Navigate, Fetch and Find - Imitation Based Methods

Chaosheng Dong chaosheng@pitt.edu **Ibrahim El Shar** ije8@pitt.edu

Tian Tong ttong1@andrew.cmu.edu **Yijia Wang** yiw94@pitt.edu

1 Introduction

We are working in the World 1. Solving World 1 requires training an agent to learn the best possible strategy to navigate the infinite horizon environment and collect rewards efficiently. This involves taking the shortest path to collect items and having a high level plan that guides the agent and prioritize where to go and what to collect. It is important because we believe that it can be related to real life problems which involve rescuing humans from various hazards. For example, in fire situations, robots can be trained to rescue people trapped in buildings/cities burning in fire. Other examples include searching for survivors of earth quakes, Tsunamis, or sinking ships. It is worth to mention that all these examples have a common ground in their environments which is "Navigate, Fetch and Find". These important humane problems are very important and are yet to be studied extensively. We hope that our current study helps in taking a step further towards this end.

2 Background

We have implemented the recommended baseline method using Deep Q-networks (DQN) [1]. To facilitate the learning process, we used a (observation, action, next observation) representation as the state. We trained our agent continuously over 2 million steps without resetting the environment.

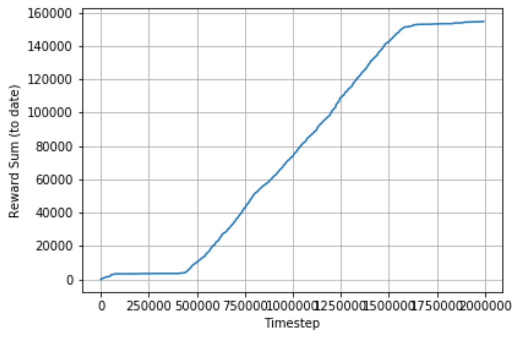
The table below shows the hyperparameters used in the DQN implementation.

Table 1: Baseline - DQN Implementation Details

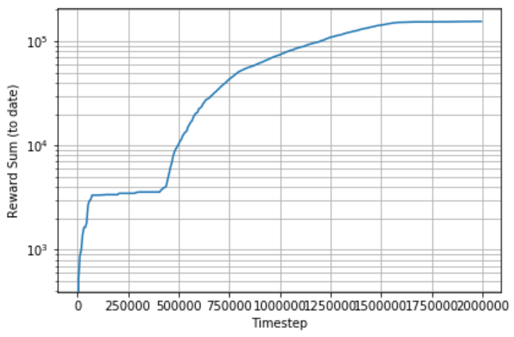
DQN	
Settings	Value
NN MLP Architecture	3 hidden layers each of size 735 with Relu activation
Optimizer	Adam
Burn-in transitions	10000
Initial epsilon	1
Final epsilon	0.1
Exploration decay steps	1×10^5
Minibatch size	32
No. of steps until target network is updated	2000
Steps before minibatch	1
Learning rate	0.001
Discount factor	1
No. of total training steps	2×10^6

Fig. 1 shows the baseline plots. In particular, the sum of rewards over time steps is shown in Fig. 1a & 1b and the average of rewards over time steps is shown in Fig. 1c & 1d.

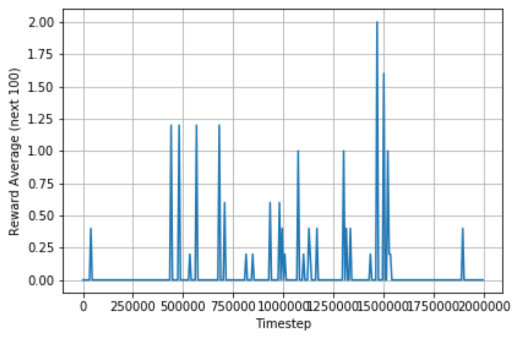
The training curve shows that the DQN agent was able to learn reasonable results with substantial learning progress. The agent achieves the highest average reward over 100 steps at about $2/3$ of the training process (1.5 million steps). At this point of training epsilon has already reached its minimum at 0.1 showing that the agent have actually learned to act in this environment with a reasonable policy.



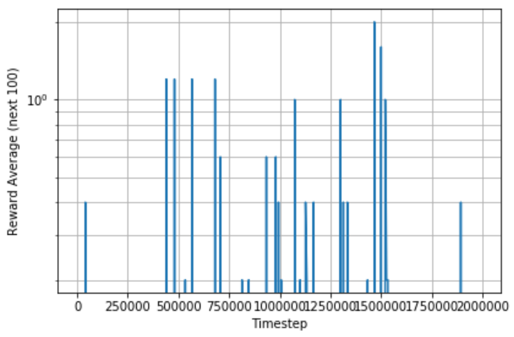
(a) Sum of rewards over time steps.



(b) Sum of rewards over time steps (log scale).



(c) Average of rewards over time steps.



(d) Average of rewards over time steps (log scale).

Figure 1: Baseline: DQN agent training curves recorded over 2 million steps

3 Related Works

Imitation learning methods [2] aim to mimic an expert policy in a given environment. The agent is trained to act in an environment from demonstrations by learning a mapping between observations and actions. In stead of following a human expert, we follow our Greedy Heuristic policy which has an outstanding performance for this environment.

Deep Q-learning from Demonstrations (DQfD) [3] reforms the DQN to take in expert demonstrations. DQfD imports expert demonstrations and trains to follow the expert in addition to minimize the DQN loss.

4 Methods

4.1 Greedy Heuristic

We first implemented a Greedy Heuristic policy. It follows the shortest path to approach the most valuable items in the vision. More specifically, the agent takes the following action with priority from high to low:

1. If the agent carries tongs, and there are diamonds in the vision, then it approaches one along the shortest path.
2. Otherwise, if there are tongs in the vision, then it approaches one along the shortest path.

3. Otherwise, if there are jellybeans in the vision, then it approaches one along the shortest path.
4. Otherwise, it approaches out of the vision along the shortest path.

To find the action described above, we remodelled the local environment within the vision. Notice that this modified environment is only used for the local simulation, during which the actual agent does not move, and the actual environment is not affected.

- Local Q-value of shape $11 \times 11 \times 4 \times 3$, representing 11×11 locations, 4 directions and 3 actions. Given a vision matrix, the location of the actual agent is always $(0, 0)$, while the direction of the actual agent is always 0.
- All simulated actions result an additional reward -1, to encourage for the shortest path.
- If the agent carries tongs, and the simulated agent arrives to a location of diamonds, then the local simulation terminates with reward 100.
- If the simulated agent arrives to a location of tongs, then the local simulation terminates with reward 40.
- If the simulated agent arrives to a location of jellybeans, then the local simulation terminates with reward 20.
- If the simulated agent arrives out of the vision, then the local simulation terminates with reward 0.

Notice that although the actual environment is never ended, the local simulation terminates once the agent finds an item. It prevents the value iteration to fall into endless loops. Also, it means that the agent will greedily approach one item, which is the most valuable one.

At each time step t, we run the value iteration algorithm to find the fixed point solution to the local Q-value, based on the current vision, and then choose the action as the optimal simulated action of the current location and direction, i.e., $A_t = \arg\max_a Q_{\text{local}}(0, 0, 0, a)$.

4.2 Imitation Learning

Motivated by the Greedy Heuristic outstanding performance, we implement an agent based on imitation learning to mimic the actions of the Greedy heuristic agent and may be generalize over unseen states. To do this, we do supervised learning on the Greedy Heuristic actions, using DAGGER [2]. The below table shows the hyperparameters used in our imitation learning implementation.

Table 2: Imitation Learning Implementation Details

Imitation Learning	
Settings	Value
NN MLP Architecture	3 hidden layers of size 512, 64, 18 respectively with Relu activation. Last layer has 3 units with softmax activation.
Optimizer	Adam
Training batch size	32
Learning rate	0.001
Discount factor	1
No. of total training steps	1×10^4

One difficulty faced with this method is that it relies heavily on the the Greedy Heuristic that has a high computational complexity. Accordingly, we found it a bit hard to effectively train our agent using this method.

Nonetheless, Fig. 3 shows the performance of this method compared to methods utilized herein.

4.3 Deep Q-learning from Demonstrations

We implemented the deep Q-learning from demonstrations (DQfD). We use the same neural network architecture as that used in DQN. In addition to minimizing the DQN loss, another loss term is added to encourage the agent to mimic the Greedy Heuristic actions.

We adopt the loss function used in [3] to update the network:

$$J(Q) = J_{DQ}(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q)$$

The four parts of loss are as follows respectively:

- $J_{DQ}(Q) = (R(s,a) + \gamma Q(s_{t+1}, a_{t+1}^{\epsilon-greedy}; \theta') - Q(s,a;\theta))^2$ is the double DQN loss, where θ' are the parameters of the target network, and $a_{t+1}^{\epsilon-greedy}$ is the ϵ-greedy policy given state s_{t+1}.
- $J_n(Q)$ is the n-step loss, $J_n(Q) = (r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n Q(s_{t+n}, a_{t+1}^{\epsilon-greedy}; \theta') - Q(s,a;\theta))^2$.
- $J_E(Q) = \max_{a \in A}[Q(s,a) + l(a_E, a)] - Q(s, a_E)$ is the margin classification loss, where a_E is expert demonstrator's action given state s, and $l(a_E, a) = 0.8 \times \mathbf{1}_{a \neq a_E}$. We use the Greedy Heuristic policy as the expert policy.
- $J_{L2}(Q)$ is the $L2$ regularization.

The weights of these losses we use are $\lambda_1 = 0.3, \lambda_2 = 1.0, \lambda_3 = 0.00001$.

At the beginning, the replay buffer is filled with transitions from expert demonstrations, and after that the agent begins to generate new transitions. It can be viewed that the expert demonstrations serve as a better initialization, instead of starting from random initialization of weights.

Table 3 show the hyperparameters used in our DQfD implementation.

Table 3: DQfD Implementation Details

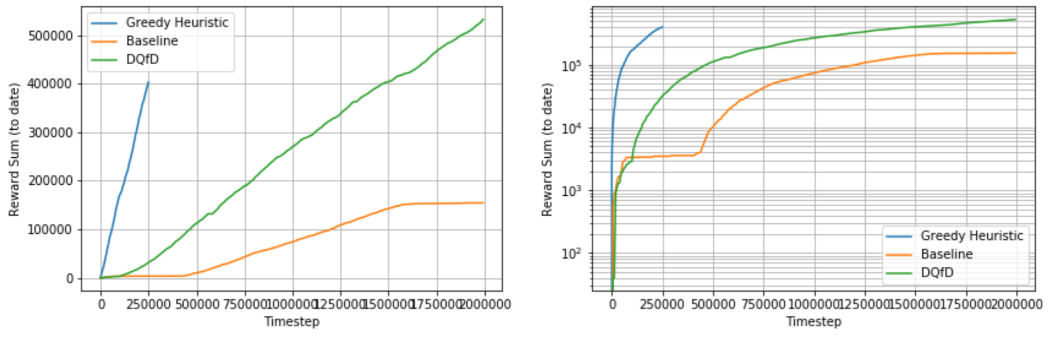
Settings	Value
NN MLP Architecture	3 hidden layers each of size 735 with Relu activation
Optimizer	Adam
Demonstration buffer size	5×10^4
Total buffer size	10^5
Initial epsilon	1
Final epsilon	0.01
Exploration decay rate	0.999
Minibatch size	64
Pre-training steps	10^4
No. of steps until selection network is updated	10
No. of steps until target network is updated	1000
Learning rate	0.001
Discount factor	1
No. of total training steps	2×10^6

5 Results

Fig. 2 shows the training results of the implemented algorithms. In particular, the sum of rewards over time steps is shown in Fig. 2a & 2b and the average of rewards over time steps is shown in Fig. 2c & 2d.
The training curves show that DQfD was able to learn substantially and at a very high rate as compared to the DQN baseline agent. Moreover, it is clear from Fig. 2 that our Greedy Heuristic agent has outperformed and by far both the baseline DQN and DQfD agents. The Greedy Heuristic agent was able to reach a reward sum of 400,000 in only 250,000 steps, Fig. 2a. DQfD, on the other hand, took an extra 1,250,000 steps to reach the same value which DQN (baseline) failed to achieve over the total course of training, 2 million steps. The superiority of the Greedy Heuristic comes however at the cost of the computational complexity which explains why we only run it for 250k steps. One can also see that at around 150 million steps DQN performance has plateaued considerably whereas DQfD continued to learn with the same stability over the course of training.

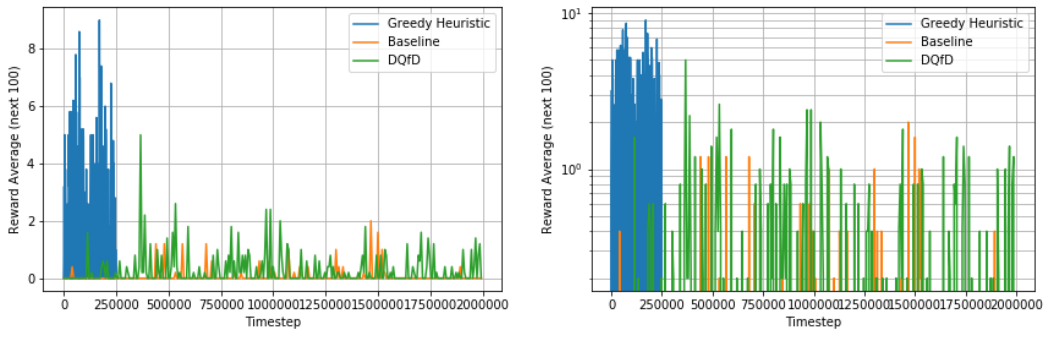
Fig. 3 shows the performance curves of the implemented methods which includes imitation learning in addition to the algorithms mentioned above. As expected, the Greedy Heuristic, imitation learning and

DQfD outperform the baseline. We believe that with enough training imitation learning performance can get even better than the current one which was obtained over 10,000 steps only.



(a) Sum of rewards over time steps.

(b) Sum of rewards over time steps (log scale).



(c) Average of rewards over time steps.

(d) Average of rewards over time steps (log scale).

Figure 2: Baseline (DQN), DQfD and Greedy Heuristic training curves: Baseline DQN and DQfD agents training curves were recorded over 2 million steps. Obtaining the Greedy Heuristic's rewards over 2 million steps was expensive so we ran it over 250k steps instead.

6 Discussion and Analysis

The baseline method implemented by DQN is a general approach for reinforcement learning, however, it may fall into inefficient policies and take very long time to improve. From our experiments, the agent usually moves in circles instead of taking the direct path to collect the items.

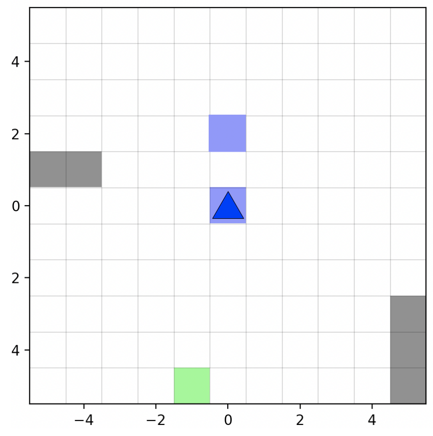
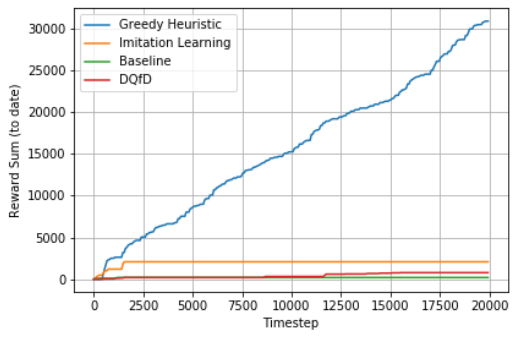
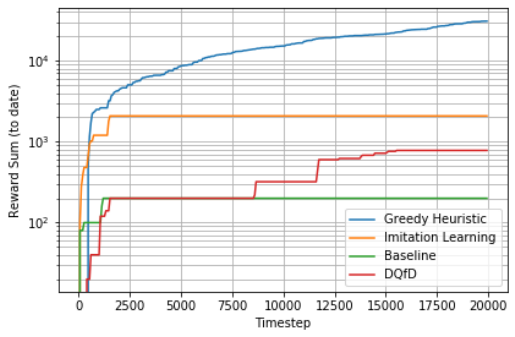


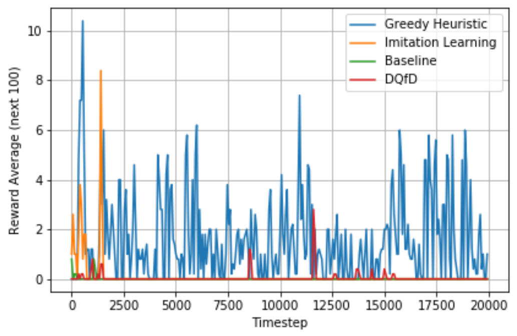
Figure 4: Example where Greedy Heuristic is not optimal: Greedy Heuristic policy is to approach the diamond (green block), but a better policy is first to approach the jellybean (blue block) and then turn back to approach the diamond (green block).



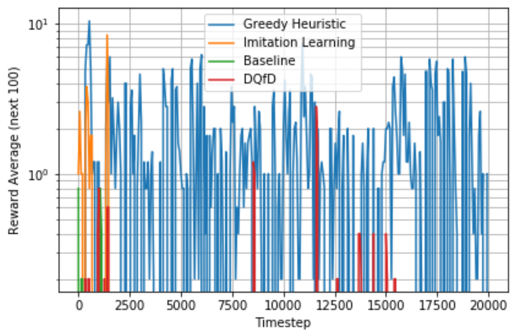
(a) Sum of rewards over time steps.



(b) Sum of rewards over time steps (log scale).



(c) Average of rewards over time steps.



(d) Average of rewards over time steps (log scale).

Figure 3: Baseline (DQN), DQfD, Imitation Learning and Greedy Heuristic performance curves: Baseline DQN and DQfD agents performance curves were recorded over 20k steps.

To find a smarter policy to approach items more efficiently, we designed the Greedy Heuristic policy, which follows the the shortest path to approach the most valuable items in the vision. The Greedy Heuristic is not optimal, for example in 4, the Greedy Heuristic policy is to approach the diamond, since it has higher reward than the jellybean, and after collecting the diamond, the agent cannot see the jellybean since the jellybean is out of the vision; but a better policy is first to approach the jellybean and then turn back to approach the diamond.

Although not optimal, we believe that the Greedy Heuristic policy behaves nearly as optimal in most situations. Our experiments show that Greedy Heuristic outperforms all other methods. A main drawback is that Greedy Heuristic is computationally costly, since it requires to find the fixed point solution for $11 \times 11 \times 4 \times 3$ local Q-value for very steps by Q-iteration.

We tried the imitation learning on Greedy Heuristic using DAGGER to perform supervised learning on Greedy Heuristic. Its behavior is consistently worse than Greedy Heuristic.

Besides that, we implemented the deep Q-learning from demonstrations (DQfD). Taking advantage of Greedy Heuristic demonstrations, the agent is able to gain much higher rewards compared to the DQN in baseline. However, it still cannot beat the Greedy Heuristic policy. We believe that since the Greedy Heuristic policy uses knowledge of the environment and is nearly optimal, DQfD, as a model-free method, is quite hard to beat that.

Moreover, one more method we implemented but did not include in this writeup is DQN with epsilon Greedy Heuristic, i.e., with probability epsilon we use the Greedy Heuristic action to learn its Q-values. Nonetheless, this method showed to be less effective than the other methods mentioned herein.

Overall, one can conclude that compared to pure reinforcement learning (RL), algorithms that involve both “planning” and RL would probably perform better. This can be seen from this study and from recent success of this kind of algorithms. We mention: Alphago Zero as an example of "planning and RL" algorithms that has proved to be successful in practice.

References

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

[2] 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.

[3] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, et al. Deep Q-learning from demonstrations. *arXiv preprint arXiv:1704.03732*, 2017.