LEARNING ROBUST GOAL SPACE WITH HYPOTHETICAL ANALOGY-MAKING

Anonymous authors
Paper under double-blind review

ABSTRACT

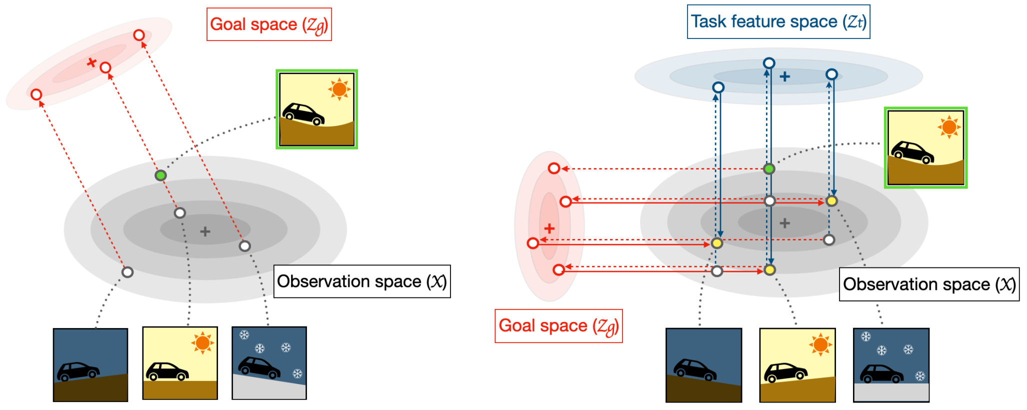
Learning compact state representations from high dimensional and noisy observations is the cornerstone of reinforcement learning (RL). However, these representations are often biased toward the current goal context and overfitted to goal-irrelevant features, making it hard to generalize to other tasks. Inspired by the human analogy-making process, we propose a novel representation learning framework called hypothetical analogy-making (HAM) for learning robust goal space and generalizable policy for RL. It consists of encoding goal-relevant and other task-related features, hypothetical observation generation with different feature combination, and analogy-making between the original and hypothetical observations using discriminators. Our model introduces an analogy-making objective that maximizes the mutual information between the generated hypothetical observation and the original observation to enhance disentangled representation. Experiments on various challenging RL environments showed that our model helps the RL agent's learned policy generalize by revealing a robust goal space.

1 INTRODUCTION

Learning policies directly from observations is a gateway to successful real-life applications such as auto-driving and robotics. However, current deep reinforcement learning (RL) algorithms trained from raw pixels have several issues undermining their applicability to real-world problems. First, they are vulnerable to common noises such as background changes (Gamrian & Goldberg, 2019; Zhang et al., 2021). They also often fail to adapt to small semantic changes, e.g., the height of the platform or the position of stars in the Climber environment in the OpenAI ProcGen benchmark suite (Cobbe et al., 2020; Kirk et al., 2021). Both are because their representations are strongly biased toward the current goal context, distracted by task-relevant easy-to-learn features (Scimeca et al., 2022), and eventually fail to learn the robust goal space (see Figure 1(a)). In fact, there exists a substantial discrepancy between the train and test performance of the vanilla PPO agent (Cobbe et al., 2020). Although a large amount of training data ($\geq$ 10k samples) could reduce the generalization gap (Cobbe et al., 2019), many practical applications cannot meet this condition for various reasons, such as cost and safety issues for data acquisition (Levine et al., 2020).

Unlike machine learning algorithms, humans have an outstanding ability to learn with a limited amount of experience (Lake et al., 2016; Kaiser et al., 2020). This ability is often ascribed to abstraction and analogy-making. Analogy-making is a central mechanism for revealing meaning and core concept from perception (Anderson, 1980; Bartha, 2019). We learn compact abstractions from noisy perception through numerous comparisons with other perceptions or imaginations by making analogies (Lakoff, 2009; Mitchell, 2021). In this paper, we hypothesize that applying the analogy-making process to learning the goal space can help RL models learn generalizable policy. Figure 1(b) shows the example case when analogy-making can find proper goal space to improve generalization.

Inspired by human analogy-making, we propose a novel representation learning framework for learning robust goal space in RL. To establish broadly generalizable policies in RL, we aim to separate the *goal context* from *task feature* by performing analogy-making between experienced and hypothetical observations. Our framework called hypothetical analogy-making (HAM) consists of three phases: encoding goal-relevant and other task-related features, hybrid observation generation, and analogy-making with discriminators. (i) HAM decomposes an observation into two independent


Goal space ($\mathcal{Z}_g$)
Observation space ($\mathcal{X}$)
Task feature space ($\mathcal{Z}_t$)
Goal space ($\mathcal{Z}_g$)
Observation space ($\mathcal{X}$)

(a) Task feature-correlated goal space   (b) Robust goal space with analogy-making

Figure 1: The analogy-making process can help learn the proper goal space: when the training dataset is small, as in (a), the learned goal space is easily biased toward task feature space. In (a), there are three training data points (white-colored), and we try to encode visual features that determines the control of the vehicle associated with the current goal of safe driving. We can find an undesirable correlation between the learned goal space and the task feature space (i.e., the weather condition) resulting in the wrong goal context encoding of the unseen observation (green-colored). By adopting the analogy-making process, we can relieve this problem. By generating hypothetical observations (yellow-colored) using learned goal and task feature space and making analogies with the original observations (white-colored) such as ”Do A and B share the same control of the vehicle?” or ”Are A and B in the same weather condition?”, we can find the proper goal space and task feature space and we can map the unseen observation correctly as in (b).

components: a goal-related code and other task-related code. (ii) By combining the codes from different tasks, HAM generates a hybrid observation, a hypothetical but realistic observation. In doing so, (iii) HAM maximizes the mutual information between the hypothetical observation and the original labels, enforcing the goal-related and other task-related information being embedded in each code. Our model learns robust goal context across varying task features by making analogies between the original and hypothetical observations generated with different task feature codes. The key contributions of our work are as follows:

- Motivated by human analogy-making, we propose a novel representation learning framework for learning robust goal context in RL. For this, we combine goal and task feature relevant encoding, hypothetical observation generation, and analogy-making between the original and hypothetical observations.
- We conducted several experiments in the Jumping Task (des Combes et al., 2018) as a proof of concept, where the observation space has relatively simple generative factors. Furthermore, we empirically show that our model outperforms prior baselines on several environments in the OpenAI ProcGen benchmark suite (Cobbe et al., 2020), which was designed to evaluate the generalization performance of RL algorithms.
- We show that our model successfully reveals a robust goal space, which is invariant to other task feature changes, using both quantitative metrics and qualitative visual results. The learned RL policy built on this robust goal space can generalize more broadly to the unseen observations.

2 Problem Formulation

2.1 Preliminaries

We use a Markov decision process (MDP) setting, in which the MDP is denoted as $\mathcal{M} = (\mathcal{X}, \mathcal{A}, R, P, \gamma)$ with an observation space $\mathcal{X}$, an action space $\mathcal{A}$, a reward function R, transition dynamics P, and a discount factor $\gamma \in [0, 1)$. A policy $\pi(\cdot|x)$ represents a probability distribution over actions given observation $x \in \mathcal{X}$. The RL agent aims to learn an optimal policy that maximizes the expected cumulative discounted rewards $\mathbb{E}_{a_t \sim \pi(\cdot|x_t)}[\sum_t \gamma^t R(x_t, a_t)]$ at timestep t starting from an initial state x_0. We define a *goal of a task* as an inherent rule of a task related to reward acquisition.

To formalize this, we consider a distribution of tasks which share the same goal, each defined as an MDP $\mathcal{M}^i \in \mathcal{M}$, where $i \in I$ and $|I|$ defines the size of the task distribution. Those MDPs share an action space $\mathcal{A}$, a reward function R, and transition dynamics P but are with disjoint observation spaces, $\mathcal{X}^i \cap \mathcal{X}^j = \emptyset$. Note that the domain of transition dynamics and reward function is a state space, not an observation space $\mathcal{X}$. A state refers to an embedding of an observation encoded only with the goal-related part of the observation. For instance, different MDPs correspond to tasks with different levels in the same environment in the OpenAI ProcGen benchmark suite (Cobbe et al., 2020) (see Appendix A.1.2). We define an union observation space $\mathcal{U}$ of all possible observation spaces, where $\mathcal{U} = \bigcup_{i \in I} \mathcal{X}^i$. We assume the RL agent has access to a collection of N training MDPs $\{\mathcal{M}^i\}_{i=1}^N$ and the index i of each. After training, the RL agent applies its policy π over the entire observation space $\mathcal{U}$ including unseen MDPs. Note that we evaluate the learned policy's zero-shot performance without any meta-learning phase.

2.2 Broadly generalizable policy

Broadly generalizable policy refers to a policy that is consistent across different environments with the same goal regardless of their visual discrepancy. It can be learned based on the ideal goal space z_g^*. The visual information necessary for reconstructing an observation x can be divided into the goal-dependent $\mathcal{I}(g, x)$ and other goal-independent information term $H(x|g)$, where g indicates the goal context of x, and $\mathcal{I}$ and H refer to mutual information and entropy, respectively. The goal-independent information reflect the intrinsic nature of the task (i.e., task structures associated with the task index i). For any $x \in \mathcal{X}^i$, the latent feature group z_g^* and z_t^* are said to be *key generative factors of the state space* $\mathcal{X}^i$ when

$$H(x) = \mathcal{I}(g, x) + H(x|g) = \mathcal{I}(g, x) + \mathcal{I}(t, x) = H(z_g^*) + H(z_t^*), \qquad (1)$$

where g and t indicate goal context labels (e.g., optimal action) and task-dependent features (e.g., task index), respectively. By combining these two mutually exclusive variables, the goal feature z_g^* ideally reflecting the goal context information and the task feature z_t^* ideally reflecting all other task-dependent information, one can reconstruct the current observation x.

The goal of learning representations for *broadly generalizable policy* is to glean goal context information from the noisy observation. This is formulated as maximizing $\mathcal{I}(g, z)$ while minimizing $\mathcal{I}(t, z)$ with respect to z, so that a policy built upon the representation z becomes robust against goal-irrelevant changes (i.e., changes in the task background space in Figure 1(b)). The optimal solution is z_g^* in our setting. By using our hypothetical analogy-making module, which makes imaginary hypothetical observations with different combinations of the generative factors of inputs and performs analogy-making using discriminators, RL agent can achieve the *broadly generalizable policy*. Our module enhances the independence of two generative factors by swapping the two features when generating hypothetical observations. We provide the details in the next section.

3 Hypothetical Analogy-Making

We implemented the hypothetical analogy-making (HAM) process with mutual information (MI) regularized generative adversarial networks (GAN) (Goodfellow et al., 2014) structure. In addition to the original GAN objective aimed at making images as realistic as possible, we train our encoder E, generator G, and discriminator D to maintain a large amount of MI between the hypothetical images $G([z_g^i, z_t^j])$ and the original labels: *goal context* label g and *task feature* label t.

The MI regularized GAN objective is as follows. For any $x^i \in \mathcal{X}^i$ and $x^j \in \mathcal{X}^j$,

$$\min_{E,G} \max_{D} L_{\mathcal{I}}(D, G, E) = L(D, G, E) - \lambda_1 \mathcal{I}(g^i, G([z_g^i, z_t^j])) - \lambda_2 \mathcal{I}(t^j, G([z_g^i, z_t^j])), \qquad (2)$$

where $[z_g^i, z_t^i] = E(x^i)$, $[z_g^j, z_t^j] = E(x^j)$, and $L(D, G, E)$ is an encoder-added version of the original GAN loss (Chen et al., 2016), which is $\mathbb{E}_{x \sim P_{data}}[logD(x)] + \mathbb{E}_{x \sim P_{data}}[log(1 - D(G(E(x))))]$.

The encoder part of the objective corresponds to the first step of HAM: decomposing the original observation into the goal context-relevant part and the others. It is followed by the generation of hypothetical observation $G([z_g^i, z_t^j])$ as a second step: imagining a hypothetical situation by replacing the background part with what we have experienced before while maintaining the original context. The MI terms, $\mathcal{I}(g^i, G([z_g^i, z_t^j]))$ and $\mathcal{I}(t^j, G([z_g^i, z_t^j]))$, correspond to the last step: keeping the context and background information the same with the original observations where each code came from. We hypothesize that the RL agent mimicking the human analogy-making process with hypothetical observations can learn a *broadly generalizable policy*.

Suppose we generated a new image with goal context code z_g^i from an image x^i and the task feature code z_t^j from another image x^j. We claim that by maximizing the mutual information between the generated image and the goal context label from which its goal context code came, $\mathcal{I}(g^i, G([z_g^i, z_t^j]))$, and the task feature label of which its task feature code came from, $\mathcal{I}(t^j, G([z_g^i, z_t^j]))$ with regard to the encoder E and generator G, we can find the optimal solution z_g^* by maximizing the lower bound of $\mathcal{I}(g, z_g)$ while minimizing $\mathcal{I}(t, z_g)$.

3.1 Model Architecture

The proposed HAM architecture is illustrated in Figure 2. All the components are trained jointly during the RL policy learning process. The following sections describe each component.

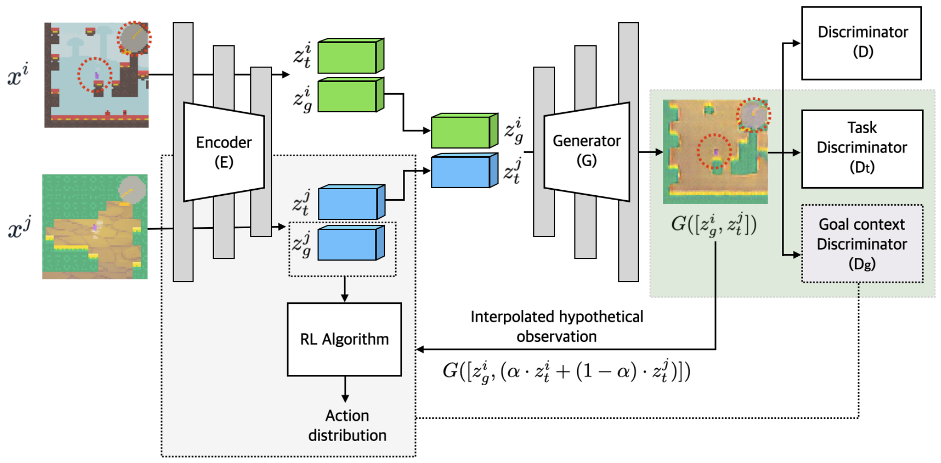


Figure 2: Architecture of Hypothetical Analogy Making model

Image Generation with Disentangled Features. Given an input pair $(x^i, x^j) \sim (\mathcal{X}^i, \mathcal{X}^j)$, we apply our encoder E to each input. Then the encoder outputs the goal context code and background code for each input, $E(x^i) = [z_g^i, z_t^i]$ and $E(x^j) = [z_g^j, z_t^j]$, respectively. Then we apply the generator G to the hybrid pair $[z_g^i, z_t^j]$. The generated image $G([z_g^i, z_t^j])$ is fed into the basic discriminator D, which evaluates how realistic the generated image is. Furthermore, we train our encoder, generator, and discriminator to generate a realistic image while learning independent goal and task-relevant features that can be combined to create a new hybrid image. The loss is calculated as follows using the non-saturating adversarial loss (Mirza & Osindero, 2014):

$$J_{GAN,hybrid}(E, G, D) = \mathbb{E}_{x^i \sim \mathcal{X}^i, x^j \sim \mathcal{X}^j, x^i \neq x^j}[-log(D(G([z_g^i, z_t^j])))]. \quad (3)$$

The detailed structures of our encoder, generator and discriminator are provided in Appendix A.1.1 and Appendix A.1.2.

Analogy-Making. The goal context and task discriminators in the green shaded area of Figure 2 makes predictions about the hypothetical observations and uses the labels of the original images for calculating the prediction loss. Note that this is the core component of our hypothetical analogy-making process. The generated images $G([z_g^i, z_t^j])$ are fed into the three modules: the basic discriminator D, task discriminator D_t, and goal context discriminator D_g. The task discriminator

D_t outputs how realistic the given generated image is compared to the reference image x^j, where the task feature code of the generated image was from x^j. The loss is given by:

$$J_{task}(E, G, D_t) = \mathbb{E}_{x^i \sim \mathcal{X}^i, x^j \sim \mathcal{X}^j}[-log(D_t(\texttt{crop}(G([z^i_g, z^j_t]))), \texttt{crops}(x^j))], \tag{4}$$

where $[z^j_t, z^j_c] = E(x^j)$ and `crop` randomly selects a fixed-sized patch of the full image and `crops` is a collection of multiple patches. We use the `crop` function to wipe the goal context out and preserve the task features from images in ProcGen benchmark. Note that in experiments with Jumping task, we use the full images as inputs for D_t without using the `crop` function because the task features of Jumping task (i.e., the height of the floor and the position of the obstacle) are lost in the cropping process.

The goal context discriminator D_g outputs whether the hybrid image shares the same action context with the original image x^i (the generated image got its goal context code from x^i) or not. In experiments with ProcGen benchmark, we utilize the action output of the RL part's policy network Q_π instead of using a separate action discriminator (gray shaded area in Figure 2). We took the action output of the policy network and trained the model in the direction of minimizing Jensen-Shannon divergence D_{JS} between the action probabilities of the original and hypothetical observations. The loss is given by:

$$J_{goal}(E, G, Q_\pi) = \mathbb{E}_{x^i \sim \mathcal{X}^i, x^j \sim \mathcal{X}^j}[D_{JS}(Q_\pi(G([z^i_g, z^j_t])) \| Q_\pi(x^i)))]. \tag{5}$$

By using the above losses, our encoder E and generator G learn to generate a hybrid image $G([z^i_g, z^j_t])$, which contains the original goal context of x^i and the task background of x^j. The goal context of x^i is highlighted with red dotted circles in Figure 2. Note that D_g and D_t compete over visual information in the observation. However, as the information in the observation can be divided into the goal context-relevant parts and the others and ideally the two groups do not overlap, both D_g and D_t can minimize their own loss without compromise.

Policy Learning. The RL agent learns its policy with the goal context code z_g. In experiments with the ProcGen benchmark, the RL agent also utilizes hybrid images to facilitate learning task background-invariant context features. In ProcGen, we use a standard Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithm for training the RL agent; note that our framework can exploit any model-free RL algorithm. PPO algorithm utilizes action-advantages $A_t = A^\pi(a_t, x_t) = Q^\pi(a_t, x_t) - V^\pi(x_t)$, and minimizes a clipped probability loss as follows:

$$\mathcal{J}_\pi(\theta) = -\mathbb{E}_{\tau \sim \pi}[\min(\rho_t(\theta)A_t, clip(\rho_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]. \tag{6}$$

We use hypothetical observations when calculating the clipped-ratio $\rho_t(\theta)$ over the recent experience collected with $\pi_{\theta_{old}}$ and updating a state-value estimator $V_\phi(x)$. We replace a certain ratio $\lambda_{mix} \in (0, 1]$ of the original observation x with hypothetical observation generated by performing task feature interpolation $G([z^i_g, (\alpha \cdot z^i_t + (1-\alpha) \cdot z^j_t)])$ using a random rate $\alpha \in [0, 1)$. See Appendix A.3.2 for the quality of task interpolated hypothetical observations.

4 Experimental Results

We ran simulations to show that our model improves generalization performance in challenging scenarios by separating goal context-relevant visual features from task background-relevant ones. As a proof of concept, we conducted several experiments, including an ablation study in the Jumping task (des Combes et al., 2018), where the observation space has relatively simple generative factors. Then we conducted evaluations in more challenging tasks in ProcGen benchmark suite (Cobbe et al., 2020), where goal-related visual features are intertwined with other task-relevant ones. We carried out evaluations in three different ways: (i) generalization performance with a small size of the training dataset, (ii) qualitative results of generated hypothetical observations showing the separation of GAN features into the goal and task-related group, and (iii) further evaluation to show the robustness of our learned goal context space. The details for each experiment are provided in Appendix A.1 and the code for replicating our experiments is available in the supplementary material.

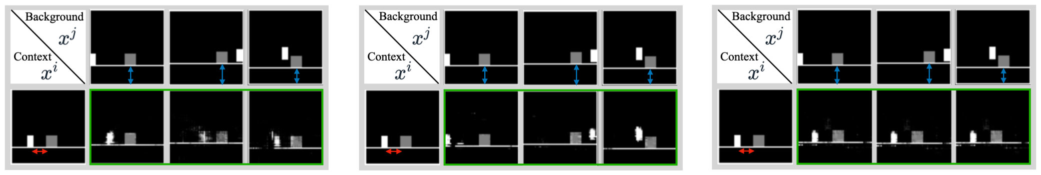
4.1 Jumping task

In the Jumping task, a white agent has to jump over a gray obstacle without touching it. The evaluation measures test performance for 286 tasks of 26 different obstacle locations and 11 floor heights while the training proceeds with only 18 of them. Depending on the distribution of training samples in the evaluation grid, the training type is classified into Wide, Narrow, and Random. Each grid measures different types of generalization (See Appendix A.1.1 for the details). Experiments in the Jumping task are intended to confirm whether our model can separate relatively simple goal context-relevant visual features from task background-relevant ones. In the Jumping task, the *distance between the agent and the obstacle* is the goal context-relevant visual feature. The *height of the floor and the location of the obstacle* are the task background-relevant visual features.

Generalization test. The following are the generalization results compared with other baseline methods and ablated models. In Table 1, we present the ratio of how many of the total 286 tasks the learned policy succeeded. The policy is trained using only 18 tasks of them. We found that our model shows the highest performance in Wide and Narrow grid configurations, notably with highly significant performance improvement in the Narrow training setting. Results in Random show relatively low performance, presumably the case where our model has difficulty finding the visual feature that determines whether the two images share the same action context or not in input data with irregular intervals. The lower performance of ablated models implies that the two losses; goal context discriminator loss and task discriminator loss have synergistic effects.

Method	Success ratio (%)		
	Wide	Narrow	Random
L_2 reg.	20.0 (1.6)	15.7 (2.8)	8.7 (2.0)
PSEs	32.4 (8.2)	9.7 (5.2)	**34.1** (9.4)
GAN	22.9 (2.2)	16.5 (3.0)	9.9 (2.1)
HAM (ours)	**34.9** (3.5)	**37.5** (7.1)	19.3 (2.8)
HAM w/o GD	25.6 (4.9)	32.9 (6.0)	13.3 (2.9)
HAM w/o TD	26.7 (4.2)	31.1 (6.4)	15.2 (4.7)

Table 1: Generalization results on Jumping task: we present the generalization results of baseline methods, our model, and ablated models with three types of evaluation grid configuration. We measure the test performances after 20k timesteps training with an imitation agent as in PSEs (Agarwal et al., 2021). The results show the mean success ratio (%) among 286 tasks over 20 runs with different seeds and the standard deviations are in parentheses.



(a) HAM (b) HAM w/o GD (c) HAM w/o TD

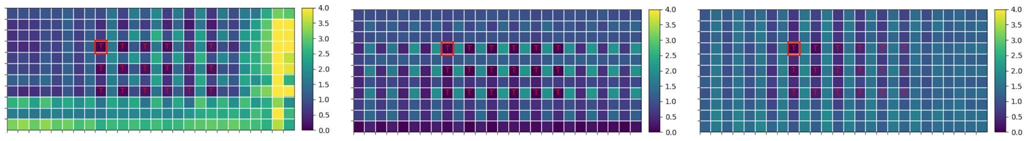
Figure 3: We show the role of each discriminator in our model by visualizing the generated hypothetical observations $G([z_g^i, z_t^j])$ (green-bordered images) in ablated models. We can find that HAM without goal context discriminator cannot separate context information (highlighted with red arrows) from x^i, and HAM without task discriminator cannot separate task information (highlighted with blue arrows) from x^j while our model separates both information successfully.

Disentanglement of visual features. To show the disentanglement quality of the learned features, we visualize hypothetical observations generated at the end of the training in Figure 3. Green-bordered images are generated by using goal context code z_g^i and task feature code z_t^j from the ith observation x^i and jth observation x^j respectively. We can find that task-relevant visual features; *the floor height and the location of the obstacle* (highlighted with blue arrows) of generated hypothetical observation are similar to its first row x^j in Figure 3. We can also see that goal context-relevant feature; *the distance between the agent and the obstacle* (highlighted with red arrows) that determines whether to jump is identical to its first column x^i. In other words, our model can effectively separate goal context-relevant visual features from others, which is in line with the significant performance improvement in the Narrow configuration which measures the extrapolation ability.

Robustness of learned goal context space. To evaluate how robust our learned goal context code is against task changes, we measure the normalized Euclidean distance (Tensmeyer & Martinez, 2017) in the representation space.

$$dist(z_g, z_g^*) = \frac{\|z_g - z_g^*\|_2}{\|z_g^*\|_2} \quad (7)$$

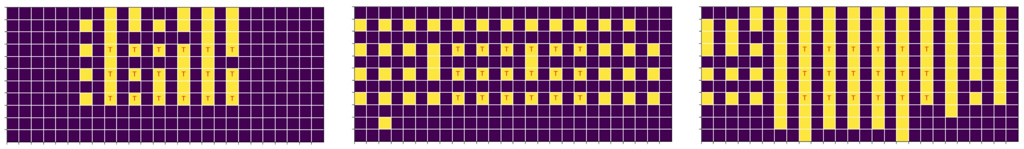
, where z^* is the left top sample in the training grid. Figure 4 shows the qualitative result that how similar our learned context space with training tasks (cells marked with T) and the one with test tasks (other cells in the entire grid) are so that learned policy can be deployed to unseen test data without errors. We observe that the color distributions of training and test task cells are much similar in our model compared to other baselines.



(a) GAN (b) L_2 reg. (c) HAM (Ours)

Figure 4: Invariance grid of learned goal context on task changes: Grid cells marked with T indicate training tasks, and a cell with a red rectangle is a standard data point for normalization. The color of each cell shows the normalized Euclidean distance between the goal context code of the current cell and the standard cell when the optimal action is *to jump*.

We also show how the learned context space connects to the generalization performance in Figure 5.



(a) GAN (b) L_2 reg. (c) HAM

Figure 5: Visualization of generalization performance of HAM and baseline methods in Narrow training grid configuration. Red letter T indicates the training tasks. Yellow tiles are solved tasks and violet tiles are tasks each method couldn't solve.

To quantitatively measure the distribution shift of the goal context code between training and test tasks, we first assume that $dist(z_g, z_g^*)$ measured in each task set follows normal distribution $N(\mu, \sigma^2)$. We then measure the area under the curve (AUC) of probability density function of test context distribution overlapped with train context distribution with the range of $[\mu_{train} - 3\,\sigma_{train}, \mu_{train} + 3\,\sigma_{train}]$. A higher value implies the test context distribution is mostly covered by the policy learned from the train context distribution. Table 2 shows that our model has the highest AUC in all three grid configurations and learns the most robust goal context space against varying task features.

Method	Area under the curve (AUC)		
	Wide	Narrow	Random
L_2 reg.	0.02 (0.06)	0.01 (0.02)	0.03 (0.14)
GAN	0.15 (0.09)	0.33 (0.26)	0.11 (0.07)
HAM	**0.72** (0.15)	**0.60** (0.19)	**0.33** (0.29)

Table 2: Quantitative measure for the robustness of learned context space: we measure the area under the curve of the probability distribution of test context space covered by train context space. The measured area can range between $[0, 1]$. The results show the mean and standard deviation averaged over 10 runs.

4.2 ProcGen benchmark

ProcGen (Cobbe et al., 2020) is a collection of unique environments designed to measure both sample efficiency and generalization in RL. In ProcGen, the train and test environments differ broadly in visual appearance and structure. We follow the environment setup in ProcGen, and we use Proximal Policy Optimization (PPO) (Schulman et al., 2017) algorithm for training our policy network.

Generalization test. We conduct the generalization test with 9 different models, including PPO, PPO with L_2 regularization, HAM (ours), HAM w/o GD (goal context discriminator ablated version), HAM w/o TD (task discriminator ablated version), and RAD (Laskin et al., 2020) with different augmentations. We choose the four representative and best-performing data augmentation techniques; gray, random crop, cutout, and color-jitter in (Laskin et al., 2020). We excluded models based on other representation learning techniques (e.g., contrastive learning (Srinivas et al., 2020), deepMDP (Gelada et al., 2019)), and IDAAC (Raileanu & Fergus, 2021) for the fair comparison because those models are based on different RL algorithms; SAC (Haarnoja et al., 2018), Distributional Q-learning (Bellemare et al., 2017), and DAAC (Raileanu & Fergus, 2021), respectively. We use three OpenAI ProcGen environments: Fruitbot, Jumper, and Climber. For more information about each environment, refer to Appendix A.1.2. We found that our model significantly outperforms all the baselines when the training dataset is small in Table 3. Notably, our model beats all the baselines trained on two times the number of training levels in Fruitbot and Jumper environments when the training dataset is the smallest; this is the most challenging scenario. We also plot the generalization gap over training level and timestep in Appendix A.3.2.

Table 3: We measure the test performances on 1000 test levels after training 20M timesteps using 50, 100 and 200 training levels. The results show the mean and standard deviation (the value after $\pm$) averaged over three runs with different seeds.

	# of levels	PPO	PPO $+L_2$ reg.	HAM	HAM -GD	HAM -TD	RAD (gray)	RAD (crop)	RAD (cutout)	RAD (color-jitter)
Fruitbot	50	6.6 $\pm$ 1.1	10.4 $\pm$2.5	**17.3** $\pm$ 0.4	16.4 $\pm$ 1.6	9.7 $\pm$ 4.9	4.6 $\pm$2.7	4.4 $\pm$2.5	8.1 $\pm$0.3	-1.4 $\pm$0.9
	100	14.5 $\pm$2.9	16.6 $\pm$2.3	**19.6** $\pm$2.1	18.4 $\pm$1.5	18.0 $\pm$0.9	7.6 $\pm$2.0	11.2 $\pm$3.8	14.7 $\pm$0.4	5.3 $\pm$4.5
	200	19.4 $\pm$1.9	20.3 $\pm$0.6	**21.3** $\pm$0.6	20.5 $\pm$1.4	19.9 $\pm$0.6	14.1 $\pm$0.6	16.1 $\pm$4.7	18.5 $\pm$2.6	19.4 $\pm$2.9
Jumper	50	4.8 $\pm$0.2	5.3 $\pm$0.3	**6.0** $\pm$0.1	5.6 $\pm$0.3	5.4 $\pm$0.3	5.0 $\pm$0.2	4.0 $\pm$0.2	5.2 $\pm$0.2	5.6 $\pm$0.2
	100	5.2 $\pm$0.5	5.8 $\pm$0.2	**6.2** $\pm$0.3	6.0 $\pm$0.1	6.1 $\pm$0.2	5.2 $\pm$0.1	5.1 $\pm$0.2	5.6 $\pm$0.1	6.1 $\pm$0.2
	200	6.0 $\pm$0.2	6.3 $\pm$0.1	**6.4** $\pm$0.1	6.4 $\pm$0.1	6.3 $\pm$0.2	5.6 $\pm$0.1	5.2 $\pm$0.7	5.4 $\pm$0.1	5.9 $\pm$0.1
Climber	50	3.4 $\pm$0.2	3.6 $\pm$0.2	**3.7** $\pm$0.1	2.8 $\pm$0.2	2.7 $\pm$0.3	3.3 $\pm$0.1	2.7 $\pm$0.6	3.3 $\pm$0.2	3.4 $\pm$0.1
	100	4.2 $\pm$0.3	**4.4** $\pm$0.2	4.1 $\pm$0.2	3.3 $\pm$0.2	3.2 $\pm$0.2	3.6 $\pm$0.1	2.8 $\pm$0.1	4.1 $\pm$0.3	4.0 $\pm$0.4
	200	4.5 $\pm$0.1	**4.9** $\pm$0.3	4.5 $\pm$0.7	3.8 $\pm$0.2	3.7 $\pm$0.2	4.4 $\pm$0.3	3.2 $\pm$0.2	4.6 $\pm$0.4	4.2 $\pm$0.5

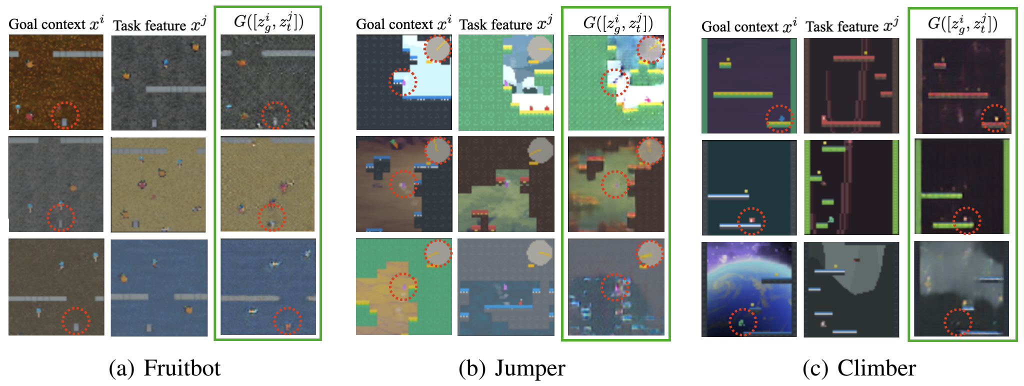


Figure 6: Generated hypothetical observations $G([z_g^i, z_t^j])$ (green-bordered) with the goal context code of x^i and task feature code of x^j. Goal-relevant visual features are highlighted with red-dotted circles.

Disentanglement of visual features. We present the generated observations from HAM around the end of the training process in Figure 6. The results imply that our model successfully separate goal context-relevant latent features from x^i and the task task-relevant ones from x^j. The task code reflects to which *task level* the generated observation belongs (e.g., the background image and the platform color). The goal context code reflects to visual features that influence the *action selection* (e.g., the structure of walls and the placement of fruits in the Fruitbot environment, the direction of the compass needle that points to the location of the carrot in the Jumper environment, and the arrangement of stairs and stars in the Climber environment). One intriguing aspect of our findings is that our model can disentangle not only style-relevant visual features but also important action-relevant visual features in Figure 6. More visual results are in Appendix A.3.2.

5 Related work

Representation learning. Many RL algorithms incorporated a representation learning process to improve generalization performance. This approach obtains robust policies by learning compact vector representations from images (Finn et al., 2015; Dwibedi et al., 2018; Lee et al., 2020; Mazoure et al., 2020). DBC (Zhang et al., 2021) measures behavioral similarity between observations using reward signal and state transition probability, and PSE (Agarwal et al., 2021) compares optimal behaviors of an agent for learning robust representations. CURL (Srinivas et al., 2020) adopts a contrastive auxiliary task that leverages different views of an augmented image. In this paper, we obtain robust representations for generalizable policy by decomposing the latent space of observation space into goal context and task space. Our model presents a new perspective on representation learning in RL. Moreover, the learned hypothetical observation generative model has further possibilities for various applications, such as virtual observation generation in the metaverse.

Regularization and data augmentation. Regularization techniques, known to be effective in supervised learning contexts such as L_2 regularization and dropout (Srivastava et al., 2014), also help RL agents generalize to unseen contexts (Cobbe et al., 2019; Igl et al., 2019). As another attempt, data augmentation techniques in RL such as RAD (Laskin et al., 2020) improve generalization to unseen tasks or levels by simply training on more diversely augmented samples. RAD utilizes different views of the same input and maximizes the MI between features taken from them. While these methods are simple and effective, the task invariance of the learned policy is due to several predetermined factors such as translation, rotation, and color. This limits the applicability of these methods to situations that deal with a specific type of invariance; for example, the color of an object affects the probability of reward acquisition. On the other hand, our model learns task code itself and makes analogies with hypothetical observations generated with different task codes.

Task background modeling. A similar attempt for task background modeling was made in model-based learning. TIA (Fu et al., 2021) learns a distractor model using reward disassociation with negative gradient flow (Ganin & Lempitsky, 2015). TIA acquires reward-related code using the model and utilizes it for planning. On the other hand, our model obtains goal context representation by dividing the latent space learned through mutual-information regularized GAN into goal-related and task-related ones using an additional analogy-making objective.

6 Conclusion

Motivated by the human analogy-making process, this paper presents a novel auxiliary process called hypothetical analogy making (HAM), which enables RL agents to learn compact and explainable goal-relevant features that can generalize to unseen tasks. HAM consists of three parts: goal context and other task-relevant encoding, hypothetical observation generation, and analogy-making. In simulations with the Jumping task and the OpenAI ProcGen benchmark, we show that our model can learn a generalizable behavioral policy by revealing the robust goal context space. Our model outperforms other state-of-the-arts in challenging settings with less training data. In subsequent analyses, we show that HAM learns the proper goal-relevant visual features by visualizing generated hypothetical observations. We also analyze how robust our learned goal context space is by comparing training and test samples' goal context code distribution. Our approach opens up a new possibility of learning generalizable inductive bias by mimicking human cognition. Furthermore, the intuitive design of HAM enables several variants, such as leveraging the sequence of action distributions or incorporating the state value together when measuring goal context similarity.

REFERENCES

Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *International Conference on Learning Representations*, 2021.

John R. Anderson. Cognitive psychology and its implications. 1980.

Paul Bartha. Analogy and analogical reasoning. *The Stanford Encyclopedia of Philosophy (Spring 2019 Edition)*, 2019. URL `https://plato.stanford.edu/archives/spr2019/entries/reasoning-analogy`.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, 2017.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2016.

Karl Cobbe, Oleg Klimov, Christopher Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, 2019.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning*, 2020.

Remi Tachet des Combes, Philip Bachman, and Harm van Seijen. Learning invariances for policy generalization. In *ICLR Workshop Track*, 2018.

Debidatta Dwibedi, Jonathan Tompson, Corey Lynch, and Pierre Sermanet. Learning actionable representations from visual observations. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1577–1584, 2018.

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and P. Abbeel. Learning visual feature spaces for robotic manipulation with deep spatial autoencoders. In *Conference on Robot Learning*, 2015.

Xiang Fu, Ge Yang, Pulkit Agrawal, and T. Jaakkola. Learning task informed abstractions. *ArXiv*, abs/2106.15612, 2021.

Shani Gamrian and Yoav Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. *ArXiv*, abs/1806.07377, 2019.

Yaroslav Ganin and Victor S. Lempitsky. Unsupervised domain adaptation by backpropagation. *ArXiv*, abs/1409.7495, 2015.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. *ArXiv*, abs/1906.02736, 2019.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. URL `https://arxiv.org/abs/1406.2661`.

Tuomas Haarnoja, Aurick Zhou, P. Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cynthia H Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems*, 2019.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, K. Czechowski, D. Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Ryan Sepassi, G. Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *ArXiv*, abs/1903.00374, 2020.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8107–8116, 2020.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *CoRR*, abs/2111.09794, 2021. URL `https://arxiv.org/abs/2111.09794`.

Brenden M. Lake, Tomer David Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2016.

George Lakoff. The neural theory of metaphor. *Law & Rhetoric eJournal*, 2009.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems*, 2020.

Alex X. Lee, Anusha Nagabandi, P. Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In *34th Conference on Neural Information Processing Systems*, 2020.

Sergey Levine, Aviral Kumar, G. Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *ArXiv*, abs/2005.01643, 2020.

Bogdan Mazoure, Rémi Tachet des Combes, Thang Van Doan, Philip Bachman, and R. Devon Hjelm. Deep reinforcement and infomax learning. *ArXiv*, abs/2006.07217, 2020.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *ArXiv*, abs/1411.1784, 2014.

Melanie Mitchell. Abstraction and analogy-making in artificial intelligence. *Annals of the New York Academy of Sciences*, 2021.

Taesung Park, Jun-Yan Zhu, Oliver Wang, Jingwan Lu, Eli Shechtman, Alexei A. Efros, and Richard Zhang. Swapping autoencoder for deep image manipulation. In *34th Conference on Neural Information Processing Systems*, 2020.

Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. *ArXiv*, abs/2102.10330, 2021.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

Luca Scimeca, Seong Joon Oh, Sanghyuk Chun, Michael Poli, and Sangdoo Yun. Which shortcut cues will dnns choose? a study from the parameter-space perspective. *ArXiv*, abs/2110.03095, 2022.

A. Srinivas, Michael Laskin, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *ICML*, 2020.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15: 1929–1958, 2014.

Chris Tensmeyer and Tony R. Martinez. Improving invariance and equivariance properties of convolutional neural networks. 2017.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021.

A Appendix

A.1 Experiment Detail

A.1.1 Jumping task

We provide information about Jumping task (des Combes et al., 2018) and show our architecture designs and additional training details.

Environment description

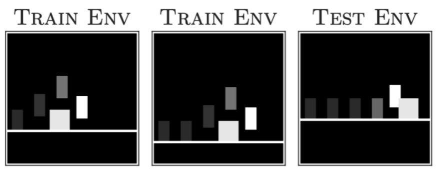


Figure 7: Example MDPs from Jumping task: In Jumping task, the agent (white block) should jump over the obstacle (gray block) without touching it. The shaded trajectory shows the optimal trajectory in training MDPs. The train and test MDPs differ in their observation space which is generated with different floor heights and obstacle positions of various ranges.

In the Jumping task, a white agent has to jump over a gray obstacle without touching it. The agent can choose between two actions: right and jump. The environment is deterministic, with the agent observing a reward of $+1$ at each time step. If the agent successfully reaches the rightmost side of the screen, it receives a reward of $+100$; if the agent touches the obstacle, the episode terminates with a negative reward of -1. The observation space is the 60x60 pixel representation of the environment, as depicted in Figure 7.

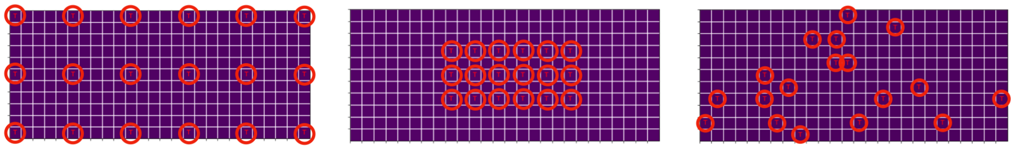


Figure 8: Evaluation grids of Wide, Narrow, and Random grid configurations.

The evaluation measures test performance for 286 tasks of 26 different obstacle locations and 11 floor heights. The training proceeds with only 18 of them. Depending on the distribution of training samples on the evaluation grid, the training type is classified into Wide, Narrow, and Random. The Wide type measures generalization through "interpolation." The Narrow type measures generalization outside the distribution through "extrapolation." Random type assumes a situation in which training and test data are sampled independently in the same distribution. Evaluation grid for each grid configuration is in Figure 8.

Architecture of HAM module

The **encoder** separates the input observation into goal context and task codes, as shown in Figure 9 (left). For the context and task code, the network consists of 3 downsampling convolution layers and 1 convolution layer with kernel sizes 8x8, 4x4, 1x1, and 1x1 and strides 4, 2, 1, and 1. The code with 64 channels is divided into the task and context codes with 32 channels each. The divided context code is fed into a linear layer which computes the policy that outputs the probability of the jump and right actions. For the design of convolution layers, we referred to Agarwal et al. (2021).

The **generator** maps the codes to an hypothetical observation, as shown in Figure 9 (right). The network consists of 1 convolution layer and 3 upsampling convolution layers. The kernel sizes and strides are symmetric to the encoder's downsampling convolution layers.

The **discriminator** is designed to determine if the observation is realistic or not. Each observation is first encoded with3 downsampling convolution layers and 1 convolution layer with kernel sizes 8x8,

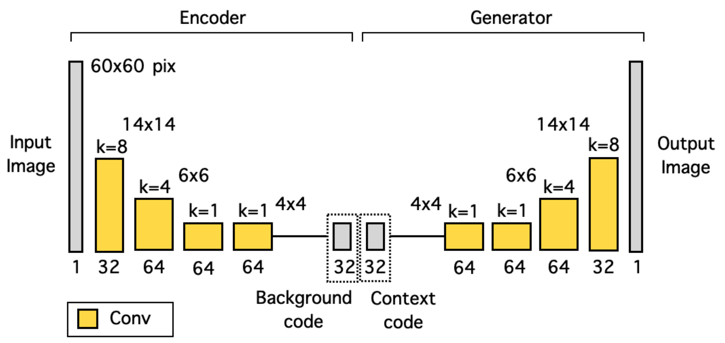


Figure 9: Architecture of encoder and generator

4x4, 1x1, and 1x1 and strides 4, 2, 1, and 1. the final prediction applies 3 dense layers to the flattened representation (see Figure 10). The **goal context discriminator** and **task discriminator** follow the identical architecture with the basic discriminator. Except that the **goal context discriminator** and **task discriminator** take an observation subtracted by the reference observation as an input to determine if the observation has the same task or context as the reference observation. They output logits with length 2 at the last dense layer.

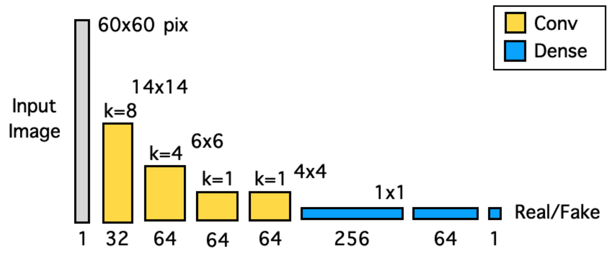


Figure 10: Architecture of discriminator: basic discriminator architecture consists of the feature extractor, which applies 3 downsampling convolution layers and 1 convolution layer with kernel sizes 8x8, 4x4, 1x1, and 1x1 and strides 4, 2, 1, and 1. And then we apply 3 dense layers to the flattened features and outputs the final prediction. The **goal context discriminator** and **task discriminator** shares the identical architecture with the basic discriminator except their final output is logits with length 2.

Training details

At each iteration, we conduct learning on the entire limitation data and train the limitation agent and the HAM module alternately in units of batches of size 256. Most hyperparameters for imitation agents with baseline methods are from Agarwal et al. (2021), such as 256 batch size (with random split), 0.003 learning rate, 0.999 decay rate, and 256 hidden dimensions. We adjusted a few hyperparameters for imitation agent with HAM module as 0.001 learning rate (imitation agent), 0.0003 learning rate (HAM module), and 0.9999 decay rate.

A.1.2 ProcGen benchmark

We provide information about environments we used in ProcGen benchmark (Cobbe et al., 2020), show our architecture designs, and share additional training details.

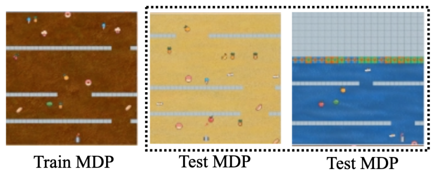
Environment descriptions

A detailed description of Fruitbot, Jumper, Climber environments that we used in ProcGen benchmark (Cobbe et al., 2020) is as follows. Example MDPs of each environment are in Figure 11.

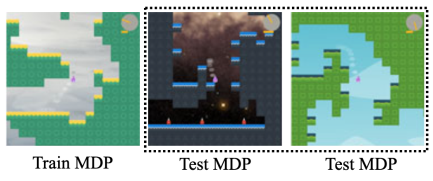
Fruitbot. A simple scrolling game where the player must navigate between gaps in walls and collect fruit along the way. The player receives a positive reward for collecting a piece of fruit and a more significant negative reward for mistakenly collecting a non-fruit object.

Jumper. An open world environment where the player, a bunny, should find the carrot which is randomly located in the map. The screen includes a compass with a needle that displays direction and distance to the carrot. Style of background, map structure, and location of enemy depending on the level.

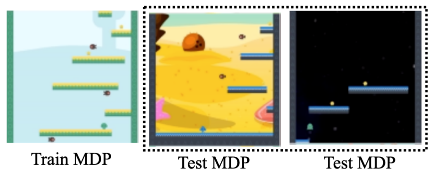
Climber. A simple platformer where the player must climb a sequence of platforms, collecting stars along the way. Collecting a star gives a small reward, and a larger reward is given for collecting all the stars in a level. If all stars are collected, the episode ends.



(a) Fruitbot



(b) Jumper



(c) Climber

Figure 11: Example MDPs from Fruitbot, Jumper, and Climber environments:

ARCHITECTURE OF HAM MODULE

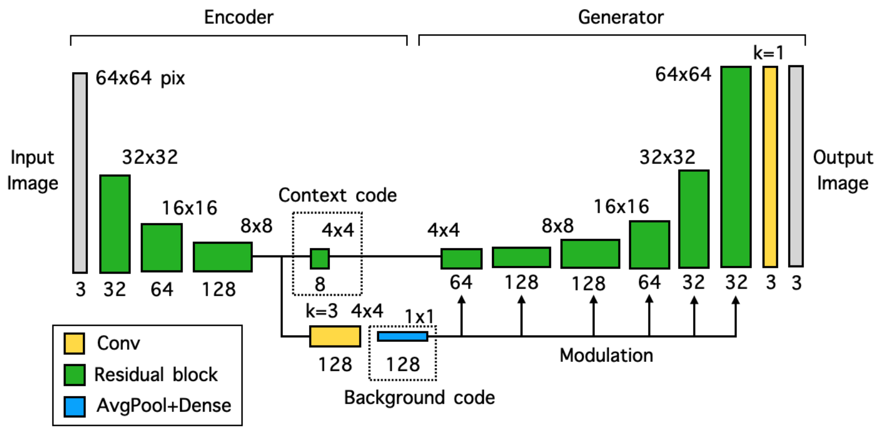


Figure 12: Architecture of encoder and generator

The **encoder** separates the input observation into goal context and task codes, as shown in Figure 12 (left). For the context code, the network consists of 4 downsampling residual blocks (He et al., 2016). For the task code, the network branches off after 3 downsampling residual blocks followed by a convolution layer and an average pooling, and a dense layer. The design of the code shape is from Swap Autoencoder (Park et al., 2020) to impose an inductive bias that task information in ProcGen benchmark is agnostic to positional information like style features. The spatial dimension in the task code is removed using average pooling and no padding in the convolution layer.

The **generator** maps the codes to an hypothetical observation, as shown in Figure 12 (right). The network uses the context code in the main branch, followed by 2 residual blocks and 4 upsampling residual blocks. The task code is injected using the weight modulation layer from StyleGAN2 (Karras et al., 2020). We generate the hypothetical observation by applying a convolutional layer at the end of the residual blocks as in Swap Autoencoder.

The **discriminator** architecture is identical to Swap Autoencoder, except we omitted a few residual blocks as the observation size in ProcGen is 64x64, relatively smaller than the image dataset (e.g., 512x512) in Swap Autoencoder. The architecture of **task discriminator** is in Figure 13. The design is to determine whether an input patch has realistic task features compared to the reference patch.

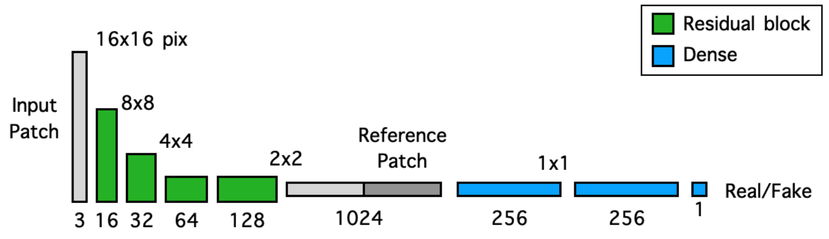


Figure 13: The architecture of task discriminator: the architecture consists of the feature extractor, which applies 3 downsampling residual blocks, 1 residual block, and the classifier with 3 dense layers. When fed into the classifier, we concatenate the flattened feature of the input patch with that of the reference patch in the channel dimension. The classifier outputs logit representing the task similarity between the input and reference images. For the reference patches, we average the extracted features of 4 patches over the batch dimension as in Park et al. (2020).

Training details

Basically, the entire HAM architecture is jointly learned in an end-to-end manner. We also used gradient flow control. For example, when updating the encoder and generator with loss function in Equation (3), the gradient flow to the goal context discriminator and task discriminator was blocked. On the contrary, the gradient flow to the encoder and generator was blocked when updating discriminators using Equation (4) and Equation (5).

For each ProcGen environment hyperparameters, we follow the environment setup in ProcGen (Cobbe et al., 2020), and hyperparameters in PPO (Schulman et al., 2017). We show a full list of hyperparameters for ProcGen experiments in Table 4 and Table 5. The hyperparameters values are unchanged across environments. We also sampled every batch randomly.

Table 4: Hyperparameters used for PPO algorithm and ProcGen environments

Hyperparameter	Value
PPO learning rate (α_{ppo})	0.0003
RMSprop optimizer epsilon	0.00001
RMSprop optimizer alpha	0.99
Discount factor for rewards	0.999
GAE coefficient	0.95
Entropy coefficient	0.01
Value loss coefficient	0.5
Max norm of gradients	0.5
Use unnormalized returns	FALSE
# of training CPU processes	32
# of forward steps in A2C	256
# of PPO epochs	3
# of batches for PPO	8
PPO clip parameter	0.2
# of environment steps to train	20M
distribution of environments	easy
Paint velocity vector	FALSE
Start level id	0

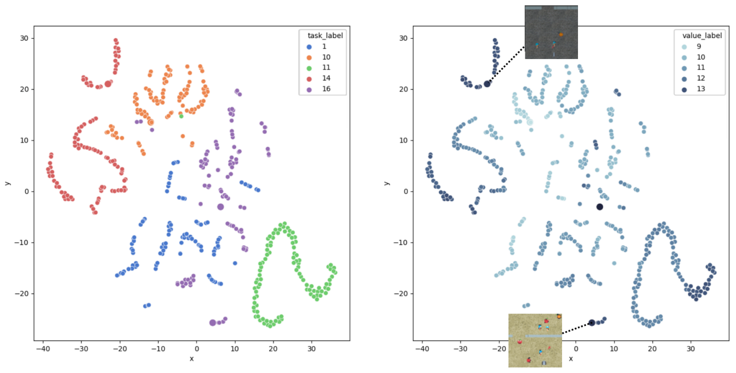
Table 5: Hyperparameters used for HAM module

Hyperparameter	Value
Encoder learning rate (α_{enc})	0.0003
Discriminator learning rate (α_{disc})	0.0001
# of batches for HAM	8
Goal context code dimension	256
GAN loss coefficient (λ_{gan})	0.5
Task loss coefficient (λ_{style})	1.0
Goal loss coefficient (λ_{action})	1.0
# of crop	8
reference # of crop	4
R1 coefficient	10
Task R1 coefficient	1.0

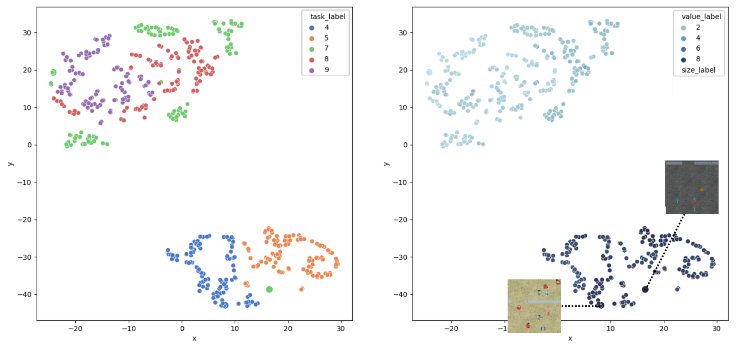
A.2 Robustness of learned goal context space

A.2.1 ProcGen benchmark

To further evaluate the robustness of the learned goal context space, we visualized the goal context code z_g of observations in $2d$ space using t-SNE in Figure 14. Because there was no optimal trajectory data in the ProcGen environment, we could not sample the training and test observations with the same goal context, as in the Jumping task. A sufficiently trained models can serve as an agent with the optimal policy, however, observations can be asynchronously sampled because the training performance differs by each baseline method. Instead, we plot the goal context code z_g of observations in various tasks using two color codes representing their task index and value respectively. We observe that our goal context codes are clustered based on their value similarity showing their robustness against task changes, while the vanilla PPO's context codes are clustered based more on the task index labels.



(a) PPO



(b) HAM

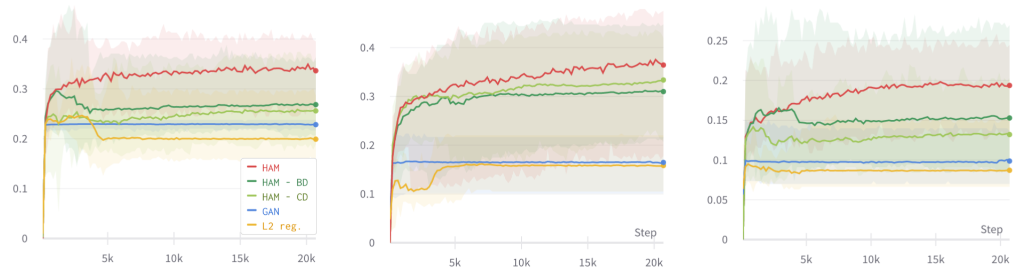
Figure 14: T-SNE visualization of goal context code: visualization of z_g trained with vanilla PPO and PPO + HAM module using 50 training levels in the Fruitbot environment. 100 observations from randomly sampled 5 different tasks. The figure on the left is color-coded with task label and the right figure is color-coded with value label. We can find that the learned context space with our HAM module is more task-invariant compared to the vanilla PPO model.

A.3 Additional results

A.3.1 Jumping task

Generalization curves

We plot generalization performance on the entire grid including unseen floor heights and obstacle positions over training timestep. We observe HAM outperforms other baseline methods including ablated versions of HAM.



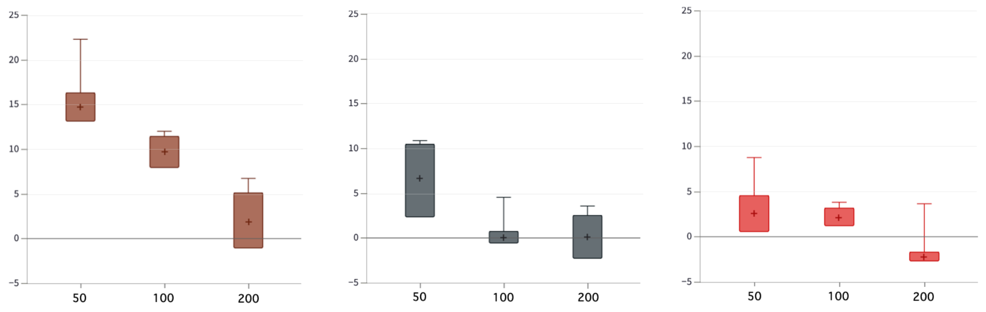
(a) Wide (b) Narrow (c) Random

Figure 15: Generalization performance over training timestep: We plot the average success ratio (# of successful tasks over # of entire tasks in the grid), and the shaded region shows the standard deviation.

A.3.2 ProcGen benchmark

Generalization gaps

We plot the generalization gap over training level in the Fruitbot environment in Figure 16. Vanilla PPO shows a significant gap with a small amount of training data, and the gap gradually reduces as the number of training data increases. On the other hand, our model maintains a small gap from the most challenging training condition. Furthermore, we found that the generalization gap of HAM on 50 training levels is compatible with that of PPO using four times more training data. PPO with L_2 regularization also shows a relatively small gap but not as small as HAM. In summary, the results show that the RL agent with HAM learns task-invariant goal context features by virtue of the hypothetical analogy-making process.



(a) PPO (b) PPO + L_2 reg. (c) HAM

Figure 16: Generalization gap over training level in the Fruitbot environment.

We also plot the generalization gap over timestep in Figure 17. We found HAM shows the smaller gap during the whole training process compared to vanilla PPO agent when the training dataset is small.

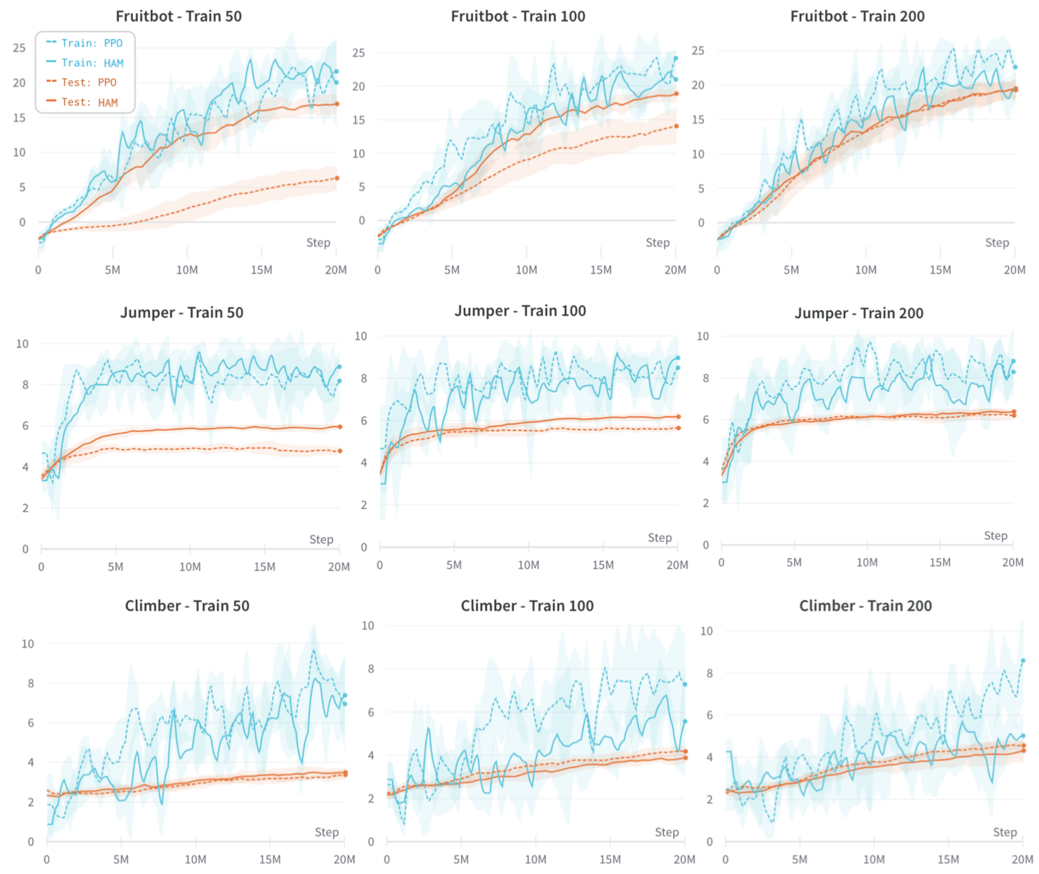


Figure 17: Generalization gap over timestep on Fruitbot, Jumper and Climber.

Generated hypothetical observations

We show additional results of hypothetical observation generation on Fruitbot, Jumper and Climber environments in Figure 18. We also show the hypothetical observation generated by performing task interpolation $G([z_g^i, (\alpha \cdot z_t^i + (1-\alpha) \cdot z_b^j)])$ using a random rate $\alpha \in [0, 1)$ in Figure 19. We can find that the context-relevant features (highlighted with red-dotted circles) are preserved as its task-relevant features vary during the interpolation process.

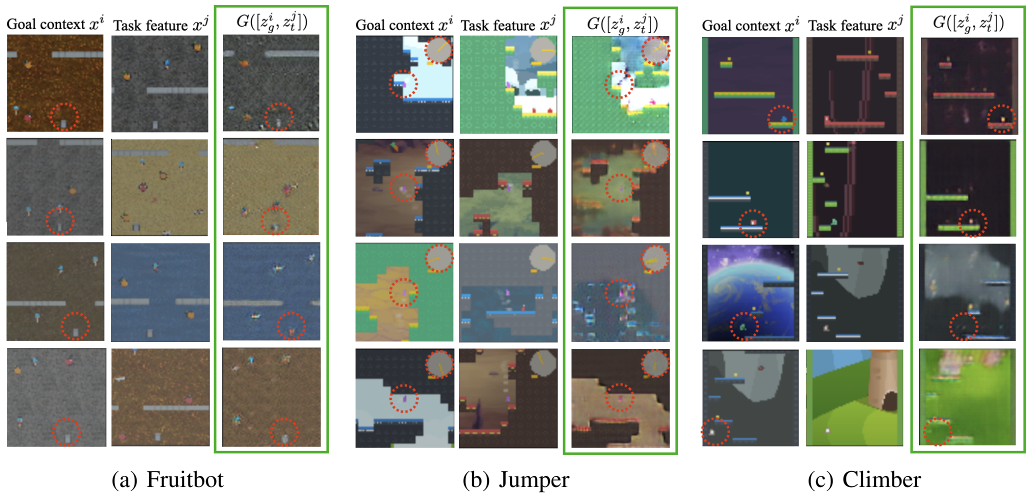


Figure 18: Generated hypothetical observations $G([z_g^i, z_t^j])$ (green-bordered) with the goal context code of x^i and task code of x^j. The goal context features are highlighted with red-dotted circles.

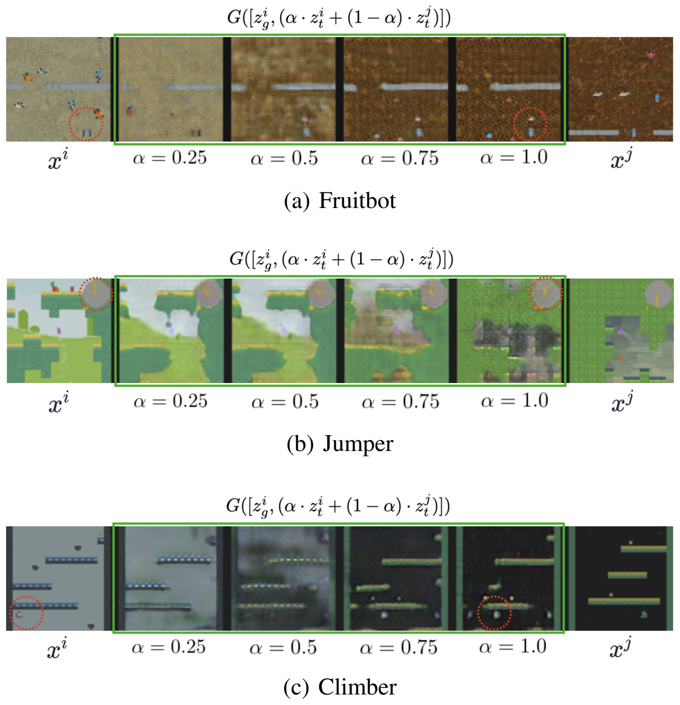


Figure 19: Generated hypothetical observations with task interpolation $G([z_g^i, (\alpha \cdot z_t^i + (1-\alpha) \cdot z_b^j)])$ on Fruitbot, Jumper and Climber environments.