IJCAI-22 Formatting Instructions

Learning rational behaviors in open-world games like Minecraft remains to be challenging for Reinforcement Learning (RL) research due to the compound challenge of partial observability, high-dimensional visual perception and delayed reward. To address this, we propose JueWu-MC, a sample-efficient hierarchical RL approach equipped with representation learning and imitation learning to deal with perception and exploration. Specifically, our approach includes two levels of hierarchy, where the high-level controller learns a policy to control over options and the low-level workers learn to solve each sub-task. To boost the learning of sub-tasks, we propose a combination of techniques including 1) action-aware representation learning which captures underlying relations between action and representation, 2) discriminator-based self-imitation learning for efficient exploration, and 3) ensemble behavior cloning with consistency filtering for policy robustness. Extensive experiments show that JueWu-MC significantly improves sample efficiency and outperforms a set of baselines by a large margin. Notably, we won the championship of the NeurIPS MineRL 2021 research competition and achieved the highest performance score ever.

Deep reinforcement learning (DRL) has shown great success in many genres of games, including board game (Silver et al., 2016), Atari (Mnih et al., 2013), simple first-person-shooter (FPS) (Huang et al., 2019), real-time strategy (RTS) (Vinyals et al., 2019), multiplayer online battle arena (MOBA) (Berner et al., 2019), etc. Recently, open-world games have been attracting attention due to its playing mechanism and similarity to real-world control tasks (Guss et al., 2021). Minecraft, as a typical open-world game, has been increasingly explored for the past few years (Oh et al., 2016; Tessler et al., 2017; Guss et al., 2019; Kanervisto et al., 2020; Skrynnik et al., 2021; Mao et al., 2021).

Compared to other games, the characteristics of Minecraft make it a suitable testbed for RL research, as it emphasizes exploration, perception and construction in a 3D open world (Oh et al., 2016). The agent is only provided with partial observability and occlusions. The tasks in the game are chained and long-term. Generally, human can make rational decisions to explore basic items and construct desired higher-level items using a reasonable amount of samples, while it can be hard for an AI agent to do so autonomously. Therefore, to facilitate the efficient decision-making of agents in playing Minecraft, MineRL (Guss et al., 2019) has been developed as a research competition platform, which provides human demonstrations and encourages the development of sample-efficient RL agents for playing Minecraft. Since the release of MineRL, a number of efforts have been made on developing Minecraft AI agents, e.g., ForgER (Skrynnik et al., 2021), SEIHAI (Mao et al., 2021).

However, it is still difficult for existing RL algorithms to mine items in Minecraft due to the compound challenge it poses, expanded below.

Long-time Horizons

In order to achieve goals (e.g., mining a diamond) in Minecraft, the agent is required to finish

a variety of sub-tasks (e.g., log, craft) that highly depend on each other. Due to the sparse reward, it is hard for agents to learn long-horizon decisions efficiently. Hierarchical RL from demonstrations (Le et al., 2018; Pertsch et al., 2020) has been explored to leverage the task structure to accelerate the learning process. However, learning from unstructured demonstrations without domain knowledge remains challenging.

High-dimensional Visual Perception

Minecraft is a flexible 3D first-person game revolving around gathering resources (i.e., explore) and creating structures and items (i.e., construct). In this environment, agents are required to deal with high-dimensional visual input to enable efficient control. However, agent's surroundings are varied and dynamic, which poses difficulties to learning a good representation.

Inefficient Exploration

With partial observability, the agent needs to explore in the right way and collect information from the environment so as to achieve goals. A naive exploration strategy can waste a lot of samples on useless exploration. Self-imitation Learning (SIL) (Oh et al., 2018) is a simple method that learns to reproduce past good behaviors to incentivize deep exploration. However, SIL is not sample-efficient because its advantage-clipping operation causes a waste of samples. Moreover, SIL does not make use of the transitions between samples.

Imperfect Demonstrations

Human demonstrations in playing Minecraft are highly distributional diverse (Kanervisto et al., 2020). Also, there exists noisy data due to the imperfection of human operation (Guss et al., 2019).

To address the aforementioned compound challenges, we develop an efficient hierarchical RL approach equipped with novel representation and imitation learning techniques. Our method makes effective use of human demonstrations to boost the learning of agents and enables the RL algorithm to learn rational behaviors with high sample efficiency.

Hierarchical Planing with Prior

We first propose a hierarchical RL (HRL) framework with two levels of hierarchy, where the high-level controller automatically extracts sub-goals in long-horizon trajectories from the unstructured human demonstrations and learns a policy to control over options, while the low-level workers learn sub-tasks to achieve sub-goals by leveraging both demonstrations dispatched by the high-level controller and interactions with environments. Our approach automatically structures the demonstrations and learns a hierarchical agent, which enables better decision over long-horizon tasks. Under our HRL framework, we devise the following key techniques to boost agent learning.

Action-aware Representation Learning

Although some prior works (Huang et al., 2019) proposed using auxiliary tasks (e.g., enemy detection) to better understand the 3D world, such methods require a large amount of labeled data. We propose a self-supervised action-aware representation learning (A2RL) technique, which learns to capture the underlying relations between action and representation in 3D visual environments like Minecraft. As we will show, A2RL not only enables effective control by learning a compact representation but also improves the interpretability of the learned policy.

Discriminator-based Self-imitation Learning

As mentioned, existing self-imitation learning is advantage-based and becomes sample-inefficient for handling tasks in Minecraft, as it wastes a lot of samples due to the clipped objective and does not utilize transitions between samples. Therefore, we propose discriminator-based self-imitation learning (DSIL) which leverages self-generated experiences to learn self-correctable policies for better exploration.

Ensemble Behavior Cloning with Consistency Filtering

Learning a robust policy from imperfect demonstrations is difficult (Wu et al., 2019). To address this issue, we first propose consistency filtering to identify the most common human behavior, and then perform ensemble behavior cloning to learn a robust agent with reduced uncertainty.

In summary, our contributions are: 1) We propose JueWu-MC, a sample-efficient hierarchical RL approach, equipped with action-aware representation learning, discriminator-based self-imitation, and ensemble behavior cloning with consistency filtering, for training Minecraft AI agents. 2) Our approach outperforms competitive baselines by a significantly large margin and achieves the best performance ever throughout the MineRL competition history. Thorough ablations and visualizations are further conducted to help understand why our approach works.

Game AI

Game has long been a preferable field for artificial intelligence research. AlphaGo (Silver et al., 2016) mastered the game of Go with DRL and tree search. Since then, DRL has been used in other more sophisticated games, including StarCraft (RTS) (Vinyals et al., 2019), Google Football (Sports) (Kurach et al., 2020), VizDoom (FPS) (Huang et al., 2019), Dota (MOBA) (Berner et al., 2019). Recently, the 3D open-world game Minecraft is drawing rising attention. Oh et al. (2016) showed that existing RL algorithms suffer from generalization in Minecraft and proposed a new memory-based DRL architecture. Tessler et al. (2017) proposed H-DRLN, a combination of a deep skill array and a skill distillation system, to promote lifelong learning and transfer knowledge among different tasks in Minecraft. Since

MineRL was held in 2019, many solutions have been proposed to learn to play in Minecraft. There works can be grouped into two categories: 1) end-to-end learning (Amiranashvili et al., 2020; Kanervisto et al., 2020; Scheller et al., 2020); 2) HRL with human demonstrations (Skrynnik et al., 2021; Mao et al., 2021). Our approach belongs to the second category. In this category, prior works leverage the structure of the tasks and learn a hierarchical agent to play in Minecraft - ForgER (Skrynnik et al., 2021) proposed a hierarchical method with forgetful experience replay to allow the agent to learn from low-quality demonstrations; Mao et al. (2021) proposed SEIHAI that fully takes advantage of the human demonstrations and the task structure.

Sample-efficient Reinforcement Learning

Our work is to build a sample-efficient RL agent for playing Minecraft, and we thereby develop a combination of efficient learning techniques. We discuss the most relevant works below.

Our work is related to recent HRL research that builds upon human priors. To expand, Le et al. (2018) proposed to warm-up the hierarchical agent from demonstrations and fine-tune with RL algorithms. Pertsch et al. (2020) proposed to learn a skill prior from demonstrations to accelerate HRL algorithms. Compared to existing works, we are faced with the highly unstructured demo in 3D first-person video games played by the crowds. We address this challenge by structuring the demonstrations and defining sub-tasks and sub-goals automatically.

Representation learning in RL has two broad directions: self-supervised learning and contrastive learning. The former (Wu et al., 2021) aims at learning rich representations for high-dimensional unlabeled data to be useful across tasks, while the latter (Srinivas et al., 2020) learns representations that obey similarity constraints in a dataset organized by similar and dissimilar pairs. Our work proposes a novel self-supervised representation learning method that can measure action effects in 3D video games.

Existing methods use curiosity or uncertainty as a signal for exploration (Pathak et al., 2017; Burda et al., 2018) so that the learned agent is able to cover a large state space. However, the exploration-exploitation dilemma, given the sample efficiency consideration, drives us to develop self-imitation learning (SIL) (Oh et al., 2018) methods that focus on exploiting past good experiences for better exploration. Hence, we propose discriminator-based self-imitation learning (DSIL) for efficient exploration.

Our work is also related to learning from imperfect demonstrations, such as DQfD (Hester et al., 2018) and Q-filter (Nair et al., 2018). Most methods in this field leverage online interactions with the environment to handle the noise in demonstrations. We propose ensemble behavior cloning with consistency filtering (EBC) which leverages imperfect demonstrations to learn robust policies in playing Minecraft.

3 Method

In this section, we first introduce our overall HRL framework, and then illustrates the details of each component.

3.1 Overview

Figure 1 shows our overall framework. We define the human demonstrations as =0,1,2,…subscript0subscript1subscript2…\mathcalD=\\tau_0,\tau_1,\tau_2,...\caligraphic_D = italic_ start_POSTSUBSCRIPT 0 end_POSTSUBSCRIPT , italic_ start_POSTSUBSCRIPT 1 end_POSTSUBSCRIPT , italic_ start_POSTSUBSCRIPT 2 end_POSTSUBSCRIPT , … where isubscript\tau_iitalic_ start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT represents a long-horizon trajectory containing states, actions and rewards. The provided demonstrations are unstructured in that there are no explicit signals to specify sub-tasks and sub-goals.

We first define atomic skill as an individual skill that gets a non-zero reward. Then, we define sub-tasks and sub-goals based on atomic skill. To define reasonable sub-tasks, we examine the degree of reward delay for each atomic skill. We keep those atomic skills with long reward delay as individual sub-tasks because they require executing a long sequence of actions to achieve a delayed reward. Meanwhile, we merge those adjacent atomic skills with short reward delay into one sub-task. By doing so, we get nnitalic_n sub-tasks (a.k.a stages) in total. To define a sub-goal for each sub-task, we extract the most common human behavior pattern and use the last state in each sub-task as its sub-goal. In this way, we get structured demonstrations (0,1,…,n-1subscript0subscript1…subscript1\mathcalD\rightarrow\\mathcalD_0,\mathcalD_1,...,\mathcalD_n-1\caligraphic_D caligraphic_D start_POSTSUBSCRIPT 0 end_POSTSUBSCRIPT , caligraphic_D start_POSTSUBSCRIPT 1 end_POSTSUBSCRIPT , … , caligraphic_D start_POSTSUBSCRIPT italic_n - 1 end_POSTSUBSCRIPT) with sub-tasks and sub-goals that are used to train the hierarchical agent. With the structured demonstrations, we train the meta-policy by imitation learning, and train the sub-policies to solve sub-tasks by leveraging both demonstrations and interactions with the environment, as described below.

3.2 Meta- and Sub-policies

Meta-policy

We train a meta-policy that maps continuous states to discrete indices (0,1,…,n-101…10,1,...,n-10 , 1 , … , italic_n - 1) that specifies which option to be used. Given state space ¯¯\bar\mathcalSover¯ start_ARG caligraphic_S end_ARG and discrete option space \mathcalOcaligraphic_O, the meta-policy is defined as m(o|s¯)subscriptsuperscriptconditional¯\pi^m_\rho(o|\bars)italic_ start_POSTSUPERSCRIPT italic_m end_POSTSUPERSCRIPT start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_o | over¯ start_ARG italic_s end_ARG), where s¯¯¯¯¯\bars\in\bar\mathcalSover¯

start_ARG italic_s end_ARG over¯ start_ARG caligraphic_S end_ARG is an inventory vector that summarizes the agent's collected items, oo\in\mathcalOitalic_o caligraphic_O is a discrete value and \rhoitalic_ represents parameters.
m(o|s¯)subscriptsuperscriptconditional¯\pi^m_\rho(o|\bars)italic_ start_POSTSUPERSCRIPT italic_m end_POSTSUPERSCRIPT start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_o | over¯ start_ARG italic_s end_ARG) specifies the conditional distribution over the discrete options. To train the meta-policy, we generate training data (s¯,i)¯(\bars,i)(over¯ start_ARG italic_s end_ARG , italic_i) where iiitalic_i represents the iiitalic_i-th stage and s¯i¯subscript\bars\in\mathcalD_iover¯ start_ARG italic_s end_ARG caligraphic_D start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT is sampled from the demonstrations of the iiitalic_i-th stage. The meta-policy is trained using negative log-likelihood (NLL) loss:

mini=0n-1s¯i[-logm(i|s¯)].subscriptsuperscriptsubscript01¯subscriptdelimited-[]subscriptsuperscriptconditional¯\footnotesize\min_\rho\sum_i=0^n-1\underset\bars\in\mathcalD_i% \mathbbE\left[-\log\pi^m_\rho(i|\bars)\right].roman_min start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT start_POSTSUBSCRIPT italic_i = 0 end_POSTSUBSCRIPT start_POSTSUPERSCRIPT italic_n - 1 end_POSTSUPERSCRIPT start_UNDERACCENT over¯ start_ARG italic_s end_ARG caligraphic_D start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT end_UNDERACCENT start_ARG blackboard_E end_ARG [- roman_log italic_ start_POSTSUPERSCRIPT italic_m end_POSTSUPERSCRIPT start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_i | over¯ start_ARG italic_s end_ARG)] . (1)
During inference, the meta-policy generates options by taking argmax on the distribution o^=argmaxom(o|s¯)^subscriptargmaxsubscriptsuperscriptconditional¯\hato=\operatorname*arg\,max_o\pi^m_\rho(o|\bars)over^ start_ARG italic_o end_ARG = start_OPERATOR roman_arg roman_max end_OPERATOR start_POSTSUBSCRIPT italic_o end_POSTSUBSCRIPT italic_ start_POSTSUPERSCRIPT italic_m end_POSTSUPERSCRIPT start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_o | over¯ start_ARG italic_s end_ARG).

Sub-policy

In Minecraft, sub-tasks can be grouped into two main types: gathering resources, and crafting items. In the first type (gathering resources), agents need to navigate and gather sparse rewards by observing high-dimensional visual inputs which are varied and dynamic. In the second type (crafting items), agents need to execute a sequence of actions robustly.

In typical HRL, action space of the sub-policies is pre-defined according to prior knowledge. However, in the MineRL 2020&2021, handcrafted action space is prohibited. Besides, action space is obfuscated in both human demonstrations and the environment. Directly learning in this continuous action space is challenging as exploration in a large continuous action space can be inefficient. Therefore, we use KMeans (Krishna and Murty, 1999) to cluster actions for each sub-task using demonstration isubscript\mathcalD_icaligraphic_D start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT, and perform reinforcement learning

and imitation learning based on the clustered action space.

In the following section, we describe how to learn sub-policies efficiently to solve these two kinds of sub-tasks.

3.3 Learning Sub-policies to Gather Resources

To efficiently solve this kind of sub-tasks, we propose action-aware representation learning as well as discriminator-based self-imitation learning to facilitate the learning process of sub-policies. We show the model architecture in Figure 2. The full algorithm is shown in Appendix Algorithm.

3.3.1 Action-aware Representation Learning

Learning compact representation is crucial to improve sample efficiency in reinforcement learning (Lesort et al., 2018). To tackle the challenge of learning good representation in 3D open world, we start by observing that in first-person 3D environments, different actions have their own effects - each action acts on a local part of the high-dimensional observations. For example, in Minecraft, the attack action aims to break and acquire the block in front of the agent, while the camera action aims to adjust the agent's camera perspective. Motivated by this observation, we propose action-aware representation learning (A2RL), to learn representation that can capture the underlying relation with actions.

To achieve so, we leverage the dynamic property from environments. Specifically, we learn a mask net on feature map for each action to capture dynamic information between the current and next states. Denote the feature map as f(s)H×Wsubscriptsuperscriptf_\theta(s)\in\mathbbR^H\times Witalic_f start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s) blackboard_R start_POSTSUPERSCRIPT italic_H × italic_W end_POSTSUPERSCRIPT and the mask net as m(s,a)[0,1]H×Wsubscriptitalic-superscript01m_\phi(s,a)\in[0,1]^H\times Witalic_m start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s , italic_a) [0 , 1] start_POSTSUPERSCRIPT italic_H × italic_W end_POSTSUPERSCRIPT, where \thetaitalic_ and italic-\phiitalic_ represent parameters of convolution neural network of the policy and mask net. Given a transition tuple (s,a,s)superscript(s,a,s^\prime)(italic_s , italic_a , italic_s start_POSTSUPERSCRIPT end_POSTSUPERSCRIPT), the loss function for training the mask is as follows:

m1()=s,a,s[ga((1-m(s,a))f(s))-f(s)2],superscriptsubscript1italic-similar-tosuperscriptdelimited-[]subscriptnormsubscriptsubscriptdirect-product1subscriptitalic-subscriptsubscriptsuperscript2\scriptsize\mathcalL_m^1(\phi)=\undersets,a,s^\prime\sim\mathcalD% \mathbbE\bigg[\Big\g_\psi_a\big((1-m_\phi(s,a))\odot f_\theta% (s)\big)-f_\theta(s^\prime)\Big_2\bigg],caligraphic_L start_POSTSUBSCRIPT italic_m end_POSTSUBSCRIPT start_POSTSUPERSCRIPT 1 end_POSTSUPERSCRIPT (italic_) = start_UNDERACCENT italic_s , italic_a , italic_s start_POSTSUPERSCRIPT

end_POSTSUPERSCRIPT caligraphic_D end_UNDERACCENT start_ARG blackboard_E end_ARG [italic_g start_POSTSUBSCRIPT italic_ start_POSTSUBSCRIPT italic_a end_POSTSUBSCRIPT end_POSTSUBSCRIPT ((1 - italic_m start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s , italic_a)) italic_f start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s)) - italic_f start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s start_POSTSUPERSCRIPT end_POSTSUPERSCRIPT) start_POSTSUBSCRIPT 2 end_POSTSUBSCRIPT] , (2)

m2()=s,a,s[-ga(m(s,a)f(s))-f(s)2],superscriptsubscript2italic-similar-tosuperscriptdelimited-[]subscriptnormsubscriptsubscriptdirect-productsubscriptitalic-subscriptsubscriptsuperscript2\scriptsize\mathcalL_m^2(\phi)=\undersets,a,s^\prime\sim\mathcalD% \mathbbE\bigg[-\Bigg_\psi_a\big(m_\phi(s,a)\odot f_\theta(s%)\big)-f_\theta(s^\prime)\Big_2\bigg],caligraphic_L start_POSTSUBSCRIPT italic_m end_POSTSUBSCRIPT start_POSTSUPERSCRIPT 2 end_POSTSUPERSCRIPT (italic_) = start_UNDERACCENT italic_s , italic_a , italic_s start_POSTSUPERSCRIPT end_POSTSUPERSCRIPT caligraphic_D end_UNDERACCENT start_ARG blackboard_E end_ARG [- italic_g start_POSTSUBSCRIPT italic_ start_POSTSUBSCRIPT italic_a end_POSTSUBSCRIPT end_POSTSUBSCRIPT (italic_m start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s , italic_a) italic_f start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s)) - italic_f start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s start_POSTSUPERSCRIPT end_POSTSUPERSCRIPT) start_POSTSUBSCRIPT 2 end_POSTSUBSCRIPT] , (3)

m()=m1()+m2(),subscriptitalic-superscriptsubscript1italic-superscriptsubscript2italic-\footnotesize\mathcalL_m(\phi)=\mathcalL_m^1(\phi)+\eta\mathcalL_% m^2(\phi),caligraphic_L start_POSTSUBSCRIPT italic_m end_POSTSUBSCRIPT (italic_) = caligraphic_L start_POSTSUBSCRIPT italic_m end_POSTSUBSCRIPT start_POSTSUPERSCRIPT 1 end_POSTSUPERSCRIPT (italic_) + italic_ caligraphic_L start_POSTSUBSCRIPT italic_m end_POSTSUBSCRIPT start_POSTSUPERSCRIPT 2 end_POSTSUPERSCRIPT (italic_) , (4)
where gasubscriptsubscriptg_\psi_aitalic_g start_POSTSUBSCRIPT italic_ start_POSTSUBSCRIPT italic_a end_POSTSUBSCRIPT end_POSTSUBSCRIPT is a linear projection function parameterized by learnable parameters asubscript\psi_aitalic_ start_POSTSUBSCRIPT italic_a end_POSTSUBSCRIPT; direct-product\odot represents element-wise product; \etaitalic_ is a hyper-parameter to trade off two objectives.

To optimize Eq 4, we use a two-stage training process. In the first stage, we train the linear projection network gasubscriptsubscriptg_\psi_aitalic_g start_POSTSUBSCRIPT italic_ start_POSTSUBSCRIPT italic_a end_POSTSUBSCRIPT end_POSTSUBSCRIPT using the following objective:

g(a)=s,a,s[ga(f(s))-f(s)2].subscriptsubscriptsimilar-tosuperscriptdelimited-[]subscriptnormsubscriptsubscriptsubscriptsubscriptsuperscript2\footnotesize\mathcalL_g(\psi_a)=\undersets,a,s^\prime\sim\mathcalD% \mathbbE\bigg[\Big\g_\psi_a\big(f_\theta(s)\big)-

f_\theta% (s^\prime)\Big_2\bigg].caligraphic_L start_POSTSUBSCRIPT italic_g end_POSTSUBSCRIPT (italic_ start_POSTSUBSCRIPT italic_a end_POSTSUBSCRIPT) = start_UNDERACCENT italic_s , italic_a , italic_s start_POSTSUPERSCRIPT end_POSTSUPERSCRIPT caligraphic_D end_UNDERACCENT start_ARG blackboard_E end_ARG [italic_g start_POSTSUBSCRIPT italic_ start_POSTSUBSCRIPT italic_a end_POSTSUBSCRIPT end_POSTSUBSCRIPT (italic_f start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s)) - italic_f start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s start_POSTSUPERSCRIPT end_POSTSUPERSCRIPT) start_POSTSUBSCRIPT 2 end_POSTSUBSCRIPT] . (5)
This objective learns to recover information of ssuperscripts^\primeitalic_s start_POSTSUPERSCRIPT end_POSTSUPERSCRIPT from ssitalic_s in latent space, which is equivalent to learning a dynamic model to predict next state given current state and action. Note that the parameter asubscript\psi_aitalic_ start_POSTSUBSCRIPT italic_a end_POSTSUBSCRIPT is dependent with action aaitalic_a. In the second stage, we fix the learned linear function gasubscriptsubscriptg_\psi_aitalic_g start_POSTSUBSCRIPT italic_ start_POSTSUBSCRIPT italic_a end_POSTSUBSCRIPT end_POSTSUBSCRIPT and use Eq 4 to optimize the mask net.

Intuitively, on the one hand, by minimizing Eq 2, the mask net will learn to mask out the parts that introduce uncertainty to the model-based prediction, while remaining adequate information to predict the next state. On the other hand, by minimizing Eq 3, the mask net will tend to pay attention to as little information as possible, trying to introduce uncertainty to the prediction. Therefore, by minimizing them jointly in Eq 4, the mask net can learn to focus on local parts of the current image that introduce uncertainty to the dynamic model. This is similar to human curiosity, which pays attention to the part that is uncertain to themselves.

A2RL is reminiscent of dynamics-based representation Whitney et al. (2019). However, dynamics-based representation learning aims to learn representation that is capable to imagine dynamics with a long horizon. Our approach aims to learn the underlying relations between representation and action by leveraging one-step dynamic prediction - this provides the agent with multi-view representations that reveal the effects of different actions. The learned representations can then be combined with any off-the-shelf RL algorithms to improve sample efficiency.

For policy-based methods, we plug our learned representations into policy (a|(1+m(s,a))f(s))subscriptconditionaldirect-product1subscriptitalic-subscript\pi_\theta(a|(1+m_\phi(s,a))\odot f_\theta(s))italic_ start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_a | (1 + italic_m start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s , italic_a)) italic_f start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s)) for effective perception and efficient back-propagation of policy gradient. For value-based methods, we combine our learned representation directly with Q-value functions Qq((1+m(s,a))f(s),a)subscriptsubscriptdirect-product1subscriptitalic-subscriptQ_\theta_q((1+m_\phi(s,a))\odot f_\theta(s),a)italic_Q start_POSTSUBSCRIPT italic_ start_POSTSUBSCRIPT italic_q end_POSTSUBSCRIPT end_POSTSUBSCRIPT ((1

+ italic_m start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s , italic_a)) italic_f start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s) , italic_a). The learning of Q-value function can be done using any Q-learning based algorithms.

3.3.2 Discriminator-based Self-imitation Learning

Self-imitation Learning (SIL) (Oh et al., 2018) is considered as a simple but effective way to solve hard-exploration tasks. SIL uses an advantage clipping technique to bias the agent towards good behaviors, which we call it as advantage-based self-imitation learning (ASIL). However, Mega blog is not sample-efficient due to the clipping mechanism. Besides, SIL does not leverage the transition between samples.

To address the issues of SIL, we propose discriminator-based self-imitation learning (DSIL). Unlike ASIL, DSIL does not use advantage clipping. Our intuition is that the agent should be encouraged to visit the state distribution that is more likely to lead to goals.

To do so, DSIL first learns a discriminator to distinguish between states from successful and failed trajectories (i.e., “good” and “bad” states), and then uses the learned discriminator to guide exploration. Specifically, We maintain two replay buffers i+superscriptsubscript\mathcalB_i^+caligraphic_B start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT and i-superscriptsubscript\mathcalB_i^-caligraphic_B start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT to store successful and failed trajectories respectively. During learning, we treat data from i+superscriptsubscript\mathcalB_i^+caligraphic_B start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT as positive samples and data from i-superscriptsubscript\mathcalB_i^-caligraphic_B start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT as negative samples to train the discriminator. Denote the discriminator as D:[0,1]:subscript01D_\xi:\mathcalS\rightarrow[0,1]italic_D start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT : caligraphic_S [0 , 1] which is parameterized by parameters \xiitalic_. We train the discriminator with the following objective:

maxsi+[logD(s)]+si-[1-logD(s)].subscriptsimilar-tosuperscriptsubscriptdelimited-[]subscriptsimilar-tosuperscriptsubscriptdelimited-[]1subscript\footnotesize\max_\xi\undersets\sim\mathcalB_i^+\mathbbE\left[% \log D_\xi(s)\right]+\undersets\sim\mathcalB_i^-\mathbbE\left[1-% \log D_\xi(s)\right].roman_max start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT start_UNDERACCENT italic_s caligraphic_B start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT start_POSTSUPERSCRIPT + end_POSTSUPERSCRIPT end_UNDERACCENT start_ARG blackboard_E end_ARG [roman_log italic_D start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s)] + start_UNDERACCENT italic_s caligraphic_B start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT start_POSTSUPERSCRIPT - end_POSTSUPERSCRIPT end_UNDERACCENT start_ARG blackboard_E end_ARG [1 -

roman_log italic_D start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s)] . (6) Intuitively, this objective encourages D(s)subscriptD_\xi(s)italic_D start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s) to output high values for good states while giving low values for bad states. For those states that are not distinguishable, D(s)subscriptD_\xi(s)italic_D start_POSTSUBSCRIPT italic_ end_POSTSUBSCRIPT (italic_s) tends to output 0.5. The learned discriminator captures the good state distribution that leads to goals and the bad state distribution that leads to failure.

We then use the trained discriminator to provide intrinsic rewards for policy learning to guide exploration. The intrinsic reward is defined as:

r¯(s,a,s)=+1,D(s)>1--1,D(s)1-% \epsilon\\ -1,&D_\xi(s^\prime)over¯ start_ARG italic_r end_ARG (italic_s , italic_a , italic_s start_POSTSUPERSCRIPT end_POSTSUPERSCRIPT) = s)\right],\bar\mathcalD_i^k% \subset\bar\mathcalD_i,k=1,2,...,K,roman_min start_POSTSUBSCRIPT italic_ start_POSTSUBSCRIPT italic_k end_POSTSUBSCRIPT end_POSTSUBSCRIPT start_UNDERACCENT italic_s , italic_a over¯ start_ARG caligraphic_D start_POSTSUBSCRIPT italic_i end_POSTSUBSCRIPT end_ARG start_POSTSUPERSCRIPT italic_k end_POSTSUPERSCRIPT end_UNDERACCENT start_ARG blackboard_E end_ARG [- roman_log italic_ start_POSTSUBSCRIPT italic_ start_POSTSUBSCRIPT italic_k end_POSTSUBSCRIPT end_POSTSUBSCRIPT (italic_a