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.

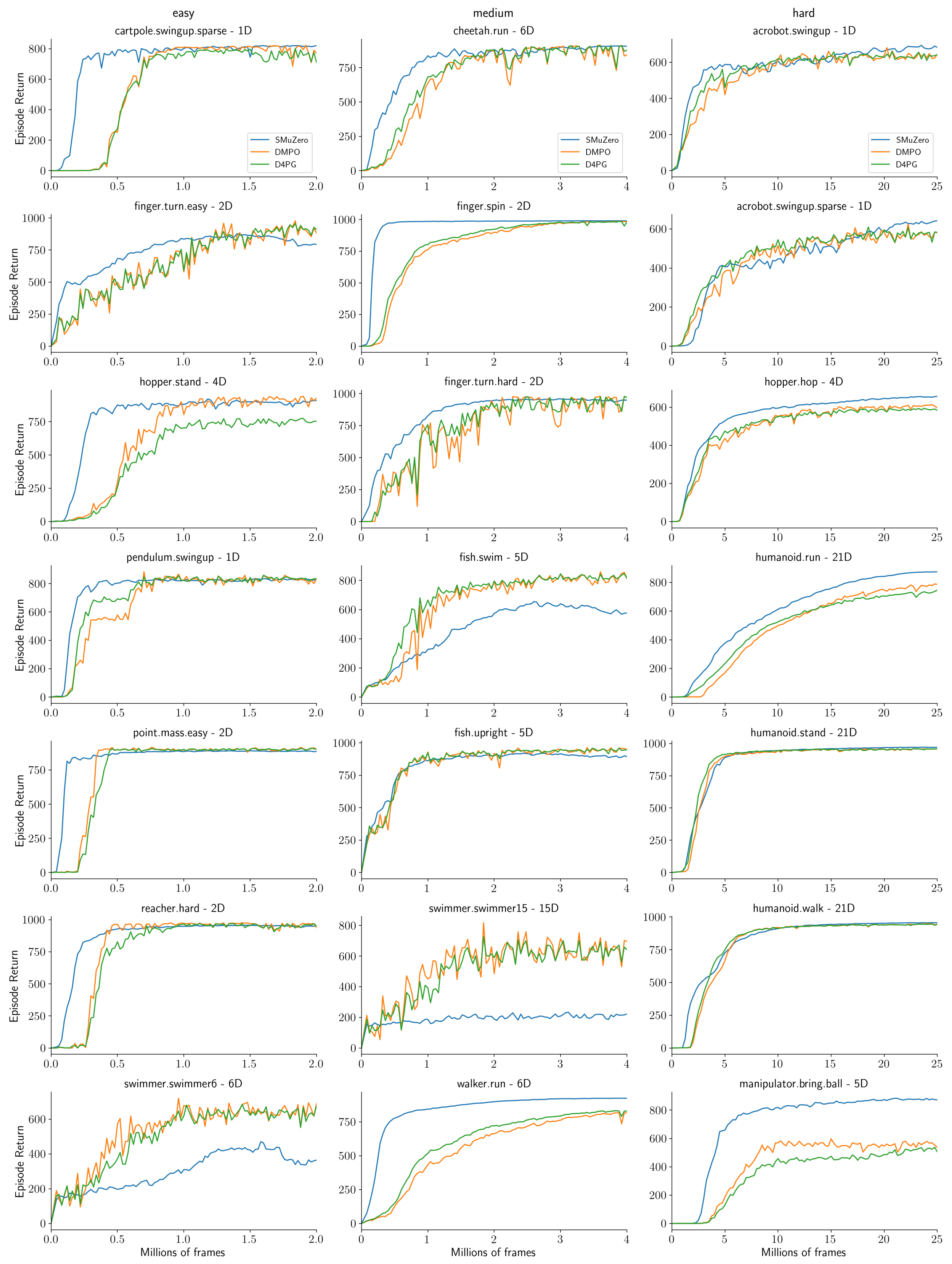

Figure 7. **Results in DM Control Suite tasks.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training compared to DMPO (Hoffman et al., 2020) and D4PG (Barth-Maron et al., 2018). The x-axis shows millions of environment frames, the y-axis mean episode return. Tasks are grouped 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.

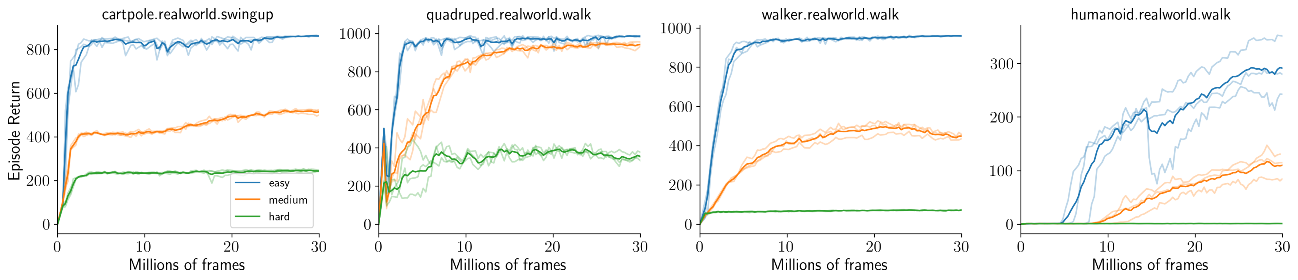

Figure 8. ***Sampled MuZero*** **results for the Real-Word RL benchmark.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training on the easy, medium and hard variations of difficulty. The x-axis shows millions of environment frames, the y-axis mean episode return. Tasks are grouped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

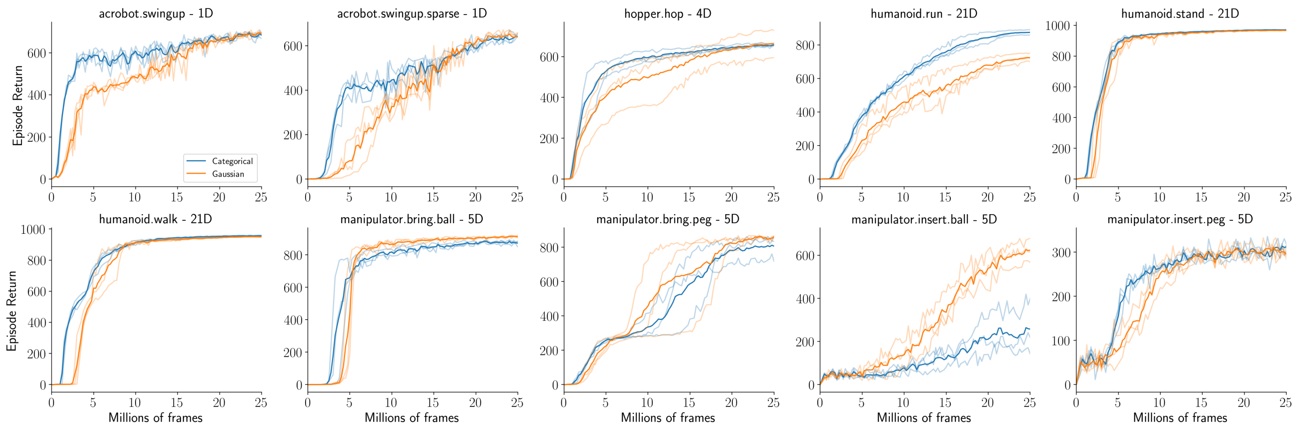

Figure 9. **Comparison between a Categorical and Gaussian parameterisation of the policy prediction for *Sampled MuZero*.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training on the DM Control Hard and Manipulator tasks.

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	**927.13**
quadruped.walk	931.61	**959.03**
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.

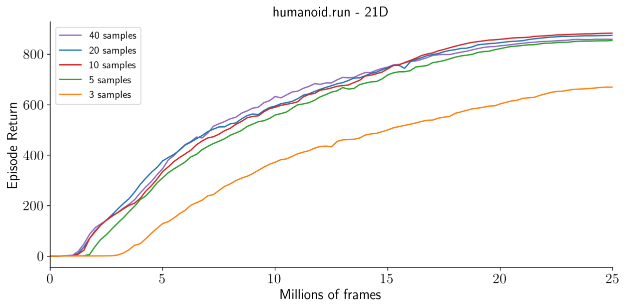

Figure 10. **Performance of *Sampled MuZero* with different number of samples on the *humanoid.run* task.** Performance of *Sampled MuZero* (6 seeds per experiment) throughout training on the DM Control Humanoid Run task.

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

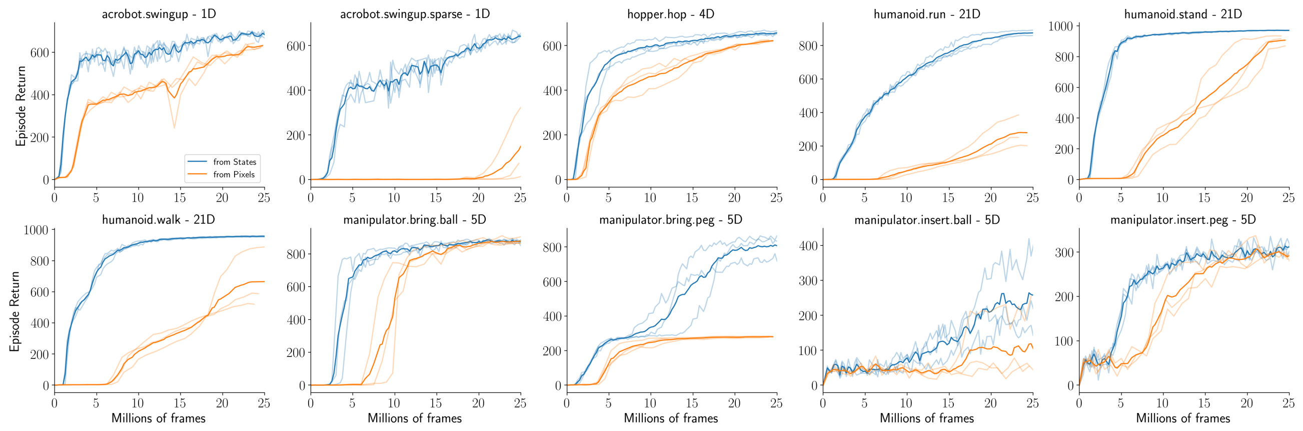

Figure 11. **Reproducibility of *Sampled MuZero* from state and raw pixel inputs on the hard and manipulator tasks.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training on the DM Control Humanoid Run task.

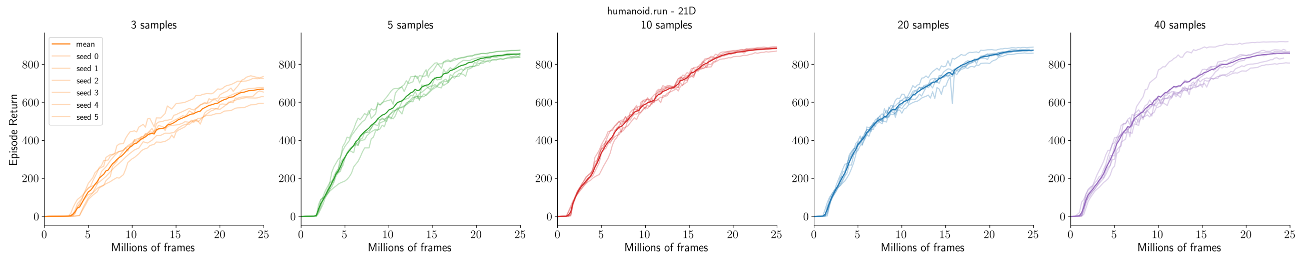

Figure 12. **Reproducibility of *Sampled MuZero* on the *humanoid.run* task with 3, 5, 10, 20 and 40 action samples.** Performance of *Sampled MuZero* (6 seeds per experiment) throughout training on the DM Control Humanoid Run task.

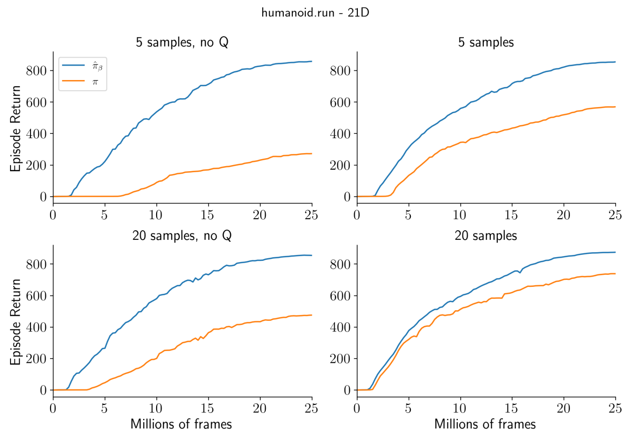

Figure 13. **Performance of *Sampled MuZero* using $\hat{\pi}_\beta$ vs π on the *humanoid.run* task.** Performance of *Sampled MuZero* (3 seeds per experiment) throughout training on the DM Control Humanoid Run task evaluated with $K = 5$ or $K = 20$ samples and with or without (no Q) evaluating the 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 $\mathcal{I}\pi$.