Learning and Planning in Complex Action Spaces

Thomas Hubert [*1] **Julian Schrittwieser** [*1] **Ioannis Antonoglou** [1] **Mohammadamin Barekatain** [1]
Simon Schmitt [1] **David Silver** [1]

Abstract

Many important real-world problems have action spaces that are high-dimensional, continuous or both, making full enumeration of all possible actions infeasible. Instead, only small subsets of actions can be sampled for the purpose of policy evaluation and improvement. In this paper, we propose a general framework to reason in a principled way about policy evaluation and improvement over such sampled action subsets. This sample-based policy iteration framework can in principle be applied to any reinforcement learning algorithm based upon policy iteration. Concretely, we propose *Sampled MuZero*, an extension of the *MuZero* algorithm that is able to learn in domains with arbitrarily complex action spaces by planning over sampled actions. We demonstrate this approach on the classical board game of Go and on two continuous control benchmark domains: DeepMind Control Suite and Real-World RL Suite.

1. Introduction

Real-world environments abound with complexity in their action space. Physical reality is continuous both in space and time; hence many important problems, most notably physical control tasks, have continuous multi-dimensional action spaces. The joints of a robotic hand can assume arbitrary angles; the acceleration of a self-driving car should vary smoothly to minimise discomfort for passengers. Discrete problems also often have high-dimensional action spaces, leading to an exponential number of possible actions. Many other domains have richly structured actions spaces such as sentences, queries, images, or serialised objects. Consequently, a truly general reinforcement learning (RL) algorithm must be able to deal with such complex action spaces in order to be successfully applied to those real-world problems.

[*]Equal contribution [1]DeepMind, London, UK. Correspondence to: Thomas Hubert <tkhubert@google.com>.

Under submission, please do not distribute.

Recent advances in deep learning and RL have indeed led to remarkable progress in model-free RL algorithms for continuous action spaces (Lillicrap et al., 2015; Schulman et al., 2017; Barth-Maron et al., 2018; Abdolmaleki et al., 2018; Hoffman et al., 2020) and other complex action spaces (Dulac-Arnold et al., 2016). Simultaneously, planning based methods have enjoyed huge successes in domains with discrete action spaces, surpassing human performance in the classical games of chess and Go (Silver et al., 2018) or poker (Brown & Sandholm, 2018; Moravčík et al., 2017). The prospect of combining these two areas of research holds great promise for real-world applications.

The model-based *MuZero* (Schrittwieser et al., 2020) RL algorithm took a step towards applicability in real-world problems by learning a model of the environment and thus unlocking the use of the powerful methods of planning in domains where the dynamics of the environment are unknown or impossible to simulate efficiently. However, *MuZero* was only applied to domains with relatively small action spaces; small enough to be in fact enumerated in full by the tree-based search at its core.

Sample-based methods provide a powerful approach to dealing with large complex actions spaces. Rather than enumerating all possible actions, the idea is to sample a small subset of actions and compute the optimal policy or value function with respect to those samples. This simple strategy is so general that it can be applied to large, continuous, or structured action spaces. Specifically, action sampling can be used both to propose improvements to the policy at each of the sampled actions, and subsequently to evaluate the proposed improvements. However, to correctly improve or evaluate the policy across the entire action space, and not just the samples, one must understand how the sampling procedure interacts with both policy improvement and policy evaluation.

In this work, we propose a framework to reason in a principled way about policy improvement and evaluation computed over small subsets of sampled actions. We show how this local information can be used to train a global policy, act and even perform explicit steps of policy evaluation for the purpose of planning and local policy iteration. This sample-based framework can in principle be applied

to any reinforcement learning algorithm based upon policy iteration. Concretely, we propose *Sampled MuZero*, an algorithmically simple extension of the *MuZero*[1] algorithm that facilitates its application to domains with complex action spaces.

To demonstrate the generality of this approach, we apply our algorithm to two continuous control benchmark domains, the DeepMind Control Suite (Tassa et al., 2018) and Real-World RL Suite (Dulac-Arnold et al., 2020). We also demonstrate that our algorithm can be applied to large discrete action spaces, by sampling the actions in the game of Go, and show that high performance can be maintained even when sub-sampling a small fraction of possible moves.

2. Related Work

Previous research in reinforcement learning for complex or continuous action spaces has often focused on model-free algorithms.

Deep Deterministic Policy Gradient (DDPG) exploits the fact that in action spaces that are entirely continuous (no discrete action dimensions), the action-value function $Q(s, a)$ can be assumed to be differentiable with respect to the action a in order to efficiently compute policy gradients (Silver et al., 2014; Lillicrap et al., 2015). Distributed Distributional Deterministic Policy Gradients (D4PG) extends DDPG by using a distributional value function and a distributed training setup (Barth-Maron et al., 2018). Trust Region Policy Optimisation (TRPO) uses a hard KL constraint to ensure that the updated policy remains close to the previous policy during the policy improvement step (Schulman et al., 2015), to avoid catastrophic collapse. Proximal Policy Optimisation (PPO) has the same goal as TRPO, but instead uses the KL-divergence as a penalty in the loss function or clipping in the value function (Schulman et al., 2017). This results in a simpler algorithm with empirically better performance. In the regime of data-efficient off-policy algorithms, recent advances have derived actor-critic algorithms that optimise a (relative-)entropy regularised RL objective such as SAC (Haarnoja et al., 2018), MPO (Abdolmaleki et al., 2018), AWR (Peng et al., 2019). Among these, MPO uses a sample based policy improvement step that can be related to our algorithm (see section 4.4). Distributional MPO (DMPO) extends MPO to use a distributional Q-function (Hoffman et al., 2020).

Model-based control for high dimensional action spaces has recently seen a resurgence of interest (see e.g. (Byravan et al., 2020; Hafner et al., 2018; 2019; Koul et al., 2020)). While most of these algorithms consider direct policy optimisation against a learned model some have considered combinations of rollout based search/planning with policy learning. (Piché et al., 2018) use planning via sequential importance sampling of action sequences sampled from a SAC policy. (Bhardwaj et al., 2020) use a learned simulator to construct K-step returns for learning a soft Q-function. Closest to our work, (Springenberg et al., 2020) consider a sample based policy update similar to ours - but using a policy improvement operator based on the KL regularised objective rather than the MCTS based policy improvement that we consider here.

Sparse sampling algorithms (Kearns et al., 1999) are an effective approach to planning in large state spaces. The main idea is to sample K possible state transitions from each state, drawn from a generative model of the underlying MDP. Collectively, these samples provide a search tree over a subset of the MDP; planning over the sampled tree provides a near-optimal approximation, for large K, to the optimal policy for the full MDP, independent of the size of the state space. Indeed, sampling is known to address the curse of dimensionality in some cases (Rust, 1997). However, sparse sampling typically enumerates all possible actions from each state, and does not address issues relating to large action spaces. In contrast, our method samples actions rather than state transitions. In principle, it would be straightforward to combine both ideas; however, we focus in this paper upon the novel aspect relating to large action spaces and utilise deterministic transition models.

There have been several previous attempts at generalising *AlphaZero* and *MuZero* to continuous action spaces. These attempts have shown that such an extension is possible in principle, but have so far been restricted to very low dimensional cases and not yet demonstrated effectiveness in high-dimensional tasks. A0C (Moerland et al., 2018) describes an extension of *AlphaZero* to continuous action spaces using a continuous policy representation and REINFORCE (Williams, 1992) to estimate the gradients for the reverse KL divergence between the neural network policy estimate and the target MCTS policy, demonstrating some learning on the 1D Pendulum task. (Yang et al., 2020) describe a similar extension of *MuZero* to continuous actions and show promising results outperforming soft actor-critic (SAC) (Haarnoja et al., 2018) on environments with 1 and 2 dimensional action spaces.

The factorised policy representation described by (Tang & Agrawal, 2020) shows good results in a variety of domains; by representing each action dimension with a separate categorical distribution it efficiently avoids the exponential explosion in the number of actions faced by a simple discretisation scheme.

[1]The discussion in this paper applies equally to *AlphaZero* and *MuZero*; in the text we will only refer to *MuZero* for simplicity.

3. Background

We consider a standard reinforcement learning setup in which an agent acts in an environment by sequentially choosing actions over a sequence of time-steps in order to maximise a cumulative reward. We model the problem as a Markov decision process (MDP) which comprises a state space $\mathcal{S}$, an action space $\mathcal{A}$, an initial state distribution, a stationary transition dynamics distribution and a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

The agent's behaviour is controlled by a policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ which maps states to a probability distribution over the action space. The return from a state is defined as the sum of discounted future rewards $G_t = \sum_i \gamma^i r(s_{t+i}, a_{t+i})$ where γ is a discount factor in $[0, 1]$. The goal of the agent is to learn a policy which maximises the expected return from the start distribution.

In order to do so, a common strategy called *policy evaluation* consists in learning a value function that estimates the expected return of following policy π from a state s_t or a state action pair (s_t, a_t). The value function can then be used in a process called *policy improvement*, to find and learn better policies by for instance increasing the probabilities of actions with higher values. The process of repeatedly doing policy evaluation followed by policy improvement is at the heart of many reinforcement learning algorithms and is called *policy iteration*.

Naturally, a lot of research focuses on improving the methods for *policy evaluation* and *policy improvement*. One direction for scaling the efficiency of both is to evaluate, from the current state, several possible actions, or even several possible future trajectories by using a model, instead of just extracting information from the trajectory that was executed. Those evaluations can then be used to build a locally better policy over those actions. Planning algorithms such as Monte Carlo Tree Search (MCTS) (Coulom, 2006) take this even further and make several local *policy iteration* steps by repeatedly performing a policy improvement step followed by an explicit local step of policy evaluation of the improved policy in the aim of generating an even better policy locally.

From this perspective, the *MuZero* algorithm can be understood as the combination of two processes of *policy evaluation* and *policy improvement*. The inner process, concretely *MuZero*'s MCTS search, provides the *policy improvement* for the outer process which in turn learns the quantities: the model, reward function, value function and the policy, necessary for the inner process. Specifically, in the outer process, *MuZero* learns a deep neural network parameterising a model, a reward function, a state-value function and a policy. *Policy improvement* is accomplished by regressing the parametric policy towards the improved policy built by *MuZero*'s MCTS search. The improved policy is also used for acting. The value function is learned using the usual tools of *policy evaluation* such as temporal-difference learning (Sutton, 1988). These two objectives coupled with the learning of the reward function drive the learning of the model. In the inner process, *MuZero*'s MCTS search takes several analytical *policy iteration* steps: values in the search tree are estimated by explicitly averaging n-step returns bootstrapped from the value function (*policy evaluation*) while visits are directed towards high policy and high value actions (*policy improvement*). This results in an improved policy and an estimate of the value of this improved policy that can be used for the outer process.

This raises a few questions, especially in the case where only a small subset of the action space can be evaluated to build the locally improved policy.

- how to select the actions or trajectories to be evaluated
- how to build a locally improved policy over those actions
- how to use the locally improved policy to learn about the global policy
- how to use it to act
- how to perform an explicit local step of policy evaluation of the improved policy for planning
- how all these steps interact with each other

In the following, we will assume that the actions to be evaluated are sampled from some proposal distribution β and that we have at our disposal some process to build a locally improved policy. We will mainly focus on the last four questions and propose a general framework to reason in a principled way about policy evaluation and improvement over such sampled action subsets.

4. Sample-based Policy Iteration

Let $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ be a policy and $\mathcal{I}\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ be an improved policy of π: $\forall s \in \mathcal{S}, v^{\mathcal{I}\pi}(s) \geq v^{\pi}(s)$. If we had complete access to $\mathcal{I}\pi$, we would directly use it for policy improvement by projecting it back onto the space of realisable policies. However, when the action space $\mathcal{A}$ is too large, it might only be feasible to compute an improved policy over a small subset of actions.

It is not immediately clear how to use this locally improved policy to perform principled policy improvement, or policy evaluation of the improved policy, because this locally improved policy only gives us information regarding the sampled actions.

We propose a framework which relies on writing both operations as an expectation with respect to the fully improved policy $\mathcal{I}\pi$ and use the samples we have to estimate these

expectations. This allows us to use the conceptually correct target $\mathcal{I}\pi$ to define the objectives and clearly surface the approximations that are introduced afterwards. Specifically, we will restrict ourselves to the general class of policy improvement operators that we call *action-independent* as defined below in 4.2.

4.1. Operator view of Policy Improvement

We use the concepts introduced by (Ghosh et al., 2020) and decompose *policy improvement* into the successive application of two operators: (a) a policy improvement operator $\mathcal{I}$ which maps any policy to a policy achieving strictly larger return; and (b) a projection operator $\mathcal{P}$, which finds the best approximation of this improved policy in the space of realisable policies. With those notations, the process of *policy improvement* can be written as $\mathcal{P} \circ \mathcal{I}$.

(Ghosh et al., 2020) showed that the policy gradient algorithm can be thought of having the following policy improvement operator: $\mathcal{I}\pi(s,a) \propto \pi(s,a)Q(s,a)$ where $Q(s,a)$ is the action-value function. They also showed that PPO's (Schulman et al., 2017) policy improvement operator is $\mathcal{I}\pi(s,a) \propto \exp(Q(s,a)/\tau)$ where τ is a temperature parameter. Similarly, MPO's (Abdolmaleki et al., 2018) policy improvement operator can be written $\mathcal{I}\pi(s,a) \propto \pi(s,a)\exp(Q(s,a)/\tau)$ and AWR (Peng et al., 2019) uses a similar form of improved policy, replacing the action-value function by the advantage function $\mathcal{I}\pi(s,a) \propto \pi(s,a)\exp(A(s,a)/\tau)$.

4.2. Action-Independent Policy Improvement Operator

We define a policy improvement operator as *action-independent* if it can be written as:

$$\mathcal{I}\pi(a|s) = f(s,a,Z(s))$$

where $Z(s)$ is a unique state dependent normalising factor defined by $\forall a \in \mathcal{A}, f(s,a,Z(s)) \geq 0$ and $\sum_{a\in\mathcal{A}} f(s,a,Z(s)) = 1$.[2]

All of the policy improvement operators described above are action-independent.

MPO Example: MPO's policy improvement operator can be written $\mathcal{I}\pi(a|s) = f(s,a,Z(s)) = \pi(s,a)\exp(Q(s,a)/\tau)/Z(s)$ and $Z(s) = \sum_a \pi(s,a)\exp(Q(s,a)/\tau)$.

4.3. Sample-Based Action-Independent Policy Improvement Operator

Let $\{a_i\}$ be K actions sampled from a proposal distribution β and $\hat{\beta}(a|s) = \frac{1}{K}\sum_i \delta_{a,a_i}$ the corresponding empirical distribution[3] which is non-zero only on the sampled actions $\{a_i\}$.

We define the sample-based action-independent[4] policy improvement operator as

$$\hat{\mathcal{I}}_\beta\pi(a|s) = (\hat{\beta}/\beta)(a|s)f(s,a,\hat{Z}_\beta(s))$$

where $\hat{Z}_\beta(s)$ is a unique state dependent normalising factor defined by $\forall a \in \mathcal{A}, (\hat{\beta}/\beta)(a|s)f(s,a,\hat{Z}_\beta(s)) \geq 0$ and $\sum_{a\in\mathcal{A}}(\hat{\beta}/\beta)(a|s)f(s,a,\hat{Z}_\beta(s)) = 1$. We have used the shorthand notation $(\hat{\beta}/\beta)(a|s)$ to mean $\hat{\beta}(a|s)/\beta(a|s)$.

MPO Example: MPO's sample-based action-independent policy improvement operator using $\beta = \pi$ would therefore be $\hat{\mathcal{I}}_\beta\pi(a|s) = \hat{\beta}(a|s)\exp(Q(s,a)/\tau)/\hat{Z}_\beta(s)$ with $\hat{Z}_\beta(s) = \sum_a \hat{\beta}(a|s)\exp(Q(s,a)/\tau)$.

4.4. Computing an expectation with respect to $\mathcal{I}\pi$

We focus in this section on evaluating for a given state s the expectation $\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s]$ of a random variable X given actions $\{a_i\}$ sampled from a distribution β and the sample-based improved policy $\hat{\mathcal{I}}_\beta\pi$.

Theorem. For a given random variable X, we have

$$\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s] = \lim_{K\to\infty}\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a|s)X(s,a)$$

Furthermore, $\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a|s)X(s,a)$ is approximately normally distributed around $\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s]$ as $K\to\infty$:

$$\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a|s)X(s,a) \sim \mathcal{N}(\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s], \frac{\sigma^2}{K})$$

where $\sigma^2 = Var_{a\sim\beta}[\frac{f(s,a,Z(s))}{\beta}X(s,a)|s]$.

Proof See Appendix E

Corollary. The sample-based policy improvement operator converges in distribution to the true policy improvement operator:

$$\lim_{K\to\infty}\hat{\mathcal{I}}_\beta\pi = \mathcal{I}\pi$$

and is approximately normally distributed around the true policy improvement operator as $K\to\infty$.

Proof. See Appendix E

We illustrate this result in Figure 1.

[2] In the continuous case, sums would be replaced by integrals.

[3] δ_{a,a_i} represents the Kronecker delta function. In the continuous case, it would be replaced by the Dirac delta function $\delta(a-a_i)$.

[4] We will omit the action-independent qualifier in the rest of the text when it is clear from the context.

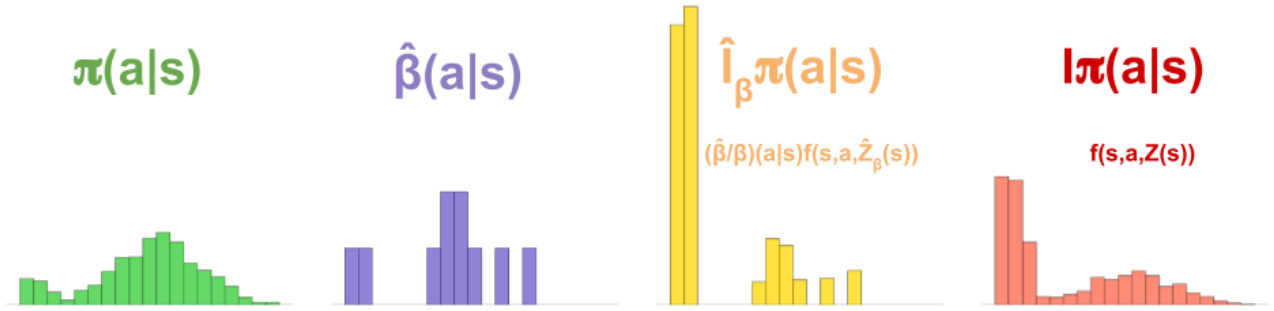

Figure 1. **Sample-based Policy Improvement.** On the left, the current policy $\pi(a|s)$. Next, K actions $\{a_i\}$ are sampled from a proposal distribution β and $\hat{\beta}(a|s)$ is the corresponding empirical distribution. A sample-based improved policy $\hat{\mathcal{I}}_\beta\pi(a|s) = (\hat{\beta}/\beta)(a|s)f(s, a, \hat{Z}_\beta(s))$ is then built. As the number of samples K increases $\hat{\mathcal{I}}_\beta\pi(a|s)$ converges to the improved policy $\mathcal{I}\pi(a|s) = f(s, a, Z(s))$.

4.5. Sample-based Policy Evaluation and Improvement

The previous expression computing an estimate of $\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s]$ using the quantity $\hat{\mathcal{I}}_\beta\pi$ and the sampled actions $\{a_i\}$ can be used for policy improvement and policy evaluation of the improved policy.

Policy improvement can be performed by for instance instantiating $X = -\log\pi_\theta$, minimising the cross-entropy between π_θ and the improved policy $\mathcal{I}\pi$: $CE = \mathbb{E}_{a\sim\mathcal{I}\pi}[-\log\pi_\theta]$.

Additionally, samples from $\mathcal{I}\pi$ can be obtained by re-sampling an action a from $\hat{\mathcal{I}}_\beta\pi$. This procedure also known as Sampling Importance Resampling (SIR) (Rubin, 1987) gives us a way to act with the improved policy and reuse the usual tools such as temporal-difference learning to do policy evaluation of the improved policy.

Finally, for instance for the purpose of planning, an explicit step of policy evaluation of the improved policy can be computed by estimating 1-step or n-step returns. Using for example $X = r + \gamma V'$ lets us backpropagate the value V' by one step in a search tree: $V(s) = \mathbb{E}_{a\sim\mathcal{I}\pi}[r + \gamma V'|s]$.

5. *Sampled MuZero*

Building on the sample-based policy iteration framework established in the previous section, we now instantiate those ideas in the context of a complete system. Concretely, we apply our sampling procedure to the *MuZero* algorithm, to produce a new algorithm that we term *Sampled MuZero*. This algorithm may be applied to any domain where *MuZero* can be applied; but furthermore can also be used, in principle, to learn and plan in domains with arbitrarily complex action spaces.

As introduced in the background section, *MuZero* may be understood as combining an inner process of policy iteration, within its Monte-Carlo tree search, and an outer process, in its overall interactions with the environment.

5.1. Inner Policy Evaluation and Improvement

Specifically, within its search tree, *MuZero* estimates values by explicitly averaging n-step returns samples (*policy evaluation*) and selects the next node to evaluate and expand by recursively maximising (*policy improvement*) over the probabilistic upper confidence tree (PUCT) bound (Silver et al., 2016)

$$arg\,max_a Q(s,a) + c(s)\cdot\pi(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

where $c(s)$ is an exploration factor controlling the influence of the policy $\pi(s,a)$ relative to the values $Q(s,a)$ as nodes are visited more often.

Naive Modification. A first approach to extending *MuZero*'s MCTS search is to search over the sampled actions $\{a_i\}$ and keep the PUCT formula unchanged, directly using the probabilities π coming from the policy network in the PUCT formula just like in *MuZero*. The search's visit count distribution f can then be used to construct the sampled-based $\hat{\mathcal{I}}_\beta\pi = \hat{\beta}/\beta f$ to correct for the effect of sampling at policy network training time and acting time, but also dynamically as the tree is built for value backpropagation (inner *policy evaluation*). Theoretically this procedure is not complicated, but in practice it might lead to unstable results because of the f/β term, especially if f is represented by normalised visit counts which have limited numerical precision.

Proposed Modification. Instead, we propose to search with probabilities $\hat{\pi}_\beta(s,a)$ proportional to $(\hat{\beta}/\beta\pi)(s,a)$, in place of $\pi(s,a)$ in the PUCT formula and directly use the resulting visit count distributions just like in *MuZero*. We use the following Theorem to justify this proposed modification.

Theorem. Let $\mathcal{I}\pi$ be the visit count distribution[5] of *MuZero*'s search using prior π when considering the whole

[5]for a given number of simulations

action space $\mathcal{A}$ and let $\mathcal{I}\hat{\pi}_\beta$ be the visit count distribution obtained by searching using prior $\hat{\pi}_\beta$. Then, $\mathcal{I}\hat{\pi}_\beta$ is approximately equal to the sample-based policy improvement associated to $\mathcal{I}\pi$. In other words, $\mathcal{I}\hat{\pi}_\beta \approx \hat{\mathcal{I}}_\beta\pi$.

Proof. See Appendix F

We can therefore directly use the results of the previous section 4.4 and in particular,

$$\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s] \approx \sum_{a\in\mathcal{A}} \mathcal{I}\hat{\pi}_\beta(s,a)X(s,a)$$

This lets us conclude that the only modification beyond sampling that needs to be made to *MuZero* is to use $\hat{\pi}_\beta$ instead of π in the PUCT formula. The rest of the *MuZero* algorithm, from estimating the values in the search tree by averaging n-step returns, to acting and training the policy network using the visit count distribution, can proceed unchanged.

Remark. Note that, if $\beta = \pi^{1/\tau}$, the probabilities $\hat{\pi}_\beta$ used in the PUCT formula can be written: $\hat{\pi}_\beta = \hat{\beta}/\beta\pi = \hat{\beta}\pi^{1-1/\tau}$. If $\tau = 1$, $\hat{\pi}_\beta$ is equal to the empirical sampling/prior distribution $\hat{\beta}$. This means that the search is guided by a potentially quasi uniform prior $\hat{\beta}$ but only evaluates relatively high probability actions. If $\tau > 1$, the search evaluates more diverse samples but is guided by more peaked probabilities $\hat{\beta}\pi^{1-1/\tau}$.

5.2. Outer Policy Improvement

Once the inner iterations of policy improvement and policy evaluation within Monte-Carlo tree search have been completed, the net result is a set of visit counts $N(s,a)$ at the root state s of the search tree, corresponding to each sampled action a. These visit counts may be normalised to provide the sample-based improved policy $\mathcal{I}\hat{\pi}_\beta(a|s) = N(s,a)/\sum_b N(s,b)$. Following the argument in the previous section, these visit counts already take account of the fact that the root actions were sampled according to β.

Hence all that remains is to project the sample-based improved policy back onto the space of representable policies, using an appropriate projection operator $\mathcal{P}$. Following *MuZero*, we choose a standard projection operator for probability distributions that selects parameters θ minimising the KL divergence $KL(\mathcal{I}\hat{\pi}_\beta||\pi_\theta)$.

5.3. Outer Policy Evaluation

To select actions, the agent samples its behaviour from its sample-based improved policy, $\mathcal{I}\hat{\pi}_\beta(a|s) = N(s,a)/\sum_b N(s,b)$. As above, we note that this already corrects for the sampling procedure in the construction of the visit counts, and hence may be used directly as a policy.

The outer policy evaluation step then follows directly from *MuZero*, i.e. a value function is trained from n-step returns, using trajectories of behaviour generated by the sample-based improved policy.

5.4. Search Tree Node Expansion

In *MuZero*, each time a leaf node is expanded, all the $N = |\mathcal{A}|$ actions of the action space are returned alongside the probabilities π the policy network assigns to each of those actions.

Proposed Modification. In *Sampled MuZero*, we instead sample $K \ll N$ actions from a distribution β and return each action a along with its corresponding probabilities $\pi(s,a)$ and $\beta(s,a)$.

We note that, if the number of simulations of the search is much bigger than K, techniques such as progressive widening (Chaslot et al., 2008) could in principle be used to dynamically sample more actions for nodes on highly visited search paths.

5.5. Sampling distribution β

In principle any sampling distribution β with a wide support can be used, including the uniform distribution. However, as only a limited number of samples can be drawn, it is preferable to sample moves that are likely according to our current estimate for the best policy, i.e. the policy network.[6]

Proposed Modification. We use $\beta = \pi$, potentially modulated by a temperature parameter. To encourage exploration and to make sure that even low prior moves have an opportunity to be reassessed from time to time, *MuZero* combines the prior π produced by the policy network with Dirichlet noise at the root of the search tree. We obtain the same behaviour in *Sampled MuZero* by also including noise in β and π, ensuring that low prior moves can be sampled and searched.

6. Experiments

We evaluated the performance of *Sampled MuZero* on a variety of reinforcement learning environments. We focus upon standard benchmark environments in which clear baselines are available for comparison. We use those benchmarks to explore two important properties of real-world applications. First, whether *Sampled MuZero* is sufficiently general to operate across discrete and continuous environments of very different types. Second, whether the algorithm is robust to sampling – that is, whether we can come close to the performance of algorithms that have access to the entire action set (and are therefore not scalable to large action spaces), when only sampling a small fraction of the action space.

[6]Note that *MuZero* with a limited number of simulations will only visit the high prior moves

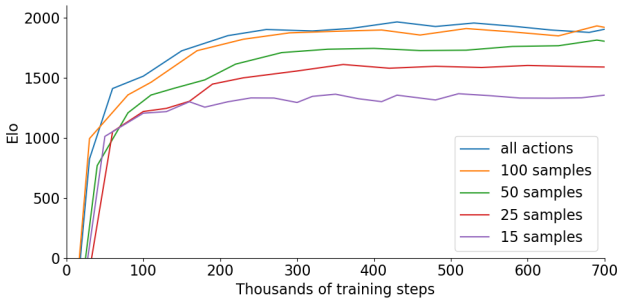

Figure 2. **Results in the classical board game of Go.** Performance of *Sampled MuZero* (1 seed per experiment) with different number $K \in \{15, 25, 50, 100\}$ of samples throughout training compared to a *MuZero* baseline that always considers all available actions (action space of size 362). Elo scale anchored to final baseline performance of 2000 Elo. As the number of sampled actions increases, performance rapidly approaches the baseline that always considers all actions.

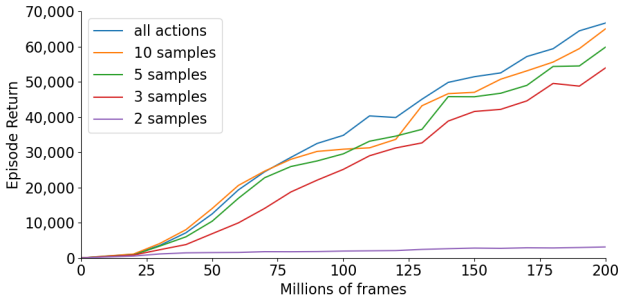

Figure 3. **Results in Ms. Pacman.** Performance of *Sampled MuZero* (1 seed per experiment) with different number of samples throughout training compared to a *MuZero* baseline that always considers all available actions. As the number of sampled actions increases, performance rapidly approaches the baseline that always considers all actions.

6.1. Go

Go has long been a challenge problem for AI, with only the *AlphaGo* (Silver et al., 2016; 2018; Schrittwieser et al., 2020) family of algorithms finally surpassing human professional players. It is a domain that requires deep and precise planning and as such is an ideal domain to put the planning capabilities of *Sampled MuZero* to the test.

Using *MuZero* as a baseline, we trained multiple instances of *Sampled MuZero* with varying number of action samples $K \in \{15, 25, 50, 100\}$ (see Figure 2). The size of the action space in 19x19 Go is 362 (all board points plus pass), so all the tested values of K only cover a small part of the action space. As expected, the performance improves as K increases, with $K = 50$ samples already closely approaching the performance of the baseline that is allowed to search over all possible actions.

6.2. Atari

We also performed the same experiment as in Figure 2 for the Arcade game of Ms. Pacman, from the classic Atari RL benchmark. The action space in Atari is of size 18. Searching with $K = 2$ samples is not sufficient for efficient learning, but already with $K = 3$ samples performance rapidly approaches the baseline that is allowed to search all possible actions without sampling (Figure 3).

6.3. DeepMind Control Suite

The DeepMind Control Suite (Tassa et al., 2018) provides a set of continuous control tasks based on MuJoCo (Todorov et al., 2012) and has been widely used as a benchmark to assess performance of continuous control algorithms. For the experiments in this paper we use the task classification and data budgets introduced in Acme (Hoffman et al., 2020), evaluating *Sampled MuZero* on the easy, medium and hard tasks. We additionally evaluated *Sampled MuZero* on the manipulator tasks which are known to be interesting and difficult.

In its most common setup, the control suite domains provide 1 dimensional state inputs (as opposed to 2 dimensional image inputs in board games and Atari as used by *MuZero*). We therefore used a variation of the *MuZero* model architecture in which all convolutions are replaced by fully-connected layers (see Appendix A for further details). For the policy prediction, we chose the factored policy representation introduced by (Tang & Agrawal, 2020), representing each dimension by a categorical distribution. There are however no difficulties in working directly with continuous actions and we show results with a policy prediction parameterised with a Gaussian distribution on the hard and manipulator tasks in the Appendix (Figure A.1).

Sampled MuZero showed good performance across the task set (Figure 7 for full results), with especially good results for tasks in the most difficult hard and manipulator categories (Figure 4) such as *humanoid.run* or the *manipulator* tasks in general.

The control suite domains can also be configured to provide raw pixel inputs instead of 1 dimensional state inputs. We ran *Sampled MuZero* on the same tasks with the same data budget (25M frames) and the same hyperparameters. As demonstrated in Figure 5, *Sampled MuZero* can be applied to efficiently learn from raw pixel inputs as well. It is particularly remarkable that *Sampled MuZero* can learn to control the 21 dimensional humanoid from raw pixel inputs only. In addition, we compared *Sampled MuZero* to the Dreamer agent (Hafner et al., 2019) in Appendix A.2, Table 2. *Sampled MuZero* equalled or surpassed the Dreamer agent's performance in all tasks, without any action repeat (Dreamer uses an action repeat of 2), observation reconstruction, or any hyperparameter re-tuning.

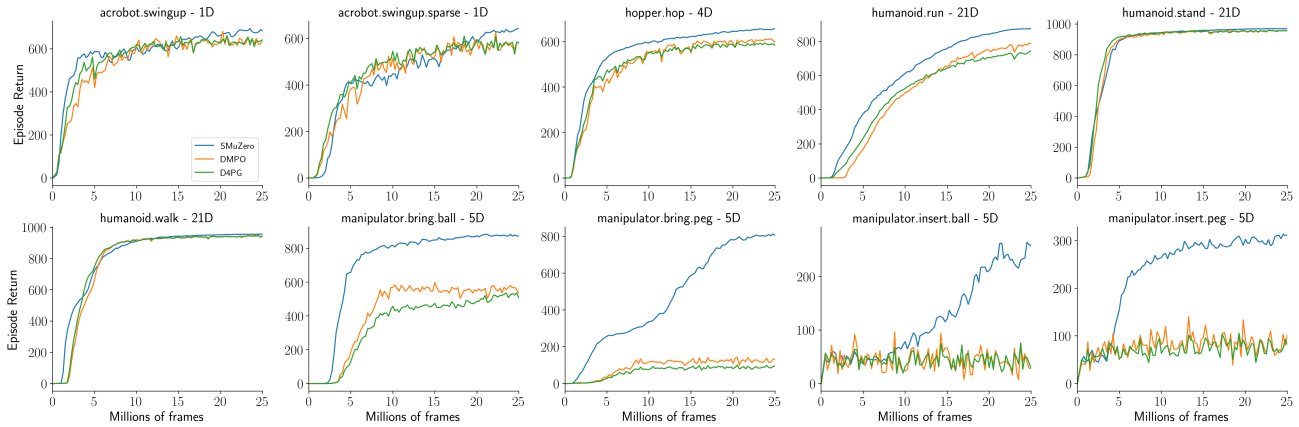

Figure 4. **Results in DM Control Suite Hard and Manipulator tasks.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training compared to DMPO (Hoffman et al., 2020) and D4PG (Barth-Maron et al., 2018). The x-axis shows millions of environment frames, the y-axis mean episode return. Hard tasks as proposed by (Hoffman et al., 2020). Plot titles include the task name and the dimensionality of the action space.

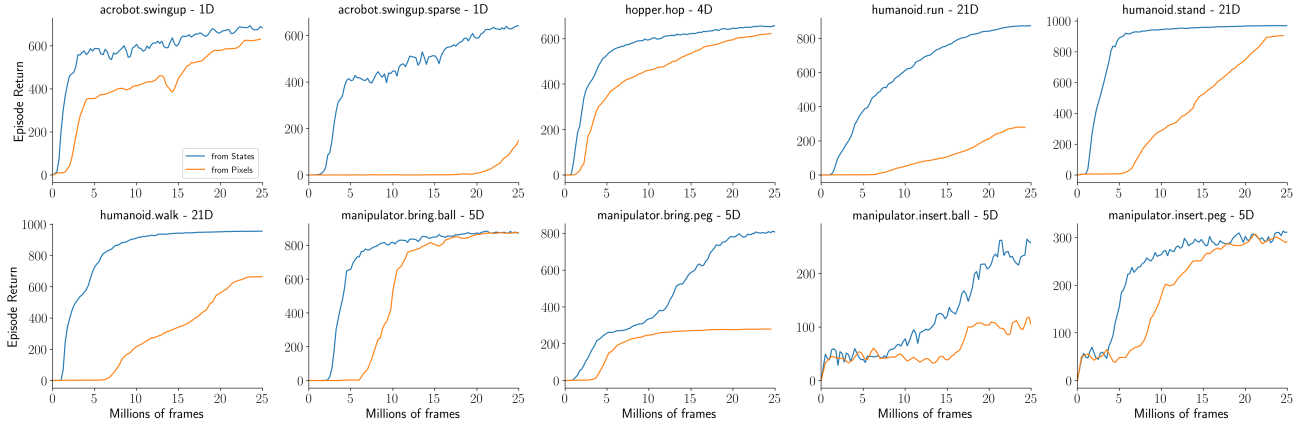

Figure 5. **Results in DM Control Suite Hard and Manipulator tasks of *Sampled MuZero* learning from raw pixel inputs.** Performance of *Sampled MuZero* (3 seeds per experiment) learning from raw pixel inputs throughout training compared to *Sampled MuZero* learning from state inputs. The x-axis shows millions of environment frames, the y-axis mean episode return.

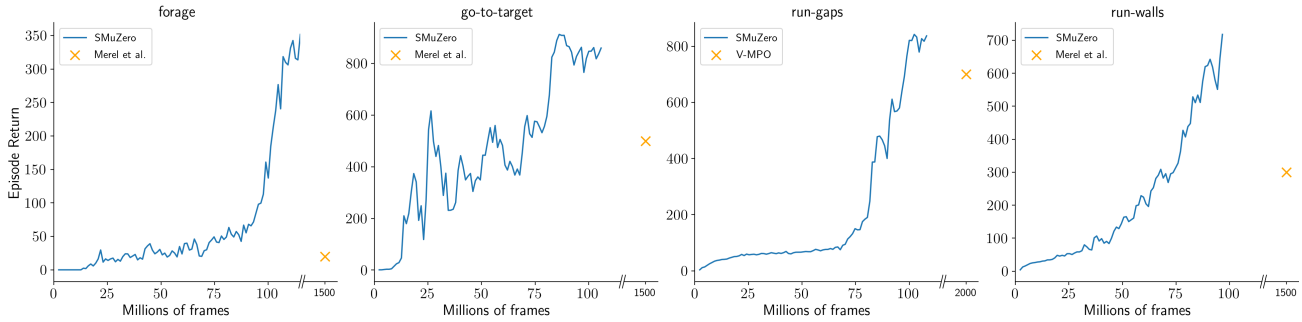

Figure 6. **Results for 56D CMU Humanoid Locomotion tasks.** Performance of *Sampled MuZero* (1 seed per experiment) throughout training in dm_control (Tassa et al., 2020) based CMU humanoid tasks, controlling a humanoid body with 56 action dimensions. *Sampled MuZero* outperforms previously reported results for both forage, go-to-target and run-walls (Merel et al., 2019) as well as run-gaps (Song et al., 2020) while using more than an order of magnitude fewer environment interactions.

To investigate the scalability to more complex action spaces, we also applied *Sampled MuZero* to the dm_control (Tassa et al., 2020) based Locomotion environment. In this set of high-dimensional tasks, the agent must control a humanoid body with 56 action dimensions to accomplish a variety of goals (Figure 6). In all tasks *Sampled MuZero* not only outperformed previously reported results, but it did so using more than an order of magnitude fewer interactions with the environment.

Finally, we investigated the impact on performance of the number of samples in the Appendix (Figure 10). We show that *Sampled MuZero* can learn high dimensional action tasks with as little as $K = 5$ samples. Furthermore, we evaluated the stability of *Sampled MuZero*, both from state inputs and raw pixel inputs, in Figure 11 and Figure 12. We show that *Sampled MuZero*'s performance is overall very reproducible across tasks and number of samples. We also verified the practical importance of using $\hat{\pi}_\beta$ instead of just π in *Sampled MuZero*'s PUCT formula in Figure 13. We find that, as suggested by the theory, it is much more robust to use $\hat{\pi}_\beta$.

6.4. Real-World RL Challenge Benchmark

The real-world Reinforcement Learning (RWRL) Challenge set of benchmark tasks (Dulac-Arnold et al., 2020) is a set of continuous control tasks that aims to capture the aspects of real-world tasks that commonly cause RL algorithms to fail. We used this benchmark to test the robustness of our proposed algorithm to complications such as delays, partial observability or stochasticity. We used the same neural network architecture as for the DeepMind Control Suite with the addition of an LSTM (Hochreiter & Schmidhuber, 1997) to deal with partial observability.

As shown in Table 1, *Sampled MuZero* significantly outperformed baseline algorithms in all three challenge difficulties. We provide full learning curve results in the Appendix (Figure 8).

Agent	Cartpole	Walker	Quadruped	Humanoid
		Easy		
DMPO	464.05	474.44	567.53	1.33
D4PG	482.32	512.44	787.73	102.92
STACX	734.40	487.75	865.80	1.21
SMuZero	**861.05**	**959.83**	**987.20**	**289.36**
		Medium		
DMPO	155.63	64.63	180.30	1.27
D4PG	175.47	75.49	268.01	1.28
STACX	398.71	94.01	466.43	1.18
SMuZero	**516.69**	**448.51**	**946.21**	**108.56**
		Hard		
DMPO	138.06	63.05	144.69	1.40
D4PG	108.20	59.85	280.75	1.27
STACX	135.26	58.11	**351.56**	1.26
SMuZero	**244.71**	**71.16**	348.09	1.19

Table 1. ***Sampled MuZero*** **results for the Real-Word RL benchmark** (RWRL). In RWRL, for each task there is an easy, medium and hard variation of increasing difficulty. DMPO and D4PG results from (Dulac-Arnold et al., 2020), STACX from (Zahavy et al., 2020). *Sampled MuZero* (3 seeds per experiment) shows strong performance throughout, especially on the highest dimensional Humanoid tasks.

7. Conclusions

In this paper we introduced a unified framework for learning and planning in discrete, continuous and structured complex action spaces. Our approach is based upon a simple principle of sampling actions. By careful book-keeping we have shown how one may take account of the sampling process during policy improvement and policy evaluation. In principle, the same sample-based strategy could be applied to a variety of other reinforcement algorithms in which the policy is updated by, or approximated by, an action-independent improvement step. Concretely, we have focused upon applying our framework to the model-based planning algorithm of *MuZero*, resulting in our new algorithm *Sampled MuZero*. Our empirical results show that the idea is both general, succeeding across a wide variety of discrete and continuous benchmark environments, and robust, scaling gracefully down to small numbers of samples. These results suggest that the ideas introduced in this paper may also be effective in larger scale applications where it is not feasible to enumerate the action space.

Acknowledgements

We would like to thank Jost Tobias Springenberg for providing very detailed feedback and constructive suggestions.

References

Abdolmaleki, A., Springenberg, J. T., Degrave, J., Bohez, S., Tassa, Y., Belov, D., Heess, N., and Riedmiller, M. Relative entropy regularized policy iteration, 2018.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization, 2016.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., TB, D., Muldal, A., Heess, N., and Lillicrap, T. Distributed Distributional Deterministic Policy Gradients, 2018.

Bhardwaj, M., Handa, A., Fox, D., and Boots, B. Informa-

tion theoretic model predictive q-learning. In *Learning for Dynamics and Control*, pp. 840–850. PMLR, 2020.

Brown, N. and Sandholm, T. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Byravan, A., Springenberg, J. T., Abdolmaleki, A., Hafner, R., Neunert, M., Lampe, T., Siegel, N., Heess, N., and Riedmiller, M. Imagined value gradients: Model-based policy optimization with transferable latent dynamics models. In *Conference on Robot Learning*, pp. 566–589. PMLR, 2020.

Chaslot, G., Winands, M., Herik, H., Uiterwijk, J., and Bouzy, B. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, 04: 343–357, 11 2008. doi: 10.1142/S1793005708001094.

Coulom, R. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pp. 72–83. Springer, 2006.

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. Deep reinforcement learning in large discrete action spaces, 2016.

Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., and Hester, T. An empirical investigation of the challenges of real-world reinforcement learning, 2020.

Ghosh, D., Machado, M. C., and Roux, N. L. An operator view of policy gradient methods, 2020.

Grill, J.-B., Altché, F., Tang, Y., Hubert, T., Valko, M., Antonoglou, I., and Munos, R. Monte-carlo tree search as regularized policy optimization, 2020.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016. URL http://arxiv.org/abs/1603.05027.

Hennigan, T., Cai, T., Norman, T., and Babuschkin, I. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Hoffman, M., Shahriari, B., Aslanides, J., Barth-Maron, G., Behbahani, F., Norman, T., Abdolmaleki, A., Cassirer, A., Yang, F., Baumli, K., Henderson, S., Novikov, A., Colmenarejo, S. G., Cabi, S., Gulcehre, C., Paine, T. L., Cowie, A., Wang, Z., Piot, B., and de Freitas, N. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020. URL https://arxiv.org/abs/2006.00979.

Kearns, M., Mansour, Y., and Ng, A. Y. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pp. 1324–1331, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Koul, A., Kumar, V. V., Fern, A., and Majumdar, S. Dream and search to control: Latent space planning for continuous control, 2020.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Loshchilov, I. and Hutter, F. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. URL http://arxiv.org/abs/1711.05101.

Merel, J., Ahuja, A., Pham, V., Tunyasuvunakool, S., Liu, S., Tirumala, D., Heess, N., and Wayne, G. Hierarchical visuomotor control of humanoids. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BJfYvo09Y7.

Moerland, T. M., Broekens, J., Plaat, A., and Jonker, C. M. A0C: AlphaZero in continuous action space, 2018.

Moravčík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

Peng, X. B., Kumar, A., Zhang, G., and Levine, S. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

Piché, A., Thomas, V., Ibrahim, C., Bengio, Y., and Pal, C. Probabilistic planning with sequential monte carlo methods. In *International Conference on Learning Representations*, 2018.

Rubin, D. B. The calculation of posterior distributions by data augmentation: Comment: A noniterative sampling/importance resampling alternative to the data augmentation algorithm for creating a few imputations when fractions of missing information are modest: The sir algorithm. *Journal of the American Statistical Association*, 82(398):543–546, 1987. ISSN 01621459. URL http://www.jstor.org/stable/2289460.

Rust, J. Using randomization to break the curse of dimensionality. *Econometrica*, 65(3):487–516, 1997. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/2171751.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. In *International Conference on Learning Representations*, Puerto Rico, 2016.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T. P., and Silver, D. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 588(7839):604–609, 2020.

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

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

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In Xing, E. P. and Jebara, T. (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 387–395, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/silver14.html.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

Song, H. F., Abdolmaleki, A., Springenberg, J. T., Clark, A., Soyer, H., Rae, J. W., Noury, S., Ahuja, A., Liu, S., Tirumala, D., Heess, N., Belov, D., Riedmiller, M., and Botvinick, M. M. V-MPO: On-policy maximum a posteriori policy optimization for discrete and continuous control. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylOlp4FvH.

Springenberg, J. T., Heess, N., Mankowitz, D., Merel, J., Byravan, A., Abdolmaleki, A., Kay, J., Degrave, J., Schrittwieser, J., Tassa, Y., et al. Local search for policy iteration in continuous control. *arXiv preprint arXiv:2010.05545*, 2020.

Sutton, R. S. Learning to predict by the methods of temporal differences. 1988. URL http://incompleteideas.net/papers/sutton-88-with-erratum.pdf.

Tang, Y. and Agrawal, S. Discretizing continuous action space for on-policy optimization, 2020.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T., and Riedmiller, M. Deepmind control suite, 2018.

Tassa, Y., Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., and Heess, N. dm_control: Software and tasks for continuous control. *arXiv preprint arXiv:2006.12983*, 2020.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. pp. 5026–5033, 10 2012. ISBN 978-1-4673-1737-5. doi: 10.1109/IROS.2012.6386109.

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

Yang, X., Duvaud, W., and Wei, P. Continuous control for searching and planning with a learned model, 2020.

Zahavy, T., Xu, Z., Veeriah, V., Hessel, M., Oh, J., van Hasselt, H. P., Silver, D., and Singh, S. A self-tuning actor-critic algorithm. *Advances in Neural Information Processing Systems*, 33, 2020.

A. DeepMind Control Suite and Real-World RL Experiments

For the continuous control experiments where the input is 1 dimensional (as opposed to 2 dimensional image inputs in board games and Atari as used by *MuZero*), we used a variation of the *MuZero* model architecture in which all convolutions are replaced by fully connected layers.

The representation function processed the input via an *input block* composed of a linear layer, followed by a Layer Normalisation and a *tanh* activation. The resulting embedding was then processed by a ResNet v2 style pre-activation residual tower (He et al., 2016) coupled with Layer Normalisation (Ba et al., 2016) and Rectified Linear Unit (ReLU) activations. We used 10 blocks, each block containing 2 layers with a hidden size of 512.

For the Real-World RL experiments, we additionally inserted an LSTM module (Hochreiter & Schmidhuber, 1997) in the representation function between the *input block* and the residual tower to deal with partial observability. We trained the LSTM using truncated backpropagation through time for 8 steps, initialised from LSTM states stored during acting, each step having the last 4 observations concatenated together, for an effective unroll step of 32 steps.

The dynamics function processed the action via an *action block* composed of a linear layer, followed by a Layer Normalisation and a *ReLU* activation. The action embedding was then added to the dynamics function's input embedding and then processed by a residual tower using the same architecture as the residual tower for the representation function.

The reward and value predictions used the categorical representation introduced in *MuZero* (Schrittwieser et al., 2020). We used 51 bins for both the value and the reward predictions with the value being able to represent values between $[-150.0, 150.0]$ and the reward being able to represent values between $[-1.0, 1.0]$. We used n-step bootstrapping with $n = 5$ and a discount of 0.99 consistent with Acme (Hoffman et al., 2020).

We used the factored policy representation introduced by (Tang & Agrawal, 2020) representing each dimension by a categorical distribution over $B = 7$ bins for the policy prediction.

To implement the network, we used the modules provided by the Haiku neural network library (Hennigan et al., 2020).

We used the Adam optimiser (Kingma & Ba, 2015) with decoupled weight decay (Loshchilov & Hutter, 2017) for training. We used a weight decay scale of $2 \cdot 10^{-5}$, a batch size of 1024 an initial learning rate of 10^{-4}, decayed to 0 over 1 million training batches using a cosine schedule:

$$\text{lr} = \text{lr}_{\text{init}} \frac{1}{2} \left(1 + \cos \pi \frac{\text{step}}{\text{max\_steps}} \right)$$

where $\text{lr}_{\text{init}} = 10^{-4}$ and $\text{max\_steps} = 10^6$.

For replay, we keep a buffer of the most recent 2000 sequences, splitting episodes into subsequences of length up to 500. Samples are drawn from the replay buffer according to prioritised replay (Schaul et al., 2016) using the same priority and hyperparameters as in *MuZero*.

We trained *Sampled MuZero* using $K = 20$ samples and a search budget of 50 simulations per move. At the root of the search tree only, we evaluated all sampled actions before the start of the search and used those to initialise the $Q(s, a)$ quantities in the PUCT formula (Appendix D). We evaluated *Sampled MuZero*'s network checkpoints throughout training playing 100 games with a search budget of 50 simulations per move and picked the move with the highest number of visits to act, consistent with previous work.

We used Acme (Hoffman et al., 2020) to produce the results for DMPO (Hoffman et al., 2020) and D4PG (Barth-Maron et al., 2018). Compared to Acme, we used bigger networks (Policy Network layers = (512, 512, 256, 128), Critic Network Layers = (1024, 1024, 512, 256)) and a bigger batch size of 1024 for better comparison. Each task was run with three seeds.

We provide full learning curve results on the DeepMind Control Suite (Figure 7) and Real-World RL (Figure 8) tasks.

A.1. Gaussian policy parameterisation

Even though a categorical policy representation was used to compute the main results, *Sampled MuZero* can also be applied working directly with continuous actions. Figure 9 shows results on the hard and manipulator tasks when the policy prediction is parameterised by a Gaussian distribution.

The performance is similar across almost all tasks but we found that Gaussian distributions are harder to optimise than their categorical counterpart and that using entropy regularisation was useful to produce better results (we used a coefficient of 5e-3). It is possible that these results could be improved with better regularisation schemes such as constraining the deviation of the mean and standard deviation as in the MPO (Abdolmaleki et al., 2018) algorithm. In contrast, we did not need to add any regularisation to train the categorical distribution.

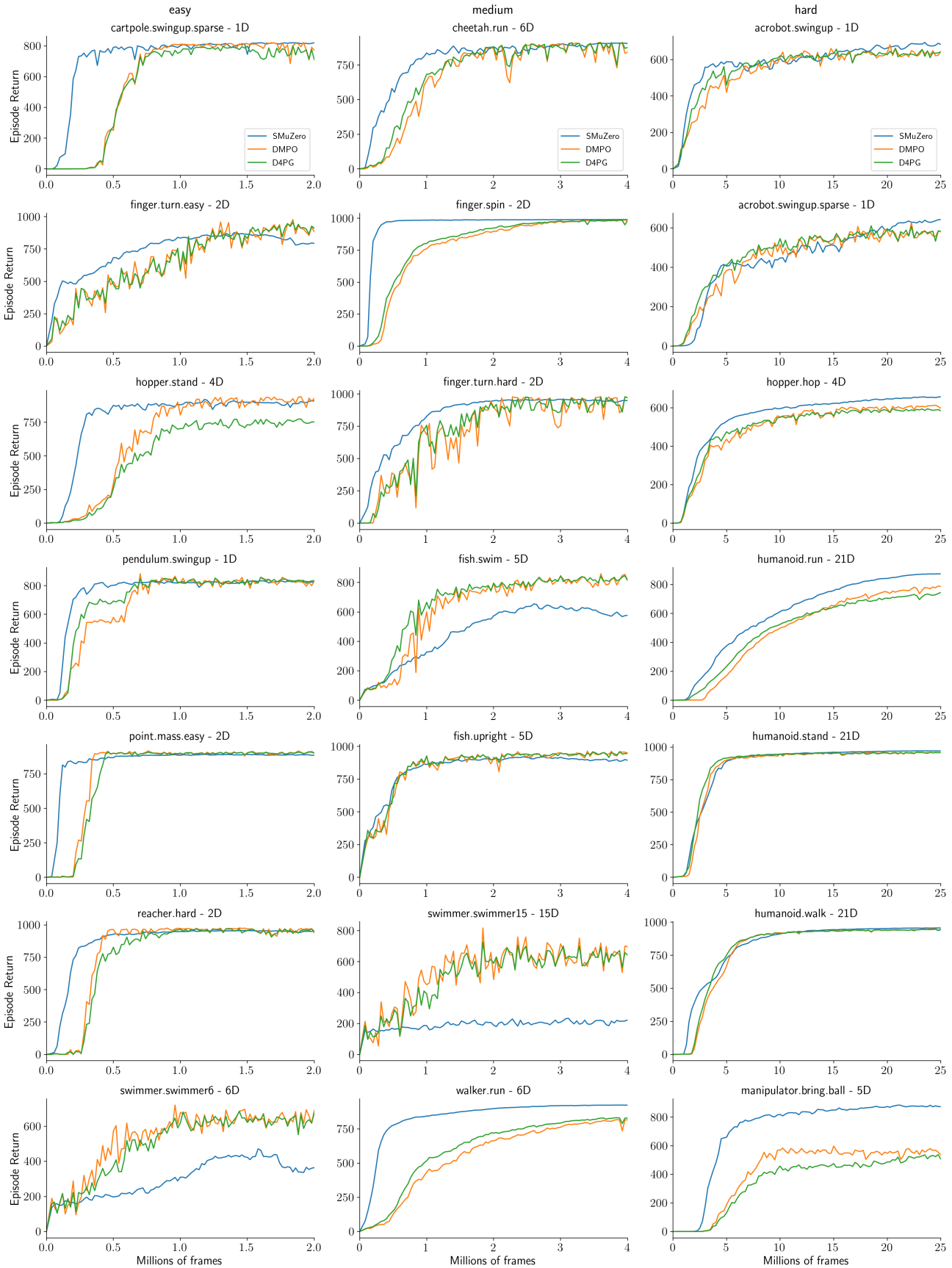

Figure 7. **Results in DM Control Suite tasks.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training compared to DMPO (Hoffman et al., 2020) and D4PG (Barth-Maron et al., 2018). The x-axis shows millions of environment frames, the y-axis mean episode return. Tasks are grouped into easy, medium and hard as proposed by (Hoffman et al., 2020). Plot titles include the task name and the dimensionality of the action space.

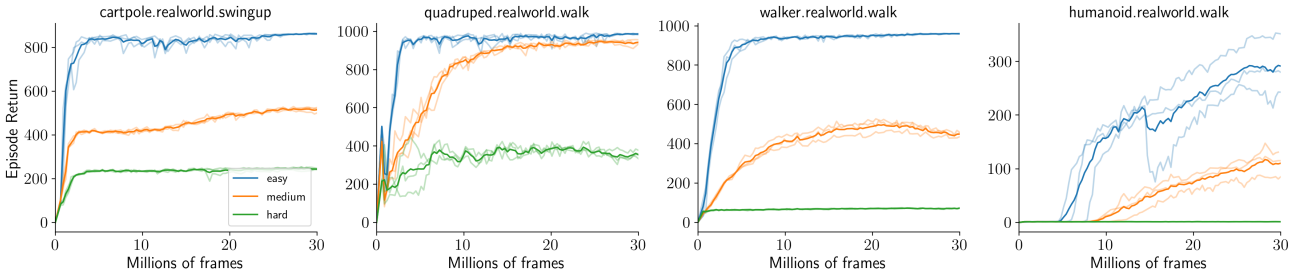

Figure 8. ***Sampled MuZero*** **results for the Real-Word RL benchmark.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training on the easy, medium and hard variations of difficulty. The x-axis shows millions of environment frames, the y-axis mean episode return. Tasks are grouped into easy, medium and hard. Plot titles include the task name.

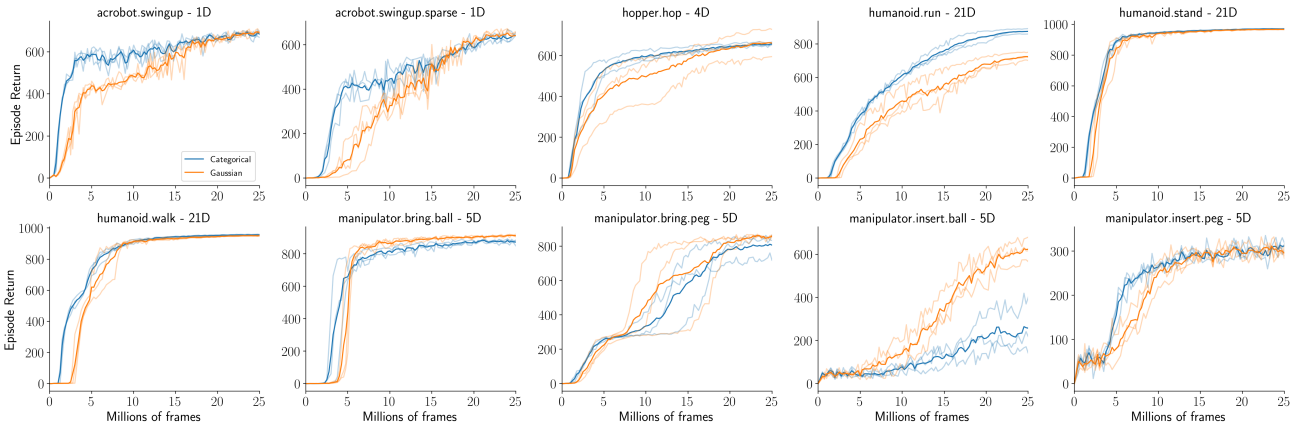

Figure 9. **Comparison between a Categorical and Gaussian parameterisation of the policy prediction for** ***Sampled MuZero*****.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training on the DM Control Hard and Manipulator tasks.

Tasks	Dreamer	SMuZero
acrobot.swingup	365.26	**417.52**
cartpole.balance	**979.56**	**984.86**
cartpole.balance_sparse	941.84	**998.14**
cartpole.swingup	833.66	**868.87**
cartpole.swingup_sparse	812.22	**846.91**
cheetah.run	894.56	**914.39**
ball_in_cup.catch	962.48	**977.38**
finger.spin	498.88	**986.38**
finger.turn_easy	825.86	**972.53**
finger.turn_hard	891.38	**963.07**
hopper.hop	368.97	**528.24**
hopper.stand	**923.72**	**926.50**
pendulum.swingup	**833.00**	**837.76**
quadruped.run	888.39	**923.54**
quadruped.walk	**931.61**	**933.77**
reacher.easy	935.08	**982.26**
reacher.hard	817.05	**971.53**
walker.run	824.67	**931.06**
walker.stand	**977.99**	**987.79**
walker.walk	961.67	**975.46**

Table 2. **Performance of *Sampled MuZero* compared to the Dreamer agent.** *Sampled MuZero* equals or outperforms the Dreamer agent in all tasks. Dreamer results from (Hafner et al., 2019).

A.2. *Sampled MuZero* from Pixels

In addition to *Sampled MuZero*'s results on the hard and manipulator tasks when learning from raw pixel inputs, we compared *Sampled MuZero* to the Dreamer agent (Hafner et al., 2019) in Table 2. We used the 20 tasks and the 5 million environment steps experimental setup defined by (Hafner et al., 2019). *Sampled MuZero* equalled or surpassed the Dreamer agent's performance in all 20 tasks, without any action repeat (Dreamer uses an action repeat of 2), observation reconstruction, or any hyperparameter re-tuning.

A.3. Ablation on the number of samples

We trained multiple instances of *Sampled MuZero* with varying number of action samples $K \in \{3, 5, 10, 20, 40\}$ on the *humanoid.run* task for which the action is 21 dimensional. We ran six seeds for each instance. Surprisingly $K = 3$ is already sufficient to learn a good policy and performance does not seem to be improved by sampling more than $K = 10$ samples (see Figure 10).

A.4. Reproducibility

In order to evaluate the reproducibility of *Sampled MuZero* from state inputs and raw pixel inputs, we show the individual performance of 3 seeds on the hard and manipulator tasks in Figure 11. Overall, the variation in performance across seeds is minimal.

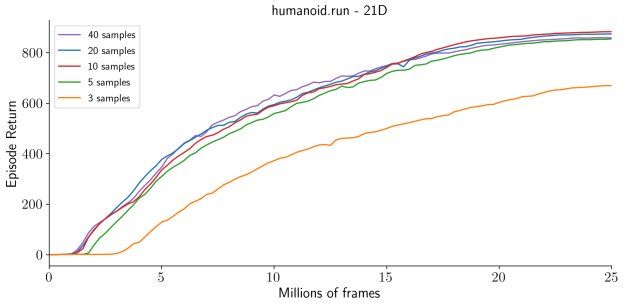

Figure 10. **Performance of *Sampled MuZero* with different number of samples on the *humanoid.run* task.** Performance of *Sampled MuZero* (6 seeds per experiment) throughout training on the DM Control Humanoid Run task.

In addition, we show the individual performance of 6 seeds when sampling $K = 3, 5, 10, 20, 40$ actions on the *humanoid.run* task. We observe that even when the number of samples is small, performance stays very reproducible across runs.

A.5. Ablation on using $\hat{\pi}_\beta$ vs π

We evaluated the practical importance of using $\hat{\pi}_\beta = \hat{\beta}/\beta\pi$ instead of just π in *Sampled MuZero*'s PUCT formula and ran experiments on the *humanoid.run* task.

We expect that as the number of samples increases, the difference will go away as $\lim_{K\to\infty} \hat{\pi}_\beta = \lim_{K\to\infty} \hat{\beta}/\beta\pi = \pi$. We therefore looked at the difference in performance when drawing $K = 5$ or $K = 20$ samples.

Furthermore, evaluating the Q values of all sampled actions at the root of the search tree before the start of the search puts more emphasis on the values and less on the prior in the PUCT formula. We therefore also show the difference in performance with and without Q evaluations (*no Q* in the figure).

The experiments in Figure 13 confirm that it is much better to use $\hat{\pi}_\beta$ when the number of samples is small and not evaluating the Q values. The performance drop of using π is attenuated by evaluating the Q values at the root of the search tree, but it is still better to use $\hat{\pi}_\beta$ even in that case.

B. Go Experiments

For the Go experiments, we mostly used the same neural network architecture, optimisation and hyperparameters used by *MuZero* (Schrittwieser et al., 2020) with the following differences. Instead of using the outcome of the game to train the value network, we used n-step bootstrapping with

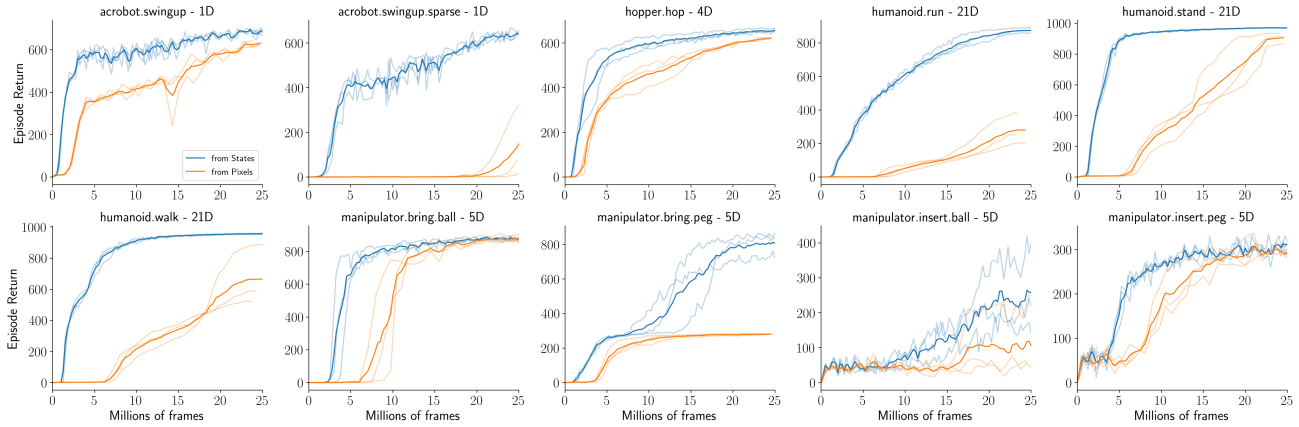

Figure 11. **Reproducibility of *Sampled MuZero* from state and raw pixel inputs on the hard and manipulator tasks.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training on the DM Control Humanoid Run task.

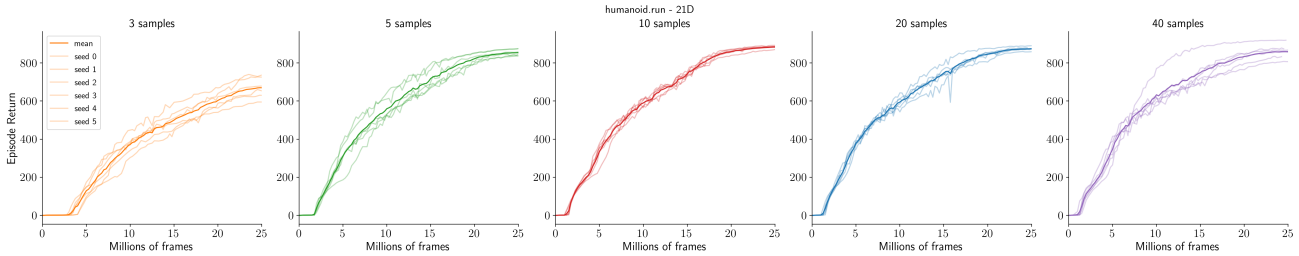

Figure 12. **Reproducibility of *Sampled MuZero* on the *humanoid.run* task with 3, 5, 10, 20 and 40 action samples.** Performance of *Sampled MuZero* (6 seeds per experiment) throughout training on the DM Control Humanoid Run task.

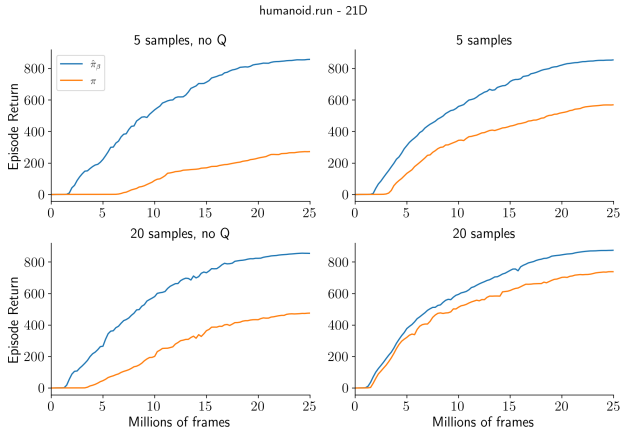

Figure 13. **Performance of *Sampled MuZero* using $\hat{\pi}_\beta$ vs π on the *humanoid.run* task.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training on the DM Control Humanoid Run task evaluated with $K = 5$ or $K = 20$ samples and with or without (no Q) evaluating the Q values of all sampled actions at the root of the search tree. It is much more robust to use $\hat{\pi}_\beta$ over π in *Sampled MuZero*.

$n = 25$ where the value used to bootstrap was the averaged predictions of a target network applied to 4 consecutive states at indices $n + i$ for $i \in [0, 3]$. We averaged multiple consecutive target network value predictions due to the alternation of perspective for value prediction in two-player games; using the average of multiple estimates ensures that learning is based on the estimates for both sides. We observed that this reduced value overfitting and allowed us to train *MuZero* while generating less data. In addition, we used a search budget of 400 simulations per move instead of 800 in order to use less computation.

We evaluated the network checkpoints of *MuZero* and *Sampled MuZero* throughout training playing 100 matches with a search budget of 800 simulations per move. We anchored the Elo scale to a final *MuZero* baseline performance of 2000 Elo.

C. Atari Experiments

For the Atari experiments, we used the same architecture, optimisation and hyperparameters used by *MuZero* (Schrittwieser et al., 2020).

We evaluated the network checkpoints of *MuZero* and *Sampled MuZero* throughout training playing 100 games with a

search budget of 50 simulations per move.

D. Search

The full PUCT formula used in *Sampled MuZero* is:

$$arg\,max_a Q(s,a) + c(s) \cdot (\frac{\hat{\beta}}{\beta}\pi)(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

where

$$c(s) = c_1 + \log\frac{1 + c_2 + \sum_b N(s,b)}{c_2}$$

with $c_1 = 1.25$ and $c_2 = 19652$ in the experiments for this paper. Note that at visit counts $N(s) = \sum_b N(s,b) \ll c_2$, the log in the exploration term is approximately 0 and the formula can be written:

$$arg\,max_a Q(s,a) + c_1 \cdot (\frac{\hat{\beta}}{\beta}\pi)(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

E. Sample-based Policy Improvement and Evaluation Proofs

Lemma. $\hat{Z}_\beta$ and Z are linked by:

$$\lim_{K\to\infty} \hat{Z}_\beta = Z$$

Proof $\hat{Z}_\beta(s)$ is defined such that $\sum_{a\in\mathcal{A}}(\hat{\beta}/\beta)(a|s)f(s,a,\hat{Z}_\beta(s)) = 1$.

Therefore

$$\begin{aligned} 1 &= \lim_{K\to\infty}\sum_{a\in\mathcal{A}}(\hat{\beta}/\beta)(a|s)f(s,a,\hat{Z}_\beta(s)) \\ &= \lim_{K\to\infty}\sum_{a\in\mathcal{A}} f(s,a,\hat{Z}_\beta(s)) \end{aligned}$$

where we used $\lim_{K\to\infty}\hat{\beta} = \beta$ to go from line 1 to 2.

We therefore have

$$\lim_{K\to\infty}\sum_{a\in\mathcal{A}} f(s,a,\hat{Z}_\beta(s)) = 1 = \sum_{a\in\mathcal{A}} f(s,a,Z(s))$$

which shows by the uniqueness of Z that $\lim_{K\to\infty}\hat{Z}_\beta = Z$.

Theorem. For a given random variable X, we have

$$\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s] = \lim_{K\to\infty}\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a|s)X(s,a)$$

Furthermore, $\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a|s)X(s,a)$ is approximately normally distributed around $\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s]$ as $K\to\infty$:

$$\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a|s)X(s,a) \sim \mathcal{N}(\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s], \frac{\sigma^2}{K})$$

where $\sigma^2 = Var_{a\sim\beta}[\frac{f(s,a,Z(s))}{\beta}X(s,a)|s]$.

Proof. We have

$$\begin{aligned} &\mathbb{E}_{a\sim\mathcal{I}\pi}[X(s,a)|s] \\ &= \mathbb{E}_{a\sim\beta}[(\mathcal{I}\pi/\beta)(a|s)X(s,a)|s] \\ &= \mathbb{E}_{a\sim\beta}[f(s,a,Z(s))/\beta(a|s)X(s,a)|s] \\ &= \lim_{K\to\infty}\sum_{a\in\mathcal{A}}(\hat{\beta}/\beta)(a|s)f(s,a,Z(s))X(s,a) \\ &= \lim_{K\to\infty}\sum_{a\in\mathcal{A}}(\hat{\beta}/\beta)(a|s)f(s,a,\hat{Z}_\beta(s))X(s,a) \\ &= \lim_{K\to\infty}\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a|s)X(s,a) \end{aligned}$$

where we used the law of large numbers to go from line 2 to 3, replacing the expectation with the limit of a sum, and the lemma to go from line 3 to 4.

Using the central limit theorem from line 2, we can also show that as $K\to\infty$,

$$\sum_{a\in\mathcal{A}}(\hat{\beta}/\beta)(a|s)f(s,a,Z(s))X(s,a) \to \mathcal{N}(\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s], \frac{\sigma^2}{K})$$

in distribution with $\sigma^2 = Var_{a\sim\beta}[\frac{f(s,a,Z(s))}{\beta}X(s,a)|s]$.

Making the approximation of swapping in $\hat{Z}_\beta$ for Z based on the lemma, we obtain that as $K\to\infty$:

$$\sum_{a\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(a|s)X(s,a) \sim \mathcal{N}(\mathbb{E}_{a\sim\mathcal{I}\pi}[X|s], \frac{\sigma^2}{K})$$

Corollary. The sample-based policy improvement operator converges in distribution to the true policy improvement operator:

$$\lim_{K\to\infty}\hat{\mathcal{I}}_\beta\pi = \mathcal{I}\pi$$

Furthermore, the sample-based policy improvement operator is approximately normally distributed around the true policy improvement operator as $K\to\infty$:

$$\hat{\mathcal{I}}_\beta\pi(a|s) \sim \mathcal{N}(\mathcal{I}\pi(a|s), \frac{\sigma^2}{K})$$

where $\sigma^2 = Var_{a\sim\beta}[\frac{f(s,a,Z(s))}{\beta}\mathbb{1}(a)|s]$.

Proof. We obtain the corollary by using $X(s,a) = \mathbb{1}(a)$ in conjunction with $\mathcal{I}\pi(a|s) = \mathbb{E}_{a\sim\mathcal{I}\pi}[\mathbb{1}(a)|s]$ and $\hat{\mathcal{I}}_\beta\pi(a|s) = \sum_{b\in\mathcal{A}}\hat{\mathcal{I}}_\beta\pi(s,b)\mathbb{1}(a)$

F. The *MuZero* Policy Improvement Operator

Recent work (Grill et al., 2020) showed that *MuZero*'s visit count distribution was tracking the solution $\bar{\pi}$ of a regularised policy optimisation problem:

$$\bar{\pi} = arg\,max_\Pi Q^T\Pi - \lambda_N KL(\pi,\Pi)$$

where KL is the Kullback–Leibler divergence and λ_N is a constant dependent on c and the total number N of simulations.

$\bar{\pi}$ can be computed analytically:

$$\bar{\pi}(a|s) = \lambda_N \frac{\pi(s,a)}{Z(s) - Q(s,a)}$$

where $Z(s)$ is a normalising factor such that $\forall a \in \mathcal{A}, \bar{\pi}(a|s) \geq 0$ and $\sum_{a \in \mathcal{A}} \bar{\pi}(a|s) = 1$.

In other words, using the terminology introduced in Section 4, *MuZero*'s policy improvement can be approximately written:

$$\mathcal{I}\pi(a|s) \approx f(s, a, Z(s))$$

where

$$f(s, a, Z(s)) = \lambda_N \frac{\pi(a|s)}{Z(s) - Q(s,a)}$$

and is therefore action-independent.

Let's consider the visit count distribution $\mathcal{I}\hat{\pi}_\beta$ obtained by searching using prior $\hat{\pi}_\beta = \hat{\beta}/\beta\pi$.

Using (Grill et al., 2020), we can write:

$$\begin{aligned}
\mathcal{I}\hat{\pi}_\beta(s,a) &\approx \lambda_N \frac{\hat{\pi}_\beta(a|s)}{\hat{Z}_\beta(s) - Q(s,a)} \\
&= \lambda_N \frac{(\hat{\beta}/\beta\pi)(a|s)}{\hat{Z}_\beta(s) - Q(s,a)} \\
&= (\hat{\beta}/\beta)(a|s) f(s, a, \hat{Z}_\beta(s))
\end{aligned}$$

where $\hat{Z}_\beta(s)$ is such that $\forall a \in \mathcal{A}, \mathcal{I}\hat{\pi}_\beta(s,a) \geq 0$ and $\sum_{a \in \mathcal{A}} \mathcal{I}\hat{\pi}_\beta(s,a) = 1$.

This shows that $\mathcal{I}\hat{\pi}_\beta$ is the action-independent sample-based policy improvement operator associated to $\mathcal{I}\pi$.