This is a follow-up of NGU

Agent57: Outperforming the Atari Human Benchmark

Adrià Puigdomènech Badia [*1] **Bilal Piot** [*1] **Steven Kapturowski** [*1] **Pablo Sprechmann** [*1] **Alex Vitvitskyi** [1]
Daniel Guo [1] **Charles Blundell** [1]

Abstract

Atari games have been a long-standing benchmark in the reinforcement learning (RL) community for the past decade. This benchmark was proposed to test general competency of RL algorithms. Previous work has achieved good average performance by doing outstandingly well on many games of the set, but very poorly in several of the most challenging games. We propose Agent57, the first deep RL agent that outperforms the standard human benchmark on all 57 Atari games. To achieve this result, we train a neural network which parameterizes a family of policies ranging from very exploratory to purely exploitative. We propose an adaptive mechanism to choose which policy to prioritize throughout the training process. Additionally, we utilize a novel parameterization of the architecture that allows for more consistent and stable learning.

1. Introduction

The Arcade Learning Environment (ALE; Bellemare et al., 2013) was proposed as a platform for empirically assessing agents designed for general competency across a wide range of games. ALE offers an interface to a diverse set of Atari 2600 game environments designed to be engaging and challenging for human players. As Bellemare et al. (2013) put it, the Atari 2600 games are well suited for evaluating general competency in AI agents for three main reasons: *(i)* varied enough to claim generality, *(ii)* each interesting enough to be representative of settings that might be faced in practice, and *(iii)* each created by an independent party to be free of experimenter's bias.

Agents are expected to perform well in as many games as possible making minimal assumptions about the domain at hand and without the use of game-specific information. Deep Q-Networks (DQN ; Mnih et al., 2015) was the first algorithm to achieve human-level control in a large number of the Atari 2600 games, measured by human normalized scores (HNS). Subsequently, using HNS to assess performance on Atari games has become one of the most widely used benchmarks in deep reinforcement learning (RL), despite the human baseline scores potentially underestimating human performance relative to what is possible (Toromanoff et al., 2019). Nonetheless, human benchmark performance remains an oracle for "reasonable performance" across the 57 Atari games. Despite all efforts, no single RL algorithm has been able to achieve over 100% HNS on all 57 Atari games with one set of hyperparameters. Indeed, state of the art algorithms in model-based RL, MuZero (Schrittwieser et al., 2019), and in model-free RL, R2D2 (Kapturowski et al., 2018) surpass 100% HNS on 51 and 52 games, respectively. While these algorithms achieve well above average human-level performance on a large fraction of the games (e.g. achieving more than 1000% HNS), in the games they fail to do so, they often fail to learn completely. These games showcase particularly important issues that a general RL algorithm should be able to tackle. Firstly, *long-term credit assignment*: which decisions are most deserving of credit for the positive (or negative) outcomes that follow? This problem is particularly hard when rewards are delayed and credit needs to be assigned over long sequences of actions, such as in the games of *Skiing* or *Solaris*. The game of *Skiing* is a canonical example due to its peculiar reward structure. The goal

Figure 1. Number of games where algorithms are better than the human benchmark throughout training for Agent57 and state-of-the-art baselines on the 57 Atari games.

[*]Equal contribution [1]DeepMind. Correspondence to: Adrià Puigdomènech Badia <adriap@google.com>.

of the game is to run downhill through all gates as fast as possible. A penalty of five seconds is given for each missed gate. The reward, given only at the end, is proportional to the time elapsed. Therefore long-term credit assignment is needed to understand why an action taken early in the game (e.g. missing a gate) has a negative impact in the obtained reward. Secondly, *exploration*: efficient exploration can be critical to effective learning in RL. Games like *Private Eye*, *Montezuma's Revenge*, *Pitfall!* or *Venture* are widely considered hard exploration games (Bellemare et al., 2016; Ostrovski et al., 2017) as hundreds of actions may be required before a first positive reward is seen. In order to succeed, the agents need to keep exploring the environment despite the apparent impossibility of finding positive rewards. These problems are particularly challenging in large high dimensional state spaces where function approximation is required.

Exploration algorithms in deep RL generally fall into three categories: randomized value functions (Osband et al., 2016; Fortunato et al., 2017; Salimans et al., 2017; Plappert et al., 2017; Osband et al., 2018), unsupervised policy learning (Gregor et al., 2016; Achiam et al., 2018; Eysenbach et al., 2018) and intrinsic motivation (Schmidhuber, 1991; Oudeyer et al., 2007; Barto, 2013; Bellemare et al., 2016; Ostrovski et al., 2017; Fu et al., 2017; Tang et al., 2017; Burda et al., 2018; Choi et al., 2018; Savinov et al., 2018; Puigdomènech Badia et al., 2020). Other work combines handcrafted features, domain-specific knowledge or privileged pre-training to side-step the exploration problem, sometimes only evaluating on a few Atari games (Aytar et al., 2018; Ecoffet et al., 2019). Despite the encouraging results, no algorithm has been able to significantly improve performance on challenging games without deteriorating performance on the remaining games without relying on human demonstrations (Pohlen et al., 2018). Notably, amongst all this work, intrinsic motivation, and in particular, *Never Give Up* (NGU; Puigdomènech Badia et al., 2020) has shown significant recent promise in improving performance on hard exploration games. NGU achieves this by augmenting the reward signal with an internally generated intrinsic reward that is sensitive to novelty at two levels: short-term novelty within an episode and long-term novelty across episodes. It then learns a family of policies for exploring and exploiting (sharing the same parameters), with the end goal of obtain the highest score under the exploitative policy. However, NGU is not the most general agent: much like R2D2 and MuZero are able to perform strongly on all but few games, so too NGU suffers in that it performs strongly on a smaller, *different* set of games to agents such as MuZero and R2D2 (despite being based on R2D2). For example, in the game *Surround* R2D2 achieves the optimal score while NGU performs similar to a random policy. One shortcoming of NGU is that it collects the same amount of experience following each of its policies, regardless of their contribution to the learning progress. Some games require a significantly different degree of exploration to others. Intuitively, one would want to allocate the shared resources (both network capacity and data collection) such that end performance is maximized. We propose allowing NGU to adapt its exploration strategy over the course of an agent's lifetime, enabling specialization to the particular game it is learning. This is the first significant improvement we make to NGU to allow it to be a more general agent.

Recent work on long-term credit assignment can be categorized into roughly two types: ensuring that gradients correctly assign credit (Ke et al., 2017; Weber et al., 2019; Ferret et al., 2019; Fortunato et al., 2019) and using values or targets to ensure correct credit is assigned (Arjona-Medina et al., 2019; Hung et al., 2019; Liu et al., 2019; Harutyunyan et al., 2019). NGU is also unable to cope with long-term credit assignment problems such as *Skiing* or *Solaris* where it fails to reach 100% HNS. Advances in credit assignment in RL often involve a mixture of both approaches, as values and rewards form the loss whilst the flow of gradients through a model directs learning.

In this work, we propose tackling the long-term credit assignment problem by improving the overall training stability, dynamically adjusting the discount factor, and increasing the backprop through time window. These are relatively simple changes compared to the approaches proposed in previous work, but we find them to be effective. Much recent work has explored this problem of how to dynamically adjust hyperparameters of a deep RL agent, e.g., approaches based upon evolution (Jaderberg et al., 2017), gradients (Xu et al., 2018) or multi-armed bandits (Schaul et al., 2019). Inspired by Schaul et al. (2019), we propose using a simple non-stationary multi-armed bandit (Garivier & Moulines, 2008) to directly control the exploration rate and discount factor to maximize the episode return, and then provide this information to the value network of the agent as an input. Unlike Schaul et al. (2019), 1) it controls the exploration rate and discount factor (helping with long-term credit assignment), and 2) the bandit controls a family of state-action value functions that back up the effects of exploration and longer discounts, rather than linearly tilting a common value function by a fixed functional form.

In summary, our contributions are as follows:

Mainly a set of minor tweaks to NGU

1. A new parameterization of the state-action value function that decomposes the contributions of the intrinsic and extrinsic rewards. As a result, we significantly increase the training stability over a large range of intrinsic reward scales.

2. A *meta-controller*: an adaptive mechanism to select

which of the policies (parameterized by exploration rate and discount factors) to prioritize throughout the training process. This allows the agent to control the *exploration/exploitation trade-off* by dedicating more resources to one or the other.

3. Finally, we demonstrate for the first time performance that is above the human baseline across all Atari 57 games. As part of these experiments, we also find that simply re-tuning the backprop through time window to be twice the previously published window for R2D2 led to superior long-term credit assignment (e.g., in *Solaris*) while still maintaining or improving overall performance on the remaining games.

These improvements to NGU collectively transform it into the most general Atari 57 agent, enabling it to outperform the human baseline uniformly over all Atari 57 games. Thus, we call this agent: Agent57.

2. Background: Never Give Up (NGU)

Our work builds on top of the NGU agent, which combines two ideas: first, the curiosity-driven exploration, and second, distributed deep RL agents, in particular R2D2.

NGU computes an intrinsic reward in order to encourage exploration. This reward is defined by combining per-episode and life-long novelty. The per-episode novelty, r_t^{episodic}, rapidly vanishes over the course of an episode, and it is computed by comparing observations to the contents of an episodic memory. The life-long novelty, α_t, slowly vanishes throughout training, and it is computed by using a parametric model (in NGU and in this work Random Network Distillation (Burda et al., 2018) is used to this end). With this, the intrinsic reward r_t^i is defined as follows:

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

where $L = 5$ is a chosen maximum reward scaling. This leverages the long-term novelty provided by α_t, while r_t^{episodic} continues to encourage the agent to explore within an episode. For a detailed description of the computation of r_t^{episodic} and α_t, see (Puigdomènech Badia et al., 2020). At time t, NGU adds N different scales of the same intrinsic reward $\beta_j r_t^i$ ($\beta_j \in \mathbb{R}^+, j \in 0, \ldots N-1$) to the extrinsic reward provided by the environment, r_t^e, to form N potential total rewards $r_{j,t} = r_t^e + \beta_j r_t^i$. Consequently, NGU aims to learn the N different associated optimal state-action value functions $Q^*_{r_j}$ associated with each reward function $r_{j,t}$. The exploration rates β_j are parameters that control the degree of exploration. Higher values will encourage exploratory policies and smaller values will encourage exploitative policies. Additionally, for purposes of learning long-term credit assignment, each $Q^*_{r_j}$ has its own associated discount factor γ_j (for background and notations on Markov Decision Processes (MDP) see App. A). Since the intrinsic reward is typically much more dense than the extrinsic reward, $\{(\beta_j, \gamma_j)\}_{j=0}^{N-1}$ are chosen so as 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).

To learn the state-action value function $Q^*_{r_j}$, NGU trains a recurrent neural network $Q(x, a, j; \theta)$, where j is a one-hot vector indexing one of N implied MDPs (in particular (β_j, γ_j)), x is the current observation, a is an action, and θ are the parameters of the network (including the recurrent state). In practice, NGU can be unstable and fail to learn an appropriate approximation of $Q^*_{r_j}$ for all the state-action value functions in the family, even in simple environments. This is especially the case when the scale and sparseness of r_t^e and r_t^i are both different, or when one reward is more noisy than the other. We conjecture that learning a common state-action value function for a mix of rewards is difficult when the rewards are very different in nature. Therefore, in Sec. 3.1, we propose an architectural modification to tackle this issue.

Our agent is a deep distributed RL agent, in the lineage of R2D2 and NGU. As such, it decouples the data collection and the learning processes by having many actors feed data to a central prioritized replay buffer. A learner can then sample training data from this buffer, as shown in Fig. 2 (for implementation details and hyperparameters refer to App. E). More precisely, the replay buffer contains sequences of transitions that are removed regularly in a FIFO-manner. These sequences come from actor processes that interact with independent copies of the environment, and they are prioritized based on temporal differences errors (Kapturowski et al., 2018). The priorities are initialized by the actors and updated by the learner with the updated state-action value function $Q(x, a, j; \theta)$. According to those priorities, the learner samples sequences of transitions from the replay buffer to construct an RL

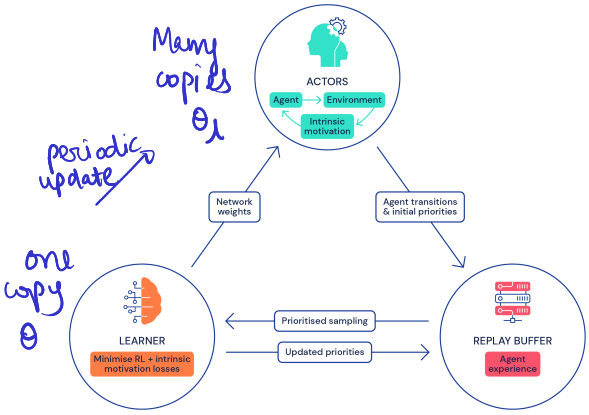


Figure 2. A schematic depiction of a distributed deep RL agent.

similar to A3C

loss. Then, it updates the parameters of the neural network $Q(x, a, j; \theta)$ by minimizing the RL loss to approximate the optimal state-action value function. Finally, each actor shares the same network architecture as the learner but with different weights. We refer as θ_l to the parameters of the $l-$th actor. The learner weights θ are sent to the actor frequently, which allows it to update its own weights θ_l. Each actor uses different values ϵ_l, which are employed to follow an ϵ_l-greedy policy based on the current estimate of the state-action value function $Q(x, a, j; \theta_l)$. In particular, at the beginning of each episode and in each actor, NGU uniformly selects a pair (β_j, γ_j). We hypothesize that this process is sub-optimal and propose to improve it in Sec. 3.2 by introducing a meta-controller for each actor that adapts the data collection process.

3. Improvements to NGU

3.1. State-Action Value Function Parameterization

The proposed architectural improvement consists in splitting the state-action value function in the following way:

$$Q(x, a, j; \theta) = Q(x, a, j; \theta^e) + \beta_j Q(x, a, j; \theta^i),$$

where $Q(x, a, j; \theta^e)$ and $Q(x, a, j; \theta^i)$ are the extrinsic and intrinsic components of $Q(x, a, j; \theta)$ respectively. The sets of weights θ^e and θ^i separately parameterize two neural networks with identical architecture and $\theta = \theta^i \cup \theta^e$. Both $Q(x, a, j; \theta^e)$ and $Q(x, a, j; \theta^i)$ are optimized separately in the learner with rewards r^e and r^i respectively, but with the same target policy $\pi(x) = \arg\max_{a \in \mathcal{A}} Q(x, a, j; \theta)$. More precisely, to train the weights θ^e and θ^i, we use the same sequence of transitions sampled from the replay, but with two different transformed Retrace loss functions (Munos et al., 2016). For $Q(x, a, j; \theta^e)$ we compute an extrinsic transformed Retrace loss on the sequence transitions with rewards r^e and target policy π, whereas for $Q(x, a, j; \theta^i)$ we compute an intrinsic transformed Retrace loss on the same sequence of transitions but with rewards r^i and target policy π. A reminder of how to compute a transformed Retrace loss on a sequence of transitions with rewards r and target policy π is provided in App. C.

In addition, in App. B, we show that this optimization of separate state-action values is equivalent to the optimization of the original single state-action value function with reward $r^e + \beta_j r^i$ (under a simple gradient descent optimizer). Even though the theoretical objective being optimized is the same, the parameterization is different: we use two different neural networks to approximate each one of these state-action values (a schematic and detailed figures of the architectures used can be found in App. F). By doing this, we allow each network to adapt to the scale and variance associated with their corresponding reward, and we also allow for the associated optimizer state to be separated for intrinsic and extrinsic state-action value functions.

Moreover, when a transformed Bellman operator (Pohlen et al., 2018) with function h is used (see App. A), we can split the state-action value function in the following way:

$$Q(x, a, j; \theta) = h\left(h^{-1}(Q(x, a, j; \theta^e)) + \beta_j h^{-1}(Q(x, a, j; \theta^i))\right).$$

In App. B, we also show that the optimization of separated transformed state-action value functions is equivalent to the optimization of the original single transformed state-action value function. In practice, choosing a simple or transformed split does not seem to play an important role in terms of performance (empirical evidence and an intuition behind this result can be found in App. H.3). In our experiments, we choose an architecture with a simple split which corresponds to h being the identity, but still use the transformed Retrace loss functions.

3.2. Adaptive Exploration over a Family of Policies

The core idea of NGU is to jointly train a family of policies with different degrees of exploratory behaviour using a single network architecture. In this way, training these exploratory policies plays the role of a set of auxiliary tasks that can help train the shared architecture even in the absence of extrinsic rewards. A major limitation of this approach is that all policies are trained equally, regardless of their contribution to the learning progress. We propose to incorporate a meta-controller that can adaptively select which policies to use both at training and evaluation time. This carries two important consequences. Firstly, by selecting which policies to prioritize during training, we can allocate more of the capacity of the network to better represent the state-action value function of the policies that are most relevant for the task at hand. Note that this is likely to change throughout the training process, naturally building a curriculum to facilitate training. As mentioned in Sec. 2, policies are represented by pairs of exploration rate and discount factor, (β_j, γ_j), which determine the discounted cumulative rewards to maximize. It is natural to expect policies with higher β_j and lower γ_j to make more progress early in training, while the opposite would be expected as training progresses. Secondly, this mechanism also provides a natural way of choosing the best policy in the family to use at evaluation time. Considering a wide range of values of γ_j with $\beta_j \approx 0$, provides a way of automatically adjusting the discount factor on a per-task basis. This significantly increases the generality of the approach.

We propose to implement the meta-controller using a non-stationary multi-arm bandit algorithm running independently on each actor. The reason for this choice, as opposed to a global meta-controller, is that each actor follows a different ϵ_l-greedy policy which may alter the choice of the

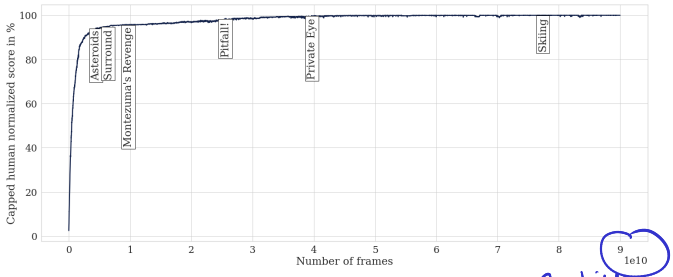


Figure 3. Capped human normalized score where we observe at which point the agent surpasses the human benchmark on the last 6 games.

optimal arm. Each arm j from the N-arm bandit is linked to a policy in the family and corresponds to a pair (β_j, γ_j). At the beginning of each episode, say, the k-th episode, the meta-controller chooses an arm J_k setting which policy will be executed. We use capital letters for the arm J_k because it is a random variable. Then the l-th actor acts ϵ_l-greedily with respect to the corresponding state-action value function, $Q(x, a, J_k; \theta_l)$, for the whole episode. The undiscounted extrinsic episode returns, noted $R^e_k(J_k)$, are used as a reward signal to train the multi-arm bandit algorithm of the meta-controller. The reward signal $R^e_k(J_k)$ is non-stationary, as the agent changes throughout training. Thus, a classical bandit algorithm such as Upper Confidence Bound (UCB; Garivier & Moulines, 2008) will not be able to adapt to the changes of the reward through time. Therefore, we employ a simplified sliding-window UCB with ϵ_{UCB}-greedy exploration. With probability $1 - \epsilon_{\text{UCB}}$, this algorithm runs a slight modification of classic UCB on a sliding window of size τ and selects a random arm with probability ϵ_{UCB} (details of the algorithms are provided in App. D).

Note that the benefit of adjusting the discount factor through training and at evaluation could be applied even in the absence of intrinsic rewards. To show this, we propose augmenting a variant of R2D2 with a meta-controller. In order to isolate the contribution of this change, we evaluate a variant of R2D2 which uses the same RL loss as Agent57. Namely, a transformed Retrace loss as opposed to a transformed n-step loss as in the original paper. We refer to this variant as R2D2 (Retrace) throughout the paper. In all other aspects, R2D2 (Retrace) is exactly the same algorithm as R2D2. We incorporate the joint training of several policies parameterized by $\{\gamma_j\}_{j=0}^{N-1}$ to R2D2 (Retrace). We refer to this algorithm as *R2D2 (bandit)*.

4. Experiments

We begin this section by describing our experimental setup. Following NGU, Agent57 uses a family of coefficients $\{(\beta_j, \gamma_j)\}_{j=0}^{N-1}$ of size $N = 32$. The choice of discounts $\{\gamma_j\}_{j=0}^{N-1}$ differs from that of NGU to allow for higher values, ranging from 0.99 to 0.9999 (see App. G.1 for details). The meta-controller uses a window size of $\tau = 160$ episodes and $\epsilon = 0.5$ for the actors and a window size of $\tau = 3600$ episodes and $\epsilon = 0.01$. All the other hyperparameters are identical to those of NGU, including the standard preprocessing of Atari frames. For a complete description of the hyperparameters and preprocessing we use, please see App. G.3. For all agents we run (that is, all agents except MuZero where we report numbers presented in Schrittwieser et al. (2019)), we employ a separate evaluator process to continuously record scores. We record the undiscounted episode returns averaged over 3 seeds and using a windowed mean over 50 episodes. For our best algorithm, Agent57, we report the results averaged over 6 seeds on all games to strengthen the significance of the results. On that average, we report the maximum over training as their final score, as done in Fortunato et al. (2017); Puigdomènech Badia et al. (2020). Further details on our evaluation setup are described in App. E.

In addition to using human normalized scores $\text{HNS} = \frac{\text{Agent}_{\text{score}} - \text{Random}_{\text{score}}}{\text{Human}_{\text{score}} - \text{Random}_{\text{score}}}$, we report the capped human normalized scores, $\text{CHNS} = \max\{\min\{\text{HNS}, 1\}, 0\}$. This measure is a better descriptor for evaluating general performance, as it puts an emphasis in the games that are below the average human performance benchmark. Furthermore, and avoiding any issues that aggregated metrics may have, we also provide all the scores that all the ablations obtain in all games we evaluate in App. H.1.

We structure the rest of this section in the following way: firstly, we show an overview of the results that Agent57 achieves. Then we proceed to perform ablations on each one of the improvements we propose for our model.

4.1. Summary of the Results

Tab. 1 shows a summary of the results we obtain on all 57 Atari games when compared to baselines. MuZero obtains the highest uncapped mean and median human normalized scores, but also the lowest capped scores. This is due to the fact that MuZero performs remarkably well in some games, such as *Beam Rider*, where it shows an uncapped score of 27469%, but at the same time catastrophically fails to learn in games such as *Venture*, achieving a score that is on par with a random policy. We see that the meta-controller improvement successfully transfers to R2D2: the proposed variant R2D2 (bandit) shows a mean, median, and CHNS that are much higher than R2D2 with the same Retrace loss. Finally, Agent57 achieves a median and mean that is greater than NGU and R2D2, but also its CHNS is 100%. This shows the generality of Agent57: not only it obtains a strong mean and median, but also it is able to obtain strong performance on the tail of games in which MuZero

Table 1. Number of games above human, mean capped, mean and median human normalized scores for the 57 Atari games.

Statistics	Agent57	R2D2 (bandit)	NGU	R2D2 (Retrace)	R2D2	MuZero
Capped mean	**100.00**	96.93	95.07	94.20	94.33	89.92
Number of games > human	**57**	54	51	52	52	51
Mean	4766.25	5461.66	3421.80	3518.36	4622.09	**5661.84**
Median	1933.49	2357.92	1359.78	1457.63	1935.86	**2381.51**
40th Percentile	1091.07	**1298.80**	610.44	817.77	1176.05	1172.90
30th Percentile	614.65	**648.17**	267.10	420.67	529.23	503.05
20th Percentile	**324.78**	303.61	226.43	267.25	215.31	171.39
10th Percentile	**184.35**	116.82	107.78	116.03	115.33	75.74
5th Percentile	**116.67**	93.25	64.10	48.32	50.27	0.03

and R2D2 catastrophically fail. This is more clearly observed when looking at different percentiles: up to the 20th percentile, Agent57 shows much greater performance, only slightly surpassed by R2D2 (bandit) when we examine higher percentiles. In Fig. 3 we report the performance of Agent57 in isolation on the 57 games. We show the last 6 games (in terms of number of frames collected by the agents) in which the algorithm surpasses the human performance benchmark. As shown, the benchmark over games is beaten in a long-tailed fashion, where Agent57 uses the first 5 billion frames to surpass the human benchmark on 51 games. After that, we find hard exploration games, such as *Montezuma's Revenge*, *Pitfall!*, and *Private Eye*. Lastly, Agent57 surpasses the human benchmark on *Skiing* after 78 billion frames. To be able to achieve such performance on *Skiing*, Agent57 uses a high discount (as we show in Sec. 4.4). This naturally leads to high variance in the returns, which leads to needing more data in order to learn to play the game. One thing to note is that, in the game of *Skiing*, the human baseline is very competitive, with a score of -4336.9, where -17098.1 is random and -3272 is the optimal score one can achieve.

In general, as performance in Atari keeps improving, it seems natural to concentrate on the tail of the distribution, i.e., pay attention to those games for which progress in the literature has been historically much slower than average. We now present results for a subset of 10 games that we call the *challenging set*. It consists of the six hard exploration games as defined in (Bellemare et al., 2016), plus games that require long-term credit assignment. More concretely, the games we use are: *Beam Rider*, *Freeway*, *Montezuma's Revenge*, *Pitfall!*, *Pong*, *Private Eye*, *Skiing*, *Solaris*, *Surround*, and *Venture*.

In Fig. 4 we can see the performance progression obtained from incorporating each one of the improvements we make on top of NGU. Such performance is reported on the selection of 10 games mentioned above. We observe that each one of the improvements results in an increment in final performance. Further, we see that each one of the improvements that is part of Agent57 is necessary in order to obtain the consistent final performance of 100% CHNS.

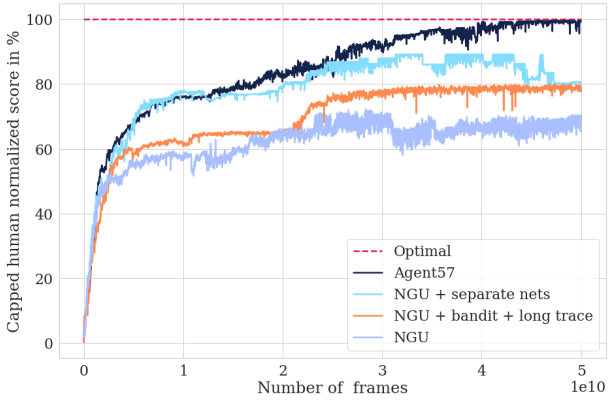


Figure 4. Performance progression on the 10-game *challenging set* obtained from incorporating each one of the improvements.

4.2. State-Action Value Function Parameterization

We begin by evaluating the influence of the state-action value function parametrization on a minimalistic gridworld environment, called "random coin". It consists of an empty room of size 15×15 where a coin and an agent are randomly placed at the start of each episode. The agent can take four possible actions (up, down, left right) and episodes are at most 200 steps long. If the agent steps over the coin, it receives a reward of 1 and the episode terminates. In Fig. 5 we see the results of NGU with and without the new parameterization of its state-action value functions. We report performance after 150 million frames. We compare the extrinsic returns for the policies that are the exploitative ($\beta_j = 0$) and the most exploratory (with the largest β_j in the family). Even for small values of the exploration rates ($\max_j \beta_j$), this setting induces very different exploratory and exploitative policies. Maximizing the discounted extrinsic returns is achieved by taking the shortest path towards the coin (obtaining an extrinsic return of one), whereas maximizing the augmented returns is achieved by avoiding the coin and visiting all remaining states (obtaining an extrinsic return of zero). In principle, NGU should be able to learn these policies jointly. However, we observe that the exploitative policy in NGU struggles to solve the task as intrinsic motivation reward scale increases. As we increase the scale of the intrinsic reward,

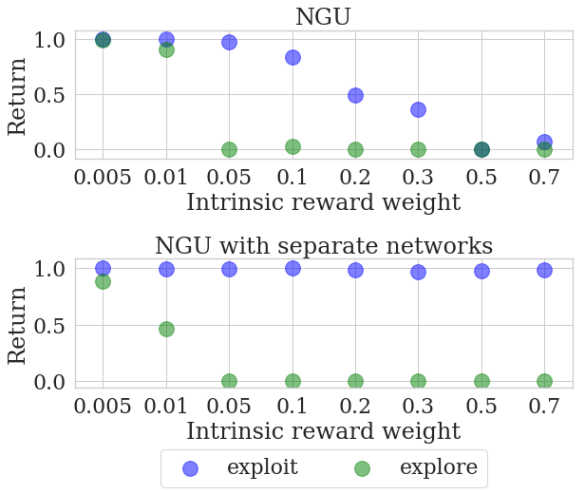


Figure 5. Extrinsic returns for the exploitative ($\beta_0 = 0$) and most exploratory ($\beta_{31} = \beta$) on "random coin" for different values of the intrinsic reward weight, β. *(Top)* NGU*(Bottom)* NGU with Separate networks for intrinsic and extrinsic values.

its value becomes much greater than that of the extrinsic reward. As a consequence, the conditional state-action value network of NGU is required to represent very different values depending on the β_j we condition on. This implies that the network is increasingly required to have more flexible representations. Using separate networks dramatically increases its robustness to the intrinsic reward weight that is used. Note that this effect would not occur if the episode did not terminate after collecting the coin. In such case, exploratory and exploitative policies would be allowed to be very similar: both could start by collecting the coin as quickly as possible. In Fig. 4 we can see that this improvement also translates to the *challenging set*. NGU achieves a much lower average CHNS than its separate network counterpart. We also observe this phenomenon when we incorporate the meta-controller. Agent57 suffers a drop of performance that is greater than 20% when the separate network improvement is removed.

We can also see that it is a general improvement: it does not show worse performance on any of the 10 games of the challenging set. More concretely, the largest improvement is seen in the case of *Surround*, where NGU obtains a score on par with a random policy, whereas with the new parametrization it reaches a score that is nearly optimal. This is because *Surround* is a case that is similar to the "random coin" environment mentioned above: as the player makes progress in the game, they have the choice to surround the opponent snake, receive a reward, and start from the initial state, or keep wandering around without capturing the opponent, and thus visiting new states in the world.

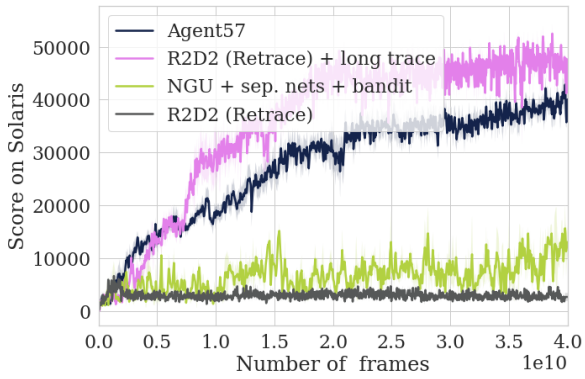


Figure 6. Solaris learning curves with small and long backprop through time window sizes for both R2D2 and Agent57.

4.3. Backprop Through Time Window Size

In this section we analyze the impact of having a backprop through time window size. More concretely, we analyze its impact on the base algorithm R2D2 to see its effect without NGU or any of the improvements we propose. Further, we also analyze its effect on Agent57, to see if any of the improvements on NGU overlap with this change. In both cases, we compare using backprop through time window sizes of 80 (default in R2D2) versus 160.

In aggregated terms over the *challenging set*, its effect seems to be the same for both R2D2 and Agent57: using a longer backprop through time window appears to be initially slower, but results in better overall stability and slightly higher final score. A detailed comparison over those 10 games is shown in App. H.2. This effect can be seen clearly in the game of *Solaris*, as observed in Fig. 6. This is also the game showing the largest improvement in terms of final score. This is again general improvement, as it enhances performance on all the *challenging set* games. For further details we report the scores in App. H.1.

4.4. Adaptive Exploration

In this section, we analyze the effect of using the meta-controller described in Sec. 3.1 in both the actors and the evaluator. To isolate the contribution of this improvement, we evaluate two settings: R2D2 and NGU with separate networks, with and without meta-controller. Results are shown in Fig. 7. Again, we observe that this is a general improvement in both comparisons. Firstly, we observe that there is a great value in this improvement on its own, enhancing the final performance of R2D2 by close to 20% CHNS. Secondly, we observe that the benefit on NGU with separate networks is more modest than for R2D2. This indicates that there is a slight overlap in the contributions of the separate network parameterization and the use of the meta-controller. The bandit algorithm can adaptively decrease the value of β when the difference in scale between

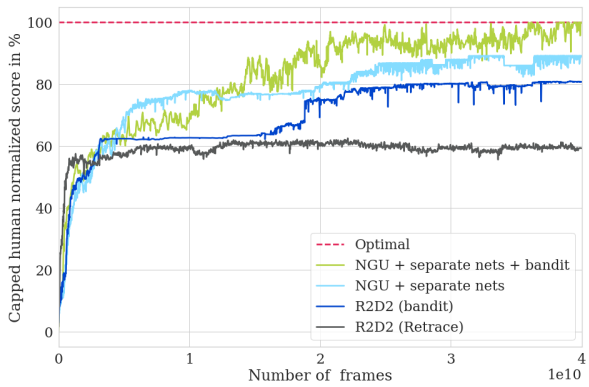


Figure 7. Performance comparison for adaptive exploration on the 10-game *challenging set*.

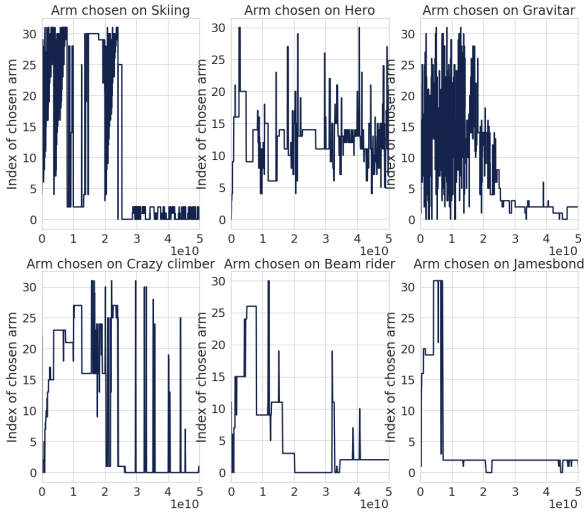


Figure 8. Best arm chosen by the evaluator of Agent57 over training for different games.

intrinsic and extrinsic rewards is large. Using the meta-controller allows to include very high discount values in the set $\{\gamma_j\}_{j=0}^{N}$. Specifically, running R2D2 with a high discount factor, $\gamma = 0.9999$ surpasses the human baseline in the game of *Skiing*. However, using that hyperparameter across the full set of games, renders the algorithm very unstable and damages its end performance. All the scores in the *challenging set* for a fixed high discount ($\gamma = 0.9999$) variant of R2D2 are reported in App. H.1. When using a meta-controller, the algorithm does not need to make this compromise: it can adapt it in a per-task manner.

Finally, the results and discussion above show why it is beneficial to use different values of β and γ on a per-task basis. At the same time, in Sec. 3 we hypothesize it would also be useful to vary those coefficients throughout training. In Fig. 8 we can see the choice of (β_j, γ_j) producing highest returns on the meta-controller of the evaluator across training for several games. Some games clearly have a preferred mode: on *Skiing* the high discount combination is quickly picked up when the agent starts to learn, and on *Hero* a high β and low γ is generally preferred at all times. On the other hand, some games have different preferred modes throughout training: on *Gravitar*, *Crazy Climber*, *Beam Rider*, and *Jamesbond*, Agent57 initially chooses to focus on exploratory policies with low discount, and, as training progresses, the agent shifts into producing experience from higher discount and more exploitative policies.

5. Conclusions

We present the first deep reinforcement learning agent with performance above the human benchmark on all 57 Atari games. The agent is able to balance the learning of different skills that are required to be performant on such diverse set of games: *exploration and exploitation* and *long-term credit assignment*. To do that, we propose simple improvements to an existing agent, *Never Give Up*, which has good performance on hard-exploration games, but in itself does not have strong overall performance across all 57 games. These improvements are i) using a different parameterization of the state-action value function, ii) using a meta-controller to dynamically adapt the novelty preference and discount, and iii) the use of longer backprop-through time window to learn from using the Retrace algorithm.

This method leverages a great amount of computation to its advantage: similarly to NGU, it is able to scale well with increasing amounts of computation. This has also been the case with the many recent achievements in deep RL (Silver et al., 2016; Andrychowicz et al., 2018; Vinyals et al., 2019). While this enables our method to achieve strong performance, an interesting research direction is to pursue ways in which to improve the data efficiency of this agent. Additionally, this agent shows an average capped human normalized score of 100%. However, in our view this by no means marks the end of Atari research, not only in terms of efficiency as above, but also in terms of general performance. We offer two views on this: firstly, analyzing the performance among percentiles gives us new insights on how general algorithms are. While Agent57 achieves great results on the first percentiles of the 57 games and holds better mean and median performance than NGU or R2D2, as MuZero shows, it could still obtain much better average performance. Secondly, as pointed out by Toromanoff et al. (2019), all current algorithms are far from achieving optimal performance in some games. To that end, key improvements to use might be enhancements in the representations that Agent57 and NGU use for exploration, planning (as suggested by the results achieved by MuZero) as well as better mechanisms for credit assignment (as highlighted by the results seen in *Skiing*).

Acknowledgments

We thank Daan Wierstra, Koray Kavukcuoglu, Vlad Mnih, Vali Irimia, Georg Ostrovski, Mohammad Gheshlaghi Azar, Rémi Munos, Bernardo Avila Pires, Florent Altché, Steph Hughes-Fitt, Rory Fitzpatrick, Andrea Banino, Meire Fortunato, Melissa Tan, Benigno Uria, Borja Ibarz, Andre Barreto, Diana Borsa, Simon Osindero, Tom Schaul, and many other colleagues at DeepMind for helpful discussions and comments on the manuscript.

References

Achiam, J., Edwards, H., Amodei, D., and Abbeel, P. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

Arjona-Medina, J. A., Gillhofer, M., Widrich, M., Unterthiner, T., Brandstetter, J., and Hochreiter, S. Rudder: Return decomposition for delayed rewards. In *Advances in Neural Information Processing Systems*, pp. 13544–13555, 2019.

Aytar, Y., Pfaff, T., Budden, D., Paine, T., Wang, Z., and de Freitas, N. Playing hard exploration games by watching youtube. In *Advances in Neural Information Processing Systems*, pp. 2930–2941, 2018.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. Successor features for transfer in reinforcement learning. In *Advances in neural information processing systems*, pp. 4055–4065, 2017.

Barto, A. G. Intrinsic motivation and reinforcement learning. In *Intrinsically motivated learning in natural and artificial systems*, pp. 17–47. Springer, 2013.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 06 2013.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Choi, J., Guo, Y., Moczulski, M., Oh, J., Wu, N., Norouzi, M., and Lee, H. Contingency-aware exploration in reinforcement learning. *arXiv preprint arXiv:1811.01483*, 2018.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Ferret, J., Marinier, R., Geist, M., and Pietquin, O. Credit assignment as a proxy for transfer in reinforcement learning. *arXiv preprint arXiv:1907.08027*, 2019.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Fortunato, M., Tan, M., Faulkner, R., Hansen, S., Badia, A. P., Buttimore, G., Deck, C., Leibo, J. Z., and Blundell, C. Generalization of reinforcement learners with working and episodic memory. In *Advances in Neural Information Processing Systems*, pp. 12448–12457, 2019.

Fu, J., Co-Reyes, J., and Levine, S. Ex2: Exploration with exemplar models for deep reinforcement learning. In *Advances in neural information processing systems*, pp. 2577–2587, 2017.

Garivier, A. and Moulines, E. On upper-confidence bound policies for non-stationary bandit problems, 2008.

Gregor, K., Rezende, D. J., and Wierstra, D. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Harutyunyan, A., Dabney, W., Mesnard, T., Azar, M. G., Piot, B., Heess, N., van Hasselt, H. P., Wayne, G., Singh, S., Precup, D., et al. Hindsight credit assignment. In *Advances in neural information processing systems*, pp. 12467–12476, 2019.

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., and Silver, D. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Hung, C.-C., Lillicrap, T., Abramson, J., Wu, Y., Mirza, M., Carnevale, F., Ahuja, A., and Wayne, G. Optimizing agent behavior over long time scales by transporting value. *Nature communications*, 10(1):1–12, 2019.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green,

T., Dunning, I., Simonyan, K., et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2018.

Ke, N. R., Goyal, A., Bilaniuk, O., Binas, J., Charlin, L., Pal, C., and Bengio, Y. Sparse attentive backtracking: Long-range credit assignment in recurrent networks. *arXiv preprint arXiv:1711.02326*, 2017.

Liu, Y., Luo, Y., Zhong, Y., Chen, X., Liu, Q., and Peng, J. Sequence modeling of temporal credit assignment for episodic reinforcement learning. *arXiv preprint arXiv:1905.13420*, 2019.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529, 2015.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1046–1054, 2016.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. In *Advances In Neural Information Processing Systems*, pp. 4026–4034, 2016.

Osband, I., Aslanides, J., and Cassirer, A. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 8617–8629, 2018.

Ostrovski, G., Bellemare, M. G., van den Oord, A., and Munos, R. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2721–2730. JMLR. org, 2017.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2): 265–286, 2007.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., Barth-Maron, G., Van Hasselt, H., Quan, J., Večerík, M., et al. Observe and look further: Achieving consistent performance on atari. *arXiv preprint arXiv:1805.11593*, 2018.

Puigdomènech Badia, A., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., and Blundell, C. Never give up: Learning directed exploration strategies. In *International Conference on Learning Representations*, 2020.

Puterman, M. L. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Savinov, N., Raichuk, A., Marinier, R., Vincent, D., Pollefeys, M., Lillicrap, T., and Gelly, S. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018.

Schaul, T., Borsa, D., Ding, D., Szepesvari, D., Ostrovski, G., Dabney, W., and Osindero, S. Adapting behaviour for learning progress, 2019.

Schmidhuber, J. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pp. 222–227, 1991.

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

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, O. X., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, pp. 2753–2762, 2017.

Toromanoff, M., Wirbel, E., and Moutarde, F. Is deep reinforcement learning really superhuman on atari? *arXiv preprint arXiv:1908.04683*, 2019.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Weber, T., Heess, N., Buesing, L., and Silver, D. Credit assignment techniques in stochastic computation graphs. *arXiv preprint arXiv:1901.01761*, 2019.

Xu, Z., van Hasselt, H. P., and Silver, D. Meta-gradient reinforcement learning. In *Advances in neural information processing systems*, pp. 2396–2407, 2018.

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 $h_{-1} = 0$.
- $j_{t-1} = j$: the 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

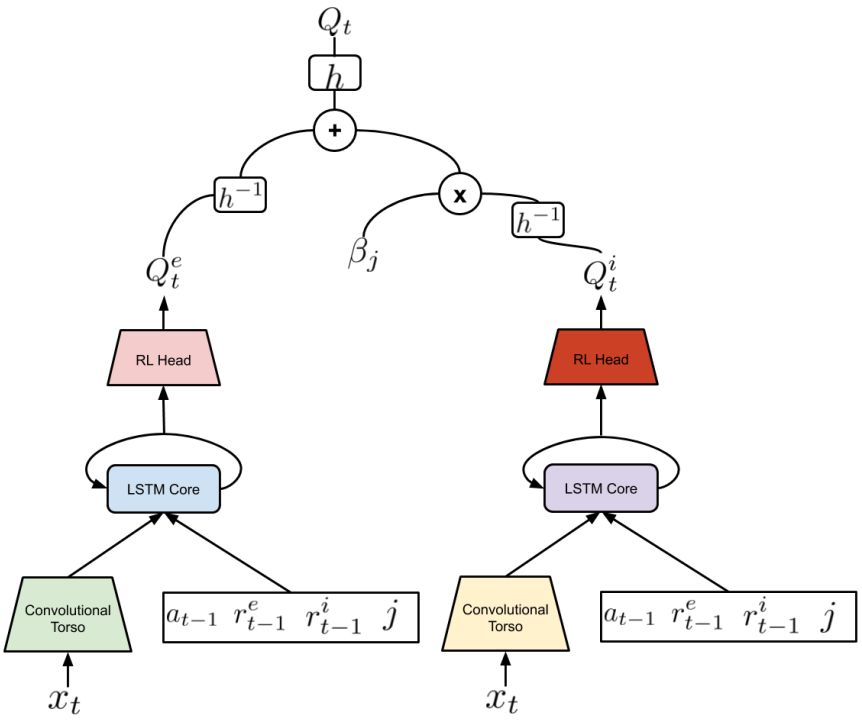


Figure 9. Sketch of the Agent57.

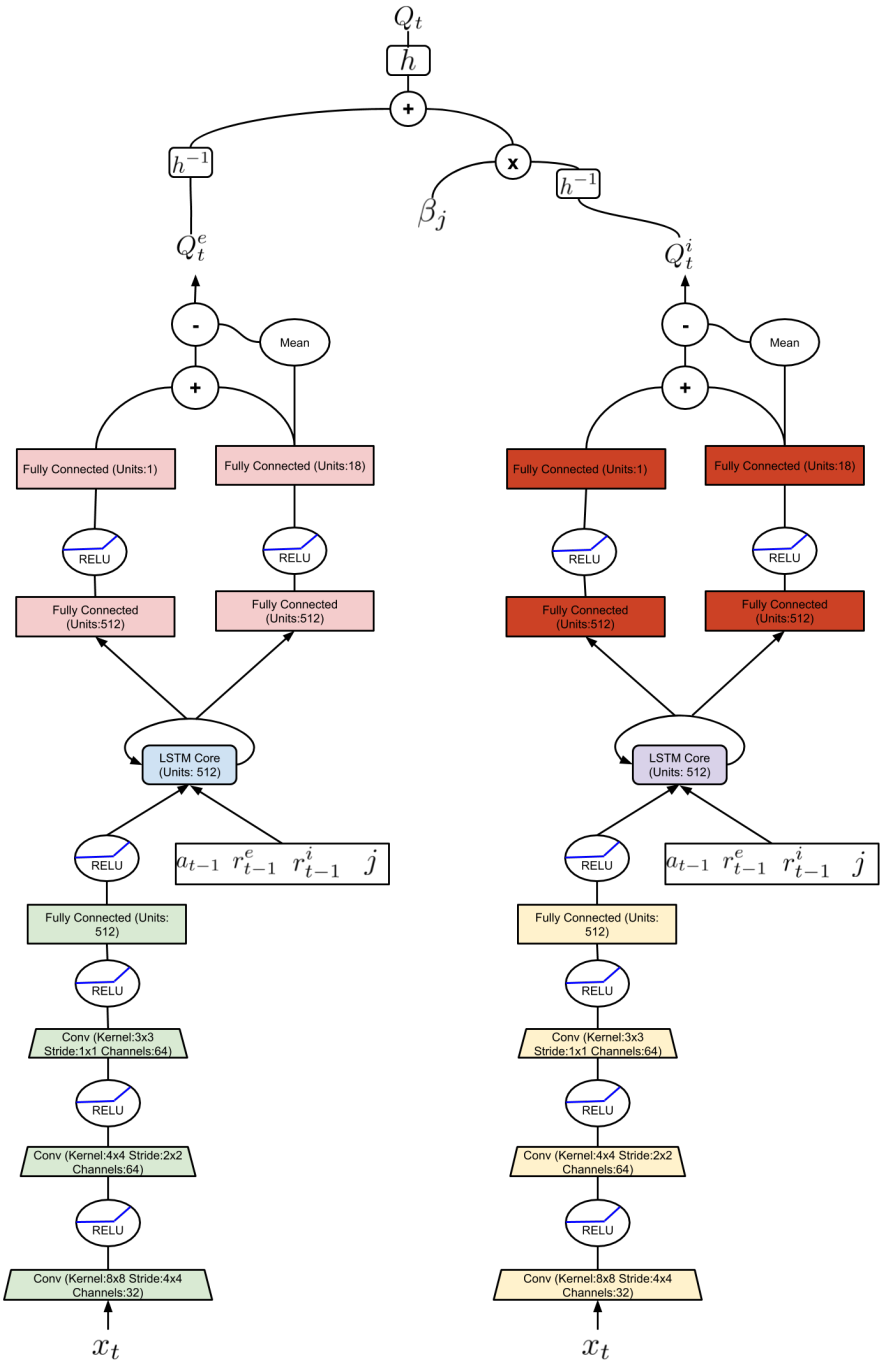
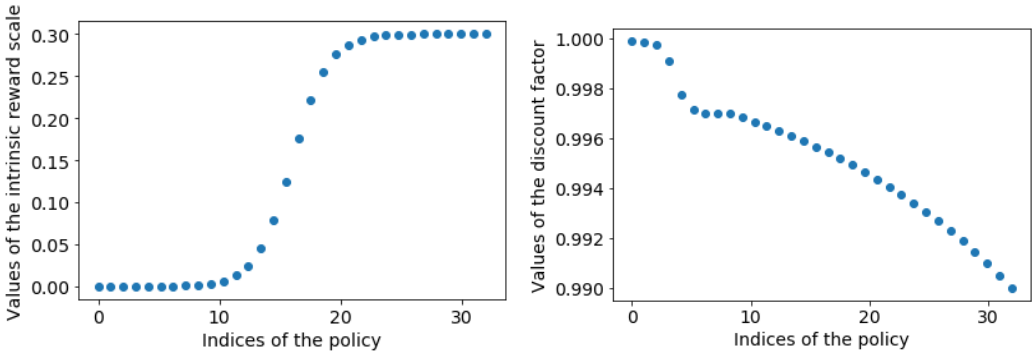


Figure 10. Detailed Agent57.

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	Agent57 small trace
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

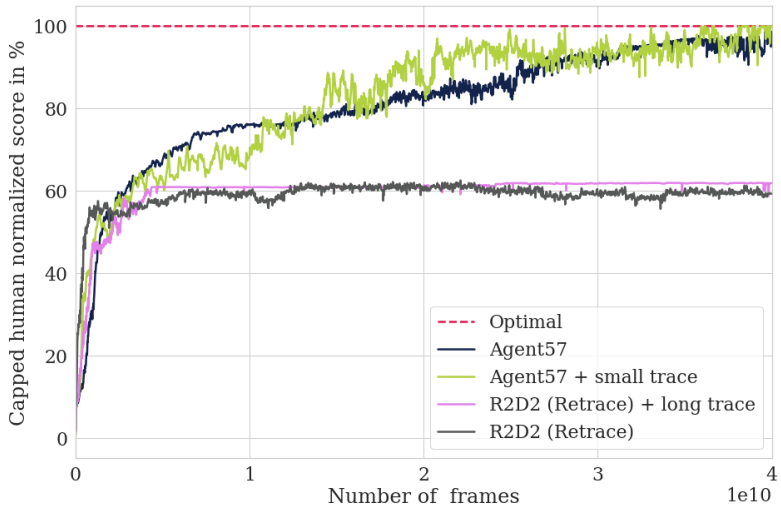


Figure 12. Performance comparison for short and long backprob window length on the 10-game *challenging set*.

H.3. Identity versus h-transform mixes comparison

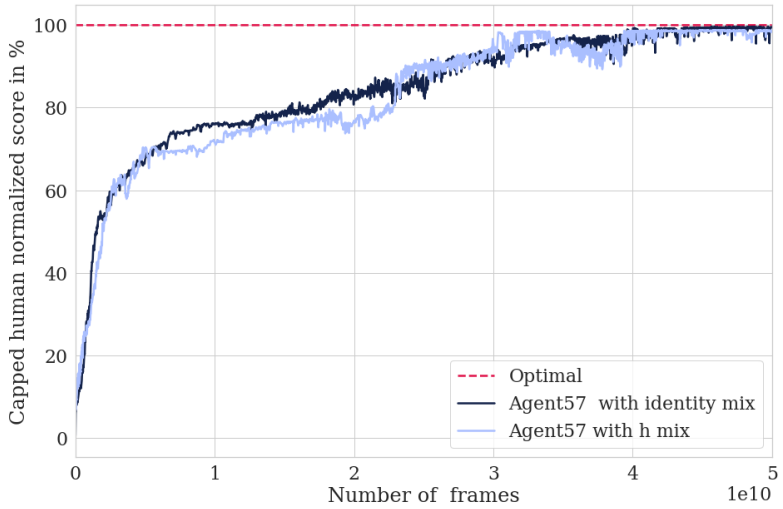


Figure 13. Performance comparison for identity versus h-transform mixes on the 10-game *challenging set*.

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 $Q_k^e(x,a) + \beta Q_k^i(x,a)$ and $h^{-1}(Q_k^e(x,a)) + \beta h^{-1}(Q_k^i(x,a))$ 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 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	464232.43 ± 7988.66	**741812.63**
amidar	1719.50	5.80	29660.08 ± 880.39	**31331.37 ± 817.79**	28634.39
assault	742.00	222.40	67212.67 ± 6150.59	110100.04 ± 346.06	**143972.03**
asterix	8503.30	210.00	991384.42 ± 9493.32	**999354.03 ± 12.94**	998425.00
asteroids	47388.70	719.10	150854.61 ± 16116.72	431072.45 ± 1799.13	**6785558.64**
atlantis	29028.10	12850.00	1528841.76 ± 28282.53	1660721.85 ± 14643.83	**1674767.20**
bank heist	753.10	14.20	23071.50 ± 15834.73	**27117.85 ± 963.12**	1278.98
battle zone	37187.50	2360.00	934134.88 ± 38916.03	**992600.31 ± 1096.19**	848623.00
beam rider	16926.50	363.90	300509.80 ± 13075.35	390603.06 ± 23304.09	**4549993.53**
berzerk	2630.40	123.70	61507.83 ± 26539.54	77725.62 ± 4556.93	**85932.60**
bowling	160.70	23.10	251.18 ± 13.22	161.77 ± 99.84	**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	863.92 ± 0.08	**864.00**
centipede	12017.00	2090.90	412847.86 ± 26087.14	908137.24 ± 7330.99	**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	**729482.83 ± 87975.74**	458315.40
defender	18688.90	2874.50	677642.78 ± 16858.59	730714.53 ± 715.54	**839642.95**
demon attack	1971.00	152.10	143161.44 ± 220.32	143913.32 ± 92.93	**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	2378.66 ± 3.66	**2382.44**
fishing derby	-38.70	-91.70	86.97 ± 3.25	90.34 ± 2.66	**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	309077.30 ± 274879.03	**631378.53**
gopher	2412.50	257.60	117777.08 ± 3108.06	129736.13 ± 653.03	**130345.58**
gravitar	3351.40	173.00	19213.96 ± 348.25	**21068.03 ± 497.25**	6682.70
hero	30826.40	1027.00	**114736.26 ± 49116.60**	49339.62 ± 4617.76	49244.11
ice hockey	0.90	-11.20	63.64 ± 6.48	**86.59 ± 0.59**	67.04
jamesbond	302.80	29.00	135784.96 ± 9132.28	**158142.36 ± 904.45**	41063.25
kangaroo	3035.00	52.00	**24034.16 ± 12565.88**	18284.99 ± 817.25	16763.60
krull	2665.50	1598.00	251997.31 ± 20274.39	245315.44 ± 48249.07	**269358.27**
kung fu master	22736.30	258.50	206845.82 ± 11112.10	**267766.63 ± 2895.73**	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	62595.90 ± 1755.82	**243401.10**
name this game	8049.00	2292.30	54386.77 ± 6148.50	138030.67 ± 5279.91	**157177.85**
phoenix	7242.60	761.40	908264.15 ± 28978.92	**990638.12 ± 6278.77**	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	**777071.30 ± 190653.94**	72276.00
riverraid	17118.00	1338.50	63318.67 ± 5659.55	93569.66 ± 13308.08	**323417.18**
road runner	7845.00	11.50	243025.80 ± 79555.98	593186.78 ± 88650.69	**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	**-3851.44 ± 517.52**	-29968.36
solaris	12326.70	1236.30	44199.93 ± 8055.50	**67306.29 ± 10378.22**	56.62
space invaders	1668.70	148.00	48680.86 ± 5894.01	67898.71 ± 1744.74	**74335.30**
star gunner	10250.00	664.00	839573.53 ± 67132.17	**998600.28 ± 218.66**	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	460596.49 ± 3139.33	**476763.90**
tutankham	167.60	11.40	**2354.91 ± 3421.43**	483.78 ± 37.90	491.48
up n down	11693.20	533.40	623805.73 ± 23493.75	702700.36 ± 8937.59	**715545.61**
venture	1187.50	0.00	**2623.71 ± 442.13**	2258.93 ± 29.90	0.40
video pinball	17667.90	0.00	992340.74 ± 12867.87	**999645.92 ± 57.93**	981791.88
wizard of wor	4756.50	563.50	157306.41 ± 16000.00	183090.81 ± 6070.10	**197126.00**
yars revenge	54576.90	3092.90	998532.37 ± 375.82	**999807.02 ± 54.85**	553311.46
zaxxon	9173.30	32.50	249808.90 ± 58261.59	370649.03 ± 19761.32	**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

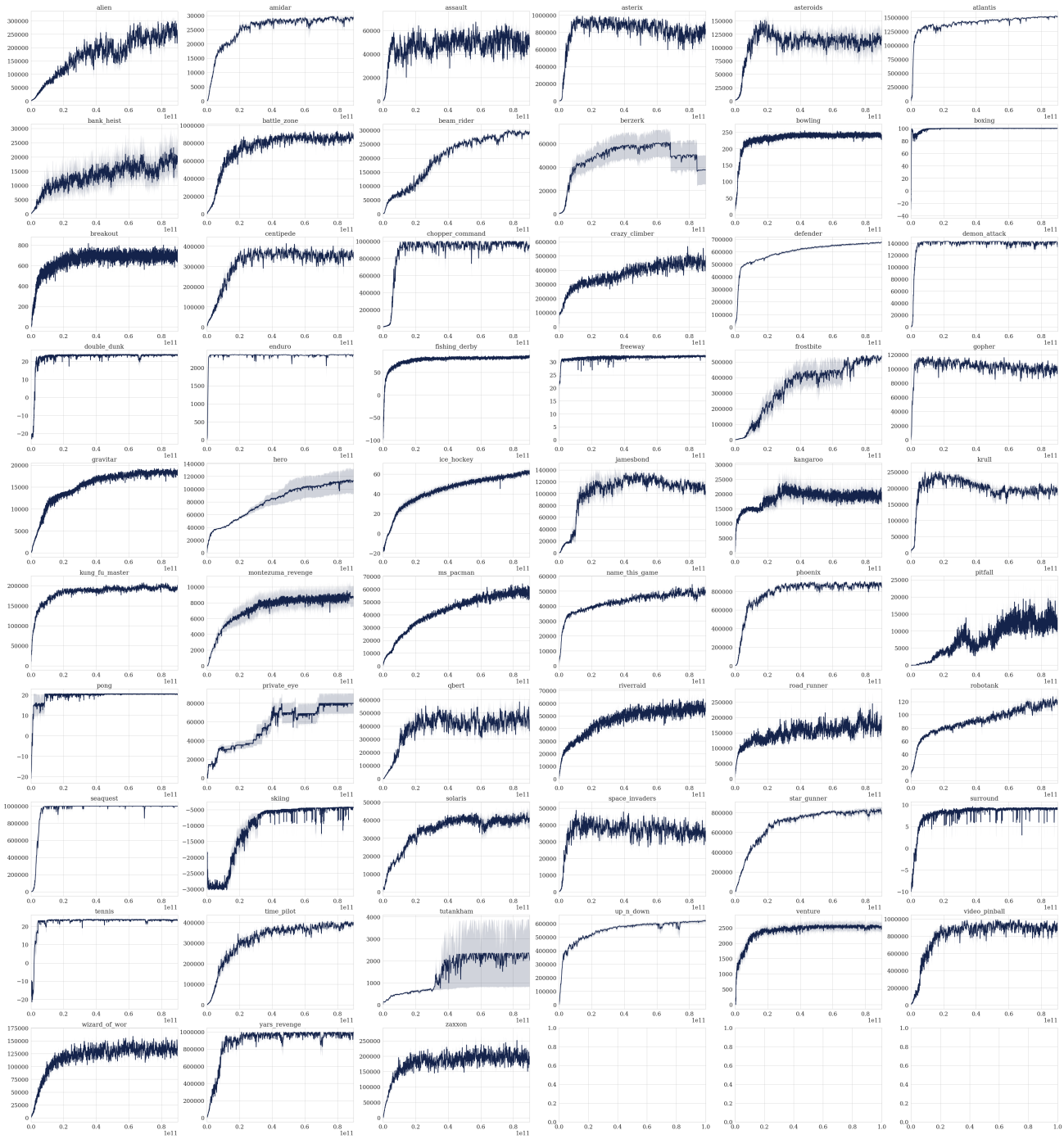


Figure 14. Learning curves for Agent57 on Atari57.

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.