DeepMind

Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

Julian Schrittwieser*, Ioannis Antonoglou*, Thomas Hubert*, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, David Silver*

*These authors contributed equally to this work.

Motivation

Bring the power of planning with MCTS and deep networks to an increasingly larger set of domains. Take advantage of planning with a learned model to learn more efficiently.

1 Planning with a Learned Model

Representation $s^0 = h_\theta(o_1, ..., o_t)$

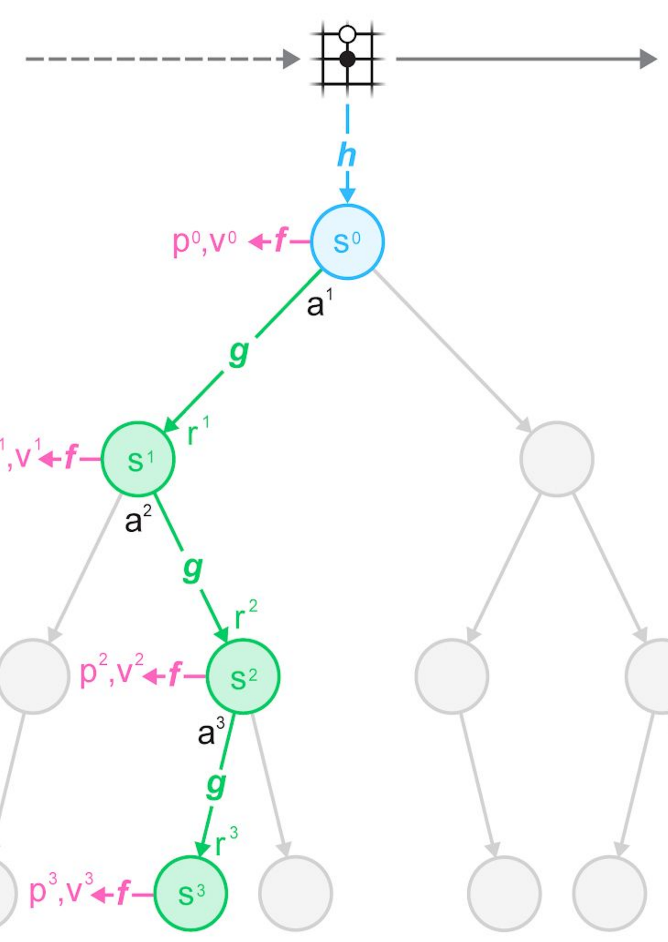
Maps from observations to hidden state of the network

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

Computes the next hidden state, given the current hidden state and action.

Prediction $\mathbf{p}^k, v^k = f_\theta(s^k)$

Predicts policy and value given a hidden state.

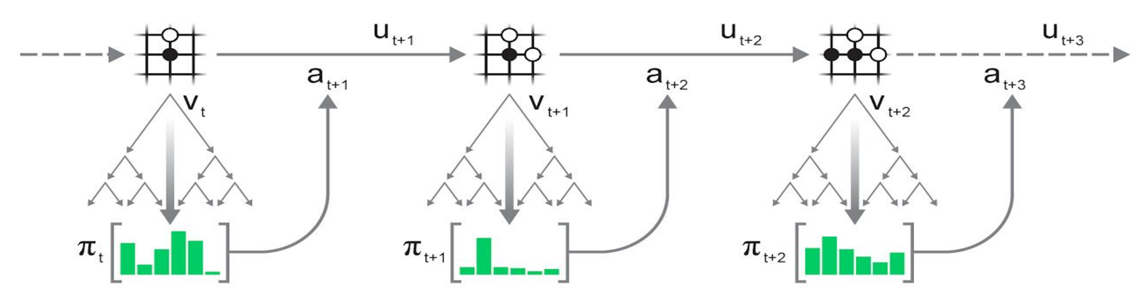


We use these three functions to plan with Monte Carlo Tree Search (MCTS) as illustrated in the diagram above, where policy and value predictions are combined according to the UCB formula:

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

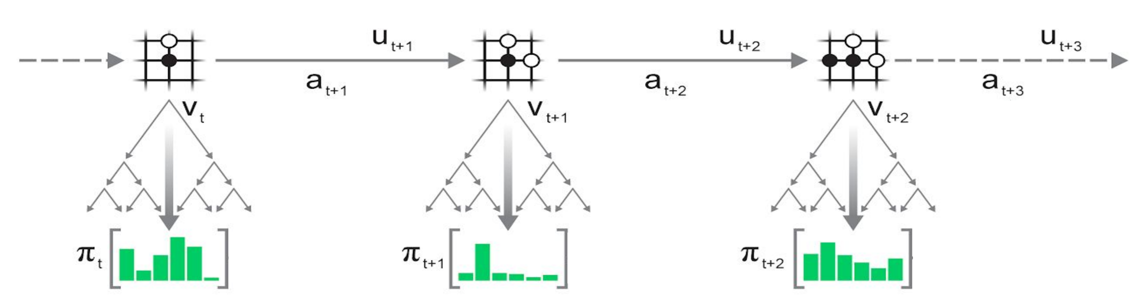
2 Data Generation

In each state, we run a Monte Carlo Tree Search using the learned model as described above; then we sample the action to play proportional to the visit counts at the root.



Reanalyze

Using existing trajectories, we re-run the MCTS to update the stored search statistics, generating new targets for the policy prediction. The new search statistics are only used to train the policy, the trajectory is unchanged.

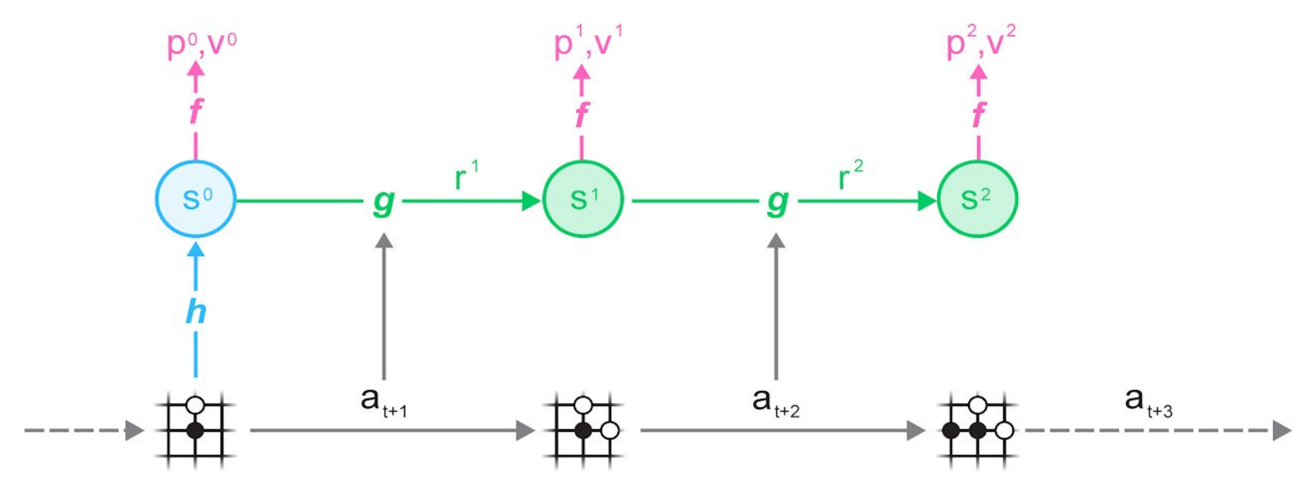


The bootstrap values in the n-step return value targets are computed in the learner using a target network.

3 Learning

We keep a replay buffer of recent trajectories from which we sample training positions according to prioritized replay.

We then unroll our network for 5 steps starting from the sampled position. On each step, the network receives the action executed in that position as an input, and has to predict reward, value and policy targets.

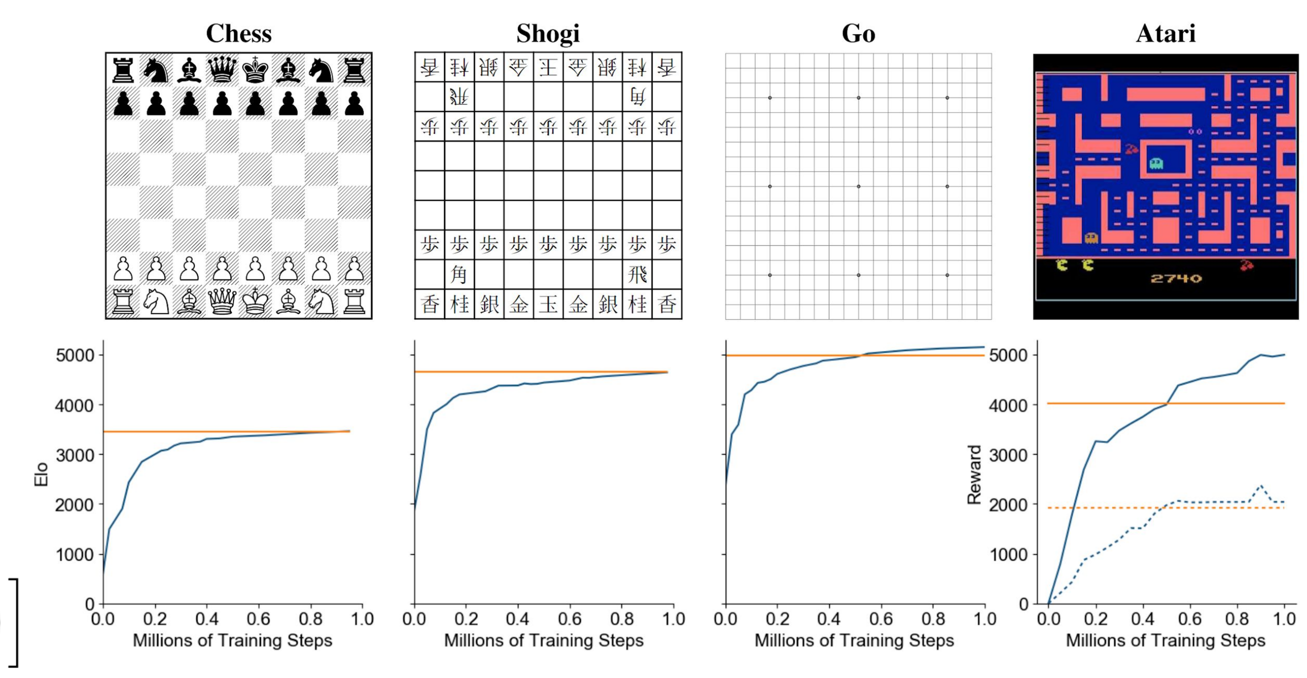


The target for the reward is the real reward observed in that transition; for the policy it is the visit distribution of the MCTS; the value is regressed towards the N-step return:

$$G^n(S_t) = \sum_{\tau=0}^{n-1} \gamma^\tau r_{\tau+t+1} + \gamma^n V(S_{t+n})$$

Results

We evaluated MuZero both on classical board games - to allow direct comparisons against AlphaZero - as well as Atari, a widely used domain for model-free agents. MuZero reached **State of the Art performance** in all of these domains, matching AlphaZero's performance in board games and exceeding previous agents in Atari.



Agent	Median	Mean	Env. Frames	Training Time
Ape-X [18]	434.1%	1695.6%	22.8B	5 days
R2D2 [21]	1920.6%	4024.9%	37.5B	5 days
MuZero	**2041.1%**	**4999.2%**	20.0B	12 hours
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

In Atari, we performed experiments in two different regimes to better compare against existing algorithms.

In the large data regime, we trained using 0.1 samples per state, for a total of 20 billion environment frames, to show scaling to maximum performance.

In the small data regime, we restricted to the standard 200 million frames by sampling each state on average 2.0 times during training, and generated 80% of the training data using reanalyze.

In both settings, MuZero achieved a **new State of the Art**.

MCTS in Single Player Games

Value Rescaling

Single player games usually have rewards and values of arbitrary scale. To obtain a value of the same order of magnitude as the prior for use in the UCB formula, we rescale the value:

$$v' = \frac{v - v_{min}}{v_{max} - v_{min}}$$

Here, the min and max can be computed over the entire search tree, or only over the current node and its children.

Intermediate Rewards

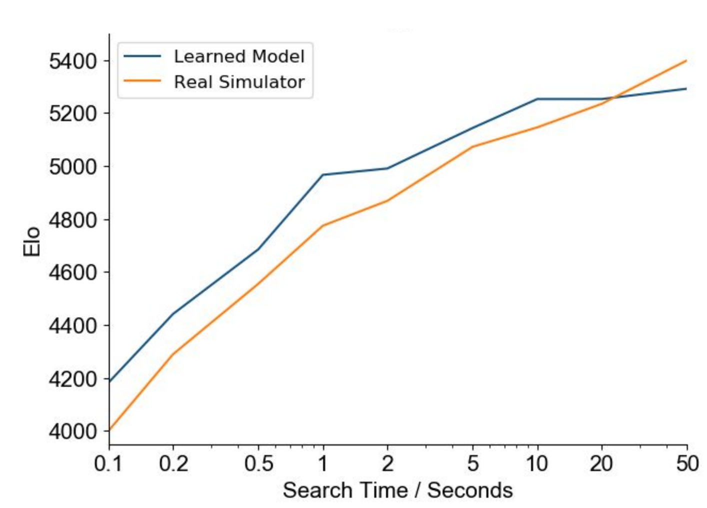
In contrast to board games like Go or Chess, single player games also commonly have intermediate rewards during an episode.

We extend the MCTS to store predicted rewards at every tree node, and the backpropagation to include rewards and discounting.

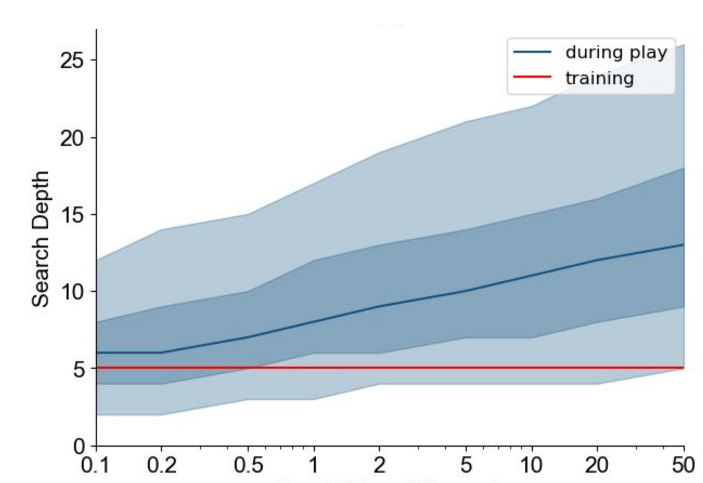
Ablations

Scaling with search time in Go

To confirm that MuZero performs as well as AlphaZero, we first investigated the scaling behaviour in Go.

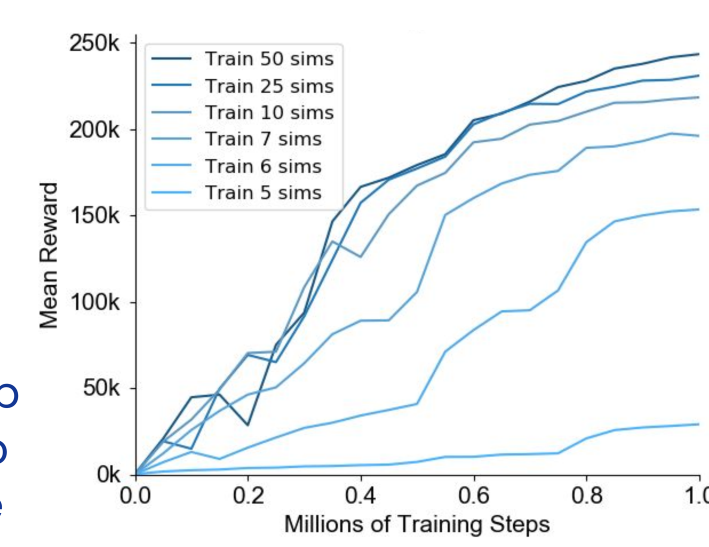


As expected, we observe increasing performance with longer search time, up to very long searches (two orders of magnitude more search time than during training) where eventually the model predictions start to degrade.



As search time is scaled up, the model is unrolled several times longer than during training. Despite this, performance scales well and remains stable even when unrolling three times longer than during training.

Policy Improvement during Training in Ms. Pacman



Next, we confirmed the benefit of search during training on the example of Ms. Pacman. We observe faster learning and increasing final performance as the number of simulations per search is increased.

Policy Improvement after Training in Atari

We also investigate the impact of search after training. We still observe an improvement in score with more search, but in contrast to Go the improvement is much smaller - due to the simpler nature of Atari, at the end of training even the policy network alone has learned to play many games perfectly, which leaves no room for further improvement through search.