Muesli Supplement

Content

A. Stochastic estimation details

In the policy-gradient term in Eq. 10, we clip the importance weight $\frac{\pi(A|s)}{\pi_b(A|s)}$ to be from $[0, 1]$. The importance weight clipping introduces a bias. To correct for it, we use β-LOO action-dependent baselines (Gruslys et al., 2018).

Although the β-LOO action-dependent baselines were not significant in the Muesli results, the β-LOO was helpful for the policy gradients with the TRPO penalty (Figure 16).

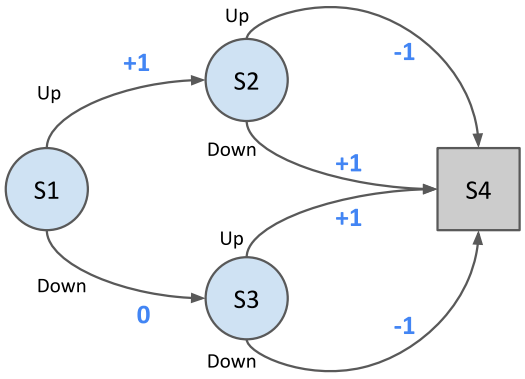

Figure 9. The episodic MDP from Figure 2, reproduced here for an easier reference. State 1 is the initial state. State 4 is terminal. The discount is 1.

B. The illustrative MDP example

Here we will analyze the values and the optimal policy for the MDP from Figure 9, when using the identical state representation $\phi(s) = \varnothing$ in all states. With the state representation $\phi(s)$, the policy is restricted to be the same in all states. Let's denote the probability of the up action by p.

Given the policy $p = \pi(up|\phi(s))$, the following are the values of the different states:

$$v_\pi(3) = p + (1-p)(-1) = 2p - 1 \tag{15}$$
$$v_\pi(2) = p \cdot (-1) + (1-p) = -2p + 1 \tag{16}$$
$$v_\pi(1) = p \cdot (1 + v_\pi(2)) + (1-p)v_\pi(3) \tag{17}$$
$$= -4p^2 + 5p - 1. \tag{18}$$

Finding the optimal policy. Our objective is to maximize the value of the initial state. That means maximizing $v_\pi(1)$. We can find the maximum by looking at the derivatives. The derivative of $v_\pi(1)$ with respect to the policy parameter is:

$$\frac{dv_\pi(1)}{dp} = -8p + 5. \tag{19}$$

The second derivative is negative, so the maximum of $v_\pi(1)$ is at the point where the first derivative is zero. We conclude that the maximum of $v_\pi(1)$ is at $p^* = \frac{5}{8}$.

Finding the action values of the optimal policy. We will now find the $q^*_\pi(\phi(s), up)$ and $q^*_\pi(\phi(s), down)$. The $q_\pi(\phi(s), a)$ is defined as the expected return after the $\phi(s)$, when *doing* the action a (Singh et al., 1994):

$$q_\pi(\phi(s), a) = \sum_{s'} P_\pi(s'|\phi(s)) q_\pi(s', a), \tag{20}$$

where $P_\pi(s'|\phi(s))$ is the probability of being in the state s' when observing $\phi(s)$.

In our example, the Q-values are:

$$q_\pi(\phi(s), up) = \frac{1}{2}(1 + v_\pi(2)) + \frac{1}{2}p \cdot (-1) + \frac{1}{2}(1-p) \tag{21}$$
$$= -2p + \frac{3}{2} \tag{22}$$
$$q_\pi(\phi(s), down) = \frac{1}{2}v_\pi(3) + \frac{1}{2}p + \frac{1}{2}(1-p)(-1) \tag{23}$$
$$= 2p - 1 \tag{24}$$

We can now substitute the $p^* = \frac{5}{8}$ in for p to find the $q^*_\pi(\phi(s), up)$ and $q^*_\pi(\phi(s), down)$:

$$q^*_\pi(\phi(s), up) = \frac{1}{4} \tag{25}$$
$$q^*_\pi(\phi(s), down) = \frac{1}{4}. \tag{26}$$

We see that these Q-values are the same and uninformative about the probabilities of the optimal (memory-less) stochastic policy. This generalizes to all environments: the optimal policy gives zero probability to all actions with lower Q-values. If the optimal policy $\pi^*(\cdot|\phi(s))$ at a given state representation gives non-zero probabilities to some actions, these actions must have the same Q-values $q^*_\pi(\phi(s), a)$.

Bootstrapping from $v_\pi(\phi(s))$ would be worse. We will find the $v_\pi(\phi(s))$. And we will show that bootstrapping from it would be misleading. In our example, the $v_\pi(\phi(s))$ is:

$$v_\pi(\phi(s)) = \frac{1}{2}v_\pi(1) + \frac{1}{2}pv_\pi(2) + \frac{1}{2}(1-p)v_\pi(3) \tag{27}$$
$$= -4p^2 + \frac{9}{2}p - 1. \tag{28}$$

We can notice that $v_\pi(\phi(s))$ is different from $v_\pi(2)$ or $v_\pi(3)$. Estimating $q_\pi(\phi(s), up)$ by bootstrapping from $v_\pi(\phi(2))$ instead of $v_\pi(2)$ would be misleading. Here, it is better to estimate the Q-values based on Monte-Carlo returns.

C. The motivation behind Conservative Policy Iteration and TRPO

In this section we will show that unregularized maximization of $\mathbb{E}_{A\sim\pi(\cdot|s)}\left[\hat{q}_{\pi_{\text{prior}}}(s,A)\right]$ on data from an older policy π_{prior} can produce a policy worse than π_{prior}. The size of the possible degradation will be related to the total variation distance between π and π_{prior}. The explanation is based on the proofs from the excellent book by Agarwal et al. (2020).

As before, our objective is to maximize the expected value of the states from an initial state distribution μ:

$$J(\pi) = \mathbb{E}_{S\sim\mu}\left[v_\pi(S)\right]. \tag{29}$$

It will be helpful to define the discounted state visitation distribution $d_\pi(s)$ as:

$$d_\pi(s) = (1-\gamma)\mathbb{E}_{S_0\sim\mu}\left[\sum_{t=0}^{\infty}\gamma^t P(S_t = s|\pi, S_0)\right], \tag{30}$$

where $P(S_t = s|\pi, S_0)$ is the probably of S_t being s, if starting the episode from S_0 and following the policy π. The scaling by $(1-\gamma)$ ensures that $d_\pi(s)$ sums to one.

From the policy gradient theorem (Sutton et al., 2000) we know that the gradient of $J(\pi)$ with respect to the policy parameters is

$$\frac{\partial J}{\partial\theta} = \frac{1}{1-\gamma}\sum_s d_\pi(s)\sum_a \frac{\partial\pi(a|s)}{\partial\theta}q_\pi(s,a). \tag{31}$$

In practice, we often train on data from an older policy π_{prior}. Training on such data maximizes a different function:

$$\text{TotalAdv}_{\text{prior}}(\pi) = \frac{1}{1-\gamma}\sum_s d_{\pi_{\text{prior}}}(s)\sum_a \pi(a|s)\,\text{adv}_{\pi_{\text{prior}}}(s,a), \tag{32}$$

where $\text{adv}_{\pi_{\text{prior}}}(s,a) = q_{\pi_{\text{prior}}}(s,a) - v_{\pi_{\text{prior}}}(s)$ is an advantage. Notice that the states are sampled from $d_{\pi_{\text{prior}}}(s)$ and the policy is criticized by $\text{adv}_{\pi_{\text{prior}}}(s,a)$. This happens often in the practice, if updating the policy multiple times in an episode, using a replay buffer or bootstrapping from a network trained on past data.

While maximization of $\text{TotalAdv}_{\text{prior}}(\pi)$ is more practical, we will see that unregularized maximization of $\text{TotalAdv}_{\text{prior}}(\pi)$ does not guarantee an improvement in our objective J. The $J(\pi) - J(\pi_{\text{prior}})$ difference can be even negative, if we are not careful.

Kakade & Langford (2002) stated a useful lemma for the performance difference:

Lemma C.1 (The performance difference lemma) *For all policies π, π_{prior},*

$$J(\pi) - J(\pi_{\text{prior}}) = \frac{1}{1-\gamma}\sum_s d_\pi(s)\sum_a \pi(a|s)\,\text{adv}_{\pi_{\text{prior}}}(s,a). \tag{33}$$

We would like the $J(\pi) - J(\pi_{\text{prior}})$ to be positive. We can express the performance difference as $\text{TotalAdv}_{\text{prior}}(\pi)$ plus an extra term:

$$J(\pi) - J(\pi_{\text{prior}}) = \text{TotalAdv}_{\text{prior}}(\pi) - \text{TotalAdv}_{\text{prior}}(\pi) + \frac{1}{1-\gamma}\sum_s d_\pi(s)\sum_a \pi(a|s)\,\text{adv}_{\pi_{\text{prior}}}(s,a) \tag{34}$$

$$= \text{TotalAdv}_{\text{prior}}(\pi) + \frac{1}{1-\gamma}\sum_s (d_\pi(s) - d_{\pi_{\text{prior}}}(s))\sum_a \pi(a|s)\,\text{adv}_{\pi_{\text{prior}}}(s,a) \tag{35}$$

$$= \text{TotalAdv}_{\text{prior}}(\pi) + \frac{1}{1-\gamma}\sum_s (d_\pi(s) - d_{\pi_{\text{prior}}}(s))\sum_a (\pi(a|s) - \pi_{\text{prior}}(a|s))\,\text{adv}_{\pi_{\text{prior}}}(s,a). \tag{36}$$

To get a positive $J(\pi) - J(\pi_{\text{prior}})$ performance difference, it is not enough to maximize $\text{TotalAdv}_{\text{prior}}(\pi)$. We also need to make sure that the second term in (36) will not degrade the performance. The impact of the second term can be kept small by keeping the total variation distance between π and π_{prior} small.

For example, the performance can degrade, if π is not trained at a state and that state gets a higher $d_\pi(s)$ probability. The performance can also degrade, if a stochastic policy is needed and the $\mathrm{adv}_{\pi_{\text{prior}}}$ advantages are for an older policy. The π would become deterministic, if maximizing $\sum_a \pi(a|s)\,\mathrm{adv}_{\pi_{\text{prior}}}(s,a)$ without any regularization.

C.1. Performance difference lower bound.

We will express a bound of the performance difference as a function of the total variation between π and π_{prior}. Starting from Eq. 36, we can derive the TRPO lower bound for the performance difference. Let α be the maximum total variation distance between π and π_{prior}:

$$\alpha = \max_s \frac{1}{2} \sum_a |\pi(a|s) - \pi_{\text{prior}}(a|s)|. \tag{37}$$

The $\|d_\pi - d_{\pi_{\text{prior}}}\|_1$ is then bounded (see Agarwal et al., 2020, Similar policies imply similar state visitations):

$$\|d_\pi - d_{\pi_{\text{prior}}}\|_1 \leq \frac{2\alpha\gamma}{1-\gamma}. \tag{38}$$

Finally, by plugging the bounds to Eq. 36, we can construct the lower bound for the performance difference:

$$J(\pi) - J(\pi_{\text{prior}}) \geq \mathrm{TotalAdv}_{\text{prior}}(\pi) - \frac{4\alpha^2\gamma\epsilon_{\max}}{(1-\gamma)^2}, \tag{39}$$

where $\epsilon_{\max} = \max_{s,a} |\,\mathrm{adv}_{\pi_{\text{prior}}}(s,a)|$. The same bound was derived in TRPO (Schulman et al., 2015).

D. Proof of Maximum CMPO total variation distance

We will prove the following theorem: *For any clipping threshold $c > 0$, we have:*

$$\max_{\pi_{\text{prior}},\mathrm{adv},s} \mathrm{D}_{\mathrm{TV}}(\pi_{\mathrm{CMPO}}(\cdot|s), \pi_{\text{prior}}(\cdot|s)) = \tanh(\frac{c}{2}).$$

Having 2 actions. We will first prove the theorem when the policy has 2 actions. To maximize the distance, the clipped advantages will be $-c$ and c. Let's denote the π_{prior} probabilities associated with these advantages as $1-p$ and p, respectively.

The total variation distance is then:

$$\mathrm{D}_{\mathrm{TV}}(\pi_{\mathrm{CMPO}}(\cdot|s), \pi_{\text{prior}}(\cdot|s)) = \frac{p\exp(c)}{p\exp(c) + (1-p)\exp(-c)} - p. \tag{40}$$

We will maximize the distance with respect to the parameter $p \in [0,1]$.

The first derivative with respect to p is:

$$\frac{d\,\mathrm{D}_{\mathrm{TV}}(\pi_{\mathrm{CMPO}}(\cdot|s), \pi_{\text{prior}}(\cdot|s))}{dp} = \frac{1}{(p\exp(c) + (1-p)\exp(-c))^2} - 1. \tag{41}$$

The second derivative with respect to p is:

$$\frac{d^2\,\mathrm{D}_{\mathrm{TV}}(\pi_{\mathrm{CMPO}}(\cdot|s), \pi_{\text{prior}}(\cdot|s))}{dp^2} = -2(p\exp(c) + (1-p)\exp(-c))^{-3}(\exp(c) - \exp(-c)). \tag{42}$$

Because the second derivative is negative, the distance is a concave function of p. We will find the maximum at the point where the first derivative is zero. The solution is:

$$p^* = \frac{1-\exp(-c)}{\exp(c) - \exp(-c)}. \tag{43}$$

At the found point p^*, the maximum total variation distance is:

$$\max_p \mathrm{D}_{\mathrm{TV}}(\pi_{\mathrm{CMPO}}(\cdot|s), \pi_{\mathrm{prior}}(\cdot|s)) = \frac{\exp(c) - 1}{\exp(c) + 1} = \tanh(\frac{c}{2}). \tag{44}$$

This completes the proof when having 2 actions.

Having any number of actions. We will now prove the theorem when the policy has any number of actions. To maximize the distance, the clipped advantages will be $-c$ or c. Let's denote the *sum* of π_{prior} probabilities associated with these advantages as $1 - p$ and p, respectively.

The total variation distance is again:

$$\mathrm{D}_{\mathrm{TV}}(\pi_{\mathrm{CMPO}}(\cdot|s), \pi_{\mathrm{prior}}(\cdot|s)) = \frac{p \exp(c)}{p \exp(c) + (1 - p) \exp(-c)} - p, \tag{45}$$

and the maximum distance is again $\tanh(\frac{c}{2})$.

We also verified the theorem predictions experimentally, by using gradient ascent to maximize the total variation distance.

E. Extended related work

We used the desiderata to motivate the design of the policy update. We will use the desiderata again to discuss the related methods to satisfy the desiderata. For a comprehensive overview of model-based reinforcement learning, we recommend the surveys by Moerland et al. (2020) and Hamrick (2019).

E.1. Observability and function approximation

1a) Support learning stochastic policies. The ability to learn a stochastic policy is one of the benefits of policy gradient methods.

1b) Leverage Monte-Carlo targets. Muesli uses multi-step returns to train the policy network and Q-values. MPO and MuZero need to train the Q-values, before using the Q-values to train the policy.

E.2. Policy representation

2a) Support learning the optimal memory-less policy. Muesli represents the stochastic policy by the learned policy network. In principle, acting can be based on a combination of the policy network and the Q-values. For example, one possibility is to act with the π_{CMPO} policy. ACER (Wang et al., 2016) used similar acting based on π_{MPO}. Although we have not seen benefits from acting based on π_{CMPO} on Atari (Figure 15), we have seen better results on Go with a deeper search at the evaluation time.

2b) Scale to (large) discrete action spaces. Muesli supports large actions spaces, because the policy loss can be estimated by sampling. MCTS is less suitable for large action spaces. This was addressed by Grill et al. (2020), who brilliantly revealed MCTS as regularized policy optimization and designed a tree search based on MPO or a different regularized policy optimization. The resulting tree search was less affected by a small number of simulations. Muesli is based on this view of regularized policy optimization as an alternative to MCTS. In another approach, MuZero was recently extended to support sampled actions and continuous actions (Hubert et al., 2021).

2c) Scale to continuous action spaces. Although we used the same estimator of the policy loss for discrete and continuous actions, it would be possible to exploit the structure of the continuous policy. For example, the continuous policy can be represented by a normalizing flow (Papamakarios et al., 2019) to model the joint distribution of the multi-dimensional actions. The continuous policy would also allow to estimate the gradient of the policy regularizer with the reparameterization trick (Kingma & Welling, 2013; Rezende et al., 2014). Soft Actor-Critic (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018) achieved great results on the Mujoco tasks by obtaining the gradient with respect to the action from an ensemble of approximate Q-functions. The ensemble of Q-functions would probably improve Muesli results.

E.3. Robust learning

3a) Support off-policy and historical data. Muesli supports off-policy data thanks to the regularized policy optimization, Retrace (Munos et al., 2016) and policy gradients with clipped importance weights (Gruslys et al., 2018). Many other methods deal with off-policy or offline data (Levine et al., 2020). Recently MuZero Reanalyse (Schrittwieser et al., 2021) achieved state-of-the-art results on an offline RL benchmark by training only on the offline data.

3b) Deal gracefully with inaccuracies in the values/model. Muesli does not trust fully the Q-values from the model. Muesli combines the Q-values with the prior policy to propose a new policy with a constrained total variation distance from the prior policy. Without the regularized policy optimization, the agent can be misled by an overestimated Q-value for a rarely taken action. Soft Actor-Critic (Haarnoja et al., 2018) and TD3 (Fujimoto et al., 2018) mitigate the overestimation by taking the minimum from a pair of Q-networks. In model-based reinforcement learning an unrolled one-step model would struggle with compounding errors (Janner et al., 2019). VPN (Oh et al., 2017) and MuZero (Schrittwieser et al., 2020) avoid compounding errors by using multi-step predictions $P(R_{t+k+1}|s_t, a_t, a_{t+1}, \ldots, a_{t+k})$, not conditioned on previous model predictions. While VPN and MuZero avoid compounding errors, these models are not suitable for planning a sequence of actions in a stochastic environment. In the stochastic environment, the sequence of actions needs to depend on the occurred stochastic events, otherwise the planning is confounded and can underestimate or overestimate the state value (Rezende et al., 2020). Other models conditioned on limited information from generated (latent) variables can face similar problems on stochastic environment (e.g. DreamerV2 (Hafner et al., 2020)). Muesli is suitable for stochastic environments, because Muesli uses only one-step look-ahead. If combining Muesli with a deep search, we can use an adaptive search depth or a stochastic model sufficient for causally correct planning (Rezende et al., 2020). Another class of models deals with model errors by using the model as a part of the Q-network or policy network and trains the whole network end-to-end. These networks include VIN (Tamar et al., 2016), Predictron (Silver et al., 2017), I2A (Racanière et al., 2017), IBP (Pascanu et al., 2017), TreeQN, ATreeC (Farquhar et al., 2018) (with scores in Table 3), ACE (Zhang & Yao, 2019), UPN (Srinivas et al., 2018) and implicit planning with DRC (Guez et al., 2019).

3c) Be robust to diverse reward scales. Muesli benefits from the normalized advantages and from the advantage clipping inside π_{CMPO}. Pop-Art (van Hasselt et al., 2016) addressed learning values across many orders of magnitude. On Atari, the score of the games vary from 21 on Pong to 1M on Atlantis. The non-linear transformation by Pohlen et al. (2018) is practically very helpful, although biased for stochastic returns.

3d) Avoid problem-dependent hyperparameters. The normalized advantages were used before in PPO (Schulman et al., 2017). The maximum CMPO total variation (Theorem 4.1) helps to explain the success of such normalization. If the normalized advantages are from $[-c, c]$, they behave like advantages clipped to $[-c, c]$. Notice that the regularized policy optimization with the popular $-\text{H}[\pi]$ entropy regularizer is equivalent to MPO with uniform π_{prior} (because $-\text{H}[\pi] = \text{KL}(\pi, \pi_{\text{uniform}}) + \text{const.}$). As a simple modification, we recommend to replace the uniform prior with π_{prior} based on a target network. That leads to the model-free direct MPO with normalized advantages, outperforming vanilla policy gradients (compare Figure 13 to Figure 1a).

E.4. Rich representation of knowledge

4a) Estimate values (variance reduction, bootstrapping). In Muesli, the learned values are helpful for bootstrapping Retrace returns, for computing the advantages and for constructing the π_{CMPO}. Q-values can be also helpful inside a search, as demonstrated by Hamrick et al. (2020a).

4b) Learn a model (representation, composability). Multiple works demonstrated benefits from learning a model. Like VPN and MuZero, Gregor et al. (2019) learns a multi-step action-conditional model; they learn the distribution of observations instead of actions and rewards, and focus on the benefits of representation learning in model-free RL induced by model-learning; see also (Guo et al., 2018; Guo et al., 2020). Springenberg et al. (2020) study an algorithm similar to MuZero with an MPO-like learning signal on the policy (similarly to SAC and Grill et al. (2020)) and obtain strong results on Mujoco tasks in a transfer setting. Byravan et al. (2020) use a multi-step action model to derive a learning signal for policies on continuous-valued actions, leveraging the differentiability of the model and of the policy. Kaiser et al. (2019) show how to use a model for increasing data-efficiency on Atari (using an algorithm similar to Dyna (Sutton, 1990)), but see also van Hasselt et al. (2019) for the relation between parametric model and replay. Finally, Hamrick et al. (2020b) investigate drivers of performance and generalization in MuZero-like algorithms.

Table 3. The mean score from the last 100 episodes at 40M frames on games used by TreeQN and ATreeC. The agents differ along multiple dimensions.

	Alien	Amidar	Crazy Climber	Enduro	Frostbite	Krull	Ms. Pacman	Q*Bert	Seaquest
TreeQN-1	2321	1030	107983	800	2254	10836	3030	15688	9302
TreeQN-2	2497	1170	104932	825	581	11035	3277	15970	8241
ATreeC-1	3448	**1578**	102546	678	1035	8227	4866	25159	1734
ATreeC-2	2813	1566	110712	649	281	8134	4450	25459	2176
Muesli	**16218**	524	**143898**	**2344**	**10919**	**15195**	**19244**	**30937**	**142431**

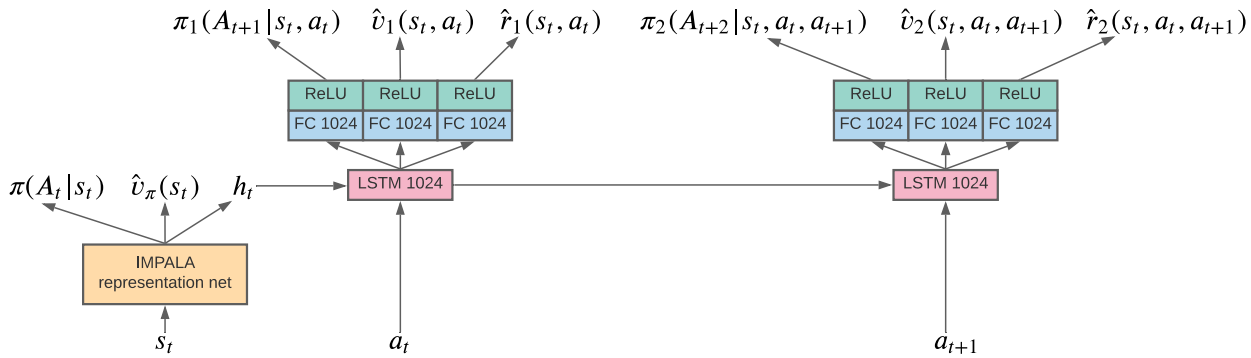

Figure 10. The model architecture when using the IMPALA-based representation network. The $\hat{r}_1(s_t,a_t)$ predicts the reward $\mathbb{E}[R_{t+1}|s_t,a_t]$. The $\hat{v}_1(s_t,a_t)$ predicts the value $\mathbb{E}[v_\pi(S_{t+1})|s_t,a_t]$. In general, $\hat{r}_k(s_t,a_{<t+k})$ predicts the reward $\mathbb{E}[R_{t+k}|s_t,a_{<t+k}]$. And $\hat{v}_k(s_t,a_{<t+k})$ predicts the value $\mathbb{E}[v_\pi(S_{t+k})|s_t,a_{<t+k}]$.

F. Experimental details

F.1. Common parts

Network architecture. The large MuZero network is used only on the large scale Atari experiments (Figure 1b) and on Go. In all other Atari and MuJoCo experiments the network architecture is based on the IMPALA architecture (Espeholt et al., 2018). Like the LASER agent (Schmitt et al., 2020), we increase the number of channels 4-times. Specifically, the numbers of channels are: (64, 128, 128, 64), followed by a fully connected layer and LSTM (Hochreiter & Schmidhuber, 1997) with 512 hidden units. This LSTM inside of the IMPALA *representation* network is different from the second LSTM used inside the model *dynamics* function, described later. In the Atari experiments, the network takes as the input one RGB frame. Stacking more frames would help as evidenced in Figure 17.

Q-network and model architecture. The original IMPALA agent was not learning a Q-function. Because we train a MuZero-like model, we can estimate the Q-values by:

$$\hat{q}(s,a) = \hat{r}_1(s,a) + \gamma \hat{v}_1(s,a), \tag{46}$$

where $\hat{r}_1(s,a)$ and $\hat{v}_1(s,a)$ are the reward model and the value model, respectively. The reward model and the value model are based on MuZero *dynamics* and *prediction* functions (Schrittwieser et al., 2020). We use a very small dynamics function, consisting of a single LSTM layer with 1024 hidden units, conditioned on the selected action (Figure 10).

The decomposition of $\hat{q}(s,a)$ to a reward model and a value model is not crucial. The Muesli agent obtained a similar score with a model of the $q_\pi(s,a)$ action-values (Figure 14).

Value model and reward model losses. Like in MuZero (Schrittwieser et al., 2020), the value model and the reward model are trained by categorical losses. The target for the value model is the multi-step return estimate provided by Retrace (Munos

et al., 2016). Inside of the Retrace, we use $\hat{q}_{\pi_{\text{prior}}}(s, a)$ action-values provided by the target network.

Optimizer. We use the Adam optimizer (Kingma & Ba, 2014), with the decoupled weight decay by (Loshchilov & Hutter, 2017). The learning rate is linearly decayed to reach zero at the end of the training. We do not clip the norm of the gradient. Instead, we clip the parameter updates to $[-1, 1]$, before multiplying them with the learning rate. In Adam's notation, the update rule is:

$$\theta_t = \theta_{t-1} + \alpha \,\text{clip}(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, -1, 1), \tag{47}$$

where $\hat{m}_t$ and $\hat{v}_t$ are the estimated moments, not value functions.

Replay. As observed by (Schmitt et al., 2020), the LASER agent benefited from mixing replay data with on-policy data in each batch. Like LASER, we also use uniform replay and mix replay data with on-policy data. To obtain results comparable with other methods, we do not use LASER's *shared* experience replay and hence compare to the LASER version that did not share experience either.

Evaluation. On Atari, the human-normalized score is computed at 200M environment frames (including skipped frames). The episode returns are collected from last 200 training episodes that *finished before* the 200M environment frames. This is the same evaluation as used by MuZero. The replayed frames are not counted in the 200M frame limit. For example, if replayed frames form 95% of each batch, the agent is trained for 20-times more steps than an agent with no replay.

F.2. Muesli policy update

The Muesli policy loss usage is summarized in Algorithm 1.

Prior policy. We use a *target* network to approximate $v_{\pi_{\text{prior}}}$, $q_{\pi_{\text{prior}}}$ and π_{prior}. Like the target network in DQN (Mnih et al., 2015), the target network contains older network parameters. We use an exponential moving average to continuously update the parameters of the target network.

In general, the π_{prior} can be represented by a mixture of multiple policies. When forming π_{CMPO}, we represented π_{prior} by the target network policy mixed with a small proportion of the uniform policy (0.3%) and the behavior policy (3%). Mixing with these policies was not a significant improvement to the results (Figure 18).

F.3. Hyperparameters

On Atari, the experiments used the Arcade Learning Environment (Bellemare et al., 2013) with sticky actions. The environment parameters are listed in Table 4.

The hyperparameters shared by all policy updates are listed in Table 5. When comparing the clipped and unclipped advantages in Figure 4, we estimated the $\text{KL}(\pi_{\text{CMPO}}, \pi)$ with exact KL. The unclipped advantages would have too large variance without the exact KL.

The hyperparameters for the large-scale Atari experiments are in Table 6, hyperparameters for 9x9 Go self-play are in Table 7 and hyperparameters for continuous control on MuJoCo are in Table 8. On Go, the discount $\gamma = -1$ allows to train by self-play on the two-player perfect-information zero-sum game with alternate moves without modifying the reinforcement learning algorithms.

F.4. Policy losses

We will explain the other compared policy losses here. When comparing the different policy losses, we always used the same network architecture and the same reward model and value model training. The advantages were always normalized.

The hyperparameters for all policy losses are listed in Table 9. We tuned the hyperparameters for all policy losses on 10 Atari games (`alien`, `beam_rider`, `breakout`, `gravitar`, `hero`, `ms_pacman`, `phoenix`, `robotank`, `seaquest` and `time_pilot`). For each hyperparameter we tried multiples of 3 (e.g., 0.1, 0.3, 1.0, 3.0). For the PPO clipping threshold, we explored 0.2, 0.25, 0.3, 0.5, 0.8.

Policy gradients (PG). The simplest tested policy loss uses policy gradients with the entropy regularizer, as in (Mnih et al.,

Algorithm 1 Agent with Muesli policy loss

Initialization:
Initialize the estimate of the variance of the advantage estimator:
$\text{var} := 0$
$\beta_{\text{product}} := 1.0$
Initialize π_{prior} parameters with the π parameters:
$\theta_{\pi_{\text{prior}}} := \theta_\pi$

Data collection on an actor:
For each step:
Observe state s_t and select action $a_t \sim \pi_{\text{prior}}(\cdot|s_t)$.
Execute a_t in the environment.
Append $s_t, a_t, r_{t+1}, \gamma_{t+1}$ to the replay buffer.
Append $s_t, a_t, r_{t+1}, \gamma_{t+1}$ to the online queue.

Training on a learner:
For each minibatch:
Form a minibatch B with sequences from the online queue and the replay buffer.
Use Retrace to estimate each return $G^v(s,a)$, bootstrapping from $\hat{q}_{\pi_{\text{prior}}}$.
Estimate the variance of the advantage estimator:
$\text{var} := \beta_{\text{var}}\text{var} + (1-\beta_{\text{var}})\frac{1}{|B|}\sum_{(s,a)\in B}(G^v(s,a) - \hat{v}_{\pi_{\text{prior}}}(s))^2$
Compute the bias-corrected variance estimate in Adam's style:
$\beta_{\text{product}} := \beta_{\text{product}}\beta_{\text{var}}$
$\widehat{\text{var}} := \frac{\text{var}}{1-\beta_{\text{product}}}$
Prepare the normalized advantages:
$\hat{\text{adv}}(s,a) = \frac{\hat{q}_{\pi_{\text{prior}}}(s,a) - \hat{v}_{\pi_{\text{prior}}}(s)}{\sqrt{\widehat{\text{var}}+\epsilon_{\text{var}}}}$
Compute the total loss:
$L_{\text{total}} = ($
$L_{\text{PG+CMPO}}(\pi, s)$ // Regularized policy optimization, Eq. 9.
$+ L_m(\pi, s)$ // Policy model loss, Eq. 13.
$+ L_v(\hat{v}_\pi, s) + L_r(\hat{r}_\pi, s))$ // MuZero value and reward losses.
Use L_{total} to update θ_π by one step of gradient descent.
Use a moving average of π parameters as π_{prior} parameters:
$\theta_{\pi_{\text{prior}}} := (1-\alpha_{\text{target}})\theta_{\pi_{\text{prior}}} + \alpha_{\text{target}}\theta_\pi$

2016). The loss is defined by

$$L_{\text{PG}}(\pi, s) = -\mathbb{E}_{A\sim\pi(\cdot|s)}\left[\hat{\text{adv}}(s,A)\right] - \lambda_{\text{H}}\text{H}[\pi(\cdot|s)]. \tag{48}$$

Policy gradients with the TRPO penalty. The next policy loss uses $\text{KL}(\pi_b(\cdot|s), \pi(\cdot|s))$ inside the regularizer. The π_b is the behavior policy. This policy loss is known to work as well as PPO (Cobbe et al., 2020).

$$L_{\text{PG+TRPOpenalty}}(\pi, s) = -\mathbb{E}_{A\sim\pi(\cdot|s)}\left[\hat{\text{adv}}(s,A)\right] - \lambda_{\text{H}}\text{H}[\pi(\cdot|s)] + \lambda_{\text{TRPO}}\,\text{KL}(\pi_b(\cdot|s), \pi(\cdot|s)). \tag{49}$$

Proximal Policy Optimization (PPO). PPO (Schulman et al., 2017) is usually used with multiple policy updates on the same batch of data. In our setup, we use a replay buffer instead. PPO then required a larger clipping threshold ϵ_{PPO}. In our setup, the policy gradient with the TRPO penalty is a stronger baseline.

Table 4. Atari parameters. In general, we follow the recommendations by Machado et al. (2018).

PARAMETER	VALUE
Random modes and difficulties	No
Sticky action probability ς	0.25
Start no-ops	0
Life information	Not allowed
Action set	18 actions
Max episode length	30 minutes (108,000 frames)
Observation size	96×96
Action repetitions	4
Max-pool over last N action repeat frames	4
Total environment frames, including skipped frames	200M

Table 5. Hyperparameters shared by all experiments.

HYPERPARAMETER	VALUE
Batch size	96 sequences
Sequence length	30 frames
Model unroll length K	5
Replay proportion in a batch	75%
Replay buffer capacity	6,000,000 frames
Initial learning rate	3×10^{-4}
Final learning rate	0
AdamW weight decay	0
Discount	0.995
Target network update rate α_{target}	0.1
Value loss weight	0.25
Reward loss weight	1.0
Retrace $\mathbb{E}_{A\sim\pi}[\hat{q}_{\pi_{\text{prior}}}(s,A)]$ estimator	16 samples
$\text{KL}(\pi_{\text{CMPO}}, \pi)$ estimator	16 samples
Variance moving average decay β_{var}	0.99
Variance offset ϵ_{var}	10^{-12}

$$L_{\text{PPO}}(\pi, s) = -\mathbb{E}_{A\sim\pi_b(\cdot|s)}\left[\min(\frac{\pi(A|s)}{\pi_b(A|s)}\,\hat{\text{adv}}(s,A), \text{clip}(\frac{\pi(A|s)}{\pi_b(A|s)}, 1-\epsilon_{\text{PPO}}, 1+\epsilon_{\text{PPO}})\,\hat{\text{adv}}(s,A))\right] - \lambda_{\text{H}}\text{H}[\pi(\cdot|s)]. \tag{50}$$

Maximum a Posteriori Policy Optimization (MPO). We use a simple variant of MPO (Abdolmaleki et al., 2018) that is not specialized to Gaussian policies. Also, we use $\pi_{\text{MPO}}(\cdot|s_{t+k})$ as the target for the policy model.

$$L_{\text{MPO}}(\pi, s_t) = \text{KL}(\pi_{\text{MPO}}(\cdot|s_t), \pi(\cdot|s_t)) + \frac{1}{K}\sum_{k=1}^{K}\text{KL}(\pi_{\text{MPO}}(\cdot|s_{t+k}), \pi_k(\cdot|s_t, a_{<t+k})) \tag{51}$$

$$s.t.\ \mathbb{E}_{S\sim d_{\pi_b}}\left[\text{KL}(\pi_{\text{MPO}}(\cdot|S), \pi(\cdot|S))\right] < \epsilon_{\text{MPO}}. \tag{52}$$

Direct MPO. Direct MPO uses the MPO regularizer $\lambda\,\text{KL}(\pi, \pi_{\text{prior}})$ as a penalty.

$$L_{\text{DirectMPO}}(\pi, s) = -\mathbb{E}_{A\sim\pi(\cdot|s)}\left[\hat{\text{adv}}(s,A)\right] + \lambda\,\text{KL}(\pi(\cdot|s), \pi_{\text{prior}}(\cdot|s)). \tag{53}$$

F.5. Go experimental details

The Go environment was configured using OpenSpiel (Lanctot et al., 2019). Games were scored with the Tromp-Taylor rules with a komi of 7.5. Observations consisted of the last 2 board positions, presented with respect to the player in three

Table 6. Modified hyperparameters for large-scale Atari experiments. The network architecture, discount and replay proportion are based on MuZero Reanalyze.

HYPERPARAMETER	VALUE
Network architecture	MuZero net with 16 ResNet blocks
Stacked frames	16
Batch size	768 sequences
Replay proportion in a batch	95%
Replay buffer capacity	28,800,000 frames
AdamW weight decay	10^{-4}
Discount	0.997
Retrace λ	0.95
$\mathrm{KL}(\pi_{\mathrm{CMPO}}, \pi)$ estimator	exact KL

Table 7. Modified hyperparameters for 9x9 Go self-play experiments.

HYPERPARAMETER	VALUE
Network architecture	MuZero net with 6 ResNet blocks
Batch size	192 sequences
Sequence length	49 frames
Replay proportion in a batch	0%
Initial learning rate	2×10^{-4}
Target network update rate α_{target}	0.01
Discount	-1 (self-play)
Multi-step return estimator	V-trace
V-trace λ	0.99

9x9 planes each (player's stones, opponent's stones, and empty intersections), in addition to a plane indicating the player's color. The agents were evaluated against GnuGo v3.8 at level 10 (Bump et al., 2005) and Pachi v11.99 (Baudiš & Gailly, 2011) with 10,000 simulations, 16 threads, and no pondering. Both were configured with the Chinese ruleset. Figure 11 shows the results versus GnuGo.

G. Additional experiments

Table 10 lists the median and mean human-normalized score across 57 Atari games. The table also lists the differences in the number of stacked frames, the amount of replay and the probability of a sticky action. The environment with a non-zero probability of a sticky action is more challenging by being stochastic (Machado et al., 2018).

In Figure 15 we compare the different ways to act and explore during training. Muesli (in blue) acts by sampling actions from the policy network. Acting proportionally to π_{CMPO} was not significantly different (in green). Acting based on the Q-values only was substantially worse (in red). This is consistent with our example from Figure 2 where acting with Q-values would be worse.

Table 8. Modified hyperparameters for MuJoCo experiments.

HYPERPARAMETER	VALUE
Replay proportion in a batch	95.8%

Table 9. Hyperparameters for the different policy losses.

HYPERPARAMETER	PG	TRPO penalty	PPO	MPO	Muesli
Total policy loss weight	3.0	3.0	3.0	3.0	3.0
Entropy bonus weight	0.003	0.0003	0.0003	0	0
TRPO penalty weight	0	0.01	0	0	0
PPO clipping ϵ_{PPO}	-	-	0.5	-	-
MPO $\text{KL}(\pi_{\text{MPO}}, \pi)$ constraint	-	-	-	0.01	-
CMPO loss weight	0	0	0	-	1.0
CMPO clipping threshold c	-	-	-	-	1.0

Table 10. Median and mean human-normalized score across 57 Atari games, after 200M environment frames. The agents differ in network size, amount of replay, the probability of a sticky action and agent training. The $\pm$ indicates the standard error across 2 random seeds. While DreamerV2 was not evaluated on `defender` and `surround`, DreamerV2 median score remains valid on 57 games, if we assume a high DreamerV2 score on `defender`.

AGENT	MEDIAN	MEAN	STACKED FRAMES	REPLAY	STICKY ACTION
DQN (Mnih et al., 2015)	79%	-	4	87.5%	0.0
IMPALA (Espeholt et al., 2018)	192%	958%	4	0%	0.0
IQN (Dabney et al., 2018)	218%	-	4	87.5%	0.0
Rainbow (Hessel et al., 2018)	231%	-	4	87.5%	0.0
Meta-gradient$\{\gamma, \lambda\}$ (Xu et al., 2018)	287%	-	4	0%	0.0
LASER (Schmitt et al., 2020)	431%	-	4	87.5%	0.0
DreamerV2 (Hafner et al., 2020)	164%	-	1	-	0.25
Muesli with IMPALA architecture	562 $\pm$3%	1,981 $\pm$66%	4	75%	0.25
Muesli with MuZero arch, replay=80%	755 $\pm$27%	2,253 $\pm$120%	16	80%	0.25
Muesli with MuZero arch, replay=95%	**1,041** $\pm$40%	2,524 $\pm$104%	16	95%	0.25
MuZero Reanalyse (Schrittwieser et al., 2021)	**1,047** $\pm$40%	2,971 $\pm$115%	16	95%	0.25

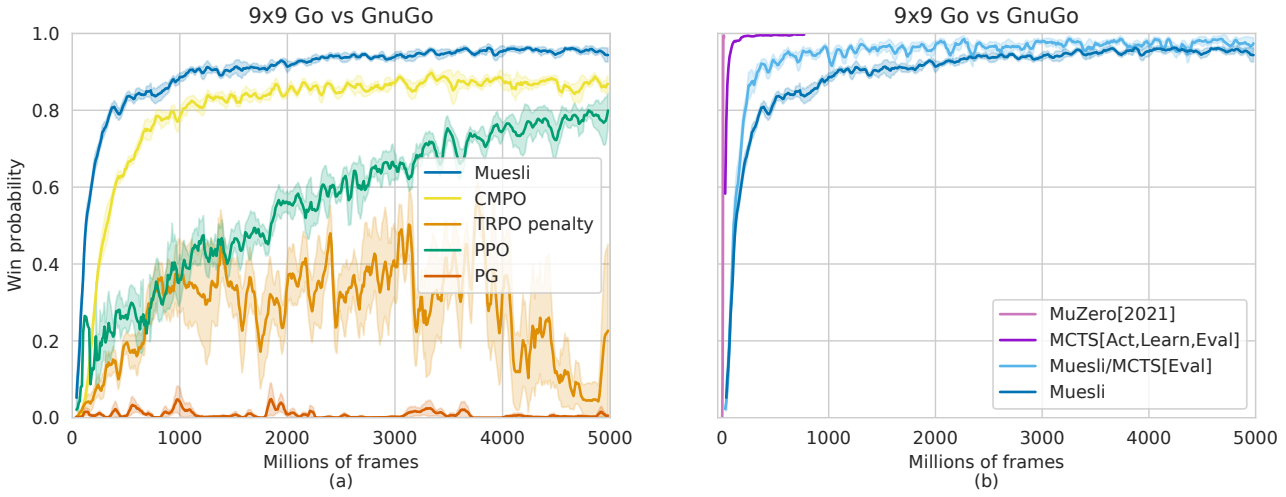

Figure 11. Win probability on 9x9 Go when training from scratch, by self-play, for 5B frames. Evaluating 3 seeds against GnuGo (level 10). **(a)** Muesli and other search-free baselines. **(b)** MuZero MCTS with 150 simulations and Muesli with and without the use of MCTS at the evaluation time only.

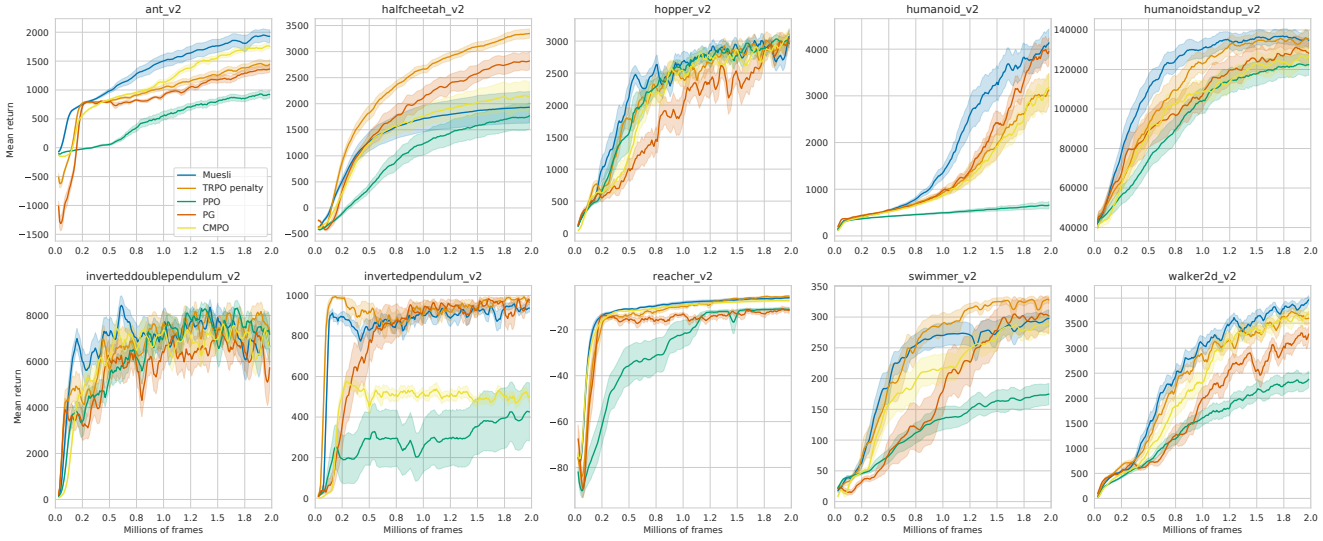

Figure 12. Mean episode return on MuJoCo environments from OpenAI Gym. The shaded area indicates the standard error across 10 random seeds.

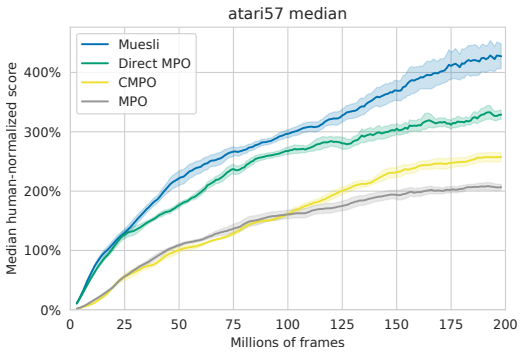

Figure 13. Median score of across 57 Atari games for different MPO variants. CMPO is MPO with clipped advantages and no constrained optimization.

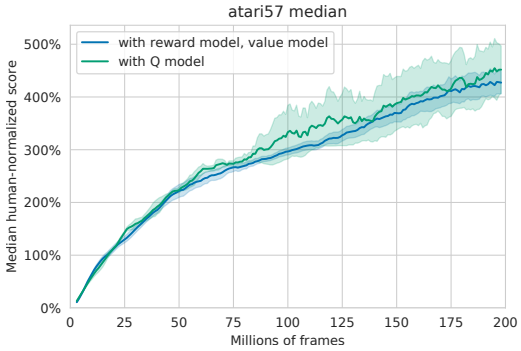

Figure 14. Median score of Muesli across 57 Atari games when modeling the reward and value or when modeling the $q_\pi(s, a)$ directly.

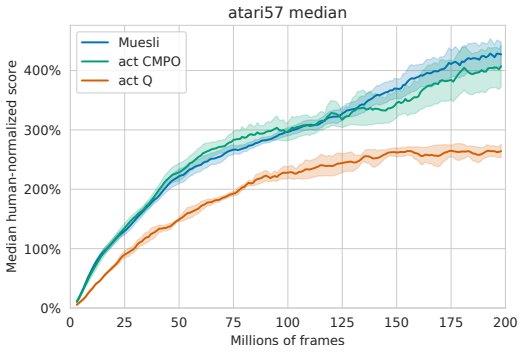

Figure 15. Median score across 57 Atari games for different ways to act and explore. Acting with π_{CMPO} was not significantly different. Acting with $\text{softmax}(\hat{q}/temperature)$ was worse.

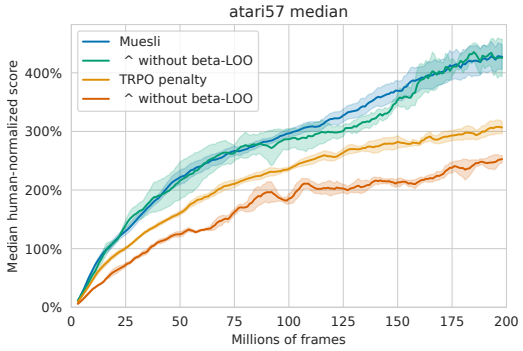

Figure 16. Median score across 57 Atari games when using or not using β-LOO action dependent baselines.

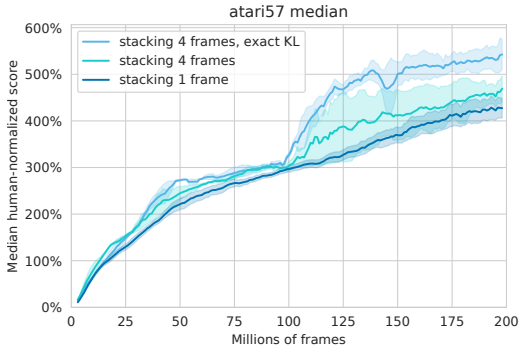

Figure 17. Median score across 57 Atari games for different numbers of stacked frames.

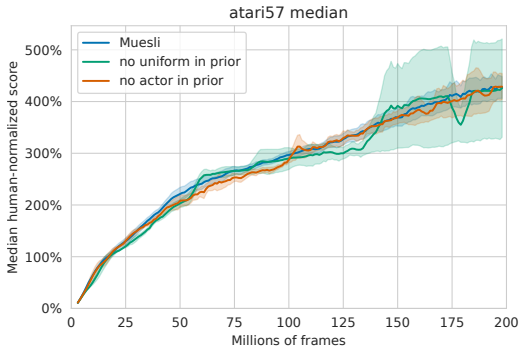

Figure 18. Median score across 57 Atari games for different π_{prior} compositions.

Table 11. The mean score from the last 200 episodes at 200M frames on 57 Atari games. The ± indicates the standard error across 2 random seeds.

GAME	Random	Human	MuZero		Muesli	
alien	228	7128	135541	±65349	**139409**	±12178
amidar	6	1720	1061	±136	**21653**	±2019
assault	222	742	29697	±3595	**36963**	±533
asterix	210	8503	**918628**	±56222	316210	±48368
asteroids	719	47389	**509953**	±33541	484609	±5047
atlantis	12850	29028	1136009	±1466	**1363427**	±81093
bank_heist	14	753	**14176**	±13044	1213	±0
battle_zone	2360	37188	320641	±141924	**414107**	±13422
beam_rider	364	16927	**319684**	±13394	288870	±137
berzerk	124	2630	19523	±16817	**44478**	±36140
bowling	23	161	156	±25	**191**	±37
boxing	0	12	**100**	±0	99	±1
breakout	2	30	778	±20	**791**	±10
centipede	2091	12017	862737	±11564	**869751**	±16547
chopper_command	811	7388	**494578**	±488588	101289	±24339
crazy_climber	10780	35829	**176172**	±17630	175322	±3408
defender	2874	18689	544320	±12881	**629482**	±39646
demon_attack	152	1971	**143846**	±8	129544	±11792
double_dunk	-19	-16	**24**	±0	-3	±2
enduro	0	861	**2363**	±2	2362	±1
fishing_derby	-92	-39	**69**	±5	51	±0
freeway	0	30	**34**	±0	33	±0
frostbite	65	4335	**410173**	±35403	301694	±275298
gopher	258	2412	**121342**	±1540	104441	±424
gravitar	173	3351	10926	±2919	**11660**	±481
hero	1027	30826	**37249**	±15	37161	±114
ice_hockey	-11	1	**40**	±2	25	±13
jamesbond	29	303	**32107**	±3480	19319	±3673
kangaroo	52	3035	13928	±90	**14096**	±421
krull	1598	2666	**50137**	±22433	34221	±1385
kung_fu_master	258	22736	**148533**	±31806	134689	±9557
montezuma_revenge	0	**4753**	1450	±1050	2359	±309
ms_pacman	307	6952	**79319**	±8659	65278	±1589
name_this_game	2292	8049	**108133**	±6935	105043	±732
phoenix	761	7243	748424	±67304	**805305**	±26719
pitfall	-229	**6464**	0	±0	0	±0
pong	-21	15	**21**	±0	20	±1
private_eye	25	**69571**	7600	±7500	10323	±4735
qbert	164	13455	85926	±8980	**157353**	±6593
riverraid	1338	17118	**172266**	±592	47323	±1079
road_runner	12	7845	**554956**	±23859	327025	±45241
robotank	2	12	**85**	±15	59	±2
seaquest	68	42055	501236	±498423	**815970**	±128885
skiing	-17098	**-4337**	-30000	±0	-18407	±1171
solaris	1236	**12327**	4401	±732	3031	±491
space_invaders	148	1669	31265	±27619	**59602**	±2759
star_gunner	664	10250	158608	±4060	**214383**	±23087
surround	-10	7	**10**	±0	9	±0
tennis	-24	-8	-0	±0	**12**	±12
time_pilot	3568	5229	**413988**	±10023	359105	±21396
tutankham	11	168	**318**	±30	252	±47
up_n_down	533	11693	**606602**	±28296	549190	±70789
venture	0	1188	866	±866	**2104**	±291
video_pinball	0	17668	**921563**	±56020	685436	±155718
wizard_of_wor	564	4757	**103463**	±3366	93291	±5
yars_revenge	3093	54577	187731	±32107	**557818**	±1895
zaxxon	32	9173	**106935**	±45495	65325	±395