arXiv:1911.08265v2 [cs.LG] 21 Feb 2020

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

Julian Schrittwieser,[1*] Ioannis Antonoglou,[1,2*] Thomas Hubert,[1*]
Karen Simonyan,[1] Laurent Sifre,[1] Simon Schmitt,[1] Arthur Guez,[1]
Edward Lockhart,[1] Demis Hassabis,[1] Thore Graepel,[1,2] Timothy Lillicrap,[1]
David Silver[1,2*]

[1]DeepMind, 6 Pancras Square, London N1C 4AG.
[2]University College London, Gower Street, London WC1E 6BT.
[*]These authors contributed equally to this work.

Abstract

Constructing agents with planning capabilities has long been one of the main challenges in the pursuit of artificial intelligence. Tree-based planning methods have enjoyed huge success in challenging domains, such as chess and Go, where a perfect simulator is available. However, in real-world problems the dynamics governing the environment are often complex and unknown. In this work we present the *MuZero* algorithm which, by combining a tree-based search with a learned model, achieves superhuman performance in a range of challenging and visually complex domains, without any knowledge of their underlying dynamics. *MuZero* learns a model that, when applied iteratively, predicts the quantities most directly relevant to planning: the reward, the action-selection policy, and the value function. When evaluated on 57 different Atari games - the canonical video game environment for testing AI techniques, in which model-based planning approaches have historically struggled - our new algorithm achieved a new state of the art. When evaluated on Go, chess and shogi, without any knowledge of the game rules, *MuZero* matched the superhuman performance of the *AlphaZero* algorithm that was supplied with the game rules.

1 Introduction

Planning algorithms based on lookahead search have achieved remarkable successes in artificial intelligence. Human world champions have been defeated in classic games such as checkers [34], chess [5], Go [38] and poker [3, 26], and planning algorithms have had real-world impact in applications from logistics [47] to chemical synthesis [37]. However, these planning algorithms all rely on knowledge of the environment's dynamics, such as the rules of the game or an accurate simulator, preventing their direct application to real-world domains like robotics, industrial control, or intelligent assistants.

Model-based reinforcement learning (RL) [42] aims to address this issue by first learning a model of the environment's dynamics, and then planning with respect to the learned model. Typically, these models have either focused on reconstructing the true environmental state [8, 16, 24], or the sequence of full observations [14, 20]. However, prior work [4, 14, 20] remains far from the state of the art in visually rich domains, such as Atari 2600 games [2]. Instead, the most successful methods are based on model-free RL [9, 21, 18] – i.e. they estimate the optimal policy and/or value function directly from interactions with the environment. However, model-free algorithms are in turn far from the state of the art in domains that require precise and sophisticated lookahead, such as chess and Go.

crackbitswilp.in

In this paper, we introduce *MuZero*, a new approach to model-based RL that achieves state-of-the-art performance in Atari 2600, a visually complex set of domains, while maintaining superhuman performance in precision planning tasks such as chess, shogi and Go. *MuZero* builds upon *AlphaZero*'s [39] powerful search and search-based policy iteration algorithms, but incorporates a learned model into the training procedure. *MuZero* also extends *AlphaZero* to a broader set of environments including single agent domains and non-zero rewards at intermediate time-steps.

The main idea of the algorithm (summarized in Figure 1) is to predict those aspects of the future that are directly relevant for planning. The model receives the observation (e.g. an image of the Go board or the Atari screen) as an input and transforms it into a hidden state. The hidden state is then updated iteratively by a recurrent process that receives the previous hidden state and a hypothetical next action. At every one of these steps the model predicts the policy (e.g. the move to play), value function (e.g. the predicted winner), and immediate reward (e.g. the points scored by playing a move). The model is trained end-to-end, with the sole objective of accurately estimating these three important quantities, so as to match the improved estimates of policy and value generated by search as well as the observed reward. There is no direct constraint or requirement for the hidden state to capture all information necessary to reconstruct the original observation, drastically reducing the amount of information the model has to maintain and predict; nor is there any requirement for the hidden state to match the unknown, true state of the environment; nor any other constraints on the semantics of state. Instead, the hidden states are free to represent state in whatever way is relevant to predicting current and future values and policies. Intuitively, the agent can invent, internally, the rules or dynamics that lead to most accurate planning.

2 Prior Work

Reinforcement learning may be subdivided into two principal categories: model-based, and model-free [42]. Model-based RL constructs, as an intermediate step, a model of the environment. Classically, this model is represented by a Markov-decision process (MDP) [31] consisting of two components: a state transition model, predicting the next state, and a reward model, predicting the expected reward during that transition. The model is typically conditioned on the selected action, or a temporally abstract behavior such as an option [43]. Once a model has been constructed, it is straightforward to apply MDP planning algorithms, such as value iteration [31] or Monte-Carlo tree search (MCTS) [7], to compute the optimal value or optimal policy for the MDP. In large or partially observed environments, the algorithm must first construct the state representation that the model should predict. This tripartite separation between representation learning, model learning, and planning is potentially problematic since the agent is not able to optimize its representation or model for the purpose of effective planning, so that, for example modeling errors may compound during planning.

A common approach to model-based RL focuses on directly modeling the observation stream at the pixel-level. It has been hypothesized that deep, stochastic models may mitigate the problems of compounding error [14, 20]. However, planning at pixel-level granularity is not computationally tractable in large scale problems. Other methods build a latent state-space model that is sufficient to reconstruct the observation stream at pixel level [48, 49], or to predict its future latent states [13, 11], which facilitates more efficient planning but still focuses the majority of the model capacity on potentially irrelevant detail. None of these prior methods has constructed a model that facilitates effective planning in visually complex domains such as Atari; results lag behind well-tuned, model-free methods, even in terms of data efficiency [45].

A quite different approach to model-based RL has recently been developed, focused end-to-end on predicting the value function [41]. The main idea of these methods is to construct an abstract MDP model such that planning in the abstract MDP is equivalent to planning in the real environment. This equivalence is achieved by ensuring *value equivalence*, i.e. that, starting from the same real state, the cumulative reward of a trajectory through the abstract MDP matches the cumulative reward of a trajectory in the real environment.

The predictron [41] first introduced value equivalent models for predicting value (without actions). Although the underlying model still takes the form of an MDP, there is no requirement for its transition model to match real states in the environment. Instead the MDP model is viewed as a hidden layer of a deep neural network. The unrolled MDP is trained such that the expected cumulative sum of rewards matches the expected value with respect to the real environment, e.g. by temporal-difference learning.

crackbitswilp.in

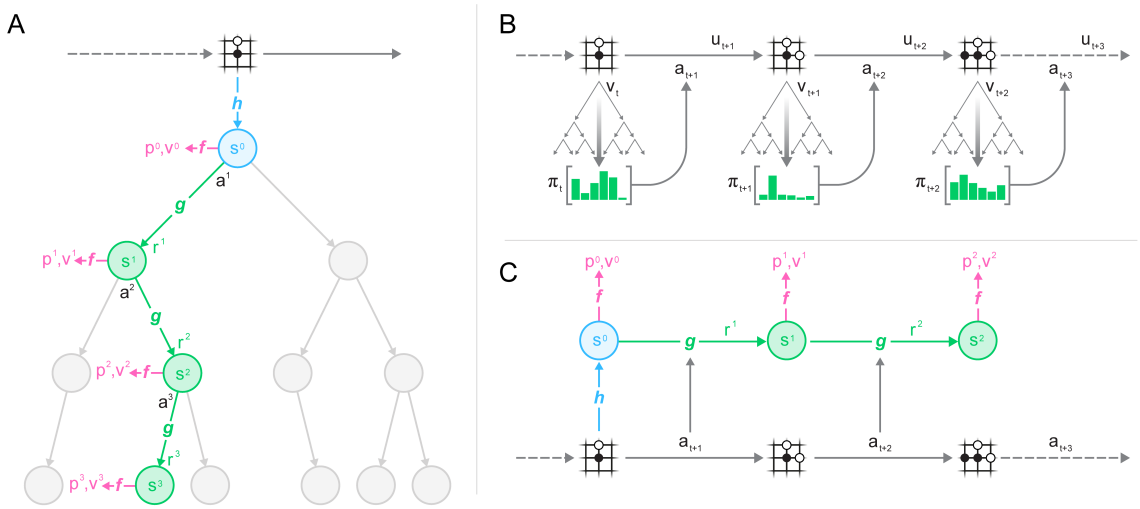


Figure 1: **Planning, acting, and training with a learned model. (A)** How *MuZero* uses its model to plan. The model consists of three connected components for representation, dynamics and prediction. Given a previous hidden state s^{k-1} and a candidate action a^k, the *dynamics* function g produces an immediate reward r^k and a new hidden state s^k. The policy p^k and value function v^k are computed from the hidden state s^k by a *prediction* function f. The initial hidden state s^0 is obtained by passing the past observations (e.g. the Go board or Atari screen) into a *representation* function h. **(B)** How *MuZero* acts in the environment. A Monte-Carlo Tree Search is performed at each timestep t, as described in A. An action a_{t+1} is sampled from the search policy π_t, which is proportional to the visit count for each action from the root node. The environment receives the action and generates a new observation o_{t+1} and reward u_{t+1}. At the end of the episode the trajectory data is stored into a replay buffer. **(C)** How *MuZero* trains its model. A trajectory is sampled from the replay buffer. For the initial step, the representation function h receives as input the past observations $o_1, ..., o_t$ from the selected trajectory. The model is subsequently unrolled recurrently for K steps. At each step k, the dynamics function g receives as input the hidden state s^{k-1} from the previous step and the real action a_{t+k}. The parameters of the representation, dynamics and prediction functions are jointly trained, end-to-end by backpropagation-through-time, to predict three quantities: the policy $\mathbf{p}^k \approx \pi_{t+k}$, value function $v^k \approx z_{t+k}$, and reward $r_{t+k} \approx u_{t+k}$, where z_{t+k} is a sample return: either the final reward (board games) or n-step return (Atari).

Value equivalent models were subsequently extended to optimising value (with actions). TreeQN [10] learns an abstract MDP model, such that a tree search over that model (represented by a tree-structured neural network) approximates the optimal value function. Value iteration networks [44] learn a local MDP model, such that value iteration over that model (represented by a convolutional neural network) approximates the optimal value function. Value prediction networks [28] are perhaps the closest precursor to *MuZero*: they learn an MDP model grounded in real actions; the unrolled MDP is trained such that the cumulative sum of rewards, conditioned on the actual sequence of actions generated by a simple lookahead search, matches the real environment. Unlike *MuZero* there is no policy prediction, and the search only utilizes value prediction.

3 *MuZero* Algorithm

We now describe the *MuZero* algorithm in more detail. Predictions are made at each time-step t, for each of $k = 1...K$ steps, by a model μ_θ, with parameters θ, conditioned on past observations $o_1, ..., o_t$ and future actions $a_{t+1}, ..., a_{t+k}$. The model predicts three future quantities: the policy $\mathbf{p}_t^k \approx \pi(a_{t+k+1}|o_1, ..., o_t, a_{t+1}, ..., a_{t+k})$, the value function $v_t^k \approx \mathbb{E}\left[u_{t+k+1} + \gamma u_{t+k+2} + ...|o_1, ..., o_t, a_{t+1}, ..., a_{t+k}\right]$, and the immediate reward $r_t^k \approx u_{t+k}$,

crackbitswilp.in

where $u_.$ is the true, observed reward, π is the policy used to select real actions, and γ is the discount function of the environment.

Internally, at each time-step t (subscripts $_t$ suppressed for simplicity), the model is represented by the combination of a *representation* function, a *dynamics* function, and a *prediction* function. The dynamics function, $r^k, s^k = g_\theta(s^{k-1}, a^k)$, is a recurrent process that computes, at each hypothetical step k, an immediate reward r^k and an internal state s^k. It mirrors the structure of an MDP model that computes the expected reward and state transition for a given state and action [31]. However, unlike traditional approaches to model-based RL [42], this internal state s^k has no semantics of environment state attached to it – it is simply the hidden state of the overall model, and its sole purpose is to accurately predict relevant, future quantities: policies, values, and rewards. In this paper, the *dynamics* function is represented deterministically; the extension to stochastic transitions is left for future work. The policy and value functions are computed from the internal state s^k by the prediction function, $\mathbf{p}^k, v^k = f_\theta(s^k)$, akin to the joint policy and value network of *AlphaZero*. The "root" state s^0 is initialized using a representation function that encodes past observations, $s^0 = h_\theta(o_1, ..., o_t)$; again this has no special semantics beyond its support for future predictions.

Given such a model, it is possible to search over hypothetical future trajectories $a^1, ..., a^k$ given past observations $o_1, ..., o_t$. For example, a naive search could simply select the k step action sequence that maximizes the value function. More generally, we may apply any MDP planning algorithm to the internal rewards and state space induced by the dynamics function. Specifically, we use an MCTS algorithm similar to *AlphaZero*'s search, generalized to allow for single agent domains and intermediate rewards (see Methods). At each internal node, it makes use of the policy, value and reward estimates produced by the current model parameters θ. The MCTS algorithm outputs a recommended policy π_t and estimated value ν_t. An action $a_{t+1} \sim \pi_t$ is then selected.

All parameters of the model are trained jointly to accurately match the policy, value, and reward, for every hypothetical step k, to corresponding target values observed after k actual time-steps have elapsed. Similarly to *AlphaZero*, the improved policy targets are generated by an MCTS search; the first objective is to minimise the error between predicted policy $\mathbf{p}_t^k$ and search policy π_{t+k}. Also like *AlphaZero*, the improved value targets are generated by playing the game or MDP. However, unlike *AlphaZero*, we allow for long episodes with discounting and intermediate rewards by *bootstrapping* n steps into the future from the search value, $z_t = u_{t+1} + \gamma u_{t+2} + ... + \gamma^{n-1} u_{t+n} + \gamma^n \nu_{t+n}$. Final outcomes $\{lose, draw, win\}$ in board games are treated as rewards $u_t \in \{-1, 0, +1\}$ occuring at the final step of the episode. Specifically, the second objective is to minimize the error between the predicted value v_t^k and the value target, z_{t+k} [1]. The reward targets are simply the observed rewards; the third objective is therefore to minimize the error between the predicted reward r_t^k and the observed reward u_{t+k}. Finally, an L2 regularization term is also added, leading to the overall loss:

$$l_t(\theta) = \sum_{k=0}^{K} l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\pi_{t+k}, \mathbf{p}_t^k) + c||\theta||^2 \tag{1}$$

where l^r, l^v, and l^p are loss functions for reward, value and policy respectively. Supplementary Figure S2 summarizes the equations governing how the *MuZero* algorithm plans, acts, and learns.

4 Results

We applied the *MuZero* algorithm to the classic board games Go, chess and shogi [2], as benchmarks for challenging planning problems, and to all 57 games in the Atari Learning Environment [2], as benchmarks for visually complex RL domains.

In each case we trained *MuZero* for $K = 5$ hypothetical steps. Training proceeded for 1 million mini-batches of size 2048 in board games and of size 1024 in Atari. During both training and evaluation, *MuZero* used 800 simulations for each search in board games, and 50 simulations for each search in Atari. The representation

[1] For chess, Go and shogi, the same squared error loss as *AlphaZero* is used for rewards and values. A cross-entropy loss was found to be more stable than a squared error when encountering rewards and values of variable scale in Atari. Cross-entropy was used for the policy loss in both cases.

[2] Imperfect information games such as Poker are not directly addressed by our method.

crackbitswilp.in

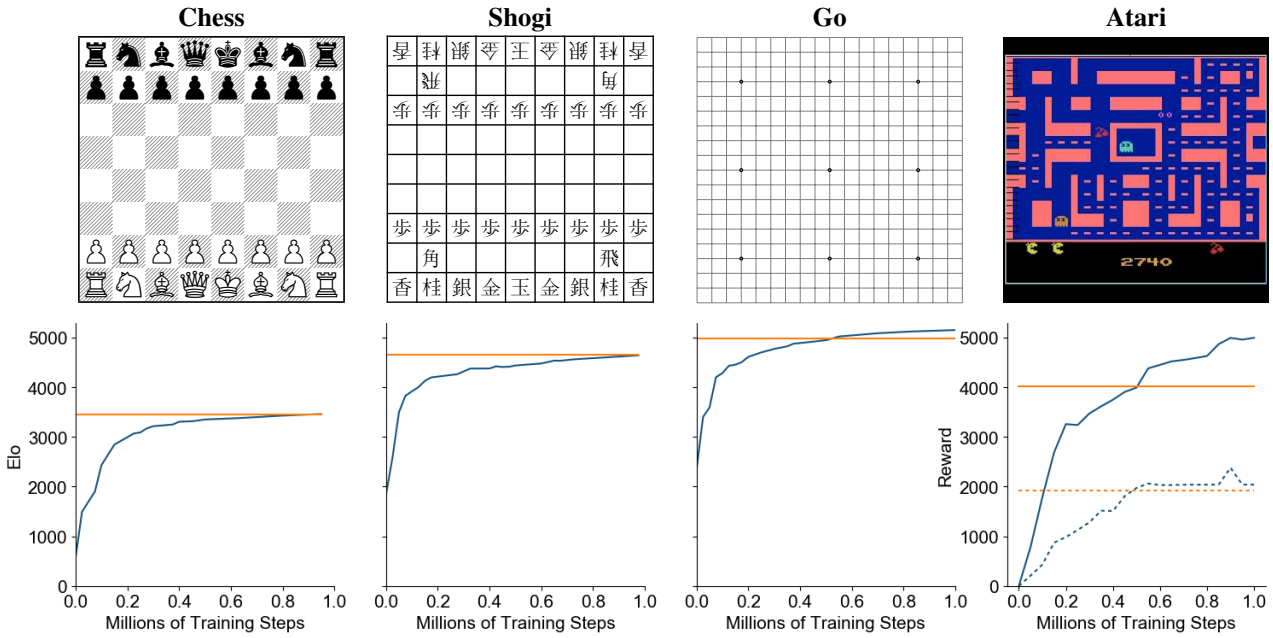


Figure 2: **Evaluation of *MuZero* throughout training in chess, shogi, Go and Atari.** The x-axis shows millions of training steps. For chess, shogi and Go, the y-axis shows Elo rating, established by playing games against *AlphaZero* using 800 simulations per move for both players. *MuZero*'s Elo is indicated by the blue line, *AlphaZero*'s Elo by the horizontal orange line. For Atari, mean (full line) and median (dashed line) human normalized scores across all 57 games are shown on the y-axis. The scores for R2D2 [21], (the previous state of the art in this domain, based on model-free RL) are indicated by the horizontal orange lines. Performance in Atari was evaluated using 50 simulations every fourth time-step, and then repeating the chosen action four times, as in prior work [25].

function uses the same convolutional [23] and residual [15] architecture as *AlphaZero*, but with 16 residual blocks instead of 20. The dynamics function uses the same architecture as the representation function and the prediction function uses the same architecture as *AlphaZero*. All networks use 256 hidden planes (see Methods for further details).

Figure 2 shows the performance throughout training in each game. In Go, *MuZero* slightly exceeded the performance of *AlphaZero*, despite using less computation per node in the search tree (16 residual blocks per evaluation in *MuZero* compared to 20 blocks in *AlphaZero*). This suggests that *MuZero* may be caching its computation in the search tree and using each additional application of the dynamics model to gain a deeper understanding of the position.

In Atari, *MuZero* achieved a new state of the art for both mean and median normalized score across the 57 games of the Arcade Learning Environment, outperforming the previous state-of-the-art method R2D2 [21] (a model-free approach) in 42 out of 57 games, and outperforming the previous best model-based approach SimPLe [20] in all games (see Table S1).

We also evaluated a second version of *MuZero* that was optimised for greater sample efficiency. Specifically, it reanalyzes old trajectories by re-running the MCTS using the latest network parameters to provide fresh targets (see Appendix H). When applied to 57 Atari games, using 200 million frames of experience per game, *MuZero Reanalyze* achieved 731% median normalized score, compared to 192%, 231% and 431% for previous state-of-the-art model-free approaches IMPALA [9], Rainbow [17] and LASER [36] respectively.

To understand the role of the model in *MuZero* we also ran several experiments, focusing on the board game of Go and the Atari game of Ms. Pacman.

First, we tested the scalability of planning (Figure 3A), in the canonical planning problem of Go. We compared the performance of search in *AlphaZero*, using a perfect model, to the performance of search in *MuZero*, using a

crackbitswilp.in

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: **Comparison of *MuZero* against previous agents in Atari**. We compare separately against agents trained in large (top) and small (bottom) data settings; all agents other than *MuZero* used model-free RL techniques. Mean and median scores are given, compared to human testers. The best results are highlighted in **bold**. *MuZero* sets a new state of the art in both settings. [a]Hyper-parameters were tuned per game.

learned model. Specifically, the fully trained *AlphaZero* or *MuZero* was evaluated by comparing MCTS with different thinking times. *MuZero* matched the performance of a perfect model, even when doing much larger searches (up to 10s thinking time) than those from which the model was trained (around 0.1s thinking time, see also Figure S3A).

We also investigated the scalability of planning across all Atari games (see Figure 3B). We compared MCTS with different numbers of simulations, using the fully trained *MuZero*. The improvements due to planning are much less marked than in Go, perhaps because of greater model inaccuracy; performance improved slightly with search time, but plateaued at around 100 simulations. Even with a single simulation – i.e. when selecting moves solely according to the policy network – *MuZero* performed well, suggesting that, by the end of training, the raw policy has learned to internalise the benefits of search (see also Figure S3B).

Next, we tested our model-based learning algorithm against a comparable model-free learning algorithm (see Figure 3C). We replaced the training objective of *MuZero* (Equation 1) with a model-free Q-learning objective (as used by R2D2), and the dual value and policy heads with a single head representing the Q-function $Q(\cdot|s_t)$. Subsequently, we trained and evaluated the new model without using any search. When evaluated on Ms. Pacman, our model-free algorithm achieved identical results to R2D2, but learned significantly slower than *MuZero* and converged to a much lower final score. We conjecture that the search-based policy improvement step of *MuZero* provides a stronger learning signal than the high bias, high variance targets used by Q-learning.

To better understand the nature of *MuZero*'s learning algorithm, we measured how *MuZero*'s training scales with respect to the amount of search it uses *during* training. Figure 3D shows the performance in Ms. Pacman, using an MCTS of different simulation counts per move throughout training. Surprisingly, and in contrast to previous work [1], even with only 6 simulations per move – fewer than the number of actions – *MuZero* learned an effective policy and improved rapidly. With more simulations performance jumped significantly higher. For analysis of the policy improvement during each individual iteration, see also Figure S3 C and D.

5 Conclusions

Many of the breakthroughs in artificial intelligence have been based on either high-performance planning [5, 38, 39] or model-free reinforcement learning methods [25, 29, 46]. In this paper we have introduced a method that combines the benefits of both approaches. Our algorithm, *MuZero*, has both matched the superhuman performance of high-performance planning algorithms in their favored domains – logically complex board games such as chess and Go – and outperformed state-of-the-art model-free RL algorithms in their favored domains – visually complex Atari games. Crucially, our method does not require any knowledge of the game rules or environment dynamics, potentially paving the way towards the application of powerful learning and planning methods to a host of real-world domains for which there exists no perfect simulator.

crackbitswilp.in

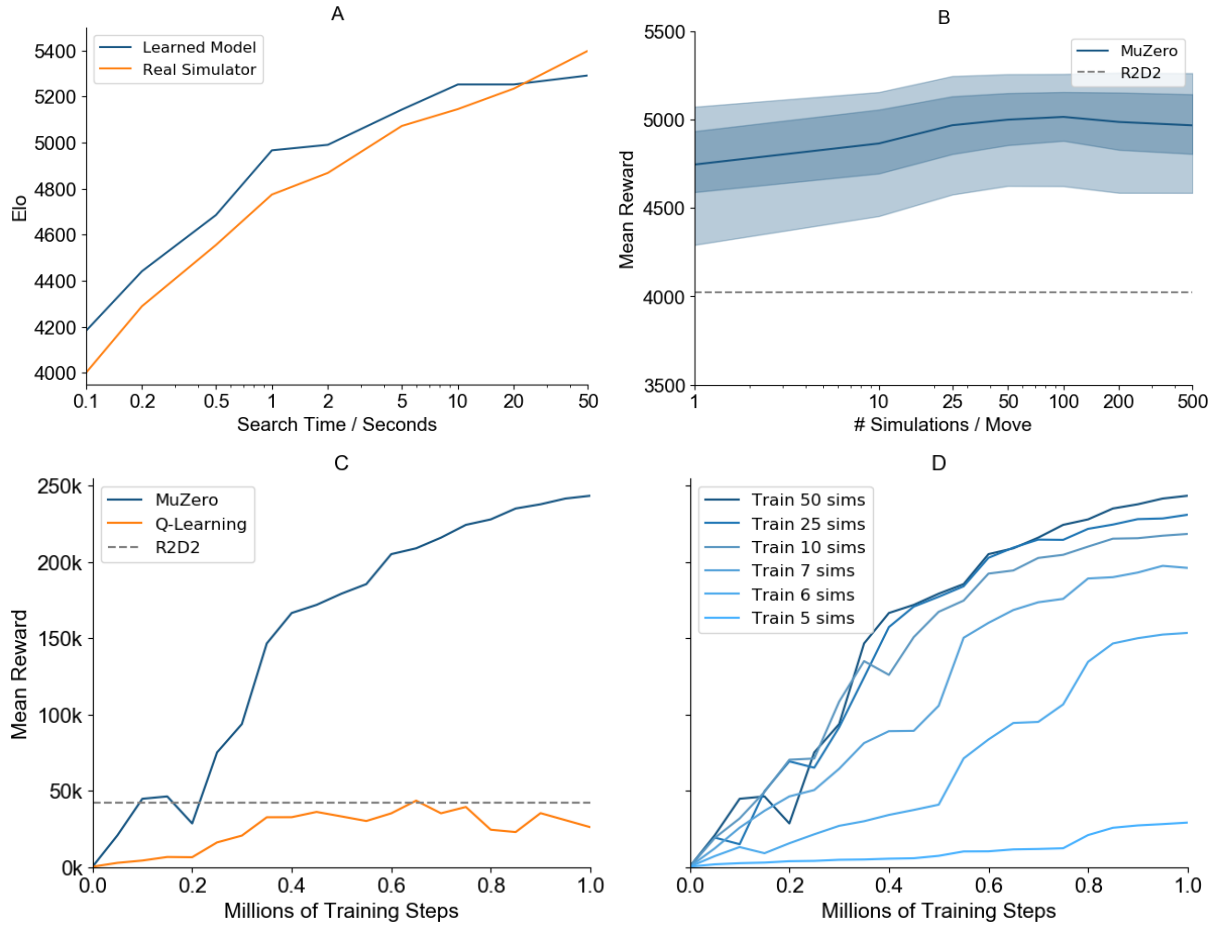


Figure 3: **Evaluations of *MuZero* on Go (A), all 57 Atari Games (B) and Ms. Pacman (C-D)**. (**A**) Scaling with search time per move in Go, comparing the learned model with the ground truth simulator. Both networks were trained at 800 simulations per search, equivalent to 0.1 seconds per search. Remarkably, the learned model is able to scale well to up to two orders of magnitude longer searches than seen during training. (**B**) Scaling of final human normalized mean score in Atari with the number of simulations per search. The network was trained at 50 simulations per search. Dark line indicates mean score, shaded regions indicate 25th to 75th and 5th to 95th percentiles. The learned model's performance increases up to 100 simulations per search. Beyond, even when scaling to much longer searches than during training, the learned model's performance remains stable and only decreases slightly. This contrasts with the much better scaling in Go (A), presumably due to greater model inaccuracy in Atari than Go. (**C**) Comparison of MCTS based training with Q-learning in the *MuZero* framework on Ms. Pacman, keeping network size and amount of training constant. The state of the art Q-Learning algorithm R2D2 is shown as a baseline. Our Q-Learning implementation reaches the same final score as R2D2, but improves slower and results in much lower final performance compared to MCTS based training. (**D**) Different networks trained at different numbers of simulations per move, but all evaluated at 50 simulations per move. Networks trained with more simulations per move improve faster, consistent with ablation (B), where the policy improvement is larger when using more simulations per move. Surprisingly, *MuZero* can learn effectively even when training with less simulations per move than are enough to cover all 8 possible actions in Ms. Pacman.

crackbitswilp.in

6 Acknowledgments

Lorrayne Bennett, Oliver Smith and Chris Apps for organizational assistance; Koray Kavukcuoglu for reviewing the paper; Thomas Anthony, Matthew Lai, Nenad Tomasev, Ulrich Paquet, Sumedh Ghaisas for many fruitful discussions; and the rest of the DeepMind team for their support.

References

[1] Kamyar Azizzadenesheli, Brandon Yang, Weitang Liu, Emma Brunskill, Zachary C. Lipton, and Animashree Anandkumar. Surprising negative results for generative adversarial tree search. *CoRR*, abs/1806.05780, 2018.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[3] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[4] Lars Buesing, Theophane Weber, Sebastien Racaniere, SM Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

[5] Murray Campbell, A. Joseph Hoane, Jr., and Feng-hsiung Hsu. Deep blue. *Artif. Intell.*, 134(1-2):57–83, January 2002.

[6] R. Coulom. Whole-history rating: A Bayesian rating system for players of time-varying strength. In *International Conference on Computers and Games*, pages 113–124, 2008.

[7] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[8] MP. Deisenroth and CE. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pages 465–472. Omnipress, 2011.

[9] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[10] Gregory Farquhar, Tim Rocktaeschel, Maximilian Igl, and Shimon Whiteson. TreeQN and ATreec: Differentiable tree planning for deep reinforcement learning. In *International Conference on Learning Representations*, 2018.

[11] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2170–2179, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[12] Cloud tpu. `https://cloud.google.com/tpu/`. Accessed: 2019.

[13] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 2455–2467, USA, 2018. Curran Associates Inc.

crackbitswilp.in

[14] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *14th European Conference on Computer Vision*, pages 630–645, 2016.

[16] Nicolas Heess, Greg Wayne, David Silver, Timothy Lillicrap, Yuval Tassa, and Tom Erez. Learning continuous control policies by stochastic value gradients. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2944–2952, Cambridge, MA, USA, 2015. MIT Press.

[17] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[18] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. In *International Conference on Learning Representations*, 2018.

[19] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[20] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[21] Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019.

[22] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[24] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1071–1079. Curran Associates, Inc., 2014.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[26] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[27] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015.

[28] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6118–6128, 2017.

[29] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

crackbitswilp.in

[30] Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado van Hasselt, John Quan, Mel Večerík, et al. Observe and look further: Achieving consistent performance on atari. *arXiv preprint arXiv:1805.11593*, 2018.

[31] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[32] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

[33] Maarten PD Schadd, Mark HM Winands, H Jaap Van Den Herik, Guillaume MJ-B Chaslot, and Jos WHM Uiterwijk. Single-player monte-carlo tree search. In *International Conference on Computers and Games*, pages 1–12. Springer, 2008.

[34] Jonathan Schaeffer, Joseph Culberson, Norman Treloar, Brent Knight, Paul Lu, and Duane Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53(2-3):273–289, 1992.

[35] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.

[36] Simon Schmitt, Matteo Hessel, and Karen Simonyan. Off-policy actor-critic with shared experience replay. *arXiv preprint arXiv:1909.11583*, 2019.

[37] Marwin HS Segler, Mike Preuss, and Mark P Waller. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604, 2018.

[38] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[39] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[40] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, October 2017.

[41] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3191–3199. JMLR. org, 2017.

[42] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[43] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[44] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

[45] Hado van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? *arXiv preprint arXiv:1906.05243*, 2019.

crackbitswilp.in

[46] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

[47] I Vlahavas and I Refanidis. Planning and scheduling. *EETN, Greece, Tech. Rep*, 2013.

[48] Niklas Wahlström, Thomas B. Schön, and Marc Peter Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *CoRR*, abs/1502.02251, 2015.

[49] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2746–2754, Cambridge, MA, USA, 2015. MIT Press.

Supplementary Materials

- Pseudocode description of the *MuZero* algorithm.
- Data for Figures 2, 3, S2, S3, S4 and Tables 1, S1, S2 in JSON format.

Supplementary materials can be accessed from the ancillary file section of the arXiv submission.

Appendix A Comparison to *AlphaZero*

MuZero is designed for a more general setting than *AlphaGo Zero* [40] and *AlphaZero* [39].

In *AlphaGo Zero* and *AlphaZero* the planning process makes use of two separate components: a simulator implements the rules of the game, which are used to update the state of the game while traversing the search tree; and a neural network jointly predicts the corresponding policy and value of a board position produced by the simulator (see Figure 1 A).

Specifically, *AlphaGo Zero* and *AlphaZero* use knowledge of the rules of the game in three places: (1) state transitions in the search tree, (2) actions available at each node of the search tree, (3) episode termination within the search tree. In *MuZero*, all of these have been replaced with the use of a single implicit model learned by a neural network (see Figure 1 B):

1) State transitions. *AlphaZero* had access to a perfect simulator of the true dynamics process. In contrast, *MuZero* employs a learned dynamics model within its search. Under this model, each node in the tree is represented by a corresponding hidden state; by providing a hidden state s_{k-1} and an action a_k to the model the search algorithm can transition to a new node $s_k = g(s_{k-1}, a_k)$.

2) Actions available. *AlphaZero* used the set of legal actions obtained from the simulator to mask the prior produced by the network everywhere in the search tree. *MuZero* only masks legal actions at the root of the search tree where the environment can be queried, but does not perform any masking within the search tree. This is possible because the network rapidly learns not to predict actions that never occur in the trajectories it is trained on.

3) Terminal nodes. *AlphaZero* stopped the search at tree nodes representing terminal states and used the terminal value provided by the simulator instead of the value produced by the network. *MuZero* does not give special treatment to terminal nodes and always uses the value predicted by the network. Inside the tree, the search can proceed past a terminal node - in this case the network is expected to always predict the same value. This is achieved by treating terminal states as absorbing states during training.

In addition, *MuZero* is designed to operate in the general reinforcement learning setting: single-agent domains with discounted intermediate rewards of arbitrary magnitude. In contrast, *AlphaGo Zero* and *AlphaZero* were designed to operate in two-player games with undiscounted terminal rewards of ± 1.

crackbitswilp.in

Appendix B Search

We now describe the search algorithm used by *MuZero*. Our approach is based upon Monte-Carlo tree search with upper confidence bounds, an approach to planning that converges asymptotically to the optimal policy in single agent domains and to the minimax value function in zero sum games [22].

Every node of the search tree is associated with an internal state s. For each action a from s there is an edge (s, a) that stores a set of statistics $\{N(s,a), Q(s,a), P(s,a), R(s,a), S(s,a)\}$, respectively representing visit counts N, mean value Q, policy P, reward R, and state transition S.

Similar to *AlphaZero*, the search is divided into three stages, repeated for a number of simulations.

Selection: Each simulation starts from the internal root state s^0, and finishes when the simulation reaches a leaf node s^l. For each hypothetical time-step $k = 1...l$ of the simulation, an action a^k is selected according to the stored statistics for internal state s^{k-1}, by maximizing over an upper confidence bound [32][39],

$$a^k = \arg\max_a \left[Q(s,a) + P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \left(c_1 + \log \left(\frac{\sum_b N(s,b) + c_2 + 1}{c_2} \right) \right) \right] \tag{2}$$

The constants c_1 and c_2 are used to control the influence of the prior $P(s,a)$ relative to the value $Q(s,a)$ as nodes are visited more often. In our experiments, $c_1 = 1.25$ and $c_2 = 19652$.

For $k < l$, the next state and reward are looked up in the state transition and reward table $s^k = S(s^{k-1}, a^k)$, $r^k = R(s^{k-1}, a^k)$.

Expansion: At the final time-step l of the simulation, the reward and state are computed by the dynamics function, $r^l, s^l = g_\theta(s^{l-1}, a^l)$, and stored in the corresponding tables, $R(s^{l-1}, a^l) = r^l, S(s^{l-1}, a^l) = s^l$. The policy and value are computed by the prediction function, $\mathbf{p}^l, v^l = f_\theta(s^l)$. A new node, corresponding to state s^l is added to the search tree. Each edge (s^l, a) from the newly expanded node is initialized to $\{N(s^l, a) = 0, Q(s^l, a) = 0, P(s^l, a) = \mathbf{p}^l\}$. Note that the search algorithm makes at most one call to the dynamics function and prediction function respectively per simulation; the computational cost is of the same order as in *AlphaZero*.

Backup: At the end of the simulation, the statistics along the trajectory are updated. The backup is generalized to the case where the environment can emit intermediate rewards, have a discount γ different from 1, and the value estimates are unbounded [3]. For $k = l...0$, we form an $l - k$-step estimate of the cumulative discounted reward, bootstrapping from the value function v^l,

$$G^k = \sum_{\tau=0}^{l-1-k} \gamma^\tau r_{k+1+\tau} + \gamma^{l-k} v^l \tag{3}$$

For $k = l...1$, we update the statistics for each edge (s^{k-1}, a^k) in the simulation path as follows,

$$\begin{aligned} Q(s^{k-1}, a^k) &:= \frac{N(s^{k-1}, a^k) \cdot Q(s^{k-1}, a^k) + G^k}{N(s^{k-1}, a^k) + 1} \\ N(s^{k-1}, a^k) &:= N(s^{k-1}, a^k) + 1 \end{aligned} \tag{4}$$

In two-player zero sum games the value functions are assumed to be bounded within the $[0, 1]$ interval. This choice allows us to combine value estimates with probabilities using the pUCT rule (Eqn 2). However, since in many environments the value is unbounded, it is necessary to adjust the pUCT rule. A simple solution would be to use the maximum score that can be observed in the environment to either re-scale the value or set the pUCT constants appropriately [33]. However, both solutions are game specific and require adding prior knowledge to the *MuZero* algorithm. To avoid this, *MuZero* computes normalized Q value estimates $\overline{Q} \in [0, 1]$ by using the minimum-maximum values observed in the search tree up to that point. When a node is reached during the selection stage, the algorithm computes the normalized $\overline{Q}$ values of its edges to be used in the pUCT rule using the equation below:

$$\overline{Q}(s^{k-1}, a^k) = \frac{Q(s^{k-1}, a^k) - \min_{s,a \in Tree} Q(s,a)}{\max_{s,a \in Tree} Q(s,a) - \min_{s,a \in Tree} Q(s,a)} \tag{5}$$

[3] In board games the discount is assumed to be 1 and there are no intermediate rewards.

crackbitswilp.in

Appendix C Hyperparameters

For simplicity we preferentially use the same architectural choices and hyperparameters as in previous work. Specifically, we started with the network architecture and search choices of *AlphaZero* [39]. For board games, we use the same UCB constants, dirichlet exploration noise and the same 800 simulations per search as in *AlphaZero*.

Due to the much smaller branching factor and simpler policies in Atari, we only used 50 simulations per search to speed up experiments. As shown in Figure 3B, the algorithm is not very sensitive to this choice. We also use the same discount (0.997) and value transformation (see Network Architecture section) as R2D2 [21].

For parameter values not mentioned in the text, please refer to the pseudocode.

Appendix D Data Generation

To generate training data, the latest checkpoint of the network (updated every 1000 training steps) is used to play games with MCTS. In the board games Go, chess and shogi the search is run for 800 simulations per move to pick an action; in Atari due to the much smaller action space 50 simulations per move are sufficient.

For board games, games are sent to the training job as soon as they finish. Due to the much larger length of Atari games (up to 30 minutes or 108,000 frames), intermediate sequences are sent every 200 moves. In board games, the training job keeps an in-memory replay buffer of the most recent 1 million games received; in Atari, where the visual observations are larger, the most recent 125 thousand sequences of length 200 are kept.

During the generation of experience in the board game domains, the same exploration scheme as the one described in *AlphaZero* [39] is used. Using a variation of this scheme, in the Atari domain actions are sampled from the visit count distribution throughout the duration of each game, instead of just the first k moves. Moreover, the visit count distribution is parametrized using a temperature parameter T:

$$p_\alpha = \frac{N(\alpha)^{1/T}}{\sum_b N(b)^{1/T}} \tag{6}$$

T is decayed as a function of the number of training steps of the network. Specifically, for the first 500k training steps a temperature of 1 is used, for the next 250k steps a temperature of 0.5 and for the remaining 250k a temperature of 0.25. This ensures that the action selection becomes greedier as training progresses.

Appendix E Network Input

Representation Function

The history over board states used as input to the representation function for Go, chess and shogi is represented similarly to *AlphaZero* [39]. In Go and shogi we encode the last 8 board states as in *AlphaZero*; in chess we increased the history to the last 100 board states to allow correct prediction of draws.

For Atari, the input of the representation function includes the last 32 RGB frames at resolution 96x96 along with the last 32 actions that led to each of those frames. We encode the historical actions because unlike board games, an action in Atari does not necessarily have a visible effect on the observation. RGB frames are encoded as one plane per color, rescaled to the range $[0, 1]$, for red, green and blue respectively. We perform no other normalization, whitening or other preprocessing of the RGB input. Historical actions are encoded as simple bias planes, scaled as $a/18$ (there are 18 total actions in Atari).

Dynamics Function

The input to the dynamics function is the hidden state produced by the representation function or previous application of the dynamics function, concatenated with a representation of the action for the transition. Actions are encoded spatially in planes of the same resolution as the hidden state. In Atari, this resolution is 6x6 (see description of downsampling in Network Architecture section), in board games this is the same as the board size (19x19 for Go, 8x8 for chess, 9x9 for shogi).

crackbitswilp.in

In Go, a normal action (playing a stone on the board) is encoded as an all zero plane, with a single one in the position of the played stone. A pass is encoded as an all zero plane.

In chess, 8 planes are used to encode the action. The first one-hot plane encodes which position the piece was moved from. The next two planes encode which position the piece was moved to: a one-hot plane to encode the target position, if on the board, and a second binary plane to indicate whether the target was valid (on the board) or not. This is necessary because for simplicity our policy action space enumerates a superset of all possible actions, not all of which are legal, and we use the same action space for policy prediction and to encode the dynamics function input. The remaining five binary planes are used to indicate the type of promotion, if any (queen, knight, bishop, rook, none).

The encoding for shogi is similar, with a total of 11 planes. We use the first 8 planes to indicate where the piece moved from - either a board position (first one-hot plane) or the drop of one of the seven types of prisoner (remaining 7 binary planes). The next two planes are used to encode the target as in chess. The remaining binary plane indicates whether the move was a promotion or not.

In Atari, an action is encoded as a one hot vector which is tiled appropriately into planes.

Appendix F Network Architecture

The prediction function $\mathbf{p}^k, v^k = f_\theta(s^k)$ uses the same architecture as *AlphaZero*: one or two convolutional layers that preserve the resolution but reduce the number of planes, followed by a fully connected layer to the size of the output.

For value and reward prediction in Atari we follow [30] in scaling targets using an invertible transform $h(x) = sign(x)(\sqrt{|x|+1} - 1 + \epsilon x)$, where $\epsilon = 0.001$ in all our experiments. We then apply a transformation ϕ to the scalar reward and value targets in order to obtain equivalent categorical representations. We use a discrete support set of size 601 with one support for every integer between -300 and 300. Under this transformation, each scalar is represented as the linear combination of its two adjacent supports, such that the original value can be recovered by $x = x_{low} * p_{low} + x_{high} * p_{high}$. As an example, a target of 3.7 would be represented as a weight of 0.3 on the support for 3 and a weight of 0.7 on the support for 4. The value and reward outputs of the network are also modeled using a softmax output of size 601. During inference the actual value and rewards are obtained by first computing their expected value under their respective softmax distribution and subsequently by inverting the scaling transformation. Scaling and transformation of the value and reward happens transparently on the network side and is not visible to the rest of the algorithm.

Both the representation and dynamics function use the same architecture as *AlphaZero*, but with 16 instead of 20 residual blocks [15]. We use 3x3 kernels and 256 hidden planes for each convolution.

For Atari, where observations have large spatial resolution, the representation function starts with a sequence of convolutions with stride 2 to reduce the spatial resolution. Specifically, starting with an input observation of resolution 96x96 and 128 planes (32 history frames of 3 color channels each, concatenated with the corresponding 32 actions broadcast to planes), we downsample as follows:

- 1 convolution with stride 2 and 128 output planes, output resolution 48x48.
- 2 residual blocks with 128 planes
- 1 convolution with stride 2 and 256 output planes, output resolution 24x24.
- 3 residual blocks with 256 planes.
- Average pooling with stride 2, output resolution 12x12.
- 3 residual blocks with 256 planes.
- Average pooling with stride 2, output resolution 6x6.

The kernel size is 3x3 for all operations.

For the dynamics function (which always operates at the downsampled resolution of 6x6), the action is first encoded as an image, then stacked with the hidden state of the previous step along the plane dimension.

crackbitswilp.in

Appendix G Training

During training, the *MuZero* network is unrolled for K hypothetical steps and aligned to sequences sampled from the trajectories generated by the MCTS actors. Sequences are selected by sampling a state from any game in the replay buffer, then unrolling for K steps from that state. In Atari, samples are drawn according to prioritized replay [35], with priority $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$, where $p_i = |\nu_i - z_i|$, ν is the search value and z the observed n-step return. To correct for sampling bias introduced by the prioritized sampling, we scale the loss using the importance sampling ratio $w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta$. In all our experiments, we set $\alpha = \beta = 1$. For board games, states are sampled uniformly.

Each observation o_t along the sequence also has a corresponding MCTS policy π_t, estimated value ν_t and environment reward u_t. At each unrolled step k the network has a loss to the value, policy and reward target for that step, summed to produce the total loss for the *MuZero* network (see Equation 1). Note that, in board games without intermediate rewards, we omit the reward prediction loss. For board games, we bootstrap directly to the end of the game, equivalent to predicting the final outcome; for Atari we bootstrap for $n = 10$ steps into the future.

To maintain roughly similar magnitude of gradient across different unroll steps, we scale the gradient in two separate locations:

- We scale the loss of each head by $\frac{1}{K}$, where K is the number of unroll steps. This ensures that the total gradient has similar magnitude irrespective of how many steps we unroll for.
- We also scale the gradient at the start of the dynamics function by $\frac{1}{2}$. This ensures that the total gradient applied to the dynamics function stays constant.

In the experiments reported in this paper, we always unroll for $K = 5$ steps. For a detailed illustration, see Figure 1.

To improve the learning process and bound the activations, we also scale the hidden state to the same range as the action input ($[0, 1]$): $s_{scaled} = \frac{s - \min(s)}{\max(s) - \min(s)}$.

All experiments were run using third generation Google Cloud TPUs [12]. For each board game, we used 16 TPUs for training and 1000 TPUs for selfplay. For each game in Atari, we used 8 TPUs for training and 32 TPUs for selfplay. The much smaller proportion of TPUs used for acting in Atari is due to the smaller number of simulations per move (50 instead of 800) and the smaller size of the dynamics function compared to the representation function.

Appendix H Reanalyze

To improve the sample efficiency of *MuZero* we introduced a second variant of the algorithm, *MuZero Reanalyze*. *MuZero Reanalyze* revisits its past time-steps and re-executes its search using the latest model parameters, potentially resulting in a better quality policy than the original search. This fresh policy is used as the policy target for 80% of updates during *MuZero* training. Furthermore, a target network [25] $\cdot, v^- = f_{\theta^-}(s^0)$, based on recent parameters θ^-, is used to provide a fresher, stable n-step bootstrapped target for the value function, $z_t = u_{t+1} + \gamma u_{t+2} + ... + \gamma^{n-1} u_{t+n} + \gamma^n v^-_{t+n}$. In addition, several other hyperparameters were adjusted – primarily to increase sample reuse and avoid overfitting of the value function. Specifically, 2.0 samples were drawn per state, instead of 0.1; the value target was weighted down to 0.25 compared to weights of 1.0 for policy and reward targets; and the n-step return was reduced to $n = 5$ steps instead of $n = 10$ steps.

Appendix I Evaluation

We evaluated the relative strength of *MuZero* (Figure 2) in board games by measuring the Elo rating of each player. We estimate the probability that player a will defeat player b by a logistic function $p(a \text{ defeats } b) = (1 + 10^{(c_{\text{elo}}(e(b) - e(a)))})^{-1}$, and estimate the ratings $e(\cdot)$ by Bayesian logistic regression, computed by the *BayesElo* program [6] using the standard constant $c_{\text{elo}} = 1/400$.

Elo ratings were computed from the results of a 800 simulations per move tournament between iterations of *MuZero* during training, and also a baseline player: either *Stockfish*, *Elmo* or *AlphaZero* respectively. Baseline

crackbitswilp.in

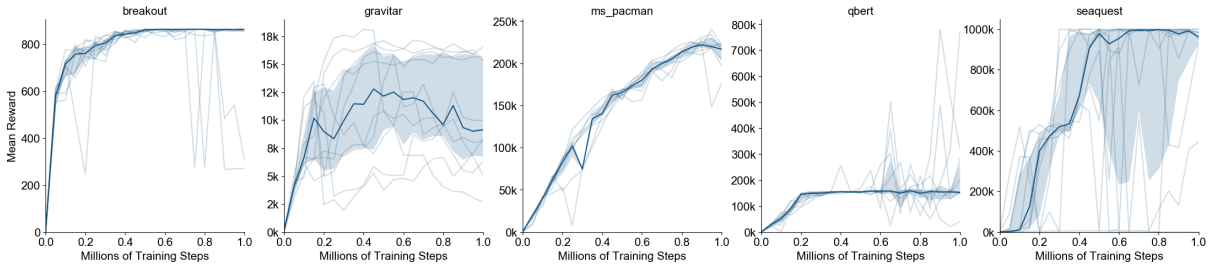


Figure S1: **Repeatability of *MuZero* in Atari for five games.** Total reward is shown on the y-axis, millions of training steps on the x-axis. Dark line indicates median score across 10 separate training runs, light lines indicate individual training runs, and the shaded region indicates 25th to 75th percentile.

players used an equivalent search time of 100ms per move. The Elo rating of the baseline players was anchored to publicly available values [39].

In Atari, we computed mean reward over 1000 episodes per game, limited to the standard 30 minutes or 108,000 frames per episode [27], using 50 simulations per move unless indicated otherwise. In order to mitigate the effects of the deterministic nature of the Atari simulator we employed two different evaluation strategies: 30 noop random starts and human starts. For the former, at the beginning of each episode a random number of between 0 and 30 noop actions are applied to the simulator before handing control to the agent. For the latter, start positions are sampled from human expert play to initialize the Atari simulator before handing the control to the agent [27].

crackbitswilp.in

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 %
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: **Evaluation of *MuZero* in Atari for individual games with 30 random no-op starts.** Best result for each game highlighted in **bold**. Each episode is limited to a maximum of 30 minutes of game time (108k frames). SimPLe was only evaluated on 36 of the 57 games, unavailable results are indicated with '-'. Human normalized score is calculated as $s_{normalized} = \frac{s_{agent} - s_{random}}{s_{human} - s_{random}}$.

crackbitswilp.in

Game	Random	Human	Ape-X [18]	*MuZero*	*MuZero* normalized
alien	128.30	6,371.30	17,732.00	**713,387.37**	11,424.9 %
amidar	11.79	1,540.43	1,047.00	**26,638.80**	1,741.9 %
assault	166.95	628.89	24,405.00	**143,900.58**	31,115.2 %
asterix	164.50	7,536.00	283,180.00	**985,801.95**	13,370.9 %
asteroids	877.10	36,517.30	117,303.00	**606,971.12**	1,700.6 %
atlantis	13,463.00	26,575.00	918,715.00	**1,653,202.50**	12,505.6 %
bank heist	21.70	644.50	**1,201.00**	962.11	151.0 %
battle zone	3,560.00	33,030.00	92,275.00	**791,387.00**	2,673.3 %
beam rider	254.56	14,961.02	72,234.00	**419,460.76**	2,850.5 %
berzerk	196.10	2,237.50	55,599.00	**87,308.60**	4,267.3 %
bowling	35.16	146.46	30.00	**194.03**	142.7 %
boxing	-1.46	9.61	**81.00**	56.60	524.5 %
breakout	1.77	27.86	757.00	**849.59**	3,249.6 %
centipede	1,925.45	10,321.89	5,712.00	**1,138,294.60**	13,533.9 %
chopper command	644.00	8,930.00	576,602.00	**932,370.10**	11,244.6 %
crazy climber	9,337.00	32,667.00	263,954.00	**412,213.90**	1,726.9 %
defender	1,965.50	14,296.00	399,865.00	**823,636.00**	6,663.7 %
demon attack	208.25	3,442.85	133,002.00	**143,858.05**	4,441.0 %
double dunk	-15.97	-14.37	22.00	**23.12**	2,443.1 %
enduro	-81.84	740.17	2,042.00	**2,264.20**	285.4 %
fishing derby	-77.11	5.09	22.00	**57.45**	163.7 %
freeway	0.17	25.61	**29.00**	28.38	110.9 %
frostbite	90.80	4,202.80	6,512.00	**613,944.04**	14,928.3 %
gopher	250.00	2,311.00	121,168.00	**129,218.68**	6,257.6 %
gravitar	245.50	3,116.00	662.00	**3,390.65**	109.6 %
hero	1,580.30	25,839.40	26,345.00	**44,129.55**	175.4 %
ice hockey	-9.67	0.53	24.00	**52.40**	608.5 %
jamesbond	33.50	368.50	18,992.00	**39,107.20**	11,663.8 %
kangaroo	100.00	2,739.00	578.00	**13,210.50**	496.8 %
krull	1,151.90	2,109.10	8,592.00	**257,706.70**	26,802.6 %
kung fu master	304.00	20,786.80	72,068.00	**174,623.60**	851.1 %
montezuma revenge	25.00	**4,182.00**	1,079.00	57.10	0.8 %
ms pacman	197.80	15,375.05	6,135.00	**230,650.24**	1,518.4 %
name this game	1,747.80	6,796.00	23,830.00	**152,723.62**	2,990.7 %
phoenix	1,134.40	6,686.20	188,789.00	**943,255.07**	16,969.6 %
pitfall	-348.80	**5,998.91**	-273.00	-801.10	-7.1 %
pong	-17.95	15.46	19.00	**19.20**	111.2 %
private eye	662.78	**64,169.07**	865.00	5,067.59	6.9 %
qbert	159.38	12,085.00	**380,152.00**	39,302.10	328.2 %
riverraid	588.30	14,382.20	49,983.00	**315,353.33**	2,281.9 %
road runner	200.00	6,878.00	127,112.00	**580,445.00**	8,688.9 %
robotank	2.42	8.94	69.00	**128.80**	1,938.3 %
seaquest	215.50	40,425.80	377,180.00	**997,601.01**	2,480.4 %
skiing	-15,287.35	**-3,686.58**	-11,359.00	-29,400.75	-121.7 %
solaris	2,047.20	**11,032.60**	3,116.00	2,108.08	0.7 %
space invaders	182.55	1,464.90	50,699.00	**57,450.41**	4,465.9 %
star gunner	697.00	9,528.00	432,958.00	**539,342.70**	6,099.5 %
surround	-9.72	5.37	6.00	**8.46**	120.5 %
tennis	-21.43	-6.69	**23.00**	-2.30	129.8 %
time pilot	3,273.00	5,650.00	71,543.00	**405,829.30**	16,935.5 %
tutankham	12.74	138.30	128.00	**351.76**	270.0 %
up n down	707.20	9,896.10	347,912.00	**607,807.85**	6,606.9 %
venture	18.00	**1,039.00**	936.00	21.10	0.3 %
video pinball	0.00	15,641.09	873,989.00	**970,881.10**	6,207.2 %
wizard of wor	804.00	4,556.00	46,897.00	**196,279.20**	5,209.9 %
yars revenge	1,476.88	47,135.17	131,701.00	**888,633.84**	1,943.0 %
zaxxon	475.00	8,443.00	37,672.00	**592,238.70**	7,426.8 %
# best	0	6	5	46	

Table S2: **Evaluation of *MuZero* in Atari for individual games from human start positions.** Best result for each game highlighted in **bold**. Each episode is limited to a maximum of 30 minutes of game time (108k frames).

crackbitswilp.in

Model

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

Search

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

Learning Rule

$$\mathbf{p}_t^k, v_t^k, r_t^k = \mu_\theta(o_1, ..., o_t, a_{t+1}, ..., a_{t+k})$$
$$z_t = \begin{cases} u_T & \text{for games} \\ u_{t+1} + \gamma u_{t+2} + ... + \gamma^{n-1} u_{t+n} + \gamma^n \nu_{t+n} & \text{for general MDPs} \end{cases}$$
$$l_t(\theta) = \sum_{k=0}^{K} l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\pi_{t+k}, p_t^k) + c||\theta||^2$$

Losses

$$l^r(u, r) = \begin{cases} 0 & \text{for games} \\ \boldsymbol{\phi}(u)^T \log \mathbf{r} & \text{for general MDPs} \end{cases}$$
$$l^v(z, q) = \begin{cases} (z - q)^2 & \text{for games} \\ \boldsymbol{\phi}(z)^T \log \mathbf{q} & \text{for general MDPs} \end{cases}$$
$$l^p(\pi, p) = \boldsymbol{\pi}^T \log \mathbf{p}$$

Figure S2: **Equations summarising the *MuZero* algorithm**. Here, $\phi(x)$ refers to the representation of a real number x through a linear combination of its adjacent integers, as described in the Network Architecture section.

crackbitswilp.in

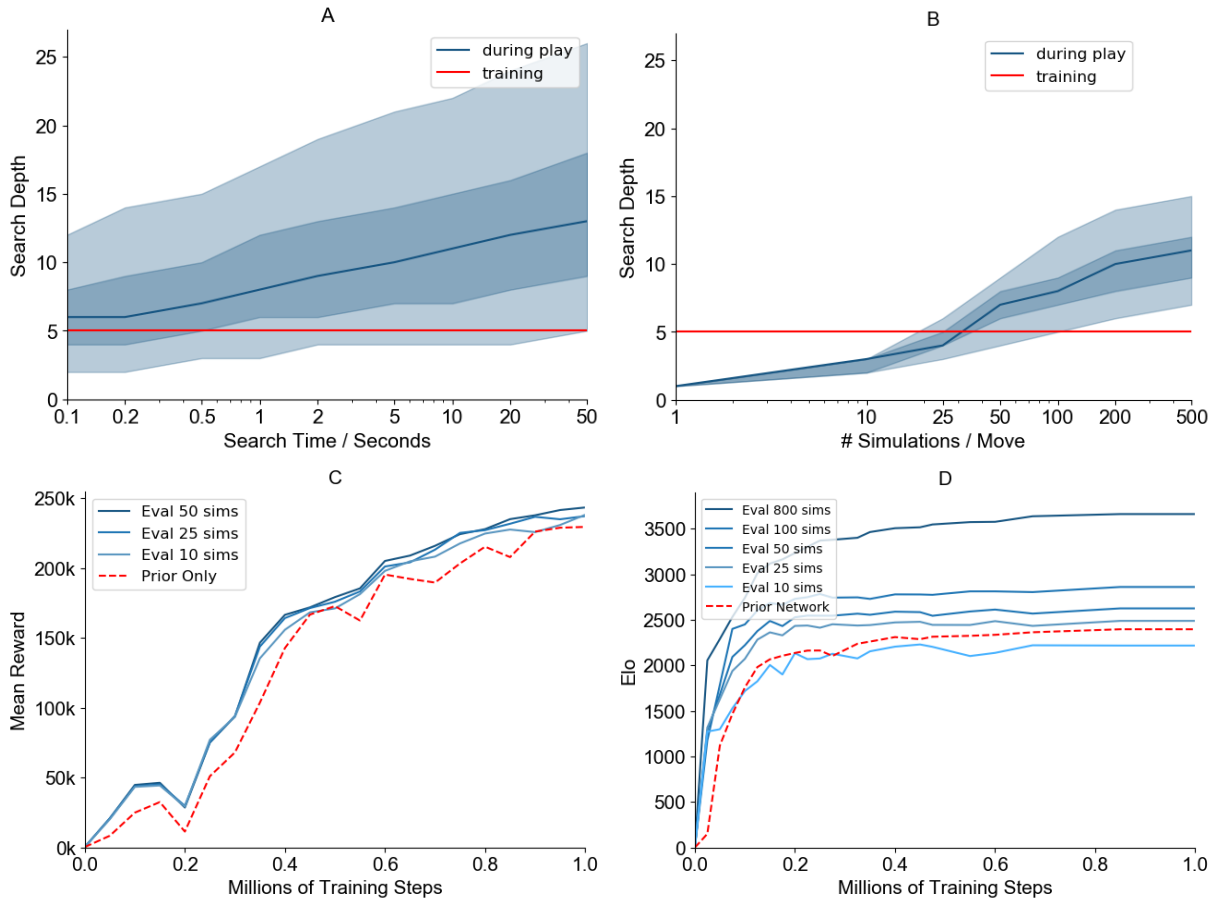


Figure S3: **Details of *MuZero* evaluations (A-B) and policy improvement ablations (C-D)**. (**A-B**) Distribution of evaluation depth in the search tree for the learned model for the evaluations in Figure 3A-B. The network was trained over 5 hypothetical steps, as indicated by the red line. Dark blue line indicates median depth from the root, dark shaded region shows 25th to 75th percentile, light shaded region shows 5th to 95th percentile. (**C**) Policy improvement in Ms. Pacman - a single network was trained at 50 simulations per search and is evaluated at different numbers of simulations per search, including playing according to the argmax of the raw policy network. The policy improvement effect of the search over the raw policy network is clearly visible throughout training. This consistent gap between the performance with and without search highlights the policy improvement that *MuZero* exploits, by continually updating towards the improved policy, to efficiently progress towards the optimal policy. (**D**) Policy improvement in Go - a single network was trained at 800 simulations per search and is evaluated at different numbers of simulations per search. In Go, the playing strength improvement from longer searches is much larger than in Ms. Pacman and persists throughout training, consistent with previous results in [40]. This suggests, as might intuitively be expected, that the benefit of models is greatest in precision planning domains.

crackbitswilp.in



Figure S4: **Learning curves of *MuZero* in Atari for individual games.** Total reward is shown on the y-axis, millions of training steps on the x-axis. Line indicates mean score across 1000 evaluation games, shaded region indicates standard deviation.

crackbitswilp.in