MuZero, PlaNet

Milan Straka

December 13, 2021

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

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

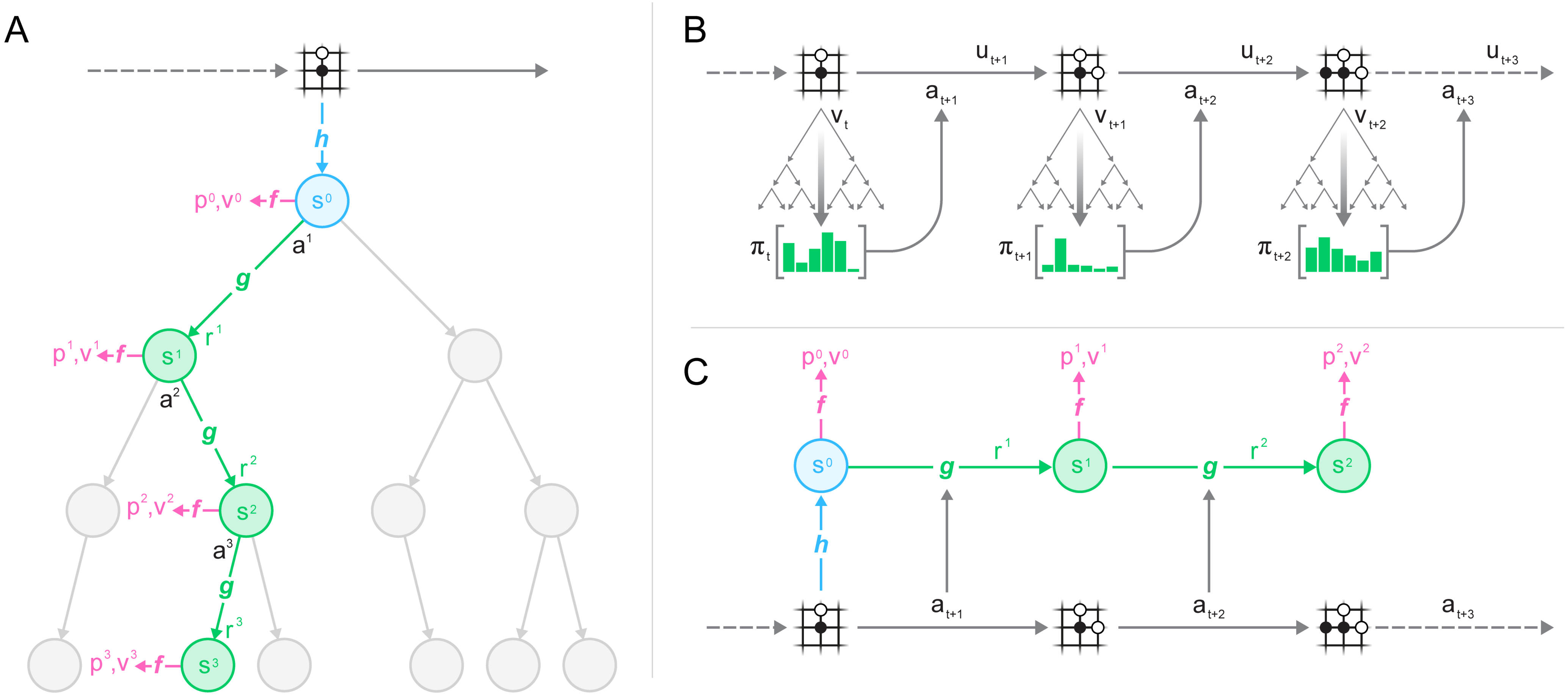


Figure 1 of the paper "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

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 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 consider 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 minimum and maximum 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 the obtained policy, $a_{t+1} \sim \boldsymbol\pi_t$.

For games, the same strategy of sampling the actions a_t is used. In the Atari domain, the actions are sampled according to 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.

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.

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.

Furthermore in Atari, the discount factor $\gamma = 0.997$ is used and the replay buffer elements are sampled according to prioritized replay and importance sampling is used to account for changing the sampling distribution.

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

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

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

Learning Rule

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

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

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

Losses

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

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

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

MuZero – Evaluation

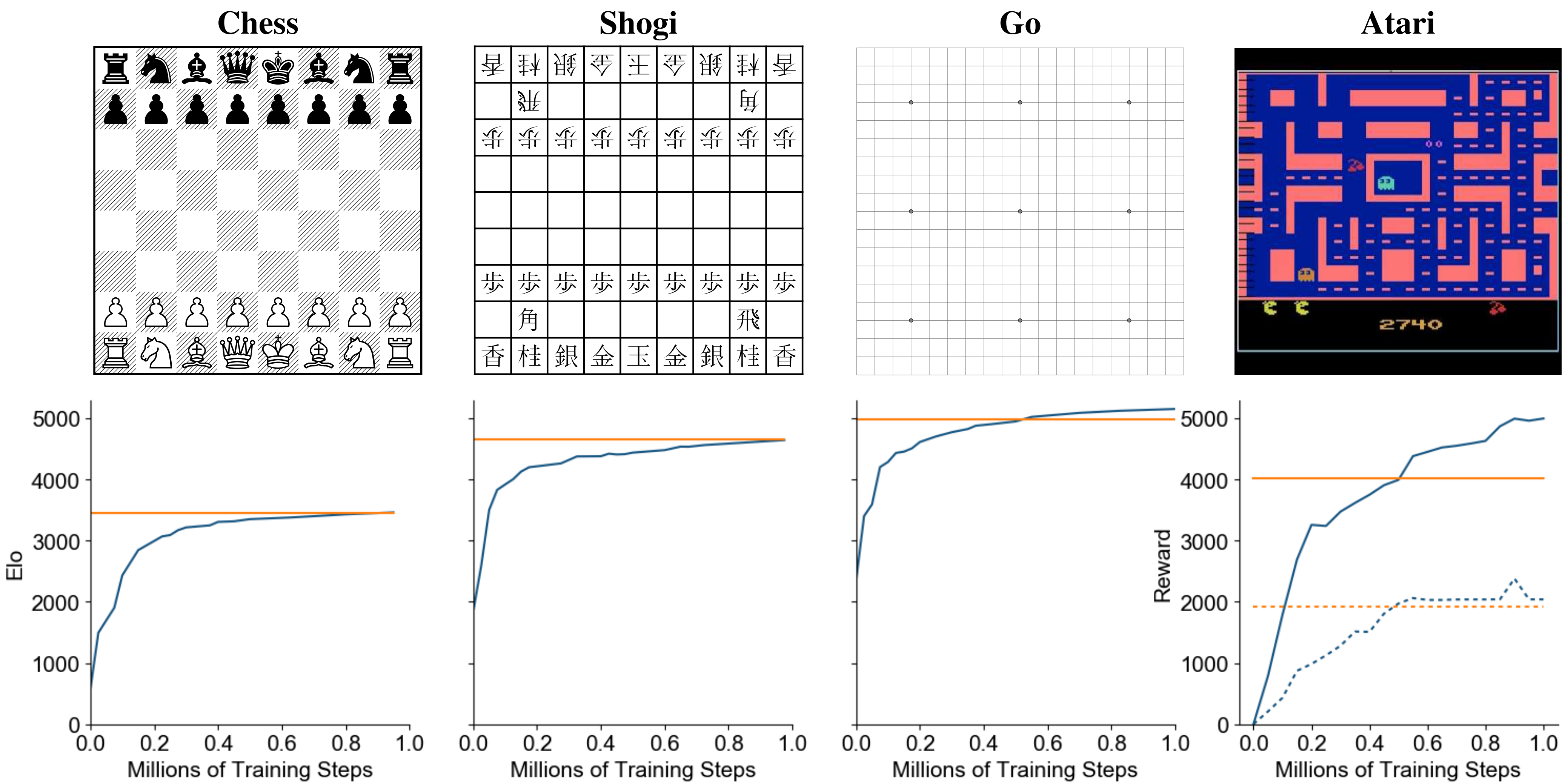


Figure 2 of the paper "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

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.

MuZero Reanalyze is optimized for greater sample efficiency. It revisits past trajectories using the network with the latest parameters (using the fresh policy in 80% of the training steps).

MuZero – Planning Ablations

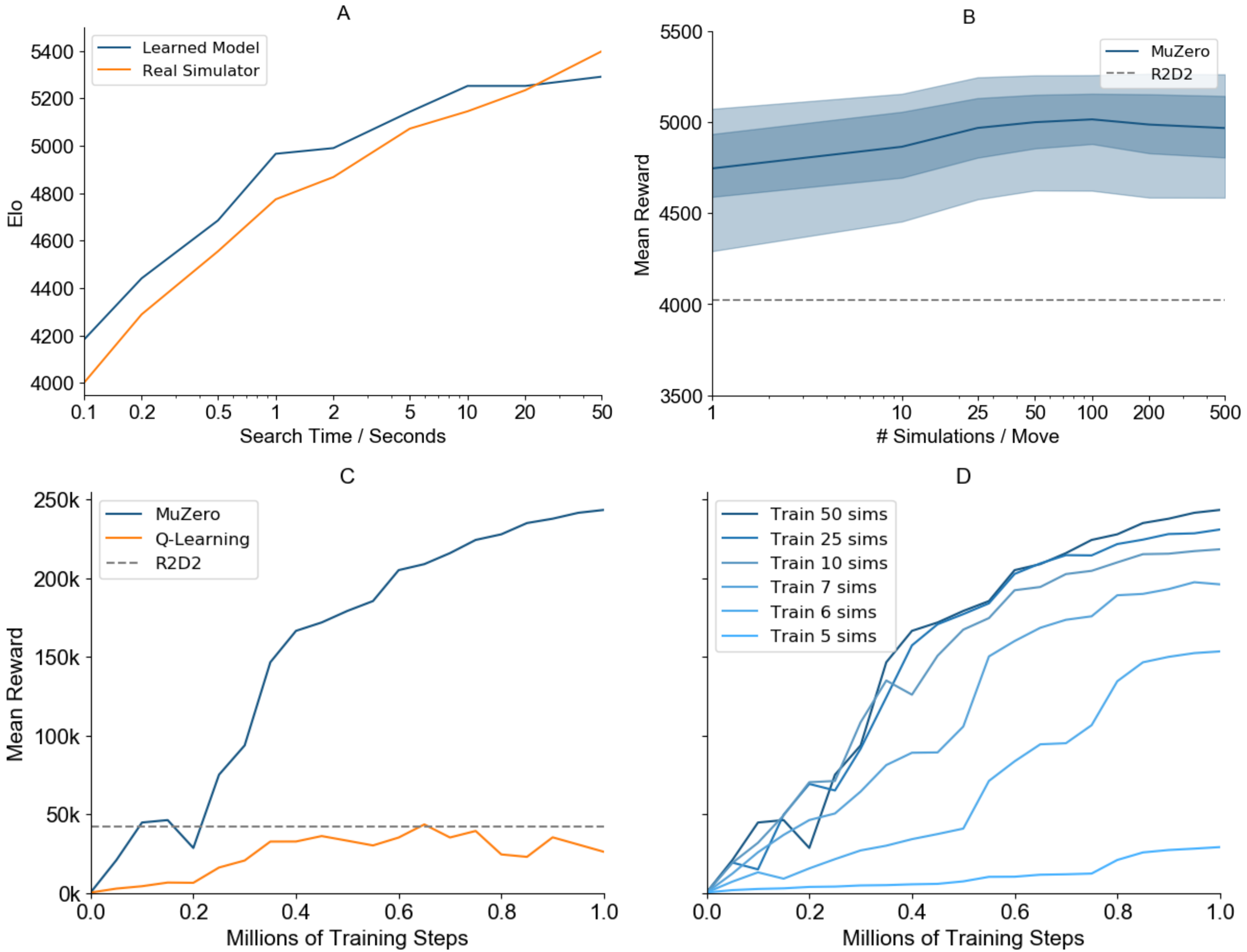


Figure 3 of the paper "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

MuZero – Planning Ablations

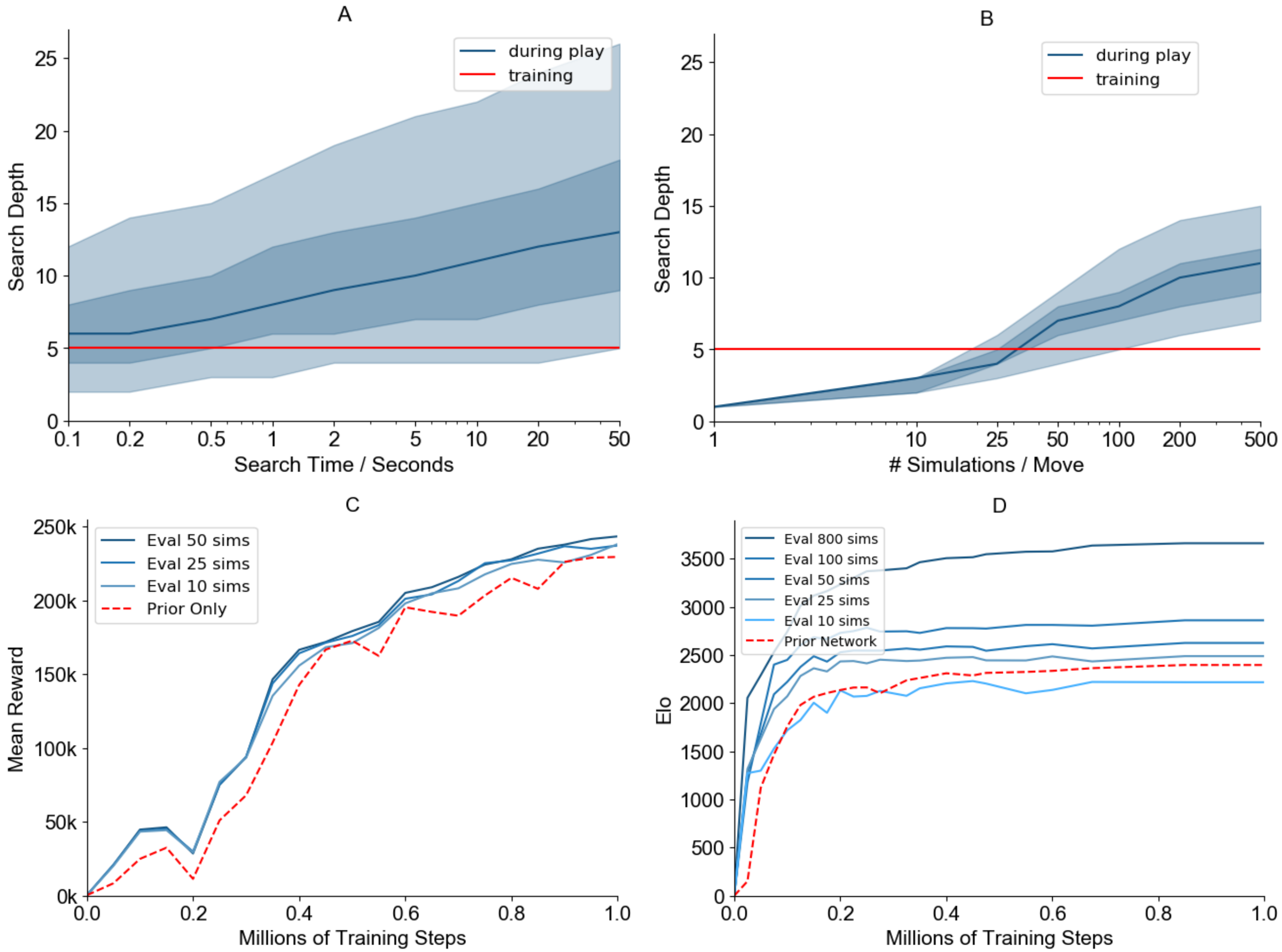


Figure S3 of the paper "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model" by Julian Schrittwieser et al.

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

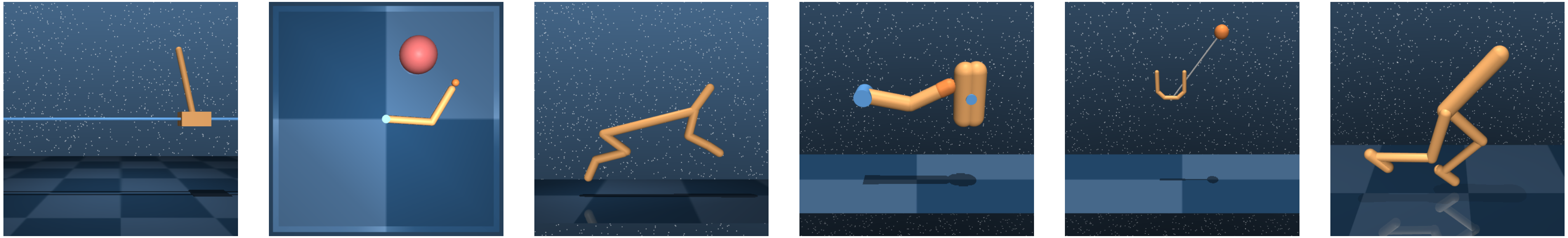
Game	Random	Human	SimPLe [20]	Ape-X [18]	R2D2 [21]	*MuZero*	*MuZero* normalized
krull	1,598.05	2,665.53	2,204.80	11,741.00	218,448.10	**269,358.27**	25,083.4 %
kung fu master	258.50	22,736.25	14,862.50	97,830.00	**233,413.30**	204,824.00	910.1 %
montezuma revenge	0.00	**4,753.33**	-	2,500.00	2,061.30	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.

PlaNet

In Nov 2018, an interesting paper from D. Hafner et al. proposed a **Deep Planning Network (PlaNet)**, which is a model-based agent that learns the MDP dynamics from pixels and then chooses actions using a CEM planner using a compact latent space.

The PlaNet is evaluated on selected tasks from the DeepMind control suite



(a) Cartpole (b) Reacher (c) Cheetah (d) Finger (e) Cup (f) Walker

Figure 1 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

The partially observable MDPs are considered in PlaNet, that follow the stochastic dynamics:

$$\begin{aligned}
\text{transition function:} \quad & s_t \sim p(s_t|s_{t-1}, a_{t-1}), \\
\text{observation function:} \quad & o_t \sim p(o_t|s_t), \\
\text{reward function:} \quad & r_t \sim p(r_t|s_t), \\
\text{policy:} \quad & a_t \sim p(a_t|o_{\le t}, a_{<t}).
\end{aligned}$$

The main goal is to train the first three – the transition function, the observation function and the reward function.

PlaNet – Data Collection

Algorithm 1: Deep Planning Network (PlaNet)

Input :

R Action repeat	$p(s_t \mid s_{t-1}, a_{t-1})$	Transition model
S Seed episodes	$p(o_t \mid s_t)$	Observation model
C Collect interval	$p(r_t \mid s_t)$	Reward model
B Batch size	$q(s_t \mid o_{\leq t}, a_{<t})$	Encoder
L Chunk length	$p(\epsilon)$	Exploration noise
α Learning rate		

Initialize dataset $\mathcal{D}$ with S random seed episodes.
Initialize model parameters θ randomly.
while *not converged* **do**
  `// Model fitting`
  for *update step* $s = 1..C$ **do**
    Draw sequence chunks $\{(o_t, a_t, r_t)_{t=k}^{L+k}\}_{i=1}^{B} \sim \mathcal{D}$ uniformly at random from the dataset.
    Compute loss $\mathcal{L}(\theta)$ from Equation 3.
    Update model parameters $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$.
  `// Data collection`
  $o_1 \leftarrow$ `env.reset()`
  for *time step* $t = 1..\lceil \frac{T}{R} \rceil$ **do**
    Infer belief over current state $q(s_t \mid o_{\leq t}, a_{<t})$ from the history.
    $a_t \leftarrow$ `planner`$(q(s_t \mid o_{\leq t}, a_{<t}), p)$, see Algorithm 2 in the appendix for details.
    Add exploration noise $\epsilon \sim p(\epsilon)$ to the action.
    for *action repeat* $k = 1..R$ **do**
      $r_t^k, o_{t+1}^k \leftarrow$ `env.step`(a_t)
    $r_t, o_{t+1} \leftarrow \sum_{k=1}^{R} r_t^k, o_{t+1}^R$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^{T}\}$

Algorithm 1 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

Because an untrained agent will most likely not cover all needed environment states, we need to iteratively collect new experience and train the model. The authors propose $S = 5$, $C = 100$, $B = 50$, $L = 50$, R between 2 and 8.

For planning, CEM algorithm capable of solving all tasks with a true model is used; $H = 12$, $I = 10$, $J = 1000$, $K = 100$.

Algorithm 2: Latent planning with CEM

Input :

H Planning horizon distance	$q(s_t \mid o_{\leq t}, a_{<t})$	Current state belief
I Optimization iterations	$p(s_t \mid s_{t-1}, a_{t-1})$	Transition model
J Candidates per iteration	$p(r_t \mid s_t)$	Reward model
K Number of top candidates to fit		

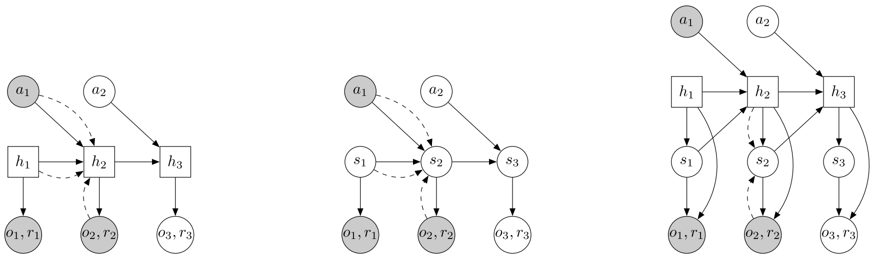
Initialize factorized belief over action sequences $q(a_{t:t+H}) \leftarrow \mathrm{Normal}(0, \mathbb{I})$.
for *optimization iteration* $i = 1..I$ **do**
  `// Evaluate J action sequences from the current belief.`
  for *candidate action sequence* $j = 1..J$ **do**
    $a_{t:t+H}^{(j)} \sim q(a_{t:t+H})$
    $s_{t:t+H+1}^{(j)} \sim q(s_t \mid o_{1:t}, a_{1:t-1}) \prod_{\tau=t+1}^{t+H+1} p(s_\tau \mid s_{\tau-1}, a_{\tau-1}^{(j)})$
    $R^{(j)} = \sum_{\tau=t+1}^{t+H+1} \mathrm{E}[p(r_\tau \mid s_\tau^{(j)})]$
  `// Re-fit belief to the K best action sequences.`
  $\mathcal{K} \leftarrow \mathrm{argsort}(\{R^{(j)}\}_{j=1}^{J})_{1:K}$
  $\mu_{t:t+H} = \frac{1}{K} \sum_{k \in \mathcal{K}} a_{t:t+H}^{(k)}, \quad \sigma_{t:t+H} = \frac{1}{K-1} \sum_{k \in \mathcal{K}} |a_{t:t+H}^{(k)} - \mu_{t:t+H}|.$
  $q(a_{t:t+H}) \leftarrow \mathrm{Normal}(\mu_{t:t+H}, \sigma_{t:t+H}^2 \mathbb{I})$
return *first action mean* μ_t.

Algorithm 2 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

PlaNet – Latent Dynamics

First let us consider a typical latent-space model, consisting of

$$\begin{aligned} \text{transition function:} \quad & s_t \sim p(s_t | s_{t-1}, a_{t-1}), \\ \text{observation function:} \quad & o_t \sim p(o_t | s_t), \\ \text{reward function:} \quad & r_t \sim p(r_t | s_t). \end{aligned}$$



(a) Deterministic model (RNN) (b) Stochastic model (SSM) (c) Recurrent state-space model (RSSM)

Figure 2 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

The transition model is Gaussian with mean and variance predicted by a network, the observation model is Gaussian with identity covariance and mean predicted by a deconvolutional network, and the reward model is a scalar Gaussian with unit variance and mean predicted by a neural network.

To train such a model, we turn to variational inference and use an encoder $q(s_{1:T} | o_{1:T}, a_{1:T}) = \prod_{t=1}^T q(s_t | s_{t-1}, a_{t-1}, o_t)$, which is a Gaussian with mean and variance predicted by a convolutional neural network.

PlaNet – Training Objective

Using the encoder, we obtain the following variational lower bound on the log-likelihood of the observations (for rewards the bound is analogous):

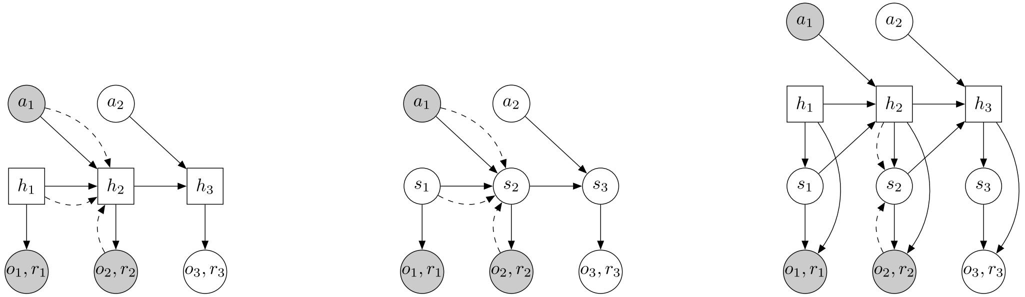
$$\begin{aligned}
&\log p(o_{1:T} | a_{1:T}) \\
&\quad = \log \int \prod_t p(s_t | s_{t-1}, a_{t-1}) p(o_t | s_t) \, \mathrm{d}s_{1:T} \\
&\quad \geq \sum_{t=1}^T \Bigg(\underbrace{\mathbb{E}_{q(s_t | o_{\leq t}, a_{<t})} \log p(o_t | s_t)}_{\text{reconstruction}} - \underbrace{\mathbb{E}_{q(s_{t-1} | o_{\leq t-1}, a_{<t-1})} D_\mathrm{KL}\big(q(s_t | o_{\leq t}, a_{<t}) \| p(s_t | s_{t-1}, a_{t-1})\big)}_{\text{complexity}} \Bigg).
\end{aligned}$$

We evaluate the expectations using a single sample and use the reparametrization trick to allow backpropagation through the sampling.

To derive the training objective, we employ importance sampling and the Jensen's inequality:

$$\begin{aligned}
&\log p(o_{1:T} | a_{1:T}) \\
&\quad = \log \mathbb{E}_{p(s_{1:T}|a_{1:T})} \prod_{t=1}^T p(o_t | s_t) \\
&\quad = \log \mathbb{E}_{q(s_{1:T}|o_{1:T},a_{1:T})} \prod_{t=1}^T p(o_t | s_t) p(s_t | s_{t-1}, a_{t-1}) / q(s_t | o_{\le t}, a_{<t}) \\
&\quad \ge \mathbb{E}_{q(s_{1:T}|o_{1:T},a_{1:T})} \sum_{t=1}^T \log p(o_t | s_t) + \log p(s_t | s_{t-1}, a_{t-1}) - \log q(s_t | o_{\le t}, a_{<t}) \\
&\quad = \sum_{t=1}^T \Big(\underbrace{\mathbb{E}_{q(s_t|o_{\le t},a_{<t})} \log p(o_t | s_t)}_{\text{reconstruction}} - \underbrace{\mathbb{E}_{q(s_{t-1}|o_{\le t-1},a_{<t-1})} D_{\mathrm{KL}}\big(q(s_t | o_{\le t}, a_{<t}) \| p(s_t | s_{t-1}, a_{t-1})\big)}_{\text{complexity}} \Big).
\end{aligned}$$

The purely stochastic transitions nevertheless struggle to store information for multiple timesteps. Therefore, the authors propose to include a deterministic path to the model, obtaining the **recurrent state-space model (RSSM)**:



(a) Deterministic model (RNN) (b) Stochastic model (SSM) (c) Recurrent state-space model (RSSM)

Figure 2 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

$$\begin{aligned}
\text{deterministic state model:} &\quad h_t = f(h_{t-1}, s_{t-1}, a_{t-1}),\\
\text{stochastic state function:} &\quad s_t \sim p(s_t|h_t),\\
\text{observation function:} &\quad o_t \sim p(o_t|h_t, s_t),\\
\text{reward function:} &\quad r_t \sim p(r_t|h_t, s_t),\\
\text{encoder:} &\quad q_t \sim q(s_t|h_t, o_t).
\end{aligned}$$

Table 1: Comparison of PlaNet to the model-free algorithms A3C and D4PG reported by Tassa et al. (2018). The training curves for these are shown as orange lines in Figure 4 and as solid green lines in Figure 6 in their paper. From these, we estimate the number of episodes that D4PG takes to achieve the final performance of PlaNet to estimate the data efficiency gain. We further include CEM planning ($H = 12, I = 10, J = 1000, K = 100$) with the true simulator instead of learned dynamics as an estimated upper bound on performance. Numbers indicate mean final performance over 5 seeds and 10 trajectories.

Method	Modality	Episodes	Cartpole Swing Up	Reacher Easy	Cheetah Run	Finger Spin	Cup Catch	Walker Walk
A3C	proprioceptive	100,000	558	285	214	129	105	311
D4PG	pixels	100,000	862	967	524	985	980	968
PlaNet (ours)	pixels	1,000	821	832	662	700	930	951
CEM + true simulator	simulator state	0	850	964	656	825	993	994
Data efficiency gain PlaNet over D4PG (factor)			250	40	500+	300	100	90

Table 1 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

PlaNet – Ablations

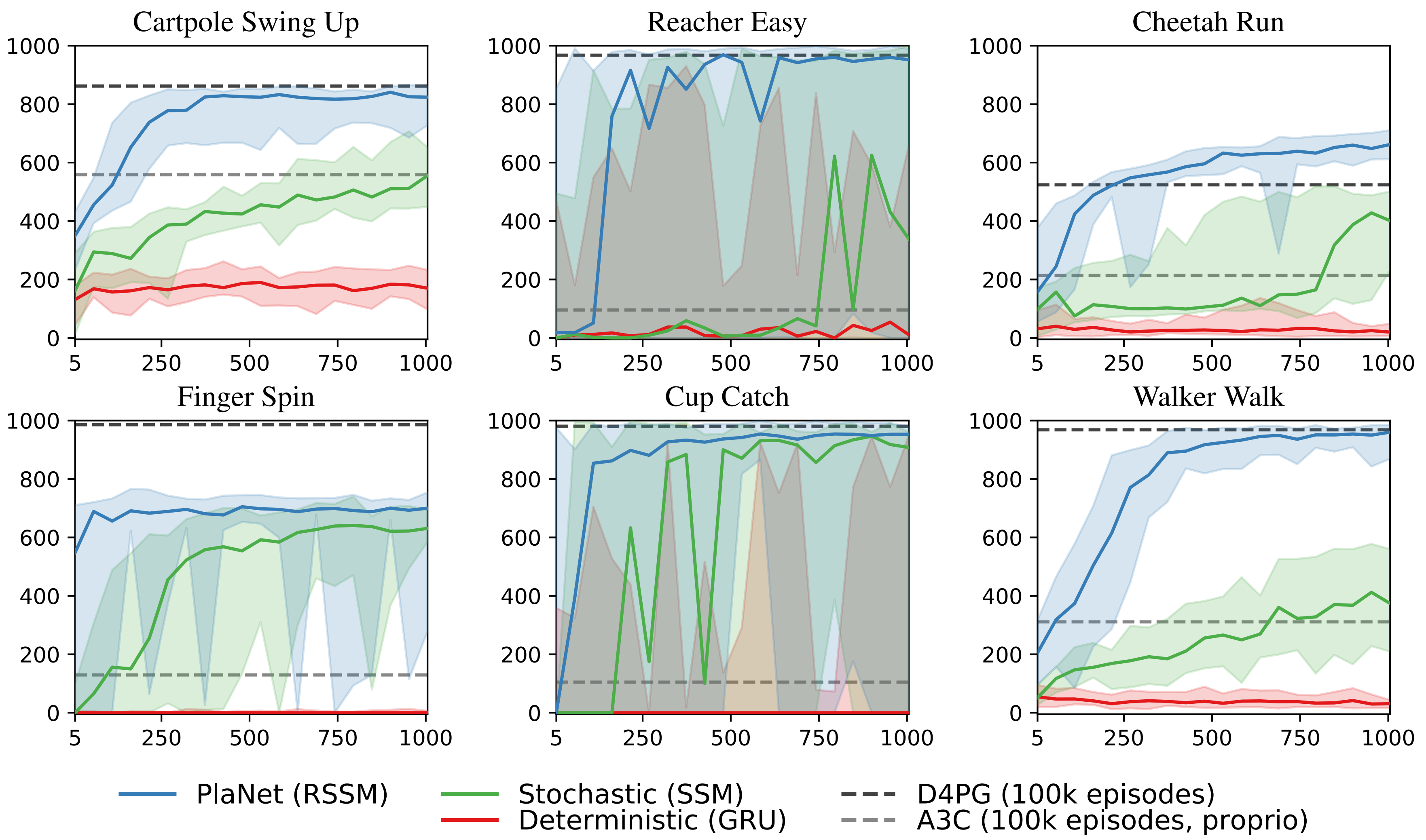


Figure 4 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551

PlaNet – Ablations

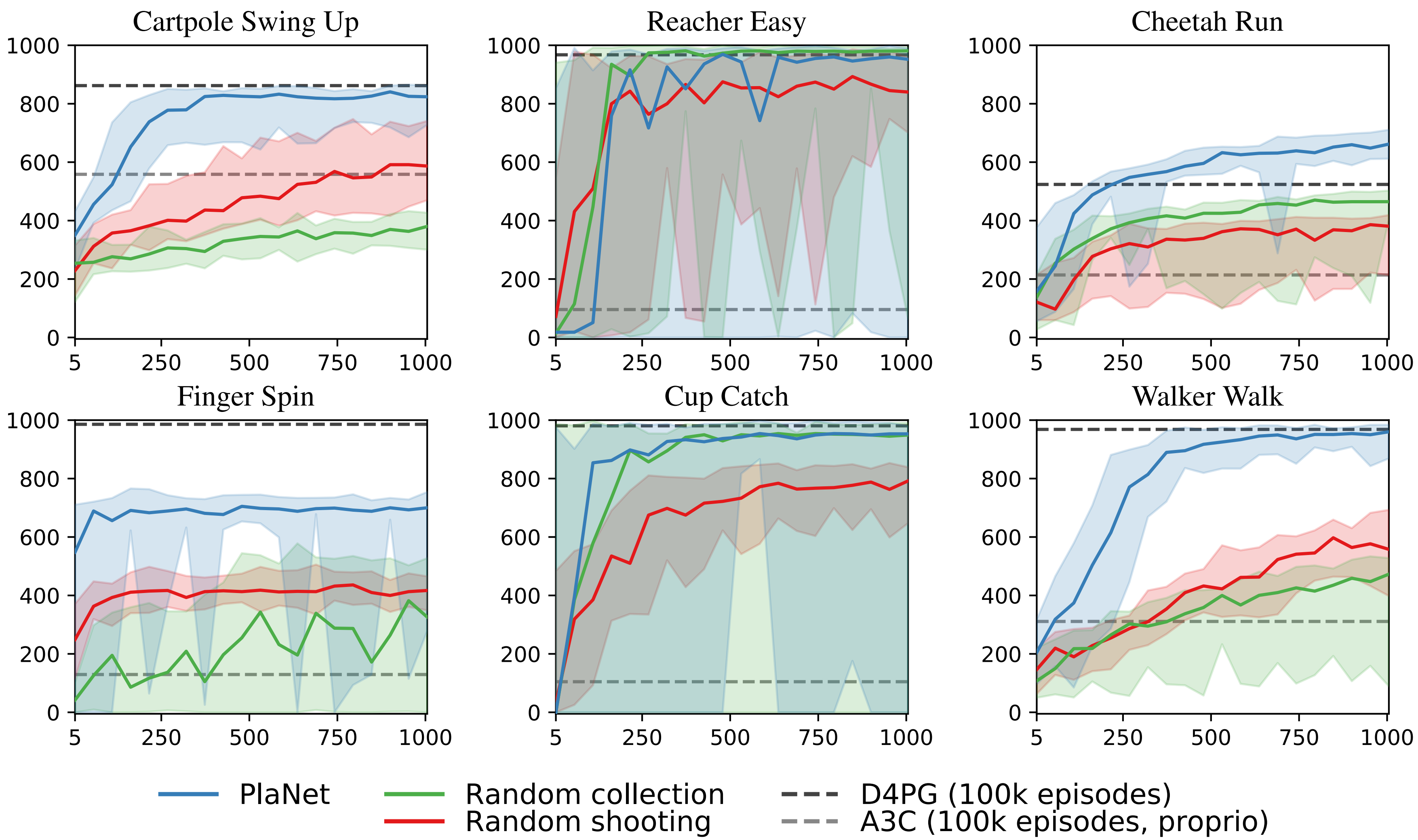


Figure 5 of "Learning Latent Dynamics for Planning from Pixels", https://arxiv.org/abs/1811.04551