AUXILIARY TASK DISCOVERY THROUGH GENERATE-AND-TEST

Banafsheh Rafiee
Department of Computing Science
University of Alberta
Canada
{rafiee}@ualberta.ca

Sina Ghiassian
Department of Computing Science
University of Alberta
Canada

Jun Jin
Huawei Technologies Canada, Ltd
Canada

Richard Sutton
Department of Computing Science
University of Alberta
Canada

Jun Luo
Huawei Technologies Canada, Ltd
Canada

Adam White
Department of Computing Science
University of Alberta
Canada

ABSTRACT

In this paper, we explore an approach to auxiliary task discovery in reinforcement learning based on ideas from representation learning. Auxiliary tasks tend to improve data efficiency by forcing the agent to learn auxiliary prediction and control objectives in addition to the main task of maximizing reward, and thus producing better representations. Typically these tasks are designed by people. Meta-learning offers a promising avenue for automatic task discovery; however, these methods are computationally expensive and challenging to tune in practice. In this paper, we explore a complementary approach to the auxiliary task discovery: continually generating new auxiliary tasks and preserving only those with high utility. We also introduce a new measure of auxiliary tasks' usefulness based on how useful the features induced by them are for the main task. Our discovery algorithm significantly outperforms random tasks and learning without auxiliary tasks across a suite of environments.

1 INTRODUCTION

The discovery question—what should an agent learn about—remains an open challenge for AI research. In the context of reinforcement learning, multiple components define the scope of what the agent is learning about. The agent's behavior defines its focus and attention in terms of data collection. Related exploration methods based on intrinsic rewards define what the agent chooses to do outside of reward maximization. Most directly, the auxiliary learning objectives we build in, including macro actions or options, models, and representation learning objectives force the agent to learn about other things beyond a reward maximizing policy. The primary question is where do these auxiliary learning objectives come from?

Classically, there are two approaches to defining auxiliary objectives that are the extremes of a spectrum of possibilities. The most common approach is for people to build the auxiliary objectives in pre-defining option policies, intrinsic rewards, and model learning objectives. Although most empirically successful, this approach has obvious limitations like feature engineering of old. At the other extreme is *end-to-end learning*. The idea is to build in as little inductive bias as possible including the inductive biases introduced by auxiliary learning objectives. Instead, we let the agent's neural network discover and adapt internal representations and algorithmic components (e.g., discovering objectives (Xu et al., 2020), update rules (Oh et al., 2020), and models (Silver et al., 2017)) just through trial and error interaction with the world. This approach remains challenging due to data efficiency concerns and in some cases shifts the difficulty from auxiliary objective design to loss function and curriculum design.

An alternative approach that exists somewhere between human design and end-to-end learning is to hand-design many tasks in the form of additional output heads on the network that must be optimized in addition to the primary learning signal. These tasks, called *auxiliary tasks*, exert pressure on the lower layers of the neural network during training, yielding agents that can learn faster (Mirowski et al., 2016; Shelhamer et al., 2016), produce better final performance (Jaderberg et al., 2016), and at times transfer to other related problems (Wang et al., 2022). This positive influence on neural network training is called the *auxiliary task effect* and is related to the emergence of good internal representations we seek in end-to-end learning. The major weakness of auxiliary task learning is its dependence on people. Relying on people for designing auxiliary tasks is not ideal because it is challenging to know what auxiliary tasks will be useful in advance and, as we will show later, poorly specified auxiliary tasks can significantly slow learning.

There has been relatively little work on autonomously discovering auxiliary tasks. One approach is to use meta learning. Meta-learning methods are higher-level learning methods that adapt the parameters of the base learning system, such as step-sizes, through gradient descent (Xu et al., 2018). This approach can be applied to learning auxiliary tasks defined via General Value Functions or GVFs (Sutton et al., 2011) by adapting the parameters that define the goal (cumulant) and termination functions via gradient-descent (Veeriah et al., 2019). Generally speaking, these meta-learning approaches require large amounts of training data and are notoriously difficult to tune (Antoniou et al., 2018).

An exciting alternative is to augment these meta-learning approaches with generate-and-test mechanisms that can discover new auxiliary tasks, which can later be refined via meta-learning. This approach has produced promising results in representation learning where simple generate-and-test significantly improves classification and regression performance when combined with back-prop (Dohare et al., 2021). Before we can combine meta-learning and generate-and-test, we must first develop the generate-and-test approach to auxiliary task discovery so that their combination has the best chance for success. Such an effort is worthy of an entire study on its own, so in this paper we leave combining the two to future work and focus on the generate-and-test approach.

Despite significant interest, it remains unclear what makes a good or bad auxiliary tasks. The meta-learning approaches do not generate human-interpretable tasks. Updating toward multiple previous policies, called the *value improvement path* (Dabney et al., 2020), can improve performance but is limited to historical tasks. The gradient alignment between auxiliary tasks and the main task has been proposed as a measure of auxiliary tasks usefulness (Lin et al., 2019; Du et al., 2018). However, the efficacy of this measure has not been thoroughly studied. Randomly generated auxiliary tasks can help avoid representation collapse (Lyle et al., 2021) and improve performance (Zheng et al., 2021), but can also generate significant interference which degrades performance (Wang et al., 2022).

In this paper we take a step toward understanding what makes useful auxiliary tasks, introducing a new generate-and-test method for autonomously generating new auxiliary tasks and a new measure of task usefulness to prune away bad ones. The proposed measure of task usefulness evaluates the auxiliary tasks based on how useful the features induced by them are for the main task. Our experimental results shows that our measure of task usefulness successfully distinguishes between the good and bad auxiliary tasks. Moreover, our proposed generate-and-test method outperforms random tasks and learning without auxiliary tasks.

2 Background

In this paper, we consider the interaction of an agent with its environment at discrete time steps $t = 1, 2, \ldots$. The current state is denoted by $S_t \in \mathcal{S}$. The agent's action $A_t \in \mathcal{A}$ is selected according to a policy $\pi : \mathcal{A} \times \mathcal{S} \to [0, 1]$, causing the environment to transition to the next state S_{t+1} emitting a reward of $R_{t+1} \in \mathbb{R}$. The goal of the agent is to find the policy π with the highest state-action value function defined as $q_\pi(s, a) \doteq \mathbf{E}_\pi[G_t | S_t = s, A_t = a]$ where $G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ is called the return with $\gamma \in [0, 1)$ being the discount factor.

To estimate the state-action value function, we use temporal-difference learning (Sutton, 1988). Specifically, we use Q-learning (Watkins & Dayan, 1992) to learn a parametric approximation $\hat{q}(s, a; \mathbf{w})$ by updating a vector of parameters $\mathbf{w} \in \mathbf{R}^d$. The update is as follows,

$$\mathbf{w_{t+1}} \leftarrow \mathbf{w_t} + \alpha \delta_t \nabla_{\mathbf{w_t}} \hat{q}(S_t, A_t; \mathbf{w_t}),$$

where $\delta_t \doteq R_{t+1} + \gamma \text{max}_a \hat{q}(S_{t+1}, a; \mathbf{w_t}) - \hat{q}(S_t, A_t; \mathbf{w_t})$ is the TD error, $\nabla_{\mathbf{w_t}} \hat{q}(S_t, A_t; \mathbf{w_t})$ is the gradient of the value function with respect to the parameters $\mathbf{w_t}$, and the scalar α denotes the step-size parameter. For action selection, Q-learning is commonly combined with an epsilon greedy policy.

We use neural networks for function approximation. We integrate a replay buffer, a target network, and the Adam optimizer with Q-learning as is commonly done to improve performance (Mnih et al., 2013).

To formulate auxiliary tasks, a common approach is to use general value functions or GVFs (Sutton et al., 2011). GVFs are value functions with a generalized notion of target and termination. More specifically, a GVF can be written as the expectation of the discounted sum of any signal of interest:

$$v_{\pi,\gamma,c}(s) \doteq \mathbf{E}_\pi[\sum_{k=0}^{\infty} (\prod_{j=1}^{k} \gamma(S_{t+j})) c(S_{t+k+1}) | S_t = s, A_{t:\infty} \pi]$$

where π is the policy, γ is the continuation function, and c is a signal of interest and is referred to as the cumulant. Similarly, a generalized state-action value function $q_{\pi,\gamma,c}(s,a)$ can be defined where the expectation is conditioned on $A_t = a$ as well as $S_t = s$. A control auxiliary tasks is one where the agent attempts to learn a π to maximize the expected discounted sum of the future signal of interest (called a control demon or control GVF in prior work).

To learn these auxiliary tasks, multi-headed neural networks are commonly used where the last hidden layer acts as the representation shared between the main task and the auxiliary tasks (Jaderberg et al., 2016). In this setting, each head corresponds to either the main task or one of the auxiliary tasks and the auxiliary tasks make changes to the representation alongside the main task via backpropagation.

3 Auxiliary task discovery through generate-and-test

We propose a new method for auxiliary task discovery based on a class of algorithms called generate-and-test. Generate-and-test was originally proposed as an approach to representation learning or feature finding. We can think of backprop with a large neural network as performing a massive parallel search in feature space (Frankle & Carbin, 2018). Backprop greatly depends on the randomness in the weight initialization to find good features. The idea of generate and test is to continually inject randomness in the feature search by continually proposing new features using a *generator*, to measure features usefulness using a *tester*, and to discard useless features. This idea has a long history in supervised learning (Sutton et al., 2014; Mahmood & Sutton, 2013), and can even be combined with backprop (Dohare et al., 2021). The same basic structure can be applied to auxiliary task discovery, which we explain next.

We use generate-and-test for discovering and retaining auxiliary tasks that induce a representation useful for learning the main task. That is, the goal is to find auxiliary tasks that induce a positive auxiliary task effect. It is challenging to recognize which auxiliary tasks induce useful representations. To do so, we first evaluate how good each feature is based on how much it contributes to the approximation of the main task action-value function. Here we define the features to be the output of neural network's last hidden layer after applying the activation function. We then identify which auxiliary task was responsible for shaping which features.

Our proposed generate-and-test method for discovering auxiliary tasks consists of a generator and a tester. The *generator* generates new auxiliary tasks and the *tester* evaluates the auxiliary tasks. The auxiliary tasks that are assessed as useful are retained while the auxiliary tasks that are assessed to be useless are replaced by newly generated auxiliary tasks. The newly generated auxiliary tasks will most likely have low utility. To prevent the replacement of newly generated auxiliary tasks, we calculate the number of steps since their generation and refer to that as their age. An auxiliary task can only be replaced if its age is bigger than some age threshold. Every T time step, some ratio of the auxiliary tasks get replaced. We refer to T as the replacement cycle and denote the replacement ratio by ρ. The pseudo-code for the proposed generate-and-test method is shown in Algorithm 1.

Algorithm 1 Generate-and-test for auxiliary task discovery

```
1:  Input: number of auxiliary tasks $n$, age threshold $\mu$, replacement cycle $T$, replacement ratio $\rho$
2:  Initialization:
3:  generate $n$ auxiliary tasks using the generator
4:  randomly initialize the base learning network
5:  set age $a_i$ for each auxiliary task to zero
6:  for Every time step do
7:      do a DQN step to update the base learning network
8:      Increase $a_i$ by one for $i = 1, \dots, n$
9:      update the utility of each auxiliary task $u(\text{aux}^i)$ for $i = 1, ..., n$ using the tester
10:     for Every $T$ time steps do
11:         Find $n\rho$ auxiliary tasks with the lowest utilities such that $a_i > \mu$
12:         replace the $n\rho$ auxiliary tasks with new auxiliary tasks generated by the generator
13:         reinitialize the input and output weights of the features induced by the $n\rho$ auxiliary tasks
14:         reset $a_i$ to zero for the $n\rho$ auxiliary tasks
15:     end for
16: end for
```

Note that the proposed method does not generate-and-test on features but on auxiliary tasks. It, however, does assess the utility of features and derives the utility of the auxiliary tasks from the utility of the features that they induced.

We propose a tester that evaluates the auxiliary tasks based on how useful the features induced by them are for the main task. When following the standard practice of jointly learning the main task and the auxiliary tasks, recognizing

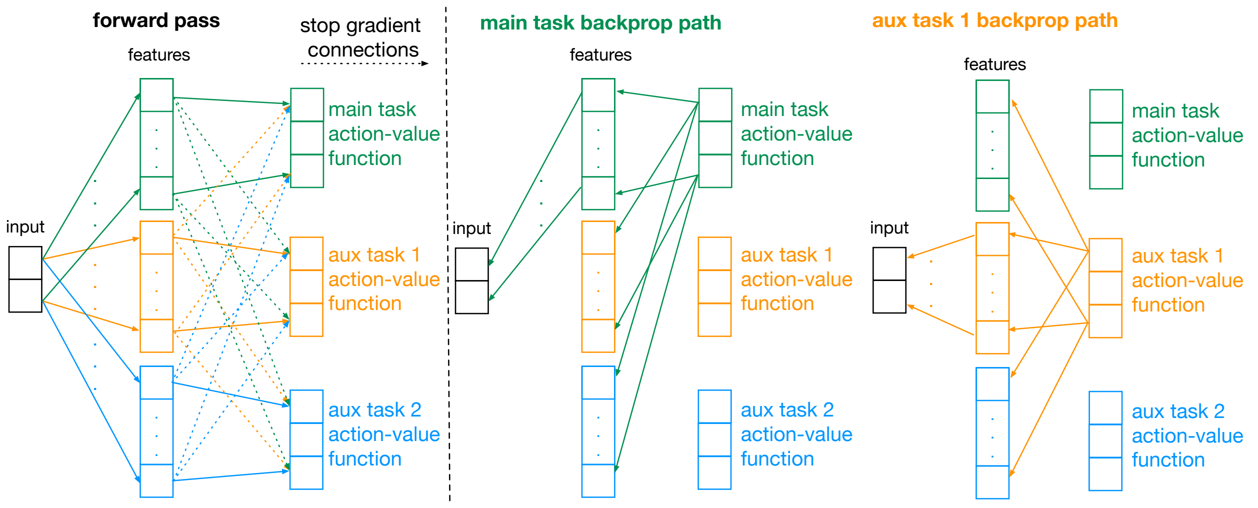
forward pass
features
stop gradient connections
input
main task action-value function
aux task 1 action-value function
aux task 2 action-value function
main task backprop path
features
input
main task action-value function
aux task 1 action-value function
aux task 2 action-value function
aux task 1 backprop path
features
input
main task action-value function
aux task 1 action-value function
aux task 2 action-value function

Figure 1: The forward pass, backward pass for the main task, and backward pass for auxiliary task 1 when using the Master-User strategy for learning auxiliary tasks alongside the main task. All features are used by all tasks in the forward pass but only modified through the gradient backpropagated from one task. The dotted arrows show stop-gradient connections. The gradients does not go back any further from these connections. When using the Master-User strategy, it is clear which auxiliary task was responsible for inducing which feature.

which feature was influenced the most by which auxiliary task is challenging. This is because all features are jointly shaped by all the tasks, both auxiliary and main. To address this issue, we use a strategy for learning the representation where all features are used by all tasks in the forward pass; however, each feature is only modified through the gradient backpropagated from one task. See Figure 1. This learning strategy is similar to the Master-User algorithm proposed for continual recurrent learning (Javed et al., 2021). Therefore, we refer to this learning strategy as the Master-User strategy. When using the Master-User strategy, it is clear which auxiliary task was responsible for inducing which feature.

As we mentioned above, the proposed tester assesses the utility of an auxiliary task based on how useful the features induced by it are for the main task. To assess each feature, the proposed tester looks at the magnitude of the outgoing weights from the feature to the main task action-value function for all actions. The greater the magnitude is, the more important the feature is. The tester also considers the trace of magnitude of each feature: the greater the trace of the feature is, the more it contributes to the approximation of the main task action-value function. The magnitude of the weights times the trace of the magnitude of the feature represents how much the feature contributes to the approximation of the main task action-value function. Therefore, the utility of feature k is defined as:

$$u_k = \bar{f}_k \times \sum_a |w_{ka}^{\text{main}}| \tag{1}$$

where the utility of feature k is denoted by u_k and $\bar{f}_k$ is a *trace* of feature k defined as:

$$\bar{f}_k \leftarrow (1-\tau)\bar{f}_k + \tau f_k \tag{2}$$

where f_k is the value of feature k at the current time step and $0 < \tau < 1$ is the trace parameter. This assessment method is similar to what has been used in generate-and-test on features (Mahmood & Sutton, 2013).

After assessing the utility of the features, the utility of each auxiliary task is set to the sum of the utility of the features shaped by it: $u(\text{aux}^i) = \sum_{k \in F^i} u_k$ where F^i are the features shaped by auxiliary task i.

We combined the proposed tester with a simple generator that randomly generates auxiliary tasks. The auxiliary task are formulated as *subgoal-reaching* GVFs where the continuation function returns 0 at the subgoals and 1 elsewhere (similar to γ in an epsiodic MDP). The cumulant is -1 everywhere and the policy is greedy. The subgoals are randomly selected from the observation space, meaning the agent is learning many policies to reach different parts of the observation space in addition to solving the main task.

4 Experimental results

In this section, we provide empirical results supporting the efficacy of the proposed generate-and-test method for auxiliary task discovery. We include results on two gridworld environments: four-rooms and maze. We also include results on the pinball environment (Konidaris & Barto, 2009), which is widely used in skill chaining, option discovery, and recently model-based planning (Lo et al., 2022). We choose these environments so that we could easily visualize the discovered auxiliary tasks and easily design good and bad auxiliary task as baselines. All environments are episodic.

In the gridworld environments, the goal is to learn the shortest path from start state to goal. The start and goal states are denoted by S and G respectively in Figure 2 and 3. At each cell, four actions are available: up, down, left, and right. There is some degree of stochasticity when transitioning from one state to another: with probability 0.5 the agent will transition in the same direction as the selected action, otherwise it will transition in one of the remaining directions with equal probability. The observation space is described with a one-hot representation with the index corresponding to the agent's position being 1. The reward is -1 on each time step.

In the pinball environment, a small ball should be navigated to the goal in a maze-like environment with simplified ball physics. In Figure 3, the pinball environment is shown with the ball shown by a grey circle. The goal and start states are denoted by S and G respectively. Collision with the obstacles causes the ball to bounce. The observation space is continuous and is described by x, y, $\dot{x}$, $\dot{y}$. The start location and goal location are at $(0.8, 0.5)$ and $(0.1, 0.1)$ respectively. The action space includes 5 actions of increasing or decreasing $\dot{x}$ or $\dot{y}$ and no change to $\dot{x}$ and $\dot{y}$. The reward is -1 at each time step. There is no episode cutoff.

Note that in the original pinball environment, the agent receives a special reward of $10,000$ upon arrival at the goal. Instead we gave a reward of -1 (like every other step) so that the scale of the action-value function for the main task and the auxiliary tasks would not be too different. When learning multiple tasks in parallel, the contribution of each task is determined by the scale of the corresponding value function (Hessel et al., 2019). Therefore, when the scale of value functions are very different, we would need to scale the reward of the main task and the cumulants of the auxiliary tasks appropriately. This issue requires an additional hyper parameter that would give our method an advantage if tuned. For this paper, we decided to focus on the case where the scale of the value function for the main task and the auxiliary tasks are similar.

We used DQN with Adam optimizer as the base learning system. We used a neural network with one hidden layer and $tanh$ activation function. (We used $tanh$ activation function so that the induced features would be all in the same range of $(-1, 1)$; however, our proposed tester should work well when other activation functions are used too. This can be investigated in future work.) For the girdworld environments, the one-hot observation vector was fed to the neural network. The hidden layer size for the baseline with no auxiliary task in four-rooms and maze were 50 and 500 respectively. The replay buffer size for four-rooms and maze were 500 and 1000 respectively. For both four-rooms and maze, we used a batch-size of 16 and target network update frequency of 100.

For the pinball environment, the 4-dimensional observation was normalized and fed to the neural network. The hidden layer size was 500. We used a replay buffer of size $10,000$, a batch-size of 16, and target network update frequency of 200.

4.1 The proposed tester reasonably evaluates the auxiliary tasks

To see how well the proposed tester evaluates the auxiliary tasks, we designed good and bad auxiliary tasks in the four-rooms environment. The hand-designed auxiliary task were formulated as subgoal-reaching GVFs with the good and bad hand-designed auxiliary tasks having hallway and corner subgoals respectively. See Figure 2. We used the Master-User architecture when learning the hand-designed auxiliary tasks.

When learning the auxiliary tasks alongside the main task using the Master-User strategy, the gradient backpropagated from the main task only modifies $\frac{1}{\#\text{auxiliary tasks}+1}$ percent of the features,. For example, in the case of learning the hallway auxiliary tasks, there are 2 auxiliary tasks. Therefore, the gradient backpropagated from the main task only modifies 33.3% of the features.

When including auxiliary tasks, we adjusted the hidden layer size of the network such that the total number of learnable parameters are roughly equal across methods. For example, in the four-rooms environment, for the case of no auxiliary tasks there are a total of $49 \times 50 + 50 \times 4 = 2650$ parameters, with an observation size of (network input size) 49 , hidden layer size of 50, and 4 actions (network output size). For the baseline with hallway auxiliary tasks, we used a smaller hidden layer size of 43 so that it results in roughly the same number of total parameters: $49 \times 43 + 43 \times 4 \times 3 = 2623$. In all the experiments, we kept the number of learnable parameters roughly equal across methods with and without auxiliary tasks.

The hallway auxiliary tasks improved learning in terms of learning speed as we expected (Figure 2, middle graph). The corner auxiliary tasks, on the other hand, hurt the performance at the early episodes and resulted in suboptimal performance in comparison to the hallway auxiliary tasks. As we mentioned earlier, when including auxiliary tasks, we used a smaller hidden layer size such that the total number of learnable parameters are equal to the case of no auxiliary tasks. It is interesting that in this environment with such a small observation space and no partial observability, dedicating a considerable percentage of the learnable parameters to learning auxiliary tasks can result in better performance compared to dedicating all the parameters to learning the main task.

The proposed tester evaluated the hallway and corner auxiliary tasks well, assigning much higher utility to the hallway auxiliary tasks and clearly indicating the corner tasks are bad (Figure 2, right graph). The utility of all hand-designed auxiliary tasks started around the same point. However, the utility of the good auxiliary tasks reached a much higher level compared to the bad auxiliary tasks over time. We also conducted a more thorough experiment in which we included all the cells as subgoals. We observed that the tester evaluated the auxiliary tasks well, assigning higher utility to the auxiliary tasks that accelerated learning the most. These results are included in the appendices.

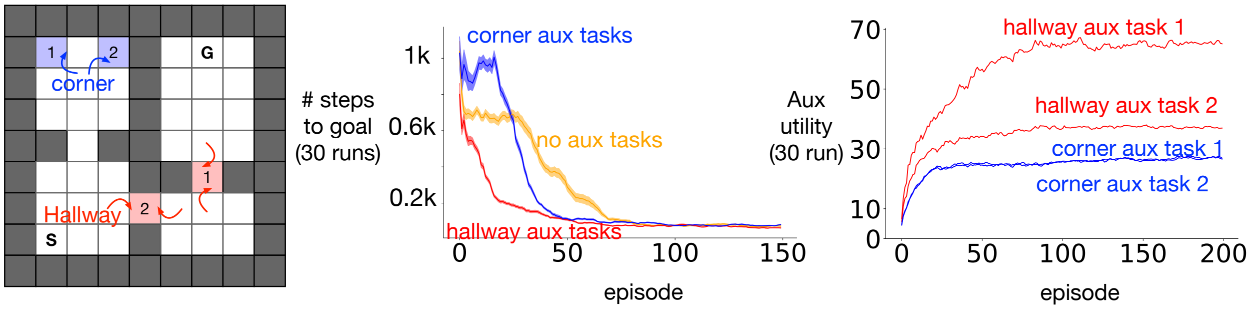

Figure 2: Left: The four-rooms environment with the subgoals corresponding to the good and bad hand-designed auxiliary tasks shown in red and blue respectively. Middle: Hallway auxiliary tasks improved the performance in terms of learning speed. The corner auxiliary tasks made learning slower in the early episodes. Right: The proposed tester evaluated the hand-designed auxiliary tasks well, giving higher utility to the hallway auxiliary tasks. The results are averaged over 30 runs and the shaded regions depict the standard error.

4.2 The generate-and-test method improves over the baseline of no auxiliary tasks

Next, we studied the performance of the base learning system when combined with the proposed generate-and-test method. The generate-and-test method uses the combination of the *random generator* and our proposed tester. The random generator produces subgoal-reaching auxiliary tasks with the subgoals randomly picked from the observation space. More specifically, in the gridworld environments, the subgoals are cells in the grid. In the pinball environment, the subgoals are determined by (x, y) and once the ball is within radius 0.035 of a subgoal, it is assumed that the agent has reached the subgoal.

We included two baselines for comparison which included the base learning system with 1) no auxiliary tasks 2) fixed random auxiliary tasks. All the auxiliary tasks were in form of subgoal-reaching tasks. For the fixed random auxiliary tasks, the subgoals where randomly picked from the observation space and kept fixed throughout learning. The number of auxiliary tasks for the baseline with fixed random auxiliary tasks was set equal to the number of auxiliary tasks for the generate-and-test method. (We also experimented with a baseline that replaces some ratio of the auxiliary tasks randomly instead of keeping them fixed. This baseline can be thought of as a generate-and-test method with a random tester and a random generator. This baseline performed worse than the baseline with fixed random auxiliary tasks. The results can be found in the appendices.)

We systematically swept the step-size parameter and report the performance of the best to ensure a fair comparison. To do so, we ran the baseline with no auxiliary tasks with different values of the step-size for 10 runs. We used the step-size that resulted in the lowest final error (last 10 percent episodes) and reran the baseline with the best step-size for 30 runs to get the final results. We repeated this process for all methods. For four-rooms, maze, and pinball the sweep over the step-sizes included $\{0.000625, 0.0025, 0.01, 0.04\}$, $\{0.00025, 0.001, 0.004\}$, and $\{0.00125, 0.0025, 0.005, 0.01, 0.02\}$.

The generate-and-test method has hyper-parameters of its own: 1) number of auxiliary tasks 2) age threshold 3) replacement cycle 4) replacement ratio 5) trace parameter. For the gridworld environments, we used 5 auxiliary tasks, age threshold of 0, replacement cycle of 500 steps, replacement ratio of 0.2, and trace parameter 0.05. For the pinball environment, we used 4 auxiliary tasks, age threshold of 0, replacement cycle of 500 steps, replacement ratio of 0.25, and trace parameter 0.01.

First, lets take a look at the learning curves for the four-rooms environment shown in Figure 3. The proposed generate-and-test method improved over the baseline with no auxiliary tasks bridging the gap between the baseline with no auxiliary tasks and the baseline with hallway auxiliary tasks. Note that generate-and-test is slower than the baseline with hallway auxiliary tasks. This is because generate-and-test is searching the space of auxiliary tasks, starting with random ones, testing them, and retaining those recognized as useful whereas the baseline with hallway auxiliary tasks starts with reasonably good auxiliary tasks from the beginning. The generate-and-test method outperformed the baseline with no auxiliary tasks in the maze and pinball environments as well (Figure 3).

Generate-and-test also outperformed the baseline with fixed random auxiliary tasks in all three environments. This suggests that the choice of the auxiliary tasks was important and generate-and-test discovered and retained useful auxiliary tasks.

Interestingly, the fixed random auxiliary tasks resulted in significant performance gain over the baseline with no auxiliary tasks in the gridworld environments (Figure 3). This is in line with the findings from the literature suggesting that random GVFs can form good auxiliary tasks for reinforcement learning (Zheng et al., 2021). In the pinball environment, however, the performance gain for the fixed random auxiliary tasks was small. This could be because the number of random auxiliary tasks is only 4 which is relatively small for the pinball environment.

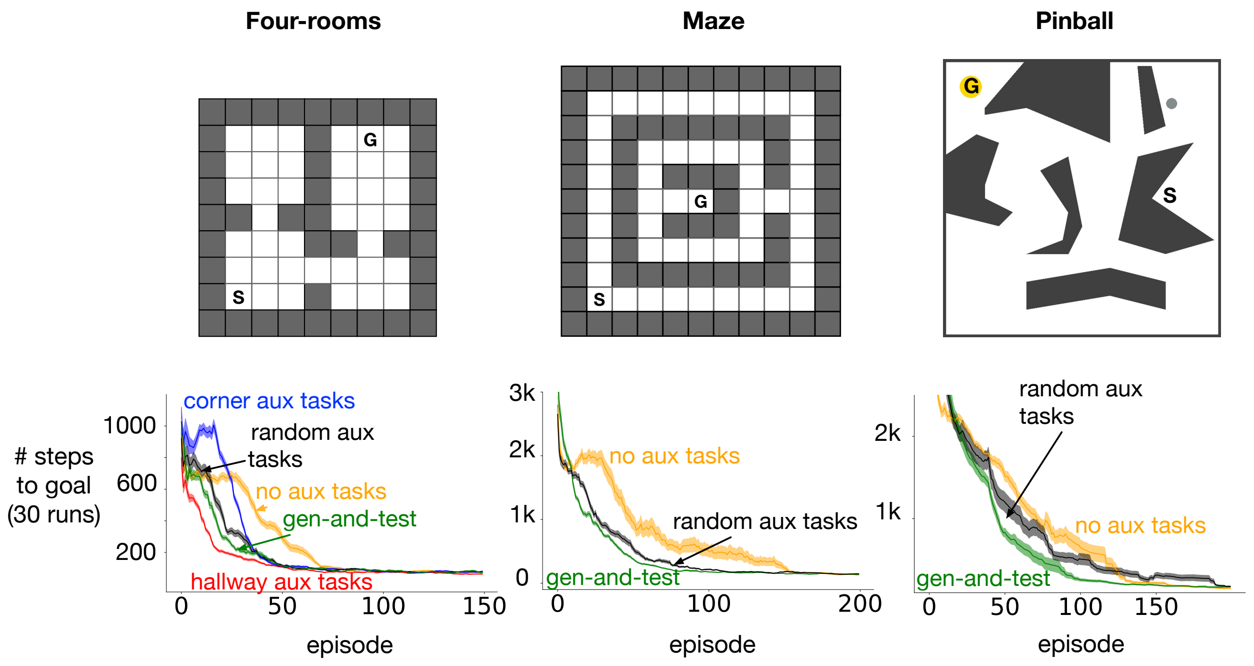

Figure 3: The learning curves for the proposed generate-and-test method (green), the baseline with no auxiliary tasks (orange), the baseline with fixed random auxiliary tasks (black). The results are averaged over 30 runs and the shaded regions depict the standard error. The proposed generate-and-test method improved over the baseline with no auxiliary tasks. Generate-and-test also outperformed the baseline with fixed random auxiliary tasks. Fixed random auxiliary tasks also resulted in performance gain over the baseline.

The auxiliary tasks discovered and retained by generate-and-test are shown in Figure 4. To plot the discovered auxiliary tasks, we ran the generate-and-test method for 30 runs and stored the auxiliary tasks that were retained. The green squares correspond to the discovered auxiliary tasks in the gridworld environments. Darker green indicates that the cell was chosen as a subgoal in many runs. For the pinball environment, the discovered auxiliary tasks are shown by green circles. In the gridworld environments, the subgoals corresponding to the discovered auxiliary tasks were close to the goal states. In the pinball environment, the discovered auxiliary tasks were more concentrated in the central areas—reasonable way-points on the path to the goal.

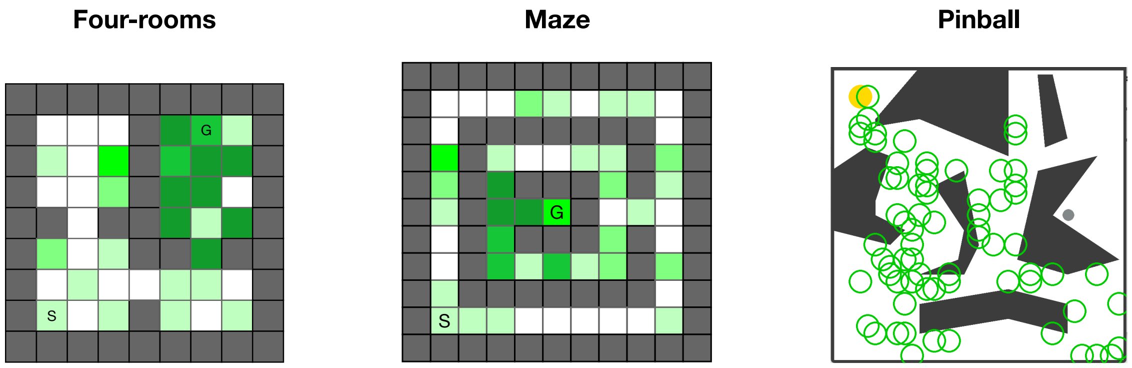

Figure 4: Example discovered auxiliary tasks in the three environments. Generate-and-test discovered reasonably good auxiliary tasks: In the gridworld environments, the subgoals corresponding to the discovered auxiliary tasks were close to the goal states. In the pinball environment, the discovered auxiliary tasks were more concentrated in the central areas.

5 The feature-attainment generator

In this section, we propose and investigate a new auxiliary task generator that has better scaling potential compared with random. While the random subgoal-reaching generator worked well in our experiments, it may not be feasible in cases with a large observation space where the space of possible subgoals is large. To improve the scalability of our generate-and-test method, we propose that the discovery method searches in the space of auxiliary tasks that are about features rather than searching in the space of auxiliary tasks that are about the input observations.

The new generator generates is based on the idea of *feature-attainment* recently introduced for option discovery in planning (Sutton et al., 2022). In feature-attainment auxiliary tasks, the goal is to maximize an individual feature of interest (or component of the representation layer) which we refer to as the target feature. For feature-attainment auxiliary tasks, the continuation function returns 0 when the target feature has its maximum value and 1 otherwise. Recall, we use $\tanh$ activation functions and thus the maximum value a feature can take on is one. The cumulant is -1 everywhere and the policy is greedy.

When generating feature-attainment auxiliary tasks, the question is what the target features should be. We propose that the generator picks the target features only from the features induced by the main task. We designed the generator such that when picking from the features induced by the main task, it looks at the score of the features calculated according to Equation 1 and picks the ones with the highest scores. Therefore, the new generate-and-test method will be searching in the space of auxiliary tasks that are about the features that 1) contribute the most to the main task action value function 2) are induced by the main task.

We tested the generate-and-test method with the feature-attainment generator on the three environments. We used 3, 8, 4 auxiliary tasks for four-rooms, maze and pinball respectively. For four-rooms and maze, we used replacement cycle of 2000 steps, replacement ratio of 0.2, and trace parameter 0.05. For the pinball environment, we used replacement cycle of 1000 steps, replacement ratio of 0.25, and trace parameter 0.01.

The generate-and-test method with the feature-attainment generator improved over the baseline with no auxiliary tasks across the three environments (Figure 5). For the first few episodes, the generate-and-test method with the feature-attainment generator did not improve over the baseline with no auxiliary tasks, unlike the variant with the random subgoal-reaching generator. We speculate that this is because the features are not useful in early learning and thus the feature attainment auxiliary tasks are somewhat arbitrary. With further learning, the features become more relevant to the main task progressively becoming better auxiliary tasks.

6 Conclusions and future work

In this paper, we proposed a new method for auxiliary task discovery. The proposed method uses a generate-and-test approach. We also introduced a new measure of auxiliary task usefulness. Through careful experimentation, we showed that: 1) The proposed tester reasonably evaluates the auxiliary tasks. 2) The generate-and-test method improves over the baseline with no auxiliary tasks.

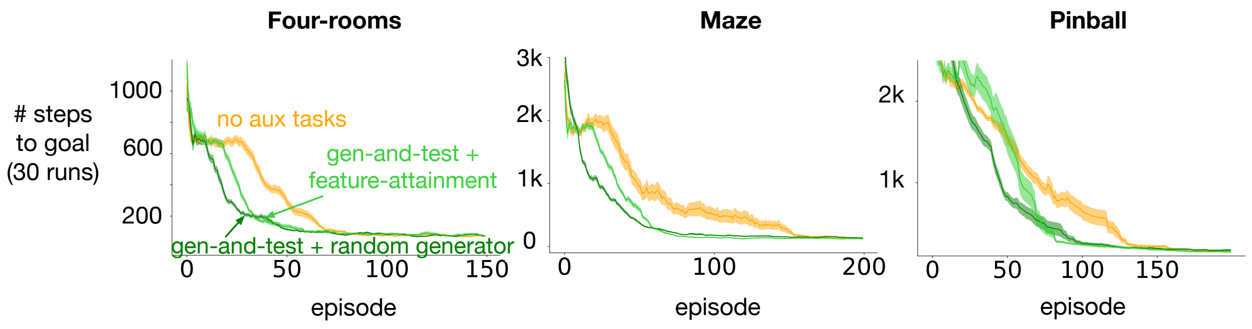

Figure 5: The learning curves for the proposed generate-and-test method with the feature-attainment generator (lime green) and the baseline with no auxiliary tasks (orange). The results are averaged over 30 runs and the shaded regions depict the standard error. The proposed generate-and-test method with the feature-attainment generator improved over the baseline with no auxiliary tasks and is potentially more salable than random auxiliary task generation.

In this paper, we took a first step toward designing a functional generate-and-test method for auxiliary task discovery. However, there is a big space of ideas to try for designing both the generator and the tester. For example, one idea is to sample subgoals from the replay buffer instead of generating random subgoals. This would be similar to the idea of hindsight experience replay (Andrychowicz et al., 2017). There are also ideas from the representation learning literature to explore for improving the tester, examples of which are tested by Shah (2023) and Elsayed & Mahmood (2023). Finally, our results with the feature attainment generator demonstrated the approach can find useful auxiliary tasks, but the next step is to try it in environments where random generation would be infeasible. We could also explore reward-respecting feature attainment subtasks (Sutton et al., 2022), that further promote auxiliary tasks that are useful for the main task.

Another future direction is to combine the meta-learning approaches with the proposed generate-and-test method. The meta-learning approaches can be used to refine the auxiliary tasks discovered by generate-and-test. It would be interesting to test the combination of meta-learning with generate-and-test in the three tested environments and see if their combination will result in discovering good auxiliary tasks, outperforming both approaches in isolation.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.

Will Dabney, André Barreto, Mark Rowland, Robert Dadashi, John Quan, Marc G Bellemare, and David Silver. The value-improvement path: Towards better representations for reinforcement learning. *arXiv preprint arXiv:2006.02243*, 2020.

Shibhansh Dohare, A Rupam Mahmood, and Richard S Sutton. Continual backprop: Stochastic gradient descent with persistent randomness. *arXiv preprint arXiv:2108.06325*, 2021.

Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Mehrdad Farajtabar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018.

Mohamed Elsayed and A Rupam Mahmood. Utility-based perturbed gradient descent: An optimizer for continual learning. *arXiv preprint arXiv:2302.03281*, 2023.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3796–3803, 2019.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Khurram Javed, Martha White, and Rich Sutton. Scalable online recurrent learning using columnar neural networks. *arXiv preprint arXiv:2103.05787*, 2021.

George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems*, 22, 2009.

Xingyu Lin, Harjatin Baweja, George Kantor, and David Held. Adaptive auxiliary task weighting for reinforcement learning. *Advances in neural information processing systems*, 32, 2019.

Chunlok Lo, Gabor Mihucz, Adam White, Farzane Aminmansour, and Martha White. Goal-space planning with subgoal models. *arXiv preprint arXiv:2206.02902*, 2022.

Clare Lyle, Mark Rowland, Georg Ostrovski, and Will Dabney. On the effect of auxiliary tasks on representation dynamics. In *International Conference on Artificial Intelligence and Statistics*, pp. 1–9. PMLR, 2021.

Ashique Rupam Mahmood and Richard S Sutton. Representation search through generate and test. In *AAAI Workshop: Learning Rich Representations from Low-Level Sensors*, volume 10, 2013.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado P van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33: 1060–1070, 2020.

Haseeb Shah. Greedy pruning for continually adapting networks. Master's thesis, University of Alberta, 2023.

Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

David Silver, Hado Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, pp. 3191–3199. PMLR, 2017.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768, 2011.

Richard S Sutton, Steven D Whitehead, et al. Online learning with random representations. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 314–321, 2014.

Richard S Sutton, Marlos C Machado, G Zacharias Holland, David Szepesvari Finbarr Timbers, Brian Tanner, and Adam White. Reward-respecting subtasks for model-based reinforcement learning. *arXiv preprint arXiv:2202.03466*, 2022.

Vivek Veeriah, Matteo Hessel, Zhongwen Xu, Janarthanan Rajendran, Richard L Lewis, Junhyuk Oh, Hado P van Hasselt, David Silver, and Satinder Singh. Discovery of useful questions as auxiliary tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

Han Wang, Erfan Miahi, Martha White, Marlos C Machado, Zaheer Abbas, Raksha Kumaraswamy, Vincent Liu, and Adam White. Investigating the properties of neural network representations in reinforcement learning. *arXiv preprint arXiv:2203.15955*, 2022.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

Zhongwen Xu, Hado P van Hasselt, and David Silver. Meta-gradient reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Zhongwen Xu, Hado P van Hasselt, Matteo Hessel, Junhyuk Oh, Satinder Singh, and David Silver. Meta-gradient reinforcement learning with an objective discovered online. *Advances in Neural Information Processing Systems*, 33:15254–15264, 2020.

Zeyu Zheng, Vivek Veeriah, Risto Vuorio, Richard L Lewis, and Satinder Singh. Learning state representations from random deep action-conditional predictions. *Advances in Neural Information Processing Systems*, 34, 2021.

A Appendix

A.1 A more thorough evaluation of the proposed tester

In this section, we provide the result for a more thorough evaluation of the proposed tester. In this experiment, we included all the cells that are not a wall cell as subgoals. There were a total of 39 subgoals. We used a neural network with a hidden layer of size 240 so each task would modify 6 features. We observed that the tester gave higher scores to the auxiliary tasks with subgoals in the top right and bottom right rooms, that is subgoals closer to the goal state (Figure 6, right).

To test whether the auxiliary tasks with subgoals closer to the goal state are indeed more useful auxiliary tasks in four-rooms, we conducted another experiment where we considered auxiliary tasks with subgoals from each room separately and compared the result with the baseline with no auxiliary tasks. More specifically, to evaluate the subgoals from each room, in each run, we picked 5 subgoals from that room and learned the corresponding auxiliary tasks alongside the main task. We did this so the number of auxiliary tasks from each room would be the same.

We observed that the auxiliary tasks from the top right and bottom right rooms accelerated learning and were indeed more useful (Figure 6, middle). In four-rooms, the tester evaluated the subgoals that were closer to the goal state as more useful. However, this is not necessarily the case in all environments. For example, in the pinball environment the subgoals concentrated in the central areas were recognized as more useful by the tester.

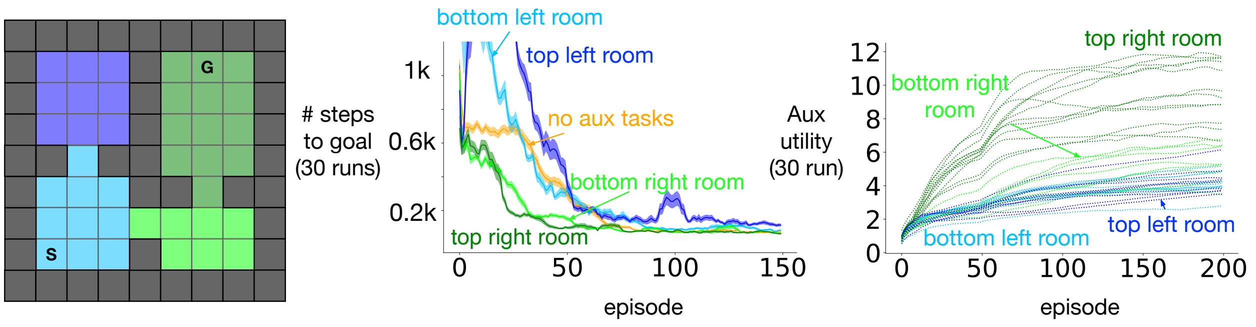

Figure 6: The tester gave higher scores to the auxiliary tasks with subgoals in the top right and bottom right rooms. The auxiliary tasks from the top right and bottom right rooms accelerated learning and were indeed more useful.

A.2 Random tester

In this section, we provide the result for a baseline with a random tester and a random generator. This baseline replaces some ratio of the auxiliary tasks every once in a while similar to the generate-and-test method. However, unlike generate-and-test, it replaces auxiliary tasks at random and not based on any measure of usefulness. Having the auxiliary tasks replaced randomly resulted in a performance worse than that of the fixed random auxiliary tasks. We speculate that the poor performance of the random tester is due to the fact that randomly removing auxiliary tasks can result in the removal of useful auxiliary tasks. As a result, the overall performance might degrade.

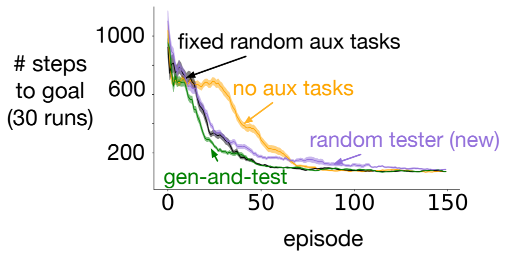

Figure 7: Having the auxiliary tasks replaced randomly resulted in a performance worse than that of the fixed random auxiliary tasks.