Deep Reinforcement Learning using Memory-based Approaches

Manish Pandey
Synopsys, Inc.
690 Middlefield Rd., Mountain View
mpandey2@stanford.edu

Dai Shen
Stanford University
450 Serra Mall, Stanford
dai2@stanford.edu

Apurva Pancholi
Omnisenz Inc
872 Bandol Way, San Ramon
apurva03@stanford.edu

Abstract

This paper focuses on the problem of navigation in a space using dynamic reinforcement learning. We build on the work by Zhu et.al. [1], and explore the performance of target-driven visual navigation with memory layers added to the network. We evaluate our models using simulated 3D indoor scenes rendered by Thor framework [1], and we show that in many cases, adding memory results in small improvements in episode path lengths for targets not trained on earlier. We use an actor-critic model with policy as the function of goal as well as current state to allows for better generalization.

1. Introduction

Reinforcement Learning (RL) enables machines and software agents to automatically determine their actions in the context of a specific environment. Agents observe environment, and compute a reward feedback (reinforcement signal) to learn behavior and take actions to maximize the reward. Applications of RL include video and board game playing [2], robotic obstacle avoidance [3], visual navigation [1], and driving. Combining deep learning with reinforcement learning, termed Deep Reinforcement Learning (DRL) is helping build systems that can at times outperform passive vision systems [6]. Recent work with deep neural networks to create agents, termed deep Q-networks [9], can learn successful policies from high-dimensional sensory inputs using end-to-end reinforcement learning.

This paper focuses on the problem of navigation in a space using DRL. The task of the agent is to navigate to a given visual target, using only visual input. This requires that the agent should learn the relationship between actions (movement in different directions), and the spatial view, and learn how to navigate towards the target. We build on the work by Zhu et.al. [1], that overcomes the limitations of traditional visual DRL agents that have the target embedded into the agent's model which requires re-training DRL agents for new model parameters to handle new target. In contrast, this paper uses the approach developed in [1], to create a target-driven model that learns a policy based on both the target and the current state. This makes it possible to avoid re-training the model for new targets.

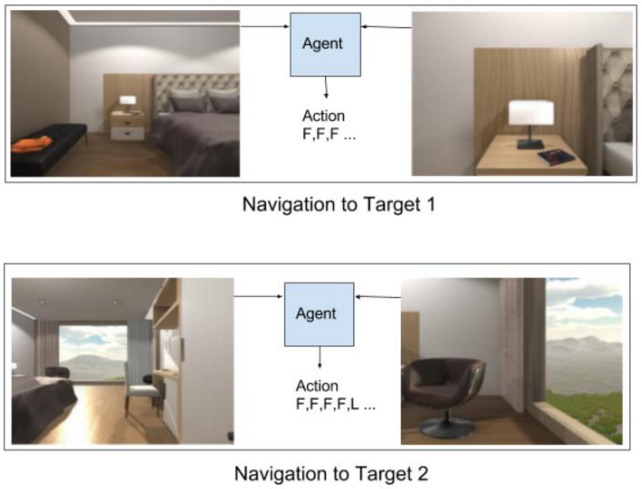


Figure 1: DL Agent with current observation and target makes navigational decisions to reach target.

For example, Figure 1 illustrates two navigation problems, where the agent takes the observation, on the left, and the image of target on the right, and determines next action to the taken. The problem in navigation to target 2 involves going to a target that is initially partially occluded, and requires navigating around an obstacle to reach the target chair (a series of move forwards 'F', followed by a left turn, 'L').

Generation of data for visual navigation can be tedious, requiring running systems and capturing images in physical space. However, we take advantage of the Thor simulation framework [1], that allows agents to navigate in a virtual space. The images in Figure 1 were generated with Thor, and we employ this framework for visual navigation for our work.

While the DRL approach described in [1] shows better performance than many other target driven approach, such as One-step Q [9], it does not maintain a history that could potentially help it 'remember' past context to make future navigational decision. In this project, we explore various

memory-based architectures such as Memory Q-Networks (MQN) [8], with single-layer and multi-layer LSTMs to determine if adding this state to the DRL model yields navigational paths with shorter trajectory lengths.

2. Related Work

Visual navigation is an active research area with a number of approaches, that can be classified as map-based [13] or map-less approaches [1]. Map-based approaches require a prior map of the environment, or reconstruct the map on-demand. In contrast, map-less approaches do not use a prior map, and do not assume a set of landmarks in the navigational environment. The advantages of map-less approaches include the ability to dynamically handle new situations and changes to the navigational landscape.

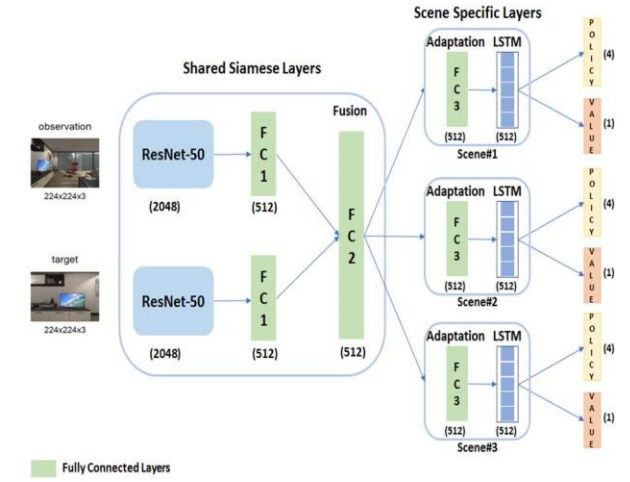
Reinforcement Learning (RL) has been applied to a variety of problems, such as robotic obstacle avoidance [2], and visual navigation [1]. Deep Reinforcement Learning (DRL), a combination of reinforcement learning with deep learning has shown unprecedented capabilities at solving tasks such as playing Atari games or the game of Go [2].

[8] has added 'context' to DRL by adding past "context" or history of observations to determine agent action with architectures such as Memory Q Network (MQN). We seek to use this idea of [8], to extend the target-driven visual navigation approach by Zhu et.al. [1], and investigate, how adding context can lead to better performance.

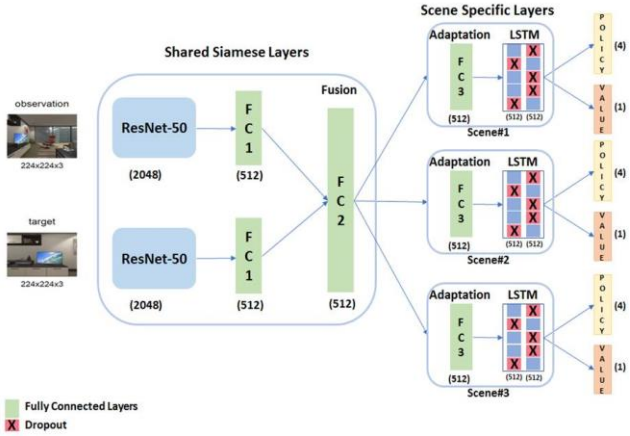
The idea of asynchronous reinforcement learning is particularly important to enable parallel training with multiple scenes to improve learning. Parallel weight updates to a global graph from multiple threads helps generalize training [15].

3. Methods

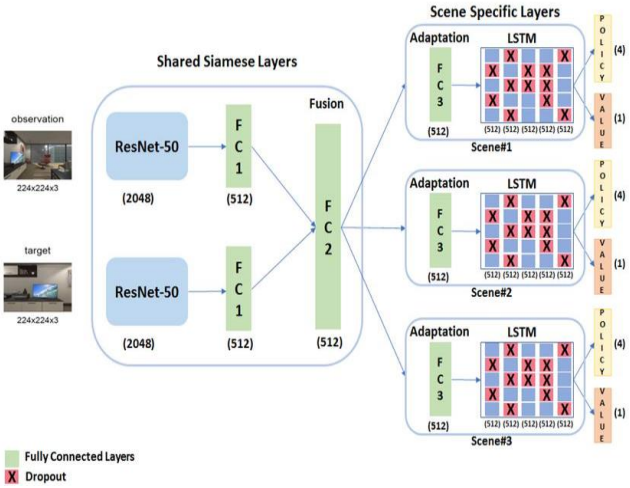
Though the Deep Reinforcement Learning yields proficient controllers for complex tasks, these controllers have limited memory and rely on being able to perceive the complete information (game screen, scenes, etc.) at each decision point. To address these shortcomings, in this paper, we introduce a new architecture for Target driven Deep reinforcement learning and investigate the effects of adding recurrency to a Deep Q-Network (DQN) by introducing recurrent LSTM layers. Our proposed architecture introduces the memory layer in the existing network architecture of deep Siamese actor-critic model proposed by [1] as is shown in Figure 2. Our proposed architecture is based on Memory Q-Network (MQN) which is a feedforward architecture that constructs the context based on only the current observation, which is very similar to MemNN except that the current input is used for memory retrieval in the temporal context of RL.



(a) Siamese actor-critic model with one LSTM layer



(b)Siamese actor-critic model with two LSTM layers, dropout



(c) Siamese actor-critic model with *n* LSTM layers, dropout

Figure 2: LSTM layer(s) added to Siamese actor-critic model in [1].

We begin by reviewing the network used in [1], which is essentially the same as Figure 2(a) except that the LSTM layer in the scene specific layer is absent. The inputs to the network are two images of the agent's current position and the target to reach. These two images are then processed by a ResNet-50 network and outputted as two 2048D vectors. These two vectors are then fed into two fully connected layers which output 2 512D vectors, which are then concatenated as one 1024D vector and fed into a second fully connected layer. The output is then a vector containing information of both the agent's current position as well as the target. This output vector is then fed into scene specific layer which consists of a third fully connected layers and two other fully connected layers for calculating policies and values.

The architecture adopts a model similar to A3C [14], where several worker networks are trained in parallel and are asynchronously synched with one global network for variable updates. The global network has exactly the same structure as the worker networks except that the global network has all the scene specific layer whereas each worker network only interacts with one scene and has only one scene specific layer. Making each worker interact with different scenes effectively separates the experience gained by each and creates a more diverse update of the network variables. This is claimed to stabilize the training process.

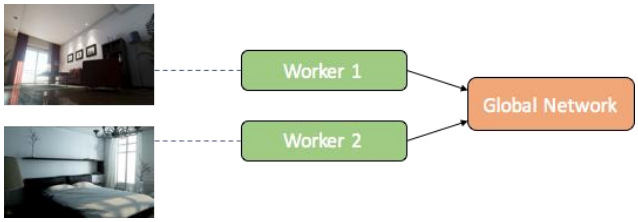


Figure 3: A3C global network and worker networks

Building on the existing architecture, we introduce memory into the network by adding LSTM layers into the scene specific layers as shown in Figure 2 (a, b, c). We also attempt the test the generalizability of our model by evaluating its performance on targets unseen during training. More specifically, we run the following experiments:

1. Train DRL model using existing architecture (Architecture without memory) on all scenes but only partial set of targets. Evaluate on the unseen targets.
2. Train and evaluate the DRL model using new architecture (with memory) on all scenes and targets.
3. Train DRL model using new architecture (with memory) on all scenes but only partial set of targets. Evaluate on the unseen targets.

In addition to the code we have developed, we have used code from https://github.com/yukezhu/icra2017-visual-navigation (non-public repository due to copyright), and https://github.com/miyosuda/async_deep_reinforce for our implementation.

4. Dataset and Features

Our training data consists of a set of simulated 3D indoor scenes rendered by the Thor framework [1]. Each of the scene consists of images created by artists to simulate the texture and lightings of the real environment. The scenes are of four common types in a household environment, namely kitchen, bathroom, bedroom, and living room. The use of the simulated scenes makes the training process much more affordable and easier to scale than training robots in real world. Figure 4 shows four example simulated scenes we captured from Thor.

Figure 4: Sample generated scene models from THOR.

For training, we use hdf5 dumps of the simulated scenes in [1, 16]. Each dump contains the agent's first-person observations sampled from a discrete grid in four directions. To be more specific, each dump stores the following information row by row:

1. observation: 300x400x3 RGB image (agent's first-person view)
2. resnet_feature: 2048-d ResNet-50 feature of the observations extracted using Keras
3. location: (x, y) coordinates of the sampled scene locations on a discrete grid with 0.5-meter offset
4. rotation: agent's rotation in one of the four cardinal directions, 0, 90, 180, and 270 degrees
5. graph: a state-action transition graph, where graph[i][j] is the location id of the destination by taking action j in location i, and -1 indicates

collision while the agent stays in the same place.

6. shortest_path_distance: a square matrix of shortest path distance (in number of steps) between pairwise locations, where -1 means two states are unreachable from each other.

5. Experiments and Results

We discuss below data from the following experiments and the results we obtained:

- Training of baseline network for multiple targets (section 5.1)
- Training of memory-enhanced network for multiple targets (section 5.2)
- Evaluation of target-driven navigation for multiple targets (with baseline and memory-enhanced) (section 5.3)
- Multi-layer LSTM architectures (section 5.4)

All experiments were performed on a Google Compute Instance with a n1-highmem-8 with 8 vCPUs, 52GB memory, and 1 Nvidia Tesla K-80 GPU.

5.1. Baseline Network

The first step in running the DRL system is to train it on a number of different scenes and targets. The DRL system implements an asynchronous actor-critic model, and this training proceeds in parallel with a total of 20 targets distributed equally across 4 scenes. An indication of the progress in training is the convergence of the episode path length, max Q value, and the rewards value, as shown in figures 5 through 7 below.

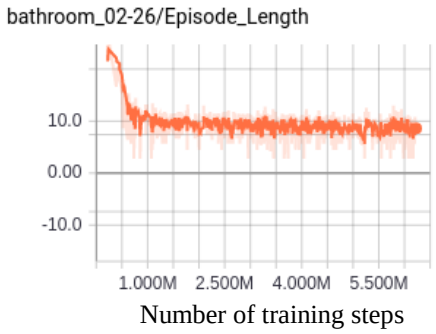


Figure 5: Episode length for baseline training for target 26. (Y-axis represents episode path length).

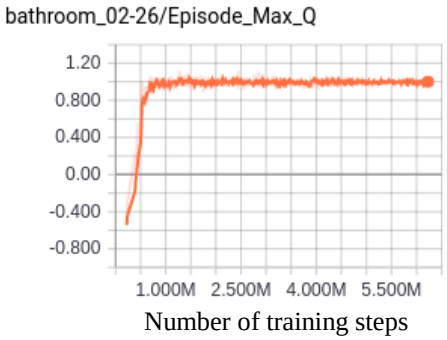


Figure 6: Max Q value during baseline training for target 26. (Y-axis represents Q value).

Figure 7: Episode reward while navigating to target 26. (Y-axis represents episode path length).

The convergence of path length to a small target value, approximately 10, in figure 5, and convergence of max-Q value to 1 is an indication of completion of training.

5.2. Memory-enhanced Network

Adding memory to the base system architecture significantly increases the number of steps needed during training. Table 1 shows that the increase in the number of training steps ranges between 89% – 174%, i.e., the number of steps can possibly increase by a factor of 3. This however is accompanied by a small decrease in episode path lengths between 2.6% to 11%. There are some cases where the path length may increase (e.g., target 43 in table).

Scene and Target	Episode Length	Steps (M)	Episode Length	Steps (M)	Episode Length Improvement	Steps Increase Percentage
	Baseline	Baseline	LSTM	LSTM	LSTM/Base	LSTM/Base
bathroom_02_26	9.10	0.96	8.20	2.63	11.0%	174%
bathroom_02_37	9.00	1.00	8.70	2.14	3.4%	114%
bathroom_02_43	8.40	1.34	8.70	2.61	-3.4%	94.8%
bathroom_02_53	8.5	1.15	8.1	2.18	4.9%	89.6%
bathroom_02_69	7.8	1.34	7.6	2.8	2.6%	109%

Table 1: Baseline vs LSTM Episode Lengths in Training.

5.3. Target-driven Navigation

Table 2 shows the results of evaluating the trained model for a subset of scenes and targets.

Scene	Target	Mean Episode Reward	Mean Episode Length	Mean Episode Collision
bathroom_02	26	9.93	7.76	0.04
	37	9.91	9.55	0.04
	43	9.93	6.95	0.11
	53	9.93	6.81	0.09
	69	9.93	7.06	0.08

Table 2: Baseline model evaluation results summary for a subset of targets.

This table is indicative of training quality, and suggests "over fitting." The true test of target driven navigation [1] is how well does the DRL architecture perform for navigating to targets that it has not been trained for, i.e., a "target-driven" evaluation.

We have validated this in a series of experiments where we train models with all targets but one, and then navigate to that model specifically. Our results for this are included in table 3.

For example, in the case of target #43, we train the network for 19 targets, excluding #43 for both the baseline and the memory enhanced models. Then, for each of these models, we evaluate the network for navigating specifically to the specified target. We repeat this experiment 100 times and report the average number of steps, reward and the number of collisions in the table. For target #43, with 100K training steps, the baseline model yields result with an average path length of 426.85 steps. The memory enhanced model has a shorter path length of 399.80 steps.

DRL Architecture	Target	Metric	Untrained	Target Driven (100K steps)	Target Driven (1M Steps)	Episode Length Improvement
Baseline	53	Episode Length	505.32	438.78	61.22	
		Reward	-0.02	0.06	8.95	
		Collision	55.35	61.81	5.00	
LSTM	53	Episode Length	452.48	371.13	121.52	-49.62%
		Reward	0.72	2.33	7.05	
		Collision	52.96	44.11	8.13	
Baseline	26	Episode Length	411.33	399.21	175.25	
		Reward	1.66	1.76	7.33	
		Collision	47.06	47.30	6.72	
LSTM	26	Episode Length	534.47	498.95	142.31	23.15%
		Reward	-0.69	0.39	6.52	
		Collision	59.5	51.51	9.05	
Baseline	37	Episode Length	728.31	615.91	6,186.81	
		Reward	-4.36	-3.05	-58.56	
		Collision	78.7	76.66	35.52	
LSTM	37	Episode Length	942.91	702.58	565.86	993.35 %
		Reward	-8.44	-6.01	-1.61	
		Collision	100.23	99.88	66.23	
Baseline	43	Episode Length	481.04	426.85	388.33	
		Reward	0.16	0.73	2.11	
		Collision	55.98	55.69	48.72	
LSTM	43	Episode Length	471.45	399.80	342.13	13.50%
		Reward	0.52	0.75	1.01	
		Collision	53.09	50.11	43.13	
Baseline	69	Episode Length	455.86	424.40	410.64	
		Reward	-1.32	-0.03	0.68	
		Collision	-71.33	64.38	58.03	
LSTM	69	Episode Length	440.86	386.47	390.72	5.10%
		Reward	0.72	0.99	2.21	
		Collision	54.27	57.27	31.35	

Table 3: Target driven navigation for baseline and memory-based models.

We measure the baseline and memory enhanced model performance for average episode length, reward and number of collisions for three cases: Untrained network, target-driven with 100K training steps, and 1 Million training steps. As we can see from the table, adding

memory in many cases helped improve the model quality with shorter evaluated paths and fewer collisions. In one case, for target #53, the path length does increase. This could be potentially due to limited number of training steps (1 Million). However, the huge amount of training needed, and available CPU/GPU time was a limiting factor for us. The Google Compute instance we used ran at the rate of 100 training steps per second. As such, the table 3 represents over 42 hours of training time (which we ran multiple times).

Excluding the outlier case of baseline target 37 result (which could be a victim of runaway gradient), the improvements in path lengths ranged from 5 – 23%, and in one case the path length got worse.

5.4. Multi-layer LSTM architectures

We have in addition run training on multi-layer LSTM implementations, described earlier in Section 3 (Figure 2(b), (c)). With two LSTM layers, using dropout, we observe faster training convergence, within a million time steps as shown in figure 8 below. This is in contrast to longer training episodes in training in the base memory layer. While we have not done target driven evaluations on this model, the episode length and reward values in training are indicative of good evaluation performance, without the disadvantage of long path lengths as in number of LSTM training steps in table 1.

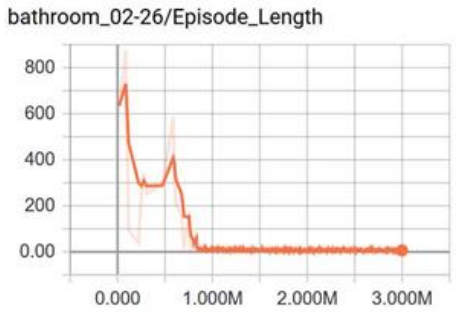


Figure 8: Multi-layer LSTM episode length (Y-axis is number of episode steps)

Figure 9: Multi-layer LSTM episode reward graph (Y-axis is episode reward)

6. Conclusion and Future Work

Our work indicates that adding memory "context" to the model helps improve the performance of target-driven visual navigation. We have validated this through multiple runs of independent targets, and have succeeded in improving upon baseline results in several cases. However, this comes at the cost of longer training episodes (up to 3X longer). Also, in some cases, the episode length may actually increase. Multi-layer LSTM-based A3C architectures seem to require fewer training cycles to converge, but require further investigation.

The experiments reported in table 3 have been run with models trained up to only a million cycles due to computational resource constraints mentioned in section 5.3. An obvious extension of this would be to train for additional 10-20 million cycles and study target-driven performance.

In addition, [8] has a number of variations on retaining a recent history of observations and context vector (for memory retrieval and action-value estimation) such as Memory Q-Network (MQN), Recurrent Memory Q-Network (RMQN), and Feedback Recurrent Memory Q-Network (FRMQN). These architectures allow the network to refine its context based on previous retrieved memory so that it can do more complex reasoning with time.

7. Honor code related information

The work in this project uses code from https://github.com/yukezhu/icra2017-visual-navigation and https://github.com/miyosuda/async_deep_reinforce . The first repository has the baseline A3C model which we have

improved upon. The first, repository, however, is not public as the Thor database is proprietary, and it has been made accessible to team members for the purpose of the CS231n project.

Acknowledgement

The team would like to acknowledge Yuke Zhu for making accessible Thor database, the visual navigation codebase, and answering numerous questions.

References

[1] Zhu, Yuke, et al. "Target-driven visual navigation in indoor scenes using deep reinforcement learning." arXiv preprint arXiv:1609.05143 (2016).
[2] Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).
[3] Kober, Jens, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey." The International Journal of Robotics Research 32.11 (2013): 1238-1274.
[4] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." Nature 529.7587 (2016): 484-489.
[5] Ba, J., Mnih, V. & Kavukcuoglu, K. Multiple object recognition with visual attention. In Proc. International Conference on Learning Representations http://arxiv.org/abs/1412.7755 (2014).
[6] Mnih, V. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015).
[7] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International Conference on Machine Learning. 2016.
[8] Oh, Junhyuk, et al. "Control of memory, active perception, and action in minecraft." arXiv preprint arXiv:1605.09128 (2016).
[9] Hausknecht, Matthew, and Peter Stone. "Deep recurrent q-learning for partially observable mdps." arXiv preprint arXiv:1507.06527 (2015).
[10] Konda, Vijay R., and John N. Tsitsiklis. "Actor-Critic Algorithms." NIPS. Vol. 13. 1999.
[11] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." AAAI. 2016.
[12] Duan, Yan, et al. "Benchmarking deep reinforcement learning for continuous control." Proceedings of the 33rd International Conference on Machine Learning (ICML). 2016.
[13] J. Borenstein, and Y. Koren, "Real time obstacle avoidance for fast mobile robots," IEEE Trans on Cybernetics, 1991.
[14] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." International Conference on Machine Learning. 2016.
[15] Juliani, A, Simple Reinforcement Learning with Tensorflow: Asynchronous Actor-Critic Agents (A3C), https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2
[16] Zhu, Yuke: ICRA 2017 paper code repository https://github.com/yukezhu/icra2017-visual-navigation