %
robotank	2.16	11.94	-	74.00	100.40	**131.13**	1,318.7 %
seaquest	68.40	42,054.71	683.30	392,952.00	**999,996.70**	999,976.52	2,381.5 %
skiing	-17,098.09	**-4,336.93**	-	-10,790.00	-30,021.70	-29,968.36	-100.9 %
solaris	1,236.30	**12,326.67**	-	2,893.00	3,787.20	56.62	-10.6 %
space invaders	148.03	1,668.67	-	54,681.00	43,223.40	**74,335.30**	4,878.7 %
star gunner	664.00	10,250.00	-	434,343.00	**717,344.00**	549,271.70	5,723.0 %
surround	-9.99	6.53	-	7.00	9.90	**9.99**	120.9 %
tennis	-23.84	-8.27	-	**24.00**	-0.10	0.00	153.1 %
time pilot	3,568.00	5,229.10	-	87,085.00	445,377.30	**476,763.90**	28,486.9 %
tutankham	11.43	167.59	-	273.00	395.30	**491.48**	307.4 %
up n down	533.40	11,693.23	3,350.30	401,884.00	589,226.90	**715,545.61**	6,407.0 %
venture	0.00	1,187.50	-	1,813.00	**1,970.70**	0.40	0.0 %
video pinball	0.00	17,667.90	-	565,163.00	**999,383.20**	981,791.88	5,556.9 %
wizard of wor	563.50	4,756.52	-	46,204.00	144,362.70	**197,126.00**	4,687.9 %
yars revenge	3,092.91	54,576.93	5,664.30	148,595.00	**995,048.40**	553,311.46	1,068.7 %
zaxxon	32.50	9,173.30	-	42,286.00	224,910.70	**725,853.90**	7,940.5 %
# best	0	5	0	5	13	37	

Table S1 of "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

AlphaZero as Regularized Policy Optimization

Recall that in AlphaZero, actions are selected according to a variant of PUCT algorithm:

$$a^* = \arg\max_a \left(Q(s,a) + C(s)P(s,a)\frac{\sqrt{N(s)}}{1+N(s,a)}\right),$$

with a slightly time-increasing exploration rate $C(s) = \log\left(\frac{1+N(s)+19625}{19625}\right) + 1.25 \approx 1.25$.

The paper *Jean-Bastien Grill et al.: Monte-Carlo Tree Search as Regularized Policy Optimization*, the authors have shown how to interpret this algorithm as a regularized policy optimization.

AlphaZero as Regularized Policy Optimization

Policy optimization is usually an iterative procedure, which in every step improves a current policy $\pi_{\boldsymbol\theta_0}$ according to

$$\pi_{\boldsymbol\theta'} \overset{\text{def}}{=} \operatorname*{arg\,max}_{\boldsymbol y \in \mathcal{S}} \boldsymbol q_{\pi_{\boldsymbol\theta_0}}^T \boldsymbol y - \mathcal{R}(\boldsymbol y, \pi_{\boldsymbol\theta_0}),$$

where $\mathcal{S}$ is a $|\mathcal{A}|$-dimensional simplex and $\mathcal{R}: \mathcal{S}^2 \rightarrow \mathbb{R}$ is an optional (usually convex) regularization term.

- with $\mathcal{R} = 0$, the above reduces to policy iteration (used for example in DQN);
- with $\mathcal{R} = 0$, if the policy is updated using a single gradient step, the algorithm reduces to policy gradient;
- when $\mathcal{R}(\boldsymbol y, \pi_{\boldsymbol\theta_0}) = -H(\boldsymbol y)$, we recover the Soft Actor Critic objective;
- for $\mathcal{R}(\boldsymbol y, \pi_{\boldsymbol\theta_0}) = D_{\mathrm{KL}}(\pi_{\boldsymbol\theta_0} \| \boldsymbol y)$ we get an analogue of the TRPO objective, which motivated PPO;
- the MPO algorithm (which we did not discuss) employs $\mathcal{R}(\boldsymbol y, \pi_{\boldsymbol\theta_0}) = D_{\mathrm{KL}}(\boldsymbol y \| \pi_{\boldsymbol\theta_0})$.

AlphaZero as Regularized Policy Optimization

Let us define the **empirical visit distribution** $\hat\pi$ as

$$\hat\pi(a|s) \overset{\text{def}}{=} \frac{1 + N(s, a)}{|\mathcal{A}| + \sum_b N(s, b)}.$$

The added plus ones makes the following analysis easier, but are not strictly necessary.

We also define the **multiplier** λ_N as

$$\lambda_N(s) \overset{\text{def}}{=} C(s) \cdot \frac{\sqrt{\sum_b N(s, b)}}{|\mathcal{A}| + \sum_b N(s, b)}.$$

With these definitions, we can rewrite the AlphaZero action selection to

$$a^* = \arg\max_a \left(Q(s, a) + \lambda_N \cdot \frac{\pi_{\boldsymbol\theta}(a|s)}{\hat\pi(a|s)}\right).$$

AlphaZero as Regularized Policy Optimization

$$a^* = \arg\max_a \left(Q(s,a) + \lambda_N \cdot \frac{\pi_{\boldsymbol\theta}(a|s)}{\hat\pi(a|s)}\right)$$

For notational simplicity, we will represent $Q(s,a)$ as a vector $\boldsymbol q$, where $q_a = Q(s,a)$, and similarly the policies as $\boldsymbol\pi_{\boldsymbol\theta}$, $\hat{\boldsymbol\pi}$.

Furthermore, for two vectors $\boldsymbol a, \boldsymbol b$, let $\frac{\boldsymbol a}{\boldsymbol b}$ denote element-wise division with $\left(\frac{\boldsymbol a}{\boldsymbol b}\right)_i \overset{\text{def}}{=} \frac{a_i}{b_i}$.

With this notation, the action selection can be succinctly written as

$$a^* = \arg\max_a \left(\boldsymbol q + \lambda_N \frac{\boldsymbol\pi_{\boldsymbol\theta}}{\hat{\boldsymbol\pi}}\right).$$

AlphaZero as Regularized Policy Optimization

Let $\bar{\boldsymbol\pi}$ be the solution of the following objective:

$$\bar{\boldsymbol\pi} \stackrel{\text{def}}{=} \arg\max_{\boldsymbol y \in \mathcal{S}} \Big(\boldsymbol q^T \boldsymbol y - \lambda_N D_\textrm{KL}(\boldsymbol\pi_{\boldsymbol\theta} \| \boldsymbol y)\Big).$$

It can be shown that this solution can be computed explicitly as

$$\bar{\boldsymbol\pi} = \lambda_N \frac{\boldsymbol\pi_{\boldsymbol\theta}}{\alpha - \boldsymbol q},$$

where $\alpha \in \mathbb{R}$ is set such that the result is a proper distribution.

- Note that $\alpha \geq \max_{b \in \mathcal{A}} \big(q_b + \lambda_N \pi_{\boldsymbol\theta}(b)\big)$, because $\bar\pi(a)$ must be at most 1.
- Furthermore, $\alpha \leq \max_{b \in \mathcal{A}}(q_b) + \lambda_N$, because we need $\sum_a \bar\pi(a) = 1$ and we combine $\sum_a \frac{\lambda_N \pi_{\boldsymbol\theta}(a)}{\max_b(q_b) + \lambda_N - q_a} \leq \sum_a \frac{\lambda_N \pi_{\boldsymbol\theta}(a)}{\lambda_N} = 1$ with the fact that $\sum_a \frac{\lambda_N \pi_{\boldsymbol\theta}(a)}{\alpha - q_a}$ is a decreasing function of $\alpha \geq \max_b q_b$.

Note the $\lambda_N \approx 1/\sqrt{N}$ decreasing the regularization for increasing number of simulations.

AlphaZero as Regularized Policy Optimization

The action a^* selected by the AlphaZero algorithm fulfills

$$a^* = \arg\max_a \left(\frac{\partial}{\partial N(s, a)} \left(\boldsymbol{q}^T \hat{\boldsymbol{\pi}} - \lambda_N D_{\mathrm{KL}}(\boldsymbol{\pi_\theta} \| \hat{\boldsymbol{\pi}}) \right) \right).$$

In other words, $\hat\pi$ “tracks” $\bar\pi$.

Furthermore, it can be also shown that for the selected action a^*,

$$\hat\pi(a^*|s) \le \bar\pi(a^*|s),$$

until in the limit, the two distributions coincide.

AlphaZero as Regularized Policy Optimization

The $\bar\pi$ can be used in the AlphaZero algorithm in several ways:

- **Act**: the action in self-play games could be sampled according to $\bar\pi(\cdot|s_\mathrm{root})$ instead of $\hat\pi$;
- **Search**: during search, we could sample the actions stochastically according to $\bar\pi$ instead of the PUCT rule;
- **Learn**: we could use $\bar\pi$ as the target policy during training instead of $\hat\pi$;
- **All**: all of the above.

AlphaZero as Regularized Policy Optimization

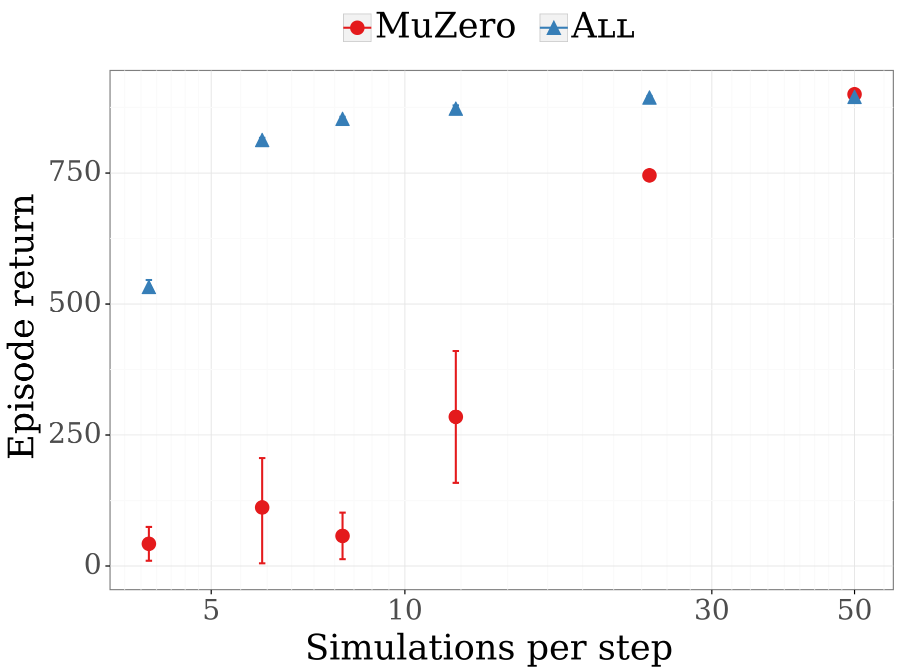


Figure 1. Comparison of the score (median score over 3 seeds) of MuZero (red: using $\hat{\pi}$) and ALL (blue: using $\bar{\pi}$) after 100k learner steps as a function of N_{sim} on Cheetah Run of the Control Suite.

Figure 1 of "Monte-Carlo Tree Search as Regularized Policy Optimization", https://arxiv.org/abs/2007.12509

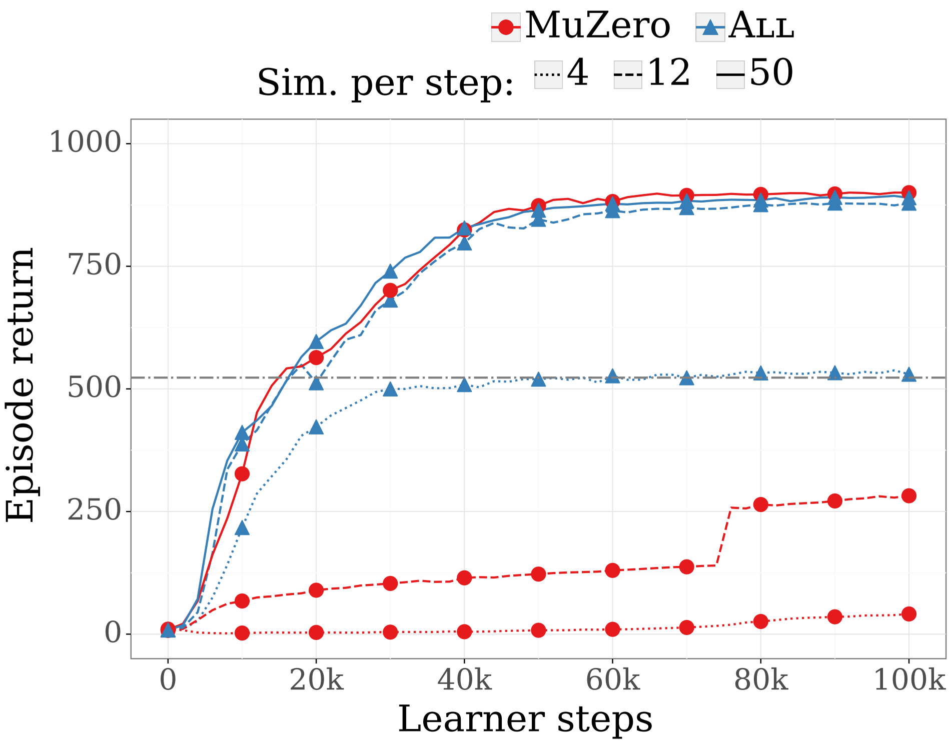


Figure 5. Comparison of the median score over 3 seeds of MuZero (red) and ALL (blue) at 4 (dotted) and 50 (solid line) simulations per step on Cheetah Run (Control Suite).

Figure 5 of "Monte-Carlo Tree Search as Regularized Policy Optimization", https://arxiv.org/abs/2007.12509

AlphaZero as Regularized Policy Optimization

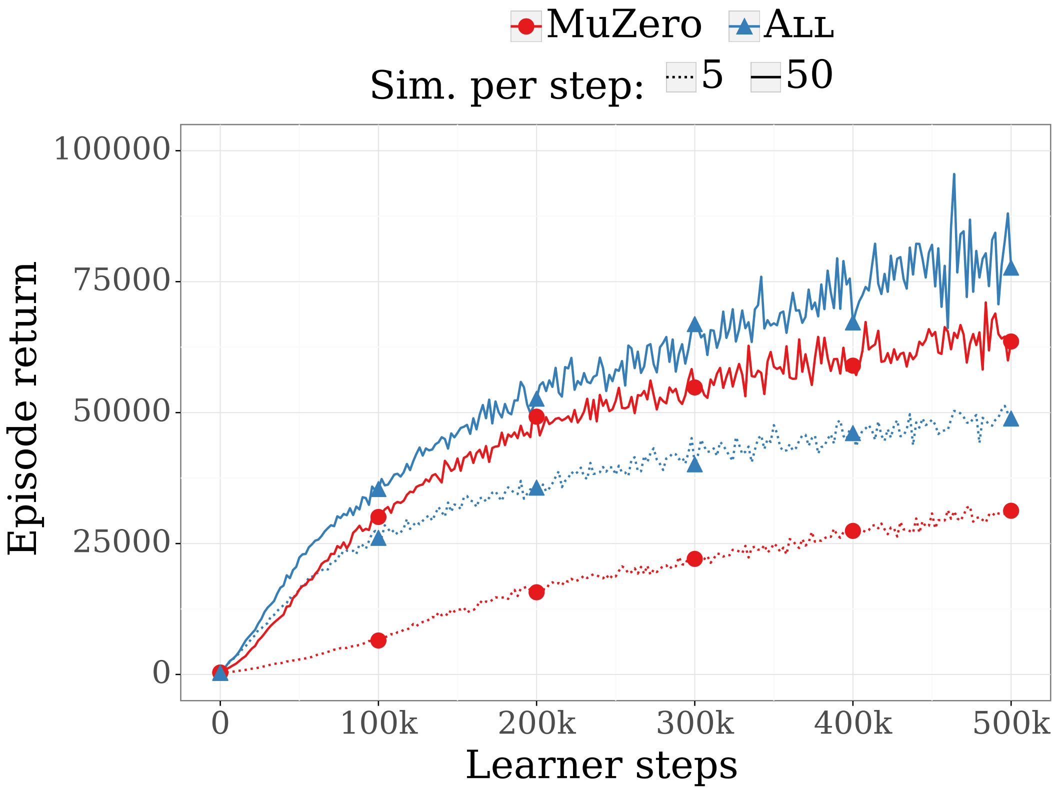


Figure 2. Comparison of median scores of MuZero (red) and ALL (blue) at $N_{\text{sim}} = 5$ (dotted line) and $N_{\text{sim}} = 50$ (solid line) simulations per step on Ms Pacman (Atari). Averaged across 8 seeds.

Figure 2 of "Monte-Carlo Tree Search as Regularized Policy Optimization", https://arxiv.org/abs/2007.12509

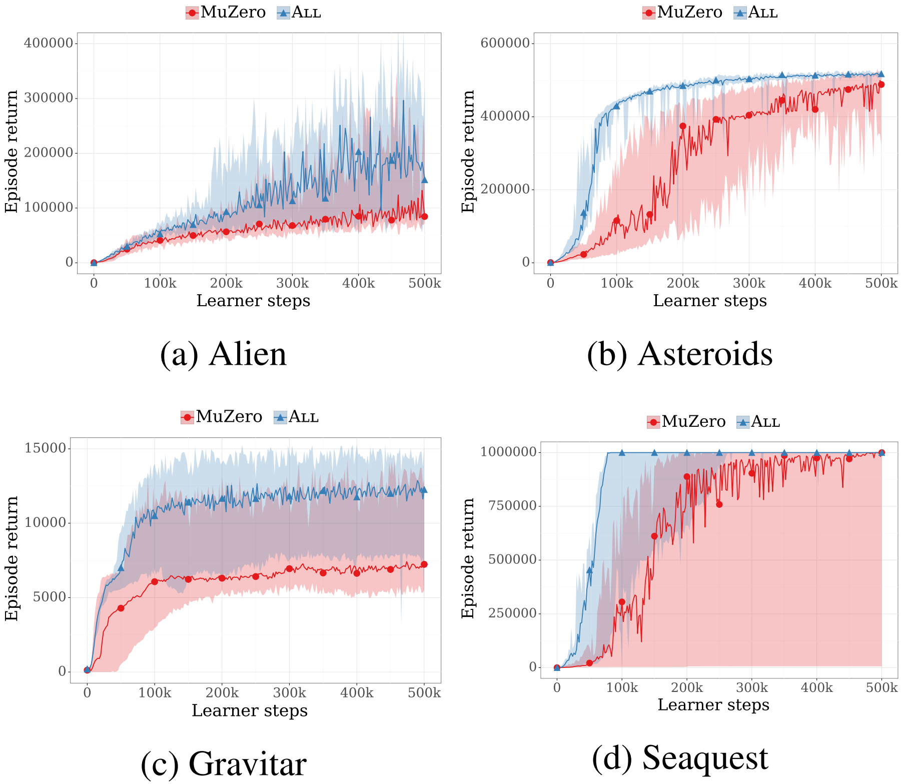
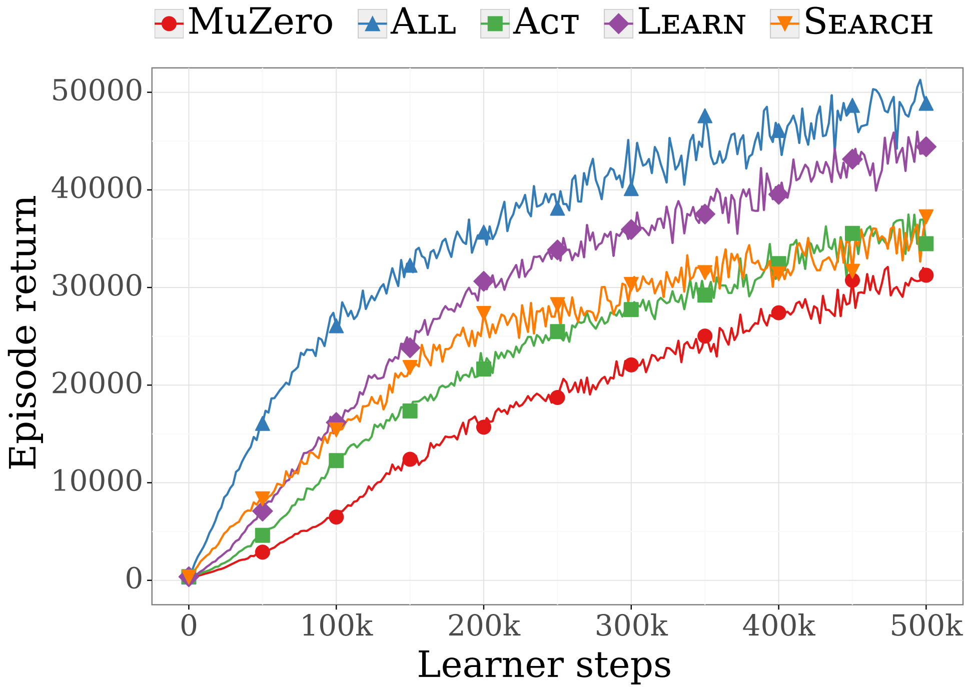


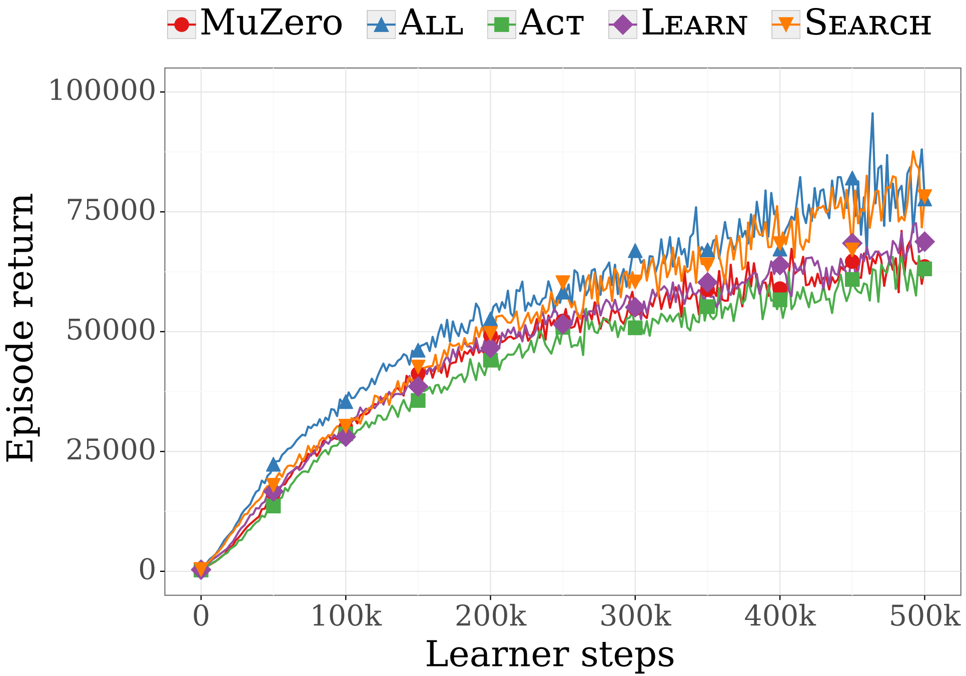
Figure 3. Comparison of median score (solid lines) over 6 seeds of MuZero and ALL on four Atari games with $N_{\text{sim}} = 50$. The shaded areas correspond to the range of the best and worst seed. ALL (blue) performs consistently better than MuZero (red).

Figure 3 of "Monte-Carlo Tree Search as Regularized Policy Optimization", https://arxiv.org/abs/2007.12509

AlphaZero as Regularized Policy Optimization



(a) 5 simulations



(b) 50 simulations

Figure 4. Ablation study at 5 and 50 simulations per step on Ms Pacman (Atari); average across 8 seeds.

Figure 4 of "Monte-Carlo Tree Search as Regularized Policy Optimization", https://arxiv.org/abs/2007.12509

Gumbel-Softmax

The **Gumbel-softmax** distribution was proposed independently in two papers in Nov 2016 (under the name of **Concrete** distribution in the other paper).

It is a continuous distribution over the simplex (over categorical distributions) that can approximate *sampling* from a categorical distribution.

Let z be a categorical variable with class probabilities $\boldsymbol{p} = (p_1, p_2, \ldots, p_K)$.

The Gumbel-Max trick (based on a 1954 theorem from E. J. Gumbel) states that we can draw samples $z \sim \boldsymbol{p}$ using

$$z = \operatorname{one-hot}\Big(\arg\max_i \big(g_i + \log p_i\big)\Big),$$

where g_i are independent samples drawn from the $\operatorname{Gumbel}(0, 1)$ distribution.

To sample g from the distribution $\operatorname{Gumbel}(0, 1)$, we can sample $u \sim U(0, 1)$ and then compute $g = -\log(-\log u)$.

Gumbel Distribution

First recall that exponential distribution $\operatorname{Exp}(\lambda)$ has

$$\operatorname{PDF}(x;\lambda) = \lambda e^{-\lambda x}, \quad \operatorname{CDF}(x;\lambda) = 1 - e^{-\lambda x}.$$

The standard $\operatorname{Gumbel}(0,1)$ distribution has

$$\operatorname{PDF}(x) = e^{-x-e^{-x}}, \quad \operatorname{CDF}(x) = e^{-e^{-x}}.$$

The Gumbel distribution can be used to model the distribution of maximum of a number of samples from the exponential distribution: if $\tilde x$ is a maximum of N samples from the $\operatorname{Exp}(1)$ distribution, we get that

$$P(\tilde x - \log N \le x) = P(\tilde x \le x + \log N) = \operatorname{CDF}_{\operatorname{Exp}(1)}\big(x + \log N\big)^N = \Big(1 - \frac{e^{-x}}{N}\Big)^N,$$

which converges to $e^{-e^{-x}} = \operatorname{CDF}_{\operatorname{Gumbel}(0,1)}(x)$ for $N \to \infty$.

Gumbel-Max Trick Proof

To prove the Gumbel-Max trick, we first reformulate it slightly.

Let l_i be logits of a categorical distribution (so that the class probabilities $\pi_i \propto e^{l_i}$), and let $g_i \sim \operatorname{Gumbel}(0, 1)$. Then

$$\pi_k = P\big(k = \arg\max_i (g_i + l_i)\big).$$

We first observe that the theorem is invariant to a scalar shift of logits, so we can without loss of generality assume that $\sum_i e^{l_i} = 1$ and $\pi_i = e^{l_i}$.

For convenience, denote $u_i \overset{\text{def}}{=} g_i + l_i$.

We will use both the PDF and CDF of a $\operatorname{Gumbel}(0, 1)$ distribution:

$$\begin{aligned}\operatorname{PDF}(g_i) &= e^{-g_i - e^{-g_i}},\\ \operatorname{CDF}(g_i) &= e^{-e^{-g_i}}.\end{aligned}$$

Gumbel-Max Trick Proof

To finish the proof, we compute

$$\begin{aligned}
P\Big(k = \arg\max_i (g_i + l_i)\Big) &= P(u_k \geq u_i, \forall_{i \neq k}) \\
&= \int P(u_k) \prod\nolimits_{i \neq k} P(u_k \geq u_i | u_k) \,\mathrm{d}u_k \\
&= \int P(g_k | g_k = u_k - l_k) \prod\nolimits_{i \neq k} P(g_i \leq u_k - l_i | u_k) \,\mathrm{d}u_k \\
&= \int e^{l_k - u_k - e^{l_k - u_k}} \prod\nolimits_{i \neq k} e^{-e^{l_i - u_k}} \,\mathrm{d}u_k \\
&= \int \pi_k e^{-u_k - \pi_k e^{-u_k}} \prod\nolimits_{i \neq k} e^{-\pi_i e^{-u_k}} \,\mathrm{d}u_k \\
&= \pi_k \int e^{-u_k - e^{-u_k} \sum_i \pi_i} \,\mathrm{d}u_k \\
&= \pi_k \int e^{-g_k - e^{-g_k}} \,\mathrm{d}g_k = \pi_k \cdot 1 = \pi_k .
\end{aligned}$$

Gumbel-Softmax

To obtain a continuous distribution, we relax the $\arg\max$ into a $\operatorname{softmax}$ with temperature T as

$$z_i = \frac{e^{(g_i + \log p_i)/T}}{\sum_j e^{(g_j + \log p_j)/T}}.$$

As the temperature T goes to zero, the generated samples become one-hot, and therefore the Gumbel-softmax distribution converges to the categorical distribution $p(z)$.

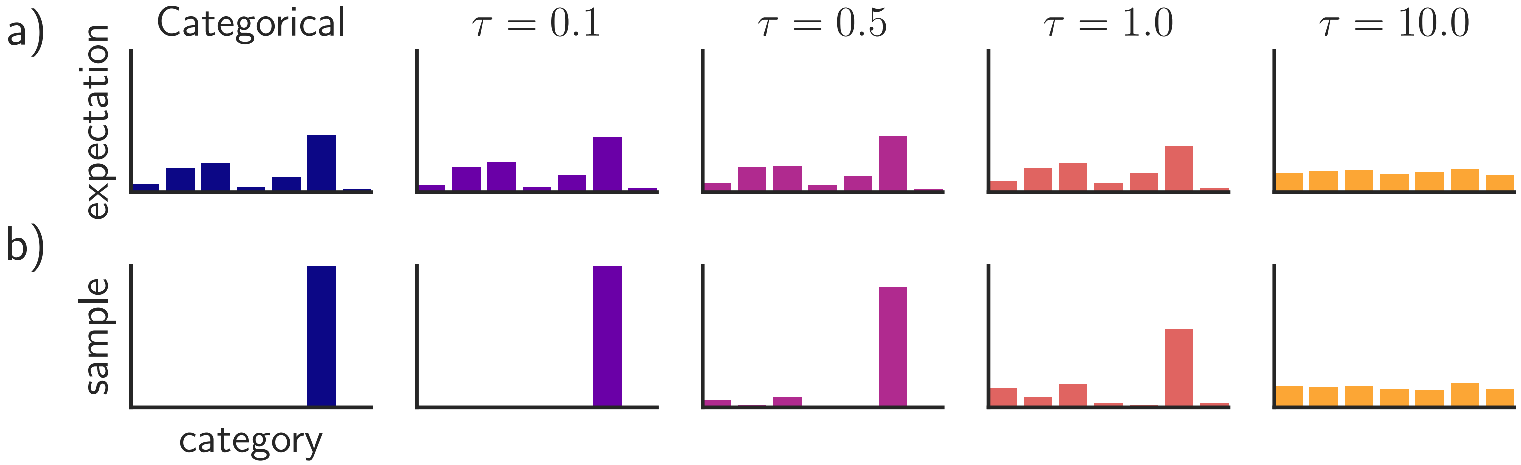


Figure 1 of "Categorical Reparameterization with Gumbel-Softmax", https://arxiv.org/abs/1611.01144

Gumbel AlphaZero and MuZero

In AlphaZero, using the MCTS visit counts as the target policy fails to improve the policy for small number of visits.

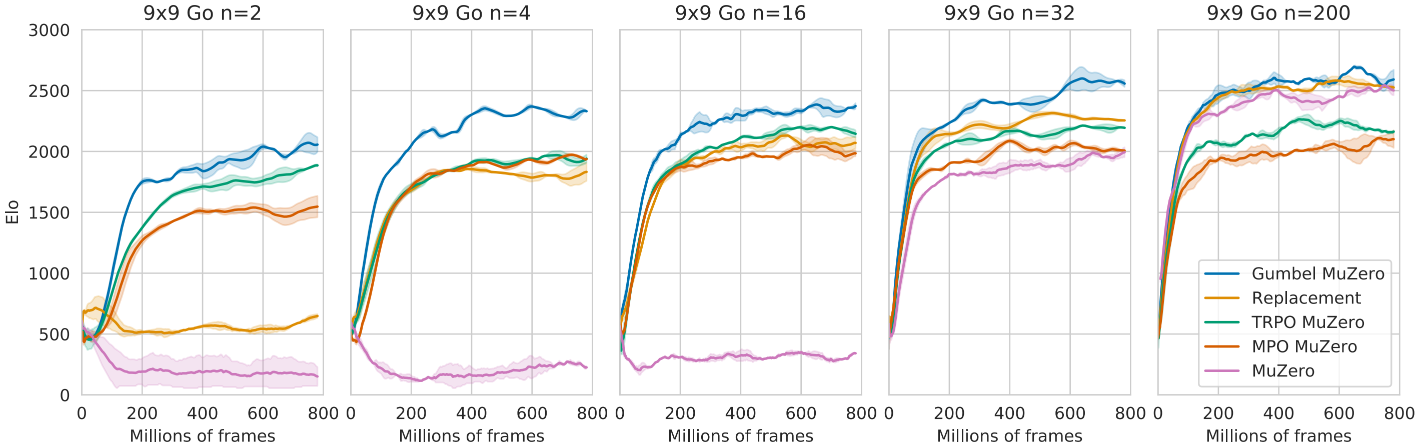


Figure 2: Elo on 9x9 Go, when training with $n \in \{2, 4, 16, 32, 200\}$ simulations. Evaluation uses 800 simulations. Shades denote standard errors from 2 seeds.

Figure 2 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO

In *Ivo Danihelka et al.: Policy Improvement by Planning with Gumbel*, several AlphaZero/MuZero improvements are proposed; among other a different target policy, which guarantees improvement.

Gumbel AlphaZero and MuZero

Let π be a categorical distributions parametrized with $\mathit{logits}(a)$. Let $\boldsymbol{g} \in \mathbb{R}^k$ be a vector of independent Gumbel(0, 1) random variables.

The Gumbel-Max trick states that

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

has a distribution $A \sim \pi$.

The Gumbel-Max trick can be generalized to **Gumbel-Top-k** trick, capable of producing n actions without replacement by considering the top n scoring actions $\operatorname{argtop}(\boldsymbol{g} + \boldsymbol{\mathit{logits}}, n)$:

$$\begin{aligned}
A_1 &= \arg\max_a \big(g(a) + \mathit{logits}(a)\big),\\
A_2 &= \arg\max_{a \neq A_1} \big(g(a) + \mathit{logits}(a)\big),\\
&\;\vdots\\
A_n &= \arg\max_{a \notin \{A_1,\ldots,A_{n-1}\}} \big(g(a) + \mathit{logits}(a)\big).
\end{aligned}$$

A. Guaranteed Policy Improvement

For a small number of simulations, PUCT does not guarantee policy improvement.

- Consider for example three actions with prior policy $(50\%, 30\%, 20\%)$ and action values $(0, 0, 1)$.
- The PUCT rule will select the first two actions.
- However, the value function of any policy considering just the first two actions is 0, which is worse than the value function of the prior policy.

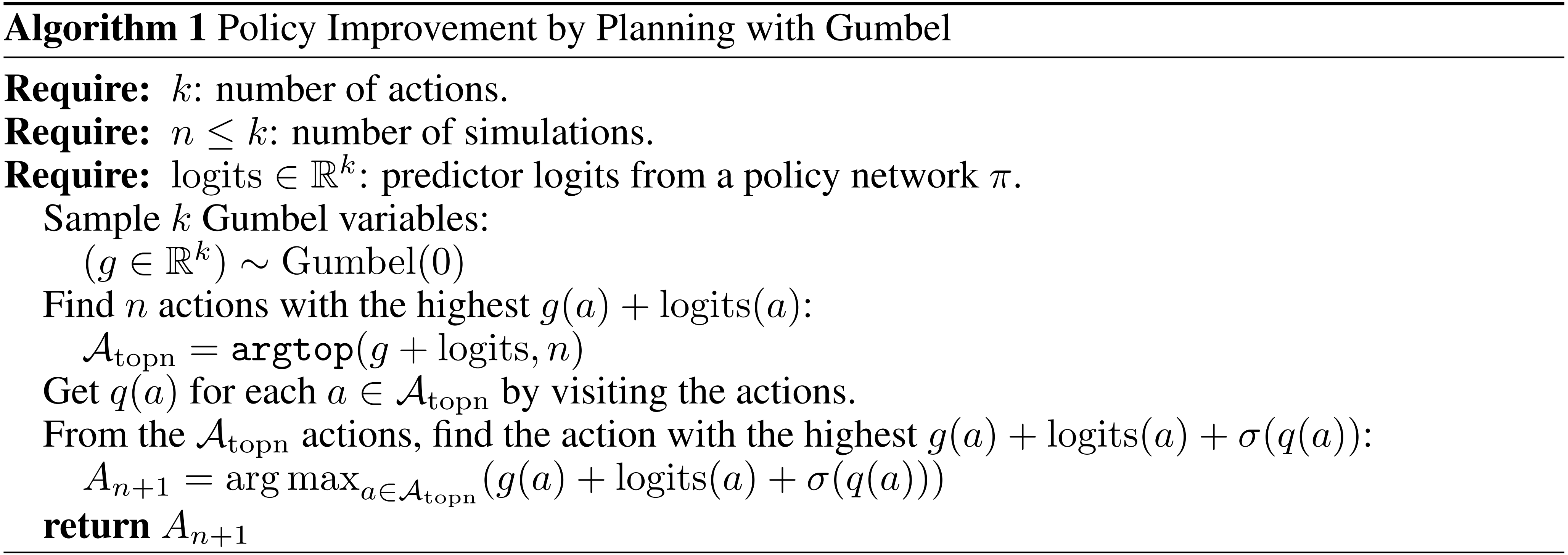
In GumbelZero, we start by sampling n actions without replacement using the Gumbel-Max trick with Gumbel noise $\boldsymbol{g}$.

Our first attempt is to define a one-hot policy selecting an action A_{n+1} such that

$$A_{n+1} = \operatorname*{arg\,max}_{a \in \{A_1, \ldots, A_n\}} \big(g(a) + \mathit{logits}(a) + \sigma(q(a))\big),$$

where σ can be any monotonically increasing transformation.

GumbelZero, A. Guaranteed Policy Improvement



Algorithm 1 Policy Improvement by Planning with Gumbel

```
Require: $k$: number of actions.
Require: $n \leq k$: number of simulations.
Require: $\text{logits} \in \mathbb{R}^k$: predictor logits from a policy network $\pi$.
    Sample $k$ Gumbel variables:
        $(g \in \mathbb{R}^k) \sim \text{Gumbel}(0)$
    Find $n$ actions with the highest $g(a) + \text{logits}(a)$:
        $\mathcal{A}_{\text{topn}} = \texttt{argtop}(g + \text{logits}, n)$
    Get $q(a)$ for each $a \in \mathcal{A}_{\text{topn}}$ by visiting the actions.
    From the $\mathcal{A}_{\text{topn}}$ actions, find the action with the highest $g(a) + \text{logits}(a) + \sigma(q(a))$:
        $A_{n+1} = \arg\max_{a \in \mathcal{A}_{\text{topn}}} (g(a) + \text{logits}(a) + \sigma(q(a)))$
    return $A_{n+1}$
```

Algorithm 1 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO

GumbelZero, A. Guaranteed Policy Improvement

A policy choosing the action

$$A_{n+1} = \operatorname*{arg\,max}_{a \in \{A_1, \ldots, A_n\}} \big(g(a) + \mathit{logits}(a) + \sigma(q(a))\big),$$

guarantees policy improvement, i.e., $\mathbb{E}[q(A_{n+1})] \geq \mathbb{E}_{a\sim\pi}[q(a)]$.

Considering a fixed Gumbel noise $\boldsymbol{g}$, we get that

$$q\big(\operatorname*{arg\,max}_{a \in \{A_1, \ldots, A_n\}} g(a) + \mathit{logits}(a) + \sigma(q(a))\big) \geq q\big(\operatorname*{arg\,max}_{a \in \{A_1, \ldots, A_n\}} g(a) + \mathit{logits}(a)\big),$$

- either the action chosen on both sides is the same and we get an equality, or
- the action on the left side is different, meaning it has larger $q(a)$.

Finally, if the inequality holds for any $\boldsymbol{g}$, it holds also in expectation. With the Gumbel-Max trick transforming the expectation of the right side to sampling an action $a \sim \pi$, we get the required $\mathbb{E}[q(A_{n+1})] \geq \mathbb{E}_{a\sim\pi}[q(a)]$.

B. Planning on Stochastic Bandit

When we get only an estimate $\hat{q}(a)$ of the action-value function, it is probably beneficial to visit an action multiple times.

Furthermore, choosing actions in the root using a UCB-like rule is not optimal:

- UCB minimizes cumulative regret, i.e., maximizes the sum of the obtained returns;
- in the root our goal is to obtain the best possible A_{n+1}, i.e., maximize just $\mathbb{E}[q(A_{n+1})]$.

The authors evaluated several simple regret minimization algorithms, and chose Sequential Halving (because it was easier to tune and does not have problem-dependent parameters).

GumbelZero, B. Planning on Stochastic Bandit

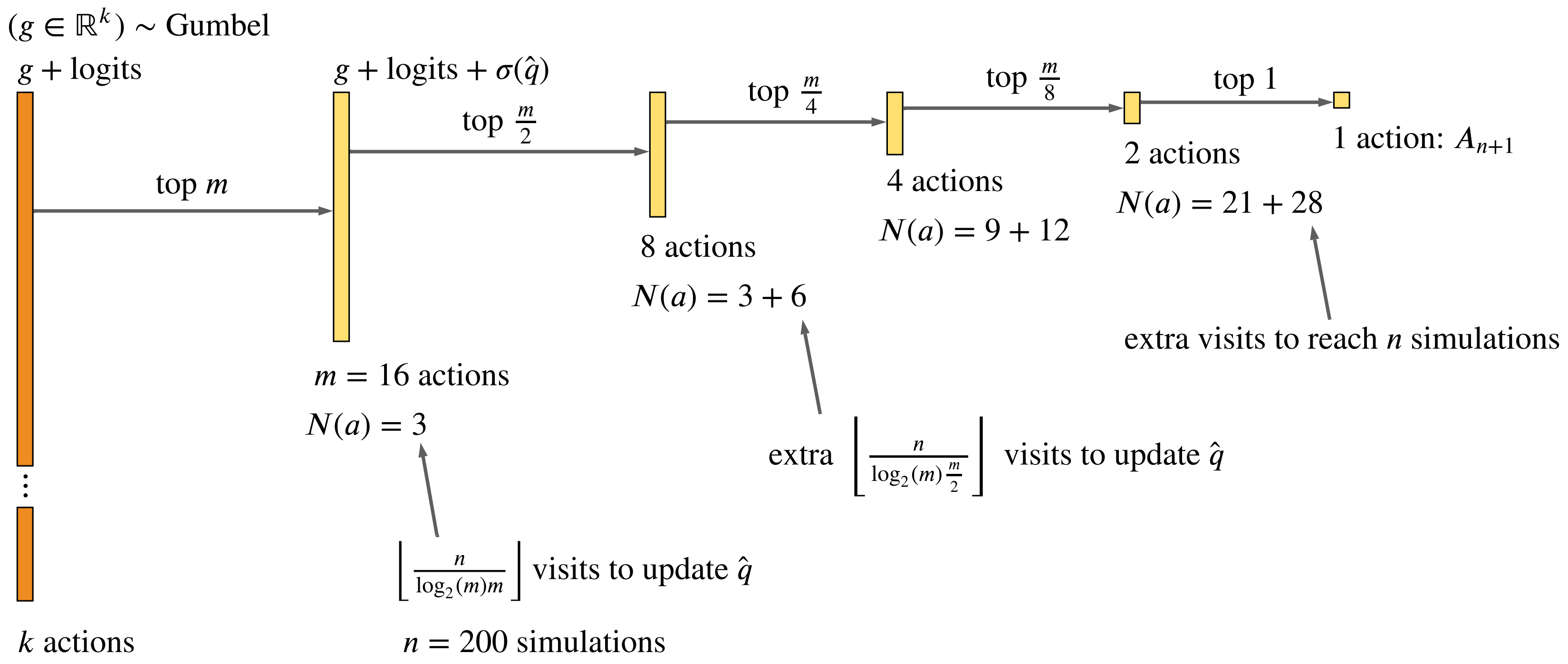

$(g \in \mathbb{R}^k) \sim$ Gumbel
$g +$ logits
top m
k actions
$g +$ logits $+ \sigma(\hat{q})$
top $\frac{m}{2}$
$m = 16$ actions
$N(a) = 3$
$\left\lfloor \frac{n}{\log_2(m)m} \right\rfloor$ visits to update $\hat{q}$
$n = 200$ simulations
top $\frac{m}{4}$
8 actions
$N(a) = 3 + 6$
extra $\left\lfloor \frac{n}{\log_2(m)\frac{m}{2}} \right\rfloor$ visits to update $\hat{q}$
top $\frac{m}{8}$
4 actions
$N(a) = 9 + 12$
top 1
2 actions
$N(a) = 21 + 28$
1 action: A_{n+1}
extra visits to reach n simulations

Figure 1: The number of considered actions and their visit counts $N(a)$, when using Sequential Halving with Gumbel on a k-armed stochastic bandit. The search uses $n = 200$ simulations and starts by sampling $m = 16$ actions without replacement. Sequential Halving divides the budget of n simulations equally to $\log_2(m)$ phases. In each phase, all considered actions are visited equally often. After each phase, one half of the actions is rejected. From the original k actions, only the best actions will remain.

Figure 1 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO

GumbelZero, B. Planning on Stochastic Bandit

Algorithm 2 Sequential Halving with Gumbel

```
**Require:** $k$: number of actions.
**Require:** $m \le k$: number of actions sampled without replacement.
**Require:** $n$: number of simulations.
**Require:** logits $\in \mathbb{R}^k$: predictor logits from a policy network $\pi$.
    Sample $k$ Gumbel variables:
        $(g \in \mathbb{R}^k) \sim \mathrm{Gumbel}(0)$
    Find $m$ actions with the highest $g(a) + \mathrm{logits}(a)$:
        $\mathcal{A}_{\mathrm{topm}} = \mathtt{argtop}(g + \mathrm{logits}, m)$
    Use Sequential Halving with $n$ simulations to identify the best action from the $\mathcal{A}_{\mathrm{topm}}$ actions,
    by comparing $g(a) + \mathrm{logits}(a) + \sigma(\hat{q}(a))$.
    $A_{n+1} = \arg\max_{a \in Remaining} (g(a) + \mathrm{logits}(a) + \sigma(\hat{q}(a)))$
    **return** $A_{n+1}$
```

Algorithm 2 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO

The authors utilize $m = \min(n, 16)$, and visit each action at least once even when n is small by visiting each action $\max\left(1, \left\lfloor \frac{n}{\lceil \log_2 m \rceil m} \right\rfloor\right)$; after n simulation, the search is always stopped.

GumbelZero, C. Better Improved Policy

Using a one-hot policy based on

$$A_{n+1} = \underset{a \in \{A_1, \ldots, A_n\}}{\arg\max} \big(g(a) + \mathit{logits}(a) + \sigma(q(a))\big)$$

results in using a simple policy loss

$$L_\textrm{simple}(\pi) = -\log \pi(A_{n+1}).$$

GumbelZero, C. Better Improved Policy

However, more information from the search might be extracted by using all action-value functions $q(a)$ produced by the search.

- First, we complete the action values using

$$\mathit{completedQ}(a) \overset{\text{def}}{=} \begin{cases} q(a) & \text{if } N(a) > 0, \\ v_\pi & \text{otherwise.} \end{cases}$$

- Then, we define improved policy as

$$\pi' = \operatorname{softmax}\big(\mathit{logits}(a) + \sigma(\mathit{completedQ}(a))\big).$$

  It can be again proven (appendix C of the paper) that π' is an improved policy, so $\mathbb{E}_{a\sim\pi'}[q(a)] \geq \mathbb{E}_{a\sim\pi}[q(a)]$.

- A natural loss is then

$$L_\text{completed}(\pi) = D_\text{KL}(\pi' \| \pi).$$

GumbelZero, C. Better Improved Policy

The authors propose to use

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

for $c_\text{visit} = 50$, $c_\text{scale} = 1.0$.

Furthermore, the authors propose a consistent approximation to v_π based on a network-predicted $\hat v_\pi$ and the $q(a)$ of the visited actions:

$$v_\text{mix} \overset{\text{def}}{=} \frac{1}{1 + \sum_b N(b)} \Bigg(\hat v_\pi + \Big(\sum\nolimits_b N(b)\Big) \frac{\sum_{a,N(a)>0} \pi(a) q(a)}{\sum_{a,N(a)>0} \pi(a)} \Bigg).$$

Overall, the algorithm denoted in the paper as Gumbel MuZero utilizes Sequential Halving with Gumbel and trains using the improved policy combining logits and action values completed by v_mix; otherwise it is the same as MuZero.

D. Action Selection in Non-Root Nodes

We might consider utilizing the improved policy π' also in the non-root nodes, by for example sampling actions from it. Additionally, the authors provide a deterministic algorithm of choosing non-root actions minimizing the difference between π' and the current visit counts:

$$a^* = \arg\min_a \sum_b \left(\pi'(b) - \underbrace{\frac{N(b) + [a = b]}{1 + \sum_c N(c)}}_{\text{normalized visit counts if taking } a}\right)^2.$$

This formula can be simplified to

$$a^* = \arg\max_a \left(\pi'(a) - \frac{N(a)}{1 + \sum_b N(b)}\right).$$

When this action selection is used, the authors call the algorithm Full Gumbel MuZero.

GumbelZero, D. Action Selection in Non-Root Nodes

$$\begin{aligned}
a^* &= \arg\min_a \sum_b \left(\pi'(b) - \frac{N(b) + [a=b]}{1 + \sum_c N(c)}\right)^2 \\
&= \arg\min_a \sum_b \left(\left(\pi'(b) - \frac{N(b)}{1 + \sum_c N(c)}\right) - \frac{[a=b]}{1 + \sum_c N(c)}\right)^2 \\
&= \arg\min_a \sum_b -2\left(\pi'(b) - \frac{N(b)}{1 + \sum_c N(c)}\right)\frac{[a=b]}{1 + \sum_c N(c)} \\
&= \arg\min_a -\sum_b \left(\pi'(b) - \frac{N(b)}{1 + \sum_c N(c)}\right)[a=b] \\
&= \arg\max_a \left(\pi'(a) - \frac{N(a)}{1 + \sum_b N(b)}\right)
\end{aligned}$$

Gumbel AlphaZero and MuZero

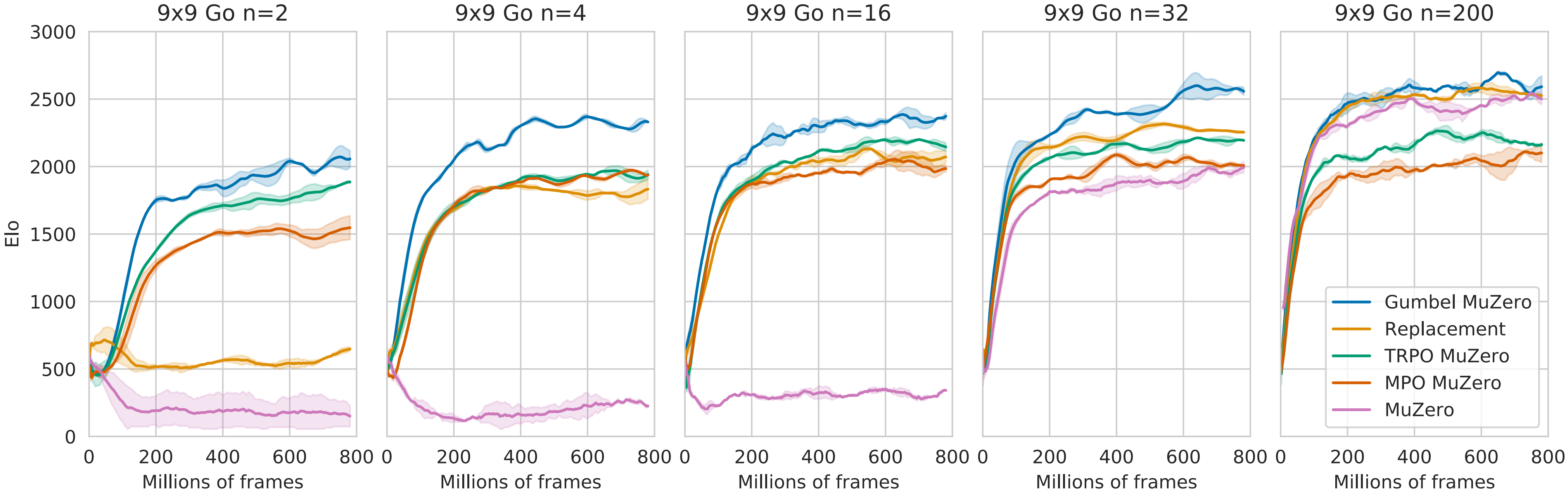


Figure 2: Elo on 9x9 Go, when training with $n \in \{2, 4, 16, 32, 200\}$ simulations. Evaluation uses 800 simulations. Shades denote standard errors from 2 seeds.

Figure 2 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO

“Replacement” is a Gumbel MuZero ablation sampling actions with replacement.
“TRPO MuZero”, “MPO MuZero” use Act+Search+Learn using the previously described regularized policy with $D_{\mathrm{KL}}(\pi \| \pi_{\mathrm{new}})$ and $D_{\mathrm{KL}}(\pi_{\mathrm{new}} \| \pi)$ regularizer, respectively.

Gumbel AlphaZero and MuZero

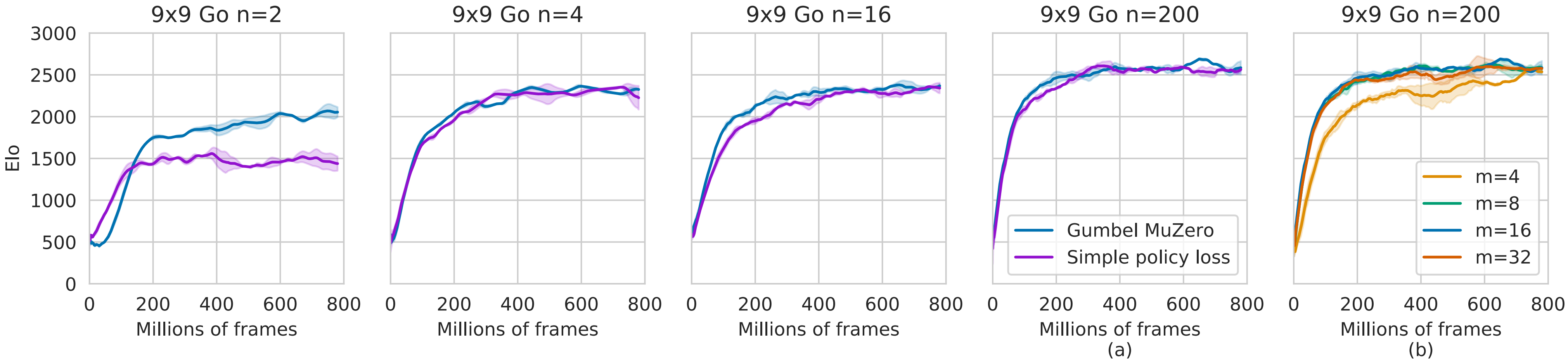


Figure 3: Gumbel MuZero ablations on 9x9 Go. **(a)** Policy loss ablations, when training with $n \in \{2, 4, 16, 200\}$ simulations. Gumbel MuZero uses the policy loss with completed Q-values. **(b)** Sensitivity to the number of sampled actions. Gumbel MuZero samples m actions without replacement.

Figure 3 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO

Gumbel AlphaZero and MuZero

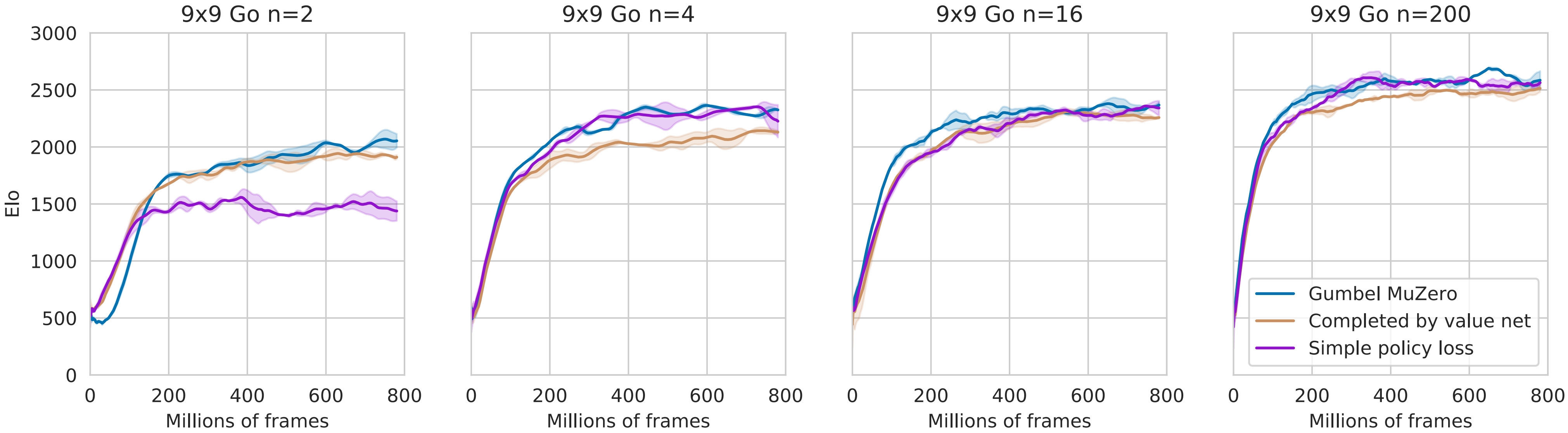


Figure 6: Detailed policy loss ablations. Gumbel MuZero uses the policy loss with Q-values completed by the v_{mix} value estimator from Appendix D. That works better than Q-values completed by the raw value network $\hat{v}_\pi$.

Figure 6 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO

Gumbel AlphaZero and MuZero

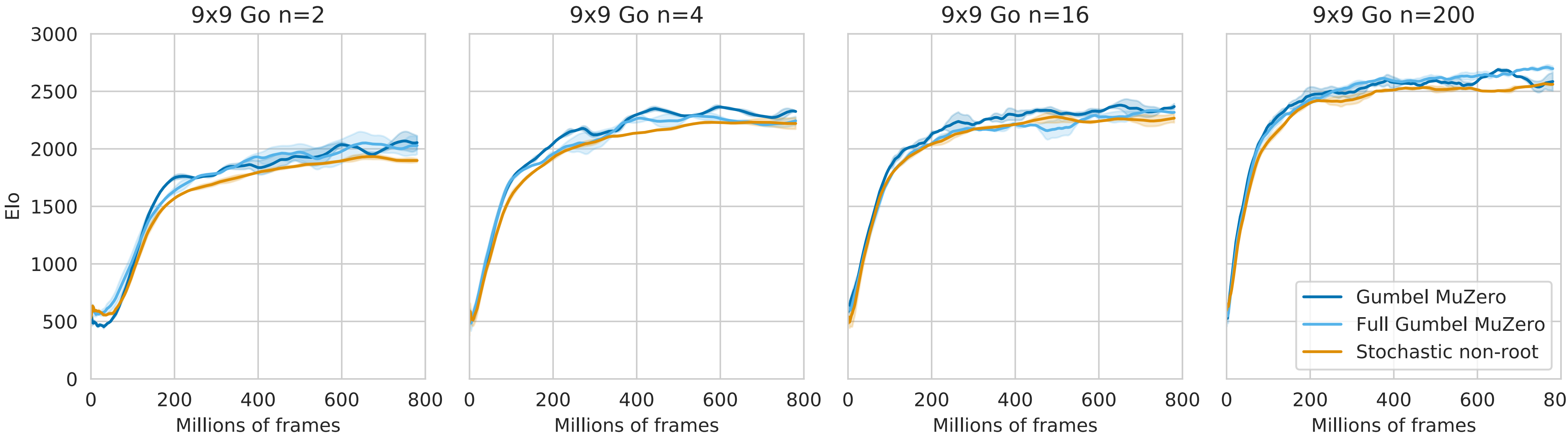


Figure 7: A comparison of different action selections at the non-root nodes. Gumbel MuZero uses the unmodified (deterministic) MuZero action selection at non-root nodes. Full Gumbel MuZero uses the deterministic action selection from Equation 14, which we compare to stochastic sampling from π' at non-root nodes.

Figure 7 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO

Gumbel AlphaZero and MuZero

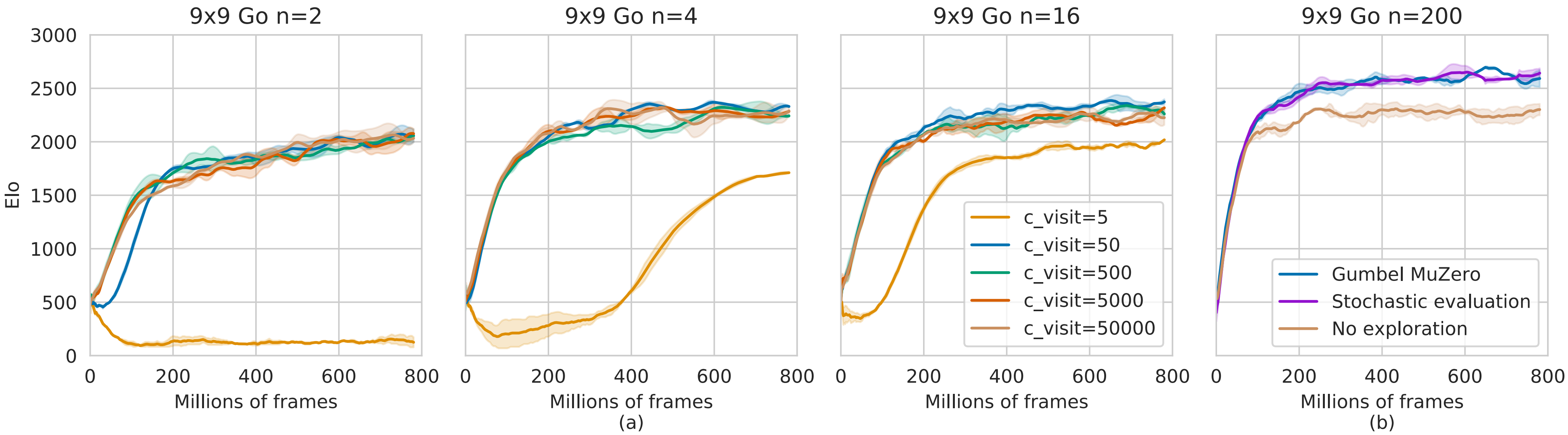


Figure 8: Additional Gumbel MuZero ablations on 9x9 Go. **(a)** Sensitivity to Q-value scaling by c_{visit}. **(b)** On the perfect-information game, Gumbel MuZero used zero Gumbel noise at evaluation. Although, evaluation with stochastic Gumbel noise is not worse. During training, MuZero and Gumbel MuZero benefit from explorative acting proportional to the visit counts.

Figure 8 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO

Gumbel AlphaZero and MuZero

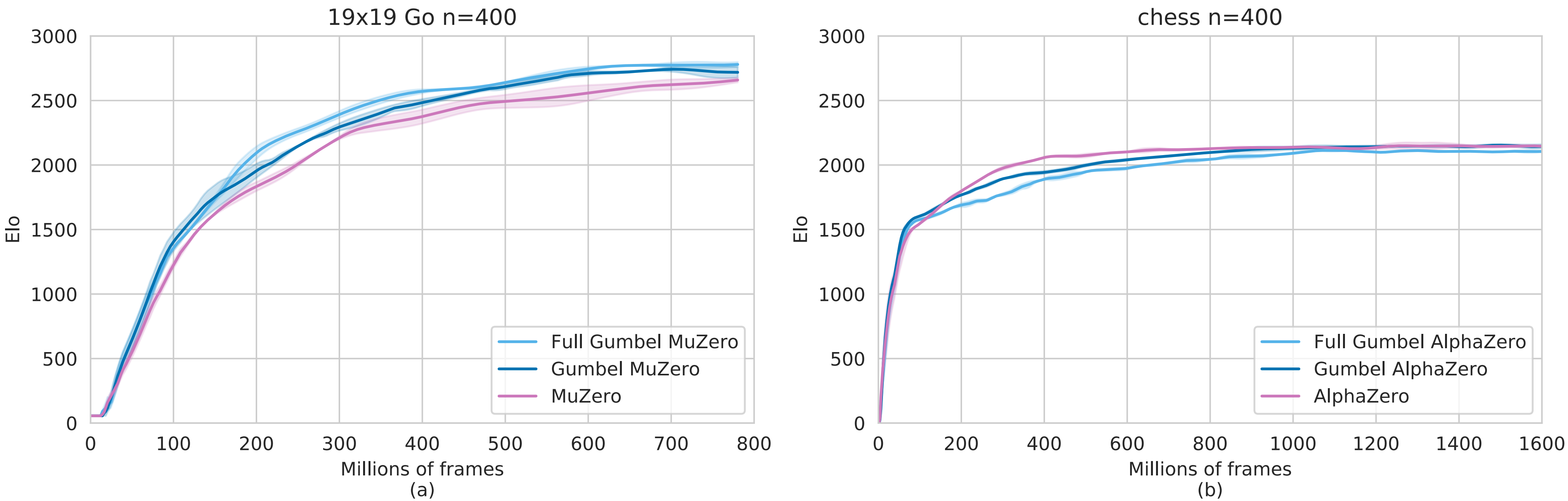


Figure 4: Large-scale experiments with $n = 400$ simulations per move. **(a)** Elo on 19x19 Go, when training MuZero. **(b)** Elo on chess, when training AlphaZero.

Figure 4 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO

Gumbel AlphaZero and MuZero

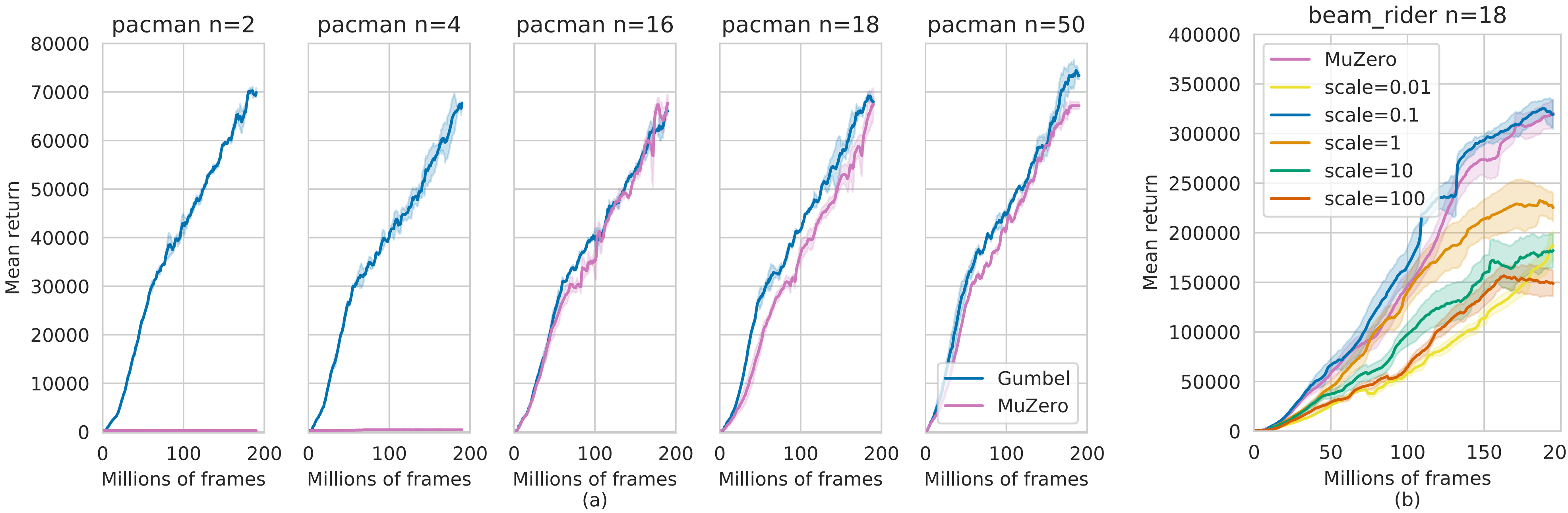


Figure 5: Atari results. **(a)** Mean return on `ms_pacman`, when training Gumbel MuZero and MuZero with $n \in \{2, 4, 16, 18, 50\}$ simulations. MuZero fails to learn from 4 or less simulations. **(b)** Mean return on `beam_rider` for Gumbel MuZero with $c_{\text{scale}} \in \{0.01, 0.1, 1, 10, 100\}$, compared to MuZero with $n = 50$ simulations. Shades denote standard errors from 10 seeds.

Figure 5 of "Policy improvement by planning with Gumbel", https://openreview.net/forum?id=bERaNdoegnO