# Efficient Reinforcement Learning through Improved Cognitive Capabilities

by

**Nicolas Bougie**

Dissertation

Submitted to the Department of Informatics
in partial fulfillment of the requirements for the degree of

**Doctor of Philosophy**

SOKENDAI

The Graduate University for Advanced Studies, SOKENDAI
March 2021

## Acknowledgements

I am very grateful to all the people who contributed in one or the other way to this thesis. First and foremost, I would like to express my sincere gratitude to my advisor Professor Ryutaro Ichise. He has been an invaluable source of insight, ideas, and guidance; but also for his patience, motivation, and immense knowledge. His advice and mentorship have been invaluable. Thank you for being my advisor and teaching me.
I would like to thank my thesis committee - Mahito Sugiyama, Ken Satoh, Tetsunari Inamura, Seiji Yamada, and Koichi Moriyama, for taking the time to give me valuable feedback. I also want to thank all the members and internship students in Ichise Laboratory, especially Takuma Ebisu, Maxime Clement, Pierre-Antoine Chemin, Valerian Acier, Khai Nguyen, Li-Kai Cheng, Hien Van Tran, Natthawut Kertkeidkachornm, Juha Hovi, Aixi Li, and Xiangwei Wang, for many helpful discussions and providing a motivating working environment. I am thankful that I had the chance to work with them, and I am proud of the work we have done together over the last 3 years.
I would like to also thank my friends, I would not have been able to write this thesis without their support. Lastly, I would not be in this position without the unconditional love and support of my family. They have been my strongest supporters in every endeavor and have always supported me. Thank you very much!

## Abstract

A long-standing goal in reinforcement learning is to create agents that can solve complex decision-making tasks through trial and error interactions with the environment. While reinforcement learning has shown impressive advances in a plethora of simulated tasks including game-playing, board-games, or robot control, these results are shyly echoed in the real world. This is often due to huge amounts of interactions required to reach decent performance, which can be intractable in real-world settings. One underlying reason for the above-mentioned shortcomings is the poor cognitive capabilities of RL agents.

In this thesis, we start by tackling one of the key problems in reinforcement learning: how to overcome the lack of curiosity in reinforcement learning algorithms? One natural form of learning is to explore the environment and accumulate knowledge. However, in the real world, rewards are naturally sparse or poorly-defined, which is an important issue since hoping to stumble into a goal state by chance (in the absence of curiosity) within an acceptable number of interactions is unlikely. Inspired by curious behaviors in animals, we develop curiosity-driven agents that can learn to solve complex problems featuring extremely sparse rewards. Precisely, our aim is at modeling and reproducing the development of a natural cognitive process in artificial agents: curiosity. In this dissertation, we focus on self-exploration algorithms that can deal with: 1) the problem of gradual skill acquisition based on the agent's knowledge of the environment in the presence of high-dimensional observations and complex dynamics, 2) stochastic environments to escape from local optima, and the "vanishing curiosity" issues of prior work that use the absolute prediction error to guide exploration - curiosity rewards soon exhaust as the prediction becomes perfect or does not improve, prematurely converging to sub-optimal policies, and 3) global exploration to encourage long-term exploration behaviors (i.e. coordinated decisions over long time horizons).

To solve challenging tasks, an agent may not solely rely on internal guidance but also leverage human knowledge of how to solve these tasks. Precisely, unlike humans that rely on their common sense priors, traditional reinforcement learning agents do not have such an ability - they learn a task from scratch. In this dissertation, we address one of the key problems in reinforcement learning: how can agents efficiently leverage human guidance to drive the learning process? Thus, we propose generic mechanisms that employ human guidance to transfer human knowledge into reinforcement learning. Our core idea is to develop novel types of guidance that require minimal human effort and provide additional meaningful information to the agent. First, we introduce guidance via the concept of *high-level information* and human-like planning, based on simple domain knowledge and visual recognition. Second, we propose an interpretable model whose internal representation is *symbols* and *rules* automatically extracted from existing datasets (domain knowledge), drastically reducing the amount of necessary human effort. Finally, in order to ensure that the learner's goal matches the domain knowledge provided, we propose *active demonstrations*; an approach operating in the low data regime that actively shares insights between the agent and the teacher. Additionally, our agent learns a wide range of skills from the same set of *goal-driven demonstrations*.

Developmental studies show that interactive capabilities in humans emerge incrementally through the aggregation of multiple internal and external feedback signals. Hence, one natural question that arises is: how to simultaneously learn from these two classes of supervision (i.e. curiosity and human guidance) to achieve human-like sample-efficient learning? To answer this question, we propose a hierarchical framework that exploits the hierarchical structure of the task to integrate different modes of supervision. Our key design principle is to introduce (non-expert) human guidance at the high-level for long-term planning and common sense reasoning, while curiosity is employed at the low-level to drive the learning of sub-tasks. These hybrid agents can be used in domains where humans struggle to provide demonstrations while reducing the amount of required feedback and interactions with the environment by several orders of magnitude. Moreover, it produces agents that exceeded the expert performance in various domains.

# Table of contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

Over the course of the last decade, AI systems have transitioned from science fiction to science fact. The framework of reinforcement learning has begun to make substantial progress on many complex sequential decision-making problems. A lot of effort has been directed towards game-playing [1, 2, 3], robot control [4, 5], or even autonomous vehicles [6, 7]. Despite all the recent breakthroughs and advances for learning a rich set of behaviors in simulated tasks, reinforcement learning agents are not yet in wide-spread use in the real world. In fact, these advances have been made possible largely thanks to the ability of reinforcement learning to learn from huge amounts of data through trial and error interactions with the environment. Some advances also lie the design of task-specific reward functions, which is a notoriously challenging engineering problem - shaping a good reward function is not suitable for real-world tasks that involve versatility, different objectives, or very large state spaces.

All these considerations lead to one fundamental question: how to improve sample efficiency in reinforcement learning agents enough that they can be practically applied to real-world tasks? We believe that developing methods which integrate human-like *cognitive capabilities* is the key to this answer. Cognitive capabilities are aspects of mental functioning, which include the capability of planning, memorizing, thinking abstractly, learning from experience, common sense reasoning [8], and making intrinsically motivated choices [9, 10, 11]. These capabilities are one of the skill sets that distinguishes efficient learning in humans from data-consuming learning in artificial agents. Thus, this thesis outlines this ideology in detail and concretizes these ideas into algorithms that are embodied with cognitive capabilities.

In particular, reinforcement learning algorithms may require prohibitive amounts of data when the extrinsic reward function is sparse - the agent receives no incentive for most of the time steps. This drawback will become pressing when attempting to scale reinforcement learning to practical tasks where the observations are often high-dimensional sensor inputs such as images and rewards naturally extremely sparse. Furthermore, while learning from scratch (also called end-to-end learning) is one of the most popular paradigms in the field of machine learning, this will require the agent to directly solve the overall task without relying on any specific domain knowledge. Hence, putting reinforcement learning agents into complex diverse real-world environments to do a variety of different tasks remains very challenging and currently often not feasible.

On the other hand, humans heavily rely on their sense of curiosity, common sense priors (domain knowledge), and external feedback to rapidly learn new tasks - their cognitive capabilities allow them to master new skills in a few attempts. For example, suppose a child walking in a park. The child is surrounded by a small lake, pine trees, and a hill. Considering the number of possible activities and continuous changes in the world, in order to gradually acquire a new skill this child will have to make well-chosen decisions that fulfill a particular objective. For instance, to climb the large pine tree, this child will use his sense of curiosity to try new behaviors, but, he will also extract initial biases and strategies based on his common sense priors. Common sense priors in humans are extracted from previously learned tasks, and/or external guidance (e.g. parents advice, imitation of other children). In addition, the human brain will naturally reward him once the goal being pursued is achieved [12] - after climbing the tree. Somehow, mapping this continual stream of observations and internal/external guidance is natural for humans but remains an open challenge in reinforcement learning. In the same situation, an end-to-end reinforcement learning agent would only make use of the goal reward - that is, when it reaches the top of the tree. Therefore, it is likely that a reinforcement learning agent will take a huge amount of trials before mastering this skill.

Motivated to expand the availability of reinforcement learning to real-world domains, we present our contributions towards endowing reinforcement learning agents with a human-like ability to achieve sample-efficient learning. We consider that such agents must be endowed with a number of "building blocks", aimed at fulfilling vital cognitive capabilities. Precisely, we propose solutions for giving RL agents cognitive capabilities to address the issues discussed above:

1. Self-supervision via curiosity-driven exploration to deal with sparse reward environments,

2. Methods for facilitating various forms of common sense reasoning via novel types of external guidance and domain knowledge,

3. Techniques for combining intrinsic motivation and human guidance.

First, we design and study curiosity-driven approaches, which aim to speed up the learning time and improve the performance of reinforcement learning agents. As mentioned above, in many real-world scenarios, learning becomes impractical or even impossible when rewards extrinsic to the agent are very sparse or missing altogether. While humans are accustomed to operating with rewards that are extremely sparse, reinforcement learning agents lack an essential cognitive capability: curiosity [11]. Curiosity refers to the capability to make intrinsically motivated choices. In detail, humans constantly seek information and knowledge about their environment, independently of an extrinsic profit [13]. Taking inspiration from these observations, multiple work have been devoted to reproducing the computations underlying such a capability. In the context of reinforcement learning, motivation/curiosity has been used to explain the need to explore complex environments and discover novel states [14, 15]. In particular, making intrinsically motivated choices was shown to allow a learner to autonomously and actively acquire complex skills and order its own learning experiences. However, in order to be useful for real-world tasks, several major limitations inherent in curiosity must be addressed:

- Most prior work in the curiosity literature maintains a model of environmental dynamics and encourages visits of states with large prediction errors. Nevertheless, this does not take into account the agent's knowledge of the environment - rather than visiting all the states, curiosity should encourage exploration of task-relevant regions of the state space (task-relevant exploration). Besides, the model may drift over time, which limits the long-term performance of such techniques. (Chapter 3)

- Another serious limitation of prior approaches is that curiosity was shown to vanish quickly with further visitations. That is, the intrinsic reward goes to zero as soon as the model becomes sufficiently accurate or the agent does not improve. In the absence of curiosity reaching the final goal becomes unlikely, prematurely converging to sub-optimal policies. A further challenge is the handling of stochastic environments. If the transitions or observations in the environment are random, then even with a perfect dynamic model, an agent trying to maximize a prediction error will tend to continuously seek out such local sources of entropy in the environment, getting trapped in local optima. (Chapter 4)

- While current methods focus on improving local exploration - exploring the consequences of short-term decisions, global exploration that involves coordinated decisions over long time horizons is beyond their reach. Namely, they do not explicitly encourage the agent to discover deep exploration strategies, failing in long-term tasks. (Chapter 5)

Second, we focus on giving agents the capability of leveraging external guidance in order to enable common sense reasoning. In the real world, reinforcement learning requires large amounts of data due to reward sparsity, but also because most systems learn tasks from scratch, i.e., without any prior knowledge. While this is convenient in simulated domains where interactions are virtually unlimited, this assumption rarely holds in the real world - training an end-to-end reinforcement learning system with no prior assumptions about the domain often induces millions or billions of interactions to reach reasonable performance. To overcome this key challenge, one solution is to integrate human guidance to facilitate various forms of common sense reasoning [8, 16] - the cognitive capability to make presumptions about the type and essence of ordinary situations based on prior experiences and domain knowledge (also called common sense priors) [17], which cannot be achieved using end-to-end reinforcement learning.

However, the problem of endowing AI systems with common sense reasoning remains a major challenge in the quest for efficient learning. In the context of reinforcement learning, human knowledge how to approach a new task is often incorporated via imitation learning, where the agent learns to imitate human demonstrated decisions. Hence, the learner develops prior assumptions about the environment and initial biases, which can be used to face a new situation that is similar or analogous to one in the demonstration data. However, imitation learning raises several serious problems: to provide meaningful demonstrations the human demonstrator has to have some good familiarity with the task, it significantly puts more burden on humans, and these approaches are not directly applicable to behaviors that are difficult for humans to demonstrate. Therefore, in this dissertation, we focus on novel forms of guidance that reduce the cost of human effort while significantly improving efficiency - by exploiting domain knowledge and the tight coupling between reasoning and learning we seek to greatly improve sample-efficiency. In this thesis, we focus on improving three aspects of human guidance:

- First, it is not clear how to represent knowledge in a way that they are easy to interpret for the agent and can generalize to many situations. That is, while demonstrations carry state-action trajectory information (i.e. low-level knowledge), it remains uncertain how to represent high-level information about the environment in a meaningful way; and how to use the information for high-level control such as planning. Moreover, since human guidance is subject to errors, it motivates one question: how to correct possible human mistakes to exceed expert-level performance? (Chapter 6)

- Second, prior work expressly create domain knowledge or demonstrations to solve the task being learned. On the other hand, many datasets exist but they cannot be used in the context of reinforcement learning - agents do not have the capability to extract task-relevant information from a dataset. This observation motivates the need for leveraging existing datasets, in order to substantially reduce human workload and learn from large amounts of prior knowledge. (Chapter 7)

- Finally, most approaches focus on a passive access to domain knowledge - the agent and the teacher cannot communicate or share information during the training process. Thus, they cannot cope with the changes in the environment (also called distribution mismatch) or may suffer from low state coverage. Moreover, we argue that human guidance should be primarily focused on hard-to-learn regions. (Chapter 8)

Finally, one natural question that arises is: how to simultaneously learn from these two forms of supervision to achieve human-like sample-efficient learning? As explained above, curiosity is an effective form of intrinsic supervision for control and execution (low-level tasks), particularly when rewards are sparse. On the other hand, human guidance is effective to enable optimal planning or reasoning, but can increase human effort and knowledge is often incomplete/imperfect. Thus, in Chapter 9, we aimed to achieve human-like sample-efficient learning by incorporating different combinations of supervision at different levels of our agent.

## 1.2 Summary of Contributions

The ability to achieve sample-efficient learning in complex and changing scenarios is essential for reinforcement learning agents to function in real-world environments. As discussed above, it is critical to improve the cognitive capabilities of agents in order to reflect recent successes in simulated tasks to the real world. To this aim, we considered three open challenges, namely: (1) learning from sparse rewards via curiosity-driven exploration (self-supervision), (2) bridging the gap between reinforcement learning and human guidance, and (3) leveraging multiple sources of intrinsic and extrinsic supervision for sample-efficient learning. We briefly summarize our contributions in the following section.

### 1.2.1 Curiosity-Driven Exploration for Sample-Efficient Learning

In Part I, our key emphasis is on building agents that continuously learn skills just from their own collected experience by using data as its own supervision - curiosity. Curiosity-driven exploration is a flexible class of self-supervision inspired by curious behaviors in animals (visiting novel states should be rewarded). It can be used to encourage agents to learn about their environments even when extrinsic feedback is rarely provided (also called sparse rewards). For example, count-based exploration [18] keeps visit counts for states and favors the exploration of states rarely visited. Another class of methods relies on building a model of environmental dynamics of the environment [19]. For instance, ICM [14] predicts the feature representation of the next state based on the current state and the action taken by the agent. Nevertheless, maximizing the prediction error tends to attract the agent to stochastic transitions, where the consequences of actions are hardly predictable [20]. This issue has motivated several recent works [20, 21] where the problem is a deterministic function of its inputs.

This thesis addresses several issues and limitations inherent in curiosity, as summarized below:

- **Skill-Based Curiosity for Intrinsically Motivated Reinforcement Learning**: First, we propose a formulation of intrinsic curiosity that can deal with complex environmental dynamics, large observation spaces, and very sparse rewards. Many prior work such as the famous ICM algorithm [14] learn a model of environmental dynamics and encourage the agent to visit regions with large prediction errors. However, they do not directly consider the agent's knowledge (i.e. the policy being learned) and its ability to reach the goal being pursued. Moreover, it tends to limit long-horizon performance due to model drift. On the other hand, the proposed method (GoCu) generates an exploration bonus based on the agent's knowledge about its environment in order to encourage the gradual acquisition of new skills, focusing its attention on task-relevant parts of the state space to speed up learning and improve sample efficiency. To this end, we propose to automatically decompose the task into easier sub-tasks and to evaluate the agent's ability to master multiple skills at once. We further present and evaluate a technique to embed skills into a latent space, in order to improve generalization of skills. The depicted method scales to high-dimensional problems, avoids the need for maintaining a model of the world, and can perform in sequential decision scenarios.

- **Exploration via Progress-Driven Intrinsic Rewards**: In spite of curiosity-driven approaches' ability to deal with some hard exploration tasks, they tend to get stuck in local optima in tasks featuring stochastic dynamics or local sources of entropy. Furthermore, these algorithms face a fundamental limitation: the intrinsic reward may vanish quickly with additional visitations, converging prematurely to sub-optimal policies. To address these pitfalls, we introduce a robust definition of curiosity that considers the agent's learning progress in terms of "quantity" and "quality" on a multi-step horizon. We postulate that focusing on hard-to-learn regions of the state space is crucial to efficiently explore. To quantify the agent's learning progress, we propose to measure the divergence between a parametric model

Table 1.1: Comparison between pure reinforcement learning (RL), a curiosity-driven exploration baseline (ICM) and the proposed curiosity-driven agents.

| | Challenge | | | | |
|---|---|---|---|---|---|
| Method | Sparse Rewards | Task-relevant Exploration | Vanishing Curiosity | Stochasticity | Global Exploration |
| RL | X | X | X | X | X |
| ICM | ✓ | X | X | X | X |
| GoCu | ✓ | ✓ | X | X | X |
| PoBP, ReBP | ✓ | ✓ | ✓ | ✓ | X |
| FaSo, IML | ✓ | ✓ | ✓ | ✓ | ✓ |

(i.e. the current model) and prior models. We develop and evaluate two types of models based on: 1) the agent's understanding of the world (ReBP), 2) the agent's policy (PoBP). We further adopt a mechanism called *episodic-skills* to promote faster learning of key skills.

- **Fast and Slow Curiosity for High-Level Exploration**: While intrinsically motivated agents hold the promise of better local exploration, solving problems that require coordinated decisions over long-time horizons remains an open problem. To overcome this drawback, we found that high-level exploration - the combination of local and deep dominant exploration phases, is essential. To this end, we introduce the concept of fast and slow curiosity that aims to incentive high-level exploration by flexibility integrating two reward components (FaSo). Our method decomposes the curiosity bonus into a fast reward that deals with local exploration and a slow reward that encourages deep exploration. We then derive another algorithm for the better incorporation of action-dependent information into the existing intrinsic calculation and a different strategy (IML) to flexibly combine intrinsic rewards, promoting lifelong learning. By doing so, the proposed method reduces the amount of interactions with the environment by several orders of magnitude while enabling deep exploration even in the presence of sparse rewards and temporally-extended exploration patterns.

Overall, we develop the means for integrating *curiosity* into reinforcement learning. Through evaluations on benchmark tasks, we show that curiosity is vital to rapidly acquire new skills and improve exploration efficiency in extremely sparse reward environments. Besides, even in the absence of explicit extrinsic reward, curiosity provides enough indirect supervision for learning interesting behaviors and skills. Furthermore, we demonstrate that introducing this cognitive capability enables our agents to master tasks featuring real-world characteristics (e.g. stochasticity, poorly-defined rewards, temporally-extended exploration patterns) that traditional methods fail to solve (Table 1.1).

### 1.2.2 Human Guidance for Sample-Efficient Learning

In Part I, we focus on guiding an end-to-end agent learning via its own supervision (i.e. curiosity). Part II takes inspiration from development in humans, they rarely attempt to learn from scratch but instead also heavily rely on their common sense priors. They extract common sense priors from guidance of other humans, and/or previously learned tasks. Such guidance is often available in real-world domains and is essential to scale reinforcement learning to practical tasks.

In the context of reinforcement learning, the most common form of external guidance is the human policy itself [22] (also called imitation learning). A human demonstrator communicates the policy by performing the task in person and providing the correct actions to the agent. Most imitation learning algorithms assume this type of guidance [23, 24]. Another form of imitation learning, inverse reinforcement learning, extracts a reward function from the demonstration data [25, 26, 27]. However, these approaches are not directly applicable to behaviors that are difficult for humans to demonstrate such as robot control or in long-time horizon tasks [28]. Moreover, this type of domain feedback can significantly increase human workload. These considerations motivate the need for designing RL systems with the capability of integrating different forms of guidance more suitable for real-world scenarios. In this dissertation, we propose initial directions towards novel types of guidance, which are less expensive than policy demonstrations and more intuitive for humans. Overall, we aim to reduce the cost of human effort far enough that it can be practically applied to real-world systems.

This thesis introduces novels types of guidance and domain knowledge as summarized below:

- **Combining Deep Reinforcement Learning with Prior Knowledge and Reasoning**: This chapter focuses on introducing human-like reasoning and domain knowledge to supervise the agent's learning. As an intuition, we enhance information given to the agent by providing human expertise. Our method (DRL-EK) augments the input of a reinforcement learning model whose input is raw pixels by adding high-level information created from simple knowledge about the task and recognized objects within frames. This type of guidance is less expensive than policy demonstrations and easy to understand for the agent, while providing additional information that can be generalized to many situations. We then extend this framework to integrate human-based reasoning and planning to improve the overall exploration strategy followed by agent. Since domain knowledge is often incomplete, we simultaneously train the reinforcement learning model to correct possible human errors. Further benefits stem from combining multiple sources of decisions (prior vs learned) to improve the robustness in the action selection.
- **Towards interpretable reinforcement learning with state abstraction driven by external knowledge**: In the above Chapter, we found that it is hard to generalize domain knowledge to new tasks. Another general issue is that most imitation-based reinforcement learning agents lack interpretability - while deep neural networks have been shown to be very effective, the structure of these models makes them difficult to be interpreted, which restricts their use to non-safety critical domains. We propose a learning framework (Sarsa-rb($\lambda$)) that addresses all of these issues at once by combining a novel reinforcement learning architecture with large amounts of domain knowledge to significantly accelerate learning, without the need for demonstrations or specific human engineering. Namely, we propose to extract symbols and first-order logic rules, and then use them to represent the internal states of our reinforcement learning model, making decisions fully interpretable and reducing the state space to be explored. Critically, these rules can be extracted from

Table 1.2: Comparison between pure imitation learning (DQfD) and the proposed agents.

| | | Challenge | | | | |
|---|---|---|---|---|---|---|
| Method | Guidance | Robust to Noise | Intrepretable | Existing Datasets | Cooperation | Target Goal(s) |
| DQfD | Demonstrations | No | No | No | Passive | Single |
| DRL-EK | High-level Knowledge / Planning | Yes | No | No | Passive | Single |
| Sarsa-rb($\lambda$) | First-order rules | Yes | Yes | Yes | Passive | Single |
| goAL | Goal-driven demonstrations | Yes | No | No | Active | Multiple |

existing datasets related to the task being learned and during the training process. To do so, we present three novel methods to extract rules, which includes analyzing the learned representation of an autoencoder. Besides, the structure of the rules can be used to maximize the benefit of past experience to face new situations. This is the key idea of the *sub-states* mechanism that exploits similarities among rules.

- **Active Goal-Driven Learning**: Above chapters focus on a passive access to the demonstrator. That is, there is no active cooperation between the agent and the human demonstrator (or the provided domain knowledge). However, to be useful in changing real-world tasks it may be necessary to promote active cooperation: learning from a fixed set of demonstrations/domain knowledge may be impracticable due to lack of state coverage or distribution mismatch - when the learner's goal deviates from the demonstrated behaviors. Besides, we are interested in learning how to reach a wide range of goals from the same set of domain knowledge. We contribute an active goal-conditioned approach (goAL) that drastically reduces expert workload by incrementally requesting partial demonstrations (i.e. short demonstrations) towards specific goals (goal-driven demonstrations). Goal-driven demonstrations are actively queried based on the agent's confidence and the agent's ability to reach the goal being pursued. We found this type of guidance to be easier to demonstrate and more intuitive for a human than full demonstrations, while significantly increasing the value information of the queries by matching the agent's needs. We then contribute a relabeling strategy to artificially generate more expert data. Finally, the proposed framework further incentivizes the sampling of goals where the disagreement between the expert and the policy is maximized.

In this part, we build generic models embodied with the capability of leveraging novel types of human guidance. We show that the proposed approaches improve various forms of common sense reasoning, enhancing data efficiency and final performance (Table 1.2). They reduce the number of required interactions and human effort enough to scale RL to practical tasks such as robot control, trading, 3D navigation, or game-playing.

### 1.2.3 Self-Supervision and Human Guidance for Sample-Efficient Learning

Parts I and II discuss how an agent could integrate internal or external guidance to achieve sample-efficient learning in the real world. However, in practice, the agent will eventually have access to multiple types of supervision (e.g. curiosity, goal-driven demonstrations, preferences). Hence, it motivates the necessity for developing methods that can utilize multiple intermediate learning signals.

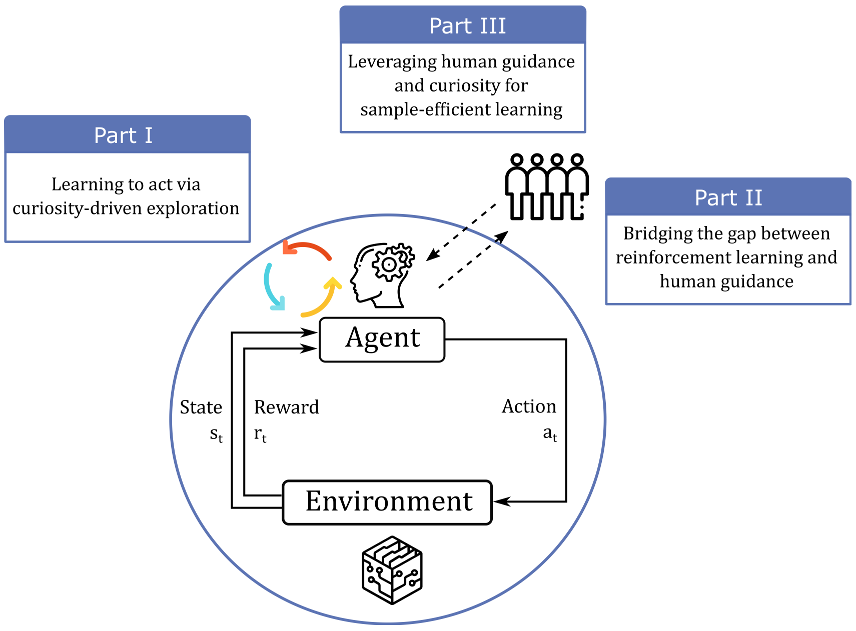

Part III
Leveraging human guidance and curiosity for sample-efficient learning
Part I
Learning to act via curiosity-driven exploration
Part II
Bridging the gap between reinforcement learning and human guidance
Agent
State $s_t$
Reward $r_t$
Action $a_t$
Environment

Figure 1.1: Outline of the thesis.

- **Hierarchical learning from human preferences and curiosity**: Inspired by learning in humans - they utilize all possible intermediate learning signals to solve challenging tasks, we design a novel hierarchical reinforcement learning method that introduces non-expert human preferences at the high-level, and curiosity to drastically speed up the convergence of subpolicies to reach any sub-goals (even in tasks with very sparse rewards). In this work, we only request human high-level feedback to the supervisor in states where the agent is unsure and struggles. Moreover, the proposed form of human guidance does not necessarily require the human trainer to be an expert at performing the task since it only requires the human to judge outcomes. We further formulate a strategy based on curiosity to automatically discover sub-goals, eliminating the need for manually designing sub-goals.

Overall, we aim to mimic the human brain's capability of aggregating a continual stream of internal and external guidance. This cognitive capability allows us to solve extremely challenging tasks in a few interactions. The method can be practically applied to a wide range of tasks, and shows substantial improvements over prior artificial agents. Moreover, we can exceed expert-level performance in several domains by actively sharing insights between the agent and a human.

## 1.3 Outline of the Thesis

The contributions of this thesis can be separated into three main parts (Figure 1.1). Each part introduces a class of approaches aimed at improving a different aspect of human-like sample-efficient learning.

- **Part 1** focuses on learning in the absence of dense or well-designed rewards. Inspired by early stages of development in animals and humans [29, 30], we build systems that drive the agent's learning via its own supervision (curiosity). First, we develop a method for learning in large state-space environments based on the agent's knowledge of the task (Chapter 3). Second, we propose a method for avoiding the well-known "vanishing curiosity" issues inherent in curiosity and handling stochastic dynamics (Chapter 4). Finally, we propose an algorithm for long-term exploration, to avoid local optima and capture the consequences of long-term decisions (Chapter 5).

- **Part 2** introduces techniques for learning in cooperation with humans via domain knowledge and human guidance. More precisely, we focus on novel forms of guidance that add a minimal human effort to improve both sample efficiency and the performance of our agent. 1) We introduce *high-level* domain knowledge and *human-based planning* to augment the agent's decision making (Chapter 6). 2) We represent the internal representation of our agent with *first-order rules* extracted from existing datasets related to the task being learned (Chapter 7). 3) The agent can actively query *goal-driven demonstrations* in hard-to-learn and uncertain regions of the state space (Chapter 8).

- **In Part 3** we then address the problem of learning from multiple types of guidance. That is, we propose a hierarchical framework that can incorporate different combinations of curiosity and human guidance at different levels, leading to dramatic reductions in both human effort and cost of exploration (Chapter 9).

In the final chapter of this thesis, we summarize our findings and discuss future avenues of research.

# Chapter 2

# Background

This chapter provides the necessary background to understand the research topics presented in the following chapters. This chapter begins with a discussion on the foundations of reinforcement learning (overview of reinforcement learning, Markov decision process, Bellman equation, and temporal-difference methods), followed by a brief introduction on different broad classes of approaches, including recent on-policy-algorithms, goal-conditioned reinforcement learning, and hierarchical reinforcement learning. Finally, we present an overview of the unsupervised image representation learning algorithms that we make use in this thesis.

## 2.1 Reinforcement Learning (RL)

Reinforcement learning [31] (RL) consists of an agent learning a policy $\pi$ by interacting with an environment (Figure 2.1). At each time-step the agent receives an observation $s_t$ and chooses an action $a_t$. The agent gets a feedback from the environment called reward, $r_t$. Rewards are given to the agent when it reaches certain desired states and are used for policy learning.

Given this reward and the current observation, the agent can update its policy to improve the future expected rewards. Assuming a discount factor $\gamma$, the future discounted rewards, called return $\mathcal{R}_t$, is defined as:

$$\mathcal{R}_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{2.1}$$

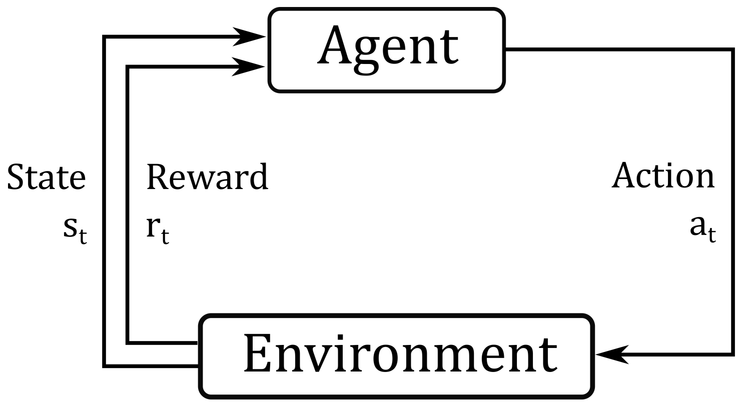


Figure 2.1: Structure of agent-environment interactions in reinforcement learning.

where $T$ is the time-step at which the epoch terminates. The agent learns to select the action with the maximum return $\mathcal{R}_t$ achievable for a given observation [32].

### 2.1.1 Markov Decision Process

A Markov Decision Process (MDP) [32] is defined as a 5-tuple $(S, A, P, r, \gamma)$, where $S$ is a set of states, $A$ is a set of possible actions, $P : S \times A \times S \to \mathbb{R}$ is a transition function, $r : S \times A \to \mathbb{R}$ is a reward function, and $\gamma \in [0, 1]$ is a discount factor. We aim to find a policy $\pi : S \to A$ that maximizes the excepted discounted rewards:

$$R_t = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r(s_t, a_t, s_{t+1})] \tag{2.2}$$

In other words, the agent selects the action in each specified state using its policy. For every state $s \in S$, $\pi(s, \cdot)$ forms a probability distribution:

$$\sum_{a \in A} \pi(s, a) = 1, \forall s \in S \tag{2.3}$$

The characteristic of an MDP is that the environment and the task follow the *Markov* property, which is defined as follows:

$$P(s_{t+1} = s, r_{t+1} = r | s_t, a_t, r_t, ..., r_1, s_0, a_0) = P(s_{t+1} = s, r_{t+1} = r | s_t, a_t), \forall s, s_t \in S, a_t \in A \tag{2.4}$$

where $P$ is the probability distribution. The above property guarantees that the probability distribution of future actions of the process depends only upon the present state, not on the sequence of events that preceded it.

### 2.1.2 Bellman Equation

The agent tries to get the most expected sum of rewards from every state it lands in. In order to achieve that, we must try to get the *optimal value function.* The Bellman equation defines this notion of *value* of a state.

The value function $V^{\pi}(s)$ which represents the expected return for a state $s$ following a policy $\pi$ is defined as follows:

$$V^{\pi}(s) = \max_{a \in A}(\sum_{s' \in S} P(s, a, s^{'})[R(s, a, s^{'}) + \gamma V^{\pi}(s^{'})]), \forall s \in S \tag{2.5}$$

The optimal value function $V^{*}(s)$ can then be calculated by using a policy evaluation technique:

$$V^{*}(s) = \max_{a \in A}(\sum_{s' \in S} P(s, a, s^{'})[R(s, a, s^{'}) + \gamma V^{*}(s^{'})]), \forall s \in S \tag{2.6}$$

We can then derive the optimal policy $\pi^{*}$ that selects actions that lead to states with the highest value:

$$\pi^*(s) = \arg\max_{a \in A}(\sum_{s' \in S} P(s,a,s')[R(s,a,s') + \gamma V^*(s')]), \forall s \in S \tag{2.7}$$

Rather than directly estimating the value function, it is common to estimate a state-action function. The state-action value function is called the Q-value function, $Q : S \times A \rightarrow \mathbb{R}$, which represents the expected sum of the discounted rewards for an agent starting at state $s$, selecting an action $a$, and then following a policy $\pi$ - whether a particular (s,a) tuple is valuable in the long run:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}\left[R_t \mid s_t = s, a_t = a\right] \\ &= \sum_{s' \in S} P(s,a,s')[R(s,a,s') + \gamma V^\pi(s')], \forall s \in S, a \in A \end{aligned} \tag{2.8}$$

The optimal policy $\pi^*$ is defined as selecting the action with the optimal Q-value, the highest expected return, followed by an optimal sequence of actions. This obeys the Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right] \tag{2.9}$$

and the optimal policy is defined as follows:

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a), \forall s \in S \tag{2.10}$$

### 2.1.3 Temporal-Difference Methods

Temporal Difference (TD) methods enable reinforcement learning in model-free settings, where no explicit knowledge of the environment model is needed. TD methods use policy iteration, which can be grouped into two broad classes: on-policy, and off-policy algorithms. They are widely used to learn an optimal policy.

A common off-policy technique to approximate $\pi \approx \pi^*$ is Q-learning [33]. The estimation of the action value function is performed iteratively by updating $Q(s,a)$. This algorithm is considered as an off-policy method since the update rule is unrelated to the policy that is learned, as follows:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma * \max_{a} Q(s_{t+1},a) - Q(s_t,a_t)\right] \tag{2.11}$$

where $0 < \alpha < 1$ is a learning rate that generally decreases over time. The choice of the action follows a policy derived from $Q$. Balancing exploration and exploitation can be done via an $\epsilon$-greedy strategy, which trade-off the exploration/exploitation dilemma [32] by decreasing the probability of exploration over time, $\epsilon$. During exploitation, the action with the highest estimated return is selected whereas a random action is sampled during exploration.

On the other hand, the on-policy algorithm, Sarsa [34], is a variant of the Q-learning algorithm. The key difference between Q-learning and Sarsa is that Sarsa is an on-policy method, which implies that the Q-values are learned based on the action performed by the current policy instead of a greedy policy. Sarsa is more conservative, meaning that Sarsa tends to avoid dangerous actions that may trigger negative rewards. This may be critical in real-world tasks where mistakes are costly such as in trading or autonomous driving. The update rule becomes:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.12}$$

Sarsa converges with probability 1 to an optimal policy as long as all the action-value states are visited an infinite number of times.

### 2.1.4 State-of-the-art On-Policy Algorithms

We now review two state-of-the-art on-policy algorithms which are used as learning methods in this thesis. Please note that the main difference with the temporal-difference approaches previously presented lies in the use of deep neural networks used as function approximators. When dealing with high-dimensional state spaces, table-based representations (e.g. Q-value function) may become infeasible and they do not generalize to unseen transition samples. This issue is tackled by representing value functions using function approximators, such as linear functions [35, 36] or artificial neural networks. In this dissertation, we use deep neural networks as function approximators. Therefore, we present more in detail two deep on-policy algorithms: A2C [37] and PPO [38].

As mentioned above, in contrast with off-policy methods such as DQN [1] that decouple the behavior and target policies, on-policy methods update their value functions based on the trajectories generated following the current policy. A2C is a synchronous variant of A3C [37] that takes advantage of parallel learning to efficiently learn. It consists in a critic that estimates the value function to criticize the actions made by the actor, and an actor that learns a policy $\pi(a|s, \theta)$ by minimizing the following loss function:

$$\mathcal{L}_{actor}(\theta) = -\mathbb{E}_{s,a\sim\pi}\left[\mathcal{R}_t - V_\pi(s) + \beta H(\pi(.|s, \theta))\right] \tag{2.13}$$

where $H(\pi(.|s, \theta))$ is the entropy to encourage exploration and avoid convergence towards a sub-optimal policy, $\beta$ controls the importance of exploration during training, and $\mathcal{R}$ is an estimation of the return. As mentioned above, the value function $V_\pi(s)$ represents the excepted return for a state $s$ following the policy $\pi$: $V_\pi(s) = \mathbb{E}_{a\sim\pi(a|s)}\left[\mathcal{R}_t | s_t = s\right]$

We now review Proximal Policy Optimization (PPO), a specific technique for optimizing policies. PPO introduces a penalty which controls the change of the policy at each iteration to reduce oscillating behaviors. The objective function becomes:

$$\mathcal{L}_{CLIP}(\theta) = \mathbb{E}_t\left[\min(\varrho_t(\theta)A_t, \mathrm{clip}(\varrho_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right] \tag{2.14}$$

where $\varrho_t$ is the probability ratio, $\varrho_t = \pi_\theta(a|s)/\pi_{\theta_{old}}(a|s)$, and $\epsilon$ is a hyperparameter. The ratio $\varrho_t$ is clipped to fall between $(1-\epsilon)$ and $(1+\epsilon)$. $A_t$ represents the advantage function $A_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ - the benefits of taking an action compared to the others, in a state.

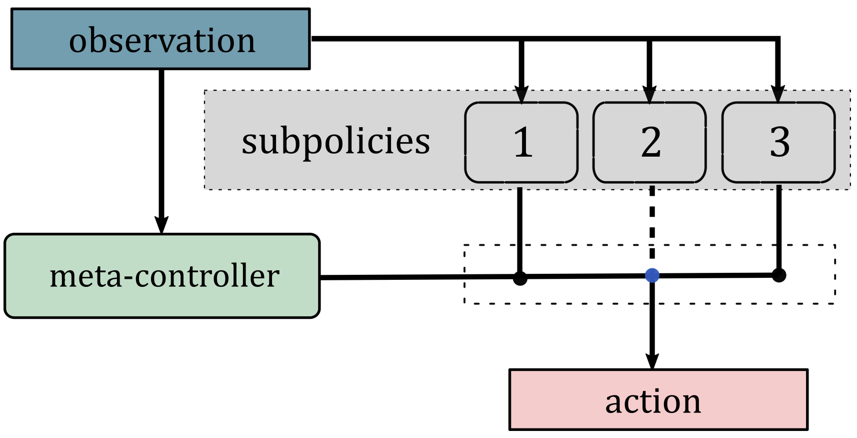


Figure 2.2: Hierarchical reinforcement learning.

### 2.1.5 Goal-Conditioned Reinforcement Learning

Here we present a brief overview of the theory of goal-conditioned reinforcement learning [39, 40]. Goal-conditioned reinforcement learning is an extension of RL to the setup where there are multiple goals we may try to reach.

Hence, rather than maintaining a policy trained to reach a single target goal (e.g. drive a car to a target position), it aims to obtain a policy that can accomplish a variety of tasks (e.g. drive a car to a target position, park the car, change of lane). To this end, we assume a set of goals $G$. The reward function and the policy are additionally conditioned on a goal, $g \in G$.

At every timestep $t$ the agent receives as input not only the current state $s_t$ but also the current goal $g$, $\pi : S \times G \rightarrow A$. The reward function becomes $r_t = r_g(s_t, a_t)$ where $r_g$ is often a binary function which represents whether the agent could reach the goal (i.e. $\mathbb{1}[s_{t+1} == g]$). The policy is trained to optimize the excepted return with respect to a goal distribution:

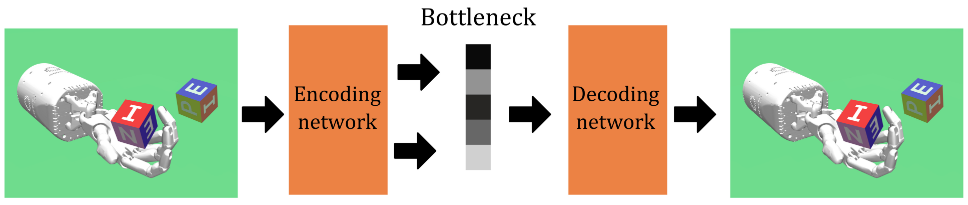
$$\mathbb{E}_{g \sim G}[\mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_0]] \tag{2.15}$$

where $G$ is the set of goals, and $E$ is a data-set of state-action-goal tuples. Goal-conditioned RL often employs a replay strategy called Hindsight Experience Replay (HER) [41] to artificially generate new transitions by relabeling goals seen along the state trajectory. Since the transition probability is not affected by the goal being pursued $g$, $g$ can be relabeled in hindsight. Thus, a transition $(s_t, a_t, s_{t+1}, g, r = 0)$ can be treated as $(s_t, a_t, s_{t+1}, g^{'} = s_{t+1}, r = 1)$.

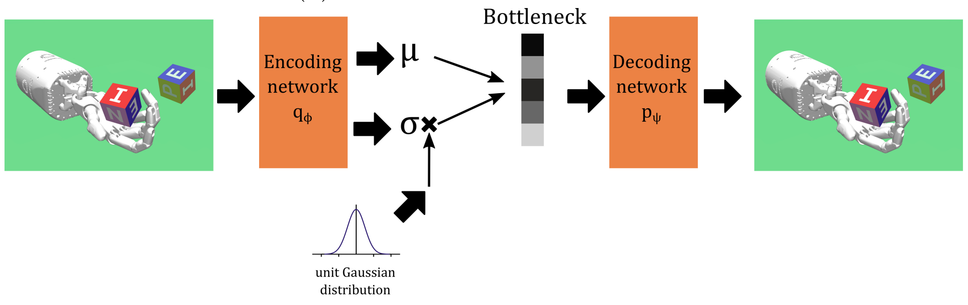
### 2.1.6 Hierarchical Reinforcement Learning

In this section we introduce notations of hierarchical reinforcement learning [42] that we use in Part III. The possibility of learning temporally extended actions (as opposed to primitive actions that last for only one time-step) has been introduced under the concept of *options* [43]. This idea plays a central role in hierarchical reinforcement learning.

In our dissertation, we consider a hierarchical RL agent with a two-level hierarchy (Figure



(a) Autoencoder architecture.


Bottleneck
Encoding network $q_\phi$
μ
σ
Decoding network $p_\psi$
unit Gaussian distribution

(b) Variational autoencoder architecture.

Figure 2.3: Architecture of an autoencoder (top) and a variational autoencoder (bottom).

2.2). The high-level policy (also called master policy or meta-controller) chooses a sub-goal that a low-level policy tries to reach. For instance, a sub-goal could be "reach the door" and the corresponding subpolicy "go left" $\rightarrow$ "jump" $\rightarrow$ "go down" $\rightarrow$ "use key". Each sub-goal corresponds to one subpolicy that can be executed by the low-level component. We denote a sub-goal as $g \in \mathcal{G}$ and a low-level action is denoted by $a \in \mathcal{A}$. The high-level policy iteratively selects a sub-goal $g$ which is then executed by the corresponding subpolicy until completion or failure. To choose this sub-goal, the high-level policy takes as input the current state $s \in \mathcal{S}$. This problem can be formalized as simultaneously learning the high-level policy $\tau : \mathcal{S} \rightarrow \mathcal{G}$ and a set of low-level policies $\pi_g : \mathcal{S} \rightarrow \mathcal{A}$. Low-level policies receive a positive reward when the sub-goal being pursued is achieved. On the other hand, the meta-controller receives an extrinsic reward provided by the environment, which indicates whether the agent is improving at solving the overall task.

## 2.2 Unsupervised Image Representation Learning

We now briefly summarize unsupervised image representation learning techniques that we build on in our methods. Unsupervised image representation learning is a popular research topic in machine learning. For instance, these approaches have been used to extract meaningful features from rich sensory inputs [44], denoise images [45], or for images compression [46]. In this section we present two common techniques: 1) autoencoders, and 2) variational autoencoders.

An autoencoder (AE) [47] consists in an encoder network learning how to compress the input data $x$ into an encoded representation $\phi(x)$, and then the original image is reconstructed $\hat{x}$ by a decoder network (Figure 2.3a). By using a low-dimensional middle layer (i.e. bottleneck layer), the model is forced to extract relevant features. The parameters of the neural network are trained to optimize a loss function $L_{AE}$ where the first term $\mathcal{L}$ penalizes the reconstruction error of the target given the input and the second term prevents overfitting, such as L1 regularization:

$$\mathcal{L}_{AE} = \mathcal{L}(x, \hat{x}) + \lambda \sum_i |a_i^{(h)}| \tag{2.16}$$

where $a_i^{(h)}$ represents the activation values in layer $h$ for observation $i$, and $\lambda$ is a scaling parameter.

In contrast, variational autoencoders (VAEs) [48] are generative models used to learn latent representation of high dimensional data such as images (Figure 2.3b). The input image is passed through an encoder network $q_\phi$ which outputs the parameters $\mu$ and $\sigma$ of a multivariate Gaussian distribution. A latent vector is sampled and the decoder network $p_\psi$ decodes it into the original state space. The parameters $\phi$ and $\psi$ of the encoder and decoder are jointly optimized to maximize:

$$\mathcal{L}(\psi, \phi; s^{(i)}) = \beta D_{KL}(q_\phi(z|s^{(i)})||p(z)) - \mathbb{E}_{q_\phi(z|s^{(i)})}\left[\log p_\psi(s^{(i)}|z)\right] \tag{2.17}$$

where the first term is a regularizer, the Kullback-Leibler divergence between the encoder distribution $q_\phi(z|s^{(i)})$ and $p(z)$. $p(z)$ is some prior specified as a standard normal distribution $p(z) = \mathcal{N}(0, 1)$. The second term is the expected negative log-likelihood - the reconstruction loss.

# Part I

# Learning to Act via Curiosity-Driven Exploration

# Chapter 3

# Skill-Based Curiosity for Intrinsically Motivated Reinforcement Learning

One of the most important problems in reinforcement learning currently is arguably to improve the agents' efficiency when rewards are sparse or poorly-defined, which is an issue for many real-world applications. Classic methods rely on getting feedback via extrinsic rewards to train the agent. However, when rewards are sparse - rewards are not frequently emitted, the agent learns very slowly or cannot learn at all.

Motivated to overcome this drawback, we present a novel curiosity-driven approach that (1) encourages the agent to gradually acquire new skills based on its knowledge of the task, (2) promotes the revisits of hard-to-learn skills, and (3) improves generalization of skills to unseen situations.

## 3.1 Introduction

Combining reinforcement learning (RL) with neural networks has led to a large number of breakthroughs in many domains. They work by maximizing extrinsic rewards provided by the environment. In well-specified reward function tasks, such as game-playing [49] or robotic control [50], RL is able to achieve human performance. In reality, many real-world domains involve reward functions that are sparse or poorly-defined.

Overcoming sparse reward scenarios is a major issue in RL since the agent can only reinforce its policy when the final goal is reached. In such scenarios, motivating the agent to explore the environment is necessary. Several works have attempted to facilitate exploration by combining the original reward with an additional reward. For instance, reward shaping [51] guides the policy optimization towards promising solutions with a supplementary reward, but, the necessity of engineering limits the applicability to specific domains. An alternative approach to address the reward design problem is to motivate the agent to explore novel states using its own curiosity - the ability to acquire new skills that might become useful in the future. Curiosity is motivated by early stages of development in humans: babies reward themselves for acquiring increasingly difficult new skills. Many formulations aim to measure the novelty of the states in order to encourage exploration of novel states. For instance, an approach [52] bases the exploration of the agent on the surprise - the ability of the agent to predict the future. Similarly, ICM [14] formulates curiosity as the error in an agent's ability to predict the consequence of its own actions on the environment. However, maintaining a model of environmental dynamics may be computationally costly, and does not consider the usefulness of the agent's knowledge to solve the task. Precisely, rather than visiting all possible states, the agent should focus on task-relevant information. Moreover, these models remain hard to train in high-dimensional state spaces such as images and tend to limit long-horizon performance due to model drift.

In this chapter, we introduce a novel technique called goal-based curiosity (GoCu). We propose an alternative solution to the curiosity mechanism by generating an exploration bonus based on the agent's knowledge about its environment. To do so, we use the idea of goals to automatically decompose a task into several easier sub-tasks, and, skills, the ability to achieve a goal given an observation. In summary, our main contribution is that we manage to bypass most pitfalls of previous curiosity-based works with the following idea: GoCU is not based on the ability to predict the future nor the novelty of the states, but on learning which skills are mastered. That is, often visiting a state will result in a high intrinsic reward unless the agent perceives that it has good knowledge about the associated skills. In detail, given an observation the agent predicts which goals it masters. We then reward the agent when the uncertainty to achieve them is high, thereby encouraging curious behaviors. Our method relies on two deep neural networks: (1) a deep neural network to embed the states and goals into a latent space and (2) a predictor network to predict the capabilities of the agent. In order to improve generalization of skills to novel scenarios, we embed the goals into a latent space using a variational autoencoder [48]. Note that skill-discovery is done in a curriculum fashion by letting the agent uses its experience to gradually discover task-relevant skills.

We first evaluate our approach on a set of sequential sparse tasks (several sequential steps are required in order to achieve the final goal) in the Minigrid [53] environment. Next, we show that our approach can scale to large environments. It can also learn policies in the MuJoCo environment, as well as in several Atari games, in a small number of steps. We compare our algorithm against proximal policy optimization (PPO) [38], and advantage actor-critic (A2C) [37] agent as well as state-of-the-art curiosity-based RL methods. We show that curiosity-driven exploration is crucial in tasks with sparse rewards and demonstrate that the proposed agent explores faster as compared to other methods.

## 3.2 Related Work

The problem of motivating the agent to explore sparse reward environments was studied in the works that we discuss in this section. Most approaches can be grouped into three classes: reward shaping and auxiliary tasks, goal conditioned learning, and intrinsic curiosity.

**Reward Shaping and Auxiliary Tasks.** Reward shaping [51] aims to guide the agent towards potentially promising solutions with an additional hand-crafted reward. However, this idea is limited to specific domains by the necessity of human engineering. Other attempts have been made to design this additional reward without supervision. For instance, Stadie *et al.* [54] improve exploration by estimating visitation frequencies for states and state-action pairs. When a state-action is not visited enough, a novelty bonus is assigned. Similarly, several works [55, 56] are based on a pseudo-count reward bonus to encourage the exploration of novel states. These methods assume that the states need to be equally visited. Obviously, this assumption is no longer valid in sequential tasks. By contrast, we let the agent learn which states need to be explored in priority. Another solution [57] learns the reward function based on an adversarial reward learning but is limited to behaviors that are easy to demonstrate. UNREAL [58] trains the agent on unsupervised auxiliary tasks. In order to perform well, the agent must acquire a good representation of the states which entails that it must explore the environment. However, this method requires experts to design the auxiliary tasks. On the other hand, our method enables the agent to discover sub-tasks without any prior belief about the target task.

**Goal Conditioned Learning.** Goal conditioned learning motivates an agent to explore by constructing a goal-conditioned policy and then optimize the rewards with respect to a goal distribution. For instance, the approach, universal value function approximators [40], trains a goal-conditioned policy but is limited to a single goal. Andrychowicz *et al.* propose an implicit curriculum by using visited states as target and improve sample efficiency by replaying episodes with different goals [41]. However, selecting relevant goals is not easy. Another work [59] presets an algorithmic framework to generate a series of increasing distant initial states from a goal or to embed the goals into a latent space and then sample the goals [60]. Similarly, Reinforcement Learning with Imagined Goals (RIG) [61] trains a policy to reach a latent goal sampled from a VAE. However, this formulation does not have a notion of which goals are hard for the learner. Instead, our method considers how difficult is to achieve a goal to help the agent to find more efficient solutions to reach it. Besides, we focus on the problem of optimizing learning of multiple skills simultaneously to improve data efficiency. Despite successes in robotic manipulation tasks, generating increasingly difficult goals is still challenging [62], especially in sequential tasks. In contrast, the proposed method bypasses this issue by training the agent to gradually acquire new skills using multiple goals at once, and adapts curiosity based on the complexity of each skill.

**Intrinsic Curiosity.** Intrinsic curiosity refers to the idea of self-motivating the agent through an intrinsic reward signal, in order to guide the agent towards the desired outcome. Most techniques can be grouped into two approaches.

The first approach to generate curiosity relies on estimating a novelty measure, in order to encourage the exploration of novel states. Schmidhuber *et al.* propose for the first time a curiosity reward proportional to the predictability of the task [63] and then, based on the learning progress [64]. To give another example [65], the agent receives a reward proportional to the *Bayesian surprise*, the estimation of which data affect the prior belief about the world of the observer. Several other works focused on maximizing a measure such as empowerment [66] or, competence progress [67]. Recently, a solution was proposed to measure uncertainty to generalize count-based exploration [55]. Nevertheless, these models are hard to train in high-dimensional state spaces such as images, or, in stochastic environments.

The second approach consists in optimizing the ability of the agent to predict the consequences of its actions. Racanière *et al.* base the exploration of the agent on the surprise - the ability of the agent to predict future [52]. A related concept [68] estimates the surprise of the agent by predicting the consequences of the actions of the agent on the environment. Namely, they use an inverse model to learn feature representations of controllable aspects

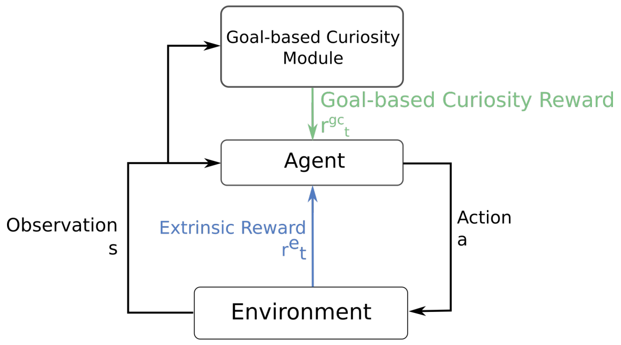


Figure 3.1: In a state $s$, the agent interacts with the environment by performing an action $a$, and receives an extrinsic reward $r^e$. A policy $\pi(s_t;\theta_P)$ is trained to optimize the sum of $r^e$ and $r^{gc}$. The intrinsic reward $r^{gc}$ is generated by the goal-based curiosity module to favor the exploration of novel or complex states.

of the environment. Nonetheless, predicting the future is unsuitable for domains in which variability is high, and tend to limit long-horizon performance due to model drift. In that work, the intrinsic curiosity combines the agent's experiences and a predictor to estimate how complex are the future states surrounding the current state; while embedding states into a latent space to improve generalization. Another line of work aims to predict the features of a fixed random neural network on the observation of the agent [20]. However, the low sample efficiency does not show clearly how to adapt this method to large scale tasks. The intrinsic motivation may include the comparison between the current observation and the observations in an episodic memory [21]. This comparison uses the concept of reachability, but is limited to episodic tasks and does not use the idea of skill learning. Our work differs by using proposing a technique that scales to continuous tasks and let the agent discover new skills and reinforce them based on their difficulty. Our solution is also more data efficient since we only need to do predictions on a small number of skills instead of on the full episodic memory. By embedding the skills and states, we also improve generalization capability of our agent. Finally, our method enables the use of efficient goal sampling strategies to reduce training time. A solution is to [69] maintain a list of behaviors, and reward the agent for exploring novel behaviors. In this chapter, we step towards this idea by considering the impact of the agent's actions on its ability to achieve skills. In the depicted method, only the agent's experience is required to predict the uncertainty to master skills, thus alleviating the need of predicting complex changes in the environment. Besides, skill discovery is carried in a curriculum way to let the agent gradually acquire task-relevant skills.

## 3.3 Skill-Based Curiosity

We consider an agent interacting with an environment; at each time step $t$ the agent performs an action and receives an extrinsic reward $r^e_t$ supplied by the environment. In sparse reward tasks, rewards are infrequent or delayed which entails that agents learn slowly.

We propose to introduce a new reward signal - goal-based curiosity reward $r^{gc}$, to encourage the agent to discover new skills (Figure 3.1). In our approach, at time $t$ the agent

**Algorithm 1** PPO + GoCu pseudo-code

```
1: Collect $D = \{s^{(i)}\}$ using random exploration
2: Train VAE on $D$ by optimizing (2.17)
3: Initialize replay buffer $R$
4: Initialize goal buffer $R_g$
5: Initialize environment env
6: Initialize goal-based curiosity module GCM
7: $t = 0$
8: for rollout r=1 to $M$ do
9:     Sample K goals from $R_g$ $g' = \{\phi(g) \sim f_p(g)\}$
10:    for l=1 to N do
11:        Sample $a_t \sim \pi(a_t|s_t)$
12:        Collect $s_{t+1}, r^e_t \sim \text{env}(s_{t+1}, r^e_t|s_t, a_t))$
13:        Calculate curiosity-based reward $r^{gc}_t = \text{GCM}(s_{t+1})$
14:        Add $s_t, s_{t+1}, r^e_t, a_t, r^{gc}_t$ to $R_r$
15:        $t+ = 1$
16:    Calculate returns $R_{e,r}$,$R_{gc,r}$ for extrinsic and curiosity-based reward
17:    Calculate advantages $A_{e,r}$,$A_{gc,r}$ for extrinsic and curiosity-based reward
18:    Calculate the new advantage $A_r = A_{e,r} + \sigma A_{gc,r}$
19:    Optimize PPO using on $R_r$ and $A_r$
20:    Update GCM every K episodes on $R$
21:    Add new goals to $R_g$ given $R_r$
```

receives the sum of these two rewards $r_t = r^e_t + r^{gc}_t$. To encourage the agent to explore the environment and favor the discovery of new skills, we design $r^{gc}$ to be higher in *complex* states. To do so, a curiosity signal is produced in novel states, and, in states involving skills that the agent seeks to reinforce. The policy $\pi(s_t; \theta_P)$ is represented by a deep neural networks. Its parameters $\theta_P$ are optimized to maximize the following equation:

$$\max_{\theta_P} \mathbb{E}_{\pi(s_t;\theta_P)}[\sum_t r_t] \tag{3.1}$$

Any RL algorithm could be used to learn the policy $\pi(s_t; \theta_P)$. In this work, we use proximal policy optimization (PPO) [38] as policy learning method. Our main contribution is the goal-based curiosity module (GCM) module that helps the agent in the quest of new knowledge. We describe more details of each step below. See Alg. 1 for a more formal description of the algorithm. In order to combine the extrinsic $r^e$ and the goal-based curiosity $r^{gc}$ reward, we decompose the return into two terms $R = R_e + R_{gc}$. These two terms are estimated by collecting samples from the environment. The state values $V_e$ and $V_{gc}$ are computed by the same critic neural network with two heads, one for each value function. Finally, we combine them to obtain the advantage function:

$$A_r = A_{e,r} + \sigma A_{gc,r} \tag{3.2}$$

where $\sigma$ controls the importance of the goal-based curiosity advantage. Note that the

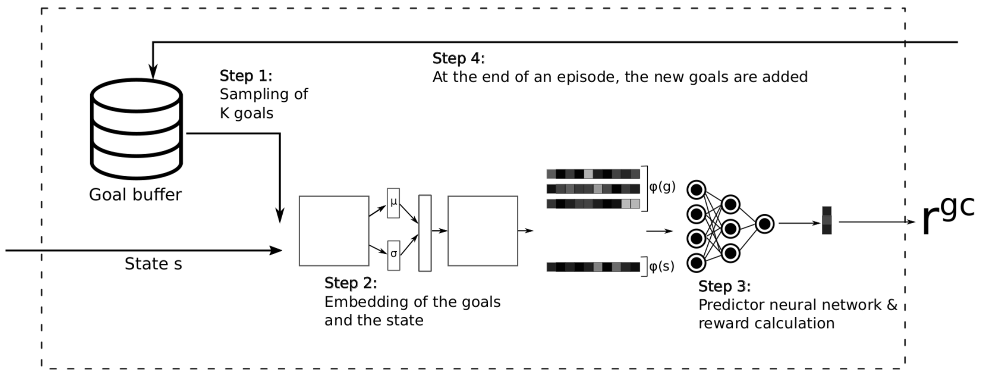


Figure 3.2: Goal-based curiosity module. The module takes as input an observation $s$, and, at the beginning of every episode samples K goals. The goals and the observation are embedded during step 2, $\phi(g)$ and $\phi(s)$ respectively. Step 3 predicts the probability that each goal is mastered; and given this vector of probabilities calculates the curiosity reward signal $r^{gc}$. At the end of each episode, multiple new goals are added to the goal buffer based on the states experienced.

measure performance that we aim to improve with GoCu is the extrinsic rewards; the intrinsic rewards only intend to guide exploration of the environment.

### 3.3.1 Goal-Based Curiosity Module (GCM)

We build the goal-based curiosity module (GCM) on the following idea. Rather than using count-based exploration, or next frame prediction error as exploration bonus, GCM rewards the states based on the uncertainty to master *skills*. We introduce the concept of *skill* that represents the ability of the agent to achieve a goal $g$ in a state $s$. A skill is mastered when the agent can achieve $g$ from $s$ with a high degree of confidence. The confidence is measured in two ways: (1) the difficulty to go from $s$ to $g$, (2) the progress of the agent to achieve this skill. As a result, curiosity incentivizes the agent to seek new knowledge and discover new skills that might lead to the final goal.

A straightforward solution is to consider the final goal as the goal to be achieved in any state. However, in sparse reward environments, reaching the final goal may be infrequent, entailing that for most of the episodes the agent only experiences failures. Therefore, training a probabilistic model for predicting if the final goal is mastered is highly inaccurate. Instead, we propose to estimate if the agent can solve multiple intermediate goals. Since these goals are easier to master - encountered more frequently, the estimation becomes more accurate while carrying information about the final goal. In the absence of domain knowledge, a general choice is to set the goals as states experienced by the agent during exploration.

In details, the goal-based curiosity module takes as input the current observation and produces the curiosity based reward $r^{gc}$. The algorithm can be broken down in fours parts (Figure 3.2). First, at the beginning of each episode, $K$ goals are sampled with $1 \leq K \in \mathbb{N}$ (Section 3.3.2). Second, at each step, we embed the current observation and the goals into a latent space using a variational autoencoder $ve : O \to \mathbb{R}^n$ (Figure 2.3b). Third, the agent predicts the probability that each goal can be achieved from the current state. Our implementation relies on a deep predictor neural network which predicts the probability that a goal is mastered $m : \mathbb{R}^n \times \mathbb{R}^n \rightarrow [0,1]$ and the curiosity based reward is calculated. To help the agent in the quest of new knowledge, we design the reward to encourage the agent to move towards regions of the state-action space that are simultaneously novels and for which the skills are "hard-to-learn". Namely, we take advantage of uncertainty given the probabilities that the goals are mastered by the agent. We give details about the reward calculation in Section 3.3.3. Finally, at the end of the episode, the GCM is updated according to the experienced observations and the predictor neural network is retrained to fit with the new knowledge of the agent (Section 3.3.4).

### 3.3.2 Goals

As described earlier, estimating if the agent can solve the entire task is challenging. Instead, we introduce the concept of *skill* to decompose the problem of reaching a distant final goal into multiple easier goals. A skill is the ability of the agent to reach a goal from a given state. Let $G$ be the space of possible goals. We suppose that every goal $g \in G$ corresponds to some predicate $f_g : S \rightarrow \{0, 1\}$ and that a skill is mastered given a state $s$ when $f_g(s) = 1$. In order to keep a consistent representation, we suppose the goal space $G$ to be the same as the state space $S$, $G = \mathbb{R}^n$.

Working with high-dimensional state spaces such as images is complex. Instead, we embed the goals and states into a latent space $Z$ using a variational autoencoder $ve$, $\phi(s) = ve(s)$ and $\phi(g) = ve(g)$. In addition to improve data efficiency, it crucially improves generalization capability of our model [70, 71]. That is, the deep predictor network can generalize knowledge for two goals sharing a similar embedded representation. We initialize the parameters of a VAE by stepping a random agent for a small number of iterations before beginning training. Then, we train a VAE on observations seen along trajectories. The goals are stored in a goal buffer $R_g$.

In practice, every episode starts with sampling a sub-set $g'$ of latent goals given a distribution function $f_p$:

$$g' = \{\phi(g) \sim f_p(g)\} \text{ sample K goals} \in G \tag{3.3}$$

In the current implementation, the probability of sampling a goal $f_p(g)$ is uniform for all the goals. In future work, we anticipate more complex distributions to take into account the difficulty of the goals.

During training, the goals aim to provide additional feedback to the agent to improve exploration. At the end of an episode, we add a mechanism to further enable sample-efficient learning. In addition to the state reached at the end of the episode, we artificially generate new goals by selecting states experienced during the episode. We discuss in Section 3.4.3 the strategies to select the new goals after experiencing an epoch. To keep the goals as different as possible, we only add the goals that are different from the existing ones. To do so, we measure the similarity between the embedded goals in a similar manner to that of the RIG [61].

$$sim(\mathrm{g1}, \mathrm{g2}) = -\|\phi(g1) - \phi(g2)\|_A \propto \sqrt{\log \mathrm{ve}_\phi(\phi(g2)|\mathrm{g1})} \tag{3.4}$$

where $g1$ and $g2$ are two goals, their embedded representation denoted by $\phi(g1)$, $\phi(g2)$ respectively, and $\mathrm{ve}_\phi$ the VAE encoder.

### 3.3.3 Reward Calculation

Given the embedding representations of the current state $\phi(s)$ and the goals, GCM produces the curiosity reward $r^{gc}$. We propose to train a predictor network to predict the probability that a skill is mastered - the probability that a goal $\phi(g)$ can be achieved from $\phi(s)$, $m(\phi(s), \phi(g))$. The prediction is made for each pair of current state - goal, and the concatenation produces a vector: $g_{active} = \langle (m(\phi(s), \phi(g_1)), ..., m(\phi(s), \phi(g_K))) \rangle$. Given these predictions we define the goal-based curiosity reward:

$$r^{gc} = \left[ \frac{(\langle \alpha \rangle - h\langle (m(\phi(s), \phi(g_1)), ..., m(\phi(s), \phi(g_K))) \rangle)}{\delta} \right] \tag{3.5}$$

where $h$ is a function mapping the probability vector $g_{active}$ to a real number which expresses the *complexity of the state.* A state is complex when skills are *hard-to-learn* - the goals can be achieved with small probabilities. The parameter $\delta$ controls the scale of the curiosity reward, and $\langle \alpha \rangle$ the sign of the intrinsic reward. In the current implementation, we use $\alpha = \langle 1.0 \rangle$, a uniform vector of 1, and $h = \max()$. The choice of $h$ depends on the desired exploration behavior. In practice, $h = \max()$ results in a positive reward unless all skills are considered as mastered by the agent, forcing a complete exploration of the environment. Another choice could be $h = \mathrm{mean}()$; the curiosity signal would decrease when a majority of skills (but not necessarily all) are mastered with high probabilities. As a consequence, the agent is forced to explore novel states, leading to a faster but possibly partial exploration of the state space.

In other words, predicting that the agent cannot achieve a goal results in a positive curiosity-based reward, $r^{gc} > 0$. Curiosity pushes the agent to explore novel regions of the state space, and discover more efficient solutions to achieve the goals. One issue with the combination of extrinsic reward and curiosity-based reward is the scaling of the intrinsic reward which may vary between tasks. In order to mitigate this scaling problem, we normalize the curiosity-based reward:

$$r_t^{gc} = \frac{r_t^{gc}}{\sigma(R_{gc})} \tag{3.6}$$

with $\sigma(R_{gc})$ an estimation of the standard deviations of the curiosity-based reward returns.

### 3.3.4 Training the Goal-Based Curiosity Module

The VAE network and predictor network are optimized every $P_{vae}$ and $P_m$ time-steps respectively. The agent's observations are collected in a memory bank and every $P_{vae}$ time-steps the VAE is retrained for 20 epochs. When the memory bank is full, a random element is replaced with the new observation. This is necessary to ensure that the representation of the goals adapts over exploration.

The update of the predictor network can be broken down into three parts: (1) training samples are collected during exploration, (2) a label is assigned to each example, and (3) the network is optimized. First, the policy interacts with the environment to produce a set of trajectories $\{\tau^1, ..., \tau^i\}$. Within each trajectory, each visited states $s_t$ is concatenated with the $d_{max}$ next states $\{(s_t \cdot s_{t+1}), (s_t \cdot s_{t+2}), ..., (s_t \cdot s_{t+d_{max}})\}$. For each training sample $(s_t, s_{t+y})$, we consider the left element to be the initial state (s), and the right element the goal that the agent aims to achieve (g).

Second, a label (mastered/unmastered) is assigned to each example - skill, $(s, g)$. To do so, we compute a complexity score for each example $sc(s, g)$ that takes into account the difficulty to go from $s$ to $g$, and the progress of the agent to achieve this skill. In details, we use a linear combination of the prior knowledge of the predictor network $p(s, g)$ and the number of time-steps $dist(s, g)$ between the state $s$ and the goal $g$ - skill difficulty:

$$sc(s,g) = \lambda prior(s,g) + \beta dist(s,g) \tag{3.7}$$

where $prior(s, g)$ is the belief of the predictor network about the skill, and the scalars $\lambda$ and $\beta$ weight each component. Given $sc(s, g)$, the probability that in a state $s$ the skill $(s, g)$ is considered mastered $p_{skill}(s, g)$ (label +1) follows:

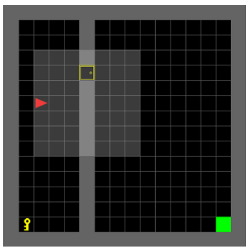
$$p_{skill}(s,g) = 1.0 - \frac{1.0}{1.0 + e^{-b(sc(s,g)-a)}} \tag{3.8}$$

where $a > 0$ controls the boundary between the two classes. The term $a$ acts as the degree of certainty to decide whether the agent should explore new parts of the environment or continue to acquire knowledge about this skill. It can be interpreted as follows: to ensure that the skills can be achieved with high certainty, $a$ should be small. Hence, only low difficulty skills are labeled as mastered (label +1). Thereby, during exploration, GCM produces a positive reward (Eq. 3.5) in states involving skills that the agent seeks to reinforce (label 0). Finally, the predictor network is optimized every $P_m = 20000$ time-steps via supervised learning.

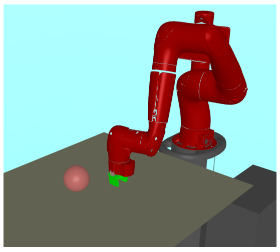
## 3.4 Experiments

### 3.4.1 Environments

We evaluate our agent on a set of tasks in the MiniGrid environment [53]. The tasks are partially observable, with varying components such as navigation, sequential solving, and planning. We consider the *Door & key* task domain that we modify to generate RGB images. An example is depicted in Figure 3.3a. It consists in a board surrounded by grey



(a) Door & key environment (Minigrid)



(b) Visual reacher environment (Mujoco)

Figure 3.3: Examples of training environments. Door & key environment with a board of size $16 \times 16$ (left). Example of task in the MuJoCo domain consisting in a 7-DoF Sawyer arm that aims to reach a target (right).

walls that the agent cannot cross. The agent is represented by a red triangle, and the final goal to reach by a green square. In order to reach the final goal, the agent has first to pick a key randomly positioned on the map, then unlock a door and finally get to the green square. The agent receives as input a 32×32 RGB image of the visible cells, represented by a light gray rectangle. The agent can choose among the 7 possible actions: *turn left*, *turn right*, *move forward*, *pick up an object*, *drop the object being carried*, *open doors - interact with objects* and *complete the task*. Solving such sparse tasks is challenging since the agent only receives a positive reward +1 when it reaches the final goal. Most of the times, the agent only receives 0 as reward. Using efficient exploration strategies such as curiosity is crucial.

### 3.4.2 Implementation details

The agent was trained for 5000000 frames. Any RL algorithm could be used to train our model. In our experiments, we combine PPO and GoCu that indicates to perform well. The PPO agent processes the input with a convolutional neural network that consists in a serie of three convolution layers, with 16,32,64 filters respectively, kernel size 2×2, stride of size 2 and padding 1, with rectified linear unit (ReLU) function as the activation function. In the experiments, we use a rollout of length 128, an entropy coefficient 0.01, batch size 256, a discount rate $\lambda = 0.99$, value loss term coefficient 0.5, and a clipping $\epsilon = 0.2$. The experiments were run with 16 parallel environments and optimized with Adam [72] for 4 optimization steps and a learning rate $\alpha = 0.0001$. At the beginning of an episode, $K = 50$ goals are sampled from the goal buffer with a maximum size 1500. The intrinsic advantage is combined in PPO with $\sigma = 0.25$ and $\delta = 10$ as scaling factor of the curiosity-based reward (Eq 3.5). We use $\lambda = 3$, $\beta = 1$ to weight the prior belief against new knowledge, and $b = 0.8$, $a = 15$ to assign the label of training samples. In our experiments, we observed that when $\lambda > 10$, the agent struggles to integrate new knowledge - discover new skills, whereas a small value (i.e. $\lambda \leqslant 2$) may result in overfitting of observations. The parameter $a$ controls the difficulty threshold to classify whether the skill is mastered or not. We found that a value $10 < a < 30$ performs well in most tasks. Since $b$ controls the boundary between the two classes, keeping a large value $b > 0.7$ makes the boundary large, crucial to improving performance of the predictor network.

We pretrained the VAE with 10 000 images collected with a random exploration policy.

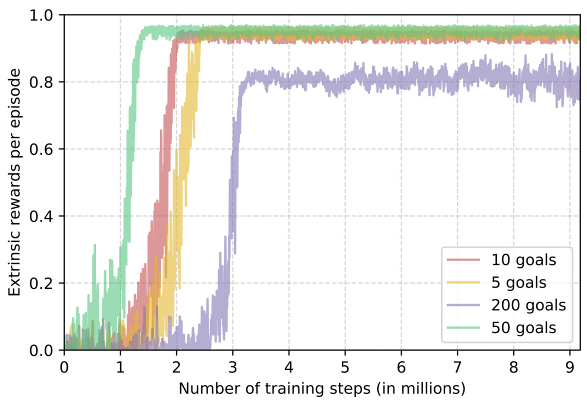


Figure 3.4: Extrinsic rewards achieved by PPO+GoCu for different number of goals in the task *Door & key 8×8*.

The encoder neural network consists in 3 convolutional layers with the following parameters (filter: 32,32,32 kernel size: 3×3, 3×3,3×3, stride size: 2,1,1) and apply a rectifier non-linearity. They are followed by a fully connected layer of size 128. The latent vector $\mu$ and $\sigma$ are of size 10. The 4 last layers are the corresponding decoding layers. We use the compressed representation, the fully connected layer that follows the encoder network, to embed the goals and states. To measure the distance between two goals (Eq 3.4), we found that $A = I$ corresponding to Euclidean distance, performs better than cosine or Mahalanobis distance.

The predictor network is a fully-connected network with 5 hidden layers of size 512. It takes as input the concatenation of the embedding of a state and a goal and outputs (softmax function) the probability that the skill is mastered. The input of each layer is batch normalized and followed by a ReLu activation function. We use Adam to optimize the predictor network, with learning rate 0.0005 on batch of size 128. Every 20000 training-steps, the network is optimized on the new training samples for 15 epochs.

### 3.4.3 Ablation Analysis

#### Learning with Variable Numbers of Goals

To isolate how much the number of goals contributes to our algorithm, we vary the number of active goals while fixing other parts of our method. We compare the performance of PPO+GoCu trained with the same hyperparameters and a number of goals $K \in \{5, 10, 50, 200\}$. For this experiment, the agents were trained on the *Door & key* task with a board of size $8 \times 8$. The goal was to evaluate the gain we can have by increasing the number of goals sampled at each epoch.

The results shown in Figure 3.4 show that increasing the number of goals greatly improves performance. However, we also note a too large number of goals, $K = 200$, makes the curiosity saturate. Experiments suggest that an excessive number of goals increases prediction errors of the predictor network. In contrast, learning an insufficient number of

Table 3.1: The table compares the average reward for various goal generation strategies, and on different tasks.

| | Time-steps to converge (in millions) | | | Average Reward (at convergence) | | |
|---|---|---|---|---|---|---|
| Method | 5×5 | 8×8 | 11×11 | 5×5 | 8×8 | 11×11 |
| random | **0.22 ± 0.05** | 2.20 ± 0.10 | 4.1± 0.42 | 0.96± 0.04 | 0.94 ± 0.05 | 0.94 ± 0.07 |
| proportional | 0.25 ± 0.01 | **1.60 ± 0.04** | **3.3± 0.16** | **0.96± 0.03** | **0.95 ± 0.05** | **0.95 ± 0.08** |
| late | 0.32 ± 0.03 | 2.56± 0.21 | 3.75± 0.36 | 0.94± 0.04 | 0.95 ± 0.06 | 0.94 ± 0.10 |

skills ($K \in \{5, 10\}$) decreases convergence speed. Note that in the $11 \times 11$ environment the results show similar trends. We observe that $K = 50$ was optimal, and in practice, $K \in \{10, 50\}$ works on many tasks.

### Goal Generation Strategies

In this section we experimentally evaluate different strategies for adding new goals to the goal buffer. At the end of an episode, $l$ new goals are added to the goal buffer, with $l$ a hyper-parameter that we empirically set to 7. We consider the following strategies to generate the new goals:

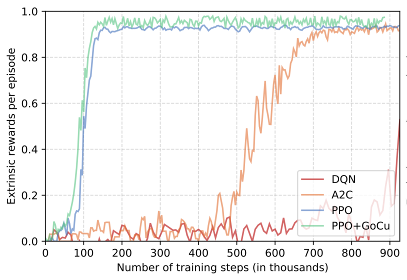
- *random* - $l$ states encountered during the last epoch are randomly added to the goal buffer,
- *proportional* - $l$ states are added to the goal buffer with a probability inversely proportional to their order of visit (the final state has the highest probability),
- *late* - $l$ states encountered during the $l * 4$ last time-steps of the epoch are randomly added to the goal buffer.

All the strategies were tested with the same hyper-parameters. The results comparing these strategies can be found in Table 3.1. We found that in the task 5×5, all the strategies lead to similar performance. As can also be seen, the *proportional* strategy is the most efficient one in term of convergence speed and average reward (at convergence) in the task 8×8 and 11×11. The *proportional* strategy achieves an almost perfect score in the three tasks.

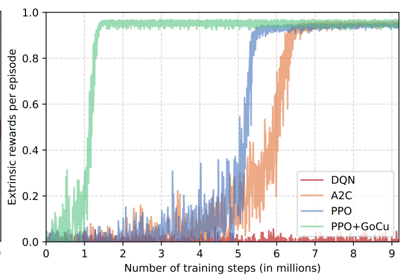
## 3.4.4 Maze Tasks

In this section, we perform a set of two experiments to evaluate the overall performance of our algorithm. First, we verify if GoCu achieves better performance than traditional RL methods. Second, we compare GoCu against state-of-the-art curiosity-based learners.

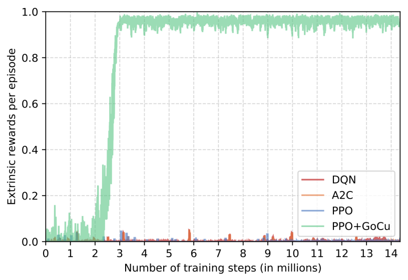
In order to verify that GoCu improves performance, we evaluate PPO [38] with and without GoCu on the *Door & key* task. Moreover, we compare against A2C [37] and DQN [73, 1]. We report the average extrinsic reward supplied by the environment. We present



(a) Door & key $5 \times 5$

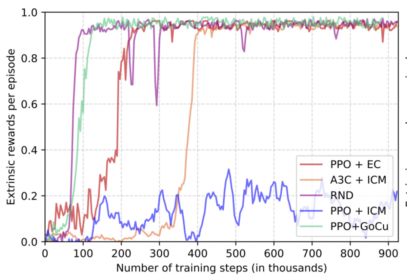


(b) Door & key $8 \times 8$

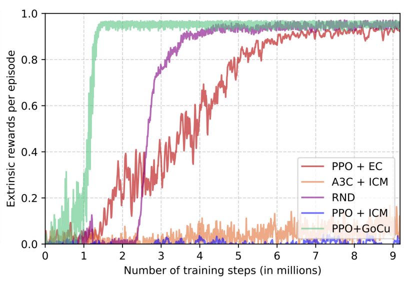


(c) Door & key $11 \times 11$

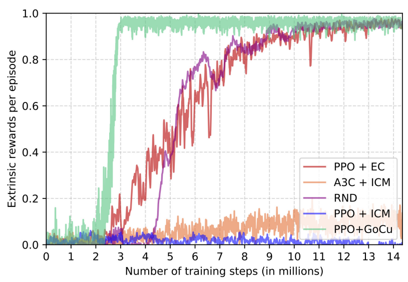
Figure 3.5: Extrinsic rewards achieved by the agent on the *Door & key* task. An episode is considered successful when the agent reaches the final goal resulting in a +1 reward.



(a) Door & key $5 \times 5$



(b) Door & key $8 \times 8$



(c) Door & key $11 \times 11$

Figure 3.6: Comparison of GoCu algorithm (green) to other algorithms on several environments with varying size (degree of sparsity).

in Figure 3.5 the evolution of the extrinsic reward achieved by the agent. In such sparse tasks, the DQN agent fails to solve the task. The learning curve shows that our agent outperforms PPO trained without GoCu as well as A2C. After converging, PPO+GoCu achieves an almost perfect score in 95% of the runs, slightly higher than PPO trained without curiosity, 93%. Only our method scales with the size of the environment and is significantly faster in term of convergence speed.

In addition, we also conduct a set of experiments to compare GoCu with RND [20], PPO+EC [21], A3C+ICM [68], and PPO+ICM [68]. We perform an evaluation using the same open-source implementation with exactly the same hyper-parameters of the original works. We ran the agents for 3 runs each for 1 millions time-steps on the *Door & key* $5 \times 5$ task, 10 millions time-steps on *Door & key* $8 \times 8$, and 15 millions time-steps for the board of size $11 \times 11$. We can see in Figure 3.6 that in 2 out 3 domains our method is significantly faster than the baselines and improves the performance of PPO, while on task $5 \times 5$, we obtain similar performance with RND. We also observe that PPO + ICM and A3C+ICM result in poor performance in the early stages of learning in the three tasks and was not able to reach the goal. Note that we didn't fine-tune hyper-parameters for each domain, that might be an opportunity to further improve performance.

## 3.4.5 Simulated Robotics Tasks

Our second set of experiments explores domains from the Mujoco control benchmark [74]. We evaluate performance on five tasks. (1) *Visual Reacher*: a 7-DoF Sawyer arm learns

Table 3.2: Comparison of seven models over five tasks in the Mujoco domain. We report the final distance to goal for 5 runs that were performed for 500K simulations steps.

| | Final Distance Goal (at convergence) | | | | |
|---|---|---|---|---|---|
| Method | Visual Reacher | Visual Door Hook | Visual Pusher | Visual Multi-Object pusher | Visual Pick and Place |
| HER | 0.09 ± 0.01 | 0.42 ± 0.02 | 0.26 ± 0.02 | 0.36 ± 0.03 | 0.19±0.01 |
| A3C+ICM | 0.15 ± 0.03 | 0.41 ± 0.04 | 0.32 ±0.04 | 0.35 ± 0.03 | 0.17 ± 0.02 |
| PPO+ICM | 0.08 ± 0.01 | 0.32 ± 0.02 | 0.11 ±0.01 | 0.21 ± 0.02 | 0.12 ± 0.01 |
| PPO+EC | 0.08 ± 0.02 | 0.11 ± 0.01 | 0.12 ±0.02 | 0.18 ± 0.03 | 0.11 ± 0.01 |
| RIG | **0.07 ± 0.02** | 0.08 ± 0.01 | 0.17 ± 0.02 | 0.10 ± 0.02 | **0.08 ± 0.01** |
| DSAE | 0.10 ± 0.01 | 0.28 ± 0.05 | 0.25 ± 0.01 | 0.27 ± 0.02 | 0.13 ± 0.01 |
| PPO+GoCu | **0.07 ± 0.01** | **0.06 ± 0.02** | **0.09 ± 0.02** | **0.06 ± 0.03** | 0.13 ± 0.02 |

to reach goal positions. The end-effector (EE) is constrained to a 2-dimensional rectangle. (2) *Visual Door Hook*: a 7-DoF Sawyer arm with a hook on the end of the effector to handle a door hook. (3) *Visual Pusher* and (4) *Visual Multi-Object pusher*: the arm aims to push one or two pucks to a target. Note that the end-effector is constrained to only move in the XY plane. (5) *Visual Pick and Place*: a Sawyer arm with a grab at the end of the effector to grab a small ball that the agent has to place at a target. We extend the tasks to produce 32×32 RGB observations. The rewards are sparse indicator functions being the distance in the final state between the objects and their goals. We show an example of two tasks in Figure 3.3b.

For these tasks, we keep the hyper-parameters of PPO unchanged for both augmented and baseline agents. We use Adam to optimize the predictor network as well as PPO. For these tasks, we pretrained the VAE with 10000 images. We compare our method with the following prior works: A3C+ICM [68], PPO+ICM [68], PPO+EC [21], HER (Hindsight Experience Replay) [41], RIG (RL with Imagined Goals) [61], and DSAE (Deep Spatial Autoencoders) [75].

The parameters of GoCu are similar as for the minigrid domain. For this range of tasks, the predictor network consists in a fully-connected network with 3 hidden layers of size 512. Adam optimizer was used with learning rate 0.0003 and batch size 64. We decrease the number of active goals to 25 and the capacity of the goal buffer to 1000.

Agents were trained for 500K time steps. We trained 5 agents separately without additional parameters tuning. Table 3.2 reports the final distance to the goal of our method against six baselines. In these experiments, we observe that HER tends to perform poorly due to the visual representation of the tasks. RIG achieves higher performance in the visual pick and place task thanks to the use of sub-goals that guide the agent towards the goal. From Table 3.2, it is clear that PPO+ICM is significantly outperformed, we suspect that the high similarity among states results in small rewards. On the other hand, our curiosity-based approach outperforms the prior methods only using visual information in 3 out of 5 domains. Overall, the results show that our method can learn policies in the robotic domain. It confirms that GoCu is a crucial element which makes learning from sparse rewards possible.

Table 3.3: Final mean score of our method and baselines on Atari games. We report the results achieved over total 100M timesteps of training, averaged over 3 seeds.

| | Maximum Mean Score (at convergence) | | |
|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Pitfall |
| A2C [37] | 13 | 574 | -15 |
| PPO [38] | 2,497 | 105 | -32 |
| RND [20] | 8,152 | 8,666 | **-3** |
| PPO+EC [21] | 8,032 | 9,258 | -10 |
| PPO+ICM [68] | 351 | 503 | -14 |
| DeepCS [76] | - | 1105 | -186 |
| Average Human [77] | 4,753 | 69,571 | 6,464 |
| PPO+GoCu (Ours) | **10,958** | **12,546** | -4 |

### 3.4.6 Game Playing Tasks

We also evaluate the proposed curiosity method on three difficult exploration Atari 2600 games from the Arcade Learning Environment (ALE) [49]: Montezuma's Revenge, Private Eye, and Pitfall. In the selected games, training an agent with a poor exploration strategy often results in a suboptimal policy. We first compare to the performance of a baseline A2C and PPO implementation without intrinsic reward. The results are shown in Table 3.3. In these games, training an agent with a poor exploration strategy results in a score close to zero, except for Montezuma's Revenge that pure PPO could partially solve.

In addition, we also compare our method against other techniques using alternative exploration bonus (Table 3.3). In Pitfall, PPO+GoCu achieves similar performance as RND. Many interactions yield negative rewards that dissuade our agent and other baselines from exploring efficiently the environment. Despite this issue, PPO+GoCu performs better than algorithms without curiosity. However, in Montezuma's Revenge, PPO+GoCu outperforms average human performance and previous state of the art. On Private PPO+GoCu's performance are higher than other baselines. The strong performance of PPO+GoCu on these tasks suggests that encouraging the agent to learn new skills is a powerful form of intrinsic motivation to enable learning in sparse-reward domains.

## 3.5 Discussion

In this chapter, we introduced a new mechanism for generating curiosity that is easy to implement, scales to high-dimensional inputs such as images, and bypasses the need for maintaining a model of the environmental dynamics. A key concept is to reward the states with associated skills difficult to master; to move the agent towards task-relevant regions of the state-action space. That is, we encourage the agent to gradually acquire novel task-relevant skills. Furthermore, by embedding the states and skills into a latent space and then working in the latent space, we can improve data efficiency as well as generalization to unseen observations.

We have shown its ability to solve complex tasks with continuous state spaces, and exceeds baseline agents in terms of overall performance and convergence speed. To support these claims, we presented an evaluation on a set of complex and sparse visual tasks from Minigrid, MuJoCo, and Atari games. The experiments have demonstrated clear improvements in the agent's capability to make intrinsically motivated choices, greatly improving sample efficiency.

A potential limitation of our proposed method as well as prior approaches is the *vanishing issue* inherent in curiosity. As the agent explores the environment and becomes more familiar with it, the intrinsic bonus may disappear and then learning is only driven by extrinsic rewards, which are often sparse in the real world. In the absence of dense rewards, hoping to stumble into a goal state by chance within an acceptable number of interactions is unlikely, which can lead the agent to prematurely converge to sub-optimal policies.

Another limitation of our proposed framework as well as most prior curiosity-driven agents is that they can get stuck in local optima. This can happen if the environment contains random state-action transitions. The intrinsic reward will be maximized by exploiting such actions which lead to hardly predictable consequences - the intrinsic novelty will be the entropy of the transition. This issue may lead the agent to fall and stay trapped in poor local optima during exploration. Since stochasticity is quite common in the real world (e.g. fine-grained details in the images) or unpredictability caused by a poor learning algorithm, making the intrinsic curiosity robust to such nuisance would be necessary to scale the method to practical problems.

In the next chapter, we propose a formulation for exploration which is able to handle the vanishing curiosity issue as well as stochastic environments.

# Chapter 4

# Exploration via Progress-Driven Intrinsic Rewards

In the previous chapter, we described a first agent whose developmental trajectory is motivated based on its knowledge of the environment. In retrospect, we concluded that a major limitation is the *vanishing* issue inherent in curiosity. In curiosity-driven exploration literature, most prior work employ the absolute prediction error as a drive to explore novel stimuli. Nevertheless, it was shown that such approaches will exhaust their curiosity very quickly [21, 20, 78], prematurely converging to sub-optimal policies. That is, curiosity rewards soon exhaust as the prediction becomes perfect or does not improve, letting the agent with no incentive to encourage revisits of states, regardless of the downstream learning opportunities.

Beyond overcoming the known "vanishing issue", another challenge for curiosity-driven agents is random environments. The agent may get trapped in local optima due to stochasticity in the environment: 1) noisy environment observations, and 2) noise in the actions. In this chapter, we discuss a method to guide the agent's learning that can handle randomness in both observations and actions.

We address both challenges by proposing a robust formulation of curiosity that uses the agent's learning progress on a multi-step horizon scale, and we study its potential in terms of developmental trajectories and sample-efficiency.

## 4.1 Introduction

Inspired by curious behavior in animals, one solution to the exploration problem in sparse reward tasks is to let the agent discover skills that will be useful later (i.e. curiosity-driven learning). A number of curiosity measurement strategies have been proposed such as count-based exploration [56] or the use of information gain [79]. Some other approaches generate a bonus based on the inability to predict the future [14]; but tend to attract the agent to states with stochastic transitions due to hardly predictable environmental dynamics, local optima. If the transitions or observations in the environment are random or contain stochasticity, then even with a perfect dynamics model, an agent trying to maximize a prediction error will tend to continuously seek out such local sources of entropy in the environment. In GoCu [80], curiosity utilizes the agent's knowledge to discover and acquire task-relevant skills. Another line of work aims to predict the features of a fixed random neural network on the observation of the agent [20]. However, capturing complex visual features remains challenging, and, in spite of their ability to capture short-term novelty, they tend to be computationally expensive. Furthermore, when using prediction errors as an intrinsic reward, the intrinsic reward may vanish quickly with additional visitations, converging prematurely to sub-optimal policies.

In this chapter, we introduce a novel definition of curiosity that considers the agent's learning progress on a multi-step horizon as exploration bonus. We postulate that rewarding hard-to-learn regions of the state space is crucial to efficiently explore. By doing so, it drives the agent to seek more knowledge about such regions. To quantify the agent's learning progress in terms of "quantity" and "quality", we propose to measure the divergence between a parametric model (i.e. the current model) and prior models. We build and evaluate two types of models based on: 1) the agent's understanding of the world, 2) the agent's policy. A key idea in our work is to measure progress on a multi-step horizon scale. This allows us to overcome the known "vanishing curiosity" issues of prior work that use the absolute prediction error to guide exploration - curiosity rewards soon exhaust as the prediction becomes perfect or does not improve. To further improve the benefits of our method, we introduce a mechanism called *episodic-skills* to repeatably revisit key regions/skills. In other words, we let the agent discover which skills are important in the environment and encourage the visits of states carrying information about them. Overall, our agent will not receive rewards for reaching states that are inherently unpredictable, making exploration robust with respect to local sources of entropy in the environment.

We benchmark our approach on a set of hard exploration tasks from Minigrid, Super Mario Bros., and Atari games. We compare our method with state-of-the-art curiosity methods. The experimental results show that our agent can escape from local optima in sparse or deceptive reward environments, to learn comparable or superior policies in most of the tasks. Results also demonstrate that progress-driven exploration is vital in tasks with stochasticity in both transitions and observations, and, that our agent explores faster as compared to other techniques.

## 4.2 Related Work

Our method is related to *encouraging exploration* in RL. Most techniques can be grouped into three classes: goal-based, count-based, and curiosity-based exploration.

Goal conditioned learning [39] constructs a goal-conditioned policy to push the agent to acquire new skills and explore novel states. Namely, they optimize average reward with respect to a goal distribution. Universal value function approximators [40] samples a fixed goal at the beginning of each episode and rewards the agent when the current goal can be achieved. Nonetheless, selecting relevant goals remains an open problem. A solution [81] and its recent follow-up [61], proposed to generate increasingly difficult goals to drive the agent towards the final goal.

Another group of works keeps visit counts for states to favor exploration of rarely visited states [82]. To extend count-based exploration to continuous state spaces, a technique [56] is to train an observation density model to supply counts. An alternative solution [83] is to discretize the states by hashing and then apply a simple count-based strategy.

However, one can expect these methods to be less effective when some *valuable* states require more attention - more visits.

This work belongs to the category of approaches that consider curiosity as exploration bonus. Some methods [14] rely on predicting environment dynamics using an inverse or a forward dynamic model. Another class of work uses prediction error in the feature space as a measure of the importance of states [20]. On the other hand, in GoCu [80] curiosity utilizes the agent's knowledge to acquire and discover task-relevant skills. The exploration can also be motivated by introducing a new term in the loss function that measures state diversity [84]. In contrast, our work aims to learn intrinsic rewards by measuring the agent's learning progress. The crucial difference here, however, is that rather than simply estimating diversity, we measure the quantity and quality of the learning progress. Additionally, our algorithm can be trained using policy-based or visual-based progress. Finally, we aim to ameliorate the performance of exploration by efficiently sampling key states (i.e. episodic skills). Exploration bonus can also be based on maximizing information gain about the agent's knowledge of the environment [79]. Another work [85] guides exploration by maximizing an entropy objective. Episodic curiosity through reachability [21] uses the number of time-steps between two states as curiosity measure. Nonetheless, one problem with these approaches is that the curiosity exhausts quickly during training. Moreover, dealing only with local novelty makes it likely for agents to get stuck in undesirable local optima. In our framework, we introduce temporally-extended learning progress to overcome these pitfalls.

## 4.3 Progress-Driven Exploration

We consider an agent performing actions in an environment. In this work we focus on environments where extrinsic rewards are sparse or missing - zero for most of the time steps. At each time step $t$, the agent collects an observation $s_t$, samples an action $a_t$ from a set of actions $A$ following a policy $\pi_{\theta_P}(s_t)$, and then receives an extrinsic reward $r_t^e$. We augment the task reward with an intrinsic exploration bonus $r_t^i$, $r_t = \alpha r_t^e + \beta r_t^i$, where $\alpha$ and $\beta$ are hyperparameters to weight the importance of both rewards. The policy $\pi_{\theta_P}(s_t)$ is represented by a deep neural network and its parameters $\theta_P$ are optimized to maximize the sum of these two rewards, $\max_{\theta_P} \mathbb{E}_{\pi_{\theta_P}}[\sum_t r_t]$.

Our main contribution is a novel intrinsic reward based on the agent's learning progress. Since learning progress is non stationary, it is useful to normalize it by diving the original intrinsic reward by a running estimate of the standard deviations of the intrinsic rewards $\bar{R}_{ib}$. We can assign a curiosity bonus via:

$$r_t^i = \left[ \frac{\bar{r}_t^i}{\sigma(\bar{R}_{ib})} \right] \tag{4.1}$$

where $\bar{r}_t^i$ is the progress-driven reward before normalization.

In the following section we present the key components of our progress-driven curiosity reward.

### 4.3.1 Learning Progress as Intrinsic Reward

Progress-driven exploration is a method to motivate the agent to explore hard-to-learn regions of the state space and escape local optima induced by poorly defined or deceptive extrinsic rewards. To this end, we aim to measure the agent's progress during training. We assume a parametric model that given the current state $s_t$ outputs a probability distribution which can be interpreted as the *agent's knowledge* of $s_t$. In this work, we compare two models: policy-based progress, and representation-based progress. We propose as intrinsic motivation to measure the distance between the current distribution and prior distributions. Concretely, we measure learning progress in terms of *quantity* and *quality*. To deal with global exploration, i.e. exploring the consequences of long-term decisions, we further propose to estimate progress at different time-scales on a batch of observations.

This process drives the agent to gather unseen observations which would maximize long-time horizon learning progress, while encouraging the revisit of *hard-to-learn* state trajectories. Moreover, using progress instead of absolute prediction errors allows us to overcome the known "vanishing curiosity" shortcoming. Our exploration bonus remains large independently of the number of visits, by adapting curiosity based on the agent's understanding of the world. In the next section, we detail how is calculated the intrinsic reward for the *policy-based* and the *representation-based* model.

### 4.3.2 Policy-based Progress (PoBP)

The policy-based progress model (PoBP) works as follows. The agent fills an observation memory with the latest observations. At every time step, we estimate the probability distributions over actions associated with the agent's policy and a set of recent policies for each observation. Given those distributions, we evaluate the agent's learning progress (along the state trajectory stored in the memory) by measuring the distance between the current policy and prior policies. Then we produce an intrinsic bonus to promote effective exploration strategies, based on learning progress. We describe more details of each step below.

Let's consider a policy $\bar{\pi}_\theta(a|s)$ which quantifies the importance of an action $a$ in a state $s$. We first define the probability distribution over actions:

$$\pi_\theta(a|s) = \frac{e^{\bar{\pi}_\theta(a|s)}}{\sum_{z \in A} e^{\bar{\pi}_\theta(z|s)}} \tag{4.2}$$

Please note that this step is only performed when the policy does not estimate a probability distribution over actions given a state. With a slight abuse of notation, we refer to the probabilistic policy as $\pi_\theta$.

We assume a batch $\Psi = \{s_1, ..., s_N\}$ of the $N$ most recent observations and $\Omega$ a set of $M$ prior policies. To fill $\Omega$, every $K$ time steps we substitute the oldest policy in memory with the current policy. Note that at the beginning of each episode the memory $\Psi$ is empty. Policy-based progress can now be estimated as the average distance between the current policy and previous policies over batch of observations:

$$\bar{r}_t^i = \frac{1}{|\Psi|}\sum_{s\in\Psi}\frac{1}{|\Omega|+1}\left[\sum_{\theta_{old}\in\Omega}\left[\eta D(\pi_{\theta_{old}}(s)||\pi_{uni}(s)) + \nu D(\pi_{\theta_{old}}(s)||\pi_\theta(s))\right] + \gamma D(\pi_\theta(s)||\pi_{uni}(s))\right] \tag{4.3}$$

where $\pi_\theta$ is the current policy, $\pi_{\theta_{old}}$ refers to a prior policy, and $\pi_{uni}$ is a fixed *uniform* policy - the probability distribution over the actions is uniform. The first term, $D(\pi_{\theta_{old}}(s)||\pi_{uni}(s))$, measures the learning progress *quality* of prior policy $\pi_{\theta_{old}}$ - it encourages the policy to be as distant as possible to the uniform distribution, which quantifies whether some actions are certainly better than others. The second term, $D(\pi_{\theta_{old}}(s)||\pi_\theta(s))$ measures the *quantity* of learning progress; it is expressed in the distance between the current policy and a prior policy. The third term, $D(\pi_\theta(s)||\pi_{uni}(s))$, is the quality of the current policy.

The distance function $D$ can be any distance measure such as KL-divergence or euclidean distance. Theoretically, the KL divergence would be a robust choice. We define the Kullback-Leibler (KL) divergence between two policies as:

$$D_{KL}(\pi(s)||\pi'(s)) = \sum_{a\in A}\pi(a|s)\log(\frac{\pi(a|s)}{\pi'(a|s)}) \tag{4.4}$$

Although KL measure performs well in many tasks, performance can degrade in some environments with deceptive rewards. In such environments, encountering negative rewards can greatly deteriorate the quality of the learned policy, occasionally causing instability in the training phase. Instead, we propose to embrace the Jensen-Shannon (JS) divergence. We extend it to calculate the distance between two policies as the following:

$$D_{JS}(\pi(s)||\pi'(s)) = \frac{1}{2}D_{KL}(\pi(s)||\pi^*(s)) + \frac{1}{2}D_{KL}(\pi'(s)||\pi^*(s)) \tag{4.5}$$

where $\pi^*(s) = \frac{1}{2}(\pi(s) + \pi'(s))$. We compare in Section 4.4.4 the impact of using KL-divergence or JS-divergence on our method.

### 4.3.3 Representation-based Progess (ReBP)

The reconstruction-based model (ReBP) assesses learning progress based on the agent's understanding of the visual features of its environment. Instead of using the probability distribution over actions, we seek to measure the agent's progress to reconstruct the observations. This approach is motivated by the idea that, to accurately reconstruct a region of the state space, the agent needs to acquire good knowledge about the environment's dynamics and objects, by interacting with them (without the need to learn or model dynamics).

To do so, we propose to measure the quality of a variational autoencoder (VAE) [48] for encoding the latent representations in the data. We assume that an observation $s$ of an agent is generated by a random latent process $z$. Learning progress can be estimated by measuring the distance between the posterior distribution $p(z|s)$ after experiencing new observation and the prior $p(z)$. Several measures such as KL divergence can be used.

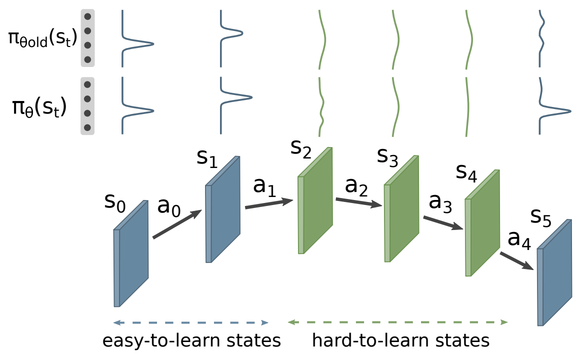


Figure 4.1: Illustration of the idea of episodic skills. Some states (in green) should be given more weight because they are hard-to-learn.

However, computing the posterior distribution $p(z|s)$ is often intractable. Instead, we use a VAE with encoder parameters $\phi$ to model the approximate posterior distribution $q_\phi(z|s)$.

Formally, let $\Psi = \{s_1, ..., s_N\}$ be a set of observations and $\Omega$ a set of prior VAEs. Similarly to PoBP, every $K$ time steps the oldest VAE is substituted with the current version. For online training of the VAE, we store the experience and make 5 epochs of training every 50,000 steps. We express the intrinsic reward as:

$$\bar{r}^i_t = \frac{1}{|\Psi|} \sum_{s \in \Psi} \frac{1}{|\Omega|+1} \left[ \sum_{\phi_{old} \in \Omega} \left[ \eta D_{KL}(q_{\phi_{old}}(z|s)||p(z)) + \nu D_{KL}(q_{\phi_{old}}(z|s)||q_\phi(z|s)) \right] + \gamma D_{KL}(q_\phi(z|s)||p(z)) \right] \tag{4.6}$$

As a result, the agent seeks out states as diverse as possible and in doing so increases the distance between $q_\phi(z|s)$ and $p(z)$ in average - agent's learning quality. In practice, $p(z)$ is specified as a standard normal distribution. On the other hand, it also favors to revisit state trajectories where the agent can improve its understanding by maximizing the distance (progress) between the current VAE $q_\phi(z|s)$ and prior VAE $q_{\phi_{old}}(z|s)$.

### 4.3.4 Episodic Skills

When performing informed exploration, we observed that 1) some states may need more attention because they are hard-to-learn or more task-relevant, 2) high-level exploration requires to balance the loss of easy immediate reward [20]. We propose a simple mechanism relying on an episodic memory to address all of these issues at once (Figure 4.1). We call a skill, $sk$, a sequence of $T$ observations (i.e. a specific region of the state-space). In the absence of domain knowledge, a general-purpose choice is to set $T = N$. The key insight is to prioritize the agent's learning of important skills, which entails that they should be given more weight. Therefore, visiting a state should trigger a positive intrinsic reward not only because it is locally informative (i.e. progress along the current trajectory), but also because it is informative about previously encountered hard-to-learn skills. That is, visiting such a state enables progress in other skills.

### Episodic Memory

The episodic memory stores a set of important skills $\{sk_1 : (s_1, .., s_T), ..., sk_C : (s_1, .., s_T)\}$. It has a limited capacity of size $C$. We define an embedding function, later used to efficiently compare or measure a distance in the episodic memory. We can embed an observation $s$ to a latent space, $\varphi(s)$. When employing *policy-based* progress, we use the representation extracted after passing the input through the sequence of convolutional layers of the policy network, whereas for *visual-based* progress, we use the mean of the VAE's encoder as the state encoding. Intuitively, we extend the above definition to skill embedding, $\varphi(sk) = (\varphi(s_1), .., \varphi(s_T))$.

### Reward Calculation

During intrinsic reward calculation, we compute an exploration bonus independently for each skill $sk$, similarly to the *policy-based* or *representation-based* method. To avoid unstable behaviors, we weight each skill reward, $\bar{r}^i_{sk}$, as being proportional to the similarity with the current observation, $s_t$. This approach is justified by the common sense that progress on distant states is unlikely to be related to the exploration of the current state. The sum of the weighted bonus of each skill is added to the intrinsic reward:

$$\bar{r}^i_t \mathrel{+}= \mathbb{E}_{sk \sim M}\left[\alpha_{sk}\bar{r}^i_{sk}\right] \tag{4.7}$$

where $\alpha_{sk}$ is the similarity between $\varphi(s_t)$ and $\varphi(sk)$. In this work, we use the average pairwise cosine distance as similarity measure.

### Memory Update

At the end of the episode, to decide if a skill should be added in memory, we first measure if the skill novelty is larger than a threshold $t_{novelty}$. Rather than operating directly in the state space, we utilize the cosine distance on skill embeddings to perform this check. This induces a discretization of the embedding space which guarantees to store skills as diverse as possible. Then, hard-to-learn skills are added to the memory if the bonus $\bar{r}^i_{sk}$ is less than $t_{hl}$. As a result, this process keeps skills for which the agent has poor knowledge and made little progress. When the capacity is exceeded we randomly substitute a skill with the new skill.

## 4.4 Experiments

### 4.4.1 Environmental Settings

In this section, we first evaluate our method trained on a fixed and randomly generated environments from the Minigrid [53] domain. Second, we compare *progress-driven exploration* against standard RL and intrinsic reward-based approaches on five Atari 2600 games [86] as well as Super Mario Bros [87], combining intrinsic rewards with sparse

Table 4.1: Reward in Montezuma's revenge with varying batch size $N$. Averages over 10 trials are reported after 600M steps of training.

| | Maximum Mean Score (at convergence) | | | | | |
|---|---|---|---|---|---|---|
| Method | $N$=2 | $N$=6 | $N$=8 | $N$=10 | $N$=15 | $N$=20 |
| PPO+PoBP(KL-divergence) | 7.553 | 9,478 | 9.521 | 8.620 | 5.231 | 5.787 |
| PPO+PoBP(JS-divergence) | 6.542 | 9,125 | 9.024 | 6.785 | 4.023 | 4.008 |
| PPO+ReBP | 6.989 | 9,125 | 7.568 | 5.874 | 2.210 | 2.087 |

or deceptive extrinsic rewards. Third, we test the proposed method in the absence of any exploration bonus on Super Mario Bros. Finally, we provide an ablation study on Montezuma Revenge [86].

As our policy learning method, we use PPO with similar hyperparameters as in the original implementation of RND [20]. The input is passed through three convolutional layers (for Minigrid) or four (other domains). We estimate learning progress on a batch of states of size $N = 6$ over $M = 4$ prior models. We utilize an episodic memory of capacity 15, and $t_{novelty} = 0.10$ (Minigrid) or $t_{novelty} = 0.25$. The frequency update $K$ of the prior model buffer, $\Psi$, is set to 50,000. The value of $\eta$, $\nu$, $\gamma$ is 1.0,0.8,0.2 respectively for PoBP, and 1.0,1.0,0.2 for ReBP. If not specified differently, we employ the JS-divergence (Eq. 4.5) to measure the divergence between two policies. To combine $r_t^e$ and $r_t^i$, we set the coefficient of extrinsic reward $\alpha = 2$, and $\beta = 1$.

### 4.4.2 Ablation Analysis

We also present an ablation study to investigate: 1) the size of batch of observations, 2) the number of prior models.

#### Batch Size of Observations

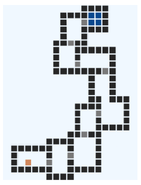
Our technique relies on batch of last observations. One legitimate question is to study the impact of the batch size, $N$, on the performance of the algorithm. We conduct a study with $N$ varying between 2 and 20. As shown in Table 4.1, when using $N = 6$ both models (PoBP, ReBP) can achieve high scores. We observed that *progress-driven exploration's* performance is reasonably robust to the choice of $N$, as long as $N \leq 8$. We can draw the observation that increasing $N$ contributes positively to the capturing of global progress, by providing more information about the current observation.

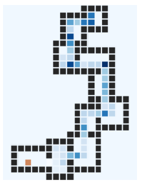
#### Number of Prior Models

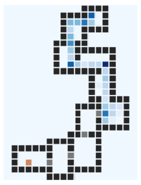
We report experiments showing the effect of increased number of prior models, $M$. Table 4.2 shows that agents trained with a larger number of prior models obtain higher mean

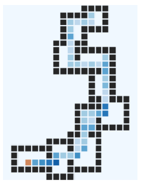
Table 4.2: Score in Montezuma's revenge using different number of prior models $M$. Averages over 10 runs are shown after 600M steps.

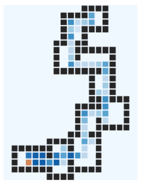
| | Maximum Mean Score (at convergence) | | | | |
|---|---|---|---|---|---|
| Method | $M$=2 | $M$=3 | $M$=4 | $M$=6 | $M$=8 |
| PPO+PoBP(KL-divergence) | 7,254 | 8,452 | 9,478 | 8,767 | 3,571 |
| PPO+PoBP(JS-divergence) | 7,642 | 9,250 | 9,125 | 7,187 | 2,028 |
| PPO+ReBP | 6,988 | 7,125 | 8,934 | 6,245 | 3,698 |

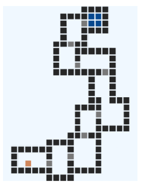

(a) Random

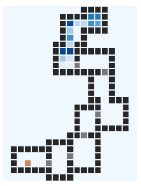

(b) RND

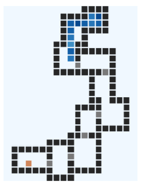

(c) ICM

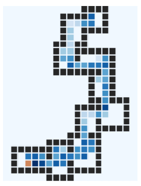

(d) PoBP


(e) ReBP


(f) Random


(g) RND


(h) ICM


(i) PoBP

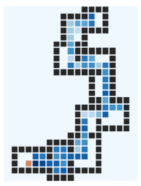

(j) ReBP

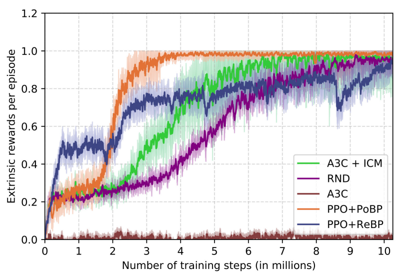
Figure 4.2: State visitation heatmaps averaged over 10 runs for different models: random, RND, ICM, PPO+PoBP, and PPO+ReBP. The models are trained for 40m frames on a fixed maze (top row) and on randomly generated mazes (bottom row) in MultiRoomN10.

returns after similar numbers of updates. On the other hand, when more than 6 prior policies are used, it tends to make progress-driven exploration unstable. We hypothesize that when using a large number of prior models, progress scores of the oldest models are much larger and therefore saturate the intrinsic rewards. The experimental results show that M=4 is a reasonable choice in most domains.

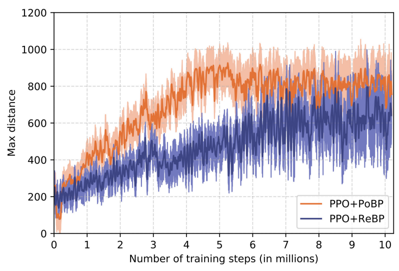
### 4.4.3 Fixed Versus Randomly Generated Environments

To investigate the ability of our agent to learn from randomly generated environments and generalize to unseen appearances, we trained the models on a fixed maze and on randomly generated mazes from the MultiRoomN10 (Figure 4.2) [53]. It consists of a board divided into several rooms. To reach the final goal (red square), it requires to discover rooms and open doors. The environment is challenging due to its sparsity - the agent only receives a positive reward (+1) when it reaches the final goal. The observations are given in the form of grayscale images of size 84×84 of the visible cells surrounding the agent.

We compare PPO+PoBP and PPO+ReBP with three baselines: random agent, RND [20], and ICM [14]. A random policy can only explore the first room. We observe that

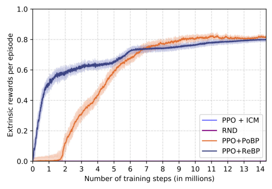


(a) Sparse reward setting

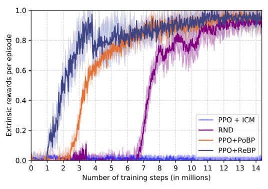


(b) No reward

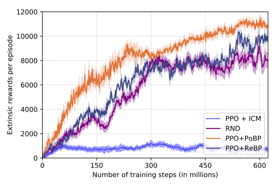
Figure 4.3: Average extrinsic reward obtained with sparse reward setting (left) and maximum distance achieved with no extrinsic reward (right) in the Super Mario Bros environment. We run every algorithm with a repeat of 10 and show all runs. Mean and standard error of the mean over trials are plotted.



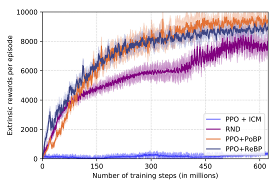
(a) MultiRoom



(b) Door & Key



(c) PrivateEye



(d) Montezuma

Figure 4.4: Reward as a function of training step for a variety of hard exploration tasks. We run every algorithm with a repeat of 10.

ICM fails to efficiently explore the environments in both scenarios. While RND is able to solve the fixed maze task, it struggles in randomly generated mazes. On the other hand, our methods could learn efficient strategies to discover all the rooms. This suggests that progress-driven exploration allows generalization across the environments and is more robust to changes in randomly generated mazes.

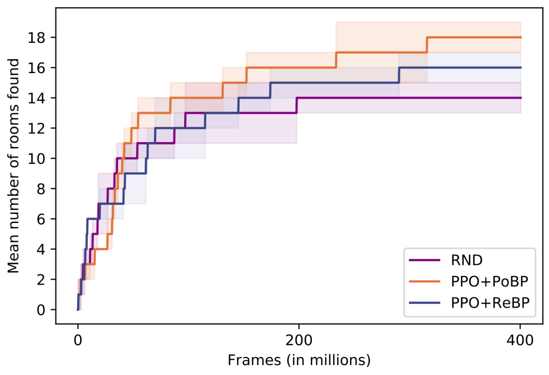


Figure 4.5: Number of rooms found during the exploration phase in Montezuma's Revenge. We run every method 10 times.

Table 4.3: Mean score of our method and baselines on Atari games. We report the results achieved over total 600M timesteps of training, averaged over 10 seeds.

| | Maximum Mean Score (at convergence) | | | | |
|---|---|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Gravitar | Pitfall | Seaquest |
| A2C [37] | 22 | 650 | 2,452 | -25 | 1,525 |
| PPO [38] | 2,475 | 98 | 3,438 | -41 | 1,785 |
| RND [20] | 8,654 | 9,026 | 4,621 | -2 | 3,280 |
| PPO+EC [21] | 7,022 | 6,254 | 3,521 | -20 | **3,485** |
| PPO+ICM [14] | 554 | 773 | 4,784 | -13 | 2,865 |
| PPO+GoCu [80] | 9,123 | 10,223 | 550 | -2 | 2,055 |
| DeepCS [76] | 3,500 | 1,105 | 881 | -186 | 3,343 |
| Average Human [77] | 4,753 | 69,571 | 3351 | 6,464 | 20,182 |
| PPO+PoBP(KL-divergence) | **9,478** | 10,876 | **5,112** | -1 | 2,874 |
| PPO+PoBP(JS-divergence) | 9,125 | **11,124** | 5,069 | **109** | 2,958 |
| PPO+ReBP | 8,934 | 8,598 | 4,915 | 0 | 3,165 |

### 4.4.4 Exploration With Sparse Extrinsic Rewards

We now report results in three sparse domains including Super Mario Bros (Figure 4.3a), Minigrid, and Atari games (Figure 4.4). We observe that the gap between our approach and the others is increasing with the degree of sparsity. On Montezuma's revenge PPO+PoBP achieves state of the art performance. It is likely caused by its ability to discover more rooms than RND model (Figure 4.5). Furthermore, when environments do not involve complex dynamics such as Minigrid, representation-based models can quickly achieve high-performance, whereas on more challenging games, policy-based models outperform other approaches. Please note that PPO+PoBP(KL-divergence) is less ideal than PPO+PoBP(JS-divergence) in tasks with deceptive rewards. It might be related to the fact that the former is more affected by a deterioration of the policy. In other tasks, the choice of distance measure has a limited impact on the performance.

Next, we further consider more exploration bonus as baselines. We conduct experiments on tasks featuring complex control dynamics, sparse extrinsic rewards, or deceptive rewards. We selected five hard exploration Atari 2600 games [86]: Montezuma's Revenge, Private Eye, Gravitar, Pitfall, and Seaquest. In these games, a poor exploration strategy often results in a suboptimal policy. The results are shown in Table 4.3. In 4 out 5 games, our agents consistently exceed the performance of baselines techniques. Precisely, it takes much less trial-and-error interactions to reach similar or superior scores than other curiosity-driven approaches. Moreover, on Montezuma's Revenge and Gravitar, they could surpass average human performance as well as the previous state of the art. On Seaquest our model's performance is comparable to that of the state of the art. We further observe that on Pitfall most algorithms fail to find any positive rewards, which entails that they cannot reinforce their policy. On the other hand, by using policy-based learning progress combined with JS-divergence, PPO+PoBP could achieve a positive score. These results demonstrate that the depicted methods achieve much greater sample-efficiency and are more robust than prior work to local sources of entropy in the environment.

Table 4.4: Average success rate on tasks from the Minigrid domain with dense and sparse settings. The results are averaged over 10 runs after 15 millions training steps.

| | Sparse | | Dense | |
|---|---|---|---|---|
| Method | MultiRoomN10 | Door&Key 16×16 | MultiRoomN10 | Door&Key 16×16 |
| PPO | 0.2±2.4 | 0.5±3.5 | 14±2.9 | 47±2.2 |
| PPO+PoBP(KL-divergence) | 78±0.3 | 92±4.8 | 75± 0.8 | 87 ± 4.6 |
| PPO+PoBP(JS-divergence) | **81±0.1** | **94±5.1** | **77 ± 0.4** | **90 ± 3.9** |
| PPO+ReBP | 77± 0.2 | **93 ± 4.6** | 72 ± 1.2 | 86 ± 4.1 |

### 4.4.5 Dense Rewards

An important characteristic of a good curiosity bonus is to avoid hurting performance in dense-reward tasks. We test this setting in MultiRoomN10 and Door&Key 16×16. In the sparse setting, the agent solely receives a (sparse) terminal reward of +1 if it reach the target and 0 otherwise. In the dense setting, the agent is rewarded for picking keys (+0.3) and opening doors (+0.3), as well as reaching the goal (+1). The results highlight (Table 4.4) that the depicted methods do not significantly deteriorate performance in dense reward tasks. Even though PPO+PoBP and PPO+ReBP perform slightly worse in the dense setting, they still greatly improve performance as compared to plain PPO.

### 4.4.6 No Extrinsic Reward

For testing the good exploration coverage of our method, we trained our agent without any extrinsic reward from the environment. Our agent only receives a curiosity-based signal to reinforce its policy. We evaluate our approach in the Super Mario Bros environment [87]. Initially, this game is played using a joystick which requires pressing simultaneously multiple buttons. In our implementation, each combination of buttons is mapped to a unique action resulting in 12 possible actions. As can be seen in Figure 4.3b, in order to remain curious the agent is pushed to explore distant regions of the state space, which entails that the overall state coverage increases over time. It highlights that in the absence of extrinsic rewards, our method provides indirect supervision for learning meaningful and diverse behaviors.

## 4.5 Discussion

In this chapter, we employed *learning progress* as a central theme in building sample-efficient agents that can explore the environment on their own from just raw sensory data. We proposed to measure the agent's learning progress in terms of *quantity* and *quality* at different time-scales on a batch of observations. We compared and evaluated two methods to estimate the agent's progress: 1) policy-based progress, 2) representation-based progress. We further contributed a method to give more attention to hard-to-learn regions of the state space - the *episodic-skill* module encourages the exploration of state-action trajectories carrying information about many skills.

We have experimentally demonstrated that our agents trained with a curiosity reward are able to learn useful behaviors in tasks featuring sparse and/or deceptive rewards, and stochastic dynamics. This method allows us to overcome the known *vanishing* curiosity issues of prior work and outperform previous state-of-the-art curiosity-driven methods on robotic, maze navigation, and complex game-playing tasks. Moreover, the experiments highlight that in the absence of extrinsic rewards, our method provides enough indirect supervision for learning useful behaviors. Finally, we demonstrated that our agent is capable of maintaining exploration in a meaningful way throughout the agent's training process, escaping from sub-optimal policies. Hence, improving the capability to make intrinsically motivated choices has resulted in greater efficiency in terms of training time and performance.

A limitation of our proposed as well as prior approaches is that they do not explicitly encourage deep exploration behaviors. We found that these methods are sufficient to deal with local exploration - exploring the consequences of short-term decisions, such as how to pass an obstacle or interact with an object. Nonetheless, achieving deep exploration - the future usefulness of decisions within some temporally-extended horizon, is beyond the reach of most curiosity-driven methods. In detail, the agent should receive enough intrinsic motivation for trying deep exploration strategies potentially leading to novel learning opportunities later.

In the next chapter, we present a formulation for exploration inspired by developmental sciences that explicitly favors deep exploration behaviors while maintaining consistent exploration both within an episode and across episodes, escaping the vanishing curiosity issue presented in this chapter.

# Chapter 5

# Fast and Slow Curiosity for High-Level Exploration

In the above chapters, we discussed different strategies for motivating the agent's exploration in sparse reward environments, including those featuring high-dimensional observations, challenging motion features, and stochastic dynamics. In retrospect, we concluded that even though these techniques lead to drastic reductions in both amounts of required interactions and cost of exploration, explicitly promoting deep exploration behaviors remains an open question. This capability is critical for solving complex real-world problems.

In this chapter, we present an approach that explicitly encourages deep exploration behaviors by maintaining exploration throughout the agent's training process and providing enough intrinsic reward for escaping from local optima. To do so, we introduce a novel formulation of curiosity that can capture meaningful visual features and salient environmental dynamics at different scales (e.g. character-level or word-level); and then build an algorithmic framework that combines two curiosity bonus components. We further derive a strategy for lifelong learning and the better incorporation of environmental motions.

## 5.1 Introduction

The problem of intrinsically-motivated *deep* exploration remains one of the major challenges in deep reinforcement learning. Several works attempt to tackle this challenge by providing a new intrinsic exploration bonus (i.e. curiosity) to the agent. For example, count-based exploration [18] keeps visit counts for states and favors the exploration of states rarely visited. Another class of methods relies on predicting motion dynamics of the environment [19]. For instance, ICM [14] predicts the feature representation of the next state based on the current state and the action taken by the agent. Nevertheless, maximizing the prediction error tends to attract the agent to stochastic transitions, where the consequences of actions are hardly predictable [20]. This issue has motivated several recent works [20, 21, 78]. Despite their ability to deal with *local exploration* - exploring the consequences of short-term decisions (e.g. how to pass an obstacle or whether to interact with a particular object), global exploration remains an open problem. *Global exploration*, also referred to as "deep exploration", considers the future usefulness of decisions within some temporally-extended horizon. Therefore, these methods generally struggle in tasks wherein a large enough intrinsic reward should be given to the agent to discover long-time horizon strategies and avoid local optima.

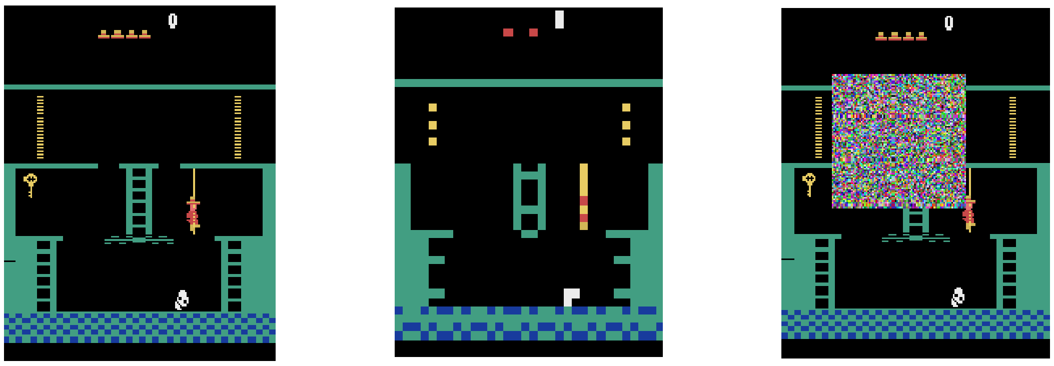

(a) Original Frame (b) Downsample Context (c) Noisy Context

Figure 5.1: Examples of contexts created from one frame (left column) of the Montezuma's Revenge environment. Middle column: downsample context. Right column: noisy context.

Some recent works have suggested that what makes human learners so efficient at learning is the brain's computational capacities to act over multiple spatial-temporal scales [88, 89], that is, humans employ fast and slow forms of learning. One natural question that arises is: how these principles relate to exploration and how can we replicate them to improve exploration? Inspired by this observation, we present Fast and Slow intrinsic curiosity (FaSo) that can deal with high-level exploration, the combination of fast and slow dominant exploration phases. We postulate that to efficiently explore its environment, the agent should combine a short-time and long-time horizon strategy. To this end, the proposed method decomposes the curiosity bonus into a fast and a slow intrinsic reward. Fast rewards deal with local exploration by rapidly adapting to evaluate the novelty of states. In contrast with fast rewards, slow rewards change slowly and remain large to explicitly encourage the exploration of distant or hard-to-reach regions of the state space (global exploration). We formulate intrinsic rewards as the reconstruction errors of the observations given their *contexts*. Namely, a *reconstructor* network takes an observation with missing or corrupted regions, the observation's context (Figure 5.1), and attempts to reconstruct the original image. We show that having two types of reconstruction tasks (i.e. fast model and slow model) can lead to different exploration behaviors characterized by different time horizons. A key difference between these models lies in their context creation strategies. Moreover, using noisy inputs forces the model to capture meaningful visual features and salient environmental dynamics at different scales (e.g. character-level or word-level). We further propose an adaptive scaling technique to modulate fast and slow rewards by measuring state diversity. Please note that since reconstruction tasks are noisy by nature, we found that our model is not affected by local sources of entropy in the environment.

We then derive a second model (IML) based on FaSo for lifelong learning and the better incorporation of environmental motions. We develop and show how lifelong learning can be achieved by modulating local rewards with deep rewards to maintain consistent exploration both within an episode and across episodes. We further incorporate the agent's actions in the reconstruction tasks to capture motion dynamics.

In summary, our main contribution is a high-level exploration mechanism relying on fast and slow rewards, which can scale to problems with complex visual observations and that is applicable to most on-policy algorithms. We propose as curiosity reward the reconstruction errors of the observations given their contexts using two different architectures of deep neural network inspired by auto-encoders. Furthermore, we trade-off local with global exploration strategies by estimating state diversity. We evaluate the performance of our method in a variety of sparse reward environments, including Minigrid, Super Mario Bros, and Atari games. We demonstrate that FaSo is preferable to the previous intrinsic reward methods in terms of exploration efficiency, especially in environments featuring sparse rewards. Experimental results also show that high-level exploration is crucial when the environment contains deceptive and extremely sparse rewards such as in Pitfall (Atari). We also demonstrate how lifelong learning can improve sample-efficiency on domains featuring very long delay between the initial state and goal states. Finally, our experiments show that incorporating curiosity can lead to dramatic improvements in performance compared to pure RL.

## 5.2 Related Work

Our work is also related to encouraging exploration in reinforcement learning. Most approaches can be grouped into three classes: count-based strategies, goal conditioned learning, and curiosity-driven exploration. This section provides a comprehensive comparison with those researches.

One line of work is to keep visit counts for states to favor exploration of rarely visited states [18, 55, 82]. To apply count-based exploration to continuous state spaces, a solution [56] is to train an observation density model to supply counts. Another solution [83] maps states to hash codes and counts state visitations with a hash table. In order to reduce the size of the count table, a method clusters states and keeps visit counts of clusters instead of the original states [90]. A prior work [91] introduces a count-based optimistic algorithm by estimating the uncertainty associated with each state. However, one can expect these methods to be less effective when some *valuable* states require more attention - more visits. In this setting an agent can visit a less frequently visited state many times, even though the value in this state is already estimated. On the other hand, the proposed curiosity formulation is based on the agent's understanding of the world. Hence, states easy to reconstruct will be less novel than states featuring sophisticated visual patterns.

Goal conditioned learning [39] motivates an agent to explore by constructing a goal-conditioned policy, and then optimize the rewards with respect to a goal distribution. While intrinsically motivated agents can easily get trapped in local optima, goal conditioned methodologies aim to maximally cover a behavioral goal space, acting as a "global" exploration strategy. For instance, universal value function approximators (UVFA) [40] construct a set of optimal value functions by a single function approximator that can generalize over both states and goals. The recent work on hindsight experience replay (HER) [41] forms an implicit curriculum by using visited states as a target. However, selecting relevant goals is not easy. A class of work [67] and its recent follow-up [81, 61], proposed to generate increasingly difficult goals to provide additional feed-back during exploration. Rather than operating directly on observations, the approach [92] learns an embedding for the goal space using unsupervised learning and then chooses the goals from that space. Another line of work [93] improves exploration by focusing on goals that provide maximal learning progress. The recent work [94], Skew-fit, proposes an exploration objective that maximizes state diversity. In particular, they use the density in a VAE latent space to model state diversity. The key idea is to learn a maximum-entropy goal distribution to match the weighted empirical distribution, where the rare states receive larger weights. In contrast, we assume a fixed goal and train a VAE to reconstruct the original image given a noisy input. Our method also works in the latent space to estimate state diversity, but state diversity is used to adjust the scaling factors of intrinsic rewards. In a similar fashion, option discovery methods [95, 96, 97] use options as sub-goals in order to add high-level supervision during training. For instance, FeUdal network [98] employs a manager and low-level policy. The manager policy guides the low-level policy by sending sub-goal signals that are not explicitly defined.

This work belongs to the category of techniques that consider curiosity as a drive to explore. Some methods [14, 54] rely on predicting environment dynamics using an inverse or forward dynamic model. Another class of approaches uses prediction errors in the feature space as measure of the importance of states [99]. For instance, RND [20] predicts the output of a randomly initialized neural network on the current state. In a different fashion, learning progress was shown to be an effective exploration bonus to escape from local optima and avoid premature convergence to sub-optimal policies [78]. In contrast, our work introduces the idea of reward decomposition to achieve flexible exploration behaviors and explicitly encourage deep exploration. Another crucial difference here is that each reward stream is independently calculated, by taking advantage of context-based features. A recent solution [100] aims to leverage motion features in the observations. They utilize the errors from a forward and backward optical flow estimation to asses the novelty of observations. In [80], the authors formulate curiosity as the ability of the agent to achieve a set of goals. By doing so, they incentivize the exploration of hard-to-learn states. Introducing a new term in the loss function that attemps to maximize state diversity was used to deal with local exploration [84]. Learning without extrinsic rewards has also been studied and is referred to as *novelty search* [101] - an evolutionary algorithm designed to escape from local optima by defining selection pressure in terms of behavior. The novelty of an event is estimated as the distance of the current behavior features to the *k-nearest* neighbors in a memory of behavior features. Episodic curiosity through reachability [21] addresses the "noisy TV" problem by considering the number of time-steps between two states as curiosity measure. Exploration bonus can also be based on maximizing information gain about the agent's knowledge of the environment [79]. As a step towards addressing this problem, an algorithm [85] guides exploration by maximizing an entropy objective. In this work, our formulation of curiosity is a deterministic problem which not only deals with local exploration but aims to introduce high-level exploration in reinforcement learning.

## 5.3 Fast and Slow Exploration

The main objective of the proposed fast and slow driven exploration (FaSo) method is to encourage high-level exploration. We focus on tasks where extrinsic rewards $r_t^e$ are sparse; zero for most of the time steps $t$. In addition to this reward, our method produces an intrinsic curiosity-based bonus $r_t^i$ that can be decomposed into a fast reward $r_t^{fast}$ and a slow reward $r_t^{slow}$. The fast reward $r_t^{fast}$ deals with local exploration by rapidly changing

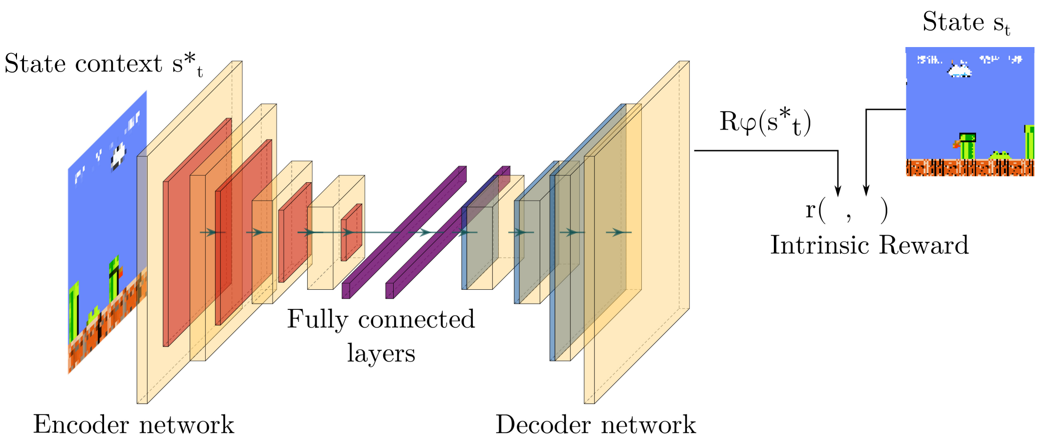


Figure 5.2: The reconstructor $R_\theta$ architecture. The image's context is passed through an encoder network and the features all around the feature map are connected using fully connected layers. The reconstruction-based curiosity reward is calculated based on the discrepancy between the original image and the reconstructed image.

over time based on the novelty of the current observation. The slow reward $r_t^{slow}$ acts as a global exploration bonus to explicitly encourage deep exploration behaviors which would move the agent towards regions of the state-action space that are novel or hard to reach. $r_t^{fast}$ and $r_t^{slow}$ are assigned by using the idea of image reconstruction error given an observation's *contexts*. Formally, we consider an agent interacting with an environment; at each time step $t$ the agent performs an action and receives an augmented reward:

$$r_t = r_t^e + r_t^i = r_t^e + \left[\alpha r_t^{fast} + \beta r_t^{slow}\right] \tag{5.1}$$

where $\alpha$ and $\beta$ are hyperparameters to weight the importance of both rewards. The policy $\pi(s_t; \theta_P)$ is represented by a deep neural network with parameters $\theta_P$. Its parameters are optimized to maximize the excepted sum of these two rewards:

$$\max_{\theta_P} \mathbb{E}_{\pi(s_t;\theta_P)}\left[\sum_t r_t\right] \tag{5.2}$$

In the following section, we describe in detail the key components. First, we present how an intrinsic reward signal is generated based on reconstruction error given observation's context. Second, we introduce two neural network models to reconstruct noisy images. Finally, we formalize how fast and slow reconstruction-based rewards are combined to achieve high-level exploration.

### 5.3.1 Reconstruction-Based Curiosity

We propose the concept of reconstruction-based curiosity as our novelty measurement scheme (Figure 5.2). At every step, the module takes the *context* of the current observation $s^*$ as its input, and reconstructs the original image $s$. In this work, the *context* of an observation refers to a version of it with one or more noisy, or corrupted regions. The pixels can be recovered by propagating local features and capturing the overall structure of the context. The discrepancy between the reconstructed image and the actual image then serves as the intrinsic reward.

We found that reconstructing an observation given its context is more robust to random perturbations or small changes in the environment, and helps to capture important visual features (see Section 5.3.1). Moreover, our formulation bypasses the need of predicting the next observation given the current observation and the agent's action [14], which tends to attract the agent to stochastic transitions to maximize the prediction error [15]. In contrast with such stochastic approaches, we propose a curiosity measure where the reconstruction is a deterministic problem.

### Context Creation

We introduce two methods to artificially extract the *context* of an observation that we present below.

- Downsample Context: The original image of size $w \times w$ is downscaled to a smaller image of size $\frac{w}{K} \times \frac{w}{K}$ using a nearest-neighbor interpolation [102] and then upscaled to the original size, introducing small artifacts. The hyperparameter $K$ controls the amount of artifacts. As an intuition, it can be viewed as a simplified version of the image.
- Noisy Context: The observation is augmented with a region of white noise which makes $K \times K$ pixels. The white noise region position changes for each observation to improve robustness. Theoretically, randomly choosing the position would be a good choice, however, in practice this solution is less than ideal in some cases (e.g. the region falls on the background). Empirically, we found that a random position within a radius $K/2$ from the center of the frame works well as a robust substitute.

An example is shown in Figure 5.1. When using downsample context creation, the reconstructor network attempts to reconstruct the detailed content of the original frame (Figure 5.1a) given its blurry / corrupted context (Figure 5.1b). When using noisy context creation, the reconstructor network aims to infer the original pixel values of a large noisy region (Figure 5.1a) based on the context of the surrounding pixels (Figure 5.1c).

### Reward Calculation

In this section we formulate the procedure to calculate reconstruction-based intrinsic reward. This involves the prediction error of a *reconstructor* network trained to reconstruct an observation given as input the observation's context.

Formally, let $s_t$ be the original observation at time $t$ and $s_t^*$ its context. The reconstructor network, $R_\theta : s^* \mapsto s$, takes the context of the observation and reconstructs the original image $s_t$. We denote the reconstructed image as $\hat{s}_t$. The parametrization of $R_\theta$ is discussed further in the next section. The reconstruction process can be formulated as:

$$\hat{s}_t = R_\theta(s_t^*) \tag{5.3}$$

This reconstruction will have some errors that can be measured using a distance function such as the euclidean distance. However, empirically we found that the euclidean distance is ineffective to distinguish noisy data (such as contexts) from the original images. Instead, we propose to embrace SSIM metric [103], a popular technique used in the field of computer vision for comparing the structural similarity of two images. Our novelty measure is based on the single-scale SSIM metric that compares corresponding pixels and their neighborhoods in two images with three metrics: luminance, contrast, and structure. The reconstruction error $e(s_t, \hat{s}_t)$ can therefore be expressed via:

$$e(s_t, \hat{s}_t) = 1.0 - \left[ \frac{1}{P} \sum_{i=1}^{P} L(s_t^i, \hat{s}_t^i)\Gamma(s_t^i, \hat{s}_t^i)S(s_t^i, \hat{s}_t^i) \right] \tag{5.4}$$

where $s_t^i$ and $\hat{s}_t^i$ are the $i^{th}$ sliding patch over $s_t$ and $\hat{s}_t$, respectively, and $P$ is the number of patchs per image. The luminance (L), contrast ($\Gamma$), and structure (S) can be computed as follows:

$$L(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \; \Gamma(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \; S(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{5.5}$$

where $\mu_x$, $\mu_y$, $\sigma_x$, $\sigma_y$ denote the mean pixel intensity and the standard deviations of pixel intensity of two image patches $x$ and $y$. Following this work [103], we use a square neighborhood of $5 \times 5$ pixels resulting in patches of size $11 \times 11$ pixels. $\sigma_{xy}$ is the sample correlation coefficient between corresponding pixels in the two patches and $C_1$, $C_2$, $C_3$ are small constants for numerical stability.

When using reconstruction error as intrinsic reward, the reward function is non-stationary and its scale can fluctuate greatly between different points in time. In order to keep a consistent scale of the intrinsic rewards, it is useful to normalize them. This can be achieved by dividing the intrinsic rewards by a running estimate of the standard deviations of the sum of discounted intrinsic rewards $\mathcal{R}_{ib}$ [15]. We can now assign a curiosity bonus $r^{ib}$ as:

$$r^{ib}(s_t) = \left[ \frac{e(s_t, R_\theta(s_t^*))}{\sigma(\mathcal{R}_{ib})} \right] \tag{5.6}$$

This approach is motivated by the idea that, as our ability to model the dynamics and salient features of a particular state improves, the agent has a better understanding and hence the intrinsic motivation is lower. We show in Section 5.3.3 how the reconstruction-based curiosity method can be used for estimating fast and slow rewards.

### 5.3.2 Image Reconstruction Architecture

In this section we consider the task of reconstructing an observation given its context. We propose two different neural network architectures inspired by autoencoders (denoted by Cb-AE and Cb-VAE) to learn $\hat{s} = R_\theta(s^*)$. They both consist of a number of convolutional and de-convolutional layers for reconstructing the original image.

### Context-Based Autoencoder

Given the context of an observation, the most direct way to reconstruct the original observation is to train an autoencoder (AE). However, autoencoder architectures cannot propagate information from one part of the feature map to another. This is because the encoder network directly feeds feature map to the decoder network and convolutional layers never connect all the locations together within a feature map. To meet this need, we introduce multiple fully connected layers between the encoder network and the decoder network (Cb-AE)(Figure 5.2) like done in [104]. By connecting the features altogether, we can expect to enable the propagation of information for all locations across the image and hence improve image reconstruction. As an intuition, when a pixel is missing or corrupted, only using the pixel's neighborhoods only enables to capture local geometric features, whereas our architecture (Cb-AE) can capture the general appearance structure of images. Therefore, pixels are reconstructed to make the overall prediction look more real. Please note that unlike autoencoders, the input image is different from the target image that alleviates the need to have a small middle layer.

During training, we further propose a novel context-based loss function $\mathcal{L}_{cb}$ that captures the overall structure of the observations. The objective is to favor the reconstruction of regions that are noisy or corrupted, and hence require more attention. Given the context $s^*$ of a state $s$, the reconstructor network $R_\theta$ generates an output $R_\theta(s^*)$ and is trained to minimize the following loss function:

$$\mathcal{L}_{cb}(s, s^*) = \|(s - R_\theta(s^*)) \odot (1 - N)\|_2 \tag{5.7}$$

where $N$ is a mask with values between 0 and 1 of the same size as the reconstructed image. The mask is automatically generated based on the type of context. When using the *downsample* context, all the pixels are corrupted and hence the mask is filled with 0. When the image is filled with white noise (i.e. *noisy* context), the mask is filled with +1 for the pixels not corrupted and 0 for the corresponding pixels within the noisy area.

### Context-Based Variational Autoencoder

We propose context-based variational autoencoder (Cb-VAE) inspired by variational autoencoders (VAEs). VAEs are known to give representations with disentangled factors [105] due to gaussian priors on the latent variables. They have been shown to be more scalable to large datasets and to capture better representations than AEs. Cb-VAE resembles a standard autoencoder, using an encoder network $q_\phi$ that outputs a latent variable $z$ given the image's context $s^*$, and then a decoder network $p_\psi$ that maps $z$ to the original image $s$. Its parameters are optimized by maximizing a variational lower bound on the likelihood function:

$$\mathcal{L}(\psi, \phi; s, s^*) = \tau D_{KL}(q_\phi(z|s^*)||p(z)) - \mathbb{E}_{q_\phi(z|s)}\left[\log p_\psi(s|z)\right] \tag{5.8}$$

where the first term is a regularizer, the Kullback-Leibler divergence between the encoder distribution $q_\phi(z|s^*)$ and $p(z)$, and the second term is equivalent to a $l_2$ loss. However, VAEs suffer from the effect of $l_2$ loss and therefore are prone to generate blurry images.

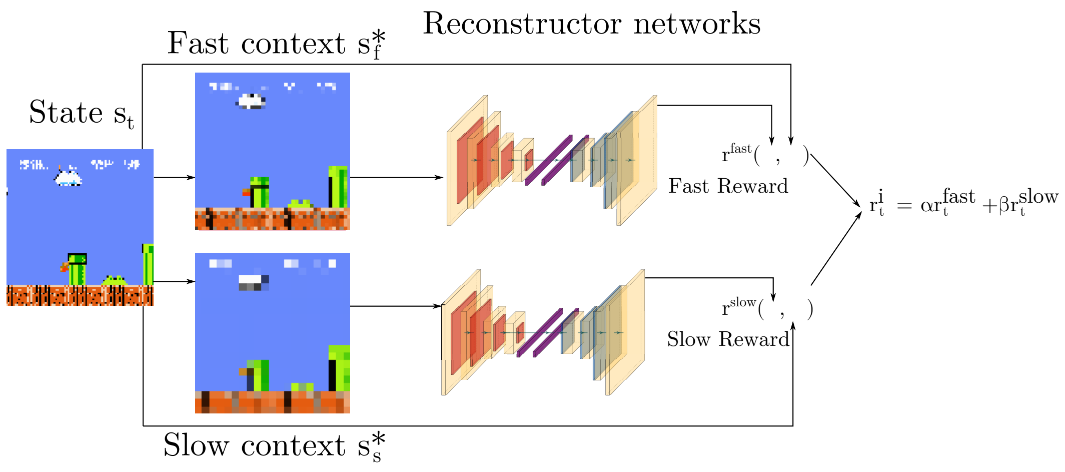


Figure 5.3: Fast and slow exploration model architecture.

We observe that it can be written as:

$$\mathcal{L}(\psi, \phi; s, s^*) = \tau D_{KL}(q_\phi(z|s^*)||p(z)) + \frac{1}{2\sigma^2}\sum_{i=1}^{N}(s^i - f_\psi(z^{(i)}))^2 \tag{5.9}$$

where $f_\psi(z^{(i)})$ is computed by the decoder network. Thus, we can rewrite this loss function to integrate any differential loss $g(s, \hat{s})$ that better captures small details and generates more sharp images (see Section 5.4.1). The loss function that Cb-VAE optimizes then becomes:

$$\mathcal{L}(\psi, \phi; s, s^*) = \tau D_{KL}(q_\phi(z|s^*)||p(z)) + \rho \cdot \mathbb{E}_{q_\phi(z|s)}[g(s, \hat{s})] \tag{5.10}$$

where $\tau$ and $\rho$ are scalars to weight the two components of the loss function, and $\hat{s} = f_\psi(z)$. Empirically, we found that minimizing the loss related to the sum of structural similarity scores works well as a robust substitute to $l_2$ [106]:

$$g(s, \hat{s}) = -\sum_i \text{MS-SSIM}(s_i, \hat{s}_i) \tag{5.11}$$

where $i$ is an index over scales and MS-SSIM is the multiscale SSIM [107]. While $l_2$ ignores the intricate characteristics of the human visual system, MS-SSIM is sensitive to changes in local structures and performs well on real-world images with different scales. For training the MS-SSIM loss, we use 5 scales.

Similarly to Cb-AE, we further modified the original architecture by adding multiple fully connected layers to connect features altogether.

### 5.3.3 Combining Fast and Slow Rewards

In this section we provide an algorithm built upon reconstruction-based curiosity to deal with high-level exploration. Although a reconstruction-based curiosity bonus can improve local exploration such as how to interact with a particular object or avoid a collision; global exploration is beyond the reach of a single curiosity-based reward. We found that the agent may get stuck in local optima or not receive enough intrinsic reward to promote long-time

**Algorithm 2** Fast and Slow intrinsic curiosity (FaSo)

```
1:  Given:
      • an on-policy RL algorithm $\pi_{\theta_P}$                                ▷ PPO, A2C
      • a replay buffer $R$
      • a fast context buffer $\Omega_f$ and a slow context buffer $\Omega_s$
      • a fast model $R_{\theta_f}$ and a slow $R_{\theta_s}$ reconstructor model
      • a context creation strategy                                  ▷ Noisy, Downsample
      • context creation parameters $K_{fast}$ and $K_{slow}$
2:  Initialize $\pi_{\theta_P}$, $R_{\theta_f}$ and $R_{\theta_s}$                  ▷ initialize neural networks
3:  Initialize $R = \{\}$, $\Omega_f = \{\}$, and $\Omega_s = \{\}$
4:  Initialize $\alpha = 0.5$ and $\beta = 0.5$
5:  for m=0,...,M do
6:      for t=0,...,H-1 do
7:          Get action $a_t = \pi(s_t|\theta_P)$
8:          Execute $a_t$ and observe next state $s_{t+1}$
9:          Create fast $s_f^*$ and slow contexts $s_s^*$ given the current observation
10:         Calculate intrinsic reward $r_t^i = \alpha \left[\frac{e(s_t,R_{\theta_f}(s_f^*))}{\sigma(\mathcal{R}_{ib}^f)}\right] + \beta \left[\frac{e(s_t,R_{\theta_s}(s_s^*))}{\sigma(\mathcal{R}_{ib}^s)}\right]$
11:         Store transition $R = R \cup (s_t, a_t, s_{t+1}, r_t^e, r_t^i)$
12:         Store $s_f^*$ in $\Omega_f$ and $s_s^*$ in $\Omega_s$
13:         Update reward normalization parameters using $i_t$
14:     Normalize the intrinsic rewards contained in $R$
15:     Update the RL model on $R$
16:     Compute state diversity on $\Omega_f$, and $\Omega_s$
17:     Assign $\alpha$ and $\beta$ based on state diversity progress at time m and m+1
18:     Periodically fine-tune $R_{\theta_f}$
19:     Periodically fine-tune $R_{\theta_s}$
```

horizons strategies. This can happen after the novelty of a state has vanished, the agent is not encouraged to visit it again, regardless of its importance on a long-time horizon. To build intuition we consider a task wherein a robot has to reach a target location behind a door. There are multiple levers but only the combination of two of them can unlock this door. To use the correct levers, the agent must give up immediate rewards associated with easy-to-reach levers and explore more distant areas. Solving such a task requires a long-term exploration bonus to escape from sub-optimal policies by compensating the loss of easy immediate extrinsic reward (i.e. keep the key) and incentivizing useful decisions on a long-time horizon. One solution to the vanishing-curiosity issue could be to decrease the learning rate of the reconstructor network, however, it tends to make the agent's learning slow as the curiosity continuously encourages the revisit of familiar states (not all regions need to be revisited).

This work introduces a different approach where the intrinsic bonus $r_t^i$ is the combination of two distinct reconstruction-based rewards:

$$r_t^i = \alpha r_t^{fast} + \beta r_t^{slow} \tag{5.12}$$

where $r_t^{fast}$ deals with local exploration and $r_t^{slow}$ is the slow reconstruction-based reward that aims to push the agent to explore long-time horizon strategies. In detail, the fast context-based model quickly learns to reconstruct observations to assess the short-term novelty of each state. On the other hand, $r_t^{slow}$ changes slowly and remains large to encourage deep exploration. The scalars $\alpha$ and $\beta$ weight the fast and slow reconstruction-based rewards.

We now explain how $r^{fast}$ and $r^{slow}$ are calculated. They are estimated by two distinct reconstruction-based curiosity models which reconstruct the original observation $s$ given its fast $s_f^*$ and slow $s_s^*$ contexts respectively, $R_{\theta_f} : s_f^* \mapsto s$ and $R_{\theta_s} : s_s^* \mapsto s$ (see Figure 5.3). They are parameterized by a set of trainable parameters $\theta_f$ and $\theta_s$. The key difference is how to generate frame contexts, $s_f^*$ and $s_s^*$, to achieve exploration behaviors with different ranges of time horizons. Let $K_{fast}$ the parameter used to create the fast contexts, we typically used $K_{fast} \times 2$ or $K_{fast} \times 4$ to create the slow contexts - slow contexts are more corrupted. Slow reward acts as a global exploration mechanism in two ways. 1) As the reconstruction task becomes more challenging when large regions of images are noisy or corrupted (i.e. slow contexts), slow reward remains large even with further state visitations. 2) The reconstruction error is maximized only in states that induce a significant shift in the state distribution (e.g. visiting a new room), driving the agent to seek out such richer and novel regions (i.e. global exploration). Please note that since slow contexts have a more blurry representation of states, thus, the reconstructor network will not see fine-grained local differences and will drive the agent to further explore novel regions. In contrast, contexts of fast rewards are nearly unique which entails that slightly deviating from previous policies - visiting novel states, or revisiting surprising states is sufficient to significantly increase the reconstruction errors; encouraging to locally explore the environment. Therefore, such a reward enables fast learning by encouraging local exploration but swiftly downmodulates states that become more familiar across episodes.

Thus, the overall intrinsic reward provided by FaSo is calculated as:

$$r_t^i = \alpha \left[ \frac{e(s_t, R_{\theta_f}(s_f^*))}{\sigma(\mathcal{R}_{ib}^f)} \right] + \beta \left[ \frac{e(s_t, R_{\theta_s}(s_s^*))}{\sigma(\mathcal{R}_{ib}^s)} \right] \tag{5.13}$$

where $e$ is the reconstruction error defined in Equation 5.4. Algorithm 2 provides an outline of the basic training loop.

One problem is how to select $\alpha$ and $\beta$ to modulate local and global exploration strategies. We found that a fixed value can be difficult to tune and not ideal. Instead, we propose an adaptive solution to weight the fast and slow intrinsic rewards based on the idea of *state diversity*. At the end of each epoch $\alpha$ and $\beta$ are selected. We compared two methodologies depending of the choice of the reconstructor network architecture:

1. **Context-based autoencoder (Cb-AE):** state diversity can be estimated by maintaining an episodic memory of the last experienced contexts, and measuring diversity progress between the memory and the memory augmented with the new observation's contexts. Let $\Omega = \{s_0^*, s_1^*, ..., s_N^*\}$ an episodic memory of the last contexts. We can estimate pairwise dissimilarities $\overline{d}$ among the contexts as follows:

$$\overline{d} = \frac{1}{|\Omega|(|\Omega|-1)} \sum_{i=0}^{N} \sum_{\substack{j=0 \\ i \neq j}}^{N} ||s_i^* - s_j^*||_2 \tag{5.14}$$

State diversity progress can be measured as the difference between pairwise dissimilarities at time $t+1$ and $t$.

$$d = \frac{\overline{d}_{t+1} - \overline{d}_t}{\sigma(D)} \tag{5.15}$$

where $\sigma(D)$ is an estimation of the standard deviations of the state diversity progresses. We then clip $d$ to be between 0 and 1 and the new experience are added to the memory bank. When the capacity is exceeded a random element is substituted in memory with the current element. The scaling factors $\alpha$ and $\beta$ are independently calculated based on their dedicated episodic memory and their value is equal to respectively $d_{fast}$ and $d_{slow}$.

2. **Context-based variational autoencoder (Cb-VAE):** we indirectly estimate the state diversity by measuring the quality of the Cb-VAE for encoding the latent representations in the data. We assume that an observation's context $s^*$ is generated by a random latent process $z$. Learning progress can be measured by measuring the distance between the posterior distribution $p(z|s^*)$ after experiencing new observation' contexts and the prior $p(z)$. Several measures such as KL divergence can be used. However, computing the posterior distribution $p(z|s^*)$ is often intractable. Instead, we propose to use Cb-VAE to model the approximate posterior $q_\phi(z|s^*)$.

   Let $\Omega = \{s_0^*, s_1^*, ..., s_N^*\}$ a set of observation's contexts, we define the state diversity for the fast and slow models as:

$$d = 1.0 - \left[ \frac{1}{N} \sum_{i=0}^{N} D_{KL}(q_\phi(z|s^{*(i)})||p(z)) \right] \tag{5.16}$$

   To ensure state-diversity measure to adapt over time, $\Omega$ is filled with contexts experienced during the last episodes and contexts collected by executing a random policy. The diversity progress is independently measured for fast and slow rewards, $\alpha = d_{fast}$, $\beta = d_{slow}$ respectively; and we clip the values to be within [0.1,1.0]. Since state-diversity greatly decreases when more states become familiar, it is useful to normalize $\alpha$ and $\beta$ by diving the state diversity by a running estimate of the standard deviations of the state diversity. By doing so, the agent seeks out to explore states as diverse as possible - this increases the distance between $p(z|s^*)$ and $p(z)$ in average, to receive large intrinsic rewards induced by high values of $\alpha$ and $\beta$.

## 5.4 Experiments

In this section, we first evaluate FaSo and present the experimental results on a variety of environments including Minigrid, Super Mario Bros, and Atari games. In Minigrid, we consider three types of hard exploration tasks: Door & Key, KeyCorridor, and Multiroom. We first present an ablation analysis of FaSo. Second, we test FaSo trained on a fixed and randomly generated mazes (Multiroom). Third, we evaluate the performance of our model on Super Mario Bros in the absence of any extrinsic reward signal. Fourth, we measure the performance of FaSo on dense reward environments. Finally, we compare the proposed algorithm with the previous curiosity-based approaches on five Atari games, Door & Key, and Super Mario Bros, combining intrinsic rewards with sparse or deceptive extrinsic rewards.

**Implementation Details (FaSo)**

In all the experiments the observations are given in the form of images. The RGB images are converted to 84×84 grayscale images. The input given to the policy network consists of the current observation concatenated with the previous three frames. We set the end of an episode to when the game ends. As our policy learning method, we use PPO with similar hyperparameters as in the original implementation of RND [20], but 32 learners. The output of the last convolutional layer is fed into a fully connected layer with 256 units. It is followed by two separate fully connected layers of size 448, used to predict the value function of each reward component (extrinsic and intrinsic advantage).

In order to find the hyperparameters for our method, we ran grid searches over the context creation techniques as well as $K_{fast}$ and $K_{slow}$, which control the degree of fast and slow contexts. On Minigrid, we select Cb-AE as the reconstructor network architecture and *downsample* contexts created with $K_{fast} = 2$ and $K_{slow} = 4$. On Super Mario Bros, we train a Cb-VAE to reconstruct the observations given their *downsample* contexts ($K_{fast} = 2$, $K_{slow} = 5$). For Atari games, we compare our method trained with *noisy* contexts ($K_{fast} = 24$, $K_{slow} = 46$) with baselines. We set the coefficients $\tau = 0.5$ and $\rho = 0.5$. The value of the memory size, $N$, is 80 for FaSo(Cb-AE), and 200 for FaSo(Cb-VAE). We estimate the sum of discounted intrinsic rewards using a discount factor of 0.99 on rollouts of length 128. $\sigma(\mathcal{R}_{ib})$ (Eq 5.6) is defined as the running average of the standard deviations of these discounted intrinsic rewards. The architectures of Cb-AE and Cb-VAE consist of a sequence of four convolutional layers with 32 filters each, stride: 2,1,1, kernel size of 3×3, and padding 1. We apply a rectifier non-linearity after each convolutional layer. The output of the last convolutional layer is passed to a serie of two fully connected layers of size 256. The last layers are the corresponding decoding layers. For online training of the reconstructor networks, we store the experience and make 5 epochs of training every 16K time steps. We found that retraining the slow reconstructor network $R_{\theta_s}$ once every 48K steps is sufficient. Note that we decouple the training speed of the reconstructor models to ensure that the models achieve the desired behaviors (fast learning or slow learning of the reconstruction task). We observed that decoupling the training speed is only useful in visually simple tasks such as Door & Key. Training is carried out with a fixed learning rate of 0.0002 using the Adam optimizer [72], with a batch size of 256.

**Environments**

We test our method in multiple environments from Minigrid [53], Super Mario Bros [87], DMLab [108], and Atari [86] games. The experiments in Minigrid and DMLab allows us to verify that our method can improve learning in sparse reward environments and its generality. The experiments in Atari games allows us to evaluate our method on tasks

Table 5.1: Ablative performance comparisons on Door & key 16×16, Super Mario Bros sparse, and Montezuma's Revenge. Averages over 10 trials are reported at different timesteps (in millions M). We report scores (mean±std) for each component of the proposed method (line 1-4), different choices of context creation method (line 5-7), different strategies to choose the position of noisy contexts (line 8-10), various predictor network loss functions (line 11-14), and the overall method (line 15-16).

| | Door & key 16×16 | | | Super Mario Bros sparse | | | Montezuma's Revenge | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | 2M | 6M | 15M | 2M | 6M | 10M | 50M | 250M | 500M |
| PPO+Fa(Cb-AE) | 0.03±0.08 | 0.88±0.15 | 0.92±0.05 | 0.30±0.08 | 0.87±0.07 | 0.89±0.03 | 4,523±252 | 8,537±189 | 8,785±341 |
| PPO+So(Cb-AE) | 0.02±0.05 | 0.65±0.21 | 0.93±0.06 | 0.21±0.05 | 0.82±0.06 | 0.81±0.05 | 2,876±345 | 6,123±512 | 8,823±472 |
| PPO+Fa(Cb-VAE) | 0.02±0.10 | 0.75±0.18 | 0.86±0.08 | 0.35±0.09 | 0.90±0.03 | 0.88±0.05 | 5,524±157 | 9,762±212 | 9,340±464 |
| PPO+So(Cb-VAE) | 0.01±0.06 | 0.61±0.18 | 0.98±0.04 | 0.18±0.07 | 0.79±0.07 | 0.74±0.04 | 3,921±415 | 8,311±475 | 9,025±378 |
| PPO+FaSo(downsample) | 0.02±0.07 | 0.79±0.17 | 0.96±0.04 | 0.24±0.03 | 0.83±0.06 | 0.93±0.05 | 3.245±165 | 6.628±200 | 8.878±622 |
| PPO+FaSo(noisy) | 0.03±0.08 | 0.76±0.15 | 0.93±0.03 | 0.27±0.06 | 0.80±0.07 | 0.91±0.06 | 3,278±180 | 6,897±225 | 9,651±442 |
| PPO+FaSo(original) | 0.03±0.05 | 0.55±0.12 | 0.78±0.07 | 0.12±0.03 | 0.66±0.10 | 0.57±0.08 | 2,128±450 | 3,287±511 | 4,425±709 |
| PPO+FaSo(random) | 0.04±0.10 | 0.64±0.23 | 0.85±0.09 | 0.11±0.08 | 0.57±0.09 | 0.76±0.08 | 2,876±237 | 5,450±487 | 6,973±904 |
| PPO+FaSo(noisy) | 0.03±0.08 | 0.76±0.15 | 0.93±0.03 | 0.27±0.06 | 0.80±0.07 | 0.91±0.06 | 3,278±180 | 6,897±225 | 9,651±442 |
| PPO+FaSo(fixed) | 0.03±0.04 | 0.71±0.10 | 0.90±0.05 | 0.22±0.05 | 0.73±0.07 | 0.85±0.05 | 3,043±340 | 5,394±412 | 8,036±667 |
| PPO+(Cb-AE/$\mathcal{L}_{cb}$) | 0.02±0.07 | 0.79±0.17 | 0.96±0.06 | 0.24±0.03 | 0.83±0.06 | 0.93±0.05 | 3,278±180 | 6,897±225 | 9,651±442 |
| PPO+FaSo(Cb-VAE/MS-SSIM) | 0.01±0.09 | 0.64±0.15 | 0.91±0.08 | 0.30±0.04 | 0.96±0.05 | 0.97±0.04 | 5,325±208 | 9,363±345 | 11,466±584 |
| FaSo+Cb-AE(MSE) | 0.01±0.12 | 0.66±0.13 | 0.92±0.06 | 0.26±0.06 | 0.64±0.08 | 0.54±0.13 | 2,748±165 | 3,846±303 | 5,025±687 |
| FaSo+Cb-VAE(MSE) | 0.02±0.07 | 0.75±0.16 | 0.97±0.05 | 0.25±0.04 | 0.68±0.06 | 0.87±0.06 | 3,142±415 | 7,424±402 | 6,560±457 |
| PPO+FaSo(Cb-AE) | 0.02±0.07 | 0.79±0.17 | 0.96±0.06 | 0.24±0.03 | 0.83±0.06 | 0.93±0.05 | 3,278±180 | 6,897±225 | 9,651±442 |
| PPO+FaSo(Cb-VAE) | 0.01±0.08 | 0.64±0.15 | 0.91±0.04 | 0.30±0.04 | 0.96±0.05 | 0.97±0.04 | 5,325±208 | 9,363±345 | 11,466±584 |

that involve deep exploration.

### 5.4.1 Ablation Analysis

We have conducted ablation studies for all the three sets of tasks (Door & key 16×16, Super Mario Bros sparse, Montezuma's Revenge) to investigate: (1) the impact of a fast/slow reward decomposition, (2) the effect of the choice of the context creation method, (3) the influence of MS-SSIM (Eq 5.11) on our method, (4) the effect of using adaptive scaling factors as part of the intrinsic reward, and (5) the performance of FaSo on "noisy-TV" tasks.

#### Fast and Slow Reward Decomposition

As described in Section 5.3.3, our method decomposes the curiosity reward into a *fast* reward and a *slow* reward. To isolate how much each reward contributes to our method, we show in Table 5.1 the performance of FaSo trained only with *fast* rewards ('PPO+Fa(Cb-AE)','PPO+Fa(Cb-VAE)') and only with *slow* rewards ('PPO+So(Cb-AE)','PPO+So(Cb-VAE)'). In Table 5.1 we see that methods trained only with *fast* rewards tend to learn faster during early training, but are outperformed later by models guided using *slow* rewards. However, we found that only a slow reward model tends to provide similar rewards for the observations of a same region of the state space, resulting in an incomplete exploration. We also observe that PPO+FaSo achieves the highest performance which indicates that it takes advantage of the two reward streams to discover more efficient exploration strategies.

The advantages of a fast and slow reward decomposition become more noticeable in environments with sparse rewards, or harder exploration such as Montezuma's revenge. Similarly to RND [20], our agents trained using a single intrinsic reward stream cannot explore all the rooms on Montezuma's revenge. In the first stage, the agent has to pick keys and open two doors. Without long-time exploration, baseline models open the first easy doors to receive the associated rewards and therefore fail (i.e. local optima). On the other hand, *slow rewards* compensate for the loss of immediate reward (no door are open) to let the agent try more global exploration behaviors (e.g. save the keys for later). In Pitfall, similar behaviors can be observed: *slow* rewards encourage the agent to travel through tunnels, useful to later collect treasures (positive rewards). Since tunnels are hard to explore and contain a lot of objects resulting in negative/deceptive rewards, baselines tend to stay in the jungle but cannot finish the level. *Slow rewards* balance negative rewards during early exploration to incentivize strategies that may result in larger rewards on a long-time scale.

### Choice of the Context Creation Method

To see the potential benefits of using observations' contexts rather original frames, we explore the performance of FaSo(Cb-VAE) with contexts created using the following strategies: downsample context, noisy context, original context (i.e. the context is the original observation, $s^* = s$) (Table 5.1). The agents trained with the original context method perform poorly. This behavior is expected since fast contexts and slow contexts are identical. It might also be related to the fact that PPO+FaSo(original) is more affected by small changes in the environment. However, for the other agents, using a context creation method (downsample or noisy) greatly improves the performance.

When using noisy contexts, another question that arises is how to choose the noise position. We experimentally evaluate different strategies for choosing the position of the white noise region. The choice of the position has a relative importance on the intrinsic bonus and the state diversity. For example, if the noisy region falls on the background or on relevant regions of the images, the reconstruction task may become easier or more difficult.

So far, the location is randomly chosen around the center of the frame (see Sec 5.3.1). Apart from it we consider the following strategies:

- full random: the location is randomly chosen within the frame without any constraint.
- fixed: the white noise region is fixed and located in the center of the observation.

We report the performance (mean±std) of the proposed method trained with noisy contexts where the noise region is: randomly selected (random), within a radius from the center $K/2$ (noisy), and fixed (fixed). As shown in Table 5.1 (line 8-10), FaSo(noisy) learns considerably faster and better than the models trained with the other strategies. We also observe that FaSo(random) is more prone to outlier and more unstable than FaSo(noisy). On the other hand, reconstructing the image from fixed noisy contexts

Table 5.2: Final mean score (± std) of our method with various scaling strategies of rewards on Atari games and average success rate (± std) on Door & Key 16×16 and Super Mario Bros. Averages over 10 runs are shown after 500M steps (Atari), 15M steps (Door & Key), and 10M steps (Super Mario Bros).

| | Maximum Mean Score (at convergence) | | | | | Success rate | |
|---|---|---|---|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Gravitar | Pitfall | Seaquest | Door & Key | Super Mario Bros |
| FaSo (Cb-AE) / N = 50 | 9,711±587 | 11,807±754 | 3,812±325 | 234±24 | 3,120±270 | 0.95 ±0.07 | 0.92 ± 0.04 |
| FaSo (Cb-AE) / N = 80 | 9,651±442 | 13,423±775 | 3,656±280 | 247±28 | 4,989±311 | 0.96 ±0.06 | 0.93 ± 0.06 |
| FaSo (Cb-AE) / N = 100 | 7,245±371 | 14,008±637 | 3,658±254 | 15±8 | 5,244±327 | 0.94±0.06 | 0.95 ± 0.07 |
| FaSo (Cb-AE) / N = 150 | 6,388±356 | 13,996±588 | 3,125±266 | -5±1 | 4,584±386 | 0.90±0.07 | 0.74 ± 0.12 |
| FaSo (Cb-AE) / N = 200 | 6,461±425 | 11,652±752 | 2,718±303 | 2±2 | 4,256±297 | 0.84 ± 0.13 | 0.68 ± 0.11 |
| FaSo (Cb-VAE) / N = 50 | 8,625±440 | 13,325±862 | 3,001±444 | 70±7 | 4,125 ± 645 | 0.72 ± 0.16 | 0.71 ± 0.12 |
| FaSo (Cb-VAE) / N = 80 | 9,121±512 | 13,311±743 | 3,257±396 | 65±10 | 4,659 ± 587 | 0.77 ± 0.13 | 0.77 ± 0.10 |
| FaSo (Cb-VAE) / N = 100 | 10,998±436 | 15,010±754 | 3,389±352 | 71±6 | 5,027 ± 463 | 0.82 ± 0.11 | 0.92 ± 0.08 |
| FaSo (Cb-VAE) / N = 150 | 11,895±487 | 15,994±712 | 3,715±318 | 180±12 | 4,826 ± 327 | 0.90 ± 0.06 | 0.97 ± 0.05 |
| FaSo (Cb-VAE) / N = 200 | 11,466±584 | 16,135±688 | 3,431±325 | 189±17 | 5,123±251 | 0.91 ± 0.04 | 0.97 ± 0.04 |
| FaSo (Cb-AE) / (Schedule 1) | 5,430 ± 462 | 11,927 ± 711 | 2,734 ± 523 | -1 ± 4 | 3,871 ± 401 | 0.90 ± 0.08 | 0.71 ± 0.09 |
| FaSo (Cb-VAE) / (Schedule 1) | 8,046 ± 613 | 13,780 ± 499 | 2,508 ± 682 | -3 ± 3 | 4,024 ± 455 | 0.86 ± 0.08 | 0.66 ± 0.08 |
| FaSo (Cb-AE) / (Schedule 2) | 5,712 ± 587 | 12,489 ± 539 | 2,115 ± 713 | 10 ± 6 | 3,915 ± 473 | 0.80 ± 0.07 | 0.72± 0.13 |
| FaSo (Cb-VAE) / (Schedule 2) | 7,369 ± 632 | 12,647 ± 500 | 2,828 ± 654 | 26 ± 12 | 4,687 ± 601 | 0.86 ± 0.11 | 0.69 ± 0.10 |
| FaSo (Cb-AE) / ($\alpha = 0.5$, $\beta = 0.5$) | 6,251±398 | 11,684±542 | 3,213±412 | 12±2 | 4,182±221 | 0.85 ± 0.06 | 0.75 ± 0.06 |
| FaSo (Cb-VAE) / ($\alpha = 0.5$, $\beta = 0.5$) | 8,750±467 | 12,459±493 | 3,312±338 | 85±6 | 4,701±186 | 0.82 ± 0.04 | 0.76 ± 0.08 |
| FaSo (Cb-AE) / ($\alpha = 0.8$, $\beta = 0.2$) | 7,465 ± 363 | 11,414 ± 522 | 3512 ± 564 | 15 ± 7 | 4,106 ± 328 | 0.82 ± 0.05 | 0.78 ± 0.04 |
| FaSo (Cb-VAE) / ($\alpha = 0.8$, $\beta = 0.2$) | 8,868 ± 484 | 11,988 ± 440 | 2532 ± 380 | 38 ± 8 | 4,789 ± 323 | 0.83 ± 0.06 | 0.81 ± 0.07 |
| FaSo (Cb-AE) / ($\alpha = 0.2$, $\beta = 0.8$) | 5,139 ± 526 | 8,234 ± 619 | 2234 ± 389 | 1 ± 2 | 3,401 ± 511 | 0.62 ± 0.15 | 0.65 ± 0.09 |
| FaSo (Cb-VAE) / ($\alpha = 0.2$, $\beta = 0.8$) | 5,340 ± 438 | 9,030 ± 782 | 2646 ± 401 | -1 ± 3 | 3,876 ± 499 | 0.64 ± 0.10 | 0.70 ± 0.12 |

tends to not capture the novelty of states where small changes are located in the center of it.

### Predictor Network Loss Function Comparison

FaSo relies on two reconstructor networks. One legitimate question is to study the impact of the choice of loss function on the performance. To answer this question, we keep other components the same and only change the loss function during training. As shown in Table 5.1, the agents trained using 'FaSo+Cb-VAE(MS-SSIM)'and 'FaSo+Cb-AE($\mathcal{L}_{cb}$)'outperform 'FaSo+Cb-VAE(MSE)'and 'FaSo+Cb-AE(MSE)', respectively. MSE loss successfully handles visually simple tasks such as on Door & key, but gives unsatisfactory performance in more complex domains such as Atari games. We conjecture that small details cannot be captured by the MSE loss and therefore MS-SSIM loss is more suitable for reconstructing complex frames.

### Adaptive Scaling of Rewards

We aim to understand the effect of using adaptive scaling of rewards. We compare the performance of FaSo(Cb-AE) and FaSo(Cb-VAE) trained with fixed scaling of rewards ($\alpha = 0.5$, $\beta = 0.5$),($\alpha = 0.8$, $\beta = 0.2$),($\alpha = 0.2$, $\beta = 0.8$) and adaptive scaling factors. Adaptive scaling factors are obtained with $N$ varying between 50 and 200. We also evaluate two schedule strategies that switch between fast ($\alpha = 0.8$, $\beta = 0.2$) and slow ($\alpha = 0.2$, $\beta = 0.8$) regimes. We report the performance of Faso trained with a slow

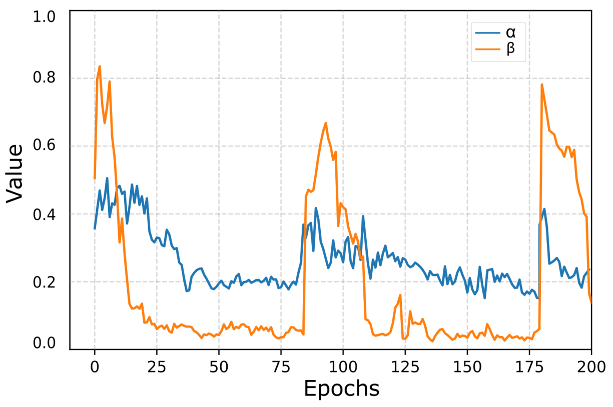


Figure 5.4: Evolution of $\alpha$ and $\beta$ across learning on the Montezuma's Revenge environment. We use Cb-VAE and downsample contexts to compute the coefficient values.

(Schedule 1) and a frequent switch in regimes (Schedule 2). Schedule 1 switches in regimes every 10 epochs and Schedule 2 every 3 epochs.

As shown in Table 5.2, methods using adaptive factors perform significantly better in most of the tasks. On Gravitar, there is only little difference between the variants although the full model worked slightly better on average. On Montezuma's revenge, the effect of adaptive weights is more clear; the agent can discover more rooms and therefore achieves a higher score. We further observe that FaSo(Cb-AE), $N = 80$, and FaSo(Cb-VAE), $N = 200$, perform well in most of the games. Similar trends can be observed on Door & Key and Super Mario Bros as well. This experiment leads us to the conclusion that FaSo is reasonably robust to the choice of $N$. In Private Eye and Door & Key, we found that a frequent switch in the regimes improves the performance compared to fixed scaling factors. However, these parameters can be difficult to tune in the absence of domain knowledge. Under this lens, having an adaptive scaling of rewards appears to be the best solution to trade-off local and global exploration strategies.

To further examine the importance of state-diversity automatic schedule, we plot the evolution of $\alpha$ and $\beta$ across learning on Montezuma's Revenge. The results are plotted in Figure 5.4. They show that $\alpha$ maintains a relatively stable value. Intuitively, exploring the state surrounding the agent enables sufficient state diversity. This is because their fast context is nearly unique, which entails that the diversity progress does not sharply decrease across many episodes. On the other hand, $\beta$ tends to produce spikes only in states that significantly drift away from the known states (e.g. a new room, a new type of obstacle). Overall, we can observe a frequent switch in regimes of fast and slow dominant phases and that "fast exploration" phases tend to be longer. Rather than fixed $\alpha$ and $\beta$, we argue that a switch in regimes is closer to how humans explore and learn.

## Randomized Environments

As pointed out by many authors [20, 21], agents that maximize the "surprise" - inability to predict the future, tend to suffer from the *TV noise* problem. For example, let us consider a curiosity formulation where the agent predicts the next observation given the current observation and agent's action. An agent maximizing this prediction error may seek out

Table 5.3: Average reward in the randomized-TV versions of Montezuma's Revenge (mean±std). Results are average over 25 random seeds after 10M timesteps of training without seed tuning.

| | Maximum Mean Score (at convergence) | | | |
|---|---|---|---|---|
| Method | Original | Noise | Noise Action $\varrho = 0.05$ | Noise Action $\varrho = 0.10$ |
| RND [20] | 8,152±653 | 3,642±902 | 6,224±647 | 5,824±733 |
| PPO+EC [21] | 8,025±770 | **4,008±823** | 7,160±845 | 6,860±862 |
| PPO+ICM [14] | 329±118 | 125±106 | 78±40 | 56±74 |
| PPO+FaSo (Cb-AE) | 9,651±442 | 3,854±779 | 8,734±611 | 7,487±884 |
| PPO+FaSo (Cb-VAE) | **11,466±584** | 3,708±806 | **9,965±599** | **8,609±740** |

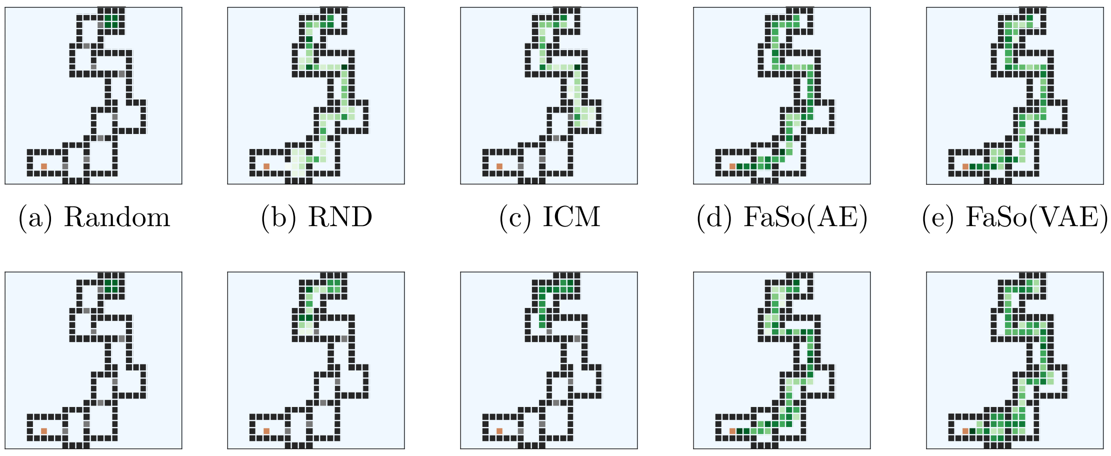
stochasticity (e.g. randomized transitions, high-frequency images) in the environment to maximize the error.

We now evaluate our method trained on randomized environments. We create versions of the Montezuma's Revenge environment with added sources of stochasticity. We test several settings:

- "Original": the original environment.
- "Noise": if the agent selects the action *jump*, a noise pattern (32×32) is displayed on the lower right of the observation - TV screen. The noise is sampled from [0,255] independently for each pixel.
- "Noise Action": if the agent selects the action *jump*, with a probability $\varrho \in \{0.05, 0.10\}$, the action performed by the agent is uniformly sampled among the possible actions.

In almost all cases, the performance of all methods deteriorates due to the stochasticity (Figure 5.3). Nevertheless, our method is reasonably robust to randomized transitions (i.e. noise action $\varrho = 0.05$ and noise action $\varrho = 0.10$). We observed that ICM gets stuck in local optima - the ICM agent frequently uses the action *jump* to maximize the prediction error. On the other hand, our formulation does not rely on the agent's action and consequently is more robust to stochastic transitions. The scores for PPO+FaSo (Cb-AE) and PPO+FaSo (Cb-VAE) are significantly higher compared to the baselines as indicated by paired t-tests at 95% confidence level ($p < 0.002$).

When adding visual noise to the environment, the performance of FaSo appears to deteriorate more. It is quite likely that visiting a state with a noise pattern produces constantly reward for such an area. That said, further analysis found that slow rewards may be large enough to escape from local optima by incentivizing the agent to try other actions. This observation motivated the use of formulations that quantify the relative improvement of the reconstruction, rather than its absolute error, but we leave it to future work to explore this direction further.



(a) Random (b) RND (c) ICM (d) FaSo(AE) (e) FaSo(VAE)

(f) Random (g) RND (h) ICM (i) FaSo(AE) (j) FaSo(VAE)

Figure 5.5: State visitation heatmaps averaged over 10 runs for different models: random, RND, ICM, FaSo(Cb-AE), and FaSo(Cb-VAE). We trained the models for 40m frames on a fixed maze (top row) and on randomly generated mazes (bottom row) in MultiRoomN10.

Table 5.4: Average success rate on fixed and randomly generated tasks from the Minigrid domain. The results are averaged over 100 runs after 40 millions training steps.

| | Fixed | | | Random | | |
|---|---|---|---|---|---|---|
| Method | MultiRoomN10 | Door&Key 16×16 | KeyCorridorS6R3 | MultiRoomN10 | Door&Key 16×16 | KeyCorridorS6R3 |
| RND | 51±1.1 | 97±0.6 | 62±0.7 | 0±3.7 | 92±4.7 | 30±7.1 |
| ICM | 18±0.3 | 65±0.8 | 23±1.2 | 0±2.1 | 3±1.2 | 21±5.6 |
| PPO+FaSo(Cb-AE) | **91±0.8** | 98±0.5 | 88±0.9 | 87±2.9 | **97±3.5** | 81±3.5 |
| PPO+FaSo(Cb-VAE) | 89±0.6 | **99±0.2** | **94±1.2** | **88±3.8** | 96±2.9 | **90±3.6** |

### 5.4.2 Fixed Versus Randomly Generated Environments

In this experiment we aim to investigate the ability of our agent to learn from randomly generated environments and generalize to unseen views or appearances. We use MultiRoom tasks from the Minigrid [53] domain. We compare PPO+FaSo(AE) and PPO+FaSo(VAE) with three baselines: random agent, RND [20], and ICM [14].

Figure 5.5 shows state visitation heatmaps on fixed (top row) and randomly generated (bottom row) mazes. We found that a random agent can only explore the first room. We also observe that ICM gets trapped in local optima in both scenarios. However, our approach discovers a large number of rooms when trained from fixed or randomly generated mazes. When trained on randomly generated mazes, existing methods are exploring much less efficiently, resulting in a poor state coverage.

To further evaluate the robustness of FaSo to random perturbations, we report in Table 5.4 the average success rate of agents trained on more environments (fixed and randomly generated) from the Minigrid domain. In all tasks, we observe that the proposed methods considerably outperforms the baselines approaches. The results further suggest that FaSo enables better generalization across the environments and is less distracted by small details that change from on environment to another.

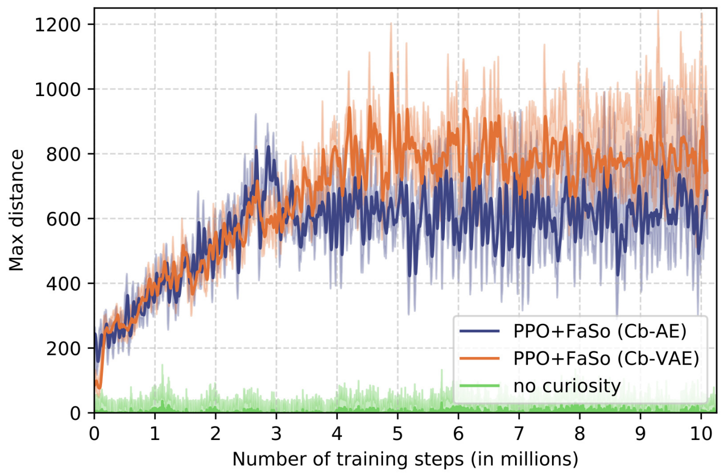


Figure 5.6: Maximum distance achieved with no extrinsic reward on Super Mario Bros. We report average distance over 10 seeds. Darker line represents mean and shaded area represents standard error.

Table 5.5: Average success rate on tasks from the Minigrid domain with dense and sparse settings. The results are averaged over 100 runs after 40 millions training steps.

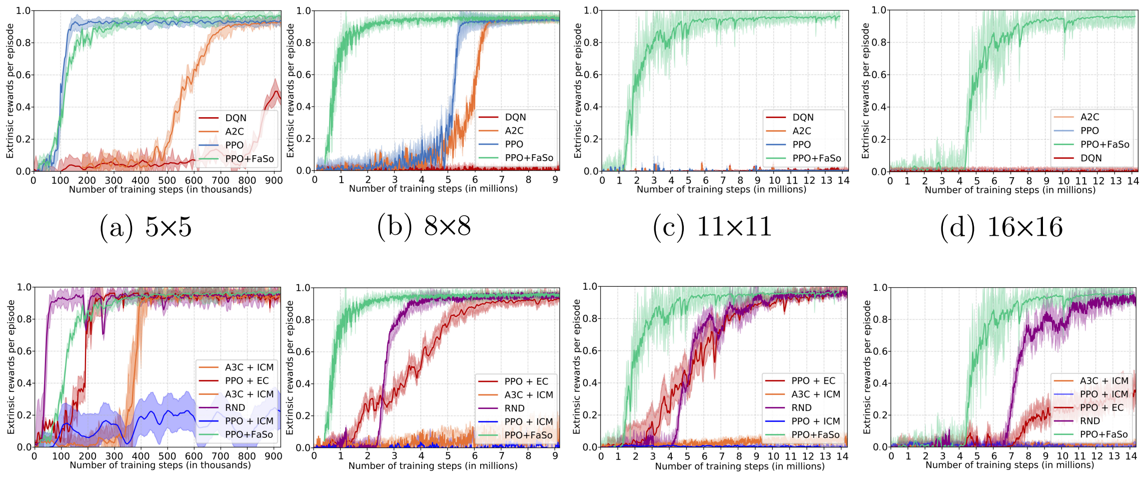
| | Sparse | | | Dense | | |
|---|---|---|---|---|---|---|
| Method | MultiRoomN10 | Door&Key 16×16 | KeyCorridorS6R3 | MultiRoomN10 | Door&Key 16×16 | KeyCorridorS6R3 |
| PPO | 0.3±4.3 | 0.0±2.1 | 0.0±1.1 | 22±3.1 | 63±1.8 | 16± 1.5 |
| PPO+FaSo(Cb-AE) | 87±2.9 | **97±3.5** | 81±3.5 | 74±1.1 | **93±1.3** | 77± 1.9 |
| PPO+FaSo(Cb-VAE) | **88±3.8** | 96±2.9 | **90±3.6** | **83±1.6** | 94±2.9 | **86±1.8** |

## 5.4.3 No Extrinsic Reward

For testing the good exploration coverage of our method, we trained our agent on Super Mario Bros without any reward from the environment. Our agent only receives a curiosity-based signal to reinforce its policy. As can be seen in Figure 5.6, in order to remain curious the agent is pushed to explore distant regions of the state space, which entails that its coverage increases over time. It highlights that in the absence of extrinsic rewards, FaSo provides enough intrinsic supervision exploration signal for learning useful behaviors. We found a statistically significant difference between PPO+FaSo(Cb-VAE) and PPO+FaSo(Cb-AE) after 2.7M steps (paired t-test, $p < 0.05$).

## 5.4.4 Dense Reward

A desirable property of the proposed method is to avoid hurting performance in tasks where rewards are dense and well-defined. We evaluate this scenario in MultiRoomN10, Door&Key 16×16, and KeyCorridorS6R3. In those tasks, the agent has to collect keys, open doors, and reach a target position. In the sparse setting, the agent is only provided a sparse terminal reward of +1 if it finds the target and 0 otherwise. In the dense setting, the agent is rewarded for collecting keys (+0.3) and opening doors (+0.3), as well as reaching the goal (+1). The results show (Table 5.5) that our method does not significantly deteriorate performance in dense reward tasks (paired t-test p>0.05), with the exception of FaSo(Cb-AE) on MultiroomN10 (p=0.0034). Even though PPO+FaSo(Cb-



(a) 5×5 (b) 8×8 (c) 11×11 (d) 16×16

(e) 5×5 (f) 8×8 (g) 11×11 (h) 16×16

Figure 5.7: Comparison of PPO+FaSo with baselines with no curiosity (top row) and agents augmented with an exploration bonus (bottom row) in Minigrid Door & Key. The hardness of the exploration task (i.e. sparsity) is gradually increased from left to right. Results are averaged over 10 random seeds. No seed tuning is performed. The shaded area shows the standard errors of 10 runs.

AE) and PPO+FaSo(Cb-VAE) perform slightly worse in the dense setting, they still greatly improve performance as compared to plain PPO.

### 5.4.5 Exploration with Sparse Extrinsic Rewards

We now report experimental results in three domains including Minigrid, Super Mario Bros, and Atari games, characterized by sparse rewards. It aims to investigate how useful our proposed method is for hard exploration tasks on which recent advanced exploration methods mainly focused.

#### Minigrid

We performed a set of two experiments on Door & Key to evaluate the overall performance of our algorithm. First, we verify if our model (PPO+FaSo) achieves better performance than traditional RL methods (DQN [1], PPO [38], A2C [37]). Second, we compare it against state-of-the-art curiosity-based learners for different degree of sparsity (i.e. size of board). We evaluated our model against RND [20], PPO+EC [21], A3C+ICM [14], and, PPO+ICM [14] that were shown to perform well in sparse reward environments.

We present in Figure 5.7 the evolution of the extrinsic reward achieved by the agents. The results of each run are averaged to provide a mean curve in each figure, and the standard error is used to make the shaded region surrounding each curve. In such sparse tasks, the learning curve shows that our model always outperforms RL baselines. Moreover, only our method scales with the size of the environment and is significantly faster in term of convergence speed. The performance gap is more pronounced in levels hard to learn

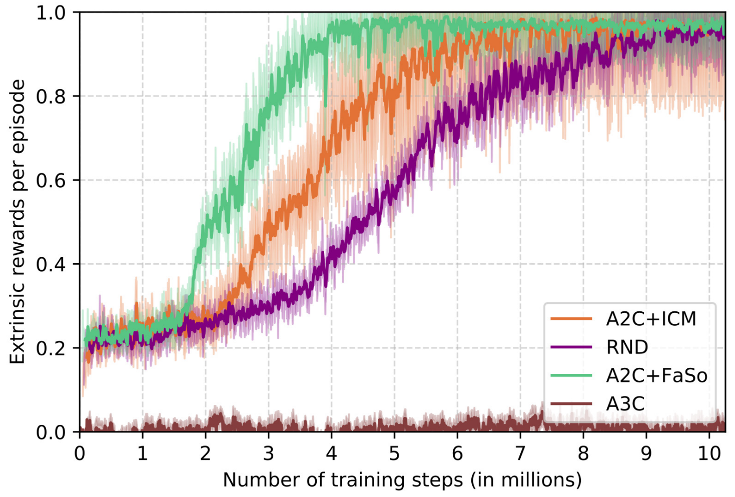


Figure 5.8: Average task reward obtained in the Super Mario Bros environment with sparse reward setting. We run every method with a repeat of 10 and show all runs. Mean and standard error of the mean over trials are plotted.

(i.e. size > 5) and a significant difference was found between our method and every other baselines (paired t-test at 95% confidence, p<0.001). In Door & Key 5×5, the difference between FaSo and PPO is, however, not statistically significant (t-test p>0.05).

We further observed that baseline methods will exhaust their curiosity quickly after experiencing unexpected events such as after *picking the key*, and therefore struggle to reach the final goal. On the other hand, we found that when using a slow curiosity reward, intrinsic reward remains large enough to encourage long-time horizon exploration strategies such as *opening the door after picking the key*, which significantly improves performance.

### Super Mario Bros

Next, we apply our method to the Super Mario Bros environment [87]. As a baseline, we compare our model to the A3C, A2C+ICM and PPO+EC algorithms. For a fair comparison with the state-of-the-art approach [14], we combine FaSo with A2C, from the open-source implementation [109]. We use the same hyperparameters as in the work [14]. Fig 5.8 shows the normalized average reward (over 10 runs) obtained for each method. The main result is that A2C+FaSo obtains a near perfect score in a smaller number of epochs than any other method. The proposed method can significantly accelerate learning compared to the state-of-the-art ICM algorithm (t-test p=0.023, t=2.26).

### Atari Games

We also evaluate the proposed curiosity method on five difficult exploration Atari 2600 games from the Arcade Learning Environment (ALE) [86]: Montezuma's Revenge, Private Eye, Gravitar, Pitfall. In the selected games, training an agent with a poor exploration strategy often results in a suboptimal policy. We compare our method to the performance of A2C and PPO without intrinsic reward. The results are shown in Table 5.6. We

Table 5.6: Final mean score of our method and baselines on Atari games. We report the results achieved over total 500M timesteps of training, averaged over 10 seeds.

| | Maximum Mean Score (at convergence) | | | | |
|---|---|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Gravitar | Pitfall | Seaquest |
| A2C [37] | 15±20 | 572±136 | 2,758±185 | -17±2 | 1,613±244 |
| PPO [38] | 2,487±942 | 103±56 | 3,438±412 | -31±5 | 1,548±341 |
| RND [20] | 8,152±653 | 8,666±1051 | 3,906±246 | -3±1 | 3,179±378 |
| PPO+EC [21] | 8,025±770 | 9,244±634 | 3,521±246 | -12±1 | 4,650±358 |
| PPO+ICM [14] | 329±118 | 485±71 | 3,447±242 | -15±2 | 2,165±223 |
| PPO+GoCu [80] | 9,123±751 | 10,223±587 | 550±625 | -2±1 | 2,055±259 |
| PPO+PoBP [78] | 9,125±856 | 11,124±601 | **5,069±256** | 109±11 | 2,958±311 |
| PPO+ReBP [78] | 8,934±911 | 8,598±599 | **4,915±187** | 0±2 | 3,165±325 |
| DeepCS [76] | 3,500 | 1,105 | 881 | -186 | 3,343 |
| Average Human [77] | 4,753 | 69,571 | 3351 | 6,464 | 20,182 |
| PPO+FaSo (Cb-AE) | 9,651±442 | 13,423±775 | 3,656±280 | **247±28** | 4,989±311 |
| PPO+FaSo (Cb-VAE) | **11,466±584** | **16,135±688** | 3,431±325 | 189±17 | **5,123±251** |

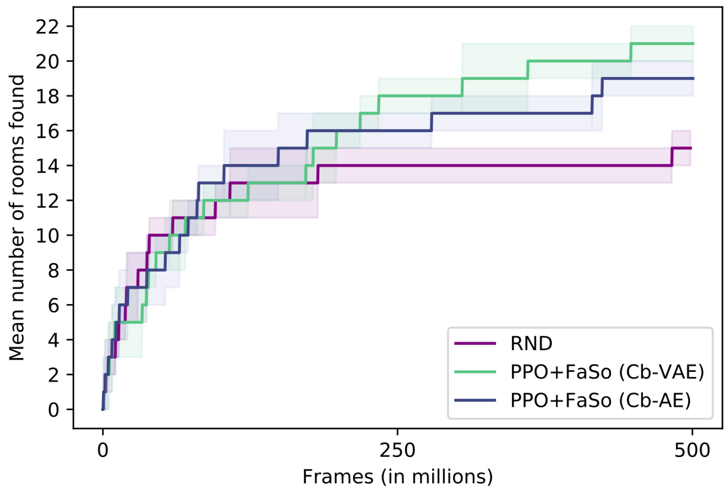


Figure 5.9: Average number of rooms (± std-error) found during the training phase on Montezuma's Revenge. We run every algorithm with a repeat of 10.

consider the mean final reward of 10 training runs with the same set of hyperparameters. It is observed that both baselines obtained a score close to zero and could not solve most of the tasks.

We further compare PPO+FaSo against various methods using different exploration strategies. To evaluate the significance of the scores, a paired t-test was conducted to compare the received average total reward in the proposed method and the best-performing methods on the five Atari 2600 games. A significant difference was found between our agents and every other methods ($p<0.001$), except on Gravitar where no significant difference was found between between RND and PPO+FaSo(Cb-AE) ($p=0.073$). As presented in Table 5.6, on Montezuma's Revenge, Seaquest and Private Eye our model outperforms other approaches that mainly deal with local exploration. It suggests that high-level exploration is vital for exploring in complex environments. For instance, on Montezuma's Revenge, FaSo(Cb-VAE) exceeds state of the art performance. It might be related to

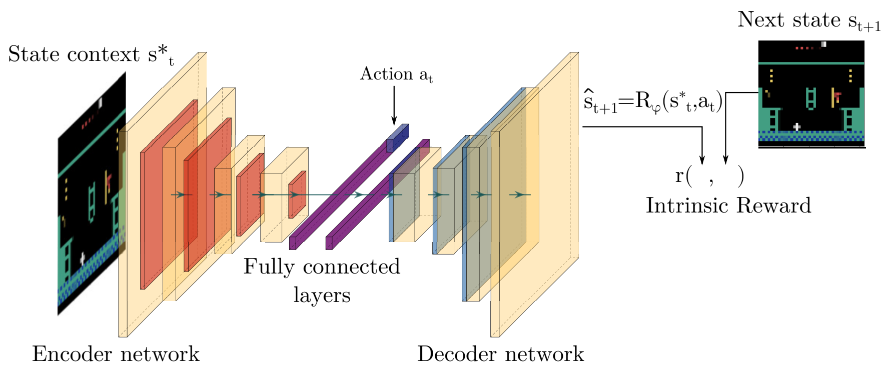


Figure 5.10: The reconstructor $R_\phi$ architecture. The current observation's context and the action are passed through an encoder network and the decoder network attempts to reconstruct the original next frame. Novelty is measured as the discrepancy between the original next frame and the reconstructed next frame.

the very fact that slow rewards are large enough to motivate the agent to discover and visit new rooms. As a result, our agent explores a larger number of rooms as compared to RND (Figure 5.9). In Pitfall, many interactions yield negative rewards that dissuade baselines from exploring efficiently the environment. We found that while the baselines focus on short-term rewards, they tend to converge prematurely to sub-optimal policies. In this task, the cognitive capability to make long-term intrinsically motivated choices is required for the agent to compensate deceptive extrinsic rewards and discover alternate policies.

## 5.5 Intrinsically Motivated Lifelong Exploration

Intrinsically Motivated Lifelong Exploration (IML) extends the idea presented above. We present a different strategy to flexibly combine intrinsic rewards - the deep signal modulates the local novelty reward. This novel strategy alleviates the need for weighting reward streams while effectively promoting lifelong learning. We also aim to develop means for the better incorporation of action-dependent information into the existing intrinsic reward formulation. To do so, the new formulation integrates the agent's action and relies on next-frame reconstruction errors. This is necessary since large changes in the environment can lead to minor visual changes.

### 5.5.1 Integrating Motion Dynamics in Reconstruction-based Curiosity

We propose a different measure of intrinsic motivation formulated as the quality of the agent to reconstruct the next state given the current observation's *context* and the executed action (Figure 5.10). The proposed architecture can be understood as a form of forward model, however, a key difference lies in the use of *context* that discards irrelevant details and forces the model to capture meaningful visual features and salient environmental dynamics at different scales (e.g. character-level or word-level), leading to different exploration behaviors characterized by different time horizons.

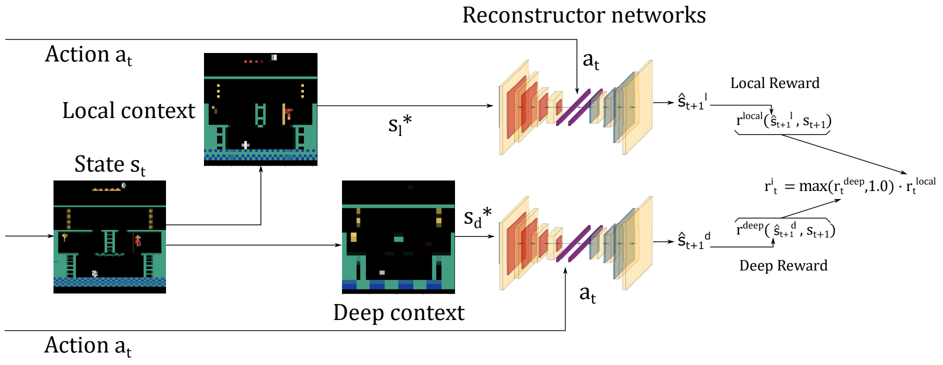


Figure 5.11: Training architecture of the proposed model. The reconstructor networks are trained separately based on the local and deep contexts.

At every time step $t$, the module takes as input the current observation $s_t$, the next state $s_{t+1}$, and the action $a_t$ executed at time $t$. Let $s_t$ be the original observation and $s_t^*$ its context. The reward calculation involves a *reconstructor* network $R_\phi : s^* \times a \mapsto s$ parameterized by $\phi$ that reconstructs the next state $s_{t+1}$ given the context of the observation and the current action. Write $\hat{s}_{t+1}$ denotes the reconstructed image. Formally, the reconstruction task can be expressed as:

$$\hat{s}_{t+1} = R_\phi(s_t^*, a_t) \tag{5.17}$$

The reconstruction will have some errors that can be interpreted as a measure of novelty and serves as an intrinsic motivation signal to the agent, $r^{cd}$:

$$r^{cd}(s_t, a_t) = \left[\frac{SSIM(s_{t+1}, \hat{s}_{t+1})}{\sigma(R_{cd})}\right] = \left[\frac{SSIM(s_{t+1}, R_\phi(s_t^*, a_t))}{\sigma(R_{cd})}\right] \tag{5.18}$$

where $SSIM$ is the single-scale SSIM metric [103], $\sigma(R_{cd})$ is the running estimate of the standard deviations of the sum of discounted intrinsic rewards, and $s_t^*$ is the context of the state $s_t$.

## 5.5.2 Integrating Lifelong Curiosity

Similarly to FaSo, the *local reward* component and the *deep exploration* component are calculated by two distinct context-driven curiosity models (Figure 5.11). However, in IML the models reconstruct the original next state $s_{t+1}$ given the local $s_l^*$ and deep $s_d^*$ contexts of the current state and the executed action, respectively, $R_{\theta_l} : s_l^* \times a \mapsto s$ and $R_{\theta_d} : s_d^* \times a \mapsto s$; with trainable parameters $\theta_l$ and $\theta_d$. A key difference also lies in the combination of $r^{deep}$ and $r^{local}$. We do so by multiplicatively modulating the local reward with a deep curiosity factor. Thus, the overall intrinsic reward provided by IML is calculated as:

$$r_t^i(s_t, a_t) = max(r_t^{deep}, 1.0) \cdot r_t^{local} \tag{5.19}$$

This can be re-written as follows:

$$r_t^i(s_t, a_t) = max(r_t^{deep}, 1.0) \cdot \left[\frac{SSIM(s_{t+1}, R_{\theta_l}(s_l^*, a_t))}{\sigma(R_{cd}^l)}\right] \tag{5.20}$$

Table 5.7: Final mean score (mean±std) of our method and baselines on Atari games. We report the results achieved over total 600M timesteps of training, averaged over 10 seeds.

| | Maximum Mean Score (at convergence) | | | | | |
|---|---|---|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Gravitar | Pitfall | Seaquest | Solaris |
| PPO [38] | 2,487±942 | 103±56 | 3,438±412 | -31±5 | 1,548±341 | 1,387 ±248 |
| PPO+RND [20] | 8,152±653 | 8,666±1051 | 3,906±246 | -3±1 | 3,179±378 | **3,282±281** |
| PPO+EC [21] | 8,025±770 | 9,244±634 | 3,521±246 | -12±1 | **4,650**±358 | 3,014 ±273 |
| PPO+ICM [14] | 329±118 | 485±71 | 3,447±242 | -15±2 | 2,165±223 | 1,330 ±108 |
| PPO+GoCu [80] | 9,123±751 | 10,223±587 | 550±625 | -2±1 | 2,055±259 | 2,726±199 |
| PPO+PoBP [78] | 9,125±856 | 11,124±601 | 5,069±256 | 109±11 | 2,958±311 | 2,875±223 |
| PPO+ReBP [78] | 8,934±911 | 8,598±599 | 4,915±187 | 0±2 | 3,165±325 | 2,945±187 |
| PPO+FaSo [110] | 9,651±442 | 13,423±775 | 3,656±280 | 247±28 | 4,589±311 | 3,154±217 |
| Average Human [77] | 4,753 | 69,571 | 3,351 | 6,464 | 20,182 | 12,327 |
| PPO+IML (ours) | **16,135±688** | **18,004±455** | **6,125±550** | **2,511±373** | 3,814 ± 504 | **3,120±338** |

where $r_t^{deep}$ is the deep exploration bonus computed by the deep component: $[SSIM(s_{t+1}, R_{\theta_d}(s_d^*, a_t))/\sigma(R_{cd}^d)]$. Intuitively, modulating the local reward by the deep reward results in a frequent switch in regimes of local and deep dominant phases, encouraging the agent to repeatedly revisit not fully explored states in its environment as well as novel regions. This combination of intrinsic rewards maintains consistent exploration throughout the agent's training process and encourages in-depth exploration.

## 5.5.3 Comparison Between FaSo and IML

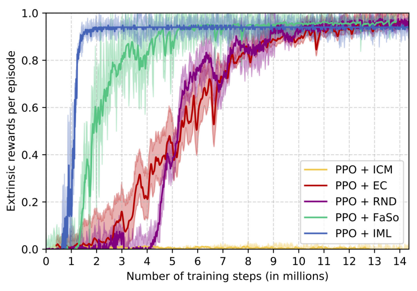
We now compare FaSo with IML on Atari games and procedurally generated tasks. We selected those tasks because they require lifelong exploration strategies.

### Implementation Details (IML)

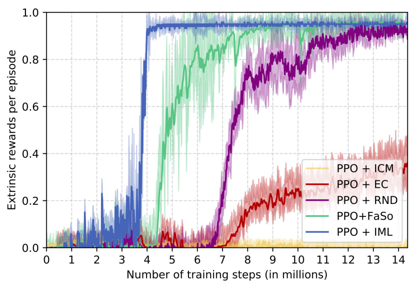
On DMLab, we employ contexts created with $K_{local} = 2$ and $K_{deep} = 5$. On Atari and Minigrid, we compare our method trained with contexts $K_{local} = 2$ and $K_{deep} = 4$. We set the coefficient of intrinsic reward $\alpha = 0.5$. The architecture of the reconstructor networks consists of a sequence of four convolutional layers with 32 filters each, stride: 2,1,1, kernel size of $3 \times 3$, and padding 1. We apply a rectifier non-linearity after each convolutional layer. For online training, we store the experience and make 5 epochs of training every 12K steps. We found that retraining the *deep* reconstructor network once every 30K steps is sufficient. Note that we decouple the training speed of the reconstructor models to ensure that the models achieve the desired behaviors. Training is carried out with a fixed learning rate of 0.0003 using the Adam optimizer.

### Atari Games

We first conduct experiments on six established hard exploration Atari 2600 games from the Arcade Learning Environment (ALE) [86] that feature very sparse rewards and com-



(a) Door & key 11×11



(b) Door & key 16×16

Figure 5.12: Average results over 10 random seeds on Minigrid (Door & key). The shaded area shows the standard errors of 10 runs.

plex visual features. The average scores are reported and compared in Table 5.7. It can be observed that IML outperforms the baselines in Montezuma's Revenge, Gravitar, and Pitfall, and Private Eye. In Seaquest, the scores of our method is comparable to those of PPO+Faso and PPO+EC. The rationale is that these games demand long-term exploration incentives, and thus are harder to be solved by using a curiosity-based bonus that vanishes quickly. For instance, on Pitfall the agent must go through regions of the state space that can yield negative rewards, which dissuade baselines from exploring efficiently. On the other hand, since our formulation of curiosity does not vanish with further state visitations while promoting lifelong exploration; it allows our agent to discover alternative ways to obtain the optimal reward.

### Procedurally Generated Environments

We now validate IML's capability for solving procedurally generated environments and generalizing to unseen views or appearances. Figure 5.12 shows the training curves on Minigrid Door & key of our method against four baselines. We can observe that PPO tends to perform poorly due to the sparsity of the tasks. From Figure 5.12, it is clear that ICM is significantly outperformed, we suspect that the high similarity among states results in small rewards. Moreover, learning effective controllable states using a simple inverse dynamics model can be challenging. On the other hand, RND can learn near-optimal policies but is slower than the proposed method. As can be further observed, IML achieves higher sample-efficient than FaSo. Overall, the results show that our method can learn effective policies in procedurally generated environments. Namely, IML can reasonably adapt to ever-changing situations and environments. It confirms that IML is a vital element that makes learning from extremely sparse rewards possible.

## 5.6 Discussion

Our work took a step towards achieving deep exploration in reinforcement learning. The proposed formulation of curiosity explicitly promotes in-depth exploration across episodes. We have constructed a mechanism based on reward decomposition and showed that the method can help exploration in challenging sparse-reward environments. The intrinsic rewards are estimated following the idea of *context-driven reconstruction* to capture salient visual features and motion dynamics at different scales of the environment. We developed two algorithms for flexibility combining local and deep rewards based on the concept of: 1) *state diversity*, or 2) lifelong learning. The experiments demonstrated the effectiveness of these approaches by achieving significant improvements in notoriously difficult tasks such as Pitfall or Montezuma's Revenge. Remarkably, to the best of our knowledge this is the first work that obtained a positive score for all six Atari games. Altogether, the experimental results suggested that enabling long-term motivation can greatly improve performance and exploration efficiency.

We validated the effectiveness of our approach by achieving significant improvements in long-horizon and challenging tasks. That being said, we acknowledge that our approach has certain limitations. In order to be useful for real-world tasks, solely relying on internal guidance seems to be an approach that is rarely taken in human and animal learning. That is, learning from scratch may be impractical in the real-world and time-consuming. In the long run, we believe that exploration algorithms should not be limited to modeling intrinsic curiosity, but should also integrate other types of external guidance as well as human expertise (prior assumptions about the domain).

# Part II

# Bridging the Gap Between Reinforcement Learning and Human Guidance

# Chapter 6

# Combining Deep Reinforcement Learning with Prior Knowledge and Reasoning

So far, we discussed how an agent can discover and acquire skills by using curiosity as its own supervision. We have developed end-to-end reinforcement learning systems that can cope with the problem of learning from sparse and poorly-defined extrinsic rewards. However, in order to be useful for real-world tasks, the agent needs to also develop capabilities for common sense reasoning by integrating external guidance such as human expertise. In practice, solely relying on internal guidance (i.e. learning from scratch) does not consider real-world scenarios where other types of external guidance are often available. Knowledge how to approach a new task can be transferred from previously learned tasks, and/or it can be extracted from the guidance a teacher. For instance, in a simple robotic manipulation task, it seems straightforward to learn an optimal policy through imitating a human demonstrator's behaviors.

In the context of reinforcement learning, the most common form of external guidance is imitation learning. Despite recent advances in imitation learning, in many cases it is impractical to use human demonstrations as guidance because: (1) some of these tasks are too challenging for even humans to perform well, (2) it often requires large amounts of demonstrations, significantly increasing the burden on the human, and (3) the human demonstrator must be highly familiar with the task and understand how to perform it. In this thesis, we design novel types of external guidance, which are less expensive and more intuitive for humans. Our central concept is to focus on forms of guidance that reduce the cost of human effort and substantially improve sample efficiency.

In the following chapter, we aim to introduce human-like planning and domain knowledge to enhance information given to the agent. The depicted method relies on (simple) high-level domain knowledge and visual recognition. The intuition behind is that this form of guidance is intuitive to provide even for a non-expert, can be easily exploited by the agent, and generalizes to many aspects of the task. We then derive a framework for the integration of human-like reasoning. It also deals with incomplete/imperfect human knowledge by correcting possible human errors based on trial-and-error interactions.

## 6.1 Introduction

Reinforcement learning is a technique that automatically learns a strategy to solve a task by interacting with the environment and learning from its mistakes. However, end-to-end RL agents often struggle in complex environments such as three-dimensional virtual worlds, resulting in a prohibitive training time and an ineffective learned policy.

As mentioned above, it is often unrealistic to expect an end-to-end reinforcement learning system to succeed with no prior assumptions about the domain (i.e. learning a task from scratch). Therefore, several methods have attempted to introduce various types of supervision into reinforcement learning systems. A powerful recent idea to tackle the problem of computational expenses is to modularise the model into an ensemble of experts [111]. Since each expert focuses on learning a different stage of the task, the reduction of the actions to consider leads to a shorter learning period. Although this approach is conceptually simple, it does not handle very challenging domains or environments with large sets of actions. In a similar fashion, a method aims to simulate the core features of human *visual intelligence* [112] by augmenting cognitive architecture with background knowledge [113], but is not directly applicable to RL and is restricted to a supervised classification problem. In detail, the idea is to leverage information about videos with external ontologies to detect events in videos and augment a supervised model with prior knowledge. A widely studied class of methods, imitation learning, provides supervision via demonstrated trajectories [114]. Inverse reinforcement learning is another form of imitation learning where a reward function is inferred from the demonstrations [25, 26, 115]. However, collecting large amounts of high-quality demonstrations is a notoriously challenging problem, and is limited to domain knowledge in the form of sequences of state-action pairs. Besides, demonstrations are only applicable to very similar situations - it is difficult to generalize demonstration data to slightly different situations. Another approach, *Symbolic Reinforcement Learning* [16, 116], combines a system that learns an abstracted representation of the environment and high-order reasoning. Nevertheless, this has several limitations, it cannot support ongoing adaptation to new environments and cannot integrate other sources of knowledge.

In this chapter, our approach focuses on combining deep reinforcement learning and external knowledge. Using external knowledge is a way to supervise the learning and enhance information given to the agent by introducing human expertise. We augment the input of a reinforcement learning model whose input is raw pixels by adding high-level information created from simple knowledge about the task and recognized objects. High-level information are easy to interpret for the agent and enable a significant speedup in the learning speed. Furthermore, this type of guidance is less expensive than policy demonstrations while providing additional information that can be generalized to many situations - a small amount of high-level information is enough to extract strategies how to approach most aspects of the task. In order to deal with the problem of imperfect domain knowledge, we combine an action selection model trained via interactions (i.e. the learned policy) with a knowledge-based decision algorithm using Q-learning [117] or a Support Vector Machine [118]. The knowledge-based decision model relies on human-like planning and reasoning to recommend the next optimal action. Therefore, we simultaneously leverage simple domain knowledge to augment the agent's inputs and introduce human-like planning and reasoning, greatly reducing the learning workload and enabling common sense reasoning.

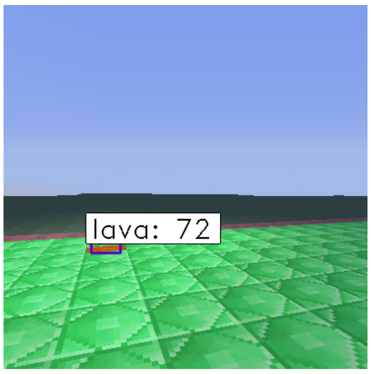


Figure 6.1: Screenshot of the environment.

In our experiments, we demonstrate that our framework successfully learns in real-time to solve a food gathering task and to find a target in a maze, in 3D partially observable environments by only using visual inputs. We evaluate our technique on two challenging 3D environments built on top of the Malmo platform, Minecraft. Our model is especially suitable for tasks involving navigation, orientation, or exploration.

## 6.2 Task & Environment

We built two environments on the top of the Malmo platform [119] to evaluate our idea. Malmo is an open-source platform that allows us to create scenarios based on the Minecraft engine. To test our model, we trained an agent to: 1) collect foods in a field with obstacles and 2) find a target in a maze. The agent can only receive partial information of the environment from his viewpoint. We only use image frames to solve the scenario. An example of screenshot with the object recognition results is shown in Figure 6.1.

### 6.2.1 Eating a Healthy Diet

The goal of the agent is to learn to have a healthy diet (Task 1). It requires to recognize the objects and learn to navigate into a 3D environment. The task consists in picking up food from the ground for 30 seconds. Food is randomly spread across the environment and four obstacles are randomly generated. Each of the 20 kinds of food has an associated reward when the agent picks it up. This reward is a number between +2 (healthy) and -2 (unhealthy). They are distributed equitably, meaning that a random agent should get a reward of 0.

The settings were: window size: *400* $\times$ *400* pixels, actions: *turn left, turn right, crouch, jump, move straight and move back* , number of objects: *200*, number of obstacles: *4*. The actions *turn left* and *turn right* are continuous actions to make the learning smoother as consecutive frames are more similar.

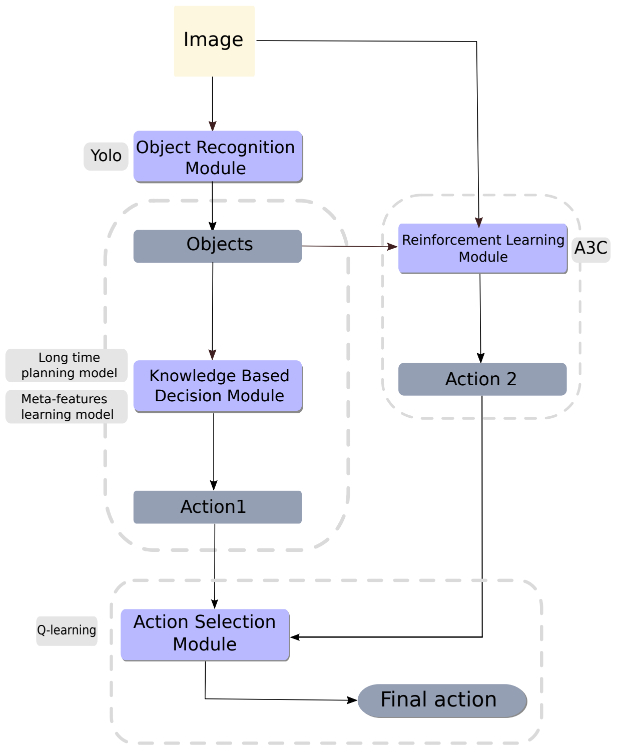


Image
Yolo
Object Recognition Module
Objects
Reinforcement Learning Module
A3C
Long time planning model
Meta-features learning model
Knowledge Based Decision Module
Action 2
Action1
Q-learning
Action Selection Module
Final action

Figure 6.2: Global architecture of DRL-EK.

### 6.2.2 Finding a Target in a Maze

We also evaluate the algorithm on an orientation task: finding a target in a maze (Task 2). A positive reward (+1) is given when the agent reaches the goal and a negative reward is given in case of timeout (-900), after 45 seconds without finding the target. Several objects are randomly placed in the maze. An episode is restarted if the agent collides with a hazardous object and a negative reward is given (-1000). We generate the objects such as the target, represented by a flag, is always reachable.

The settings and the actions are the same as in the *Eating a healthy diet* task except for the number of different objects. We limit the possible objects to 15 including 5 hazardous objects. In total, 30 objects are generated at each episode.

## 6.3 Deep Reinforcement Learning Augmented With External Knowledge

Figure 6.2 describes the global architecture of our new framework called DRL-EK. It consists of four modules: an Object Recognition Module, a Reinforcement Learning Module, a Knowledge Based Decision Module, and an Action Selection Module.

The object recognition module identifies the objects within the current image and generates high-level features. These features of the environment are then used to augment the raw image input to the reinforcement learning module. In parallel, the knowledge based decision module selects another action by combining external knowledge and the object recognition module outputs. To manage the trade-off between these two sources of decision we use an action selection module. The chosen action is then acted by the agent and the modules are updated based on the obtained reward.

### 6.3.1 Object Recognition Module

Injecting external knowledge requires to understand the scene at a high-level in order to be interpreted by a human. The easiest way to understand an image is to identify the objects. For example, it is intuitive to give more importance to the actions ***turn*** or ***jump*** than the action ***move straight*** when an obstacle is in front of the agent.

To recognize the objects, the module uses the You Only Look Once (YOLO) [120][121] library which is based on a deep convolutional neural network. As input, we use an RGB image of size 400×400 pixels. YOLO predicts in real time the bounding boxes, the labels, and confidence scores between 0 and 100 of the objects. An example is shown in Figure 6.1. We trained YOLO on a dataset of 25 000 images with twenty different classes corresponding to the food that is presented in the environment.

The model is trained off-line before starting the learning into the environment. The neural network architecture is adapted from the one proposed by Redmon *et al.* (2016) for the Pascal VOC dataset [121]. In order to recognize small objects, the size of cells is decreased from 7 to 5 pixels and the number of bounding boxes for each cell is increased from 2 to 4.

In addition to the identified objects, the module creates feature information about the current frame. To generate these high-level abstraction features we combine the recognized objects and external knowledge. They are then used as input by the reinforcement learning module and the knowledge based decision module. We designed three types of features *presence of objects*, *important area* and *important area with eligibility traces.*

#### Presence Of Objects Features

The first type of features is a vector of booleans which indicates whether an object appears or not within the current image. The size of this vector is the number of different objects in the environment. Since some objects are not helpful to solve the task, we can decide to only take some of the objects into account based on our knowledge about the task.

#### Important Area Features

Since the position of the objects is important, we encode information about objects within each area of the image. We split the image into $k$ rectangles vertically and horizontally. So, the number of areas is $k^2$ and for each one we compute a score (Figure 6.3). The score of an area is the sum of the score of the objects within this area. External knowledge can be introduced by shaping the score of the objects. Based on our knowledge about the task, we manually defined the scores to indicate whether or not an object is important to

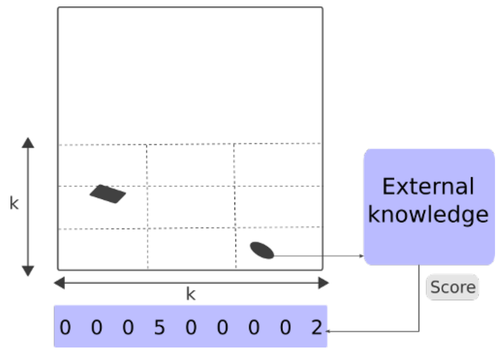


Figure 6.3: Important areas of an image.

solve the task. To tackle problems with partially observable environments, we keep track of recent information by concatenating the array of scores of the current frame with the arrays of the two previous frames.

In our experiments, the top half of the images only contains the sky so we computed the important area features on the half bottom of the images. We gave a score of -15/+5 to foods we think is unhealthy (cake,cookie) / healthy (meat, fruit) and 0 for the others. That way, if an area contains a healthy food such as a fruit and a sweet food, then the score of the area will be lower than an area containing only a fruit or no object. We set the number of rectangles to 3 (9 areas in total: 3×3). We found that with a higher number of areas the amount of encoded information is bigger but information quality of each area is worse than with 3 areas.

### Important Area Features with Eligibility Traces

Important areas with eligibility traces [122] features is a variant of the important area features (IAF) [123], suitable for orientation tasks such as *finding a target in a maze*. As with IAF, we compute a score for each part of the image but the score associated to each object is adapted over time depending on the observed rewards. The idea behind is that some objects are more important than the others to solve the task. Adapting the scores turned out to be critical in guiding the algorithm to solve tasks.

Our algorithm maintains a parametrized score-function which maps the objects to their score. At the beginning of the training, we initialize the scores using our prior knowledge. The scores are adapted over time to fit with the rewards. Instead of computing the value of each part of the image with the fixed scores, we use the updated scores of the objects.

Since objects have not an immediate impact on the reward, we use an eligibility trace mechanism to back-propagate the rewards. At each iteration, the new score $score(obj_i)'$ of an object $obj_i$ is estimated as follows:

$$score(obj_i)' = score(obj_i) + \alpha(e_t(obj_i) \times r) \tag{6.1}$$

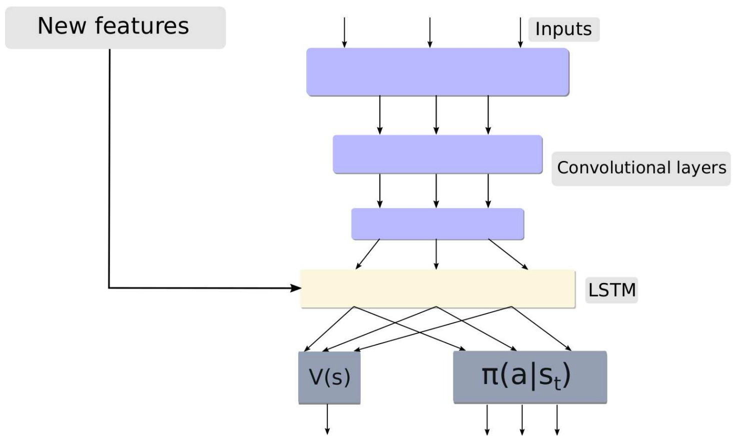


Figure 6.4: Injection of new features into the reinforcement learning module (A3C).

with $\alpha$ the learning rate and $e_t(obj_i)$ the eligibility trace of $obj_i$:

$$e_t(obj_i) = \begin{cases} \lambda e_{t-1}(obj_i) + 1, & \text{if } s = s_t \text{ and } a = a_t \\ \lambda e_{t-1}(obj_i) & \text{otherwise} \end{cases} \tag{6.2}$$

Our contribution here is to provide a simple technique to adapt our prior knowledge to fit the task. Please note that we used a linear learning rate decay to learn the eligibility traces and an eligibility trace decay $\lambda = 0.5$.

### 6.3.2 Reinforcement Learning Module

For a computer, learning from an image is difficult and requires a lot of training steps. To deal with it, the entry point of most of the reinforcement learning models is a recurrent convolutional neural network [124] to extract temporal and spatial features of the image.

In this work, we trained a deep reinforcement learning model to perform policy learning and we modified the neural network structure to incorporate external knowledge. In addition to the image input, we injected *presence of objects* or *important area* features which are created by the object recognition module. In the neural network, we give to a Long Short Term Memory (LSTM) [125] the output of the last convolutional layer concatenated with the new features (Figure 6.4). The next layers of the neural network are two separated fully-connected layers to estimate the value function $V(s)$ and the policy $\pi(a|s_t)$. The purpose is to help the model at the beginning of the training to recognize and focus on objects. The new features augment the raw image input to the reinforcement learning model by adding high-level information. For example, from presence of objects features the model can decide which actions are allowed or not. If a $door$ is detected some of the actions may become irrelevant such as $jumping$.

The choice of the reinforcement learning model highly depends on the environment. Since the model at each time-step takes an input and outputs an action, we can easily substitute most of the reinforcement learning techniques such as Deep Q-learning (DQN) [126], Deep Deterministic Gradient Policy (DDPG) [127], Dueling Network [128] or Asynchronous Actor-Critic Agents(A3C) [37] by using a recurrent convolutional neural network as state approximator.

A3C is the most suitable model to solve our task. We tested and empirically searched the best parameters such as a good convolutional neural network architecture and the choice of the optimizer of this model. It provides a baseline to evaluate the importance of each module of our architecture on the final policy.

Working directly with 400×400 pixel images is too computationally demanding. We apply image preprocessing before training A3C. The raw frame is resized to 200×200 pixels. To decrease the storage cost of the images we convert the image scale from $0-255$ to $0-1$.

We set the number of workers of A3C to 3 and a convolutional recurrent neural network is used to approximate the states. The reason why we use a recurrent neural network is because the environment is partially observable. The input of the neural network of A3C estimator consists in a 200×200×3 image. The 4 first layers convolve with the following parameters (filter: 32,32,32,32, kernel size: 8×8, 4×4,3×3,2×2, stride size: 2,2,2,1) and apply a rectifier nonlinearity. It is followed by a LSTM layer of size 128 to incorporate the temporal features of the environment. Two separate fully connected layers predict the value function and a policy function, a distribution of probability over the actions. We use RMSProp [129] as optimization technique with $\epsilon = 10^{-6}$ and minibatches of size 32 for training.

## 6.3.3 Knowledge Based Decision Module

We believe that the agent is not able to accurately understand and take into account the objects of the environment. A human can easily understand and make a decision from high-level features such as the utility or name of an object. Getting this level of abstraction is difficult but we can help the machine by giving it less low-level information such as color of pixels but more high-level information such as the importance of an area of the image.

Moreover, when the reinforcement learning module is fed with the images and the presence of objects or important areas features, the training time is long due to the size of state space. The knowledge based decision module is able to select an action using external knowledge and high-level features generated by the object recognition module and without direct access to the image. We propose two different approaches, a knowledge reasoning model or a meta-feature learning model

**Knowledge Reasoning Model**

Our technique for solving the task relies on human reasoning and prior knowledge. Our previous approach [123] hard encodes a set of rules. This may require many modifications to be adapted to new environments. Our contribution here is to provide a new architecture able to store, retrieve and reason on knowledge regardless of the environment. In addition, abstracting rules from environments allow a better control and an easier transfer among domains. To deal with environment abstraction, we divide the task into three steps:

preprocessing, knowledge representation, and reasoning.

The preprocessing file enables us to deal with complex and non-intuitive inputs. This step can be easily adapted to new environments. The purpose is to generate features that can be interpreted and stored. In case of images, a preprocessing can be the recognition of the objects within a frame. The output of the preprocessing file is then used to check which rules are satisfied. The rules are saved in a second file. They associate a pattern to an action and allow to introduce complex external knowledge about the task. An example of a simple rule is, if the object *cookie* is on the left of the image then the action *turn left* is forbidden.

A pattern is a conjunction of variables which can be arbitrarily complex. The variables represent significant events in the task. For example, in task involving driving a car, a variable could be *(speed between 20 and 50 km/h)* and an example of pattern is (*(speed between 20 and 50 km/h)* $\wedge$ *(pedestrian crossing the road)*).

Given an observation $obs_t$, the active rules are the rules for which all their variables are active. Given a priority order previously defined, one rule is selected.

The recommended action is selected in the reasoning file which combines traditional test-case algorithms and planning. It takes as input the selected rule and the output of the preprocessing file. We store in an array the sequence of planned actions. At each time-step, the model checks if the previously planned sequence of actions is still the optimal one and if it is not the case (for example the next action is *jump* but there is no obstacle) the algorithm updates it, otherwise the first action in the array is returned. If the selected action is related to the movement, the model estimates the best angle and the necessary number of steps to perform it. Finally, the first of the planned actions is returned.

Please note that to decrease the number of rules, we discretized the image space into four areas: *center, left, right, other.* We designed 43 rules to prevent the agent from going in the direction of the food we think is dangerous. To avoid static behavior, we give more priority to the actions *turn left, turn right, move straight* than the others.

### Meta-feature Learning Model

In our previous approach, we manually create rules for reasoning on high-level features. Here, to automatically learn the rules and select the optimal action from them, we use a deep reinforcement learning model such as DQN or dueling network. Unlike the reinforcement learning module which uses the image, the only input is high-level features such as *important areas* or *presence of objects.* As the input is much smaller than an image, a simple neural network can be trained to approximate the states. The smaller number of parameters leads to a faster learning than a model trained from visual information.

In experiments, we trained a dueling network combined with a double deep Q-learning (DDQN). It empirically gives a smoother learning than most of the other reinforcement learning models. A neural network approximates the states. It consists in 3 fully connected layers of size 100 with a rectifier nonlinearity activation function. Network was trained using the *Adam* algorithm [130], learning rate of $10^{-3}$ and minibatches of size 32. As input, we used a slightly modified version of the important area features outputted by the object recognition module. To create important area features, we filtered the objects too far and the objects with a confidence score less than 0.25. Taking into account an object such as *grass* is irrelevant and makes the learning more difficult. We only used dangerous or very healthy (10 objects out of 20) objects and removed 2 objects that we know are difficult to distinguish.

### 6.3.4 Action Selection Module

The module aggregates the actions proposed by the reinforcement learning module and the knowledge based decision module to select the action that the agent will perform in the environment. The goal is to take advantage of the fast learning of the knowledge based decision module and the quality of the policy learned by the reinforcement learning module. An important aspect of the action selection, is selecting an action with the highest expected return but also detecting error patterns to correct them. An error must be detected when an action which has not been proposed could offer a higher *return*. In this section, we describe in detail a reinforcement learning approach and a supervised learning approach.

#### Reinforcement Learning for Action Selection

We train a Deep Q-learning model to select the best action, detect and correct the error patterns. There is no restriction on the possible actions meaning that the final action may be different from the two proposed actions if an error is detected. We encode the proposed action by the two modules into two indicator vectors. An indicator vector is a binary vector with only one unit turned on to indicate the recommended action. The neural network input is the concatenation of these two vectors.

The Q-learning algorithm is trained using a Boltzmann distribution (Equation 6.3) as explorer and experience replay (each experience is stored into memory and the algorithm is run on randomly sampled batch) with a memory of size $10^6$. Equation 6.3 gives the probability of selecting an action in a given state $s$.

$$P_s(a) = \frac{exp(Q(s,a)/\tau)}{\sum_{a' \in A} exp(Q(s,a')/\tau)} \tag{6.3}$$

The Q-network is composed of 2 hidden fully connected layers of size 50 and are followed by rectified linear units.

#### Supervised Learning for Action Selection

An alternative approach to the action selection problem is to train a supervised learning model. It restricts the problem to a classification task with two possible classes: *-1* reinforcement learning module and *+1* knowledge-based decision module.

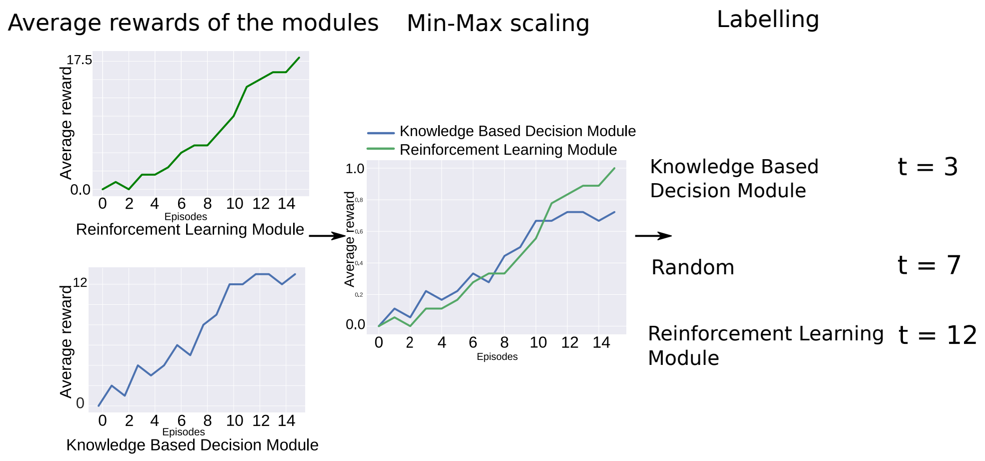


Figure 6.5: Data labeling using average reward as tag.

We create the dataset during the training process. At each iteration, a new observation is added to the examples and we label them according to performance of each module (Figure 6.5). First, each module is evaluated to estimate its average reward over iterations. In order to compare the scores, we scale the rewards using a Min-Max normalization, with min, the minimum reward of the two average rewards and reciprocally for the maximum.

The class assignment is performed as follows:

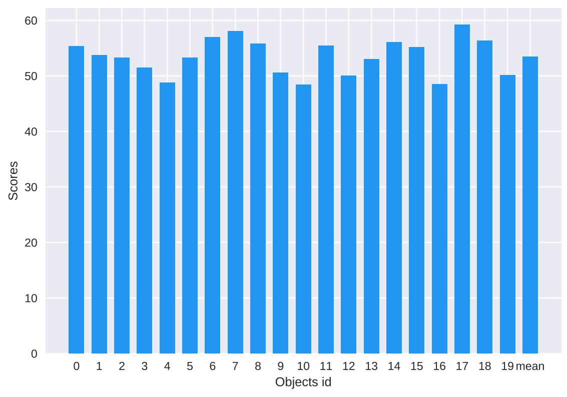
$$P(class) = \begin{cases} -1 & \text{if } \beta_{RL} \geq \beta_{KBD} \\ 1 & \text{otherwise} \end{cases} \tag{6.4}$$

with $\beta_{RL_t} = \mathcal{N}(\bar{reward}(RL)_t, \sigma^2)$, and $\beta_{KBD_t} = \mathcal{N}(\bar{reward}(KBD)_t, \sigma^2)$, where $\sigma^2 = 0.05$ and $reward(x)_t$ the scaled average reward of the module $x$ at the iteration $t$.

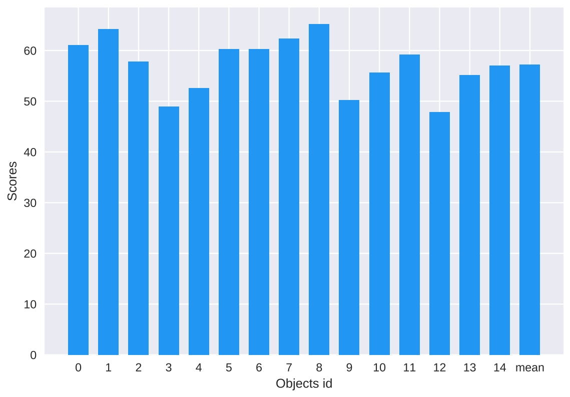
This means that we assign the label of the module with the highest average reward, and, when the rewards for both modules are similar we randomly assign a label. To deal with the classification problem, we build a support vector machine trained with a stochastic gradient descent optimization. It allows us to update the model over time. The input variables are the same as with the *Reinforcement Learning* approach. We convert the predicted class by the SVM to an action and then act it in the environment. Please note that the SVM was trained using a *l2* regularization term ([131]), a Gaussian kernel [132] and the penalty parameter $C = 5.0$

## 6.4 Experiments

We conducted several experiments for evaluating our architecture. In all our experiments, we set the discount factor to 1.0. According to our different tests, on average the best reward that a perfect agent can get on task 1 in 30 seconds is 9.



(a) Task 1



(b) Task 2

Figure 6.6: Average precision over all the classes obtained by the object recognition module for Task 1 (left) and Task 2 (right).

### 6.4.1 Object Recognition

We evaluated our object recognition module for understanding the correctness of obtained object information in the environment. Figure 6.6a reports the object recognition module performance on the *eating a healthy diet* task. In this experiment, we measured the mean average precision (mAP) as the error metric. The results are similar to the results presented by the authors (Redmon *et al.*, 2017) [121] on the Pascal VOC. dataset [133]. We obtained a mean average precision of 53.47. Although other libraries could offer higher performance, the real-time detection was the main criterion for selecting YOLO.

We noticed that most of the errors are false positives (68.3%) whereas the false negatives (31.7%) are uncommon. It leads to an agent with a policy more greedy and safer. As shown in the figure, the average precision is similar for every class. We also report the average precision for the objects of the task 2 (Figure 6.6b). On this task, we obtained a mAP of 57.19. The best average precision was for the *object 8* (lava block) and the worse for the *object 12* (obsidian block) with an average precision of 47.9 and 65.2 respectively. The mean average precision is similar among the two tasks. Note that the performance of YOLO is slightly affected by the complexity of the objects such as their shape, color, or size. We could not establish a direct link between the visual complexity of an object and the average precision. We hypothesize that it mostly depends on the quality of the training examples.

### 6.4.2 Knowledge Reasoning Model

Next, we tested the knowledge reasoning model in the knowledge based decision module to evaluate the effectiveness of this approach on the Task 1. We only utilized the object recognition module and the knowledge based decision module in our framework. The knowledge reasoning model performs much better than a random agent with an average reward of 3.3 (Figure 6.7), since the expected reward of a random agent is 0. The most likely cause of the wide variance is the difficulty to handle all possible cases with manually created rules. For instance, the agent has difficulty in gathering food near obstacles. Since

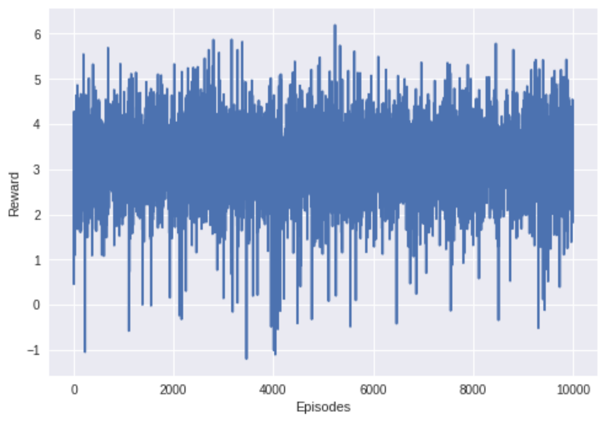


Figure 6.7: Evolution of the reward of the knowledge reasoning model.

there is no learning, the quality of the agent only depends of the quality of the rules and is not able to converge. On the other hand, from the first episode the average reward is much higher than any other learning based models.

### 6.4.3 Meta-feature Learning Model

To evaluate performance of the meta-feature learning model we use as a baseline the knowledge reasoning model. We trained the model to gather food in a 3D environment (task 1). We optimized its parameters, by sampling hyper-parameters from categorical distributions:

- Number of areas sampled from $\{4, 9, 16, 25\}$
- Number of hidden layers from $[1, 5]$
- Size of hidden layers sampled from $\{25, 50, 100, 200, 300\}$

Figure 6.8 reports an example of hyper-parameter optimization results. Each cell corresponds to a configuration of parameters. As can be seen on the figure, a number of hidden layers larger than two or a large number of areas results in lower performance. The best hyper-parameters are 9 areas, and a neural network with 3 hidden fully-connected layers of size 100. Training time is about 4 hours for each configuration on a Nvidia Titan-X GPU.

Figure 6.9 shows how the average total reward of the meta-feature learning model evolves during training with the optimal settings. The dueling network architecture effectively learns to solve the task from the important area features. The learning is fast during the first 3000 episodes and the average reward quickly converges around 5.4. It is also interesting to note that this approach rapidly achieves higher performance than the knowledge reasoning model. Automatic rule learning is more effective than manual rule construction. Unfortunately, the rules cannot be represented in a human-interpretable way.

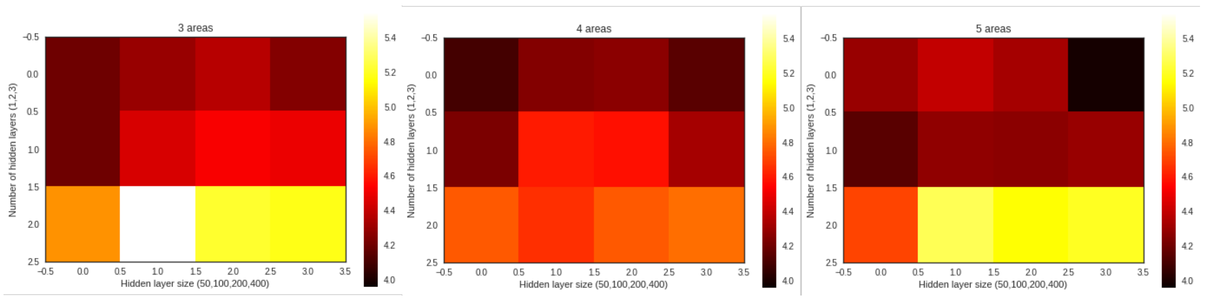


Figure 6.8: Average rewards of the meta-feature learning model with different parameters. The rewards were averaged over 200 episodes after 5000 training episodes.

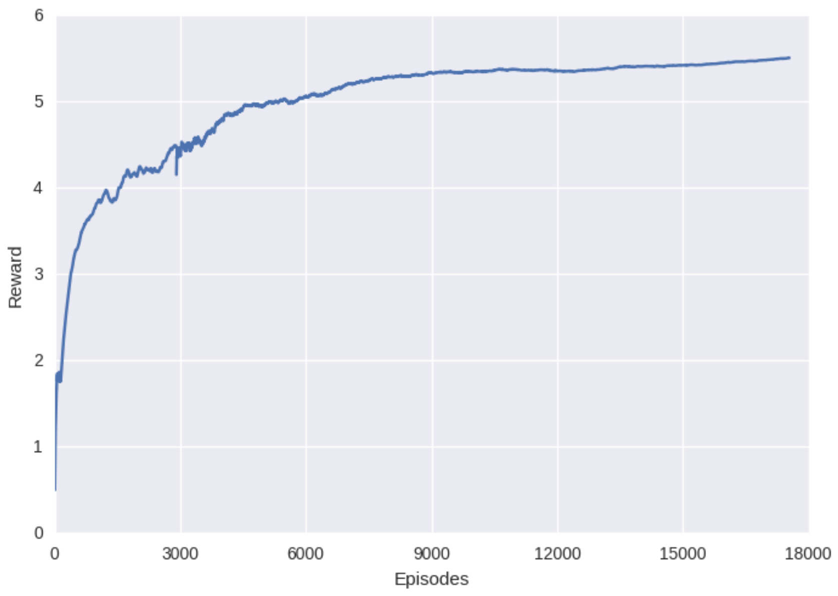


Figure 6.9: Average reward using meta-feature learning.

### 6.4.4 Action Selection

Table 6.1 reports the frequency of time the agent is able to reach the target in the maze (Task 2) using a supervised learning based action selection module and a reinforcement learning based action selection module. We obtained the results by running the corresponding agent 5 times. The agent was trained for 10000 episodes and then tested for 500 episodes.

Table 6.1: Frequency of time the agent reaches the target. The first row shows performance for a SVM as classifier and the second row for a Deep Q-learning algorithm. The last row consists in a random action selection between the two proposed actions.

| Settings | Frequency |
|---|---|
| SVM | 0.421 |
| Deep Q-learning | 0.624 |
| Random | 0.372 |

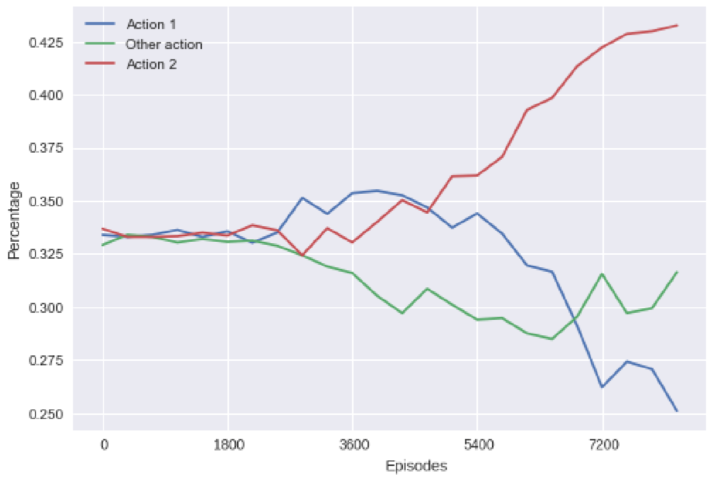


Figure 6.10: Frequency of selection of each action.

The Deep Q-learning method outperforms the SVM technique and a random selection. In particular, the low performance of the supervised approach can be caused by the difficulty to annotate the dataset. However, further analysis shows that the action selection module, trained with a SVM exhibits higher performance during first 4000 iterations than with a DQN. This indicates that the lack of adaption capability of the SVM is harming the quality of the resulting policies.

We also evaluated the characteristics of the action selection module (task 1). We report the percentage of actions which is selected from the knowledge based decision module (action 1) and from the reinforcement learning module (action 2) against other actions. We measured the frequency of selection of each action every 350 episodes. As shown in Figure 6.10, the action selection module at the beginning selects equally the actions then more the action 1 and gradually give more importance to the action 2. The results confirm our intuition, the module selects the action of the most efficient module and adapts over time the trade-off between the sources of decision to always select the best one.

### 6.4.5 Overall Performance

Finally, we report the average reward of our whole framework trained using the injection of important areas features with eligibility traces into the reinforcement learning module and a meta-feature learning model as knowledge based decision module. A Deep Q-learning was used to select the best action. Figure 6.11 compares our proposed method with the best performing reinforcement learning methods on the Task 1. These models learn the policy only using raw pixels.

DRL-EK boosts A3C by injecting important area features with eligibility traces. To select these features, we compared *A3C+presence of objects features*, *A3C+important area features*(A3C*) and *A3C+important area features with eligibility traces* (Table 6.2). In both cases, the results show that adding a new input to the reinforcement learning module improves the quality of the policy.

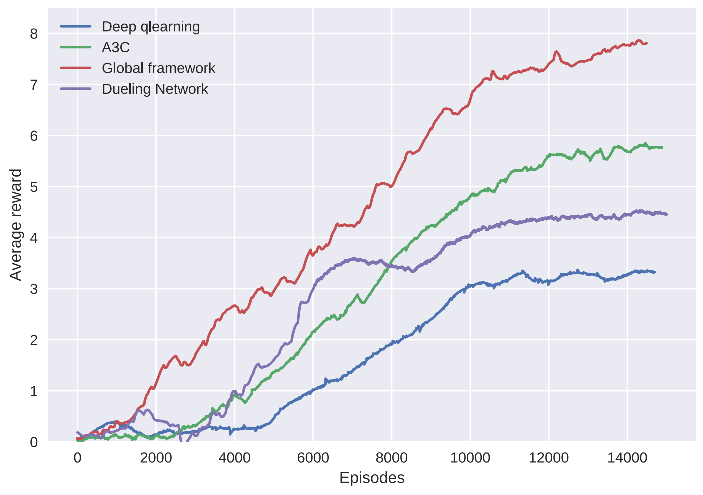


Figure 6.11: Performance of DRL-EK comparing to DQN, Dueling Network and A3C on Task 1.

Table 6.2: The table compares average reward for various features injected into A3C. The reinforcement learning module was evaluated alone for 12 000 episodes.

| Settings | Rewards |
|---|---|
| A3C | 5.6 |
| A3C + *presence of objects features* | 5.8 |
| A3C + *important area features* (A3C*) | 6.1 |
| A3C* + *eligibility traces* | 7.3 |

As can be seen, DQN gives the worst results with an average reward of 3.2, $\approx 40\%$ less than A3C after converging. After 12 000 episodes, the average reward of the dueling network architecture trained with a double deep Q-learning is around 4.4 while A3C is able to achieve an average reward of 5.6. Surprisingly the meta-feature learning model trained alone (Figure 6.9) achieves higher performance than learning only from the image with a dueling network or a DQN model. Asynchronous advantage actor-critic tends to learn faster than any other reinforcement learning based models. We believe this is due to the 3 parallel workers of A3C that offer a nonlinear significant speedup. These results show that the proposed architecture, DRL-EK, outperforms the baselines. Its average reward is around 15% better than A3C after 14 000 episodes. Moreover, the performance of DRL-EK at the beginning of the training is significantly better than all other baselines. One thing to note is that the action selection module tends to select an action different from action 1 and 2 (Figure 6.10). The continuous increase of the average reward of DRL-EK and this observation indicates that the action selection module is partially able to learn to correct the errors.

We also report learning curve on the Task 2 (Figure 6.12). We observe similar results as in the task 1. DRL-EK converges to a higher average reward, and clearly improves over A3C and A3C*. In average, our model is able to reach the target about 60% of the time whereas A3C and A3C* achieve a score of 40% and 50% respectively. The experiments demonstrate the importance of each module of our system. Please note that

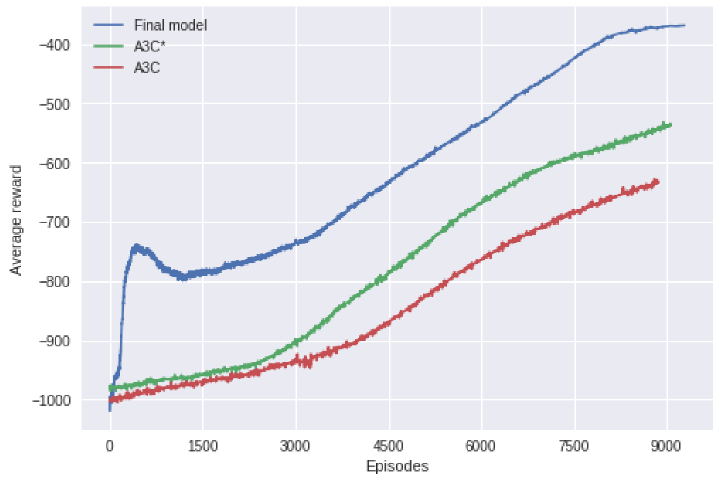


Figure 6.12: Comparison of performance curves between DEL-EK, A3C, A3C* on Task 2.

with an average time for one step of 0.43 seconds on a Nvidia Titan-X (Pascal) GPU, DRL-EK can be trained in real time. In general we observed that the agent can generalize a small amount of high-level knowledge to most situations encountered throughout the training process, significantly improving the performance. For instance, the agent rapidly understands how to survive by coupling its ability for learning and reasoning with the provided domain knowledge. Moreover, learning from high-level knowledge reduces the human effort and eliminates the need for demonstration data, enabling common sense reasoning in tasks that are too challenging for even humans to perform well.

## 6.5 Discussion

We proposed a new architecture to combine deep reinforcement learning with external knowledge. We demonstrated its ability to solve complex tasks in 3D partially observable environments with images as input. Our central thesis is enhancing the agent's inputs by generating high-level features of the environment. These high-level features provide general prior assumptions about the environment that can be applied to most situations. Further benefits stem from efficiently combining two sources of decision and adapting prior knowledge to fit the tasks. Moreover, our approach can be easily adapted to solve new tasks with a very limited amount of human work. We have demonstrated the efficacy of the proposed architecture to greatly reduce the learning time and learn more effective policies by enhancing the agent's decision-making with human-based planning and reasoning.

Our method opens several avenues of research. The depicted method and most prior work integrate human guidance and domain knowledge designed expressly to solve the task being learned. However, we feel that if we want to further reduce the amount of human involvement and improve the performance of our agent, it is important to develop means for the incorporation of existing datasets into reinforcement learning. This would enable the agent to leverage large amounts of (already created) knowledge with a minimal human workload. All these considerations lead to two fundamental questions: how to extract meaningful information from datasets related to the current task?, and how to represent the information in a way that they can be understood by the agent?

Another limitation of our proposed as well as prior methods is that they lack interpretability - their operation is largely opaque to humans, rendering them unsuitable for domains in which verifiability is important. This issue limits the applicability of these methods to non-critical real-world tasks, excluding for example autonomous vehicles, or medicine. More troubling, the difficulty to extract a humanly-comprehensible chain of reasons for the action choice makes these methods less suitable for active cooperation with humans.

# Chapter 7

# Towards Interpretable Reinforcement Learning with State Abstraction Driven by External Knowledge

In the previous chapter, we presented a first agent whose choice of action is augmented with high-level human guidance, facilitating various forms of common sense reasoning. In retrospect, we concluded that the agent lacked an important component: interpretability. Furthermore, we did not take into account the possibility that already created datasets related to the task being learned often exist. In the context of real-world tasks, it would be desirable to automatically extract meaningful domain knowledge from existing datasets, leading to a drastic reduction in human effort.

In this chapter, we focus on giving a reinforcement learning agent the capability of leveraging existing human expertise. To do so, we build an agent whose internal representation is first-order logic rules automatically extracted from existing datasets related to the task being learned. We propose different strategies to discover these rules; and we show that the natural structure of the rules can be used to 1) improve generalization, and 2) extract a humanly-comprehensible chain of reasons for the action choice of the agent.

## 7.1 Introduction

Deep reinforcement learning methods present a number of challenges. First, they suffer from lack of interpretability. While deep neural networks have been shown to be very effective, the structure of these models makes them difficult to be interpreted, which restricts their use to non-safety critical domains, excluding for example, medicine or law. Second, they require large datasets to be efficient. To build their representation from complex data such as images, neural networks need a large amount of data which entails that they learn slowly. They typically require millions of steps to learn good control policies. As a consequence, DRL cannot be directly applicable to real-world tasks such as robots [134] or recommendation systems [135], emphasizing the need of sample-efficient RL. Third, they suffer from low generalization capability in the sense that their ability to determine similarities among previously encountered situations is limited. In deep reinforcement learning, this abstraction is achieved by a neural network. However, DRL tends to generalize poorly on seemingly minor changes in the task [16, 136].

Motivated to overcome these shortcomings, we propose a learning framework that addresses all of these issues at once by combining simple interpretable reinforcement learning and prior domain knowledge [137], adding a minimal human overhead. Our algorithm leverages large amounts of domain knowledge to significantly accelerate learning without the need for demonstrations [138] or specific human engineering. Namely, we propose a new variant of the Sarsa($\lambda$) algorithm [122], which is based on the idea of learning policies that are humanly-comprehensible. The basic concept of this method is to represent the states of the policy as understandable rules, reducing the state space as well as the amount of data needed to learn an efficient state representation. In this work, we present and compare different strategies to discover these first-order rules by analyzing the agent's experience and datasets relevant to the current task, based on: 1) human knowledge, 2) the patterns learned by a random forest, and 3) the latent representation learned by a deep neural network. Besides, their structure can be used to maximize the benefits of past experiences to face new situations (i.e. generalization). This is the key idea of the sub-states mechanism which exploits similarities among rules. Sub-states allow a more frequent update of the Q-values thereby smoothing and speeding-up the learning. Furthermore, we adapt eligibility traces and the learning rate, which turned out to be critical in guiding the algorithm to solve tasks. Finally, we introduce extra supervision during early training by using external knowledge to initialize the parameters of our model.

We demonstrate the effectiveness of this approach across various applications such as visual tasks or time series. We find that our agent learns effective policies in a small number of iterations and exhibits higher performance and faster training than the best generally-applicable reinforcement learning methods.

## 7.2 Related Work

In this section, we outline the state-of-the-art methods in reinforcement learning to address each of the shortcomings presented before, namely, interpretability, data efficiency, and generalization.

A key component of many reinforcement learning algorithms is neural networks. They contain a lot of implicit knowledge about the problems but need to be explainable - they should provide easy-to-interpret reasons for the choice of an action. Kim *et al.* propose to interpret neural networks using visual interpretation [139]. Specifically, a visual attention model is used to train a convolution network. A causal filtering can determine which input regions influence the output. However, this method cannot be easily interpreted by an algorithm and is only suitable for visual domains. Symbolic reinforcement learning [116] aims to solve the lack of interpretability of neural networks as well as improving their generalization capabilities. For example, Garnelo *et al.* combine a back-end deep neural network to learn a symbolic representation and a front-end interpretable reinforcement model that learns the interactions between the objects [16]. Although the agent could learn to master a navigation task, this has not yet been shown to work in rich visual environments. Instead of representing policies by neural networks, Verma *et al.* represent policies using a high-level interpretable language [140]. During the first step, a neural policy network is trained, and then a high-level policy is "extracted". This approach seems not to be directly applicable to environments in which simulations are costly.

Another active area in reinforcement learning is improving data efficiency. A line of work (Bougie *et al.*) involves external knowledge to help the agent to focus on important features of the environment during training [137]. This gives prior assumptions about the domain to the system, reducing training time of the agent. At present, only simple knowledge can be introduced and the lack of interpretability limits the usability of this method. Another line of work [141] proposes to use deep auto-encoders as pre-processing stage of visual RL. Deep auto-encoders were shown capable to learn robust feature representations. We manage to bypass the lack of interpretability issue by learning robust features using an auto-encoder and then extracting interpretable state representations. In recent years, one way of learning faster has been extensively studied: imitation learning. It enables agents to learn an optimal policy through imitating a human demonstrator's behaviors [142, 138]. In a slightly different spirit, inverse reinforcement learning extracts a reward function from demonstrations and then trains a policy to maximize it [25, 26]. However, as mentioned above, these approaches are not directly applicable to behaviors that are difficult for humans to demonstrate. Moreover, demonstrations are specifically collected to solve the current task. On the other hand, in this work we hypothesize that leveraging existing sources of task-relevant information such as datasets about the environment is essential to reduce human workload. Moreover, integrating prior assumptions about the domain directly onto the internal representation of our model provides additional and richer information than state-action trajectories (i.e. imitation learning). For instance, consider an autonomous vehicle. Rather than collecting a human demonstrator's behaviors that are expensive, only contain trajectory information, and are solely designed for one task; we aim to integrate general knowledge such as traffic rules or information about potentially dangerous situations by analyzing widely available "driving" datasets.

Generalization capability field aims to facilitate transfer learning among observations, central in reinforcement learning to reduce the amount of training data. Compact state representation [143] area focuses on creating an abstract representation of the states [144]. It enables a faster learning than training the agent on the raw data without facing the drawbacks of deep learning. For instance, Andre *et al.* hierarchically abstract the states by decomposing the states into subroutines [145] but has been limited to simple domains. Another method [146] applies a state aggregation technique to reduce the number of state-action pairs that relies on estimating the similarity among the pairs of states. The main drawback is how to compute the similarity between complex objects such as pixels or time-series. All the previously cited approaches suffer from lack of interpretability which reduces their usage in critical applications such as autonomous driving. Bougie *et al.* use an abstract representation of the states to improve generalization while having decisions fully interpretable [147]. This work extends this idea; we aim to reduce the amount of human work by proposing a new deep unsupervised method to generate the rules. In addition, we increase the generalization capability of the agent with better use of sub-states. Finally, we conduct more extensive evaluations of our algorithm on two different domains.

## 7.3 Rule-based Sarsa($\lambda$)

We propose a method, rule-based Sarsa (Sarsa-rb), to enable Sarsa in continuous spaces by injecting external knowledge (Figure 7.1). Sarsa-rb is divided into two stages: rule

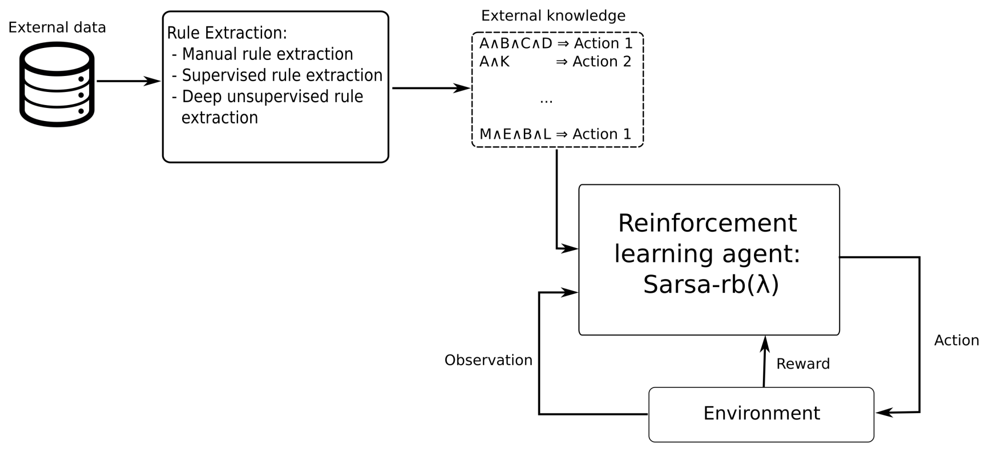


Figure 7.1: Rule-based Sarsa($\lambda$) architecture. The extracted rules are fed into the reinforcement learning agent to learn efficient policies and improve state abstraction.

extraction, and, learning (e.g. reinforcement learning agent) (Sarsa-rb($\lambda$)).

At the top level, the rule extraction module extracts symbols from various sources of (existing) data and then generates rules to describe the environment. We formalize this problem as learning a mapping $\phi$ (i.e. a set of rules) from a state $s_t$ to its abstract representation $\bar{s}_t$, where $\bar{s}_t = \phi(s_t)$; $\phi(s_t) \in \bar{S}$. Since the rules can be created using our prior knowledge about the task and external source of data, we refer to them as *external knowledge*. Besides the general idea that the state representation has the role of encoding and compressing essential information about the task while discarding irrelevant states, it enables to inject prior assumptions about the domain.

The second stage is a reinforcement learning system trained to maximize the reward signal. Training Sarsa in the raw state space is undesirable not only because the structure of the images makes them difficult to interpret, but also because it is hard to predict pixels directly. The rule-based representation constructed in stage one can now be jointly used to enhance state representation in Sarsa and to efficiently initialize the Q-values. As in Sarsa, the agent estimates the Q-values, however, each state is represented by a rule-based representation. At this point, the advantage of representing the states by rules becomes clear. Their compositional structure makes possible to combine and recombine the rules and interpret them. Furthermore, the present architecture maps high-dimensional raw input into a lower-dimensional rule space which reduces the number of Q-values to estimate. In addition, we propose a new technique to update the Q-values of Sarsa that relies on sub-states as generalization mechanism.

To further improve data efficiency and generalization ability of Sarsa-rb, we propose the idea of sub-states. The key insight behind our new mechanism is to exploit the similarities among the rules to help the agent to reason in similar situations. At each iteration, it takes the current observation and instead of updating only one Q-value, the Q-values sharing similarities are also updated, leading to a significant speed-up. Finally, to take advantage of the sub-states, we adapt the original learning rate update and eligibility trace $\lambda$ used in Sarsa, Sarsa-rb($\lambda$).

### 7.3.1 Rule Extraction

The purpose of this first stage is to extract the rules that can be used to represent and compress the observations of our environment. One common solution for extracting rules consists in extracting a set of meaningful features for each training example, and then creating a rule describing the overall observation. However, such an approach often has an abundance of irrelevant or redundant information. Moreover, describing a complex environment may require a huge number of rules, drastically increasing the training time of an agent whose internal representation is based on these rules. Rather than describing one observation with one rule, we propose to extract rules representing common and task-relevant patterns in the dataset. Hence, one observation can be represented by a combination of rules. By doing so, the depicted framework captures meaningful features and ignores irrelevant details, reducing the number of rules necessary to represent a complex environment. Moreover, during the agent's training, we can update the Q-values of states sharing common patterns/rules, significantly reducing the training time.

We present three methods to extract rules. One consists in manually creating them according to our knowledge about the task. Supervised and deep unsupervised extraction methods retrieve patterns from external sources of data. Specifically, the deep unsupervised rule generation method was designed to extract rules from visual inputs by taking advantage of deep learning [148] and extends the idea depicted by Garnelo *et al.* [16]. The core idea is to extract rules describing the environment based on local interactions between objects. These sources of data can be various such as annotated datasets similar to the current environment, or any datasets containing relevant information about the task. For instance, for a trading task we can use as external sources of data several stock market datasets from other companies. The intuition behind is that among other companies, we can extract patterns that are shared with the current task.

#### Manual Rule Extraction

One technique to address the rule generation relies on human or background knowledge about the domain. For instance, in a car driving task, such knowledge can be retrieved from traffic rules. Another possible application of our model is automatic trading, for which we can use expertise about time-series and stock markets. Moreover, several previous works about feature engineering can be integrated and combined such as candlestick patterns [149]. This stock-market analysis technique estimates the trend of the share price by identifying common patterns into time series.

#### Supervised Rule Extraction

In real-world environments, the rules can be automatically captured by supervised machine learning methods. We follow a similar idea of Mashayekhi *et al.* [150]. This method extracts the rules from a random forest [151], an ensemble of decision trees [152]. A decision tree consists of several nodes that branch to two sub-trees based on a threshold value on a variable. We call leaf nodes the terminal nodes. A single decision tree has a very limited generalization capability and a high variance [153]. Several ensemble models such

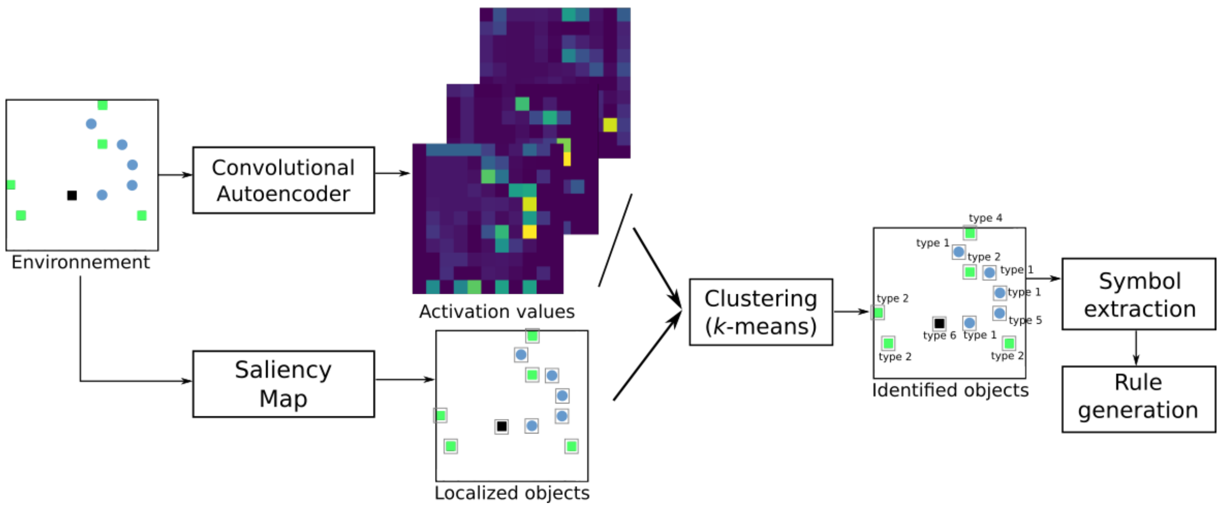


Figure 7.2: Deep unsupervised rule extraction method.

as random forest reduce the variance by building many trees and making prediction based on a consensus to among the decision trees. A simple tree traversal method can directly extract patterns from the trees. The recommended action associated to each pattern can be retrieved using simple heuristics such as depicted in Section 7.4.1, manually annotated, or, let empty without affecting much performance after convergence.

### Deep Unsupervised Rule Extraction

The goal of this method (Figure 7.2) is to extract symbols and then generate rules in an unsupervised manner, which represent local interactions between objects. This technique is well-suited for visual environments.

The first stage consists in extracting and recognizing the objects from the latent representation learned by a deep neural network. The next stage generates symbols that represent these objects. Finally, we construct a set of rules by estimating the relevance of each symbol: the position, the color, and the relative position of the objects.

The first stage extracts the objects within an image. We use a deep unsupervised neural network in order to extract features from images. Specifically, we train a convolutional autoencoder [141] and use the compress representation (i.e. latent representation) to identify where are the objects as well as recognize them. As shown by Garnelo *et al.* [16] objects are characterized by high activation values throughout the layers of the compressed image representation. Note that with natural images, state-of-the-art methods in unsupervised learning such as PixelVAE [154] can be trained to learn a useful latent representation. In addition, similar objects share a similar activation spectra independently of their position in the image. Therefore, we can extract the position of the objects by extracting the areas with high activation values. One way to classify an area as containing an object or not relies on a fixed threshold. However, using a threshold [16] requires manual tuning to fit the environment. Instead, our technique relies on salient areas of the original image. For each pixel, the salient value is scaled between 0 and 1 and the probability that a pixel is considered as part of an object is equal to this salient value. The corresponding activation

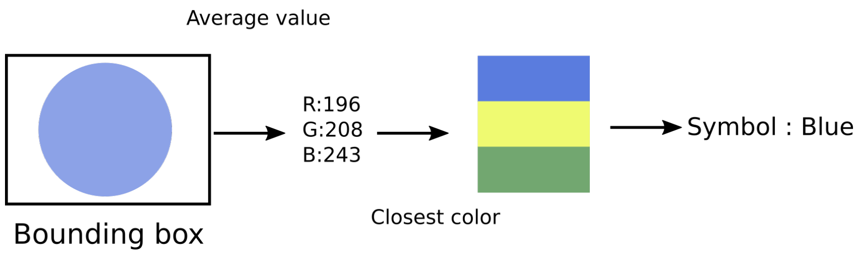


Figure 7.3: Overview of the symbol color extraction method.

values in the compressed representation are then extracted and considered as potentially representing an object.

The second stage characterizes the objects by comparing their activation spectra. This is done by using an unsupervised clustering algorithm. Given the short length of activation spectra, we applied a $k$-means method [155] to group the objects according to their spectra. The idea behind is that similar objects have similar activation spectra and therefore will belong to the same clusters. A $k$-means method is then trained using the spectra extracted from the external sources of data (stage 1). When a new image is observed by the agent, the image is processed following a similar pipeline, however, the $k$-means method is not retrained but only used to predict the label of each identified objects. Since we build an end-to-end model without supervision, we label the new objects with the cluster ID number.

The information extracted at the end of this stage is the position of the objects within the images, their label represented by an integer as well as their bounding boxes. In addition to the type of objects and their positions, we construct two other symbols: the color of the objects and their relative position.

The first symbol represents the colors of the objects. The color of an object is typically the average RGB value of the pixels within the bounding box. This bounding box was found by using the salient map areas. Given the average RGB values, we assign the symbol as the closest color among: *red, blue, green, yellow, red, blue, black, gray, purple*. As can be seen in Figure 7.3, the number of possible colors can be augmented to fit the complexity of the task.

The second symbol describes the neighborhood of the objects. We encode the positions of objects relative to other objects within a radius $r$. This approach is justified by the common sense that the interactions of an object with its closest objects are more likely to have an impact on the reward than the interactions with farther objects. To do this, we divide the neighborhood of an object into areas of 45 degrees and for each one, we store the label of the closest object. In case there is no object in any direction within the radius $r$, we store $-1$. The final representation is an array, the concatenation of all the objects for each angle. Figure 7.4 depicts the process of encoding the relative object positions. There is only one object within the radius $r$ represented by the light gray circle around the agent (dark gray pixel). Since its relative angle is around 330 °, the only value different value of $-1$ in the array is 2, the label of the object.

Once the symbols are extracted, the last stage is to generate a set of rules describing the objects within frames. Note that we only take into account the observed frames followed

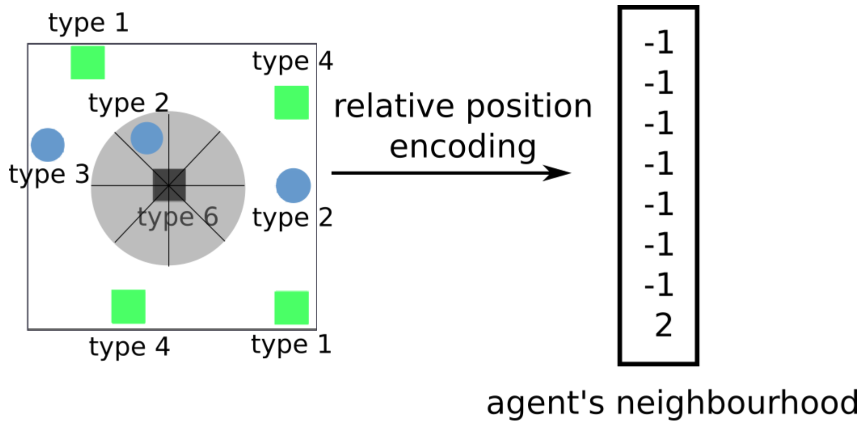


Figure 7.4: Relative position extraction method. The objects within a radius $r$ are taken into account and ther clostest objects are saved as symbols according to their angle with the considered object.

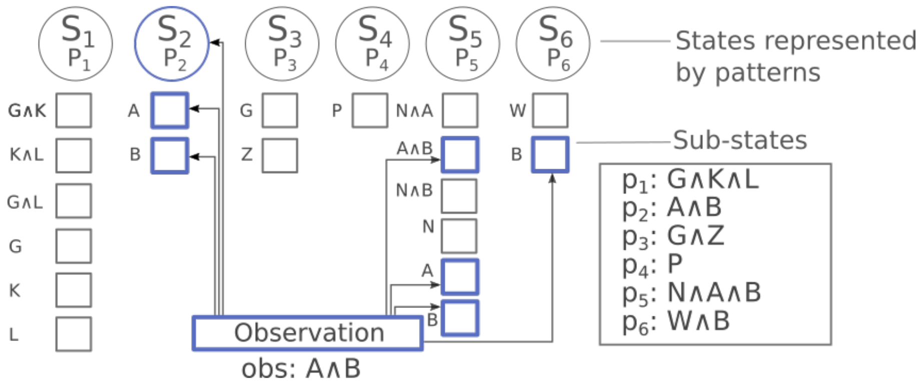


Figure 7.5: An illustration of the update of the Q-function. The Q-values of the active state $s2$ and its sub-states are updated. The sub-states sharing similar information with $s2$, in blue, are also updated.

by a reward greater than zero or negative. This method is justified by the idea that other symbols are not relevant since they don't have an immediate impact on the rewards received by the agent. Given the set of symbols, we construct the associated conjunctions of variables $symbol_1 \land symbol_2 \land ... \land symbol_n$. We refer to them as patterns since each one describes an image. The C most frequent patterns are kept and used to describe frames as rules. These generated rules are then used to train our algorithm, Sarsa-rb($\lambda$). Please note that in future work with richer environments, the proposed method can be easily modified to integrate more sophisticated symbols based on simple prior assumptions about the target environment.

### 7.3.2 Learning Algorithm

Once extracted, we use the rules in the reinforcement learning stage. The Sarsa algorithm maintains a parametrized Q-function which maps the states to their Q-values:

$$Q : S \times A \to \mathbb{R} \tag{7.1}$$

Instead of extending Sarsa to continuous state space by discretizing it, our representation relies on a rule-based representation. We propose to learn a Q-function which maps the state representations $\bar{S}$ to their Q-values:

$$Q : \bar{S} \times A \rightarrow \mathbb{R} \tag{7.2}$$

We propose to represent abstract states $\bar{S}$ by the rules extracted in stage 1. A rule $r$ associates a pattern $p$ to a recommended action, which we denote as $a_R$

$$\{\phi(s_t) = r | r : p \rightarrow a_R\} \tag{7.3}$$

where the pattern $p$ is used as the *activation function* of the state. Specifically, a state is active when the associated pattern $p$ satisfies the current observation. In addition, we improve the Q-value initialization $Q(\phi(s), a)_{t=0}$ by using the action recommended $a_R$ by the pattern. The recommended action belongs to the recommended action space $A_r$, where we set $A_r = A$ to ensure generality. Note that recommended actions are only used to guide the agent at the start of learning, and then the Q-values are learned from agent's interactions with the environment.

As depicted before, a rule associates a pattern $p$ to an action $a$, $p \rightarrow a_R$. A pattern $p$ is an arbitrarily complex conjunction of variables. The variables represent significant events in the task. For example, in a task involving driving a car, a variable could be *(speed between 20 and 50 km/h)* and an example of pattern is (*(speed between 20 and 50 km/h)* $\wedge$ *(pedestrian crossing the road)*). Finally, the rule which links this pattern to an action (e.g *brake*, *turn left*, etc.) could be:

$$\textit{(speed between 20 and 50 km/h)} \wedge \textit{(pedestrian crossing the road)} \quad \Longrightarrow \quad \textit{brake} \tag{7.4}$$

When the agent receives an observation, the active state is the state for which its associated pattern is satisfied, in other words, all its variables are active. Since no pattern is always satisfied, we added an "empty" state. This is the default state, active regardless of the input. Note that we capture only important information by filtering out irrelevant rules, the less frequent ones.

To improve generalization capability of our agent, we propose a sub-states mechanism. The sub-states are constructed by augmenting each Q-value with an ensemble of sub-states. We define the sub-states as its sub-patterns, the combinations of the variables. For example, given two patterns $p_1$: $A \wedge B \wedge C$ and $p_2$: $B \wedge C \wedge D$ the sub-patterns are $A \wedge B, B \wedge C, A \wedge C$ and $B \wedge C, C \wedge D, B \wedge D$ respectively. These two patterns are similar and share one sub-state $B \wedge C$, which indicates that the associated Q-values may be similar.

The Figure 7.5 shows our model structure. To simplify the representation we show the structure for only one action, however, the agent maintains a parametrized Q-function for each possible action. In summary, the states of Sarsa are replaced by rules. Their associated patterns are used as activation functions of the states. In addition, each state is augmented with one or more sub-states represented by the squares. The state $\phi(s_2)$ represented by the pattern $p_2 : A \wedge B$ has two possible sub-states corresponding to the two possible sub-patterns $A$ and $B$.

In Sarsa, the Q-values are uniformly initialized. We introduce a new method to take advantage of prior knowledge by initializing the Q-values according to the recommended actions of the rules:

$$Q(\phi(s), a)_{t=0} = \begin{cases} \mathcal{N}(\mu, \sigma^2), & \text{if } a = a_R \\ 0 & \text{otherwise} \end{cases} \tag{7.5}$$

where $\mu$ is the mean, $\sigma^2$ the variance, and $a_R$ the action recommended by the rule associated with $\phi(s)$ (Eq 7.3). In order to accelerate learning by appropriately specifying initial Q-values, for a state $\phi(s)$, the initial value of the action recommended by the rule follows a normal distribution centered around $\mu$, greater than 0, to follow our prior knowledge. For the other Q-values, the agent starts out knowing nothing, they are initialized to zero.

This rule-based representation can now be used to learn an effective policy. Our contribution here is to propose a technique to jointly use prior knowledge and reinforcement learning to decrease training time of the agent (rule-based representation) and to avoid learning from scratch (Q-value initialization). Since the number of abstract states is $|\bar{S}|$ (i.e. the number of extracted rules), and the total of Q-values to estimate $|\bar{S}| \times |A| << |S| \times |A|$, our algorithm can be trained in large state space domains.

### 7.3.3 Estimation of the Q-values

Accurately estimate each Q-value would result in a very long training time due to the infrequent visit of most of the states. However, our estimation relies on a sub-states technique which aims to enable fast generalization across observations. We also provide modifications of eligibility traces and learning rate to take advantage of sub-states during the estimation of the Q-values.

#### Sub-states as Generalization Mechanism

We introduce a *sub-states* mechanism to improve data-efficiency and generalization ability of our reinforcement learning agent. The main drawback of TD algorithms is that only one Q-value is updated at each iteration, entailing that they learn slowly. These algorithms can be augmented using eligibility traces to propagate the reward to the previous states. This work introduces a novel approach to jointly update the similar states and back-propagate the reward to the previous states. The goal is to get most of the benefits of the shared information among the rules while keeping the rest of the Sarsa algorithm intact and efficient. In order to implement this mechanism, we augment each Q-value with its sub-patterns (Figure 7.5). Note that to limit the number of sub-states, we limit the size of the sub-rules to conjunctions of at least 3 variables.

We provide modifications to the estimation and update of the Q-values inspired by Sarsa to incorporate sub-states. Our estimation of a Q-value $Q^{'}(\phi(s), a)$ takes into account the Q-value itself $Q(\phi(s), a)$ and the value of its sub-states. Intuitively, this estimation improves generalization among states by incorporating Q-values of similar states and

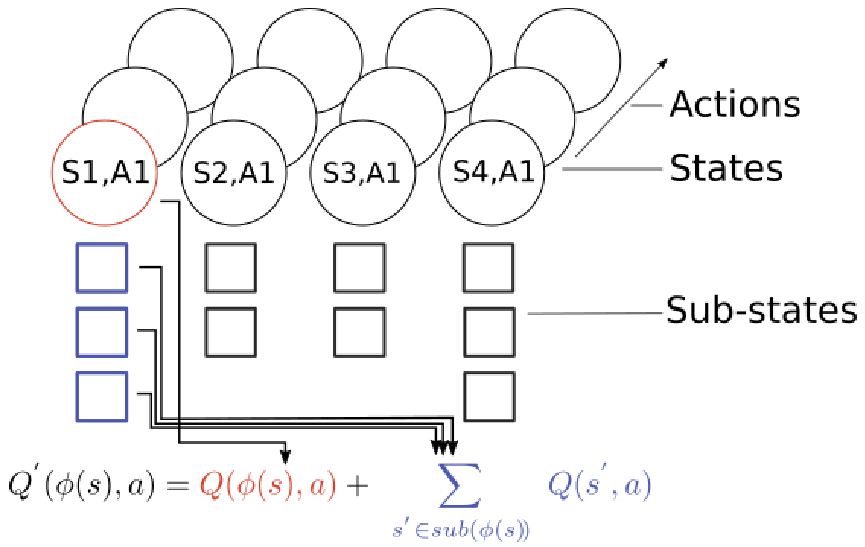

Actions
States
Sub-states
S1,A1
S2,A1
S3,A1
S4,A1
$Q'(\phi(s),a) = Q(\phi(s),a) + \sum_{s' \in sub(\phi(s))} Q(s',a)$

Figure 7.6: Estimation of a Q-value, $Q'(\phi(s),a)$, with the sub-states technique. In addition to the Q-value $Q(\phi(s),a)$ itself, the sub-states values $Q(s',a)$ are taken into account.

sub-states encountered previously:

$$Q'(\phi(s),a) = Q(\phi(s),a) + \sum_{s' \in sub(\phi(s))} Q(s',a) \tag{7.6}$$

with $sub(\phi(s))$ the sub-states of a state $\phi(s)$. We modified the update rule of states to incorporate sub-states:

$$Q'_{t+1}(\phi(s),a) = Q_t(\phi(s),a) + \alpha_{Q'(\phi(s),a)t}[r_{t+1} + \gamma Q'_t(\phi(s)_{t+1},a_{t+1}) - Q'_t(\phi(s)_t,a_t)]E_t(\phi(s),a) \tag{7.7}$$

and for sub-states $s'$, is defined as:

$$Q_{t+1}(s',a) = Q_t(s',a) + \alpha[r_{t+1} + \gamma Q_t(s'_{t+1},a_{t+1}) - Q_t(s'_t,a_t)]E_t(s',a) \tag{7.8}$$

where $E_t$ refers to the eligibility trace for a state-action pair, and $\alpha_{Q'(\phi(s),a)t}$ indicates the learning rate specific to the updated Q-value (Section 7.3.3). Our approach to backpropagate the rewards to all similar Q-values is to increment the eligibility traces of the similar sub-states. We illustrate the Q-value estimation process for a simple example in Figure 7.6. $Q(s',a)$ refers to the estimation of the value of the sub-state $s'$ given the action $a$. Adding this term grounds the values of the unvisited states, and makes the value induced by the values of the similar visited states.

A frequent and early update of the sub-states turned out to be critical in fast estimation of the Q-values, inducing a much faster training.

## Eligibility Traces

Directly implementing Sarsa-rb is proved to be slow learning in environments with sparse rewards. Our method, Sarsa-rb($\lambda$), is derived from Sarsa($\lambda$). Adding n-steps returns helps to propagate the current reward $r_t$ to the earlier states. We allow a propagation of $r_t$ to the earlier sub-states by changing their eligibility traces. The idea behind is that a sub-state similar to the current state is likely to get a similar reward by following the same action. The update of the current state $\phi(s)$ remains unchanged from Sarsa($\lambda$):

$$\begin{cases} E_t(\phi(s),a) = \lambda E_{t-1}(\phi(s),a) + 1 & \text{if the current state is } \phi(s) \\ E_t(\phi(s),a) = \lambda E_{t-1}(\phi(s),a) & \text{otherwise} \end{cases} \tag{7.9}$$

Similarly for sub-states:

$$\begin{cases} E_t(s',a) = \lambda E_{t-1}(s',a) + e^{-sim(s',\phi(s))}, & \text{if } s' \text{ is a sub-states of } \phi(s) \\ E_t(s',a) = \lambda E_{t-1}(s',a) + \frac{(e^{-sim(s',\phi(s))})^2}{P}, & \text{for the sub-states sharing} \\ & \text{at least 2 variables} \\ E_t(s',a) = \lambda E_{t-1}(s',a) & \text{otherwise} \end{cases} \tag{7.10}$$

where $E(\phi(s),a)$ represents the eligibility trace for state $\phi(s)$, $E(s',a)$ is the eligibility trace of sub-state $s'$ for a given action $a$, and $sim(s',\phi(s))$ denotes the similarity score between the sub-state $s'$ and the state $\phi(s)$. We define the similarity function as the number of different variables between a sub-state $s'$ and a state $\phi(s)$, $sim(s',\phi(s)) = |s' \cup \phi(s)| - |s' \cap \phi(s)|$. We bounded the score between 0 (identical) and 1. Note that we only take into account the sub-states sharing at least two variables.

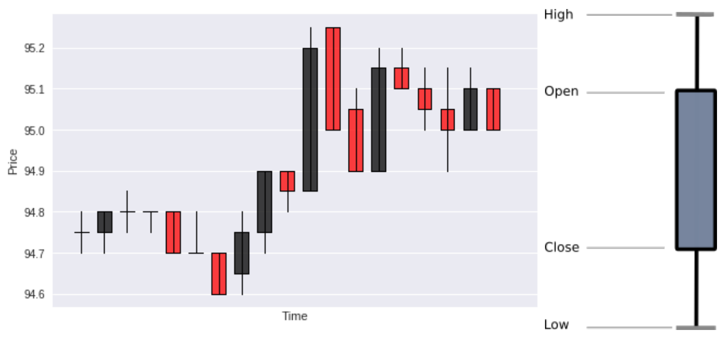
Since sub-states are often updated, we avoid exploding eligibility trace values by adding an exponential decay, and a constant $P$ which determines the scale of update signal. Intuitively, a high value decreases the update of the sub-states sharing only a few similar sub-patterns with the current state. Note that similarly to the original idea, when the eligibility trace of a sub-state $E_t(s',a) \approx 0$, the associated Q-value is not updated, but is always taken into account to estimate the Q-values of other states and sub-states. Updates performed in this manner allow estimating Q-values more accurately. Our experiments also suggest that sub-states technique decreases the number of necessary visits to accurately estimate Q-values and yields faster convergent policies.

### Adaptive Learning Rate

The linear learning rate $\alpha$ in regular Sarsa assumes that the states are equally visited. Obviously, this assumption is no longer valid. One approach to this problem is to change the update rate of the states according to the frequency of visit of the states and their sub-states. We turn to a learning rate specific to each Q-value $Q'(s_t,a_t)$:

$$\alpha_{Q'(\phi(s),a)t} = \frac{1.0}{(1.0 + visit(\phi(s),a)_t + \sqrt{visit(sub(\phi(s)),a)_t})} \tag{7.11}$$

We add the term $visit(\phi(s),a)_t$ which refers to the number of visits of the Q-value $Q(\phi(s),a)$. The term $sub(\phi(s))$ defines the ensemble of the sub-states of the state $(\phi(s))$ and $visit(sub(\phi(s)),a)_t$ is the total number of visits of all its sub-states. A state with its sub-states $sub(\phi(s))$ often visited will be estimated more accurately than a state without sub-states and hence doesn't need to be updated much.



(a) An example of OHLC chart (open, high, low, close)

(b) Strucure of one observation

Figure 7.7: Example of a sample of data from the environment. The left plot shows the time series and the right plot is the structure of one observation.

## 7.4 Experiments

For our experiments, we use two environments that both follow the Open-AI Gym structure [156]. The first environment is a trading task from real stock market data. This task involves large continuous state spaces. We use this task to compare the two rule creation methods (manual rule generation and supervised rule generation). The second task is more challenging and consists of a visual navigation problem using images as the input of our model. This task is used to evaluate our deep unsupervised rule extraction technique. To ensure a fair evaluation, we pretrained the baseline algorithms on the datasets used to generate the rules.

### 7.4.1 Trading Tasks

We evaluated our agent, Sarsa-rb($\lambda$), on the OpenAI trading environment, a complex and fluctuating simulation from real stock market data (Figure 7.7a). The observations (Figure 7.7b) are given to the agent in the form of a vector of 4 continuous variables that were recorded during a one minute interval: the open price, the close price, and the highest/lowest price. The action set consists of 3 actions: *Buy*, *Hold* and *Sell*. The reward is computed according to the win/loose after buying or selling. Each training episode is followed by a testing episode to evaluate the average reward of the agent on another subset of the same stock price. Each episode was played until the training data are consumed, approximatively $10^5$ iterations.

Our system learns to trade on a minutely stock index. In total, we used 4 datasets with a duration varying between 2 years and 5 years. One stock index was used to train the agent, and the other three as external sources of knowledge to generate the rules. To fairly evaluate the model, we trained it on 80% of the training examples and evaluate the performance on the remaining 20%. We ran a grid search over the parameters to initialize the Q-values and found that $\mu$ the mean equals to 0.25 and $\sigma$ equals to 0.2 were the best parameters. We use $P = 100$ as decay factor of eligibility traces. In the case of manually created rules, we first compute the percentage increase in the share price 14 days later

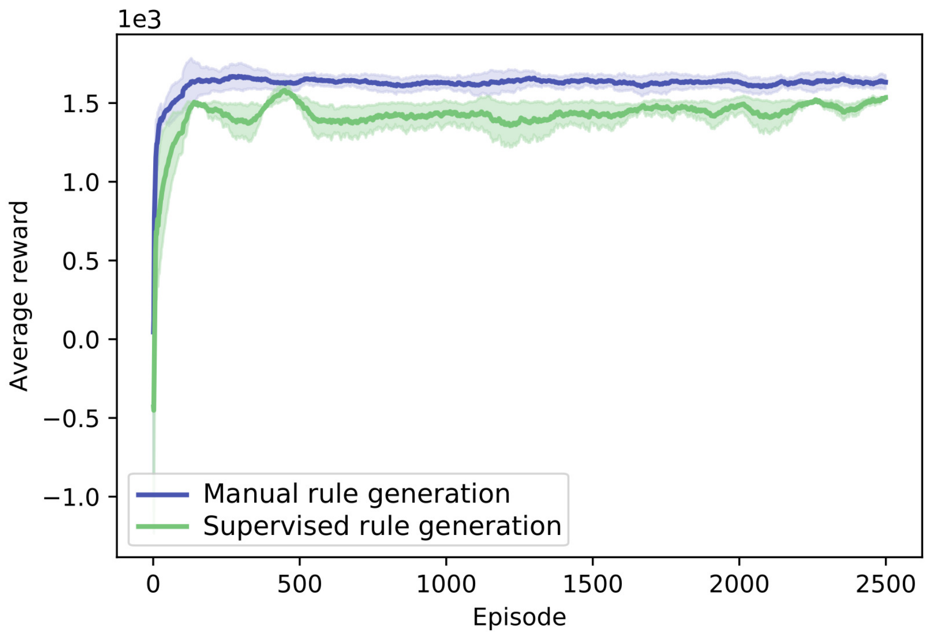


Figure 7.8: Performance curves of Sarsa-rb using a selection of techniques to create the rules: background knowledge based (blue) and extracted from a random forest (green).

and then estimate an optimal action associated with each pattern. In total, we took into account 40 candlestick patterns.

We follow a simplified technique used by Mashayekhi *et al.* to generate the rules from a random forest [150]. Briefly, we extract the patterns top to bottom and filter the patterns to avoid redundancy. To construct the trees, we automatically annotate 6000 samples into 3 classes. Each sample is the aggregation of the last 5 prices. We labeled the dataset according to the price $p_{diff}$ increase 14 days later ($p_{diff}$ >=0.5%, $p_{diff}$ <=-0.5%, 0.5%< $p_{diff}$ >-0.5%) to train a random forest. We compute $p_{diff}$ as the average between the open and close price. In order to limit the number of rules and since the impact on accuracy was minimal, we build 20 trees with a maximum height of 4. In total, we retrieved 855 rules. For each pattern, the predicted class (i.e. the trend of the stock market) was used to recommended an action.

### Rule Creation Performance

First, we evaluate the two rule creation techniques discussed in Section 7.3.1 and their impact on the average reward. After creating the rules on 3 stock prices, we obtained between 855 and 3240 rules, resulting in a maximum of $3240 \times 3$ Q-values to estimate. Figure 7.8 shows the performance of Sarsa-rb with its internal states created using the different methods. The results are obtained by running Sarsa-rb without sub-states and eligibility traces, and the same hyper-parameters to allow a controlled experiment focused on rule effectiveness.

As can be seen in Figure 7.8, the agent trained using background knowledge based rules achieves the highest score on average. Performance of the agent was mainly affected by

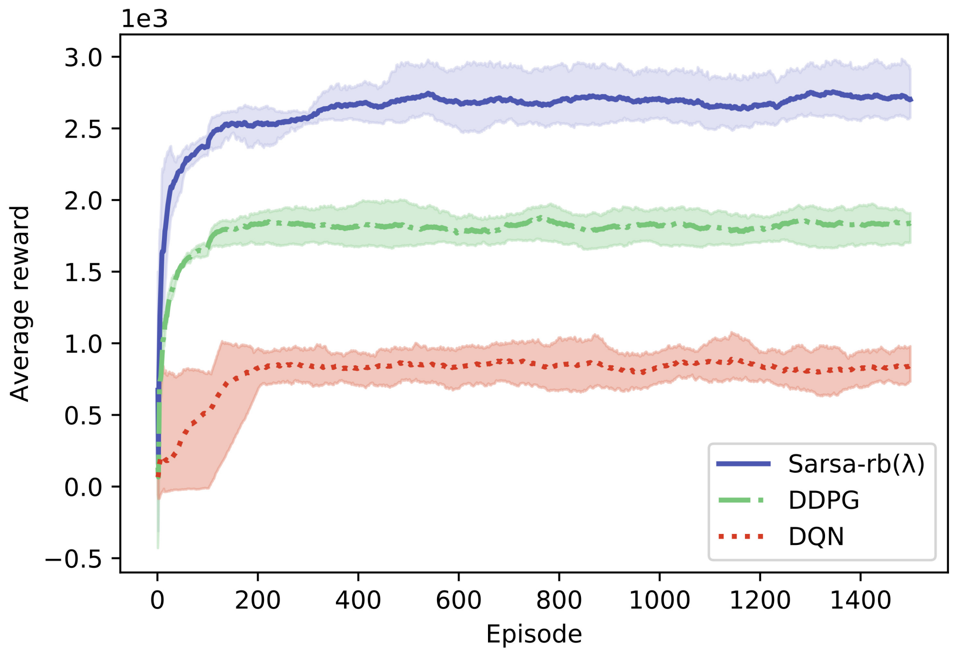


Figure 7.9: Performance curves for a selection of algorithms: original Deep Q-learning algorithm (red), Deep Deterministic Policy Gradients algorithm (green) and Sarsa-rb($\lambda$) (blue).

the quality of the rules, however, we could not establish a link between the number of rules and the quality of the model. Furthermore, the rules can be combined to take advantage of each technique - we can make use of rules extracted by different algorithms, which is impossible when directly representing the policy by a neural network.

### Overall Performance

We evaluated the performance of the learned policy using the proposed sub-states mechanism. We compare Sarsa-rb($\lambda$) with two baselines, a deep recurrent Q-learning model[124] and a DDPG model [4]. DDPG method was shown to perform well in trading tasks [157, 158]. Since the feasible trading actions is in a discrete set, the output of the actor is a vector of $n$ real numbers. The action to take corresponds to the index of the maximum value.

For this evaluation, we individually tuned the hyper-parameters of each model. We decreased the learning rate from $\alpha = 0.3$ to $\alpha = 0.0001$, and increased the eligibility trace from $\lambda = 0.8$ to $\lambda = 0.995$, then used $\lambda = 0.9405$ and $P = 100$. Each parameter value was sampled within the given interval, and the algorithm evaluated using those values. We use an $\epsilon$-greedy policy as the behavior policy $\pi$. It chooses a random action with a probability $\epsilon$ and an optimal action with a probability $1 - \epsilon$. In our experiments, $\epsilon$ is set to 0.01. The plots are averaged over 5 runs. Finally, we used the external knowledge based rules as the states of Sarsa-rb($\lambda$). Note that since Sarsa or Sarsa($\lambda$) cannot learn from continuous state spaces, we don't report their performance.

We report the learning curve on the testing data in Figure 7.9. Sarsa-rb($\lambda$) always achieve

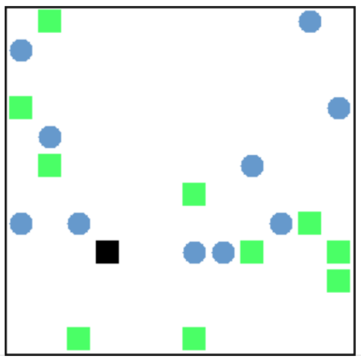


Figure 7.10: A board of the visual navigation environment. The agent is represented by a gray square, the food to collect by the blue circles and the walls by the green squares.

a score higher than DQN and DDPG. From Figure 7.9, it is clear that Sarsa-rb($\lambda$) improves over DQN - we observe after converging an average reward around 3.3 times higher. DDPG appears less fluctuating than Sarsa-rb($\lambda$) but also less effective.

The key concepts of our algorithm are sub-states and their capacity to transfer knowledge. By representing the observations with rules we can easily understand when a situation is analogous to a previously encountered set of situations. Moreover, sub-states allow to transfer knowledge from previous similar situations and even to transfer knowledge from partially encountered events. In the current algorithm, this capability is achieved through a simple similarity measure, but in the future we aim to develop a more efficient similarity measure. Furthermore, it appears that grafting domain knowledge directly onto the agent's internal representation leads to better common sense reasoning - the agent exhibits capabilities for generalizing commonsense priors to unseen situations, resulting in higher performance. It turns out to be critical in complex environments where traditional forms of human guidance such as demonstrations are deemed infeasible but datasets are widely available.

### 7.4.2 Visual Navigation Tasks

As a benchmark for our visual rule extraction method, we developed a simple navigation task. The environment consists of a board with obstacles, and objects of different shapes and colors (Figure 7.10). The agent learns to collect food, represented as blue circles, and to avoid walls, represented as green squares. The agent has to learn to navigate using one of the possible actions (*up*,*right*,*down*,*left*). Encountering a food results in a positive reward (+5), a wall, a negative reward (-5) while moving to an empty cell results in a negative reward (-0.2).

The goal of the agent is to collect the five "food" objects randomly positioned across the map. Eating the last food object results in a positive reward (+20) whereas going out the map gives a negative reward (-5) and restarts the game. The agent receives as input a 2D RGB image corresponding to what the agent sees around it. The raw frame is resized to $84 \times 84$ pixels. To decrease the storage cost of the images we convert the image scale from $0 - 255$ to $0 - 1$. Note that we limit the maximum time to 1 minute for the agent to find a strategy to get a maximum reward. The rules were extracted from a dataset

Table 7.1: The table evaluates performance in terms of average reward. We compare Sarsa-rb($\lambda$) with Deep Q-learning (DQN) and Proximal Policy Optimization (PPO) algorithm.

| | 20000 episodes | 80000 episodes | 160000 episodes |
|---|---|---|---|
| Sarsa-rb($\lambda$) | 2.01±0.23 | 9.4±1.18 | 16.6±1.4 |
| DQN | -3.2±0.85 | -3.3±0.83 | -3.2±0.83 |
| PPO | 1.8±1.0 | 8.9±3.0 | 13.2±3.4 |
| DQfD | 5.2 ± 0.33 | 9.2 ± 1.06 | 12.1 ± 1.01 |

of 100 000 images of the environment following the deep unsupervised rule extraction method. We trained a convolutional autoencoder on 100 000 random boards in order to extract features from images. To have an accurate representation of the environment, the position of the agent varied randomly as well as the objects whose numbers were randomly selected. For our experiments, the input of the neural network consists of a 84×84×3 RGB image. The first 3 layers convolve with the following parameters (filter: 64,32,32, kernel size: 3×3,3×3,3×3). Each one was followed by a 32×32 pooling layer. The last 4 layers are the corresponding decoding layers. We use the compressed representation, the middle layer, to recognize the objects. Since we do not assume strong prior knowledge about the task, we set the number of clusters of the $k$-means model to $\boldsymbol{K} = 8$. Finally, We kept the 6000 most frequent patterns to build the rules.

### Overall Performance

We compared our agent trained on the visual navigation task to several methods from the literature, which are considered to be effective for visual problems. We compared our algorithm against tuned versions of proximal policy optimization (PPO)[38], deep q-learning (DQN), and, deep q-learning from demonstrations (DQfD) [138]. DQfD pretrains a DQN agent with demonstrations as source of prior knowledge. For our experiments, we had a human player play the game. In total, 6237 transitions were recorded and used to pre-train DQN. We trained 5 agents for each algorithm with the same settings. The average reward is shown in Table 7.1. Note that for these two deep learning algorithms, we used the same policy network architecture as used by Mnih *et al.* [1].

The depicted approach outperforms standard DQN in terms of convergence speed and quality of policy. Moreover, Sarsa-rb($\lambda$) is more stable than PPO. Although DQfD achieves a higher average reward at the start of learning, our model converges towards a better policy. We should emphasize that our method requires much less human effort compared to DQfD since we eliminate the need to expressly create domain knowledge to solve the task being learned (e.g. demonstrations). Besides, our technique preserves interpretability while learning a good policy, making it more suitable for real-world tasks. By representing the observations of the environment with understandable rules, we produced a humanly-comprehensible model.

## 7.5 Discussion

We discussed learning from existing sources of knowledge in the context of sample-efficient reinforcement learning. By introducing large amounts of existing prior knowledge into reinforcement learning architectures, our agent can learn interpretable and compact representations of the environment without specific human engineering. Besides, since its internal states are represented by first-order logic rules, we can analyze which objects, as well as symbols, are involved in the choice of the action and their importance. Specifically, the Q-values can be interpreted to evaluate the weight of each feature on the choice of the action. Finally, we can control the importance of knowledge transfer by giving more or less importance to sub-states (i.e. generalization capability). To support these claims, we presented an exhaustive evaluation on time series tasks and visual tasks. We observed that our system dramatically outperforms neural network-based methods in a range of different domains - grafting domain knowledge onto the agent's internal representation improves various forms of common sense reasoning. We also studied the effect of rules mined by analyzing the latent representation learned by a deep neural network, as well as rules extracted from random decision trees. The evaluations demonstrated that our approach efficiently learns and generalizes domain knowledge - rules, to setups which were not demonstrated in the datasets. We have also shown its ability to solve complex tasks with continuous state spaces, and exceeds baseline agents in terms of overall performance and convergence speed.

One limitation of our approach is that we passively access to human guidance. That is, there is no active cooperation during the training process between the teacher and the student (the agent). It would be interesting to investigate ways for an agent to actively cooperate and share insights with the teacher. Ensuring that the agent's goal matches the provided domain knowledge is necessary to scale the method to practical problems characterized by changing complex scenarios. Additionally, we believe that the proposed method should also enable the agent to identify the need for and communicate for specific domain knowledge throughout the agent's training.

# Chapter 8

# Active Goal-Driven Learning

In chapters 6 and 7, we have seen how utilizing human guidance can help us reduce training time and improve final performance. However, these methods and most prior work are still restricted to the setting in which there is no active cooperation between the teacher and the agent. In other words, the agent cannot request feedback from the teacher when it struggles or based on its knowledge of the task. Besides, in real tasks, the agent cannot cope with the changes in the environment - learning from a fixed set of demonstrations may be impracticable due to lack of state coverage or distribution mismatch (when the learner's goal deviates from the demonstrated behaviors).

This chapter presents a goal-conditioned reinforcement learning algorithm that can be applied to such problems. Crucially, we introduce the concept of active goal-driven demonstrations to query the demonstrator only in hard-to-learn and uncertain regions of the state space. The algorithmic framework relies on a novel form of human feedback called *goal-driven demonstrations*, which are easier and more intuitive to demonstrate for a teacher than full demonstrations while precisely matching the agent's needs.

## 8.1 Introduction

A line of work for overcoming the above-mentioned issues is goal-conditioned learning, a form of self-supervision that constructs a goal-conditioned policy to learn how to reach multiple goals [39, 40]. This idea was extended in Hindsight Experience Replay (HER) [41] to artificially generate new transitions by relabeling goals seen along the state trajectory. Nevertheless, it may still require a large amount of data to capture complex policies. Since it is often unrealistic to expect an end-to-end reinforcement learning system to rapidly succeed with no prior assumptions about the domain (i.e. learning a task from scratch), several methods have attempted to introduce external supervision into reinforcement learning systems. For instance, an approach [159] leverages human preferences as feedback signal to reduce the amount of human involvement. Nonetheless, it was shown that preferences are an inefficient way of soliciting information from humans [159]. In the context of reinforcement learning, the most common form of external supervision is imitation learning. Imitation learning seeks to learn tasks from demonstrated state-action trajectories [160, 22]. For instance, DQFD [114] improves initial performance by pre-training the policy with demonstrations. However, learning from human demonstrations suffers from three problems: (1) it is hard to obtain a broad state coverage of task-relevant regions from trajectories demonstrated without a specific goal, (2) it usually has an abundance of irrelevant or redundant information, (3) it assumes that the learner's goal matches the teacher's demonstrated behaviors. Additionally, most imitation learning algorithms learn policies that achieve a single task.

In this chapter, we contribute an active goAL-conditioned approach (goAL) that drastically reduces expert workload by incrementally requesting partial demonstrations towards specific goals, *goal-driven demonstrations*. Contrary to pure demonstrations, goal-driven demonstrations do not aim to demonstrate the overall task or all possible situations. Instead, goal-driven demonstrations fulfill particular goals that are actively selected based on the agent's knowledge about its environment. Especially, the proposed framework allows an agent to jointly identify states where feedback is most needed and communicate for specific domain knowledge throughout the training process. Our method relies on an *imitator* network trained to clone a novel form of human feedback: *goal-driven demonstrations*. Given its prediction, we augment the policy loss with a simple auxiliary objective. Rather than using a fixed set of demonstrations, goal-driven demonstrations are actively queried based on the imitator's confidence and the ability of the agent to reach the goal being pursued. We build and compare two techniques to estimate the agent's confidence: 1) Bayesian-confidence, 2) quantile-confidence; and study a relabeling strategy that extracts additional information from the demonstrated trajectories. We found goal-driven demonstrations to be easier to demonstrate for a human than full demonstrations, while significantly increasing the value information of the queries by matching the agent's needs. We further propose a method for prioritizing the sampling of important goals - in places where the disagreement between the expert and the policy is large.

We evaluate our approach on several tasks from the Mujoco benchmark suite [74, 161] including Fetch and ShadowHand. Experimental results show that goAL outperforms previous approaches in most of the tasks with a significantly lower number of demonstrations. We also show that our method can generalize to unseen states while being robust to incomplete or noisy demonstrations. Remarkably, goAL produced agents that exceeded the expert performance in multiple tasks.

The main contributions of this work are summarized as follows:

- We propose a new framework, Active Goal-Driven Learning, which is the first work to use active goal-driven demonstrations to the best of our knowledge.
- We contribute a method to query the demonstrator only in states where the agent struggles and is not confident, maximizing the expected value of information of the queries and drastically reducing human effort.
- We propose two novel confidence-based query strategies to evaluate the confidence along a state-action trajectory.
- We contribute a goal-sampling technique that maximizes the agent's learning progress.
- We provide a comprehensive comparison between the proposed methodology and a number of baselines, evaluated on complex robotic tasks.

## 8.2 Related Work

Methods for providing external supervision largely divide into two categories: imitation learning and learning from interactive human feedback. Our work builds upon goal-conditioned learning combines aspects of both. We briefly introduce these techniques in this section.

**Imitation learning and RL.** Multiple works seek to combine deep reinforcement learning with human demonstrations. For instance, DQfD [114] pre-trains a Q-learning agent on the expert demonstration data. This idea was extended to handle continuous action spaces such as in robotic tasks [162], as well as to actor-critic architectures [163]. POfD [164] proposes to follow demonstrations in early learning stages for exploration, and then let the agent explores new states on its own. A recent follow-up [165] introduces an expert loss in DDPG [4] and proposes to filter suboptimal demonstrations based on the Q-values (Q-filter). It is assumed that there is a fixed set of demonstration data. In contrast, we are interested in actively sharing insights between the teacher and the agent. Therefore, we propose a novel imitation loss function which leverages goal-driven demonstrations and a goal-conditioned framework to actively request feedback to the teacher when the agent struggles, reducing both the training time and the number of demonstrations while ensuring that the goal-driven demonstrations match the agent's needs during the training process. In this work, we use the Q-filter method [165] in a goal-conditioned setting, and we further adapt it to filter transitions where the agent action is significantly better than the demonstrator action. Another solution is to represent a policy as a set of Gaussian mixture models [166]. However, they consider a fixed target goal setting, and the method is not directly applicable to continuous action spaces. In a different spirit, AlphaGo [2] trains a policy network to classify positions according to expert moves. A way of dealing with sparse rewards consists in introducing a curious replay mechanism and demonstrations [167]. In recent years, an emerging strategy combines generative adversarial networks and reinforcement learning (GAIL) [24]. However, GAIL was shown to suffer from the *drift prediction error* and instability during training. DAGGER [50] requests supervision at each step, and takes an action sampled from a mixture distribution of the demonstrator and the agent. The idea was extended in Deeply AggreVaTeD [168] to work in environments with continuous action spaces. Another method [169] constructs a goal-conditioned policy to visit similar states as the expert. That is, they employ the idea of discriminability as a central theme in building agents that can leverage demonstrations. Rather than solely using goals to condition the policy, we use goals to enable active cooperation between the teacher and the agent. Namely, we propose a novel form of human guidance, *goal-driven demonstrations.* Goal-driven demonstrations do not intend to cover all possible scenarios or demonstrate the overall task, but guide the agent to fulfill particular goals when the agent struggles, drastically reducing the number of required queries and being more intuitive for the demonstrator than pure demonstrations. In order to identify the need for specific domain knowledge throughout the agent's training, we also contribute a framework to identify areas where feedback is most needed based on the agent's confidence and its ability to reach the goal pursued. The selected goals are then sent to the teacher, enabling the demonstrator to understand the agent's needs. In addition, the imitation loss is used in a different way in our method; and we develop a different strategy for relabeling goal-driven demonstrations, which ensures that only optimal transitions are recorded.

The present work further differs by proposing a *prioritized goal sampling* method in order to focus on task-relevant goals, reducing the overall learning time of the algorithm. Another form of imitation learning is inverse reinforcement learning (IRL) [25] where a reward function is inferred from the demonstrated trajectories. IRL has been applied to several domains including navigation [142], and autonomous flight [22]. However, in many cases it is impractical to demonstrate long-term tasks, and most agents focus on learning a single task from a set of demonstrations. Finally, IRL algorithms assume that the observed behavior is optimal.

**Learning from interactive human feedback.** Most methods that focus on learning from interactive human feedback [170, 171, 172, 159, 173] query the human to drive learning. For example, TAMER [174] trains the policy from feedback in high-dimensional state space. The learner may receive feedback in the form of sequences of actions planned by a teacher [175]. Uncertainty-based query was used in [176] but is limited to DQN [1], limiting the possible applications of this method. In contrast, our method can be combined with most of off-policy RL algorithms; and introduces the idea of goal-driven demonstrations. Some authors [177] consider multiple demonstrators performing different tasks and the agent must actively select which one to request for advice. Another solution [178] is to block unsafe actions by training a module from expert feedback. However, it requires the expert to identify all unsafe situations by watching an agent play. To deal with the problem of query selection, it is possible to select sufficiently different unqueried data [179]. In this work, we propose to only request feedback to the supervisor in states where the agent is unsure and struggles.

**Goal-conditioned RL.** Goal-conditioned reinforcement learning [39] constructs a goal-conditioned policy to push the agent to acquire new skills and explore novel states. Universal value function approximators [40] samples a fixed goal at the beginning of each episode and rewards the agent when the current goal can be achieved. Nonetheless, selecting relevant goals remains an open problem. A solution [81] and its recent follow-up [61], proposed to generate increasingly difficult goals to drive the agent towards the final goal. The method [92] learns an embedding for the goal space using unsupervised learning and then choose the goals from that space. The recent work [94], Skew-fit, proposes an exploration objective that maximizes state diversity. The key idea is to learn a maximum-entropy goal distribution to match the weighted empirical distribution, where the rare states receive larger weights. Another line of work [93] focuses on goals that provide maximal learning progress. Our method, which builds on top of HER, provides an order of magnitude of speedup by taking advantage of very few goal-driven demonstrations. We further introduce a novel goal sampling strategy based on the disagreement with the demonstrator.

## 8.3 Goal-Driven Imitation

The challenges of injecting expert feedback into DRL are two folds. First, expert demonstrations are limited, which entails that the agent needs to efficiently leverage a small amount of demonstration data. Although a number of algorithms could in principle be used to learn from demonstrations, standard methods can suffer from poor performance. This can happen when the state coverage of the expert trajectories is too narrow, or due to

**Algorithm 3** Active Goal-Driven Learning (goAL)

```
1: Given:
     • an off-policy RL algorithm $\pi_{\theta_\pi}$                    ▷ DQN, DDPG, NAF
     • a replay buffer $R$ and an expert trajectory buffer $\Omega$
     • an imitator network $f_\theta$
     • a reward function $r_g : S \times A \times G \rightarrow \mathbb{R}$      ▷ e.g. $r_g(s,a,g) = ||s-g||_2^2$
2: Initialize $\pi_{\theta_\pi}$ and $f_\theta$
3: Initialize $R = \{\}$ and $\Omega = \{\}$                       ▷ initialize neural networks
4: for m=0,...,M do
5:     Receive a goal $g$ and initial state $s_0$
6:     for t=0,...,H-1 do
7:         Get action $a_t = \pi(s_t, g|\theta_\pi)$
8:         Execute $a_t$ and observe next state $s_{t+1}$
9:         Store transition $R = R \cup (s_t, a_t, s_{t+1}, g, r)$
10:    Save the current trajectory $\varphi = \{s_0, a_0, s_1, ..., , g\}$ and the final state $s_k = s_{H-1}$
11:    $C(\varphi) = \mathbb{1}[s_k \neq g]\mathbb{E}_{(s_t,g)\sim\varphi} c(s_t, g)$          ▷ Query Selection
12:    if $C(\varphi) > t_{qry}$ then
13:        Query the expert with $g$ as the target goal and receive $\tau$ =
           $\{(s_0, a_0), (s_1, a_1), ..., (s_k, g)\}$                  ▷ Query the demonstrator
14:        Relabel $\tau$ and store tuples in $\Omega$                    ▷ Expert Relabeling
15:        Fine-tune $f_\theta$ on $\Omega$
16:    for t=0,...,H-1 do
17:        Sample a set of additional goals $G \in \{s_{t+1}, .., s_{H-1}\}$ with probability $p(s,g)$ ▷
           Prioritized Goal Sampling
18:        for $g' \in G$ do
19:            Store transition $R = R \cup (s_t, a_t, s_{t+1}, g', r_g(s_t, a_t, g'))$
20:    for j=0,...,$N_{opt}$-1 do
21:        Sample a minibatch $B$ from $R$
22:        Calculate $L_C$ on $B$                              ▷ Goal-Driven Imitation Loss
23:        Update policy loss $L^D = \lambda_1 L - \lambda_2 L_e$ and optimize $\theta_\pi$
```

a discrepancy between the agent's goal and the demonstrated data. Second, demonstrating the entire task trajectory multiple times is an inefficient way of soliciting information from humans, lacking generalization to new target goals.

The framework of *active goal-driven learning* (goAL) provides us a mechanism to mitigate these problems by incrementally querying *goal-driven* demonstrations (Figure 8.1). Our approach (Algorithm 3) introduces human feedback into goal-conditioned learning via Hindsight Experience Replay. Specifically, the agent receives feedback in the form of short goal-driven demonstrations - the tutor is requested to reach a specific goal. We decide how to query goal-driven demonstrations based on the agent's needs and the expected value of information of the query, drastically reducing the number of required demonstrations. In the following section we describe the key components of our method.

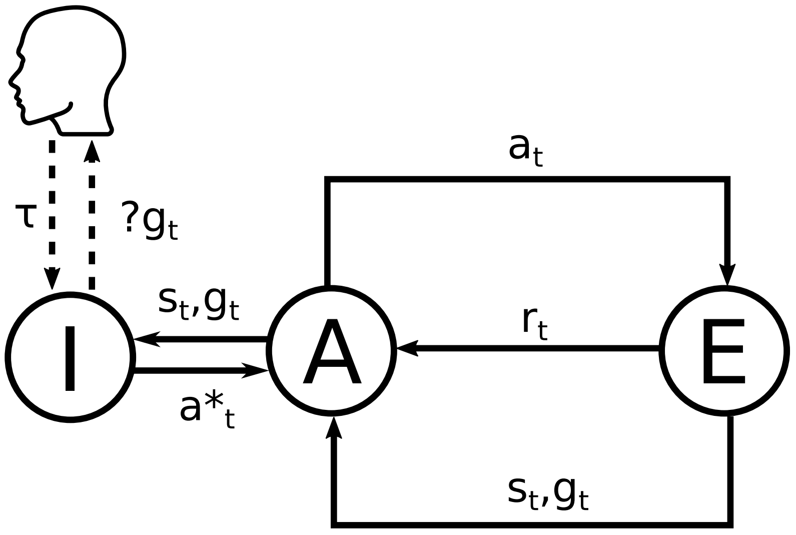


Figure 8.1: Active Goal-Driven Learning (goAL). The imitator predicts the action the demonstrator would have taken given a pair of state and goal $(s_t, g_t)$. When the agent fails to reach the goal being pursued $g_t$, a new demonstration $\tau$ with $g_t$ as the target goal may be queried.

### 8.3.1 Goal-Driven Imitation

We assume a small dataset of tuples $(s_i, g_i, a_i)$ extracted from expert trajectories, $\mathbf{\Omega}$. A trajectory segment (also called goal-driven demonstration) is a sequence of observations and actions, $\tau = \{(s_0, a_0), (s_1, a_1), ..., (s_k, g)\}$, where $g$ indicates the goal pursued by the demonstrator. Our method involves an imitator network $f : S \times G \longrightarrow \mathbb{R}^n$ that mimics expert behaviors, parameterized by a set of trainable parameters $\theta$. The imitator network is trained with a regression loss $\bar{L}$: it predicts the action the demonstrator would have taken given a pair of state and goal $(s_i, g_i)$, $a_i^* = f(s_i, g_i|\theta)$. In the absence of domain knowledge, a general-purpose choice is to train $f_\theta$ with a regression loss, the mean-squared-error, $\bar{L} = \frac{1}{|\Omega|} \sum_{i=1}^{|\Omega|} ||a_i^* - a_i||_2^2$.

A contribution of this chapter consists in augmenting the policy loss with an extra term to accommodate the goal-driven expert data. Given a minibatch of $T$ transitions, the imitation loss is given by:

$$L_e = \frac{1}{T} \sum_{t=1}^{T} ||\pi(s_t, g_t|\theta_\pi) - f(s_t, g_t|\theta)||_2^2 \tag{8.1}$$

where $\pi$ is the current policy parameterized by $\theta_\pi$. In order to allow the agent to significantly outperform the demonstrator - deviate significantly from the expert demonstrations, we use a Q-filter function [165] in a goal-conditioned setting, which we extend to increase the gap between "optimal" and "sub-optimal" transitions. In order to ensure that a transition provided by the demonstrator is significantly better than the agent's policy, we propose to filter irrelevant transitions via: $\mathbb{1}_{Q(s_t, f(s_t, g_t), g_t) - (Q(s_t, \pi(s_t, g_t), g_t) - \eta|Q(s_t, \pi(s_t, g_t), g_t)|) > 0}$, where $\eta$ is a positive constant. This filtering enables our agent to improve significantly beyond the expert demonstrations.

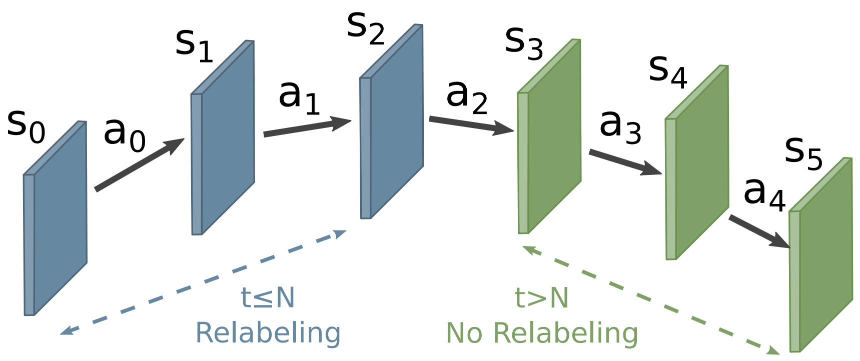


Figure 8.2: Relabeling strategy. The states within a reachability threshold $N$ are used as goals.

The overall loss used to update the policy network is a combination of two losses:

$$L^D = \lambda_1 L + \lambda_2 L_e \tag{8.2}$$

where $L$ indicates the loss function of any arbitrary DRL algorithms, and $\lambda_1$ and $\lambda_2$ weight each loss components. Adding this auxiliary objective provides the agent both the intention of the demonstrator and the ability to discover alternative strategies. Next, we show how to artificially increase the amount of demonstrations by relabeling expert data.

### 8.3.2 Expert Relabeling

To further enable sample-efficient learning in the real world, we present a relabeling strategy to artificially generate more expert data. In other words, expert relabeling is a type of data augmentation on the provided goal-driven demonstrations. As mentioned earlier, we collect expert demonstrations in the form of trajectories, $\tau = \{(s_0, a_0), (s_1, a_1), ..., (s_k, g)\}$, where $g$ is the goal being pursued. The idea behind this method is that in a state $s_i$, the associated action $a_i$ can be used to reach $g$, as well as new goals $\{s_{i+1}, .., s_k\}$ - the transition probability is not affected by the goal being pursued $g$. We found that selecting all the future states like done in goalGail [169] not an ideal solution, since distant goals can be reached using different actions. Besides, in the context of active learning, adding imaginary sub-optimal samples may create conflicts when the agent later on receives new feedback from the overseer. Instead, we restrict the creation of new imaginary samples to only short slices of the original trajectory. To do so, we propose to use the number of times-steps to approximate the distance between two states. We relabel future states when this distance is lower than a threshold $N$ (Figure 8.2): $\{s_{i+1}, .., s_{min(i+N,k)}\}$. By artificially generating new demonstrations, we can convert a single transition $(s, g, a)$ into potentially many valid training examples, which is particularly useful to decrease the number of queries to the demonstrator.

### 8.3.3 Query Selection

An important component in this method is query selection, in which the agent needs to decide which goals to query for demonstration. Our approach to decide when to query is based on 1) the ability of the agent to reach the goal pursued in the episode, 2) the confidence in the action prediction of the imitator network. By requesting demonstrations in hard-to-learn and low confidence situations, the depicted algorithm eliminates repetitive demonstrations of already learned goals of the task and provides human feedback to the agent when it struggles.

After experiencing each episode $\varphi$, we evaluate the confidence of the imitator along the state-action trajectory, if the goal was failed. For simplicity, a slight abuse of notation is made by using $\boldsymbol{C}$ to denote the query score. A score above a threshold $t_{qry}$ results in a goal-driven query - the demonstrator is requested to demonstrate how to achieve the failed goal. We formally define the overall function to estimate $\boldsymbol{C}$ as:

$$C(\varphi) = \mathbb{1}[s_k \neq g]\mathbb{E}_{(s_t,g)\sim\varphi}c(s_t, g) \tag{8.3}$$

where $s_k$ is the final state of the trajectory, $g$ is the goal being pursued, and $c$ is an estimation function of the confidence for the pair $(s_t, g)$. Every time a new demonstration is collected, the training transitions are recorded in $\boldsymbol{\Omega}$ and we make 50 epochs of training. Rather than using bootstrapping as in prior work, which adds a significant overload, we propose two novel methods (*quantile-confidence* and *bayesian-confidence*) to estimate prediction confidence, $c(s_t, g)$. The parametrization is discussed further in the next section.

### Quantile-Confidence

One common solution for estimating confidence in the prediction relies on ensemble-based uncertainty estimates, as done by Christiano et al. (2017) [170]. However, such an approach tends to be computationally expensive [159] and inaccurate when operating in the low data regime (with very few data). Instead, we develop a simple architecture for estimating confidence in the prediction, which has little/no computational cost, works with most existing imitation-based models, and is more robust against outliers. We propose to embrace deep quantile regression to estimate model confidence. Rather than only predicting the mean, the last layer of $f_\theta$ is used to predict each quantile separately. Assuming a set of goal-driven demonstration data $\boldsymbol{\Omega}$, we run a regression algorithm to train $f_\theta$ with a novel loss:

$$\bar{L}(q) = \frac{1}{|\Omega|}\sum_{i=1}^{|\Omega|}\rho(f(s_i, g_i|\theta) - a_i, q) \tag{8.4}$$

where

$$\rho(\varepsilon, q) = \begin{cases} q\varepsilon, & \text{if } \varepsilon \geq 0 \\ (q-1)\varepsilon, & \text{if } \varepsilon < 0 \end{cases} \tag{8.5}$$

where $q$ is the required quantile ($0 < q < 1$), and $a_i$ is the action the demonstrator took. We typically use (0.3,0.5,0.8) as quantiles. We can express *quantile-confidence*, $c(s_t, g)$, by measuring the prediction interval between the largest $q''$ and smallest $q'$ quantile, $c(s_t, g) = \left|f(s_t, g|\theta)_{q''} - f(s_t, g|\theta)_{q'}\right|$. Please note that we use the median quantile (q=0.5) in Equation 8.1.

**Bayesian-Confidence**

Imitator network confidence can also be modeled using bayesian models. However, in the context of RL, their computational cost can be prohibitive. This problem can be mitigated by using an estimation of Bayesian inference. It was shown that the use of dropout can be interpreted as a Bayesian approximation of Gaussian process [180]. Therefore, we introduce a dropout layer before every weight layer of our imitator network. To estimate predictive confidence, we collect the results of stochastic forward passes through the imitator network:

$$c(s_t, g) = \mathbb{E}_{d_j \sim D}[f^{d_j}(s_t, g|\theta) - p]^2 \tag{8.6}$$

where $f^{d_j}(s_t, g|\theta)$ represents the model with dropout mask $d_j$, $D$ is a set of dropout masks, and $p$ is the predictive posterior mean, $p = \mathbb{E}_{d_j \sim D} f^{d_j}(s_t, g|\theta)$. Since the forward passes can be done concurrently, the method results in a running time identical to that of standard dropout. We can expect the variance of unknown and far-away tuples to be larger than known tuples. Please note that one advantage of using dropout is that it allows the imitator network to "smooth out" much of the noise in the data, making the imitator network more robust to noise in the demonstration data. Besides, this technique is particularly effective in the low-data regime to improve generalization of demonstrated behaviors. We compare in Section 8.4.2 the impact of each strategy on our method.

### 8.3.4 Prioritized Goal Sampling

In the future HER sampling strategy, the new goals are randomly selected along the future state-action trajectory. However, an RL agent can learn more effectively from some goals than from others. Typically, some goals may be useful to the agent, but might become less when the agent competence increases. Prioritized goal sampling (PGS) assigns high sampling priority to key goals - in places where the expert and the agent strongly disagree. As a criterion to quantify this disagreement, we measure the divergence in the action recommendation between the imitator-network and the policy, which indicates how "surprising" or "hard-to-learn" the goal is. Given a state $s_t$ seen along an episode of $T$ states and $g$ the current goal, we define the probability of sampling $s_t$ as a new goal:

$$p(s_t, g) = \frac{||\pi(s_t, g|\theta_\pi) - f(s_t, g|\theta)||_2^2}{\sum_{j=t}^{T} ||\pi(s_j, g|\theta_\pi) - f(s_j, g|\theta)||_2^2} \tag{8.7}$$

where $||\pi(s_t, g|\theta_\pi) - f(s_t, g|\theta)||_2^2$ is the deviation between the policy and expert. As a result, PGS capitalizes on large disagreement to encourage sampling of goals that potentially lead to large learning progress.

## 8.4 Experiments

In this section, we first describe implementation details and the tasks to be completed by the agent. Then, we conduct experiments in multiple tasks from the Mujoco suite [74, 161]. Finally, we answer the following questions:

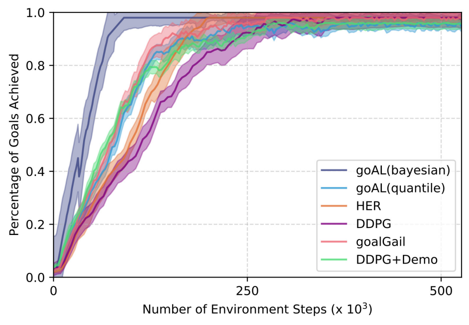
- What is the impact of the relabeling threshold on the imitator network training?
- How important is the weight of the imitation loss?
- Is goAL robust to noisy demonstrations?
- Can goAL generalize to unseen goals?
- Does our method increase the state coverage of demonstrations?
- Is prioritized goal sampling an efficient way to select goals?
- What is the impact of the query budget on the performance?

## 8.4.1 Implementation Details and Tasks

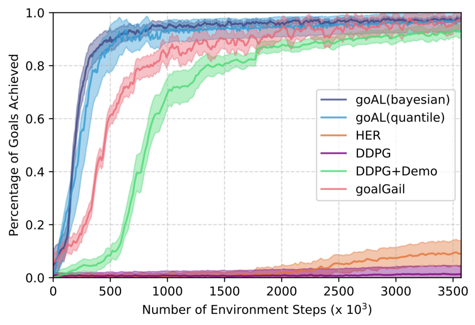
Experiments are conducted on eight robotic tasks implemented in MujoCo. In the first set of experiments, we consider four manipulation tasks (Fetch tasks) where the agent controls a 7-DoF Sawyer arm: (1) Fetch Reach, (2) Fetch Push, (3) Fetch Pick & Place, and (4) Fetch Slide. The end-effector (EE) is constrained to a 2-dimensional rectangle. In the second set of experiments (ShadowHand tasks), we evaluate our framework on significantly more challenging tasks with very sparse rewards and larger action spaces. The agent is trained to manipulate physical objects via a human-like robot hand: (1) Hand Manipulate Block, (2) Hand Manipulate Egg, (3) Hand Manipulate Pen, and (4) Hand Reach. The observations are given in the form of continuous values and the action-space is also continuous. The performance metric we use is the percentage of goals that the agent is able to reach. An episode is considered successful if the distance between the agent and the goal at the end of the episode is less than a threshold defined by the task.

As our policy learning method, we rely on DDPG with HER. We refer to our algorithm as active goal-driven learning (goAL). The critic and imitator networks consist in 4 fully-connected layers with 256 hidden units. ReLU is used as the activation function expect for the last layer that used tanh, and the output value is scaled to the range of each action dimension. Their parameters are optimized given as input pairs of state-goal. Training is carried out with a fixed learning rate of $10^{-3}$ using the Adam optimizer [72], with a batch size of 256.

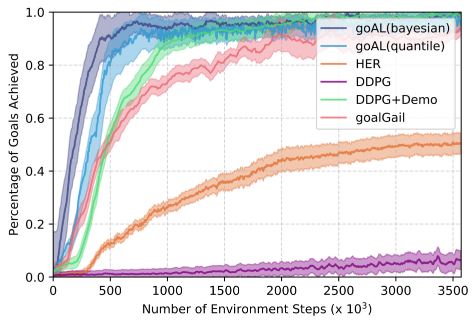
In order to select the hyperparameters used for Fetch and ShadowHand tasks, we ran a grid search with the ranges shown in Section 8.4.4 and we used 10 seeds on the eight tasks. For Fetch tasks, we use a query budget of 20 and we set $t_{qry} = 0.32$ (Bayesian-confidence) and $t_{qry} = 0.43$ (quantile-confidence). For ShadowHand tasks, we use a query budget of 50, $t_{qry} = 0.58$ (Bayesian-confidence), and $t_{qry} = 0.27$ (quantile-confidence). In all our experiments, Bayesian-confidence is estimated based on 500 dropout masks with $p = 0.1$. The weights of loss components were $\lambda_1 = 1$ and $\lambda_2 = 0.003$ unless stated otherwise and we anneal the imitation loss weight $\lambda_2$ by 0.98 per 500 rollouts collected. As relabeling constant $N$, we set the constant equal to half of the maximum number of time-steps per episode. Since we account for the possibility that the learned policy outperforms expert demonstrations, we employ the depicted Q-filter strategy with $\eta = 0.015$.



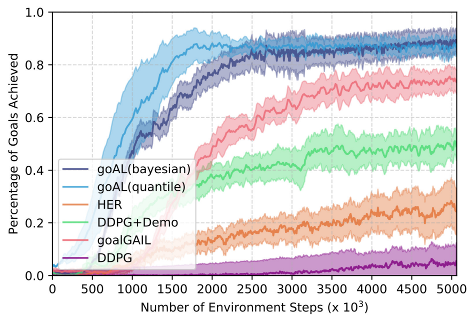
(a) Fetch Reach



(b) Fetch Push



(c) Fetch Pick & Place

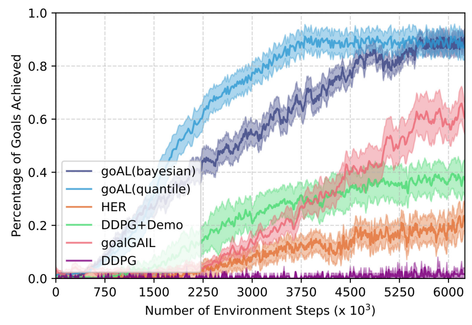


(d) Fetch Slide

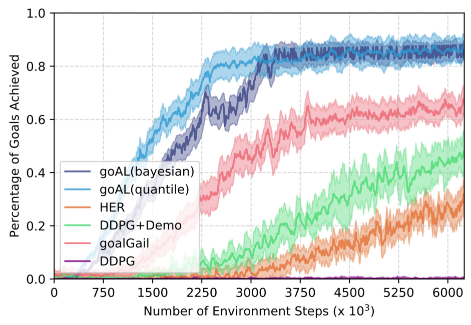
Figure 8.3: Learning curves averaged over 10 runs (±std) for different models: goAL(bayesian), goAL(quantile), DDPG, HER, DDPG+Demo, and goalGail. The models are trained on robotic tasks from the Fetch environment.

### 8.4.2 Fetch Robotic Tasks

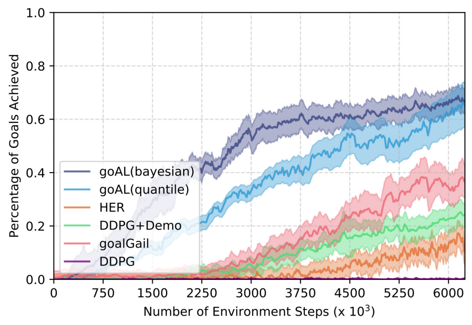
We first perform experiments on four different Fetch tasks from the robotic domain built on top of Mujoco: Fetch Reach, Fetch Push, Fetch Pick and Place, and Fetch Slide. We first evaluate DDPG with Hindsight Experience Replay (HER) [41] with and without active-goal driven learning (goAL). Moreover, we compare our method against several baselines including DDPG [4], DQfD [114], goalGail [169], and DDPG+Demo [165]. Please note that we replace DQN by DDPG as learning algorithm in DQfD. We use 20 demonstrations to guide goalGAIL and 128 for the other methods. To generate these demonstrations, we randomly sample goals and request the expert to reach them. We show learning curves in Figure 8.3. Our method can learn comparable or superior policies using a significantly smaller number of demonstrations. For instance, on Pick and Place, in average only 9 queries were made by goAL. As expected, it ends up reaching similar final performance, however, our method has a faster convergence rate. As can be observed, despite the fast early learning of baselines, their convergence speed becomes similar to that of the original HER after extracting all the knowledge from the demonstrations. On the other hand, incrementally querying new demonstrations enables us to overcome this problem, while keeping the number of demonstrations very low. Quantile-confidence can be useful to obtain a more comprehensive analysis of the agent's confidence, and is more robust to extreme outliers in the goal-driven demonstrations (e.g. providing the wrong action). Therefore, we believe that this approach should be used when the feedback are provided by a non-expert. On the other hand, Bayesian-confidence is particularly effective to "smooth out" much of the noise in the demonstrations or randomness in the environment, and enables a better generalization of goal-driven demonstrations in the low-data



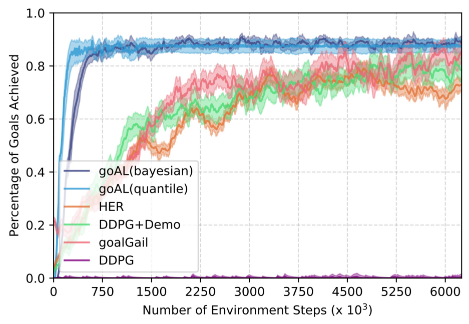
(a) Hand Manipulate Block



(b) Hand Manipulate Egg



(c) Hand Manipulate Pen



(d) Hand Reach

Figure 8.4: Learning curves (mean±std) on ShadowHand tasks averaged over 10 runs for different models: goAL(bayesian), goAL(quantile), DDPG, HER, DDPG+Demo, and goalGail.

regime. Overall, this experiment highlights that goAL drastically reduces the training time in complex environments and the number of necessary interactions with humans.

### 8.4.3 ShadowHand Robotic Tasks

In addition to the first robotic tasks, we evaluate our methodology in a more challenging set of environments (ShadowHand): Hand Manipulate Block, Hand Manipulate Egg, Hand Manipulate Pen, and Hand Reach. We provide an expert trajectory dataset of 100 demonstrations to goalGAIL and 400 to the other methods. Figure 8.4 plots the learning curve of all the models. We can observe that our strategy helps to greatly improve convergence speed. Unlike our algorithm, prior methods passively access the demonstration data, so we actively provide help to our agent when it struggles. On these tasks, we found that goalGAIL does not receive enough supervision to achieve optimal policies. Results highlight that the gap between our approach and the others is increasing with the degree of sparsity. Overall, these results show that our method is capable of adapting the set of demonstration trajectories to match the learner's goal and hence escape the known "distribution mismatch" issue of prior work. Remarkably, the final performance of goAL is not capped and can even exceed expert-level performance. To the best of our knowledge, this is the first approach operating in the low demonstration regime (less than 50 demonstrations) that achieves a near-optimal score on the four ShadowHand tasks.

Table 8.1: Learning locomotion in Mujoco using different positive thresholds $N$ when training the imitator network (line 1-8) or using different imitation loss weights (line 9-12). Results are averager over 10 random seeds (±std). No seed tuning is performed.

| | Percentage of goals achieved | | | |
|---|---|---|---|---|
| Method | Fetch Reach | Fetch Push | Fetch Pick & Place | Fetch Slide |
| goAL(bayesian) / N=0.25 | 0.91±0.03 | 0.95±0.05 | 0.90±0.03 | 0.87±0.04 |
| goAL(quantile) / N=0.25 | 0.90±0.04 | 0.92±0.02 | 0.92±0.04 | 0.90±0.02 |
| goAL(bayesian) / N=0.50 | 0.97±0.02 | 0.96±0.04 | 0.96±0.04 | 0.87±0.06 |
| goAL(quantile) / N=0.50 | 0.96±0.02 | 0.94±0.03 | 0.95±0.03 | 0.90±0.05 |
| goAL(bayesian) / N=0.75 | 0.96±0.03 | 0.97±0.05 | 0.93±0.05 | 0.88±0.04 |
| goAL(quantile) / N=0.75 | 0.95±0.04 | 0.94±0.03 | 0.93±0.04 | 0.86±0.04 |
| goAL(bayesian) / N=1.0 | 0.93±0.04 | 0.94±0.05 | 0.88±0.05 | 0.79±0.08 |
| goAL(quantile) / N=1.0 | 0.92±0.05 | 0.92±0.06 | 0.91±0.07 | 0.88±0.04 |
| goAL(bayesian) / $\lambda_2 = 0.001$ | 0.95±0.03 | 0.93±0.05 | 0.94±0.07 | 0.88±0.08 |
| goAL(quantile) / $\lambda_2 = 0.001$ | 0.96±0.04 | 0.92±0.04 | 0.91±0.05 | 0.89±0.05 |
| goAL(bayesian) / $\lambda_2 = 0.003$ | 0.97±0.02 | 0.96±0.04 | 0.96±0.04 | 0.87±0.06 |
| goAL(quantile) / $\lambda_2 = 0.003$ | 0.96±0.02 | 0.94±0.03 | 0.95±0.03 | 0.90±0.05 |
| goAL(bayesian) / $\lambda_2 = 0.006$ | 0.97±0.02 | 0.95±0.06 | 0.92±0.05 | 0.90±0.04 |
| goAL(quantile) / $\lambda_2 = 0.006$ | 0.94±0.04 | 0.93±0.03 | 0.93±0.04 | 0.86±0.06 |

### 8.4.4 Ablation Analysis

We also present an ablation study to investigate: 1) the importance of the relabeling threshold, 2) the impact of the weight of the imitation loss, 3) the robustness to imperfect demonstrations, 4) the generalization to unseen goals, 5) the state coverage of queries, 6) the impact of prioritized goal sampling, and 7) the size of query budget.

#### Relabeling Threshold in Imitator Network Training

Relabeling the goal-driven demonstrations requires a threshold $N$ to separate "optimal" from "sub-optimal" transitions. Thus, the trained policy implicitly depends on this threshold. Precisely, adding potentially sub-optimal transitions may hurt the performance of the agent. We conduct a study where the threshold $N$ is varied from 0.25 to 1.0. A threshold of 0.25 means that N is equal to one-fourth of the maximum of time-steps per episode. Table 8.1 (line 1-8) shows that relabeling all the goals hurts the performance. This can happen for mainly two reasons: 1) conflicts arising from sub-optimal samples recorded in the demonstration buffer, and 2) since the agent's confidence for imaginary sub-optimal samples is high, the agent struggles to identify regions where feedback is the most needed. A reasonable choice of N is between 0.5 or 0.75. Hence, the depicted relabeling strategy helps the performance of the algorithm.

Table 8.2: Percentage of goals achieved from imperfect demonstrations, where $\varrho$ is the probability of providing a sub-optimal action. The results are averaged across 10 random seeds (±std). We set the number of queries to 20.

| | $\varrho$=0.05 | | | $\varrho$=0.1 | | | $\varrho$=0.2 | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | Fetch Push | Fetch Pick & Place | Fetch Slide | Fetch Push | Fetch Pick & Place | Fetch Slide | Fetch Push | Fetch Pick & Place | Fetch Slide |
| goalGail | 0.91±0.04 | 0.84±0.06 | 0.72±0.08 | 0.88±0.05 | 0.78±0.11 | 0.70±0.09 | 0.74±0.08 | 0.62±0.14 | 0.62±0.13 |
| DDPG+Demo | 0.89±0.05 | 0.81±0.07 | 0.48±0.08 | 0.72±0.08 | 0.62±0.14 | 0.30±0.12 | 0.51±0.15 | 0.47±0.18 | 0.21±0.20 |
| goAL(bayesian) | **0.94±0.06** | 0.83±0.06 | 0.90±0.10 | **0.91±0.08** | **0.80±0.10** | **0.87±0.12** | **0.85±0.06** | **0.77±0.08** | **0.84±0.14** |
| goAL(quantile) | 0.92±0.05 | **0.94±0.13** | **0.91±0.07** | 0.88±0.08 | 0.72±0.14 | 0.69±0.07 | 0.77±0.13 | 0.66±0.12 | 0.62±0.10 |

### Weight of Imitation Loss

Combining the policy loss and imitation loss involves a hyperparameter $\lambda_2$, which weights the importance of the imitation loss. One legitimate question is to study the impact of this hyperparameter on the performance of the agent. Ideally, the policy performance should not be too sensitive to this hyperparameter. We perform a study for various values of $\lambda_2$ in $0.001, 0.003, 0.006$. Table 8.1 (line 9-12) shows that the goAL performance is robust to the choice of this hyperparameter.

### Robustness to Imperfect Demonstrations

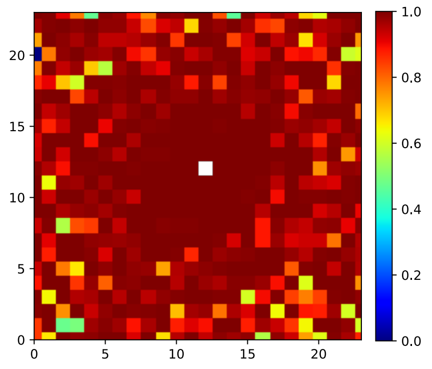
In the above experiments, we assume perfect demonstrations. However, the expert might select not the best action or even lack knowledge about a goal. We study how our agents perform when imperfect demonstrations are generated by the demonstrators. In order to generate imperfect demonstrations, we add normal noise $\mathcal{N}(0, \sigma^2)$ to the teacher actions with a probability $\varrho \in \{0.05, 0.1, 0.2\}$ and $\sigma = 0.03$. We report in Table 8.2 the performance of our framework and several baselines. We observe that goAL can still achieve acceptable performance. For instance, the success rate of the proposed method remains larger than 0.80 on the three tasks ($\varrho = 0.05$). Even though goAL(bayesian) performs slightly worse in the imperfect setting, it still improves performance as compared to the prior methods. A reason is that dropout allows the imitator to "smooth out" much of the noise in the data, making goAL(bayesian) robust to noisy demonstrations. Moreover, annealing the imitation loss weight and filtering the sub-optimal demonstrations allow us to escape from poor local optima, improving significantly beyond the (imperfect) expert demonstrations. The results demonstrate that our method is reasonably robust to noise in the demonstrations, and hence non-expert can provide a feedback signal to the agent.

### Generalization to unseen goals

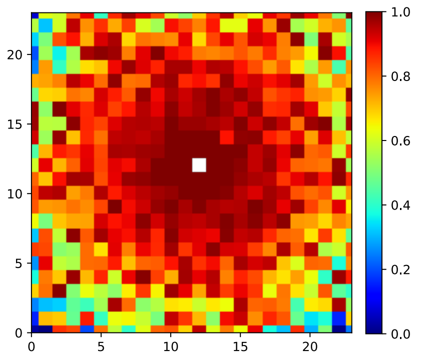
In the previous section, we showed that our method learns to achieve a wide range of goals. However, it remains unclear whether the agent has achieved this by "generalizing demonstrated trajectories". To investigate this question, we train our agent on a set of goals and evaluate its performance on a different set of goals (without additional queries). From Table 8.3, we see that the agent can generalize to unseen goals, with a slight loss in the performance. As the agent has already learned about parts of the environment, it can leverage known similar goals to face an unseen situation. We can further observe that

Table 8.3: Evaluation of the agent trained on several Mujoco tasks ("fine-tuned"). We then evaluate the agent's performance on a different set of goals without querying new demonstrations ("run as"). We report the results averaged over 10 seeds (±std).

| | Percentage of goals achieved | | | |
|---|---|---|---|---|
| Method | Fetch Reach (0.5M) | Fetch Push (3M) | Fetch Pick & Place (3M) | Fetch Slide (3M) |
| Run as (bayesian) | 0.95±0.03 | 0.84±0.08 | 0.93±0.11 | 0.85±0.10 |
| Run as (quantile) | 0.93±0.04 | 0.79±0.06 | 0.85±0.08 | 0.72±0.09 |
| Fine-tuned(bayesian) | 0.97±0.02 | 0.96±0.04 | 0.96±0.04 | 0.87±0.06 |
| Fine-tuned(quantile) | 0.96±0.02 | 0.94±0.03 | 0.95±0.03 | 0.90±0.05 |



(a) With confidence



(b) Without confidence

Figure 8.5: State visitation heatmaps of the demonstrations queried by the goAL agent with Bayesian-confidence (left) and without confidence prediction (right). The agent starts in the middle of the board (white square).

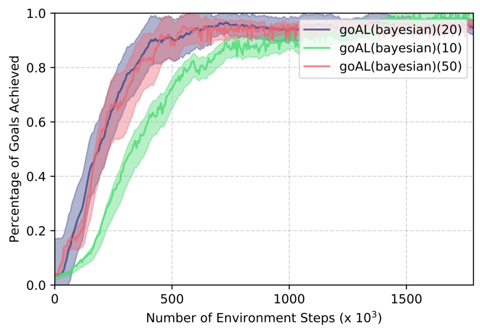
goAL(bayesian) tends to generalize to unseen situations better than goAL(quantile). We hypothesize that using dropout in the imitator network prevents the network to overfit the provided goal-driven demonstrations. Moreover, the results highlight that Bayesian-confidence is slightly more accurate to estimate the agent's confidence than quantile-confidence, improving the value information of the queries. Experimental results suggest that this form of human guidance allows the learner to generalize domain knowledge to unseen situations.

## State Coverage of Queries

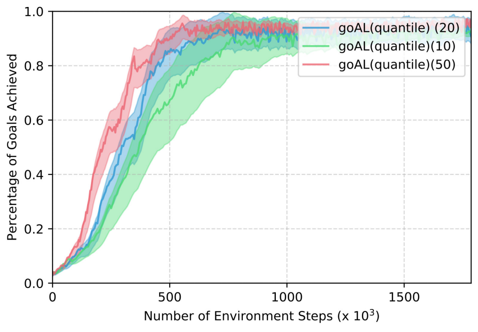
In this experiment, we show that our method is efficient at querying demonstrations and can keep the level of redundancy at a minimum. To do so, we show state visitation heatmaps of trajectories queried by our method over 100 runs on a simple 2D navigation environment. The agent starts in the center of the box, and can take actions to directly move its position. In this task, the agent needs to navigate itself to a target position ($x$,$y$) that is randomly generated by the environment. Figure 8.5 illustrates that trajectories requested using our method cover most of the states. One reason is that similar goals are not requested for demonstrations since the confidence is large enough. Thus, the agent explores further and make queries in places where it struggles. On the other hand, as the agent (without Bayesian-confidence) fails to reach the goals being pursued, it quickly

Table 8.4: Learning locomotion in Mujoco, with and without prioritized goal sampling (PGS). Results are averager over 10 random seeds (±std). No seed tuning is performed.

| | Percentage of goals achieved (convergence speed $\times 10^3$) | | | |
|---|---|---|---|---|
| Method | Fetch Reach | Fetch Push | Fetch Pick & Place | Fetch Slide |
| goAL(bayesian) | 0.97 (122) | 0.96 (982) | 0.96 (812) | 0.87 (1850) |
| goAL(quantile) | 0.96 (241) | 0.94 (1023) | 0.95 (1073) | 0.90 (2609) |
| goAL(bayesian) without PGS | 0.96 (157) | 0.97 (1256) | 0.95 (1229) | 0.85(2044) |
| goAL(quantile) without PGS | 0.96 (270) | 0.95 (883) | 0.94 (1627) | 0.91 (2953) |



(a) goAL(bayesian)



(b) goAL(quantile)

Figure 8.6: Learning curves averaged over 10 runs (±std) on the Fetch Pick & Place task, with different query budgets.

exhausts its query budget by making redundant queries. As a result, we can expect imitation learning to be efficient only near the center of the box. This issue becomes even more noticeable as the size or the complexity of the environment is increased. In general we discovered that the depicted algorithm improves state coverage and eliminates the need for unnecessary demonstrations of already acquired behaviors.

### Using Prioritized Goal Sampling

One legitimate question is to study the impact of the prioritized goal sampling on the performance of the algorithm. We conduct a study with and without prioritized goal sampling (PGS). As shown in Table 8.4, PGS produces faster learning in all eight environments. For example, on Fetch Push, PGS reduces the number of interactions to reach converge by $\approx 27\%$ for goAL(bayesian) and $\approx 15\%$ for goAL(quantile). Furthermore, the results show that indeed PGS does not deteriorate performance and in some cases our agent can reach higher final performance than running pure goal sampling. It confirms that the most important goals for replays are the ones where the disagreement in the prediction is maximized.

#### Query Budget

We finally report evaluations showing the effect of increased query budget. Figure 8.6 demonstrates that agents trained with a larger query budget obtain higher mean returns after similar numbers of updates. However, despite a small query budget, our method can still learn near optimal policies. We can draw the observation that as the query budget increases, the learning effect on the agent gradually improves. However, for the results with 20 and 50 queries, we can see that even if the number of queries significantly differs, the difference in learning effect can be negligible. This can happen when queried demonstrations cover a broad state space and therefore the agent does not need to make additional queries. As a result, our method leverages a small amount of demonstrations that cover task-relevant regions of the state space and outperforms the baselines by a large margin (see Section 8.4.2).

## 8.5 Discussion

We have constructed a mechanism to utilize a novel form of human guidance, goal-driven demonstrations, along with goal-conditioned reinforcement learning. Goal-driven demonstrations do not intend to cover all the state-space or reach the final goal, but guide the agent to fulfill particular intermediate goals when it struggles. In order to greatly reduce the number of required demonstrations, we proposed to actively query the human demonstrator in states where the agent struggles and its confidence in the action prediction is low. In this work, we introduced and studied the effect of two strategies to measure the model's confidence. This novel form of human guidance is less expensive and more intuitive than policy demonstrations, and ensures that the provided knowledge match the agent's needs, hence escaping the known "distribution mismatch" issues of prior work. Even very small amounts of feedback (less than 50 queries) let us outperform prior imitation-based approaches on Fetch and ShadowHand tasks. We also study the effect of doing expert relabeling as a type of data augmentation on the provided goal-driven demonstrations. Remarkably, the depicted algorithmic framework can match the basic demonstration-level performance and even exceed expert-level performance. Furthermore, goAL learns to reach a wide range of configurations from the same set of demonstrated trajectories. These results suggest that goAL improves the capability for common sense reasoning in agents, which greatly enhances learning efficiency and could help to expand the possible applications of RL.

The proposed approach opens several avenues of research. An exciting future direction is to integrate curiosity into the proposed framework. In regions where the agent learns in the absence of external guidance, curiosity could help to improve sample-efficiency and incentivize the agent to outperform the performance of the expert. Another possible improvement is combining the idea of active feedback/queries with other forms of guidance, less reliant on low-level human knowledge to enable non-expert feedback.

# Part III

# Leveraging Human Guidance and Curiosity for Sample-Efficient Learning

# Chapter 9

# Hierarchical Learning from Human Preferences and Curiosity

So far, we have addressed the sample efficiency issue via internal or external supervision. Nevertheless, contrary to humans that can simultaneously aggregate multiple intermediate learning signals, we have not considered the scenario in the context of reinforcement learning where multiple types of supervision are available. We believe that achieving human-like sample-efficient learning is closely related to this capability of the human brain. All these considerations lead to one fundamental question: how to integrate different forms of supervision into reinforcement learning?

In this chapter, we propose to hierarchically decompose the task into a set of sub-tasks and incorporate different combinations of supervision at different levels. We explore (non-expert) human guidance in the form of high-level preferences between sub-goals, leading to drastic reductions in both human effort and cost of exploration. Here, the human trainer is not necessarily an expert at performing the task since it only requires the human to judge outcomes. We also make use of curiosity to effectively drive the learning of low-level subpolicies. Finally, we demonstrate how curiosity relates to sub-goal discovery.

## 9.1 Introduction

Reinforcement learning algorithms often require a large number of interactions to reach decent performance, which can be intractable in real-world settings. A strategy to escape these pitfalls is hierarchical reinforcement learning (HRL) [42] that decomposes the overall task into easier short-term sub-tasks. However, it may be slow to learn subpolicies and it remains challenging to order subpolicies in the absence of dense task rewards. Besides, HRL usually requires to manually define a set of sub-goals.

Another successful class of methods for dealing with such problems is imitation learning (IL) [23] in which the agent learns by observing and possibly querying an expert [181]. Nevertheless, these approaches are not directly applicable to behaviors that are difficult for humans to demonstrate such as robot control or temporally-extended tasks, and assume that the human demonstrator has some familiarity with the task. A solution is to leverage other types of human guidance such as "preferences" [170] or "weak feedback" [28]. These types of human guidance were shown to be easier to demonstrate for humans and reduce the amount of human involvement. For instance, preference-based learning has been applied to game-playing as well as robot control [170, 159]. However, we argue that providing low-level preferences between pairs of trajectory segments may be an inefficient way of soliciting humans, and could be expensive and/or subject to error. A different paradigm provides an intrinsic exploration bonus (i.e. curiosity) to the agent. For example, count-based exploration [18] keeps visit counts for states and favors the exploration of states rarely visited. Curiosity can also be measured as the error in predicting the consequences of the agent's actions on the environment [14], a prediction task where the problem is a deterministic function of its inputs [20, 78], or explicitly promote in-depth exploration [182]. Prior work have demonstrated that curiosity is a sufficient exploration signal to improve control and execution in agents (e.g. learning to reach a sub-goal in HRL), however, learning a task from scratch can still require a prohibitively time-consuming amount of exploration of the state-action space in order to find a good policy. Namely, reasoning and planning based on common sense priors [16] is beyond the reach of agents trained with no prior assumptions about the domain. In order to expand the availability of DRL, it is necessary to combine the strengths of both of these techniques. In other words, human preferences is a good source of guidance for high-level decision-making such as planning/reasoning, while curiosity is a good source of endogenous motivation for exploring the consequences of low-level actions on the environment (control and execution), like how to pass an obstacle or whether to interact with a particular object.

In this chapter, we propose a hierarchical reinforcement learning algorithm. At a high-level, a meta-controller policy is trained to select sub-goals given (non-expert) human high-level preferences between pairs of sub-goals as a feedback signal, leading to dramatic reductions in both human workload and degree of familiarity necessary to provide feedback. Besides, human preferences provide prior assumptions about the domain to the agent, avoiding learning from scratch and enabling common sense reasoning. We decide how to query preferences based on an approximation to the confidence in the action selection. At a low-level, we introduce curiosity to drive the learning of subpolicies (i.e. low-level control and execution). Such an approach is motivated by learning in animals, they utilize all possible intermediate learning signals (e.g. curiosity, preferences) to solve challenging tasks. We further contribute a strategy to automatically discover sub-goals that relies on the agent's curiosity, alleviating the need for an expert to define sub-goals. As an intuition, we found that manually created sub-goals are generally associated with large spikes in the curiosity, which indicate meaningful events. This strategy promotes active cooperation between levels of the policy by communicating potential novel meaningful sub-goals throughout the training process. Our experiments take place in three domains: 2D navigation tasks in Minigrid, robotics tasks in the physics simulator MuJoCo, and Atari games in the Arcade Learning Environment (ALE). We show that a small amount of feedback from a non-expert human suffices to rapidly learn both standard RL tasks and novel hard-to-specify behaviors such as playing Montezuma's Revenge or dexterous manipulation with robotic arms, notorious for their extremely sparse rewards and complexity.

The key contributions of this article are as follows:

- We present a two-level hierarchical algorithm that introduces human high-level preferences between pairs of sub-goals at the high-level to drastically reduce human workload, while being more intuitive for humans and enabling non-expert feedback.

- We introduce a technique to request very few preferences, in states where the agent's confidence is low.

- We propose to use curiosity at the low-level to drive exploration and reduce the amount of interactions to learn subpolicies (i.e. low-level policies). Next, we derive a method to automatically discover sub-goals through the idea of *curiosity-driven sub-goal discovery.*

## 9.2 Related Work

**Imitation Learning.** DQfD [114] pre-trains a Q-learning agent on the expert demonstration data. This idea was extended to handle continuous action spaces such as in robotic tasks [162], as well as to actor-critic architectures [163]. A recent follow-up [165] introduces an expert loss in DDPG [4] and proposes to filter suboptimal demonstrations based on the Q-values. Another solution is to represent a policy as a set of Gaussian mixture models [166]. In a different spirit, AlphaGo [2] trains a policy network to classify positions according to expert moves. A way of dealing with sparse rewards consists in introducing a curious replay mechanism and demonstrations [167]. In recent years, an emerging strategy combines generative adversarial networks and reinforcement learning (GAIL) [24]. However, GAIL was shown to suffer from the *drift prediction error* and instability during training. Another method [169] constructs a goal-conditioned policy to visit similar states as the expert. A different form of imitation learning is inverse reinforcement learning (IRL) [25] that uses demonstrated trajectories to extract a reward function. IRL has been applied to several domains including navigation [142, 27], and autonomous flight [22]. However, in many cases it is impractical to demonstrate long-term tasks and IRL algorithms assume that the observed behavior is optimal. Overall, it is not clear how to scale imitation learning to much more complex behaviors that are difficult to collect. Another limitation is how to learn when the agent's goal deviates from the demonstrated trajectories. This work differs by leveraging high-level guidance - human preferences between pairs of sub-goals, alleviating the need for collecting low-level trajectories.

**Interactive Human Feedback.** Most methods that focus on learning from interactive human feedback [170, 171, 172, 159, 173] query the human to drive learning. Especially, it is possible to query trajectory preferences [170], which can be combined with fixed demonstrations [159]. In contrast, the depicted method introduces the idea of high-level preferences, and combines human feedback with curiosity that yields better-than-expert performance by guiding the agent in the quest of knowledge beyond the expert's knowledge. Another example, TAMER [174], trains the policy from feedback in high-dimensional state spaces. The learner may also receive feedback in the form of sequences of actions planned by a teacher [175]. Some authors [177] consider multiple demonstrators performing different tasks and the agent must actively select which one to request for advice. Another solution [178] is to block unsafe actions by training a module from expert feedback. However, it requires the expert to identify all unsafe situations by watching an agent play. To deal with the problem of query selection, it is possible to select sufficiently different unqueried data [179]. In this work, we only request high-level feedback to the supervisor in states where the agent is unsure and struggles. Moreover,

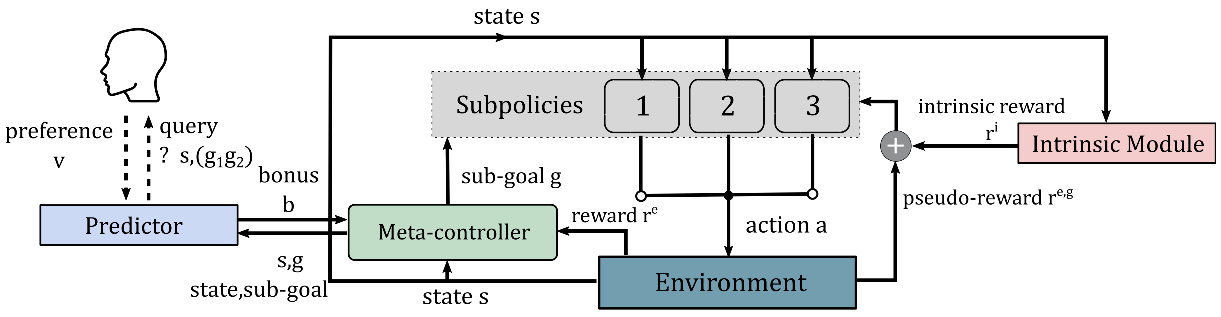


Figure 9.1: Hierarchical learning from human preferences and curiosity. The predictor network is trained based on high-level preferences provided by a demonstrator. The predictor network provides a bonus $\boldsymbol{b}$ to the meta-controller for agreeing with human preferences. In addition to the pseudo-reward $\boldsymbol{r^{e,g}}$, the subpolicies receive an intrinsic reward $\boldsymbol{r^i}$.

it does not necessarily require the human trainer to be an expert at performing the task since the proposed form of guidance only requires the human to judge outcomes. We also use the structure of the task and introduce curiosity to drive the agent's exploration of sub-tasks for which human feedback are not necessary, further reducing human effort.

**Sample-Efficient Hierarchical Reinforcement Learning.** Many tasks are hierarchically structured, which entails that they can be decomposed and gradually solved. A closely related hierarchical RL work to ours is the approach named hg-DAgger [183] in which the agent can request high-level and low-level demonstrations, and a signal to indicate if the high-level sub-goal was chosen correctly. At each level, the learning task becomes a typical imitation learning problem. In this work, we primarily focus on preferences to enable non-expert feedback and expand the possible applications of HRL, including in tasks with behaviors difficult to demonstrate. Moreover, we overcome the need for expert-engineered sub-goals by introducing a method to automatically discover sub-goals during exploration. Humans can also provide policy sketches that are high-level sub-task labels [184]. A forward model has also been used as a measure of novelty to improve sample efficiency in actor-critic HRL [185]. Integrating temporal abstraction and intrinsic motivation has been applied to game-playing [186]. On the other hand, our method is driven by a Random Network Distillation [20] error that gradually downmodulates states that become progressively familiar across the agent's learning, escaping from the known "Noisy-TV" issue inherent in curiosity.

## 9.3 Leveraging Preferences and Curiosity

The challenges of injecting expert feedback into DRL are two folds. First, in many cases it is impractical to use human policy as guidance because some of these tasks are too challenging for even humans to perform well. Second, providing low-level feedback can be time-consuming and significantly increase human workload.

The framework of hierarchical learning from human preferences (HhP) provides us a mechanism to mitigate these problems (Figure 9.1). Our approach introduces (non-expert) human high-level feedback into hierarchical reinforcement learning via preferences over sub-goals. This provides a significant speedup in learning while requiring less human effort. Although a number of algorithms could in principle be used to learn the low-level subpolicies, they often require to manually engineer sub-goals and rely on large amounts of interactions. In contrast, our formulation introduces curiosity to enable self-discovery of sub-goals and speed up the learning of subpolicies.

In the following section, we first describe high-level components of our method and then low-level components.

### 9.3.1 High-Level Preferences

In this section, we describe high-level components of our algorithm. When learning from demonstrated trajectories, the policy is trained to clone a human (expert) demonstrator on the task. Nevertheless, to provide meaningful demonstrations, the demonstrator has to have some knowledge and familiarity with the current task. As a result, learning from direct demonstrations of trajectories or high-level actions (i.e. sub-goals) is significantly more costly than requesting human (non-expert) preferences, which only uses the ability to judge outcomes. Moreover, rather than relying on trajectory preferences (i.e. low-level preferences) we propose to query high-level preferences, decreasing the amount of feedback required by several orders of magnitude.

To this end, we assume a set of sub-goals $\mathcal{G}$ and access to a supervisor that can provide preferences over sub-goals. Our method maintains a meta-controller (also called high-level policy) $\tau : \mathcal{S} \rightarrow \mathcal{G}$ and a *predictor* network that estimates a reward function from the annotator's preferences $\hat{p} : \mathcal{S} \times \mathcal{G} \rightarrow \mathbb{R}$, each parametrized by deep neural networks. Our predictor network takes a state and a sub-goal as input and outputs an estimate of the corresponding reward. In our settings, preferences are collected while the agent is training, during the experiment. The proposed framework can be divided into three processes: (1) training the predictor network to fit the high-level preferences collected from human so far, (2) optimizing the meta-controller, and (3) selecting new queries.

#### Training the Predictor Network

Our model maintains a predictor network $\hat{p}$ that mimics human high-level preferences. It takes as input the current state $s_t$ and the sub-goal $g$ recommended by the meta-controller, and outputs a reward that "criticizes" the meta-controller's recommendation, $b_t \in \mathbb{R}$.

In order to train the predictor model, the human overseer is given a visualization of a state $s$ and a pair of two sub-goals $(g_1, g_2)$, in the form of images. The human then indicates which sub-goal is favoured, that the two sub-goals are equally good, or that the overseer is unable to compare the two sub-goals. We record tuples $(s, g_1, g_2, v)$ in a database $\mathcal{R}$, where $s$ is a state, $g_1$ and $g_2$ are two distinct sub-goals, and $v$ is the score given by the demonstrator. Write $g_1 \succ g_2$ to indicate that the human preferred sub-goal $g_1$ to sub-goal $g_2$, and $g_1 \not\succ g_2$ to indicate that the two sub-goals are equally good. In our implementation we use labels for encoding preferences (i.e. score $v$): $[1, 0, 0]$ denotes $g_1 \succ g_2$, $[0, 1, 0]$ denotes $g_2 \succ g_1$, and $[0, 0, 1]$ denotes $g_1 \not\succ g_2$.

Assuming the data-set $\mathcal{R}$ of tuples, $\mathcal{R} = \{(s, g_1, g_2, v), ...\}$, we train the predictor network $\hat{p}$ to minimize the cross-entropy loss, $\mathcal{L}$, between these predictions and the actual judgment labels. Since we discard tuples that are incomparable, the loss function can be written as follows:

$$\mathcal{L}(\hat{p}, \theta^*, \mathcal{R}) = -\frac{1}{|\mathcal{R}|} \sum_{(s,g_1,g_2,v)\in\mathcal{R}} \log(P[g_1 \succ g_2])v_0 + \log(P[g_2 \succ g_1])v_1 \tag{9.1}$$

where $\theta^*$ is the set of parameters of the predictor network, $v_i$ corresponds to the $i$'th element of one-hot encoded label of the sample $(s, g_1, g_2)$, $(g_1, g_2)$ is a pair of sub-goals, and $P$ is the probability of preferring a sub-goal. We calculate this probability via:

$$P[g_1 \succ g_2] = \frac{e^{\hat{p}(s,g_1|\theta^*)}}{e^{\hat{p}(s,g_1|\theta^*)} + e^{\hat{p}(s,g_2|\theta^*)}} \tag{9.2}$$

Therefore, the overall loss can be re-written as:

$$\mathcal{L}(\hat{p}, \theta^*, \mathcal{R}) = -\frac{1}{|\mathcal{R}|} \sum_{(s,g_1,g_2,v)\in\mathcal{R}} \log\left[\frac{e^{\hat{p}(s,g_1|\theta^*)}}{(e^{\hat{p}(s,g_1|\theta^*)} + e^{\hat{p}(s,g_2|\theta^*)})}\right]v_0 + \log\left[\frac{e^{\hat{p}(s,g_2|\theta^*)}}{(e^{\hat{p}(s,g_2|\theta^*)} + e^{\hat{p}(s,g_1|\theta^*)})}\right]v_1 \tag{9.3}$$

Please note that, $\hat{p}_i(s, g|\theta^*) \geq 0$ since the output layer is a softmax. This loss function is a specialization of the Bradley-Terry model [187] for estimating score functions from pairwise preferences. Similarly to the low-level preference model [170], we resort to indirect training of this model via preferences expressed by the teacher. However, here, rather than working on low-level segment trajectories, our method operates on sub-goals, making the training much less reliant on large amounts of preferences to be accurate and more intuitive for the overseer while being computationally more efficient.

Besides, it has several advantages. 1) It is more natural and faster for a human to compare a pair of sub-goals that coming with an optimal sub-goal. 2) When a new sub-goal is added to memory, it only requires to compare the new sub-goal with current best sub-goals. 3) Human with partial knowledge about the task (i.e. non-expert) can provide feedback signal since they can provide information about pairs of sub-goals. That is, if an expert has poor knowledge about one pair of sub-goals, it can still provide information about other pairs of sub-goals. On the other hand, in standard imitation learning, human cannot provide any feedback - the teacher cannot selects the next optimal sub-goal or action. As a result, we found that a non-expert can provide valuable feedback that can be used to expand the possible applications of deep reinforcement learning agents.

### Optimizing the Meta-Controller

We can train the meta-controller $\tau$ based on high-level preference elicitation provided by the predictor network. At time-step $t$, the meta-controller network receives the current state $s_t$ and recommends the next sub-goal $g$. In this paper, we consider that, along with the extrinsic reward, the meta-controller also receives a *preference* bonus $b_t$ that rewards the meta-controller for agreeing with human preferences. Please note that since the reward function $b_t$ is often non-stationary, it is useful to normalize the scale of the rewards. In order to keep the rewards on a consistent scale, we normalize the preference reward by dividing it by a running estimate of the standard deviations of the preference returns.

Thus, the reward $r_t^e$ that receives the meta-controller for selecting the sub-goal $g$ in a state $s_t$ is the following:

$$r_t^e = \beta \cdot R_e + (1-\beta) \cdot b_t = \beta \cdot R_e + (1-\beta) \cdot \overline{\hat{p}(s_t, g|\theta^*)} \tag{9.4}$$

where $R_e$ is the extrinsic return obtained by selecting the sub-goal $g$ and then following the subpolicy $\pi_g$, $\beta$ is a hyperparameter to weight the importance of the preference bonus, and $\overline{\hat{p}(s_t, g|\theta^*)}$ is the normalized reward calculated by the predictor network. Please note that this formulation accounts for the possibility that human preferences can be suboptimal. Rather than always selecting the preference-based sub-goal, we provide a bonus to the meta-controller. This enables the meta-controller to correct possible human mistakes based on interactions with the environment and discover alternative options that are unknown to the demonstrator.

### Selecting Queries

One common solution to decide how to query preferences relies on ensemble-based uncertainty estimates, as done by Christiano et al. (2017) [170]. However, such an approach tends to be computationally expensive [159]. Moreover, we found this strategy less accurate when operating in the low data regime (with very few preferences), as in our work.

Therefore, we introduce a different approach to decide how to query preferences based on the predictor's confidence. This prompts the predictor to query preferences only for sub-goals that are found with a low degree of confidence. The predictor network's confidence can be modeled using bayesian models. However, in the context of RL, their computational cost can be prohibitive. This problem can be mitigated by using an estimation of Bayesian inference. It was shown that the use of dropout can be interpreted as a Bayesian approximation of Gaussian process [180]. Therefore, we introduce a dropout layer before every weight layer of our predictor network. To estimate predictive confidence $c(s,g)$ of a tuple $(s,g)$, we collect the results of stochastic forward passes through the predictor network and then measure the variance in the prediction:

$$c(s,g) = \mathbb{E}_{d_j \sim D}[\hat{p}^{d_j}(s,g|\theta^*) - p]^2 \tag{9.5}$$

where $\hat{p}^{d_j}(s,g|\theta^*)$ represents the model's prediction with dropout mask $d_j$, $D$ is a set of dropout masks, and $p$ is the predictive posterior mean, $p = \mathbb{E}_{d_j \sim D}\hat{p}^{d_j}(s,g|\theta)$. A score above a threshold $t_{qry}$ results in a query. Since the forward passes can be done concurrently, the method results in a running time identical to that of standard dropout. We can expect the variance of unknown and far-away tuples to be larger than known tuples. Furthermore, using a large number of dropout masks makes this approach much more accurate than training an ensemble of models, while preventing overfitting of recorded preferences.

### 9.3.2 Combining Low-level Policies with Curiosity

At the low-level, we train subpolicies $\pi_g$ where $g$ is the sub-goal pursued by the subpolicy. In particular we can use any standard off-policy reinforcement learning algorithm like DDPG where each learner accumulates its own experience. Similarly to prior work on hierarchical reinforcement learning, we assume access to a *terminal(s,g)* function that indicates the termination of the sub-task and a *done(s,g)* function that indicates a failure or success of the goal being pursued. Hence, we can derive a pseudo-reward function $r^{e,g}$. The subpolicies are trained to maximize this excepted pseudo-reward, defined as follows:

$$r^{e,g} = \begin{cases} 1 & \text{if done(s,g)} \\ -1 & \text{if terminal(s,g)} \wedge \neg\text{done(s,g)} \\ 0 & \text{otherwise} \end{cases} \tag{9.6}$$

However, this pseudo-reward function $r^{e,g}$ may be very sparse depending on the time-horizon of the sub-task. This will become pressing when attempting to scale this method to practical tasks where the number of steps to reach the sub-goal being pursued can be large or the sub-task be very challenging. In this chapter, we extend the formulation of the pseudo-reward to tackle sparse reward sub-tasks, drastically reducing the number of interactions to learn subpolicies.

To this end, we propose to make use of a curiosity-based intrinsic reward, $r^i$. This bonus is summed up with the sub-task reward (i.e. pseudo-reward), making rewards dense and more suitable for learning, without the need for handcrafting a reward function for each sub-goal.

Overall, at every time step $t$, $\pi_g$ receives the following reward:

$$r_t = r_t^{e,g} + \alpha \cdot r_t^i \tag{9.7}$$

where $\alpha$ is a hyperparameter of our method to weight reward components, $r^{e,g}$ is the pseudo-reward, and $r^i$ is the intrinsic reward. In all our experiments, we use Random Network Distillation (RND) [20] as formulation of curiosity, which consists of two neural networks. A fixed network takes an observation to an embedding $f : \mathcal{S} \to \mathbb{R}^k$ and a reconstructor networks that aims to predict the output of $f$ on the current observation, $\hat{f} : \mathcal{S} \to \mathbb{R}^k$. Please note that only the reconstructor network is trained by gradient descent to minimize the prediction distance with $f$. The intrinsic reward $r^i$ is calculated via:

$$r_t^i = ||\hat{f}(s_{t+1}) - f(s_{t+1})||^2 \tag{9.8}$$

An important "trick" that we found is to use a global estimator of curiosity rather than an estimator for each subpolicy. Intuitively, the agent does not need to revisit states that have already been visited by other subpolicies. Furthermore, using a global curiosity estimator ensures that only relevant parts of the state space are collected by the agent. That is, the agent collects experience that has never been explored (or potentially not fully explored) by other subpolicies and that are relevant for reaching the goal being pursued, $g$.

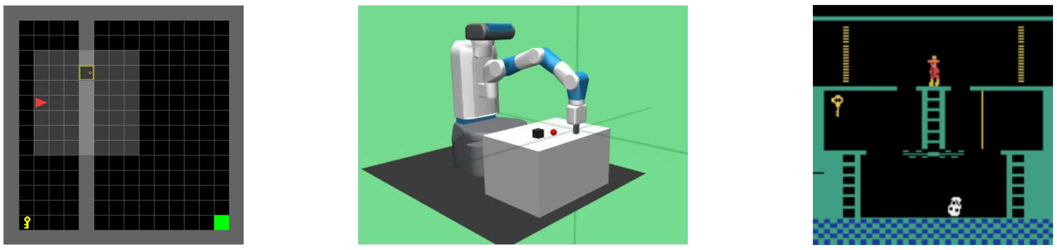

(a) Door & key (b) Fetch Push (c) MR

Figure 9.2: Frames from Door & Key, Fetch Push, and Montezuma's Revenge (MR).

**Intrinsic Skill Discovery**

Rather than relying on manually created sub-goals as done in most prior work, we introduce *Intrinsic Skill Discovery* (ISD) to automatically discover sub-goals with a minimal computational overhead. Intuitively, it has been shown that spikes in the intrinsic curiosity mostly correspond to meaningful events [182, 20]. For instance, in Montezuma's Revenge, large spikes correspond to events such as *passing an obstacle*, *picking an object*, *interacting with a torch*, or *using a ladder*. Therefore, we can hypothesize that some of the states where curiosity spikes can be considered as potential sub-goals. Moreover, we would like to emphasize that irrelevant sub-goals will be discarded (i.e. not selected) by the high-level policy.

The proposed method works as follows. We consider $\boldsymbol{\Omega}$ a set of $\boldsymbol{M}$ prior curiosity models (e.g. RND [20]). To fill $\boldsymbol{\Omega}$, every $\boldsymbol{T}$ time steps we substitute the oldest model in memory with the current model. By doing so, we can measure the curiosity progress at different time-scales. Curiosity progress $\rho(s_t)$ in a state $s_t$ can now be estimated by measuring the average distance between the current model and previous models at different time-scales:

$$\rho(s_t) = \frac{1}{|\Omega|} \sum_{\theta_{old} \in \Omega} \left[ ||\hat{f}(s_{t+1}|\theta) - f(s_{t+1}|\theta)||^2 - ||\hat{f}(s_{t+1}|\theta_{old}) - f(s_{t+1}|\theta_{old})||^2 \right] \tag{9.9}$$

where $||\hat{f}(s_{t+1}|\cdot) - f(s_{t+1}|\cdot)||^2$ is the curiosity estimation parametrized by a set of trainable parameters $\theta$ (current model) or $\theta_{old}$ (prior model). After the curiosity progress score computation, a new sub-goal with a score in the lowest 5-percentile is added to memory if the similarity with any other sub-goals is smaller than a similary threshold $b_{similarity}$. This check is necessary for the following reason: the threshold $b_{similarity}$ induces a discretization in the sub-goal space which enables to store "distinct enough" sub-goals.

## 9.4 Experiments

In this section, we first describe implementation details and the tasks to be completed by the agent. Then, we answer the following questions:

- Is HhP robust to noisy human preferences?
- Is automatic sub-goal discovery an efficient way to design sub-goals?

- How much do preferences and curiosity help?
- What is the impact of the query budget on the performance?

Finally, we conduct experiments in multiple tasks from the Minigrid environment, MuJoCo suite, and Atari benchmark suite (Figure 9.2).

### 9.4.1 Implementation Details

In this section, we refer to our algorithm as hierarchical human preferences (HhP). In all the experiments the observations are given in the form of images. The RGB images are converted to 84×84 grayscale images. The input given to the policy network consists of the current observation concatenated with the previous three frames. We use DDQN with prioritized experience replay (Minigrid and Atari) or DDPG (MuJoCo) at the low-level of our algorithm, and RND with similar hyperparameters as in the original implementation [20]. At the high-level, the predictor network consists in a sequence of three convolutional layers with (32,64,64) filters each, stride: 4,2,1, kernel size of: 8×8,4×4,3×3, and padding 1. We apply a rectifier non-linearity after each convolutional layer. The output of the last convolutional layer is passed to a serie of two fully connected layers of size 256. Training is carried out with a fixed learning rate of $10^{-3}$ using the Adam optimizer [72], with a batch size of 128. For Minigrid tasks, we use $t_{qry} = 0.26$. For Atari games, we set $t_{qry} = 0.42$. We set the coefficient of rewards $\beta = 0.7$ and $\alpha = 0.5$. In MuJoCo we set the query budget to 750 and 3000 in Atari games. The frequency update $T$ of the prior model buffer, $\Omega$, is set to 15,000 and we found $M = 4$ to be sufficient. In all our experiments, predictor confidence is estimated based on 500 dropout masks with $p = 0.1$.

### 9.4.2 Environments

We conduct experiments on several sparse reward environments:

- Minigrid: In Minigrid [53], the world is a partially observable grid. Each tile in the grid contains nothing or one object: ball, box, door, wall, or door. An observation consists of the visible cells surrounding the agent. The agent can choose among 7 possible actions: turn left, turn right, move forward, pick up an object, drop the object being carried, open a door, and complete the task. The Door & Key task consists of two rooms connected by a door. The agent has to pick up the key in order to unlock the door and then get to the goal. In KeyCorridor, the task is similar to Door & Key but there are multiple rooms and multiple doors. In Multiroom, the agent has to open a serie of doors to reach the final goal. In ObstrMaze, the doors are locked, the keys are hidden in boxes and doors are obstructed by balls. Solving such sparse tasks is challenging since the object locations are randomized and the agent only receives a positive reward +1 when it reaches the final goal. These tasks also require sequential decision making (e.g. saving the key to open a distant door) to reach the final goal.
- MuJoCo: In the MuJoCo environment, the agent controls a 7-DoF Sawyer arm.

Table 9.1: Final mean performance (± std) of our method with various sub-goal creation strategies on Atari games and average success rate (± std) on KeyCorridorS4R3 and Fetch Pick & Place. Averages over 10 runs are shown after 100M steps (Atari), 5M steps (KeyCorridorS4R3), and 1.5M steps (Fetch Pick & Place).

| | Maximum Mean Score (at convergence) | | | | Success rate | |
|---|---|---|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Gravitar | Pitfall | KeyCorridorS4R3 | Pick & Place |
| HhP / $\varrho$=0.05 | 18,657±1,168 | 66,145±1,609 | 2,941±963 | 992±205 | 0.90±0.04 | 0.94 ±0.04 |
| HhP / $\varrho$=0.1 | 16,569±1,453 | 63,693±2,272 | 2,836±1,231 | 759±301 | 0.84±0.07 | 0.79 ±0.07 |
| HhP / $\varrho$=0.2 | 15,028±1,625 | 58,304±2,263 | 2,564±1,456 | 704±416 | 0.77±0.12 | 0.76 ±0.09 |
| HhP / random | 12,698±2,865 | 49,217±6,580 | 2,547±687 | 576±220 | 0.65±0.11 | 0.39 ±0.14 |
| HhP / ISD | 19,914±979 | 68,874±1,265 | 3,100±714 | 1,288±169 | 0.91±0.02 | 0.97 ±0.03 |

The end-effector (EE) is constrained to a 2-dimensional rectangle. We consider four Fetch tasks where the agent controls the robotic arm: (1) Fetch Reach, (2) Fetch Push, (3) Fetch Pick & Place, and (4) Fetch Slide.

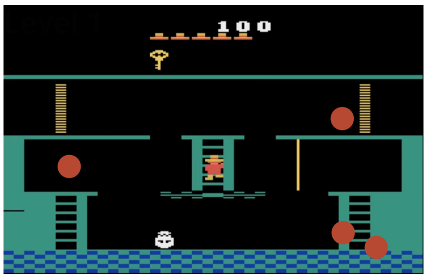
- Atari: We conduct experiments in the Arcade Learning Environment [86], including: Montezuma's Revenge, Private Eye, Gravitar, Pitfall, Seaquest, and Solaris. These are hard exploration games that might also contain deceptive rewards such as in Pitfall. In these games, the agent has to navigate in complex environments, explore labyrinths, avoid enemies, pass obstacles, or collect objects. In Montezuma's Revenge the player explores rooms filled with enemies, obstacles, traps, in an underground labyrinth. In Pitfall, in order to recover 32 treasures, the player must maneuver through numerous hazards, including pits, quicksands, and rolling logs. In Seaquest, the agent must shoot enemies to survive.

### 9.4.3 Ablation Analysis

We have conducted ablation studies on some Atari games to investigate: (1) the robustness to imperfect preferences, (2) the impact of the sub-goal discovery strategy, (3) the impact of the components on the performance, and (4) the impact of the query budget on the performance.

#### Robustness to Imperfect Preferences

In the above experiments, we assume perfect preferences (i.e. the demonstrator always provides optimal feedback). However, the teacher might select not the best preferences or even lack knowledge about a sub-goal. We study how our agent performs when imperfect preferences are generated by the teacher. In order to generate imperfect preferences, we randomly provide a non-optimal preference with a probability $\varrho \in \{0.05, 0.1, 0.2\}$. We report in Table 9.1 (line 1-3) the performance of our framework. We observe that HhP can still achieve acceptable performance. For instance, on Montezuma's Revenge, scores obtained by our method remain close to the best scores. Even though the proposed method performs slightly worse in the imperfect setting, it still improves performance as compared to the prior methods. A reason is that the agent can leverage high-level


(a) Expert labelling

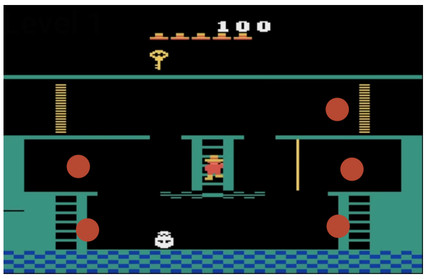

(b) Automatic discovery

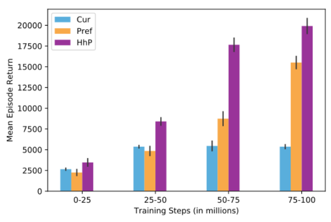
Figure 9.3: Screenshots from Montezuma's Revenge: manually created sub-goals (left), and automatically discovered sub-goals (right). The sub-goals (the agent's position) are denoted by a red dot.

preferences of similar sub-goals and learn from interactions to correct mistakes made by humans. The experimental results demonstrate that our method is reasonably robust to noise in the preferences, and hence a non-expert teacher can provide a feedback signal to the agent.

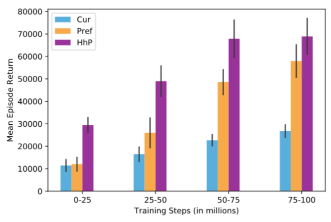
### Impact of the Sub-goal Discovery Strategy

To see the potential benefits of using an automatic sub-goal creation strategy, we explore the performance of HhP with sub-goals created using the following strategies: randomly discovered, and automatically discovered (ISD) (Table 9.1, line 4-5). Note that in complex and temporally-extended tasks like Atari games, manually creating all sub-goals is deemed infeasible. During late training, the agents trained with ISD always reached higher performance. It validates that our strategy can discover a diverse range of sub-goals that cover key events of the task. On the other hand, randomly selecting sub-goals deteriorates the performance. Nevertheless, even with randomly discovered sub-goals, the meta-controller and the teacher can still discard irrelevant sub-goals and select meaningful sub-goals, keeping the performance in an acceptable range. However, the final performance of HhP trained with randomly discovered sub-goals is capped since the agent cannot keep learning once all the meaningful sub-goals are mastered.

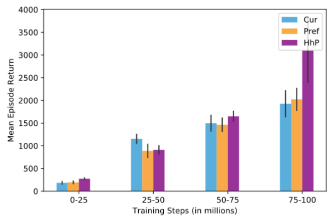
To further validate the depicted strategy, we show the sub-goals discovered in the first room of Montezuma's Revenge and compare it against the optimal sub-goals created by an expert [183]. For simplicity, we report in Figure 9.3 the location of the agent for each sub-goal. We can see that the automatically discovered sub-goals are very similar with manually created sub-goals. For instance the sub-goal *pick a key* was found by both strategies. A noticeable difference is that our strategy found a larger number of sub-goals compared to experts. Thus, this experiment validates that ISD can be used as a robust alternative to manually created sub-goals.



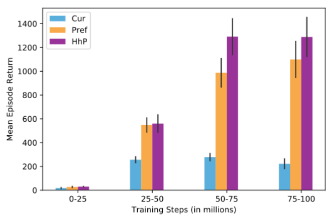
(a) Montezuma's Revenge



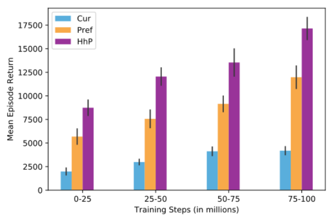
(b) Private Eye



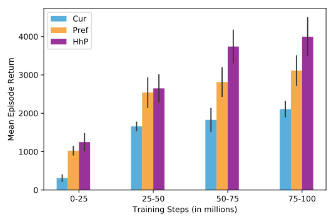
(c) Gravitar



(d) Pitfall



(e) Seaquest



(f) Solaris

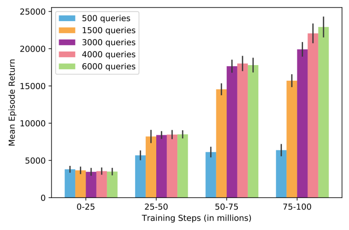
Figure 9.4: Performance for different experimental setups on 6 Atari games after 100M agent steps. Results are averaged over 10 runs (±std).

**Impact of the Components on the Performance**

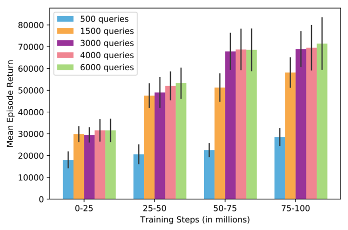
In order to measure how much do high-level preferences and curiosity help, we compare the following experimental setups:

- Curiosity (Cur): The model is trained solely based on its own curiosity.
- Preferences (Pref): The model is trained from human high-level preferences, without curiosity.
- Full model (HhP): The model is trained from human high-level preferences and curiosity.

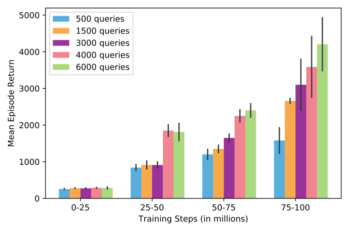
As can be observed in Figure 9.4, *Cur* produces faster learning during the early stages of the agent's training. For instance, on Pitfall after 50M training steps the agent equipped with curiosity achieves a score of $\approx 260$. However, as selecting the next sub-goal becomes increasingly difficult, learning without human preferences hurts the performance. We generally observe that curiosity significantly reduces training time of subpolicies, but has a limited impact on the gain in performance. On the other hand, learning from human high-level preferences (*Pref*) shows large improvements compared to *Cur* - human preferences help the agent to rapidly order the subpolicies. Nevertheless, learning without curiosity increases the exploration workload, requiring more interactions to reach similar performance as HhP. This experiment demonstrates that curiosity-driven exploration plays a central role in reducing the number of trial-and-errors and human guidance enables various forms of common sense reasoning, improving the overall performance.



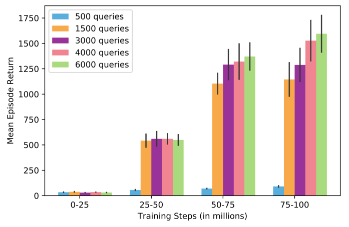
(a) Montezuma's Revenge



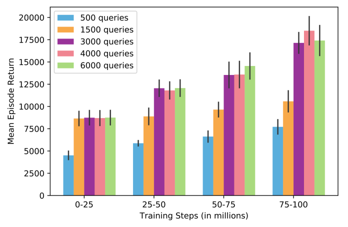
(b) Private Eye



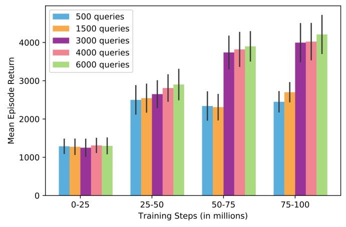
(c) Gravitar



(d) Pitfall



(e) Seaquest



(f) Solaris

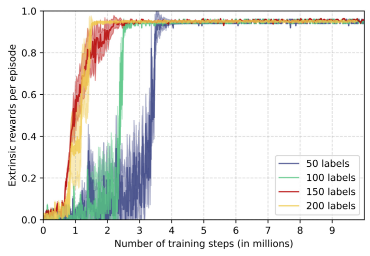
Figure 9.5: Performance of our method on 6 Atari games after 100M agent steps, for different query budgets. Results are averaged over 10 runs (±std).

### Query Budget of Preferences

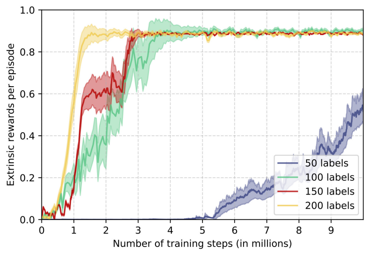
We finally report evaluations showing the effect of increased query budget. Figure 9.5 demonstrates that agents trained with a larger query budget obtain higher mean returns after similar numbers of updates. However, despite a small query budget (less than 4000), our method can still learn near-optimal policies. We can draw the observation that as the query budget increases, the learning effect on the agent gradually improves. Nonetheless, for the results with 3000 and 6000 queries, we can see that even though the number of queries significantly differs, the difference in learning effect can be negligible. This can happen when the queried preferences cover a broad enough number of sub-goals and therefore the agent does not need to make additional queries. As a result, our method leverages a small amount of preferences that cover critical sub-goals, leading to dramatic reductions in both human effort and cost of exploration.

## 9.4.4 Procedurally Generated Environments

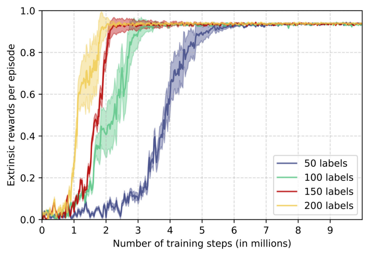
We now perform experiments on a set of four procedurally generated tasks in the Minigrid environment to evaluate the overall performance of our algorithm and its generalization ability to unseen views or appearances. Please note that due to a large number of random environment instances, this domain requires a very large number of samples for tabular algorithms. Figure 9.6 depicts the training performance on Door & 16 × 16, KeyCorridorS4R3, ObstrMaze1Q, and MultiRoomN12S10. The results of each run are averaged to provide a mean curve in each figure, and the standard error is used to make the shaded region surrounding each curve. In all cases, the use of the proposed method results in significant improvements in the performance of the policy, leading to near-optimal policies. Figure 9.6 shows that all the tasks can be solved with very few queries (i.e. less than 200 queries). As can be further seen, on Door & 16 × 16 and ObstrMaze1Q, even though HhP (50 queries) primarily progresses more slowly than HhP trained with a larger



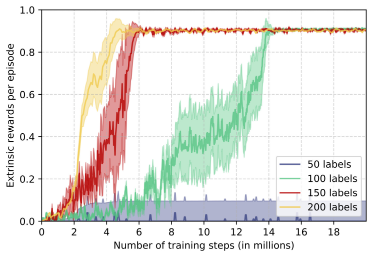
(a) Door & 16 × 16



(b) KeyCorridorS4R3



(c) ObstrMaze1Q



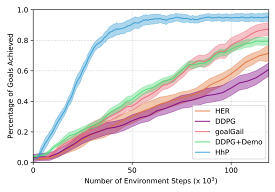
(d) MultiRoomN12S10

Figure 9.6: Performance of HhP for different numbers of queries (labels) on a variety of procedurally generated tasks in MiniGrid: Door & 16 × 16, KeyCorridorS4R3, ObstrMaze1Q, and MultiRoomN12S10. All curves (mean±std) are averaged over 10 random runs.

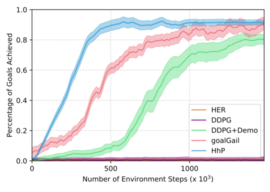
query budget (HhP (150 queries), HhP (200 queries)), it is ultimately capable of achieving similar performance. Overall, leveraging the proposed form of human guidance - high-level preferences, enables our agent to rapidly reach good performance and generalize the provided domain knowledge to unseen appearances.

### 9.4.5 Robotic Tasks

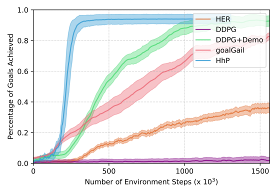
In this section we evaluate the agent on four different tasks (Fetch) from the robotic domain built on top of MuJoCo: Fetch Reach, Fetch Push, Fetch Pick and Place, and Fetch Slide. We compare our method against several baselines including DDPG [4], HER [41], DDPG+Demo [165], and goalGail [169]. We show learning curves in Figure 9.7. Our method can learn comparable or superior policies using a significantly smaller number of human feedback than goalGAIL and DDPG+Demo. For instance, on Pick and Place, on average only 156 queries were made by HhP. As excepted, it ends up reaching similar final performance, however, our method has a faster convergence rate, reducing the amount of required interactions with the environment. Furthermore, unlike our algorithm, DDPG+Demo passively access the demonstration data, so we actively provide help to our agent when it struggles. We can further observe that incrementally querying preferences over sub-goals keeps the number of required preferences very low, while enabling non-expert demonstrations - it is more intuitive for a human to judge in which direction to move the robotic arm than controlling the robotic arm. We conclude that our method provides the capability to effectively learn from multiple types of internal and external supervision.



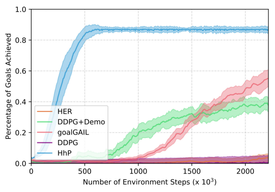
(a) Fetch Reach



(b) Fetch Push



(c) Fetch Pick & Place



(d) Fetch Slide

Figure 9.7: Learning curves averaged over 10 runs for different models: HhP, DDPG, HER, DDPG+Demo, and goalGail. The models are trained on robotic tasks from the Fetch environment.

Table 9.2: Final mean performance (mean±std) of our method and baselines on Atari games. We report the results of our method achieved over total 100M timesteps of training, averaged over 10 seeds. Some historical papers did not consider games, in which case the score is displayed as "-".

| | Maximum Mean Score (at convergence) | | | | | |
|---|---|---|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Gravitar | Pitfall | Seaquest | Solaris |
| PPO [38] | 1,259±610 | 50±37 | 1,826±255 | -21±6 | 664±258 | 1,021 ±199 |
| PPO+RND [20] | 8,152±653 | 8,666±1051 | **3,906±246** | -3±1 | 3,179±378 | 3,282±281 |
| DQfD [114] | 4,739 | 40,908 | 1,693 | 57 | 12,361 | 2,616 |
| Imitation [114] | 576 | 43,047 | 248 | 182 | 195 | 3,589 |
| Pref (No-Demo) [170] | 23 ± 5 | 256 ± 69 | - | - | 1,011 ± 216 | - |
| Pref (Demo) [159] | 2,829 ± 714 | 50,159 ± 11,657 | - | - | 515 | 142 |
| Average Human [77] | 4,753 | 69,571 | 3,351 | 6,464 | 20,182 | 12,327 |
| HhP (ours) | **19,914 ± 979** | **68,874 ± 8,265** | 3,100 ± 714 | **1,288 ± 169** | **17,143 ± 1,231** | **3,996 ± 511** |

## 9.4.6 Hard Exploration Games

We also test the proposed method on six difficult exploration Atari 2600 games from the Arcade Learning Environment (ALE) [86]: Montezuma's Revenge, Private Eye, Gravitar, Pitfall, Seaquest, and Solaris. In the selected games, training an agent with a poor exploration strategy often results in a suboptimal policy. We compare our method to the performance of PPO [38], PPO+RND [20], DQfD [114], Imitation [114], Pref (No-Demo) [170], and Pref (Demo) [159]. The results are shown in Table 9.2. We consider the final mean performance of 10 training runs with the same set of hyperparameters. It is observed the plain PPO algorithm obtained a score close to zero and could not solve most of the tasks. On the other hand, a hierarchical decomposition of the tasks drastically reduces the number of required interactions. We also found that human preferences is vital - even with significantly more labels, DQfD fails to reach scores comparable to our method. We hypothesize that human demonstrations often deviate from the agent's goal and do not enable efficient common sense reasoning - they cannot be used by the agent for planning or reasoning but are limited to specific situations. Please note that Gravitar is an exception, the curiosity-based approach PPO+RND is hard to beat. On Private Eye, our method achieves a mean score higher than Pref (Demo) trained without curiosity. Overall, on the six tasks, using human high-level guidance enables our agent to achieve various forms of common sense reasoning and avoids learning from scratch, surpassing standard reinforcement learning baselines. Moreover, even with access to a smaller number of labels, our method outperforms other preference-based methods that make use of low-level preferences. One reason is that HhP requires much less human queries to efficiently drive the agent's learning. We observed a reduction by a factor of $\approx 2$ over models that use low-level preferences (Pref(No-Demo), Pref(Demo)). The depicted method also drastically reduces the total amount of feedback compared to DQfD on Montezuma's Revenge (17949), Private Eye (10899), Gravitar (15377), Pitfall (35347), Seaquest (57453), and Solaris (28552). We should also emphasize that comparing pairs of sub-goals appears to be easier than providing an optimal action or comparing low-level trajectories. Besides, we further observed that using curiosity is critical for reducing the number of low-level interactions, especially in tasks with complex dynamics like Montezuma's Revenge or Private Eye.

## 9.5 Discussion

This chapter takes a step towards achieving human-like sample-efficient learning by acquiring the capability to learn from multiple intermediate learning signals. We have constructed a mechanism to utilize high-level preferences along with curiosity. Precisely, to greatly reduce the human involvement, we proposed to introduce non-expert human guidance in the form of high-level preferences between sub-goals. Even very small amounts of preferences let us outperform prior imitation-based approaches on multiple sparse reward tasks. We further make use of curiosity to improve sample efficiency of low-level subpolicies and we derive a technique to automatically discover sub-goals, which alleviates the need to handcraft sub-goals. In the depicted work, human preferences provide prior assumptions about the domain to the agent, avoiding learning from scratch and facilitating various forms of common sense reasoning. Precisely, grafting human preferences onto the agent allows it to make decisions based on its own common sense (without having to experience a situation) and improves the capability of the agent for planning. On the other hand, curiosity is used for control of sub-tasks too challenging for even humans to perform well. Furthermore, curiosity enables our agent to learn to outperform the demonstrator.

The experiments demonstrate the effectiveness of this approach by achieving significant improvements on notoriously difficult tasks such as game-playing (Atari games), visual navigation (Mingrid), and robot control (MuJoCo). Remarkably, our agent exceeds expert-level performance in several domains by a large margin. To the best of our knowledge, we were among the first to integrate internal and external guidance for sample-efficient reinforcement learning.

While these are just the initial steps, we believe that the proposed research direction is promising and its exploration will be useful to the research community.

# Chapter 10

# Conclusion

## 10.1 Discussion and Future Work

In recent years, deep reinforcement learning has attracted attention in a variety of application domains. In spite of recent successes in simulated domains featuring complex motions and many degrees of freedom, solving real-world tasks remains an open problem. For many of these tasks, significant amounts of data and/or computation is required to reach reasonable performance. Namely, when working in the real world, learning from massive amounts of data may be impractical or too costly. More troubling, real-world tasks often involve poorly-defined or sparse rewards, further increasing the number of training steps by several orders of magnitude.

This thesis contributes to the state-of-the-art in the domain of reinforcement learning. Our specific focus was on sample-efficient learning to expand the availability of reinforcement learning to real-world domains. Motivated by this ambition, we devoted our research to make reinforcement learning agents much less reliant on large amounts of interactions by improving their cognitive capabilities. Overall, we aimed to take a step towards human-like sample-efficient learning. To this end, we considered three open problems: (1) learning with sparse rewards via curiosity-driven exploration, (2) leveraging human guidance and domain knowledge, (3) learning from heterogeneous types of internal and external supervision. Our main contributions are summarized in the next section.

In Part I, we have proposed to improve exploration efficiency when extrinsic rewards are sparse or poorly-defined. This is a serious problem since hoping to reach a goal state by chance is most likely to be futile for all but the simplest of environments - the agent cannot be improved until a reward is obtained. In contrast with humans that are accustomed to operate with rewards that are so sparse, reinforcement learning lack an important cognitive capability: curiosity. In order to enable learning in such a setting and drastically reduce training time, we have designed curiosity-driven agents. Curiosity is a way of encouraging the agent to visit unexplored regions and acquire new skills that might come useful for pursuing rewards in the future. While several formulations of curiosity have been proposed in the literature, modeling and reproducing the development of a natural curiosity capability in artificial agents remains an open question. In this thesis, we tackled three major limitations inherent in curiosity.

- First, we proposed to generate an exploration bonus based on the agent's knowledge about its environment in order to encourage the gradual acquisition of new skills. Most prior work aims to model environmental dynamics, however, it tends to limit long-horizon performance due to model drift and does not directly consider the agent's knowledge - rather than visiting all possible states the agent should focus on task-relevant regions. Our approach (GoCu) escapes these drawbacks by operating directly on the agent's policy and encouraging the acquisition of task-relevant skills (Chapter 3).

- Second, a potential limitation of our proposed method as well as prior approaches is the *vanishing issue* inherent in curiosity. As the agent becomes more familiar with its environment, curiosity may vanish quickly during training, leaving the agent with no incentive to further explore the environment and reducing its feedback to extrinsic reward only. Moreover, curiosity-driven agents tend to fall and stay trapped in poor local optima due to local sources of entropy in the environment (e.g. random transitions). On the other hand, we addressed both challenges by proposing a robust formulation of curiosity, PoBP, that uses the agent's learning progress on a multi-step horizon scale (Chapter 4).

- Finally, while very good results have been achieved on some hard exploration tasks, these algorithms face a fundamental limitation: they do not explicitly encourage and promote deep exploration. That is, curiosity only captures the consequences of short-term decisions on the environment. Global exploration that involves coordinated decisions over long time horizons is beyond the reach of most methods. In Chapter 5, we presented a formulation of curiosity that can capture meaningful visual features and salient environmental dynamics at different scales; and then built an algorithmic framework (FaSo) that combines two curiosity components, explicitly promoting deep exploration. Please note that exploration is robust with respect to local sources of entropy in the environment.

A first contribution of this thesis has been to show that an agent consistently achieves the tasks much faster and more efficiently with the proposed curiosity mechanisms. The objective was to reproduce a seemingly natural cognitive capability of humans: intrinsic motivation. Humans dedicate much time and energy to exploring and gathering information; and often the search for information is unrelated of a foreseeable extrinsic gain/reward, as if learning were reinforcing in and of itself. This behavior emerges through our capability to make intrinsically motivated choices, our intrinsic desire to learn and understand. This thesis partially reached this objective, as we have demonstrated that our methods drive the agent to acquire novel skills, seek novel situations, and continuously reinforce its own knowledge about the environment. In order to measure the improvements of this capability over prior work, we presented evaluations on a wide range of tasks and compared the depicted methods to state-of-the-art curiosity-driven methods. To fairly compare the algorithms, we utilized the same learning algorithms (PPO or A2C) and hyperparameters (with the exception of a few prior work that are fundamentally different). Thus, our methods and prior work only differ by the way they make intrinsically motivated choices. To quantify the improvements of the above-mentioned cognitive capability, we mainly focused on two criteria: the agent's performance, and the number of training steps to converge. For the three methods we presented results on multiple standard benchmark tasks, providing general indications about the performance of each model and allowing us to compare them together. Nevertheless, these criteria may not be enough to assess specific properties such as whether the agent develops the capability to make intrinsically motivated choices in the long term, or whether the agent's motivation is robust to local

| | Characteristics | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Environments | Sparse Rewards | Very Sparse Rewards | Deceptive Rewards | Discrete Actions | Continuous Actions | Complex Dynamics | Stochasticity | Procedurally Generated |
| Minigrid | ✓ | | | ✓ | | | | ✓ |
| Super Mario Bros | | ✓ | | ✓ | | ✓ | | |
| Atari | | ✓ | ✓ | ✓ | | ✓ | | |
| Malmo | ✓ | | ✓ | ✓ | | | | ✓ |
| DMLab | ✓ | | | ✓ | | | ✓ | ✓ |
| Trading | ✓ | | | | ✓ | ✓ | | |
| Fetch Robotic | ✓ | | | | ✓ | | | ✓ |
| ShadowHand Robotic | | ✓ | | | ✓ | ✓ | | ✓ |

Table 10.1: Characteristics of the environments used to evaluate our methods.

sources of entropy in the environment. Thus, we also carried numerous ablation analyses and used additional environments based on the properties that we wanted to evaluate. We summarize below which model is most suitable for each environment, as well as the characteristics of each model.

In order to assess overall performance of our agents and specific properties that we wanted to evaluate, we utilized a large number of environments featuring different characteristics. The Minigrid environment provides a set of sequential tasks with various degrees of sparsity. One difficulty is the very large number of instances since the tasks are randomly generated; and the need for the agent to understand how to save/use objects. In general, environmental dynamics are relatively simple. The Super Mario Bros environment involves much more delay between the rewards and more complex visual features. In Atari games, the agent has to discover complex exploration patterns and learn from extremely delayed rewards. Besides, some games feature deceptive rewards and a large range of complex game mechanics even for human players, allowing a more diverse set of behaviors as it includes multiple distinct objects/enemies with their own courses of evolution under the agent's actions. In contrast to these environments, the Malmo environment consists of a set of 3D tasks where the agent must learn to navigate, orientate and control. The DMLab environment has similar properties but contains stochasticity in the actions and/or observations. We also evaluated some approaches on non-visual tasks, a trading environment. In this environment the actions are continuous and the observations are given in the form of continuous values. Finally, we also used two robotic environments where actions are continuous. In Fetch Robotic, rewards are sparse and tasks relatively simple. In ShadowHand Robotic, rewards are extremely sparse and the large degree of freedom in the action space makes these tasks much more complex. Moreover, these dexterous in-hand manipulation tasks have a huge number of controllable parameters and challenging simulated dynamics. We show a summary of the environmental characteristics in Table 10.1. Note that in some experiments we modified the environments in order to test additional properties (e.g. stochasticity, dense rewards).

Through evaluations on a wide range of domains, we found intrinsic motivation (i.e. curiosity) to be an essential capability for enabling reinforcement learning in the real world, where rewards are naturally extremely sparse. We demonstrated that the depicted approaches outperformed state-of-the-art methods both in sample-efficiency and learning performance. We also studied whether curiosity may hurt the performance in some settings. We found that curiosity always improves the performance compared to pure reinforcement learning, except when rewards are dense. In this setting, curiosity may encourage the agent to try sub-optimal exploration strategies, increasing the exploration workload. We evaluated this scenario in multiple environments that we modified to provide dense rewards. Even though curiosity may slightly increase the training time, the experimental results showed that our methods did not significantly deteriorate the performance. We conjecture that curiosity-driven agents have a relatively low chance to occasionally perform notably worse when extrinsic rewards are dense. Please note that curiosity may improve final performance when the dense rewards are poorly-defined, by encouraging the agent to escape from local optima. Another possible issue with the proposed formulations of motivation is that they solely deal with a single observation input. Typically, we presented mechanisms that take as input an image or a set of continuous values (e.g. gripper positions, sensor data). Although prior work and benchmark environments both use this setting, it remains unclear how to derive curiosity-driven mechanisms that can handle multi-sensor tasks.

We now summarize our findings regarding the capabilities of our curiosity-driven approaches as well as the impact on the choice of the model. In Part I, we compared and studied the three approaches on multiple (sparse) domains including Atari games, Super Mario Bros, Minigrid, Mujoco, and DMLab. We observed that FaSo could achieve the highest performance in the majority of domains. The capability to achieve high-level exploration makes FaSo the most suitable approach for long-term tasks and hard exploration tasks - with extremely sparse rewards, such as Atari Games and Super Mario Bros. This method is also the most effective to deal with procedurally generated tasks or changing environments such as Minigrid. In the absence of task reward - tasks with no extrinsic reward (e.g. Super Mario Bros (No-reward)), PoBP and FaSo generally achieve similar performance, both receiving enough indirect supervision for learning useful behaviors. One might question why this scenario is interesting. In fact, it can be used to determine how likely it is to stumble into the task reward in the first place. Moreover, learning without task reward may be useful to learn a set of skills/primitive actions, and reuse them in a more complex hierarchical system. On the other hand, PoBP is very robust to distractor rewards, as well as local sources of entropy in the environment (stochasticity). Especially, PoBP is extremely robust to local optima and therefore should be prioritized in tasks featuring randomness, such as in DMLab. In robotic environments (e.g. Mujoco), it is natural to decompose the overall task into a set of skills and gradually learn them. Thus, GoCu can be used to rapidly learn skills that could be applied to new settings.

In this thesis, we established results that show significant improvements in sample efficiency. We would like to emphasize that these results translate to the agent's training time. Since the depicted curiosity-driven methods have a limited computational cost, we can expect to also significantly reduce the training time of our agents. Precisely, we measured the additional computational cost of the proposed methods and estimated that GoCu, PoBP, and FaSo are respectively $\times 1.23$, $\times 1.07$, $\times 1.45$ slower than plain PPO. These results were obtained for agents trained on visual inputs of size $84 \times 84 \times 3$ pixels and averaged over three tasks: Montezuma's Revenge, Minigrid 16×16, and Super Mario Bros. Note that indeed these results may slightly fluctuate depending on the machine architecture. These observations suggest that curiosity is a vital cognitive capability to enable learning from sparse rewards and reduce training time.

In Part II, we have developed the means for integrating human guidance into reinforcement learning. While curiosity was shown to be a powerful incentive to guide an end-to-end agent (i.e. learning from scratch) when rewards are sparse, human-like sample-efficient learning is also closely related to the capability of the human brain for common sense reasoning and leveraging prior assumptions about the domain. In the context of reinforcement learning, this capability can be achieved via the integration of human guidance. Precisely, the most common form of guidance is imitation learning. However, in many cases it is impractical to use human policy as guidance because demonstrations: 1) may not be available when tasks are too challenging for even humans to perform well, and 2) drastically increase human workload. In Part II, we presented initial directions towards novel types of guidance that are less expensive and more intuitive for humans, enabling human guidance in the real world and drastically improving sample efficiency.

- First, in Chapter 6 we introduced human-like planning and domain knowledge to enhance information given to the agent (DRL-EK). The central concept is to augment the agent's input with high-level information that are easy to interpret for the agent and applicable to many situations. We then derived a framework for the integration of human-like planning based on the high-level information and simple visual recognition, improving sample efficiency and facilitating various forms of common sense reasoning.
- Second, in Chapter 7, we gave a reinforcement learning agent the capability of leveraging existing human expertise. Rather than integrating human guidance and domain knowledge designed expressly to solve the task being learned, we wanted to reduce human effort by leveraging large datasets related to the task being learned. Moreover, most methods in the reinforcement learning literature lack interpretability which may limit their applicability in some real-world domains. Therefore, we presented methods for extracting rules from existing datasets; and then built an interpretable agent whose internal representation is based on these rules (Sarsa-rb($\lambda$)).
- Third, one limitation of our methods as well as most prior work is that the agent and the teacher cannot actively share insights. Hence, the agent cannot cope with the changes in the environment or query the teacher when it struggles. To overcome these challenges, we introduced the concept of active goal-driven demonstrations to query the demonstrator only in hard-to-learn and uncertain regions of the state space (goAL). Active demonstrations are easier and more intuitive to provide for a demonstrator than full demonstrations while precisely matching the agent's needs (Chapter 8).

The second contribution of this thesis has been to show that efficiency in reinforcement can be greatly improved by grafting suitable domain knowledge onto the agent. As described above, deep reinforcement learning algorithms require huge amounts of interactions also because they learn from scratch (i.e. with no prior assumptions about the domain), which entails that they lack a vital cognitive capability: common sense reasoning. Learning without prior knowledge seems to be an approach that is rarely taken in human - they use their common sense reasoning to extract initial biases as well as strategies on how to approach a problem. Humans are born with a part of the brain that is prewired to be receptive to the world and they leverage knowledge from previously learned tasks that lead to extremely efficient common sense reasoning, allowing them to learn new skills within a few trials. To achieve this level of efficiency in AI systems, research in cognitive systems has shown that many of these challenges can be addressed by exploiting domain knowledge and the coupling between domain knowledge and learning [17, 188].

As mentioned above, we have proposed novel forms of human guidance that reduce human effort and substantially improve the agent's efficiency. We have evaluated the former by comparing the number of queries to a demonstrator, the amount of demonstration data, or the difficulty to create domain knowledge (e.g. the need for an expert to craft domain knowledge, which tasks are suitable for a form of human guidance). On the other hand, quantifying the improvements of our agents for common sense reasoning and the impact on the agent's performance is challenging. Especially, when comparing different forms of human guidance, it is difficult to evaluate whether a form of guidance improves the agent's capability for common sense reasoning. An agent may perform better purely because it receives much more domain knowledge (e.g. demonstrations of all the possible situations) than prior work, but its capability for common sense reasoning may remain identical. In order to evaluate the improvements in common sense reasoning, we have made choices regarding which kind of environment to use and which properties to include in them. For instance, we have evaluated Sarsa-rb($\lambda$) on a trading task and used distinct stock market data as domain knowledge. By doing so, we have demonstrated the capability of the agent to generalize knowledge to unseen situations. On the other hand, prior work (e.g. DQfD) fail to generalize demonstrations, clearly showing improvements of Sarsa-rb($\lambda$) for common sense reasoning. We have also used the following metrics: the agent's final performance, the training speed, and the amount of knowledge/interactions with humans. We think that the combination of these metrics provides meaningful insights to compare our methods to baselines, and measure progresses. Besides, these metrics are important to understand and determine which form of guidance is more suitable for which task. Finally, we have carried ablation analyses to specifically evaluate how much the proposed forms of guidance help common sense reasoning and assess the efficiency of the different components. For example, we have designed tasks to test whether the agents can use their common sense prior when domain knowledge is corrupted / imperfect.

Overall, the depicted methods can be employed in complex tasks, depending on the available knowledge. For instance, in a task where only simple knowledge can be collected, then DRL-EK should be prioritized. On the other hand, if a teacher is available, goAL is likely to be the best performing technique. Please note that even though the active demonstrations are expressly generated for the task being learned, in this work the agent learns to reach a wide range of configurations from the same set of demonstrated trajectories. We believe that this is a good trade-off between task-specific knowledge (Chapter 6) and general knowledge (Chapter 7). Finally, when datasets are available and/or interpretability is important, Sarsa-rb($\lambda$) can be used to learn a policy in a few trials. While these methods have yet not been combined, we can expect further improvements by integrating together different sources of human guidance. Namely, it is straightforward to augment the policy loss of DRL-EK with a simple auxiliary objective to integrate active goal-driven demonstrations, in order to ensure that the provided knowledge match the agent's goal. The idea of *high-level information* (DRL-EK) could help learning of most reinforcement learning agents or be integrated into Sarsa-rb($\lambda$) to augment the model's input. Overall, improving various forms of common sense reasoning is vital for substantially improving sample efficiency in reinforcement learning.

In Part I and II, we studied the effects of internal and external guidance on the agents' efficiency. We found curiosity to be effective for exploring the consequences of low-level actions on the environment, especially when rewards are sparse. On the other hand, optimal planning or reasoning can be improved via external supervision. In order to expand the availability of reinforcement learning agents to real-world domains - make them less reliant on large amounts of interactions, it is necessary to combine the strengths of both of these approaches. In other words, human guidance should be used for high-level decision-making such as planning/reasoning, while curiosity is a powerful source of endogenous motivation. Especially, we cannot expect to always use human guidance for low-level control or execution since human knowledge are often incomplete, noisy, or difficult to collect.

- In Chapter 9, we proposed a hierarchical reinforcement learning model that leverages the hierarchical structure of the task to integrate different modes of supervision at different levels. At the high-level, we explored non-expert guidance in the form of high-level preferences between sub-goals, leading to drastic reductions in both human effort and cost of exploration. At the low-level, we made use of curiosity to further improve sample efficiency and drive the learning of subpolicies, particularly in tasks featuring sparse rewards. We further demonstrated how curiosity relates to sub-goal discovery.

The central idea in Part III was to combine the strengths of curiosity and human guidance. This idea can be built into a system by grafting human guidance and curiosity at different levels of the agent's decision making, greatly reducing the number of required interactions to reach reasonable performance while keeping human effort at a minimum. For instance, the depicted method reduces the total amount of feedback by a factor $\approx 6$ on Montezuma's Revenge and $\approx 11$ on Pitfall, compared to the well-known DQfD algorithm (see details in Chapter 9). To assess how intrinsic motivation and common sense reasoning interact together, we presented an evaluation in environments with specific properties. We observed that common sense reasoning drastically improves the agent's performance. On the other hand, intrinsic motivation reduces the training time of the agent. Additionally, curiosity is the key to better-than-expert performance by guiding the agent in the quest of knowledge beyond the expert's knowledge. We further evaluated the model with and without human guidance / curiosity, in order to test how much "building blocks" contribute to learning policies. Overall, the agent achieved much greater sample efficiency and performance than pure curiosity-driven approaches and human-guided agents.

We should also emphasize that an important achievement in Section II and Section III was to drastically reduce human effort and enable human guidance in tasks that are too challenging for even humans to perform well. So far, human guidance was often limited to demonstrations, however, this form of guidance may be impractical in real-world settings, narrowing down the possible applications of reinforcement learning. For instance, demonstrating many possible situations in complex real-world tasks (e.g. autonomous driving) is deemed infeasible. In light of this observation, we presented approaches that leverage novel forms of guidance. In DRL-EK, rather than using low-level knowledge (e.g. demonstrations), we leverage simple high-level information about the environment, which can be created by a non-expert in just a few minutes. In Sarsa-rb($\lambda$), we eliminate the need to expressly create domain knowledge by learning from existing datasets. In goAL, we only request short goal-driven demonstrations when the agent struggles and is not confident, reducing the number of demonstrations from several hundred to less than 50 (ShadowHand) or 20 (Fetch). To further reduce human involvement and enable non-expert feedback, we have extended the idea of active cooperation to high-level human preferences between sub-goals (HhP). The central theme in goAL and HhP is to allow the agent to actively identify the need for and request domain knowledge for specific regions or skills. As a result, the agents operate with increasing autonomy as they improve at the tasks, eliminating the need for unnecessary domain knowledge of already acquired skills, and reducing both the training time and the workload of the demonstrator.

Our contributions open several avenues of research. Now, we describe some of the future directions that are linked to extending the approaches presented in this thesis.

**Hierarchical Intrinsic Motivation.** The presented curiosity-driven methods dealt with different issues inherent in curiosity, such as how to discover long-term exploration behaviors. Ideally, it would be desirable to also take into account the hierarchy of the task like done by humans [189]. It was shown that humans do not solely rely on their sense of curiosity, but also generate intermediate goals. Integrating this finding is our research is desirable, however, many questions remain open: How do humans select these goals? How do these goals relate to curiosity? We believe that understanding human intelligence and psychology is a key factor in solving these questions.

**Few-Shot Style Exploration.** In this dissertation (Part I), agents learn a set of skills following their intrinsic motivation. However, we do not explore the idea that skills can be reused in another task. Especially, in a new task, the intrinsic novelty of a state should depend on the skills already mastered by the agent. For instance, if the agent knows how to manipulate an object or avoid an obstacle, the agent should be encouraged to acquire different skills. Even though transfer in reinforcement learning [190, 191, 192] has been extensively studied, driving exploration in new tasks in a few-shot style remains an open question. In the long run, we believe that formulations of curiosity should not be limited to the task being learned, but should also take into account the situations already encountered by the agent and prior knowledge.

**Understanding the Limitations of Demonstrators.** Leveraging human guidance to train an agent often follows a teacher-student paradigm. In recent years a lot of effort has been directed towards making the student (i.e. the agent) more knowledgeable. However, we believe that mutual understanding is equally important. Especially, understanding the limitations of humans would allow us to create more effective systems. In such a setting, the student could select queries that can be easily understood by the teacher, or queries where guidance is the most needed and human effort is minimal. It was shown that differences arise in behaviors when humans demonstrate or teach [193]. This study has shown that understanding these differences can benefit when learning from showing versus doing. Ultimately, we believe that understanding the limitations of demonstrators for other types of guidance (e.g preferences) will help to create more functional AI systems.

## 10.2 Summary of Contributions

In Chapter 3 we looked at the exploration problem in complex environments where rewards are sparse or poorly-defined. We dealt with the question of how to drive the agent's learning based on its knowledge of the task in order to drastically improve its sample efficiency. In contrast to classical approaches that maintain a model of environmental dynamics, we introduced the concept of skill-based intrinsic motivation that works directly on the agent's policy. Our central idea is to incentivize the discovery of new skills and guide exploration towards promising solutions, helping the agent to incrementally and continually build up useful new skills and knowledge based on what it has already learned. We also proposed and studied a technique to embed skills and states into a latent space, in order to improve generalization to unseen situations. Evaluated on several domains including Atari games and MuJoCo, our algorithm outperformed state-of-the-art competitors both in terms of quality of learned policies and sample efficiency. In particular, this algorithm performed well in the presence of sequential decision problems, and high-dimensional state spaces.

In Chapter 4 we then addressed the problem of local optima and premature convergence to sub-optimal policies inherent in curiosity. Most prior curiosity-driven techniques involve the absolute prediction error to guide immediate exploration [14, 15]. However, such approaches suffer from two major limitations: 1) they tend to attract the agent to states with stochastic transitions due to hardly predictable environmental dynamics - local optima, and 2) the curiosity rewards soon exhaust as the prediction becomes perfect or does not improve (also called the "vanishing curiosity" issue) - converging prematurely to sub-optimal policies. On the other hand, the proposed model introduced a robust formulation of curiosity that does not quickly vanish and escapes from local optima. We consider the agent's learning progress on a multi-step horizon as a curiosity signal. To quantify the agent's learning progress in terms of "quantity" and "quality", we proposed to measure the divergence between a parametric model (i.e. the current model) and prior models. We introduced and evaluated two types of models based on: 1) the agent's understanding of the world, 2) the agent's policy. We further developed a mechanism called episodic-skills to direct the focus of attention on task-relevant skills. Our method significantly improved exploration and sample-efficiency, especially in tasks featuring complex exploration patterns, stochasticity, and/or deceptive rewards. To the best of our knowledge, we were the first solving environments characterized by both sparse and deceptive rewards (e.g. Pitfall).

In the first chapters of the thesis, curiosity does not explicitly encourage deep exploration. While these methods hold the promise of better local exploration, solving problems that require coordinated decisions over long-time horizons remains an open question. In Chapter 5, we presented a framework that explicitly promotes deep exploration behaviors. The central idea is to decompose the curiosity bonus into a fast reward that deals with local exploration and a slow reward that encourages deep exploration. These rewards are calculated following the idea of noisy reconstruction, a novel technique that can capture meaningful visual features and salient environmental dynamics at different scales (e.g. character-level or word-level). We further proposed an adaptive scaling technique to modulate fast and slow rewards by measuring state diversity. We then derived a framework for lifelong learning via the better incorporation of environmental motions and a different strategy for combining reward components. As far as we are aware, this was the first generic framework that can outperform prior state-of-the-art with much less trial and error interactions by a large margin. Moreover, the proposed method is suitable for a wide range of long-horizon domains where rewards are naturally extremely sparse; including robotic, visual navigation, visual control, and sequential tasks.

In Chapter 6, we considered external forms of supervision via human guidance and domain knowledge. We integrated human-like reasoning and domain knowledge to guide the agent and enhance information given to the agent, thanks to human expertise. To do so, the agent's input is augmented with high-level information representing task-relevant knowledge that can be easily interpreted by the agent and applicable to many aspects of the task. To create high-level information, we designed an algorithm that relies on visual recognition and simple domain knowledge. We further proposed to integrate human-based reasoning and planning based on the high-level information. Our algorithmic framework dealt with incomplete or noisy human knowledge by combining prior and learned action selection. This generic framework introduced human guidance with a minimal human workload, but drastically reduced the amount of required training data. Evaluated on multiple domains, our algorithms outperformed state-of-the-art end-to-end methods by a large margin. Overall, human guidance enables learning from very few interactions even in fairly complex tasks such as 3D navigation, and enhances the agent's decision-making with planning and reasoning.

In the above Chapter, we did not address the interpretability issue. Moreover, we did not consider the situation where already created datasets related to the task are available. In the context of real-world scenarios, it would be desirable to automatically extract domain knowledge from existing datasets to further reduce human involvement. Therefore, in Chapter 7 we presented a novel reinforcement learning model that uses as its internal representation first-order logic rules automatically extracted from datasets related to the task being learned. Critically, such an internal representation makes the model fully interpretable. To ground the rules, we proposed to analyze the latent representation learned by an autoencoder, combining the strengths of deep learning and human knowledge. We further make use of the structure of the rules for improving the generalization capabilities of our agent. We evaluated the approach on a wide range of domains including trading and visual navigation. We first observed that automatically created rules can achieve similar performance as manually created rules. The evaluations also showed that our approach efficiently learns and generalizes rules to setups which were not demonstrated in the datasets, indicating improvements in the agent's capability for common sense reasoning. To the best of our knowledge, we were among the first to propose approaches for extracting rules from deep neural networks, and then to use these rules as the internal representation of an interpretable reinforcement learning model.

In Chapters 6 and 7, we supposed passive access to external guidance. However, in complex real-world tasks, we cannot always assume that the learner's goal matches the (fixed) domain knowledge provided. Moreover, active cooperation between a teacher and the agent is central for sample-efficient learning. In Chapter 8, we addressed these issues by incrementally querying short demonstrations towards specific goals during the agent's training, *goal-driven demonstrations*. At the core, the method is based on the idea of active goal-conditioned imitation learning. We formulated an algorithm to decide how to query *goal-driven demonstrations* based on the expected value of information of the query and the agent's confidence, greatly reducing the number of queries. We further contributed a relabeling strategy to artificially generate more expert demonstrations. Finally, we proposed to give more weight to goals for which the agent's learning will be maximized (i.e. strong disagreement with the teacher). Through evaluation on benchmark robotic tasks, we showed that our strategy allows our agent for substantially faster learning and is particularly effective in the low data regime, where not many queries can be made. This was the first approach that can achieve dexterous in-hand manipulation with less than 50 (partial) demonstrations. Additionally, we were able to reach a wide range of configurations with the same set of goal-driven demonstrations.

So far, we have presented techniques for introducing internal or external supervision into reinforcement learning. However, we did take into account the possibility that multiple types of (internal and external) supervision are available, common in the real world. In Chapter 9, we aimed to achieve human-like sample-efficient learning by integrating different modes of supervision into hierarchical reinforcement learning. Therefore, we designed a hierarchical reinforcement learning model that can simultaneously integrate high-level human preferences and curiosity. The intuition behind is that humans are good at planning and high-level supervision, while control and low-level tasks can be "easily" learned via curiosity-driven reinforcement learning. We further developed a strategy based on curiosity to automatically discover sub-goals throughout the agent's training process, alleviating the need for an expert to design sub-goals. We evaluated our framework in a wide range of tasks, including tasks that are too challenging for even humans to perform well. The evaluations highlighted that a few high-level preferences is a sufficient supervision signal for solving many tasks - grafting suitable domain knowledge onto the agent's decision-making facilitates various forms of common sense reasoning. Moreover, experimental results demonstrated that curiosity enables our hybrid model to exceed expert-level performance in several domains As far as we know, we were among the first to integrate internal and external guidance for sample-efficient reinforcement learning. Overall, our method radically reduced the number of trial and error interactions.

## 10.3 Summary of the Thesis

These contributions together demonstrate that improving cognitive capabilities in reinforcement learning agents is essential to drastically enhance sample efficiency and performance. These capabilities inspired by learning in humans make artificial agents suitable for real-world domains. Namely, curiosity is a vital capability for guiding exploration - learning control and execution, particularly when rewards are sparse. On the other hand, sample efficiency can also be greatly increased by leveraging human guidance, which enables common sense reasoning and avoids learning from scratch. These two types of supervision are not exclusive; they can be complementarily employed to facilitate different aspects of learning. Especially, leveraging both internal motivation and external guidance allowed us to reduce the number of interactions by several orders of magnitude and to outperform human experts in a wide range of domains such as game-playing, trading, 3D navigation, or robot control. Hence, the proposed algorithms have improved efficiency in reinforcement learning agents enough that they can be practically applied to real-world tasks. In the future, these algorithms could be used to develop sample-efficient and learning machines, with substantial benefits for society as a whole.

# References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[3] Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[4] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2016.

[5] H Jin Kim, Michael I Jordan, Shankar Sastry, and Andrew Y Ng. Autonomous helicopter flight via reinforcement learning. In *Proceedings of Advances in neural information processing systems*, pages 799–806, 2004.

[6] Markus Kuderer, Shilpa Gulati, and Wolfram Burgard. Learning driving styles for autonomous vehicles from demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2641–2646, 2015.

[7] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[8] Murray Shanahan, Matthew Crosby, Benjamin Beyret, and Lucy Cheke. Artificial intelligence and the common sense of animals. *Trends in Cognitive Sciences*, 24(11):862 – 872, 2020.

[9] Peter Robinson. *Abilities to Learn: Cognitive Abilities*, pages 17–20. 2012.

[10] Linda S. Gottfredson. Mainstream science on intelligence: An editorial with 52 signatories, history, and bibliography. *Intelligence*, 24(1):13 – 23, 1997.

[11] Richard M Ryan and Edward L Deci. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, 25(1):54–67, 2000.

[12] Yael Niv. Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3):139–154, 2009.

[13] Jacqueline Gottlieb, Pierre-Yves Oudeyer, Manuel Lopes, and Adrien Baranes. Information-seeking, curiosity, and attention: computational and neural mechanisms. *Trends in cognitive sciences*, 17(11):585–593, 2013.

[14] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the International Conference on International Conference on Machine Learning*, page 2778–2787, 2017.

[15] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. In *Proceedings of the The International Conference on Learning Representations*, 2019.

[16] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

[17] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017.

[18] Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[19] Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

[20] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *Proceedings of the International Conference on Learning Representations*, 2019.

[21] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. In *Proceedings of the International Conference on Learning Representations*, 2019.

[22] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the international conference on Machine learning*, pages 1–8, 2004.

[23] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

[24] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Proceedings of Advances in neural information processing systems*, pages 4565–4573, 2016.

[25] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 663–670, 2000.

[26] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 1–8, 2004.

[27] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the National Conference on Artificial Intelligence*, page 1433–1438, 2008.

[28] Ruohan Zhang, Faraz Torabi, Lin Guan, Dana H Ballard, and Peter Stone. Leveraging human guidance for deep reinforcement learning tasks. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 6339–6346, 2019.

[29] Alison Gopnik, Andrew N Meltzoff, and Patricia K Kuhl. *The scientist in the crib: Minds, brains, and how children learn.* William Morrow & Co, 1999.

[30] Linda Smith and Michael Gasser. The development of embodied cognition: Six lessons from babies. *Artificial life*, 11(1-2):13–29, 2005.

[31] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[32] Richard S Sutton and Andrew G Barto. *Reinforcement learning: an introduction.* MIT press Cambridge, 1998.

[33] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[34] Gavin Adrian Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical report, University of Cambridge, October 04 1994.

[35] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4(Dec):1107–1149, 2003.

[36] Steven Bradtke and Andrew Barto. Linear least-squares algorithms for temporal difference learning. *Machine learning*, 22(1-3):33–57, 1996.

[37] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of International conference on Machine Learning*, pages 1928–1937, 2016.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[39] Leslie Pack Kaelbling. Learning to achieve goals. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 1094–1098, 1993.

[40] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the International conference on machine learning*, 2015.

[41] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Proceedings of Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[42] Peter Stone and Manuela Veloso. Layered learning. In *Proceedings of the European Conference on Machine Learning*, pages 369–381, 2000.

[43] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[44] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J Kennedy. Relational autoencoder for feature extraction. In *Proceedings of the International Joint Conference on Neural Networks*, pages 364–371, 2017.

[45] M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, 2006.

[46] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. In *Proceedings of the International Conference on Learning Representation*, 2017.

[47] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of International conference on machine learning workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[48] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations*, 2014.

[49] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1–8, 2007.

[50] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[51] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the international conference on machine learning*, volume 99, pages 278–287, 1999.

[52] Sébastien Racanière, Theophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *Proceedings of Advances in Neural Information Processing Systems*, 2017.

[53] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[54] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[55] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[56] Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proceedings of the International Conference on Machine Learning*, pages 2721–2730, 2017.

[57] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2018.

[58] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. 2017.

[59] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Proceedings of the International Conference on Robot Learning*, 2017.

[60] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *Proceedings of the International conference on machine learning*, pages 1515–1528, 2018.

[61] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Proceedings of Advances in Neural Information Processing Systems*, pages 9191–9200, 2018.

[62] Manfred Eppe, Sven Magg, and Stefan Wermter. Curriculum goal masking for continuous deep reinforcement learning. *CoRR*, abs/1809.06146, 2018.

[63] Jürgen Schmidhuber. Curious model-building control systems. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, 1991.

[64] Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proceedings of the International Conference on Simulation of Adaptive Behavior: From animals to animats*, pages 222–227, 1991.

[65] Laurent Itti and Pierre F Baldi. Bayesian surprise attracts human attention. In *Proceedings of Advances in neural information processing systems*, pages 547–554, 2006.

[66] Christoph Salge, Cornelius Glackin, and Daniel Polani. Changing the environment based on empowerment as intrinsic motivation. *Entropy*, 16(5):2789–2819, 2014.

[67] Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

[68] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the International Conference on International Conference on Machine Learning*, pages 2778–2787, 2017.

[69] Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evol. Comput.*, 19(2):189–223, June 2011.

[70] Shengjia Zhao, Hongyu Ren, Arianna Yuan, Jiaming Song, Noah Goodman, and Stefano Ermon. Bias and generalization in deep generative models: An empirical study. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems*, pages 10792–10801. 2018.

[71] Danilo J. Rezende, Shakir Mohamed, Ivo Danihelka, Karol Gregor, and Daan Wierstra. One-shot generalization in deep generative models. In *Proceedings of the International Conference on International Conference on Machine Learning*, pages 1521–1529, 2016.

[72] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the The International Conference on Learning Representations*, 2015.

[73] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[74] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[75] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *Proceedings of the International Conference on Robotics and Automation*, pages 512–519, 2016.

[76] Christopher Stanton and Jeff Clune. Deep curiosity search: Intra-life exploration improves performance on challenging deep reinforcement learning problems. In *Proceedings of the International Conference on International Conference on Machine Learning*, 2019.

[77] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the International conference on machine learning*, 2016.

[78] Nicolas Bougie and Ryutaro Ichise. Exploration via progress-driven intrinsic rewards. In *Proceedings of the International Conference on Artificial Neural Networks*, volume 22, 2020.

[79] Rein Houthooft, Xi Chen, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1109–1117. 2016.

[80] Nicolas Bougie and Ryutaro Ichise. Skill-based curiosity for intrinsically motivated reinforcement learning. *Machine Learning*, pages 493—-512, 2019.

[81] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *Proceedings of the International Conference on Machine Learning*, pages 1515–1528, 2018.

[82] Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[83] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Proceedings of Advances in neural information processing systems*, pages 2753–2762, 2017.

[84] Zhang-Wei Hong, Tzu-Yun Shann, Shih-Yang Su, Yi-Hsiang Chang, Tsu-Jui Fu, and Chun-Yi Lee. Diversity-driven exploration strategy for deep reinforcement learning. In *Proceedings of Advances in neural information processing systems*, pages 10489–10500, 2018.

[85] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Machine Learning Research*, pages 1861–1870, 2018.

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

[87] Christian Kauten. Super Mario Bros for OpenAI Gym. `https://github.com/Kautenja/gym-super-mario-bros`, 2018.

[88] Mathew Botvinick, Sam Ritter, Jane X Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis. Reinforcement learning, fast and slow. *Trends in cognitive sciences*, 2019.

[89] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258, 2017.

[90] David Abel, Alekh Agarwal, Fernando Diaz, Akshay Krishnamurthy, and Robert E Schapire. Exploratory gradient boosting for reinforcement learning in complex domains. *Proceedings of the International Conference on Machine Learning Workshop on Abstraction in Reinforcement Learning*, 2016.

[91] Jarryd Martin, Suraj Narayanan Sasikumar, Tom Everitt, and Marcus Hutter. Count-based exploration in feature space for reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, page 2471–2478, 2017.

[92] Alexandre Pere, Sébastien Forestier, Olivier Sigaud, and Pierre-Yves Oudeyer. Unsupervised learning of goal spaces for intrinsically motivated goal exploration. In *Proceedings of the International Conference on Learning Representations*, 2018.

[93] Sébastien Forestier, Yoan Mollard, and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

[94] Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2019.

[95] Marios C Machado, Marc G Bellemare, and Michael Bowling. A laplacian framework for option discovery in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 2295–2304, 2017.

[96] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

[97] Yuu Jinnai, Jee Won Park, David Abel, and George Konidaris. Discovering options for exploration by minimizing cover time. In *Proceedings of the International Conference on Machine Learning*, pages 3130–3139, 2019.

[98] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 3540–3549, 2017.

[99] Alexander S Klyubin, Daniel Polani, and Chrystopher L Nehaniv. Empowerment: A universal agent-centric measure of control. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 128–135, 2005.

[100] Hsuan-Kung Yang, Po-Han Chiang, Min-Fong Hong, and Chun-Yi Lee. Exploration via flow-based intrinsic rewards. In *Proceedings of the The International Conference on Learning Representations*, 2020.

[101] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, pages 189–223, 2011.

[102] Dianyuan Han. Comparison of commonly used image interpolation methods. In *Proceedings of the International Conference on Computer Science and Electronics Engineering*, pages 1556–1559, 2013.

[103] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[104] Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in ***beta***-vae. *arXiv preprint arXiv:1804.03599*, 2018.

[105] Irina Higgins, Loic Matthey, Xavier Glorot, Arka Pal, Benigno Uria, Charles Blundell, Shakir Mohamed, and Alexander Lerchner. Early visual concept learning with unsupervised deep learning. *arXiv preprint arXiv:1606.05579*, 2016.

[106] Jake Snell, Karl Ridgeway, Renjie Liao, Brett D Roads, Michael C Mozer, and Richard S Zemel. Learning to generate images with perceptual similarity metrics. In *Proceedings of the IEEE International Conference on Image Processing*, pages 4277–4281, 2017.

[107] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *Proceedings of the Conference on Signals, Systems & Computers*, volume 2, pages 1398–1402, 2003.

[108] Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[109] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[110] Nicolas Bougie and Ryutaro Ichise. Combining local and global exploration via intrinsic rewards. *Proceedings of the Annual Conference of JSAI*, pages 2K6ES205–2K6ES205, 2020.

[111] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2140–2146, 2017.

[112] Antonio Lieto, Christian Lebiere, and Alessandro Oltramari. The knowledge level in cognitive architectures: Current limitations and possible developments. *Cognitive Systems Research*, pages 1–42, 2017.

[113] Oltramari and Lebiere. Using ontologies in a cognitive-grounded system: Automatic action recognition in video-surveillance. 2012.

[114] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Z Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations. In *Proceedings of the Annual Meeting of the Association for the Advancement of Artificial Intelligence*, 2018.

[115] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58, 2016.

[116] A. d'Avila Garcez, A. Resende Riquetti Dutra, and E. Alonso. Towards Symbolic Reinforcement Learning with Common Sense. *ArXiv e-prints*, April 2018.

[117] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[118] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[119] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 4246–4247, 2016.

[120] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[121] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[122] Satinder P Singh and Richard S Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158, 1996.

[123] Nicolas Bougie and Ryutaro Ichise. Deep reinforcement learning boosted by external knowledge. In *Proceedings of the Annual ACM Symposium on Applied Computing*, pages 331–338, 2018.

[124] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[125] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[126] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. *ArXiv e-prints*, December 2013.

[127] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2016.

[128] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the International conference on machine learning*, pages 1995–2003, 2016.

[129] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, April 2012.

[130] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, 2014.

[131] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the International conference on Machine learning*, page 78, 2004.

[132] S Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689, 2003.

[133] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[134] Todd Hester and Peter Stone. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning*, 90(3):385–429, 2013.

[135] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering*, (6):734–749, 2005.

[136] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

[137] Nicolas Bougie and Ryutaro Ichise. Deep reinforcement learning boosted by external knowledge. In *Proceedings of the ACM Symposium on Applied Computing*, pages 331–338, 2018.

[138] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[139] Jinkyu Kim and John Canny. Interpretable learning for self-driving cars by visualizing causal attention. In *Proceedings of International Conference on Computer Vision*, pages 2961–2969. IEEE, 2017.

[140] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, pages 5045–5054, 2018.

[141] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8, 2010.

[142] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[143] Matthew Rosencrantz, Geoff Gordon, and Sebastian Thrun. Learning low dimensional predictive representations. In *Proceedings of the International Conference on Machine learning*, page 88, 2004.

[144] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine learning*, 43(1-2):7–52, 2001.

[145] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 119–125, 2002.

[146] Mohamed K Gunady and Walid Gomaa. Reinforcement learning generalization using state aggregation with a maze-solving problem. In *Proceedings of the Conference on Electronics, Communications and Computers*, pages 157–162, 2012.

[147] Nicolas Bougie and Ryutaro Ichise. Abstracting reinforcement learning agents with prior knowledge. In *Proceedings of the International Conference on Principles and Practice of Multi-Agent Systems*, pages 431–439. Springer, 2018.

[148] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[149] Steve Nison. *Japanese candlestick charting techniques: a contemporary guide to the ancient investment techniques of the Far East*. Penguin, 2001.

[150] Morteza Mashayekhi and Robin Gras. Rule extraction from random forest: the rf+hc methods. In *Proceedings of the Canadian Conference on Artificial Intelligence*, pages 223–237, 2015.

[151] Mahesh Pal. Random forest classifier for remote sensing classification. *Proceedings of the International Journal of Remote Sensing*, 26(1):217–222, 2005.

[152] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[153] Yoshua Bengio, Olivier Delalleau, and Clarence Simard. Decision trees do not generalize to new variations. *Computational Intelligence*, 26(4):449–467, 2010.

[154] Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. Pixelvae: A latent variable model for natural images. In *Proceedings of the International Conference on Learning Representations*, 2017.

[155] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[156] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. https://github.com/openai/gym, 2016.

[157] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522*, 2018.

[158] Akhil Raj Azhikodan, Anvitha GK Bhat, and Mamatha V Jadhav. Stock trading bot using deep reinforcement learning. In *Innovations in Computer Science and Engineering*, pages 41–49. Springer, 2019.

[159] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari. In *Proceedings of Advances in neural information processing systems*, pages 8011–8023, 2018.

[160] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[161] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[162] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[163] Xiaoqin Zhang and Huimin Ma. Pretraining deep actor-critic reinforcement learning algorithms with expert demonstrations. *arXiv preprint arXiv:1801.10459*, 2018.

[164] Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In *Proceedings of the International Conference on Machine Learning*, pages 2469–2478, 2018.

[165] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 6292–6299, 2018.

[166] Sonia Chernova and Manuela Veloso. Confidence-based policy learning from demonstration using gaussian mixture models. In *Proceedings of the International joint conference on Autonomous agents and multiagent systems*, pages 1–8, 2007.

[167] Guoyu Zuo, Qishen Zhao, Jiahao Lu, and Jiangeng Li. Efficient hindsight reinforcement learning using demonstrations for robotic tasks with sparse rewards. *International Journal of Advanced Robotic Systems*, 17, 2020.

[168] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *Proceedings of the International Conference on Machine Learning*, pages 3309–3318, 2017.

[169] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. In *Proceedings of Advances in Neural Information Processing Systems*, pages 15298–15309, 2019.

[170] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Proceedings of Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

[171] Aaron Wilson, Alan Fern, and Prasad Tadepalli. A bayesian approach for policy learning from trajectory preference queries. In *Proceedings of Advances in neural information processing systems*, pages 1133–1141, 2012.

[172] Kory W Mathewson and Patrick M Pilarski. Actor-critic reinforcement learning with simultaneous human control and feedback. *arXiv preprint arXiv:1703.01274*, 2017.

[173] Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *The Journal of Machine Learning Research*, 18(1):4945–4990, 2017.

[174] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1545–1554, 2018.

[175] Nicolas Bougie, Li Kai Cheng, and Ryutaro Ichise. Combining deep reinforcement learning with prior knowledge and reasoning. *ACM SIGAPP Applied Computing Review*, 18(2):33–45, 2018.

[176] Si-An Chen, Voot Tangkaratt, Hsuan-Tien Lin, and Masashi Sugiyama. Active deep q-learning with demonstration. *Machine Learning*, pages 1–27, 2019.

[177] Aaron P Shon, Deepak Verma, and Rajesh PN Rao. Active imitation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 756–762, 2007.

[178] William Saunders, Girish Sastry, Andreas Stuhlmueller, and Owain Evans. Trial without error: Towards safe reinforcement learning via human intervention. In *Proceedings of the International Conference on Autonomous Agents and MultiAgent Systems*, pages 2067–2069, 2018.

[179] Daniel Hsu. A new framework for query efficient active imitation learning, 2019.

[180] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the International Conference on Machine Learning*, pages 1050–1059, 2016.

[181] Nicolas Bougie and Ryutaro Ichise. Active goal-driven learning. 2020.

[182] Nicolas Bougie and Ryutaro Ichise. Fast and slow curiosity for high-level exploration in reinforcement learning. *Applied Intelligence*, 2020.

[183] Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudik, Yisong Yue, and Hal Daumé, III. Hierarchical imitation and reinforcement learning. Proceedings of Machine Learning Research, pages 2917–2926, 2018.

[184] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the International Conference on Machine Learning*, pages 166–175, 2017.

[185] Frank Röder, Manfred Eppe, Phuong D. H. Nguyen, and Stefan Wermter. Curious hierarchical actor-critic reinforcement learning. In Igor Farkaš, Paolo Masulli, and Stefan Wermter, editors, *Proceedings of Artificial Neural Networks and Machine Learning*, 2020.

[186] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Proceedings of Advances in neural information processing systems*, pages 3675–3683, 2016.

[187] Maurice G Kendall and B Babington Smith. On the method of paired comparisons. *Biometrika*, 31(3/4):324–345, 1940.

[188] Tiago Mota and Mohan Sridharan. Commonsense reasoning and knowledge acquisition to guide deep learning on robots. In *Robotics: Science and Systems*, 2019.

[189] Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L Griffiths, and Alexei A Efros. Investigating human priors for playing video games. In *Proceedings of the International Conference on Machine Learning*, 2018.

[190] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[191] Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.

[192] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. In *Proceedings of the The International Conference on Learning Representations*, 2016.

[193] Mark K Ho, Michael Littman, James MacGlashan, Fiery Cushman, and Joseph L Austerweil. Showing versus doing: Teaching by demonstration. In *Proceedings of Advances in neural information processing systems*, pages 3027–3035, 2016.