Semi-reward Function Problems in Reinforcement Learning

Dong-geon Lee and Hyeoncheol Kim

Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea.

Abstract. Applying reinforcement learning agents to the real-world is important. Designing the reward function has problems, especially when it needs to intricately reflect the real-world or requires burden human effort. Under such circumstances, we propose a semi-reward function. This system is intended that each agent can go toward an individual goal when a collective goal is not defined in advance. The semi-reward function, does not require sophisticated reward design, is defined by 'not allowed actions' in the environments without any information about the goal. A tutorial-based agent can sequentially determine actions based on its current state and individual goal. It can be learned through a semi-reward function and toward its own goal. For the combination of these two, we constructed training method to reach the goal. We demonstrate that agents trained in arbitrary environments could go toward it own goal even if they are given different goals in different environments.

Keywords: Reinforcement Learning, Reward Function, Reward Engineering, Transformer-based Agent, Goal-based Agent

1 Introduction

Reinforcement Learning (RL) has demonstrated utility in addressing numerous real-world problems [1][2], yet many issues remain unresolved. One particularly challenging aspect is the application of transfer-trained policies or agents to real-world scenarios [3]. This difficulty often stems from the dichotomy between the need for accuracy and the multifaceted nature of reality, i.e., the multiplicity and complex interconnectivity of factors in the real world pose inherent limitations to accurately designing models or reward functions.

A prominent concern lies in the design or engineering of a reward function [4], given that accurately reflecting a real-world reward system in the design of a reward function proves exceedingly challenging. Despite recent remarkable achievements in RL and reward modeling, further efforts are still required to enhance these aspects [5][6]. Another challenge arises from the difficulty in uniformly generating goals within the environment. Even in identical environments, agents may need to pursue different goals, and multiple goals may be attainable. Strategies such as subdividing goals into sub-goals or training multi-agent systems to interact within the same environment are being explored, However, this factor underscores the challenge of

David C. Wyld et al. (Eds): DMSE, CSEIT, NeTCoM, SPM, CIoT, NCS, NLPD, ArIT, CMLA – 2024
pp. 249-268, 2024. CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.141420

transferring agents to real-world applications.

In this study, we introduce a semi-reward function[1] to alleviate the design burden associated with reward functions and goals. In contrast to conventional methods that primarily concentrate on learning the value of actions or behaviors upon goal attainment, this semi-reward function aims to train agents to pursue their goals exclusively by imposing penalties. Instead of specifically crafting rewards, this approach generally constrains undesired actions while imparting information about the goal, thereby reducing design complexity.

Nevertheless, agents still necessitate a model capable of receiving goal information as input and predicting appropriate actions based on it. Therefore, we propose a tutorial-based agent model that can learn according to a new reward function and advance toward its goals. This model undergoes training based on environmental observations and its own objectives, striving to navigate toward its goal from the current state within the same constraints.

Our contribution is original system of reward function and agent. Building upon partial observations, we propose an agent model wherein different agents can pursue their own unique goals, as well as a methodology to train them. Furthermore, we showcase the capability to deploy a trained agent in a previously un-seen environment.

2 Methodology

2.1 Related works

RL can be approached in various ways, with the two primary methodologies being value-based and policy-based [7]. Value-based RL entails estimating the values of actions or states themselves, given a state, and then performing actions based on this estimation. It is a fundamental learning method in RL, with models like Q-learning demonstrating success [8][9][10]. However, one limitation is the potentially extended duration required to estimate actions for all situations. In contrast, policy-based RL focuses on learning policies for selecting sequential actions in different situations [11]. Among these methods, a prominent policy optimization algorithm is Proximal Policy Optimization (PPO) [12]. PPO updates the policy while constraining its scope, facilitating the learning of stable policies across diverse scenarios and making it widely applicable in various domains.

Offline RL represents a form of RL where learning is conducted from generated trajectories without direct interaction with the environment [13]. In this approach, trajectories comprising sequential states and actions experienced by either an arbitrarily generated policy or a sufficiently trained expert interacting with the envi-

[1] https://github.com/Dong-geonLEE/Semi-reward-Function-Problems

ronment are stored in a buffer. Subsequently, the learning target utilizes only the trajectories stored in the buffer to refine the policy. Representative cases include [14] and [15]. One advantage of offline RL is its capacity to learn policies within a short timeframe without direct interaction with the environment. However, a drawback lies in its high dependency on generated data, posing challenges in adapting to unseen environments easily.

There are two main approaches to applying a transformer in RL. The first approach involves utilizing expert or human experience to generate demonstrations that are then used in supervised learning to train a transformer [16][17][18]. Demonstrations are used as labels, and the loss from them is used to train the transformer. Examples of this approach include the Decision Transformer [19] and Dreamer [20][21]. The second approach involves creating a world model for transfer learning, in which transformers are instantiated in the form of agents. This approach entails generating a world model that represents real-world problems and creating a model capable of learning features for transfer [22][23].

2.2 Problem Definition

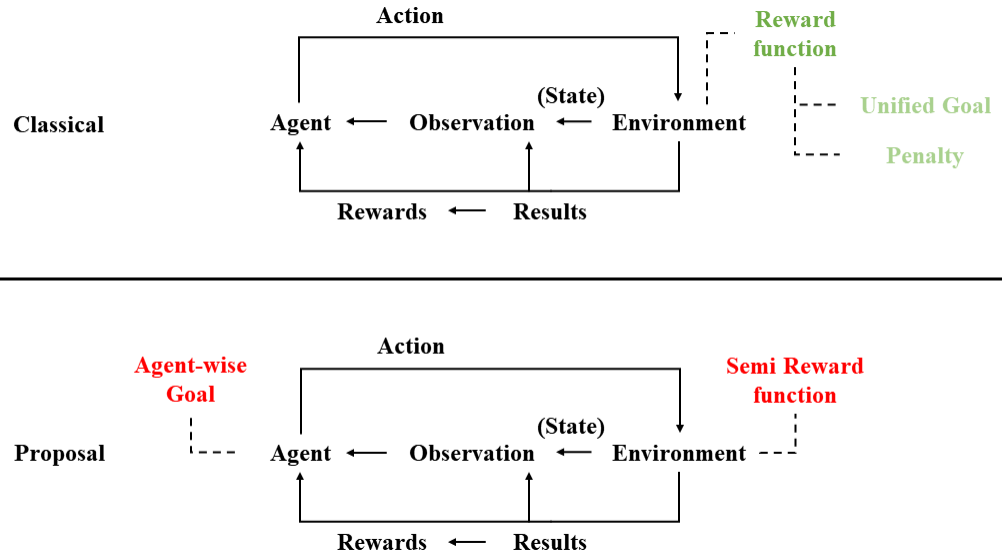

Fig. 1. Difference between classical RL and semi-reward function problems. In classical RL, the reward function, which generates the reward signal, exists as part of the environment and operates independently of the agent. In contrast, when applying a semi-reward function, the goal information from the reward function shifts toward the agent, thereby defining agent-wise goals.

This study addresses a scenario where goals are individually specified rather than collectively defined. Specifically, we examine an environment with predefined constraints such as input shape, objects, or disallowed actions. Upon encountering this environment, each new agent pursues its own distinct goal, which is established as one of the existing states within the environment. We denote this goal assigned to each agent as an **agent-wise goal**.

Because the environment lacks knowledge of the goal, when goals are allocated on an agent-wise basis, the environment cannot furnish a reward for successful goal attainment unless additional information about each agent is provided. Consequently, conventional reward functions prove inadequate. In an RL environment, the reward function is crafted to guide agent learning [26][27]. Thus, agents learn based on trajectories from episode inception to conclusion, guided by the reward signal.

Our novel reward function is termed **semi-reward function**. Diverging from a conventional reward signal, as **Fig.1**, the semi-reward function exclusively comprises information pertaining to penalties and maximum steps. In the case of goals, agents set their objective as reaching their individual goal, and upon determining that they have achieved their agent-wise goal, they receive a positive reward signal. At this juncture, the environment facilitates agents in conducting simulations while adhering to the constraints.

Pursuing an agent-wise goal entails training the agent toward its specific objective [28]. The reward function plays a crucial role in dictating agents' behavior within the environment [29]. With the modification of the reward function, the need for an agent aligned with the new function arises. Through the utilization of the semi-reward function, the agent assimilates two categories of environmental information: actions that contravene penalties and actions taken by the agent to achieve the goal.

2.3 Environment

The experiment utilized the MiniGrid environment [33], an RL library tailored for goal-oriented tasks. MiniGrid facilitates 2D grid image simulations, emphasizing environments conducive to agent self-control and comprehension of complex visual observation.

Within MiniGrid, diverse subenvironments can be generated to fulfill various missions. These missions encompass objectives such as reaching a goal object, locating a specific object of a particular color, acquiring a key within a room, or accessing objects in separate rooms via opened doors. The agent undertakes actions aligned with the designated mission, interacting with a range of objects including doors, keys, and boxes, while executing maneuvers such as turning left, turning right, picking up objects, or toggling them. MiniGrid served as the environment for semi-reward function experiments for the following reasons:

1. MiniGrid offers extensive flexibility in object creation, enabling the designation of agent-wise goals to states existing within the environment. In essence, to furnish agents with a variety of goals for generalization, the environment must encompass diverse states.
2. The environment and agent adhere to identical constraints regardless of the task or mission at hand. Subenvironments within MiniGrid share commonalities in actions, observations, and objects, constituting a cohesive framework conducive to transferring trained agents across different subenvironments under uniform constraints.
3. MiniGrid facilitates the provision of partial observations, a critical aspect for addressing the challenge of agent-wise goals. To assess observations without additional environmental context beyond the goal, the environment must present disparities between the agent's state within the environment and the observations provided.

Althogh theoretical RL uses information of both penalty and reward signal, practical RL environment is implemented with reward signal when agent reaches the goal, otherwise zero rewards when agent reaches step limits or failure defined moments. MiniGrid is one of that environment, for instance, in 'LabaGap[2]' reward is defined as 'a reward of '$1 - 0.9 * (step\_count/max\_steps)$' is given for success, and '0' for failure', and termination condition is defined as 1. the agent reaches the goal. 2. the agent falls into lava. 3. Timeout (see max_steps).

3 Tutorial-Based Model

Consequently, there arises a necessity for a trainable agent capable of adapting to a newly formulated reward function. Specifically, to employ a semi-reward function effectively, agents capable of pursuing agent-wise goals based on provided observations and training methodologies conducive to achieving such goals are indispensable. Trained agents must possess the capability to ascertain the status of their goals, including identifying relevant objects or tasks. In light of this, along with observation outcomes, agents must discern the requisite actions to attain their objectives.

We advocate for a **tutorial-based model** to serve as the framework for these adaptable agents. Analogous to attending tutorial sessions when delving into an unfamiliar domain and subsequently delving into more advanced coursework, this model empowers agents to undertake diverse missions predicated on their acquired knowledge of objects and actions within a given environment.

The flowchart depicting the tutorial-based model is delineated in **Figure 2**. Within

[2] https://minigrid.farama.org/environments/minigrid/LavaGapEnv/

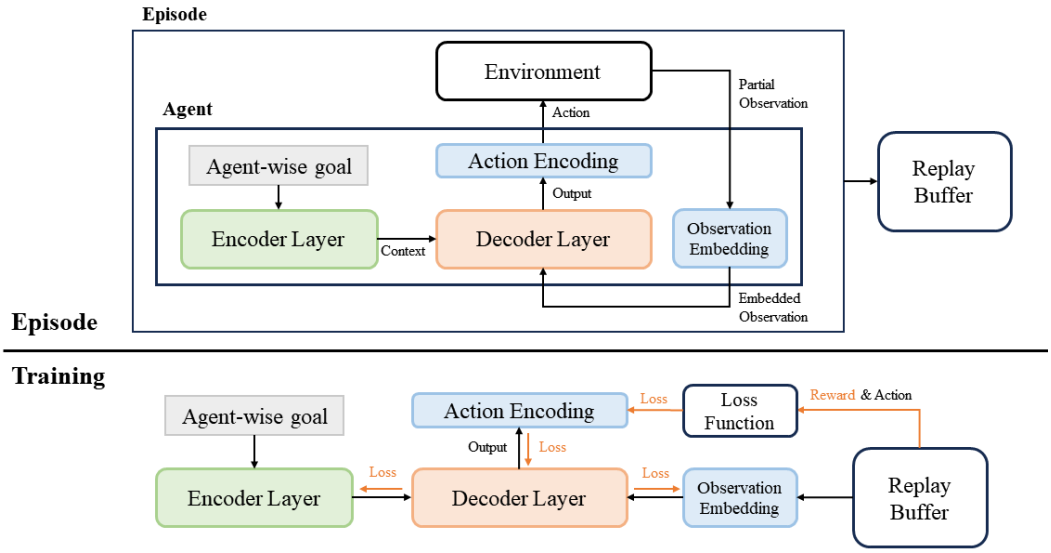

Fig. 2. Diagram of tutorial-based model. In the tutorial-based model, the agent adopts a transformer-based encoder-decoder architecture. The primary role of the agent is to predict the optimal actions based on the partial observation provided by the environment and the agent's individual goal. The agent goes through two main stages: episodes and training, iteratively during which it learns. Learning occurs akin to conventional deep learning approaches, using loss functions for training.

the training paradigm, agents undergo iterative learning cycles alternating between episodes and training stages. Here, an agent refers to an encoder-decoder structure equipped with embedding layers, partitioned into an encoder responsible for processing agent-wise goals and extracting pertinent information, and a decoder tasked with determining actions based on the current observation input.

During the episode stage, the agent determines actions based on observations and agent-wise goal information while maintaining frozen parameters. Beginning from the initial state until the episode's conclusion, the agent sequentially selects actions, acquires observations, actions, rewards, and additional episode-related information, storing these data in a buffer. Storage transpires on an episode-by-episode basis, with the process persisting until the goal is reached or actions cease due to specified conditions.

In the subsequent training stage, the agent parameters were updated on an episode-by-episode basis. Leveraging the information stored in the buffer from the episode stage, the loss value is computed, and backward propagation is executed. At this juncture, the encoder, decoder, and embedding layers undergo upgrades.

Algorithm 1 Algorithm to training method of tutorial-based model

```
Require: Environment Env, Agent-wise goal Goal, Agent Agent, Total Steps T
  embedded_goal = Agent.observation_embedding(Goal)
  while steps < T do
      Buffer = Agent.episode_buffer.__init__()
      observation = Env.reset()
      while not done do
          action = Agent.predict(observation, embedded_goal
          Env.step(action)
          Buffer[t] = observation, action
          t, step += 1
      end while
      Clip = Buffer.cliping()
      for step in range(Clip) do
          output = Agent.predict(Buffer[step, observation], embedded_goal)
          loss = Agent.calculation_loss(Buffer[step, reward])
          masked_output = Agent.masking(action, output, loss)
          loss.backward()
      end for
  end while
```

The iterative nature of the learning process, which was divided into two stages. Analogously, it entails collecting offline data from the environment, storing it, and iteratively computing losses and refining policies. The specific algorithm encapsulating the tutorial-based model is outlined as **Algorithm 1.**

The entire training procedure is bifurcated into two stages to compute the loss value utilizing the reward signal. Given the inherent delay in reward acquisition, rewards are exclusively dispensed at episode terminations. Consequently, episodes are conducted until their culmination, following which parameter updates occur.

3.1 Environment with Semi-reward Function

A key distinction lies in the origin of rewards: negative rewards are inherently provided by the environment, while positive rewards are determined by the agent. Furthermore, in instances where an episode is prematurely terminated upon reaching the maximum step limit (termed "truncated"), the most significant penalty is imposed. This feature is designed to prevent the agent from becoming passive. Initially, the agent learns through trial and error, experiencing failures and garnering negative rewards enroute to its goal. Throughout this learning process, if the penalty incurred from truncation exceeds that of negative signals, the agent may opt for overly cautious actions.

Other environmental components remain unchanged. The agent initialization method, state transitions, and provision of partial observations to the agent remain consistent. However, a conceptual shift is required for agent-wise goals. In classical RL,

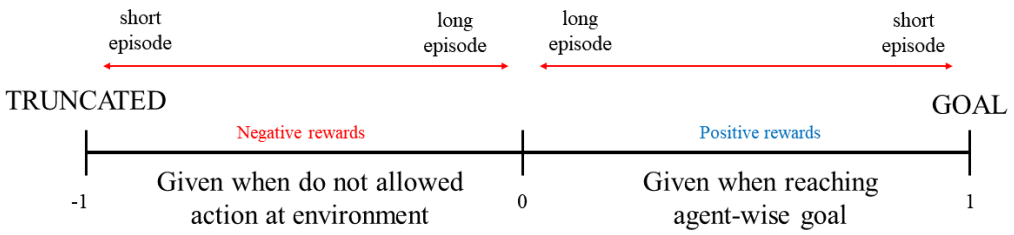

Fig. 3. Components of the semi-reward function at a scale of [-1, 1]. This figure illustrates the representation of a semi-reward function within a specific range. Rewards resulting in outcomes less than 0 are designated as negative rewards, while those greater than 0 are termed positive rewards. Negative rewards are assigned when the agent takes actions not permitted by the environment, whereas positive rewards are self-assigned by the agent upon reaching its goal.

it is presumed that no environmental information is provided to the agent. Conversely, setting agent-wise goals necessitates furnishing fundamental information about the environment's objects. Endeavors such as attempting to catch fish while scaling mountains should be avoided.

3.2 Agent

The agent for the semi-reward function is constructed based on a transformer architecture [34]. Several factors influenced this choice. Firstly, it necessitates the ability to generate agent-wise goal information and convey it effectively to the agent. This entails a model capable of accommodating diverse inputs, observations, and goals in varying formats. Moreover, transformers excel in handling long-term dependencies and facilitating parallel processing of inputs. They are extensively utilized in pretraining and transfer-learning scenarios.

The agent determines suitable actions for the current state by assimilating both partial observations and its own goal. Action determination is facilitated through an encoder-decoder model. Optimal actions are those that adhere to environmental constraints within the given observation and facilitate progress toward the agent-wise goal.

The encoder, serving as a context extractor, receives input information pertaining to the agent-wise goal at each action prediction step. This process is independent of observations and is engineered to accommodate distinct goals for each agent. On the other hand, the decoder employs the contextual information extracted by the encoder and the partial observation at each step to ascertain the optimal action.

The scalability of the agent is achieved by adjusting the numbers of encoder and decoder layers, allowing adaptation to the complexity of the environment, missions,

and goals. This scalability enables the agent to navigate diverse environments effectively. Typically, as the environmental size increases, along with the number of feasible actions and discoverable objects, additional layers become imperative to ensure robust performance.

Action Prediction The output of the decoder represents the predicted value for each action. For instance, in an environment offering three available actions, the decoder's output size is (1x3), with each number denoting the predicted value for each action based on the agent-wise goal and observation. During the episode stage, this output serves as a weight for each action post-application of the softmax function, thereby stochastically determining the actions.

Stochastic action selection is integral to the transformer architecture's utilization. It prevents consecutive selection of the same action during action prediction directly or in a greedy manner. Stochasticity fosters exploration by introducing opportunities to choose novel actions, even in analogous situations. Furthermore, it substantially amplifies differences in output values among actions.

Observation Embedding The input shape to the transformer is a 1D vector. Consequently, it necessitates an embedding process to convert the 2D partial observation provided by the environment into a 1D vector format. This embedding process, termed observation embedding, serves to transform each step performed within an episode into a format compatible with single-token input. Post-embedding, each scene within a step is converted into a singular vector, subsequently fed into the decoder. To facilitate this, we established an embedding layer concurrently trained with the transformer. This embedding layer functions to input observations for action prediction within an episode or to input the agent-wise goal into the encoder.

Agent-wise Goal To establish agent-wise goals, we employed a methodology wherein the agent's desired state is translated into goal information for input. The most straightforward approach entails converting the agent's position within the observation into the position of the object it aims to reach. For instance, when presented with a 7x7 partial observation and the agent's position is (3, 6), we substitute the agent object with the goal object at that specific position.

3.3 Loss

Labels are not employed in the training process of the tutorial-based agent. In supervised forms of RL, such as imitation learning and offline RL, loss calculation involves computing the disparity between the predicted output and the label values of the data. However, in models targeting agent-wise goals, generating trajectories

using labels is impractical, rendering the conventional loss calculation method inapplicable. Instead, a tutorial-based agent derives loss based on the reward.
If R_E represents the reward for episode E, the loss at step t can be calculated as follows:

$$R_t = -R_E \times \alpha^{(T-t)}$$
$$L_t = (R_t)^3$$

where α is a hyperparameter determining the decay rate, and T is the total number of steps performed int the episode. In the first step, the sign of R_E is inverted so that episodes receiving negative rewards yield positive losses, and vice versa. Furthermore, the decay rate, escalating to the power of the gap between the total number of steps and the current step, is multiplied. This amplifies the loss for steps nearing the episode's conclusion, reflecting the typical observation that rewards tend to escalate as the goal is approached.

A loss function accommodating both negative and positive rewards is indispensable. Conventional loss functions, like the mean squared error (MSE), often converge to a single point, potentially impeding goal achievement if utilized directly. Therefore, a novel loss function is requisite. To address both reward types, a loss function with opposing signs and a symmetrical shape was adopted, primarily employing the cubic form. Given the semi-reward function's range of [-1, 1], employing a corresponding loss function prevents loss divergence to infinity. Detailed alterations induced by the loss function are delineated in the Appendix, as **Figure 5**.

3.4 Training Methods

During the training stage, parameter updates rely on information gathered in the episode stage. This section elucidates the learning process during the training phase. To update parameters via back-propagation of the loss, determining each parameter's contribution to the output is essential. While storing parameter values for every step is plausible, it imposes a considerable memory burden. Thus, in this model, we adopt the parameter recalculation method.

Observations are stored in a buffer, with identical values re-inputted without parameter updates. Consequently, we obtain parameter values utilized for action prediction. Employing this approach, we derive parameter values and solely update parameters involved in predicting the action chosen at that step, while masking other actions during the training phase.

Updating parameters at every step can potentially lead to divergence errors if the training stage becomes overly protracted or if the size of the propagated gradients significantly increases. Therefore, we opt for parameter updates by clipping the episode and focusing on only a portion of the entire process. If the end-of-episode

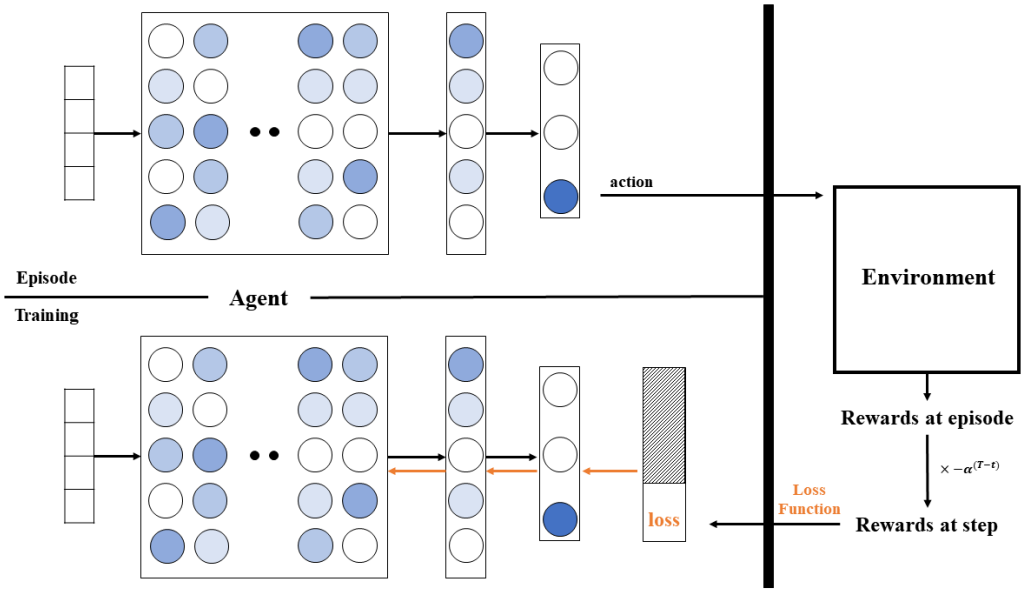

Fig. 4. Training methods for tutorial-based model. It is the process of updating the agent's parameters through rewards at the same step of the episode and training stage. The agent recalculates the output paramter used for action prediction during the training stage. After that, agent backpropagates the loss converted from the obtained reward. The agent can reflect the contribution of the parameter in action prediction. Due to the delayed reward, is should be noted that the episode and training stage do not occur simultaneously in the actual learning process.

(EOE) step is smaller than the clipping step, the entire episode is stored in the buffer. Conversely, if the EOE step surpasses the clipping step, only consecutive steps up to the clipping size are utilized.

A notable departure from classical RL is that this model does not engage in random action selection [27]. This is different from stochastic action selection at **3.2**, because it means agent selects actions sorely randomly without transformer's parameter calculation and action prediction. Random action selection is usually used to implements exploration in RL environment. The process involves accumulating updated parameters using loss, and exploration may impede parameter convergence. However, it dose not mean that exploration not occur. From the initialization of parameters and during action prediction and traninig stage, agent will experience diverse actions at various states.

On the other hand, the necessity to design a semi-reward function for training the tutorial-based agent is also intertwined with random action selection. If negative rewards are excluded, leaving only rewards between 0 and 1, there exists a high

likelihood that only the action initialized with the largest value will be consistently chosen. In most instances, the reward attained in this scenario is zero, resulting in a loss of zero throughout the entire training phase. Consequently, these parameters remain unchanged.

4 Experiment

The purpose of this experiment was to observe the results when an agent trained in a specific environment within MiniGrid was tested in a different subenvironment. Different environments refer to variations such as changes in the size of the environment while maintaining the same mission or providing different missions altogether. Through this, we aim to demonstrate that even without uniformly generating rewards in the environment, it is possible to train agents that pursue different goals using semi-reward functions and a tutorial-based agent. The experiment compared two models:

1. The first policy was trained using PPO [12] in the original MiniGrid environment. Policy was trained by Stable-baselines3[3] library [36].
2. The second model was a tutorial-based model, trained with the application of a semi-reward function in the MiniGrid[4] environment.

In the existing MiniGrid environment, apart from scenarios involving specific obstacles like obstacle objects and lava, a predefined reward is given upon completing the mission. If the mission is not completed and the maximum number of steps is reached, a reward of zero is assigned. However, we defined some additional negative rewards as illustrated in the following examples:

- Attempting to execute a forward action while facing a wall.
- Attempting to interact with an object (e.g., a wall or floor) that cannot be toggled.
- Attempts to drop an item without holding anything.

Additionally, as mentioned, a negative reward of -1 is assigned when reaching the maximum step limit. Other penalties (such as obstacles, lava) are working same with existing MiniGrid. The hyper-parameters used in expertiments are in appendix, as **table 4, 5.** Additionally, during episode generation, stochastic action selection based on the output of the decoder is used. However, during evaluation, the action with the highest parameter value was selected.

As previously mentioned, each experiment involved training a model in one environment and subsequently transferring the agent to a different environment to

[3] https://stable-baselines.readthedocs.io/en/master/index.html
[4] https://minigrid.farama.org/

evaluate its performance. In all evaluations, the agents were compared based on **the average and variation of rewards** from multiple episodes. It should be noted that PPO has a reward value range of [0, 1], while OURS has a reward value range of [-1, 1]. Difference of range is beacuse, as will be mentioned in 4.4, two models are not appropriates in each others' environments.

For the comparison two models, the higher average reward fundamentally means the agent have more proper sequential actions to achieve goals in evaluation. In usual, the high rewards' variation means that there are not only successful episodes but failure episodes in evaluations. On the contrary, if variation is shown around 0, it means agents could reach the goal stably. For instance, if there are high average reward around 1 and low std reward, it could be the result which agent could reach the goal stably in every episodes of evaluations. Despite of difference range, two agents could compared with this standards.

In the follow results, there are max steps of each agent in each sub-environments and it sizes. Each max steps means that the minimum max steps of learning to measure the performance; in other words, could reach the goal for any episodes in evaluations. In this experiment, we would like to evaluate the performance of completely learned agent. Therefore, we compared each agent with the average and variation of rewards. However, change of average reward according to the steps in training stage would shown either the performence of the agent and the stability of learning. This aspect, in tutorial-based agent, is mentioned in **ch 5**. In conclusion, cause of the instability in training stage, we evaluated tutorial-based agent in early steps, therefore the max steps appears in difference.

4.1 Evaluation at Same Goal Object

First of all, the basic and simplest environment is used. In this part, the same agent-wise goal is given in every agent and episode, but changed the size of the grid of the environment. Basically, through a combination of semi-reward function and tutorial-based agent, training an agent that performs the given mission was possible. However, when evaluating the agent in the training environment itself, the OURS agent obtained a lower average reward than the PPO agent, as shown in **Table 1**.

The performance of the agent changes when the size of the environment changed. Trained agents show different patterns at small sizes (5x5, 6x6) and large sizes (8x8, 16x16). In the case of small- sized grids, PPO agents show high performance, but they cannot reach the goal at all in large-sized environments, whereas OURS agents show consistent performance in all sizes. Conversely, in the case of large-sized grids, PPO agents show consistent performance in all sizes, whereas OURS agents show decreasing performance as the size decreases.

Table 1. Evaluation results from same goal object environments. Agent is evaluated in environments with the same goal object while only changing the size of the grid. The training subenvironment is named 'Empty', where an empty grid space with a green box goal object is provided without any obstacles. Each row represents the trained environment, and each column represents the environment used in evaluation. The Initialization of agent's start position was randomly determined in OURS at all sizes and PPO at 5x5, 6x6, while for PPO at 8x8 and 16x16, it was fixed.

		Eval Env	**Empty**							
Training Env		Method (max steps)	5x5		6x6		8x8		16x16	
			MEAN	STD	MEAN	STD	MEAN	STD	MEAN	STD
Empty	5x5	**PPO (5e4)**	0.9613	0.0081	0.4893	0.4894	0	0	0	0
		OURS (1e5)	0.7075	0.0811	0.7525	0.0986	0.7615	0.0663	0.4563	0.1938
	6x6	**PPO (5e4)**	0.9712	0.0097	0.87	0.2902	0	0	0	0
		OURS (1e5)	-0.883	0.036	0.7413	0.0812	0.6447	0.5495	0.6193	0.5536
	8x8	**PPO (5e6)**	0.9568	0.0244	0.8613	0.2874	0.9613	0	0.9762	0
		OURS (1e5)	-0.8965	0.0699	-0.874	0.0463	0.8481	0.0506	0.7638	0.1209
	16x16	**PPO (5e6)**	0.2928	0.4446	0.9318	0.0444	0.9578	0	0.9754	0
		OURS (2e5)	-0.18	0.7981	0.3705	0.6833	0.7840	0.0893	0.5683	0.5372

4.2 Evaluation at Different Goal Object

Table 2. Evaluation results from random goal object environments. The agent trains in the same subenvironemnt used in **4.1**, but evaluates in other subenvironments with same penalties. In the training environment, the goal object is given as the same object in all episodes. However, in the evaluation environment, a random object on random grid is generated, and the mission is to locate that object. This subenvironment dose not exist in MiniGrid, PPO agents are also evaluated in environment with semi-reward function.

		Eval Env	**Random Object**							
Training Env		Method (max steps)	5x5		6x6		8x8		16x16	
			MEAN	STD	MEAN	STD	MEAN	STD	MEAN	STD
Empty	5x5	**PPO (1e5)**	0.7514	0.5841	-0.03	0.9706	-0.4101	0.9010	-1.0	0
		OURS (1e5)	-0.0342	0.9485	0.104	0.8956	0.7085	0.5717	0.6235	0.7548
	6x6	**PPO (1e5)**	0.7604	0.587	0.9295	0.0242	0.1651	0.9514	-0.014	0.9860
		OURS (1e5)	-0.7478	0.5181	0.101	0.9015	0.319	0.8656	0.454	0.8348
	8x8	**PPO (5e5)**	0.7442	0.5817	0.9280	0.0334	0.7505	0.584	-0.2124	0.9647
		OURS (5e5)	-0.764	0.53	-0.558	0.7159	0.6019	0.7359	0.2475	0.9138

The experiment in this part attempted to show whether the trained agent could reach a different agent-wise goal in the new episode. For this purpose, the evaluation environment has the same type of mission but a different agent-wise goal with other types of goal object. The results are in **table 2**, similar to those at **4.1**. The performance reduction of the OURS agent is relatively small when evaluated on a larger grid, but conversely, the reduction of the PPO agent was small as it went to

the smaller sizes.

The OURS agent shows different results according to the grid size of the environment. Agents trained in relatively small-sized environments guarantee performance even when transferred to large-sized environments. However, agents trained in relatively large-sized environments show sharp performance reductions when transferred to small-sized environments. This is presumed to occurs because the agent is given a fixed size of partial observations. In small-sized environments, the majority of observations will be padded, and only a very small portion will contain information compared to the pattern of observation given in a large environment.

4.3 Evaluation at Other Subenvironment

Table 3. Evaluation results from three environments. The agent trained in one environment was evaulated in multiple environments. There are a total of three evaluation environments, and all objects and penalties in each environment were limited to those that could be experienced in the training environment. The evaluation was conducted with the agent that obtained highest average reward in the training environment. In the DistShift, on the 8x8 size grid, the version varies depending on the position of obstacles.

Training Env		**Eval Env**	**Empty**							
		Method	5x5		6x6		8x8		16x16	
		(max stes)	MEAN	STD	MEAN	STD	MEAN	STD	MEAN	STD
Dist Shift	ROW2	**PPO (5e6)**	0.0911	0.2973	0	0	0	0	0	0
		OURS (1e5)	-0.537	0.6582	-0.8245	0.0387	0.6582	0.5339	0.6933	0.1167
	ROW4	**PPO (5e6)**	0	0	0	0	0	0	0	0
		OURS (1e5)	-0.8335	0.0405	-0.9001	0.0622	0.4672	0.7283	0.5735	0.5046
		Eval Env	**DistShift**							
		Method	ROW 2				ROW 4			
		(max steps)	MEAN		STD		MEAN		STD	
	ROW2	**PPO (5e6)**	0.9536		0		0		0	
		OURS (1e5)	0.7536		0.0163		0.7671		0.0447	
	ROW4	**PPO (5e6)**	0		0		0.9607		0	
		OURS (1e5)	0.6031		0.5272		0.7716		0.048	
		Eval Env	**LavaGap**							
		Method	5x5		6x6		7x7			
		(max steps)	MEAN	STD	MEAN	STD	MEAN	STD		
	ROW2	**PPO (5e6)**	0	0	0	0	0	0		
		OURS (1e5)	-0.406	0.8233	-0.526	0.7106	-0.7135	0.5364		
	ROW4	**PPO (5e6)**	0	0	0	0	0	0		
		OURS (1e5)	-0.925	0.0361	-0.7393	0.5614	-0.964	0.0278		

In this part, the evaluation environment is different from the training environments, providing a different type of mission and agent-wise goal to the agent. For

each environment, objects and penalities are limited to those that could be experienced in the training stage. It is reasonable that the trained agent does not perform well in the evaluation environment where there are more objects or situations that have not been experienced before. In the case of OURS agents in the Empty, even if they were trained in DistShift, which is a new type of subenvironments, they shows similar results as in the **4.1**. As a consequence, as in **table 3**, the tutorial-based agent guarantees performance in a different environment if it has been experienced before.

PPO agents show high performance in the environment where each agent is trained, but not all of them reach their goals in other environments. However, in the case of PPO, it is a method of optimizing policies rather than predicting the value of observations. Thus, it is a method of learning how to reach the goal within trained environment. Therefore, rather than interpreting that the PPO agent was not trained well, it seems reasonable to conclude that predicting the value of observations and actions is more suitable to toward diverse agent's goal.

In the case of the lavagap environment, OURS agent is either not achieving the goal at all (ROW4), or not reliably reaching the goal (ROW2). Based on the results of previous experiments, it can be assumed that this was mostly caused by the small size of the evaluation environment compared to the training environment.

4.4 Combinations of Policies and Environments

The results of existing RL models' learning and evaluation in a semi-reward function environment were not recorded. Briefly to conclude, learning did not occur when agent used existing learning methods. It could be assumed that this is occurred becuase experiencing success in a semi-reward function environment is very difficult. In existing RL, the learning method could be thought to focus on finding and following the successful path first, since rewards are given when they are successful. However, in the case of semi-reward, there are incomparably more cases where learning is stopped before achieving success. Taking this into account, in order to utilize the semi-reward function, a special agent such as a tutorial-based one is required, and vice versa. Therefore, only comparative experiments were conducted on agents learned with PPO algorithms.

5 Limitations

The tutorial-based agent with a semi-reward function exhibits the following limitations:

1. The training agent displays instability. Instances occur during training where parameters diverge to infinity or become trapped in local minima, hindering

convergence. Moreover, there exists a notable variance in learning speed depending on initial values. Instability may arise from the introduction of loss through an external module, generated separately and input into the system using a different method.

2. A distinct method is necessary to enable agents to autonomously assess their goals. Currently, agent-wise goal setting is implemented by creating goals based on the mission. Particularly for the 'DoorKey' mission, it was divided into three stages and provided sequentially to the agent at each state as sub-goals.
3. Adjustment of the maximum steps per episode is required. The current setting is less than the typical maximum number of steps used. While training with algorithms such as PPO, even if the maximum number of steps was higher, the number of steps tended to decrease as the number of episodes increased. However, this model fails to determine the optimal steps, and the number of steps tends to remain consistent.

6 Conclusion

Utilizing a semi-reward function reduces the engineering effort required to define only the disallowed actions in an environment. Additionally, under the same constraints, the tutorial-based agent can provide some guarantee of performance even when transferred to new environments. With combination of semi-reward functions and tutorial-based agent, the agent toward its own goal could be created and trained.

References

1. Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., and Henighan, T.: 'Training a helpful and harmless assistant with reinforcement learning from human feedback', arXiv preprint arXiv:2204.05862, 2022.
2. Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R Ruiz, F.J., Schrittwieser, J., and Swirszcz, G.: 'Discovering faster matrix multiplication algorithms with reinforcement learning', Nature, 2022, 610, (7930), pp. 47-53.
3. Zhu, Z., Lin, K., Jain, A.K., and Zhou, J.: 'Transfer learning in deep reinforcement learning: A survey', IEEE Transactions on Pattern Analysis and Machine Intelligence, 2023.
4. Knox, W.B., Allievi, A., Banzhaf, H., Schmitt, F., and Stone, P.: 'Reward (mis) design for autonomous driving', Artificial Intelligence, 2023, 316, pp. 103829.
5. Baker, B., Akkaya, I., Zhokov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., and Clune, J.: 'Video pretraining (vpt): Learning to act by watching unlabeled online videos', Advances in Neural Information Processing Systems, 2022, 35, pp. 24639-24654.
6. Kramár, J., Eccles, T., Gemp, I., Tacchetti, A., McKee, K.R., Malinowski, M., Graepel, T., and Bachrach, Y.: 'Negotiation and honesty in artificial intelligence methods for the board game of Diplomacy', Nature Communications, 2022, 13, (1), pp. 7214.
7. Arulkumaran, K., Deisenroth, M.P., Brundage, M., and Bharath, A.A.: 'Deep reinforcement learning: A brief survey', IEEE Signal Processing Magazine, 2017, 34, (6), pp. 26-38.
8. Hasselt, H.: 'Double Q-learning', Advances in neural information processing systems, 2010, 23.

9. Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., and Osband, I.: 'Deep q-learning from demonstrations', Proceedings of the AAAI conference on artificial intelligence.: 'Book Deep q-learning from demonstrations' (2018, edn.), pp.
10. Kostrikov, I., Nair, A., and Levine, S.: 'Offline reinforcement learning with implicit q-learning', arXiv preprint arXiv:2110.06169, 2021.
11. Haarnoja, T., Tang, H., Abbeel, P., and Levine, S.: 'Reinforcement learning with deep energy-based policies', International conference on mahine learning.: 'Book Reinforcement learning with deep energy-based policies' (PMLR, 2017, edn.), pp. 1352-1361.
12. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O.: 'Proximal policy optimization algorithms', arXiv preprint arXiv:1707.06347, 2017.
13. Levine, S., Kumar, A., Tucker, G., and Fu, J.: 'Offline reinforcement learning: Tutorial, review, and perspectives on open problems', arXiv preprint arXiv:2005.01643, 2020.
14. Fujimoto, S., Meger, D., and Precup, D.: 'Off-policy deep reinforcement learning without exploration', International conference on machine learning.: 'Book Off-policy deep reinforcement learning without exploration' (PMLR, 2019, edn.), pp. 2052-2062.
15. Kumar, A., Zhou, A., Tucker, G., and Levine, S.: 'Conservative q-learning for offline reinforcement learning', Advances in Neural Information Processing Systems, 2020, 33, pp. 1179-1191.
16. Badrinath, A., Flet-Berliac, Y., Nie, A., and Brunskill, E.: 'Waypoint transformer: Reinforcement learning via supervised learning with intermediate targets', Advances in Neural Information Processing Systems, 2024, 36.
17. Hua, J., Zeng, L., Li, G., and Ju, Z.: 'Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning', Sensors, 2021, 21, (4), pp. 1278.
18. Huang, W., Zhou, Y., He, X., and Lv, C.: 'Goal-guided transformer-enabled reinforcement learning for efficient autonomous navigation', IEEE Transactions on Intelligent Transportation Systems, 2023.
19. Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I.: 'Decision transformer: Reinforcement learning via sequence modeling', Advances in neural information processing systems, 2021, 34, pp. 15084-15097.
20. Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M.: 'Dream to control: Learning behaviors by latent imagination', arXiv preprint arXiv:1912.01603, 2019.
21. Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J.: 'Learning latent dynamics for planning from pixels', International conference on machine learning.: 'Book Learning latent dynamics for planning from pixels' (PMLR, 2019, edn.), pp. 2555-2565.
22. Chebotar, Y., Vuong, Q., Hausman, K., Xia, F., Lu, Y., Irpan, A., Kumar, A., Yu, T., Herzog, A., and Pertsch, K.: 'Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions', Conference on Robot Learning.: 'Book Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions' (PMLR, 2023, edn.), pp. 3909-3928.
23. Zhang, W., Wang, G., Sun, J., Yuan, Y., and Huang, G.: 'STORM: Efficient stochastic transformer based world models for reinforcement learning', Advances in Neural Information Processing Systems, 2024, 36.
24. Li, W., Luo, H., Lin, Z., Zhang, C., Lu, Z., and Ye, D.: 'A survey on transformers in reinforcement learning', arXiv preprint arXiv:2301.03044, 2023.
25. Szepesvári, C.: 'Algorithms for reinforcement learning' (Springer nature, 2022. 2022).
26. Shakya, A.K., Pillai, G., and Chakrabarty, S.: 'Reinforcement learning algorithms: A brief survey', Expert Systems with Applications, 2023, pp. 120495.
27. Sutton, R.S., and Barto, A.G.: 'Reinforcement learning: An introduction' (MIT press, 2018. 2018).
28. Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S.J., and Dragan, A.: 'Inverse reward design', Advances in neural information processing systems, 2017, 30.
29. Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S.: 'Diversity is all you need: Learning skills without a reward function', arXiv preprint arXiv:1802.06070, 2018.

30. Liu, M., Zhu, M., and Zhang, W.: 'Goal-conditioned reinforcement learning: Problems and solutions', arXiv preprint arXiv:2201.08299, 2022.
31. Andres, A., Villar-Rodriguez, E., and Del Ser, J.: 'Towards Improving Exploration in Self-Imitation Learning using Intrinsic Motivation', 2022 IEEE Symposium Series on Computational Intelligence (SSCI): 'Book Towards Improving Exploration in Self-Imitation Learning using Intrinsic Motivation' (IEEE, 2022, edn.), pp. 890-899.
32. Wan, S., Tang, Y., Tian, Y., and Kaneko, T.: 'DEIR: efficient and robust exploration through discriminative-model-based episodic intrinsic rewards', arXiv preprint arXiv:2304.10770, 2023.
33. Chevalier-Boisvert, M., Dai, B., Towers, M., Perez-Vicente, R., Willems, L., Lahlou, S., Pal, S., Castro, P.S., and Terry, J.: 'Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks', Advances in Neural Information Processing Systems, 2024, 36.
34. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., and Polosukhin, I.: 'Attention is all you need', Advances in neural information processing systems, 2017, 30.
35. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K.: 'Bert: Pre-training of deep bidirectional transformers for language understanding', arXiv preprint arXiv:1810.04805, 2018.
36. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N.: 'Stable-baselines3: Reliable reinforcement learning implementations', Journal of Machine Learning Research, 2021, 22, (268), pp. 1-8.

Authors

Dong-geon Lee Currently he is pursuing his Masters in Computer Science from the Korea University. His research interests include reinforcement learning and machine learning.

Hyeoncheol Kim obtained his PhD in 1998 from the University of Florida, US in the area of computer & information science & engineering. He is currently serving the Korea University, from 1999. He has been research on rule extraction algorithms in shallow feed-forward neural networks and rule based knowledge extraction or interaction with machines. He also focused on machine learning, knowledge-based explainable AI and education in computer science and AI.

Appendix

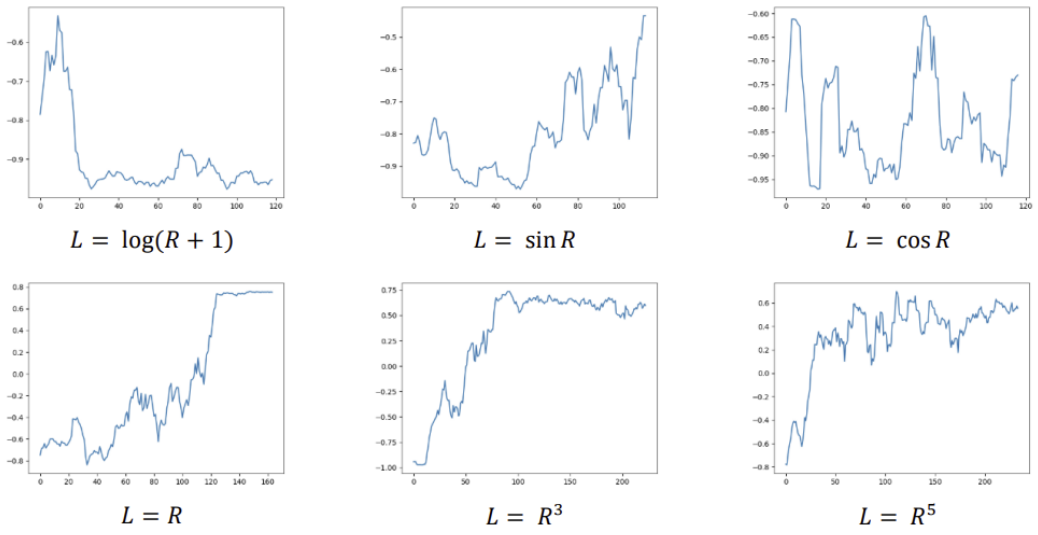

Fig. 5. Rewards in entire training process by loss functions. Average rewards during entire training with differnet loss functions at same training environment. In experiments, we selects the most stable loss functions.

Table 4. Manuscripts of hyperparameters used in tutorial-based agent.

Number of encoder / decoder layers	The number of encoder/decoder layers in the tutorial-based agent. In most experiments, maintaining the same number of layers increases performance.
Small value for action masking	A small value is used for action masking when updating the parameters during the training stage.
Maximum steps	Number of maximum steps in one episode.
Total steps	Total steps of entire learning process.
Learning rate	Learning rate.
Number of actions	Number of actions defined in the environment.
Size of observation embedding	The size of observation embedding input into the agent. Increasing when size of partial observation increases contributing to performace improvement.

Table 5. Hyperparameters used for training in Experiments.

Hyperparameters	Empty				RandomObj				DistShift	
	5x5	6x6	8x8	16x16	5x5	6x6	8x8	16x16	row=2	row=4
Number of encoder layers	2	3	4	4	2	3	4	5	4	4
Number of decoder layers	2	3	4	4	2	3	4	5	4	4
Small value for action masking	1e-5			5e-5	1e-5	1e-6	1e-8		1e-7	
Maximum steps	20	40	60	100	50	60	80	120	80	100
Total steps	1e5			4e5	1e5		5e5	1e6	1e5	
Learning rate	5e-5				5e-7				5e-5	

© 2024 By AIRCC Publishing Corporation. This article is published under the Creative Commons Attribution (CC BY) license.