An Evaluation Study of Intrinsic Motivation Techniques applied to Reinforcement Learning over Hard Exploration Environments

Alain Andres[1,2], Esther Villar-Rodriguez[1], and Javier Del Ser[1,2]

[1] TECNALIA, Basque Research & Technology Alliance (BRTA), 48160 Derio, Spain
[2] University of the Basque Country (UPV/EHU), 48013 Bilbao, Spain
{alain.andres, esther.villar, javier.delser}@tecnalia.com

Abstract. In the last few years, the research activity around reinforcement learning tasks formulated over environments with sparse rewards has been especially notable. Among the numerous approaches proposed to deal with these hard exploration problems, intrinsic motivation mechanisms are arguably among the most studied alternatives to date. Advances reported in this area over time have tackled the exploration issue by proposing new algorithmic ideas to generate alternative mechanisms to measure the novelty. However, most efforts in this direction have overlooked the influence of different design choices and parameter settings that have also been introduced to improve the effect of the generated intrinsic bonus, forgetting the application of those choices to other intrinsic motivation techniques that may also benefit of them. Furthermore, some of those intrinsic methods are applied with different base reinforcement algorithms (e.g. PPO, IMPALA) and neural network architectures, being hard to fairly compare the provided results and the actual progress provided by each solution. The goal of this work is to stress on this crucial matter in reinforcement learning over hard exploration environments, exposing the variability and susceptibility of avant-garde intrinsic motivation techniques to diverse design factors. Ultimately, our experiments herein reported underscore the importance of a careful selection of these design aspects coupled with the exploration requirements of the environment and the task in question under the same setup, so that fair comparisons can be guaranteed.

Keywords: Reinforcement Learning · Intrinsic Motivation · Exploration-Exploitation · Hard Exploration · Sparse Rewards.

1 Introduction

Over decades, Reinforcement Learning (RL) has been widely acknowledged as a rich and ever-growing research area within Artificial Intelligence aimed to efficiently deal with complex tasks [1,2]. One of the key components of the success of RL algorithms is to define a suitable reward function that reflects the objective of the task at hand. However, the design of appropriate reward functions

is often difficult – even unfeasible – depending on the peculiarities of the environment and task to be optimized. In this context, the study of environments with so-called *sparse rewards* has gained attention in the last few years. In such scenarios, the RL agent is just positively rewarded when accomplishing the goal, which is representative of manifold problems that arise from real-world applications [3]. Nevertheless, such *hard exploration* RL problems are more complex to address due to sparse informative feedback delivered by the environment, requiring effective means to balance between exploration and exploitation during the agent's learning process.

The aforementioned challenge can be overcome through Intrinsic Motivation (IM, [4]), Imitation Learning [5] and Inverse Reinforcement Learning [6], among other strategies. This work gravitates around the first (IM), which is used to encourage the agent to explore the environment by its inherent satisfaction of curiosity [7]. In practice, the concept of *curiosity* is translated to the RL domain in the form of an intrinsic bonus r_i, which is combined with the extrinsic reward provided by the environment $r = r_e + \beta r_i$ through a weighting factor β. Besides proposing different ways to generate the intrinsic reward r_i, current state-of-the-art algorithms (e.g., RIDE [8], NGU [9]) also apply novel methods to weight and scale such rewards, being those applicable to prior approaches such as Intrinsic Curiosity Module (ICM, [10]) and Random Network Distillation (RND, [11]). Unfortunately, when different IM-based schemes are compared to each other, those reward scaling techniques are not always in use, making it unclear whether the identified performance gaps are due to the exploration methods themselves or must be attributed to other design choices (i.e., the variation of intrinsic coefficient weights or the architecture of the models inside the RL agent).

Analogously to what is claimed in other performance evaluation works reported recently in [12,13], a fundamental matter in this research area is to discriminate which design criteria impact most on the performance of the RL agent. This is specially relevant in hard exploration environments, since it is known that under such circumstances, the proficiency of the agent is very sensitive regarding the configuration of its compounding modules. For this reason, this manuscript aims to fairly evaluate IM-based solutions present in the state of the art trying to decouple the solver approach from additional weighting and scaling techniques. Under this rationale, this work also incorporates the naive version of IM modules to study their benefit and ascertain the actual advantage of the algorithmic proposal when generating intrinsic rewards. Furthermore, the impact of having different neural network architectures in actor-critic agents and IM modules poses another question that lacks an informed answer in the current literature.

To sum up, this paper investigates the quantitative impact of different design choices when implementing IM-based techniques to understand their relevance when used in agents deployed over sparse reward scenarios. Hence, our contributions are three-fold: (1) we adopt curiosity mechanisms with different implementation choices that impact on how the intrinsic rewards are processed, (2) we conduct a study with multiple current state-of-the-art intrinsic motivation techniques where we compare them fairly in order to evaluate the improvement

of generating rewards with different approaches; and (3) we break down experiments, results and conclusions in the interest of providing the reader with independent performance analysis of the set of modules and parameterizations.

The rest of the manuscript is structured as follows: Section 2 overviews works related to intrinsic motivation in RL, while Section 3 details the factors and the choices to be taken into account when resorting to IM techniques. Next, Section 4 presents the experimental setup designed to achieve empirical evidence. Section 5 discusses the obtained results. Finally, Section 6 concludes the paper and outlines future research to be developed from this research on.

2 Related Work and Contribution

Before proceeding with the details of this work, we briefly review insights coming from recent research about intrinsic motivation mechanisms to deal with sparse rewards. In the absence of a dense reward function and/or when having hard exploration problems, intrinsic motivation mechanisms have turned up as an effective workaround to overcome poor exploration behaviour. These techniques generate artificial intrinsic rewards based on the novelty of a state[3], which relates to how curious an agent will be when arriving to that state. The less novel a state is, the less curious the agent should be [4]. In this context, several approaches have been proposed up to now to generate such exploration bonuses.

One mechanism to generate the aforementioned intrinsic rewards is by adopting a visitation count strategy, also referred to as *count-based* methods. In this case, intrinsic rewards are assumed to be inversely proportional to the number of counts $N(s)$ that a given state s has been visited, e.g. $r_i^{counts} = 1/\sqrt{N(s)}$. This is a simple, yet effective, solution to quantify the degree to which a state is *unknown* for the agent. However, counts are only applicable when dealing with discrete state spaces. Contrarily, when having more complex domains with continuous state spaces, density models [14], hash functions [15] and also successor features [16] can be applied to extend the concept of counts.

An alternative strategy to produce intrinsic motivation rewards is the use of *prediction-error* methods which, as their name suggests, generate intrinsic rewards based on the error when predicting the consequence of an agent's action in the environment. The aforementioned Intrinsic Curiosity Module (ICM) proposed in [10] belongs to this family of strategies, and operates by learning a state representation that just models the elements that the agent can control and those elements that can affect him. For this purpose, the intrinsic reward is generated based on the prediction error of the next state in a learned latent space:

$$r_i^{ICM} = ||\widehat{\phi}(s_{t+1}) - \phi(s_{t+1})||_2, \tag{1}$$

where $\phi(\cdot)$ denotes the learned latent space mapping; $\widehat{\phi}(s_{t+1})$ is an estimation taking into account $\phi(s_t)$ and the actual action a_t; s_t is the state visited at time t; and $||\cdot||_2$ stands for the L_2 (Euclidean) norm.

[3] Depending on the task under consideration, the novelty can be associated to the very last performed action and/or the next state visited by the agent in the trajectory.

Another approach is the use of RND introduced in [11]. Under this strategy, two identical networks are randomly initialized, where one of the networks takes the role of predictor $\widehat{\phi}$ aiming to mimic the output of the other network – namely, the target $\phi(\cdot)$, whose parameters are fixed after initialization. The reward is generated as an MSE loss between the outputs of both networks:

$$r_i^{RND} = ||\widehat{\phi}(s_{t+1}) - \phi(s_{t+1})||^2. \tag{2}$$

Built upon the idea of ICM, a recent work [8] introduced RIDE to use the same mechanism to learn the state embeddings, but they differ on how exploration bonuses are generated. In RIDE, this bonus is given by the difference between two consecutive states in their latent space:

$$r_i^{RIDE} = ||\phi(s_{t+1}) - \phi(s_t)||_2. \tag{3}$$

With this change, RIDE encourages the agent to perform actions that have an impact on the environment. Moreover, in the spirit of combining *experiment-* and *episode-level* [17] exploration to avoid the agent going back and forth between a sequence of states, the reward is discounted by the episodic state visitation counts:

$$r_i^{RIDE} = \frac{||\phi(s_{t+1}) - \phi(s_t)||_2}{\sqrt{N_{ep}(s_{t+1})}}, \tag{4}$$

where $N_{ep}(s_{t+1})$ denotes the episodic count of visits of state s_{t+1}. Following this idea of combining two levels of exploration (*experiment-* and *episode-*), the Never-Give-Up (NGU) approach in [9] employs different intrinsic weights set in several parallel agents feeding the same network, which parameterizes each agent by making the neural network subject to the intrinsic coefficient used by each of them. A more aggressive strategy is the contribution of [18], BeBold/NoveID, a solution that goes beyond the boundaries of explored regions which only rewards (intrinsically) the first time the agent visits a given state in an episode. The Fast and Slow (FaSo) intrinsic curiosity introduced in [19] combines local and global exploration by generating two different intrinsic rewards, depending on the quality of the reconstruction of two contexts built from the same state.

Rather than proposing a new intrinsic generation module, the present work offers a study combining different design choices made in recent solutions and fairly compare them under equal experimental conditions. This being said, other benchmarks/studies have been done in recent times: to begin with, [20] evaluates the performance of different exploration bonuses (pseudo-counts, ICM, RND and noisy networks) in the whole Atari 2600 suite with Rainbow [21]. By contrast, [22] carried out a large-scale study based exclusively on prediction error bonuses (ICM) over 54 environments, where they investigated the efficacy of using different feature learning methods with Proximal Policy Optimization (PPO, [23]). Our work also connects with [12,13,24], a series of evaluation studies aimed to understand what choices among high- and low-level algorithmic options affect the learning process: as such, the studies in [12,13] focus on on-policy deep actor-critic methods (examining different policy losses, architectures and advantage estimators), whereas [24] addresses Adversarial Imitation Learning related decisions (multiple reward functions and observation normalization methods).

Contribution: To the best of our knowledge, there is no prior work that exhaustively evaluates different choices for the implementation of intrinsic motivation strategies. Our study takes a step further by analyzing different weight and scale strategies for the combination of intrinsic and extrinsic rewards, as well as the impact of adopting different neural networks architectures and dimensions. The design choices here evaluated are applicable to any intrinsic curiosity generation module, so that conclusions about which ones are the most suitable given a task and an environment with sparse rewards can be drawn.

3 Methodology of the Study

After reviewing different solutions proposed in the literature to cope with hard exploration issues with IM techniques, we now proceed by describing the methodology adopted in this study to gauge the advantages and drawbacks of design choices that are present in some of them, giving an informed hint of their utility when extrapolated to the rest of IM solutions. From an overarching point of view, the methodology is driven by the pursuit of responses to three research questions (RQ):

- RQ1: Does the use of a static, parametric or adaptive decaying intrinsic coefficient weight β affect the agent's training process?
- RQ2: Which is the impact of using episodic counts to scale the intrinsic bonus? Is it better to use episodic counts than to just consider the first time a given state is visited by the agent?
- RQ3: Is the choice of the neural network architecture crucial for the agent's performance and learning efficiency?

Departing from these questions, the following methodology has been devised:

3.1 RQ1: Varying the Intrinsic Reward Coefficient β

In general, it is not advisable to combine raw extrinsic and intrinsic reward signals directly due to their potentially diverging value scales. Moreover, even if taking values from comparable ranges, the agent could need to grant more importance to exploration than to exploitation at specific periods. In fact, in sparse rewards settings, the explorer role of the agent must be strengthen and enlarged in comparison to the exploitative behaviour to guide the agent by an artificial bonus in the absence of knowledge about the target task. This balance between exploration and exploitation is usually controlled by the intrinsic reward coefficient β, whose value is often tuned manually depending on the environment and task to be accomplished. A priory, this value might be fixed and kept unaltered, or dynamically updated, as is further explained in what follows:

Static β: commonly, the β coefficient is stationary along the whole training. In such cases, we refer to this fixed and default value as β_s. On this basis, diverse

fixed intrinsic coefficient values can be used to learn a family of policies with different exploration-exploitation balances, so as to concentrate on maximizing the extrinsic reward (a policy with $\beta = 0$) while maintaining a degree of exploration (rest of policies with $\beta > 0$) [9]. Contrarily to the rest of approaches, when using multiple (fixed) intrinsic coefficients training more than one agent is required.

Dynamic β: to focus on the extrinsic signals provided by the environment, it is interesting to modulate the weight given to the intrinsic rewards generated by the agent in a dynamic fashion. Without loss of generality, in our work we consider two different options: parametric decay and adaptive decay. For the *parametric* decay, the value of β decreases by following a modified sigmoid function, which parametrically controls the smoothness of the decay:

$$\beta_t = A + \frac{K - A}{\left(1 + \exp\left(-16B\left(1 - \frac{t}{F}\right)\right)\right)^{20}} \tag{5}$$

where K is a value proven to deliver a good performance and well balanced trade-off between exploration and exploitation (e.g., the fixed value β_s that one could select under a fixed β strategy); A is the final value of β, which can be defined from K (e.g. $A = K/100$) to reflect that at the end of the learning process, the agent should receive hardly any intrinsic signal bonus; and F denotes the number of frames (= sample, steps) we expect the whole train to have. Moreover, B permits to control the *smoothness* of the progression of β throughout the training (Figure 1). Note that this parametric decay can also be used to sample different β values for each policy learned by means of the approach with multiple static intrinsic coefficients β, by defining F as the number of agents.

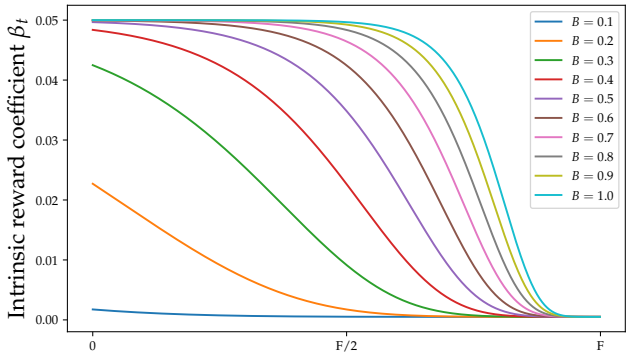


Fig. 1: Example of the parametric decay evolution of β_t for multiple values of the smoothness control parameter B, with $K = 0.05, A = 0.0005$ and $F = 2e7$.

In turn, we can vary the intrinsic coefficient by adopting an `adaptive decay strategy`. Motivated by [25] for concurrent environments, we propose to calculate a decay factor d_i^τ[4] based on the ratio between the agent's intrinsic return

[4] Rollout is denoted as τ, whereas the *i-th* rollout is denoted as τ_i.

at the current rollout, G_i^τ, and the averaged historical intrinsic return values in past rollouts H^τ:

$$\beta_i^\tau = \beta_s d_i^\tau = \beta_s \min\left[\frac{G_i^\tau}{H^\tau}, 1\right] = \beta_s \min\left[\frac{G_i^\tau}{\frac{1}{K}\sum_{k=0}^{K=i} G_k^\tau}, 1\right], \tag{6}$$

where K is the total number of rollouts the agent collected from the beginning of the training process up to the present rollout i. Consequently, under this rationale the agent is discouraged from exploring those trajectories that are more familiar than the average and means less novelty. Furthermore, the intrinsic return during the training may vary due to the non-stationary nature of the intrinsic reward generation process. Thereby, to stabilize the training, instead of leveraging the whole historical data, we also propose the use of a moving average with a sliding window, H_ω^τ, which strictly considers just the latest returns and avoids the case of discouraging the exploration due to large initial intrinsic returns that may well bias the decay factor calculation.

3.2 RQ2: Episodic State Counts versus First-Visit Scaling

As defined in [17], there are different periods in which the exploration mode can be carried out: *step-level*, *experiment-level*, *episode-level*, or *intra-episodic*. Over the years the use of *step-level* exploration (i.e. ϵ-greedy) has proven to yield good results in a diversity of simple RL environments. However, advances in learning algorithms have paved the way towards RL problems of higher complexity, in which the exploration is one of the critical parts to be addressed. As has been already argued in the introduction, hard exploration problems can be tackled by letting the agent explore the environment by its inherent satisfaction (intrinsic motivation) rather than being guided by environment provided extrinsic feedback signals. Nevertheless, intrinsic motivation techniques are prone to a quick vanishing of the rewards over the course of the training, reducing attractiveness as the training evolves. This condition is exacerbated when facing long-time horizon problems [19]. Actually, by analyzing the rewards obtained during a concrete episode, few differences in terms of novelty are appreciated between similar/close states, even if one has been already visited and the other remains unexplored. This is due to the persistence of curiosity-related information from past episodes (*experiment-level*), which is propagated forward during the agent's training leaving little novelty difference between similar (even identical) states inside the scope of the same episode. Additionally, in environments where state transitions are reversible, using intrinsic rewards to guide the exploration can lead into an agent bouncing back and forth between sequences of states that are more novel than others in the same episode [8,18].

As a solution to this issue, recent studies [8,9,18,19,26] have introduced a visitation count term so that they combine two degrees of novelty rather than just one: local (*episode-level*) and global (*experiment-level*). By virtue of episodic visitation counts, the agent is encouraged to visit as many different states as possible within an episode. However, approaches at the forefront of the state of the

art (i.e., ICM, RND) do not implement this idea to scale their rewards. In this context, it is unclear whether new proposed IM modules outperform previous approaches due to state-count regularization or to conceptually new algorithmic schemes. If state-counts regularization contributed to improve the performance, already proposed IM schemes that do not implement it and also future IM methods could adopt this strategy to meliorate their designs. Additionally, our experimentation incorporates a more aggressive variation that rewards the agent only when it visits a given state for the first time within the episode [18].

3.3 RQ3: Sensitiveness to the Neural Network Architectures

In the literature related to RL, plenty of network architecture proposals have been used to solve any given problem. As an example, the work in [27] simplified the architectures previously proposed in [8], yet achieving similar results[5]. However, they rely on different base RL algorithms (PPO [23] and IMPALA [28], respectively), thereby hindering a fair comparison, a proper interpretability and attribution of the reported performance results.

To avoid this issue, our specific experimentation evaluates the effect of the network architecture on the performance of the RL agent by considering a fixed RL algorithm and IM module, and by assessing several network configurations. By reporting the dimensions and characteristics of different neural network architectures and the performance of RL agents using them, we can gain intuition about the performance improvement (degradation) incurred when increasing (decreasing) the complexity of the neural architectures in use. Our experiments also measure the required amount of time when using those architectures, so that latency implications can be examined. This third research question is also aligned with practical concerns arising when deciding on which implementation is more suitable for a real-world deployment, specially in resource-constrained scenarios (e.g. embedded robotic devices).

4 Experimental Setup

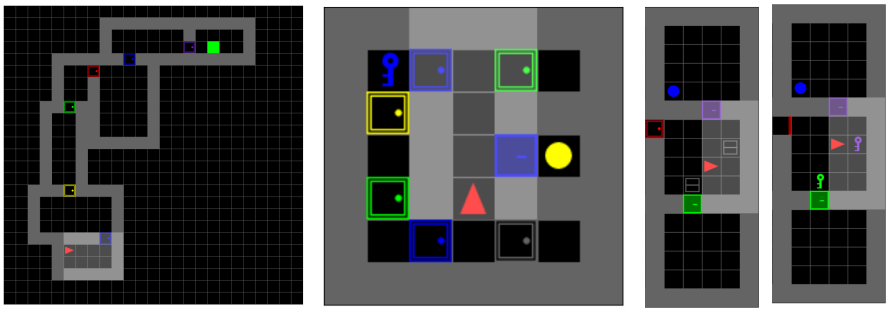
We answer RQ1, RQ2 and RQ3 over procedurally generated RL tasks from the Minimalistic Gridworld Environment (MiniGrid [29]). This framework allows creating RL tasks of varied levels of difficulty, does not strictly make use of images as observations, and most importantly, runs fast, thereby easing the implementation of massive RL benchmarks.

4.1 Environments

To design a representative benchmark for the study, among all the possible RL environments that can be selected/generated in MiniGrid, we consider 1) those

[5] We note that the choice of the neural network architecture is not just for the actor-critic modules, but also for IM approaches that hinge on neural computation.

labeled as `MultiRoomNXSY` (shortened as `MNXSY`, with `X` denoting the number of rooms and `Y` their size), 2) `KeyCorridorS3R3` (`KS3R3`); and 3) `ObstructedMaze2Dlh` (`O2Dlh`). These scenarios belong to hard exploration tasks (i.e., rewards are sparse), in which the agent fails to complete the task without the help of any IM mechanism. Refer to Figure 2 for further information about each scenario and its associated goal.



(a) `scenarios/MN7S8` (b) `KS3R3` (c) `O2Dlh`

Fig. 2: Examples of MiniGrid scenarios (a) `MN7S8`: the agent has to open multiple doors to reach the distant goal (green square); (b) `KS3R3`: the agent has to first collect the blue key in order to open the door of the room leading to the yellow ball that must be picked (c) `O2Dlh`: the agent has to discover keys hidden below some boxes, take the proper key and open the door to the blue ball (target).

By default, observations in these tasks are essentially egocentric and partially observable views of the environment, where a 7×7 tile set in the direction that the agent is facing composes the observation. Concretely, an observation is featured by a 7x7x3 matrix, being the 3 features of the last dimension information of interest such as type, colour and status of the object (e.g., doors, keys, balls, or walls) placed in the specific tile. Notice that the agent is incapable to see through walls or doors. 7 basic actions are available to solve all scenarios: `turn left`, `turn right`, `move forward`, `pick up` (an object, for instance keys or balls), `drop` the object (if carried), `toggle` (open doors, interact with objects) and `done`. Nevertheless, some of these actions are only useful at specific locations, whereas others become useless for certain tasks (for instance, `pick/drop` and `done` in `MNXSY` environments).

Not all the environments require the same amount of steps to be solved. Thus, in `MNXSY` environments a maximum number of $20 \cdot$ `X` steps is set to make it dependent on the number of rooms. Consequently, the three considered environments that fall within this set (`MN7S4`, `MN7S8` and `MN10S4`) are assumed to take at most 140, 140 and 200 steps, respectively. For `KS3R3` 270 and for `O2Dlh` 576 steps are set as maximum. The rewards are valued according to the number of steps taken. The optimal average extrinsic returns that the agent can achieve are 0.77 (`MN7S4`), 0.76 (`MN10S4`), 0.65 (`MN7S8`), 0.9 (`KS3R3`), and 0.95 (`O2Dlh`). Actually, since they are procedurally generated environments, each scenario's

final reward can slightly change due to the variance on the minimum required steps. In our case, we get these values by taking the median value of an optimal policy (equal to other previous reported optimal results [18]). Moreover, we also refer as suboptimal behavior to those policies that managed to obtain at least a 95% of the optimal score. In terms of complexity, `MN7S4` and `MN10S4` are the easiest ones to solve, followed by `MN7S8` and `KS3R3` which are harder. Finally, `O2Dlh` is the most difficult task in the benchmark.

4.2 Baselines and Hyperparameters

All our experiments will employ PPO [23] as the main RL algorithm. On top of it, we will use state-of-the-art IM techniques in order to obtain intrinsic rewards to augment the exploration efficiency, in which a naive PPO model fails [27]: COUNTS[6], RND [11], ICM [10] and RIDE [8]. For PPO we use a discount factor γ equal to 0.99, a clipping factor $\epsilon = 0.2$, 4 epochs per train step and $\lambda = 0.95$ for GAE [30]. We use 16 parallel environments to gather rollouts of size 128. Hence, we set a total horizon of $2,048$ steps between updates. Moreover, a batch size equal to 256 is considered. Unless otherwise specified, the following values - selected from an off-line grid search procedure over `MN7S4` - will be used to configure the intrinsic coefficient and entropy: $\beta = 0.05$ and $\varepsilon = 0.0005$ for RND, ICM and RIDE; $\beta = 0.005$ and $\varepsilon = 0.0005$ for COUNTS. In what refers to the dynamic update of β, we select $B = 0.5$ in Expression (5) as it represents a balanced trade-off for the agent to explore in the early stages of the training process, evolving towards a behavior mainly driven by extrinsic signals.

4.3 Network Architectures

Finally, experiments around RQ3 are performed with two different neural network architectural designs, which differ in terms of the type of neural layers (and design) and their number of trainable parameters. Following Figure 3, on one hand a *lightweight* neural architecture as in RAPID[27] is considered, in which both the actor and the critic are made of 2FC with 64 neurons each. This dual FC-64 architecture also applies to the embedding networks required for RND, ICM and RIDE. Additionally, we include a more sophisticated neural design based on what is proposed in RIDE [8], where both the actor and critic are combined into a two-headed (one for the policy, the other for the critic) shared network with 3 convolutional neural layers (32 3×3 filters, stride equal to 2, and padding 1) and a FC-256 layer. This last architecture will be deemed the *default* architecture to endow the agent with more learning capabilities and to ensure that it is not limited by a restricted network.

[6] In this case, we take advantage of the 2D grid (discrete state space) and map each state directly to a dictionary when using COUNTS. Nevertheless, when facing more complex state spaces pseudo-counts [14] can be applied as an alternative as in [20].

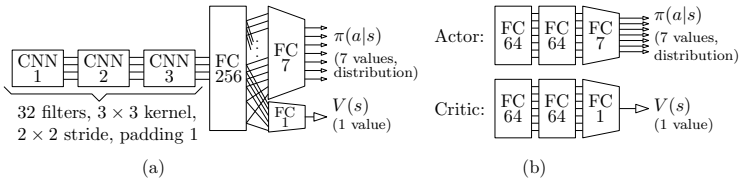


Fig. 3: (a) Sophisticated/default and (b) lightweight network architectures.

5 Results and Analysis

This section is devoted to present experiment results and answer the research questions posed in Section 3. Scripts and results have been made available in a public GitHub repository (`https://github.com/aklein1995/intrinsic_motivation_techniques_study`) to foster reproducibility and follow-up studies. For all the experiments described in this section we provide the mean and standard deviation of the average return computed over the past 100 episodes, performing 3 different runs (each with a different seed) to account for the statistical variability of the results.

RQ1: Does the use of a static, parametric or adaptive decaying intrinsic coefficient weight β affect the agent's training process?

Our first set of results compares the multiple weighting strategies introduced in Section 3.1, which differently tune the importance granted to the intrinsic rewards with respect to extrinsic signals coming from the environment. The results are shown in Table 1, where it is straightforward to note that RIDE outperforms COUNTS and RND. At this point we remind the reader that RIDE is configured with episodic count scaling, in accordance with the final solution proposed in [8]. Count-based generated rewards seem to be the best solution when facing easy exploration scenarios (`MN7S4` and `MN10S4`), but its performance degrades when facing scenarios that require more sophisticated exploration strategies. A similar pattern can be observed when analyzing the results of RND, which is unable to solve `MN7S8` and `O2Dlh` with any kind of weighting strategy. Contrarily, RIDE manages to solve all the tasks by its basic implementation, although it obtains better results when using more sophisticated weighting exploration strategies.

We now focus the discussion on gaps arising from the use of different weighting strategies. The static (default) weighting strategy (indicated with a suffix *_s* appended to each approach) is surpassed by any of the other proposed weighting approaches in the majority of the cases. When using multiple static values (*_ngu*), the only approach that takes advantage of such strategy is RND, yielding worse results for both COUNTS and RIDE in all the cases. This might happen due the slow pace at which the intrinsic rewards values decay in RND in reference to the other strategies. Moreover, the error outputs higher amplitude values than those of RIDE, then being RND a better candidate to get benefit of applying the *_ngu* strategy by the use of agents with smaller intrinsic coefficient

Table 1: Results of different IM strategies over several MiniGrid scenarios with static (*_s*), multiple static (*_ngu*) (as in NGU [9]), a parametric (*_pd*) or adaptive decay (*_ad*) weight β to modulate the importance of the intrinsic bonus in the computation of the reward. Cell values denote the training episodes ($1e6$ scale) at which the optimal average extrinsic return is achieved; in parentheses, episodes at which 95% of the optimal average extrinsic return is reached. Best results for every (IM strategy, scenario) combination are highlighted in bold.

	MN7S4	MN10S4	MN7S8	KS3R3	O2Dlh
COUNTS_*s*	**0.93** (0.86)	1.87 (1.78)	> 30	> 30	> 50
COUNTS_*ngu*	1.17 (1.11)	2.67 (2.35)	> 30	> 30	> 50
COUNTS_*pd*	0.96 **(0.83)**	2.27 (1.67)	> 30	**22.91 (22.49)**	> 50
COUNTS_*ad*	1.03 (0.92)	**1.81 (1.65)**	24.23 (24.10)	> 30	> 50
COUNTS_*ad*1000	1.03 (0.92)	**1.81 (1.65)**	23.63 (23.56)	> 30	> 50
RND_*s*	3.83 (3.78)	7.84 (7.79)	> 30	10.83 (9.72)	> 50
RND_*ngu*	2.69 (2.62)	5.78 (5.75)	> 30	8.12 (7.50)	> 50
RND_*pd*	4.04 (3.94)	6.02 (5.99)	> 30	9.24 (8.07)	> 50
RND_*ad*	**2.02 (1.39)**	**3.21 (2.65)**	> 30	**6.02 (5.43)**	> 50
RND_*ad*1000	3.62 (1.42)	3.59 (3.50)	> 30	7.47 (6.66)	> 50
RIDE_*s*	**2.49** (1.82)	2.27 (2.14)	4.00 (3.68)	6.63 (4.39)	30.88 (25.87)
RIDE_*ngu*	3.85 (2.40)	2.59 (1.26)	> 30	7.18 (3.91)	36.07 (29.96)
RIDE_*pd*	5.20 (2.14)	5.01 (1.96)	**3.73 (3.49)**	6.42 (3.87)	29.27 **(20.84)**
RIDE_*ad*	2.89 **(0.91)**	**1.60 (0.99)**	> 30	5.93 (2.99)	**27.65** (20.91)
RIDE_*ad*1000	2.54 **(0.91)**	**1.60 (0.99)**	3.88 (3.70)	**4.70 (3.00)**	28.00 (23.01)

weights. On the other hand, the use of parametric decay (*_pd*), which decreases the weight of the intrinsic reward as the train progresses to favor exploration, provides significant gains in almost all simulated scenarios. This approach is similar to *_ngu* although, instead of using multiple agents with different static intrinsic coefficients, it modulates a single value during the course of training. Hence, when employing *_pd* strategy, COUNTS is able to get a valid solution in KS3R3, RND improves all its scores and RIDE improves its behaviour in the most challenging scenarios MN7S8, KS3R3 and O2Dlh. Nevertheless, *_ngu* and *_pd* highly depend on the intrinsic coefficients given to each agent and the evolution of a single intrinsic coefficient during training, respectively. This strongly impacts on the agent's performance for a given scenario and dictates when those approaches might be better.

Finally, the use of adaptive decay (*_ad*) produces better results in COUNTS and RND when compared to the static case (*_s*). For RIDE, however, this statement does not strictly hold true, as its performance degrades in MN7S4 and MN7S8 (the agent does not even solve the task in the latter). We hypothesize that this is because the initial intrinsic returns are too high and calculating the historical average intrinsic returns biases the decay factor calculation. As outlined in Section 3.2, a workaround to bypass this issue is to calculate returns with a moving average over a window of ω steps/rollouts. We hence include in the benchmark an adaptive decay with a window size of $\omega = 1000$ rollouts (*_ad*1000). With this modification, RIDE improves its behavior in all the complex scenarios. Never-

theless, _*ad*1000 performs slightly worse than _*ad* in RND, but never worse than its static counterpart _*s*. In general, _*ad*1000 promotes higher intrinsic coefficient values than _*ad*, as the calculated average return is better fit to the actual return values. This leads to a lower decay value and a higher intrinsic coefficient, forcing the agent to explore more intensely than with _*ad* (but less than with _*s*).

RQ2: Which is the impact of using episodic counts to scale the intrinsic bonus? Is it better to use episodic counts than to just consider the first time a given state is visited by the agent?

Answers to this second question can be inferred from the results of Table 2. A first glance at this table reveals that the use of episodic counts or first-time visitation strategies for scaling the generated intrinsic rewards leads to better results. In the most challenging environments (`MNS78`, `KS3R3` and `O2Dlh`), these differences are even wider, as they require a more intense and efficient exploration by the agent. In fact, when the training stage is extended to cope with the resolution of a more complex task, intrinsic rewards also decrease, inducing a lower explorative behaviour in the agent the more the train is lengthened. What is more, the agent is not encouraged to collect/visit as many different states as possible. Hence, in those scenarios the baseline implementation of intrinsic motivation (_*noep*) may fail, but with these scaling strategies the problem is resolved (i.e. COUNTS and RND in `O2Dlh`). By contrast, in environments requiring less exploration (`MN7S4` and `MN10S4`), differences are narrower when using *episode-level* exploration and may be counterproductive in some cases (i.e. COUNTS at `MN10S4` with _1*st*).

Table 2: Comparison of different IM strategies when using no scaling (_*noep*), episodic (_*ep*) or first-time visit (_1*st*) to scale the generated intrinsic reward and combine two types of exploration degrees. Interpretation as in Table 1.

	`MN7S4`	`MN10S4`	`MN7S8`	`KS3R3`	`O2Dlh`
COUNTS_*noep*	0.93 (0.86)	1.87 (1.78)	> 30	> 30	> 50
COUNTS_*ep*	**0.76** (0.56)	**1.55 (1.47)**	2.77 (2.56)	3.99 (2.00)	**33.17 (29.79)**
COUNTS_1*st*	0.85 **(0.48)**	> 20	**1.64 (1.42)**	**1.97 (1.19)**	45.26 (37.29)
RND_*noep*	3.83 (3.78)	7.84 (7.79)	> 30	10.83 (9.72)	> 50
RND_*ep*	1.41 (0.96)	1.72 (1.34)	3.60 (3.30)	**4.31** (2.63)	**18.54** (14.07)
RND_1*st*	**1.18 (0.59)**	**1.36 (0.78)**	**1.97 (1.72)**	4.78 **(2.29)**	21.19 **(9.88)**
RIDE_*noep*	4.71 (4.54)	5.29 (5.20)	> 30	11.44 (9.63)	39.68 (35.15)
RIDE_*ep*	**2.49** (1.82)	**2.27 (2.14)**	4.00 (3.68)	6.63 (4.39)	**30.88 (25.87)**
RIDE_1*st*	3.17 **(1.34)**	3.27 (2.33)	**1.95 (1.83)**	**5.13 (2.26)**	32.14 (28.03)
ICM_*noep*	2.67 (2.55)	> 20	> 30	8.02 (6.75)	34.04 (26.78)
ICM_*ep*	3.25 (1.26)	**1.68** (1.59)	> 30	5.32 (3.14)	**19.05** (13.87)
ICM_1*st*	**1.56 (0.87)**	1.90 **(1.07)**	**2.11 (1.77)**	**4.72 (4.23)**	20.74 **(10.09)**

To better understand the superiority of RIDE over ICM [8], we also evaluate the performance of both approaches under equal conditions, with (_*ep*, _1*st*) and without (_*noep*) scaling strategies. In this way, we can examine the actual

improvement between the two types of exploration bonus strategies. Surprisingly, ICM gives better results in almost all the cases for the analyzed scenarios, yet exhibiting a larger variance in several environments that lead to failure (`MN10S4,MN7S8`). The reason might lie in how RIDE encourages the agent to perform actions that affect the environment forcing the agent to assess all possible actions, so that the entropy in the policy distribution decays slowly. This hypothesis is buttressed by the results obtained in `MN7S4` and `MN10S4`: we recall that there are 3 useless actions in these scenarios (`pick up`, `drop` and `done`), and RIDE performs clearly worse (except for the *_ep* case in `MN7S4`). In complex scenarios, when those actions are relevant for the task, performance gaps between RIDE and ICM become narrower.

Finally, for the sake of completeness in the results exposed in RQ1 and RQ2, Figure 4 shows the training convergence plots of COUNTS, RND and RIDE for different weighting and scaling strategies. These plotted curves permit to visually analyse the performance during the training process.

RQ3: Is the choice of the neural network architecture crucial for the agent's performance and learning efficiency?

One of the most tedious parts when implementing an algorithm is to determine which network architectures to use. First of all, when using an actor-critic RL framework it is necessary to establish whether a single but two-headed network or two different (and independent) networks will be adopted for the actor and the critic modules. In addition, some IM approaches are based on neural networks to generate the intrinsic rewards. In this work we evaluate two of those solutions: RND and RIDE, evaluating the contribution of different neural network architectures to the overall performance of the agent. We use similar architectures to the ones used in RIDE [8] and RAPID [27][7]: (1) a two-headed shared actor-critic network built upon convolutional and dense layers and (2) two independent MLP networks for the actor and the critic, respectively (more information in Section 3.3 for more information). Moreover, we fix the RL algorithm (PPO) and detail the number of parameters and time taken for the forward and backward passes in each network for an informed comparison.

Table 3 informs about these details of the neural architectures in use for COUNTS, RND and RIDE. The table reports the differences in terms of the number of parameters of each network, and the latency taken by the sum of both forward and backward passes through those IM modules (we note that COUNTS uses a dictionary and not a neural network). In addition, we summarize the total number of parameters depending on the IM module that has been implemented, together with the actor-critic parameters. Referred to the total elapsed time, we report the total amount of time required for a rollout collection. This elapsed time takes into account both the forward and backward passes in the IM modules, and just the forward pass across the actor-critic,

[7] Even with different neural architectures and base RL algorithms, they successfully solve the same tasks in MiniGrid.

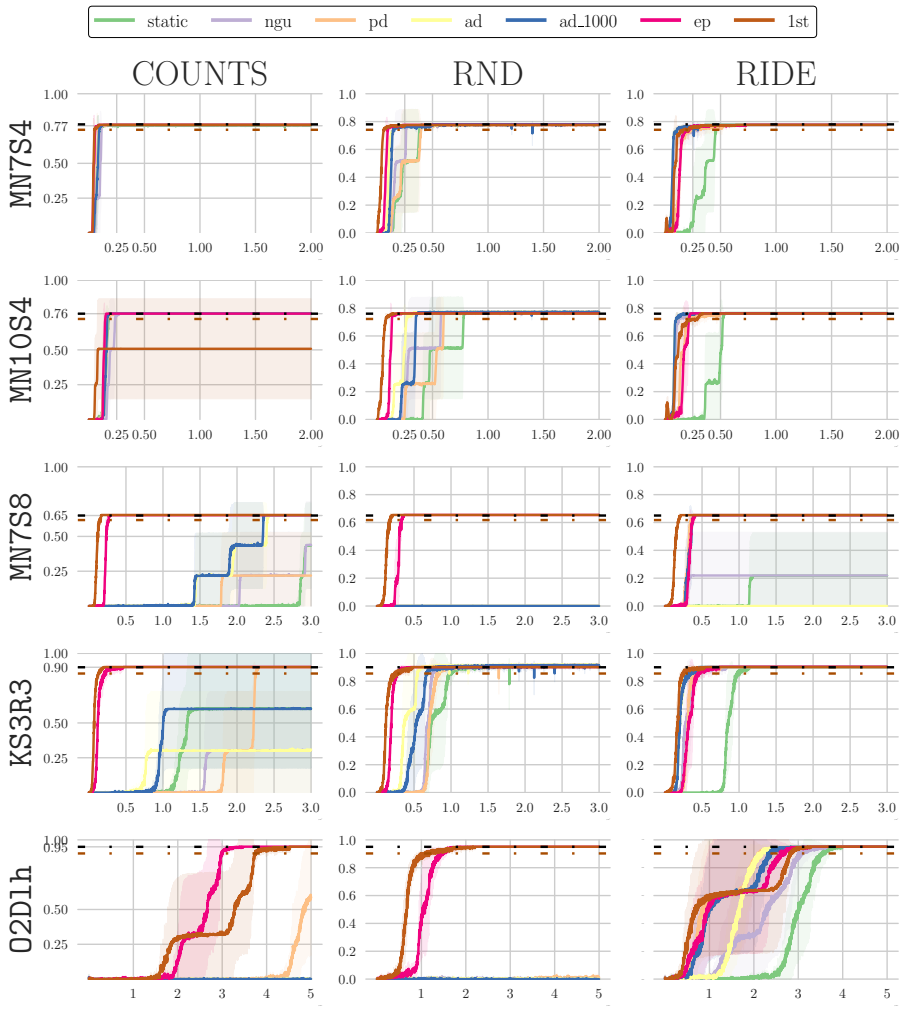


Fig. 4: Convergence plots of the schemes reported in Tables 1 and 2. Each column represents a Intrinsic Motivation type (COUNTS, RND and RIDE from left to right); each row represents the different scenarios (`MN7S4`, `MN10S4`, `MN7S8`, `KS3R3` and `O2Dlh`, from top to bottom). All figures depict the average extrinsic return as a function of the number of training frames/steps (in a scale of $1e7$). For each scenario, optimal and suboptimal scores are highlighted with horizontal black and brown lines, respectively.

among other operations executed when collecting samples. Times are calculated when executing the experiments over an Intel(R) Xeon(R) CPU E3-1505M v6 processor running at 3.00GHz. The performance of the agent configured with these network configurations is shown in Table 4.

Table 3: Comparison between the network architectures described in Section 3.3.

	Lightweight (*lw*)		Default	
	Parameters	Time (ms)	Parameters	Time (ms)
Actor	14,087	-	-	
Critic	13,697	-	-	
Actor+Critic	27,784	-	29,896	-
Dictionary	-	83.66	-	95.11
Total COUNTS	27,784	724.25	29,896	937.37
Embedding	13,632	-	19,392	-
RND	27,264	336.39	38,784	721.64
Total RND	55,048	986.13	68,937	1,408.42
Inverse	12,871	-	18,439	-
Forward	12,928	-	18,464	-
Embedding	13,632	-	19,392	-
RIDE	39,431	388.84	56,295	844.43
Total RIDE	67,215	1,177.75	86,191	1,791.70

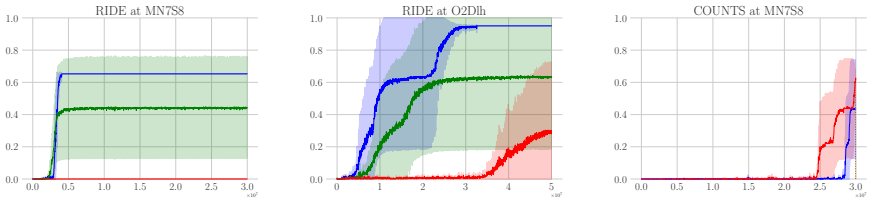


Fig. 5: Convergence plots of COUNTS and RIDE for some scenarios when using the default network (blue), *_lw_im*(green) and *_lw_tot*(red). All the figures depict the average extrinsic return as a function of the number of training frames.

Several observations can be drawn by inspecting these tables. On one hand, when reducing the number of parameters in both the actor-critic and the IM modules (*_lw_tot*), the agent's behaviour is critically deteriorated. This occurs even with COUNTS (Table 4), where the modification should have had less impact as the generation of intrinsic rewards does not depend on a neural network, but on a dictionary. When inspecting the performance of RIDE, its performance gets worse in all cases except for `MN7S4`, where the exploration requirements are the lowest among all the analyzed scenarios. Consequently, this modification can be less harmful. As for RND, the lightweight configuration of the networks makes the tasks not solvable by the agent.

The number of parameters to be learned is mostly dependent on the IM networks under consideration, whereas joining the actor and the critic into a single two-headed network barely increases the dimensionality requirements[8]. Nevertheless, the time required to perform a forward pass increases in approximately 25% when an unique actor-critic network is employed (Table 3). Moreover, by

[8] We note that the number of parameters is slightly increased, but they also differ in the type of layers that are used in each network (the two-headed network uses CNNs while the independent actor-critic only uses dense layers.

Table 4: Performance obtained with Counts, RND and RIDE when 1) using the default network configurations, 2) a lightweight architecture for the IM modules and keeping actor-critic with a default configuration (*_lw_im*), and 3) when both the IM and the actor-critic modules are implemented with the lightweight networks (*_lw_tot*). Values in the cells represent the training episode (in a scale of 1e6) when the optimal average extrinsic return is achieved. Within brackets, the training episode when an suboptimal behavior is accomplished.

	MN7S4	MN10S4	MN7S8	KS3R3	O2Dlh
COUNTS	0.93 (0.86)	1.87 (1.78)	> 30	> 30	> 50
COUNTS*_lw_im*	0.93 (0.86)	1.87 (1.78)	> 30	> 30	> 50
COUNTS*_lw_tot*	1.64 (1.48)	2.52 (2.36)	> 30 (29.96)	> 30	> 50
RND	3.86 (3.79)	7.84 (7.79)	> 30	10.84 (9.72)	> 50
RND*_lw_im*	5.66 (5.44)	6.68 (6.61)	> 30	10.97 (9.45)	> 50
RND*_lw_tot*	> 20	> 20	> 30	> 30	> 50
RIDE	2.49 (1.82)	2.27 (2.14)	4.01 (3.38)	6.63 (4.39)	30.88 (25.87)
RIDE*_lw_im*	1.63 (1.31)	1.75 (1.53)	> 30	9.44 (5.08)	> 50
RIDE*_lw_tot*	1.42 (1.05)	> 20	> 30	8.00 (5.69)	> 50

using a single network, part of the parameters of the network are shared between the actor and the critic, which can induce more instabilities but also a faster learning (as the model may share features between the actor and the critic and require less samples to learn a given task). With this in mind, we carry out an additional ablation study considering only the reduction of parameters at IM modules, and maintaining the actor-critic as a single two-head network. Such results are provided in the 2nd row of every group of results in Table 4 (*_lw_im*).

These results evince that when using RND*_lw_im*, slightly worse results are achieved with respect to RND with the default network setup. However, its performance does not degrade dramatically down to failure as with RND*_lw_tot*. Hence, using parameter sharing in a single network yields a faster learning process for this case. Regarding RIDE*_lw_im*, in some cases (`MN7S4` and `MN10S4`) it attains better results, whereas in `MN7S8` and `KS3R3` it suffers from a notorious performance decay (`MN7S8` is not solved). It can also be observed that the use of the single actor-critic network might be beneficial when reducing the complexity of the IM network (*_lw_im*), as it mitigates the performance degradation in 3 out of 5 scenarios (yet `MN7S8` and `O2Dlh` are not solved) when compared to separated actor-critic networks (*_lw_tot*), which fail over `MN7S8`, `O2Dlh` and `MN10S4`).

Finally, we include Figure 5 in order to help the reader draw deeper conclusions and gain insights about the behaviour of the learning process. It can be seen that in the two cases in which RIDE*_lw_im* failed (namely, `MN7S8` and `O2Dlh`), in two out of the three experiments that were run (*seeds*) the agent learned to solve the task, which underscores the impact of using different actor-critic architectures. Moreover, with the default actor-critic approach and using the COUNTS approach, the agent is also able to solve the `MN7S8` task in two out of the three runs. When using COUNTS*_lw_tot*, the agent reaches suboptimal performance and almost the optimal one within the frame budget.

6 Conclusion

In this work we have studied the actual impact of selecting different design choices when implementing IM solutions. More concretely, we have evaluated multiple weighting strategies to give different importance when combining the intrinsic and extrinsic rewards. Moreover, we have analysed the effect of applying distinct exploration degree levels along with the influence of the complexity of the network architectures on the performance of both actor-critic and IM modules. To conduct the study we have utilized environments belonging to MiniGrid as benchmark to test the quality of proposed schemes in a variety of tasks demanding from hard to very hard intensity of exploratory behaviour.

On one hand, we have shown that using a static intrinsic coefficient might not be the best strategy if we focus on sample-efficiency. Adaptive decay strategies have proved to be the most promising ones, although they require a good parameterisation of the sliding window. Parameter decay approach, in turn, have performed competently but it is subject to a decay parametrisation which could be more dependent of the task at hand than the previous scheme, which makes this strategy more sensitive to the environment and the task to be solved (as it happens with $\epsilon-$greedy strategies in Q-learning). The use of multiple agents as in NGU [9], each featuring a different exploration-exploitation balance also suffers from this parametrisation but reports worse results.

On the other hand, the use of *episode-level* exploration along with *experiment-level* strategies seem to be preferable when having environments with hard exploration requirements. It is not a clear winner/preference between episodic counts and first visitation strategies as their performance is not only subject to the environment, but also to the selected IM strategy, although both achieve significant improvement in the performance. Hence, we encourage the implementation of any of these strategies in follow-up IM-related studies.

Last but not least, we have analyzed the impact of modifying the neural network architecture in both the actor-critic and IM modules. The results show that reducing the number of parameters at the IM modules deteriorate the performance of the agent, making it fail in some challenging scenarios which are feasible for the complex neural configuration. What is more, when reducing the IM network dimensions, it is preferable to use a shared two-headed actor-critic as it provides better results, although it is not clear whether those results are due to the use of a single neural network (and the underlying parameter sharing and common feature space for the actor and the critic) or to the adoption of different architectures (e.g. CNNs). Further research is necessary in this direction.

We hope this work can guide readers in the implementation of intrinsic motivation strategies to address tasks with (1) a lack of dense reward functions or (2) at hard exploration scenarios where the classic exploration techniques are insufficient. Aligned with the purposes of academic and industry communities, we make all the experiments available and provide the code to ensure reproducibility [3]. In the future, we will intent to extend these analysis to more environments and algorithms in order to have more representative results.

Acknowledgments

The authors would like to thank the Basque Government for its funding support through the ELKARTEK program (3KIA project, KK-2020/00049) and the research group MATHMODE (T1294-19). A. Andres receives funding support from the same institution through its BIKAINTEK PhD support program.

References

1. David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.
2. Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv:1909.07528*, 2019.
3. Andreas Holzinger. Introduction to machine learning & knowledge extraction (make). *Mach. Learn. Knowl. Extr.*, 1(1):1–20, 2019.
4. Arthur Aubret, Laetitia Matignon, and Salima Hassas. A survey on intrinsic motivation in reinforcement learning. *arXiv:1908.06976*, 2019.
5. Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in Neural Information Processing Systems*, 29, 2016.
6. Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization, 2016.
7. Dragoș Grigorescu. Curiosity, intrinsic motivation and the pleasure of knowledge. *Journal of Educational Sciences & Psychology*, 10(1), 2020.
8. Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *arXiv:2002.12292*, 2020.
9. Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never give up: Learning directed exploration strategies. *arXiv:2002.06038*, 2020.
10. Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787, 2017.
11. Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv:1810.12894*, 2018.
12. Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv:2006.05990*, 2020.
13. Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2020.
14. Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in Neural Information Processing Systems*, 29, 2016.

15. Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2753–2762, 2017.
16. Marlos C Machado, Marc G Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *AAAI Conference on Artificial Intelligence*, volume 34(4), pages 5125–5133, 2020.
17. Miruna Pîslar, David Szepesvari, Georg Ostrovski, Diana Borsa, and Tom Schaul. When should agents explore? *arXiv:2108.11811*, 2021.
18. Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. Noveld: A simple yet effective exploration criterion. *Advances in Neural Information Processing Systems*, 34, 2021.
19. Nicolas Bougie and Ryutaro Ichise. Fast and slow curiosity for high-level exploration in reinforcement learning. *Applied Intelligence*, 51(2):1086–1107, 2021.
20. Adrien Ali Taiga, William Fedus, Marlos C Machado, Aaron Courville, and Marc G Bellemare. On bonus-based exploration methods in the arcade learning environment. *arXiv:2109.11052*, 2021.
21. Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2018.
22. Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv:1808.04355*, 2018.
23. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.
24. Manu Orsini, Anton Raichuk, Léonard Hussenot, Damien Vincent, Robert Dadashi, Sertan Girgin, Matthieu Geist, Olivier Bachem, Olivier Pietquin, and Marcin Andrychowicz. What matters for adversarial imitation learning? *Advances in Neural Information Processing Systems*, 34, 2021.
25. Xiao Jing, Zhenwei Zhu, Hongliang Li, Xin Pei, Yoshua Bengio, Tong Che, and Hongyong Song. Divide and explore: Multi-agent separate exploration with shared intrinsic motivations, 2022.
26. Mathieu Seurin, Florian Strub, Philippe Preux, and Olivier Pietquin. Don't do what doesn't matter: Intrinsic motivation with action usefulness. *arXiv:2105.09992*, 2021.
27. Daochen Zha, Wenye Ma, Lei Yuan, Xia Hu, and Ji Liu. Rank the episodes: A simple approach for exploration in procedurally-generated environments. *arXiv:2101.08152*, 2021.
28. Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416, 2018.
29. Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.
30. John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv:1506.02438*, 2015.