Active Hierarchical Imitation and Reinforcement Learning in Continuous Tasks

Yijun Gu
School of Computer Science
Georgia Institute of Technology
yjgu@gatech.edu

Yaru Niu
School of Electrical and Computer Engineering
Georgia Institute of Technology
yaruniu@gatech.edu

Zuoxin Tang
School of Computer Science
Georgia Institute of Technology
ztang315@gatech.edu

***Abstract*—For tasks with sparse reward and long horizon, for example, maze navigation of a complex agent in our case, it is hard to learn useful policies directly with Reinforcement Learning (RL). In order to tackle this problem, we propose to use a hierarchical structure. The low-level controller interacts directly with the environment, and the high-level controller generates subgoals for the low-level controller to follow. We propose to use Data Aggregation Method (DAgger) with human demonstrators so that the agent can transfer to new tasks quickly with a pre-trained low-level policy. On top of that, we also test different ways of uncertainty calculation for active learning and apply it to the DAgger learning process. For the generalization ability of the agent, we experiment on ways to enhance the agent's perception ability with a computer vision-based method, for example, a visual autoencoder.**

***Index Terms*—Imitation Learning, Hierarchical Reinforcement Learning, Active Learning**

I. Introduction

Both Hierarchical Reinforcement Learning (HRL) and Imitation Learning (IL) can be very efficient in an appropriate setting. IL can be very efficient if we have an oracle or human demonstrator with (near) optimal performance. In [31], Sun and Bagnell compare the theoretical bound of performance for both RL and IL under some simple cases, and they show IL outperforms RL for a polynomial factor in general Markov Decision Process (MDP). It also has been shown that some temporal abstraction in HRL can help the agent to explore in more semantically meaningful action space, and thus improve the sample efficiency of whole RL algorithm [25].

Does combining two approaches further help us in complex tasks? In Hierarchical Imitation and Reinforcement Learning (HIRL) [20], the authors combine both methods in discrete state-action space, together with some strategies that restrict the place where learning occurs. They show that their approach can decrease the expert's cost in training the agent, therefore, outperform other hierarchical approaches. In this work, we use a similar approach as in HIRL: training high-level controller with IL and training low-level controller with RL. Since our task, the navigation problem with a complex agent is in continuous state and action space, we cannot directly apply HIRL.

In the continuous setting, we often want to learn a set of policies together that is parameterized by the goal, and the problem is often referred to as contextual policy learning or multi-task learning. Some typical examples are locomotion task and robot arm manipulation task like [1], [27]. In our task, both the high-level controller and low-level controller are parameterized by their goal. Training such a policy is time-consuming. The model training in [27] requires several days of computation. As mentioned above, we expect that the agent can learn fast if it combines IL and uses a hierarchical structure. To further increase sample efficiency in this setting, we adopt an active learning method and compare it with directly applying DAgger [29] on the high-level controller.

As done in [27], we want to use some visual system so that the agent can avoid obstacles instead of blindly following a specific path. In our efforts to train the agent with some perception ability, we also obtain some empirical results of state-space embedding as well as training visual autoencoder. Although we do not achieve our expected goals, we analyze the possible reasons for failures for further improvement.

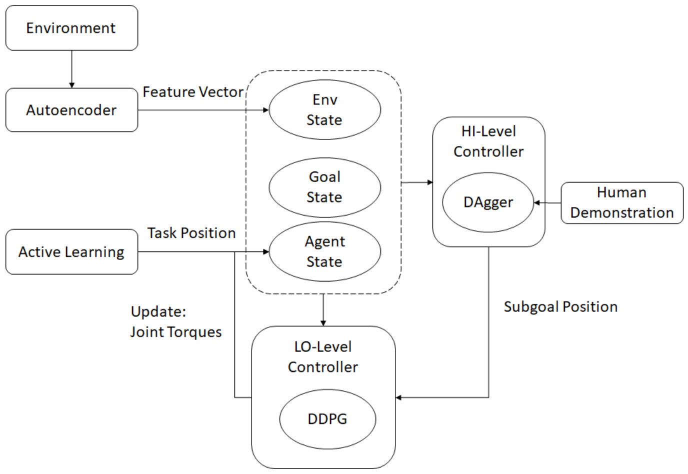


Fig. 1. AHIRL System Module

II. Related Work

A. Dataset Aggregation

Sequential prediction problems arise commonly in practice. Ross and Bagnell propose an iterative meta-algorithm DAgger [29], which trains a stationary deterministic policy, that can be seen as a no-regret algorithm in an online learning setting. The approach is guaranteed to perform well under its induced distribution of states. We incorporate the approach into our hierarchical settings. Other frameworks develop from DAgger like [31] can even outperform expert's performance, we do not use it because it is hard for humans to give an estimation of expected cost-to-go or sample an accurate value of it within a small number of roll-outs.

B. Hierarchical Reinforcement Learning

Building agents that can learn hierarchical policies is a long-standing problem in Reinforcement Learning, for example, decompose MDP into smaller MDPs [7] or Feudal RL that introduces spacial or temporal abstraction into the learning process [6]. Some theoretical results are given, where we can adopt options into MDPs and then obtain a semi-Markov decision process for high-level policies so that the agent will learn faster [32]. Recently, there are many new pieces of research that incorporate deep neural networks with hierarchical reinforcement learning. There are several automated HRL techniques that can work in discrete domains [16] with a manually designed intrinsic reward for low-level policy. Further works like Option-Critic architecture [3] and FeUdal Network [34] address the problem of predefined subgoals as well as extend HRL to continuous space. The main problem of HRL is that high-level policy learning is unstable due to the fact that low-level policy is changing. To address this issue, we can either use Hindsight Experience Replay [1] which is proposed in Hierarchical Actor-Critic (HAC) [21] or optimizing the posterior probability of the subgoal given low-level trajectories [24]. While most of the HRL methods only use a two-layer structure, HAC extends the framework to an arbitrary number of layers and each layer can be trained in parallel. It is further shown that HRL can be regarded as a multi-agent RL task and the non-stationary problem can be addressed in this view [15].

C. Neuron Encoder

Deep Neural Networks (DNN) have been quite successful in unsupervised learning of sparse representations of high-dimensional image data. This includes pre-training deep networks by stacking flat multi-layer perceptions. One of the successful techniques in the well-known MNIST letter recognition tasks is based on a deep autoencoder approach which can learn a mapping of the raw input information to a condensed information vector [11], [17]–[19]. Other methods include learning to predict the next image observation in order to select an ideal action. [8], [10]. In our task, we want to train a compressed embedding of our maze configuration which has rich information so that our agent can take this input and learn how to avoid obstacles. We expect that, with the concatenation of agent state and this embedding as enhanced state and with sufficient training samples, the agent can even learn how to avoid obstacles in an unseen environment.

D. Active Learning

Active Learning (AL) is a popular approach for classification in the semi-supervised learning setting. The expert provides labels only when the agent asks for labels. The approach aims at saving the expert's labeling cost and it is very similar to our goal. In the context of LfD, researchers have proposed active learning methods based on confidence calculation [5], [22] or even learn how to active learn [4]. Despite the fact that it is a popular LfD method, we can only find a few pieces of research on the combination of multi-task learning and AL [9], [30]. The method we adopt in our task is inspired by the work in [30], where the author uses AL for navigation tasks and the agent is selecting tasks (starting and ending points) for the human demonstrator. We also adopt the method by Hafner [12], where they use noise contrastive priors to estimate the reliable uncertainty of the neural networks.

E. Combining RL and LfD

The idea of combining IL and RL is not new [13], [26]. The proposed method Deep Q-Learning from Demonstration (DQfD) in [13] is taking IL as a "pre-training" step by pre-populating the replay buffer with demonstrations. The pre-populated training examples can be used to warm start the agent. In HIRL [20], the combination of IL and RL is in the form of interaction between meta controller and low-level controller instead of warm start the agent. However, the previous works [13], [20], [26] only focus on the application in discrete state and action spaces. In contrast, we build our pre-training IL and HRL in continuous tasks with continuous feedback.

III. Methods

The proposed framework of our work is shown in Figure 1, consisting of a high-level and a low-level controller, AL part, and autoencoder. We will describe each part in the following sections.

A. Hierarchical Imitation and Reinforcement Learning

We introduce a framework Hybrid Hierarchical Imitation Learning (HHIRL) that can effectively learn the levels in a multi-level hierarchy in parallel in continuous spaces. We construct the environment with a two-level hierarchy; the high level corresponds to choosing subtasks and the low level corresponds to executing subtasks. We train our high-level meta-controller using DAgger and low-level controller using a revised Deep Deterministic Policy Gradient (DDPG) method [23].

We develop our method mainly from Hierarchical Actor-Critic (HAC) which is a Hierarchical Reinforcement Learning (HRL) framework. Meanwhile, we have not found a baseline combining HIL and RL structures to solve problems in continuous spaces, so as a whole we will compare our method to leading HRL frameworks that can work in continuous state and action spaces, such as HAC and HIRO. We also use DAgger as our baseline algorithm of the high-level IL hierarchy. The details of the baselines we have implemented or improved up to date are described as follows. A high level description is shown as Algorithm 1.

Algorithm 1 HHIRL

```
Input: Environment, and pretrained low-level policy
Output: Learned high-level policy
 1: Randomly initialize high-level policy $\pi_{\text{HI}}$
 2: Initialize dataset $\mathcal{D} \leftarrow \emptyset$
 3: Initialize mixing factor $\beta \leftarrow 1.0$
 4: repeat
 5:     Sample end goal $g$ and initial state $s_0$ from environment
 6:     repeat
 7:         Get $(s, \pi^*(s))$ from demonstrator
 8:         Generate subgoal $g_{\text{LO}} \leftarrow \beta\pi_{\text{HI}} + (1-\beta)\pi^*$
 9:         $\mathcal{D} \leftarrow \mathcal{D} \bigcup \{(s, a)\}$
10:         Run low-level policy for fix horizon $H$
11:         if Reach update episodes then
12:             repeat
13:                 $\pi_{\text{HI}} \leftarrow \pi_{\text{HI}} - \nabla\ell(s, a)$
14:             until number of update
15:             $\beta \leftarrow \beta/d$
16:         end if
17:         Execute low-level policy for horizon $H$ given $g_{\text{LO}}$
18:     until Goal reached or maximum number of action taken
19: until 100 episodes
```

1) HAC: : HAC trains each hierarchy independently by training each level as if the lower level policies are already optimal. It designs the framework with up to three-level hierarchy and uses an actor-critic method for all levels. To implement this baseline in the four-room maze environment, we trained both 2-level and 3-level HAC on a Macbook with CPU settings. The 2-level HAC was trained with 11949 episodes using about 5 hours, and the 3-level one was trained overnight. In our test, the success rates for 2-level and 3-level are roughly 45% and 85%, respectively. In many failure cases, the subgoals tend to be generated near the walls, which will make the agent turn over, or even in the walls.

2) DAgger: : To improve the performance of HAC, we started with the 2-level hierarchical structure and replaced the high-level RL with IL, which means the human will provide demonstrations of subgoals. We used DAgger method with high-level demonstration data (subgoals) collected incrementally and interactively. Since here providing demonstrations for joint control in lower level is almost impossible, we only consider using DAgger above the base level.

B. Autoencoder

To enable the agent to generalize the environment and take action more precisely, we consider construct an embedding architecture and train our AHIRL algorithm based on both encoded embedding information and temporal information. Recently, deep encoder neural networks (e.g. autoencoder) have shown their effectiveness in many visual analysis tasks as well as image compression and recognition. Therefore, we build a deep autoencoder-based embedding architecture for our environment information compression. The task of our autoencoder is to take visual data as input and learn a mapping of the data information s_t to a condensed information vector $z_t = \phi(s_t)$. Our embedding consists of an encoder, a deconvolutional decoder. We expect this architecture will help the agent adapt more quickly to the previously unseen environment and reduce its cost for human demonstration.

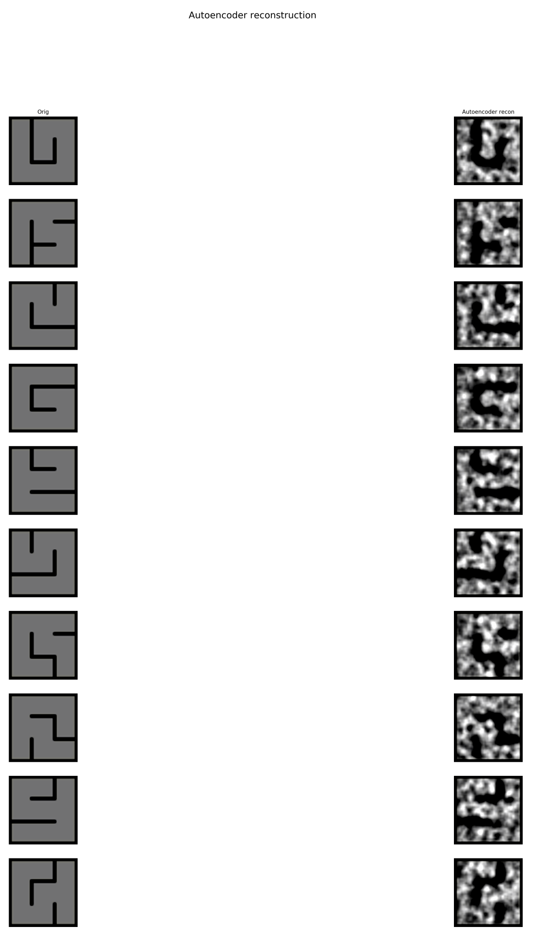


Fig. 2. Autoencoder Reconstruction Sample Results

1) A neural autoencoder: The encoder network compresses a space of possible visual data of the environment to a lower d-dimensional space: $\chi \rightarrow \Re^d$. We first resize the raw image data to 3x64x64 for the input layer using bicubic interpolation. The input layer is followed by a down-sample max pooling layer with a 2x2 filter a 2x2 stride. And the pooling layer is followed by three fully connected layers reducing the number of its predecessor by a factor of 3, 8, 2. All the fully connected layers are activated by an Exponential Linear Unit(ELU) function which tends to converge cost to zero faster and produces more accurate results and has a weight initialization of mean 0 and variance $0.1^2/(inputsize)$ and a bias initialization of 0.01. The final output vector of the encoder is 64. The decoder network produces the reconstructed high dimensional images: $\Re^d \rightarrow \chi$. It consists of a fully connected layer and a transposed deconvolutional layer. The fully connected layer expands the image from 64 to 34x34x64. And the deconvolutional layer

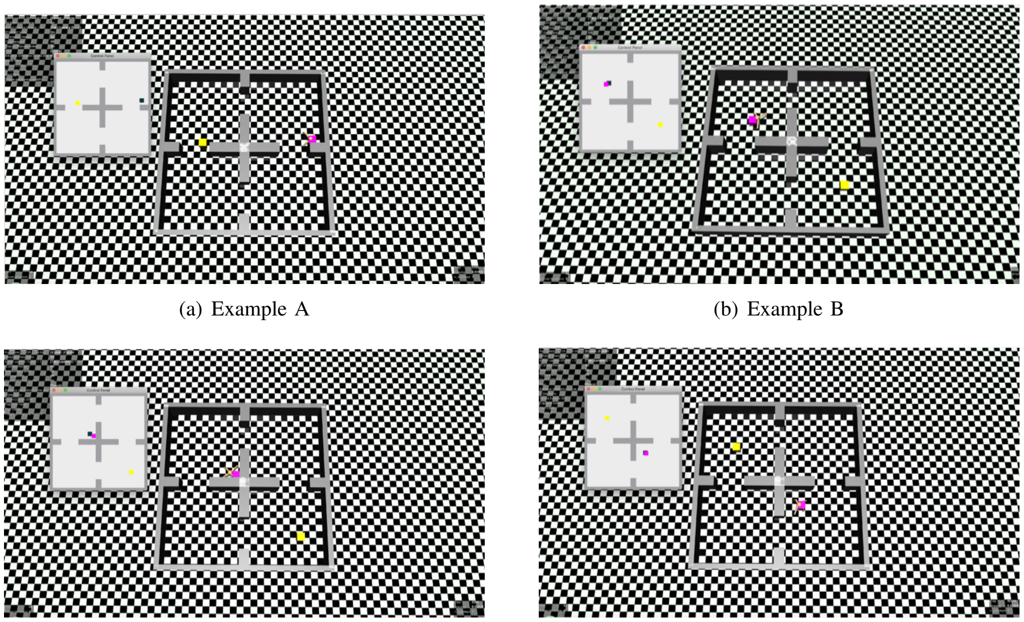

(a) Example A (b) Example B (c) Example C (d) Example D

Fig. 3. Examples of the initialized positions that active learning algorithms are very likely to choose

compresses the image channels from 64 to 3 with a kernel size of 5x5, a stride of 2 and an output padding of 2. And we crop the decoded images to the same sizes of the input images and normalize the outputs.

2) Training: We implement our autoencoder model with Pytorch 10.1. The training set of the deep encoder network consists of 10000 images of randomly generated environment settings. Each image has a size of $340 * 340$ digit points and we manually compress the image to a size of $64 * 64$ digit points for our autoencoder model. We use Adaptive moment estimation (Adam) [14] method to train our network with a learning rate of 0.00001 and optimization with 20 epochs. We measure the mean squared error (squared $L2$ norm) between each element in the input image data x and reconstructed image data y as the criterion. Figure 2 shows the result images of the test samples compressed and reconstructed by our deep autoencoder. From the results, we can see our autoencoder is able to generalize the features of the environments (e.g. wall positions and length) but because of our manual compression, the reconstructed images are not exactly the same as the original one and the noise of the reconstructed images is quite large.

C. Active Learning

1) Encoding to AHIRL: The output-layer of the encoder network delivers an encoding of the high-dimensional input image to a sparse representation. We save the encoding as an embedding into the framework. Every time we train our AHIRL and randomly generate one environment, we encode the environment image to a feature vector and add the vector and the action information together to the state description.

Our AL method is built upon the framework of HHIRL, the framework is shown as Algorithm 2. Different from the "model change" method, we take the idea from epistemic uncertainty, which is a scientific systematic uncertainty due to limited data and knowledge. And in continuous spaces, it is intuitive to measure the uncertainty of the model using the variance of the output by the policy. In our task, with the aim of choosing the initialized sub-state (in accordance with the goal state) of the agent, we design and use two AL methods presented as follows.

Algorithm 2 AHIRL

```
Input: Environment, and pretrained low-level policy
Output: Learned high-level policy
   Randomly initialize high-level policy $\pi_{\text{HI}}$
2: Initialize dataset $\mathcal{D} \leftarrow \emptyset$
   Initialize mixing factor $\beta \leftarrow 1.0$
4: repeat
       Sample end goal $g$ from environment
6:     Sample initial position (initialized $s_g$) by AL
       Run HHIRL in the following steps
8: until 100 episodes
```

2) Noise-Based: In a goal-achieving task, the agent state and goal state do not have to have the same dimension, and commonly the agent state dimension should include that of the goal state. And the noise-based method should be based on the assumption that adding noise to the sub-dimension of the agent state should not change the expect of the optimal action, or it can be meaningless to calculate the variance of the output.

Let s represent the full dimension state of the agent state, s_g the sub-dimension of the agent state which is consistent with the goal state. First, we randomly choose a number of s_g as the candidates of the initialized sub-state. Then we add random uniform noise to other sub-dimension states s_o of the agent state rather than s_g. Then we can get n agent state regarding the same s_g. Then we choose the s_g based on the equation written as

$$s_g^* = \arg\max_{s_g \in S_g} \sum_{i=1}^{n} [\pi_\theta(s_i(s_g)) - \bar{\pi}_{\theta_i}(s_i(s_g))]^2 \tag{1}$$

where S_g is the candidate pool of the s_g, and $s_i(\cdot)$ is the function to get the state after adding an random uniform noise to s_o, and π_θ is the current policy. In our task, s_g is the cartesian position of the agent. The algorithm is shown as Algorithm 4.

Algorithm 3 Noise-based Active Learning

```
Input: Environment, candidate pool size, and number of
    different noises, current high-level policy $\pi_\theta$, end goal $g$
Output: Initialized position $s_g^*$
    Initialize noise buffer $\mathcal{N} \leftarrow \emptyset$
    Initialize state dataset $\mathcal{S} \leftarrow \emptyset$
 3: repeat
        Randomize noise $n$
        $\mathcal{N} \leftarrow \mathcal{N} \bigcup \{n\}$
 6: until number of different noises
    repeat
        Randomly choose a position $s_g$ from environment
 9:     repeat
            Add noise from $\mathcal{N}$ to $s_o$
            Get new state $s$
12:         $\mathcal{S} \leftarrow \mathcal{S} \bigcup \{s\}$
        until number of different noises
        Compute the variance with $S$, $g$ and $\pi_\theta$ based on
        equation (1)
15:     Store variance
        $\mathcal{S} \leftarrow \emptyset$
    until candidate pool size
18: Choose $s_g^*$ based on equation (1)
```

3) Multiple-Policy: Another AL method we have employed is similar to bagging in ensemble learning. We train n policies with the same training dataset and test how much these policies agree given a randomly selected state. This method can be formally written as

$$s_g^* = \arg\max_{s_g \in S_g} \sum_{i=1}^{n} [\pi_{\theta_i}(s(s_g)) - \bar{\pi}_{\theta_i}(s(s_g))]^2 \tag{2}$$

where S_g is the candidate pool of the s_g, and $s(\cdot)$ is a fixed mapping function from s_g to , and $\pi_t heta_i$ is train i policy.

Algorithm 4 Multi-policy Active Learning

```
Input: Environment, candidate pool size, and number of
    different noises, current high-level policy bag $\{\pi_{\theta_i}\}_{i=1}^n$,
    end goal $g$
Output: Initialized position $s_g^*$
    repeat
        Randomly choose a position $s_g$ from environment
 3:     Get new state $s$
        Compute the variance with $s$, $g$ and $\{\pi_{\theta_i}\}_{i=1}^n$ based on
        equation (2)
        Store variance
 6: until candidate pool size
    Choose $s_g^*$ based on equation (2)
```

4) Comments: After training for over 200 hundred episodes with pure DAgger, we use our AL methods to see what kind of initialized position will be likely to be chosen. Figure 3 shows some examples we usually meet. In Figure 3 (b), (c), (d), the agent and the goal form highly symmetric layouts, in which even human demonstrator will have difficulty with choosing the path. Figure 3 (a), (c), (d) show that the agent is very likely to choose some corner as its initialized position and wait for the demonstration feeding.

IV. Experiments

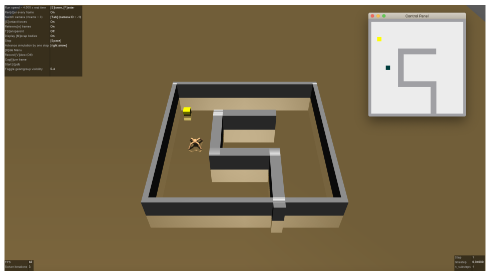
To evaluate our framework, we perform multiple experiments on reaching the target object in a simulated robotics environment developed in MuJoCo [33] as shown in Figure 4. A video showing our experiments is available at https://youtu.be/6NeZo5rELBw

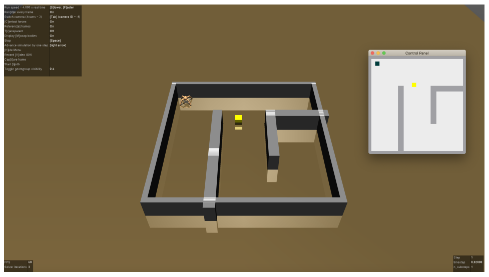
A. Experiment Setup

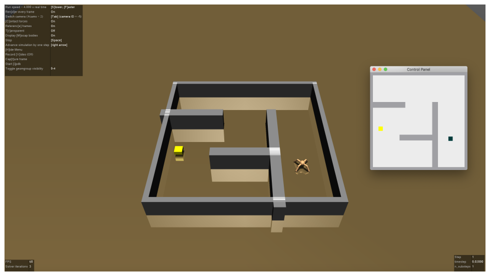
Our environment consists of a 3D maze and a simulated ant agent. The maze is 17mm*17mm with different layouts in each episode. We generate an inconstant number of rectangle walls with Depth First Search in the maze. The ant agent is equivalent to the standard Rllab Ant with a gear range of (-30, 30). We task our agent to reach the specified position in the maze with a time limit of 500 steps. It gets a reward of 0 when it reaches the goal and a reward of -1 otherwise. We run the task on three different experiments: "Pure DAgger", "Noise Based", "Multi Policy". For our experiments, we find twelve human operators to complete 100 demonstration tasks for each algorithm. The demonstrators are required to be able to robustly guide the ant agent to reach the final goal. The expert should always show a subgoal position toward the goal but also make sure not be too far from the current position of the agent.

B. Implementation Details

We compare the performance of the tasks with three different algorithms, "Pure DAgger", "Noise Based", "Multi Policy". For "Pure DAgger", it is our baseline HHIRL algorithm with a high-level meta-controller using DAgger and a low-level controller using DDPG. In DAgger, starting from a probability, we use a decreasing probability with a factor of 1.05 for


(a) maze A


(b) maze B


(c) maze C

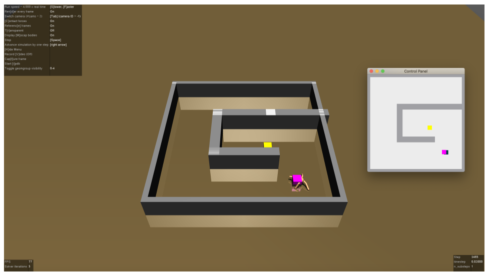
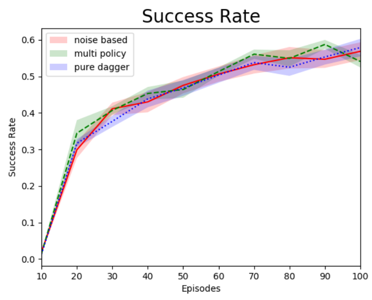
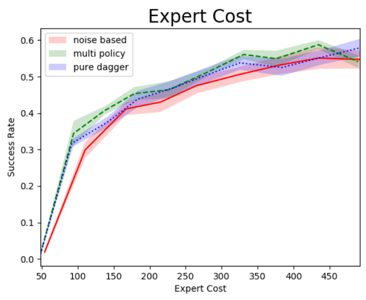

(d) maze D

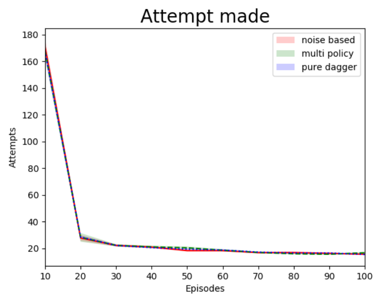
Fig. 4. Maze with different layouts



(a) Success Rate



(b) Expert Cost



(c) Attempts

Fig. 5. Experiment Results over 300 test episodes. (a) Success rate per episode. (b) Success rate versus the expert cost. (c) Attempts per episode.

the agent to take the human demonstration position as its next subgoal. For ”Noise Based”, we add the noise-based AL method to initialize the start location of the agent and train with the same HHIRL algorithm as ”Pure DAgger”. For ”Multi Policy”, we add the multi-policy AL method to initialize the start location and also train with the same HHIRL algorithm.

We train all three algorithms for 100 episodes with human demonstrators continuously giving subgoal positions. One episode is terminated either when the ant agent reaches the goal or it falls or at the time limit. For ”Pure DAgger”, we apply the algorithm for all the 100 episodes. For ”Noise Based” and ”Multi Policy”, we use ”Pure DAgger” to train the first 50 episodes, and use the corresponding algorithm to train the remaining 50 episodes. We test the algorithms over 300 test episodes and record the data.

We measure the performance by success rate, expert cost, and attempts. Success Rate is defined as the average rate of successful task completion over 100 test episodes, on the random environment not used for training. Expert Cost is defined as the average number of demonstrations in one episode over the test episodes. And Attempt is defined as the average number of subgoals generated for the agent to reach the goal in one episode over the test episodes. The results are shown in the following section.

We also test HAC algorithm and has already mentioned the training process and testing results in Method Section A.

C. *Results and Discussion*

We first test our autoencoder embedding and find that the agent can not get enough prior knowledge of the environment from the embedding. The agent always chooses to move toward the center of the maze. We discuss that the result is caused by the insufficient training data and episodes of the embedding as well as the image over-compression discussed in the Method section B.

Then we discuss the results of our experiments. The Figure 5(a) displays the median as well as the range(variance) from minimum to maximum success rate over 12 human demonstration learning policies. It shows that all of the three algorithms have an increased success rate approaching 60% which outperforms flat HAC algorithm(45%). This is because providing human demonstration interactively correct the agent from generating a subgoal position that is too far from the optimal path. Though the three algorithms have a similar ascending trend, they actually have different variances. The "Noise Based" has the largest variance while the "Multi Policy" has the smallest variance. As shown, "Noise Based" is less stable than the other two. The Figure 5(b) displays the same success rate as a function of the expert cost. All three algorithms require more expert costs to get a higher success rate. Among three algorithms, "Multi Policy" saves the most in expert cost and "Noise Based" saves the least. It is because the "Noise Based" generates too many unusual positions and makes the learning policy less stable and relies more on human demonstrators. The Figure 5(c) displays the attempts the agent tries on each episode. And all of them decrease suddenly when trained for the first 20 episodes, and have a slow decrease after 20 episodes. There is no obvious difference among them. Through the results, we conclude that AL does help choose the state or position that needs guidance and multi-policy AL does do better than the flat HHIRL algorithm. However, AL might also increase the uncertainty of the agent due to its asking providing guidance from the corner or the position that human may also not so certain about. Apart from the results, we also find that our performance is largely depended on the quality of the demonstration. Therefore, it is meaningful to help the novice to provide good demonstrations for experiments.

V. Conclusion and Future Work

We presented AHIRL, a hierarchical guidance framework and have shown how AHIRL improves learning multi-level policies in continuous tasks and reduces the cost of expert feedback in both hierarchical reinforcement learning and active imitation learning. In future work, we will consider using Convolutional Neural Network (CNN) for high-level policy learning followed by the idea of Averaged Deep Q Networks(Averaged-DQN) [2]. Rather than constructing a deep encoder embedding, train the image data of the environment with DAgger and output the subgoals. Our IL approach . Due to that, we consider improve IL algorithm by leveraging cost information using Aggregate Values to Imitate method (AggreVaTe) [28] or differentiable AggreVaTe (AggreVaTeD) [31].

Acknowledgment

This is a course project for CS 8803 Interactive Robot Learning in Georgia Institute of Technology. We thank Prof. Matthew Gombolay and Muyleng Leng for their guidance and assistance with this work.

References

[1] 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. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[2] Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 176–185. JMLR. org, 2017.

[3] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[4] Kalesha Bullard, Yannick Schroecker, and Sonia Chernova. Active learning within constrained environments through imitation of an expert questioner. *arXiv preprint arXiv:1907.00921*, 2019.

[5] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34:1–25, 2009.

[6] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

[7] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

[8] Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. *arXiv preprint arXiv:1710.05268*, 2017.

[9] Alexander Fabisch and Jan Hendrik Metzen. Active contextual policy search. *The Journal of Machine Learning Research*, 15(1):3371–3399, 2014.

[10] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[11] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[12] Danijar Hafner, Dustin Tran, Alex Irpan, Timothy Lillicrap, and James Davidson. Reliable uncertainty estimates in deep neural networks using noise contrastive priors. *arXiv preprint arXiv:1807.09289*, 2018.

[13] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Abdul Rahman Kreidieh, Samyak Parajuli, Nathan Lichtle, Yiling You, Rayyan Nasr, and Alexandre M Bayen. Inter-level cooperation in hierarchical reinforcement learning. *arXiv preprint arXiv:1912.02368*, 2019.

[16] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[17] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

[18] Sascha Lange, Martin Riedmiller, and Arne Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2012.

[19] Sascha Lange and Martin A Riedmiller. Deep learning of visual control policies. In *ESANN*, 2010.

[20] Hoang M Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. *arXiv preprint arXiv:1803.00590*, 2018.

[21] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. 2018.

[22] Mingkun Li and Ishwar K Sethi. Confidence-based active learning. *IEEE transactions on pattern analysis and machine intelligence*, 28(8):1251–1261, 2006.

[23] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[24] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018.

[25] Ofir Nachum, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine. Why does hierarchy (sometimes) work so well in reinforcement learning? *arXiv preprint arXiv:1909.10618*, 2019.

[26] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[27] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[28] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

[29] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[30] David Silver, J Andrew Bagnell, and Anthony Stentz. Active learning from demonstration for robust autonomous navigation. In *2012 IEEE International Conference on Robotics and Automation*, pages 200–207. IEEE, 2012.

[31] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3309–3318. JMLR. org, 2017.

[32] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[33] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[34] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549. JMLR. org, 2017.