Human Cognition-Inspired High-Level Decision-Making for Reinforcement Learning Agents

Dossa, Rousslan Fernand Julien

(Degree)
博士（工学）

(Date of Degree)
2023-03-25

(Date of Publication)
2024-03-01

(Resource Type)
doctoral thesis

(Report Number)
甲第8655号

(URL)
https://hdl.handle.net/20.500.14094/0100482403

Doctoral Thesis

Human Cognition-Inspired High-Level

Decision-Making for Reinforcement Learning Agents

人間の認知に基づく強化学習エージェントのための

高度な意思決定

January, 2023

Graduate School of System Informatics
Department of Information Science
Kobe University

DOSSA ROUSSLAN FERNAND JULIEN

Human Cognition-Inspired High-Level Decision Making for Reinforcement Learning Agents

Rousslan Fernand Julien Dossa

Abstract

Reinforcement learning (RL) methods can achieve performance superior to humans in a wide range of complex tasks and uncertain environments, ranging from video game playing based on incomplete pixel observations, robotics control, autonomous driving, chip placement for PCB, and even discovery of faster computational algorithms. RL can thus be considered a promising avenue for the automation of critical tasks that occur in various industries and modern society. This is due to the relatively higher generality of RL methods akin to that of animals and humans, which require a relatively lower amount of supervision and domain knowledge to solve a given task when compared to classical automation methods.

Nevertheless, high performance is not always the sole focus when considering the practical application of RL methods in a societal context such as game AI or autonomous driving. Namely, a greedily performing agent can frustrate and inconvenience surrounding human users that interact with such agent, lead to physical damages, or material losses in more extreme cases. This fact makes the case for the need for more human-like agents. While imitation learning methods can reproduce the behavior of a human expert and build a human-like agent, its performance is limited to that of the expert. This thesis first presents a training scheme to construct

a human-like and efficient agent by mixing reinforcement and imitation learning for discrete and continuous action space problems. The proposed hybrid agent achieves higher performance than a strict imitation learning agent and exhibits more human-like behavior in the context of video game playing.

In contrast to current RL methods, the ability of humans to efficiently understand and learn to solve complex tasks with relatively limited data is attributed to our hierarchically organized decision-making process. Meanwhile, sample efficiency is yet another long-standing challenge for RL agents, especially in long-horizon, sequential decision-making tasks with sparse and delayed rewards. Hierarchical reinforcement learning (HRL) augments RL agents with temporal abstraction to improve their efficiency in such complex tasks. However, the decision-making process of most HRL methods is often based directly on dense low-level information, while also using fixed temporal abstraction. This thesis further tackles the matter of sample efficiency in RL by introducing the hierarchical world model (HWM), geared toward capturing more flexible high-level, temporally abstract, and low-level dynamics of the task. On top of providing the theoretical foundation for integrating the proposed HWM with the HRL framework, empirical experiments demonstrate improvement in sample efficiency and final performance when using the HWM over classical model-based RL.

Finally, this thesis emphasizes the importance of low-level code optimizations used in the implementation of RL methods which do not receive much attention despite being critical to the performance of RL algorithms. The fourth chapter of this thesis investigates the effect of one such optimization known as “early stopping” whose usage is seldom documented in public implementation of the Proximal Policy Optimization (PPO) algorithm. This optimization technique, which is referred to as KLE-Stop,

stops the policy network update within an epoch if the mean Kullback-Leibler (KL) Divergence between the target policy and current policy becomes too high. Empirical experiments were conducted to show the importance of KLE-Stop and its more conservative variant KLE-Rollback when they are used in conjunction with other common code-level optimizations. The results demonstrate how such early stopping optimizations mitigate the sensitivity of the PPO to the number of policy network updates while being more robust to their respective hyperparameters.

Overall, this thesis addresses the critical challenges that arise on the path of applying RL methods for automating complex real-world tasks by providing a principled framework to efficiently train robust and reliable RL agents that exhibit human-like behaviors and a more efficient high-level decision-making inspired by human cognition.

Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Background

Recent advances in hardware capability and theoretical developments in the field of deep learning (DL) have spurred the rise in popularity of artificial neural networks (ANNs) and their applications [27, 28, 43]. Owing to their generality, the use of ANNs has spread from the field of computational science and cross-pollinated with a plethora of other fields, ranging from computer vision [17, 36, 56, 57, 82, 112], natural language processing [8, 18, 112], speech recognition [92], and even protein folding [46, 111], to only cite a few.

Reinforcement learning (RL) is one such field that also happens to be the core topic of this thesis. Originally motivated by the study of the psychology of animal and human learning and the problem of learning through trial-and-error, RL has seen subsequent growth and formalization inspired by optimal control and theories of dynamical systems [106]. As a consequence, modern RL has come to encompass the study of the decision-making of intelligent agents in arbitrary environments, and how such agents can learn viable solutions to achieve a given goal through a process of trial and error. Such roots make computational reinforcement learning an incredibly

promising avenue to further our understanding of animal and human cognition and decision-making. Recent studies [63, 64] have even shown how RL agents can be used as a controlled experimental setting to substitute for biological agents in neuroscience experiments. More specifically, a fully embodied virtual rodent was trained using RL and analyzed with tools from the neuroscience literature [71] to emphasize potential relationships between biological rodent brains and their artificial alternative.

RL is further intertwined with other learning paradigms that exist under the umbrella of deep learning. One such paradigm is referred to as *supervised learning* (SL). In the SL paradigm, the agent or *model* is given a pre-determined and finite set of input samples and their corresponding *label*., which is considered as the *correct answer*. An SL model is thus geared to approximate the function that maps from a given input data to the correct label [28]. For example, a function *that can determine if a given input image features a dog or not*, corresponding to the classification problem, or a function *that expresses the cost of a house as a function of time and geographical location*, a task that can be considered as the archetypal regression problem [28, 106]. A robust and generalizing SL model is also expected to extrapolate the learned approximation to samples outside of the training distribution. Different approaches that frame the reinforcement learning problem as a supervised learning problem have also been proposed. One such approach is referred to as *imitation learning*, which is further elaborated in Chapter 2. Imitation learning aims to recover expert policies based on a finite amount of samples drawn from their interaction with an environment [15, 39, 42, 106]. This approach is somewhat limited to tasks where expert policies are available, and where the finite nature of the training dataset contains enough information to capture optimal policies. Nevertheless, imitation learning

can be used to *bootstrap* the learning of RL agents by pre-training on available and potentially sub-optimal datasets to learn representations that are useful in downstream reinforcement learning tasks [51, 77, 98]. *Upside Down Reinforcement Learning* is a recent approach that reframes reinforcement learning under the guise of a supervised learning problem with rewards as labels and actions as an additional context to the agent's input [91, 103]. This enables the use of powerful supervised learning methods [36, 82, 112] to achieve competitive results with traditional RL methods, excelling when expert data is available.

Unsupervised learning (UL) is yet another paradigm of deep learning closely related to reinforcement learning. Unlike supervised learning which requires each sample in the dataset to be assigned a ground truth label, unsupervised learning usually works with unlabelled data [28]. UL methods leverage a class of generic *unsupervised objective functions* to learn latent structures underlying unlabelled data [28]. The *variational auto-encoder* (VAE) [50] is a representative of such unsupervised learning methods. Given a dataset of unlabelled images of handwritten digits ranging from 0 to 9 [58], the VAE leverages an encoder (compression) and decoder (reconstruction) architecture to learn salient features of the digit images compressed into a low-dimensional latent space. Unsupervised learning is even more complementary to reinforcement learning. It allows to extract more information from the data samples generated through the interaction of RL agents with the environment by using auxiliary, unsupervised learning objectives alongside the reward signal of RL [33–35, 45]. This results in more useful learned features which have been demonstrated to be incredibly useful to RL agents for better decision-making, but also in learning macro skills that can hasten exploration in, and completion of downstream RL tasks [9, 23, 45, 97].

Reinforcement learning thus clearly demarcates itself from its supervised or unsupervised learning [28, 106] siblings by focusing on algorithms and solutions that can learn optimal sequences of action based on a numerical reward signal depending on the goal of the agent is expected to achieve. Beside supervision via numerical reward signal, another core idea of reinforcement learning is the concept of *policy*. In the RL framework, the policy is a mapping from perceived states of the environments to actions. Depending on the goal of the task at hand, an optimal policy would map a given state to an action or sequence of actions that lead to the completion of said task. In the crudest cases, a policy can be a simple conditional statement, a lookup table, or a parameterized function (ANN) to handle more complex scenarios [28, 106]. The policy component essentially represents the behavior of the RL agent. Depending on the nature of the task, the policy can be structured to be either deterministic for straightforward cases, or stochastic to support tasks with uncertainty [31, 62]. Another salient idea of reinforcement learning is the concept of *value function*. Unlike the reward which represents the short-term value of being in a current state and taking an arbitrary action, the value component is a medium to long-term estimate of the current state with respect to the end goal. The value component can be either used to articulate a policy thus resulting in the family of RL methods referred to as *value-based RL agents* [14, 67, 89], or complement learning the policy component, resulting in the family of *policy gradient* and *actor-critic* agents [62, 94, 95, 106, 118].

Reinforcement learning methods that rely solely on policy and value functions can be further consolidated in the group of *model-free reinforcement learning* (MfRL) methods. Empirically, such methods are fast to train, but require large amounts of samples [14, 62, 67, 94, 95, 106]. Sample collection lends itself to scaling, which

can be achieved by increasing computational resources, as well as combining it with parallelization techniques [61, 78, 117]. MfRL methods excel in tasks that mostly require *reflexive behavior* (myopic, short-term planning), and have held onto the state-of-the-art performance on various benchmark tasks [4, 109].

Aside from model-free RL methods, their *model-based reinforcement learning* (MbRL) counterpart has recently been catching up to state-of-the-art performances on RL tasks. This is mainly due to the increased capacity of modern ANNs which enables the dynamics of complex environments [12, 33–35, 43]. As suggested by their denomination, MbRL methods include an additional concept that occurs in RL referred to as *world models*. World models are generally learned in an unsupervised or self-supervised fashion [29, 34] based on trajectories generated from the interactions of agents with the environments. When accurate enough, world models can be used in place of the original environment to generate synthetic trajectory samples which are then used to further refine the policy and value components. This is reminiscent of the ability of humans to simulate various situations in their minds and even conceptualize mental plans before executing them in physical reality [106, 110]. A model-based RL agent usually boast higher sample efficiency than its model-free counterpart, a direct consequence of the former's ability to use synthetic samples for learning. However, the gain in sample efficiency often comes at the cost of slower training, as learning a world model generally implies a more complex agent architecture and requires higher computational resources. Nevertheless, this trade-off can be practically advantageous compared to model-free alternatives in tasks where safety is paramount such as autonomous driving, or tasks where sampling has a higher cost, such as physical robot locomotion [119].

Empowered by ANNs and the advances in machine learning, RL agents have become able to solve a wide range of complex tasks, such as video game playing based on incomplete pixel observations [32–35, 72, 94, 95, 115], robotics control [73, 74, 119], autonomous driving [44, 87, 96], chip placement for PCB [65], and even discovering faster matrix multiplication algorithms [24], with a relatively limited amount of supervision. In contrast with traditional methods of automation, RL agents are a promising avenue for the automation of real-world high-impact tasks, especially given the context of aging demographics and the ever-increasing complexity of the industrial processes of human civilization [76].

1.2 Problem description

Despite the recent advances and development of RL methods, there is still room for improvement before they can be consistently trained and reliably deployed in complex real-world scenarios. The challenges under examination in the present thesis are threefold.

1. First, from a qualitative perspective, RL agents to be deployed in the real world are expected to demonstrate societally acceptable behavior with consideration for their surroundings. This is especially critical as those agents will have to account for and interact with human agents. However, RL agents are trained to maximize an arbitrary objective function corresponding to the task of interest. This usually results in overly greedy agents that do not necessarily align with the safety and reliability criterion expected from them in society and would create more harm than good [21, 83, 90, 101]. In the context of video games, overly

greedy RL agents can result in levels of difficulty that frustrate the human players and reduce the level of satisfaction and engagement of human users [83, 90]. In the more critical cases such as deploying RL-based autonomous driving agents or industrial robots, overly performant and unpredictable agents can result in physical harm and even loss of life [21, 87, 96, 101].

2. The second hurdle in the path of reliably training RL agents on complex real-world scenarios is the sample inefficiency of the learning process itself. Unlike humans, RL agents require a magnitude of orders higher number of interactions with the environment to explore the available state-space through trial-and-error, then learn satisfactory enough policy that sole the task at hand. Although this challenge can be partially addressed using parallelism [61, 72, 115, 117] to accelerate the sample collection process, such alternative would be limited in real-world by physical reality. On the other hand, cognitive and behavioral science literature suggests that the learning efficiency demonstrated by animals, and more specifically humans is at least in part due to their ability to plan and make decisions at higher levels of asbtraction [3,6,32,110]. Under the traditional RL framework, trial-and-error fueled exploration is usually performed at the finest time scale, which becomes untractable in the highly complex tasks that are ubiquitous in real-world scenarios. This suggests that current RL methods are limited in their efficiency by the lack of temporally abstract exploration and high-level decision-making. Such restriction essentially limits the range of tasks that can be automated using RL methods.

3. Finally, from a practically grounded perspective, a usually overlooked but

highly relevant aspect in the RL pipeline is the efficiency and reliability of the implementation and training procedure themselves. Namely, implementation details that might be considered negligible and irrelevant to the final performance of the agent can nevertheless have an unexpected impact [13, 22, 38]. More specifically, the cost of training successful RL agents is heavily tied to the structure of the neural networks, hyperparameters choice, and the training pipeline used [22, 38]. Such costs also increase proportionally to the difficulty of the task at hand, especially when it comes to critical real-world tasks with complex observation and action space. Therefore, the practical cost of RL-based automation can be as much, if not higher than that of traditional automation methods, which ultimately defeats the purpose of RL in the first place. This makes the case for the need to improve RL methods from the conceptual level, down to the practical (implementation) level by shaving off as much overhead as possible from the training algorithm itself [13, 22].

1.3 Structure

This thesis aims to address each of the aforementioned challenges to further bridge the gap between theoretical RL and the latter's practical applications, thus broadening the range of tasks that can be reliably automated using RL methods.

Chapter 2 introduces a hybrid of reinforcement and imitation learning that trades off between high performance and human-likeliness of RL agents that could be deployed among human agents in a societal context. Chapter 3 presents a human cognition-inspired model-based hierarchical reinforcement learning architecture that

leverages the concept of temporally abstract exploration and high-level decision-making to improve sample efficiency. Finally, Chapter 4 examines the low-level and more practical aspects of RL agents implementation and highlights implementation techniques that improve the efficiency of the training process itself, as well as the robustness and reliability of the resulting RL agents.

Chapter 2 Hybrid of Reinforcement and Imitation Learning for Human-Like Agents

2.1 Introduction

Reinforcement learning (RL) achieved impressive progress in several areas including Go [100], autonomous driving [44, 87, 96, 114], and video games [14, 66, 68, 94, 95]. The RL trains an agent to maximize future rewards, and the trained agent occasionally outperforms human experts. However, high performance should not be the sole focus when considering the application of RL methods. For example, when a non-player character (NPC) is trained via RL, it can become too strong to be defeated by human players, which will lead to frustration and an inability to enjoy the game. Similarly, an efficient autonomous vehicle can inconvenience surrounding cars and pedestrians by taking abrupt turns or more generally by exhibiting selfish behavior or creating a sense of fear even when the behavior is efficient and safe in reality. Hence, an approach that builds an agent which behaves like a human is desirable.

Imitation learning (IL) trains an agent to learn a policy from training data provided by a human expert [75, 85, 104]. An agent trained with IL is thus expected to exhibit human-like behavior. However, IL is not a goal-oriented method and is dependent on the demonstration data provided by a human expert. Therefore, the performance of an IL agent is likely to be capped to the former's. Various studies have leveraged human expert demonstrations to improve the performance of the RL agent [10, 39, 55, 69]. However, there is a paucity of studies on the human-likeliness factor.

This study aims to produce an agent that behaves like a human while retaining high performance by RL. This work introduces a hybrid agent based on RL and IL and demonstrates how the performance level of an RL agent and the tendency of human expert behavior can be transferred via IL to a student agent. The proposed approach is applied to an original game and the Atari57 suite's Gopher game [4] as environments of discrete action spaces. It was also applied to the TORCS racing simulator [121] as an environment of a continuous action space to demonstrate its potential in real-life applications, such as autonomous driving, where a human-like agent is highly desirable and of utmost importance. A performance test and sensitivity test (like the "Turing test") conducted in a double-blind fashion demonstrated that the proposed approach built agents that achieve better performance and exhibit more human-like behaviors when compared to those of IL and RL agents, respectively.

This chapter is thereby structured as follows: first Section 2.2 reviews the existing works in the field, then respectively lays the background of the methods which were built upon as the proposed method in Section 2.3. Next, the experiments conducted using the proposed method, followed by the corresponding results and discussions are presented in Section 2.4. Finally, the chapter is concluded in Section 4.6.

2.2 Related Work

2.2.1 Improving reinforcement learning performance from expert demonstrations

Across several topics such as robotics, general games, or autonomous driving, several studies have aimed to combine reinforcement learning (RL) and imitation learning (IL). The precursors of the aforementioned studies include Chang et al. [10] who proposed locally optimal learning to search (LOLS) intended for structured prediction tasks. This is based on the more general learning to search (L2S) method [15, 81, 84, 85]. Specifically, LOLS involves random sampling from either the learning policy or reference policy (expert policy) given an arbitrary state during the rollouts. Subsequently, it minimizes the objective loss function and guarantees convergence to a policy that is at least as good as the reference policy.

Following this, Nair et al. [69] extended the concept via the usage of expert demonstrations to overcome the difficulties of exploration in tasks with high dimensional state-action space or sparse rewards. They introduced a demonstration buffer that stores trajectories sampled from an expert, which are subsequently used in combination with the trajectories sampled from the learning policy to update the value function. Thus, the learning policy is oriented toward more promising avenues and consequently speeds up the training process. The authors demonstrated the effectiveness of their method by applying it to a 7-DOF fetch robotic arm to solve a block stacking problem with a continuous action space. The proposed method achieved a relatively better exploration when compared to the baselines. It also exhibited a performance exceeding

the expert's performance in solving the stacking task when provided with a few additional simulator tweaks.

Similarly, Hester et al. [39] proposed deep Q-Learning from demonstrations (DQfD) that also leverages a very small amount of demonstration data to significantly accelerate the training process. The agent is initially pretrained on the expert's demonstration data via a combination of both supervised and temporal difference (TD) losses based on the vanilla deep Q-learning (DQN) [68] and is subsequently trained by interacting with the domain with its pretrained policy as usual. Hence, the agent learns to imitate the expert's policy while improving it with the self-learned policy via RL. The resultant policy outperforms pure reinforcement learning baseline (double dueling DQN [116]) in 41 out of the 42 games that are experimented on with a significantly faster convergence rate.

More recently, Balaguer et al. [2] focused on a robotic arm problem, namely towel folding with cooperative manipulators through momentum fold. Their proposed algorithm leveraged human demonstrations to reduce the search space of the RL learner, thereby allowing it to converge faster to a satisfactory policy. Furthermore, Sun et al. [104] introduced Truncated HORizon Policy Search (THOR), which is based on the idea of reshaping the reward [70] and leveraged even sub-optimal expert demonstrations to shorten the learner's planning horizon, thereby significantly accelerating the learning process. Additionally, their algorithm keeps track of a disadvantage function between the expert and learner policy and prioritizes the optimization of the self-learned policy when the policy induced by the expert is determined as sub-optimal, and thus the final policy out-performs the expert.

Le et al. [55] introduced hierarchically guided IL and RL in the hierarchical

reinforcement learning framework (where a high-level policy or meta-controller is trained to select a sub-goal to be solved by low-level sub-policies until the final goal is achieved). By training the meta-controller following IL of a high-level interactive expert, hierarchical guidance limits the training of the RL sub-policies to the relevant state space and consequently improves the sample efficiency of the training process while reducing the cost of expert demonstrations.

The aforementioned studies mainly focus on improving the performance, sampling efficiency, and training speed of the RL agents. By contrast, the present study investigates the feasibility of grasping desirable features to characterize a human expert while improving the performance following an RL agent or self-learned behavior.

2.2.2 Building human-like behavior based on human demonstrations

To the best of the authors' knowledge, a conceptually close work was only explored by Hecker et al. [37]. They focused on autonomous driving and proposed a formalism to achieve accurate and comfortable human-like driving. The method achieves better driving accuracy by augmenting the data set consisting of car-mounted front-facing image data of human driving with a manually engineered set of map features. The learning policy also incorporates long-short term neural networks to consider previously executed maneuvers, and thus its output is sequential as opposed to traditional point-wise predictions. Furthermore, the driving loss is enhanced with an adversarial loss that discriminates between human-like driving and policy learning via RL in addition to a loss that penalizes abrupt steering or acceleration over a short period, thereby

resulting in a final policy that is close to the human expert's demonstrations. However, the human-likeliness of the behavior is systematically evaluated by clustering the driving data of the human expert and learned policy's predictions and measuring their similarity. Although such an evaluation method is pragmatic, it does not guarantee a measurement of human impression, namely because they are equivalent to the metrics used for imitation learning of a human expert.

2.3 Methods

2.3.1 Preliminaries

Reinforcement Learning

Traditionally, a simple Reinforcement Learning (RL) problem is approached through the formal lens of the Markov Decision Process (MDP). An MDP is defined as a 5-tuple $< \mathbb{S}, \mathbb{A}, P, R, \gamma >$, where $\mathbb{S}$ denotes a set of states, $\mathbb{A}$ denotes a set of actions, $P_{s'}(s, a) = P(s_{t+1} = s' | s_t = s, a_t = a)$ denotes the probability that state s transits to state s' due to action a on time-step t, $R(s, a)$ denotes the reward received after the transition from state s to the state s', and γ $(0 < \gamma \leq 1)$ denotes a discount factor that represents the importance of future rewards when compared to the present rewards. On every time-step t during training, the agent obverses a state $s_t \in \mathbb{S}$ and produces an action $a_t \in \mathbb{A}$. In response, the environment returns the corresponding reward r_t and next state s_{t+1}.

The Deep Q-network (DQN) [67,68] algorithm is one of the earliest RL methods that achieved reliable results over a wide range of tasks with discrete action spaces [4, 7].

From an arbitrary state s_t at time-step t, a DQN agent is trained to predict the expected return $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ where T denotes the terminal time-step. The agent subsequently takes an action that maximizes R_t based on the prediction. The optimal action-value function is defined as $Q^*(s_t, a_t) = \max_\pi \mathbb{E}[R_t | s = s_t, a = a_t]$, and applying the Bellman equation to this yields the following expression:

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathcal{E}}[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) | s_t, a_t]$$

where $\mathcal{E}$ represents the transition function of the environment. During training, the agent estimates R_t by iteratively updating the action-value function $Q(\phi(s), a)$ where ϕ denotes a function that produces the fixed length representation of state s. Thus, the action-value function $Q(\phi(s), a)$ converges to the optimal $Q_i \rightarrow Q^*$ as the update iteration $i \rightarrow \infty$ [106].

Although DQN achieves strong performance over a high dimensional state space, it was proven as limited to low dimension discrete action spaces. Therefore, for continuous action spaces, the definition of the action space is overloaded so that $\mathbb{A} \subset \mathbb{R}^N$ is continuous instead. Analogous to the discrete case the goal thus translates to obtaining a policy π that maximizes the expected return from the start distribution $J = \mathbb{E}_{r_i, s_i \sim \mathcal{E}, a_i \sim \pi}[R_0]$. The deep deterministic policy gradient (DDPG) method [62] is based on the deterministic policy gradient (DPG) method [99], which defines the policy of the RL agent as an actor function ($\mu(s|\theta^\mu)$ parametrized by θ^μ and mapped from a state s_t to a corresponding action a_t) and a critic function $Q(s_t, a_t)$ to approximate the value of a state-action pair (s_t, a_t). The actor update is subsequently performed based

on the *policy gradient* as defined by David Silver et al. [99].

The introduction of deep and non-linear approximators makes the update of the critic function $Q(s, a|\theta^Q)$ prone to divergence. Thus, the DDPG method leverages "soft" target updates by creating copies $\mu'(s|\theta^{\mu'})$ and $Q'(s, a|\theta^{Q'})$ to be used in computing the target values. However, the target networks are updated by slowly tracking original networks: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ with $\tau << 1$ to increase the stability of the training.

Given that a large action space requires considerably more exploration to converge to the optimal solution, an exploration policy $\mu'(s_t) = \mu(s_t|\theta_t^{\mu}) + \mathcal{N}$ with an added noise sampled from a noise process $\mathcal{N}$ (which is selected based on the nature of the task or the environment [79]) is adopted.

Imitation Learning

Let the policy followed by the expert players during demonstrations correspond to the optimal policy π^*, and the agent learns a policy π that approximates the optimal policy π^*. A traditional approach involves training an agent via supervised learning. Expert human players provide trajectories that consist of sequences of state-action pairs because the training data follows an optimal policy π^*. The agent's policy is trained to predict actions based on the provided states and thereby imitates the behavior of the expert, albeit in a point-wise fashion.

In the discrete action space case, knowledge distillation is an approach that trains a student model based on teacher model(s) where the teacher model trains the student model by using *soft labels* instead of the one-hot label commonly used in traditional supervised learning [41]. When compared to the one-hot label, the values of each

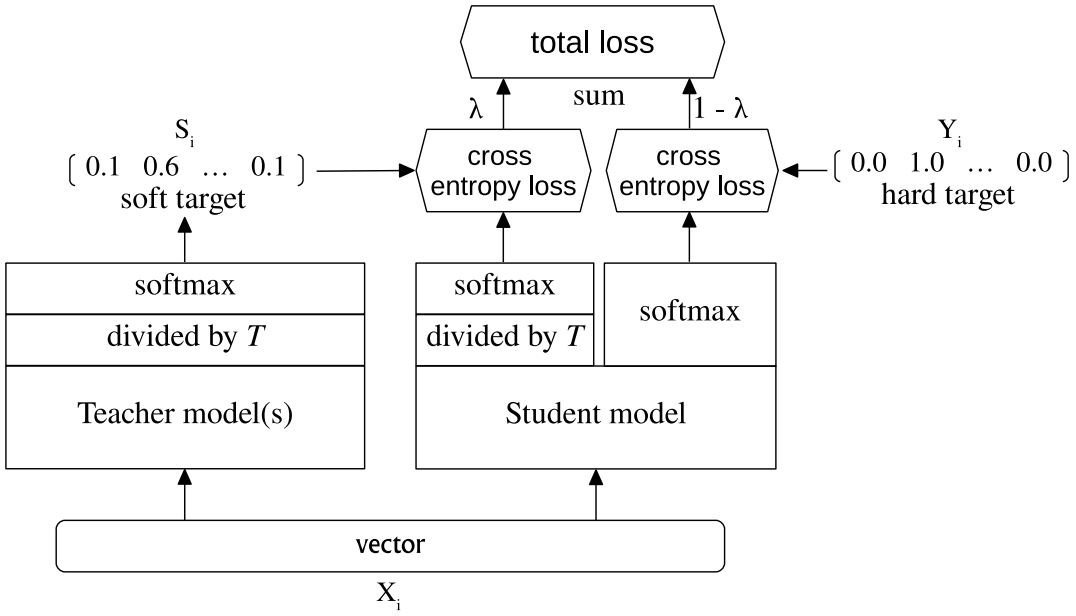


Fig. 2.1: Procedure of knowledge distillation.

dimension of a soft label include more information. With a visual input of a cat, of course, the probability of it being classified as "cat" by the model is the highest, and the probability of "dog" exceeds that of "carrot", and this matches with the fact a cat is visually more similar to a dog than to a carrot. The procedure of knowledge distillation is shown in Fig. 2.1. The approach achieves a higher-performance model with less training data, and it is also useful in terms of compressing large models. An extant study [86] extended the approach to reinforcement learning area, and this is termed as *policy distillation*. Thus, a student model that achieves higher performance is successfully trained by distilling the policy of the teacher model. Furthermore, the applicability of policy distillation to multi-task problems is demonstrated by distilling policies of teacher models trained for different tasks.

Furthermore, in the continuous action space case, generative adversarial imitation

learning (GAIL) [42] is an IL method rooted in both inverse reinforcement learning (IRL) and classical RL. Specifically, the IRL initially fits a cost function c from a family of functions C which assigns a low cost to the expert policy π_E and high cost to the other policies π:

$$\max_{c \in C} \left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_{\pi}\left[c(s,a)\right] \right) - \mathbb{E}_{\pi_E}\left[c(s,a)\right]$$

where Π denotes the set of all stationary stochastic policies mapping from $\mathbb{S}$ to $\mathbb{A}$, and $H(\pi) \triangleq \mathbb{E}_{\pi}[-\log \pi(a|s)]$ denotes the γ-discounted causal entropy of the policy π [42]. Conversely, classical RL determines the optimal policy from the cost function c extracted by IRL

$$RL(c) = \underset{\pi \in \Pi}{\operatorname{argmin}} -H(\pi) + \mathbb{E}_{\pi}\left[c(s,a)\right]$$

corresponding to the expert policy π_E. To bypass the intermediate IRL step, the original study introduced the concept of occupancy measure ρ_π of a policy π, and this is interpreted as the distribution of state-action pairs that an agent encounters while navigating the environment by the following said policy [42]. A function approximator $D_w : \mathbb{S} \times \mathbb{A} \rightarrow (0, 1)$ is used to discriminate between the occupancy measure ρ_π of the student policy and expert ρ_{π_E} by minimizing the sum of their respective expectations

over generated trajectories:

$$\mathcal{L}(\pi, \pi_E) = \mathbb{E}_{\pi}\left[\log\left(D_w(s,a)\right)\right] + \mathbb{E}_{\pi_E}\left[\log\left(1 - D_w(s,a)\right)\right] - \lambda H(\pi) \text{ for } \lambda \geq 0$$

until D_w is unable to distinguish between the π and π_E, thereby implying that the student learned the expert's policy.

The training of the student agent itself is based on the Trust Region Policy Optimization (TRPO) algorithm introduced in [94] where instead of the classical reward, the algorithm is provided with the similarity between the trajectories resulting from the student's policy and that of the expert. More specifically, the student agent is updated using the TRPO rule with the cost function $\log\left(D_w(s,a)\right)$ by taking a KL-constrained gradient step with respect to

$$\mathbb{E}_{\tau\sim\pi}[\nabla_\theta \log \pi(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta),$$
$$\text{where } Q(\bar{s},\bar{a}) = \mathbb{E}_{\tau\sim\pi}[\log\left(D_w(s,a)\right) | s_0 = \bar{s}, a_0 = \bar{a}].$$

2.3.2 Proposed hybrid agent

Recall that this chapter aims to build an agent with a behavior similar to a human expert while retaining the high performance of an RL agent. The problem is separated into the two following subproblems: (1) building an agent that exhibits a superhuman performance and (2) building an agent that selects actions similar to a human expert. The two subproblems are tackled by RL and IL, respectively. This overall task can

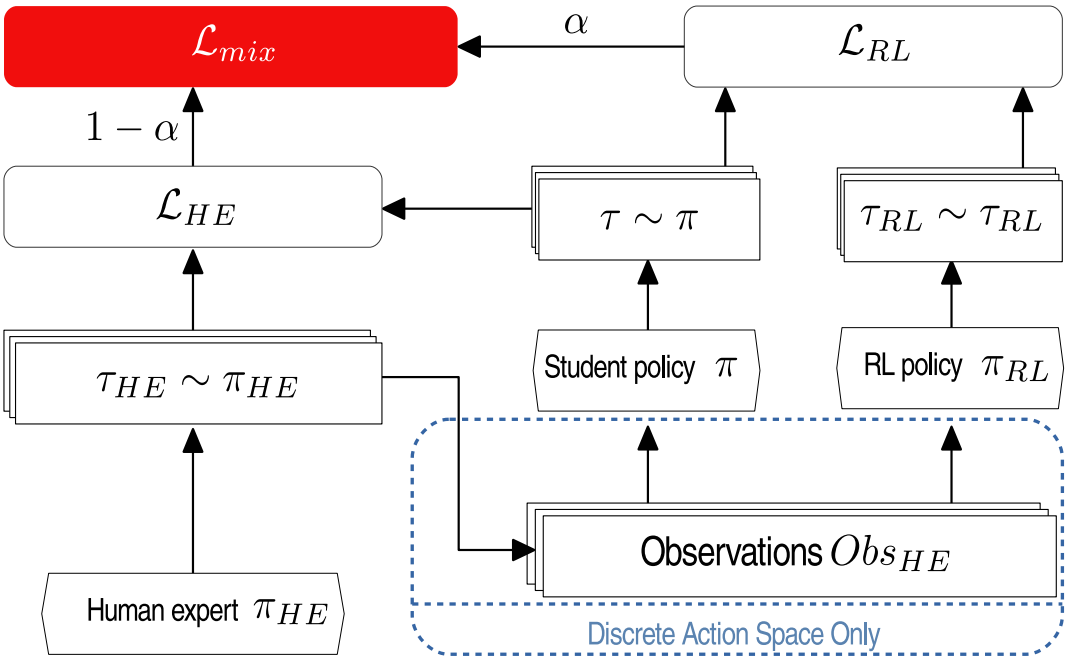


Fig. 2.2: Conceptual diagram of the proposed hybrid loss

thus be understood as a multi-task learning problem. The proposal at the core of this study consists in mixing RL and IL via policy distillation [41] for the discrete action space case and via adversarial imitation learning for the continuous action space case.

Generally, the objective function of IL is expressed as $\mathcal{L}_{\pi^*}(\pi)$, where π^* denotes a teacher policy to be imitated by the student policy π. Subsequently, the proposed objective function is formalized as:

$$\mathcal{L}_{mix}\left(\pi; \pi_{RL}, \pi_{HE}\right) = \alpha \mathcal{L}_{\pi_{RL}}\left(\pi\right) + \left(1-\alpha\right) \mathcal{L}_{\pi_{HE}}\left(\pi\right),$$

where π_{RL} denotes an optimal policy built by RL, π_{HE} denotes a policy of a human expert (HE), and α denotes a trade-off coefficient between the RL policy and human

expert.

Discrete action space

In the case of discrete action space, the objective function of IL is the following cross-entropy loss:

$$\mathcal{L}_{\pi^*}(\pi) = \mathbb{E}_s\left[-\sum_a \pi^*(a|s)\log\pi(a|s)\right],$$

by following previous studies on IL and distillation [41, 86]. An RL policy π_{RL} is expressed as a mathematically modeled probability while the policy π_{HE} of a human expert is given in terms of empirical demonstrations. Previous studies on distillation demonstrated that a student model improves its performance by a large margin if it reproduces soft labels (i.e., probability of teacher's output obtained via a high temperature T) in addition to hard labels (i.e., correct labels expressed by 0 or 1 probabilities). It is not possible to obtain soft labels of the policy for distillation. Given this, human expert demonstrations π_{HE} are used as hard labels and the policy $\pi_{RL}^{(T)}$ of a DQN model [68] are used as soft labels with a temperature T. Note that a typical policy of a DQN model employs $T = 0.0$ (i.e., provides hard labels). Subsequently, the final objective function is expressed as follows:

$$\mathcal{L}_{mix}(\pi) = \alpha\,\mathbb{E}_s\left[-\textstyle\sum_a \pi_{\mathrm{RL}}^{(T)}(a|s)\log\pi(a|s)\right] + (1-\alpha)\,\mathbb{E}_s\left[-\textstyle\sum_a \pi_{HE}(a|s)\log\pi(a|s)\right].$$

It should be noted that the distribution of state s over which the objective is averaged depends on the trajectories sampled from policies. Naturally, the RL policy π_{RL} and human expert policy π_{HE} are used for the first and second terms, respectively. However, when the two policies are completely different, the student policy π encounters a high variance between their actions and yields poor results. The human expert policy π_{HE} is thus used as a substitute for the distribution for sampling the state s for both terms. This sampling strategy was confirmed to improve performance and yield a consistent behavior [41, 42].

Continuous action space

While policy distillation can theoretically be applied to continuous action space tasks, depending on the formulation of the action space itself, simply mixing the actions provided by multiple experts is likely to induce a wrong hybrid policy. Namely in the autonomous driving task, a common formulation would be as such: the action of *steering left* is bounded to -1.0, *steering right* is bounded to 1.0, while the *no steering* is bound to 0.0 . Next, let us consider the case of learning a hybrid of two expert policies when avoiding an obstacle, but with one expert avoiding said obstacle by steering left, while the second one avoids the obstacle by steering right. By mixing these two policies using policy distillation as in the discrete action space case, the hybrid policy is bound to go straight ahead, thus crashing into the obstacle and resulting in a failure. The proposed method instead aims to imitate the hybrid of the two expert policies but based on their respective probability density distribution over the state-action space, therefore motivating the choice of the GAIL method introduced in Section 2.3.1.

GAIL requires empirical trajectories $\tau^* \sim \pi^*$ obtained from a teacher policy π^* for training. The objective function of the discriminator that is to be maximized by D_w and to be minimized by a student policy π is as follows:

$$\mathcal{L}_{\pi^*}(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\log\left(D_w(s,a)\right)\right] + \mathbb{E}_{\tau^*\sim\pi^*}\left[\log\left(1 - D_w(s,a)\right)\right],$$

where τ denotes trajectories $\tau \sim \pi$ sampled from a student policy π. With a human expert and an RL agent providing trajectories $\tau_{HE} \sim \pi_{HE}$ and $\tau_{RL} \sim \pi_{RL}$, respectively, the hybrid loss function is expressed as follows:

$$\begin{aligned}\mathcal{L}_{mix}(\pi) =& \mathbb{E}_{\tau\sim\pi}\left[\log\left(D_w(s,a)\right)\right] \\ &+ \alpha\, \mathbb{E}_{\tau_{RL}\sim\pi_{RL}}\left[\log\left(1 - D_w(s,a)\right)\right] \\ &+ (1-\alpha)\, \mathbb{E}_{\tau_{HE}\sim\pi_{HE}}\left[\log\left(1 - D_w(s,a)\right)\right],\end{aligned} \tag{2.3.1}$$

Intuitively, given that the discriminator is trained to recognize an *intermediate* hybrid policy between the RL and human expert policies, the student model — which is trained to learn a policy to fool the discriminator to classify its actions as being the expert's — is expected to converge to the said hybrid policy, thereby exhibiting behaviors inherited from both experts. More specifically, for a given state-action pair (s, a), the discriminator D_w produces a score that can be interpreted as the likelihood of the given state-action pair is that of the human or RL hybrid policy. Minimizing the first term $\mathbb{E}_{\tau\sim\pi}\left[\log\left(D_w(s,a)\right)\right]$ with respect to the weights w incentivizes the

discriminator D_w to assign a low score to samples generated by the student model. On the other hand, minimizing the hybridization terms

$$\alpha \, \mathbb{E}_{\tau_{RL} \sim \pi_{RL}} [\log(1 - D_w(s, a))] + (1 - \alpha) \, \mathbb{E}_{\tau_{HE} \sim \pi_{HE}} [\log(1 - D_w(s, a))]$$

encourages the model to assign a high score to state-action pairs matching the action distribution of either expert, with α to trade-off between each expert policy. Therefore, by iterating over such minimax objective, such hybrid generative adversarial imitation learning implicitly reduces the divergence between the action distribution of the student model, and that of the hybrid of RL and human expert.

In contrast to the discrete action space case, the results indicated that the proposed method successfully trained a hybrid policy although the demonstration sets of both experts are sampled independently of each other. This can be attributed to the relatively low variation between the RL and human expert behaviors and the adversarial training framework. Figure 2.2 recapitulates the complete procedure for both action spaces.

2.4 Experiments and Results

2.4.1 Apple Game

Experimental setting

The proposed approach was first applied to an original game termed as *Apple game*, illustrated by the screenshot provided in Fig. 2.3. This environment consists of a two-

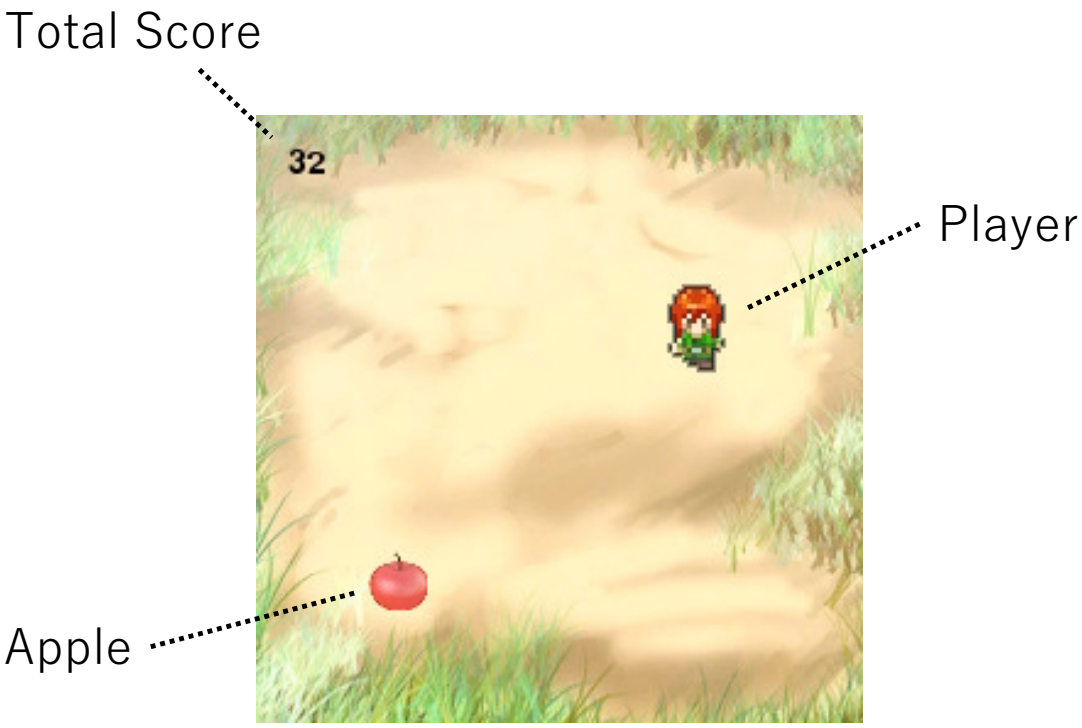


Fig. 2.3: Screenshot of the Apple game.

dimensional plane on which the agent can move freely in any direction. Additionally, apples are spawned randomly at an unpredictable location for the player to take. Once the player's avatar has collected an apple, a new one will spawn in another location on the two-dimensional plane. In the case an apple is not picked up in a fixed amount of time, it disappears and another apple is spawned at a different location. The score of the agent is increased by one point for every apple it picks up. The objective of the agent is thus to maximize the number of apples picked up in a limited amount of time. In this study, this interval was fixed to 30 seconds. Once an apple is spawned, it is always visible until it disappears. Therefore the agent can easily find a sequence of actions to collect it. This environment was selected given the straightforward formulation of the objective and the relative ease of evaluating the optimal trajectory

taken by an arbitrary agent, as well as its behavior. The diverse combinations of actions give rise to a plurality of behaviors that can help characterize the agent that is playing. Namely, to reach an apple located in the upper left corner, the agent can sequentially move "up" then "left". The agent can also move diagonally toward the apple by using the combination "up and left". The discrete action space $\mathcal{A}$ consists of 9 actions, i.e. moving in either one of the 8 possible directions in a 2D plane or keeping still. The action space is thus represented by $\mathcal{A} = [(-1,-1),(0,-1),...,(1,1)]$, $|\mathcal{A}| = 9$. From the evaluation perspective, given the location of the apple and the player's avatar, it is easy to determine the shortest possible path. This is especially useful to verify whether or not an agent has converged to an optimal policy.

From the human and RL agent, 16,000 frames of gameplay were collected, then randomly sampled into training and test sets with sizes corresponding to 14,500 and 1,500, respectively, during the training process to maximize the test accuracy. A DQN [68] model was trained to serve as a baseline for performance comparison, as well as the RL teacher. The model was trained over $8 \cdot 10^6$ frames of gameplay, with a replay buffer of size $2 \cdot 10^5$, and an exploration coefficient decaying from 1.0 to 0.1, the other hyperparameters being set to their respective default values as per the OpenAI Baselines implementation [19]. The student model was trained using Adam optimizer [49] with a learning rate of 10^{-4} and a dropout ratio of 0.5 [102]. Additionally, a preparatory search for the α parameter was performed to determine the most suitable one for the task. The duration of one episode is set to 30 seconds, and the number of apples collected by the player during that time interval is considered as the return of the episode.

In addition to the performance evaluation of each model, a sensitivity test in a

double-blind fashion to evaluate the human-likeliness of agent behaviors was also conducted. The test involved 26 participants (23 males and 3 females) wherein ages ranged from 27 to 59 with an average age of 44 years, which had no prior exposure to the research materials, demonstrations, or previous studies. Each participant was initially introduced to the rules of the game and provided with the opportunity to play the game to get familiar with its mechanics and how a human would play. In the Apple game case, each participant was presented with two 15 seconds videos of each model playing the game and requested to label it as either a human or a machine, as well as providing comments which motivated such a decision.

Results and Discussion

Table 2.1: Results of Apple game

	Game Score				**Sensitivity Test**
Agent	**Max**	**Min**	**Avg**	**Std.**	**Identified as Human**
Human	27	11	18.71	2.86	**32**
DQN (RL)	53	15	**36.27**	5.44	4
BC (IL)	29	3	17.57	4.37	<u>27</u>
Proposed (RL+IL)	35	11	<u>22.02</u>	3.70	22

Best and second best results are emphasized in bold and underlined, respectively.
The evaluated hybrid model corresponds to a trade-off coefficient of $\alpha = 0.93$.
The sensitivity test for the Apple game involved 25 participants only.

For the Apple game, the score evaluation process for each agent consisted in computing the average of the scores achieved over 400 episodes. In the Apple game, the score denotes the number of apples collected in a 30-s interval. The score variation of the proposed hybrid agent with respect to the change in the trade-off coefficient α

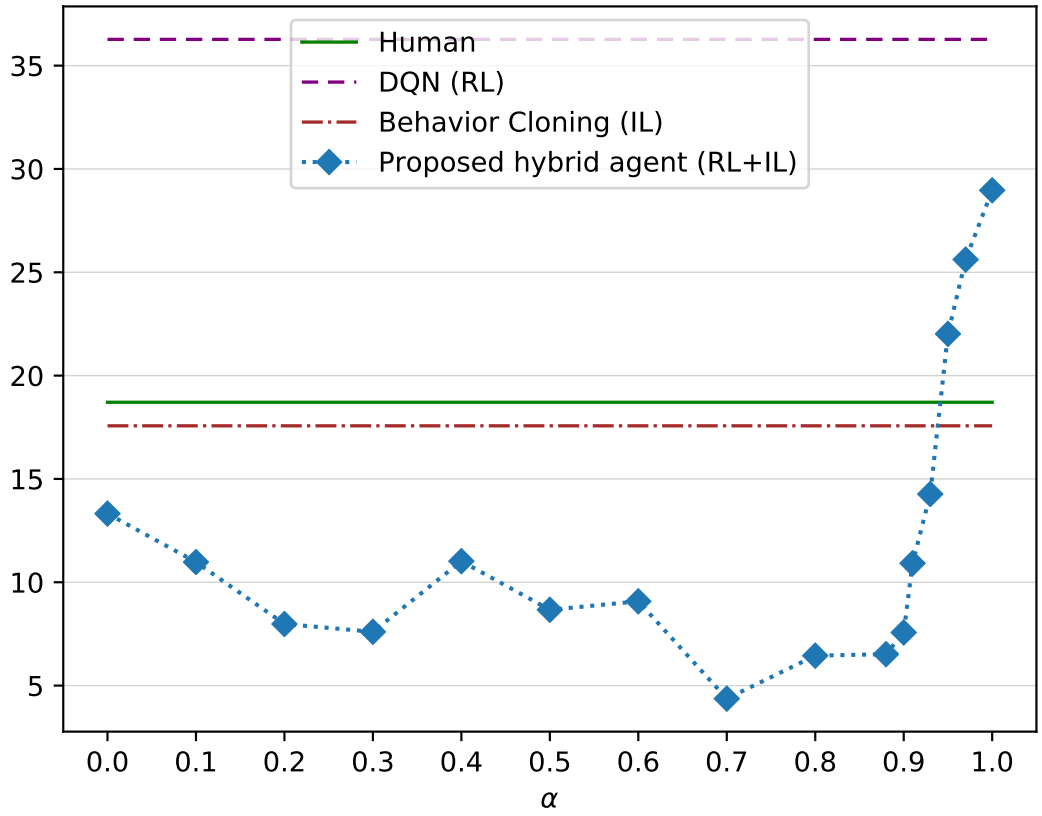


Fig. 2.4: Effect of the trade-off coefficient α on the proposed hybrid agent's score for the Apple Game. The average scores of the other agents are also provided for reference purposes.

is shown in Fig. 2.4 with the average scores of the human expert, DQN agent, and imitation of the human expert by using classical supervised learning.

At $\alpha = 0.0$ (which represents the distillation of human expert behavior only), the trained policy achieved lower than average performance, and this subsequently continued to decrease when α increased. With respect to a middle-range trade-off coefficient α, mixing the probability distributions over the action space received from the human expert and RL agent resulted in a flat probability distribution over the

action space, thereby making the agent follow a random policy and resulting in lower performance as shown in Fig. 2.4. The score subsequently rises progressively from $\alpha = 0.7$ until it reaches its peak at $\alpha = 1.0$, and this was equivalent to a distillation of the DQN agent's policy only.

Given the limited availability of the sensitivity test, the representative trade-off coefficient alpha was selected based on the corresponding performance. Following the performance of the hybrid agent obtained across various values of α (Fig. 2.4), it can be seen that the proposed method works better with larger values in this environment. This could be attributed to the wide gap in performance between the RL agent and the human expert. More specifically, the gap in performance between the RL agent and the human experts suggests the policies they respective follow are different. Furthermore, given the high variance of human expert behavior, their policies become hard to mix, thus resulting in a poor hybrid policy. The goal of this study being not just a high-performing agent, but also an agent that retains a human-like behavior, the value $\alpha = 0.93$ was settled for as the smallest experimented value that performs better than the human expert.

Following the results documented in Table 2.1 and given the simple principle of the game (i.e., moving the player avatar to the spawning apples), the DQN method achieved the highest scores, and this was followed by the proposed method and finally the human agent and its imitation. With respect to human-likeliness, the human agent was placed first. The proposed hybrid agent exhibited more human-like behavior than the RL agent. It also surpassed the human expert and IL agent in terms of the score and was followed by the human imitation agent. The DQN agent achieved the least human-like behavior.

Additionally, the hybrid agent mitigated a few of the behaviors that gave out the DQN agent, namely a rapid reaction time and mechanical movement towards the apple. This was corroborated by the comments collected from the survey participants including "a moderate reaction span when the apple spawns", "the player takes some time before moving toward the apple", or "keeps going on even after collecting the apple". Similar comments were also used to characterize the behavior of the human expert, thus further illustrating the similarity of the proposed hybrid agent with the latter.

This would suggest that the proposed approach indeed balances the human-like behavior and exhibited high performance in the game.

Table 2.2: Results of Gopher

	Game Score				**Sensitivity Test**
Agent	**Max**	**Min**	**Avg**	**Std.**	**Identified as Human**
Human	81	2	23.87	19.81	<u>29</u>
DQN (RL)	246	0	**40.30**	36.81	17
BC (IL)	126	0	23.91	23.79	**31**
Proposed (RL+IL)	132	0	<u>26.05</u>	24.31	**31**

Best and second best results are emphasized in bold and underlined, respectively. The evaluated hybrid model corresponds to a trade-off coefficient of $\alpha = 0.8$

2.4.2 Atari57 Suite: Gopher Game

Experimental setting

The proposed method was also applied to the game of the Atari57 suite [4] termed as *Gopher*. The dynamics and the observation space of the Gopher game are more

Fig. 2.5: A screenshot of *Gopher*.

complex than those of the Apple game. This enables evaluation of the effectiveness of the proposed method in the context of a wider state-action space. An optimal agent in the Gopher game is also expected to act strategically which further raises the complexity of the task. Namely, the goal consists in preventing the "gopher" — a mole-like creature — from digging its way out of the ground and stealing the carrots under the

protection of the player, the latter being tasked with moving laterally and filling up the holes as the gopher digs them. A screenshot of the game is shown in Fig. 2.5. The training data provided by the human expert and RL agent consisted of 55,000 frames each, which were also randomly sampled in training and test sets with sizes of 50,000 and 5,000, respectively. The DQN model that served as an RL teacher was trained over $1.001 \cdot 10^7$ frames of gameplay, a replay buffer of 10^6, and the exploration coefficient decaying from 1.0 to 0.01 while the other hyperparameters remained set to their respective default value as per the OpenAI Baselines implementation [19]. The student model was also trained by following the same method used for the Apple game.

Similarly to the Apple game, a sensitivity test in a double-blind fashion on top of the performance evaluation of each model was also conducted. After being introduced to the rules of the game and familiarizing themselves with the game mechanics, each participant was also presented with two 15 seconds videos of each model playing the game and requested to label it as either a human or a machine, as well as providing comments which motivated such decision.

Results and Discussion

Experiments were additionally conducted on the Atari57 suite's Gopher game namely to investigate the trade-off coefficient α variation's effect on the performance of the proposed hybrid agent, the results being documented in Fig. 2.6. The score of the hybrid agent remained around the human expert's average score for $\alpha \in [0.1, 0.6]$ before rising from $\alpha = 0.7$.

As for the Apple game, the gap in performance between the RL agent and the

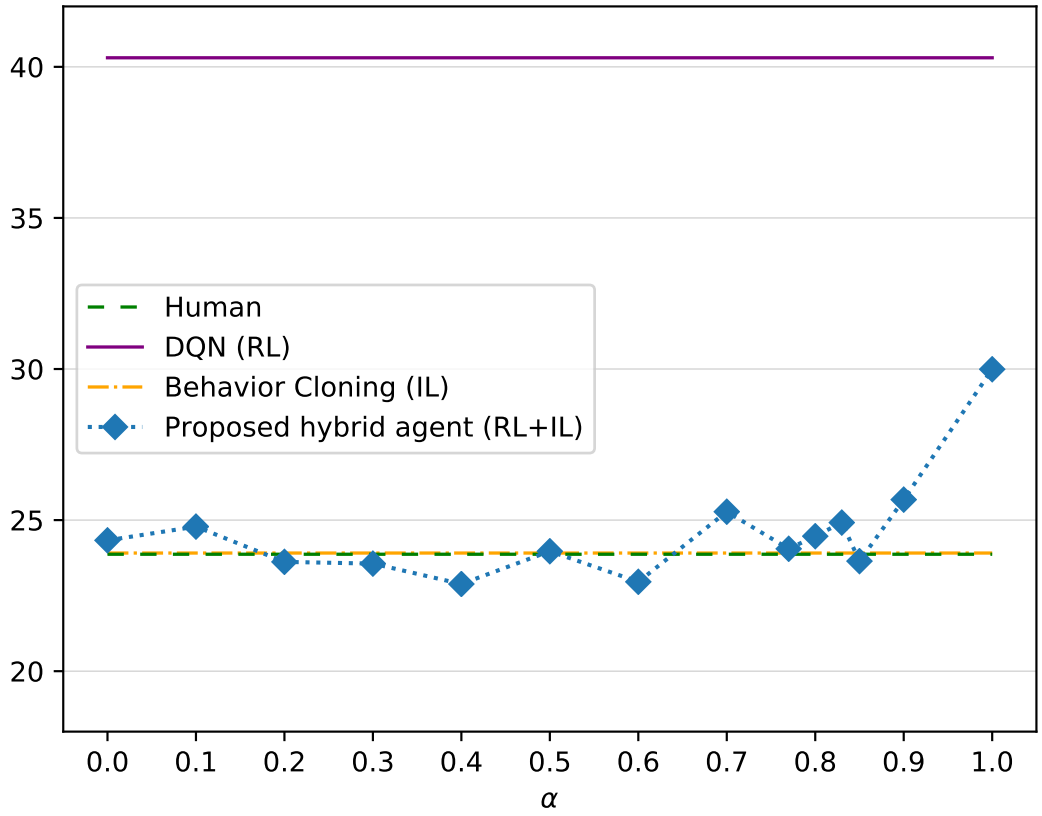


Fig. 2.6: Effect of the trade-off coefficient α on the proposed hybrid agent's score for Gopher. The average scores of the other agents are also provided for reference purposes.

human expert was also high. Henceforth, relatively large values of α are needed to offset the difficulty of mixing the policies that arise given said gap. As a representative of the hybrid agents, a value of $\alpha = 0.8$ was chosen, being the smallest value of α experimented with that also performed better than the human expert.

From a performance viewpoint, the DQN agent obtained the highest score by a large margin. This was followed by the hybrid agent that was trained via the proposed method and finally by the human agent and its imitation.

Table 2.2 compares the hybrid agent with the others. The hybrid agent (despite prioritizing the RL label by using $\alpha = 0.8$ to compute the weighted sum of the teachers' labels during the training) only exhibited an improvement of 3 points over the human agent and was still considerably behind the DQN agent's score. The RL agent achieved the best results although only a few participants identified it as a human. In the game, the hybrid agent performs comparatively or superiorly to the IL agent in terms of both criteria, namely game score and human-like behavior. Also, the proposed hybrid agent and IL agent were recognized as humans in the sensitivity test at a similar level to the human agent. This is supported by the comments collected from the participant of the sensitivity test such as "the player seems to move strategically to fill up the holes", "the player follows the movement of the gopher and fills the holes as they are dug out", or "the reaction of the player to the movement of the gopher is slow", which also match the comments characterizing the demonstrations of the human expert.

Meanwhile, the DQN agent was identified as the least human-like by the participants, which transpired in comments such as "the player systematically fills the holes with great precision", "the movements are machine-like", "the reaction time is very high", "unnatural lateral movement while filling the holes", or "tries to needlessly fill regions without holes".

Hence, it can be concluded that the proposed method builds an agent with a human-like behavior by combining the goal-oriented learning scheme of RL and the tendency of the human expert's behavior.

2.4.3 Torcs

Experimental setting

Fig. 2.7: A screenshot of Torcs.

The Apple and Gopher game being cases of the discrete action space environment, one might also wonder about the effectiveness of the proposed method in a continuous action space case. Additionally, considering the potential real-world applicability and social impact of the proposed method, the rise in popularity of the autonomous driving task, as well as the need for autonomous agents to be able to interact with other human agents, the Torcs Racing Car Simulator [121] was opted for, in virtue

of being a well-known environment that is used for autonomous driving AI research (see [54, 124], as the representative task for the continuous action space case). A screenshot of the game is shown in Fig. 2.7. The Gym Torcs environment [123] as a basis for the experiments in this chapter.

The agent observations consisted of 65 features including LIDAR-like sensors keeping track of the distance between the car and edges of the track or the opponents, current speed, and acceleration with a bi-dimensional continuous action space to control the steering and throttling of the car while using the raced distance as the reward function. This format allows for more precision when used as state representation for the agent when compared to 2D pictures of the road for example. As a result, the potential noise and error caused by feature extractors can be virtually eliminated. This allows us to objectively train the different agents as well as evaluate their performance.

Furthermore, the Torcs racing car simulator was upgraded to simulate a situation where the human decision factor would stand out more. This was performed by generating obstacles in the form of stationary bots and adding player demonstration recording support, which was used to record 220 episodes of 60 s of gameplay, which corresponded to 792,000 transitions. The RL agent was based on the DDPG implementation in the OpenAI Baselines [19] and was adapted to the autonomous driving problem. This DDPG agent was first trained for 10^7 training timesteps by using the Adam [49] optimizer, learning rates of the critic and the actor components set to 10^{-4} and 10^{-3}, and a replay memory of size 10^6, before being used to generate the necessary trajectories needed to train the proposed hybrid model.

A strict IL part consisted in imitating a human expert based on recorded data of the latter while driving in the same environment configuration as the RL experiment via the

OpenAI [19] implementation of the GAIL method and its default hyperparameter values except for the training timestep limit that was set to $5 \cdot 10^6$, empirically determined to allow convergence. The proposed hybrid agent based on the GAIL implementation with the discriminator loss modified based on Eq. 2.3.1 as described in section 2.3.2 was also trained with the same environment configuration. The GAIL-based human imitation agent and hybrid agent were trained with the discriminator learning rate set to 10^{-3} and a KL constraint of 10^{-2} via the Adam optimizer [49] over $5 \cdot 10^6$ and $7.5 \cdot 10^6$ training timesteps, respectively. Furthermore, the impact of the trade-off coefficient was explored by experimenting with several values to determine the most appropriate value for the sensitivity test. Following the search over $alpha$ values, a sensitivity test in a double-blind fashion was also conducted to evaluate the human-likeliness of each reference method as well as the proposed one. Similarly to the Apple and Gopher games, the sensitivity test was also preceded by a trial phase, during which each participant got to know the rule of the game, as well as familiarize himself with the controls and game mechanics. In the Torcs Racing Car Simulator case, each participant was presented with two 30 seconds videos of each model playing the game and requested to label it as either a human or a machine, as well as providing comments which motivated such a decision.

Results and Discussion

Finally, to evaluate the performance of a given agent in the Torcs Racing Car Simulator, the average distance traveled by the said agent during a time frame of 5 seconds is used to gauge the effective progress of the agents across the racing track. Moreover, the score for a time frame is set to 0 if the agent crashes or exits the racing track. Then,

Table 2.3: Results of Torcs

	Game Score				Sensitivity Test
Agent	**Max**	**Min**	**Avg**	**Std.**	**Identified as Human**
Human	696	588	637	31.1	26
DDPG (RL)	823	818	**819**	0.5	16
GAIL (IL)	608	23	527	72.4	<u>27</u>
Proposed (RL+IL)	817	107	<u>661</u>	179	**32**

Best and second best results are emphasized in bold and underlined, respectively. The evaluated hybrid model corresponds to a trade-off coefficient of $\alpha = 0.5$

the impact of the trade-off coefficient on the hybrid agent was investigated following the results documented in Fig. 2.8.

For $\alpha = 0.0$, the hybrid agent is equivalent to the default imitation of the human expert with the GAIL method. The hybrid agent's score then rises progressively with $\alpha \rightarrow 0.5$, thereby demonstrating a gain in performance obtained from the DDPG agent's demonstrations. From $\alpha = 0.5$ onward, the performance of the hybrid agent comes closer to that of the RL agent, but with moderate results for extreme values of α. Specifically, the deterministic nature of the DDPG agent's trajectories caused the hybrid agent to overfit the DDPG agent behavior, and thus it performed poorly during the testing phase.

In contrast to the Apple and the Gopher game, the performance gap between the RL agent and the human expert in this context was relatively lower. Moreover, the constraint of the "steering" and "acceleration" action to the $[-1, 1]$ range, as well as their continuous nature, resulted in easier-to-mix demonstrations for the hybridization process. A hybrid agent that performs better than the human expert can thus be obtained with smaller values of the trade-off coefficient α. Consequently, the hybrid

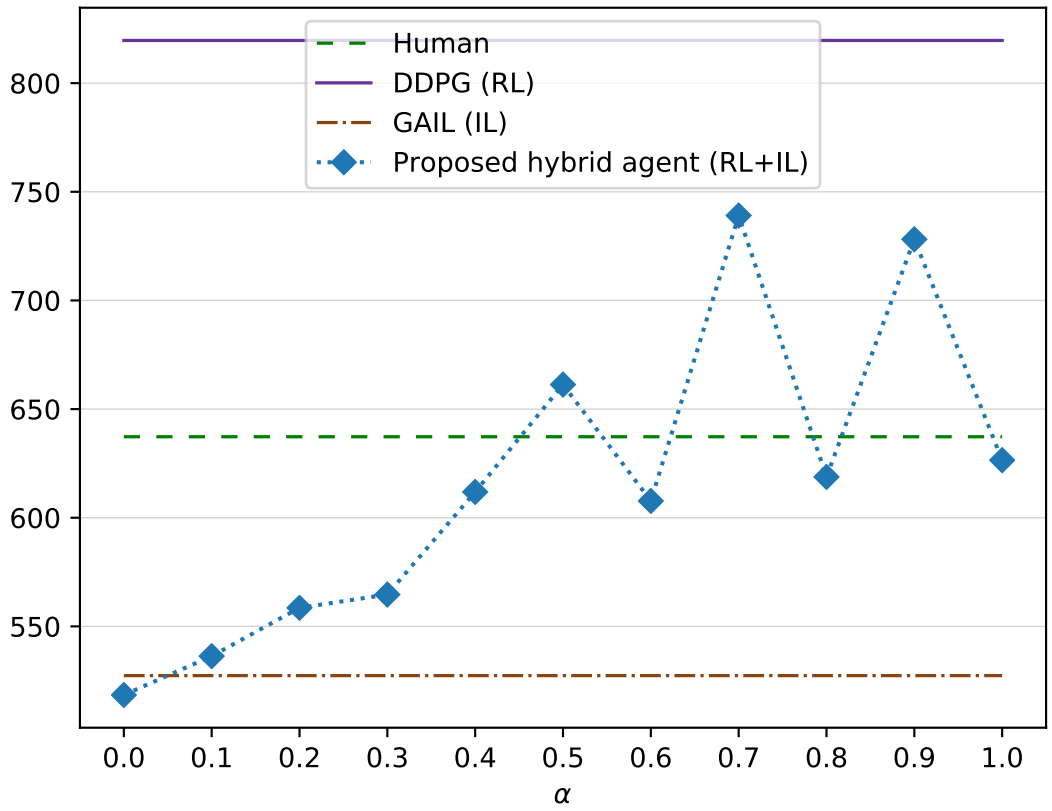


Fig. 2.8: Effect of the trade-off coefficient α on the proposed hybrid agent's score for Torcs. The average scores of the other agents are also provided for reference purposes.

agent corresponding to $\alpha = 0.5$ was used as the representative agent in the following experiments.

The first evaluation phase consisted in measuring the performance of the proposed model and comparing it relative to the human, GAIL human imitation, and DDPG agent respectively, as listed in Table 2.3.

The GAIL method can imitate expert trajectories generated by RL policies (results of [42]) or the deterministic bots coming with the Torcs Racing Car Simulator. However, it was less effective when applied to a human expert's trajectories and

achieved approximately half the score of the latter at best. This could be attributed to the human expert's policy being relatively more complex and difficult to approximate with standard neural network structures.

Furthermore, the proposed hybrid agent captured and demonstrated specific behaviors of the human expert and RL agent, namely a higher speed than the imitated human expert and deceleration during turns respectively. This was supported by the recurring comments collected from the participants of the sensitivity test, such as "the player decelerates before taking a turn or passing opponents", or "maintains a relatively constant speed overall, but with acceleration when the path follows a straight line". Similar comments were also given by the sensitivity test participants when characterizing the human expert demonstrations. Additionally, it also completed full laps like the human expert and DDPG agent.

The DDPG agent was least likely to be recognized as a human due to its high-performance behavior with comments such as "drives really fast" or "take turns at high speed". The other three agents were recognized as a human at a similar level. It can thus be concluded that the hybrid model was at least more likely to be recognized as a human than the DDPG agent, while still achieved higher performance than the human agent and its imitation.

2.5 Conclusion

This chapter introduced a method to build a hybrid agent that behaves in a human-like fashion imitated from a human expert while retaining some of the high performance displayed by pure reinforcement learning agents with the ultimate aim of obtaining the

best of both worlds. The proposed method is based on state-of-the-art reinforcement and imitation learning algorithms at the time and has two variants for the discrete and continuous action spaces respectively.

The proposed method was applied to an original game and a game of the Atari57 suite for the discrete action space case and the Torcs Racing Car Simulator for the continuous action space case. Upon evaluation of its objective performance, its human-likeliness was also evaluated via a sensitivity test. While human-acceptable agents can be considered as the goal of this study, the human-likeliness heuristic of an agent was used as an alternative, albeit narrower measure for a human-acceptable behavior. The rationale is that an agent behaving in a way deemed human-like is also likely to be deemed acceptable by other human players when used as game AI for example. The "human-acceptable" criterion, however, would require a definition from scratch for each task, and its evaluation is correspondingly difficult.

Nevertheless, the agents trained using the proposed method successfully exhibited behaviors borrowed from both experts, namely an increase in performance following the reinforcement learning agent and a human-like behavior following the human counterpart. More specifically, the performance of the hybrid agents surpassed that of the human expert in each environment experimented on. A thorough analysis of comments collected from the participants of the double-blind sensitivity test further showcased the similarity in behavior of both the proposed agents and the corresponding human expert. In light of the conducted experiments, the sensitivity evaluation approach adopted in this study would benefit from a larger population sample and preferably formed with participants that are familiar enough with the games and simulations that were used to train the different agents.

Chapter 3 Toward Human Cognition-inspired High-Level Decision Making For Hierarchical Reinforcement Learning Agents

3.1 Introduction

Despite the recent success of contemporary deep reinforcement learning (DRL) methods [14, 31, 33, 67, 95], they are still considerably sample inefficient when applied to long-horizon, sequential decision-making tasks, which usually overlap with sparse and delayed rewards problems, further increasing their complexity.

A growing body of studies in human behavior, cognitive science, neuroscience, and computational biology suggests that human behavior is hierarchically organized [6]. This is theorized to be a critical part that allows us to efficiently understand and learn to solve various challenging tasks with relatively limited amounts of data compared to RL methods [110, 122].

Inspired by this, the hierarchical reinforcement learning (HRL) framework aims to improve the efficiency of conventional (flat) RL by introducing temporal abstraction in the decision-making process of an agent [3, 6, 16, 108]. While existing HRL methods have empirically demonstrated a considerable improvement in efficiency over conventional DRL methods [25, 30, 52, 60, 113], most HRL methods rely on *fixed length temporal abstraction*. Moreover, the decision-making occurring at higher levels in the agent's hierarchy is still often based on the *observations at the lowest level*.

This chapter introduces the *hierarchical world model* (HWM), a world modeling method that aims to better fit the theorized hierarchical structure of the human decision-making process. Owing to its hierarchical structure, the proposed model inherently provides (1) a *temporally abstract state representation* summarizing an arbitrary number of lower-level states, and (2) an adaptive temporal abstraction mechanism to divide long-horizon, sequential decision tasks into smaller tasks of variable lengths.

Preliminary experiments in using the HWM as a representation learning mechanism for a Dreamer-based agent [33, 35] demonstrated improvements in sample efficiency and final performance of the agent. Further analysis of the representations learned by the proposed HWM also suggests that the HWM can learn a hierarchically structured internal state representation that aligns with human understanding and intuition [5, 110] on a given task.

Finally, the potential compatibility of the HWM with the HRL framework is shown by expanding the underlying theory of the proposed HWM to incorporate more rigorous hierarchical decision-making. This would bring the practical HRL framework closer to the human decision-making it is originally inspired from, and serve as a tool for analyzing RL agents' decision-making. In practice, this could help broaden the gamut of tasks to which deep RL methods can be successfully applied for automation.

3.2 Preliminaries

3.2.1 Reinforcement learning and hierarchical reinforcement learning

For more generality, let us consider the finite time horizon partially observable Markov decision processes (POMDP) framework, denoted by the tuple $(\mathbb{S}, \mathbb{A}, P_{s,a}, R, \mathbb{O}, P_o, H)$, where $\mathbb{S}$ is the state set, $\mathbb{A}$ is the action set, $P_{s,a} : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow [0, 1]$ is the state transition probability, $R : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}$ the reward function, $\mathbb{O}$ the set of partial observations, $P_o : \mathbb{S} \rightarrow \mathbb{O}$ the emission probability distribution and H the horizon (maximum episode length). The decision process of an RL agent can be represented by a stochastic policy $\pi : \mathbb{S} \times \mathbb{A}$, which defines the probability of action a being selected given a state s. From a probabilistic perspective [59, 105], the POMDP modeling the dynamics of the system and the decision-making of the agent can be formally

expressed as the following generative process:

$$p(O, S, A) = \prod_{t=1}^{H} p(o_t|s_t)\, p(s_t|s_{t-1}, a_{t-1})\, \pi(a_t|o_t) \tag{3.2.1}$$

In the hierarchical RL (HRL) framework, the policy of the agent is decomposed into hierarchically structured component policies. The lowest level in the hierarchy operates at the finest time scale, the same as a flat RL policy. On the other hand, the higher levels in the hierarchy operate at a coarser time scale, thus resulting in temporal abstraction. This would allow HRL agents to explore the state space more efficiently by leveraging sequential combinations of low-level policies (also referred to as *options* or *skills*), allowing for a more efficient sequential decision-making [3, 6, 108] in long-horizon tasks. Concurrently, temporal abstraction allows for a better credit assignment through time, which is especially helpful in long-horizon, sparse and delayed reward tasks [108, 113].

Consider the simplest case of a two-level hierarchical agent composed of π^{H} and π^{L} respectively representing the high and low-level policies. π^{H} selects a *skill* on which π^{L} will be conditioned. In practice, π^{H} is usually conditioned on the low-level observations, and its time scale is often set to a fixed length denoted as k hereafter [25, 30, 52, 113]. Denoting the *skill* space by $\mathbb{E}$, and augmenting the

generative process of the POMDP in Eq. 3.2.1 yields to following generative process:

$$p(O, S, A, E) = \prod_{\tau=0}^{H/k} \pi^{\mathrm{H}}(e_\tau | o_{k\tau}) \prod_{t=k\tau}^{k(\tau+1)} p(o_t|s_t) \\ p(s_t|s_{t-1}, a_{t-1})\, \pi^{\mathrm{L}}(s_t|o_t, e_\tau). \tag{3.2.2}$$

The graphical models for Eq. 3.2.1 and Eq. 3.2.2 are illustrated in Fig. 3.1 (a) and (b), respectively.

3.2.2 Hierarchically organized behavior

A growing body of studies [5, 6, 110, 122] at the intersection of neuroscience, cognitive science, psychology, and computational biology suggests that the human decision-making process is hierarchically organized. For example, when faced with a task such as *making a trip abroad*, humans will generally divide it into a sequence of sub-tasks such as *booking the flight*, *packing the luggage*, *driving to the airport*, *boarding the plane*, and so on. Each sub-task can be further divided into sub-sub-tasks, down to the finest granularity of actions such as bodily movements. This concept is referred to as *temporal abstraction* and allows us to efficiently learn, plan, and act in a wide gamut of activities. It is a fundamental principle underlying the HRL framework introduced in 3.2.1, where each of the aforementioned sub-tasks would be realized by learning and executing the appropriate *skill*.

While existing HRL methods [25, 30, 52, 60, 108, 113] do structure the decision-making process of the agent hierarchically, the *skill* selection at higher levels in the hierarchy is more often than not based on the observations at the finest level of the

hierarchy. In the case of example task *making a trip abroad*, this is akin to having the high-level policy decide to *drive to the airport* while using a very exhaustive representation of the current state of the agent, instead of the more abstract state *luggage packed and ready to go to the airport*.

Following this line of thought, [110, 122] suggest that our hierarchical decision-making process is intertwined with a corresponding *temporally abstract state representation*. Intuitively, such abstracted state representation would align with intermediate sub-goals, milestones, or *bottleneck states* that contribute to solving the overall task, thereby greatly simplifying planning, exploration, and execution.

The ability to create long-term plans and strategies without necessarily interacting with the world happens to also be tied to such abstract state representation. This is made possible by leveraging our internal model of the world, which is theorized to also be hierarchical. For example, humans can imagine both the detailed process of *folding shirt* (low-level), as well as the more abstract process of *putting folded shirts into the suitcase* (high-level). Botvinick et al. empirically demonstrated the benefit of having a *temporally abstract model* [5]. They proposed a model-based HRL variant of the *options framework* [108], where the standard HRL agent is augmented with a temporally abstract model that allows the agent to directly plan at a coarser time scale. This resulted in a great improvement in performance, but also sample efficiency. However, the *skills*, and the temporally abstract model were manually engineered, thus limiting the generality of the proposed approach.

As suggested by [3], it would be desirable to endow HRL agents with *an additional mechanism that allows autonomous extraction of temporally abstract state, and the corresponding temporal dynamics model*. Moving toward a human cognition-inspired

decision-making process, it becomes relevant to consider recent methods for dynamics learning that fall under the umbrella of *world modeling*, to complement standard HRL agents with a general, end-to-end mechanism for discovery and learning of temporally abstract state representations and dynamics.

3.2.3 World models

The broad range of methods that have marked the recent resurgence of model-based RL (MbRL) are referred to as *world models* [29, 33–35]. Such methods have not only demonstrated either competitive or superior performance to the leading model-free RL methods but also improved the sample efficiency of the agents.

One of the key factors behind the success of world modeling methods is the introduction of self-supervised learning objectives [50] to learn more compact and meaningful representations of the internal state belief s_t in Fig. 3.1 (a). This allowed separating the decision-making component (policy) from the raw and usually noisy pixel-based observations, greatly simplifying the learning of the decision-making process. Additionally, world modeling techniques leverage Recurrent Neural Networks (RNN) [12, 28, 43], to approximate the state transition dynamics $p(s_t|s_{t-1}, a_{t-1})$ from Eq. 3.2.1. Such approximation can then be used to collect samples in an *imaginary* environment in the place of the real environment, leading to a drastic improvement in sample efficiency.

Through the series of *Dreamer* agents [33, 35], Hafner et al. take this concept one step further by seamlessly incorporating the prediction of the reward and episode termination with an actor-critic [31, 95] policy into the model. Leveraging the differentiable dynamics of the latter allows *Dreamer* agents to directly improve their

policy in an end-to-end manner over simulated trajectories.

To further emphasize the gap between the desideratum of this study and the existing world modeling methods, let us consider how a world model enables RL agents to also create plans (sequences of actions) without rolling out the physical or simulated environment. This feature of models is indeed a close, albeit drastically simplified version of our ability as humans to mentally simulate various scenarios and their development for a given task. Namely, humans can maintain a set of intricately organized internal beliefs about the surrounding environments and even the actors that populate them. A practical example would be a game of chess, where a professional player is planning multiple moves, anticipating various strategies of his opponent, and developing counter moves. For an example more grounded in daily life activities, let us again consider the task of planning a trip. For us, such planning is a process that ranges over low-level details such as "what documents to prepare", "which clothes to take ?", "what time to wake up at ?", to more and more abstract levels such as "taking the train from city A to city B both in country C" then "taking the plane from city B to city D in country E", and so on [110].

As humans, our decision-making process is indeed not limited to low-level and densely detailed state beliefs (representations) of the world. Most, if not all world model methods, however, only estimate the internal state belief and transition dynamics at the finest time scale. Inspired by the theorized hierarchically structured model of animals and humans presented in 3.2.2, this chapter seeks to augment conventional modeling methods with a hierarchical structured dynamics model.

3.2.4 Variational temporal abstraction

The variational temporal abstraction (VTA) [48] framework was introduced as a discovery method for temporally hierarchical structure and representation in sequential data. Formally, VTA assumes the existence of a sequence of observations $O = \{o_1, o_2, ...o_H\}$ of length H that can be decomposed into N non-overlapping sub-sequences $O = (O_1, O_2, ..., O_N)$, such that each sub-sequence $O_i = \{o^i_{1:l_i}\}$ has length l_i, and $\sum_{i=1}^{N} l_i = H$. Each observation o_t is generated from the corresponding *low-level state* w_t, such that each observation sub-sequence O_i is associated with a low-level state sub-sequence W_i. Finally, each low-level state sub-sequence W_i is assumed to have been generated from a *temporally abstract state* z_i.

To efficiently separate sub-sequences W_i corresponding to different z_i, the VTA framework leverages a binary random variable M referred to as *binary boundary indicator* instead of modeling both the number of sub-sequences N and their lengths L. At an arbitrary time-step t, the binary indicator m_t specifies whether or not a new sub-sequence starts at time step $t+1$. The generative process for an observed sequence O is formally defined as follows:

$$p(O, W, Z, M) = \prod_{t=1}^{H} p(o_t|w_t)\, p(w_t|z_t, w_{t-1}, m_{t-1}) \\ p(z_t|z_{t-1}, m_{t-1})\, p(m_t|w_t). \tag{3.2.3}$$

The *temporally abstract* state z and the low-level state abstraction w are approximated using the proposed hierarchical recurrent state-space model, which is trained using

sequential variational inference [50].

While it does provide an adaptable mechanism to learn temporally abstracted dynamics and the corresponding abstract states, it cannot be directly incorporated into a hierarchically structured decision-making process. Namely, it does not model the influence of the actions of either low or high-level actions originating as causal components of the observed sequences.

3.3 Proposed method

3.3.1 Theoretical model

This section introduces the *hierarchical world model* (HWM), which combines conventional world modeling techniques and VTA to capture flexible, temporally abstract dynamics, and the corresponding state representations structured hierarchically.

First, let us consider the world model structure of the Dreamer agents [33, 35] introduced in Section 3.2.3. For a given task formalized as a POMDP, such a world model approximates the internal belief state s_t as a single random variable using sequential variation inference. Following the VTA [48] framework, it is assumed that the internal belief state s_t itself can be decomposed into the hierarchically structured components w_t, and z_t, with the boundary indicator m_t determining when the temporally abstract transition takes place.

Since the main interest of this study is learning the dynamics of the environment, let us consider the sequence of observations as generated by a fixed policy π, which influence is hereafter represented by the random variable A. Based on Eq. 3.2.1, and

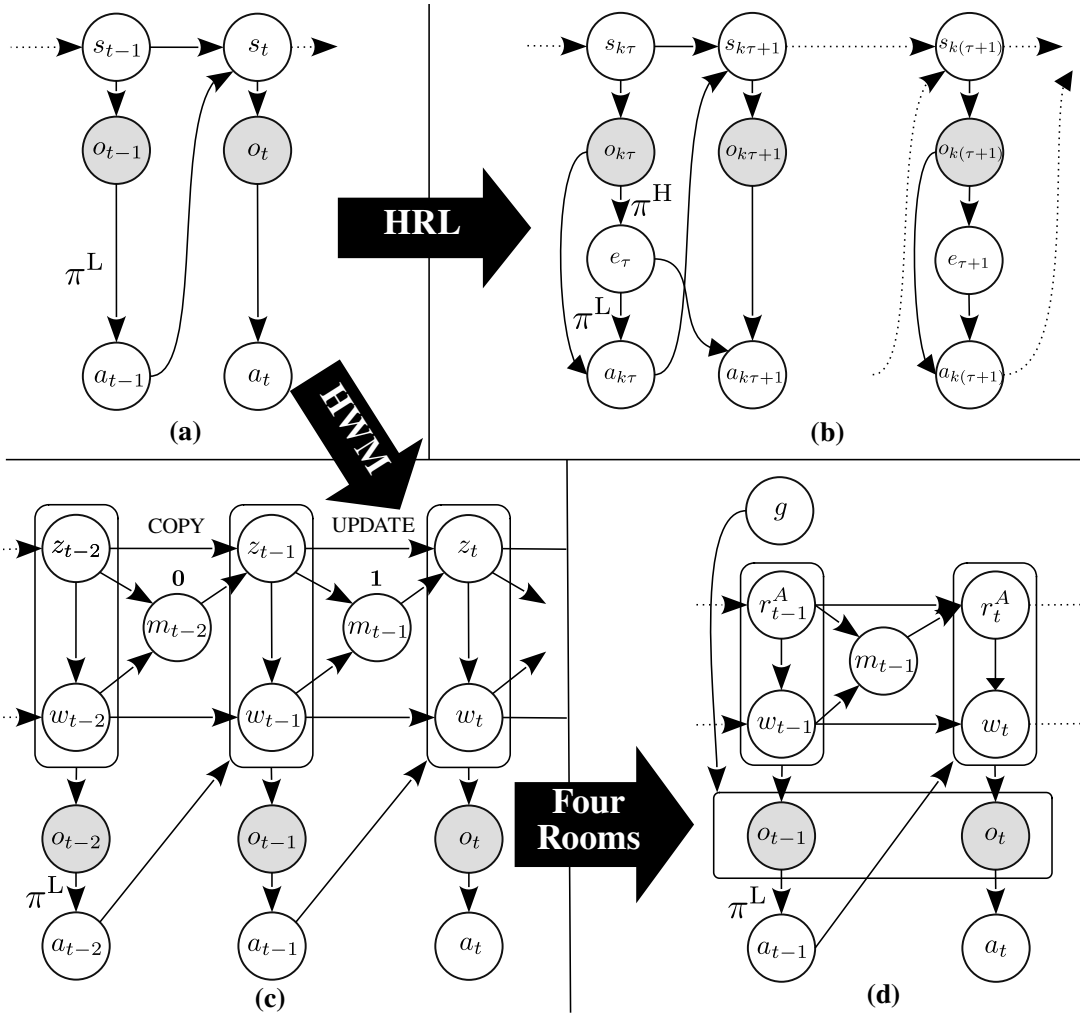


Fig. 3.1: (a) Graphical model of a POMDP task dynamics and decision-making process of a flat RL agent. (b) Graphical model of the temporal abstraction introduced to the task following the HRL framework. (c) Dynamic Bayesian network of the POMDP task with a hierarchically structured state representation following the proposed HWM. (d) Dynamic Bayesian network of the proposed HWM applied to the *Four Rooms* task. The stochastic variable g denotes the goal of the agent, and r^A denotes the room of the agent. The variables g and r^A would correspond to the z variable in the proposed HWM formulation. Finally, w denotes more detailed information about the agent relative to the room it is located in.

Eq. 3.2.3 the generative process the proposed model is based upon is obtained formally defined as follows:

$$p(O, W, Z, M|A) = \prod_{t=1}^{H} p(o_t|w_t, z_t)\, p(m_t|w_t, z_t, a_t) \\ p(z_t|z_{t-1}, m_{t-1}, w_{t-1}, a_{t-1})\, p(w_t|z_t, w_{t-1}, a_{t-1}). \tag{3.3.1}$$

The corresponding graphical model is documented as Fig. 3.1 (c).

3.3.2 Learning and inference of task dynamics

The generative process proposed in Eq. 3.3.1 is modeled using a hierarchical recurrent state-space model [48, 88]. More specifically, $p_\theta(o_t|w_t, z_t)$ is parameterized as the decoder component of a variational auto-encoder [50]. The prior over the temporally abstract transition is modeled by $p_\theta(z_t|z_{t-1}, m_{t-1}, w_{t-1}, a_{t-1})$ approximated using deep neural networks.

To improve the modeling of long-term dynamics, z_t is decomposed into a deterministic component c_t and stochastic component v_t [33–35, 48]. The deterministic transitions for c_t are modeled using the following rule:

$$c_t = \begin{cases} c_{t-1} & \text{if } m_{t-1} = 0 \text{ (COPY)} \\ f_{\text{z-rnn}}(z_{t-1}, h_{t-1}, c_{t-1}) & \text{otherwise (UPDATE)} \end{cases}$$

where $f_{\text{z-rnn}}$ is a GRU neural network [12, 48], and h_{t-1} represents the preceding

sequence of (w, a) pairs. The stochastic component v_t is implemented as a Normal distribution: $v_t \sim \mathcal{N}(\mu_v(c_t), \sigma_v(c_t))$, where μ_v and σ_v are parameterized by their respective densely connected feed-forward neural networks. The temporally abstract state z_t is thus obtained by concatenating c_t and v_t.

Similarly, the low-level state w_t is also decomposed into deterministic and stochastic components, respectively denoted by h_t and y_t. Unlike in VTA [48], h_t is seamlessly updated like in Clockwork VAEs [88] using the rule:

$$h_t = f_{\text{w-rnn}}\big(w_{t-1}, a_{t-1}, h_{t-1}\big),$$

where $f_{\text{w-rnn}}$ is the GRU neural network associated with the low-level state transitions. The stochastic component y_t is implemented using a normal distribution $y_t \sim \mathcal{N}(\mu_y(h_t), \sigma_y(h_t))$, where μ_y and σ_y are parameterized by their respective densely connected feed-forward neural networks. The concatenation of h_t and y_t is then used to represent the low-level state w_t.

Finally, the prior boundary detector $p_\theta(m_t|w_t, z_t, a_t)$ is parameterized using a densely connected neural network with a final sigmoid activation function.

The hierarchy of state representation components and the corresponding dynamics is inferred [50] using the parameterized variational distribution $q_\phi(Z, W, M|O, A)$. The latter is decomposed as follows:

$$q_\phi(Z, W, M|O) = q_\phi(M|O) \prod_{t=1}^{H} q_\phi(w_t|z_t, M, O)\, q_\phi(z_t|M, O), \tag{3.3.2}$$

with $q_\phi(M|O) = \prod_t \text{Bernoulli}(m_t|\sigma(\varphi(O)))$, where σ is the sigmoid function, and φ is a temporal convolution operation over a sequence of observations. Both $q_\phi(w_t|z_t, M, O)$ and $q_\phi(z_t|M, O)$ are approximated using the mean field approximation-based method [48, 88].

The proposed model can be trained by maximizing the variational lower bound (VLB) which is derived from Eq. 3.3.1 as follows:

$$
\begin{aligned}
\log p(O|A) &= \log \sum_M \int_{Z,W,A} \prod_{t=1}^{H} p(o_t|w_t, z_t)\, p(m_t|w_t, z_t, a_t) \\
&\qquad\qquad p(z_t|z_{t-1}, m_{t-1}, w_{t-1}, a_{t-1})\, p(w_t|z_t, w_{t-1}, a_{t-1}) \\
&= \log \sum_M \int_{Z,W} \prod_{t=1}^{H} p(o_t|w_t, z_t)\, p(m_t|w_t, z_t) \\
&\qquad\qquad p(z_t|z_{t-1}, m_{t-1}, w_{t-1})\, p(w_t|z_t, w_{t-1}) \\
&= \log \sum_M \int_{Z,W} p(O|W, Z)\, p(M|W, Z)\, p(Z|Z', M', W')\, p(W|W', Z') \\
&\geq \sum_M \int_{Z,W} q(W, Z, M|O) \\
&\qquad\qquad \log \frac{p(O|W, Z)\, p(M|W, Z)\, p(Z|Z', M', W')\, p(W|W', Z')}{q(W, Z, M|O)} \\
&= \sum_M \int_{Z,W} q(W, Z, M|O) \Big[\log p(O|W, Z) + \\
&\qquad\qquad \log \frac{p(M|W, Z)\, p(Z|Z', M', W')\, p(W|W', Z')}{q(Z, W, M|O)} \Big] \\
&= \underbrace{\mathbb{E}_{q(Z,W,M|O)} \Big[\log p(O|W, Z) \Big]}_{\text{reconstruction}} - \underbrace{KL\Big[q(Z, W, M|O) \,||\, p(Z, W, M) \Big]}_{\text{divergence}}
\end{aligned}
\tag{3.3.3}
$$

In practice, the parameter vectors θ and ϕ can thus be learned by maximizing the

VLB derived from Eq. 3.3.3 such that:

$$\log p(O|A) \geq \mathop{\mathbb{E}}_{q_\phi}\Big[\log p_\theta(O|W,Z)\Big] - \mathrm{KL}\Big[q_\phi(Z,W,M|O) \,||\, p_\theta(Z,W,M)\Big]. \quad (3.3.4)$$

Therefore, for a given sequence of observation-action pairs $\{(o_t, a_t)\}_{t=1}^{H}$, the VLB derived in Eq. 3.3.4 is approximated as:

$$\begin{aligned} J_{\mathrm{HWM}}(\theta,\phi) = \sum_{t=1}^{T} \log p_\theta(o_t|w_t,z_t) - \mathrm{KL}\Big[q_\phi(z_t) \,||\, p_\theta(z_t)\Big] - \\ \mathrm{KL}\Big[q_\phi(w_t) \,||\, p_\theta(w_t)\Big] - \mathrm{KL}\Big[q_\phi(m_t) \,||\, p_\theta(m_t)\Big]. \end{aligned} \quad (3.3.5)$$

The first term in Eq. 3.3.5 corresponds to the reconstruction objective across the observed sequence. The last three terms correspond to the Kullback-Leibler (KL) divergence between the generative and variational distributions used to approximate temporally abstract state z_t dynamics, the low-level state w_t's dynamics, and the sequence segmentation based on the boundary indicator m_t, respectively. More specifically, the derivation of the empirical KL divergence term is obtained following the subsequent procedure based on Eq. 3.3.3. For simplicity, the θ and ϕ parameters are omitted.

$$\mathrm{KL}[q \,||\, p] = \sum_{M} \int_{Z,W} q(Z,W,M|O)\Big[\log q(Z,W,M|O) - \log p(Z,W,M)\Big] \quad (3.3.6)$$

$$
\begin{aligned}
&= \sum_M q(M|O) \Bigg[\log q(M|O) + \sum_t \int_{Z,W} q(Z,W|M,O) \\
&\qquad \log \frac{q(z_t|M,O)\, q(w_t|z_t,M,O)}{p(z_t|z_{t-1},m_{t-1})\, p(w_t|w_{t-1},z_{t-1})\, p(m_t|w_t,z_t)} \Bigg] \\
&\approx \sum_M q(M|O) \Bigg[\log q(M|O) + \sum_t \int_{z_t,w_t} q(z_t,w_t,|M,O) \\
&\qquad \log \frac{q(z_t|M,O)\, q(w_t|z_t,M,O)}{p(z_t|z_{t-1},m_{t-1})\, p(w_t|w_{t-1},z_{t-1})\, p(m_t|w_t,z_t)} \Bigg] \\
&= \sum_M q(M|O) \Bigg[\log q(M|O) + \sum_t \mathrm{KL}\Big[q'(z_t) \,||\, p'(z_t) \Big] + \\
&\qquad \int_{z_t,w_t} q(z_t,w_t,|M,O) \log \frac{q(w_t|z_t,M,O)}{p(w_t|w_{t-1},z_{t-1})\, p(m_t|w_t,z_t)} \Bigg] \\
&= \sum_M q(M|O) \Bigg[\log q(M|O) + \sum_t \mathrm{KL}\Big[q'(z_t) \,||\, p'(z_t) \Big] + \\
&\qquad \int_{z_t} \underbrace{q'(z_t)}_{\text{sample } z_t} \mathrm{KL}\Big[q'(w_t) \,||\, p'(w_t) \Big] - \log p(m_t|w_t,z_t) \Bigg] \\
&\approx \sum_M q(M|O) \Bigg[\log q(M|O) + \sum_t \mathrm{KL}\Big[q'(z_t) \,||\, p'(z_t) \Big] + \\
&\qquad \mathrm{KL}\Big[q'(w_t) \,||\, p'(w_t) \Big] - \log p(m_t|w_t,z_t) \Bigg] \\
&= \sum_M q(M|O) \Bigg[\log \frac{\prod_t q(m_t|O)}{\prod_t p(m_t|w_t,z_t)} + \sum_t \mathrm{KL}\Big[q'(z_t) \,||\, p'(z_t) \Big] + \\
&\qquad \mathrm{KL}\Big[q'(w_t) \,||\, p'(w_t) \Big] \Bigg]
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{t'} \mathrm{KL}\Big[q'(m_{t'}) \,||\, p'(m_{t'})\Big] + \sum_{M} \underbrace{q(M|O)}_{\text{sample } m_t} \Bigg[\\
&\qquad\qquad \sum_{t} \mathrm{KL}\Big[q'(z_t) \,||\, p'(z_t)\Big] + \mathrm{KL}\Big[q'(w_t) \,||\, p'(w_t)\Big] \Bigg] \\
&\approx \sum_{t} \mathrm{KL} \underbrace{\Big[q'(m_t) \,||\, p'(m_t)\Big]}_{\text{sequence decomposition}} + \underbrace{\mathrm{KL}\Big[q'(z_t) \,||\, p'(z_t)\Big]}_{\text{temporal abstraction}} + \underbrace{\mathrm{KL}\Big[q'(w_t) \,||\, p'(w_t)\Big]}_{\text{low-level state abstraction}}
\end{aligned}
\tag{3.3.7}
$$

$$
\begin{aligned}
\text{where } \; & q'(z_t) = q(z_t|M, O) \;\text{ and }\; p'(z_t) = p(z_t|z_{t-1}, m_{t-1}, w_{t-1}, a_{t-1}), \\
& q'(w_t) = q(w_t|z_t, M, O) \;\text{ and }\; p'(w_t) = p(w_t|w_{t-1}, a_{t-1}, z_t), \\
& q'(m_t) = q(m_t|O) \;\text{ and }\; p'(m_t) = p(m_t|w_t, z_t, a_t)
\end{aligned}
$$

3.3.3 Incorporation with an actor-critic

To enable decision-making based on the proposed HWM, the model is further extended with a reward predictor $p_\theta(\hat{r}_t|w_t, z_t)$. This predictor is approximated as a Normal distribution $\hat{r}_t \sim \mathcal{N}(\mu_{\hat{r}}(w_t, z_t), 1)$ where the mean $\mu_{\hat{r}}(w_t, z_t)$ is parameterized using a feed-forward neural network. The reward predictor is then trained to maximize the likelihood of the reward r_t collected during trajectory sampling. Formally, this corresponds to minimizing the objective function $J_{\mathrm{R}}(\theta, \phi) = \sum_{t=1}^{H} -\log p_\theta(r_t|w_t, z_t)$ jointly with $J_{\mathrm{HWM}}(\theta, \phi)$ defined in Eq. 3.3.5.

Following [33, 35], the actor-critic is composed of an action and a value model. In the scope of this chapter, let the action model be denoted as the policy $\pi_\psi(a_h|w_h, z_h)$ to be a distribution over discrete actions. The value model is defined as a state value

function that estimates the average reward-to-go from a given state (w_h, z_h), formally expressed as follows:

$$V_\psi^t = V_\psi(w_t, z_t) \approx \mathbb{E}_{q_\phi(\cdot|w_t,z_t)}\Big[\sum_{h=t}^{t+H} \gamma^{h-t}\hat{r}_h\Big].$$

Here, ψ denotes the vector of parameters that form the layers of the deep neural networks expressing both π_ψ and V_ψ.

The training objective of the value function is to regress the V_ψ^t estimate to the state value target denoted as V_λ^t. The latter is estimated using the λ-return estimation [33,35]. The objective function for the value learning components is thus formally expressed as:

$$J_{\mathrm{V}}(\psi) = \mathbb{E}_{p_\theta(w_t,z_t)}\Big[\sum_{h=t}^{t+H} \frac{1}{2}\big\|V_\psi(w_h, z_h) - V_\lambda^h\big\|^2\Big] \tag{3.3.8}$$

The role of the actor component is to select the action that maximizes leads to states with maximal value estimation. Exploration is implicitly incorporated by regularizing the policy entropy following [31,35]. This is formally expressed as follows:

$$J_\pi(\psi) = \mathbb{E}_{p_\theta(w_t,z_t)}\Big[\sum_{h=t}^{t+H} -\log \pi_\psi(a_t|w_h, z_h)\,\mathrm{sg}(V_\psi^h) - \eta\,\mathcal{H}\big[a_h|w_h, z_h\big]\Big] \tag{3.3.9}$$

with *sg* denoting the *gradient stopping operation*, $\mathcal{H}$ the entropy of the policy π_ψ, and η the entropy regularization coefficient.

Finally, the actor-critic component objective function $J_{\mathrm{AC}}(\psi)$ is then constructed

as follows:

$$J_{\mathrm{AC}}(\psi) = J_{\pi}(\phi) + J_{\mathrm{V}}(\psi). \tag{3.3.10}$$

The overall cost function of the proposed HWM as a representation learning component, and the actor-critic component to be minimized is thus expressed as follows:

$$J(\theta, \phi, \psi) = J_{\mathrm{HWM}}(\theta, \phi) + J_{\mathrm{R}}(\theta, \phi) + J_{\mathrm{AC}}(\psi). \tag{3.3.11}$$

While in practice the actor-critic component is jointly trained with the proposed HWM, let us emphasize that $J_{\mathrm{AC}}(\psi)$ is minimized over *simulated trajectories* of length H, instead of directing relying on data sampled during rollouts of the agent [14, 30, 95]. Those simulated trajectories are produced by the approximated dynamics transitions $p_\theta(w_t)$, $p_\theta(z_t)$, and $p_\theta(m_t)$ of the proposed HWM that were described in Section 3.3.1 and 3.3.2.

3.3.4 HRL extension of the decision-making

This section motivates how the proposed HWM can be further leveraged in the context of the HRL framework. This ties into bringing the decision-making process of HRL agents toward one that is more aligned with the hierarchically organized behavior observed in animals and humans, as described in Section 3.2.2.

To this end, let us first extend the proposed generative process in Eq. 3.2.1 to account for the decision-making of an HRL agent. The resulting generative process is

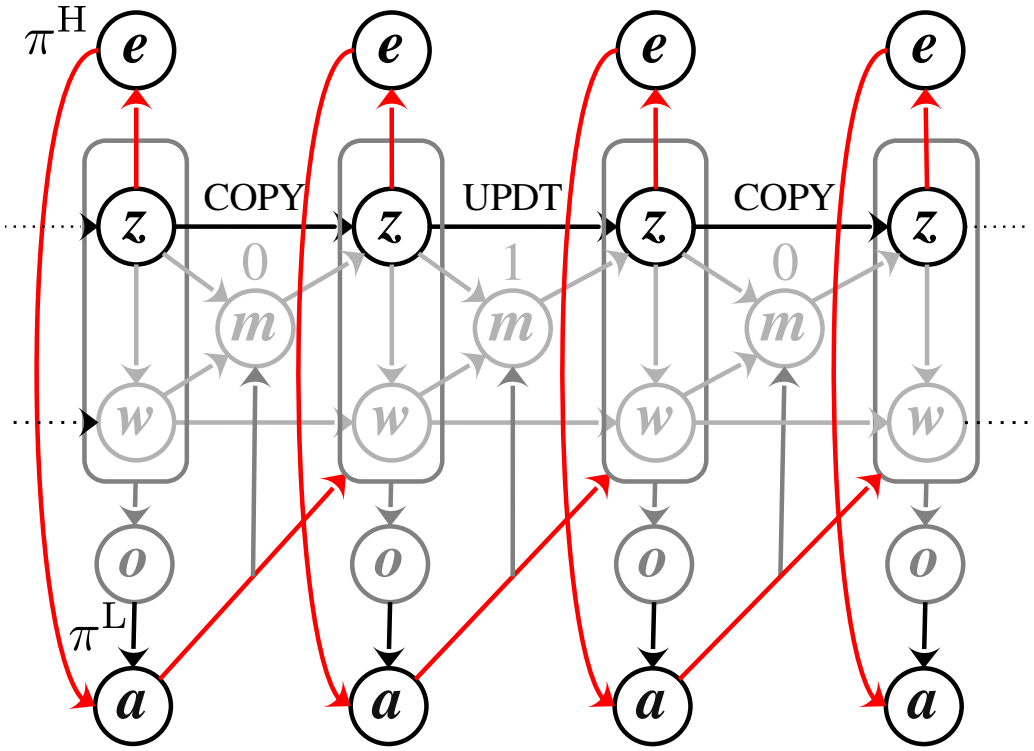


Fig. 3.2: Dynamic Bayesian network representing the generative process combining the HWM and HRL frameworks. The red line emphasizes the influence of the high-level policy's action over the high-level, temporally abstract state z's transition modeled by the proposed HWM.

illustrated in Fig. 3.2 while being formally expressed as:

$$
\begin{aligned}
p(O, W, Z, M, A, E) = \prod_{t=1}^{H} \; & p(o_t|w_t, z_t)\, p(m_t|w_t, z_t, a_t) \\
& p(z_t|z_{t-1}, m_{t-1}, w_{t-1}, a_{t-1}) \\
& p(w_t|z_t, w_{t-1}, a_{t-1}) \\
& \pi^{\mathrm{L}}(a_t|w_t, z_t, e_t)\, \pi^{\mathrm{H}}(e_t|z_t).
\end{aligned}
\tag{3.3.12}
$$

Marginalizing out the observation O, low-level state representation component W, boundary indicator M, and the low-level action A variables from Eq. 3.3.12 allows the recovery of the *temporally abstract* MDP:

$$p(Z, E) = \prod_{\tau} p(z_\tau | z_{\tau-1}, e_{\tau-1})\, \pi^{\mathrm{H}}(e_\tau | z_\tau). \tag{3.3.13}$$

Such temporally abstract MDP can be considered as a simplified version of the task to solve. The motivation for such an abstraction mechanism is intuited in Fig. 3.3.

Given the example of the *Four Rooms* task introduced in Section 3.4, this abstract MDP would correspond to a simpler problem of navigating between the eight rooms, as illustrated at the layer (c) in Fig. 3.3. Namely, the high-level policy π^{H} would thus be set to solve the task of making the overall agent reach a given room by guiding the low-level policy. At the low level, the corresponding *sub-policy* expressed by π^{L} executes the necessary sequence of actions that concretizes the high-level instruction, as proposed in the HRL framework [3, 16, 25, 60, 108].

By doing so, the resulting *HWM HRL Dreamer* agent is expected to expedite the exploration process in virtue of said exploration being conducted over a temporally abstracted and principled state space. This also happens to align with how a human would decompose such a task, and learn how to solve it efficiently, as described in Section 3.2.2.

Next, the training objective for each component of the aforementioned *HWM HRL Dreamer* agent is derived. From the theoretical perspective, the temporally abstract MDP derived in Eq. 3.3.13 happens to exactly match the standard MDP formulation

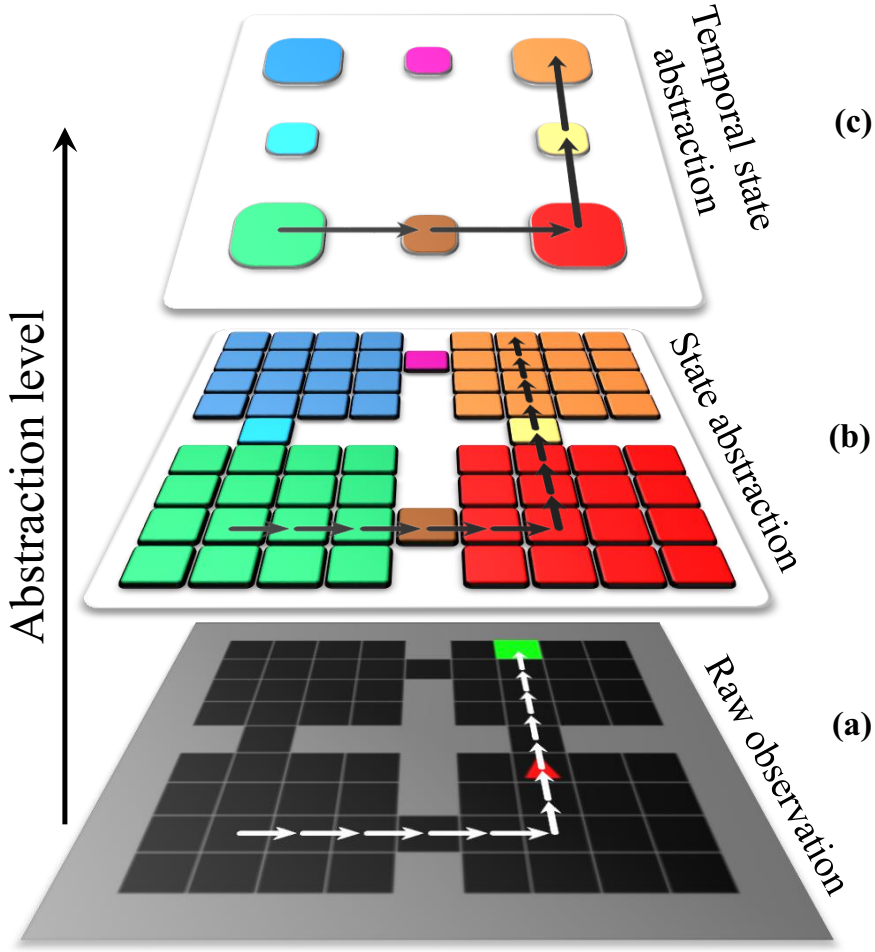


Fig. 3.3: Conceptual diagram based on the *Four Rooms* task, illustrating the different levels of abstractions that the proposed HWM is designed to model. At the level of raw pixel-based observations (a), the optimal trajectory of the agent from its starting room to the goal tile is depicted using white arrows. From that layer (a), humans can generally conceptualize a simpler state abstraction (b) that focuses on each cell of the maze that the agent can traverse. This abstraction layer would be equivalent to the low-dimensional latent space that a world model such as the one in Dreamer learns. Finally, layer (c) represents the temporally abstract state space that the proposed HWM is specifically designed to model. At this layer, all the cells that form a *room* of the maze are considered as an aggregate state. This simplifies the original task to the problem of *navigating from the starting room to the room that contains the goal (exit)*.

upon which RL algorithms are based upon [106]. This is intuitively illustrated in Fig. 3.3. Consequently, any RL method that is proven to work with a task that follows the standard MDP formulation can also be applied at the temporally abstract layer.

The proposed *HWM Dreamer* agent described in Section 3.3.3 thus appears a natural fit for such an extension. First, the state value estimator corresponding to the low-level policy is overloaded with the high-level action [89]. By doing so, the state value estimator at the low level approximates the value of being in a given state when following an arbitrary instruction from the high-level policy. Formally, this is equivalent to replacing all occurrences of $V_\psi(w_h, z_h)$ in Eq. 3.3.8 with $V^{\mathrm{L}}_\psi(w_h, z_h, e_h)$. This overloaded state value objective function is hereafter referred to as $J^{\mathrm{L}}_{\mathrm{V}}(\psi)$ to indicate that it corresponds to the low-level policy π^{L}_ψ. Namely,

$$J^{\mathrm{L}}_{\mathrm{V}}(\psi) = \mathbb{E}_{p_\theta(w_t, z_t)}\Big[\sum_{h=t}^{t+H} \frac{1}{2}\Big\|V^{\mathrm{L}}_\psi(w_h, z_h, e_h) - V^{\mathrm{L},h}_\lambda\Big\|^2\Big] \tag{3.3.14}$$

Similarly, the actor's objective function $J_\pi(\psi)$ defined in Eq. 3.3.9 is also overloaded to account for the guidance of the low-level policy π^{L}_ψ by the high-level policy π^{H}_ψ as follows:

$$J^{\mathrm{L}}_\pi(\psi) = \mathbb{E}_{p_\theta(w_t, z_t)}\Big[\sum_{h=t}^{t+H} -\log \pi^{\mathrm{L}}_\psi(a_t|w_h, z_h, e_h)\,\mathrm{sg}(V^{\mathrm{L},h}_\psi) - \eta\,\mathcal{H}\big[a_h|w_h, z_h, e_h\big]\Big]. \tag{3.3.15}$$

At the high level, the policy π^{H}_ψ outputs a high-level action based on the high-level component Z of the proposed hierarchical state representation described in

Section 3.3.2. The learning objective for the high-level state value estimator V_ψ^{H} is thus derived following the same principles as the for the traditional state value case described in Section 3.3.3. This is formally expressed as follows:

$$J_{\mathrm{V}}^{\mathrm{H}}(\psi) = \mathbb{E}_{p_\theta(z_t)}\Big[\sum_{\tau=t}^{(t+H)/k} \frac{1}{2}\big\|V_\psi^{\mathrm{H}}(z_\tau) - V_\lambda^{\mathrm{H},\tau}\big\|^2\Big] \tag{3.3.16}$$

where τ designates a time step at the coarser, temporally abstract level. To simplify the notation, it is assumed to temporally abstract transition occurs at a fixed interval k, hence the values of τ ranging from the arbitrary time step t to $(t+H)/k$, with H being the maximal length of a trajectory simulated using the learned dynamics. In practice, such transitions are of variable length and dictated by the prior distribution over the boundary indicator described in Section 3.3.2.

Consequently, the learning objective for π_ψ^{H} is similarly deduced from Eq. 3.3.9 as follows:

$$J_\pi^{\mathrm{H}}(\psi) = \mathbb{E}_{p_\theta(z_t)}\Big[\sum_{\tau=t}^{(t+H)/k} -\log \pi_\psi^{\mathrm{H}}(e_\tau|z_\tau)\,\mathrm{sg}(V_\psi^{\mathrm{H},\tau}) - \eta\,\mathcal{H}\big[e_\tau|z_\tau\big]\Big]. \tag{3.3.17}$$

Aggregating the actor-critic losses for the low-level and high-level policies as

$$J_{\mathrm{AC}}^{\mathrm{L}}(\psi) = J_\pi^{\mathrm{L}}(\psi) + J_{\mathrm{V}}^{\mathrm{L}}(\psi)$$

and

$$J^{\mathrm{H}}_{\mathrm{AC}}(\psi) = J^{\mathrm{H}}_{\pi}(\psi) + J^{\mathrm{H}}_{\mathrm{V}}(\psi)$$

respectively, the final training objective for an *HWM HRL Dreamer* agent can be expressed as:

$$J_{\text{HWM-HRL}}(\theta, \phi, \psi) = J_{\text{HWM}}(\theta, \phi) + J_{\mathrm{R}}(\theta, \phi) + J^{\mathrm{L}}_{\mathrm{AC}}(\psi) + J^{\mathrm{H}}_{\mathrm{AC}}(\psi) \tag{3.3.18}$$

3.4 Experimental setting

The experiments are grounded in one of the representative tasks of the HRL setting referred to as *Four Rooms* [108], illustrated in Fig. 3.3 (a). This study was built on top of the publicly available implementation referred to as the *MiniGrid-FourRooms-v0* environment, provided by [11]. By default, this environment provides RGB images as observations to the RL agent.

In *Four Rooms*, the agent (red triangle) illustrated in Fig. 3.3) (a) has to reach the exit (green tile) within a maximal episode length of $H = 100$ steps. The layout of the maze, which determines the position of the *doors* is fixed across all episodes. Both the starting position of the agent and the goal are set randomly at the beginning of each episode. The action space is simplified to three actions: *turn left*, *turn right*, and *move forward*.

For this study, let us consider a *decomposed state representation* of s_t, illustrated in Fig. 3.1 (d). Namely, the state of the whole maze can be efficiently described using 3 random variables. First, let G encode the room of the goal, as well as the relative

x and y coordinates of the goal in said room. Next, the agent position in the maze can be decomposed into the room of the agent, denoted as the random variable R^A, and the variable W that encodes the relative x and y coordinates of the agent in R^A, as well as the direction it is facing. This is derived from the observation that the information encoded by G is fixed across an episode, while the room of the agent R^A changes less frequently when compared to the x and y coordinates, or the direction of the agent encoded in W. Notice that this compact representation can be matched with the two-level hierarchy of the proposed HWM as illustrated in Fig. 3.1 (c), by collapsing both G and R^A into the variable Z, because they both change at a slower pace than W. This is motivated by the experimental results of [88], stipulating that in a hierarchically structured recurrent state-space model, slowly changing information in the observations is encoded into higher levels of the latent variable hierarchy. Meanwhile, fast-changing components are encoded at the lower levels.

The first experimental phase is designed to demonstrate how the proposed HWM captures adaptable temporally abstracted state dynamics. To this end, an RL agent instantiated as Proximal Policy Optimization (PPO) algorithm [95] is trained using the reference implementation provided in the *Stable Baselines 3* RL algorithm library [80]. The pre-trained PPO agent is used to generate a dataset of $25,000$ observation-action pairs (o_t, a_t), corresponding to $1,736$ distinct episode trajectories. This dataset is then used to train an instance of the proposed HWM without an actor-critic component, to maximize the objective function $J_{\text{HWM}(\theta,\phi)}$ derived in Eq. 3.3.5.

The second experimental phase aims to investigate whether or not the proposed HWM is a valid world modeling method, and if it provides any performance or sample efficiency improvement over classical world modeling methods. Using the

Dreamer agent [33, 35] as baseline, this chapter investigates two variants based on the proposed HWM. Namely, the first variant denoted as *HWM Dreamer* leverages the same configuration as the *Dreamer* baseline agent, while substituting standard the vanilla world-modelwith the proposed HWM as representation learninig component, as described in Section 3.3.3. In other words, the *HWM Dreamer* is used to isolate the contribution of the proposed HWM itself. The second variant hereafter referred to as *HWM HRL Dreamer* combines the hierarhical state representation learned by the proposed HWM and the extended HRL agent described in Section 3.3.4. This variant implements the *temporally abstract*, high-level decision-making inspired by that of animal and human cognition, as motivated in Section 3.2.2, and is used to investigate the benefit or lack thereof of abstract decision-making and temporally abstract exploration in model-based HRL agents. Due to computational limitations, the agents are trained for $2,000,000$ environment steps, corresponding to $400,000$ model and actor-critic update steps for *MiniGrid-FourRoooms-v0*.

3.5 Results

3.5.1 Hierarchically structured state representation

Recall that the purpose of the HWM is to provide (1) a *temporally abstract state representation* summarizing an arbitrary number of lower-level states while at the same time filling the role of a world model [29, 33, 35]. Additionally, the model should also provide (2) an adaptive temporal abstraction mechanism to divide trajectories into coherent sub-sequences [48].

To evaluate the proposed method, a trained instance of the proposed HWM following the setting described in Section 3.4 is fed trajectories of observation-action pairs (o_t, a_t). For each trajectory, the first temporally abstract state z_0 is inferred using the learned posterior $q_\phi(z_t)$. For $t > 0$, the temporal transition is modeled using the learned $p_\theta(z_t)$. The lower-level state w_t at each step is inferred using the posterior $q_\phi(w_t)$, identically to Dreamer agents [33, 35]. The boundary indicator m_t is estimated using the learned prior distribution $p_\theta(m_t)$. Finally, each observation o_t is reconstructed using the learned decoder $p_\theta(o_t|w_t, z_t)$.

From lines (a) and (e) in Fig. 3.4, it can be seen that the HWM manages to accurately reconstruct the provided observations, while modeling the high and low-level state transitions, thus satisfying the requirement (1). Requirement (2) is also satisfied, as the prior boundary indicator $p_\theta(m_t)$ can accurately predict the change in the room of the agent. This is indicated in Fig. 3.4 by a change of color at the next step every time the prior boundary indicator detects a new segment. In this specific case, the proposed trajectory segmentation also coincides with the intuited abstracted dynamics that were derived for the *Four Rooms* task, as illustrated in Fig. 3.1 (d). Through the lines (b), (c), and (d) in Fig. 3.4, this study aims to illustrate what part of the reconstructed observation each of the latent variables z_t and w_t is responsible for. From (b), it can be observed that only the layout of the maze is reconstructed when zeros are passed as input to the decoder $p_\theta(o_t|w_t, z_t)$. Moreover, there are smeared traces of red color over the mazes, reminiscent of the agent depiction. The agent itself, however, is missing in the reconstructed observation.

When conditioning the decoder on the high-level state z_t only, the goal tile, as well as a blurry depiction of the agent, appear in the reconstructions, illustrated by line (c).

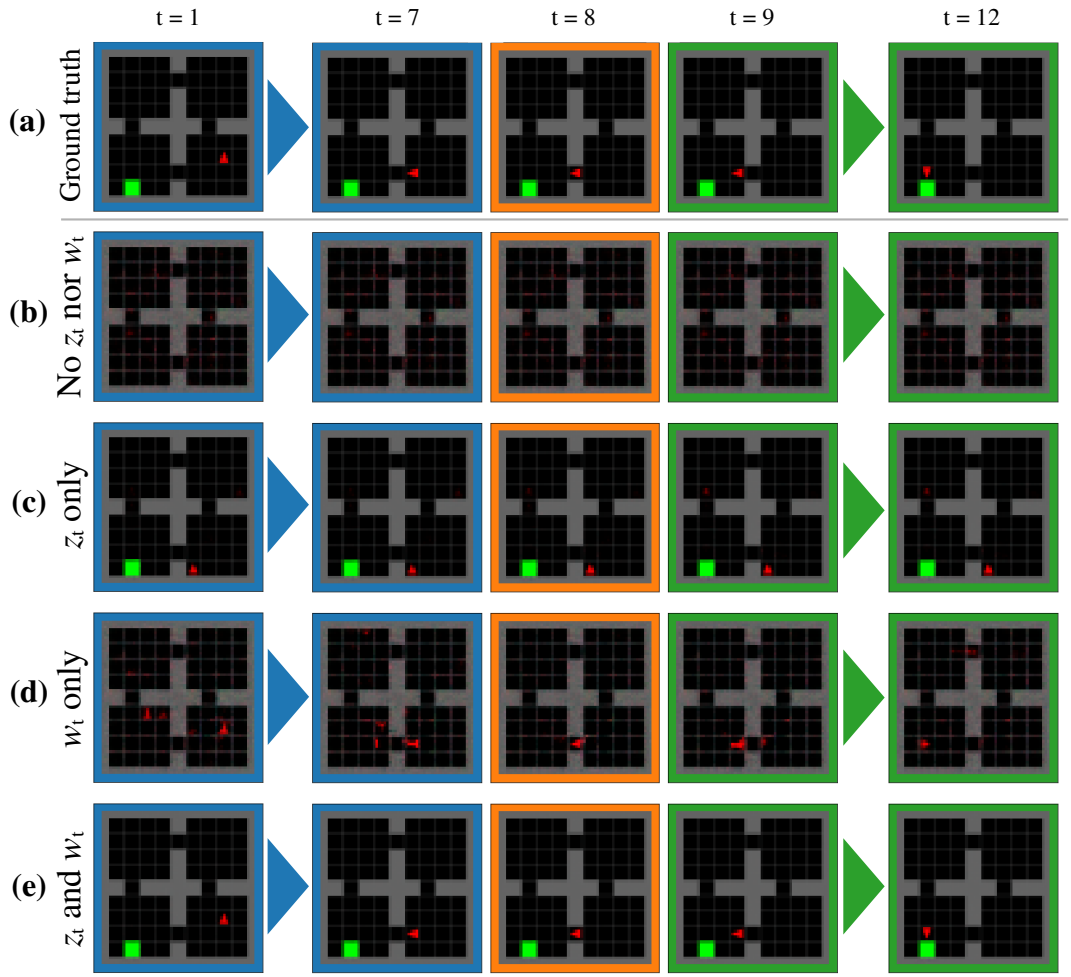


Fig. 3.4: Illustrative example of the reconstruction and segmentation of an arbitrary trajectory performed by the proposed HWM. Each color represents a segment that corresponds to a temporally abstract state s_t. The segmentation is performed using the learned prior boundary distribution $m_t \sim p_\theta(m_t)$. Intermediate steps of long segments are omitted. Row (a) shows the ground truth observation of the trajectory. From row (b) to (e), the hierarchically structured components z_t and w_t of the HWM's state belief are progressively turned on just before reconstructing the observed frame.

On the other hand, when conditioning the decoder solely on w_t, not only is the goal absent from the frame but the position of the agent in the maze becomes ambiguous,

as illustrated in line (d).

These results suggest that the information about the goal and the room of the agent tend to be encoded at the higher level by z_t, while other, more *refined* information is encoded at the lower level by w_t. Notice also how the traces of red are present in rows (b) and (d) become absent once z_t has been incorporated in row (c). The red traces could be attributed to the ambiguity regarding the information about the agent's position in the state representation used for the reconstruction. This further strengthens the observation that z_t also encodes information that contributes to precisely situating the agent in the maze, which is also corroborated by [88].

One caveat would be that the role of encoding information about the agent seems to be shared by the combination of w_t and z_t. Namely, on line (c) at time step $t = 8$, the blurry agent is depicted in the room corresponding to the previous segment of $t \in \{1, ..., 7\}$. Concurrently, on line (d) at time steps $t \in \{7, 8, 9\}$, the agent is localized in the correct room, albeit with a less defined depiction. This could be solved by further refining the HWM's structure and using methods such as [40, 125] to better disentangle the information captured toward different levels of the hierarchy.

3.5.2 Sample efficiency and performance improvement

Fig. 3.5 documents the averaged episodic return across four seeds for *MiniGrid-FourRooms-v0*. First, the *HWM Dreamer* (green line) that uses the proposed HWM instead of the reference world model [33, 35] demonstrates a non-negligible gain in sample efficiency, when compared to the baseline *Dreamer* agent (blue line). Namely, the added constraint of learning a hierarchically structured state representation incentivizes the HWM to disentangle correlated aspects of the task's dynamics into

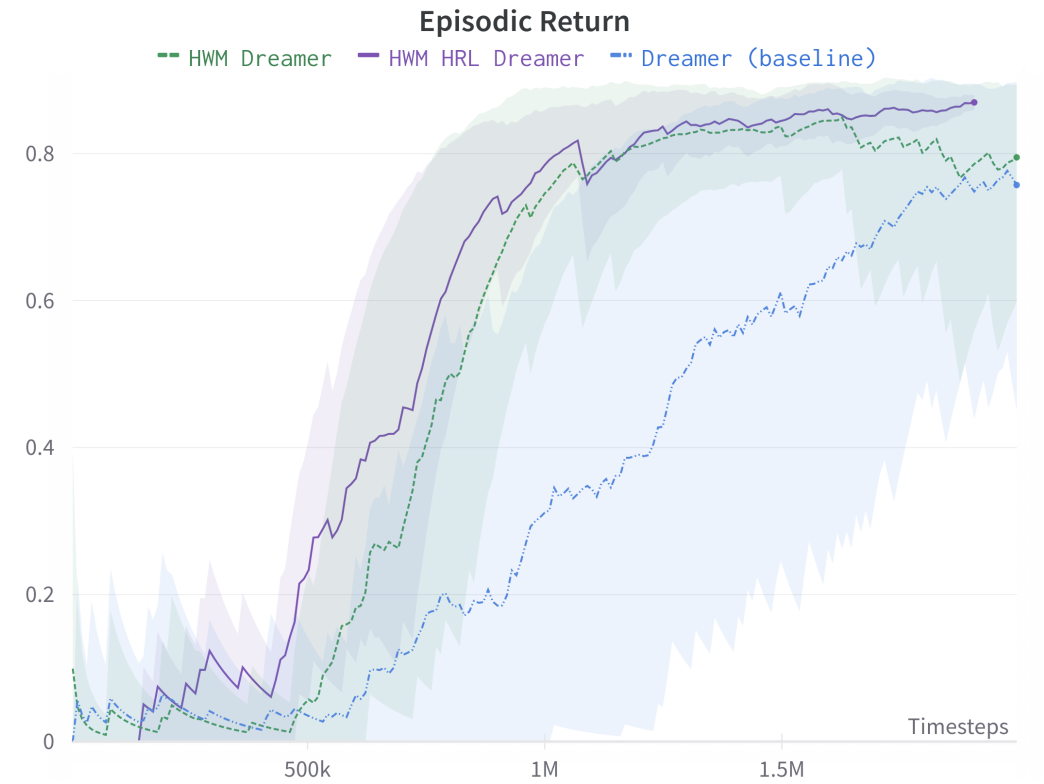


Fig. 3.5: Averaged episodic return of each agent variant across 4 seeds. The vertical axis holds the episodic return values, while the horizontal one shows the number of time steps (environment interactions).

different (hierarchical) levels of the state feature vector. This in turn allows the downstream critic and policy deep neural networks to estimate the optimality of the value and action more effectively than conventional representation learning in the reference world modeling methods [34, 35], resulting in a drastic improvement in sample efficiency compared to the *Dreamer* baseline. Moreover, it also has comparable performance with the Dreamer baseline.

This would suggest that owing to its hierarchically structured internal state representation, the proposed HWM further learns a state representation that accelerates

and stabilizes the learning of the actor-critic component.

Next, the proposed *HWM HRL Dreamer* (purple line) that uses both the HWM and the model-based HRL decision-making proposed in 3.3.4 demonstrates further improvement in final performance as well as sample efficiency when compared to both *HWM Dreamer* and the Dreamer baseline. This can be attributed to the benefit of temporally abstract exploration that allows the high-level policy to better guide the lower level across the maze, as intuited in 3.3. However, given the relative simplicity of the *MiniGrid-FourRooms-v0* investigated, the gain in sample efficiency from high-level exploration is limited, leading to a marginal improvement of *HWM HRL Dreamer* over *HWM Dreamer*. The gain in sample efficiency of the former is expected to increase proportionally to the difficulty of the task. For example, in mazes with larger rooms, higher count of rooms, or generally more intricate labyrinth layouts, exploring at the temporally abstract level is expected to yield better sample efficiency compared to baseline methods, which is a promising avenue to explore in future works.

3.6 Conclusion

This study leveraged the recent progress in world modeling methods and the framework of variation temporal abstraction to propose the hierarchical world model (HWM). The proposed model captures both temporally abstract and granular dynamics, as well as the corresponding hierarchically structured state representations.

Owing to its design, the HWM maintains the ability of world model methods [29, 33–35] to provide a compact state representation that is also hierarchically structured for standard MbRL agents. This was verified in our preliminary experiments, which

demonstrated an increase in sample efficiency and final performance for a Dreamer agent that used the proposed HWM instead of its original world model component. A complementary set of experiments also show that the proposed HWM can learn semantically meaningful, temporally abstract state representations that happen to align with human understanding of the task to solve. Empirically, the proposed *HWM HRL Dreamer* agent was demonstrated to achieve higher sample efficiency and maintains stable final performance when compared to the baseline trained under the same experimental setting. Additionally, this chapter also demonstrates how the adaptive temporal abstraction mechanism provided by the HWM can be extended onto an HRL agent, thus building toward a human cognition-based, hierarchically structured decision-making process.

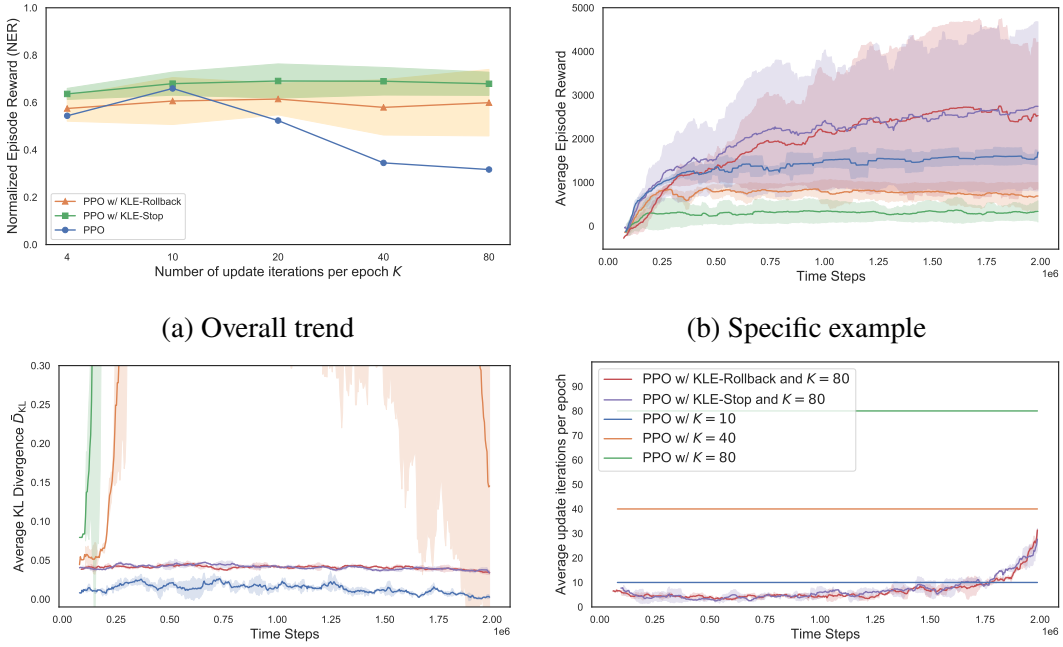
As previously motivated in Chapter 1 and 2, while imitating the behavior of human experts based on available demonstration can yield high-performing agents that can still behave predictably from the human perspective, it is far from sufficient when deploying RL agents on real-world tasks. Namely, besides their qualitative requirements, one practical limitation of RL agents is the efficiency of the training pipeline itself. Therefore, training satisfactory enough RL agents can require computational resources and training time proportional to the complexity of the task that society might benefit from automating via RL. The deployment of RL methods in real-world, high-impact tasks would thus benefit from any gain in efficiency at a lower level of the training process. Consequently, the next chapter dives into the investigation of low-level implementation practice of trust region-based RL methods, to further improve the efficiency, reliability, and robustness of the trained agents.

Chapter 4 An Empirical Investigation of Early Stopping Optimization in Proximal Policy Optimization

4.1 Introduction

The plethora of DRL method developed in the recent has been accompanied by a broad variety of implementations framework and techniques [62, 67, 94, 95]. One caveat stems from the freedom in implementation and the openness to interpretation of most DRL methods is that the same algorithm can yield drastically different results, depending on the structure of the implementation or even the software framework used.

Recent work by Engstrom, Ilyas et al. [22] suggests that code-level optimizations, which are optimization techniques employed in the implementation of a given algorithm that often does not appear on the published pseudo-code, could be fully responsible



(a) Overall trend

(b) Specific example

(c) KL Explosion

(d) Dynamic versus static K

Fig. 4.1: (a) shows the Normalized Episode Reward (NER) of PPO, PPO with KLE-Stop, and PPO with KLE-Rollback for varying K, averaged over 11 tasks of robotic control; its shaded region represents the variance against different d_{target}. As a specific example in Halfcheetah-v2, (b) demonstrates that the KLE-Stop and KLE-Rollback are less susceptible to degenerate performance when K is high, (c) illustrates the KL explosion phenomenon that occurs when K is set too high for the PPO algorithm, and (d) shows the average update iteration per epoch; early-stopping optimizations dynamically adjust K based on the mean KL divergence $\bar{D}_{\text{KL}}$ between the target and current policy.

for a significant portion of the algorithm's performance improvement, pointing out that these optimizations require more attention from the research community. This chapter delves into the study of the effect of one such optimization that is initially referred to as "early stopping" [1] in the *openai/spinningup* library's implementation of the Proximal Policy Optimization (PPO) algorithm [95]. The key idea of early stopping methods is to measure how much is the policy changing with each update

(using the *Kullback-Leibler* Divergence [53]), and prevent updates that change the policy too abruptly. Hereafter, the standard version of early stopping is referred to as *KLE-Stop*, while using (KL Enforcement-Stop) to refer to the early stopping which stops any further updates when KL divergence between policy updates goes over a predefined threshold. Following a thorough survey of related works, the KLE-Stop concept appeared to not have been discussed in any existing literature. Intuitively, KLE-Stop is a flexible method to dynamically determine the best "number of update iterations per epoch" (K) to apply to the policy network at each epoch, based on a heuristic. KLE-Stop is interesting, since existing published work does not provide any insights regarding the choice of the K hyper-parameter for PPO, and the values used differ across publications and implementations. For example, Schuman et al. [95] uses $K = 10$ for Mujoco, and $K = 15$ for Roboschool, $K = 4$ for Atari environments, while *openai/spinningup* uses a surprising $K = 80$ coupled with KLE-Stop for Mujoco.

More specifically, the KLE-Stop technique works as follows: after every update iteration applied to the policy network in PPO, the average Kullback–Leibler (KL) divergence [53] $\bar{D}_{\mathrm{KL}}$ between the target policy and current policy is measured. If such divergence is greater than a preset threshold d_{target}, KLE-Stop will stop future policy updates. The motivation behind KLE-Stop is clearly to make sure $\bar{D}_{\mathrm{KL}}$ does not get any larger by continuing to perform gradient steps if $\bar{D}_{\mathrm{KL}} > d_{\mathrm{target}}$. However, notice KLE-Stop does not guarantee that $\bar{D}_{\mathrm{KL}} \leq d_{\mathrm{target}}$. In an unlikely situation, the last gradient step could make $\bar{D}_{\mathrm{KL}}$ to be significantly larger than d_{target}. Therefore, a more conservative variant dubbed *KLE-Rollback*, which rolls back to the policy just before the threshold was crossed is also included in the experimental study.

Experiments are conducted with PPO that is augmented with all the code-level

optimization implemented in *openai/baselines* to study if early stopping optimizations (KLE-Stop and KLE-Rollback) could improve PPO's performance, robustness, and sample efficiency. The main findings of those experiments are as follows:

1. **PPO's sensitivity to K:** PPO's performance was found to be highly sensitive to the hyper-parameter "number of update iterations per epoch" K. Fig. 4.1a shows the overall trend of the performance of PPO with varying K. From $K = 4$ to $K = 20$, the Normalized Episode Reward (NER), averaged over 11 tasks, of PPO increases progressively until reaching the peak. From $K = 40$ to $K = 80$, however, NER decreases as K increases. This phenomenon is referred to as "catastrophic unlearning", which occurs when the number of policy network updates becomes too high (see Fig. 4.1b as an example, where agents trained with PPO, $K = 80$ learn some useful policy in the beginning then quickly degenerate). In addition, it can also be observed that the drop in performance exhibited by PPO agents with $K = 40$ and $K = 80$ is highly correlated with the "KL explosion" that can be observed in Fig. 4.1c.

2. **Early stopping optimizations mitigate such sensitivity:** Even when K is large, early stopping optimizations overall avoid the "catastrophic unlearning" that happens in the absence of early stopping optimizations. As shown in Fig.4.1a, the agents trained with $K = 20$, $K = 40$, $K = 80$, and early stopping optimizations are shown to reach comparable performance to agents trained with $K = 10$ and $K = 20$ without early stopping optimizations. This is likely due to the dynamic adjustment of K at each epoch. As shown in Fig. 4.1d, the actual number of update iterations per epoch for PPO with early stopping

optimizations could be small (about 3-7) at the beginning of training. At the late stages of training, the code-level optimization of learning rate annealing makes the mean KL divergence between policy updates small, and therefore the actual update iterations per epoch become higher.

3. **Early stopping optimizations could be an alternative to tuning on K:** As observed in Fig. 4.1a, a small-to-medium K (e.g. $K = 10$ or $K = 20$) generally produce good performance with or without early stopping optimizations for the tasks experimented upon. However, it is not always possible to know the best K for the desired task. Early stopping optimizations intuitively serve as a convenient replacement to tuning on K. Namely, the user could set a large K and early stopping optimizations will help to automatically determine the best number of update iterations for each epoch. In addition, the shaded regions in Fig. 4.1a show the variance of early stopping optimization with varying d_{target}, showing the early stopping optimization is less sensitive to its hyper-parameter d_{target} than PPO is to K.

The remainder of this chapter is structured as follows. Section 4.2 presents the background on policy gradient algorithms. Section 4.3 elaborates on early stopping optimizations and lists out other code-level optimizations found in popular DRL libraries, which were used in this study. Section 4.4 details the experiments conducted to study the effectiveness of early stopping optimizations, Section 4.5 presents the experiment results and discussion, and Section 4.6 concludes and discusses future work.

4.2 Preliminaries

This section introduces the background on Proximal Policy Optimization algorithms [95], used in the subsequent experiments.

4.2.1 Policy Gradient algorithms

Let us recall the definition of the standard Reinforcement Learning problem in the setting of Markov Decision Process (MDP) denoted as $(\mathbb{S}, \mathbb{A}, P, \rho_0, r, \gamma, T)$, with the set of states $\mathbb{S}$, the set of actions $\mathbb{A}$, the state transition probability $P : \mathbb{S}\times\mathbb{A}\times\mathbb{S} \to [0, 1]$, the initial state distribution $\rho_0 : \mathbb{S} \to [0, 1]$, the reward function $r : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, the discount factor is γ, and the maximum episode length T. A stochastic policy $\pi_\theta : \mathbb{S} \times \mathbb{A} \to [0, 1]$, parameterized by a parameter vector θ, defines the probability of an action given a state. The typical objective is to maximize the expected discounted return:

$$J = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \gamma^t r_t \right], \tag{4.2.1}$$

where τ is the trajectory $(s_0, a_0, r_0, s_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1})$, $s_t \sim P(\cdot|s_{t-1}, a_{t-1})$, $s_0 \sim \rho_0$, $a_t \sim \pi_\theta(\cdot|s_t)$, and $r_t = r\,(s_t, a_t)$.

The notation $P(\cdot|s_{t-1}, a_{t-1})$ represents the states transition distribution given the previous state s_{t-1} and action a_{t-1}, and the notation $s_t \sim P(\cdot|s_{t-1}, a_{t-1})$ represents that the state s_t visited at time t is sampled from $P(\cdot|s_{t-1}, a_{t-1})$. Similarly, $\pi_\theta(\cdot|s_t)$ represents the action distribution given state s_t, and $a_t \sim \pi_\theta(\cdot|s_t)$ means the action a_t at time t is sampled from $\pi_\theta(\cdot|s_t)$.

The policy gradient algorithms aim to maximize the expected discounted rewards

by doing gradient ascent $\theta = \theta + \nabla_\theta J$, where $\nabla_\theta J$ is the *policy gradient* of the expected discounted return with respect to the policy parameter vector θ. Earlier work proposed the following policy gradient estimate to the objective J [106, 107]:

$$\mathbb{E}_\tau \left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) \sum_{k=0}^{\infty} \gamma^k r_{t+k} \right], \tag{4.2.2}$$

This gradient estimate, however, suffers from large variance [106] and the following gradient estimate is suggested instead:

$$\mathbb{E}_\tau \left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) A(\tau, V, t) \right], \tag{4.2.3}$$

where $A(\tau, V, t)$ is the General Advantage Estimation (GAE) [93], which measures "how good is a_t compared to the usual actions", and $V : S \to \mathbb{R}$ is the state-value function. The use of GAE has been shown to significantly improve variance and yield strong empirical results [93].

4.2.2 Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) [94] is a recent algorithm introduced by Schulman et al. to (1) allow the policy update to reuse trajectory generated by the current policy $\pi_{\theta_{old}}$ parameterized by θ_{old} to improve sampling efficiency via importance sampling, and (2) constrain the amount of policy change (i.e. ensure the trust region) to improve stability via KL divergence. To this end, TRPO updates the

policy by solving the following optimization problem with constraints:

$$\max_{\theta} \quad \mathbb{E}_{\tau}\left[\sum_{t=0}^{\infty} \gamma^{t} \rho_{t}(\theta) A(\tau, V, t)\right] \tag{4.2.4}$$

$$\text{s.t.} \quad \mathbb{E}_{s \sim d^{\pi_{\theta_{old}}}}\left[D_{\mathrm{KL}}\left(\pi_{\theta_{\mathrm{old}}}(\cdot|s) \| \pi_{\theta}(\cdot|s)\right)\right] \leq \delta \tag{4.2.5}$$

where $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the importance sampling ratio between the target policy π_θ and the behavior policy $\pi_{\theta_{old}}$, $d^{\pi_{\theta_{old}}}$ is the on-policy state distribution under $\pi_{\theta_{old}}$, and D_{KL} measures the KL divergence between distributions. The practical algorithm to solve this constrained optimization problem of TRPO involves natural gradient descent, which could be computationally expensive and difficult to implement.

4.2.3 Proximal Policy Optimization

To alleviate the complexity of TRPO, Schulman et al. proposed the Proximal Policy Optimization (PPO) algorithm [95] with a simpler clipped surrogate objective:

$$\max_{\theta} \mathbb{E}_{\tau}\left[\sum_{t=0}^{\infty}\left[\min\left(k \rho_{t}(\theta) A_{\pi_{\theta}}\left(s_{t}, a_{t}\right), \operatorname{clip}\left(\rho_{t}(\theta), 1-\varepsilon, 1+\varepsilon\right) A(\tau, V, t)\right)\right]\right], \text{ with} \tag{4.2.6}$$

$$\text{clip}\left(\rho_t(\theta), 1-\varepsilon, 1+\varepsilon\right) = \begin{cases} 1-\varepsilon & \text{if } \rho_t(\theta) < 1-\varepsilon \\ 1+\varepsilon & \text{if } \rho_t(\theta) > 1+\varepsilon \\ \rho_t(\theta) & \text{otherwise} \end{cases}$$

In particular, the objective of PPO removes the constraint in Equation 4.2.5 and uses the clipped importance sampling ratio clip $(\rho_t(\theta), 1-\varepsilon, 1+\varepsilon)$ to encourage the target policy to stay within a reasonable trust-region of the behavior policy. The optimization process thus consists in maximizing the simplified objective of PPO with respect to the policy parameter vector θ, as in Eq. 4.2.6. Overall, PPO has been shown to outperform classical TRPO in several tasks, while being a simpler and faster algorithm.

4.3 Code-Level Optimizations

In addition to the base algorithm above, there are many code-level optimizations usually implemented to enhance the performance and stability [22] of the PPO algorithm. These optimizations include various normalization and clipping techniques for the input data as well as for gradient updates. A full list of these optimizations can be found in Section 4.3.2. Engstrom, Ilyas et al. [22] find these optimizations to be critical to the performance of PPO. They showed that without these code-level optimizations, PPO fails to stay within the trust region and performs much worse than with these optimizations. Empirically, they also performed an ablation study and demonstrate that the clipped objective, which is the key feature under PPO's theory, is not responsible

for PPO's performance improvement over TRPO, at least not by itself.

4.3.1 Early stopping optimizations

Since PPO removes the constraint in Equation 4.2.5, the clipped objective of PPO only gives incentives but no guarantees for the desired KL constraints. Empirically, the KL Divergence over time produced by PPO without any code-level optimizations grows *exponentially* [22]. During the reproduction study of the PPO algorithm, the existence of a code-level optimization not mentioned in existing literature, namely, "Early Stopping" (KLE-Stop) implemented in *openai/spinningup* but not in *openai/baselines* came to light. Instead of solely relying on the clipped surrogate objective introduced in PPO [95], KLE-Stop first measures the mean KL divergence $\bar{D}_{\text{KL}}$ between the target policy at update iteration k (denoted as π_{θ_k} hereafter), and the current policy $\pi_{\theta_{old}}$ based on the states sampled from the minibatch $\mathcal{M}$:

$$\bar{D}_{\text{KL}}\left(\theta_k, \theta_{old}\right) = \mathbb{E}_{s\sim\mathcal{M}}\left[D_{\text{KL}}\left(\pi_{\theta_k}(\cdot|s) \| \pi_{\theta_{old}}(\cdot|s)\right)\right] \tag{4.3.1}$$

If $\bar{D}_{\text{KL}}$ is greater than some desired KL divergence value d_{target}, then the update is deemed to be *outside* of the trust region. More specifically, let us consider two different implementations of this concept (Algorithm 4.1):

1. **KLE-Stop.** When $\bar{D}_{\text{KL}}(\theta_k, \theta_{old}) > d_{target}$, the algorithm stops any further update steps on the current batch and continues to generate new batches and perform updates. This behavior is the default implementation found in *openai/spinningup*.

2. **KLE-Rollback.** When $\bar{D}_{\mathrm{KL}}(\theta_k, \theta_{old}) > d_{target}$, the algorithm rolls back to the state of the policy before the update iteration k, then continues to generate new batches and perform updates. A study of KLE-Rollback was deemed relevant because KLE-Stop, by design, will result in new policies that are right outside of the trust region, which may or may not be desirable.

Intuitively, both early stopping optimizations present *algorithmic benefits* to PPO. Namely,

1. **Strong Enforcement of Trust Region.** The early stopping optimizations do not just give incentives for letting agents remain in the trust region. Rather, they make sure that the agents stay close to the trust region (if using KLE-Stop) or stay within the trust region (if using KLE-Rollback). This enforcement behavior could be especially desirable when the agents are stuck in local minima. The early stopping optimizations thus effectively ensure that the policy change of the agent stays within the target KL divergence threshold d_{target}.

2. **Faster Execution.** Since early stopping optimizations could early abort the update of the policy network weights, and immediately continue to generate new batches, they marginally reduce the wall time required for the whole training.

Algorithm 4.1 summarizes the training procedure of PPO while emphasizing the two early stopping mechanisms investigated.

4.3.2 Other Code-level Optimizations

Including KLE, which is the focus of this chapter, the full list of the code-level optimizations that are found in popular DRL libraries (e.g. *openai/baselines*, *ope-*

Algorithm 4.1 PPO with Early Stopping Optimization

while Total number of steps $\leq N_T$ **do**
  Initialize batch storage $\mathcal{D}$ of size N
  Run episode $1, 2, ..., M$ to fill up $\mathcal{D}$ according to $\pi_{\theta_{old}}$
  Let $\theta_0 \leftarrow \theta_{old}$
  for Update iteration $k = 1, 2, ..., K$ **do**
    for $\mathcal{M} \subseteq \mathcal{D}$ **do**
      Let θ_k be the policy parameter obtained through policy update on θ_{k-1} given $\mathcal{M}$
    end for
    Measure $\bar{D}_{\text{KL}}(\theta_k, \theta_{old})$ based on the last $\mathcal{M}$
    if $\bar{D}_{\text{KL}}(\theta_k, \theta_{old}) > d_{target}$ **then**
      break If KLE-Stop is used
      break and $\theta_k \leftarrow \theta_{k-1}$ if KLE-Rollback is used
    end if
  end for
  $\theta_{old} \leftarrow \theta_k$
end while

nai/spinningup), their corresponding description, and the permanent links to the files where these optimizations were found are as follows:

1. **"Early Stopping" (KLE-Stop)[1]:** As discussed above.

2. **Normalization of Advantages[2]:** After calculating the advantages based on GAE, the advantages vector is normalized by subtracting its mean and divided by its standard deviation.

3. **Normalization of Observation[3]:** The observation is pre-processed before feeding to the PPO agent. The raw observation was normalized by subtracting

[1] https://github.com/openai/spinningup/blob/038665d62d569055401d91856abb287263096178/spinup/algos/pytorch/ppo/ppo.py#L269-L271

[2] https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/model.py#L139

[3] https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/common/vec_env/vec_normalize.py#L4

its running mean and divided by its variance; then the raw observation is clipped to a range, usually $[-10, 10]$.

4. **Rewards Scaling[4]:** Similarly, the reward is pre-processed by dividing the running variance of the discounted the returns, following by clipping it to a range, usually $[-10, 10]$.

5. **Value Function Loss Clipping[5]:** The value function loss is clipped in a manner that is similar to the PPO's clipped surrogate objective:

$$V_{loss} = \max\left[\left(V_{\theta_t} - V_{targ}\right)^2, \left(V_{\theta_{t-1}} + \text{clip}\left(V_{\theta_t} - V_{\theta_{t-1}}, -\varepsilon, \varepsilon\right) - V_{targ}\right)^2\right]$$

where V_{targ} is calculated by adding $V_{\theta_{t-1}}$ and the GAE $A(\tau, V)$.

6. **Adam Learning Rate Annealing[6]:** The Adam [49] optimizer's learning rate is set to decay as the number of timesteps agent trained increase.

7. **Mini-batch updates[7]:** The PPO implementation of the *openai/baselines* also uses mini-batches to compute the gradient and update the policy instead of the whole batch data such as in *open/spinningup*.

[4] https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/common/vec_env/vec_normalize.py#L4

[5] https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/model.py#L68-L75

[6] https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/ppo2.py#L135

[7] https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/ppo2.py#L160-L162

8. **Global Gradient Clipping[8]:** For each update iteration in an epoch, the gradients of the policy and value network are clipped so that the "global ℓ_2 norm" (i.e. the norm of the concatenated gradients of all parameters) does not exceed 0.5.

9. **Orthogonal Initialization of weights[9]:** The weights and biases of fully connected layers use with orthogonal initialization scheme with different scaling. For the experiments in this study, however, the scaling value of 1 for historical reasons.

10. **Hyperbolic tangent activations[10]**: The activation functions of the hidden layers are always set to use the hyperbolic tangent function.

4.4 Experiments

The section presents the details of the experiments conducted to study early stopping mechanisms. As the control group, the performance of PPO was evaluated using $K \in \{4, 10, 20, 40, 80\}$. As the experimental group, the performance of PPO augmented with early stopping optimizations using $d_{\text{target}} \in \{0.015, 0.020, 0.030, 0.050\}$ and $K \in \{4, 10, 20, 40, 80\}$ was evaluated. All of the experiments were conducted using 2 random seeds and 11 tasks of robotics control (Reacher-v2, Pusher-v2, Thrower-v2, Striker-v2, InvertedPendulum-v2, HalfCheetah-v2, Hopper-v2, Swimmer-v2, Walker2d-v2, Ant-v2, and Humanoid-v2). For other hyper-parameters, the default

[8] https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/ppo2/model.py#L107

[9] https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/a2c/utils.py#L58

[10] https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/common/models.py#L75

settings from *openai/baselines* were used. The detailed list of hyper-parameters used is documented in Table 4.1.

Table 4.1: The list of experiment parameters and their values.

Parameter Names	Parameter Values
Total Time steps	2,000,000
γ (Discount Factor)	0.99
λ (for GAE)	0.97
η (Entropy Regularization Coefficient)	0.2
ε (PPO's Clipping Coefficient)	0.2
ω (Gradient Norm Threshold)	0.5
α_π Policy's Learning Rate	0.0003
α_v Value Function's Learning Rate	0.0003

For a more intuitive analysis and comparison across tasks, the Normalized Episode Reward (NER) was used. The NER can is obtained as follows: let $r^E_{\min}$ and $r^E_{\max}$ respectively stand for the minimum and maximum episode reward received across all runs for a given task E, the NER for E based on actual episode reward r^E_{actual} is obtained as follows:

$$r^E_{normalized} = \frac{r^E_{actual} - r^E_{\min}}{r^E_{\max} - r^E_{\min}}.$$

4.5 Results and Discussion

The average NER of the last 50 episodes and 11 tasks is reported in Table 4.2. In addition, the average number of actual update iterations for early stopping optimizations of the first 500,000 timesteps and 11 tasks is also reported in Table 4.3. The motivation for using 500,000 timesteps for evaluating the actual number of update iterations is that the code-level optimization of learning rate annealing would make $\bar{D}_{\mathrm{KL}}(\theta_k, \theta_{old})$

small and therefore resulting in an abnormally large number of update iterations in late stages of training as shown in Fig. 4.1d. For the sake of objective completeness, the episode rewards obtained by PPO with $K = 10$ without early stopping optimizations are also provided in Table 4.4 in the Section 4.5.4, which matches the performance from published sources [95]. Below are the major findings of this chapter:

Table 4.2: The average normalized episode reward over the last 50 episodes and over 11 popular tasks.

	4	10	20	40	80
PPO	0.54	**0.65**	0.52	0.34	0.31
PPO w/ KLE-Rollback, $d_{\text{target}} = 0.015$	0.53	0.43	**0.58**	0.52	0.57
PPO w/ KLE-Rollback, $d_{\text{target}} = 0.02$	0.50	0.62	0.57	**0.65**	0.57
PPO w/ KLE-Rollback, $d_{\text{target}} = 0.025$	0.60	0.52	0.60	0.60	**0.72**
PPO w/ KLE-Rollback, $d_{\text{target}} = 0.03$	0.59	0.70	**0.72**	0.67	0.69
PPO w/ KLE-Rollback, $d_{\text{target}} = 0.05$	0.66	0.70	0.67	0.66	**0.71**
PPO w/ KLE-Stop, $d_{\text{target}} = 0.015$	**0.61**	0.61	0.60	0.61	0.59
PPO w/ KLE-Stop, $d_{\text{target}} = 0.02$	0.61	0.62	0.68	**0.70**	0.66
PPO w/ KLE-Stop, $d_{\text{target}} = 0.025$	0.64	**0.73**	0.68	0.64	0.72
PPO w/ KLE-Stop, $d_{\text{target}} = 0.03$	0.67	0.72	0.65	**0.77**	0.67
PPO w/ KLE-Stop, $d_{\text{target}} = 0.05$	0.62	0.69	**0.82**	0.71	0.73

4.5.1 PPO's sensitivity to K

The agent's performance was found to be highly sensitive to the hyper-parameter "number of update iterations per epoch" K. On one hand, when K is small (e.g. $K = 4$), the agent "under learns" and its improvement is slow, on average reaching an episode reward that is 50.63% of the best episode reward achieved. On the other hand, when K is large (e.g. $K = 80$), the agent "catastrophically unlearns" useful policies, on average reaching an episode reward that is 20.08% of the best episode reward achieved, which

Table 4.3: Average number of update iterations for early stopping optimizations over 11 popular tasks within the first 500,000 time steps.

	4	10	20	40	80
PPO w/ KLE-Rollback (0.015)	2.943	3.974	4.374	4.586	4.572
PPO w/ KLE-Rollback (0.02)	3.204	4.807	5.635	6.149	6.420
PPO w/ KLE-Rollback (0.025)	3.420	5.594	6.892	7.865	8.628
PPO w/ KLE-Rollback (0.03)	3.624	6.192	8.080	9.588	10.53
PPO w/ KLE-Rollback (0.05)	3.922	8.292	11.82	15.45	18.09
PPO w/ KLE-Stop (0.015)	2.850	3.772	4.145	4.439	4.515
PPO w/ KLE-Stop (0.02)	3.146	4.590	5.370	5.799	6.184
PPO w/ KLE-Stop (0.025)	3.384	5.407	6.580	7.636	8.540
PPO w/ KLE-Stop (0.03)	3.584	6.063	7.799	9.245	10.56
PPO w/ KLE-Stop (0.05)	3.915	8.131	11.48	14.89	16.86

is even worse than when $K = 4$. In comparison, a well-tuned K (e.g. $K = 10$), which is the default setting for *openai/baselines* reaches 72.58% of the best episode reward achieved.

4.5.2 Early stopping optimizations mitigate such sensitivity

When K is large, early stopping optimizations overall helps the agent to avoid the "catastrophic unlearning" that happens without early stopping optimizations. The source of this mitigation is likely due to the reduced number of update iterations per epoch. As shown in Table 4.3, even when K is large such as 80, the actual update iterations done by agents with early stopping optimizations are within the range of $[6, 18]$, as per Table 4.3. Although such sensitivity is mitigated, note that the agents trained with early stopping optimizations with $K = 10$ perform as well, and sometimes even better than the default setting $K = 10$ without early stopping optimizations.

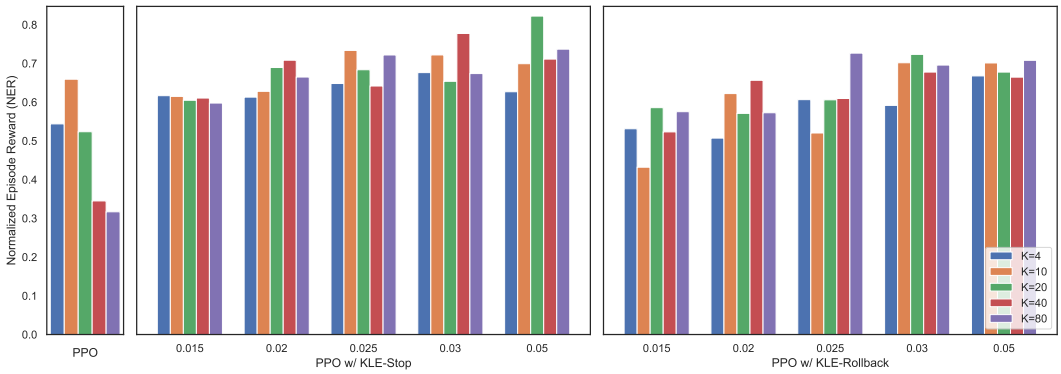


Fig. 4.2: Investigating the respective sensitivity of PPO with KLE-Stop (Middle) and KLE-Rollback (Right) to the d_{target} hyper-parameter. For easier comparison, the classical version with does not use KLE-based early stop is also provided and denoted as PPO (Left).

Table 4.4: The average episode reward of PPO without early stopping optimizations across 2 random seeds for different numbers of training updates K

	4	10	20	40	8
Ant-v2	1070.8	**3164.9**	445.8	45.5	3.6
HalfCheetah-v2	**2649.5**	1626.2	1470.3	714.3	284.3
Hopper-v2	706.9	**2442.5**	1565.4	1015.8	463.8
Humanoid-v2	**1655.4**	839.6	557.8	420.0	465.4
InvertedPendulum-v2	762.5	**1000.0**	881.5	**1000.0**	639.0
Pusher-v2	-54.3	**-32.0**	-39.0	-75.0	-69.0
Reacher-v2	-8.6	-7.9	**-5.8**	-14.1	-10.0
Striker-v2	-244.6	**-216.4**	-270.1	-292.2	-331.8
Swimmer-v2	100.2	**109.1**	116.8	85.5	81.6
Thrower-v2	-64.7	**-63.7**	-82.9	-68.1	-64.0
Walker2d-v2	**4203.4**	3742.1	1568.4	212.2	475.4

4.5.3 Early stopping optimizations as an alternative to tuning K

Although a small-to-medium K (e.g. $K = 10$ or $K = 20$) generally produces good performance with or without early stopping optimizations, obtaining such high-

performance K for the desired task is not always possible without much tuning. As an alternative to spending time on finding the optimal K, the users could conveniently use early stopping optimizations with a large K, and early stopping optimizations will automatically determine an appropriate K for each epoch. Although the agents' performance can still be affected by different d_{target}, early stopping optimizations appear to be less sensitive to d_{target} than PPO is to K. As an example, the agents trained with $K = 80$ and early stopping optimizations are shown to reach varying performance from 55.84% to 76.89% depending on the choice of d_{target} and selection of KLE-Stop or KLE-Rollback, which is much better than the 20.08% achieved by agents trained without early stopping optimizations.

4.5.4 Averaged results across all environments

Fig. 4.2 summarises the investigation of the sensibility of the baseline PPO algorithm with respect to the fixed number of iterations used K (left block), as well as the sensibility of the KLE-Stop and KLE-Rollback variants of PPO with respect to the d_{target} hyperparameter (the two blocks on the right).

It can be seen that the performance of the baseline PPO version varies drastically depending on the value of K Furthermore, either the KLE-Stop or KLE-Rollback variant of the PPO algorithm exhibits a more stable performance across their respect to d_{target}. It also performs better than the baseline PPO on average.

Table 4.4 summarises the average episode returns obtained by the baseline PPO in each environment, without using the early stopping mechanism investigated in this work. First, a reproduction study was conducted to match the results obtained by the reference implementation provided in *openai/baselines*, before investigating the

impact of each KLE-Stop and KLE-Rollback variant of the PPO algorithm.

4.5.5 Detailed results of all the experiments

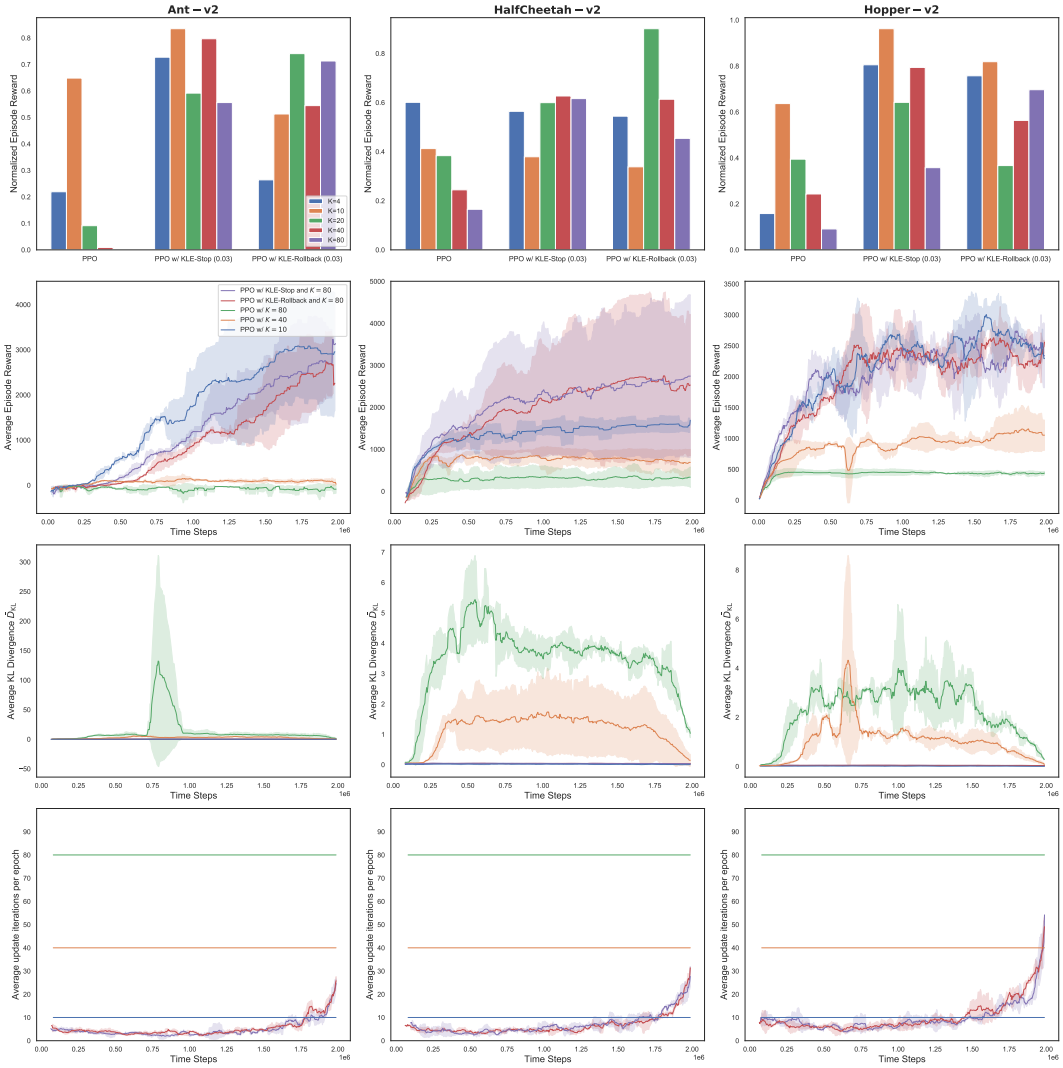


Fig. 4.3: Normalized Episode Reward (NER), Average Episode Reward, Average KL Divergence, and Average Update Iterations per epoch for each environment (one environment per column, part 1)

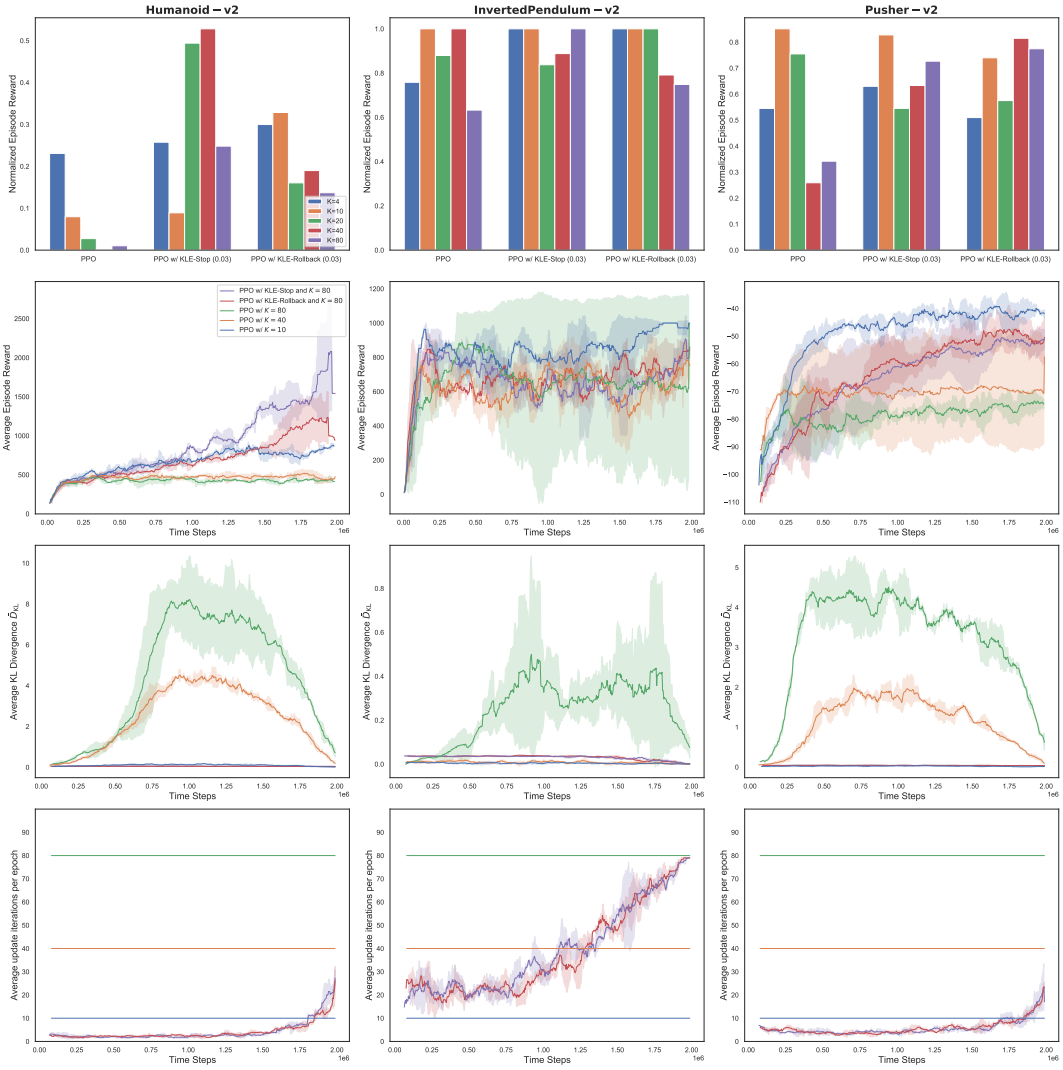


Fig. 4.4: Normalized Episode Reward (NER), Average Episode Reward, Average KL Divergence, and Average Update Iterations per epoch for each environment (one environment per column, part 2)

Fig. 4.3, Fig. 4.4, Fig. 4.5, and Fig. 4.6 document the detailed result of the investigation of the baselines PPO and its KLE-Stop, and KLE-Rollback early stopping variants for each robotic control task.

As mentioned in Section 4.5, for $K = 10$, PPO with KLE-Stop overall performs

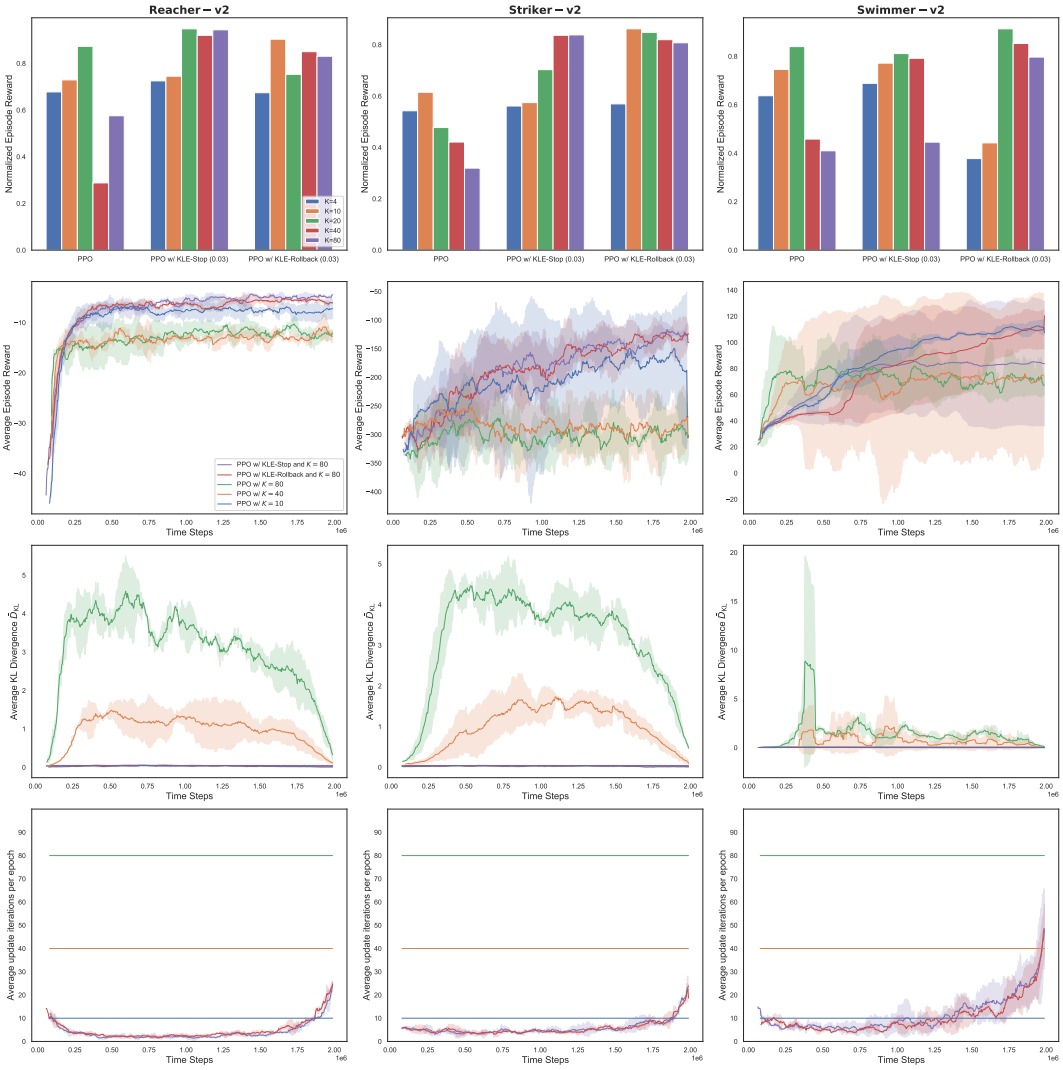


Fig. 4.5: Normalized Episode Reward (NER), Average Episode Reward, Average KL Divergence, and Average Update Iterations per epoch for each environment (one environment per column, part 3)

comparatively to the baselines PPO for the same K. Using $K = 10$ as a reference, a detailed, task-wise analysis of their respective performance (Fig. 4.3, Fig. 4.4, Fig. 4.5, and Fig. 4.6 reveals that for $K = 10$, PPO with KLE-Stop and $d_{target} = 0.03$ performs worse than the baselines PPO for the same K in 3 out of the 11 tasks experimented

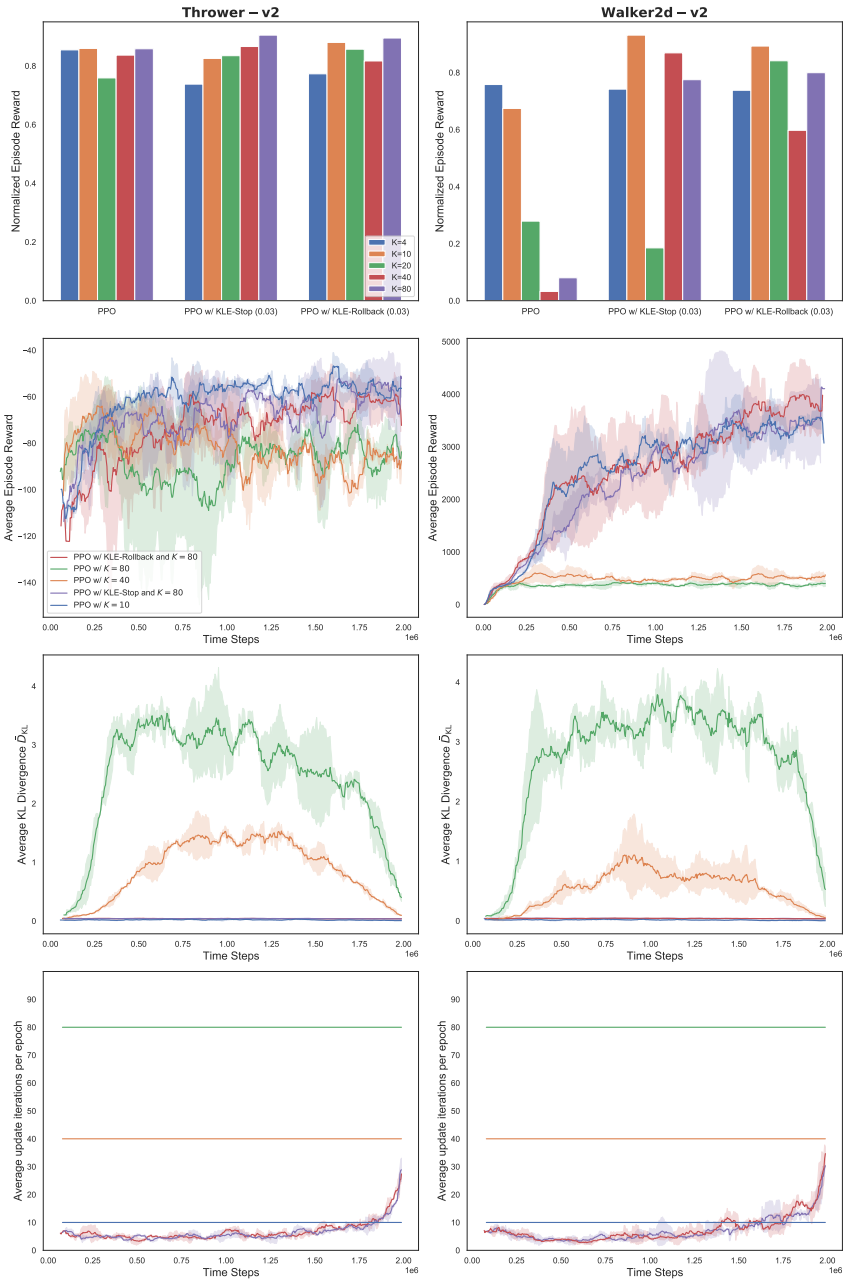


Fig. 4.6: Normalized Episode Reward (NER), Average Episode Reward, Average KL Divergence, and Average Update Iterations per epoch for each environment (one environment per column, part 4)

upon, and only by a slight margin. On the other tasks where it outperforms the baseline, however, it sometimes does so by a considerable margin, such as in Ant-v2 (Fig. 4.3), Hopper-v2 (Fig. 4.3), and Walker2D-v2 in (Fig. 4.6).

Since the PPO algorithm relies upon a fixed, and reasonably small amount of gradient update to the policy and value networks, high values of $K(\geq 20)$ highly correlate with a worsening in performance (Fig. 4.1a). For $K = 4$, the lowest number of updates per epoch investigated, the general assumption is that the agent "under learns" and its improvement is slow. For the same amount of update per epoch, PPO with early stopping optimization (PPO-KLE Stop with $d_{target} = 0.03$ still outperforms its baseline PPO counterpart on average (Fig. 4.2). In some environments, it also outperforms it by a high margin. For example, in Ant-v2 (Fig. 4.3), Hopper-v2 (Fig. 4.3), and the Inverted Pendulum-v2 (Fig. 4.3) tasks. Additionally, following Table 4.3, note that PPO-KLE Stop ($d_{target} = 0.03$) will perform on average less than $K = 4$ updates per epoch. This suggests that despite doing fewer updates than the baseline PPO with $K = 4$, it can still achieve a performance comparable to the best baseline PPO ($K = 10$). Combined with the comparison between PPO with $K = 10$ and its early stopping counterpart, this observation could suggest that the enforcement of the trust region in PPO using a fixed amount of updates per epoch might not be optimal. Instead, more flexible and systematic enforcement via early stopping ([94], based on heuristics such as KLE-Stop) would be more robust across tasks and allow the agent to achieve even better performance.

4.6 Conclusion

In this chapter, the empirical effect of early stopping optimizations when used in conjunction with other common code-level optimizations for PPO was investigated. The experiments conducted in this study show that 1) the performance of PPO is sensitive to the hyper-parameter "number of update iterations per epoch" K of the policy network, 2) early stopping optimizations mitigate such sensitivity by dynamically adjusting the number of policy network update iterations applied within an epoch, and 3) early stopping optimizations could serve as a convenient alternative to tuning on K since the experiments emphasizing d_{target} show that early stopping optimization is less sensitive to d_{target} than PPO is to K.

Overall, this suggests that taking into consideration the low-level implementation details and as well as early stopping mechanisms for trust region-based RL algorithms result in more reliable and robust agents. In practice, this enables faster training of RL agents, thus reducing computational cost and time required for iteration. This is an important aspect of RL research that impacts how fast and reliably RL agents can be deployed in real-world tasks.

Chapter 5 Conclusion

Overall, this thesis builds toward addressing critical challenges that arise in the quest of leveraging the reinforcement learning framework as a principled set of tools for automating real-world and high-impact tasks.

First, Chapter 2 addresses the practical problem of safely and conveniently deploying RL agents in real-world scenarios that retain the high performance of the RL paradigm, while also retaining a human-like behavior to improve the safety and comfort of surrounding human agents. The proposed hybrid of reinforcement learning and imitation learning of human demonstration has been demonstrated to achieve the best of both worlds. In the context of automating critical tasks with RL methods, the proposed method allows for maintaining the performance advantage of pure RL methods, while providing relatively safer and predictable RL agents that could be effectively deployed for real-world tasks.

Next, Chapter 3 demonstrates the increased sample efficiency of a human cognition-inspired model-based hierarchical reinforcement learning framework, which is critical to making RL methods a viable automation tool for real-world high-impact tasks. More specifically, the RL agents resulting from the proposed approach are shown to better scale to long-horizon sequential decision-making tasks that are ubiquitous to

real-world scenarios. Such agents can leverage the concept of temporally abstract exploration akin to the *mental time travel* ability exhibited by humans. By doing so, they can form better plans and perform more abstract, high-level decision-making, thus exhausting the state-action space of a given task more efficiently to discover viable policies with a more realistic cost. Additionally, this chapter has also demonstrated how the high-level decision-making of the proposed agents can align with human intuition, thus improving the interpretability and predictability of RL agents.

Finally, Chapter 4 demonstrates the benefits of leveraging lower-level implementation techniques such as early stopping mechanisms during the stochastic gradient descent phase in on RL methods based on trust regions (PPO [95]). Agents trained using the investigated early stopping mechanism are empirically shown to produce more robust policies across domains and sometimes reach higher performance when compared to their classical counterparts. This could help when applying existing RL algorithms onto new domains by reducing computation cost and iteration time, thus drastically speeding up the process of training and deploying RL agents in real-world scenarios.

Nevertheless, a lot more still needs to be done to further reduce the gap between theoretical reinforcement learning and its practical application in real-world scenarios. Some promising avenues for future work highlighted from the studies underlying this thesis include, but are not limited to:

(a) the development of a more comprehensive framework to integrate human user preference with the proposed hybrid of imitation and reinforcement learning,

(b) augmenting the proposed hybrid of imitation and reinforcement learning frame-

work with continuous learning and active inference to achieve RL agents that perpetually learn and co-evolve with the environment and the user,

(c) the development of multi-agent reinforcement learning that cooperates with other artificial agents as well as humans,

(d) the development of a hierarchical world model with more than two levels of hierarchy to scale to more complex long-horizon sequential decision-making tasks,

(e) the development of the proposed model-based hierarchical reinforcement learning framework to support modeling of other agents' beliefs and preference for multi-agent cooperation,

(f) the incorporation of inference mechanisms based on attention modules for better temporal state abstraction into the proposed model-based hierarchical reinforcement learning framework,

(g) the implementation of low-level optimizations of the sequence processing and inference mechanisms of temporal abstraction for better scalability of the proposed model-based hierarchical reinforcement learning agents.

References

[1] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018. Available at `https://github.com/openai/spinningup`.

[2] B. Balaguer and S. Carpin. Combining imitation and reinforcement learning to fold deformable planar objects. In *Proceedings of the International Conference on Intelligent Robots and Systems (IEEE/RSJ IROS)*, pages 1405–1412, 2011.

[3] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 2003.

[4] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *"Journal of Artificial Intelligence Research"*, 2013.

[5] Matthew Botvinick and Ari Weinstein. Model-based hierarchical reinforcement learning and human action control. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 2014.

[6] Matthew M. Botvinick, Yael Niv, and Andrew G. Barto. *Hierarchically organised behaviour and its neural foundations: a reinforcement-learning perspective*, page 264–299. "Cambridge University", 2011.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[9] Victor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giro-I-Nieto, and Jordi Torres. Explore, Discover and Learn: Unsupervised Discovery of State-Covering Skills. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1317–1327. PMLR, 13–18 Jul 2020.

[10] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. Learning to Search Better than Your Teacher. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 37, pages 2058–2066, 2015.

[11] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym, 2018.

[12] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[13] Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2020–2027. PMLR, 2021.

[14] Will Dabney, Georg Ostrovski, David Silver, and Remi Munos. Implicit quantile networks for distributional reinforcement learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2018.

[15] Hal Daumé III, John Langford, and Stéphane Ross. Efficient programmable learning to search. *CoRR*, abs/1406.1837, 2014.

[16] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*. Morgan-Kaufmann, 1992.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[19] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. OpenAI Baselines, 2017.

[20] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-Explore: A New Approach for Hard-Exploration Problems. *arXiv preprint arXiv:1901.10995*, 2019.

[21] Mohamed El-Shamouty, Xinyang Wu, Shanqi Yang, Marcel Albus, and Marco F. Huber. Towards safe human-robot collaboration using deep reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4899–4905, 2020.

[22] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation Matters in Deep RL: A Case Study on PPO and TRPO. In *International Conference on Learning Representations*, 2019.

[23] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *ArXiv*, abs/1802.06070, 2019.

[24] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610:47 – 53, 2022.

[25] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *CoRR*, abs/1704.03012, 2017.

[26] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[27] Kunihiko Fukushima, Ken-ichi Nagahara, and Hayaru Shouno. Training neocognitron to recognize handwritten digits in the real world. In *Proceedings of the 2nd AIZU International Symposium on Parallel Algorithms / Architecture Synthesis*, PAS '97, page 292. IEEE Computer Society, 1997.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[29] David Ha and Jürgen Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018.

[30] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018.

[31] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[32] Danijar Hafner, Kuang-Huei Lee, Ian Fischer, and Pieter Abbeel. Deep Hierarchical Planning from Pixels, 2022.

[33] Danijar Hafner, T. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *ArXiv*, abs/1912.01603, 2020.

[34] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels, 2018.

[35] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models, 2020.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[37] Simon Hecker, Dengxin Dai, and Luc van Gool. Learning accurate, comfortable and human-like driving. abs/1903.10995, 2019.

[38] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[39] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Learning from demonstrations for real world reinforcement learning. *CoRR*, abs/1704.03732, 2017.

[40] Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations, 2017*, 2017.

[41] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[42] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016.

[43] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[44] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement

learning. In *Proceedings of the International Conference on Robotics and Automation (IEEE ICRA)*, pages 2034–2039, 2018.

[45] Max Jaderberg, Volodymyr Mnih, Wojciech M. Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *ArXiv*, abs/1611.05397, 2017.

[46] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596, 2021.

[47] Sham Kakade and John Langford. Approximately Optimal Approximate Reinforcement Learning. In *Proceedings of the 19th International Conference on Machine Learning*, 2002.

[48] Taesup Kim, Sungjin Ahn, and Yoshua Bengio. Variational Temporal Abstraction. *arXiv e-prints*, 2019.

[49] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[50] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

[51] Ksenia Konyushkova, Konrad Zolna, Yusuf Aytar, Alexander Novikov, Scott E. Reed, Serkan Cabi, and Nando de Freitas. Semi-supervised reward learning for offline reinforcement learning. *ArXiv*, abs/2012.06899, 2020.

[52] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016.

[53] Solomon Kullback and Richard A Leibler. On Information and Sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[54] Ben Lau. Using keras and deep deterministic policy gradient to play torcs, 2016.

[55] Hoang M. Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. *CoRR*, abs/1803.00590, 2018.

[56] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[57] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 2015.

[58] Yann LeCun and Corinna Cortes. MNIST Handwritten Digit Database. 2010.

[59] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *CoRR*, abs/1805.00909, 2018.

[60] Alexander C. Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning. *CoRR*, abs/1906.05862, 2019.

[61] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. Gpu-accelerated robotic simulation for distributed reinforcement learning. *CoRR*, abs/1810.05762, 2018.

[62] 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. *arXiv preprint arXiv:1509.02971*, 2015.

[63] Grace W Lindsay, Josh Merel, Tom Mrsic-Flogel, and Maneesh Sahani. Divergent representations of ethological visual inputs emerge from supervised, unsupervised, and reinforcement learning. *arXiv preprint arXiv:2112.02027*, 2021.

[64] Josh Merel, Diego Aldarondo, Jesse Marshall, Yuval Tassa, Greg Wayne, and Bence Ölveczky. Deep neuroethology of a virtual rodent. *arXiv preprint arXiv:1911.09451*, 2019.

[65] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim M. Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin

Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, Will Hang, Emre Tuncer, Ananda Babu, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. Chip placement with deep reinforcement learning. *ArXiv*, abs/2004.10746, 2020.

[66] 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 the International Conference on Machine Learning (ICML)*, volume 48, pages 1928–1937, 2016.

[67] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[68] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[69] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *Proceedings of International Conference on Robotics and Automation (IEEE ICRA)*, pages 6292–6299, 2018.

[70] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceed-*

ings of International Conference on Machine Learning (ICML), pages 278–287. Morgan Kaufmann, 1999.

[71] Hamed Nili, Cai Wingfield, Alexander Walther, Li Su, William D. Marslen-Wilson, and Nikolaus Kriegeskorte. A toolbox for representational similarity analysis. *PLoS Computational Biology*, 10, 2014.

[72] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

[73] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. *CoRR*, abs/1910.07113, 2019.

[74] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[75] Juan Ortega, Noor Shaker, Julian Togelius, and Georgios N Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 4(2):93–104, 2013.

[76] Sindhu Padakandla. A survey of reinforcement learning algorithms for dynamically varying environments. 54(6), 2021.

[77] Tom Le Paine, Sergio Gomez Colmenarejo, Ziyun Wang, Scott E. Reed, Yusuf Aytar, Tobias Pfaff, Matthew W. Hoffman, Gabriel Barth-Maron, Serkan Cabi, David Budden, and Nando de Freitas. One-Shot High-Fidelity Imitation: Training Large-Scale Deep Nets with RL. *ArXiv*, abs/1810.05017, 2018.

[78] Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*. JMLR.org, 2020.

[79] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *CoRR*, abs/1706.01905, 2017.

[80] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

[81] Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Hc-search: A learning framework for search-based structured prediction. *The Journal of Artificial Intelligence Research (JAIR)*, 50, 2014.

[82] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[83] Shaghayegh Roohi, Christian Guckelsberger, Asko Relas, Henri Heiskanen, Jari Takatalo, and Perttu Hämäläinen. Predicting game difficulty and engagement using ai players. 5, 2021.

[84] Stéphane Ross and J. Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *CoRR*, abs/1406.5979, 2014.

[85] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15, pages 627–635, 2011.

[86] Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *CoRR*, abs/1511.06295, 2015.

[87] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[88] Vaibhav Saxena, Jimmy Ba, and Danijar Hafner. Clockwork variational autoencoders. In *Advances in Neural Information Processing Systems*, 2021.

[89] Tom Schaul, Dan Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning*. Journal of Machine Learning Research, 2015.

[90] Jasse Schell. *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann, 2008.

[91] Juergen Schmidhuber. Reinforcement Learning Upside Down: Don't Predict Rewards - Just Map Them to Actions. *ArXiv*, abs/1912.02875, 2019.

[92] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised Pre-Training for Speech Recognition. In *Proc. Interspeech 2019*, pages 3465–3469, 2019.

[93] J. Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *CoRR*, abs/1506.02438, 2016.

[94] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[95] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[96] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *CoRR*, abs/1610.03295, 2016.

[97] Archit Sharma, Shixiang Shane Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised skill discovery. In *ICLR 2020*, 2020.

[98] Wenjie Shi, Gao Huang, Shiji Song, Zhuoyuan Wang, Tingyu Lin, and Cheng Wu. Self-Supervised Discovering of Interpretable Features for Reinforcement Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:2712–2724, 2022.

[99] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *Proceedings of the International Conference on International Conference on Machine Learning (ICML)*, volume 32, pages I–387–I–395. JMLR, 2014.

[100] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[101] Nanda Kishore Sreenivas and Shrisha Rao. Safe deployment of a reinforcement learning robot using self stabilization. *Intelligent Systems with Applications*, 16:200105, 2022.

[102] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, and Yoshua Bengio. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, page 1929–1958, 2014.

[103] Rupesh Kumar Srivastava, Pranav Shyam, Filipe Wall Mutz, Wojciech Jaśkowski, and Juergen Schmidhuber. Training Agents using Upside-Down Reinforcement Learning. *ArXiv*, abs/1912.02877, 2019.

[104] Wen Sun, J. Andrew Bagnell, and Byron Boots. Truncated horizon policy search: Combining reinforcement learning & imitation learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[105] Xudong Sun and Bernd Bischl. Tutorial and survey on probabilistic graphical model and variational inference in deep reinforcement learning. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019.

[106] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[107] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning With Function Approximation. In *Advances in Neural Information Processing Systems*, 2000.

[108] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999.

[109] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[110] Momchil S. Tomov, Samyukta Yagati, Agni Kumar, Wanqian Yang, and Samuel J. Gershman. Discovery of hierarchical representations for efficient planning. *PLOS Computational Biology*, 2020.

[111] Mihaly Varadi, Stephen Anyango, Mandar Deshpande, Sreenath Nair, Cindy Natassia, Galabina Yordanova, David Yuan, Oana Stroe, Gemma Wood, Agata Laydon, Augustin Žídek, Tim Green, Kathryn Tunyasuvunakool, Stig Petersen, John Jumper, Ellen Clancy, Richard Green, Ankur Vora, Mira Lutfi, Michael Figurnov, Andrew Cowie, Nicole Hobbs, Pushmeet Kohli, Gerard Kleywegt, Ewan Birney, Demis Hassabis, and Sameer Velankar. AlphaFold Protein Structure Database: massively expanding the structural coverage of protein-sequence space with high-accuracy models. *Nucleic Acids Research*, 50(D1):D439–D444, 11 2021.

[112] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[113] 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 34th International Conference on Machine Learning - Volume 70*. Journal of Machine Learning Research, 2017.

[114] Bhanu Vikas. Deep Reinforcement Learning approach to Autonomous Navigation. 2017.

[115] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[116] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings the International Conference on Machine Learning (ICML)*, volume 48, pages 1995–2003, 2016.

[117] Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, Zhongwen Xu, and Shuicheng Yan. EnvPool: A Highly Parallel Reinforcement Learning Environment Execution Engine, 2022.

[118] Lilian Weng. Policy gradient algorithms. 2018. Available at https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html.

[119] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and P. Abbeel. Daydreamer: World models for physical robot learning. *ArXiv*, abs/2206.14176, 2022.

[120] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger B. Grosse, and Jimmy Ba. Scalable Trust-Region Method for Deep Reinforcement Learning Using Kronecker-Factored Approximation. *CoRR*, abs/1708.05144, 2017.

[121] Bernhard Wymann, Christos Dimitrakakis, Andrew D Sumner, Eric Espié, Christophe Guionneau, and Rémi Coulom. Torcs, the open racing car simulator, v1.3.5. 2013.

[122] Liyu Xia and Anne G. E. Collins. Temporal and state abstractions for efficient learning, transfer and composition in humans. *bioRxiv*, 2020.

[123] Naoto Yoshida. Gym TORCS, 2016.

[124] Yurong You. Torcs for reinforcement learning.

[125] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Information maximizing variational autoencoders. *ArXiv*, abs/1706.02262, 2017.

Publication List

Peer-Reviewed Journal Articles

1. Shengyi Huang, Rousslan F. J. Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, João G.M. Araújo, "CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms", *Journal of Machine Learning Research (JMLR)*, vol. 23, 2022

2. Rousslan F. J. Dossa, Shengyi Huang, Santiago Ontañón, Takashi Matsubara, "An Empirical Investigation of Early Stopping Optimizations in Proximal Policy Optimization", *IEEE Access*, vol. 9, pp. 117981-117992, 2021

3. Rousslan F. J. Dossa, Xinyu Lian, Hirokazu Nomoto, Takashi Matsubara, Kuniaki Uehara, "Hybrid of Reinforcement and Imitation Learning for Human-Like Agents", *IEICE Transactions on Information and Systems*, vol. E103.D, pp. 1960-1970, 2020

International Conference Proceedings

4. Rousslan F. J. Dossa, Takashi Matsubara, "Toward Human Cognition-inspired High-Level Decision Making For Hierarchical Reinforcement Learning Agents",

in *Proceedings of the Decision Awareness in Reinforcement Learning (DARL) Workshop at the International Conference on Machine Learning (ICML)*, 2022

5. Shengyi Huang, Rousslan F. J. Dossa, Antonin Raffin, Anssi Kanervisto, Weisun Wang, "The 37 Implementation Details of Proximal Policy Optimization", in *Proceedings of the Internal Conference on Learning Representations (ICLR) 2022 Blog Track*, 2022

6. Rousslan F. J. Dossa, Takashi Matsubara, "Toward Human Cognition-inspired High-Level Decision Making For Hierarchical Reinforcement Learning Agents", in *Proceedings of the Non-Linear Science Workshop (NLSW) at the International Symposium on Nonlinear Theory and Its Applications (NOLTA)*, 2021

7. Rousslan F. J. Dossa, Xinyu Lian, Hirokazu Nomoto, Takashi Matsubara, Kuniaki Uehara, "A Human-Like Agent Based on a Hybrid of Reinforcement and Imitation Learning", in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2019

Domestic Conference Proceedings

8. Rousslan F. J. Dossa, Takashi Matsubara, "Toward Human Cognition-inspired High-Level Decision Making For Hierarchical Reinforcement Learning Agents", in *IEICE Technical Committee on Complex Communication Science (CCS), IEICE Technical Report*, vol. 121, no. 253, pp. 61-66, 2021-

9. Rousslan F. J. Dossa, Xinyu Lian, Hirokazu Nomoto, Takashi Matsubara, Kuniaki Uehara, "Hybrid Reinforcement and Imitation Learning for Human-Like

Agents", in *IEICE Technical Committee on NeuroComputing (NC), IEICE Technical Report*, vol. 119, no. 88, pp. 69-74, 2019

10. Rousslan F. J. Dossa, Xinyu Lian, Hirokazu Nomoto, Takashi Matsubara, Kuniaki Uehara, "Building a Human-Like Agent Based on a Hybrid of Reinforcement and Imitation Learning", in *Proceedings of 33rd Annual Conference of the Japanese Society for Artificial Intelligence (JSAI)* (in Japanese), 2019

Preprint

11. Shengyi Huang, Anssi Kanervisto, Antonin Raffin, Weixun Wang, Santiago Ontañón, Rousslan F. J. Dossa, "A2C is a special case of PPO", *ArXiv Preprint*, vol. abs/2025.09123, 2022

Structure of this thesis

The relationship between the author's publications and the chapters of the present thesis is detailed as follows. First, Chapter 2 is based on the publication item 3, which introduces a hybrid of reinforcement and imitation learning for human-like agents:

- Rousslan F. J. Dossa, Xinyu Lian, Hirokazu Nomoto, Takashi Matsubara, Kuniaki Uehara, "Hybrid of Reinforcement and Imitation Learning for Human-Like Agents", *IEICE Transactions on Information and Systems*, vol. E103.D, pp. 1960-1970, 2020

Chapter 3 is based on the publication item 4. It proposes a hierarchical world model and extends its application with the hierarchical reinforcement learning framework, building toward agents capable of high-level decision-making akin to that of animals and humans:

- Rousslan F. J. Dossa, Takashi Matsubara, "Toward Human Cognition-inspired High-Level Decision Making For Hierarchical Reinforcement Learning Agents", in *Proceedings of the Decision Awareness in Reinforcement Learning (DARL) Workshop at the International Conference on Machine Learning (ICML)*, 2022

Finally, Chapter 4 is based on the publication item 2. It highlights the impact of the low-level implementation detail referred to as impact *optimization early stopping* mechanism on the robustness and final performance of PPO agents:

- Rousslan F. J. Dossa, Shengyi Huang, Santiago Ontañón, Takashi Matsubara, "An Empirical Investigation of Early Stopping Optimizations in Proximal Policy Optimization", *IEEE Access*, vol. 9, pp. 117981-117992, 2021

Acknowledgements

I would like to express my deepest gratitude to Professor Kuniaki Uehara and at the time associate Professor Takashi Matsubara for allowing me to join their lab at Kobe University and for their invaluable guidance throughout the course of my research as a student, a master's degree student, and a doctoral student. I am deeply grateful for Professor Uehara's broad expertise and guidance, as well as for Professor Matsubara's meticulous care and guidance on the details of machine learning theory and practical implementation. I am also deeply grateful to Professor Tetsuya Takiguchi for further accommodating my experience as a doctoral student at his laboratory. I would also like to extend my gratitude to the professors at the Kobe University Center for International Education for bringing me up to speed in Japanese, and greatly facilitating my integration into the research and daily life activities.

More over, I would like to express my gratitude to my labmates during my master's and doctoral studies, from whom I learned a lot. More specifically, I would like to thank senior Zeng Xiao for his guidance and encouragement during the early phase of my research and journey in the reinforcement learning field, as well as senior Xinyu Lian for the tremendous support during the experiments and advice, without which this work would not have been possible, as well as senior Xuejiao Deng for her support

and collaboration. Additionally, I would like to extend my thanks to Hama Kenta, Kusano Kouki, Kenia Ukai, and Zhou Boqian for their input, thoughtful discussions, assistance with Japanese writing and presentations, thesis proofreading, and other related academic activities.

Last but not least, I would like to express my gratitude to the reinforcement learning community for the passionate discussions and ideas exchanges, along with all the maintainers and contributors of RL libraries and codebase, namely the OpenAI Baselines, OpenAI SpinningUp, the Stable-Baselines as well as the Google's Dopamine repositories. I am also grateful to the members, collaborators, and co-authors of the CleanRL library development team Shengyi Huang, Anthonin Raffin, Niju, and many others for the invaluable insights, and practical knowledge earned during our study groups. Additionally, I would like to thank Taesup Kim and Danijar Hafner for publishing their respective work upon which part of the present thesis was built.

My sincere gratitude goes to the members of EQUOS RESEARCH Co., Ltd, for organizing the human sensitivity test to evaluate the trained agents to the participants for taking the time out of their busy schedules.

Finally, I would like to express my deepest gratitude to my family, for their thoughtful care and support throughout my life so far, and beyond.

Sincerely, Rousslan Fernand Julien Dossa.

Preliminary Examination Q&A Session

The present addendum documents the proceeds of the Q&A session with the esteemed jury members of the preliminary examination of this thesis held on December 7th, 2022.

1. **Q**: Regarding Chapter 2, how does the generative adversarial objective function lead to the student model converging to the hybrid policy of both experts in the continuous action space?

   A: The generative adversarial imitation learning objective function proposed in Chapter 2, Section 2.3.2 as Eq. 2.3.1 is as follows:

$$\begin{aligned}\mathcal{L}_{mix}(\pi) =& \mathbb{E}_{\tau\sim\pi}\left[\log\left(D_w(s,a)\right)\right] \\ &+ \alpha\, \mathbb{E}_{\tau_{RL}\sim\pi_{RL}}\left[\log\left(1 - D_w(s,a)\right)\right] \\ &+ (1-\alpha)\ \mathbb{E}_{\tau_{HE}\sim\pi_{HE}}\left[\log\left(1 - D_w(s,a)\right)\right],\end{aligned}$$

   Given a state-action pair (s, a), the discriminator D_w produces a continuous value in range $[0, 1]$, which can be interpreted as the likelihood of the given

state-action pair is that of the human or RL hybrid policy. Minimizing the first term $\mathbb{E}_{\tau\sim\pi}\left[\log\left(D_w\left(s,a\right)\right)\right]$ with respect to the weights w incentivizes the discriminator D_w to assign a low score to samples generated by the student model. On the other hand, minimizing the hybridization terms

$$\alpha\,\mathbb{E}_{\tau_{RL}\sim\pi_{RL}}[\log(1-D_w(s,a))]+(1-\alpha)\,\mathbb{E}_{\tau_{HE}\sim\pi_{HE}}[\log(1-D_w(s,a))]$$

encourages the model to assign a high score to state-action pairs matching the action distribution of either expert, with α to trade-off between each expert policy.

Therefore, by iterating over such minimax objective, such hybrid generative adversarial imitation learning implicitly reduces the divergence between the action distribution of the student model and that of the hybrid of RL and human expert. This has been further emphasized in Section 2.3.2 of Chapter 2.

2. **Q**: Regarding Chapter 2, would it be possible to address the matter of RL agent's human-likeliness from a low-level or implementation-level perspective akin to the proposed method in Chapter 4?

   A: Chapter 4 investigates the impacts of the early stopping mechanisms on the consistency of the resulting RL agents. While having more consistent and robust agents can benefit downstream methods such as the hybrid generative adversarial imitation learning used in Chapter 2, it is ultimately orthogonal to the latter.

   It would be difficult to realize a human-likely agent purely based on RL and

low-level implementation constraints because concepts such as *human-likeliness* and *human-friendliness* are hard to quantify objectively.

3. **Q**: Regarding Chapter 2, what separates the proposed hybrid of RL and human expert with other methods that leverage human expert data ?

   A: The majority of existing works combining human expert demonstrations and RL leverage the former to either train an agent purely based on supervised learning (imitation learning, behavior cloning). The second major train was to use expert demonstrations to pre-train various components of the RL agents, such as state feature extractors, before further finetuning the neural networks using the RL paradigm. Overall, these studies emphasized the quantitative evaluation of the resulting agents, disregarding more qualitative aspects of *human-likeliness* or *human-friendliness* of said agents.

   On the other hand, the study presented in Chapter 2 takes into account such qualitative aspects through a sensitivity test conducted in a double-blind review fashion. Such sensitivity test provides insights into the human-likeliness of the agents that were trained using the proposed joint objective combining both human expert demonstrations and the RL paradigm.

4. **Q**: Regarding Chapter 3, what warrants the human cognition-inspired high-level decision-making label, given that the proposed method essentially performs a hierarchical abstraction of the state representation ?

   A: As motivated in Section 3.2.2 of Chapter 3, a growing body of studies at the

intersection of neuroscience, cognitive science, psychology, and computational biology suggests that the human decision-making process is hierarchically organized. Tangential to the ability to perform different kinds of abstraction of observations, the ability of humans to also perform abstraction across the temporal dimension allows the simplification of the planning process for the original task into *pseudo-tasks* of coarser temporal granularity. This is especially useful for tasks that require sequential decision-making over a long horizon and usually end up being tasks with sparse rewards.

While the proposed hierarchical world model provides a (i) *hierarchically structured* latent state representation, it also provides a (ii) flexible mechanism that segments the trajectories of the agent into correlated groups of time steps. The proposed *HWM HRL Dreamer* agent leverages the coarse task dynamics that stem from (i) and (ii) to allow the high-level policy of the contemporary hierarchical reinforcement learning framework to plan and perform decision-making at a temporally abstract level, akin to how human plan over a longer horizon. More specifically, the *high-level decision-making* occurs by having the high-level policy π^H act solely based on the *temporally abstract, high-level component* Z of the state representation learned by the hierarchical world model. Each high-level action is triggered by following the signal of the *boundary indicator variable* M. The intuition of the underlying theory is motivated in Section 3.3.4 of Chapter 3, and illustrated in Fig. 3.2, reproduced below for convenience. Henceforth the denomination of *high-level decision-making* that is based on human cognition.

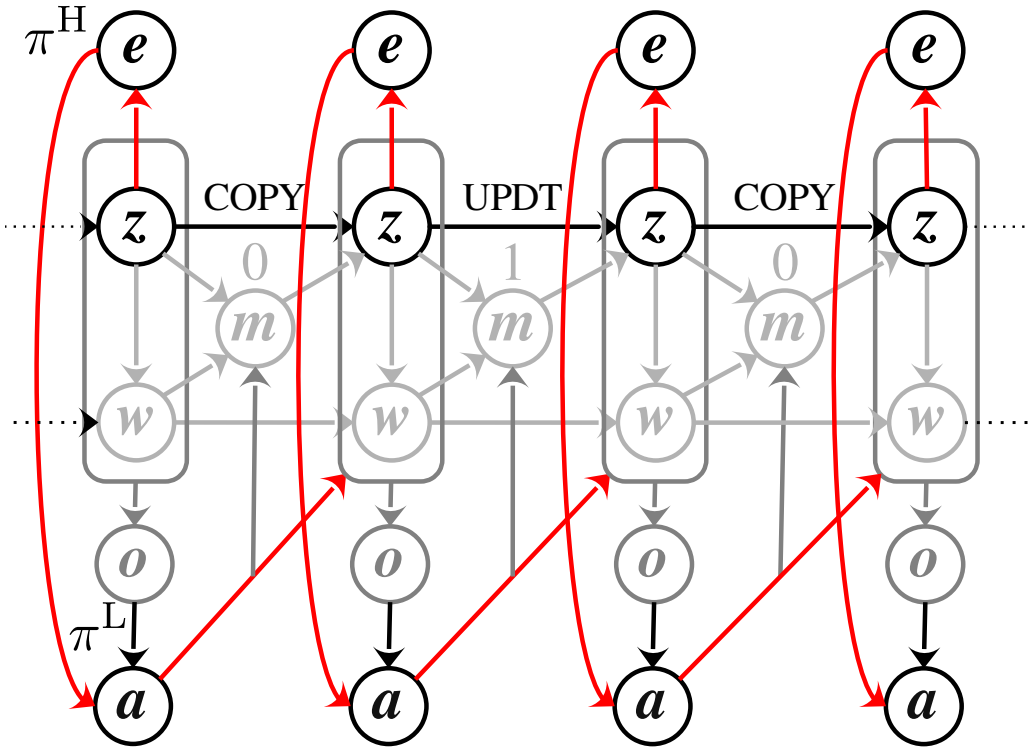


Fig. 5.1: Dynamic Bayesian network representing the generative process combining the HWM and HRL frameworks. The red line emphasizes the influence of the high-level policy's action over the high-level, temporally abstract state z's transition modeled by the proposed HWM.

5. **Q**: Regarding Chapter 3, what are the typical dimensions of the latent state belief S ? Also, what would be the dimension of the corresponding latent state belief formed by the concatenation of the Z and W components of the proposed hierarchical world model ?

   A: The *Dreamer-v2* algorithm that was used as an experimental baseline for the proposed hierarchical world model typically learns a 1624-dimensional latent state S. For fairness, the proposed hierarchical world model uses 812-dimensional vectors to represent the high and low-level state representation Z

and W, so that the combination of Z and W is equivalent to S.

6. **Q**: Regarding Chapter 3, namely the chart of the performance and sample efficiency in Section 3.5.2 (included below for convenience), what is the reason for the marginal gain in sample efficiency of the *HWM HRL Dreamer* over *HWM Dreamer*, compared to that of *HWM Dreamer* over the *Dreamer* baseline?

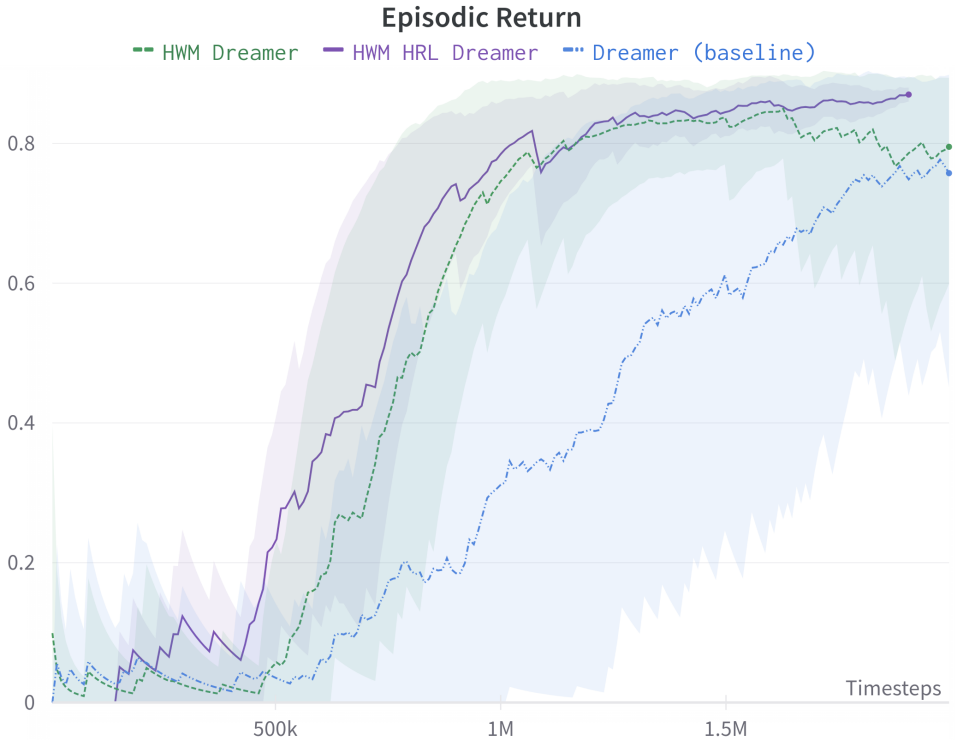


Fig. 5.2: Averaged episodic return of each agent variant across 4 seeds. The vertical axis holds the episodic return values, while the horizontal one shows the number of time steps (environment interactions).

A: Owing to the added constraint of learning a hierarchically structured state representation, *HWM Dreamer* learns latent state beliefs that disentangle correlated aspects of the task's dynamics in the state feature vector. This in turn allows the

downstream critic and policy deep neural networks to estimate the optimality of the value and action more effectively than traditional representation learning in world modeling methods, hence the drastic improvement in sample efficiency compared to the *Dreamer* baseline. *HWM HRL Dreamer*, builds on top of the hierarchically structured state representation learned by the proposed model to improve exploration of the overall agent through the guidance of the *high-level policy* that is based solely on the *temporally abstract, high-level component* Z. Given the relative simplicity of the *MiniGrid-FourRooms-v0* however, the gain in sample efficiency from high-level exploration is limited, hence the marginal improvement of *HWM HRL Dreamer* over *HWM Dreamer*. The gain in sample efficiency of the former is expected to increase proportionally to the difficulty of the task. For example, in mazes with larger rooms, higher count of rooms, or generally more intricate labyrinth layouts, exploring at the temporally abstract level is expected to yield better sample efficiency compared to baseline methods, which is to be empirically verified in current and future endeavors related to this chapter.

This has been reflected in Section 3.5.2 of Chapter 3 as an additional discussion of the performance and sample efficiency results.

Doctoral Thesis, Kobe University

"Human Cognition-Inspired High-Level Decision-Making for Reinforcement Learning Agents", 150 pages

Submitted on January, 20th, 2023.

The date of publication is printed in the cover of repository version published in Kobe University Repository Kernel.