What Can Learned Intrinsic Rewards Capture?

Zeyu Zheng [*] [†] [1] Junhyuk Oh [*] [2] Matteo Hessel [2] Zhongwen Xu [2] Manuel Kroiss [2] Hado van Hasselt [2] David Silver [2] Satinder Singh [2]

Abstract

The objective of a reinforcement learning agent is to behave so as to maximise the sum of a suitable scalar function of state: the *reward*. These rewards are typically given and immutable. In this paper, we instead consider the proposition that the reward function itself can be a good locus of learned knowledge. To investigate this, we propose a scalable meta-gradient framework for learning useful intrinsic reward functions across multiple lifetimes of experience. Through several proof-of-concept experiments, we show that it is feasible to learn and capture knowledge about long-term exploration and exploitation into a reward function. Furthermore, we show that unlike policy transfer methods that capture "how" the agent should behave, the learned reward functions can generalise to other kinds of agents and to changes in the dynamics of the environment by capturing "what" the agent should strive to do.

1. Introduction

Reinforcement learning (RL) agents can store knowledge in their policies, value functions, state representations, and models of the environment dynamics. These components can be the *loci* of knowledge in the sense that they are structures in which knowledge, either learned from experience by the agent's algorithm or given by the agent-designer, can be deposited and reused. The objective of the agent is defined by a reward function, and the goal is to learn to act so as to maximise cumulative rewards. In this paper we consider the proposition that the reward function itself is a good locus of knowledge. This is unusual (but not novel) in that most prior work treats the reward as given and immutable, at least as far as the learning algorithm is concerned. In fact, agent designers often do find it convenient to modify the reward function given to the agent to facilitate learning. It is therefore useful to distinguish between two kinds of reward functions (Singh et al., 2010): *extrinsic* rewards define the task and capture the designer's preferences over agent behaviour, whereas *intrinsic* rewards serve as helpful signals to improve the learning dynamics of the agent.

Most existing work on intrinsic rewards falls into two broad categories: task-dependent and task-independent. Both are typically designed by hand. Hand-designing *task-dependent* rewards can be fraught with difficulty as even minor misalignment between the actual reward and the intended bias/goals can lead to unintended and sometimes catastrophic consequences (Clark & Amodei, 2016). *Task-independent* intrinsic rewards are also typically hand-designed, often based on an intuitive understanding of animal/human behaviour or on heuristics on desired exploratory behaviour. It can, however, be hard to match such task-independent intrinsic rewards to the specific learning dynamics induced by the interaction between agent and environment. In this paper, we are interested in the comparatively under-explored possibility of *learned* (not hand-designed) task-dependent intrinsic rewards. Although there have been a few attempts to learn useful intrinsic rewards from experience (Singh et al., 2009; Zheng et al., 2018), how to capture complex knowledge such as exploration across episodes into a reward function remains an open question.

We emphasise that it is *not* our objective to show that rewards are a *better* locus of learned knowledge than others; the best locus likely depends on the kind of knowledge that is most useful in a given task. In particular, knowledge captured in rewards provides guidance on "what" the agent should strive to do while knowledge captured in policies provides guidance on "how" an agent should behave. Knowledge about "what" captured in rewards is indirect and thus slower to make an impact on behaviour because it takes effect through learning, while knowledge about "how" can directly have an immediate impact on behaviour. At the same time, because of its indirectness the former can generalise better to changes in dynamics and different learning agents, as we empirically show in this paper.

[*]Equal contribution [†]Work done during an internship at DeepMind. [1]University of Michigan [2]DeepMind. Correspondence to: Zeyu Zheng <zeyu@umich.edu>, Junhyuk Oh <junhyuk@google.com>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

How should we measure the usefulness of a learned reward function? Ideally, we would like to measure the effect the learned reward function has on the learning dynamics. Of course, learning happens over multiple episodes, indeed it happens over an entire lifetime. Therefore, we choose *lifetime return*, the cumulative extrinsic reward obtained by the agent over its entire lifetime, as the main objective. To this end, we adopt the multi-lifetime setting of the Optimal Rewards Framework (Singh et al., 2009) in which an agent is initialised randomly at the start of each lifetime and then faces a stationary or non-stationary task drawn from some distribution. In this setting, the only knowledge that is transferred across lifetimes is the reward instead of the policy. Specifically, the goal is to learn a single intrinsic reward function that, when used to adapt the agent's policy using a standard episodic RL algorithm, ends up optimising the cumulative extrinsic reward over its lifetime.

In previous work, good reward functions were found via exhaustive search, limiting the range of applicability. We develop a more scalable gradient-based method for learning intrinsic rewards by exploiting the fact that the interaction between the policy update and the reward function is differentiable (Zheng et al., 2018). Moreover, unlike the prior work, we parameterise the reward function by a recurrent neural network unrolled over the entire lifetime and train it to maximise lifetime return, which is crucial for the reward function to capture long-term temporal dependencies (e.g., novelty of states across episodes). To handle long-term credit assignment that spans the lifetime, we use a lifetime value function that estimates the remaining lifetime return.

Our main contributions and findings are as follows: (1) Through carefully designed environments, we show that learned intrinsic reward functions can capture a rich form of knowledge such as long-term exploration (e.g., exploring uncertain states) and exploitation (e.g., anticipating environment changes) across multiple episodes. To our knowledge, this is the first work that shows the feasibility of learning such complex knowledge into reward functions. (2) We show that "what to do" knowledge captured by the reward functions can generalise to changing dynamics of the environment and new learning agents, whereas policy transfer methods do not generalise well, which provides insights into the usefulness of rewards as a locus of knowledge.

2. Related Work

Hand-designed Rewards There is a long history of work on designing rewards to accelerate learning in reinforcement learning. Reward shaping aims to design task-specific rewards towards known optimal behaviours, typically requiring domain knowledge. Both the benefits (Randlöv & Alström, 1998; Ng et al., 1999; Harutyunyan et al., 2015) and the difficulty (Clark & Amodei, 2016) of task-specific reward shaping have been studied. On the other hand, many intrinsic rewards have been proposed to encourage exploration, inspired by animal behaviours. Examples include prediction error (Schmidhuber, 1991a;b; Oudeyer et al., 2007; Gordon & Ahissar, 2011; Mirolli & Baldassarre, 2013; Pathak et al., 2017), surprise (Itti & Baldi, 2006), weight change (Linke et al., 2019), and state-visitation counts (Sutton, 1990; Poupart et al., 2006; Strehl & Littman, 2008; Bellemare et al., 2016; Ostrovski et al., 2017). Although these kinds of intrinsic rewards are not domain-specific, they are often not well-aligned with the task that the agent is solving, and ignore the effect on the agent's learning dynamics. In contrast, our work aims to learn intrinsic rewards from data that take into account the agent's learning dynamics without requiring prior knowledge from a human.

Rewards Learned from Experience There have been a few attempts to learn useful intrinsic rewards from data. Singh et al. (2009) introduced the Optimal Reward Framework which aims to find a good reward function that allows agents to solve a distribution of tasks using exhaustive search. The empirical study only showed simple intrinsic reward functions such as preference over certain objects due to the inefficient exhaustive search method employed. Although there have been follow-up works (Sorg et al., 2010; Guo et al., 2016) that use a gradient-based method, they consider a non-parameteric policy using Monte-Carlo Tree Search. Our work is closely related to LIRPG (Zheng et al., 2018) which proposed a meta-gradient method to learn intrinsic rewards. However, LIRPG considers a single task in a single lifetime with a myopic episode return objective, which is limited in that it does not allow exploration across episodes or generalisation to different agents. In contrast, our approach takes into account both the long-term effect of intrinsic rewards on the learning dynamics and the lifetime history of the agent. We show this is crucial for capturing long-term knowledge, such as seeking for novel states across episodes, which is not achieved in previous work.

Meta-learning for Exploration and Task Adaptation Meta-learning (Schmidhuber et al., 1996; Thrun & Pratt, 1998) has recently received considerable attention in RL. Recent advances include few-shot adaptation (Finn et al., 2017a), few-shot imitation (Finn et al., 2017b; Duan et al., 2017), model adaptation (Clavera et al., 2019), and inverse RL (Xu et al., 2019). In particular, our work is related to the prior work on meta-learning good exploration strategies (Wang et al., 2016; Duan et al., 2016; Stadie et al., 2018; Xu et al., 2018a) in that both perform temporal credit assignment across episode boundaries by maximising rewards accumulated beyond an episode. Unlike the prior work that aims to directly transfer an exploratory policy, our framework indirectly drives exploration via a reward function which can be reused by different learning agents.

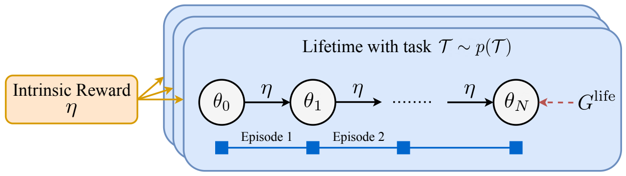


Figure 1. Illustration of the proposed intrinsic reward learning framework. The intrinsic reward r_η is used to update the agent's parameter θ_i throughout its lifetime which consists of many episodes. The goal is to find the optimal intrinsic reward parameters η^* across many lifetimes that maximises the lifetime return (G^{life}) given any randomly initialised agents and possibly non-stationary tasks drawn from some distribution $p(\mathcal{T})$.

Meta-learning Update Rules There have been a few studies that directly meta-learn how to update the agent's parameters via meta-parameters including discount factor and returns (Xu et al., 2018b), auxiliary tasks (Schlegel et al., 2018; Veeriah et al., 2019), unsupervised learning rules (Metz et al., 2019), and RL objectives (Bechtle et al., 2019). Our work also belongs to this category in that our meta-parameters are the reward function used in the agent's update. In particular, our multi-lifetime formulation is similar to ML^3 (Bechtle et al., 2019). However, the meta-learned loss in ML^3 cannot generalise to different agent-environment interfaces, whereas intrinsic rewards can as shown in Section 6. In addition, we propose to use the lifetime return as opposed to the myopic episodic objective of ML^3, which is crucial for cross-episode exploration.

Cognitive Study on Exploration-Exploitation. Several cognitive science studies on the exploration-exploitation dilemma (Cohen et al., 2007; Wilson et al., 2014) have shown that humans use both a random exploration strategy (Thompson, 1933; Watkins, 1989) and an information-seeking strategy (Gittins, 1974; 1979) when facing uncertainty. Computationally, the former can be easily implemented, whereas the latter usually requires carefully hand-crafted methods to guide the agent's behaviour. In this work, we hypothesize and empirically verify that an information-seeking intrinsic reward function can naturally emerge if it is *useful* for solving the tasks. The condition of being useful resembles a recent study (Dubey & Griffiths, 2019) which posited that a rational agent should explore in a way such that the usefulness of its knowledge is maximised.

3. The Optimal Reward Problem

We first introduce some terminology.

- **Agent**: A learning system interacting with an environment. On each step t the agent selects an action a_t and receives from the environment an observation s_{t+1} and an *extrinsic* reward r_{t+1} defined by a task $\mathcal{T}$. The agent chooses actions based on a policy $\pi_\theta(a_t|s_t)$ parameterised by θ.

- **Episode**: A finite sequence of agent-environment interactions until the end of the episode defined by the task. An episode return is defined as: $G^{\text{ep}} = \sum_{t=0}^{T_{\text{ep}}-1} \gamma^t r_{t+1}$, where γ is a discount factor, and the random variable T_{ep} gives the number of steps until the end of the episode.

- **Lifetime**: A finite sequence of agent-environment interactions until the end of training defined by an agent-designer, which can consist of multiple episodes. The *lifetime return* is $G^{\text{life}} = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$, where γ is a discount factor, and T is the number of steps in the lifetime.

- **Intrinsic reward**: A reward function $r_\eta(\tau_{t+1})$ parameterised by η, where $\tau_t = (s_0, a_0, r_1, d_1, s_1, \ldots, r_t, d_t, s_t)$ is a lifetime history with (binary) episode terminations d_i.

The Optimal Reward Problem (Singh et al., 2010), illustrated in Figure 1, aims to learn the parameters of the intrinsic reward such that the resulting rewards achieve a learning dynamic for an RL agent that maximises the lifetime (extrinsic) return on tasks drawn from some distribution. Formally, the objective function is defined as:

$$J(\eta) = \mathbb{E}_{\theta_0 \sim \Theta, \mathcal{T} \sim p(\mathcal{T})} \left[\mathbb{E}_{\tau \sim p_\eta(\tau|\theta_0)} \left[G^{\text{life}} \right] \right], \quad (1)$$

where Θ and $p(\mathcal{T})$ are an initial policy distribution and a distribution over possibly non-stationary tasks respectively. The likelihood of a lifetime history τ is $p_\eta(\tau|\theta_0) = p(s_0) \prod_{t=0}^{T-1} \pi_{\theta_t}(a_t|s_t) p(d_{t+1}, r_{t+1}, s_{t+1}|s_t, a_t)$, where $\theta_t = f(\theta_{t-1}, \eta)$ is a policy parameter as updated with update function f, which is policy gradient in this paper.[1] Note that the optimisation of η spans multiple lifetimes, each of which can span multiple episodes.

Using the lifetime return G^{life} as the objective instead of the conventional episodic return G^{ep} allows exploration across multiple episodes as long as the lifetime return is maximised in the long run. In particular, when the lifetime is defined as a fixed number of episodes, we find that the lifetime return objective is sometimes more beneficial than the episodic return objective, even for the episodic return performance measure. However, different objectives (e.g., final episode return) can be considered depending on the definition of what a good reward function is.

[1] We assume that the policy parameter is updated after each time-step throughout the paper for brevity. However, the parameter can be updated less frequently in practice.

Algorithm 1 Learning intrinsic rewards

```
Input: $p(\mathcal{T})$: Task distribution
Input: $\Theta$: Randomly-initialised policy distribution
Initialise intrinsic reward $\eta$ and lifetime value $\phi$
repeat
  Initialise task $\mathcal{T} \sim p(\mathcal{T})$ and policy $\theta \sim \Theta$
  while lifetime not ended do
    $\theta_0 \leftarrow \theta$
    for $k = 1, 2, \dots, N$ do
      Generate a trajectory using $\pi_{\theta_{k-1}}$
      Update policy $\theta_k \leftarrow \theta_{k-1} + \alpha \nabla_{\theta_{k-1}} J_\eta(\theta_{k-1})$
      using intrinsic rewards $r_\eta$ (Eq. 3)
    end for
    Update intrinsic reward function $\eta$ using Eq. 4
    Update lifetime value function $\phi$ using Eq. 6
    $\theta \leftarrow \theta_N$
  end while
until $\eta$ converges
```

4. Meta-Learning Intrinsic Reward

We propose a meta-gradient approach (Xu et al., 2018b; Zheng et al., 2018) to solve the optimal reward problem. At a high-level, we sample a new task $\mathcal{T}$ and a new random policy parameter θ at each lifetime iteration. We then simulate an agent's lifetime by updating the parameter θ using an intrinsic reward function r_η (Section 4.1) with policy gradient (Section 4.2). Concurrently, we compute the meta-gradient by taking into account the effect of the intrinsic rewards on the policy parameters to update the intrinsic reward function with a lifetime value function (Section 4.3). Algorithm 1 gives an overview of our algorithm. The following sections describe the details.

4.1. Architectures

The intrinsic reward function is a recurrent neural network (RNN) parameterised by η, which produces a scalar reward on arriving in state s_t by taking into account the history of an agent's lifetime $\tau_t = (s_0, a_0, r_1, d_1, s_1, ..., r_t, d_t, s_t)$. We claim that giving the lifetime history across episodes as input is crucial for balancing exploration and exploitation, for instance by capturing how frequently a certain state is visited to determine an exploration bonus reward. The lifetime value function is a separate recurrent neural network parameterised by ϕ, which takes the same inputs as the intrinsic reward function and produces a scalar value estimation of the expected future return within the lifetime.

4.2. Policy Update

Each agent interacts with an environment and a task sampled from a distribution $\mathcal{T} \sim p(\mathcal{T})$. However, instead of directly maximising the extrinsic rewards defined by the task, the agent maximises the intrinsic rewards (r_η) by using policy gradient (Williams, 1992; Sutton et al., 2000):

$$J_\eta(\theta) = \mathbb{E}_\theta\left[\sum_{t=0}^{T_{\text{ep}}-1} \bar{\gamma}^t r_\eta(\tau_{t+1})\right] \tag{2}$$

$$\nabla_\theta J_\eta(\theta) = \mathbb{E}_\theta\left[G^{\text{ep}}_{\eta,t} \nabla_\theta \log \pi_\theta(a|s)\right], \tag{3}$$

where $r_\eta(\tau_{t+1})$ is the intrinsic reward at time t, and $G^{\text{ep}}_{\eta,t} = \sum_{k=t}^{T_{\text{ep}}-1} \bar{\gamma}^{k-t} r_\eta(\tau_{k+1})$ is the return of the intrinsic rewards accumulated over an episode with discount factor $\bar{\gamma}$.

4.3. Intrinsic Reward and Lifetime Value Update

To update the intrinsic reward parameters η, we directly take a meta-gradient ascent step using the overall objective (Equation 1). Specifically, the gradient is (see the supplementary material for derivation)

$$\nabla_\eta J(\eta) = \mathbb{E}_{\theta_t, \mathcal{T}}\left[G^{\text{life}}_t \nabla_{\theta_t} \log \pi_{\theta_t}(a_t|s_t) \nabla_\eta \theta_t\right], \tag{4}$$

The chain rule is used to get the meta-gradient ($\nabla_\eta \theta_t$) as in previous work (Zheng et al., 2018). The computation graph of this procedure is illustrated in Figure 1.

Computing the true meta-gradient in Equation 4 requires backpropagation through the entire lifetime, which is infeasible as each lifetime can involve thousands of policy updates. To partially address this issue, we truncate the meta-gradient after N policy updates but approximate the lifetime return $G^{\text{life},\phi}_t \approx G^{\text{life}}_t$ using a *lifetime value function* $V_\phi(\tau)$ parameterised by ϕ, which is learned using a temporal difference learning from n-step trajectory:

$$G^{\text{life},\phi}_t = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\phi(\tau_{t+n}) \tag{5}$$

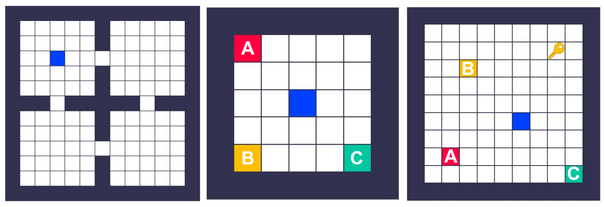
$$\phi' = \phi + \alpha'(G^{\text{life},\phi}_t - V_\phi(\tau_t))\nabla_\phi V_\phi(\tau_t), \tag{6}$$

where α' is a learning rate. In our empirical work, we found that the lifetime value estimates were crucial to allow the intrinsic reward to perform long-term credit assignments across episodes (Section 5.6).

5. Empirical Investigations

We present the results from our empirical investigations in two sections. In this section, the experiments and domains are designed to answer the following research questions:

- What kind of knowledge is learned by the intrinsic reward?
- How does the distribution of tasks influence the intrinsic reward?



(a) Empty Rooms (b) ABC (c) Key-Box

Figure 2. Illustration of domains. (a) The agent needs to find the goal location which gives a positive reward, but the goal is not visible to the agent. (b) Each object (A, B, and C) gives rewards. (c) The agent is required to first collect the key and visit one of the boxes (A, B, and C) to receive the corresponding reward. All objects are placed in random locations before each episode.

- What is the benefit of the lifetime return objective over the episode return?
- When is it important to provide the lifetime history as input to the intrinsic reward?

5.1. Experimental Setup

We investigate these research questions in the grid-world domains illustrated in Figure 2. For each domain, we trained an intrinsic reward function across many lifetimes and evaluated it by training an agent using the learned reward. We implemented the following baselines.

- Extrinsic-EP: A policy is trained with extrinsic rewards to maximise the episode return.
- Extrinsic-LIFE: A policy is trained with extrinsic rewards to maximise the lifetime return.
- Count-based (Strehl & Littman, 2008): A policy is trained with extrinsic rewards and count-based exploration bonus rewards.
- ICM (Pathak et al., 2017): A policy is trained with extrinsic rewards and curiosity rewards based on an inverse dynamics model.

Note that these baselines, unlike the learned intrinsic rewards, do not transfer any knowledge across different lifetimes. Throughout Sections 5.2-5.5, we focus on analysing what kind of knowledge is learned by the intrinsic reward depending on the nature of environments. We discuss the benefit of using the lifetime return and considering the lifetime history when learning the intrinsic reward in Section 5.6. The details of implementation and hyperparameters are described in the supplementary material.

5.2. Exploring Uncertain States

We designed 'Empty Rooms' (Figure 2a) to see whether the intrinsic reward can learn to encourage exploration of uncertain states like novelty-based exploration methods. The goal is to visit an invisible goal location, which is fixed within each lifetime but varies across lifetimes. An episode terminates when the goal is reached. Each lifetime consists of 200 episodes. From the agent's perspective, its policy should visit the locations suggested by the intrinsic reward. From the intrinsic reward's perspective, it should encourage the agent to go to unvisited locations to locate the goal, and then to exploit that knowledge for the rest of the lifetime.

Figure 3 shows that the learned intrinsic reward was more efficient than extrinsic rewards and count-based exploration when training a new agent. We observed that the intrinsic reward learned two interesting strategies as visualised in Figure 4. While the goal is not found, it encourages exploration of unvisited locations, because it learned the knowledge that there exists a rewarding goal location somewhere. Once the goal is found the intrinsic reward encourages the agent to exploit it without further exploration, because it learned that there is only one goal. This result shows that curiosity about uncertain states can naturally emerge when various states can be rewarding in a domain, even when the rewarding states are fixed within an agent's lifetime.

5.3. Exploring Uncertain Objects

In the previous domain, we considered uncertainty of where the reward (or goal location) is. We now consider dealing with uncertainty about the value of different objects. In the 'Random ABC' environment (see Figure 2b), for each lifetime the rewards for objects A, B, and C are uniformly sampled from $[-1, 1]$, $[-0.5, 0]$, and $[0, 0.5]$ respectively but are held fixed within the lifetime. A good intrinsic reward should learn that: 1) B should be avoided, 2) A and C have uncertain rewards, hence require systematic exploration (first go to one and then the other), and 3) once it is determined which of the two A or C is better, exploit that knowledge by encouraging the agent to repeatedly go to that object for the rest of the lifetime.

Figure 3 shows that the agent learned a near-optimal exploration-and-then-exploitation method with the learned intrinsic reward. Note that the agent cannot pass information about the reward for objects across episodes, as usual in reinforcement learning. The intrinsic reward can propagate such information across episodes and help the agent explore or exploit appropriately. We visualised the learned intrinsic reward for different actions sequences in Figure 5. The intrinsic rewards encourage the agent to explore towards A and C in the first few episodes. Once A and C are explored, the agent exploits the largest rewarding object. Throughout training, the agent is discouraged to visit B through negative intrinsic rewards. These results show that avoidance and curiosity about uncertain objects can potentially emerge if the environment has various or fixed rewarding objects.

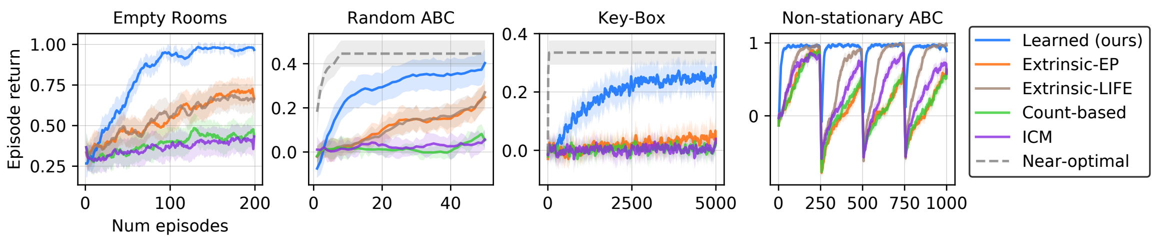


Figure 3. Evaluation of different reward functions averaged over 30 seeds. The learning curves show agents trained with our intrinsic reward (blue), with the extrinsic reward using the episodic return objective (orange) or the lifetime return objective (brown), and with a count-based exploration reward (green). The dashed line corresponds to a hand-designed near-optimal exploration strategy.

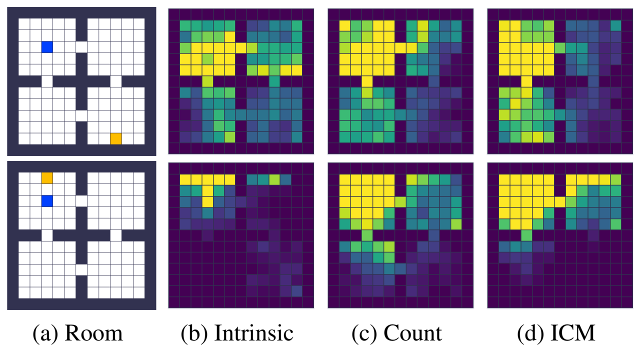


Figure 4. Visualisation of the first 3000 steps of an agent trained with different reward functions in Empty Rooms. (a) The blue and yellow squares represent the agent and the *hidden* goal, respectively. (b) The learned reward encourages the agent to visit many locations if the goal is not found (top). However, when the goal is found early, the intrinsic reward makes the agent exploit it without further exploration (bottom). (c-d) Both the count-based and ICM rewards tend to encourage exploration (top) but hinders exploitation when the goal is found (bottom).

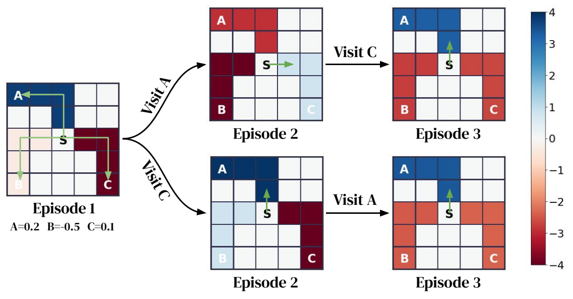


Figure 5. Visualisation of the learned intrinsic reward in Random ABC, where the extrinsic rewards for A, B, and C are 0.2, -0.5, and 0.1 respectively. Each figure shows the sum of intrinsic rewards for a trajectory towards each object (A, B, and C). In the first episode, the intrinsic reward encourages the agent to explore A. In the second episode, the intrinsic reward encourages exploring C if A is visited (top) or vice versa (bottom). In episode 3, after both A and C are explored, the intrinsic reward encourages revisiting A (both top and bottom).

5.4. Exploiting Invariant Causal Relationship

To see how the intrinsic reward deals with causal relationship between objects, we designed 'Key-Box', which is similar to Random ABC except that there is a key in the room (see Figure 2c). The agent needs to collect the key first to open one of the boxes (A, B, and C) and receive the corresponding reward. The rewards for the objects are sampled from the same distribution as Random ABC. The key itself gives a neutral reward of 0. Moreover, the locations of the agent, the key, and the boxes are randomly sampled for each episode. As a result, the state space contains more than 3 billion distinct states and thus is infeasible to fully enumerate. Figure 3 shows that learned intrinsic reward leads to a near-optimal exploration. The agent trained with extrinsic rewards did not learn to open any box. The intrinsic reward captures that the key is necessary to open any box, which is true across many lifetimes of training. This demonstrates that the intrinsic reward can capture causal relationships between objects when the domain has this kind of invariant dynamics.

5.5. Dealing with Non-stationarity

We investigated how the intrinsic reward handles non-stationary tasks within a lifetime in our 'Non-stationary ABC' environment. Rewards are as follows: for A is either 1 or -1, for B is -0.5, for C is the negative value of the reward for A. The rewards of A and C are swapped every 250 episodes. Each lifetime lasts 1000 episodes. Figure 3 shows that the agent with the learned intrinsic reward quickly recovered its performance when the task changes, whereas the baselines take more time to recover. Figure 6 shows how the learned intrinsic reward encourages the learning agent to react to the changing rewards. Interestingly, the intrinsic reward has learned to prepare for the change by giving negative rewards to the exploitation policy of the agent a few episodes before the task changes. In other words, the intrinsic reward reduces the agent's commitment to the current best rewarding object, thereby increasing entropy in the current policy in anticipation of the change, eventually making it easier to adapt quickly. This shows that the intrinsic

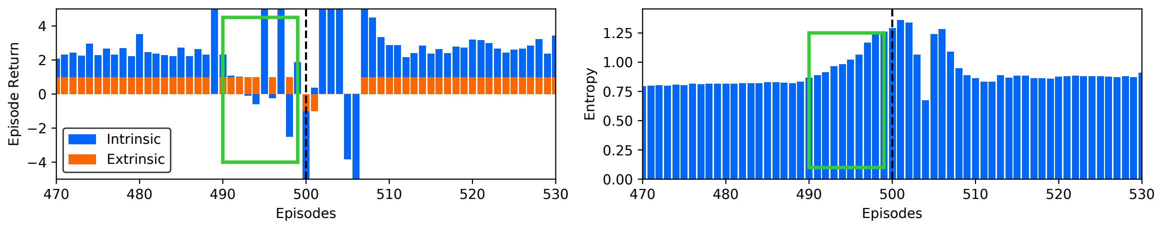


Figure 6. Visualisation of the agent's intrinsic and extrinsic rewards (left) and the entropy of its policy (right) on Non-stationary ABC. The task changes at 500th episode (dashed vertical line). The intrinsic reward gives a negative reward even before the task changes (green rectangle) and makes the policy less deterministic (entropy increases). As a result, the agent quickly adapts to the change.

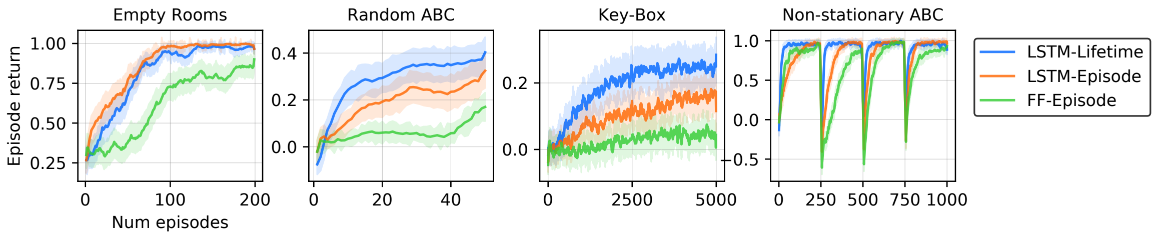


Figure 7. Evaluation of different intrinsic reward architectures and objectives. For 'LSTM' the reward network has an LSTM taking the lifetime history as input. For 'FF' a feed-forward reward network takes only the current time-step. 'Lifetime' and 'Episode' means the lifetime and episodic return as objective respectively.

reward can capture the (regularly) repeated non-stationarity across many lifetimes and make the agent intrinsically motivated not to commit too firmly to a policy, in anticipation of changes in the environment.

5.6. Ablation Study

To study relative benefits of the proposed technical ideas, we conducted an ablation study 1) by replacing the long-term lifetime return objective (G^{life}) with the episodic return (G^{ep}) and 2) by restricting the input of the reward network to the current time-step instead of the entire lifetime history. Figure 7 shows that the lifetime history was crucial to achieve good performance. This is reasonable because all domains require some past information (e.g., object rewards in Random ABC, visited locations in Empty Rooms) to provide useful exploration strategies. It is also shown that the lifetime return objective was beneficial on Random ABC, Non-stationary ABC, and Key-Box. These domains require exploration across multiple episodes in order to find the optimal policy. For example, collecting an uncertain object (e.g., object A in Random ABC) is necessary even if the episode terminates with a negative reward. The episodic value function would directly penalise such an under-performing exploratory episode when computing meta-gradient, which prevents the intrinsic reward from learning to encourage exploration across episodes. On the other hand, such behaviour can be encouraged by the lifetime value function, as long as it provides useful information to maximise the lifetime return in the long term.

6. Generalisation via Rewards

As noted above, rewards capture knowledge about what an agent's goals should be rather than how it should behave. At the same time, transferring the latter in the form of policies is also feasible in our domains presented above. Here we confirm it by implementing and presenting results for the following two meta-learning methods:

- MAML (Finn et al., 2017a): A policy meta-learned from a distributions of tasks such that it can adapt quickly to the given task after a few parameter updates.
- RL^2 (Duan et al., 2016; Wang et al., 2016): An RNN policy unrolled over the entire lifetime to maximise the lifetime return, which is pre-trained on a distributions of tasks.

Although all the methods we implemented including ours are designed to learn useful knowledge from a distribution of tasks, they have different objectives. Specifically, the objective of our method is to learn knowledge that is useful for training "randomly-initialised policies" by capturing "what to do", whereas the goal of policy transfer methods is to directly transfer a useful policy for fast task adaptation by transferring "how to do" knowledge. In fact, it can be more efficient to transfer and reuse pre-trained policies instead of restarting from a random policy and learning using the learned rewards given a new task. Figure 8 indeed shows that RL^2 performs better than our intrinsic reward

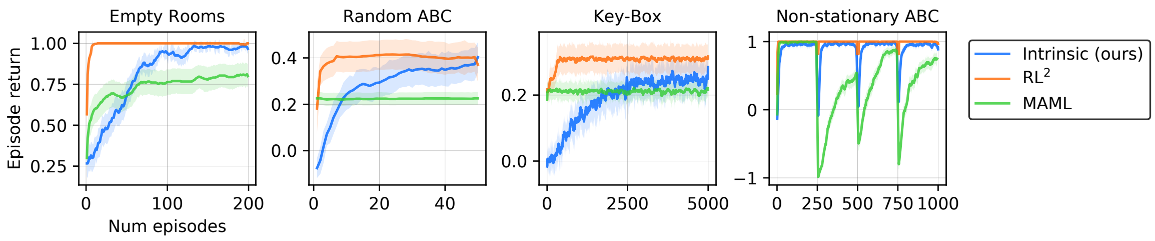
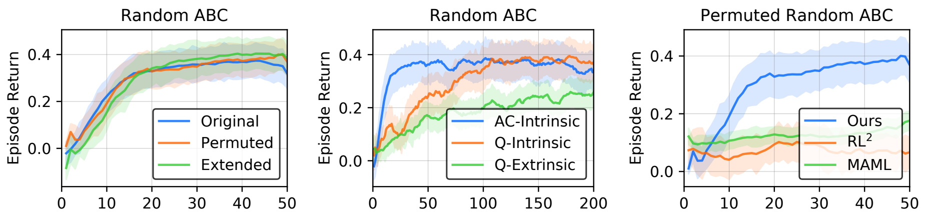


Figure 8. Comparison to policy transfer methods.



(a) Action space (b) Algorithm (c) Comparison to baselines

Figure 9. Generalisation to new agent-environment interfaces in Random ABC. (a) 'Permuted' agents have different action semantics. 'Extended' agents have additional actions. (b) 'AC-Intrinsic' is the original actor-critic agent trained with the intrinsic reward. 'Q-Intrinsic' is a Q-learning agent with the intrinsic reward learned from actor-critic agents. 'Q-Extrinsic' is the Q-learning agent with the extrinsic reward. (c) The performance of the policy transfer baselines with permuted actions during evaluation.

approach. It is also shown that MAML and RL^2 achieve good performance from the beginning, as they have already learned how to navigate the grid worlds and how to achieve the goals of the tasks. In our method, on the other hand, the agent starts from a random policy and relies on the learned intrinsic reward which indirectly tells it what to do. Nevertheless, our method outperforms MAML and achieves a comparable asymptotic performance to RL^2.

6.1. Generalise to New Agent-Environment Interfaces

In fact, our method can be interpreted as an instance of RL^2 with a particular decomposition of parameters (θ and η), which uses policy gradient as a recurrent update (see Figure 1). While this modular structure may not be more beneficial than RL^2 when evaluated with the same agent-environment interface, such a decomposition provides clear semantics of each module: the policy (θ) captures "how to do" while the intrinsic reward (η) captures "what to do", and this enables interesting kinds of generalisations as we show below. Specifically, we show that "what" knowledge captured by the intrinsic reward can be reused by many different learning agents as follows.

Generalise to Unseen Action Spaces We first evaluated the learned intrinsic reward on new action spaces. Specifically, the intrinsic reward was used to train new agents with either 1) permuted actions, where the semantics of left/right and up/down are reversed, or 2) extended actions, with 4 additional actions that move diagonally. Figure 9a shows that the intrinsic reward provided useful rewards to new agents with different actions, though it was not trained with those actions. This is feasible because the intrinsic reward assigns rewards to the agent's state changes rather than its actions. The intrinsic reward captures "what to do", which makes it feasible to generalise to new actions, as long as the goal remains the same. On the other hand, it is unclear how to generalise RL^2 and MAML in this way.

Generalise to Unseen Learning Algorithms We further investigated how general the learned intrinsic reward is by evaluating it on agents with different learning algorithms. Specifically, after training the intrinsic reward from actor-critic agents, we evaluated it by training new agents through Q-learning while using the learned intrinsic reward as denoted by 'Q-Intrinsic' in Figure 9b. Interestingly, it turns out that the learned intrinsic reward is general enough to be useful for Q-learning agents, even though it was trained for actor-critic agents. Again, it is unclear how to generalise RL^2 and MAML in this way.

Comparison to Policy Transfer Although it is impossible to apply the learned policy from RL^2 and MAML when we extend the action space or when we change the learning algorithm, we can do so when we only permute the actions. As shown in Figure 9c, both RL^2 and MAML generalise poorly when the action space is permuted for Random ABC, because the transferred policies are highly biased to the original action space. Again, this result highlights the difference between "what to do" knowledge captured by our approach and "how to do" knowledge captured by policies.

7. Conclusion

We revisited the optimal reward problem (Singh et al., 2009) and proposed a more scalable gradient-based method for learning intrinsic rewards across lifetimes. Through several proof-of-concept experiments, we showed that the learned non-stationary intrinsic reward can capture regularities within a distribution of environments or, over time, within a non-stationary environment. As a result, they were capable of encouraging both exploratory and exploitative behaviour across multiple episodes. In addition, some task-independent notions of intrinsic motivation such as curiosity emerged when they were effective for the distribution over tasks across lifetimes the agent was trained on. We also showed that the learned intrinsic rewards can generalise to different agent-environment interfaces such as different action spaces and different learning algorithms, whereas policy transfer methods fail to generalise to such changes. This highlights the difference between the "what" kind of knowledge captured by rewards and the "how" kind of knowledge captured by policies. The flexibility and range of knowledge captured by intrinsic rewards in our proof-of-concept experiments encourages further work towards combining different loci of knowledge to achieve greater practical benefits.

Acknowledgement

We thank Joseph Modayil for his helpful feedback on the manuscript.

References

Bechtle, S., Molchanov, A., Chebotar, Y., Grefenstette, E., Righetti, L., Sukhatme, G., and Meier, F. Meta-learning via learned loss. *arXiv preprint arXiv:1906.05374*, 2019.

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.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., and Wanderman-Milne, S. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Clark, J. and Amodei, D. Faulty reward functions in the wild. *CoRR*, 2016. URL https://blog.openai.com/.

Clavera, I., Nagabandi, A., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyztsoC5Y7.

Cohen, J. D., McClure, S. M., and Yu, A. J. Should i stay or should i go? how the human brain manages the trade-off between exploitation and exploration. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 362(1481):933–942, 2007.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. RL^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Duan, Y., Andrychowicz, M., Stadie, B., Ho, O. J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. One-shot imitation learning. In *Advances in Neural Information Processing Systems*, pp. 1087–1098, 2017.

Dubey, R. and Griffiths, T. L. Reconciling novelty and complexity through a rational analysis of curiosity. *Psychological Review*, 2019.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017a.

Finn, C., Yu, T., Zhang, T., Abbeel, P., and Levine, S. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pp. 357–368, 2017b.

Gittins, J. A dynamic allocation index for the sequential design of experiments. *Progress in statistics*, pp. 241–266, 1974.

Gittins, J. C. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society: Series B (Methodological)*, 41(2):148–164, 1979.

Gordon, G. and Ahissar, E. Reinforcement active learning hierarchical loops. In *The 2011 International Joint Conference on Neural Networks*, pp. 3008–3015. IEEE, 2011.

Guo, X., Singh, S., Lewis, R., and Lee, H. Deep learning for reward design to improve monte carlo tree search in atari games. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pp. 1519–1525. AAAI Press, 2016.

Harutyunyan, A., Devlin, S., Vrancx, P., and Nowe, A. Expressing arbitrary reward functions as potential-based advice. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 2652–2658. AAAI Press, 2015.

Itti, L. and Baldi, P. F. Bayesian surprise attracts human attention. In *Advances in neural information processing systems*, pp. 547–554, 2006.

Linke, C., Ady, N. M., White, M., Degris, T., and White, A. Adapting behaviour via intrinsic reward: A survey and empirical study. *arXiv preprint arXiv:1906.07865*, 2019.

Metz, L., Maheswaranathan, N., Cheung, B., and Sohl-Dickstein, J. Meta-learning update rules for unsupervised representation learning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HkNDsiC9KQ`.

Mirolli, M. and Baldassarre, G. Functions and mechanisms of intrinsic motivations. In *Intrinsically Motivated Learning in Natural and Artificial Systems*, pp. 49–72. Springer, 2013.

Ng, A. Y., Harada, D., and Russell, S. J. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287. Morgan Kaufmann Publishers Inc., 1999.

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

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

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2778–2787. JMLR. org, 2017.

Poupart, P., Vlassis, N., Hoey, J., and Regan, K. An analytic solution to discrete bayesian reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 697–704. ACM, 2006.

Randlöv, J. and Alström, P. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 463–471. Morgan Kaufmann Publishers Inc., 1998.

Schlegel, M., Patterson, A., White, A., and White, M. Discovery of predictive representations with a network of general value functions, 2018. URL `https://openreview.net/forum?id=ryZElGZ0Z`.

Schmidhuber, J. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pp. 1458–1463, 1991a.

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

Schmidhuber, J., Zhao, J., and Wiering, M. Simple principles of metalearning. *Technical report IDSIA*, 69:1–23, 1996.

Singh, S., Lewis, R. L., and Barto, A. G. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pp. 2601–2606. Cognitive Science Society, 2009.

Singh, S., Lewis, R. L., Barto, A. G., and Sorg, J. Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2010.

Sorg, J., Lewis, R. L., and Singh, S. Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems*, pp. 2190–2198, 2010.

Stadie, B., Yang, G., Houthooft, R., Chen, P., Duan, Y., Wu, Y., Abbeel, P., and Sutskever, I. The importance of sampling inmeta-reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 9280–9290, 2018.

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. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pp. 216–224. Morgan Kaufmann, 1990.

Sutton, R. S., McAllester, D. A., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pp. 1057–1063, 2000.

Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

Thrun, S. and Pratt, L. Learning to learn: Introduction and overview. In *Learning to learn*, pp. 3–17. Springer, 1998.

Veeriah, V., Hessel, M., Xu, Z., Rajendran, J., Lewis, R. L., Oh, J., van Hasselt, H. P., Silver, D., and Singh, S. Discovery of useful questions as auxiliary tasks. In *Advances in Neural Information Processing Systems*, pp. 9306–9317, 2019.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. M. Learning to reinforcement learn. *ArXiv*, abs/1611.05763, 2016.

Watkins, C. J. C. H. Learning from delayed rewards. 1989.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Wilson, R. C., Geana, A., White, J. M., Ludvig, E. A., and Cohen, J. D. Humans use directed and random exploration to solve the explore–exploit dilemma. *Journal of Experimental Psychology: General*, 143(6):2074, 2014.

Xu, K., Ratner, E., Dragan, A., Levine, S., and Finn, C. Learning a prior over intent via meta-inverse reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 6952–6962, 2019.

Xu, T., Liu, Q., Zhao, L., and Peng, J. Learning to explore via meta-policy gradient. In *International Conference on Machine Learning*, pp. 5459–5468, 2018a.

Xu, Z., van Hasselt, H. P., and Silver, D. Meta-gradient reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2396–2407, 2018b.

Zheng, Z., Oh, J., and Singh, S. On learning intrinsic rewards for policy gradient methods. In *Advances in Neural Information Processing Systems*, pp. 4644–4654, 2018.

A. Derivation of Intrinsic Reward Update

Following the conventional notation in RL, we define $v_{\mathcal{T}}(\tau_t|\eta, \theta_0)$ as the state-value function that estimates the expected future lifetime return given the lifetime history τ_t, the task $\mathcal{T}$, initial policy parameters θ_0 and the intrinsic reward parameters η. Specially, $v_{\mathcal{T}}(\tau_0|\eta, \theta_0)$ denotes the expected lifetime return at the starting state, i.e.,

$$v_{\mathcal{T}}(\tau_0|\eta, \theta_0) = \mathbb{E}_{\tau \sim p_\eta(\tau|\theta_0)}\left[G^{\text{life}}\right],$$

where G^{life} denotes the lifetime return in task $\mathcal{T}$. We also define the action-value function $q_{\mathcal{T}}(\tau_t, a_t|\eta, \theta_0)$ accordingly as the expected future lifetime return given the lifetime history τ_t and an action a_t.

The objective function of the optimal reward problem is defined as:

$$\begin{aligned} J(\eta) &= \mathbb{E}_{\theta_0 \sim \Theta, \mathcal{T} \sim p(\mathcal{T})}\left[\mathbb{E}_{\tau \sim p_\eta(\tau|\theta_0)}\left[G^{\text{life}}\right]\right] && (7) \\ &= \mathbb{E}_{\theta_0 \sim \Theta, \mathcal{T} \sim p(\mathcal{T})}\left[v_{\mathcal{T}}(\tau_0|\eta, \theta_0)\right], && (8) \end{aligned}$$

where Θ and $p(\mathcal{T})$ are an initial policy distribution and a task distribution respectively.

Assuming the task $\mathcal{T}$ and the initial policy parameters θ_0 are given, we omit $\mathcal{T}$ and θ_0 for the rest of equations for simplicity. Let $\pi_\eta(\cdot|\tau_t) = \pi_{\theta_t}(\cdot|s_t)$ be the probability distribution over actions at time t given the history τ_t, where $\theta_t = f_\eta(\tau_t, \theta_0)$ is the policy parameters at time t in the lifetime. We can derive the meta-gradient with respect to η by the following:

$$\begin{aligned} &\nabla_\eta J(\eta) \\ &= \nabla_\eta v(\tau_0|\eta) \\ &= \nabla_\eta \left[\sum_{a_0} \pi_{\theta_0}(a_0|\tau_0) q(\tau_0, a_0|\eta)\right] \\ &= \sum_{a_0} \left[\nabla_\eta \pi_{\theta_0}(a_0|\tau_0) q(\tau_0, a_0|\eta) + \pi_{\theta_0}(a_0|\tau_0) \nabla_\eta q(\tau_0, a_0|\eta)\right] \\ &= \sum_{a_0} \left[\nabla_\eta \pi_{\theta_0}(a_0|\tau_0) q(\tau_0, a_0|\eta) + \pi_{\theta_0}(a_0|\tau_0) \nabla_\eta \sum_{\tau_1, r_0} p(\tau_1, r_0|\tau_0, a_0)\Big(r_0 + v(\tau_1|\eta)\Big)\right] \\ &= \sum_{a_0} \left[\nabla_\eta \pi_{\theta_0}(a_0|\tau_0) q(\tau_0, a_0|\eta) + \pi_{\theta_0}(a_0|\tau_0) \sum_{\tau_1} p(\tau_1|\tau_0, a_0) \nabla_\eta v(\tau_1|\eta)\right] \\ &= \mathbb{E}_{\tau_t}\left[\sum_{a_t} \nabla_\eta \pi_\eta(a_t|\tau_t) q(\tau_t, a_t|\eta)\right] \\ &= \mathbb{E}_{\tau_t}\left[\nabla_\eta \log \pi_\eta(a_t|\tau_t) q(\tau_t, a_t|\eta)\right] \\ &= \mathbb{E}_{\tau_t}\left[G_t \nabla_\eta \log \pi_\eta(a_t|\tau_t)\right] \\ &= \mathbb{E}_{\tau_t}\left[G_t \nabla_{\theta_t} \log \pi_{\theta_t}(a_t|s_t) \nabla_\eta \theta_t\right], \end{aligned}$$

where $G_t = \sum_{k=t}^{T-1} r_k$ is the lifetime return given the history τ_t, and we assume the discount factor $\gamma = 1$ for brevity. Thus, the derivative of the overall objective is:

$$\nabla_\eta J(\eta) = \mathbb{E}_{\theta_0 \sim \Theta, \mathcal{T} \sim p(\mathcal{T})}\left[\mathbb{E}_{\tau_t \sim p(\tau_t|\eta, \theta_0)}\left[G_t \nabla_{\theta_t} \log \pi_{\theta_t}(a_t|s_t) \nabla_\eta \theta_t\right]\right]. \tag{9}$$

B. Experimental Details

B.1. Implementation Details

We used mini-batch update to reduce the variance of meta-gradient estimation. Specifically, we ran 64 lifetimes in parallel, each with a randomly sample task and randomly initialised policy parameters. We took the average of the meta-gradients from each lifetime to compute the update to the intrinsic reward parameters (η). We ran 2×10^5 updates to η at training time. All hidden layers in the neural networks used ReLU as the activation function. We used arctan activation on the output of the intrinsic reward. The hyperparameters used for each domain are described in Table 1.

Table 1. Hyperparameters.

Hyperparameters	Empty Rooms	Random ABC	Key-Box	Non-stationary ABC
Time limit per episode	100	10	100	10
Number of episodes per lifetime	200	50	5000	1000
Trajectory length	8	4	16	4
Entropy regularisation	0.01	0.01	0.01	0.05
Policy architecture	Conv(filters=16, kernel=3, strides=1)-FC(64)			
Policy optimiser	SGD	SGD	Adam	SGD
Policy learning rate (α)	0.1	0.1	0.001	0.1
Reward architecture	Conv(filters=16, kernel=3, strides=1)-FC(64)-LSTM(64)			
Reward optimiser	Adam			
Reward learning rate (α')	0.001			
Lifetime VF architecture	Conv(filters=16, kernel=3, strides=1)-FC(64)-LSTM(64)			
Lifetime VF optimiser	Adam			
Lifetime VF learning rate (α')	0.001			
Outer unroll length (N)	5			
Inner discount factor ($\bar{\gamma}$)	0.9			
Outer discounter factor (γ)	0.99			

B.2. Domains

We will consider four task distributions, instantiated within one of the three main gridworld domains shown in Figure 2. In all cases the agent has four actions available, corresponding to moving up, down, left and right. However the topology of the gridworld and the reward structure may vary.

B.2.1. Empty Rooms

Figure 2(a) shows the layout of the *Empty Rooms* domain. There are four rooms in this domain. The agent always starts at the centre of the top-left room. One and only one cell is rewarding, which is called the goal. The goal is invisible. The goal location is sampled uniformly from all cells at the beginning of each lifetime. An episode terminates when the agent reaches the goal location or a time limit of 100 steps is reached. Each lifetime consists of 200 episodes. The agent needs to explore all rooms to find the goal and then goes to the goal afterwards.

B.2.2. ABC World

Figure 2(b) shows the layout of the *ABC World* domain. There is a single 5 by 5 room, with three objects (denoted by A, B, C). All object provides reward upon reaching them. An episode terminates when the agent reaches an object or a time limit of 10 steps is reached. We consider two different versions of this environment:*Random ABC* and *Non-stationary ABC*. In the Random ABC environment, each lifetime has 50 episodes. The reward associated with each object is randomly sampled for each lifetime and is held fixed within a lifetime. Thus, the environment is stationary from an agent's perspective but non-stationary from the reward function's perspective. Specifically, the rewards for A, B, and C are uniformly sampled from $[-1, 1]$, $[-0.5, 0]$, and $[0.0.5]$ respectively. The optimal behaviour is to explore A and C at the beginning of a lifetime to assess which is the better, and then commits to the better one for all subsequent episode. In the non-stationary ABC environment, each lifetime has 1000 episodes. The rewards for A, B, and C are 1, -0.5, and -1 respectively. The rewards for A and C swap every 250 episodes.

B.2.3. Key Box World

Figure 2(c) shows the *Key Box World* domain. In this domain, there is a key and three boxes, A, B, and C. In order to open any box, the agent must pick up the key first. The key has a neutral reward of 0. The rewards for A, B, and C are uniformly sampled from $[-1, 1]$, $[-0.5, 0]$, and $[0, 0.5]$ respectively for each lifetime. An episode terminates when the agent opens a box or a time limit of 100 steps is reached. Each lifetime consists of 5000 episodes.

B.3. Hand-designed near-optimal exploration strategy for Random ABC

We hand-designed a heuristic strategy for the Random ABC domain. We assume the agent has the prior knowledge that B is always bad and A and C have uncertain rewards. Therefore, the heuristic is to go to A in the first episode, go to C in the second episode, and then go to the better one in the remaining episodes in the lifetime. We view this heuristic as an upper-bound because it always finds the best object and can arbitrarily control the agent's behaviour.

C. Pseudocode

We provide an illustrative implementation of two core functions based on JAX (Bradbury et al., 2018). The provided code simulates the interaction between a single agent and an intrinsic reward function. However, in practice, one can use `jax.pmap` and `jax.vmap` to simulate parallel lifetimes with a shared intrinsic reward function.

C.1. Agent Update

```
"""
Copyright 2020 DeepMind Technologies Limited.


Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

https://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
"""

import jax
from jax import numpy as jnp
import tree  # https://github.com/deepmind/tree


def inner_update(learner_state, eta, rollout):
  """Inner update function.

  Args:
    learner_state: A namedtuple containing theta, opt_state, and core_state.
      theta is the parameters for the policy and the value function;
      opt_state is the optimizer state of the Adam optimizer;
      core_state is the hidden state for the RNN cores
        of the intrinsic reward function and the lifetime value function.
    eta: The parameters for the intrinsic reward function and the lifetime value function.
    rollout: A trajectory with length T + 1, where T is the inner unroll length.

  Returns:
    A new learner_state after one inner-loop update.
  """
  theta = learner_state.theta
  opt_state = learner_state.opt_state
  core_state = learner_state.core_state

  # unroll_eta_tp1 unrolls the the intrinsic reward function and
  # the lifetime value function on the rollout. Here tp1 is short for
  # ``t plus 1'' because the length of the rollout is T + 1.
  # The implementation of unroll_eta_tp1 is omitted here.
  (r_in, unused_v_lifetime), new_core_state = unroll_eta_tp1(eta, core_state, rollout)

  # discounted_return_fn is a util function that computes the discounted return in RL.
  # The implementation is omitted as it is not the core of our algorithm.
  # A reference implementation can be found at
  # https://github.com/deepmind/rlax/blob/master/rlax/_src/multistep.py
  returns = discounted_return_fn(
    rewards=r_in[1:],
    discounts=rollout.episode_discount[1:]*episode_discount,
    bootstrap_value=rollout.v[-1])
  advantages = returns - rollout.v[:-1]

  def inner_loss(theta, rollout, returns, advantages):
    """Inner loss function (surrogate).

    Args:
      theta: The parameters of the policy and the value function.
      rollout: A trajectory with length T + 1, where T is the inner unroll length.
      returns: A tensor with shape [T].
        The return for the last state-action in rollout is dropped.
      advantages: A tensor with shape [T].
        The advantage for the last state-action in rollout is dropped.

    Returns:
      A single scalar, the standard actor-critic loss.
    """
    logits, v = theta_apply(theta, rollout.observation[:-1])
    masks = rollout.lifetime_discount[:-1]
    # policy_gradient_loss_fn and entropy_loss_fn are util functions that
    # compute the policy gradient loss and entropy regularisation loss.
    # The implementations are omitted as they are not the core of our algorithm.
    # A reference implementation can be found at
    # https://github.com/deepmind/rlax/blob/master/rlax/_src/policy_gradients.py
    pg_loss = policy_gradient_loss_fn(
      logits=logits,
      actions=rollout.action[:-1],
      advantages=advantages,
      weights=masks,
      backprop=True)
    entropy_loss = entropy_loss_fn(logits=logits, weights=masks)
    baseline_loss = 0.5 * jnp.sum(jnp.square(v - returns) * masks)
    return pg_loss + baseline_cost * baseline_loss + inner_entropy_cost * entropy_loss

  # jax.grad is a JAX primitive which derives the gradient of a function.
  dl_dtheta = jax.grad(inner_loss)(theta, rollout, returns, advantages)

  # inner_opt_update updates the optimiser statistics.
  update, new_opt_state = inner_opt_update(dl_dtheta, opt_state)
  new_theta = tree.map_structure(lambda p, u: p + u, theta, update)

  new_learner_state = LearnerState(
    theta=new_theta,
    opt_state=new_opt_state,
    core_state=new_core_state)
  return new_learner_state
```

C.2. Intrinsic Reward and Lifetime Value Function Update

```
"""
Copyright 2020 DeepMind Technologies Limited.


Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

https://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
See the License for the specific language governing permissions and
limitations under the License.
"""

import jax
from jax import numpy as jnp
import tree  # https://github.com/deepmind/tree


def outer_update(learner_state, meta_learner_state, rollouts):
  """Outer update function.

  Args:
    learner_state: A namedtuple containing theta, opt_state, and core_state.
      theta is the parameters for the policy and the value function;
      opt_state is the optimizer state of the Adam optimizer;
      core_state is the hidden state for the RNN cores
        of the intrinsic reward function and the lifetime value function.
    meta_learner_state: A namedtuple containing eta and opt_state.
      eta is the parameters for the intrinsic reward and the lifetime value function;
      opt_state is the optimizer state of the Adam optimizer;
    rollouts: A sequence of N + 1 trajectory, each with length T + 1,
      where N is the outer unroll length and T is the inner unroll length.

  Returns:
    A new meta_learner_state after one outer-loop update.
  """

  def outer_loss(eta, learner_state, rollouts):
    """Outer loss function.

    Args:
      eta: The parameters for the intrinsic reward function and the lifetime value
                                                           function.
      learner_state: A namedtuple containing theta, opt_state, and core_state.
        theta is the parameters for the policy and the value function;
        opt_state is the optimizer state of the Adam optimizer;
        core_state is the hidden state for the RNN cores
        of the intrinsic reward function and the lifetime value function.
      rollouts: A sequence of N + 1 trajectory, each with length T + 1,
        where N is the outer unroll length and T is the inner unroll length.

    Returns:
      A single scalar, the sum of the losses for the intrinsic reward function
      and the lifetime value function.
    """
    # Unroll the inner-loop updates.
    all_r_ex = []
    all_lifetime_discount = []
    all_masks = []
    all_action = []
    all_logits = []
    all_v_lifetime = []
    core_state = learner_state.core_state
    for rollout in rollouts:
      all_r_ex.append(rollout.reward[1:])
      all_lifetime_discount.append(rollout.lifetime_discount[1:])
      all_masks.append(rollout.lifetime_discount[:-1])
      all_action.append(rollout.action[:-1])

      logits, unused_v = theta_apply(learner_state.theta, rollout.observation[:-1])
      all_logits.append(logits)

      # unroll_eta_tp1 unrolls the the intrinsic reward function and
      # the lifetime value function on the rollout. Here tp1 is short for
      # ''t plus 1'' because the length of the rollout is T + 1.
      # The implementation of unroll_eta_tp1 is omitted here.
      (unused_r_in, v_lifetime), core_state = unroll_eta_tp1(eta, core_state, rollout)
      all_v_lifetime.append(v_lifetime[:-1])
      bootstrap_value_v_lifetime = v_lifetime[-1]

      learner_state = inner_update(learner_state, eta, rollout)

    # Compute lifetime return and advantage.
    #
    # discounted_return_fn is a util function that computes the discounted return in RL.
    # The implementation is omitted as it is not the core of our algorithm.
    # A reference implementation can be found at
    # https://github.com/deepmind/rlax/blob/master/rlax/_src/multistep.py
    all_r_ex = jnp.concatenate(all_r_ex)
    all_lifetime_discount = jnp.concatenate(all_lifetime_discount)
    lifetime_return = discounted_return_fn(
      rewards=all_r_ex,
      discounts=all_lifetime_discount*lifetime_discount,
      bootstrap_value=bootstrap_value_v_lifetime)
    all_v_lifetime = jnp.concatenate(all_v_lifetime)
    lifetime_advantage = lifetime_return - all_v_lifetime

    # Outer-loop policy gradient loss.
    # Note that the updated policy is evaluated by the *next* rollout.
    #
    # policy_gradient_loss_fn and entropy_loss_fn are util functions that
    # compute the policy gradient loss and entropy regularisation loss.
    # The implementations are omitted as they are not the core of our algorithm.
    # A reference implementation can be found at
    # https://github.com/deepmind/rlax/blob/master/rlax/_src/policy_gradients.py
    valid_logits = jnp.concatenate(all_logits[1:])
    valid_action = jnp.concatenate(all_action[1:])
    pg_mask = jnp.concatenate(all_masks[1:])
    valid_lifetime_advantage = lifetime_advantage[-len(valid_logits):]
    pg_loss = policy_gradient_loss_fn(
      logits=valid_logits,
      actions=valid_action,
      advantages=valid_lifetime_advantage,
      weights=pg_mask,
      backprop=False)
    entropy_loss = entropy_loss_fn(logits=valid_logits, weights=pg_mask)

    # Lifetime value function regression loss.
    # Note that we do NOT update the lifetime value function on the last rollout,
    # which is for computing the meta-gradient only.
    valid_v_lifetime = all_v_lifetime[-len(valid_logits):]
    valid_lifetime_return = lifetime_return[:len(valid_logits)]
    baseline_mask = jnp.concatenate(all_masks[:-1])
    baseline_loss = 0.5 * jnp.sum(
      jnp.square(valid_v_lifetime - valid_lifetime_return) * baseline_mask)

    return pg_loss + baseline_cost * baseline_loss + outer_entropy_cost * entropy_loss

  # jax.grad is a JAX primitive which derives the gradient of a function.
  eta = meta_learner_state.eta
  dm_deta = jax.grad(outer_loss)(eta, learner_state, rollouts)

  # outer_opt_update updates the optimiser statistics.
  opt_state = meta_learner_state.opt_state
  update, new_opt_state = outer_opt_update(sum_dm_deta, opt_state)
  new_eta = tree.map_structure(lambda p, u: p + u, eta, update)

  new_meta_learner_state = MetaLearnerState(
      eta=new_eta,
        opt_state=new_opt_state,
    )
    return new_meta_learner_state
```