Imitating Shortest Paths for Visual Navigation with Trajectory-aware Deep Reinforcement Learning

Long-Huei Chen*[1], Mohana Prasad Sathya Moorthy*[1], Pratyaksh Sharma*[1], and Prasad Kawthekar*†[1]

[1]Department of Computer Science, Stanford University

Abstract

Target-driven Visual Navigation is an important problem in robotics that presents challenges in both Computer Vision and Reinforcement Learning. To tackle this problem, we propose extensions to the deep-siamese actor-critic (DSAC) model proposed in [1] that achieve state-of-the-art performance on the CVPR THOR datatset. The main contribution of this paper is a trajectory-aware variant of DSAC model achieving shorter path lengths to trained targets. We report further extensions that utilize imitation learning and symmetry principles to achieve 50x faster training to convergence using supervision, and greater generalizability to untrained targets.

1. Introduction

1.1. Motivation

A key task necessary for the advancement of robotics is the ability to interact with physical objects and generate suitable actions based on the surrounding environment. It is infeasible for traditional non-machine learning algorithms to scale and exhaustively capture all the possible interactions and cases. A Reinforcement Learning framework is especially suitable for this task—which enables agents to navigate and interact with the environment and thereby explore the relationships between actions and environmental changes. Such models suffer from data inefficiency and require several episodes of trial and error to converge. Most of these episodes initially are going to be bad and could be costly to be performed in a physical environment. Such a method to enable a robot to learn and correct its actions is impractical in real-world scenario.

Consider the example of a robot navigating and interacting with objects in a indoor home environment. A task that we want the robot to perform might be to bring a knife from the kitchen. Training this bot in a reinforcement setting will lead to several unsuccessful attempts before it learns. These attempts can be costly, way too many to be done in practice and more importantly unsafe to be performed in real-world. [1] provides a novel solution to this problem by using a simulated virtual environment with high quality 3D scenes and a physics engine to model real word interactions. RL agents can be trained in this virtual environment before taken to the physical world.

The paper [1] introduced simulation framework called The House Of inteRactions (AI2-THOR). It enables us to collect large number of visual observations for different actions. For example, an agent can freely navigate (i.e.move and rotate) in various realistic indoor scenes, and is able to have low- and high-level interactions with the objects (e.g., applying a force or opening/closing a microwave). The paper proposes a target-aware model, which takes visual task objective as input and learns a policy that embeds the target goal and just the current state. Essentially, an agent learns to take its next action conditioned on its current state and target. We measure the performance based on the number of steps taken to reach the target.

1.2. Problem Statement

We build on the CVPR-THOR challenge, where the task is to navigate towards a set of objects in indoor scenes based only on visual input. It utilizes the AI2-THOR simulator [1] which enables navigation in a set of near-photo-realistic indoor scenes (such as those presented in Fig. 1), and also provides an interface to interact with the physical objects (obeying the laws of physics) in the environment. A basic model of our project has as input an image of the current scene at a particular location in the environment as produced by the simulator, along with a static image of the target. The model produces a series of steps (turn left, turn right, move forward, move backward) which in turn navigate the agent to the specified target.

In our work, we explore three extensions to the model proposed in [1] and evaluate its performance. First, We

*Equal contribution.
†Not enrolled in CS 231N.

Figure 1: Example input to the indoor scenes navigation model. The target image is given as a static snapshot taken by the robot's camera at the desired location, and the observation image is defined as the present view of the imaginary robot at the current location in the indoors scene.

question this assumption of the action being just a function of the current state and the target. A key intuition that we relied on is that different training episodes share information. Also the action you need to take at a particular current scene depends not just on the current state but on the path you have traversed. We incorporate Recurrant Neural Networks with LSTM cells to perform a long-term path aware target driven navigation to test our hypothesis.

Secondly, we found that the model proposed in [1] takes a lot of time to train. We propose a imitation learning based approach to incorporate prior domain knowledge into the training phase. During the training phase, we assume a map of the environment is available and give the agent access to information about shortest paths to some targets.

Lastly, The model proposed in [1] performs poorly on targets not seen during the training phase. We found that our imitation learning model using Dagger also failed to generalize well. To make the model generalize better to targets not seen during training, we propose a simple yet powerful extention to the model that significantly increase the model capability to extend to new targets: symmetry-design in scene-nonspecific shared-weights Siamese layer in DAgger. We exploit the fact that taking one step forward is reverse of taking one step backward and design a symmetric model where we randomly swap the current position scene and the target scene (and the actions in action space) with a probability of 1/2.

2. Related Work

2.1. Visual Navigation

Visual navigation is an important task for robotics and has been extensively studied in the literature. Research in this area can be split into 3 main categories:

1. **Map based navigation**: This approach requires a map of the entire (or partial) arena and there are several work to find the best decision/action at each position. This area was extensively researched in the early 1990s. [2] prescribes repeated sequence of perception and navigation where the robot collects data from its sensors, builds a local map and locates itself in the global map using fuzzy logic. During the navigation phase a planner based on A* algorithm is put in place. [3, 4] are two other approaches for real time obstacle avoidance.

2. **Map building** based navigation: In this approach the system explores the environment first and builds its own map. Navigation process usses this stored representation. [5] were one of the first to start this effort. This approach was popular in the 2000s. [6, 7, 8] all propose vision based approach to global map building based on input from camera fixed on a navigating robot.

3. **Mapless Navigation**: In this approach, the system doesn't build a map at all. They do not need knowledge of the environment and can navigate based on the current observed environment. Most of the methods involve some form of obstacle avoidance given the input image. [9, 10] are two such papers.

2.2. Role of Memory

In neural networks, memory models remember the past and takes decisions based on history. LSTM is a popular example. Memory has been shown to add performance in deep RL agent's tasks such as in this paper [11]. In [12] propose new memory based deep reinforcement learning architectures for Minecraft and discuss how it can tackle the problems of current RL systems like partial observability, delayed rewards, high dimensional visual observation, etc better. They are show that their model generalizes well for unseen environments better.

[13] replaces the first post-convolutional fully connected layer with a recurrent LSTM to build a Deep Recurrent Q-Network (DRQN). They show that even though recurrency does not provide a great advantage when learning to play the game, it can adapt better at evaluation time if the quality of observations changes. Also they show that for the same length of history a DRQN performs better than stacking history length of frames at a DQN's input layer.

2.3. DAgger: Imitation Learning

The goal of imitation learning for sequential prediction problems is to train an agent to mimic expert behavior for some task. However, naively training an agent on a dataset generated by expert's policy violates the usual i.i.d assumptions made in statistical learning. Intuitively, since the agent only sees state distributions that correspond to an expert policy it is unable to learn error-correcting behaviour when it encounters new states during evaluation, which causes compounding errors. This leads to breakdown of theoretical guarantees for imitation learning and often also leads to unstable performance in practice.

[14] proposed DAgger algorithm (Dataset Aggregation) to overcome this limitation. DAgger works by iteratively sampling states using the current learnt policy (or a weighted mixture of the current policy and the expert policy), observing (requesting) the expert's policy for the new states, and updating the current policy based on these observations. [14] provide theoretical guarantees for the quality of the policy learnt using DAgger, and also present impressive gains over other imitation learning algorithms. These performance improvements have been replicated in several other application domains for imitation learning.

Most relevant to our problem setting is the application of DAgger in [15]. [15] train an end-to-end learning mapper and planner for visual navigation, and as alternative architecture to [1]. [15] use DAgger algorithm with scheduled sampling and sampling probability annealed with inverse sigmoid decay to train their model with shortest paths precomputed for the navigation tasks.

3. Model Architecture

We first describe the target-driven policy-learning model proposed by Zhu et. al. [1], which is the state-of-the-art and achieved the best results among other modeling approaches such as A3C or single-step Q learning. We build on the existing architecture [1] by adding scene-specific LSTM cells after the last hidden layer [2] and by adopting a imitation-learning protocol, which describe in later sections.

3.1. Target-driven Asynchronous Advantageous Actor-Critic Model for Visual Navigation

1. *Observations and goals*: Both the current observation of the environment and the target are simulated images that are taken by the agents RGB camera in the first-person view.

2. *Action space*: To simplify our simulation of the robotic agent's exploration of the environment, we learn from a high level of abstraction and presume that the agent can select from these four actions: moving forward, moving backward, turning left, and turning right.

3. *Rewards*: Our goal is to minimize the steps of the trajectory to the targets. Therefore we add a goal-completion award of $+10.0$, along with a immediate reward of -0.01 as a small time penalty for each step taken.

[1]Baseline implementation by Yuke Zhu (`https://github.com/yukezhu`); currently unavailable online.

[2]Rewritten based on Miyosuda's TensorFlow implementation at `https://github.com/miyosuda/async_deep_reinforce`

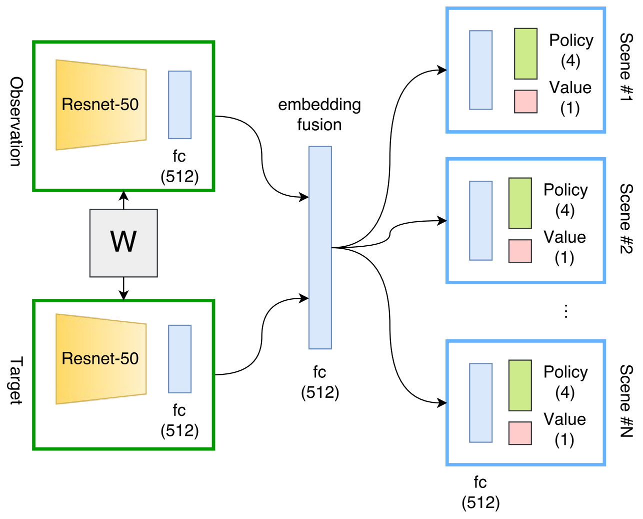


Figure 2: Deep siamese actor-critic model. The input are images of the current observation of the agent and the target, and parameters in the green squares are shared. The ResNet-50 layers are pre-trained on ImageNet and fixed during training.

3.2. Input Image and Siamese Layer

The input to the network are two images of the agent's first-person vision of the current observation in the environment and the target (See Fig. 2). We apply weight-sharing deep Siamese networks to both the observation and the goal, which allows discriminative embeddings to be learned and the geometric relationships between the images to be preserved as they are projected into the same embedding space. The two embeddings are then fused to form a joint representation representing the relationship between the environment and the target in the agent's quest to reach the target.

3.3. Scene-specific Layers

The fused representation is then passed along scene-specific layers. The reason we include scene-specific layers is to capture the spatial characteristics of a scene that are core to the success of the navigation. Similarly to the advantage actor-critic models, policy and value function outputs are generated from the scene specific layers to represent the current status of the agent.

3.4. Asynchronous Training Protocol

Traditional deep reinforcement-learning training protocols often employ replay memory and dual target network. Asynchronous methods, alternatively, explore the environment in parallel by spinning multiple threads local networks, accumulating gradients from the subnetworks, which are then back-propagated to the global network [11]. The final global network can then be tested after it's sufficiently trained based on updates from the local networks.

4. Trajectory-aware Navigation with Memory

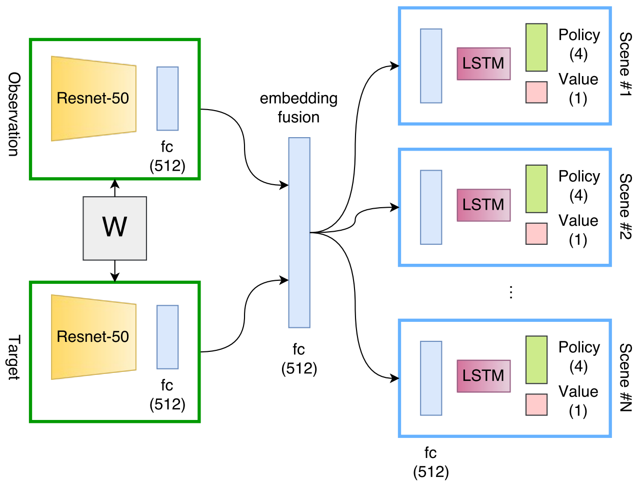


Figure 3: Deep siamese actor-critic model with memory cells extension. LSTM cells are inserted in the scene-specific layers.

4.1. Motivation

Memory models are a type of neural network models that can remember the past experience and use history to do decision making. One of the most popular memory models is the Long Short-Term Memory (LSTM) which has enjoyed success in tasks such as natural language text compression, unsegmented connected handwriting recognition, and automatic speech recognition—all of which are problems that benefit from relaxing the Markovian assumption.

As described in previous sections the deep-siamese actor-critic model [1] uses a feedforward network without memory. The implicit assumption here is that the agent's actions can be completely determined based on the current state (last four frames observed). A natural extension of this model is to relax the above *Markovian* assumption: allowing the agent's actions to be determined from the sequence of previous steps taken. Adding memory to RL agents has been shown to improve their performances in many tasks: [11] compares the performance of LSTM extensions of A3C protocol versus an A3C-feedforward baseline on 57 Atari games.

4.2. Approach

In this section, we propose extensions to the deep-siamese actor-critic model making it trajectory aware. In the first of such extensions, we add a LSTM cell right after the embedding fusion layer (see Figure 6). Our trajectory-aware memory model extension include a recurrent neural network with LSTM cells for each scene-specific layers, which see that the recent path taken by the agent is preserved as an internal state in the memory cell.

Specifically, during forward propagation we pass the scene-specific fully connected layer through a LSTM cell of the recurrent network. The hidden state of each LSTM cell is saved from step to step as a preserved representation of recent path taken [12]. During back propagation, however, the LSTM cell is unrolled for the recent history taken by agent in the local network, including all after the last time the parameters are synchronized from the global to the local network in our asynchronous training protocol. This way all exploration done by the local network starting from the initial weights from the global network can be accumulated in a single update.

For the TensorFlow implementation, we use a dynamic RNN (using a LSTM cell) unrolled over 5 time steps. While in general RNN models are harder to train than their feed-forward counterparts, we find this particular architecture especially unstable during training (see analysis in the Experiments below).

We also tried other places to add LSTM cells, many of the attempts didn't give good performance. We experimented with adding a recurrent neural network layer at the fused representation stage. Remember, a deep Siamese network with shared weights gives embedding for the goal and the observation. We fuse these two representations to get a fused representation. We made this layer a recurrent one with LSTM cells. We believed adding memory here can capture the necessary history information and also doesn't disrupt the symmetry present in the network (so that the shared weights of Siamese network are learned better). Our experiments showed that this specific model did not converge and even if it did for some scenes, it performed badly.

4.3. Implementation Details

In our TensorFlow implementation of the trajectory-aware model, we use a hidden state size of 512 for the LSTM cell. Each scene-specific layer has its own LSTM cell and correspondingly an un-shared hidden state. The cell is unrolled for 5 time-steps while training.

While in general RNN models are harder to train than their feed-forward counterparts (issues with vanishing and exploding gradients), training such models is especially difficult for RL problems. We train our model separately for 4 scenes (bathroom_02, bedroom_04, kitchen_02, living_room_08) each on 5 targets for 4 million time steps. We use an RMSProp optimizer with $\alpha = 0.99$, $\epsilon = 0.1$ and vary the learning rate log-uniformly within $[10^{-4}, 10^{-2}]$. Based on our monitoring of 1) average episode length, 2) max Q value per episode, and 3) per episode reward during train-

Model \ Scene	bathroom_02	bedroom_04	kitchen_02	living_room_08
Baseline [1]	7.46	14.46	**21.54**	15.52
Scene-specific LSTM	**7.24**	**13.46**	32.65	**14.81**

Table 1: Comparison of our proposed trajectory-aware model with the baseline on average path length (across 100 episodes) for 5 targets each across 4 scenes.

ing, we note that training converges for all 4 scenes within 4 million time steps.

4.4. Evaluation

4.4.1 Single Scene Training

For evaluation, we report the average path (across 100 episodes) for 5 targets each across the 4 scenes trained. The results in Table 1 support our hypothesis that allowing the agent to take trajectory-aware decisions can lead to improvements. For the scenes bathroom_02, bedroom_04, and living_room_08, the trajectory-aware model with scene-specific LSTM performs marginally better than the feed-forward baseline. On the other hand, for kitchen_02 (which is arguably a harder scene to navigate), the trajectory-aware model performs significantly worse than the baseline. We windup this part of our experiment with the conclusion that while simple trajectory-aware models are able to outperform the baseline, further experimentation needs to be carried out to obtain significant improvements.

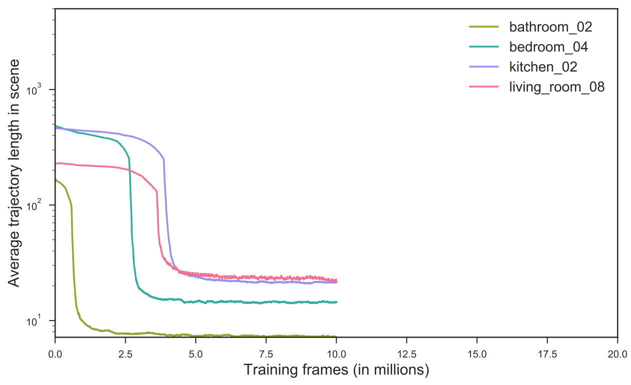
4.4.2 Cross-Scene Training

In addition to testing by training the model on each scene, separately, we also engaged in the original experiment set forth in the baseline implementation, namely training all 4 scenes with 5 targets each in an asynchronous manner. To compare the performance of such models, we look at the resulting time until convergence in the two models during training.

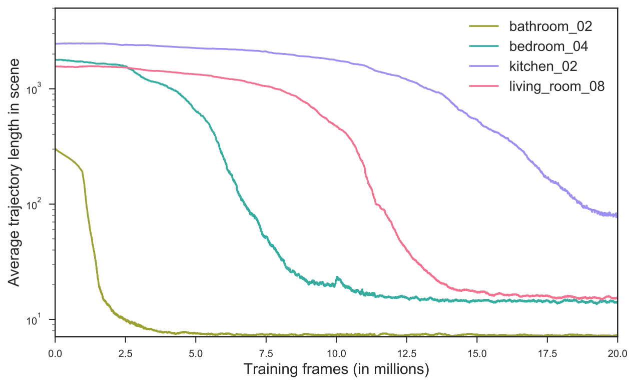
As evident from Fig. 4, when the LSTM extension is added the model takes around 2x - 4x more time until convergence. This is despite marginal or no better resulting shortest path from the LSTM model. We therefore conclude that the existing time history input in the FF model is sufficient to enable the agent to learn path based on recent trajectory, and an memory extension mostly hurts performance without adding much benefit.

4.4.3 Agent Action in Trajectories

To understand qualitatively how the different design affects the agent's actual navigational performance in the scene, we trained the baseline FF and the A3C-LSTM model on 5 targets in the largest scene 'living_room_08', and test them on



(a) Target-driven A3C-FF



(b) Target-driven A3C-LSTM with memory extension

Figure 4: Convergence of trajectories length during training time. With the LSTM extension the model requires 2x - 4x more training time before convergence, and the resulting minimum path length are only marginally smaller.

one of the trained targets. The resulting trajectory length is reported in Tab. 2.

By watching videos of the actual trajectories produced by these 2 agents (Supplementary Material V1), we observe that the baseline agent, without the LSTM extension, makes a few more mistaken steps turning around in a corner and colliding to the wall beside (Fig. 5). This is in line with our original hypothesis that the LSTM extension would help in case that the agent is stuck at corners and need to rely on

Model	Trajectory length
Baseline [1]	11
Scene-specific LSTM	9

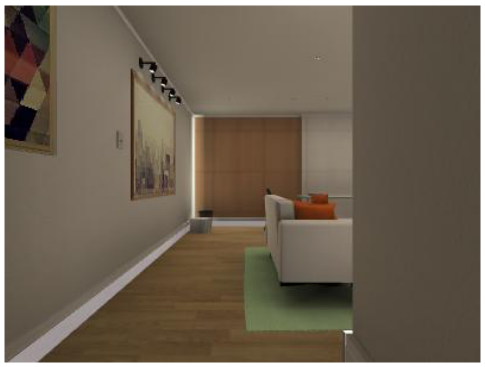
Table 2: In a single example trajectory, steps required for each agent to reach to the specified target. Videos of their respective paths is located in Supplementary Material V1.

previous trajectories to figure a way forward. Even though the resulting benefit is minimal, it remains true that our extension did help the model in the way we intended.

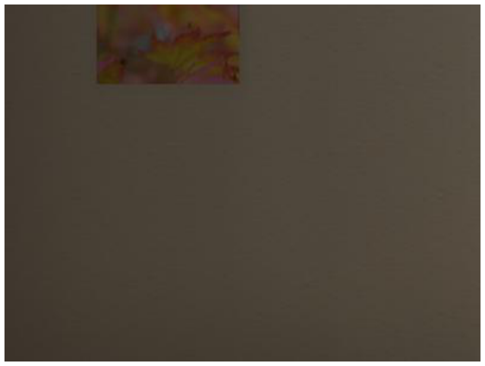
5. Shortest Path Imitation Learning

5.1. Imitation Learning with DAgger Algorithm

We investigate the use of prior domain knowledge to improve the agent's performance in path finding in terms of training time and length of paths recovered. Specifically, we



(a) Agent view at time step 04



(b) Agent view at time step 05

Figure 5: The type of scene view that the FF agent has most trouble navigating. With the LSTM extension the agent was able to avoid turning around in corner collision like this in its trajectory. See Supplementary Material V1 for full videos of the agents' action.

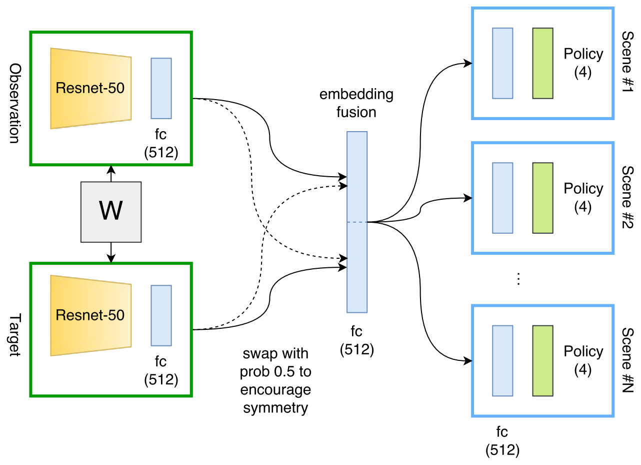


Figure 6: Deep siamese actor-critic model with symmetry. Current observation and target states are flipped with equal probability during training and evaluation to encourage generalization.

consider the case where the agent has indirect access to exact shortest paths to some targets in the environment. In this section we evaluate the agent's ability to learn/imitate/clone the paths to these targets.

We frame the problem of training the agent to navigate along shortest paths as interactive supervised learning. Specifically, the agent learns a model $a(s, t, e)$ to sample an action from a softmax probability distribution a when it is in state s while navigating to target t in environment e. The function a is approximated by the neural net architecture shown in Figure X. We use the DAgger imitation learning protocol [14] for training the model, with a's loss function being the multi-class cross-entropy w.r.t the action probabilities for navigating along the true shortest path from s to t in e.

5.1.1 Experiment Setup

The experiment consists of training the model to mimic shortest paths to 5 random targets from each of the 20 scenes (environment) of the THOR dataset. The agent is trained concurrently across 100 parallel threads, similar to the A3C training protocol [11], with each thread corresponding to one target-scene pair. Training comprises of 2 million steps across all threads. Each thread's training can be considered to be split in episodes, where each episode corresponds to a random start position and an episode ends either when the model finds a path from this start position to the target or is unable to do so in 5000 steps. After training finishes, the model is evaluated on 100 episodes (each corresponding to a random start position) for each of the 100 trained targets (5 per scene).

We compare the average lengths of paths found across all 100 episodes and for all trained targets with a similar experiment from [1], which train their model for 100 million steps. 3.

5.1.2 Evaluation: Imitating Shortest Paths

As expected, with expert spervision of true shortest paths the model is able to find shorter paths with signficantly lesser training (50x) than when it has to rely on reinforcement learning with rewards that approximate its goal.

5.2. Encouraging Symmetry in Imitation Learning

5.2.1 Motivation

During our initial testing, even though the vanilla DAgger agent [14] finds much shorter path extremely efficiently, we acknowledged that it still has trouble generalizing to new targets unseen during training time. We are therefore the first to propose a simple yet powerful modification to the model that significantly increase the model's capability to generalize to new targets: symmetry-design in scene-nonspecific shared-weights Siamese layer in DAgger.

To encourage symmetry between our two input channels and help to model generalize better to new testing targets, we design the model to randomly swap the two input (observation/target) with 0.5 probability. The two input are still projected to the same embedding space (since the input Siamese layer share weights), but the two input now have a change to be fused in an opposite order.

The reasoning behind the symmetry extension is as follows: in the shared-weight Siamese layer, both the current timeframe observation and the target image are cast towards the same embedding space. However in the fusion layer, half of the neurons are only ever exposed to the static target image embedding, and are therefore unlikely to perform well once we switch to a never-before-seen new target image during testing.

5.2.2 Evaluation: Generalization to New Targets

To test our how the new symmetrical DAgger training agent produces agents that can navigate to new targets, we setup experimentation comparing across model design, number of training targets and the distance of new test targets. For comparison across **model design**, we evaluate performance on the three models: Target-driven A3C Baseline [1], vanilla DAgger [14], and DAgger + symmetry design. We train each model with 4 different **number of training targets**: 1, 2, 4, 8, and evaluate all 4×3 training setups with 4 kinds of **new test targets**: locations that are 1, 2, 4, and 8 steps away from the trained targets.

We train and test all models on 'living_room_04', the 3rd largest scene out of 20 in the dataset. We measure agent performance by the **new target success rate**—the percentage of times the agent reach the new target under 500 steps—as the agent either succeeds or fails in its conquest to reach the new target, and during failure the agent often continues exploration indefinitely and skews the average path length.

5.2.3 Analysis

We analyze the agent performance across model design, number of trained targets and distance of test targets.

- *Model design*: It's clear both DAgger-based models perform significantly better than the baseline A3C. However, with the increase of both the number of trained targets, we see the the differences among the model seem to fall somewhat.

- *Number of trained targets*: while the asymmetrical baseline and DAgger seem to rise in accuracy as the number of trained targets increases, the symmetrically designed imitation learning agent seems to fall somewhat. However, for the DAgger + symmetry agent trained on 4 or 8 targets, the DAgger + symmetry model generalize very well as the difference among the test targets seems minimal regardless of distance.

- *Distance of testing targets*: The asymmetrical DAgger and A3C models seem to share similar overall trends as to their performance with respect to the test target distance. With the DAgger + symmetry agent, however, we observe that the accuracy is way higher when trained on 1 or 2 targets only and tested on a close new target.

5.2.4 Agent action in trajectories

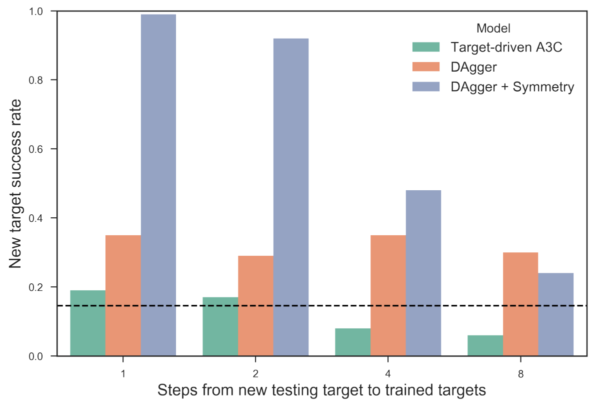
By watching videos of the actual trajectories produced by these 3 agents (refer to videos in Supplementary Material V2), we observe that the target-driven agent seems to make a bunch of missteps repeating the same state (like turn left, then right, then left, then right again). We hypothesize that compared to the DAgger imitation learning agent, in the target-driven model the expert policy is not provided during training, so similar locations or orientations at a location would be virtually equivalent in their value function and the critic is unlikely to help the agent learn a policy that leads to the target directly.

6. Conclusion

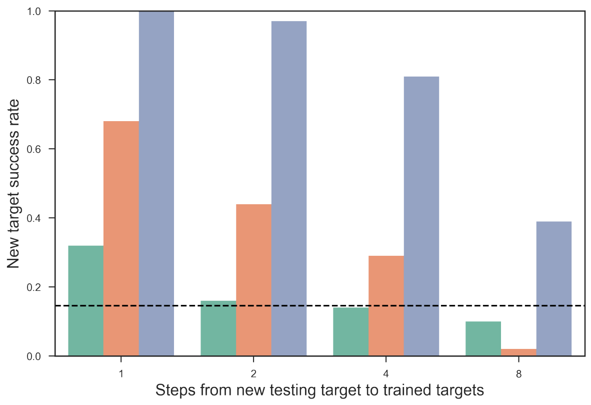
We propose and train trajectory-aware RL agents by incorporating LSTM layers in our model architecture and observe that this leads to marginal performance gains in terms of shorter paths found. The successes are not without shortcomings: performance gains are not consistent across trained scenes and such models take significantly longer

Method	**Average Path Length**
Random Walk	2744.3
Shortest Path	17.6
1-step Q	2539.2
A3C	723.5
Baseline [1]	210.7
Baseline [1] + DAgger [14]	**52.7** (Shortest Path=13.1)

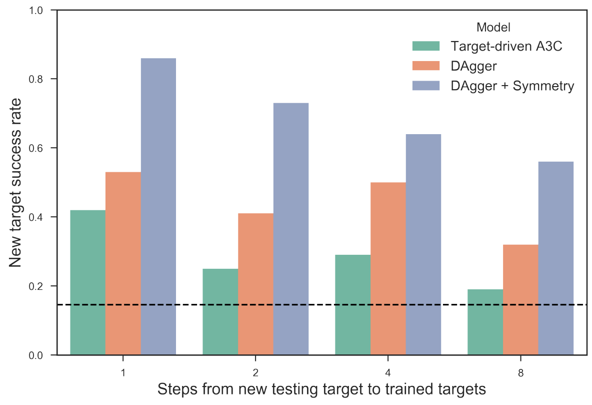
Table 3: Comparison of various methods on average path lengths found for visual navigation.



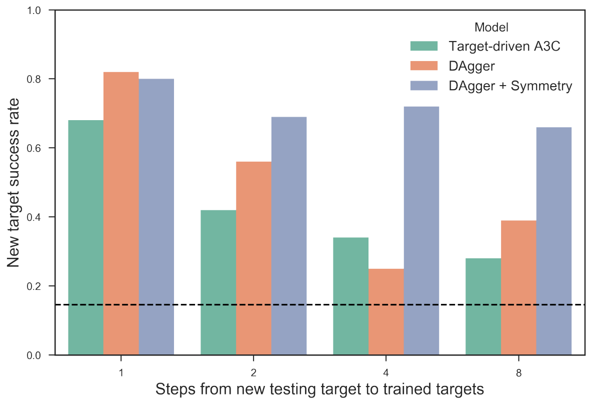
(a) Training with 1 target



(b) Training with 2 targets



(c) Training with 4 targets



(d) Training with 8 targets

Figure 7: Comparing agent performance across models, different number of training targets, and distance of test targets.

training to converge. Even so, our work illustrates the potential of memory-models for the target-driven visual navigation problem. For future work, we plan to experiment with other additions such as attention to try and improve performance.

We are the first to propose a symmetry designed extension to the target-driven model adopted for navigation task. The simple yet powerful extension, when combined with the imitation learning DAgger algorithm, allows the model to learn in 50x less training time a superior policy to the vanilla A3C baseline. The symmetry design also encourage the agent to generalize much better to new, unseen targets. In future work, we plan to experiment with noisier/inaccurate shortest path expert estimates for training the model (for ex- Euclidean shortest path directions) to test if the model can handle real-world scenarios.

References

[1] 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," in *IEEE International Conference on Robotics and Automation*, 2017.

[2] G. Oriolo, M. Vendittelli, and G. Ulivi, "On-line map building and navigation for autonomous mobile robots," in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, vol. 3. IEEE, 1995, pp. 2900–2906.

[3] D. Kim and R. Nevatia, "Symbolic navigation with a generic map," *Autonomous Robots*, vol. 6, no. 1, pp. 69–88, 1999.

[4] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on systems, man, and cybernetics*, vol. 19, no. 5, pp. 1179–1187, 1989.

[5] H. P. Moravec, "The stanford cart and the cmu rover," *Proceedings of the IEEE*, vol. 71, no. 7, pp. 872–884, 1983.

[6] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera." in *ICCV*, vol. 3, 2003, pp. 1403–1410.

[7] M. Tomono, "3-d object map building using dense object models with sift-based recognition features," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 1885–1890.

[8] D. Wooden, "A guide to vision-based map building," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 94–98, 2006.

[9] P. Saeedi, P. D. Lawrence, and D. G. Lowe, "Vision-based 3-d trajectory tracking for unknown environments," *IEEE transactions on robotics*, vol. 22, no. 1, pp. 119–136, 2006.

[10] A. Remazeilles, F. Chaumette, and P. Gros, "Robot motion control from a visual memory," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 5. IEEE, 2004, pp. 4695–4700.

[11] 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*, 2016, pp. 1928–1937.

[12] J. Oh, V. Chockalingam, S. Singh, and H. Lee, "Control of memory, active perception, and action in minecraft," *arXiv preprint arXiv:1605.09128*, 2016.

[13] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *arXiv preprint arXiv:1507.06527*, 2015.

[14] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning."

[15] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," *arXiv preprint arXiv:1702.03920*, 2017.