Monte-Carlo tree search as regularized policy optimization

Jean-Bastien Grill, Florent Altché, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, Rémi Munos

▶ **To cite this version:**

Jean-Bastien Grill, Florent Altché, Yunhao Tang, Thomas Hubert, Michal Valko, et al.. Monte-Carlo tree search as regularized policy optimization. International Conference on Machine Learning, 2020, Vienna, Austria. hal-02950136

HAL Id: hal-02950136
https://inria.hal.science/hal-02950136

Submitted on 27 Sep 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Monte-Carlo tree search as regularized policy optimization

Jean-Bastien Grill [*1] **Florent Altché** [*1] **Yunhao Tang** [*1 2] **Thomas Hubert** [3] **Michal Valko** [1]
Ioannis Antonoglou [3] **Rémi Munos** [1]

Abstract

The combination of Monte-Carlo tree search (MCTS) with deep reinforcement learning has led to significant advances in artificial intelligence. However, AlphaZero, the current state-of-the-art MCTS algorithm, still relies on handcrafted heuristics that are only partially understood. In this paper, we show that AlphaZero's search heuristics, along with other common ones such as UCT, are an approximation to the solution of a specific regularized policy optimization problem. With this insight, we propose a variant of AlphaZero which uses the *exact* solution to this policy optimization problem, and show experimentally that it reliably outperforms the original algorithm in multiple domains.

1. Introduction

Policy gradient is at the core of many state-of-the-art deep reinforcement learning (RL) algorithms. Among many successive improvements to the original algorithm (Sutton et al., 2000), *regularized policy optimization* encompasses a large family of such techniques. Among them trust region policy optimization is a prominent example (Schulman et al., 2015; 2017; Abdolmaleki et al., 2018; Song et al., 2019). These algorithmic enhancements have led to significant performance gains in various benchmark domains (Song et al., 2019).

As another successful RL framework, the AlphaZero family of algorithms (Silver et al., 2016; 2017b;a; Schrittwieser et al., 2019) have obtained groundbreaking results on challenging domains by combining classical deep learning (He et al., 2016) and RL (Williams, 1992) techniques with Monte-Carlo tree search (Kocsis and Szepesvári, 2006). To search efficiently, the MCTS action selection criteria takes inspiration from bandits (Auer, 2002). Interestingly, AlphaZero employs an *alternative* handcrafted heuristic to achieve super-human performance on board games (Silver et al., 2016). Recent MCTS-based MuZero (Schrittwieser et al., 2019) has also led to state-of-the-art results in the Atari benchmarks (Bellemare et al., 2013).

Our main contribution is connecting MCTS algorithms, in particular the highly-successful AlphaZero, with MPO, a state-of-the-art model-free policy-optimization algorithm (Abdolmaleki et al., 2018). Specifically, we show that the *empirical visit distribution* of actions in AlphaZero's search procedure approximates the solution of a regularized policy-optimization objective. With this insight, our second contribution a *modified version of AlphaZero* that comes significant performance gains over the original algorithm, especially in cases where AlphaZero has been observed to fail, *e.g.,* when per-search simulation budgets are low (Hamrick et al., 2020).

In Section 2, we briefly present MCTS with a focus on AlphaZero and provide a short summary of the model-free policy-optimization. In Section 3, we show that AlphaZero (and many other MCTS algorithms) computes approximate solutions to a family of regularized policy optimization problems. With this insight, Section 4 introduces a modified version of AlphaZero which leverages the benefits of the policy optimization formalism to improve upon the original algorithm. Finally, Section 5 shows that this modified algorithm outperforms AlphaZero on Atari games and continuous control tasks.

2. Background

Consider a standard RL setting tied to a Markov decision process (MDP) with state space $\mathcal{X}$ and action space $\mathcal{A}$. At a discrete round $t \geq 0$, the agent in state $x_t \in \mathcal{X}$ takes action $a_t \in \mathcal{A}$ given a policy $a_t \sim \pi(\cdot|s_t)$, receives reward r_t, and transitions to a next state $x_{t+1} \sim p(\cdot|x_t, a_t)$. The RL problem consists in finding a policy which maximizes the discounted cumulative return $\mathbb{E}_\pi[\sum_{t\geq 0} \gamma^t r_t]$ for a discount factor $\gamma \in (0, 1)$. To scale the method to large environments, we assume that the policy $\pi_\theta(a|x)$ is parameterized by a neural network θ.

[*]Equal contribution [1]DeepMind, Paris, FR [2]Columbia University, New York, USA [3]DeepMind, London, UK. Correspondence to: Jean-Bastien Grill <jbgrill@google.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

2.1. AlphaZero

We focus on the AlphaZero family, comprised of AlphaGo (Silver et al., 2016), AlphaGo Zero (Silver et al., 2017b), AlphaZero (Silver et al., 2017a), and MuZero (Schrittwieser et al., 2019), which are among the most successful algorithms in combining model-free and model-based RL. Although they make different assumptions, all of these methods share the same underlying search algorithm, which we refer to as *AlphaZero* for simplicity.

From a state x, AlphaZero uses MCTS (Browne et al., 2012) to compute an improved policy $\hat{\pi}(\cdot|x)$ at the root of the search tree from the prior distribution predicted by a policy network $\pi_\theta(\cdot|x)$[1]; see Eq. 3 for the definition. This improved policy is then distilled back into π_θ by updating θ as $\theta \leftarrow \theta - \eta\nabla_\theta \mathbb{E}_x[D(\hat{\pi}(\cdot|x), \pi_\theta(\cdot|x))]$ for a certain divergence D. In turn, the distilled parameterized policy π_θ informs the next local search by predicting priors, further improving the local policy over successive iterations. Therefore, such an algorithmic procedure is a special case of generalized policy improvement (Sutton and Barto, 1998).

One of the main differences between AlphaZero and previous MCTS algorithms such as UCT (Kocsis and Szepesvári, 2006) is the introduction of a learned prior π_θ and value function v_θ. Additionally, AlphaZero's search procedure applies the following action selection heuristic,

$$\arg\max_a \left[Q(x,a) + c \cdot \pi_\theta(a|x) \cdot \frac{\sqrt{\sum_b n(x,b)}}{1+n(x,a)} \right], \quad (1)$$

where c is a numerical constant,[2] $n(x,a)$ is the number of times that action a has been selected from state x during search, and $Q(x,a)$ is an estimate of the Q-function for state-action pair (x,a) computed from search statistics and using v_θ for bootstrapping.

Intuitively, this selection criteria balances exploration and exploitation, by selecting the most promising actions (high Q-value $Q(x,a)$ and prior policy $\pi_\theta(a|x)$) or actions that have rarely been explored (small visit count $n(x,a)$). We denote by N_{sim} the simulation budget, *i.e.,* the search is run with N_{sim} simulations. A more detailed presentation of AlphaZero is in Appendix A; for a full description of the algorithm, refer to Silver et al. (2017a).

2.2. Policy optimization

Policy optimization aims at finding a globally optimal policy π_θ, generally using iterative updates. Each iteration updates the current policy π_θ by solving a local maximization problem of the form

$$\pi_{\theta'} \triangleq \arg\max_{\mathbf{y}\in\mathcal{S}} \; \mathcal{Q}_{\pi_\theta}^\intercal \mathbf{y} - \mathcal{R}(\mathbf{y}, \pi_\theta), \quad (2)$$

where $\mathcal{Q}_{\pi_\theta}$ is an estimate of the Q-function, $\mathcal{S}$ is the $|\mathcal{A}|$-dimensional simplex and $\mathcal{R} : \mathcal{S}^2 \to \mathbb{R}$ a convex regularization term (Neu et al., 2017; Grill et al., 2019; Geist et al., 2019). Intuitively, Eq. 2 updates π_θ to maximize the value $\mathcal{Q}_{\pi_\theta}^\intercal \mathbf{y}$ while constraining the update with a regularization term $\mathcal{R}(\mathbf{y}, \pi_\theta)$.

Without regularizations, *i.e.,* $\mathcal{R} = 0$, Eq. 2 reduces to policy iteration (Sutton and Barto, 1998). When π_θ is updated using a single gradient ascent step towards the solution of Eq. 2, instead of using the solution directly, the above formulation reduces to (regularized) policy gradient (Sutton et al., 2000; Levine, 2018).

Interestingly, the regularization term has been found to stabilize, and possibly to speed up the convergence of π_θ. For instance, trust region policy search algorithms (TRPO, Schulman et al., 2015; MPO Abdolmaleki et al., 2018; V-MPO, Song et al., 2019), set $\mathcal{R}$ to be the KL-divergence between consecutive policies $\text{KL}[\mathbf{y}, \pi_\theta]$; maximum entropy RL (Ziebart, 2010; Fox et al., 2015; O'Donoghue et al., 2016; Haarnoja et al., 2017) sets $\mathcal{R}$ to be the negative entropy of $\mathbf{y}$ to avoid collapsing to a deterministic policy.

3. MCTS as regularized policy optimization

In Section 2, we presented AlphaZero that relies on model-*based* planning. We also presented policy optimization, a framework that has achieved good performance in model-*free* RL. In this section, we establish our main claim namely that AlphaZero's action selection criteria can be interpreted as approximating the solution to a regularized policy-optimization objective.

3.1. Notation

First, let us define the *empirical visit distribution* $\hat{\pi}$ as

$$\hat{\pi}(a|x) \triangleq \frac{1 + n(x,a)}{|\mathcal{A}| + \sum_b n(x,b)}. \quad (3)$$

Note that in Eq. 3, we consider an extra visit per action compared to the acting policy and distillation target in the original definition (Silver et al., 2016). This extra visit is introduced for convenience in the upcoming analysis (to avoid divisions by zero) and does not change the generality of our results.

We also define the *multiplier* λ_N as

$$\lambda_N(x) \triangleq c \cdot \frac{\sqrt{\sum_b n_b}}{|\mathcal{A}| + \sum_b n_b}, \quad (4)$$

[1] We note here that terminologies such as *prior* follow Silver et al. (2017a) and do not relate to concepts in Bayesian statistics.

[2] Schrittwieser et al. (2019) uses a c that has a slow-varying dependency on $\sum_b n(x,b)$, which we omit here for simplicity, as it was the case of Silver et al. (2017a).

where the shorthand notation n_a is used for $n(x,a)$, and $N(x) \triangleq \sum_b n_b$ denotes the number of visits to x during search. With this notation, the action selection formula of Eq. 1 can be written as selecting the action $a^\star$ such that

$$a^\star(x) \triangleq \arg\max_a \left[Q(x,a) + \lambda_N \cdot \frac{\pi_\theta(a|x)}{\hat{\pi}(a|x)} \right]. \quad (5)$$

Note that in Eq. 5 and in the rest of the paper (unless otherwise specified), we use Q to denote the *search* Q-values, *i.e.,* those estimated by the search algorithm as presented in Section 2.1. For more compact notation, we use bold fonts to denote vector quantities, with the convention that $\frac{\mathbf{u}}{\mathbf{v}}[a] = \frac{\mathbf{u}[a]}{\mathbf{v}[a]}$ for two vectors $\mathbf{u}$ and $\mathbf{v}$ with the same dimension. Additionally, we omit the dependency of quantities on state x when the context is clear. In particular, we use $\mathbf{q} \in \mathbb{R}^{|\mathcal{A}|}$ to denote the vector of search Q-function $Q(x,a)$ such that $\mathbf{q}_a = Q(x,a)$. With this notation, we can rewrite the action selection formula of Eq. 5 simply as[3]

$$a^\star \triangleq \arg\max \left[\mathbf{q} + \lambda_N \frac{\boldsymbol{\pi_\theta}}{\hat{\boldsymbol{\pi}}} \right]. \quad (6)$$

3.2. A related regularized policy optimization problem

We now define $\bar{\pi}$ as the solution to a regularized policy optimization problem; we will see in the next subsection that the visit distribution $\hat{\pi}$ is a good approximation of $\bar{\pi}$.

Definition 1 ($\bar{\boldsymbol{\pi}}$). *Let $\bar{\boldsymbol{\pi}}$ be the solution to the following objective*

$$\bar{\boldsymbol{\pi}} \triangleq \arg\max_{\mathbf{y} \in \mathcal{S}} [\mathbf{q}^\intercal \mathbf{y} - \lambda_N \mathrm{KL}[\boldsymbol{\pi_\theta}, \mathbf{y}]], \quad (7)$$

where $\mathcal{S}$ is the $|\mathcal{A}|-$dimensional simplex and KL *is the KL-divergence.*[4]

We can see from Eq. 2 and Definition 1 that $\bar{\boldsymbol{\pi}}$ is the solution to a policy optimization problem where $\mathcal{Q}$ is set to the search Q-values, and the regularization term $\mathcal{R}$ is a *reversed* KL-divergence weighted by factor λ_N.

In addition, note that $\bar{\pi}$ is as a smooth version of the $\arg\max$ associated to the search Q-values $\mathbf{q}$. In fact, $\bar{\pi}$ can be computed as (Appendix B.3 gives a detailed derivation of $\bar{\pi}$)

$$\bar{\boldsymbol{\pi}} = \lambda_N \frac{\boldsymbol{\pi_\theta}}{\alpha - \mathbf{q}}, \quad (8)$$

where $\alpha \in \mathbb{R}$ is such that $\bar{\pi}$ is a proper probability vector. This is slightly different from the softmax distribution obtained with $\mathrm{KL}[\mathbf{y}, \pi_\theta]$, which is written as

$$\arg\max_{\mathbf{y} \in \mathcal{S}} [\mathbf{q}^\intercal \mathbf{y} - \lambda_N \mathrm{KL}[\mathbf{y}, \boldsymbol{\pi_\theta}]] \propto \pi_\theta \exp\left(\frac{\mathbf{q}}{\lambda_N}\right).$$

[3] When the context is clear, we simplify for any $\mathbf{x} \in \mathbb{R}^{|\mathcal{A}|}$, that $\arg\max[\mathbf{x}] \triangleq \arg\max_a \{\mathbf{x}[a], a \in \mathcal{A}\}$.

[4] We apply the definition $\mathrm{KL}[\mathbf{x}, \mathbf{y}] \triangleq \sum_a \mathbf{x}[a] \log \frac{\mathbf{x}[a]}{\mathbf{y}[a]}$.

Remark The factor λ_N is a decreasing function of N. Asymptotically, $\lambda_N = \tilde{O}(1/\sqrt{N})$. Therefore, the influence of the regularization term decreases as the number of simulation increases, which makes $\bar{\pi}$ rely increasingly more on search Q-values $\mathbf{q}$ and less on the policy prior π_θ. As we explain next, λ_N follows the design choice of AlphaZero, and may be justified by a similar choice done in bandits (Bubeck et al., 2012).

3.3. AlphaZero as policy optimization

We now analyze the action selection formula of AlphaZero (Eq. 1). Interestingly, we show that this formula, which was *handcrafted*[5] independently of the policy optimization research, turns out to result in a distribution $\hat{\pi}$ that closely relates to the policy optimization solution $\bar{\pi}$.

The main formal claim of this section that AlphaZero's search policy $\hat{\pi}$ *tracks* the exact solution $\bar{\pi}$ of the regularized policy optimization problem of Definition 1. We show that Proposition 1 and Proposition 2 support this claim from two complementary perspectives.

First, with Proposition 1, we show that $\hat{\pi}$ approximately follows the gradient of the concave objective for which $\bar{\pi}$ is the optimum.

Proposition 1. *For any action $a \in \mathcal{A}$, visit count $n \in \mathbb{R}^{\mathcal{A}}$, policy prior $\pi_\theta > 0$ and Q-values $\mathbf{q}$,*

$$a^\star = \arg\max_a \left[\frac{\partial}{\partial n_a} (\mathbf{q}^\intercal \hat{\boldsymbol{\pi}} - \lambda_N \mathrm{KL}[\boldsymbol{\pi_\theta}, \hat{\boldsymbol{\pi}}]) \right], \quad (9)$$

with $a^\star$ being the action realizing Eq. 1 as defined in Eq. 5 and $\hat{\pi} = (1+\mathbf{n})/(|\mathcal{A}| + \sum_b n_b)$ as defined in Eq. 3, is a function of the count vector extended to real values.

The only thing that the search algorithm eventually influences through the tree search is the visit count distribution. If we could do an infinitesimally small update, then the greedy update maximizing Eq. 8 would be in the direction of the partial derivative of Eq. 9. However, as we are *restricted by a discrete update*, then increasing the visit count as in Proposition 1 makes $\hat{\pi}$ track $\bar{\pi}$. Below, we further characterize the selected action $a^\star$ and assume $\pi_\theta > 0$.

Proposition 2. *The action $a^\star$ realizing Eq. 1 is such that*

$$\hat{\pi}(a^\star|x) \leq \bar{\pi}(a^\star|x). \quad (10)$$

To acquire intuition from Proposition 2, note that once $a^\star$ is selected, its count $n_{a^\star}$ increases and so does the total count N. As a result, $\hat{\pi}(a^\star)$ increases (in the order of $\mathcal{O}(1/N)$) and further approximates $\bar{\pi}(a^\star)$. As such, Proposition 2 shows that the action selection formula encourages

[5] Nonetheless, this heuristic could be interpreted as loosely inspired by bandits (Rosin, 2011), but was adapted to accommodate a prior term π_θ.

the shape of $\hat{\pi}$ to be close to that of $\bar{\pi}$, until in the limit the two distributions coincide.

Note that Proposition 1 and Proposition 2 are a special case of a more general result that we formally prove in Appendix D.1. In this particular case, the proof relies on noticing that

$$\arg\max_a \left[\mathbf{q}_a + c \cdot \pi_\theta(a) \cdot \frac{\sqrt{\sum_b \boldsymbol{n}_b}}{1+\boldsymbol{n}_a}\right] \tag{1}$$

$$= \arg\max_a \left[\pi_\theta(a) \cdot \left(\frac{1}{\hat{\pi}(a)} - \frac{1}{\bar{\pi}(a)}\right)\right]. \tag{11}$$

Then, since $\sum_a \hat{\pi}(a) = \sum_a \bar{\pi}(a)$ and $\hat{\pi} > 0$ and $\bar{\pi} > 0$, there exists at least one action for which $0 < \hat{\pi}(a) \leq \bar{\pi}(a)$, *i.e.,* $1/\hat{\pi}(a) - 1/\bar{\pi}(a) \geq 0$.

To state a formal statement on $\hat{\pi}$ approximating $\bar{\pi}$, in Appendix D.3 we expand the conclusion under the assumption that $\bar{\pi}$ is a constant. In this case we can derive a bound for the convergence rate of these two distributions as N increases over the search,

$$||\bar{\boldsymbol{\pi}} - \hat{\boldsymbol{\pi}}||_\infty \leq \frac{|\mathcal{A}| - 1}{|\mathcal{A}| + N}, \tag{12}$$

with $\mathcal{O}(1/N)$ matching the lowest possible approximation error (see Appendix D.3) among discrete distributions of the form $(k_i/N)_i$ for $k_i \in \mathbb{N}$.

3.4. Generalization to common MCTS algorithms

Besides AlphaZero, UCT (Kocsis and Szepesvári, 2006) is another heuristic with a selection criteria inspired by UCB, defined as

$$\arg\max_a \left[\mathbf{q}_a + c \cdot \sqrt{\frac{\log(\sum_b \boldsymbol{n}_b)}{1+\boldsymbol{n}_a}}\right]. \tag{13}$$

Contrary to AlphaZero, the standard UCT formula does not involve a prior policy. In this section, we consider a slightly modified version of UCT with a (learned) prior π_θ, as defined in Eq. 14. By setting the prior π_θ to the uniform distribution, we recover the original UCT formula,

$$\arg\max_a \left[\mathbf{q}_a + c \cdot \sqrt{\pi_\theta(a) \cdot \frac{\log(\sum_b \boldsymbol{n}_b)}{1+\boldsymbol{n}_a}}\right]. \tag{14}$$

Using the same reasoning as in Section 3.3, we now show that this modified UCT formula also tracks the solution to a regularized policy optimization problem, thus generalizing our result to commonly used MCTS algorithms.

First, we introduce $\bar{\pi}_{\text{UCT}}$, which is tracked by the UCT visit distribution, as:

$$\bar{\pi}_{\text{UCT}} \triangleq \arg\max_{\mathbf{y} \in \mathcal{S}} \mathbf{q}^T \mathbf{y} - \lambda_N^{\text{UCT}} D(\mathbf{y}, \boldsymbol{\pi}_{\boldsymbol{\theta}}), \tag{15}$$

where $D(\boldsymbol{x}, \boldsymbol{y}) \triangleq 2 - 2\sum_i \sqrt{\boldsymbol{x}_i \cdot \boldsymbol{y}_i}$ is an f-divergence[6]

$$\text{and} \quad \lambda_N^{\text{UCT}}(x) \triangleq c \cdot \sqrt{\frac{\log \sum_b \boldsymbol{n}_b}{|A| + \sum_b \boldsymbol{n}_b}}.$$

Similar to AlphaZero, λ_N^{UCT} behaves[7] as $\tilde{O}\left(1/\sqrt{N}\right)$ and therefore the regularization gets weaker as N increases. We can also derive tracking properties between $\bar{\pi}_{\text{UCT}}$ and the UCT empirical visit distribution $\hat{\pi}_{\text{UCT}}$ as we did for AlphaZero in the previous section, with Proposition 3; as in the previous section, this is a special case of the general result with any f-divergence in Appendix D.1.

Proposition 3. *We have that*

$$\arg\max_a \left[\mathbf{q}_a + c \cdot \sqrt{\pi_\theta(a) \cdot \frac{\log(\sum_b \boldsymbol{n}_b)}{1+\boldsymbol{n}_a}}\right]$$

$$= \arg\max_a \left[\sqrt{\pi_\theta(a)} \cdot \left(\frac{1}{\sqrt{\hat{\pi}(a)}} - \frac{1}{\sqrt{\bar{\pi}_{UCT}(a)}}\right)\right]$$

and

$$a^\star_{UCT} = \arg\max_a \left[\frac{\partial}{\partial \boldsymbol{n}_a}\left(\mathbf{q}^\intercal \hat{\boldsymbol{\pi}} - \lambda_N^{UCT} D[\boldsymbol{\pi}_{\boldsymbol{\theta}}, \hat{\boldsymbol{\pi}}]\right)\right]. \tag{16}$$

To sum up, similar to the previous section, we show that UCT's search policy $\hat{\pi}_{\text{UCT}}$ tracks the exact solution $\bar{\pi}_{\text{UCT}}$ of the regularized policy optimization problem of Eq. 15.

4. Algorithmic benefits

In Section 3, we introduced a distribution $\bar{\pi}$ as the solution to a regularized policy optimization problem. We then showed that AlphaZero, along with general MCTS algorithms, select actions such that the empirical visit distribution $\hat{\pi}$ actively approximates $\bar{\pi}$. Building on this insight, below we argue that $\bar{\pi}$ is preferable to $\hat{\pi}$, and we propose three complementary algorithmic changes to AlphaZero.

4.1. Advantages of using $\bar{\pi}$ over $\hat{\pi}$

MCTS algorithms produce Q-values as a by-product of the search procedure. However, MCTS does not directly use search Q-values to compute the policy, but instead uses the visit distribution $\hat{\pi}$ (search Q-values implicitly influence $\hat{\pi}$ by guiding the search). We postulate that this degrades the performance especially at low simulation budgets N_{sim} for several reasons:

[6] In particular $D(x,y) \geq 0, D(x,y) = 0 \implies x = y$ and $D(x,y)$ is jointly convex in x and y (Csiszár, 1964; Liese and Vajda, 2006).

[7] We ignore logarithmic terms.

1. When a promising new (high-value) leaf is discovered, many additional simulations might be needed before this information is reflected in $\hat{\pi}$; since $\bar{\pi}$ is directly computed from Q-values, this information is updated instantly.
2. By definition (Eq. 3), $\hat{\pi}$ is the ratio of two integers and has limited expressiveness when N_{sim} is low, which might lead to a sub-optimal policy; $\bar{\pi}$ does not have this constraint.
3. The prior π_θ is trained against the target $\hat{\pi}$, but the latter is only improved for actions that have been sampled at least once during search. Due to the deterministic action selection (Eq. 1), this may be problematic for certain actions that would require a large simulation budget to be sampled even once.

The above downsides cause MCTS to be highly sensitive to simulation budgets N_{sim}. When N_{sim} is high relative to the branching factor of the tree search, i.e., number of actions, MCTS algorithms such as AlphaZero perform well. However, this performance drops significantly when N_{sim} is low as showed by Hamrick et al. (2020); see also *e.g.*, Figure 3.D. by Schrittwieser et al. (2019).

We illustrate the effect of simulation budgets in Figure 1, where x-axis shows the budgets N_{sim} and y-axis shows the episodic performance of algorithms applying $\hat{\pi}$ vs. $\bar{\pi}$; see the details of these algorithms in the following sections. We see that $\hat{\pi}$ is highly sensitive to simulation budgets while $\bar{\pi}$ performs consistently well across all budget values.

4.2. Proposed improvements to AlphaZero

We have pointed out potential issues due to $\hat{\pi}$. We now detail how to use $\bar{\pi}$ as a replacement to resolve such issues.[8] Appendix B.3 shows how to compute $\bar{\pi}$ in practice.

ACT: acting with $\bar{\pi}$ AlphaZero acts in the real environment by sampling actions according to $a \sim \hat{\pi}(\cdot|x_{\text{root}})$. Instead, we propose to to sample actions sampling according to $a \sim \bar{\pi}(\cdot|x_{\text{root}})$. We label this variant as ACT.

SEARCH: searching with $\bar{\pi}$ During search, we propose to stochastically sample actions according to $\bar{\pi}$ instead of the deterministic action selection rule of Eq. 1. At each node x in the tree, $\bar{\pi}(\cdot)$ is computed with Q-values and total visit counts at the node based on Definition 1. We label this variant as SEARCH.

LEARN: learning with $\bar{\pi}$ AlphaZero computes locally improved policy with tree search and distills such improved policy into π_θ. We propose to use $\bar{\pi}$ as the target policy in place of $\hat{\pi}$ to train our prior policy. As a result, the parameters are updated as

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathbb{E}_{x_{\text{root}}} \Big[\mathbb{KL}[\bar{\pi}(\cdot|x_{\text{root}}), \pi_\theta(\cdot|x_{\text{root}})] \Big], \quad (17)$$

where x_{root} is sampled from a prioritized replay buffer as in AlphaZero. We label this variant as LEARN.

ALL: combining them all We refer to the combination of these three independent variants as ALL. Appendix B provides additional implementation details.

Remark Note that AlphaZero entangles search and learning, which is not desirable. For example, when the action selection formula changes, this impacts not only intermediate search results but also the root visit distribution $\hat{\pi}(\cdot|x_{\text{root}})$, which is also the learning target for π_θ. However, the LEARN variant partially disentangles these components. Indeed, the new learning target is $\bar{\pi}(\cdot|x_{\text{root}})$ which is computed from search Q-values, rendering it less sensitive to e.g., the action selection formula.

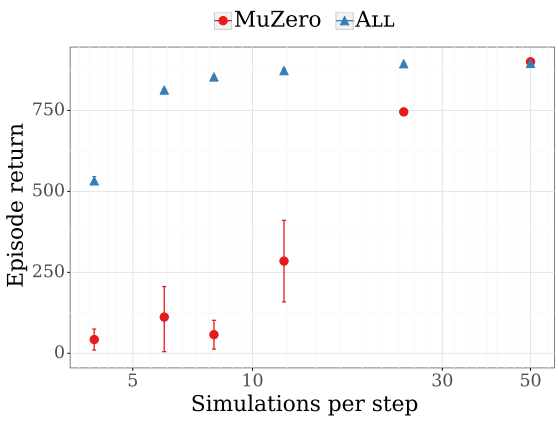


Figure 1. Comparison of the score (median score over 3 seeds) of MuZero (red: using $\hat{\pi}$) and ALL (blue: using $\bar{\pi}$) after 100k learner steps as a function of N_{sim} on Cheetah Run of the Control Suite.

4.3. Connections between AlphaZero and model-free policy optimization.

Next, we make the explicit link between proposed algorithmic variants and existing policy optimization algorithms. First, we provide two complementary interpretations.

LEARN as policy optimization For this interpretation, we treat SEARCH as a blackbox, *i.e.,* a subroutine that takes a root node x and returns statistics such as search Q-values.

Recall that policy optimization (Eq. 2) maximizes the objective $\approx \mathcal{Q}_{\pi_\theta}^{\intercal}\mathbf{y}$ with the local policy $\mathbf{y}$. There are many model-

[8] Recall that we have identified three issues. Each algorithmic variant below helps in addressing issue 1 and 2. Furthermore, the LEARN variant helps address issue 3.

free methods for the estimation of $\mathcal{Q}_{\pi_\theta}$, ranging from Monte-Carlo estimates of cumulative returns $\mathcal{Q}^{\pi_\theta} \approx \sum_{t\geq 0} \gamma^t r_t$ (Schulman et al., 2015; 2017) to using predictions from a Q-value critic $\mathcal{Q}_{\pi_\theta} \approx \mathbf{q}_\theta$ trained with off-policy samples (Abdolmaleki et al., 2018; Song et al., 2019). When solving $\bar{\pi}$ for the update (Eq. 17), we can interpret LEARN as a policy optimization algorithm using tree search to estimate $\mathcal{Q}_{\pi_\theta}$. Indeed, LEARN could be interpreted as building a Q-function[9] critic $\mathbf{q}_\theta$ with a tree-structured inductive bias. However, this inductive bias is not built-in a network architecture (Silver et al., 2017c; Farquhar et al., 2017; Oh et al., 2017; Guez et al., 2018), but constructed online by an algorithm, i.e., MCTS. Next, LEARN computes the locally optimal policy $\bar{\pi}$ to the regularized policy optimization objective and distills $\bar{\pi}$ into π_θ. This is exactly the approach taken by MPO (Abdolmaleki et al., 2018).

SEARCH as policy optimization We now unpack the algorithmic procedure of the tree search, and show that it can also be interpreted as policy optimization.

During the forward simulation phase of SEARCH, the action at each node x is selected by sampling $a \sim \bar{\pi}(\cdot|x)$. As a result, the full imaginary trajectory is generated consistently according to policy $\bar{\pi}$. During backward updates, each encountered node x receives a backup value from its child node, which is an exact estimate of $Q^{\bar{\pi}}(x, a)$. Finally, the local policy $\bar{\pi}(\cdot|x)$ is updated by solving the constrained optimization problem of Definition 1, leading to an improved policy over previous $\bar{\pi}(\cdot|x)$. Overall, with N_{sim} simulated trajectories, SEARCH optimizes the root policy $\bar{\pi}(\cdot|x_{\text{root}})$ and root search Q-values, by carrying out N_{sim} sequences of MPO-style updates across the entire tree.[10] A highly related approach is to update local policies via policy gradients (Anthony et al., 2019).

By combining the above two interpretations, we see that the ALL variant is very similar to a full policy optimization algorithm. Specifically, on a high level, ALL carries out MPO updates with search Q-values. These search Q-values are also themselves obtained via MPO-style updates within the tree search. This paves the way to our major revelation stated next.

Observation 1. ALL *can be interpreted as regularized policy optimization. Further, since $\hat{\pi}$ approximates $\bar{\pi}$, AlphaZero and other MCTS algorithms can be interpreted as approximate regularized policy optimization.*

[9]During search, because child nodes have fewer simulations than the root, the Q-function estimate at the root slightly underestimates the acting policy Q-function.

[10]Note that there are several differences from typical model-free implementations of policy optimization: most notably, unlike a fully-parameterized policy, the tree search policy is tabular at each node. This also entails that the MPO-style distillation is exact.

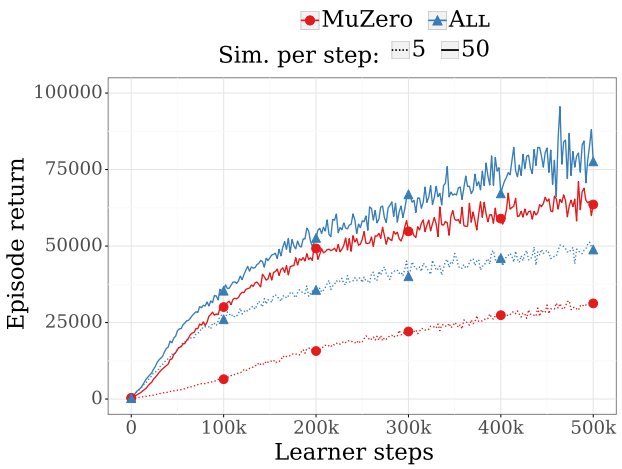


Figure 2. Comparison of median scores of MuZero (red) and ALL (blue) at $N_{\text{sim}} = 5$ (dotted line) and $N_{\text{sim}} = 50$ (solid line) simulations per step on Ms Pacman (Atari). Averaged across 8 seeds.

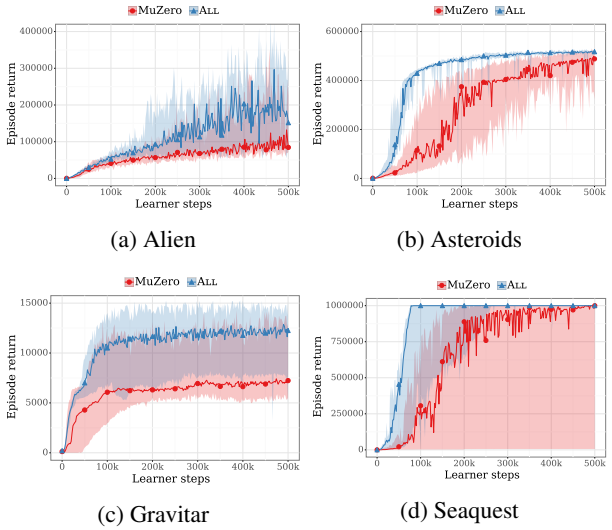
5. Experiments

In this section, we aim to address several questions: **(1)** How sensitive are state-of-the-art hybrid algorithms such as AlphaZero to low simulation budgets and can the ALL variant provide a more robust alternative? **(2)** What changes among ACT, SEARCH, and LEARN are most critical in this variant performance? **(3)** How does the performance of the ALL variant compare with AlphaZero in environments with large branching factors?

Baseline algorithm Throughout the experiments, we take MuZero (Schrittwieser et al., 2019) as the baseline algorithm. As a variant of AlphaZero, MuZero applies tree search in learned models instead of real environments, which makes it applicable to a wider range of problems. Since MuZero shares the same search procedure as AlphaGo, AlphaGo Zero, and AlphaZero, we expect the performance gains to be transferable to these algorithms. Note that the results below were obtained with a scaled-down version of MuZero, which is described in Appendix B.1.

Hyper-parameters The hyper-parameters of the algorithms are tuned to achieve the maximum possible performance for baseline MuZero on the Ms Pacman level of the Atari suite (Bellemare et al., 2013), and are identical in all experiments with the exception of the number of simulations per step N_{sim}.[11] In particular, no further tuning was required for the LEARN, SEARCH, ACT, and ALL variants, as was expected from the theoretical considerations of Section 3.

[11]The number of actors is scaled linearly with N_{sim} to maintain the same total number of generated frames per second.



(a) Alien (b) Asteroids

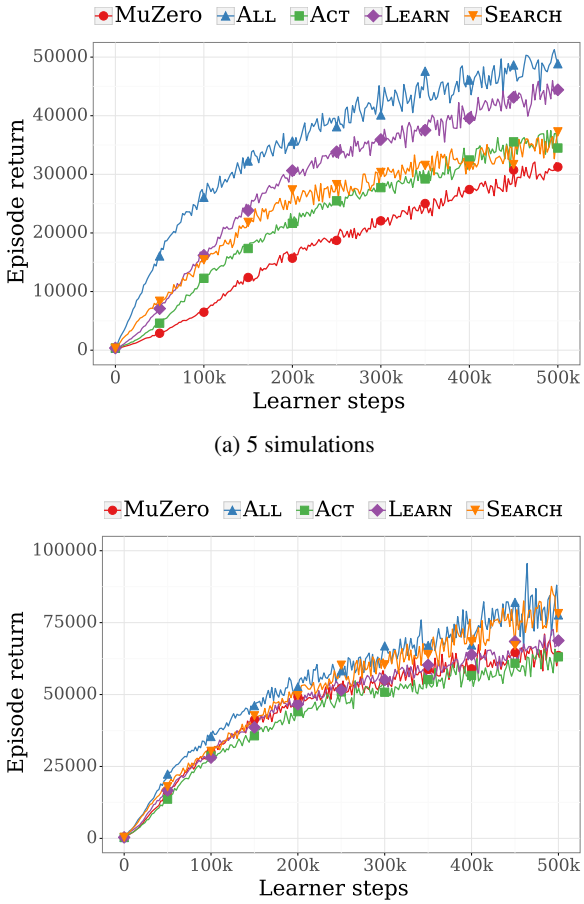
(c) Gravitar (d) Seaquest

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

5.1. Search with low simulation budgets

Since AlphaZero solely relies on the $\hat{\pi}$ for training targets, it may misbehave when simulation budgets N_{sim} are low. On the other hand, our new algorithmic variants might perform better in this regime. To confirm these hypotheses, we compare the performance of MuZero and the ALL variant on the Ms Pacman level of the Atari suite at different levels of simulation budgets.

Result In Figure 2, we compare the episodic return of ALL vs. MuZero averaged over 8 seeds, with a simulation budget $N_{\text{sim}} = 5$ and $N_{\text{sim}} = 50$ for an action set of size $|\mathcal{A}| \leq 18$; thus, we consider that $N_{\text{sim}} = 5$ and $N_{\text{sim}} = 50$ respectively correspond to a low and high simulation budgets relative to the number of actions. We make several observations: **(1)** At a relatively high simulation budget, $N_{\text{sim}} = 50$, same as Schrittwieser et al. (2019), both MuZero and ALL exhibit reasonably close levels of performance; though ALL obtains marginally better performance than MuZero; **(2)** At low simulation budget, $N_{\text{sim}} = 5$, though both algorithms suffer in performance relative to high budgets, ALL significantly outperforms MuZero both in terms of learning speed and asymptotic performance; **(3)** Figure 6 in Appendix C.1 shows that this behavior is consistently observed at intermediate simulation budgets, with the two algorithms starting to reach comparable levels of performance when $N_{\text{sim}} \geq 24$ simulations. These observations confirm the intuitions from Section 3. **(4)** We provide results on a subset of Atari games in Figure 3, which show that the performance gains due to $\bar{\pi}$ over $\hat{\pi}$ are also observed in other levels than Ms Pacman; see



(a) 5 simulations

(b) 50 simulations

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

Appendix C.2 for results on additional levels. This subset of levels are selected based on the experiment setup in Figure S1 of Schrittwieser et al. (2019). Importantly, note that the performance gains of ALL are consistently significant across selected levels, even at a higher simulation budget of $N_{\text{sim}} = 50$.

5.2. Ablation study

To better understand which component of the ALL contributes the most to the performance gains, Figure 4 presents the results of an ablation study where we compare individual component LEARN, SEARCH, or ACT.

Result The comparison is shown in Figure 4, we make several observations: **(1)** At $N_{\text{sim}} = 5$ (Figure 4a), the main improvement comes from using the policy optimization solution $\bar{\pi}$ as the learning target (LEARN variant); using $\bar{\pi}$ during search or acting leads to an additional marginal improve-

ment; **(2)** Interestingly, we observe a different behavior at $N_{\text{sim}} = 50$ (Figure 4b). In this case, using $\bar{\pi}$ for learning or acting does not lead to a noticeable improvement. However, the superior performance of ALL is mostly due to sampling according to $\bar{\pi}$ during search (SEARCH).

The improved performance when using $\bar{\pi}$ as the learning target (LEARN) illustrates the theoretical considerations of Section 3: at low simulation budgets, the discretization noise in $\hat{\pi}$ makes it a worse training target than $\bar{\pi}$, but this advantage vanishes when the number of simulations per step increases. As predicted by the theoretical results of Section 3, learning and acting using $\bar{\pi}$ and $\hat{\pi}$ becomes equivalent when the simulation budget increases.

On the other hand, we see a slight but significant improvement when sampling the next node according to $\bar{\pi}$ during search (SEARCH) regardless of the simulation budget. This could be explained by the fact that even at high simulations budget, the SEARCH modification also affect deeper node that have less simulations.

5.3. Search with large action space – continuous control

The previous results confirm the intuitions presented in Sections 3 and 4; namely, the ALL variation greatly improves performance at low simulation budgets, and obtain marginally higher performance at high simulation budgets. Since simulation budgets are relative to the number of action, these improvements are critical in tasks with a high number of actions, where MuZero might require a prohibitively high simulation budgets; prior work (Dulac-Arnold et al., 2015; Metz et al., 2017; Van de Wiele et al., 2020) has already identified continuous control tasks as an interesting testbed.

Benchmarks We select high-dimensional environments from DeepMind Control Suite (Tassa et al., 2018). The observations are images and action space $\mathcal{A} = [-1, 1]^m$ with m dimensions. We apply an action discretization method similar to that of Tang and Agrawal (2019). In short, for a continuous action space m dimensions, each dimension is discretized into K evenly spaced atomic actions. With proper parameterization of the policy network (see, *e.g.*, Appendix B.2), we can reduce the effective branching factor to $Km \ll K^m$, though this still results in a much larger action space than Atari benchmarks. In Appendix C.2, we provide additional descriptions of the tasks.

Result In Figure 5, we compare MuZero with the ALL variant on the CheetahRun environment of the DeepMind Control Suite (Tassa et al., 2018). We evaluate the performance at low ($N_{\text{sim}} = 4$), medium ($N_{\text{sim}} = 12$) and "high" ($N_{\text{sim}} = 50$) simulation budgets, for an effective action space of size 30 ($m = 6$, $K = 5$). The horizontal line corresponds to the performance of model-free D4PG also

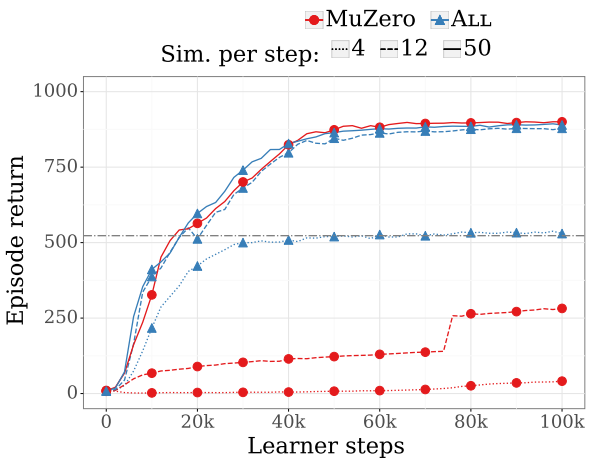


Figure 5. Comparison of the median score over 3 seeds of MuZero (red) and ALL (blue) at 4 (dotted) and 50 (solid line) simulations per step on Cheetah Run (Control Suite).

trained on pixel observations (Barth-Maron et al., 2018), as reported in (Tassa et al., 2018). Appendix C.2 provides experimental results on additional tasks. We again observe that ALL outperforms the original MuZero at low simulation budgets and still achieves faster convergence to the same asymptotic performance with more simulations. Figure 1 compares the asymptotic performance of MuZero and ALL as a function of the simulation budget at 100k learner steps.

Conclusion In this paper, we showed that the action selection formula used in MCTS algorithms, most notably AlphaZero, approximates the solution to a regularized policy optimization problem formulated with search Q-values. From this theoretical insight, we proposed variations of the original AlphaZero algorithm by explicitly using the exact policy optimization solution instead of the approximation. We show experimentally that these variants achieve much higher performance at low simulation budget, while also providing statistically significant improvements when this budget increases.

Our analysis on the behavior of model-based algorithms (i.e., MCTS) has made explicit connections to model-free algorithms. We hope that this sheds light on new ways of combining both paradigms and opens doors to future ideas and improvements.

Acknowledgements The authors would like to thank Alaa Saade, Bernardo Avila Pires, Bilal Piot, Corentin Tallec, Daniel Guo, David Silver, Eugene Tarassov, Florian Strub, Jessica Hamrick, Julian Schrittwieser, Katrina McKinney, Mohammad Gheshlaghi Azar, Nathalie Beauguerlange, Pierre Ménard, Shantanu Thakoor, Théophane Weber, Thomas Mesnard, Toby Pohlen and the DeepMind team.

References

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. (2018). Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. (2020). Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20.

Anthony, T., Nishihara, R., Moritz, P., Salimans, T., and Schulman, J. (2019). Policy gradient search: Online planning and expert iteration without search trees. *arXiv preprint arXiv:1904.03646*.

Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N., and Lillicrap, T. (2018). Distributional policy gradients. In *International Conference on Learning Representations*.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Bubeck, S., Cesa-Bianchi, N., et al. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122.

Csiszár, I. (1964). Eine informationstheoretische ungleichung und ihre anwendung auf beweis der ergodizitaet von markoffschen ketten. *Magyer Tud. Akad. Mat. Kutato Int. Koezl.*, 8:85–108.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). Openai baselines.

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. (2015). Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*.

Farquhar, G., Rocktäschel, T., Igl, M., and Whiteson, S. (2017). TreeQN and ATreeC: Differentiable tree-structured models for deep reinforcement learning. *arXiv preprint arXiv:1710.11417*.

Fox, R., Pakman, A., and Tishby, N. (2015). Taming the noise in reinforcement learning via soft updates. *arXiv preprint arXiv:1512.08562*.

Geist, M., Scherrer, B., and Pietquin, O. (2019). A theory of regularized markov decision processes. *arXiv preprint arXiv:1901.11275*.

Google (2020). Cloud TPU — Google Cloud. https://cloud.google.com/tpu/.

Grill, J.-B., Domingues, O. D., Ménard, P., Munos, R., and Valko, M. (2019). Planning in entropy-regularized Markov decision processes and games. In *Neural Information Processing Systems*.

Guez, A., Weber, T., Antonoglou, I., Simonyan, K., Vinyals, O., Wierstra, D., Munos, R., and Silver, D. (2018). Learning to search with mctsnets. *arXiv preprint arXiv:1802.04697*.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org.

Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Pfaff, T., Weber, T., Buesing, L., and Battaglia, P. W. (2020). Combining Q-learning and search with amortized value estimates. In *International Conference on Learning Representations*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., and Silver, D. (2018). Distributed prioritized experience replay. In *International Conference on Learning Representations*.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*.

Liese, F. and Vajda, I. (2006). On divergences and informations in statistics and information theory. *IEEE Transactions on Information Theory*, 52(10):4394–4412.

Metz, L., Ibarz, J., Jaitly, N., and Davidson, J. (2017). Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*.

Neu, G., Jonsson, A., and Gómez, V. (2017). A unified view of entropy-regularized Markov decision processes. *arXiv preprint arXiv:1705.07798*.

O'Donoghue, B., Munos, R., Kavukcuoglu, K., and Mnih, V. (2016). Combining policy gradient and Q-learning. *arXiv preprint arXiv:1611.01626*.

Oh, J., Singh, S., and Lee, H. (2017). Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6118–6128.

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

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2019). Mastering Atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017a). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017b). Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., et al. (2017c). 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.

Song, H. F., Abdolmaleki, A., Springenberg, J. T., Clark, A., Soyer, H., Rae, J. W., Noury, S., Ahuja, A., Liu, S., Tirumala, D., et al. (2019). V-MPO: On-policy maximum a posteriori policy optimization for discrete and continuous control. *arXiv preprint arXiv:1909.12238*.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Tang, Y. and Agrawal, S. (2019). Discretizing continuous action space for on-policy optimization. *arXiv preprint arXiv:1901.10500*.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). DeepMind control suite. *arXiv preprint arXiv:1801.00690*.

Van de Wiele, T., Warde-Farley, D., Mnih, A., and Mnih, V. (2020). Q-learning in enormous action spaces via amortized approximate maximization. *arXiv preprint arXiv:2001.08116*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Ziebart, B. D. (2010). *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Carnegie Mellon University, USA.

A. Details on search for AlphaZero

Below we briefly present details of the search procedure for AlphaZero. Please refer to the original work (Silver et al., 2017a) for more comprehensive explanations.

As explained in the main text, the search procedure starts with a MDP state x_0, which is used as the root node of the tree. The rest of this tree is progressively built as more simulations are generated. In addition to Q-function $Q(x, a)$, prior $\pi_\theta(x, a)$ and visit counts $n(x, a)$, each node also maintains a reward $R(x, a) = r(x, a)$ and value $V(x)$ estimate.

In each simulation, the search consists of several parts: **Selection**, **Expansion** and **Backup**, as below.

Selection. From the root node x_0, the search traverses the tree using the action selection formula of Eq. 1 until a leaf node x_l is reached.

Expansion. After a leaf node x_l is reached, the search selects an action from the leaf node, generates the corresponding child node x_c and appends it to the tree $\mathcal{T}$. The statistics for the new node are then initialized to $Q(x_c, a) = \min_{x\in\mathcal{T},a'\in\mathcal{A}} Q(x, a')$ (pessimistic initialization), $n(x, a) = 0$ for $\forall a \in \mathcal{A}$.

Back-up. The back-up consists of updating statistics of nodes encountered during the forward traversal. Statistics that need updating include the Q-function $Q(x, a)$, count $n(x, a)$ and value $V(x)$. The newly expanded node n_c updates its value $V(x)$ to be either the Monte-Carlo estimation from random rollouts (e.g. board games) or a prediction of the value network (e.g. Atari games). For the other nodes encountered during the forward traversal, all other statistics are updated as follows:

$$\begin{aligned} V(x) &\leftarrow (V(x) \cdot \sum_b n(x, b) + (R(x, a) + \gamma V(\text{child}(x, a))/(1 + \sum_b n(x, b)) \\ Q(x, a) &\leftarrow R(x, a) + \gamma V(\text{child}(x, a)), \\ n(x, a) &\leftarrow n(x, a) + 1, \end{aligned}$$

where $\text{child}(x, a)$ refers to the child node obtained by taking action a from node x.

Note that, in order to make search parameters agnostic to the scale of the numerical rewards (and, therefore, values), Q-function statistics $Q(x, a)$ are always normalized by statistics in the search tree before applying the action selection formula; in practice, Eq. 1 uses the normalized $Q_z(x, a)$ defined as:

$$Q_z(x, a) = \frac{Q(x, a) - \min_{x\in\mathcal{T},a\in\mathcal{A}} Q(x, a)}{\max_{x\in\mathcal{T},a\in\mathcal{A}} Q(x, a) - \min_{x\in\mathcal{T},a\in\mathcal{A}} Q(x, a)}. \tag{18}$$

B. Implementation details

B.1. Agent

For ease of implementation and availability of computational resources, the experimental results from Section 5 were obtained with a scaled-down version of MuZero (Schrittwieser et al., 2019). In particular, our implementation uses smaller networks compared to the architecture described in Appendix F of (Schrittwieser et al., 2019): we use only 5 residual blocks with 128 hidden layers for the dynamics function, and the residual blocks in the representation functions have half the number of channels. Furthermore, we use a stack of only 4 past observations instead of 32. Additionally, some algorithmic refinements (such as those described in Appendix H of (Schrittwieser et al., 2019)) have not been implemented in the version that we use in this paper.

Our experimental results have been obtained using either 4 or 8 Tesla v100 GPUs for learning (compared to 8 third-generation Google Cloud TPUs (Google, 2020) in the original MuZero paper, which are approximately equivalent to 64 v100 GPUs). Each learner GPU receives data from a separated, prioritized experience replay buffer (Horgan et al., 2018) storing the last 500000 transitions. Each of these buffers is filled by 512 dedicated CPU actors[12], each running a different environment instance. Finally, each actor receives updated parameters from the learner every 500 learner steps (corresponding to

[12]For 50 simulations per step; this number is scaled linearly as $12 + 10 \cdot N_{\text{sim}}$ to maintain a constant total number of frames per second when varying N_{sim}.

approximately 4 minutes of wall-clock time); because episodes can potentially last several minutes of wall-clock time, weights updating will usually occur within the duration of an episode. The total score at the end of an episode is associated to the version of the weights that were used to select the final action in the episode.

Hyperparameters choice generally follows those of (Schrittwieser et al., 2019), with the exception that we use the Adam optimizer with a constant learning rate of 0.001.

B.2. Details on discretizing continuous action space

AlphaZero (Silver et al., 2017a) is designed for discrete action spaces. When applying this algorithm to continuous control, we use the method described in (Tang and Agrawal, 2019) to discretize the action space. Although the idea is simple, discretizing continuous action space has proved empirically efficient (Andrychowicz et al., 2020; Tang and Agrawal, 2019). We present the details below for completeness.

Discretizing the action space We consider a continuous action space $\mathcal{A} = [-1, 1]^m$ with m dimensions. Each dimension is discretized into $K = 5$ bins; specifically, the continuous action along each dimension is replaced by K atomic categorical actions, evenly spaced between $[-1, 1]$. This leads to a total of K^m actions, which grows exponentially fast (e.g. $m = 6$ leads to about 10^4 joint actions). To avoid the curse of dimensionality, we assume that the parameterized policy can be factorized as $\pi_\theta(\mathbf{a}|x) = \Pi_{i=1}^m \pi_\theta^{(i)}(a_i|x)$, where $\pi_\theta^{(i)}(a_i|x)$ is the marginal distribution for dimension i, $a_i \in \{1, 2...K\}$ is the discrete action along dimension i and $\mathbf{a} = [a_1, a_2...a_m]$ is the joint action.

Modification to the search procedure Though it is convenient to assume a factorized form of the parameterized policy (Andrychowicz et al., 2020; Tang and Agrawal, 2019), it is not as straightforward to apply the same factorization assumption to the Q-function $Q(x, \mathbf{a})$. A most naive way of applying the search procedure is to maintain a Q-table of size K^m with one entry for each joint action, which may not be tractable in practice. Instead, we maintain m separate Q-tables each with K entries $Q_i(x, a_i)$. We also maintain m count tables $n(x, a_i)$ with K entries for each dimension.

To make the presentation clear, we detail on how the search is applied. At each node of the search tree, we maintain m tables each with K entries as introduced above. The three core components of the tree search are modified as follows.

- **Selection.** During forward action selection, the algorithm needs to select an action $\mathbf{a}$ at node x. This joint action $\mathbf{a}$ has all its components a_i selected independently, using the action selection formula applied to each dimension. To select action at dimension i, we need the Q-table $Q_i(x, a_i)$, the prior $\pi_\theta^{(i)}(a_i|x)$ and count $n(x, a_i)$ for dimension i.

- **Expansion.** The expansion part does not change.

- **Back-up.** During the value back-up, we update Q-tables of each dimension independently. At a node x, given the downstream reward $R(x, a)$ and child value $V(\text{child}(x, \mathbf{a}))$, we generate the target update for each Q-table and count table as $Q(x, a_i) \leftarrow R(x, a) + \gamma V(\text{child}(x, \mathbf{a}))$ and $n(x, a_i) \leftarrow n(x, a_i) + 1$.

The m small Q-tables can be interpreted as maintaining the marginalized values of the joint Q-table. Indeed, let us denote by $Q(x, \mathbf{a})$ the joint Q-table with K^m entries. At dimension i, the Q-table $Q(x, a_i)$ increments its values purely based on the choice of a_i, regardless of actions in other dimension $a_j, j \neq i$. This implies that the Q-table $Q(x, a_i)$ marginalizes the joint Q-table $Q(x, \mathbf{a})$ via the visit count distribution.

Details on the learning At the end of the tree search, a distribution target $\hat{\pi}$ or $\bar{\pi}$ is computed from the root node. In the discretized case, each component of the target distribution is computed independently. For example, $\hat{\pi}_i$ is computed from $N(x_0, a_i)$. The target distribution derived from constrained optimization $\bar{\pi}_i$ is also computed independently across dimensions, from $Q(x_0, a_i)$ and $N(x_0, a_i)$. In general, let $\pi_{\text{target}}(\cdot|x)$ be the target distribution and $\pi_{\text{target}}^{(i)}(\cdot|x)$ its marginal for dimension i. Due to the factorized assumption on the policy distribution, the update can be carried out independently for each dimension. Indeed, $\text{KL}[\pi_{\text{target}}(\cdot|x), \pi_\theta(\cdot|x)] = \sum_{i=1}^m \text{KL}[\pi_{\text{target}}^{(i)}(\cdot|x), \pi_\theta^{(i)}(\cdot|x)]$, sums over dimensions.

B.3. Practical computation of $\bar{\pi}$

The vector $\bar{\pi}$ is defined as the solution to a multi-dimensional optimization problem; however, we show that it can be computed easily by dichotomic search. We first restate the definition of $\bar{\pi}$,

$$\bar{\pi} \triangleq \arg\max_{\mathbf{y}\in\mathcal{S}} \big[\mathbf{q}^T\mathbf{y} - \lambda_N \mathrm{KL}[\pi_\theta, \mathbf{y}]\big]. \tag{8}$$

Let us define

$$\forall a \in \mathcal{A} \quad \pi_\alpha[a] \triangleq \lambda_N \frac{\pi_\theta[a]}{\alpha - \mathbf{q}[a]} \quad \text{and} \quad \alpha^\star \triangleq \max\left\{\alpha \in \mathbb{R} \text{ s.t } \sum_b \pi_\alpha[b] = 1\right\}. \tag{19}$$

Proposition 4.

$$(i) \quad \pi_{\alpha^\star} = \bar{\pi} \tag{20}$$

$$(ii) \quad \alpha^\star \geq \alpha_{min} \triangleq \max_{b\in\mathcal{A}}(q[b] + \lambda_N \cdot \pi_\theta[b]) \tag{21}$$

$$(iii) \quad \alpha^\star \leq \alpha_{max} \triangleq \max_{b\in\mathcal{A}} q[b] + \lambda_N \tag{22}$$

$$\tag{23}$$

As $\sum_b \bar{\pi}_\alpha[b]$ is strictly decreasing on $\alpha \in (\alpha_{\min}, \alpha_{\max})$, Proposition 4 guarantees that $\bar{\pi}$ can be computed easily using dichotomic search over $(\alpha_{\min}, \alpha_{\max})$.

Proof of (i).

Proof. The proof start the same as the one of Lemma 3 of Appendix D.1 setting $f(x) = -\log(x)$ to get

$$\exists\alpha \quad q + \lambda_N \cdot \frac{\pi_\theta}{\bar{\pi}} = \alpha\mathbb{1}, \tag{24}$$

with $\mathbb{1}$ being the the vector such that $\forall a \ \mathbb{1}_a = 1$. Therefore there exists $\alpha \in \mathbb{R}$ such that

$$\bar{\pi} = \frac{\lambda_N \cdot \pi_\theta}{\alpha - q} \tag{25}$$

Then α is set such that $\sum_b \bar{\pi}_b = 1$ and $\forall b \ \bar{\pi}_b \geq 0$. □

Proof of (ii).

Proof.

$$\forall a \quad 1 \geq \bar{\pi}[a] = \frac{\lambda_N \cdot \pi_\theta[a]}{\alpha - q[a]} \implies \forall a \quad \alpha \geq q[a] + \lambda_N \cdot \pi_\theta[a] \tag{26}$$

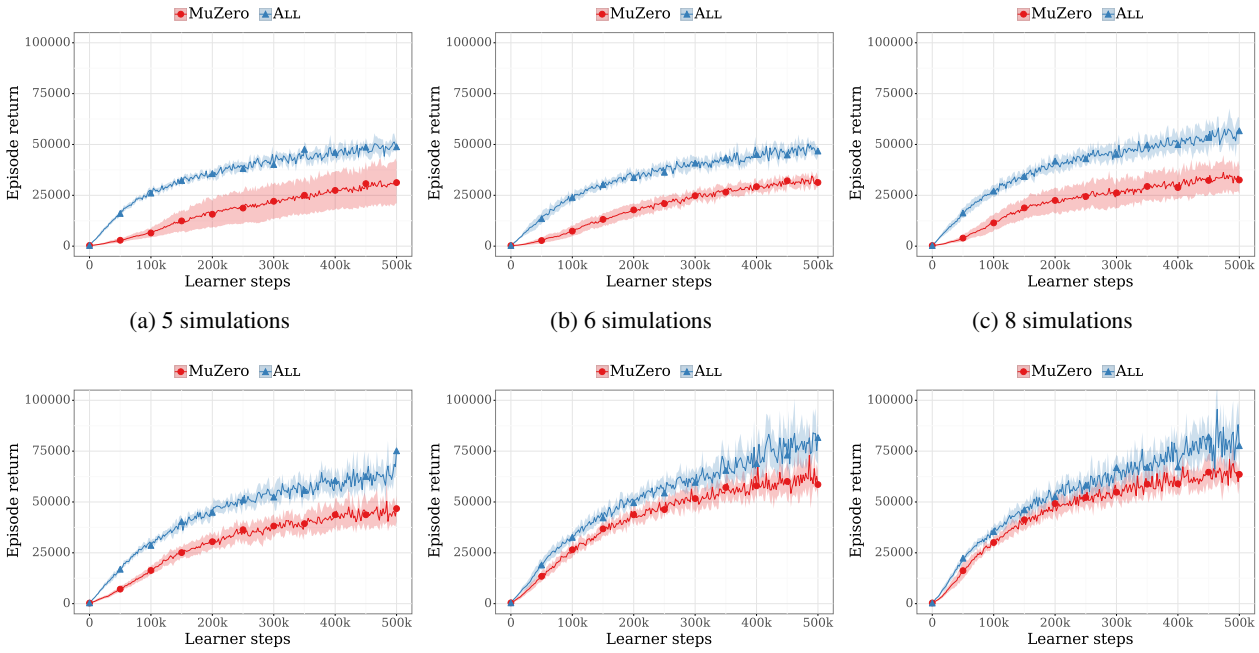
□

Proof of (iii).

Proof.

$$\sum_b \pi_{\alpha_{\max}}[b] = \sum_b \frac{\lambda_N \cdot \pi_\theta[b]}{\max_{c\in\mathcal{A}} q[c] + \lambda_N - q[b]} \leq \sum_b \frac{\lambda_N \cdot \pi_\theta[b]}{\lambda_N} = 1 \tag{27}$$

We combine this with the fact that $\sum_b \pi_\alpha[b]$ is a decreasing function of α for any $\alpha > \max_b q[b]$, and $\sum_b \pi_{\alpha^\star}[b] = 1$. □



(a) 5 simulations (b) 6 simulations (c) 8 simulations

(d) 12 simulations (e) 24 simulations (f) 50 simulations

Figure 6. Dispersion between seeds at different number of simulations per step on Ms Pacman.

C. Additional experimental results

C.1. Complements to Section 5.1

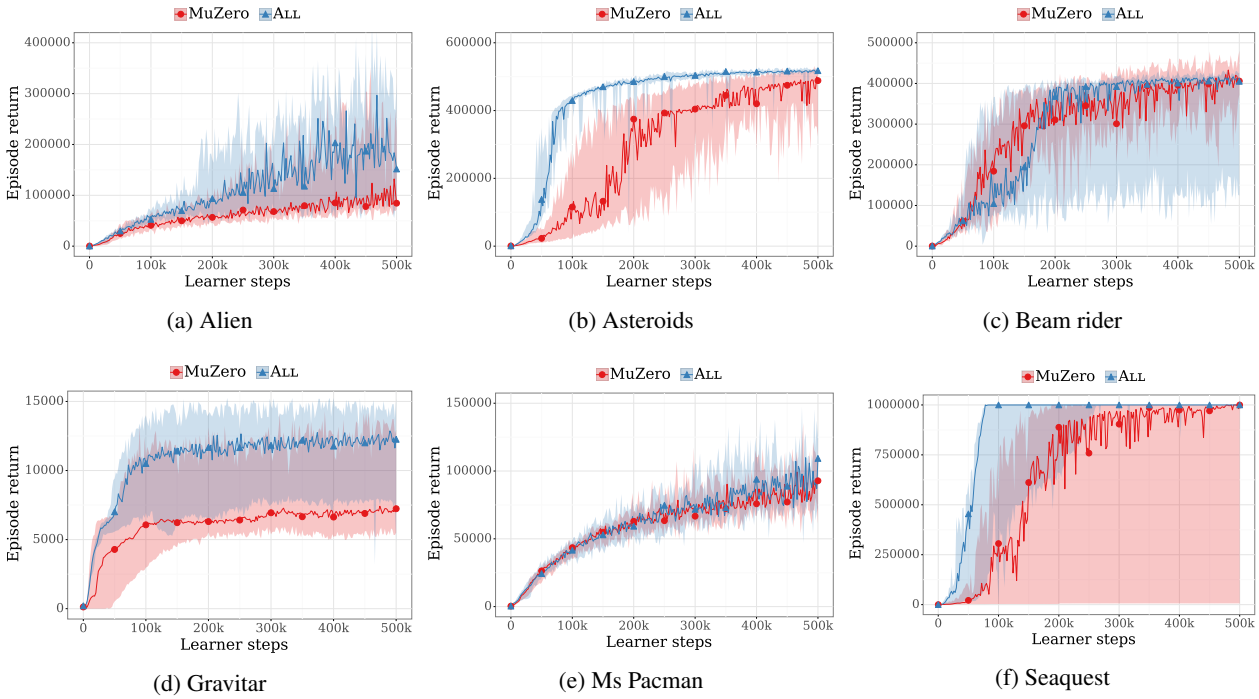
Figure 6 presents a comparison of the score obtained by our MuZero implementation and the proposed ALL variant at different simulation budgets on the Ms. Pacman level; the results from Figure 2 are also included fore completeness. In this experiment, we used 8 seeds with 8 GPUs per seed and a batch size of 256 per GPU. We use the same set of hyper-parameters for MuZero and ALL; these parameters were tuned on MuZero. The solid line corresponds to the average score (solid line) and the 95% confidence interval (shaded area) over the 8 seeds, averaged for each seed over buckets of 2000 learner steps without additional smoothing. Interestingly, we observe that ALL provides improved performance at low simulation budgets while also reducing the dispersion between seeds.

Figure 7 presents a comparison of the score obtained by our MuZero implementation and the proposed ALL variant on six Atari games, using 6 seeds per game and a batch size of 512 per GPU and 8 GPUs; we use the same set of hyper-parameters as in the other experiments. Because the distribution of scores across seeds is skewed towards higher values, we represent dispersion between seeds using the min-max interval over the 6 seeds (shaded area) instead of using the standard deviation; the solid line represents the median score over the seeds.

C.2. Complements to Section 5.3

Details on the environments The DeepMind Control Suite environments (Tassa et al., 2018) are control tasks with continuous action space $\mathcal{A} = [-1, 1]^m$. These tasks all involve simulated robotic systems and the reward functions are designed so as to guide the system for accomplish e.g. locomotion tasks. Typically, these robotic systems have relatively low-dimensional sensory recordings which summarize the environment states. To make the tasks more challenging, for observations, we take the third-person camera of the robotic system and use the image recordings as observations to the RL agent. These images are of dimension $64 \times 64 \times 3$.

Figures 9 to 12 present a comparison of MuZero and ALL on a subset of 4 of the medium-difficulty (Van de Wiele et al., 2020) DeepMind Control Suite (Tassa et al., 2018) tasks chosen for their relatively high-dimensional action space among these medium-difficulty problems ($n_{\text{dim}} = 6$). Figure 8 compare the score of MuZero and ALL after 100k learner steps on



(a) Alien (b) Asteroids (c) Beam rider

(d) Gravitar (e) Ms Pacman (f) Seaquest

Figure 7. Comparison of median score over 6 seeds of MuZero and ALL on six Atari games with 50 simulations per step. The shaded area correspond the the best and worst seeds.

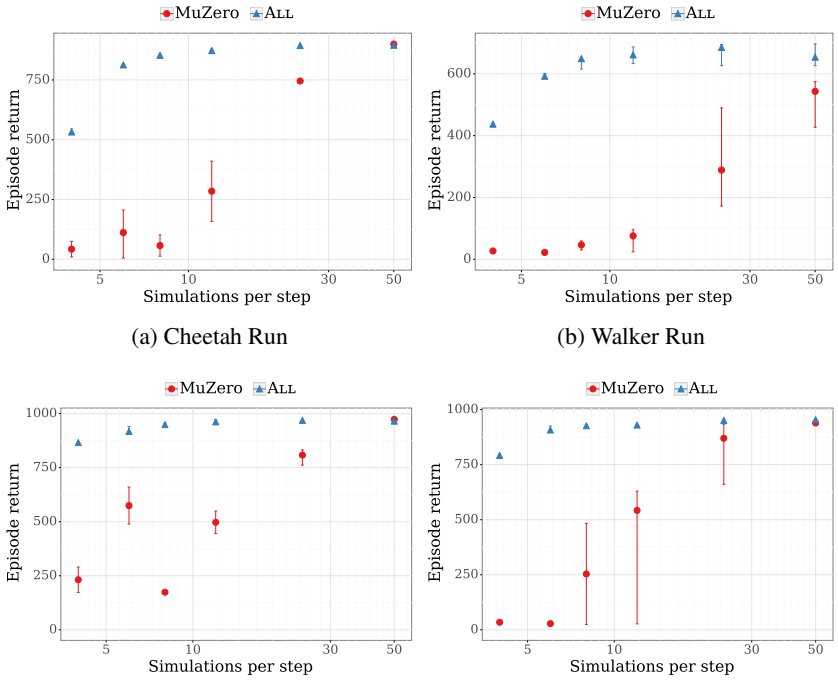
these four medium difficulty Control problems. These continuous control problems are cast to a discrete action formulation using the method presented in Appendix B.2; note that these experiments only use pixel renderings and not the underlying scalar states.

These curves present the median (solid line) and min-max interval (shaded area) computed over 3 seeds in the same settings as described in Appendix C.1. The hyper-parameters are the same as in the other experiments; no specific tuning was performed for the continuous control domain. The horizontal dashed line corresponds to the performance of the D4PG algorithm when trained on pixel observations only (Barth-Maron et al., 2018), as reported by (Tassa et al., 2018).

C.3. Complemantary experiments on comparison with PPO

Since we interpret the MCTS-based algorithms as regularized policy optimization algorithms, as a sanity check for the proposal's performance gains, we compare it with state-of-the-art proximal policy optimization (PPO) (Schulman et al., 2017). Since PPO is a near on-policy optimization algorithm, whose gradient updates are purely based on on-policy data, we adopt a lighter network architecture to ensure its stability. Please refer to the public code base (Dhariwal et al., 2017) for a review of the neural network architecture and algorithmic details.

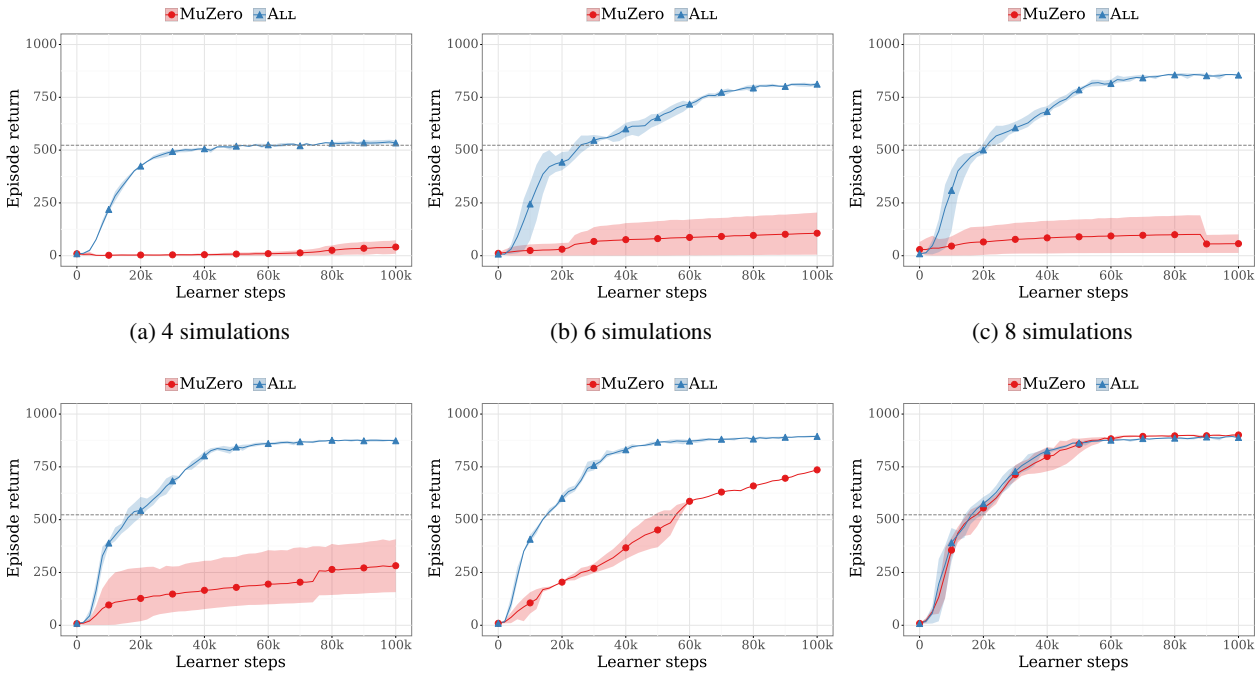
To assess the performance of PPO, we train with both state-based inputs and image-based inputs. State-based inputs are low-dimensional sensor data of the environment, which renders the input sequence strongly Markovian (Tassa et al., 2018). For image-based training, we adopt the same inputs as in the main paper. The performance is reported in Table 1 where each score is the evaluation performance of PPO after the convergence takes place. We observe that state-based PPO performs significantly better than image-based PPO, while in some cases it matches the performance of ALL. In general, image-based PPO significantly underperforms ALL.



(a) Cheetah Run (b) Walker Run

(c) Walker Stand (d) Walker Walk

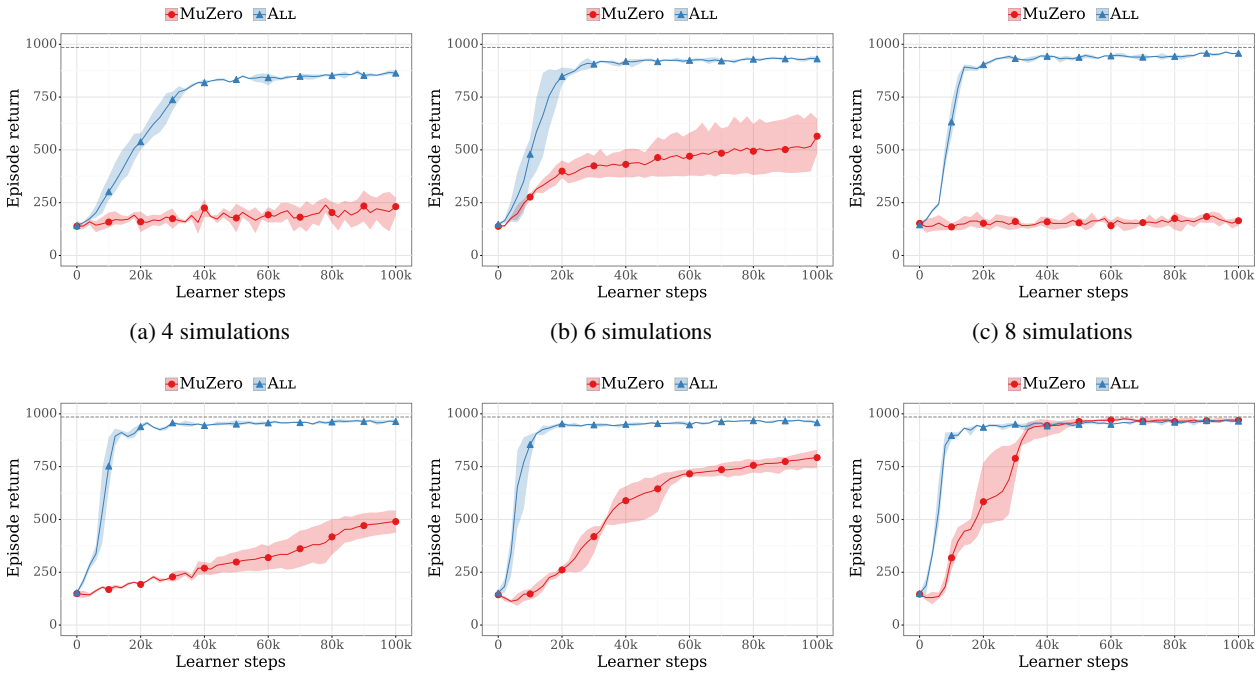
Figure 8. Score of MuZero and ALL on Continuous control tasks after 100k learner steps as a function of the number of simulations N_{sim}.



(a) 4 simulations (b) 6 simulations (c) 8 simulations

(d) 12 simulations (e) 24 simulations (f) 50 simulations

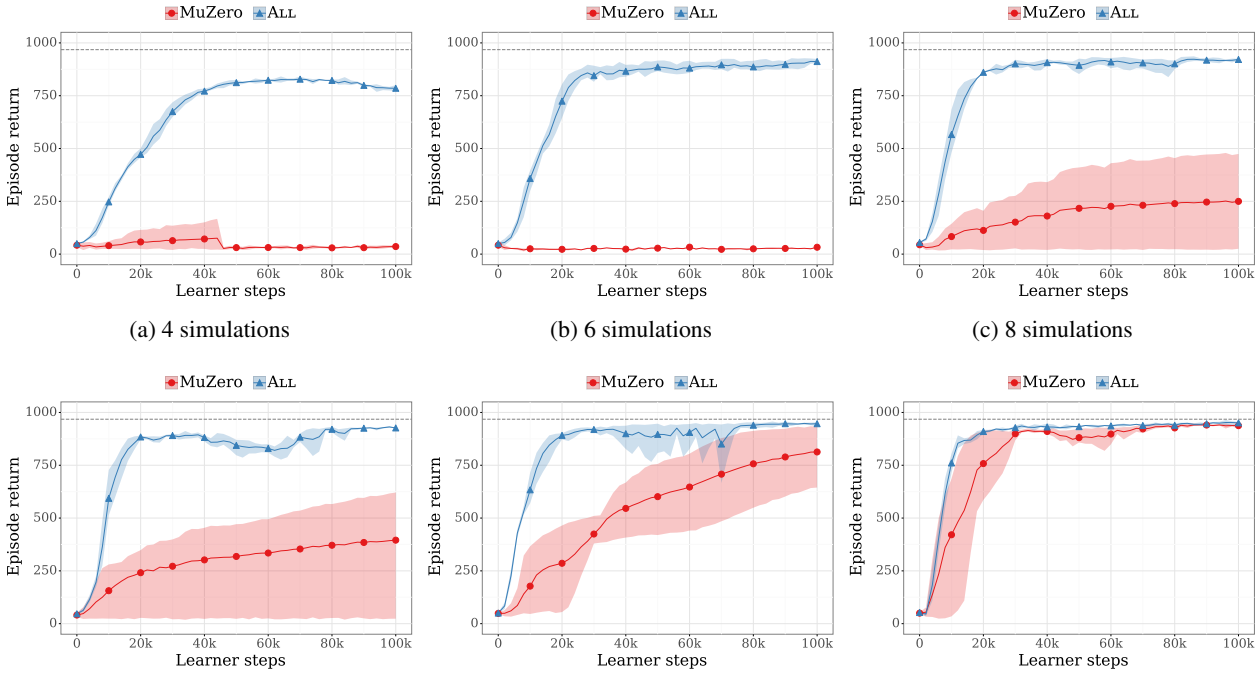
Figure 9. Comparison of MuZero and ALL on Cheetah Run.



(a) 4 simulations (b) 6 simulations (c) 8 simulations

(d) 12 simulations (e) 24 simulations (f) 50 simulations

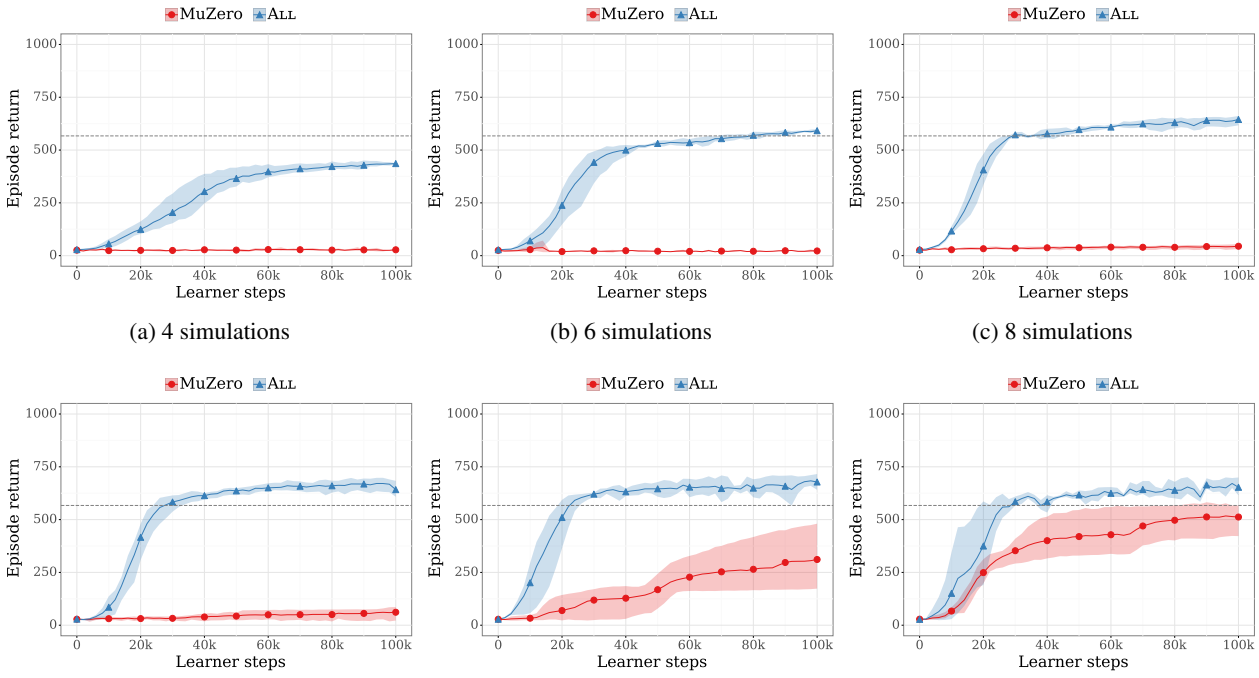
Figure 10. Comparison of MuZero and ALL on Walker Stand.



(a) 4 simulations (b) 6 simulations (c) 8 simulations

(d) 12 simulations (e) 24 simulations (f) 50 simulations

Figure 11. Comparison of MuZero and ALL on Walker Walk.



(a) 4 simulations (b) 6 simulations (c) 8 simulations

(d) 12 simulations (e) 24 simulations (f) 50 simulations

Figure 12. Comparison of MuZero and ALL on Walker Run.

Benchmarks	**PPO (state)**	**PPO (image)**	**MuZero (image)**	**ALL(image)**
WALKER-WALK	406	270	925	941
WALKER-STAND	937	357	959	951
WALKER-RUN	340	71	533	644
CHEETAH-RUN	538	285	887	882

Table 1. Comparison to the performance of PPO baselines on benchmark tasks. The inputs to PPO are either state-based or image-based. The performance is computed as the evaluated returns after the training is completed, averaged across 3 random seeds.

D. Derivations for Section 3

D.1. Proof of Proposition 1, Eq. 11 and Proposition 2.

We start with a definition of the f-divergence (Csiszár, 1964).

Definition 2 (f-*divergence*)**.** *For any probability distributions p and q on $\mathcal{A}$ and function $f : \mathbb{R} \to \mathbb{R}$ such that f is a convex function on $\mathbb{R}$ and $f(1) = 0$, the f-divergence D_f between p and q is defined as*

$$D_f(p, q) = \sum_{b \in \mathcal{A}} q(b) f\left(\frac{p(b)}{q(b)}\right) \tag{28}$$

Remark 1. *Let D_f be a f-divergence,*

(1) $\forall x, y\ D(x, y) \geq 0$ *(2)* $D(x, y) = 0 \iff x = y$ *(3) $D(x, y)$ is jointly convex in x and y.*

We states four lemmas that we formally prove in Appendix D.2.

Lemma 1.

$$\nabla_\pi \left(\mathbf{q}^T \pi - \lambda \cdot D_f[\pi, \pi_\theta]\right) = \mathbf{q} - \lambda \cdot f'\left(\frac{\pi}{\pi_\theta}\right) \tag{29}$$

Where π_θ is assumed to be non zero. We now restate the definition of $\hat{\pi}[a] \triangleq \frac{n_a+1}{N+|\mathcal{A}|}$.

Lemma 2.

$$\arg\max_a \left[\frac{\partial}{\partial n_a}\left(\mathbf{q}^T \pi - \lambda \cdot D_f[\hat{\pi}, \pi_\theta]\right)\right] = \arg\max_a \left[\mathbf{q_a} - \lambda \cdot f'\left(\frac{\hat{\pi}(a)}{\pi_\theta(a)}\right)\right] \tag{30}$$

Now we consider a more general definition of $\bar{\pi}$ using any f-divergence for some $\lambda_f > 0$ and assume $\pi_\theta > 0$,

$$\bar{\pi}_f \triangleq \arg\min_{\mathbf{y} \in \mathcal{S}} \mathbf{q}^T \mathbf{y} - \lambda_f D_f(\pi_\theta, \mathbf{y}). \tag{31}$$

We also consider the following action selection formula based on f-divergence D_f.

$$a_f^\star \triangleq \arg\max_a \left[q_a - \lambda_f \cdot f'\left(\frac{\hat{\pi}}{\pi_\theta}\right)\right], \quad \text{with.} \tag{32}$$

Lemma 3.

$$\arg\max_a \left[q_a - \lambda_f \cdot f'\left(\frac{\hat{\pi}(a)}{\pi_\theta(a)}\right)\right] = \arg\max_a \left[f'\left(\frac{\bar{\pi}_f(a)}{\pi_\theta(a)}\right) - f'\left(\frac{\hat{\pi}(a)}{\pi_\theta(a)}\right)\right]. \tag{33}$$

Lemma 4.

$$\hat{\pi}(a_f^\star) \leq \bar{\pi}(a_f^\star). \tag{34}$$

Applying Lemmas 2 and 4 with the appropriate function f directly leads to Proposition 1, Proposition 2, and Proposition 3. In particular, we use

$$\text{For AlphaZero:} \qquad f(x) = -\log(x) \tag{35}$$

$$\text{For UCT:} \qquad f(x) = 2 - 2\sqrt{x} \tag{36}$$

Algorithm	Function $f(x)$	Derivative $f'(x)$	Associated f-divergence	Associated action selection formula
—	$x \cdot \log(x)$	$\log(x) + 1$	$D_f(p, q) = \mathrm{KL}(p, q)$	$\arg\max_a q_a + \frac{c}{\sqrt{N}} \cdot \log\left(\frac{\pi_\theta(a)}{n_a+1}\right)$
UCT	$2 - 2\sqrt{x}$	$-\frac{1}{\sqrt{x}}$	$D_f(p, q) = 2 - 2\sum_{b \in \mathcal{A}} \sqrt{p_a \cdot q_a}$	$\arg\max_a q_a + c \cdot \sqrt{\frac{\pi_\theta}{n_a+1}}$
AlphaZero	$-\log(x)$	$-\frac{1}{x}$	$D_f(p, q) = \mathrm{KL}(q, p)$	$\arg\max_a q_a + c \cdot \pi_\theta \cdot \frac{\sqrt{N}}{n_a+1}$

D.2. Proofs of Lemmas 1 to 4

Proof of Lemma 1

Proof. For any action $a \in \mathcal{A}$ using basic differentiation rules we have

$$\big(\nabla_\pi\big(\mathbf{q}^T\pi - \lambda_f \cdot D_f[\pi,\pi_\theta]\big)\big)[a] = \frac{\partial}{\partial\pi_a}\left[\sum_{b\in\mathcal{A}}(q_b\cdot\pi_b) - \lambda_f\sum_{b\in\mathcal{A}}\pi_\theta(b)f\left(\frac{\pi_b}{\pi_\theta(b)}\right)\right] \tag{37}$$

$$= \frac{\partial}{\partial\pi_a}\left[\sum_{b\in\mathcal{A}}(q_b\cdot\pi_b)\right] - \lambda_f\cdot\frac{\partial}{\partial\pi_a}\sum_{b\in\mathcal{A}}\pi_\theta(b)f\left(\frac{\pi_b}{\pi_\theta(b)}\right) \tag{38}$$

$$= q_a - \lambda_f\cdot\pi_\theta(a)\cdot\frac{\partial}{\partial\pi_a}f\left(\frac{\pi_a}{\pi_\theta(a)}\right) \tag{39}$$

$$= q_a - \lambda_f\cdot f'\left(\frac{\pi_a}{\pi_\theta(a)}\right) = \left(\mathbf{q} - \lambda_f\cdot f'\left(\frac{\pi}{\pi_\theta}\right)\right)[a] \tag{40}$$

□

Proof of Lemma 2

Proof.

$$\frac{\partial}{\partial n_a}\big(\mathbf{q}^T\pi - \lambda_f\cdot D_f[\hat\pi,\pi_\theta]\big) = \frac{\partial}{\partial n_a}\left[\sum_{b\in\mathcal{A}}\left(q_b\cdot\frac{n_b+1}{|\mathcal{A}|+\sum_{c\in\mathcal{A}}n_c}\right) - \lambda_f\sum_{b\in\mathcal{A}}\pi_\theta(b)\cdot f\left(\frac{n_b+1}{(|\mathcal{A}|+\sum_{c\in\mathcal{A}}n_c)\cdot\pi_\theta(b)}\right)\right] \tag{41}$$

$$= \beta + \frac{q_a}{|\mathcal{A}|+\sum_{c\in\mathcal{A}}n_c} - \frac{\lambda_f}{|\mathcal{A}|+\sum_{c\in\mathcal{A}}n_c}f'\left(\frac{n_a+1}{(|\mathcal{A}|+\sum_{c\in\mathcal{A}}n_c)\cdot\pi_\theta(b)}\right) \tag{42}$$

$$= \beta + \frac{1}{|\mathcal{A}|+\sum_{c\in\mathcal{A}}n_c}\left(q_a - \lambda_f f'\left(\frac{\hat\pi(a)}{\pi_\theta(b)}\right)\right), \tag{43}$$

where $\beta = -\dfrac{\sum_{b\in\mathcal{A}}\left[q_b - \lambda_f f'\left(\frac{\hat\pi(b)}{\pi_\theta(b)}\right)\right]}{\big(|\mathcal{A}|+\sum_{c\in\mathcal{A}}n_c\big)^2}$ is independent of a. Also because $\frac{1}{|\mathcal{A}|+\sum_{c\in\mathcal{A}}n_c} > 0$ then

$$\arg\max_a \frac{\partial}{\partial n_a}\big(\mathbf{q}^T\pi - \lambda_f\cdot D_f[\hat\pi,\pi_\theta]\big) = \arg\max_a\left[\mathbf{q}_a - \lambda_f\cdot f'\left(\frac{\hat\pi(a)}{\pi_\theta(a)}\right)\right] \tag{44}$$

$$\tag{45}$$

□

Proof of Lemma 3

Proof. The Eq. 31 is a differentiable strictly convex optimization problem, its unique solution satisfies the KKT condition requires (see Section 5.5.3 of Boyd and Vandenberghe, 2004) therefore there exists $\alpha \in \mathbb{R}$ such that for all actions a,

$$\nabla_{\bar\pi}\big(\mathbf{q}^T\bar\pi - \lambda_f D_f(\pi_\theta,\bar\pi)\big) = \alpha\mathbb{1} \tag{46}$$

where $\mathbb{1}$ is the vector constant equal one: $\forall a\ \mathbb{1}_a = 1$. Using Lemma 1 setting π to $\bar\pi$ we get

$$\exists\alpha \quad q - \lambda_f\cdot f'\left(\frac{\bar\pi}{\pi_\theta}\right) = \alpha\mathbb{1}. \tag{47}$$

$$q - \lambda_f \cdot f'\left(\frac{\hat{\pi}}{\pi_\theta}\right) = q_a - \lambda_f \cdot \left(f'\left(\frac{\hat{\pi}}{\pi_\theta}\right) + f'\left(\frac{\bar{\pi}}{\pi_\theta}\right) - f'\left(\frac{\bar{\pi}}{\pi_\theta}\right)\right) \tag{48}$$

$$= \alpha \mathbb{1} + \lambda_f \cdot \left(f'\left(\frac{\bar{\pi}}{\pi_\theta}\right) - f'\left(\frac{\hat{\pi}}{\pi_\theta}\right)\right) \tag{49}$$

$$\arg\max\left[q - \lambda_f \cdot f'\left(\frac{\hat{\pi}}{\pi_\theta}\right)\right] = \arg\max\left[\alpha \mathbb{1} + \lambda_f \cdot \left(f'\left(\frac{\bar{\pi}}{\pi_\theta}\right) - f'\left(\frac{\hat{\pi}}{\pi_\theta}\right)\right)\right] \tag{50}$$

$$= \arg\max\left[f'\left(\frac{\bar{\pi}}{\pi_\theta}\right) - f'\left(\frac{\hat{\pi}}{\pi_\theta}\right)\right] \qquad \text{(because } \lambda_f > 0\text{)} \tag{51}$$

$$\tag{52}$$

□

Proof of Lemma 4

Proof. Since $\sum_a \hat{\pi}(a|x) = \sum_a \bar{\pi}(a|x) = 1$, there exists at least an action a_0 for which $0 \leq \hat{\pi}(a_0|x) \leq \bar{\pi}(a_0|x)$ then $0 \leq \frac{\hat{\pi}(a_0|x)}{\pi_\theta(a|x)} \leq \frac{\bar{\pi}(a_0|x)}{\pi_\theta(a|x)}$ as $\pi_\theta(a|x) > 0$. Because f is convex then f' is increasing and therefore

$$f'\left(\frac{\bar{\pi}(a_0|x)}{\pi_\theta(a_0|x)}\right) - f'\left(\frac{\hat{\pi}(a_0|x)}{\pi_\theta(a_0|x)}\right) \geq 0. \tag{53}$$

Now using Lemma 3

$$f'\left(\frac{\bar{\pi}(a_f^\star|x)}{\pi_\theta(a_f^\star|x)}\right) - f'\left(\frac{\hat{\pi}(a_f^\star|x)}{\pi_\theta(a_f^\star|x)}\right) \geq f'\left(\frac{\bar{\pi}(a_0|x)}{\pi_\theta(a_0|x)}\right) - f'\left(\frac{\hat{\pi}(a_0|x)}{\pi_\theta(a_0|x)}\right) \tag{54}$$

We put Equations (53) and (54) together

$$f'\left(\frac{\bar{\pi}(a_f^\star|x)}{\pi_\theta(a_f^\star|x)}\right) - f'\left(\frac{\hat{\pi}(a_f^\star|x)}{\pi_\theta(a_f^\star|x)}\right) \geq 0 \tag{55}$$

Finally we use again that f' is increasing and $\pi_\theta > 0$ to conclude the proof

$$\hat{\pi}(a_f^\star|x) \leq \bar{\pi}(a_f^\star|x) \tag{56}$$

□

D.3. Tracking property in the constant $\bar{\pi}$ case

Let π be some target distribution independent of the round $t \geq 0$. At each round t, starting from $t = 1$, an action $a_t \in \mathcal{A}$ is selected and for any $t \geq 0$, we define

$$p_t(a) \triangleq \frac{n_t(a_t) + 1}{|\mathcal{A}| + \sum_b n_t(b)},$$

where for any action $a \in \mathcal{A}$, $n_t(a)$ is the number of rounds the action a has been selected,

$$\forall a \in \mathcal{A} \quad n_t(a) \triangleq \sum_{i \leq t} \delta(a_t = a) \quad \text{ and } \delta(a_t = a) \triangleq 1 \text{ if and only if } a_t = a.$$

Proposition 5. *Assume that for all rounds $t \geq 1$, and for the chosen action $a_t \in \mathcal{A}$ we have*

$$p_t(a_t) \leq \pi(a_t). \tag{57}$$

Then, we have that

$$\forall a \in \mathcal{A}, t \geq 1 \quad |\pi(a) - p_t(a)| \leq \frac{|\mathcal{A}| - 1}{|\mathcal{A}| + t}.$$

Before proving the proposition above, note that $\mathcal{O}(1/t)$ is the best approximation w.r.t. t, since for any integer $k \geq 0$, taking $\pi(a) = (\frac{1}{2} + k)/(|\mathcal{A}| + t)$, we have that for all $n \geq 0$,

$$\left|\pi(a) - \frac{n+1}{|\mathcal{A}|+t}\right| \geq \frac{1}{2}\frac{1}{|\mathcal{A}|+t},$$

which follows from the fact that $\forall k, n \in \mathbb{N}$, $\left|\frac{1}{2} + k - (n+1)\right| = \left|k - n - \frac{1}{2}\right| \geq \frac{1}{2}$.

Proof. By induction on the round t, we prove that

$$\forall t \geq 1, a \in \mathcal{A} \quad p_t(a) \leq \pi(a) + \frac{1}{|\mathcal{A}|+t}. \tag{58}$$

At round $t = 1$, Eq. 58 holds as for any action a, $n_t(a) \geq 0$ therefore $p_t(a) \leq 1$. Now, let us assume that Eq. 58 holds for some $t \geq 1$. We have that for all a,

$$\frac{1 + n_t(a)}{|\mathcal{A}| + \sum_b n_t(b)} \leq \pi(a) + \frac{1}{|\mathcal{A}|+t}.$$

Note that at each round, there is exactly one action chosen and therefore, $\sum_b n_t(b) = t$. Furthermore, for $a' \neq a_{t+1}$, we have that $n_{t+1}(a') = n_t(a')$, since a' has not been chosen at round $t+1$. Therefore, for $a' \neq a_{t+1}$,

$$p_{t+1}(a') = \frac{n_{t+1}(a')+1}{|\mathcal{A}|+t+1} = \frac{n_t(a')+1}{|\mathcal{A}|+t+1} \leq \frac{|\mathcal{A}|+t}{|\mathcal{A}|+t+1}p_t(a') \leq \frac{|\mathcal{A}|+t}{|\mathcal{A}|+t+1}\left(\pi(a') + \frac{1}{|\mathcal{A}|+t}\right) \leq \pi(a') + \frac{1}{|\mathcal{A}|+t+1}.$$

Now, for the chosen action, $n_{t+1}(a_{t+1}) = n_t(a_{t+1}) + 1$. Using our assumption stated in Eq. 57, we have that

$$p_{t+1}(a_{t+1}) = \frac{n_{t+1}(a_{t+1})+1}{|\mathcal{A}|+t+1} = \frac{n_t(a_{t+1})+1+1}{|\mathcal{A}|+t+1} \leq \frac{n_t(a_{t+1})+1}{|\mathcal{A}|+t+1} + \frac{1}{|\mathcal{A}|+t+1} \leq \pi(a_{t+1}) + \frac{1}{|\mathcal{A}|+t+1},$$

which concludes the induction. Next, we compute a lower bound. For any action $a \in \mathcal{A}$ and round $t \geq 1$,

$$p_t(a) = 1 - \sum_{b \neq a} p_t(b) \geq 1 - \sum_{b \neq a}\left(\pi(b) + \frac{1}{|\mathcal{A}|+t}\right) = \left(1 - \sum_{b \neq a}\pi(b)\right) - \sum_{b \neq a}\frac{1}{|\mathcal{A}|+t} = \pi(a) - \frac{|\mathcal{A}|-1}{|\mathcal{A}|+t}.$$

We have for any action $a \in \mathcal{A}$,

$$\pi(a) - \frac{|\mathcal{A}|-1}{|\mathcal{A}|+t} \leq p_t(a) \leq \pi(a) + \frac{1}{|\mathcal{A}|+t}.$$

Since when $|\mathcal{A}| = 1$, then by definition, $p_t(a) = \pi(a) = 1$ and for all rounds $t \geq 1$, we get

$$||\pi - p_t||_\infty \leq \frac{|\mathcal{A}|-1}{|\mathcal{A}|+t}.$$

□