A. Background on MDP

A Markov decision process (MDP; Puterman, 1990) is a tuple $(\mathcal{X}, \mathcal{A}, P, r, \gamma)$, with $\mathcal{X}$ being the state space, $\mathcal{A}$ being the action space, P the state-transition distribution maps each state-action tuple (x, a) to a probability distribution over states (with $P(y|x, a)$ denoting the probability of transitioning to state y from x by choosing action a), the reward function $r \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$ and $\gamma \in]0, 1[$ the discount factor. A stochastic policy π maps each state to a distribution over actions ($\pi(a|x)$ denotes the probability of choosing action a in state x). A deterministic policy $\pi_D \in \mathcal{X}^{\mathcal{A}}$ can also be represented by a distribution over actions π such that $\pi(\pi_D(x)|x) = 1$. We will use one or the other concept with the same notation π in the remaining when the context is clear.

Let $\mathcal{T}(x, a, \pi)$ be the distribution over trajectories $\tau = (X_t, A_t, R_t, X_{t+1})_{t\in\mathbb{N}}$ generated by a policy π, with $(X_0, A_0) = (x, a)$, $\forall t \geq 1, A_t \sim \pi(.|X_t)$, $\forall t \geq 0, R_t = r(X_t, A_t)$ and $\forall t \geq 0, X_{t+1} \sim P(.|X_t, A_t)$. Then, the state-action value function $Q^\pi_r(x, a)$ for the policy π and the state-action tuple (x, a) is defined as:

$$Q^\pi_r(x, a) = \mathbb{E}_{\tau\sim\mathcal{T}(x,a,\pi)}\left[\sum_{t\geq 0} R_t\right].$$

The optimal state-action value function Q^* is defined as:

$$Q^*_r(x, a) = \max_\pi Q^\pi_r(x, a).$$

where the $\max$ is taken over all stochastic policies.

Let define the one-step evaluation Bellman operator T^π_r, for all functions $Q \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$ and for all state-action tuples $(x, a) \in \mathcal{X} \times \mathcal{A}$, as:

$$T^\pi_r Q(x, a) = r(x, a) + \gamma \sum_{b\in\mathcal{A}} \sum_{x'\in\mathcal{X}} \pi(b|x)P(x'|x, a)Q(x', b).$$

The one-step evaluation Bellman operator can also be written with vectorial notations:

$$T^\pi_r Q = r + \gamma P^\pi Q,$$

where P^π is a transition matrix representing the effect of acting according to π in a MDP with dynamics P. The evaluation Bellman operator is a contraction and its fixed point is Q^π_r.

Finally let define the greedy operator $\mathcal{G}$, for all functions $Q \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$ and for all state $x \in \mathcal{X}$, as:

$$\mathcal{G}(Q)(x) = \arg\max_{a\in\mathcal{A}} Q(x, a).$$

Then, one can show (Puterman, 1990), via a fixed point argument, that the following discrete scheme:

$$\forall k \geq 0, \quad \begin{cases} \pi_k = \mathcal{G}\left(Q_k\right), \\ Q_{k+1} = T^{\pi_k}_r Q_k, \end{cases}$$

where Q_0 can be initialized arbitrarily, converges to Q^*_r. This discrete scheme is called the one-step value iteration scheme.

Throughout the article, we also use transformed Bellman operators (see Sec. C.2). The one-step transformed evaluation Bellman operator $T^\pi_{r,h}$, for all functions $Q \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$ and for all state-action tuples $(x, a) \in \mathcal{X} \times \mathcal{A}$, can be defined as:

$$T^\pi_{r,h} Q(x, a) = h\left(r(x, a) + \gamma \sum_{b\in\mathcal{A}} \sum_{x'\in\mathcal{X}} \pi(b|x)P(x'|x, a)h^{-1}(Q(x', b))\right),$$

where h is a monotonically increasing and invertible squashing function that scales the state-action value function to make it easier to approximate for a neural network. In particular, we use the function h:

$$\forall z \in \mathbb{R}, \quad h(z) = \text{sign}(z)(\sqrt{|z| + 1} - 1) + \epsilon z,$$
$$\forall z \in \mathbb{R}, \quad h^{-1}(z) = \text{sign}(z)\left(\left(\frac{\sqrt{1 + 4\epsilon(|z| + 1 + \epsilon)} - 1}{2\epsilon}\right) - 1\right),$$

with ϵ a small number. The one-step transformed evaluation Bellman operator can also be written with vectorial notations:

$$T^{\pi}_{r,h}Q = h\left(r + \gamma P^{\pi} h^{-1}(Q)\right).$$

Under some conditions on h (Pohlen et al., 2018) and via a contraction argument, one can show that the transformed one-step value iteration scheme:

$$\forall k \geq 0, \quad \begin{cases} \pi_k = \mathcal{G}\left(Q_k\right), \\ Q_{k+1} = T^{\pi_k}_{r,h} Q_k, \end{cases}$$

where Q_0 can be initialized arbitrarily, converges. We note this limit $Q^*_{r,h}$.

B. Extrinsic-Intrinsic Decomposition

For an intrinsically-motivated agent, the reward function r is a linear combination of the intrinsic reward r^i and the extrinsic reward r^e:

$$r = r^e + \beta r^i.$$

One can compute the optimal state-action value function Q^*_r via the value iteration scheme:

$$\forall k \geq 0, \quad \begin{cases} \pi_k = \mathcal{G}\left(Q_k\right), \\ Q_{k+1} = T^{\pi_k}_{r} Q_k, \end{cases}$$

where Q_0 can be initialized arbitrarily.

Now, we want to show how we can also converge to Q^*_r using separate intrinsic and extrinsic state-action value functions. Indeed, let us consider the following discrete scheme:

$$\forall k \geq 0, \quad \begin{cases} \tilde{\pi}_k = \mathcal{G}\left(Q^e_k + \beta Q^i_k\right), \\ Q^i_{k+1} = T^{\tilde{\pi}_k}_{r^i} Q^i_k, \\ Q^e_{k+1} = T^{\tilde{\pi}_k}_{r^e} Q^e_k, \end{cases}$$

where the functions (Q^e_0, Q^i_0) can be initialized arbitrarily.

Our goal is simply to show that the linear combination of extrinsic and intrinsic state-action value function $\tilde{Q}_k$:

$$\forall k \geq 0, \tilde{Q}_k = Q^e_k + \beta Q^i_k.$$

verifies a one-step value iteration scheme with respect to the reward $r = r^e + \beta r^i$ and therefore converges to Q^*_r. To show that let us rewrite $\tilde{Q}_{k+1}$:

$$\begin{aligned} \tilde{Q}_{k+1} &= Q^e_{k+1} + \beta Q^i_{k+1}, \\ &= T^{\tilde{\pi}_k}_{r^e} Q^e_k + \beta T^{\tilde{\pi}_k}_{r^i} Q^i_k, \\ &= r^e + \beta r^i + \gamma P^{\tilde{\pi}_k}(Q^e_k + \beta Q^i_k), \\ &= T^{\tilde{\pi}_k}_{r^e + \beta r^i}(Q^e_k + \beta Q^i_k), \\ &= T^{\tilde{\pi}_k}_{r} \tilde{Q}_k. \end{aligned}$$

Therefore we have that $\tilde{Q}_k$ satisfies a value iteration scheme with respect to the reward $r = r^e + \beta r^i$:

$$\forall k \geq 0, \quad \begin{cases} \tilde{\pi}_k = \mathcal{G}\left(\tilde{Q}_k\right), \\ \tilde{Q}_{k+1} = T^{\tilde{\pi}_k}_{r} \tilde{Q}_k, \end{cases}$$

and by the contraction property:

$$\lim_{k \longrightarrow \infty} \tilde{Q}_k = Q^*_r.$$

This result means that we can compute separately Q^e_k and Q^i_k and then mix them to obtain the same behavior than if we had computed Q_k directly with the mixed reward $r^e + \beta r^i$. This implies that we can separately compute the extrinsic

and intrinsic component. Each architecture will need to learn their state-action value for different mixtures β and then act according to the greedy policy of the mixture of the state-action value functions. This result could also be thought as related to Barreto et al. (2017) which may suggest potential future research directions.

The same type of result holds for the transformed state-action value functions. Indeed let us consider the optimal transformed state-action value function $Q^*_{r,h}$ that can be computed via the following discrete scheme:

$$\forall k \geq 0, \quad \begin{cases} \pi_k = \mathcal{G}\left(Q_k\right), \\ Q_{k+1} = T^{\pi_k}_{r,h} Q_k, \end{cases}$$

where Q_0 can be initialized arbitrarily.

Now, we show how we can compute $Q^*_{r,h}$ differently using separate intrinsic and extrinsic state-action value functions. Indeed, let us consider the following discrete scheme:

$$\forall k \geq 0, \quad \begin{cases} \tilde{\pi}_k = \mathcal{G}\left(h\left(h^{-1}(Q^e_k) + \beta h^{-1}(Q^i_k)\right)\right), \\ Q^i_{k+1} = T^{\tilde{\pi}_k}_{r^i,h} Q^i_k, \\ Q^e_{k+1} = T^{\tilde{\pi}_k}_{r^e,h} Q^e_k, \end{cases}$$

where the functions (Q^e_0, Q^i_0) can be initialized arbitrarily.

We want to show that $\tilde{Q}_k$ defines as:

$$\forall k \geq 0, \quad \tilde{Q}_k = h\left(h^{-1}(Q^e_k) + \beta h^{-1}(Q^i_k)\right),$$

verifies the one-step transformed value iteration scheme with respect to the reward $r = r^e + \beta r^i$ and therefore converges to $Q^*_{r,h}$. To show that let us rewrite $\tilde{Q}_{k+1}$:

$$\begin{aligned} \tilde{Q}_{k+1} &= h\left(h^{-1}(Q^e_{k+1}) + \beta h^{-1}(Q^i_{k+1})\right), \\ &= h\left(h^{-1}(T^{\tilde{\pi}_k}_{r^e,h} Q^e_k) + \beta h^{-1}(T^{\tilde{\pi}_k}_{r^i,h} Q^i_k)\right), \\ &= h\left(r^e + \gamma P^{\tilde{\pi}_k} h^{-1}(Q^e_k) + \beta r^i + \gamma P^{\tilde{\pi}_k} \beta h^{-1}(Q^i_k)\right), \\ &= h\left(r^e + \beta r^i + \gamma P^{\tilde{\pi}_k}(h^{-1}(Q^e_k) + \beta h^{-1}(Q^i_k))\right), \\ &= h\left(r + \gamma P^{\tilde{\pi}_k} h^{-1}(\tilde{Q}_k)\right) \\ &= T^{\tilde{\pi}_k}_{r,h} \tilde{Q}_k. \end{aligned}$$

Thus we have that $\tilde{Q}_k$ satisfies the one-step transformed value iteration scheme with respect to the reward $r = r^e + \beta r^i$:

$$\forall k \geq 0, \quad \begin{cases} \tilde{\pi}_k = \mathcal{G}\left(\tilde{Q}_k\right), \\ \tilde{Q}_{k+1} = T^{\tilde{\pi}_k}_{r,h} Q_k, \end{cases}$$

and by contraction:

$$\lim_{k \longrightarrow \infty} \tilde{Q}_k = Q^*_{r,h}.$$

One can remark that when the transformation h is the identity, we recover the linear mix between intrinsic and extrinsic state-action value functions.

C. Retrace and Transformed Retrace

Retrace (Munos et al., 2016) is an off-policy RL algorithm for evaluation or control. In the evaluation setting the goal is to estimate the state-action value function Q^π of a target policy π from trajectories drawn from a behaviour policy μ. In the control setting the goal is to build a sequence of target policies π_k and state-action value functions Q_k in order to approximate Q^*.

The evaluation Retrace operator $T_r^{\mu,\pi}$, that depends on μ and π, is defined as follows, for all functions $Q \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$ and for all state-action tuples $(x,a) \in \mathcal{X}\times\mathcal{A}$:

$$T_r^{\mu,\pi}Q(x,a) = \mathbb{E}_{\tau\sim\mathcal{T}(x,a,\mu)}\left[Q(x,a) + \sum_{t\geq 0}\gamma^t\left(\prod_{s=1}^{t}c_s\right)\delta_t\right],$$

where the temporal difference δ_t is defined as:

$$\delta_t = r_t + \gamma\sum_{a\in A}\pi(a|X_{t+1})Q(X_{t+1},a) - Q(X_t,A_t),$$

and the trace coefficients c_s as:

$$c_s = \lambda\min\left(1, \frac{\pi(A_s|X_s)}{\mu(A_s|X_s)}\right),$$

where λ is a fixed parameter $\in [0,1]$. The operator $T_r^{\mu,\pi}$ is a multi-step evaluation operator that corrects the behaviour of μ to evaluate the policy π. It has been shown in Theorem 1 of Munos et al. (2016) that Q_r^π is the fixed point of $T_r^{\mu,\pi}$. In addition, Theorem 2 of Munos et al. (2016) explains in which conditions the Retrace value iteration scheme:

$$\forall k\geq 0,\quad \begin{cases}\pi_k = \mathcal{G}\left(Q_k\right),\\ Q_{k+1} = T_r^{\mu_k,\pi_k}Q_k,\end{cases}$$

converges to the optimal state-action value function Q^*, where Q_0 is initialized arbitrarily and $\{\mu_k\}_{k\in\mathbb{N}}$ is an arbitrary sequence of policies that may depend on Q_k.

As in the case of the one-step Bellman operator, we can also define a transformed counterpart to the Retrace operator. More specifically, we can define the transformed Retrace operator $T_{r,h}^{\mu,\pi}$, for all functions $Q \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$ and for all state-action tuples $(x,a) \in \mathcal{X}\times\mathcal{A}$:

$$T_{r,h}^{\mu,\pi}Q(x,a) = h\left(\mathbb{E}_{\tau\sim\mathcal{T}(x,a,\mu)}\left[h^{-1}(Q(x,a)) + \sum_{t\geq 0}\gamma^t\left(\prod_{s=1}^{t}c_s\right)\delta_t^h\right]\right),$$

where the temporal difference δ_t^h is defined as:

$$\delta_t^h = r_t + \gamma\sum_{a\in A}\pi(a|X_{t+1})h^{-1}(Q(X_{t+1},a)) - h^{-1}(Q(X_t,A_t)).$$

As in the case of the Retrace operator, we can define the transformed Retrace value iteration scheme:

$$\forall k\geq 0,\quad \begin{cases}\pi_k = \mathcal{G}\left(Q_k\right),\\ Q_{k+1} = T_{r,h}^{\mu_k,\pi_k}Q_k,\end{cases}$$

where Q_0 is initialized arbitrarily and $\{\mu_k\}_{k\in\mathbb{N}}$ is an arbitrary sequence of policies.

C.1. Extrinsic-Intrinsic Decomposition for Retrace and Transformed Retrace

Following the same methodology than App .B, we can also show that the state-action value function can be decomposed in extrinsic and intrinsic components for the Retrace and transformed Retrace value iteration schemes when the reward is of the form $r = r^e + \beta r^i$.

Indeed if we define the following discrete scheme:

$$\forall k\geq 0,\quad \begin{cases}\tilde{\pi}_k = \mathcal{G}\left(Q_k^e + \beta Q_k^i\right),\\ Q_{k+1}^i = T_{r^i}^{\tilde{\mu}_k,\tilde{\pi}_k}Q_k^i,\\ Q_{k+1}^e = T_{r^e}^{\tilde{\mu}_k,\tilde{\pi}_k}Q_k^e,\end{cases}$$

where the functions (Q_0^e, Q_0^i) can be initialized arbitrarily and $\{\tilde{\mu_k}\}_{k\in\mathbb{N}}$ is an arbitrary sequence of policies. Then, it is straightforward to show that the linear combination $\tilde{Q}_k$:

$$\forall k\geq 0, \tilde{Q}_k = Q_k^e + \beta Q_k^i,$$

verifies the Retrace value iteration scheme:

$$\forall k \geq 0, \quad \begin{cases} \tilde{\pi}_k = \mathcal{G}\left(\tilde{Q}_k\right), \\ \tilde{Q}_{k+1} = T_r^{\tilde{\mu}_k, \tilde{\pi}_k} \tilde{Q}_k, \end{cases}$$

Likewise, if we define the following discrete scheme:

$$\forall k \geq 0, \quad \begin{cases} \tilde{\pi}_k = \mathcal{G}\left(h\left(h^{-1}(Q_k^e) + \beta h^{-1}(Q_k^i)\right)\right), \\ Q_{k+1}^i = T_{r^i,h}^{\tilde{\mu}_k, \tilde{\pi}_k} Q_k^i, \\ Q_{k+1}^e = T_{r^e,h}^{\tilde{\mu}_k, \tilde{\pi}_k} Q_k^e, \end{cases}$$

where the functions (Q_0^e, Q_0^i) can be initialized arbitrarily and $\{\tilde{\mu_k}\}_{k\in\mathbb{N}}$ is an arbitrary sequence of policies. Then, it is also straightforward to show that $\tilde{Q}_k$ defines as:

$$\forall k \geq 0, \quad \tilde{Q}_k = h\left(h^{-1}(Q_k^e) + \beta h^{-1}(Q_k^i)\right),$$

verifies the transformed Retrace value iteration scheme:

$$\forall k \geq 0, \quad \begin{cases} \tilde{\pi}_k = \mathcal{G}\left(\tilde{Q}_k\right), \\ \tilde{Q}_{k+1} = T_{r,h}^{\tilde{\mu}_k, \tilde{\pi}_k} Q_k, \end{cases}$$

C.2. Retrace and Transformed Retrace Losses for Neural Nets.

In this section, we explain how we approximate with finite data and neural networks the Retrace value iteration scheme. To start, one important thing to remark is that we can rewrite the evaluation step:

$$Q_{k+1} = T_r^{\mu_k, \pi_k} Q_k,$$

with:

$$Q_{k+1} = \underset{Q \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}}{\arg\min} \|T_r^{\mu_k, \pi_k} Q_k - Q\|,$$

where $\|.\|$ can be any norm over the function space $\mathbb{R}^{\mathcal{X}\times\mathcal{A}}$. This means that the evaluation step can be seen as an optimization problem over a functional space where the optimization consists in finding a function Q that matches the target $T_r^{\mu_k, \pi_k} Q_k$.

In practice, we face two important problems. The search space $\mathbb{R}^{\mathcal{X}\times\mathcal{A}}$ is too big and we cannot evaluate $T_r^{\mu_k, \pi_k} Q_k$ everywhere because we have a finite set of data. To tackle the former, a possible solution is to use function approximation such as neural networks. Thus, we parameterize the state action value function $Q(x, a; \theta)$ (where θ is the set of parameters of the neural network) also called online network. Concerning the latter, we are going to build sampled estimates of $T_r^{\mu_k, \pi_k} Q_k$ and use them as targets for our optimization problem. In practice, the targets are built from a previous and fixed set of parameters θ^- of the neural network. $Q(x, a; \theta^-)$ is called the target network. The target network is updated to the value of the online network at a fixed frequency during the learning.

More precisely, let us consider a batch of size B of finite sampled sequences of size H: $D = \{(x_s^b, a_s^b, \mu_s^b = \mu(a_s^b|x_s^b), r_s^b, x_{s+1}^b)_{s=t}^{t+H-1}\}_{b=0}^{B-1}$ starting from (x_t^b, a_t^b) and then following the behaviour policy μ. Then, we can define the finite sampled-Retrace targets as:

$$\begin{aligned} \hat{T}_r^{\mu,\pi} Q(x_s^b, a_s^b; \theta^-) &= Q(x_s^b, a_s^b; \theta^-) + \sum_{j=s}^{t+H-1} \gamma^{j-s} \left(\prod_{i=s+1}^{j} c_{i,b}\right) \delta_{j,b} \\ c_{i,b} &= \lambda \min\left(1, \frac{\pi(a_i^b|x_i^b)}{\mu_i^b}\right), \\ \delta_{j,b} &= r_j^b + \gamma \sum_{a\in A} \pi(a|x_{j+1}^b) Q(x_{j+1}^b, a; \theta^-) - Q(x_j^b, a_j^b; \theta^-), \end{aligned}$$

where $\pi(a|x)$ is the target policy.

Once the targets are computed, the goal is to find a parameter θ that fits those targets by minimizing the following loss function:

$$L(D, \theta, \theta^-, \pi, \mu, r) = \sum_{b=0}^{B-1} \sum_{s=t}^{t+H-1} \left(Q(x_s^b, a_s^b; \theta) - \hat{T}_r^{\mu,\pi} Q(x_s^b, a_s^b; \theta^-) \right)^2.$$

This is done by an optimizer such as gradient descent for instance. Once θ is updated by the optimizer, a new loss with new targets is computed and minimized until convergence.

Therefore in practice the evaluation step of the Retrace value iteration scheme $Q_{k+1} = T_r^{\mu_k,\pi_k} Q_k$ is approximated by minimizing the loss $L(D, \theta, \pi, \mu)$ with an optimizer. The greedy step $\pi_k = \mathcal{G}(Q_k)$ is realized by simply being greedy with respect to the online network and choosing the target policy as follows: $\pi = \mathcal{G}(Q(x, a; \theta))$.

In the case of a transformed Retrace operator, we have the following targets:

$$\begin{aligned}
\hat{T}_{r,h}^{\mu,\pi} Q(x_s^b, a_s^b; \theta^-) &= h\left(h^{-1}(Q(x_s^b, a_s^b; \theta^-)) + \sum_{j=s}^{t+H-1} \gamma^{j-t} \left(\prod_{i=s+1}^{j} c_{i,b} \right) \delta_{s,b}^h \right) \\
c_{i,b} &= \lambda \min\left(1, \frac{\pi(a_i^b|x_i^b)}{\mu_i^b}\right), \\
\delta_{j,b} &= r_j^b + \gamma \sum_{a \in A} \pi(a|x_{j+1}^b) h^{-1}(Q(x_{j+1}^b, a; \theta^-)) - h^{-1} Q(x_j^b, a_j^b; \theta^-).
\end{aligned}$$

And the transformed Retrace loss function is:

$$L(D, \theta, \theta^- \pi, \mu, r, h) = \sum_{b=0}^{B-1} \sum_{s=t}^{t+H-1} \left(Q(x_s^b, a_s^b; \theta) - \hat{T}_{r,h}^{\mu,\pi} Q(x_s^b, a_s^b; \theta^-) \right)^2.$$

D. Multi-arm Bandit Formalism

This section describes succinctly the multi-arm bandit (MAB) paradigm, upper confidence bound (UCB) algorithm and sliding-window UCB algorithm. For a more thorough explanation and analysis we refer the reader to Garivier & Moulines (2008).

At each time $k \in \mathbb{N}$, a MAB algorithm chooses an arm A_k among the possible arms $\{0, \dots, N-1\}$ according to a policy π that is conditioned on the sequence of previous actions and rewards. Doing so, it receives a reward $R_k(A_k) \in \mathbb{R}$. In the stationary case, the rewards $\{R_k(a)\}_{k \geq 0}$ for a given arm $a \in \{0, \dots, N-1\}$ are modelled by a sequence of i.i.d random variables. In the non-stationary case, the rewards $\{R_k(a)\}_{k \geq 0}$ are modelled by a sequence of independent random variables but whose distributions could change through time.

The goal of a MAB algorithm is to find a policy π that maximizes the expected cumulative reward for a given horizon K:

$$\mathbb{E}_\pi \left[\sum_{k=0}^{K-1} R_k(A_k) \right].$$

In the stationary case, the UCB algorithm has been well studied and is commonly used. Let us define the number of times an arm a has been played after k steps:

$$N_k(a) = \sum_{m=0}^{k-1} \mathbf{1}_{\{A_m = a\}}.$$

Let us also define the empirical mean of an arm a after k steps:

$$\hat{\mu}_k(a) = \frac{1}{N_k(a)} \sum_{m=0}^{k-1} R_k(a) \mathbf{1}_{\{A_m = a\}}.$$

The UCB algorithm is then defined as follows:

$$\begin{cases} \forall 0 \leq k \leq N-1, \quad A_k = k \\ \forall N \leq k \leq K-1, \quad A_k = \arg\max_{1 \leq a \leq N} \hat{\mu}_{k-1}(a) + \beta \sqrt{\frac{\log(k-1)}{N_{k-1}(a)}} \end{cases}$$

In the non-stationary case, the UCB algorithm cannot adapt to the change of reward distribution and one can use a sliding-window UCB in that case. It is commonly understood that the window length $\tau \in \mathbb{N}^*$ should be way smaller that the horizon K. Let us define the number of times an arm a has been played after k steps for a window of length τ:

$$N_k(a, \tau) = \sum_{m=0 \vee k-\tau}^{k-1} \mathbf{1}_{\{A_m = a\}},$$

where $0 \vee k - \tau$ means $\max(0, k - \tau)$. Let define the empirical mean of an arm a after k steps for a window of length τ:

$$\hat{\mu}_k(a, \tau) = \frac{1}{N_k(a, \tau)} \sum_{m=0 \vee k-\tau}^{k-1} R_k(a) \mathbf{1}_{\{A_m = a\}}.$$

Then , the sliding window UCB can be defined as follows:

$$\begin{cases} \forall 0 \leq k \leq N-1, \quad A_k = k \\ \forall N \leq k \leq K-1, \quad A_k = \arg\max_{1 \leq a \leq N} \hat{\mu}_{k-1}(a, \tau) + \beta \sqrt{\frac{\log(k-1 \wedge \tau)}{N_{k-1}(a, \tau)}} \end{cases}$$

where $k - 1 \wedge \tau$ means $\min(k - 1, \tau)$.

In our experiments, we use a simplified sliding window UCB with $\epsilon_{\texttt{UCB}}$-greedy exploration:

$$\begin{cases} \forall 0 \leq k \leq N-1, \quad A_k = k \\ \forall N \leq k \leq K-1 \text{ and } U_k \geq \epsilon_{\texttt{UCB}}, \quad A_k = \arg\max_{0 \leq a \leq N-1} \hat{\mu}_{k-1}(a, \tau) + \beta \sqrt{\frac{1}{N_{k-1}(a, \tau)}} \\ \forall N \leq k \leq K-1 \text{ and } U_k < \epsilon_{\texttt{UCB}}, \quad A_k = Y_k \end{cases}$$

where U_k is a random value drawn uniformly from $[0, 1]$ and Y_k a random action drawn uniformly from $\{0, \ldots, N-1\}$.

E. Implementation details of the distributed setting

Replay buffer: it stores fixed-length sequences of *transitions* $\xi = (\omega_s)_{s=t}^{t+H-1}$ along with their priorities p_ξ. A transition is of the form $\omega_s = (r^e_{s-1}, r^i_{s-1}, a_{s-1}, h_{s-1}, x_s, a_s, h_s, \mu_s, j_s, r^e_s, r^i_s, x_{s+1})$. Such transitions are also called *timesteps* and the length of a sequence H is called the *trace length*. In addition, adjacent sequences in the replay buffer overlap by a number of timesteps called the *replay period* and the sequences never cross episode boundaries. Let us describe each element of a transition:

- r^e_{s-1}: extrinsic reward at the previous time.
- r^i_{s-1}: intrinsic reward at the previous time.
- a_{s-1}: action done by the agent at the previous time.
- h_{s-1}: recurrent state (in our case hidden state of the LSTM) at the previous time.
- x_s: observation provided by the environment at the current time.
- a_s: action done by the agent at the current time.
- h_s: recurrent state (in our case hidden state of the LSTM) at the current time.
- μ_s: the probability of choosing the action a_s.

- $j_s = j$: index of the pair (γ_j, β_j) chosen at a beginning of an episode in each actor by the multi-arm bandit algorithm (fixed for the whole sequence).
- r_s^e: extrinsic reward at the current time.
- r_s^i: intrinsic reward at the current time
- x_{s+1}: observation provided by the environment at the next time.

In our experiment, we choose a trace length of 160 with a replay period of 80 or a trace length of 80 with a replay period of 40. Please refer to (Kapturowski et al., 2018) for a detailed experimental of trade-offs on different treatments of recurrent states in the replay. Finally, concerning the priorities, we followed the same prioritization scheme proposed by Kapturowski et al. (2018) using a mixture of max and mean of the TD-errors in the sequence with priority exponent $\eta = 0.9$.

Actors: each of the L actors shares the same network architecture as the learner but with different weights θ_l, with $0 \leq l \leq L-1$. The l-th actor updates its weights θ_l every 400 frames by copying the weights of the learner. At the beginning of each episode, each actor chooses, via a multi-arm bandit algorithm, an index j that represents a pair (γ_j, β_j) in the family of pairs $(\{\beta_j, \gamma_j)\}_{j=0}^{N-1}$. In addition, the recurrent state is initialized to zero. To act, an actor will need to do a forward pass on the network in order to compute the state-action value for all actions, noted $Q(x_t, ., j; \theta_l)$. To do so the inputs of the network are :

- x_t: the observation at time t.
- r_{t-1}^e: the extrinsic reward at the previous time, initialized with $r_{-1}^e = 0$.
- r_{t-1}^i: the intrinsic reward at the previous time, initialized with $r_{-1}^i = 0$.
- a_{t-1}: the action at the previous time, a_{-1} is initialized randomly.
- h_{t-1}: recurrent state at the previous time, is initialized with $h_{-1} = 0$.
- $j_{t-1} = j$: a one-hot index of the pair (β_j, γ_j) chosen by the multi-arm bandit algorithm (fixed for all the episode).

At time t, the $l-$th actor acts ϵ_l-greedy with respect to $Q(x_t, ., j; \theta_l)$:

$$\begin{cases} \text{If: } U_t < \epsilon_l, a_t = Y_t, \\ \text{Else: } a_t = \arg\max_{a \in \mathcal{A}} Q(x_t, a, j; \theta_l), \end{cases}$$

where U_t is a random value drawn uniformly from $[0, 1]$ and Y_t a random action drawn uniformly from $\mathcal{A}$. The probability μ_t associated to a_t is therefore:

$$\begin{cases} \text{If: } U_t < \epsilon_l, \mu_t = \frac{\epsilon_l}{|\mathcal{A}|}, \\ \text{Else: } \mu_t = 1 - \epsilon_l \frac{|\mathcal{A}|-1}{|\mathcal{A}|}, \end{cases}$$

where $|\mathcal{A}|$ is the cardinal number of the action space, 18 in the case of Atari games. Then, the actor plays the action a_t and computes the intrinsic reward r_t^i and the environment produces the next observation x_{t+1} and the extrinsic reward r_t^e. This process goes on until the end of the episode.

The value of the noise ϵ_l is chosen according to the same formula established by Horgan et al. (2018):

$$\epsilon_l = \epsilon^{1+\alpha \frac{l}{L-1}}$$

where $\epsilon = 0.4$ and $\alpha = 8$. In our experiments, we fix the number of actors to $L = 256$. Finally, the actors send the data collected to the replay along with the priorities.

Evaluator: the evaluator shares the same network architecture as the learner but with different weights θ_e. The evaluator updates its weights θ_l every 5 episodes frames by copying the weights of the learner. Unlike the actors, the experience produced by the evaluator is not sent to the replay buffer. The evaluator alternates between the following states every 5 episodes:

- **Training bandit algorithm**: the evaluator chooses, via a multi-arm bandit algorithm, an index j that represents a pair (γ_j, β_j) in the family of pairs $(\{\beta_j, \gamma_j)\}_{j=0}^{N-1}$. Then it proceeds to act in the same way as the actors, described above. At the end of the episode, the undiscounted returns are used to train the multi-arm bandit algorithm.

- **Evaluation**: the evaluator chooses the greedy choice of index j, $\arg\max_{1\leq a\leq N} \hat{\mu}_{k-1}(a)$, so it acts with (γ_j, β_j). Then it proceeds to act in the same way as the actors, described above. At the end of 5 episodes and before switching to the other mode, the results of those 5 episodes are average and reported.

Learner: The learner contains two identical networks called the online and target networks with different weights θ and θ^- respectively (Mnih et al., 2015). The target network's weights θ^- are updated to θ every 1500 optimization steps. For our particular architecture, the weights $\theta = \theta^e \cup \theta^i$ can be decomposed in a set of intrinsic weights θ^e and θ^i that have the same architecture. Likewise, we have $\theta^- = \theta^{-,e} \cup \theta^{-,i}$. The intrinsic and extrinsic weights are going to be updated by their own transformed Retrace loss. θ^e and θ^i are updated by executing the following sequence of instructions:

- First, the learner samples a batch of size B of fixed-length sequences of transitions $D = \{\xi^b = (\omega_s^b)_{s=t}^{t+H-1}\}_{b=0}^{B-1}$ from the replay buffer.

- Then, a forward pass is done on the online network and the target with inputs $\{(x_s^b, r_{s-1}^{e,b}, r_{s-1}^{i,b}, j^b, a_{s-1}^b, h_{s-1}^b)_{s=t}^{t+H}\}_{b=0}^{B-1}$ in order to obtain the state-action values $\{(Q(x_s^b, ., j^b; \theta^e), Q(x_s^b, ., j^b; \theta^{-,e}), Q(x_s^b, ., j^b; \theta^i), Q(x_s^b, ., j^b; \theta^{-,i}))_{s=t}^{t+H}\}_{b=0}^{B-1}$.

- Once the state-action values are computed, it is now easy to compute the transformed Retrace losses $L(D, \theta^e, \theta^{-,e}, \pi, \mu, r^e, h)$ and $L(D, \theta^i, \theta^{-,i}, \pi, \mu, r^i, h)$ for each set of weights θ^e and θ^i, respectively, as shown in Sec .C. The target policy π is greedy with respect to $Q(x_s^b, ., j^b; \theta^e) + \beta_{j_s^b} Q(x_s^b, ., j^b; \theta^i)$ or with respect to $h\left(h^{-1}(Q(x_s^b, ., j^b; \theta^e)) + \beta_{j_s^b} h^{-1}(Q(x_s^b, ., j^b; \theta^i))\right)$ in the case where we want to apply a transform h to the mixture of intrinsic and extrinsic state-action value functions.

- The transformed Retrace losses are optimized with an Adam optimizer.

- Like NGU, the inverse dynamics model and the random network distillation losses necessary to compute the intrinsic rewards are optimized with an Adam optimizer.

- Finally, the priorities are computed for each sampled sequence of transitions ξ^b and updated in the replay buffer.

Computation used: in terms of hardware we train the agent with a single GPU-based learner, performing approximately 5 network updates per second (each update on a mini-batch of 64 sequences of length 160. We use 256 actors, with each one performing ~ 260 environment steps per second on Atari.

F. Network Architectures

Both extrinsic and intrinsic networks use a dueling head (Wang et al., 2015).

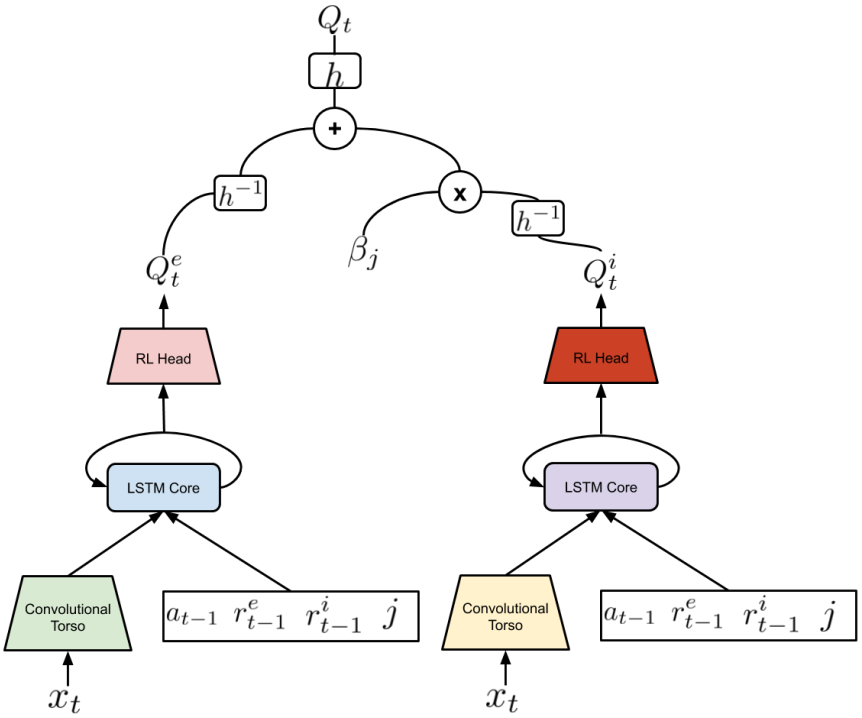


Figure 9. Sketch of the Agent57.

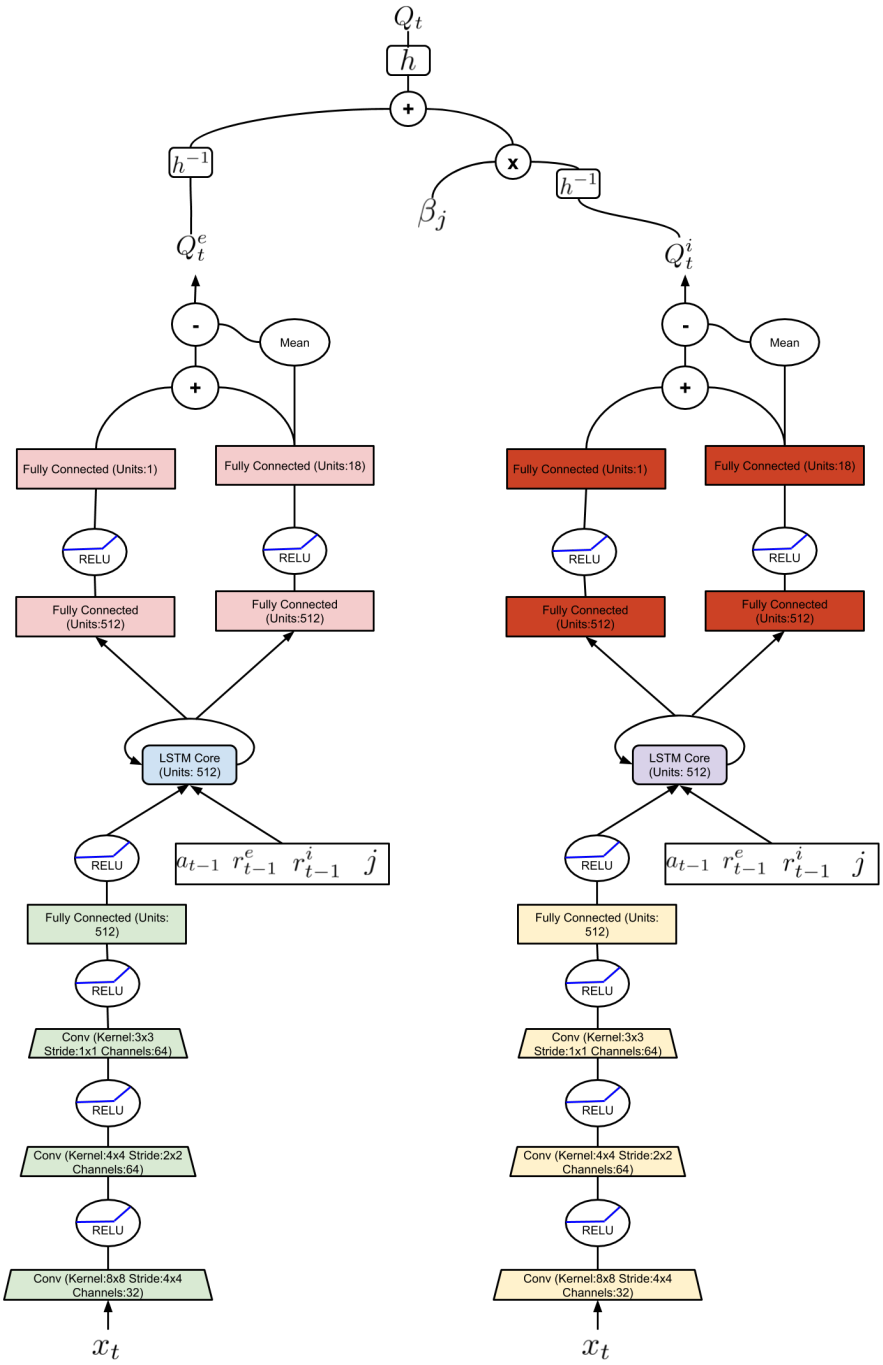
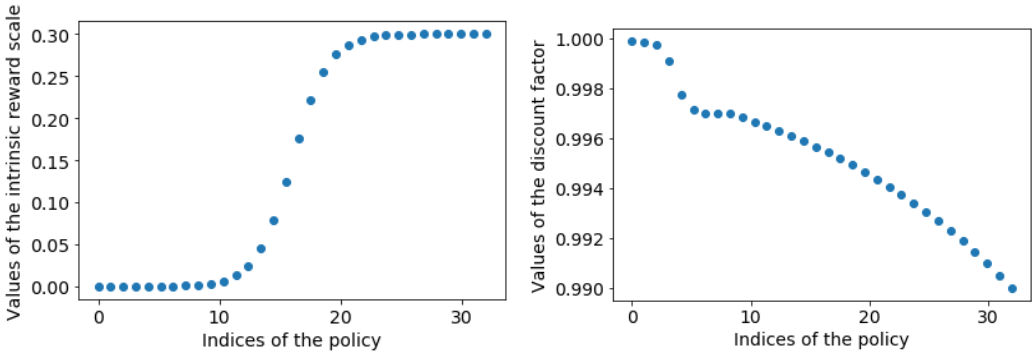


Figure 10. Detailed Agent57.

G. Hyperparameters

G.1. Values of β and γ

The intuition between the choice of the set $\{(\beta_j, \gamma_j)\}_{j=0}^{N-1}$ is the following. Concerning the β_j we want to encourage policies which are very exploitative and very exploratory and that is why we choose a sigmoid as shown in Fig. 11(a). Concerning the γ_j we would like to allow for long term horizons (high values of γ_j) for exploitative policies (small values of β_j) and small term horizons (low values of γ_j) for exploratory policies (high values of β_j). This is mainly due to the sparseness of the extrinsic reward and the dense nature of the intrinsic reward. This motivates the choice done in Fig. 11(b).



(a) Values taken by the $\{\beta_i\}_{i=0}^{N-1}$ (b) Values taken by the $\{\gamma_i\}_{i=0}^{N-1}$

Figure 11. Values taken by the $\{\beta_i\}_{i=0}^{N-1}$ and the $\{\gamma_i\}_{i=0}^{N-1}$ for $N = 32$ and $\beta = 0.3$.

$$\beta_j = \begin{cases} 0 & \text{if } j = 0 \\ \beta = 0.3 & \text{if } j = N-1 \\ \beta \cdot \sigma(10 \frac{2j-(N-2)}{N-2}) & otherwise \end{cases}, \quad \gamma_j = \begin{cases} \gamma_0 & \text{if } j = 0 \\ \gamma_1 + (\gamma_0 - \gamma_1)\sigma(10\frac{2i-6}{6}) & \text{if } j \in \{1, \dots, 6\} \\ \gamma_1 & \text{if } j = 7 \\ 1 - \exp\left(\frac{(N-9)\log(1-\gamma_1)+(j-8)\log(1-\gamma_2)}{N-9}\right) & otherwise \end{cases}$$

where $N = 32$, $\gamma_0 = 0.9999$, $\gamma_1 = 0.997$ and $\gamma_2 = 0.99$.

G.2. Atari pre-processing hyperparameters

In this section we detail the hyperparameters we use to pre-process the environment frames received from the Arcade Learning Environment. On Tab. 2 we detail such hyperparameters. ALE is publicly available at https://github.com/mgbellemare/Arcade-Learning-Environment.

Hyperparameter	Value
Max episode length	30 min
Num. action repeats	4
Num. stacked frames	1
Zero discount on life loss	$false$
Random noops range	30
Sticky actions	$false$
Frames max pooled	3 and 4
Grayscaled/RGB	Grayscaled
Action set	Full

Table 2. Atari pre-processing hyperparameters.

G.3. Hyperparameters Used

The hyperparameters that we used in all experiments are exactly like those of NGU. However, for completeness, we detail them below in Tab. 3. We also include the hyperparameters we use for the windowed UCB bandit.

Hyperparameter	**Value**
Number of mixtures N	32
Optimizer	AdamOptimizer (for all losses)
Learning rate (R2D2)	0.0001
Learning rate (RND and Action prediction)	0.0005
Adam epsilon	0.0001
Adam beta1	0.9
Adam beta2	0.999
Adam clip norm	40
Discount r^i	0.99
Discount r^e	0.997
Batch size	64
Trace length	160
Replay period	80
Retrace λ	0.95
R2D2 reward transformation	$\text{sign}(x) \cdot (\sqrt{\lvert x \rvert + 1} - 1) + 0.001 \cdot x$
Episodic memory capacity	30000
Embeddings memory mode	Ring buffer
Intrinsic reward scale β	0.3
Kernel ϵ	0.0001
Kernel num. neighbors used	10
Replay capacity	$5e6$
Replay priority exponent	0.9
Importance sampling exponent	0.0
Minimum sequences to start replay	6250
Actor update period	100
Target Q-network update period	1500
Embeddings target update period	once/episode
Action prediction network L2 weight	0.00001
RND clipping factor L	5
Evaluation ϵ	0.01
Target ϵ	0.01
Bandit window size	90
Bandit UCB β	1
Bandit ϵ	0.5

Table 3: Agent57 hyperparameters.

G.4. Hyperparameters Search Range

The ranges we used to select the hyperparameters of Agent57 are displayed on Tab. 4.

Hyperparameter	**Value**
Bandit window size τ	$\{160,\ 224,\ 320, 640\}$
Bandit $\epsilon_{\texttt{UCB}}$	$\{0.3,\ 0.5,\ 0.7\}$

Table 4. Range of hyperparameters sweeps.

H. Experimental Results

H.1. Atari 10: Table of Scores for the Ablations

Games	R2D2 (Retrace) long trace	R2D2 (Retrace) high gamma	NGU sep. nets	NGU Bandit	NGU + sep. nets + bandit
beam rider	287326.72 ± 5700.31	**349971.96 ± 5595.38**	151082.57 ± 8666.19	249006.62 ± 19662.62	244491.89 ± 25348.14
freeway	**33.91 ± 0.09**	32.84 ± 0.06	32.91 ± 0.58	26.43 ± 1.66	32.87 ± 0.12
montezuma revenge	566.67 ± 235.70	1664.89 ± 1177.26	**11539.69 ± 1227.71**	7619.70 ± 3444.76	7966.67 ± 2531.58
pitfall	0.00 ± 0.00	0.00 ± 0.00	15195.27 ± 8005.22	2979.57 ± 2919.08	**16402.61 ± 10471.27**
pong	**21.00 ± 0.00**	21.00 ± 0.00	21.00 ± 0.00	20.56 ± 0.28	21.00 ± 0.00
private eye	21729.91 ± 9571.60	22480.31 ± 10362.99	63953.38 ± 26278.51	43823.40 ± 4808.23	**80581.86 ± 28331.16**
skiing	-10784.13 ± 2539.27	-4596.26 ± 601.04	-19817.99 ± 7755.19	**-4051.99 ± 569.78**	-4278.86 ± 270.96
solaris	**52500.89 ± 2910.14**	14814.76 ± 11361.16	44771.13 ± 4920.53	43963.59 ± 5765.41	17254.14 ± 5840.70
surround	**10.00 ± 0.00**	10.00 ± 0.00	9.77 ± 0.23	-7.57 ± 0.05	9.60 ± 0.20
venture	2100.00 ± 0.00	1774.89 ± 83.79	**3249.01 ± 544.19**	2228.04 ± 305.50	2576.98 ± 394.84

H.2. Backprop window length comparison

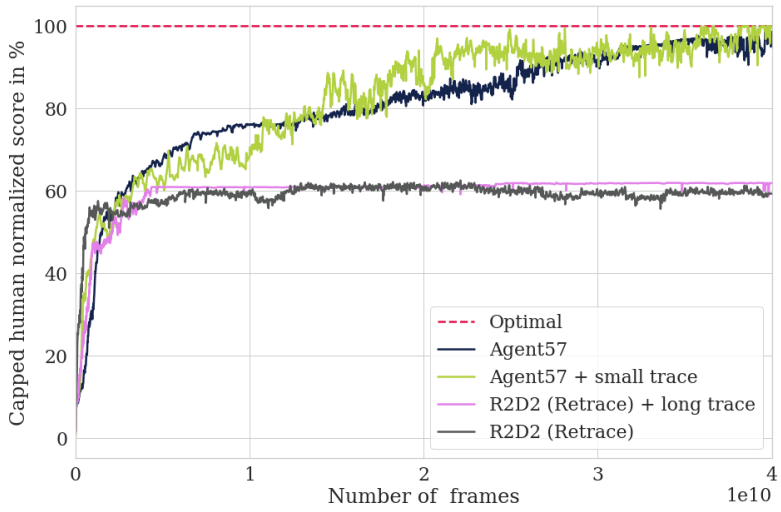


Figure 12. Performance comparison for short and long backprob window length on the 10-game *challenging set*.

H.3. Identity versus h-transform mixes comparison

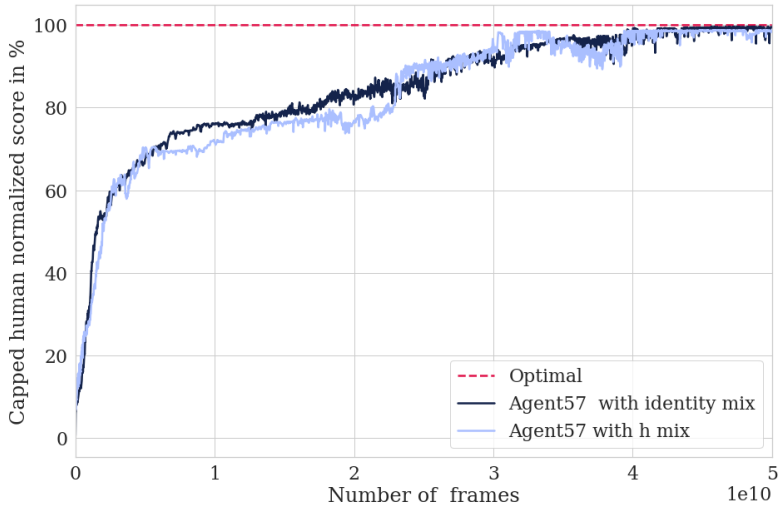


Figure 13. Performance comparison for identity versus h-transform mixes on the 10-game *challenging set*.

As shown in Fig H.3, choosing an identity or an h-transform mix does not seem to make a difference in terms of performance. The only real important thing is that a combination between extrinsic and intrinsic happens whether it is linear or not. In addition, one can remark that for extreme values of β ($\beta = 0$, $\beta >> 1$), the quantities $Q^e_k(x,a) + \beta Q^i_k(x,a)$ and $h^{-1}(Q^e_k(x,a)) + \beta h^{-1}(Q^i_k(x,a))$ have the same $\arg\max_{a \in \mathcal{A}}$ because h^{-1} is strictly increasing. Therefore, this means that on the extremes values of β, the transform and normal value iteration schemes converge towards the same policy. For in between values of β, this is not the case. But we can conjecture that when a transform operator and identity mix are used, the value iteration scheme approximates a state-action value function that is optimal with respect to a non-linear combination of the intrinsic and extrinsic rewards r^i, r^e, respectively.

H.4. Atari 57 Table of Scores

Games	Average Human	Random	Agent57	R2D2 (Bandit)	MuZero
alien	7127.70	227.80	297638.17 ± 37054.55	464883.72 ± 65894.16	**741812.63**
amidar	1719.50	5.80	29660.08 ± 880.39	**31125.16 ± 728.63**	28634.39
assault	742.00	222.40	67212.67 ± 6150.59	108428.85 ± 2116.98	**143972.03**
asterix	8503.30	210.00	991384.42 ± 9493.32	**999174.84 ± 171.58**	998425.00
asteroids	47388.70	719.10	150854.61 ± 16116.72	430848.49 ± 5250.10	**678558.64**
atlantis	29028.10	12850.00	1528841.76 ± 28282.53	1655915.92 ± 10333.29	**1674767.20**
bank heist	753.10	14.20	23071.50 ± 15834.73	**26030.12 ± 1760.95**	1278.98
battle zone	37187.50	2360.00	934134.88 ± 38916.03	**993532.51 ± 1593.45**	848623.00
beam rider	16926.50	363.90	300509.80 ± 13075.35	384466.78 ± 12163.43	**454993.53**
berzerk	2630.40	123.70	61507.83 ± 26539.54	75327.00 ± 4347.40	**85932.60**
bowling	160.70	23.10	251.18 ± 13.22	195.32 ± 94.32	**260.13**
boxing	12.10	0.10	100.00 ± 0.00	**100.00 ± 0.00**	100.00
breakout	30.50	1.70	790.40 ± 60.05	863.82 ± 0.11	**864.00**
centipede	12017.00	2090.90	412847.86 ± 26087.14	858609.27 ± 69946.39	**1159049.27**
chopper command	7387.80	811.00	999900.00 ± 0.00	**999900.00 ± 0.00**	991039.70
crazy climber	35829.40	10780.50	565909.85 ± 89183.85	**704871.91 ± 79819.90**	458315.40
defender	18688.90	2874.50	677642.78 ± 16858.59	730014.57 ± 1192.36	**839642.95**
demon attack	1971.00	152.10	143161.44 ± 220.32	143854.95 ± 112.12	**143964.26**
double dunk	-16.40	-18.60	23.93 ± 0.06	**24.00 ± 0.00**	23.94
enduro	860.50	0.00	2367.71 ± 8.69	2379.88 ± 4.44	**2382.44**
fishing derby	-38.70	-91.70	86.97 ± 3.25	91.00 ± 2.83	**91.16**
freeway	29.60	0.00	32.59 ± 0.71	**34.00 ± 0.00**	33.03
frostbite	4334.70	65.20	541280.88 ± 17485.76	406665.87 ± 262725.94	**631378.53**
gopher	2412.50	257.60	117777.08 ± 3108.06	129540.81 ± 268.58	**130345.58**
gravitar	3351.40	173.00	**19213.96 ± 348.25**	18868.69 ± 3136.73	6682.70
hero	30826.40	1027.00	**114736.26 ± 49116.60**	47331.28 ± 4662.63	49244.11
ice hockey	0.90	-11.20	63.64 ± 6.48	**86.81 ± 0.74**	67.04
jamesbond	302.80	29.00	135784.96 ± 9132.28	**158734.23 ± 1116.23**	41063.25
kangaroo	3035.00	52.00	**24034.16 ± 12565.88**	18038.22 ± 753.41	16763.60
krull	2665.50	1598.00	251997.31 ± 20274.39	225260.19 ± 49729.04	**269358.27**
kung fu master	22736.30	258.50	206845.82 ± 11112.10	**267487.46 ± 2437.72**	204824.00
montezuma revenge	4753.30	0.00	**9352.01 ± 2939.78**	3000.00 ± 0.00	0.00
ms pacman	6951.60	307.30	63994.44 ± 6652.16	60152.05 ± 3138.09	**243401.10**
name this game	8049.00	2292.30	54386.77 ± 6148.50	137023.58 ± 4928.28	**157177.85**
phoenix	7242.60	761.40	908264.15 ± 28978.92	**992489.75 ± 7436.02**	955137.84
pitfall	6463.70	-229.40	**18756.01 ± 9783.91**	0.00 ± 0.00	0.00
pong	14.60	-20.70	20.67 ± 0.47	**21.00 ± 0.00**	21.00
private eye	69571.30	24.90	**79716.46 ± 29515.48**	40700.00 ± 0.00	15299.98
qbert	13455.00	163.90	580328.14 ± 151251.66	**788781.26 ± 159574.70**	72276.00
riverraid	17118.00	1338.50	63318.67 ± 5659.55	96331.50 ± 27876.09	**323417.18**
road runner	7845.00	11.50	243025.80 ± 79555.98	585363.97 ± 1576.09	**613411.80**
robotank	11.90	2.20	127.32 ± 12.50	**144.00 ± 0.00**	131.13
seaquest	42054.70	68.40	999997.63 ± 1.42	**999999.00 ± 0.00**	999976.52
skiing	-4336.90	-17098.10	-4202.60 ± 607.85	**-3682.75 ± 485.24**	-29968.36
solaris	12326.70	1236.30	44199.93 ± 8055.50	**62021.43 ± 651.94**	56.62
space invaders	1668.70	148.00	48680.86 ± 5894.01	67619.96 ± 1660.12	**74335.30**
star gunner	10250.00	664.00	839573.53 ± 67132.17	**998270.32 ± 1070.75**	549271.70
surround	6.50	-10.00	9.50 ± 0.19	**10.00 ± 0.00**	9.99
tennis	-8.30	-23.80	23.84 ± 0.10	**24.00 ± 0.00**	0.00
time pilot	5229.20	3568.00	405425.31 ± 17044.45	461687.66 ± 2991.91	**476763.90**
tutankham	167.60	11.40	**2354.91 ± 3421.43**	467.52 ± 35.58	491.48
up n down	11693.20	533.40	623805.73 ± 23493.75	703949.97 ± 10411.25	**715545.61**
venture	1187.50	0.00	**2623.71 ± 442.13**	2237.20 ± 36.56	0.40
video pinball	17667.90	0.00	992340.74 ± 12867.87	**999231.39 ± 429.53**	981791.88
wizard of wor	4756.50	563.50	157306.41 ± 16000.00	182733.67 ± 12412.89	**197126.00**
yars revenge	54576.90	3092.90	998532.37 ± 375.82	**999775.14 ± 7.89**	553311.46
zaxxon	9173.30	32.50	249808.90 ± 58261.59	361955.14 ± 17904.80	**725853.90**

Agent57: Outperforming the Atari Human Benchmark

Games	Agent57	NGU	R2D2 (Retrace)	R2D2
alien	297638.17 ± 37054.55	312024.15 ± 91963.92	228483.74 ± 111660.11	**399709.08 ± 106191.42**
amidar	29660.08 ± 880.39	18369.47 ± 2141.76	28777.05 ± 803.90	**30338.91 ± 1087.62**
assault	67212.67 ± 6150.59	42829.17 ± 7452.17	46003.71 ± 8996.65	**124931.33 ± 2627.16**
asterix	991384.42 ± 9493.32	996141.15 ± 3993.26	998867.54 ± 191.35	**999403.53 ± 76.75**
asteroids	150854.61 ± 16116.72	248951.23 ± 7561.86	345910.03 ± 13189.10	**394765.73 ± 16944.82**
atlantis	1528841.76 ± 28282.53	**1659575.47 ± 4140.68**	1659411.83 ± 9934.57	1644680.76 ± 5784.97
bank heist	23071.50 ± 15834.73	20012.54 ± 20377.89	16726.07 ± 10992.11	**38536.66 ± 11645.73**
battle zone	934134.88 ± 38916.03	813965.40 ± 94503.50	845666.67 ± 51527.68	**956179.17 ± 31019.66**
beam rider	**300509.80 ± 13075.35**	75889.70 ± 18226.52	123281.81 ± 4566.16	246078.69 ± 3667.61
berzerk	61507.83 ± 26539.54	45601.93 ± 5170.98	**73475.91 ± 8107.24**	64852.56 ± 17875.17
bowling	251.18 ± 13.22	215.38 ± 13.27	**257.88 ± 4.84**	229.39 ± 24.57
boxing	**100.00 ± 0.00**	99.71 ± 0.25	100.00 ± 0.00	99.27 ± 0.35
breakout	790.40 ± 60.05	625.86 ± 42.66	859.60 ± 2.04	**863.25 ± 0.34**
centipede	412847.86 ± 26087.14	596427.16 ± 7149.84	**737655.85 ± 25568.85**	693733.73 ± 74495.81
chopper command	999900.00 ± 0.00	999900.00 ± 0.00	999900.00 ± 0.00	**999900.00 ± 0.00**
crazy climber	**565909.85 ± 89183.85**	351390.64 ± 62150.96	322741.20 ± 23024.88	549054.89 ± 39413.08
defender	677642.78 ± 16858.59	684414.06 ± 3876.41	681291.73 ± 3469.95	**692114.71 ± 4864.99**
demon attack	143161.44 ± 220.32	143695.73 ± 154.88	**143899.22 ± 53.78**	143830.91 ± 107.18
double dunk	23.93 ± 0.06	-12.63 ± 5.29	**24.00 ± 0.00**	23.97 ± 0.03
enduro	2367.71 ± 8.69	2095.40 ± 80.81	2372.77 ± 3.50	**2380.22 ± 5.47**
fishing derby	86.97 ± 3.25	34.62 ± 4.91	**87.83 ± 2.78**	87.81 ± 1.28
freeway	32.59 ± 0.71	28.71 ± 2.07	**33.48 ± 0.16**	32.90 ± 0.11
frostbite	**541280.88 ± 17485.76**	284044.19 ± 227850.49	12290.11 ± 7936.49	446703.01 ± 63780.51
gopher	117777.08 ± 3108.06	119110.87 ± 463.03	119803.94 ± 3197.88	**126241.97 ± 519.70**
gravitar	**19213.96 ± 348.25**	14771.91 ± 843.17	14194.45 ± 1250.63	17352.78 ± 2675.27
hero	**114736.26 ± 49116.60**	71592.84 ± 12109.10	54967.97 ± 5411.73	39786.01 ± 7638.19
ice hockey	63.64 ± 6.48	-3.15 ± 0.47	86.56 ± 1.21	**86.89 ± 0.88**
jamesbond	**135784.96 ± 9132.28**	28725.27 ± 2902.52	32926.31 ± 3073.94	28988.32 ± 263.79
kangaroo	24034.16 ± 12565.88	**37392.82 ± 6170.95**	15185.87 ± 931.58	14492.75 ± 5.29
krull	251997.31 ± 20274.39	150896.04 ± 33729.56	149221.98 ± 17583.30	**291043.06 ± 10051.59**
kung fu master	206845.82 ± 11112.10	215938.95 ± 22050.67	228228.90 ± 5316.74	**252876.65 ± 10424.57**
montezuma revenge	9352.01 ± 2939.78	**19093.74 ± 12627.66**	2300.00 ± 668.33	2666.67 ± 235.70
ms pacman	**63994.44 ± 6652.16**	48695.12 ± 1599.94	45011.73 ± 1822.30	50337.02 ± 4004.55
name this game	54386.77 ± 6148.50	25608.90 ± 1943.41	74104.70 ± 9053.70	**74501.48 ± 11562.26**
phoenix	908264.15 ± 28978.92	**966685.41 ± 6127.24**	937874.90 ± 22525.79	876045.70 ± 25511.04
pitfall	**18756.01 ± 9783.91**	15334.30 ± 15106.90	-0.45 ± 0.50	0.00 ± 0.00
pong	20.67 ± 0.47	19.85 ± 0.31	20.95 ± 0.01	**21.00 ± 0.00**
private eye	79716.46 ± 29515.48	**100314.44 ± 291.22**	34601.01 ± 5266.39	18765.05 ± 16672.27
qbert	580328.14 ± 151251.66	479024.20 ± 98094.39	434753.72 ± 99793.58	**771069.21 ± 152722.56**
riverraid	**63318.67 ± 5659.55**	40770.82 ± 748.42	43174.10 ± 2335.12	54280.32 ± 1245.60
road runner	243025.80 ± 79555.98	151326.54 ± 77209.43	116149.17 ± 18257.21	**613659.42 ± 397.72**
robotank	127.32 ± 12.50	11.62 ± 0.67	**143.59 ± 0.29**	130.72 ± 9.75
seaquest	999997.63 ± 1.42	**999999.00 ± 0.00**	999999.00 ± 0.00	999999.00 ± 0.00
skiing	**-4202.60 ± 607.85**	-24271.33 ± 6936.26	-14576.05 ± 875.96	-17797.59 ± 866.55
solaris	**44199.93 ± 8055.50**	7254.03 ± 3653.55	6566.03 ± 2209.91	11247.88 ± 1999.22
space invaders	48680.86 ± 5894.01	48087.13 ± 11219.39	36069.75 ± 23408.12	**67229.37 ± 2316.31**
star gunner	839573.53 ± 67132.17	450096.08 ± 158979.59	420337.48 ± 8309.08	**923739.89 ± 69234.32**
surround	9.50 ± 0.19	-9.32 ± 0.67	9.96 ± 0.01	**10.00 ± 0.00**
tennis	23.84 ± 0.10	11.06 ± 6.10	**24.00 ± 0.00**	7.93 ± 11.36
time pilot	405425.31 ± 17044.45	368520.34 ± 70829.26	452966.67 ± 5300.62	**454055.63 ± 2205.07**
tutankham	**2354.91 ± 3421.43**	197.90 ± 7.47	466.59 ± 38.40	413.80 ± 3.89
up n down	623805.73 ± 23493.75	630463.10 ± 31175.20	**679303.61 ± 4852.85**	599134.12 ± 3394.48
venture	**2623.71 ± 442.13**	1747.32 ± 101.40	2013.31 ± 11.24	2047.51 ± 20.83
video pinball	992340.74 ± 12867.87	973898.32 ± 20593.14	964670.12 ± 4015.52	**999697.05 ± 53.37**
wizard of wor	157306.41 ± 16000.00	121791.35 ± 27909.14	134017.82 ± 11871.88	**179376.15 ± 6659.14**
yars revenge	998532.37 ± 375.82	997642.09 ± 455.73	998474.20 ± 589.50	**999748.54 ± 46.19**
zaxxon	249808.90 ± 58261.59	129330.99 ± 56872.31	114990.68 ± 56726.18	**366028.59 ± 49366.03**

H.5. Atari 57 Learning Curves

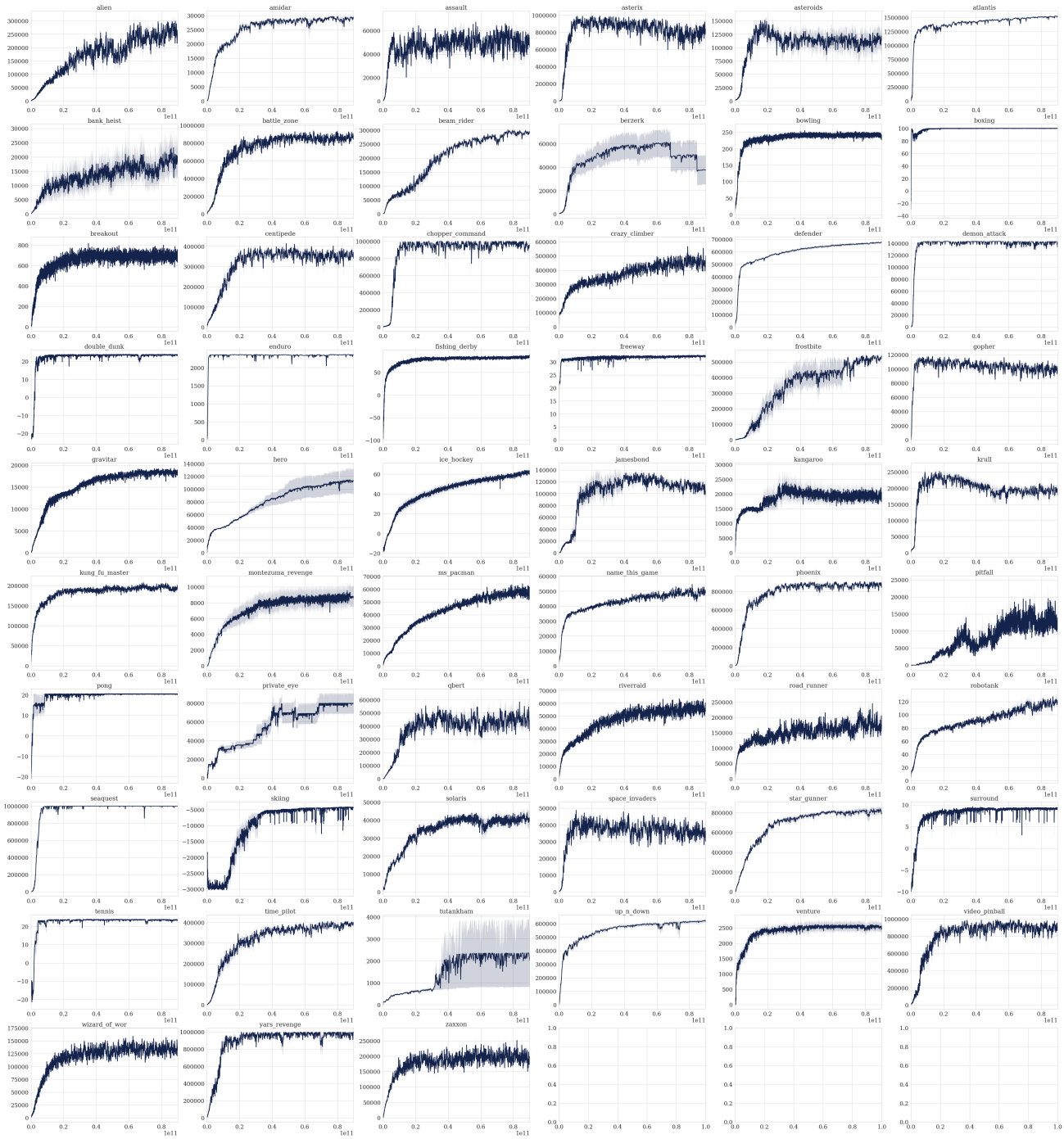


Figure 14. Learning curves for Agent57 on Atari57.

H.6. Videos

We provide several videos in https://sites.google.com/corp/view/agent57. We show

- **Agent57 on all 57 games:** We provide an example video for each game in the Atari 57 sweep in which Agent57 surpasses the human baseline.
- **State-action Value Function Parameterization:** To illustrate the importance of the value function parametrization we show videos in two games *Ice Hockey* and *Surround*. We show videos for exploitative and exploratory policies for both NGU and Agent57. In *Ice Hockey*, exploratory and exploitative policies are quite achieving very different scores. Specifically the exploratory policy does not aim to score goals, it prefers to move around the court exploring new configurations. On the other hand, NGU with a single architecture is unable to learn both policies simultaneously, while Agent57 show very diverse performance. In the case of *Surround* NGU is again unable to learn. We conjecture that the exploratory policy chooses to loose a point in order to start afresh increasing the diversity of the observations. Agent57 is able to overcome this problem and both exploitative and exploratory policies are able to obtain scores surpassing the human baseline.
- **Adaptive Discount Factor:** We show example videos for R2D2 (bandit) and R2D2 (retrace) in the game *James Bond*. R2D2 (retrace) learns to clear the game with a final score in the order of 30,000 points. R2D2 (bandit) in contrast, learns to delay the end of the game to collect significantly more rewards with a score around 140,000 points. To achieve this, the adaptive mechanism in the meta-controller selects policies with very high discount factors.
- **Backprop Through Time Window Size:** We provide videos showing example episodes for NGU and Agent57 on the game of *Solaris*. In order to achieve high scores, the agent needs to learn to move around the grid screen and look for enemies. This is a long term credit assignment problem as the agent needs to bind the actions taken on the grid screen with the reward achieved many time steps later.

I. Intrinsic reward computation

We begin by providing a general overview of the computation of the proposed intrinsic reward r_t^i. The reward is composed of two blocks: an *episodic novelty module* and an (optional) *life-long novelty module*, represented in red and green respectively in Fig. 15 (right).

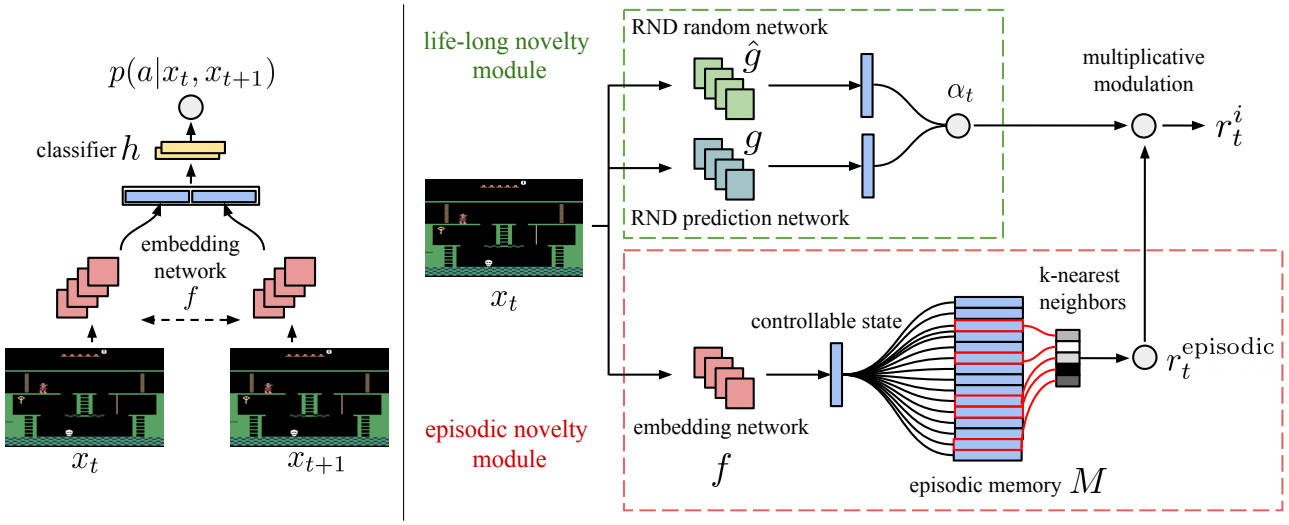


Figure 15. (left) Training architecture for the embedding network (right) Reward computation.

The episodic novelty module computes our episodic intrinsic reward and is composed of an episodic memory, M, and an embedding function f, mapping the current observation to a learned representation that we refer to as controllable state. At the beginning of each episode, the episodic memory starts completely empty. At every step, the agent computes an episodic intrinsic reward, r_t^{episodic}, and appends the controllable state corresponding to the current observation to the memory M.

To determine this episodic reward, the current observation is compared to the content of the episodic memory. Larger differences produce larger episodic intrinsic rewards.

A life-long (or inter-episodic) novelty module provides a long-term novelty signal to control the amount of exploration across episodes. We do so by multiplicatively modulating the exploration bonus r_t^{episodic} with a life-long curiosity factor, α_t. Note that this modulation will vanish over time, reducing our method to using the non-modulated reward. Specifically, we combine α_t with r_t^{episodic} as follows (see also Fig. 15 (right)):

$$r_t^i = r_t^{\text{episodic}} \cdot \min\left\{\max\left\{\alpha_t, 1\right\}, L\right\}$$

where L is a chosen maximum reward scaling.

Embedding network: $f : \mathcal{O} \to \mathbb{R}^p$ maps the current observation to a p-dimensional vector corresponding to its controllable state. Given a triplet $\{x_t, a_t, x_{t+1}\}$ composed of two consecutive observations, x_t and x_{t+1}, and the action taken by the agent a_t, we parameterise the conditional likelihood as $p(a|x_t, x_{t+1}) = h(f(x_t), f(x_{t+1}))$, where h is a one hidden layer MLP followed by a softmax. The parameters of both h and f are trained via maximum likelihood. This architecture can be thought of as a Siamese network with a one-layer classifier on top, see Fig. 15 (left) for an illustration.

Episodic memory and intrinsic reward: The episodic memory M is a dynamically-sized slot-based memory that stores the controllable states in an online fashion (Pritzel et al., 2017). At time t, the memory contains the controllable states of all the observations visited in the current episode, $\{f(x_0), f(x_1), \ldots, f(x_{t-1})\}$. Inspired by theoretically-justified exploration methods turning state-action counts into a bonus reward (Strehl & Littman, 2008), we define our intrinsic reward as

$$r_t^{\text{episodic}} = \frac{1}{\sqrt{n(f(x_t))}} \approx \frac{1}{\sqrt{\sum_{f_i \in N_k} K(f(x_t), f_i)} + c}$$

where $n(f(x_t))$ is the counts for the visits to the abstract state $f(x_t)$. We approximate these counts $n(f(x_t))$ as the sum of the similarities given by a kernel function $K : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$, over the content of M. In practice, pseudo-counts are computed using the k-nearest neighbors of $f(x_t)$ in the memory M, denoted by $N_k = \{f_i\}_{i=1}^k$. The constant c guarantees a minimum amount of *pseudo-counts* (fixed to 0.001 in all our experiments). Note that when K is a Dirac delta function, the approximation becomes exact but consequently provides no generalisation of exploration required for very large state spaces. Following (Blundell et al., 2016; Pritzel et al., 2017), we use the inverse kernel for K,

$$K(x, y) = \frac{\epsilon}{\frac{d^2(x,y)}{d_m^2} + \epsilon}$$

where ϵ is a small constant (fixed to 10^{-3} in all our experiments), d is the Euclidean distance and d_m^2 is a running average of the squared Euclidean distance of the k-th nearest neighbors. This running average is used to make the kernel more robust to the task being solved, as different tasks may have different typical distances between learnt embeddings.

Integrating life-long curiosity: The RND (Burda et al., 2018) modulator α_t is defined by introducing a random, untrained convolutional network $g : \mathcal{O} \to \mathbb{R}^k$, and training a predictor network $\hat{g} : \mathcal{O} \to \mathbb{R}^k$ that attempts to predict the outputs of g on all the observations that are seen during training by minimizing $\text{err}(x_t) = ||\hat{g}(x_t; \theta) - g(x_t)||^2$ with respect to the parameters of $\hat{g}$, θ. We then define the modulator α_t as a normalized mean squared error, as done in (Burda et al., 2018): $\alpha_t = 1 + \frac{err(x_t) - \mu_e}{\sigma_e}$, where σ_e and μ_e are running standard deviation and mean for $\text{err}(x_t)$.

Now, we present the algorithm for computing the episodic reward in Alg. 1, provide the values of the common hyperparameters in Tab. 5 and recall some useful notations:

- $f(x_t)$ is the embedding for the observation at time t. The embedding network f is trained by minimizing an inverse dynamics loss.
- M the episodic memory containing at time t the previous embeddings $\{f(x_0), f(x_1), \ldots, f(x_{t-1})\}$.
- k is the number of nearest neighbours.
- $N_k = \{f_i\}_{i=1}^k$ is the set of k-nearest neighbours of $f(x_t)$ in the memory M.

- K the kernel defined as $K(x, y) = \frac{\epsilon}{\frac{d^2(x,y)}{d_m^2} + \epsilon}$ where ϵ is a small constant, d is the Euclidean distance and d_m^2 is a running average of the squared Euclidean distance of the k-nearest neighbors.
- c is the pseudo-counts constant.
- ξ cluster distance.
- s_m maximum similarity.

Algorithm 1 Computation of the episodic intrinsic reward at time t: r_t^{episodic}.

Input : M; k; $f(x_t)$; c; ϵ; ξ; s_m; d_m^2
Output: r_t^{episodic}

Compute the k-nearest neighbours of $f(x_t)$ in M and store them in a list N_k Create a list of floats d_k of size k /* The list d_k will contain the distances between the embedding $f(x_t)$ and its neighbours N_k. */
for $i \in \{1, \ldots, k\}$ **do**
  $d_k[i] \leftarrow d^2(f(x_t), N_k[i])$
end
Update the moving average d_m^2 with the list of distances d_k /* Normalize the distances d_k with the updated moving average d_m^2. */
$d_n \leftarrow \frac{d_k}{d_m^2}$ /* Cluster the normalized distances d_n i.e. they become 0 if too small and 0_k is a list of k zeros. */
$d_n \leftarrow \max(d_n - \xi, 0_k)$ /* Compute the Kernel values between the embedding $f(x_t)$ and its neighbours N_k. */
$K_v \leftarrow \frac{\epsilon}{d_n + \epsilon}$ /* Compute the similarity between the embedding $f(x_t)$ and its neighbours N_k. */
$s \leftarrow \sqrt{\sum_{i=1}^{k} K_v[i]} + c$ /* Compute the episodic intrinsic reward at time t: r_t^i. */
if $s > s_m$ **then**
  $r_t^{\text{episodic}} \leftarrow 0$
else
  $r_t^{\text{episodic}} \leftarrow \frac{1}{s}$

Hyperparameter	Value
Learning rate (RND and Action prediction)	0.0005
Adam epsilon	0.0001
Adam beta1	0.9
Adam beta2	0.999
Adam clip norm	40
Batch size	64
Episodic memory capacity	30000
Embeddings memory mode	Ring buffer
Intrinsic reward scale β	0.3
Kernel ϵ	0.0001
Kernel num. neighbors used k	10
Kernel cluster distance ξ	0.008
Kernel pseudo-counts constant c	0.001
Kernel maximum similarity s_m	8
Embeddings target update period	once/episode
Action prediction network L2 weight	0.00001
RND clipping factor L	5

Table 5: Common hyperparameters for the reward computation.