Rogue-Gym: A New Challenge for Generalization in Reinforcement Learning

Yuji Kanagawa
Graduate School of Arts and Sciences
The University of Tokyo
Tokyo, Japan
kanagawa-yuji968@g.ecc.u-tokyo.ac.jp

Tomoyuki Kaneko
Interfaculty Initiative in Information Studies
The University of Tokyo
Tokyo, Japan
kaneko@graco.c.u-tokyo.ac.jp

***Abstract*—In this paper, we propose Rogue-Gym, a simple and classic style roguelike game built for evaluating generalization in reinforcement learning (RL). Combined with the recent progress of deep neural networks, RL has successfully trained human-level agents without human knowledge in many games such as those for Atari 2600. However, it has been pointed out that agents trained with RL methods often overfit the training environment, and they work poorly in slightly different environments. To investigate this problem, some research environments with procedural content generation have been proposed. Following these studies, we propose the use of roguelikes as a benchmark for evaluating the generalization ability of RL agents. In our Rogue-Gym, agents need to explore dungeons that are structured differently each time they start a new game. Thanks to the very diverse structures of the dungeons, we believe that the generalization benchmark of Rogue-Gym is sufficiently fair. In our experiments, we evaluate a standard reinforcement learning method, PPO, with and without enhancements for generalization. The results show that some enhancements believed to be effective fail to mitigate the overfitting in Rogue-Gym, although others slightly improve the generalization ability.**

***Index Terms*—roguelike games, reinforcement learning, generalization, domain adaptation, neural networks**

I. Introduction

Reinforcement learning (RL) is a key method for training AI agents without human knowledge. Recent advances in deep reinforcement learning have created human-level agents in many games, such as those for Atari 2600 [1] and the game DOOM [2], using only pixels as inputs. This method could be applied to many domains, from robotics to the game industry.

However, it is still difficult to generalize learned policies between tasks even for current state of the art RL algorithms. Recent studies (e.g., by Zhang et al. [3] and by Cobbe et al. [4]) have shown that agents trained by RL methods often overfit the *training environment* and perform poorly in a *test environment*, when the test environment is not exactly the same as the training environment. This is an important problem because test environments are often differ somewhat from training environments in many applications of reinforcement learning. For example, in real world applications including self-driving cars [5], agents are often trained via simulators or designated areas but need to perform safely in real world situations that are similar to but different from their training environments. For agents to act appropriately in unknown situations, they need to properly generalize their policies that they learned from the training environment. Generalization is also important in transfer learning, where the goal is to transfer a policy learned in a training environment to another similar environment, called the *target environment*. We can use this method to reduce the training time in many applications of RL. For example, we can imagine a situation in which we train an enemy in an action game through experience across fewer stages and then transfer the enemy to a higher number of other scenes via generalization.

In this paper, we propose the Rogue-Gym environment, a simple roguelike game built to evaluate the generalization ability of RL agents. As in the original implementation of Rogue, it has very diverse and randomly generated dungeon structures on each floor. Thus, there is no pattern of actions that is always effective, which makes the generalization benchmark in Rogue-Gym sufficiently fair. Instead, in Rogue-Gym agents have to generalize abstract subgoals like getting coins or going downstairs through their action sequences that consist of concrete actions (e.g., moving left). Rogue-Gym is designed so that the environment an agent encounters, which includes dungeon maps, items, and enemies, is configurable through a random seed. Thus, we can easily evaluate the generalization score in Rogue-Gym by using random seeds different from those used in training. Since many other properties including the size of dungeons and the presence of enemies, are completely configurable, researchers can easily adjust the difficulty of learning so that the properties are complex enough and difficult for simple agents to solve but can still be addressed by the state-of-the-art RL methods.

In our experiments, we evaluate a popular DRL algorithm with or without generalization methods in Rogue-Gym. We show that some of the methods work poorly in generalization, although they successfully improve the training scores through learning. In contrast, some of these methods, like L2 regularization, achieve better generalization scores than those of the baseline methods, but the results are not sufficiently effective. Therefore, Rogue-Gym is a novel and challenging domain for further studies.

A part of this work was supported by JSPS KAKENHI Grant Number 18K19832 and by JST, PRESTO.

978-1-7281-1884-0/19/$31.00 ©2019 IEEE

II. Background

We follow a standard notation and denote a Markov decision process $\mathcal{M}$ by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R}$ is the immediate reward function that maps a state to a reward, and $\mathcal{P}$ is the state transition probability. We denote the policy of an agent by $\pi(a|s)$, that is, the probability of taking an action $a \in \mathcal{A}$ given a state $s \in \mathcal{S}$. In an instance of MDP, the goal of reinforcement learning [6](RL) is to get the optimal policy that maximizes the expected total reward by repeatedly taking an action, observing a state, and getting a reward. In this paper, we consider the episodic setting, in which the total reward is defined as $R = \sum_{t=0}^{T} \mathcal{R}(s_t)$, where t denotes the time step, and the initial state s_0 is sampled from the initial state distribution $\mathcal{P}^0$.

One of the famous classes of RL algorithms is policy gradients. Suppose that a policy π is parameterized by a parameter vector θ. Then, we can denote the policy by π_θ and the gradient of the expected sum of the reward by $\nabla_\theta \mathbb{E}[R]$. The goal of policy gradient methods is to maximize $\mathbb{E}[R]$ by iteratively updating θ on the basis of estimating of $\nabla_\theta \mathbb{E}[R]$ with agents' experience.

Deep reinforcement learning refers to RL methods that use deep neural networks as function approximators. DRL enables us to train RL agents given only screen pixels as states, through the use of deep convolutional neural networks (CNNs) [1]. PPO [7] is one of the state-of-the-art deep policy gradient methods, and we use it as a baseline method in this paper.

III. Related Work

Farebrother et al. [8] proposed the use of ALE [9], an environment based on an Atari2600 emulator, to evaluate generalization in RL. They conducted experiments by using different game modes of Atari 2600 games introduced by Machado et al. [10] and showed that regularization techniques like L2 regularization mitigates the overfitting of DQN [1]. However, the number of environments is limited in ALE, which allows us to tune algorithms for specific environments.

To increase the number of training/evaluation environments, procedural content generation is considered to be a promising method. Zhang et al. [3] conducted experiments by using simple 2D gridworld mazes generated procedurally and showed that some of the typical methods used for mitigating overfitting in RL often fail.

Cobbe et al. [4] proposed the CoinRun environment, which procedurally generates short 2D action games that have different backgrounds and stage structures. They showed that large neural network architectures and standard regularization methods such as batch normalization [11] help policy generalization in CoinRun. In addition, it is notable that both Zhang et al. and Cobbe et al. reported that increasing the number of training levels helps generalization.

Juliani et al. [12] proposed Obstacle Tower, where the player explores procedurally generated 3D dungeons from a third person perspective. Inspired by *Montezuma's Revenge*, one of the most difficult games that can be played in ALE, they designed Obstacle Tower to include factors like sparse rewards, which makes the task hard for RL algorithms. In experiments, they showed that the state-of-the-art algorithms including PPO struggle to generalize learned policies in Obstacle Tower.

Our work is most similar to Cobbe et al. [4] and Juliani et al. [12] in proposing a new environment with procedural content generation for evaluating generalization, though we place more importance on customizability and reasonable difficulty.

In addition to regularization, state representation learning [13] is also a promising approach for generalization. The key idea is the use of abstract state representations to bridge the gap between a training environment and a test environment. We can obtain such a representation via unsupervised learning methods such as variational autoencoders (VAE) [14].

Higgins et al. [15] adopted this idea for RL and proposed DARLA, which learns disentangled state representations by using β-VAE [16] from randomly collected observations and then learns a policy by using these representations. They manually set up training and test environments by changing the colors and/or locations of objects in 3D navigation tasks in DeepMind Lab [17]. They showed that DARLA improves generalization scores in these tasks. We evaluated β-VAE in our experiments.

IV. Rogue-Gym Environment

In this section, we introduce the Rogue-Gym environment, which is a simple roguelike game built for evaluating the generalization performance of RL agents.

To fairly evaluate the generalization ability of RL algorithms, we claim that structural diversity across training and test environments is important, in addition to a sufficient number of test environments. In the context of evaluating an RL agent in a single task, Machado et al. [10] claimed that the stochasticity of an environment is important by showing that a simple algorithm that memorizes only an effective sequence of actions performs well in a deterministic environment. This kind of hack is also possible in a generalization setting if the training and test environments do not have diverse structures and share an undesirable structural similarity. For example, in the *normal* task of CoinRun [4], stages share a common structure in that the player is initially on the left side of the stage, and the coin of each stage is placed on the right side. This means that we can perform well by always moving or jumping to the right in almost all stages. In fact, we observed that a random agent that selects only `right` and `right-jump` completed about the 62% of stages.

On the basis of this claim, we propose the use of roguelikes as a testbed for generalization. In this paper, we use the term *roguelike* as a subgenre of role-playing video games, where a player explores procedurally generated dungeons[1]. Our Rogue-Gym is a variant of roguelike and inherits the following properties desirable for evaluating generalization:

1) it is naturally integrated with procedural generation and provides us with a lot of test environments,

[1]Note that this definition is popular and consistent with the description on Wikipedia https://en.wikipedia.org/wiki/Roguelike

Fig. 1: Screenshot of Rogue-Gym

TABLE I: Meaning of characters

Character	Meaning
@	Player
.	Floor
#	Passage
|, −	Wall
*	Gold
%	Downstairs
+	Door
A-Z	Enemy (disabled in experiments in Sect. VI)

2) it has several behaviors agents need to generalize, such as finding doors or fighting enemies, and
3) it has very diverse dungeon structures, which prevents memorizing hacks.

In addition to these crucial properties, the following conditions are also important for enabling research on various learning methods with various computing resources available:

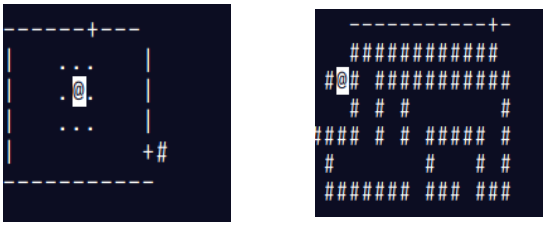
1) easy to customize and
2) able to change the difficulty with sufficient granularity.

We believe that customizability is especially important since the learning time required by deep RL algorithms heavily depends on the screen size.

To satisfy all of the properties, we create and present a simple roguelike named Rogue-Gym, with a clean implementation made from scratch by the first author, following the behavior of the original Rogue as accurately as possible. This is because other roguelike games popular for human players, such as NetHack and Cataclysm: DDA, are often too complex for RL agents. Also, the implementation of the original Rogue was written in old style C and is hard to modify.

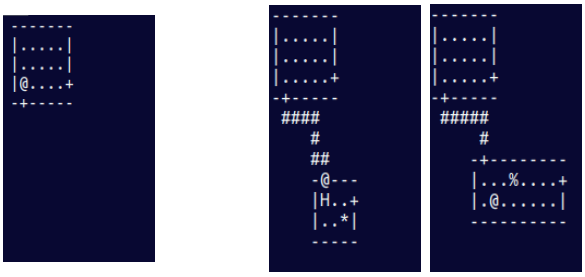
Fig. 1 shows a screenshot of Rogue-Gym. Like many roguelike games, it has a command line interface based on ASCII characters. Table I summarizes the meanings of characters.

In Rogue-Gym, the mission of the player is to get the Amulet of Yendor hidden on the deepest floor by finding the way to get downstairs on each floor. One floor consists of several rooms and passages but still has many combinatorial patterns, which makes it desirable for evaluating generalization. As shown in Fig. 2, in addition to normal rooms, Rogue-Gym has



(a) Dark room (b) Maze room

Fig. 2: Example of room variants



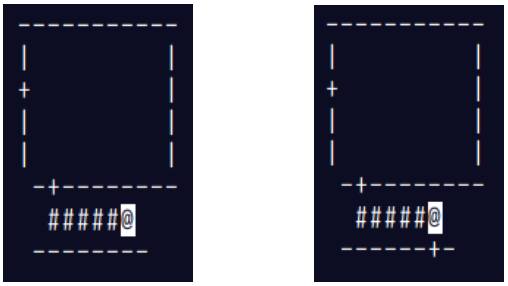
(a) Initial state (b) Possible future states

Fig. 3: The same observation may lead to different future states; while there is a door to the south of the player, the rest of the map is hidden before the door is opened (left). The agent may be faced with different states depending on the random seed after passing through the door (right).

There are dark rooms, in which only nine grids around the player are visible, and maze rooms consisting of passages that are arranged in a complicated manner. Rogue-Gym also has a variety of transition dynamics. Since only the areas that an agent has visited are visible by default, there can be situations where current states are the same but future states are different depending on the random seed, as shown in Fig. 3. In addition, Rogue-Gym is partially observable because of hidden passages and doors. As shown in Fig. 4, in Rogue-Gym, passages and doors are sometimes hidden and block the player's way. In this situation, if the player uses the `search` command, passages and doors can appear at a certain probability.

Rogue-Gym is available at the GitHub repository[2]. It is written in Rust in order to speed up performance but designed so that it can be called from many programming languages. The Python API is the main interface for training AI agents, of which binary packages are available at PyPI[3] and installable via `pip install rogue_gym`. As shown by the example code in Fig. 5, the API allows users to configure Rogue-Gym flexibly via Python's `dict` or JSON. We can change the size of the screen, kinds of enemies and items, and so on. The `RogueEnv` class inherits the `Env` class of OpenAI Gym [18], a standard library for defining RL environments. Thus, we believe our Rogue-Gym is compatible with much existing RL code and easy to use. An observation for an agent is encoded

[2]https://github.com/kngwyu/rogue-gym (Accessed:2019-05-29)
[3]https://pypi.org/project/rogue-gym/ (Accessed:2019-05-29)



(a) Before `search` (b) After `search`

Fig. 4: Example of hidden door: a door exists but is not observable (left). With (several uses of) the search command, the door was discovered (right).

```python
from rogue_gym.envs import RogueEnv
CONFIG = {
    'width': 32, 'height': 16,
    'dungeon': {
        'style': 'rogue',
        'room_num_x': 2, 'room_num_y': 2
     }
}
env = RogueEnv(max_steps=100, config_dict=CONFIG)
rewards = 0
state = env.reset()
for i in range(10):
    # move right
    state, reward, done, _ = env.step('l')
    rewards += reward
```

Fig. 5: Python API example of Rogue-Gym

as an image having the same number of binary channels as the number of ASCII characters, following a standard procedure (e.g., Silver et al. [19]). This encoding is suitable for the input of a convolutional neural network (CNN).

V. Generalization Problem in Rogue-Gym

In this section, we describe a concrete problem in Rogue-Gym that can be used to evaluate the generalization ability of RL algorithms. Here, our goal of generalization is to obtain abstract strategies, such as exploring new passages and rooms, from a limited number of samples.

To make the notations clear, we consider generalization in RL as a form of domain adaptation [20] problems where the target domain is unknown. Following a study by Higgins et al. [15], we define domain adaptation in RL as a problem in which we train agents in the source MDP $\mathcal{M}_S = (\mathcal{S}_S, \mathcal{A}_S, \mathcal{R}_S, \mathcal{P}_S)$ with $\mathcal{P}_S^0$, and then evaluate them in the target MDP $\mathcal{M}_T = (\mathcal{S}_T, \mathcal{A}_T, \mathcal{R}_T, \mathcal{P}_T)$ with $\mathcal{P}_T^0$. Though the set of states $\mathcal{S}_S$ and $\mathcal{S}_T$ can be different, we assume that the action spaces $\mathcal{A}_S$ and $\mathcal{A}_T$ are the same and that reward functions $\mathcal{R}_S$, $\mathcal{R}_T$ and transition $\mathcal{P}_S$, $\mathcal{P}_T$ share some underlying structure. This problem setting looks different from but is essentially the same as those by Zhang et al. [3] or by Cobbe et al. [4].

In the experiments in this paper, we define the source environment $(\mathcal{M}_S, \mathcal{P}_S^0)$ as s set of random number seeds, and also define the target environment $(\mathcal{M}_T, \mathcal{P}_T^0)$ as a different set of random number seeds that does not overlap with the source ones. Since the number of target environments is sufficiently large, we treat the score for the target environment as the generalization score.

TABLE II: All actions used in experiments

Command	Meaning
`.`	No operation
`h`	Move left
`j`	Move up
`k`	Move down
`l`	Move right
`n`	Move right down
`b`	Move left up
`u`	Move right up
`y`	Move left down
`>`	Go downstairs
`s`	Search around player

Configurations other than random seeds for dungeons are the same in the source and target domains. In all experiments, we configured the screen size to be 32×16 without enemies and items. Each episode ended at 500 time steps. This configuration makes the problem not too difficult and sufficiently easy to learn. The action space was discrete with 11 dimensions, as listed in Table. II.

Rewards consist of gold that an agent gathers throughout an episode. In addition, we give 50 golds as pseudo rewards each time an agent reaches the next floor to adjust the difficulty to be suitable for standard RL algorithms. Hence, reward functions $\mathcal{R}_S$ and $\mathcal{R}_T$ are different due to the difference in the state space, but they still share an underlying structure in that an agent can get a reward when:

- it arrives at a grid with `*`, and
- it arrives at a grid with `%` and selects `downstairs` as action.

Transition probabilities $\mathcal{P}_{S,T}$ are also different, but they share an underlying game rule that indicates how the player can move in dungeons, which includes stochasticity like that shown in Fig. 3.

As metric of generalization ability, we simply use the average value of rewards in all target environments, which can be denoted by $\mathbb{E}_\pi \left[\frac{1}{|\mathcal{M}_T|} \sum_{m \in \mathcal{M}_T} \sum_{t=0} \mathcal{R}_m(s_t) \right]$. In the following sections, we call this metric the *generalization score*.

VI. Evaluating Generalization Methods

In this section, we use Rogue-Gym to evaluate several methods used for deep reinforcement learning and discuss their performance by using the generalization score defined in the previous section. Our code for the experiments is available at the GitHub repository[4].

A. Reinforcement Learning Methods

We used PPO [7] as our baseline because it was the best among popular RL algorithms including DQN [1] and

[4]https://github.com/kngwyu/rogue-gym-agents-cog19 (Accessed:2019-05-29)

TABLE III: PPO parameters

Number of workers N	32
Rollout length	125
Value coef.	0.5
Entropy coef.	0.01
γ	0.99
GAE λ	0.95
Num. epochs	10
Clipping parameter ϵ	0.1
Minibatch size	200
Learning rate of Adam	2.5e-4
ϵ of Adam	1.0e-4

A2C [2], [21] in our preliminary experiments. We compare the following six enhancements to encourage generalization on top of PPO.

1) PPO SMALL: PPO with small CNN
2) PPO LARGE: PPO with large CNN
3) PPO BATCHNORM: PPO with large CNN and batch normalization
4) PPO L2: PPO with large CNN and L2 regularization
5) VAE PPO: PPO with VAE
6) β-VAE PPO: PPO with β-VAE

PPO Small uses a network with three CNN layers, which is similar to the one used in DQN [1]. PPO Large uses a network that is almost the same as the one used in IMPALA [22], which has 15 CNN layers and 6 residual connections. We adopted this large architecture because it was effective in a study by Cobbe et al. [4]. The hyper-parameters of PPO were mostly taken from the study [7] and listed in Table. III.

As standard regularization methods, we adopted batch normalization (PPO BATCHNORM) and L2 regularization (PPO L2). Batch normalization performed best among the three regularization methods used by Cobbe et al. [4], and L2 regularization is reported to improve regularization in RL both by Cobbe et al. [4] and Farebrother et al. [8]. We used 10^{-4} as a weight decay parameter λ of PPO L2.

VAE PPO and β-VAE PPO were adopted to evaluate the effectiveness of disentangled state representation learning, which was shown to be useful in a study by Higgins et al. [15]. We adopted β-VAE PPO in a simpler manner than their DARLA. For simplicity in our implementation, reconstruction loss was calculated without denoising autoencoders, while two versions, with and without denoising autoencoders, were used for DARLA. In addition, our β-VAE PPO trains β-VAE simultaneously with a policy by using parameter sharing, while DARLA trains β-VAE with random actions before learning a policy. This is because it is difficult in Rogue-Gym to obtain sufficiently diverse observations with only random actions. We show a computation graph of β-VAE PPO in Fig. 6.

VAE PPO is a special case of β-VAE PPO [16], where $\beta = 0$. It is used for comparison with $\beta > 0$ cases, which learns disentangled representations. We used $\beta = 4.0$ for β-VAE PPO. Also, we added the score of the random agent to the bottom line.

As a training score, we used episodic rewards averaged over total 1 000 trials (100 episodes for each of 10 seeds $[0, 9]$))

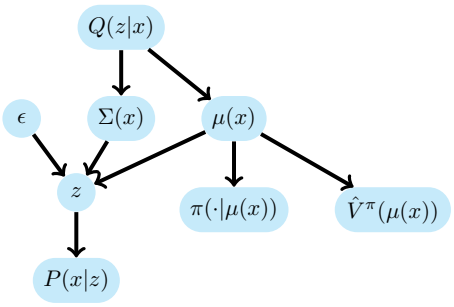


Fig. 6: Computation graph of β-VAE PPO, where x is the input, z is a latent variable, π is a policy, $\hat{V}$ is baseline used for the policy gradient, and μ and σ are the mean and variance of a Gaussian distribution.

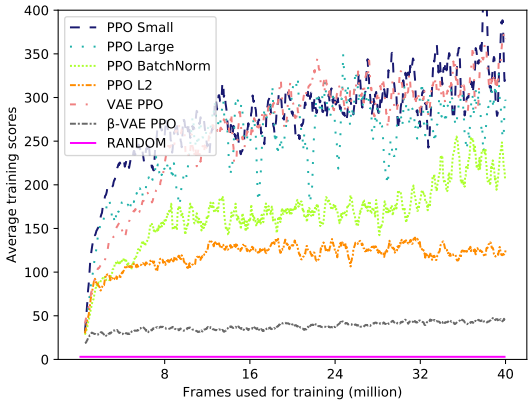
for the random agent and used rewards averaged over 1000 consecutive episodes in the training time for other agents. As generalization scores, we used rewards averaged over total 10 000 episodes (10 episodes for each of 1 000 seeds) for all agents.

B. Difference in Training and Generalization Scores

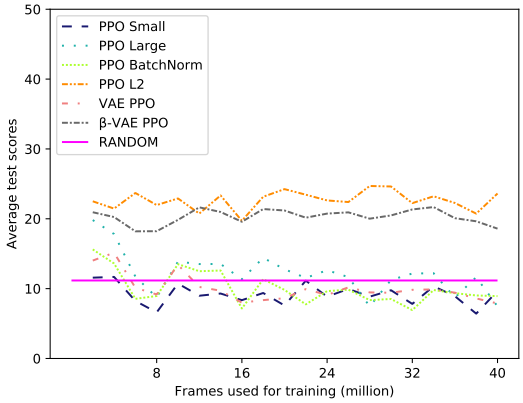
We compared the performances of the methods described in the previous subsection. Fig. 7 shows the training and generalization scores of these methods with respect to the number of game frames experienced during training time. We used $[0, 9]$ as the set of random seeds of the source environment, and $[1000, 2000)$ of the target. With respect to the training scores shown in Fig. 7a, all learning methods improved the score much better than the score of random agents. We can see that PPO SMALL or LARGE, and VAE PPO performed the best in the source environment, and β-VAE PPO performed significantly worse.

In contrast, Fig. 7b shows that generalization scores were much lower than the training scores for all methods and hardly improved during the training time. The most notable result is that of PPO LARGE. It performed slightly better than PPO SMALL, which supports the claim by Cobbe et al. [4]. However, the effectiveness of increasing the size of neural networks was relatively lower than the results reported in the work by Cobbe et al., in which a larger architecture performed about 20% better than a smaller one. From this result, we can claim that it is less effective in Rogue-Gym to improve the memorization ability of algorithms, and, thus this benchmark is fair. Also, it is surprising that PPO BATCHNORM performed worse than PPO SMALL and PPO LARGE because it performed better in Cobbe et al. in both the training and test phases.

Only PPO L2 and β-VAE PPO were clearly better than a random agent, although their generalization scores were significantly worse than for the training. PPO L2 performed the best in the target environment among all methods we used, which was consistent with the results obtained by Farebrother et al. [8] and Cobbe et al. [4]. β-VAE PPO was second-best among all of these methods in the target environment,



(a) Training scores



(b) Generalization scores

Fig. 7: Results with 10 training environments: top figure shows training scores and bottom figure shows generalization scores.

while VAE PPO overfitted as did the other methods. This supports the idea that that disentangled representation learning is effective for generalization [15].

It is notable that all methods other than PPO L2 and β-VAE PPO significantly overfitted. All methods performed better than the random agent at first but gradually overfitted and performed worse than the random agent at the end of the training. This observation is notable since such large overfitting was not observed in previous studies like those by Cobbe et al. [4] and by Farebrother et al. [8].

C. Effectiveness of Diversity in Training Environment

We investigated how an increase in the number of training seeds (i.e., dungeons) would improve the generalization scores. In existing works by Zhang et al. [3] and Cobbe et al. [4], it is reported that increasing the number of training levels significantly improves the performance of generalization. Accordingly, we conducted experiments with the number of training seeds set to 20 ($[0, 19]$) or 40 ($[0, 39]$), which is an increase from 10 in the previous experiments.

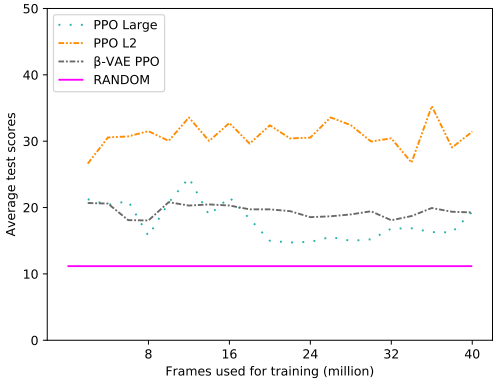


Fig. 8: Generalization scores with 20 training seeds

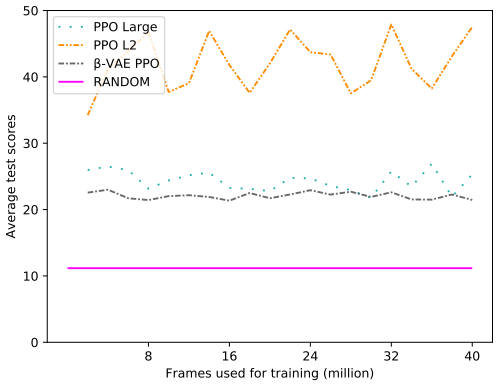


Fig. 9: Generalization scores with 40 training seeds

In this experiment, we used β-VAE PPO and PPO L2 since these two methods performed better than the others in the previous experiment. In addition, we adopted PPO LARGE as the baseline since it has almost the same number of neural network parameters as these two methods.

Fig. 8 and Fig. 9 show the results with 20 and 40 training seeds, respectively. We can see that increasing the number of training seeds improved the generalization performance for all methods. However, this improvement was not significant for β-VAE PPO. In addition, it is notable that the generalization score of PPO LARGE seemed constant through the training time with 40 training seeds. This is an improvement since the gradual overfitting (the decrease in generalization score) observed in the case with 10 or 20 training seeds did not appear here.

D. Effectiveness of Exploration

Having a policy with higher entropy means that an agent tends to act more randomly and to explore better. To investigate the effectiveness of exploration in Rogue-Gym, we ran agents 10 times for each training seed and measured the entropy of their policies averaged over $(10 \times \#\text{seeds})$ episodes.

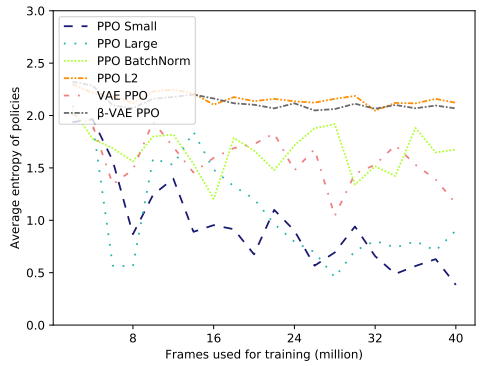


Fig. 10: Entropy of policies with 10 training seeds

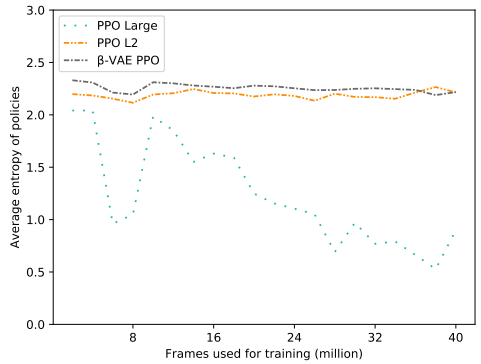


Fig. 11: Entropy of policies with 20 training seeds

Fig. 10 shows the average entropy of policies with 10 training seeds. We can see that β-VAE PPO and PPO L2 had policies with higher entropy, and the entropy of the other methods decreased gradually, corresponding to the evaluation scores in Fig. 7b.

Fig. 11 and Fig. 12 show the entropy of policies with 20 and 40 training seeds, respectively. We can see that PPO-L2, which had the best generalization score, also had the highest or similar entropy during training. Therefore, we can suppose that having diverse policies is important for obtaining a decent generalization score. However, the entropy of PPO-L2 policies with 20 or 40 training seeds was similar, although their generalization scores were different. Also, the entropy of β-VAE PPO was similar to that of PPO-L2, while the generalization scores significantly differed between the two methods. Hence, we can conclude that the randomness of the policies was important for generalization, but there is certainly another factor that enables agents to generalize their policies.

E. Detailed Analysis of Learned Policy

We compared the behaviors of two agents in a specific dungeon to investigate what kind of generalized behaviors they obtained. Concretely, we used a PPO LARGE agent with 10 training levels as the representative in overfitted agents and a

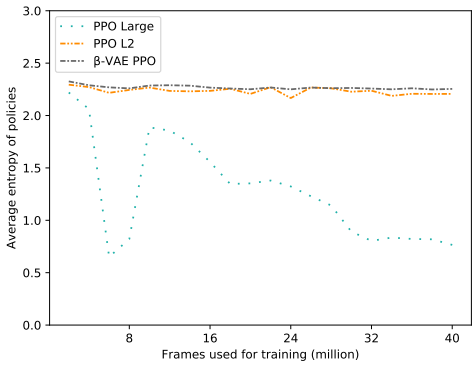


Fig. 12: Entropy of policies with 40 training seeds

TABLE IV: Generalization scores for seed 1008

Method	Number of training levels	Score
PPO LARGE	10	0.6
PPO L2	40	503.9

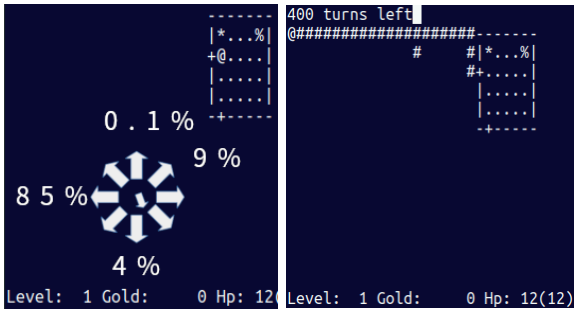
PPO L2 agent with 40 training levels as a generalized agent. To highlight the difference of their behavior, we used the dungeon in random seed 1008, which is a relatively easier stage. The scores of these agents were averaged over 10 trials and are summarized in Table IV.

We show the policy for the initial state of the overfitted agent in Fig. 13a. Each arrow means an action of moving or going down stairs. The probabilities are shown for the actions of the three highest probabilities as well as that of moving up. We expect two actions for generalized agents; moving up to get gold, or moving right to approach the downstairs. However, in the policy of the overfitted agent, moving left was the best action with a high probability of 85%, and the probability of getting golds was only 0.1%. Moving left was not appropriate because doing so made it more difficult for the agent to go down to the next floor. In fact, the agent was not able to go the next level and got stuck in a passage after some turns passed, as shown in Fig. 13b.

We show the policy for the initial state of the generalized agent in Fig. 14a. The probability of getting gold was only 1%. It was much higher than that of the overfitted agent but still low. Also, we can see that this agent tended to move down. We can not say it was an effective action, but it was better than moving to the left. Though we did not see this agent move straight to the right, the agent often succeeded at going downstairs and reaching deeper levels (5), as shown in Fig. 14b.

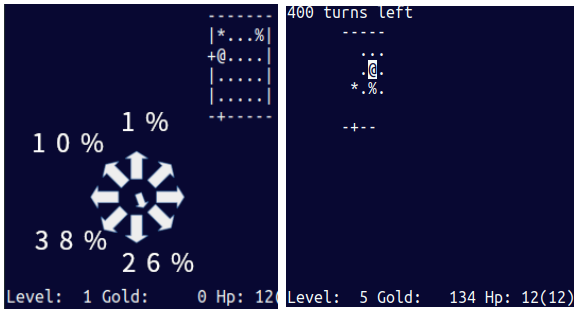
VII. CONCLUSION

In this paper, we proposed the Rogue-Gym environment, which is suitable for researching generalization in reinforcement learning. Based on roguelike games, it is built with diverse types of dungeons, fully configurable enemies and items, and has a programming interface compatible with OpenAI Gym. Rogue-Gym provides a sufficient number of



(a) Policy for initial state (b) After 100 steps

Fig. 13: Probability of actions of overfitted agent in initial state (left) and state after 100 steps (right)



(a) Policy for initial state (b) After 100 steps

Fig. 14: Probability of actions of generalized agent in initial state (left) and state after 100 steps (right)

dungeons by procedural content generation. Since the dungeons in Rogue-Gym share abstract rules but do not share a specific structure, it is difficult to get a higher score by memorizing only good sequences of actions in Rogue-Gym. Therefore, we believe that the generalization benchmark in Rogue-Gym is fair.

We evaluated existing methods for generalization with PPO in Rogue-Gym. The difficulty was modestly suppressed by using a small screen size and by disabling enemies and items so that good policies in the game would be surely learnable with state-of-the-art methods. Our results showed that training scores successfully improved during training, but that generalization scores measured in unseen dungeons hardly improved and were much lower than the training scores. Among the generalization methods, L2 regularization was the most effective, though there is much room for improvement. Also, we observed that their entropy of their policies of the agents was higher for agents with better generalization scores than others. This suggests that agents need to keep their entropy sufficiently high to improve their generalization skills. The use of maximum entropy reinforcement learning methods like soft actor-critic [23] would be interesting future work. Since memory is sometimes useful for handling partial observability when playing Rogue-Gym, the use of recurrent models (e.g., LSTM) should also be investigated.

REFERENCES

[1] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*, 2016, pp. 1928–1937.

[3] C. Zhang, O. Vinyals, R. Munos, and S. Bengio, "A study on overfitting in deep reinforcement learning," *CoRR*, vol. abs/1804.06893, 2018.

[4] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," *CoRR*, vol. abs/1812.02341, 2018.

[5] A. Kendall *et al.*, "Learning to drive in a day," *CoRR*, vol. abs/1807.00412, 2018.

[6] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[8] J. Farebrother, M. C. Machado, and M. Bowling, "Generalization and regularization in DQN," *CoRR*, vol. abs/1810.00123, 2018.

[9] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *CoRR*, vol. abs/1207.4708, 2012.

[10] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *J. Artif. Intell. Res.*, vol. 61, pp. 523–562, 2018.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, 2015, pp. 448–456.

[12] A. Juliani, A. Khalifa, V. Berges, J. Harper, H. Henry, A. Crespi, J. Togelius, and D. Lange, "Obstacle tower: A generalization challenge in vision, control, and planning," *CoRR*, vol. abs/1902.01378, 2019. [Online]. Available: http://arxiv.org/abs/1902.01378

[13] T. Lesort, N. D. Rodríguez, J. Goudou, and D. Filliat, "State representation learning for control: An overview," *Neural Networks*, vol. 108, pp. 379–392, 2018.

[14] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," *CoRR*, vol. abs/1312.6114, 2013.

[15] I. Higgins *et al.*, "DARLA: improving zero-shot transfer in reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, 2017, pp. 1480–1490.

[16] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "β-vae: Learning basic visual concepts with a constrained variational framework," in *ICLR*, 2017.

[17] C. Beattie *et al.*, "Deepmind lab," *CoRR*, vol. abs/1612.03801, 2016.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[19] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[20] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*.

[21] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 5279–5288.

[22] L. Espeholt *et al.*, "IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, 2018, pp. 1406–1415.

[23] T. Haarnoja, A. Zhou, P. Abbeel, and S. 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, ICML 2018*, 2018, pp. 1856–1865.