Learning Intrinsically Motivated Options to Stimulate Policy Exploration

Anonymous Authors[1]

Abstract

A Reinforcement Learning (RL) agent needs to find an optimal sequence of actions in order to maximize rewards. This requires consistent exploration of states and action sequences to ensure the policy found is optimal. One way to motivate exploration is through *intrinsic* rewards, i.e. agent-induced rewards to guide itself towards interesting behaviors. We propose to learn from such intrinsic rewards through *exploration options*, i.e. additional temporally-extended actions to call separate policies (or "Explorer" agents) associated to an intrinsic reward. We show that this method has several key advantages over the usual method of weighted sum of rewards, mainly task-transfer abilities and scalability to multiple reward functions.

1. Introduction

At the core of the Reinforcement Learning paradigm lies the exploration-exploitation dilemma – the agent wants to maximize its reward, using the knowledge at hand, but also needs to look for better behaviors. Despite impressive recent breakthroughs, state-of-the-art RL methods in high-dimensional environments still rely on randomness in action- or parameter-space to explore; with for example Noisy Nets (Fortunato et al., 2018) in Rainbow (Hessel et al., 2018), entropy maximization in Soft Actor-Critic (Haarnoja et al., 2018), or just ϵ-greedy in Deep Q-Network (Mnih et al., 2015). The latter, in particular, has the most naive exploration procedure possible. Indeed, it is *state-independent*, without *memory*, and the exploration behavior emerging from it is, by construction, *single-step* and purely *random*. We would like ways to consistently explore the environment, or in other words, sample-efficiently span the state-action space.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

In environments where the MDP can be considered from a tabular point-of-view, it is possible to extract powerful Exploration-Exploitation balances. Methods that achieve this, e.g. MBIE-EB (Strehl & Littman, 2008), often use an *intrinsic* reward (Oudeyer & Kaplan, 2009), a reward generated by the agent (as opposed to *extrinsic* environment rewards) to motivate alternative behaviors that might be useful for the true task learning.

For more complex and higher-dimensional environments, a very wide range of methods have been proposed, such as *pseudo-count*-based methods (Bellemare et al., 2016; Ostrovski et al., 2017; Fox et al., 2018) or *curiosity*-based methods (Schmidhuber, 2010; Houthooft et al., 2016; Pathak et al., 2017). The resulting behavior, from *intrinsic reward alone*, is a consistent policy that actively tries to find surprising elements in the environment (Burda et al., 2019a). The gap in efficiency with a random exploration policy is striking, and motivated this work.

To integrate such intrinsic rewards R_i into an RL framework with extrinsic reward R_e, the traditional approach is to carefully tune a reward-shaping hyperparameter β in the final reward expression $R = R_e + \beta R_i$. Several drawbacks come with this, such as hard fine-tuning of β, loss of task-transfer potential, constraints on the reward function, and restriction to a single intrinsic reward function.

Our proposed alternative is to decouple the policy into an *Explorer* and an *Exploiter*, and to add an **exploration option** to the set of actions of the Exploiter (see Fig. 1). It is an additional action that the Exploiter can use to let the Explorer act for some time, before taking back control.

The objective and contribution of this work is in: first, shifting the attention towards decoupled agents in intrinsic reward settings; second, opening up the path for a new family of intrinsic reward functions; third, allowing the learning from multiple intrinsic reward functions; and fourth, taking a step towards an agent with several task-independent and learnt policies built-in.

In Section 2, we present related work, mainly intrinsic motivation techniques and existing approaches to learn from such signals. In Section 3, we introduce in more detail the method, its theoretical advantages, and implementation. In

Section 4, we provide experiments in a tabular setting to support our claims. In Section 5, we discuss ways to export the method in function approximation, as well as limitations. Section 6 concludes the paper.

2. Related Work

2.1. Options and Hierarchical RL

An **option** (Sutton et al., 1999) is a generalization of actions to cover a sequence of primitive actions or other options. Options are often linked with Hierarchical RL (Barto & Mahadevan, 2003), allowing a meta-agent to use multiple controller agents or other meta-agents, which enables reasoning over *temporally extended* states and actions.

2.2. Exploration without intrinsic motivation

Most recent state-of-the-art works in Deep RL explore without intrinsic motivation and focus on much simpler exploration methods, generally relying on randomness in action- or parameter-space.

Deep Q-Network (Mnih et al., 2015) simply uses ϵ-greedy. An improvement was found in *Noisy Networks* (Fortunato et al., 2018), which learn perturbations of the network weights to drive exploration, relying on the idea that a single change in weight can yield significant changes in policy. Along with ϵ-greedy, they have been found to perform relatively well in comparison to intrinsically motivated agents (Taiga et al., 2020). However, is it impossible to extract just the exploration policy from the network, diminishing its interest from a task transfer and generalization point-of-view.

Entropy-regularization methods (Williams, 1992) try to prevent the agent from overfitting to a task or falling into local minima by solving it while acting as randomly as possible. Most notably, *Soft Actor-Critic* (Haarnoja et al., 2018) is an off-policy actor-critic algorithm that achieved excellent and very stable results on hard control tasks.

2.3. Intrinsic Motivation

Intrinsic motivation is of interest because it "leads organisms to engage in exploration, play, and other behavior driven by curiosity in the absence of externally-supplied rewards" (Barto, 2013). We describe a number of methods that could be used to generate intrinsic reward to train our Explorer.

Curiosity-based methods use *prediction error* from a learnt forward model F to generate intrinsic reward, fundamentally $R_i = ||s_{t+1} - F(s_t, a_t)||$, motivating the search for *surprising* elements and transitions (Pathak et al., 2017; Burda et al., 2019a; Kim et al., 2019).

Count-based methods count state-action visits $N(s, a)$ and generate reward for states that are not often visited, e.g. $R_i = 1/\sqrt{N(s, a)}$. This motivates the search for *novel* states. To extend to complex state-spaces, recent work uses *density models* to generate a *pseudo*-count (Bellemare et al., 2016; Ostrovski et al., 2017; Martin et al., 2017).

Intrinsic motivation is not limited to exploration – indeed, a line of work focuses rather on **skill learning**. Methods within this branch (Achiam et al., 2018; Eysenbach et al., 2019) often use an intrinsic reward to motivate *diversity*, i.e. ease of *distinction* of the skills.

Such methods motivate designing agents equipped with *multiple* intrinsic reward functions. A thorough survey and classification of intrinsic motivation can be found in (Aubret et al., 2019).

2.4. Combining Intrinsic and Extrinsic rewards

Although extensive work has been done on the different ways to generate intrinsic rewards, surprisingly little work exists on incorporating such rewards with the environment's; the traditional approach uses the weighted sum $R = R_e + \beta R_i$ and fine-tunes β.

In the paper introducing Random Network Distillation (Burda et al., 2019b), a very efficient method for intrinsic motivation based on state reconstruction error, the authors shortly propose to decouple the network's heads into an exploration and exploitation head. However, first, they limit this idea to using different discount rates; and second, they only apply it to the value function estimator, not to the policy network, effectively merging both policies into one in the process. In contrast, we decouple the agents all the way to the policies, conserving knowledge and distinct behaviors of both.

The *engaged climber method* (Perotto, 2015) also proposed to decouple an exploration and exploitation strategy, where the Explorer is trained with a version of a curiosity-based reward. The focus of the author is on the timing to switch between strategies, presenting a set of conditions to decide whether to switch. However, our work differs in two main ways: First, their algorithm and hyperparameter choices assume knowledge of the MDP; and second, their switching method relies heavily on the reward sign, and is not learnt; while we use options.

In a preliminary paper empirically studying the success of Hierarchical RL (Nachum et al., 2019), the authors introduce *Explore & Exploit*, the closest approach we find to ours. They train an Explorer and Exploiter with respectively intrinsic (from goal specification) and extrinsic rewards. The core differences with our method is that a temporary policy is chosen from a fixed $\{Explorer : 0.2, Exploiter : 0.8\}$ distribution. The Exploiter can therefore not deliberately *choose* to explore, nor can-he choose not to, as the exploration is deliberately not formulated as an option. The au-

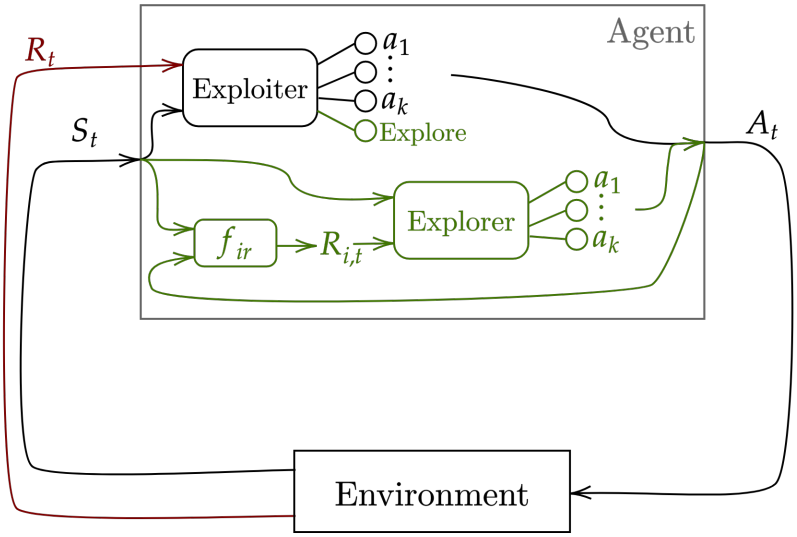

Figure 1. Architecture of the agent with our proposed extension in green. The Exploiter is trained using the extrinsic reward R_t (red) while the Explorer uses the intrinsic reward $R_{i,t}$. Note that the time-step t is incremented at the Environment step, hence $f_{i,r}$ takes (A_t, S_{t+1}) as input.

thors only use their algorithm for comparison purposes in a HRL setting, while our focus is in learning from intrinsic motivation.

3. Method

We assume that the agent is equipped with an **intrinsic reward function** $f_{ir}(s, a, s') = R_i$ to generate a pseudo-reward R_i from the (s, a, s') transition. This function's purpose is to help the agent consistently explore diverse behaviors to find the optimal policy, with examples provided in Section 2.3, and is expected to generate an interesting behavior *without extrinsic reward* at all. We focus on how to efficiently use this reward as a learning signal to improve sample-efficiency and performance.

3.1. Traditional approach drawbacks

The traditional approach, which can be called *weighted sum reward*, is to fine-tune a β parameter in the total reward expression $R = R_e + \beta R_i$, where R_e is the task's reward, called *extrinsic*. It basically comes down to tuning a single hyperparameter to balance out exploration and exploitation. However, multiple problems arise from this approach.

Constraints on the reward function The weighted sum assumes that $R_i \to 0$, so that the agent can converge towards optimal behavior. First, this means that it constraints the exploration to the beginning of the learning, while as pointed out in (Sutton & Barto, 2018), "*any method that focuses on the initial conditions in any special way is unlikely to help with the general non-stationary case*". Indeed, we constrain our f_{ir} function to disappear asymptotically; however, there is always ground for exploration. Second, if the reward function is non-stationary or too expressive, it can cause convergence issues: "*when intrinsic rewards evolve over time, the agent generally cannot find an optimal stationary policy*" (Aubret et al., 2019).

Loss of knowledge and generalization First, if the reward function disappears asymptotically, the learnt exploratory behavior will be lost – the agent will forget *how* to explore. Second, even if it did not, in combining both policies into one, it is impossible to extract either the greedy or exploratory behavior at any time. If no greedy behavior can be extracted, the performance will be hindered as long as the intrinsic reward does not disappear. If no exploratory behavior can be extracted, we lose valuable task-independent knowledge about the environment. Third, in combining the policies, it is unclear when the greedy behavior will win over the exploratory one, which might impede multi-step exploration.

Hyperparameter fine-tuning In the weighted sum expression, fine-tuning β is reward-dependent, non-intuitive, and can therefore be tedious: "*without due care, the optimal policy can be altered or even completely obscured by the intrinsic rewards*" (Fortunato et al., 2018). We will see that there is not always a good balance.

Extensions to multiple reward functions If multiple intrinsic reward functions $f_{ir}^{(j)} = R_i^{(j)}$ are available, the grid-search fine-tuning of $\beta^{(j)}$ parameters in the total $R = R_e + \sum_j \beta^{(j)} R_i^{(j)}$ reward expression quickly becomes impractical. Even with few reward functions, the rewards $R_i^{(j)}$ are hard to compare intuitively, both in amplitude and decaying rates, so we are likely to lose most of the behaviors in the fine-tuning process.

3.2. Proposed alternative

Our focus is therefore on finding a way to benefit from an intrinsic motivation function f_{ir} without such downsides. The key insight to our approach lies in the fact that a *purely* intrinsically motivated agent can still learn powerful exploration policies (Burda et al., 2019a).

3.2.1. Learning from the intrinsic reward

Our proposition can be summarized in the term *decoupling*: training a different agent per reward function available. We call the agent trained with extrinsic reward $r(s, a, s') = R_e$ the **Exploiter**; all other agents, trained with intrinsic rewards $f_{ir}^{(j)}(s, a, s') = R_i^{(j)}$, are called **Explorers**.

3.2.2. Calling the exploration policy

We would like the agent to learn when to explore, and when to exploit. For this, we equip the Exploiter with an **exploration option**, i.e. an additional action to call the Explorer. For n Explorers, if $k = |\mathcal{A}|$ with $\mathcal{A}$ the set of possible actions, then the options are $a_{explorer_j} = a_{k+j}, \forall j \leq n$. The overall agent architecture with $n = 1$ Explorer is shown in Fig. 1, with additions in green.

When called, an Explorer acts for the next c_{switch} timesteps[1], before "switching" back to the Exploiter. Learning an option termination function is beyond the scope of this paper. Usage of the option leads to *multi-step* exploration, which corresponds to an *exploratory temporal abstraction* and has empirically been shown, in some settings, to be important for Hierarchical RL (Nachum et al., 2019).

3.2.3. Training

The reward $\hat{R}_{t+1} = \sum_{k=0}^{c_{switch}-1} \gamma^k R_{t+k+1}$ and bootstrapping discount $\gamma^{c_{switch}}$ are used for option learning.[2].

All agents –Exploiter, or Explorers– have access to the primitive actions for natural control; only the Exploiter has access to more, with as many options as Explorers available. The agents are all trained **off-policy** from the $(S_0, A_0, ..., S_t, A_t, S_{t+1})$ stream of data; using extrinsic reward $r(S_t, A_t, S_{t+1}) = R_e$ for the Exploiter, and intrinsic rewards $f_{ir}^{(j)}(S_t, A_t, S_{t+1}) = R_i^{(j)}$ for Explorers.

3.2.4. Under-sampling problem

When the action space is large, the options might get drowned in the possibilities, and get sampled less. To prevent this, under an ϵ-greedy Exploiter policy, we can choose a probability p_{ex} to select an exploration option when acting non-greedily (under probability ϵ). At one extreme, $p_{ex} = 1/(k+n)$ leads back to normal ϵ-greedy action selection (which can therefore be seen as a default hyperparameter choice). At the other extreme, choosing $p_{ex} = 1$ means the Exploiter will exclusively use an option and call an Explorer when acting non-greedy. This would mean that ϵ-greedy is now, with probability ϵ, causing the agent to use consistent, multi-step exploration – instead of a single step of random action selection. Note that in this setting, we should make sure that the Explorers explore the whole state-action space, in order to ensure thorough exploration.

3.3. Comparison with the weighted sum method

In this subsection we answer the problems that arise from using the *weighted sum* reward scheme, $R = R_e + \beta R_i$, mentioned in subsection 3.1, point-by-point.

Constraints on the reward function Since we decoupled the Exploiter and Explorer, the Explorer is now an independent agent that can learn from any reward function f_{ir}. It does not matter if the function is non-stationary, or expressive, or does not disappear in time.

Loss of knowledge and generalization The task-independent knowledge about the environment –regarding *how to explore*, or any other skill the Explorers might have learnt– is now contained in the Explorers as usable policies. In the context of task task-transfer, the Exploiter alone can be switched out for a new Exploiting agent, while keeping the set of $f_{ir}^{(j)}$, options and learnt Explorers intact (in green on Fig. 1).

Hyperparameter tuning The β exploration hyperparameter is removed completely from the method, which means we do not need to fine-tune it to balance out exploitation and exploration. The agent can decide how much it explores. Note that in return, we have a whole Explorer agent to take care of.

Extensions to multiple reward functions When using Explorers and Exploration Options, the extension to multiple reward functions $f_{ir}^{(j)}$ is natural and requires little extra work.

[1] Notation from Explore&Exploit, (Nachum et al., 2019).

[2] Full notations, simplified for clarity, should include minimum operators with T, the termination time in episodic environments.

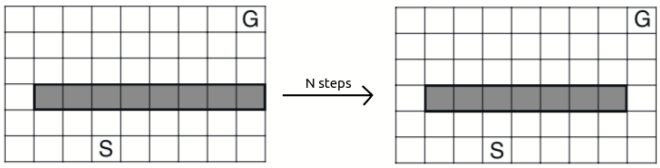

Figure 2. Shortcut Maze environment. The agent starts in state S and a reward of 1 is distributed when the agent reaches the terminal state G. The shortcut opens up after N steps.

4. Experiments

4.1. Environment and Setting

We evaluate our method in a tabular setting, on the Shortcut Maze environment (Sutton & Barto, 2018), where a shorter path is opened up after $N = 30k$ steps (Fig 2). Note that the environment is only interesting if the agent did indeed find the first longer path, or finding the shorter path becomes easy. We implement this environment with a sparse reward of 1 when the agent reaches the goal G, and set $\gamma = 0.9 < 1$ so the agent looks for the shortest geodesic path. We use a learning rate of $\alpha = 0.1$ for all learnt agents.

The agent is evaluated in a testing phase, using the greedy behavior (the Exploiter still has access to the options). During such testing phases –but not in training–, the agent is limited to 100 steps to reach the goal. We average the results over 50 runs.

4.2. Comparison with baselines

We compare our ExploreOption with other model-free baselines for exploration. The first baseline (referred-to as "QL") is a simple ϵ-greedy QLearning (Watkins & Dayan, 1992) with ϵ annealed from 1 to 0.1 over $5k$ steps. A constant $\epsilon = 0.1$ is used for all other QLearning agents. The second is an Optimistic QLearning (referred-to as "QL_O"), with $Q_1 = 1$. The third is a QLearning agent with a visit-count reward inspired from MBIE-EB[3] (Strehl & Littman, 2008), $R_i = 1/\sqrt{N(s')} =: f_{ir}(s, a, s')$ with total reward $R = R_e + \beta R_i$, which we refer to as QLearning+VisitCounts ("QL+VC"). Since we want the values of states to go down with visits, we use an optimistic initialization at $Q_1 = \beta/(1-\gamma)$, which solves the first update with target $0 + \beta/\sqrt{(1)} + \gamma Q_1$. The ExploreOption agent is implemented with a QLearning Exploiter and Explorer ($n = 1$), and the Explorer uses the same optimistic values[4] as QL+VC and is trained with reward R_i alone. Note that the Explorer does not have access to the `done` flag indicating a terminal state, and sees the terminal $G \rightarrow S$ transition

[3]We use visit-counts $N(s')$ instead of $N(s, a)$ to speed up learning in a deterministic environment.

[4]using $\beta = 1$, but this hyperparameter has no effect here.

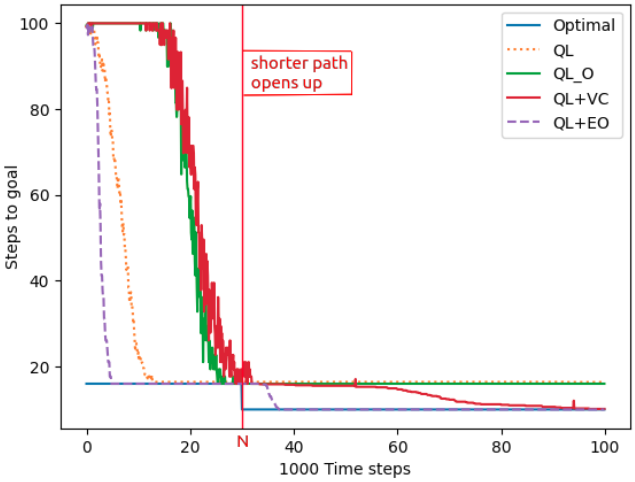

Figure 3. Number of steps to the goal over training steps. Comparison of QLearning with annealed ϵ (QL), QLearning with Optimistic starts (QL_O), QLearning with a Visit-Count based reward (QL+VC), and QLearning with an ExploreOption (QL+EO).

as any other (with a random action). This is essential for positive reward functions, as otherwise the agent will be pushed off the goal state, resulting in biased exploration. The whole agent is referred-to as "QL+EO". The hyperparameters are fixed using the best values of the studies from the following paragraphs.

The results are shown in Fig 3. As expected, QLearning alone with an annealed ϵ does not find the short path, having overfit to the longer one. Optimistic values alone do not help much for this task, as the path opens up after the agent's expectations wear down. The algorithm is therefore simply slower than QLearning to find the first path. Since QL+VC uses optimistic values, its slow convergence is similar, and even slightly slower to ensure visit-count based exploration – it barely has time to find the first path before the second opens up. In comparison, QL+EO finds the first path even faster than QLearning alone, due to proper exploration, while it finds the second path much faster than the visit-count based approach. As we decoupled exploration from exploitation, the agent is able to benefit from consistent exploration without harming the greedy behavior.

4.3. c_{switch} and intrinsic reward functions

The c_{switch} hyperparameter controls the number of steps the Explorer acts before the Exploiter takes the control back. With $c_{switch} = 1$, the Explorer has the least time and expressiveness, close to ϵ-greedy. With $c_{switch} = \infty$ (implemented arbitrarily as $c_{switch} = 3600$), the Explorer plays until the end of the episode. The hyperparameter can be associated with the β hyperparameter from the weighted sum scheme, as c_{switch} appears when decoupling to get rid of β,

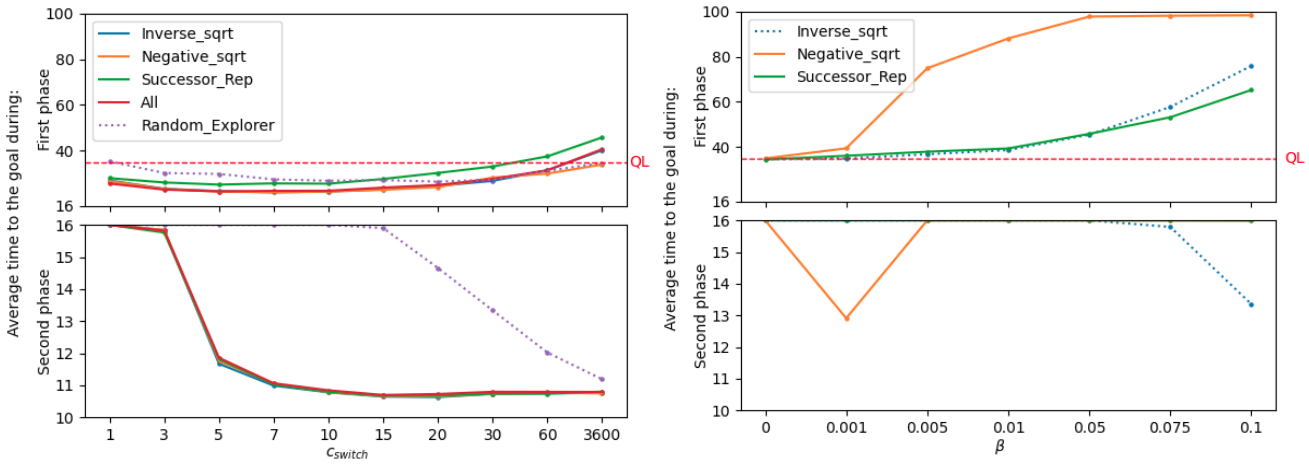

Figure 4. (lower is better) Average time to the goal, during training, before (top plot) and after (bottom plot) the path opens up (N steps); for varying hyperparameters. The x axis is not linear, only a set of increasing values of the hyperparameter that are of interest. Comparison of learning-based exploration methods and hyperparameters using different reward functions. Each plot reports the average performance before (up) and after (down) the path opens up. (left:) QLearning+ExploreOption; (right:) QLearning+VisitCounts.

and β also controls how much we care about exploration. In this section we compare these two hyperparameters, as well as the approaches' sensitivity to f_{ir}, on a set of 3 intrinsic reward functions.

The first intrinsic reward function we use is the inverse square-root of the visit-counts from the baselines, giving more reward to less-often visited states: $f_{ir}(s,a,s') = 1/\sqrt{N(s')}$, which we refer to as `Inverse_sqrt`.

Based on the same intuition, the second function we use is $f_{ir}(s,a,s') = -\sqrt{N(s')}$ referred to as `Negative_sqrt`. This function explodes in time, but we use it to show that decoupling allows for much more expressive reward functions, and functions that do not disappear in time. Note that regardless of the value of $\beta > 0$, the exploration reward will eventually lead the QL+VC agent.

The third function we use is based on the Successor Representation (SR)(Dayan, 1993), defined as $\Psi_\pi(s,s') := \mathbb{E}_{\pi,p}\left[\sum_{t=0}^{\infty}\gamma^t \mathbb{I}\{S_t = s'\} | S_0 = s\right]$, and can be interpreted as a form of generalization or distance over states, based on the similarities of the states they lead to, i.e. their successors. The SR can conveniently be learnt with TD(0) and it has been shown recently (Machado et al., 2018) that the norm of the SR can be used as an intrinsic reward signal, which makes it one of the most general intrinsic reward functions available. The resulting intrinsic reward is then $f_{ir}(s,a,s') = 1/\|\Psi(s)\|_1$, referred to as `Successor_Rep`. We use the same optimistic starts as for `Inverse_sqrt`.

In order to compare the reward functions and hyperparameters, we plot the average time the agent takes to reach the goal during the first part of training (before the path opens up; top subplot) and second part of training (after the path opens up; bottom subplot) when varying the hyperparameters c_{switch} and β. Different reward functions correspond to different curves, as the legend details – the results are shown in Fig. 4.

First, let's focus on our approach (QL+EO – left plots, first three curves of the legend). We observe that the method is extremely robust to changes in the function: the results are virtually the same, with a slight delay for the `Successor_Rep` to converge in the first phase. Regarding c_{switch}, as expected, the method fails to find the second path, or is exceedingly slow, for the lowest values of c_{switch} (for 1 and 3). In such cases, the Explorer does not have enough time to take the Exploiter out of the sub-optimal trajectory, hence the shorter path is never found, leading to an average (and worst possible) performance of 16 steps to the goal (lower plot). Remember that without the option, the Exploiter is a normal ϵ-greedy QLearning algorithm. Note that the highest values of c_{switch} take the most time to find the first path, which is likely due to the agent being taken out the temporary optimal path when the task is straightforward. Any $c_{switch} \geq 5$ finds the shorter path consistently and very fast. Even the first path is found faster with intermediate values of c_{switch}. These observations align with the Hierarchical RL study (Nachum et al., 2019): multi-step exploration, i.e. through temporal abstractions, is essential to achieve our performance. Since most values show good results, we arbitrarily selected $c_{switch} = 15$ for the baselines.

The robustness of the approach to changes in the intrinsic reward function is striking, especially since `Negative_sqrt` is so expressive, and `Successor_Rep`

does not come with the guarantees that `Inverse_sqrt` possesses. QL+EO benefits just as well from all of them, as long as they lead to consistent exploration.

In comparison, QL+VC (right plots) has much less attractive results. `Inverse_sqrt`, as mentioned in the comparison, suffers from a difficult β tuning, where high values lead to very delayed finding of the first path, while lower values simply fail to find the second path altogether. This shows that trying to combine both policies into one leads to balancing out when one or the other takes the lead, rather than benefiting from both behaviors. The `Successor_Rep` finds the first path slightly faster, but no value of β was found to discover the second path at all. Regarding `Negative_sqrt`, averaging does not display the severe instabilities, and the average time in the second phase is actually often higher than 16. The value $\beta = 0.001$ is unstable at first but settles down to the optimal path; however, the visit counts will eventually blow the agent out of the correct trajectory.

4.4. Multiple Explorers

As we have seen in Section 4.2, we might want to learn from multiple reward functions, as they offer a spectrum of behaviors. Since our ExploreOption approach scales naturally with the number of reward functions available, we can try this idea with all 3 intrinsic functions used before: `Inverse_sqrt`, `Negative_sqrt` and `Successor_Rep`. The result are shown in the curve "All" from Fig. 4 (left).

The outcome is virtually the same as for any of those functions separately, with very close performance to `Inverse_sqrt` in particular. Small to no changes to performance compared to the functions alone was to be expected, as they all provided very similar results, all focus on exploration, and the spectrum of behaviors that can emerge from such a simple Gridworld is limited. However, the point is that the integration of several intrinsic reward functions – and therefore several resulting intrinsically motivated policies – was trivial and did not harm performance. We believe this is a fundamental improvement on the traditional approach to intrinsic motivation.

4.5. Random Explorer

Learning an additional agent and an intrinsic function can be computationally expensive. This raises the question of the necessity of learning an exploratory policy: how much would the performance or sample-efficiency be affected if the Explorer did not require resources? We answer this question by replacing the (learnt) QLearning algorithm of the Explorer by a random Explorer agent. The results are reported on the left of Fig 4, "Random_Explorer" curve. Note that for $c_{switch} = 1$, we fall back to QLearning. For all higher values, the first path is found faster, showing

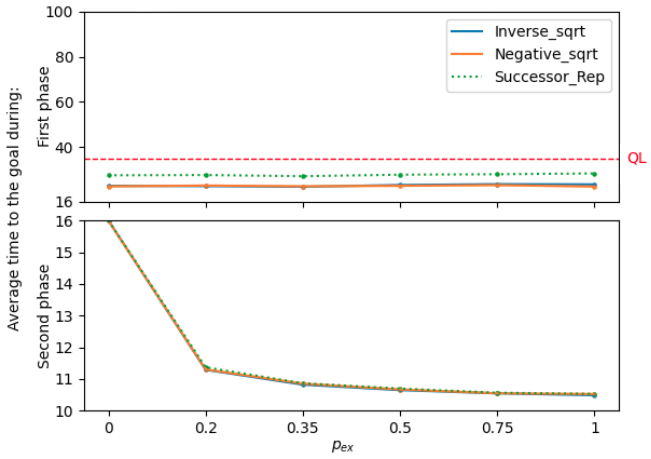

Figure 5. (lower is better) Varying p_{ex} for the ExploreOption.

that multi-step exploration, even with a random policy, is valuable. Interestingly, the agent finds the short path for the highest values of c_{switch}, even though performance is slower than the learnt agents, and less robust to c_{switch}.

The gap in performance between the agents with a learnt or random Explorer is already big, but it is bound to *widen considerably* with more complex environments and intrinsic reward functions, where the random behavior can become useless[5].

4.6. Probability of the ExploreOption.

As introduced in subsection 3.2.4, the p_{ex} hyperparameter allows to sample the options more frequently in an ϵ-greedy setting.

Fig 5 provides a study of the p_{ex} hyperparameter. For the special case $p_{ex} = 0$, it is interesting to see that although the method is nearly similar to QLearning, the agent can still select the option when breaking ties randomly after initialization, hence the agent still finds the first path faster than QLearning. Apart from $p_{ex} = 0$ none of the values prevent the agent from finding the shortest path, and the algorithm is relatively robust to it. Lower probabilities take slightly longer to find the path, since exploration is less frequent. We used the intuitive balance $p_{ex} = 0.5$ so far.

4.7. Option learning rate

In a tabular setting, each state-action pair (s, a) has its own Q value, and in particular, we learn the $Q(\cdot, a_{explore})$ values for all states. However, intuitively, proper usage of the option requires function approximation in order to generalize over states. To test this hypothesis, we tried setting

[5]Compare, for example, the exploration behavior of the learnt agent on Mario versus a random agent (Burda et al., 2019a)

the option learning rate to 0 – the option would only be used through ϵ-greedy, and not learnt. We found that *the performance did not change*: the agent did not need to *learn* to use the option to benefit form the Explorer's behavior off-policy. In the experiments so far, we used arbitrarily the learning rate $\alpha_{explore} = 0.001$, but any value $\alpha_{explore} \leq \alpha$ could work.

This observation is important, and we can summarize what we learnt from these experiments. The gap in performance of QL+EO with QL and QL+VC in this tabular setting can be attributed to just three main factors: first, multi-step exploration, i.e. $c_{switch} > 1$; second, an independent Explorer from which to learn off-policy; and third, a learnt (consistent) Explorer.

5. Discussion

5.1. Limitations

A major flaw of the algorithm presented here is the arbitrary, and potentially hard to tune, c_{switch} parameter. Several alternatives are possible, such as learning c_{switch} (Lakshminarayanan et al., 2017; Sharma et al., 2017); learning an option termination function (Bacon et al., 2017); or learning a `stop` action for the Explorer. The method proved robust to c_{switch} in a tabular setting; future work will study it in more complex environments.

Our method relies heavily on the quality of the f_{ir} functions to generate rewards that would independently lead to a powerful exploration policy. We should keep in mind that these functions are not perfect, and might not properly span the state-action space.

5.2. Learning from alternative signals

In Horde (Sutton et al., 2011), *General Value Functions* (GVFs) are a generalization of value-functions to *pseudo-rewards* and *pseudo-discounts*. Horde is then a collection of multiple GVF learners, called *demons*. *Control demons* learn action-value functions, while *prediction demons* learn values functions. This framework generates a massive **knowledge representation** of the environment based on value functions.

GVFs and Horde share many similarities with the framework of *auxiliary task learning* (Jaderberg et al., 2017; Mirowski et al., 2017). There the agent learns, in addition to the task, *auxiliary tasks* that can be control tasks (e.g. pixel control, feature control), or prediction tasks (e.g. Depth or reward prediction). The weights of the function approximator are shared with all tasks in order to build a powerful and general **representation**.

From these viewpoints, all our Explorers can be considered control tasks or demons, amassing knowledge about the environment through the reward function they try to maximize. In addition to this, an important observation from the intrinsic motivation literature is that virtually all methods rely on learning interesting properties about the MDP, e.g. a Forward Model, an Inverse Dynamics Model, the Successor Representation, or density models. All of these can then be seen as prediction tasks or demons, also gathering task-independent knowledge about the environment.

Therefore, combining everything, the final agent is a multi-headed function approximator, learning all Exploiter, Explorers and $f_{ir}^{(j)}$ jointly and off-policy. This agent not only builds a very strong task-independent representation; it also contains a lot of information about the environment - how to explore, or other behaviors; as well as predictive knowledge such as forward models or a Successor Representation. This agent would be very easily transferred across tasks – we only need to switch in a new Exploiter head. This will be the object of future work.

6. Conclusion

We introduced a new approach for learning from intrinsic motivation. We carried out experiments with QLearning agents on the simple non-stationary Shortcut Maze environment, where exploration is needed to find a shortcut. We demonstrated empirically that its sample-efficiency, adaptability and exploration performance vastly surpass that of the traditional approach, a weighted sum of rewards. We showed that:

1. Exploration through temporal abstractions over several steps, especially with a learnt Explorer, lead to much better sample-efficiency and exploration;
2. Our method adapts much better to miscellaneous intrinsic reward functions than the usual $R = R_e + \beta R_i$;
3. Decoupling exploiting and exploring policies improves both exploration and exploitation while preserving knowledge about the environment and scaling easily with several reward functions.

We presented a view from GVFs and auxiliary tasks, which vastly broadens the scope of the method for multiple Explorers and reward functions, giving glimpses of an agent with a strong representation, world knowledge, control knowledge and task-transfer abilities.

References

Achiam, J., Edwards, H., Amodei, D., and Abbeel, P. Variational Option Discovery Algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

Aubret, A., Matignon, L., and Hassas, S. A survey on Intrinsic Motivation in Reinforcement Learning. *arXiv preprint arXiv:1908.06976*, 2019.

Bacon, P.-L., Harb, J., and Precup, D. The Option-Critic Architecture. In *AAAI Conference on Artificial Intelligence*, 2017.

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

Barto, A. G. and Mahadevan, S. Recent Advances in Hierarchical Reinforcement Learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

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.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. Large-Scale Study of Curiosity-Driven Learning. In *International Conference on Learning Representations*, 2019a. URL https://openreview.net/forum?id=rJNwDjAqYX.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by Random Network Distillation. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=H1lJJnR5Ym.

Dayan, P. Improving Generalization for Temporal Difference Learning: The Successor Representation. *Neural Computation*, 5(4):613–624, 1993.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity Is All You Need: Learning Skills without a Reward Function. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SJx63jRqFm.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. Noisy Networks for Exploration. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rywHCPkAW.

Fox, L., Choshen, L., and Loewenstein, Y. DORA The Explorer: Directed Outreaching Reinforcement Action-Selection. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ry1arUgCW.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Dy, J. and Krause, A. (eds.), *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*, 2018.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. VIME: Variational Information Maximizing Exploration. In *Advances in Neural Information Processing Systems*, pp. 1109–1117, 2016.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *International Conference on Learning Representations*, 2017.

Kim, H., Kim, J., Jeong, Y., Levine, S., and Song, H. O. EMI: Exploration with Mutual Information. In *International Conference on Machine Learning*, 2019.

Lakshminarayanan, A. S., Sharma, S., and Ravindran, B. Dynamic Action Repetition for Deep Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*, 2017.

Machado, M. C., Bellemare, M. G., and Bowling, M. Count-Based Exploration with the Successor Representation. *arXiv preprint arXiv:1807.11622*, 2018.

Martin, J., Narayanan, S. S., Everitt, T., and Hutter, M. Count-Based Exploration in Feature Space for Reinforcement Learning. In *International Joint Conference on Artificial Intelligence*, pp. 2471–2478, 2017.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., and Hadsell, R. Learning to Navigate in Complex Environments. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJMGPrcle.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level Control through Deep Reinforcement Learning. *Nature*, 518 (7540):529–533, 2015. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., and Levine, S. Why Does Hierarchy (Sometimes) Work So Well in Reinforcement Learning? *CoRR*, abs/1909.10618, 2019. URL http://arxiv.org/abs/1909.10618.

Ostrovski, G., Bellemare, M. G., van den Oord, A., and Munos, R. Count-Based Exploration with Neural Density Models. In *International Conference on Machine Learning*, volume 70, pp. 2721–2730. JMLR. org, 2017.

Oudeyer, P.-Y. and Kaplan, F. What is Intrinsic Motivation? A Typology of Computational Approaches. *Frontiers in neurorobotics*, 1:6, 2009.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-Driven Exploration by Self-Supervised Prediction. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17, 2017.

Perotto, F. S. Looking for the Right Time to Shift Strategy in the Exploration-Exploitation Dilemma. *Schedae Informaticae*, 24:73–82, 2015.

Schmidhuber, J. Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

Sharma, S., Lakshminarayanan, A. S., and Ravindran, B. Learning to Repeat: Fine Grained Action Repetition for Deep Reinforcement Learning. In *International Conference on Learning Representations*, 2017.

Strehl, A. L. and Littman, M. L. An analysis of model-based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Sutton, R. S., Precup, D., and Singh, S. P. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.*, 112(1-2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1. URL https://doi.org/10.1016/S0004-3702(99)00052-1.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. Horde: a Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. In *International Conference on Autonomous Agents and Multiagent Systems*, volume 2, pp. 761–768, 2011.

Taiga, A. A., Fedus, W., Machado, M. C., Courville, A., and Bellemare, M. G. On Bonus Based Exploration Methods In The Arcade Learning Environment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJewlyStDr.

Watkins, C. J. and Dayan, P. Q-Learning. *Machine learning*, 8(3-4):279–292, 1992.

Williams, R. J. Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning. *Machine learning*, 8(3-4):229–256, 1992.