PopArt Normalization, R2D2, MuZero

Milan Straka

December 14, 2020

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

PopArt Normalization

An improvement of IMPALA from Sep 2018, which performs normalization of task rewards instead of just reward clipping. PopArt stands for *Preserving Outputs Precisely, while Adaptively Rescaling Targets*.

Assume the value estimate $v(s; \boldsymbol\theta, \sigma, \mu)$ is computed using a normalized value predictor $n(s; \boldsymbol\theta)$

$$v(s; \boldsymbol\theta, \sigma, \mu) \stackrel{\text{def}}{=} \sigma n(s; \boldsymbol\theta) + \mu$$

and further assume that $n(s; \boldsymbol\theta)$ is an output of a linear function

$$n(s; \boldsymbol\theta) \stackrel{\text{def}}{=} \boldsymbol\omega^T f(s; \boldsymbol\theta - \{\boldsymbol\omega, b\}) + b.$$

We can update the σ and μ using exponentially moving average with decay rate β (in the paper, first moment μ and second moment υ is tracked, and the standard deviation is computed as $\sigma = \sqrt{\upsilon - \mu^2}$; decay rate $\beta = 3 \cdot 10^{-4}$ is employed).

PopArt Normalization

Utilizing the parameters μ and σ, we can normalize the observed (unnormalized) returns as $(G - \mu)/\sigma$ and use an actor-critic algorithm with advantage $(G - \mu)/\sigma - n(S; \boldsymbol\theta)$.

However, in order to make sure the value function estimate does not change when the normalization parameters change, the parameters $\boldsymbol\omega, b$ used to compute the value estimate

$$v(s; \boldsymbol\theta, \sigma, \mu) \stackrel{\text{def}}{=} \sigma\Big(\boldsymbol\omega^T f(s; \boldsymbol\theta - \{\boldsymbol\omega, b\}) + b\Big) + \mu$$

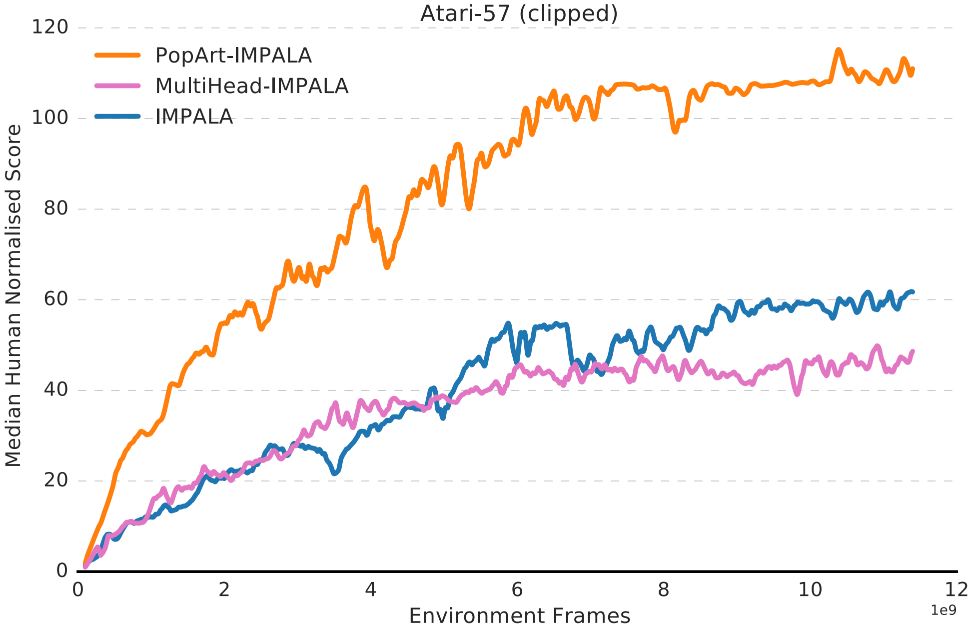
are updated under any change $\mu \rightarrow \mu'$ and $\sigma \rightarrow \sigma'$ as

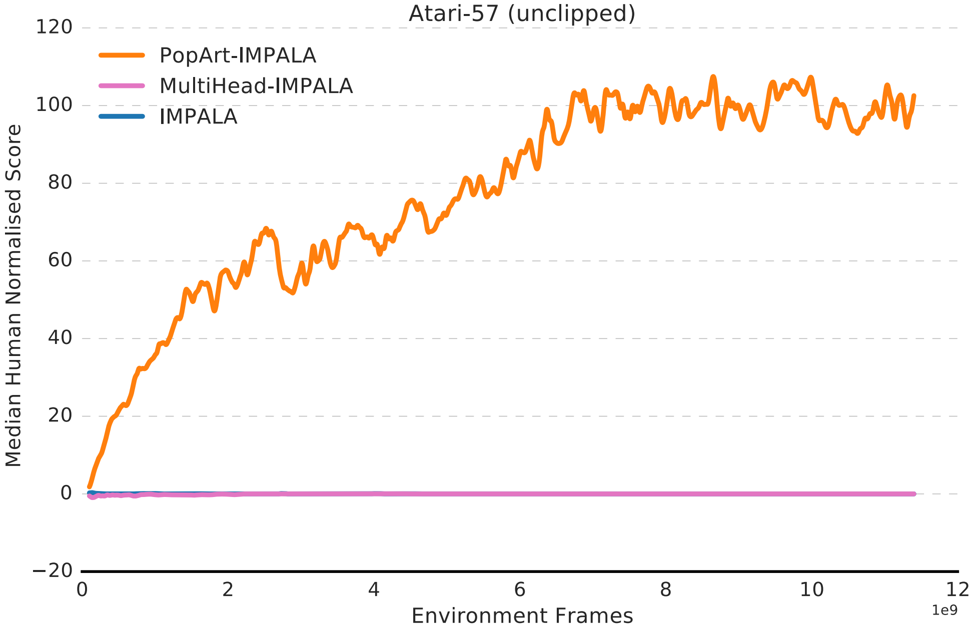
$$\begin{aligned} \boldsymbol\omega' &\leftarrow \frac{\sigma}{\sigma'}\boldsymbol\omega, \\ b' &\leftarrow \frac{\sigma b + \mu - \mu'}{\sigma'}. \end{aligned}$$

In multi-task settings, we train a task-agnostic policy and task-specific value functions (therefore, $\boldsymbol\mu$, $\boldsymbol\sigma$ and $\boldsymbol n(s; \boldsymbol\theta)$ are vectors).

Agent	Atari-57 Random	Atari-57 Human	Atari-57 (unclipped) Random	Atari-57 (unclipped) Human	DmLab-30 Train	DmLab-30 Test
IMPALA	59.7%	28.5%	0.3%	1.0%	60.6%	58.4%
PopArt-IMPALA	110.7%	101.5%	107.0%	93.7%	73.5%	72.8%

Table 1 of paper "Multi-task Deep Reinforcement Learning with PopArt" by Matteo Hessel et al.





Figures 1, 2 of paper "Multi-task Deep Reinforcement Learning with PopArt" by Matteo Hessel et al.

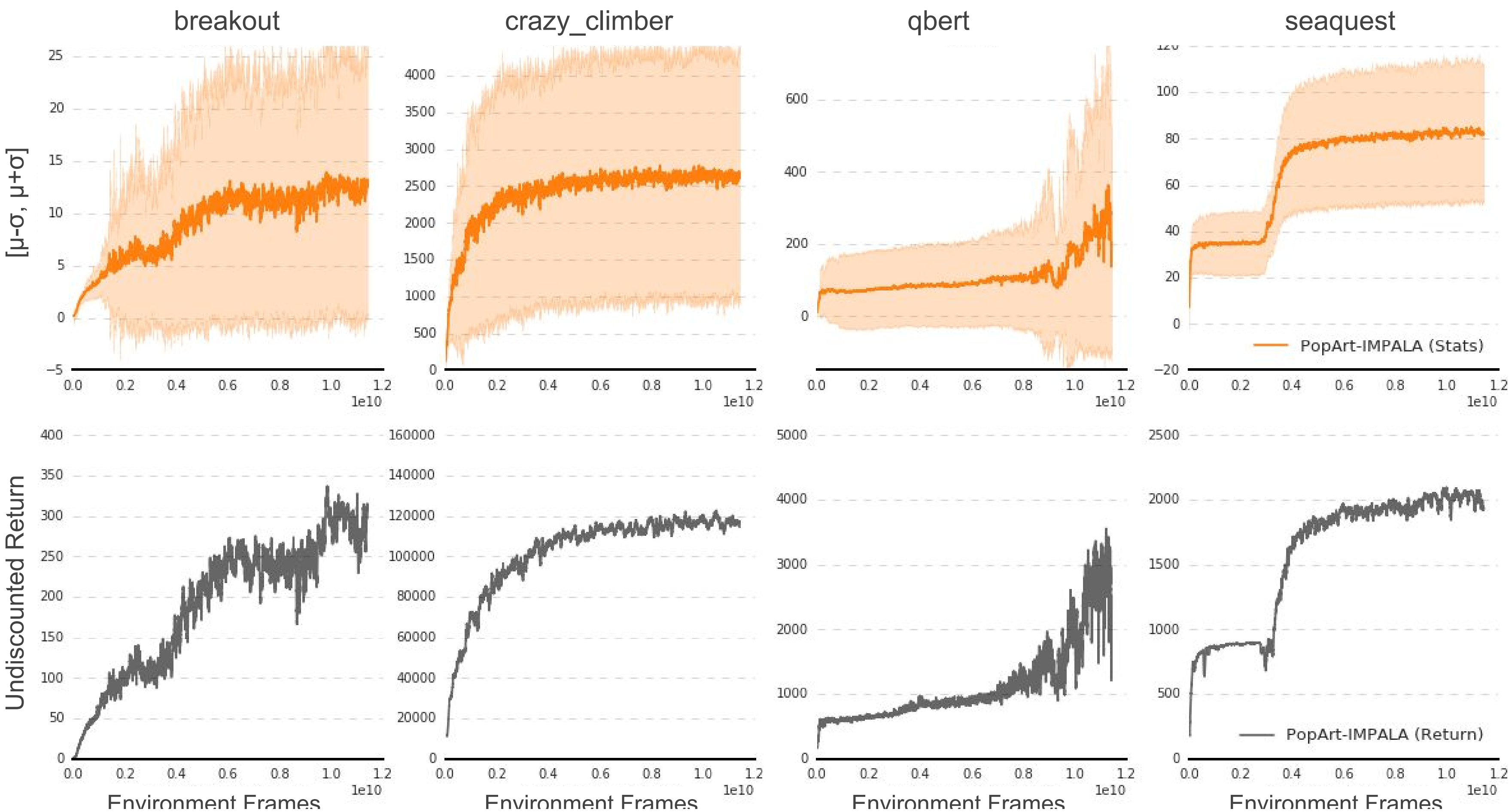
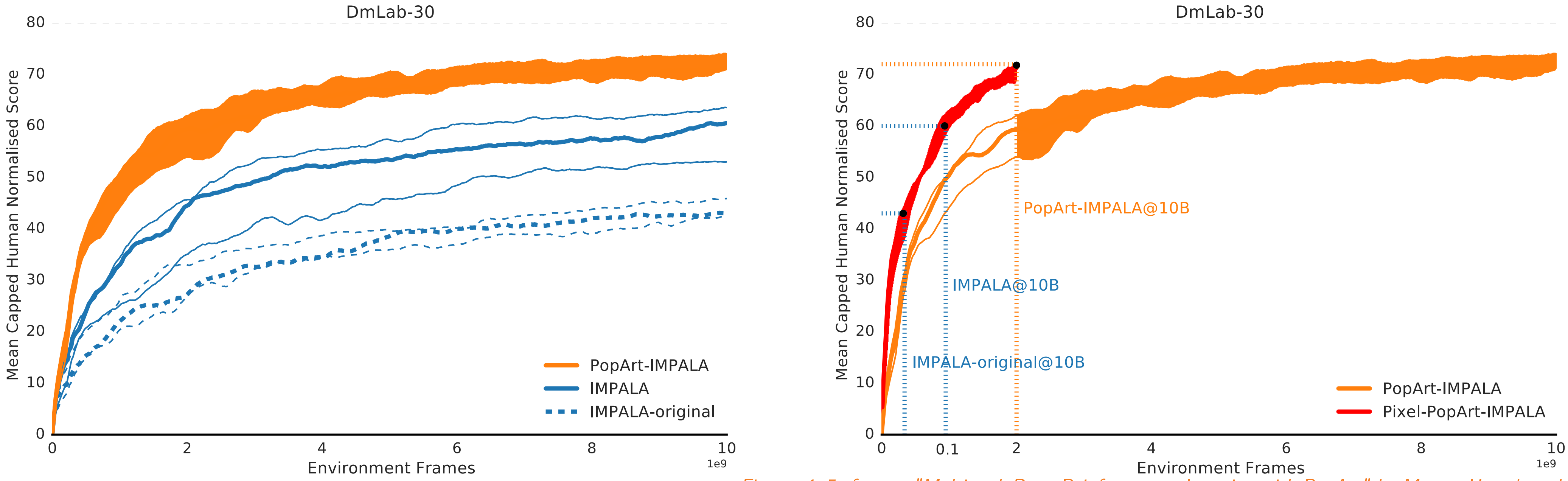


Figure 3 of paper "Multi-task Deep Reinforcement Learning with PopArt" by Matteo Hessel et al.

Normalization statistics on chosen environments.



Figures 4, 5 of paper "Multi-task Deep Reinforcement Learning with PopArt" by Matteo Hessel et al.

Transformed Rewards

So far, we have clipped the rewards in DQN on Atari environments.

Consider a Bellman operator $\mathcal{T}$

$$(\mathcal{T}q)(s,a) \stackrel{\text{def}}{=} \mathbb{E}_{s',r\sim p}\Big[r + \gamma \max_{a'} q(s',a')\Big].$$

Instead of reducing the magnitude of rewards, we might use a function $h: \mathbb{R} \rightarrow \mathbb{R}$ to reduce their scale. We define a transformed Bellman operator $\mathcal{T}_h$ as

$$(\mathcal{T}_h q)(s,a) \stackrel{\text{def}}{=} \mathbb{E}_{s',r\sim p}\Big[h\Big(r + \gamma \max_{a'} h^{-1}\big(q(s',a')\big)\Big)\Big].$$

Transformed Rewards

It is easy to prove the following two propositions from a 2018 paper *Observe and Look Further: Achieving Consistent Performance on Atari* by Tobias Pohlen et al.

1. If $h(z) = \alpha z$ for $\alpha > 0$, then $\mathcal{T}_h^k q \xrightarrow{k\rightarrow\infty} h \circ q_* = \alpha q_*$.

   The statement follows from the fact that it is equivalent to scaling the rewards by a constant α.

2. When h is strictly monotonically increasing and the MDP is deterministic, then $\mathcal{T}_h^k q \xrightarrow{k\rightarrow\infty} h \circ q_*$.

   This second proposition follows from

   $$h \circ q_* = h \circ \mathcal{T} q_* = h \circ \mathcal{T}(h^{-1} \circ h \circ q_*) = \mathcal{T}_h(h \circ q_*),$$

   where the last equality only holds if the MDP is deterministic.

Transformed Rewards

The authors use the following transformation for the Atari environments

$$h(x) \stackrel{\text{def}}{=} \operatorname{sign}(x)\left(\sqrt{|x|+1}-1\right)+\varepsilon x$$

with $\varepsilon = 10^{-2}$. The additive regularization term ensures that h^{-1} is Lipschitz continuous.

It is straightforward to verify that

$$h^{-1}(x) = \operatorname{sign}(x)\left(\left(\frac{\sqrt{1+4\varepsilon(|x|+1+\varepsilon)}-1}{2\varepsilon}\right)^2-1\right).$$

Recurrent Replay Distributed DQN (R2D2)

Proposed in 2019, to study the effects of recurrent state, experience replay and distributed training.

R2D2 utilizes prioritized replay, n-step double Q-learning with $n=5$, convolutional layers followed by a 512-dimensional LSTM passed to duelling architecture, generating experience by a large number of actors (256; each performing approximately 260 steps per second) and learning from batches by a single learner (achieving 5 updates per second using mini-batches of 64 sequences of length 80).

Instead of individual transitions, the replay buffer consists of fixed-length $(m=80)$ sequences of (s, a, r), with adjacent sequences overlapping by 40 time steps.

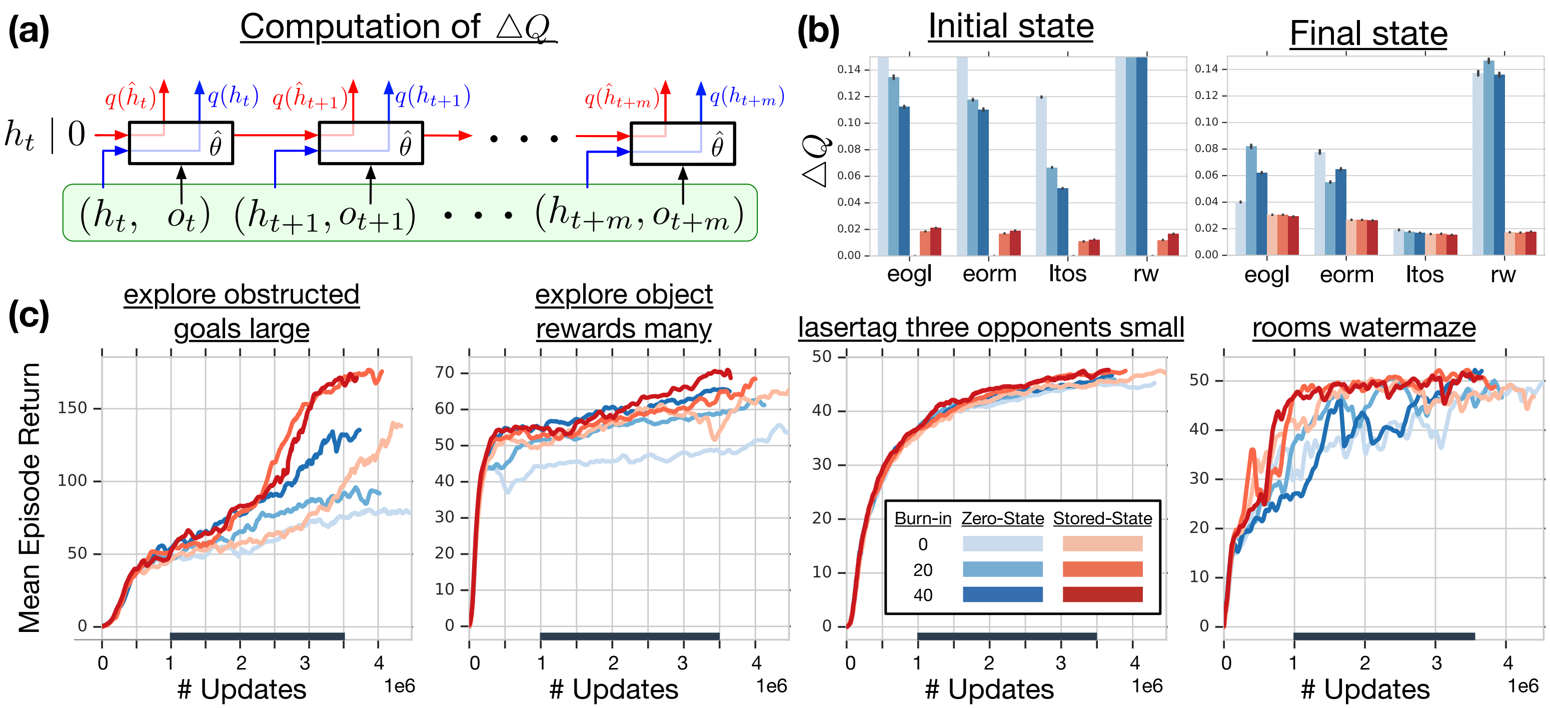


Figure 1: Top row shows Q-value discrepancy ΔQ as a measure for recurrent state staleness. **(a)** Diagram of how ΔQ is computed, with green box indicating a whole sequence sampled from replay. For simplicity, $l = 0$ (no burn-in). **(b)** ΔQ measured at first state and last state of replay sequences, for agents training on a selection of DMLab levels (indicated by initials) with different training strategies. Bars are averages over seeds and through time indicated by bold line on x-axis in bottom row. **(c)** Learning curves on the same levels, varying the training strategy, and averaged over 3 seeds.

Figure 1 of the paper "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Recurrent Replay Distributed DQN (R2D2)

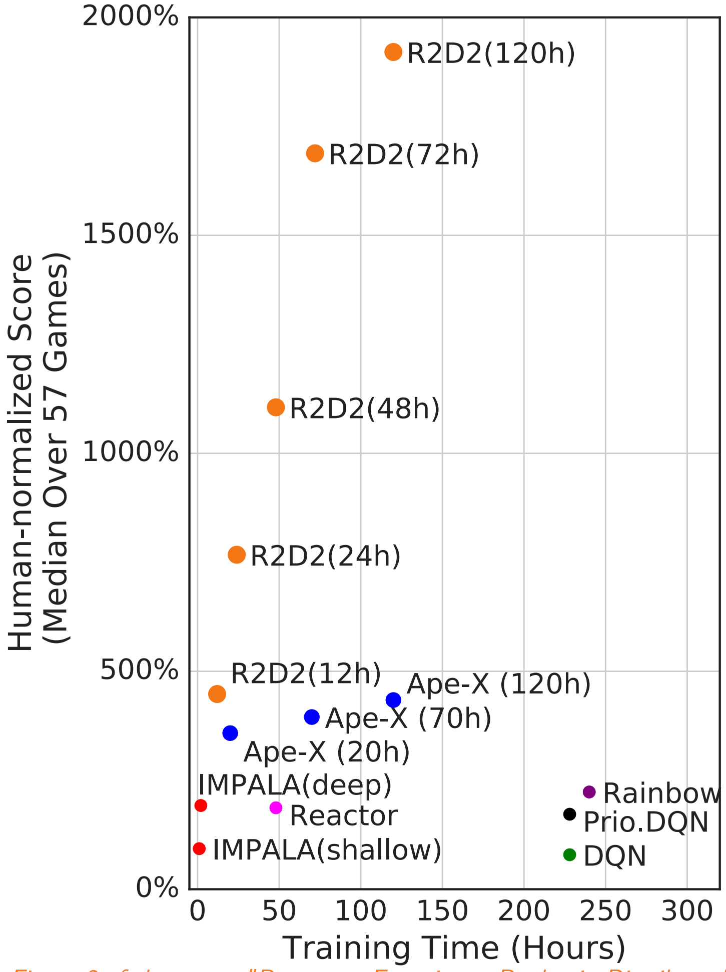


Figure 2 of the paper "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

	Atari-57		DMLab-30	
Human-Normalized Score	**Median**	**Mean**	**Median**	**Mean-Capped**
Ape-X (Horgan et al., 2018)	434.1%	1695.6%	–	–
Reactor (Gruslys et al., 2018)	187.0%	–	–	–
IMPALA, deep (Espeholt et al., 2018)	191.8%	957.6%	49.0%	45.8%
IMPALA, shallow (re-run)	–	–	89.7%	73.6%
IMPALA, deep (re-run)	–	–	**107.5%**	85.1%
R2D2+	–	–	99.5%	**85.7%**
R2D2, feed-forward	589.2%	1974.4%	–	–
R2D2	**1920.6%**	**4024.9%**	96.9%	78.3%

Table 1 of the paper "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Number of actors	256
Actor parameter update interval	400 environment steps
Sequence length m	80 (+ prefix of $l = 40$ in burn-in experiments)
Replay buffer size	4×10^6 observations (10^5 part-overlapping sequences)
Priority exponent	0.9
Importance sampling exponent	0.6
Discount γ	0.997
Minibatch size	64 (32 for R2D2+ on DMLab)
Optimizer	Adam (Kingma & Ba, 2014)
Optimizer settings	learning rate $= 10^{-4}$, $\varepsilon = 10^{-3}$
Target network update interval	2500 updates
Value function rescaling	$h(x) = \operatorname{sign}(x)(\sqrt{\lvert x\rvert + 1} - 1) + \epsilon x,\ \epsilon = 10^{-3}$

Table 2: Hyper-parameters values used in R2D2. All missing parameters follow the ones in Ape-X (Horgan et al., 2018).

Table 2 of the paper "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

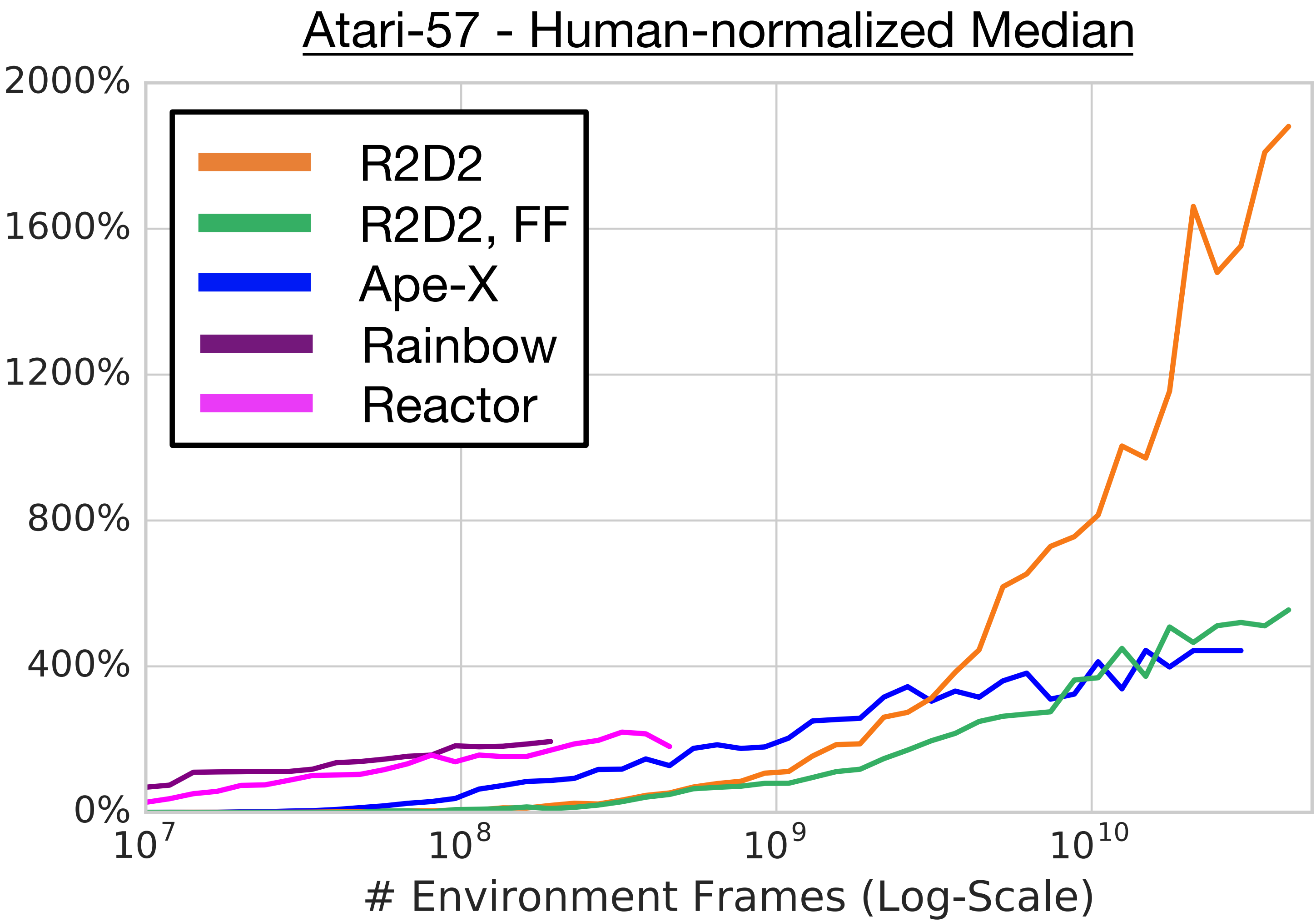


Figure 9 of the paper "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Recurrent Replay Distributed DQN (R2D2)

Ablations comparing the reward clipping instead of value rescaling (**Clipped**), smaller discount factor $\gamma = 0.99$ (**Discount**) and **Feed-Forward** variant of R2D2. Furthermore, life-loss **reset** evaluates resetting an episode on life loss, with **roll** preventing value bootstrapping (but not LSTM unrolling).

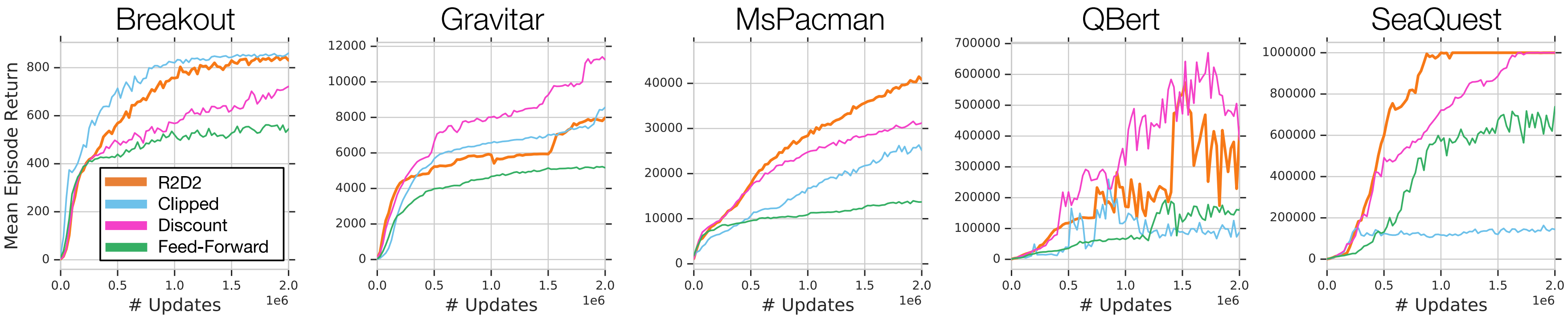


Figure 4 of the paper "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

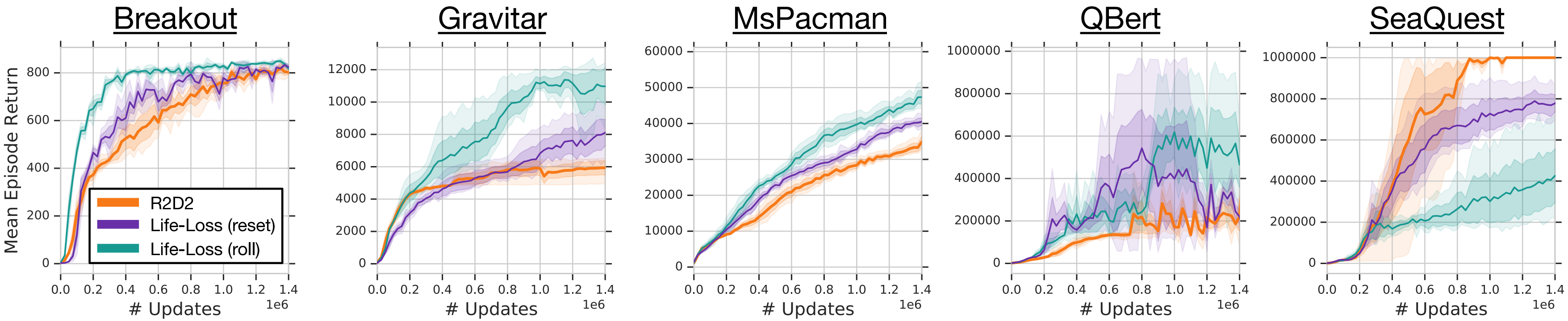


Figure 7 of the paper "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

Utilization of LSTM Memory During Inference

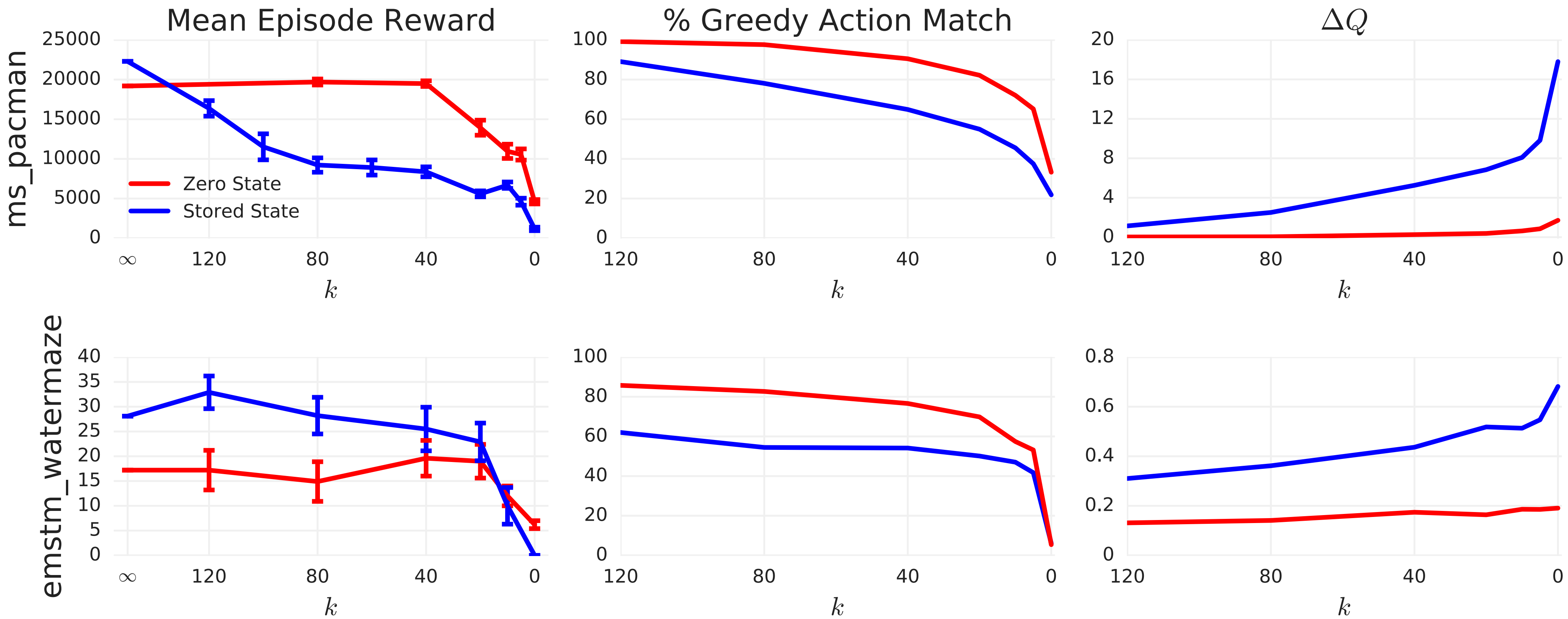


Figure 5 of the paper "Recurrent Experience Replay in Distributed Reinforcement Learning" by Steven Kapturowski et al.

The MuZero algorithm extends the AlphaZero by a **trained model**, alleviating the requirement for a known MDP dynamics.

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 $\boldsymbol{p}_t^k \approx \pi(a_{t+k+1} | o_1, \ldots, o_t, a_{t+1}, \ldots, a_{t+k})$,
- 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.

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, $\boldsymbol{r}^k, \boldsymbol{s}^k = g_\theta(\boldsymbol{s}^{k-1}, \boldsymbol{a}^k)$, simulates the MDP dynamics and predicts an immediate reward $\boldsymbol{r}^k$ 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 $\boldsymbol{p}^k, v^k = f_\theta(\boldsymbol{s}^k)$, computes the policy and value function, similarly as in AlphaZero.
- The representation function, $\boldsymbol{s}^0 = h_\theta(o_1, \ldots, o_t)$, generates an internal state encoding the past observations.

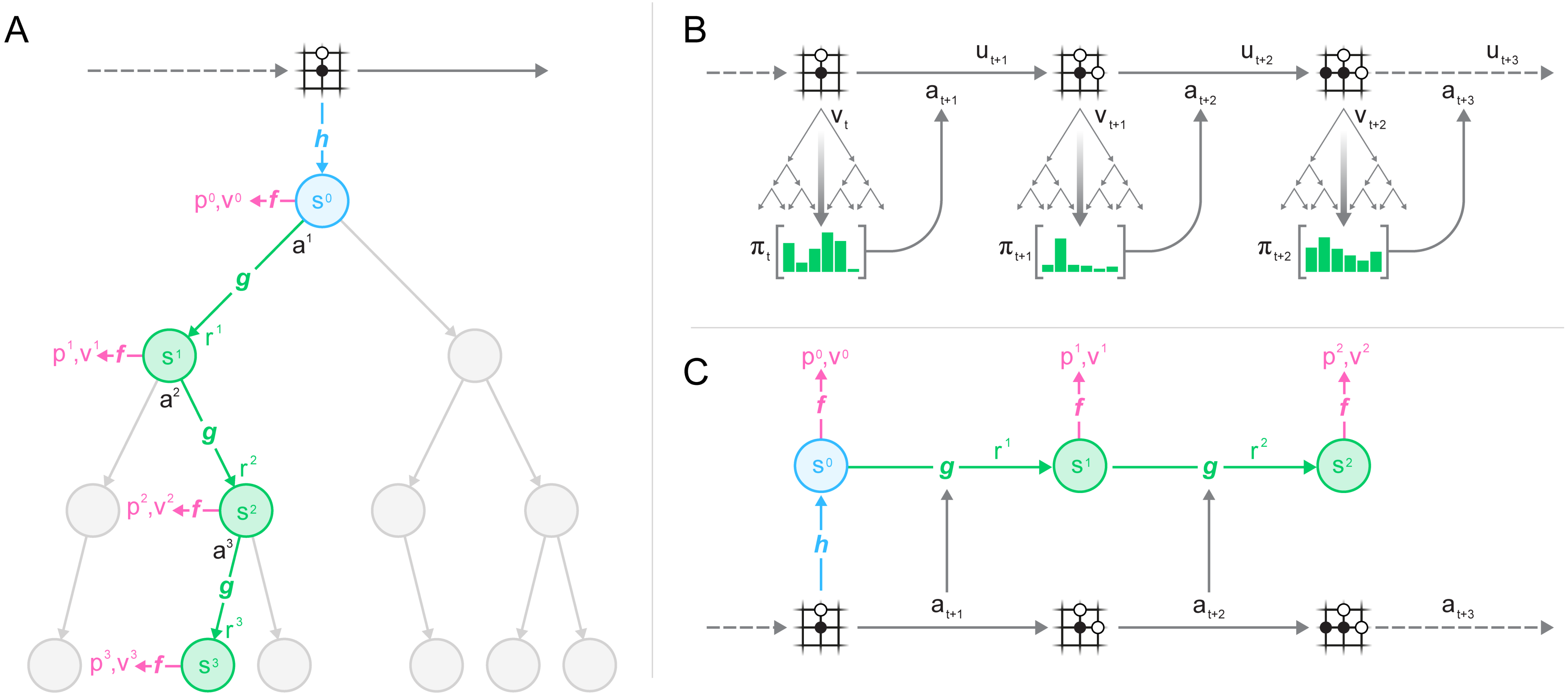


Figure 1 of the paper "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

To select a move, we employ a MCTS algorithm similar to the AlphaZero. It produces a policy π_t and value estimate ν_t, and an action is then sampled from the policy $a_{t+1} \sim \pi_t$.

During training, we utilize a sequence of k moves. We estimate the return using bootstrapping $z_t = u_{t+1} + \gamma u_{t+2} + \ldots + \gamma^{n-1} u_{t+n} + \gamma^n \nu_{t+n}$. The values $k = 5$ and $n = 10$ are used in the paper.

The loss is then composed of the following components:

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

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.

Model

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

Search

$$\nu_t, \pi_t = MCTS(s_t^0, \mu_\theta)$$

$$a_t \sim \pi_t$$

Learning Rule

$$\mathbf{p}_t^k, v_t^k, r_t^k = \mu_\theta(o_1, \ldots, o_t, a_{t+1}, ..., a_{t+k})$$

$$z_t = \begin{cases} u_T & \text{for games} \\ u_{t+1} + \gamma u_{t+2} + ... + \gamma^{n-1} u_{t+n} + \gamma^n \nu_{t+n} & \text{for general MDPs} \end{cases}$$

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

Losses

$$\mathcal{L}^r(u, r) = \begin{cases} 0 & \text{for games} \\ \boldsymbol{\phi}(u)^T \log \mathbf{r} & \text{for general MDPs} \end{cases}$$

$$\mathcal{L}^v(z, q) = \begin{cases} (z-q)^2 & \text{for games} \\ \boldsymbol{\phi}(z)^T \log \mathbf{q} & \text{for general MDPs} \end{cases}$$

$$\mathcal{L}^p(\pi, p) = \boldsymbol{\pi}^T \log \mathbf{p}$$

MuZero – Evaluation

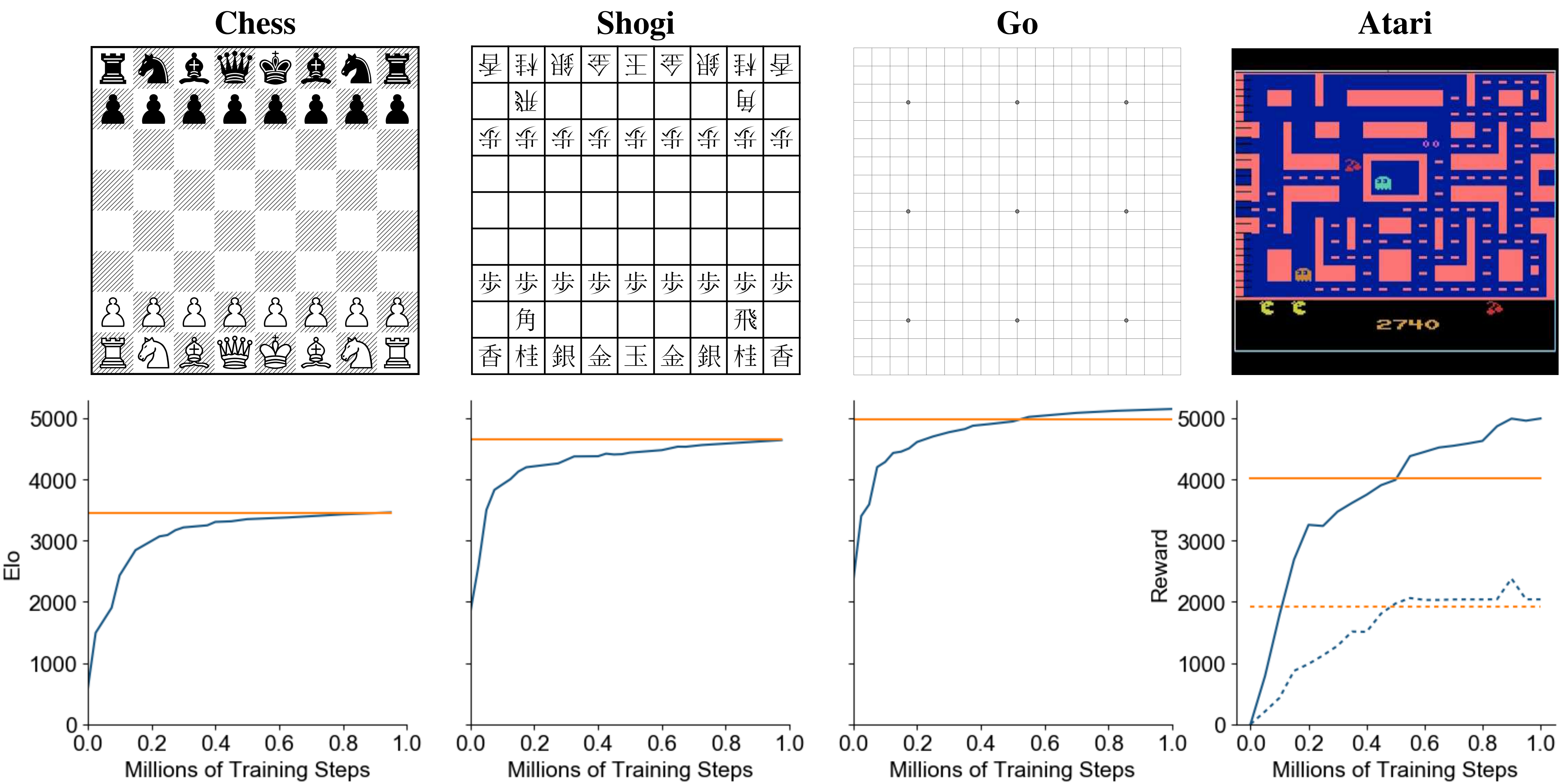


Figure 2 of the paper "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

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 the paper "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

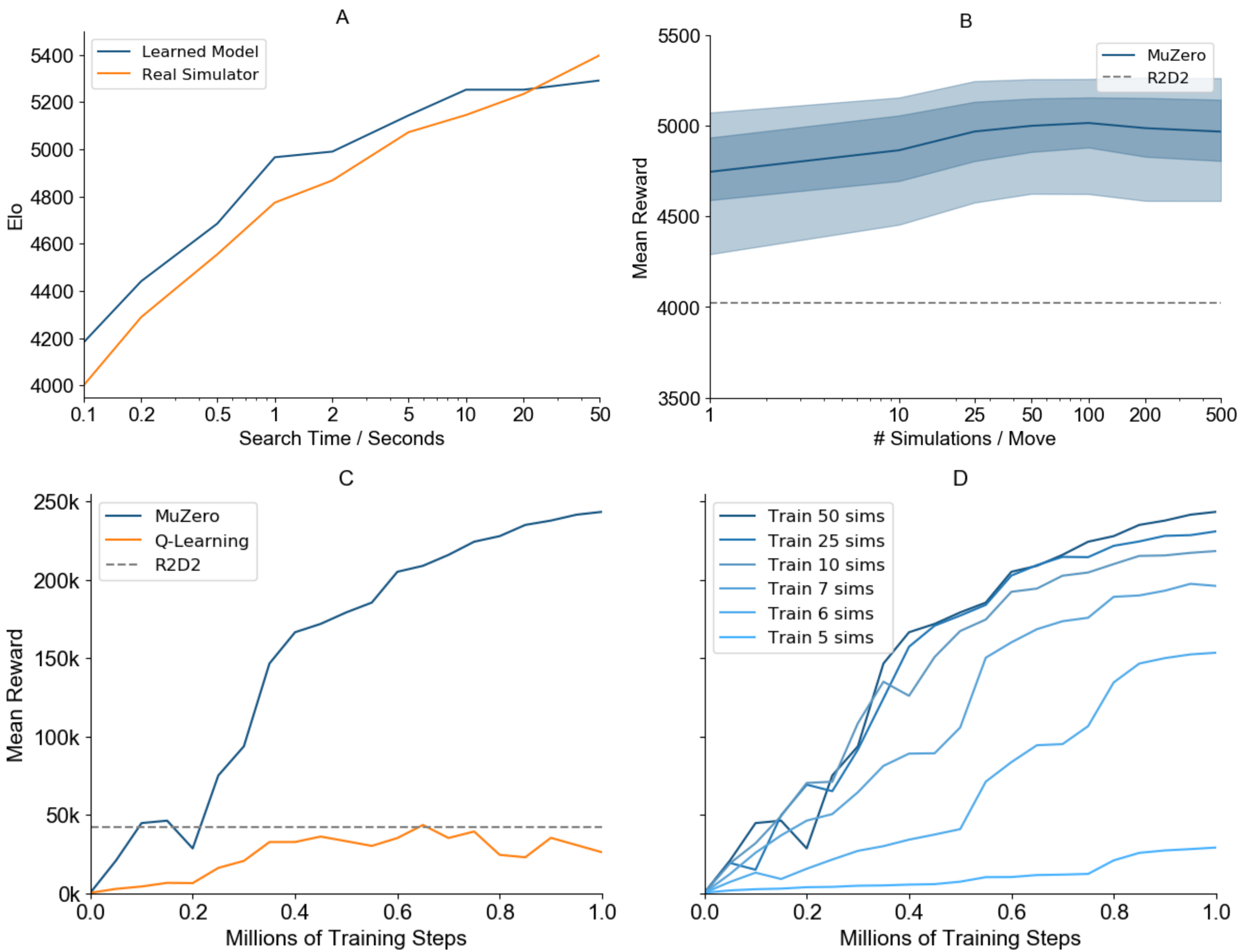


Figure 3 of the paper "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

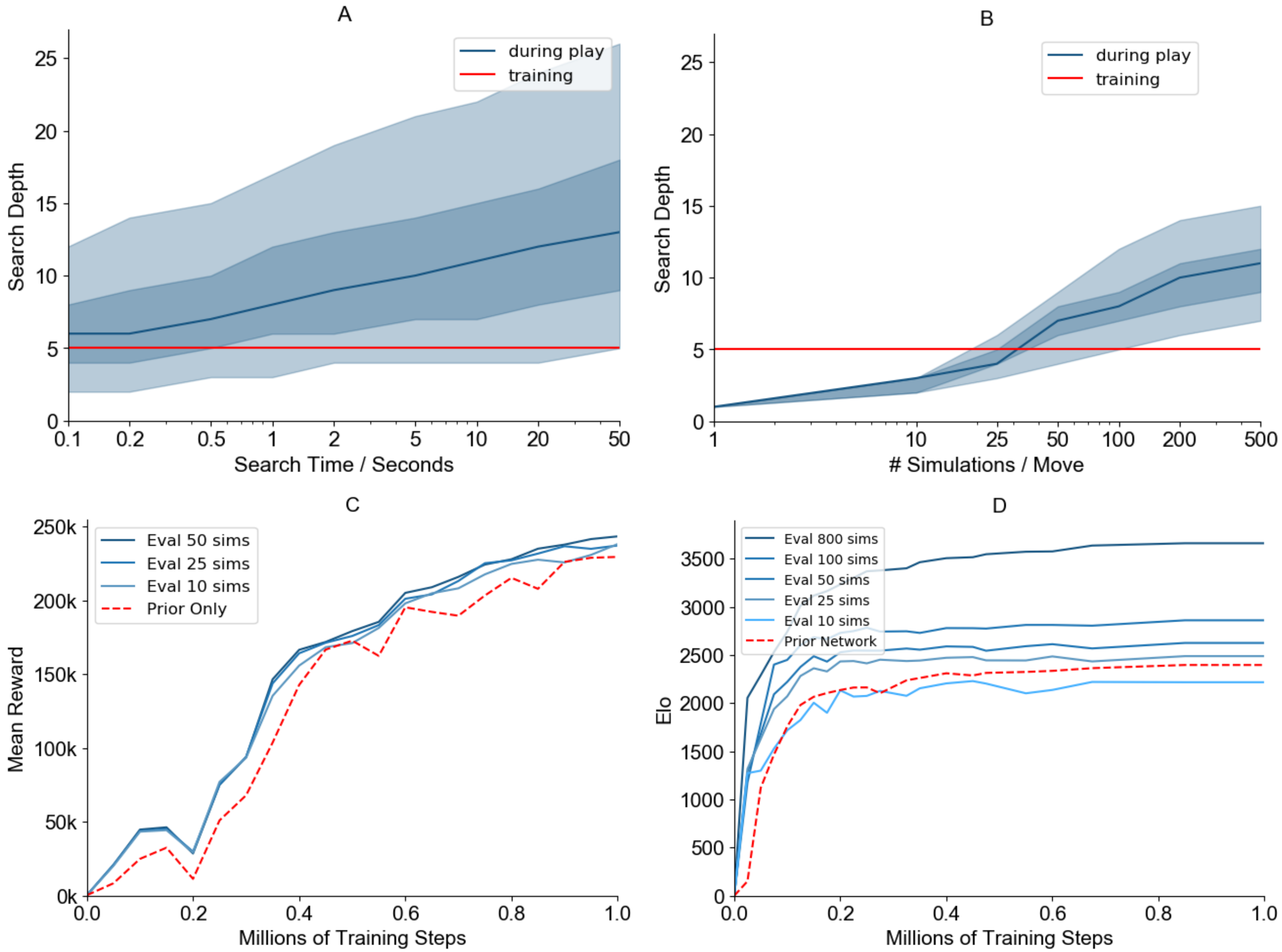


Figure S3 of the paper "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

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 the paper "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 the paper "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.