Imitation Learning with THOR

Albert Liu
albertpl@stanford.edu

Abstract

The recently proposed House Of inteRactions (AI2-THOR) framework [35] provides an simulation environment for high quality 3D scenes. Together with THOR, a Target-driven model is introduced to improve generalization capabilities. Imitation learning or learning by demonstration is known to be more effective in communicating task. In this project, we extend the Target-driven model by exploring both established and state-of-the-art imitation learning methods. First we detail our network architecture and training procedure. Then we show that end-to-end deep neural network based imitation learning methods is applicable to the high-dimension environment with raw visual inputs, such as THOR. Finally we analyze our experiments and results.

1. Introduction

The use of Robots are rapidly expanding in industrial production and our everyday life. However, autonomous Robot control remains challenging, which is the task to optimize the trajectory according to a control policy automatically. It has been active research topic in vision and control domains for many years[2] [18]. Lately, with the advent of deep learning in computer vision and reinforcement learning, visual inputs based learning methods are gaining popularity [19].

In this project, we focus on the problem of navigating in a simulated indoor environment to determine the desired path between a starting point and a given target point using only visual inputs. This problem can be naturally phrased as a reinforcement learning problem that the robot autonomously discover an optimal behavior ($i.e.$ policy) through trial-an-error interaction with its environment. Usually the learned policy through Deep Reinforcement Learning (DRL) approach is dependent on current state and the goal is hardcoded in neural network [24]. Therefore it is task specific and has to be re-trained for a new task even the model may share common network architecture, which is not computationally efficient. This is especially problematic for mobile robot navigation, where the network has to be re-trained for each new goal. It is typically very expensive to go through every single goal. Zhu *et al*. [35] proposes a target-driven model that combines observation images and task goal into inputs and learn a policy that is conditioned on both and thus avoid re-training for each task. This works well when the tasks share similar statistics of trajectories. However, one drawback for such approach is that it is often hard or impractical to specify reward function for policy search [20] [10] in practice.

Imitation learning methods eliminate the needs of handcrafting reward functions which can be more difficult than providing demonstrations. We aim to extend the target-driven model by exploring the imitation learning methods in which agents learn by observing expert's demonstrations. Imitation learning turns policy search into supervised learning and the supervision is provided through expert's demonstrations. The methods we explored include

- Dataset Aggregation (DAgger) [27] add more on policy data to expert's trajectories by labeling agent's trajectories
- Generative Adversarial Imitation Learning (GAIL) [14] which provides effective way of recovering expert's rewards functions and then extract a policy

The training of our methods takes as input: (i) agent's observation images/task goal image, and (ii) expert's trajectories, $e.g.$ pairs of observation images/task goal image and expert's actions. We evaluate our methods over navigation results collected from the simulation environment [35].

2. Related Work

Imitation learning is the problem that agents learn by expert demonstrations. Recent survey papers include (Hussein *et al*. 2017 [15]; Argall 2009 *et al*. [3] ; Calinon 2009 [4]). Two main paradigms within imitation learning are behavior cloning, in which agents mimic expert's behaviors and the policy is trained directly on expects' trajectories (e.g. Bojrski *et al*. 2016 [5]); and inverse reinforcement learning (IRL) (Ng and Russell, 2000 [25]) recovers reward functions from expert demonstrations. Behavior Cloning suffers compounding error (Ross *et al*. [27] [26]) since the

distributions of state-action pairs following agent's policy in test time does not necessarily match the training trajectories from expert demonstrations. DAgger (Ross *et al*. [27]), one of the common solutions, bring agent's trajectories distribution closer to expert's trajectories distribution by asking expert to label additional data points from agent's sample trajectories. On the other hand, many IRL algorithms can be very computationally expensive to run because that it is a nested RL problem by nature, $i.e.$ we need to find the optimal reward functions and then derive the optimal policy given the reward function. Generative Adversarial Imitation Learning (GAIL, Ho and Ermon 2016 [14] uses generative model and discriminative classifier to find a policy that match the distribution of expert's state-action pairs and doesn't assume the knowledge of environment dynamics. Very recently Duan *et al*. (2017 [10]) introduces a framework that can learn policy by a single demonstration of a task. However it is not clear if it is makes sense for navigation tasks where demonstrations for one target is not directly applicable for a different target. We will explore Behavioral Cloning, DAgger and GAIL in our experiments.

Reinforcement Learning (RL, Sutton and Barto, 1998, [32]) has long been applied in robotics tasks. Recent survey papers include (Kober *et al*. 2009 [18]; Li 2017 [21]). RL enables robots to find the optimal behavior through trial and error experience with its environment. RL and Deep RL (DRL) has seen great success recently. For real robots, (Levine *et al*. 2015 [19]) use DRL to learn policy that map the raw images into torques at robot motors. (Silver *et al*. 2016 [30]) combine Behavioral Cloning, policy gradient RL and Monte-Carlo tree search to beat the world best GO players. However, RL tends to need a large number of trajectories and typically reward functions are manually engineered [10]. The former is time consuming and we will show it in our experimental results. The latter is often considered significantly harder than providing a demonstrations (Ng and Russell, 2000, [25]).

In (Mnih *et al*. 2016 [23], multiple agents are trained asynchronously with the shared parameters. Similarly, in our experiments, we will train several agents and update the shared parameters concurrently.

(Bonin-Font *et al*. 2008 [6]) gives a survey on methods for visual navigation, where two major approaches are discussed: *map-based navigation* and *map-less navigation*. Map-based methods require a global map of the environment to make navigation decisions ($e.g.$ [7] [17] or reconstructs a map on the fly [8] [33]. Map-less methods ($e.g.$ [9] [28]) focus on obstacle avoidance given the input images. The target driven model and our extension doesn't require a prior map and considered *map-less*.

Figure 1. Sample images from four different scenes in the THOR (from left to right, top to bottom) : *bathroom_02, bedroom_04, kitchen_02, living_room_03*

Figure 2. Viewpoints of a sample navigation task in the *kitchen_02 scene, from one starting point (top left) to the task goal (bottom). At each viewpoint, the action that the agent takes is represented as a blue arrow at bottom right of the image.*

3. Dataset and Feature

We use The House Of inteRactions (THOR) framework [35] as our environment, which provides high quality images as viewpoints when agents interacts with the environment. Figure 1 illustrates sample images from different scenes. Figure 2 shows a sample visual navigation task. We then describe the details of the environment model before we introduce the methods and experiments.

1. *Dynamics*: There are 20 scenes with 130∼1000 im-

ages per scene and we use four scenes: *bathroom_02, bedroom_04, kitchen_02, living_room_03*. Each scene space is represented as a *grid-world* with a constant step size of 0.5 meters. Each node in the grid has four possible connections to its neighbors at most, mapping to four viewpoints with 90 degree apart. The transition table is deterministic in our experiments and we use it to derive expert policy.

2. *State space*: Each viewpoint at each node is represented by one image with a resolution of $84 \times 84 \times 3$. We call this *agent observation* and this is not to be confused with the *observation* in MDP and will be clarified further. The task goal is also represented as one of such images which gives the flexibility for specifying any new instances of the tasks. The idea of having task goal and agent observation interchangeable and sharing common network layers across tasks makes it possible to transfer knowledge from one navigation task to another. Essentially both agent observation and task goal are concatenated as the MDP observation/state. The rest of the report will use observation to mean agent observation for simplicity. We take transfer learning approach and use the extracted $2048D$ feature vector for each image which is pretrained on ImageNet with *Resnet-50*. During training, the two *ResNet-50* are frozen and therefore CNN codes from the *ResNet-50* is our image features. And we use 4 frames of images to represent the observation and goal respectively. So the state space is a high dimension space with a dimensionality of $2048 \times 4 \times 2$.

3. *Actions space*: THOR models actions at a higher level of abstraction above physics and mechanics details. For our navigation tasks, it considers four actions: moving forward, moving backward turning left and turning right with a constant step size. Each forward/backward move agent exactly one node away from its current location and the viewpoint is updated accordingly, assuming the action is a legal move. If the action is not defined, $e.g.$ agent is against the wall and the action is moving backward, the agent stays put and viewpoint doesn't change. Therefore the action space is a discrete set of commands, $i.e. forward, backward, left, right$

4. *Reward design*: As we use Imitation learning, we don't need to manually engineer the reward functions.

4. Methods

4.1. Problem Formalization

We consider the problem as minimizing the steps the agent takes to go from any start location to a given target point by executing sequence of actions. Each sequence contains finite number of tuple $\{$agent observations s, actions a, rewards $r\}$ and one common goal g. We call the pair s, g the state of the agent. Each sequence begins by drawing an initial location (one node in the scene space), from distribution $\mu(s_0)$ and form the initial state s_0, g. On each subsequent step $t = 1, 2, \ldots$, agent draw an action a_t from distribution $\pi(a_t|s_t, g)$. π is called stochastic policy which gives probabilistic mapping from observation-goal to action. The environment then returns reward and determine the next location, defined by transition table based function $env(s, a)$, which is unknown to the agent. The *terminal state* (s_T, g) is the location agent arrives at goal g or whenever agent perform maximum number of steps in one sequence. This can be described by

$$\begin{aligned} s_0 &\sim \mu(s_0) \\ a_0 &\sim \pi(a|s_0, g) \\ s_1, r_0 &= env(s_0, a_0) \\ a_1 &\sim \pi(a|s_1, g) \\ s_2, r_1 &= env(s_0, a_0) \\ &\ldots \\ a_{T-1} &\sim \pi(a|s_{T-1}, g) \\ s_T, r_{T-1} &= env(s_{T-1}, a_{T-1}) \end{aligned}$$

The sequence of state, actions, rewards, goal makes one trajectory τ. We define expert as the agent that has access to the transition table and therefore can derive the optimal policy (i.e. expert policy π_E), $e.g.$ using depth first search algorithm. The problem objective is defined as minimizing T with respect to π, given the expert trajectories τ_E which is sampled from expert policy π_E.

Then we introduce a notation. We use expectation with respect to a policy π to denote the expectation with respect to the trajectories τ which is sampled from π, initial station distribution $\mu(s_0)$ and environment transition function $env(s, a)$, $e.g.$

$$E_\pi[f(x)] = E_\tau[\sum_{t=0}^{T-1} \gamma^t f(x|\pi)]$$

where $f(x)$ is any function that maps $x \in R^d$ to a scalar This is a slight abuse of the notation as π is not a random variable that we average over to compute the expectation. π is the condition we draw the random variable, $i.e. \tau$. But it allow us to rewrite the problem objective as

$$\max_\pi E_\pi[r(s, g, a)]$$

where $r(s, g, a)$ is the reward function that is unknown to the agent and only the output is given to agent.

4.2. GAIL

The idea behind GAIL is to match the distribution of state-action in agent's trajectories (called occupancy measure in [14]) to expert's. A generative model G is trained to produce agent's stochastic policies π_θ. A discriminative classifier D_ω is used to distinguish the trajectories drawn from π_E and π_θ. And the training objective is to find the saddle point for

$$E_{\pi_\theta}[\log(D_\omega(s,g,a))] + E_{\pi_E}[\log(1 - D_\omega(s,g,a))] - \lambda H(\pi)$$

Where $H(\pi)$ is the casual entropy and defined as

$$H(\pi) = E_{\pi_\theta}[-log\pi(a|s,g)]$$

Intuitively, G is trained to minimize the objective function to confuse D whereas D is optimized to maximize the objective function. And there is clear connection to the Generative Adversarial Nets (GAN) [11].

From IRL's perspective, the discriminator D can also be interpreted as discovering the cost function c from a family of functions C with the optimization objective of

$$\max_{c \in C} E_{\pi_\theta}[c(s,g,a)] - E_{\pi_E}[c(s,a)] - H(\pi)$$

if the cost function, which is the negative of reward function (this is to be consistent with notations in [14]), is defined as

$$c(s,g,a) = log(D_\omega(s,g,a))$$

4.2.1 Network architecture

Fig 3 illustrates the network architecture of our model for GAIL. We use three neutral networks: parameterized stochastic policy is approximated as policy network denoted as π_θ; baseline estimator denoted as V_ϕ which is an approximation of the state based value function; discriminative classifier denoted as D_ω.

Policy and value network share most of the layers. Inputs to both policy and value network are observation images and task goal images. The policy and value network can be split into feature extraction layer (two ResNet-50), siamese layer, scene specific layers. During training, the weights of pre-trained ResNet-50 is not updated. We keep the siamese network from the target-driven model as the embedding layer to learn the mapping between observation-goal feature space to the same spatial embedding space. The policy network and value network share these two layers because we believe both policy and value estimator can benefit from the same learned embedding feature which is supposed to represent the geometric relation between current location and target goal [35]. For scene specific layers, separate FC networks map the embedding feature into probability distribution over actions for policy network and scalar baseline estimator for value network respectively.

We now introduce the training procedure. During training, we sample trajectories from agents and experts and then alternate between updating discriminator D_ω, value network V_ϕ and policy network π_θ. Details are described in the following sections.

4.2.2 Training for Value network V_ϕ

The loss function for value network is MSE between value network output $V(s,g)$ and an estimation of the value function based on D_ω.

$$L = (V(s,g) - \hat{V})^2$$

Specifically the value function is estimated as

$$\hat{V}(s,g) = E[\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} | s_t = s; a_{t'} \sim \pi]$$

The value network is used as baseline estimator to reduce the variance when estimate gradients for updating policy network. We use the discounted version to downgrade the effects of rewards in the future on current state, in order to reduce estimate variance at cost of bias.

4.2.3 Training Discriminative classifier D_ω

We follow [31] to use WGAN-GP [12] loss function to train discriminator D. Algorithm 1 describes the update procedure for ω for one trajectory.

Algorithm 1 Update D with WGAN-GP loss for one trajectory. By default, $\lambda = 10, \beta_1 = 0, \beta_2 = 0.9$

for i=1,...,m **do**
  Sample $(s_E, a_E \sim \pi_E)$ from expert trajectory
  Sample $(s_\theta, a_\theta \sim \pi_\theta)$ from agent trajectory
  Sample a random number $\epsilon \sim U[0,1]$
  $\hat{s} \leftarrow \epsilon s_E + (1-\epsilon)s_\theta$, $\hat{a} \leftarrow \epsilon a_E + (1-\epsilon)a_\theta$
  $L^{(i)} \leftarrow D_\omega(s_\theta, g, a_\theta)] - D_\omega(s_E, g, a_E)] + \lambda(||\nabla_\omega D(\hat{s}, g, \hat{a})||_2 - 1)^2$
end for
$\omega \leftarrow Adam(\nabla_\omega \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, \omega, \alpha, \beta_1, \beta_2)$

4.2.4 Training policy network π_θ

Following [14], we use TRPO [29] to update policy network. The idea behind TRPO is to update policy so that it improve certain surrogate objective but changes as little as possible, which is measure by KL divergence. The surrogate objective function is an local approximation of the expectation of the return function over trajectories and defined as

$$L(\theta) = E_{s,a \sim \pi_{old}}[\frac{\pi(a|s,g)}{\pi_{old}(a|s,g)} A^{\pi_{old}}(s,g,a)]$$

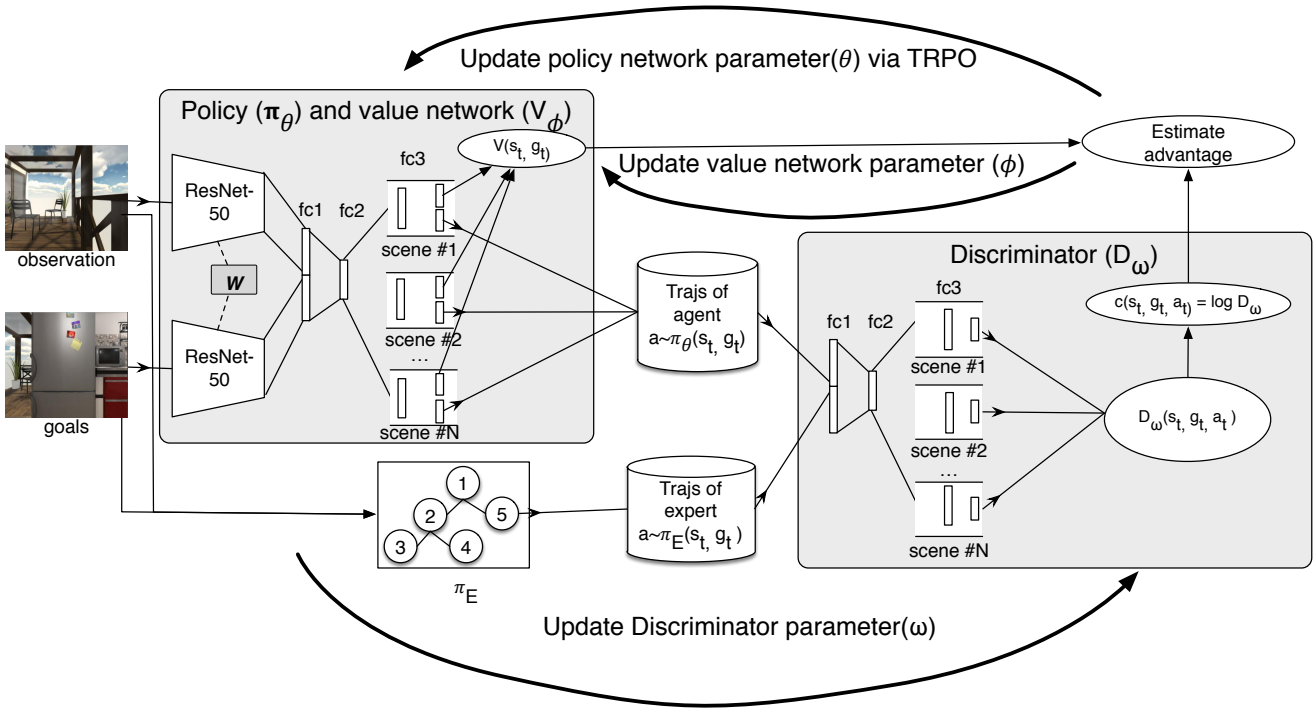

Figure 3. Network architecture overview for GAIL: parameterized stochastic policy is approximated as policy network π_θ; baseline estimator V_ϕ which is an approximation of the state based value function; discriminative classifier denoted as D_ω

subject to $\overline{KL}_{\pi_{old}}(\pi_\theta) < \delta$ where δ is the upper bound KL divergence.

Let $F = \frac{\partial^2 \overline{KL}_{\pi_{old}}(\pi_\theta)|_{\theta=\theta_{old}}}{\partial \theta^2}$ and $g = \frac{\partial L_{\pi_{old}}(\pi_\theta|_{\theta=\theta_{old}}}{\partial \theta}$

Algorithm 2 TRPO update for one trajectory, by default $\zeta_1 = 0.5, \zeta_2 = 0.1$

1: **for** i=1,. . . ,m **do**
2: $r^{(i)} \leftarrow \log D_\omega(s^{(i)}, g^{(i)}, a^{(i)})$
3: $v^{(i)} \leftarrow V_\phi(s^{(i)}, g^{(i)})$
4: $A^{(i)} \leftarrow r^{(i)} + \gamma v^{(i+1)} - v^{(i)}$
5: **end for**
6: $L(\theta) \leftarrow \frac{1}{m}\sum_{i=1}^{m} \frac{\pi(a|s,g)}{\pi_{old}(a|s,g)} A^{(i)}$
7: $g \leftarrow \nabla_\theta L(\theta)$
8: $s_{unscaled} \leftarrow F^{-1} g$
9: $s \leftarrow \sqrt{\frac{2\delta}{s^T_{unscaled} F s}} s_{unscaled}$
10: **for** i=1,. . . ,m **do**
11: $\theta = \theta_{old} + \zeta_1^{i-1} s$
12: **if** $\overline{KL}_{\pi_{old}}(\pi_\theta) < \delta$ AND $L(\theta) - L(\theta_{old}) > \zeta_2 * s^t g$ **then**
13: **break**
14: **end if**
15: **end for**
16:

We describe the TRPO update in Algorithm 2. Step 8 is approximated using any conjugate gradient algorithm, such as *scipy.sparse.linalg.cg* in [16]. Step 10 to 15 is called line search, which is to ensure improvement over surrogate objective while the KL divergence constrains are satisfied. Note both surrogate objective and KL divergence are non-linear in parameter θ space.

4.3. DAgger

4.3.1 Network architecture

Fig 4 illustrates the network architecture for DAgger. Inputs to policy network, which it share architecture with GAIL, are observation images and task goal images. The output of the policy network is a probability distribution over action.

4.3.2 Algorithm

Algorithm 3 describes the steps for DAgger training procedure. BC has the identical network architecture and similar supervised learning training procedure. The key difference between DAgger and BC is that DAgger query expert to label action for its trajectories whereas BC use expert trajectories for training only.

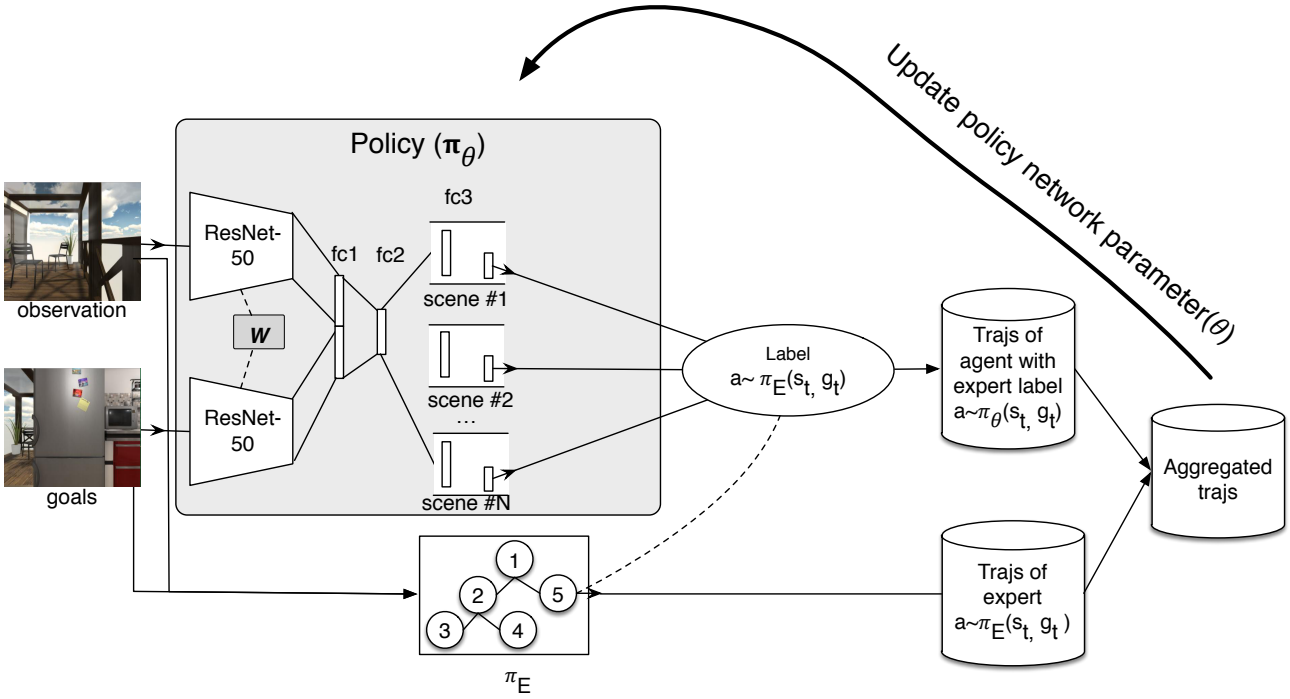

Figure 4. Network architecture for DAgger. The policy network is identical to GAIL.

Algorithm 3 DAgger Algorithm for target-driven visual navigation

1: Sample T step trajectories using π_E^l for each task l.
2: Get dataset $D^l = (s^l, g^l, \pi_E^l)$
3: **for** $t = 1 \rightarrow N$ **do**
4: Train π_θ^l on D^l
5: Sample T step trajectories using π_θ^l
6: Get dataset $D_t^l = (s^l, g^l, a^l \sim \pi_E^l(s^l, g^l))$
7: Aggregate datasets: $D^l \leftarrow D^l \cup D_t^l$
8: **end for**

4.3.3 Label smoothing

The loss function of the policy network is the standard cross entropy loss for softmax function.

$$L(\theta) = \sum_{\tau \sim D} \sum_{a} \pi_E(a|s,g) log(\pi_\theta(a|s,g)]$$

We use label smoothing on expert policy which is defined as

$$\pi_E(a|s,g) = (1-\epsilon)\mathbf{1}\{a = a^*\} + \epsilon/K$$

where $K = 4$, a^* gives the shortest path between current location s and the task goal g.

5. Experiments

5.1. Setup

We develop our modes in tensorflow [1] and the codes are based on (Zhu *et al*. 2017 [34]), referencing codes from [31], [22] and [13]. We perform training on a Nvidia Pascal TITAN X GPU, following training algorithms described in section 4. And we train 20 threads in parallel with each thread handling one unique task, except for GAIL. We compare results from the following models

1. **Shortest Path** represents the trajectories from expert and provides the optimal results. This is directly computed from environment's transition table which is not part of the inputs to any other models.

2. **Target-driven** is the model we base on. We take the codes from [34] and use default settings to train 20 targets.

3. **Behavior Cloning** Each update in BC is computed on 10 Trajectory. The learning rate is $2.7e-3$.

4. **DAgger** [27] adds more training data from agent's trajectories. The learning rate is $1.0e-4$ and we use 10 trajectories per batch.

5. **GAIL** [14] turns out to be very difficult to train. Even with all the tricks and optimization improvements we

Scenes	bathroom_02	bedroom_04	living_room_08	kitchen_02
# of locations	180	408	676	468

Table 1. Number of locations in each scenes

Method	bathroom_02	bedroom_04	living_room_08	kitchen_02
Shortest Path	6.51	12.38	12.08	14.45
Target-driven	7.53	14.03	15.36	21.41
DAgger(20)	6.96	12.92	13.40	18.28
GAIL(1)	N/A	N/A	N/A	16.19
BC (20)	6.98	13.67	14.32	17.27

Table 2. Training results with trained targets and random staring point

Method	bathroom_02	bedroom_04	living_room_08	kitchen_02
Target-driven	808	980	779	941
DAgger (20)	980	958	966	981
BC (20)	960	958	904	957

Table 3. Test results with random target and random starting points.

described in section 4.2, we cannot get the training converged nicely when we train all tasks concurrently within the project timeline. Therefore the results are reported after training for one task in scene kitchen_02. We use BC to bootstrap the training, $i.e.$ we use BC to train the policy network for 8000 images initially and then continue with GAIL training. The maximum KL divergence is set to $1e-3$, learning rate for value network $2e-3$, learning rate for discriminator $1e-6$, and we use 100 trajectories per batch and default values for other parameters.

We report the average steps ($i.e.$ average trajectory length) per episode per scene it takes for an agent to go from a starting point, which is randomly sampled, to a task goal. For each episode, if agent cannot reach the goal location after 1000 steps, we terminate the episode and record 1000 as the episode length. Each result is averaged over 100 episodes.

5.2. Results

First Table 1 gives an idea of the number of locations per scenes. We present the training results for all training targets in Table 2. These are training results since the same task goals are used during training. Then we randomly choose task goal to evaluate the generalization ability of the trained network and the results are show in Table 3.

It is clear that there is significant overfitting (or generalization issues). The training logs in Fig 5 indicates learning converges and but it is probably converging to bad minimum. The training results of imitation methods are better than those of RL methods while the results for random targets are worse.

We then investigate how the distance between training goal and test goal affects the performance. In order to be comparable, we follow the same methods as in [35] to evaluate new targets that are at fixed distance from the nearest trained targets (specifically 1, 2, and 4). We use *success rate*, which is defined as the percentage of trajectories that are shorter than 500 steps), as measurement metric. As shown in Table 4, the success rate drops as the task goal moves away from the trained target. This also points to generalization related issue.

Method	one step	2 steps	4 steps
DAgger (20)	0.28	0.26	0.20
BC (20)	0.42	0.34	0.28

Table 4. Success rate, i.e. percentage of episode shorter than 500 steps. The higher, the better. Evaluate on targets that are one, two and four steps away from trained targets

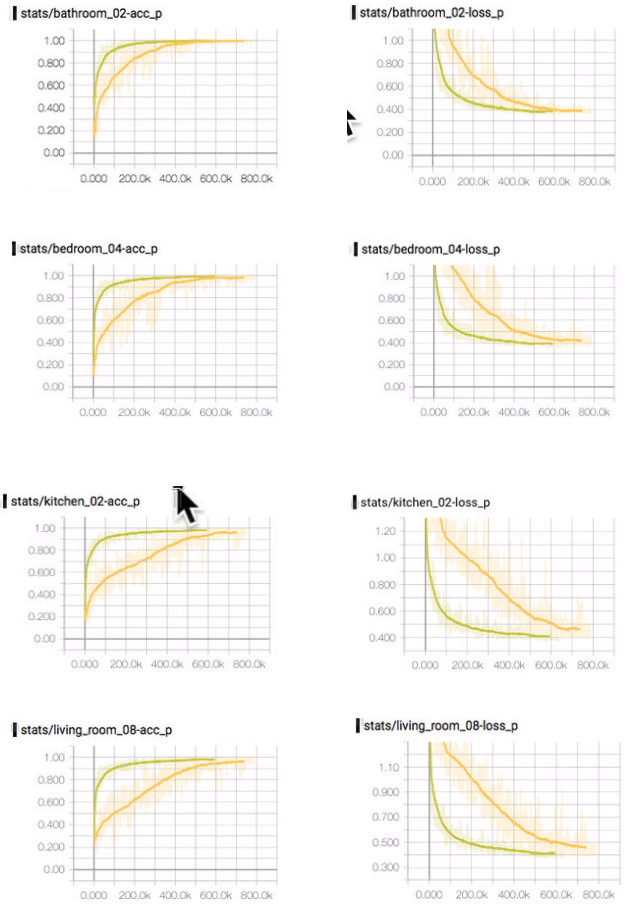

Figure 5. Training loss and accuracy for BC and DAgger. In all diagrams, yellow represents DAgger and green for BC. For both, training converges within 600 thousands steps.

5.3. Discussions

1. All imitation learning methods converges and give better training results than Target-driven methods. This is expected as supervised learning gives much stronger backpropagation signals.

2. GAIL, or likely any GAN based methods, appears to be very hard to train. Especially in our case, there are multiple tasks/scenes, each may drive optimization to different directions. Not to mention that the discriminator is designed to work against the generative model.

3. We have seen overfitting for our imitation learning implementation where it doesn't generalize very well when the goal is further away from the training targets. DAgger appears to suffer overfitting more than BC.

4. We notice an issue in early development where the trained policy gives prediction in extreme confidence, *e.g*. assigning almost 100% to one action and zero to others. This is problematic if the predicted action is wrong. As a result the agent will be stuck for the end of episode. We use smooth labeling in DAgger and BC training.

6. Conclusion/Future Work

In this project, we evaluate imitation methods on the visual navigation tasks with THOR framework. We show that deep neural network based imitation methods can be used with high dimension visual inputs. However we run into generalization issues with our implementation. The other challenge is training GAIL in concurrent tasks setting. For both we would like to investigate further in the future.

7. Acknowledgments

We would like to thank Yuke Zhu for his great advices on the project and providing the THOR environment to enable this project.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] S. Alatartsev, S. Stellmacher, and F. Ortmeier. Robotic task sequencing problem: A survey. *Journal of Intelligent & Robotic Systems*, 80(2):279, 2015.

[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[4] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[6] F. Bonin-Font, A. Ortiz, and G. Oliver. Visual navigation for mobile robots: A survey. *Journal of intelligent and robotic systems*, 53(3):263, 2008.

[7] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on systems, man, and cybernetics*, 19(5):1179–1187, 1989.

[8] F. Dayoub, T. Morris, B. Upcroft, and P. Corke. Vision-only autonomous navigation using topometric maps. In *Intelligent robots and systems (IROS), 2013 IEEE/RSJ international conference on*, pages 1923–1929. IEEE, 2013.

[9] A. De, K. S. Bayer, and D. E. Koditschek. Active sensing for dynamic, non-holonomic, robust visual servoing. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6192–6198. IEEE, 2014.

[10] Y. Duan, M. Andrychowicz, B. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba. One-shot imitation learning. *arXiv preprint arXiv:1703.07326*, 2017.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[13] J. Ho and S. Ermon. https://github.com/openai/imitation.

[14] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[15] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):21, 2017.

[16] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed ¡today¿].

[17] D. Kim and R. Nevatia. Symbolic navigation with a generic map. *Autonomous Robots*, 6(1):69–88, 1999.

[18] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[19] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

[20] S. Levine and V. Koltun. Guided policy search. In *ICML (3)*, pages 1–9, 2013.

[21] Y. Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[22] Y. Li, J. Song, and S. Ermon. https://github.com/YunzhuLi/InfoGAIL.

[23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[25] A. Y. Ng, S. J. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670, 2000.

[26] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *AISTATS*, volume 3, pages 3–5, 2010.

[27] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6, 2011.

[28] P. Saeedi, P. D. Lawrence, and D. G. Lowe. Vision-based 3-d trajectory tracking for unknown environments. *IEEE transactions on robotics*, 22(1):119–136, 2006.

[29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

[30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[31] Stanford CS231N staff. http://cs231n.github.io/.

[32] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[33] D. Wooden. A guide to vision-based map building. *IEEE Robotics & Automation Magazine*, 13(2):94–98, 2006.

[34] Y. Zhu. https://github.com/caomw/icra2017-visual-navigation.

[35] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *arXiv preprint arXiv:1609.05143*, 2016.