Implementing Inverse Q-Learning

Celine Aubuchon and Timothy Ho

November 2023

Introduction

In the field of reinforcement learning where decision-making is often modeled by Markov Decision Processes, a significant subset of problems exists where the reward function and state dynamics are completely unknown. Some attempts to solve these problems in cases where expert trajectories are readily available come in the form of imitation learning (IL), where a policy is inferred from demonstration data. One dominant class of algorithms under imitation learning is inverse reinforcement learning (IRL), where a reward function is learned. A known challenge in inferring a reward function is the fact that different reward functions could give rise to the same trajectories. Conversely, even similar reward functions that differ in slight but crucial aspects may give rise to very different trajectories[1].

Some researchers have attempted to alleviate the challenge of navigating the large space of possible reward functions by placing reasonable constraints on the reward function estimation—including natural heuristics [5] and, more recently, constraints extracted from expert data [4]. While these approaches improve the efficiency of IRL, constraints can be challenging to model and apply in different settings. Garg et al. propose instead a novel imitation learning method to learn a Q-function, IQ-Learn, which implicitly defines both a reward and policy function[3] while avoiding the need to apply additional constraints.

In our work, we re-implement the core algorithm of IQ-Learn to train a model offline using expert demonstrations, and compare its performance to a plain deep Q-network of the same architecture. Our main interest was to examine the difference in how these training methods affect performance and meaningful learning, rather than test the method's efficiency compared to other IRL methods, as was the focus of the original paper. To this end, we chose the GridWorld environment because of the tractability of visualizing rewards in a constrained state space and its amenability to modification, which helps us to distinguish between pure imitation and meaningful learning about the environment. Our implementation of GridWorld features three levels of difficulty, which we use to challenge and assess the limits of what the agents learn. We find that compared to a plain DQN, IQ-Learn matches its performance without reward information or such an abundance of training data. We also find that IQ-Learn is able to recover rewards proportional to the ground truth rewards in our GridWorld environment. Through our visualization tests, we also find evidence that the networks may have learned the position of the win state, but failed to represent obstacles like walls and boundaries.

Implementation

Gridworld Environment

We designed a discrete GridWorld environment with adjustable width and height, number of wall obstacles, and randomization options for wall positions, player start positions, and win state positions per game. A player in the GridWorld can make moves using the *step* method, which accepts one of four possible actions: up, right, down, or left. This method takes an action, updates the GridWorld state to reflect the player's new position within the grid, and returns a reward. The environment reward function is defined as:

$$reward = \begin{cases} 10 & \text{if } player_{xy} == win_{xy} \\ -1 & \text{else} \end{cases}$$

In our experiments, we tested three different grid types of increasing difficulty:

- **static walls**: Five static walls are placed in a fixed arrangement across games. The win state is always in the bottom-right corner.
- **random walls**: Five walls are randomly placed for each game. The win state remains fixed in the bottom-right corner.
- **random walls & random win states**: Five walls are randomly placed and the win state is randomly placed for each game.

For each grid type, the player is spawned in a random position at the start of each game. The player has 100 moves to travel to the win state before the game is over.

Training

A key requirement of our presented work is to train neural networks to play GridWorld. We train two fully-connected networks using Deep-Q learning and IQ-Learn [3]. In our implementation, the networks' goal is to learn a Q-function over state-action pairs, which represent the "desirability" of each action in each state. With randomly-initialized weights, the network begins with a sub-optimal Q-function in the space of possible Q-functions. Over the course of training, the networks navigate this space of possible Q-functions by minimizing loss between the current Q-function and an estimation of a better Q-function. The DQN and IQ-Learn loss functions crucially differ in the information that is used to measure this loss.

The DQN measures the loss between the current estimate of Q-values and a less biased estimate of Q-values that incorporates observed rewards. Typically, in deep reinforcement learning, network training is done online, which can accelerate convergence to an optimal policy. However, this is not strictly necessary. Given sufficient data, offline learning based on even random actions can be used to optimize the Q-function given that these actions occasionally yield positive rewards.

In stark contrast, IQ-Learn is challenged by the absence of environment rewards. It instead leverages the expertise of its training data to represent the better Q-function that

it compares its current Q-function to. The loss function simply minimizes the difference between the estimate of Q based on the expert's behavior and the estimate of V. This loss function penalizes any preferences that do not match the expert's behavior. Crucially, the design of this loss function relies on the assumption that the expert data used in training is optimal since it uses it to define its target Q function.

In our experiments, we compare a DQN trained online to an IQ-Learn network trained offline with human expert data. Our key metrics for comparison are the efficiency of the training and the performance of the models during gameplay.

Agents

We created an Agent parent class, that has a method *get_action* which outputs actions given a GridWorld state. The Agent can also store information passed into *inform_result* about results of its most recent outputted action. The default Agent chooses actions randomly from the action space. It has three child classes: HumanAgent, DQNAgent, and IQLearnAgent.

HumanAgent The HumanAgent class implements *get_action* by accepting keyboard input. This class was created primarily to collect human expert data that could both be used to train the IQLearnAgent and compare the performance of any computer-controlled agent to a human standard. The choice of the GridWorld environment was partially motivated by the ease with which we could control this human data for two reasons: (1) near-perfect performance is easy to achieve without practice for a human, which allowed us to generate large expert datasets for each grid type, and (2) game play data is easy to manipulate since the state and action spaces are discrete and intuitive.

DQNAgent The DQNAgent is initialized with an instance of the Model class, which handles batch training and inference through a fully connected neural network to output Q-values for each action. As the DQNAgent plays GridWorld, it accumulates gameplay data into a replay buffer used for training, where the data include the states, actions, rewards, next states, and termination statuses for every timestep. The DQNAgent takes an epsilon-greedy approach to training. The aim of this training is to improve the efficiency with which the agent navigates the player to the win state by minimizing the loss between the current estimate of Q-values and a less biased estimate of Q-values that incorporate observed rewards (algorithm 1).

IQLearnAgent The IQLearnAgent functions similarly to the DQNAgent, except it trains offline using an expert buffer that is pre-filled before gameplay. The loss function for the IQLearnAgent is based on the offline IQ loss function [3], which penalizes any preferences that don't match the expert behavior (algorithm 2).

Algorithm 1 DQN Training

$s, s', a, r, d \leftarrow \text{buffer}(batch\_size)$
$Q(s, \cdot) \leftarrow \text{Model}(s)$
$Q(s', \cdot) \leftarrow \text{Model}(s')$
$V(s') \leftarrow \max(Q(s', \cdot))$
$x \leftarrow Q(s, a)$
$y \leftarrow r + \neg d * \gamma * V(s')$
$loss \leftarrow \text{MSELoss}(x, y)$

Algorithm 2 IQLearn Training

$s, s', a, r, d \leftarrow \text{buffer}(batch\_size)$
$Q(s, \cdot) \leftarrow \text{Model}(s)$
$Q(s', \cdot) \leftarrow \text{Model}(s')$
$V(s') \leftarrow \max(Q(s', \cdot))$
$x_1 \leftarrow Q(s, a)$
$x_2 \leftarrow V(s) \leftarrow \max(Q(s, \cdot))$
$y \leftarrow \neg d * \gamma * V(s')$
$loss \leftarrow -\text{mean}(x_1 - y) + \text{mean}(x_2 - y)$

Pseudocode for the loss calculation in *train* methods of the DQNAgent and IQLearnAgent classes. The DQN uses a standard Bellman Error loss. The IQLearnAgent, which specifically uses an offline version of IQ-Learn, aims to balance the Q value of an expert's action with the model's value for the same state.

Orchestrator

The Orchestrator class is an interface between a GridWorld and an agent. The Orchestrator is constructed using an instance of GridWorld and an instance of an Agent and also accepts some parameters to control how many moves, or optionally how many complete games, the agent should play. The Orchestrator additionally controls training for the DQNAgent and IQ-Learn agent, calling their *train* methods between intervals of gameplay steps that are determined by the agent's *training_freq* parameter.

Calling the *play* method of the Orchestrator begins gameplay between the agent and the GridWorld. This class monitors the performance of the agent by collecting each of the agents' actions and the resulting game states into a trajectory dataset and measuring the efficiency of the agent's path for each game. This data is intended to be used for analysis.

Data

A key difference in the training of the two models is the source of gameplay data. Our plain DQN is trained online while collecting its own data from interacting with the GridWorld environment. The IQ-Learn model instead trains offline using expert data. We collected 5000 steps of GridWorld gameplay for each grid type using our HumanAgent class. Visitation rates for each state across these 5000 steps (Fig. 1) indicate that there are sufficient samples present in each dataset to train the models.

For the static walls grid type, the IQ-Learn agent was trained with 300 randomly sampled expert trajectories, while for the random walls and random winstates grid types, it was trained with 450 trajectories.

For both DQN and IQ-Learn, we applied a small range of random noise to the states to avoid 'dead neurons' during network training, since much of the grid representation is identical during the trajectories of single games.

Figure 1: Visitation heat maps for each human expert dataset. Each cell displays the count of state visits over the 5000 game steps. The black squares in the Static Walls set represent the static arrangement of walls both in the games played by the expert and the training of the networks.

Results

A DQN and IQ-Learn network was trained for each grid type with matching hyperparameters. The normalized loss per training step for each network is plotted in Fig. 2 to compare the convergence rate between DQN and IQ-Learn. Across all three grid types, IQ-Learn converges faster than the DQN and reaches its terminal average loss within the first 200 steps. This indicates that IQ-Learn can predict expert behavior well after minimal training. While the DQN appears to converge with low variability in the static walls condition, its losses are high and unstable for the random walls and random win states grid types. Under the more challenging randomized grid configurations, it seems that the DQN is not able to accurately predict expected return.

To assess gameplay performance, we measured the distance ratio between the path the agent took and the distance between its start position and the win state for each game. In a grid without obstacles, the minimum perfect distance ratio is 1. The average distance ratios of the expert datasets for the static walls, random walls, and random win states grid types were 1.00, 1.02, and 1.03 respectively.

To track the performance of the models during training, we measured their average distance ratios for 24 games in testing blocks every 50 training steps. Fig. 3 plots the average distance ratios for each testing block (left) by grid type. For all grid types, IQ-Learn achieved its best average level of performance by the first testing block. For the DQN in the static walls condition, it required almost ten-fold training steps to achieve this level of performance. For both the random walls and the random win states grid types, the performance across testing blocks is highly unstable. It appears that the IQ-Learn network may achieve slightly better performance in the early testing blocks of the random walls condition compared to the DQN, but the DQN gradually improves to match it by the 300th testing block.

We also tested each network for 5000 timesteps after training was complete (see Fig.

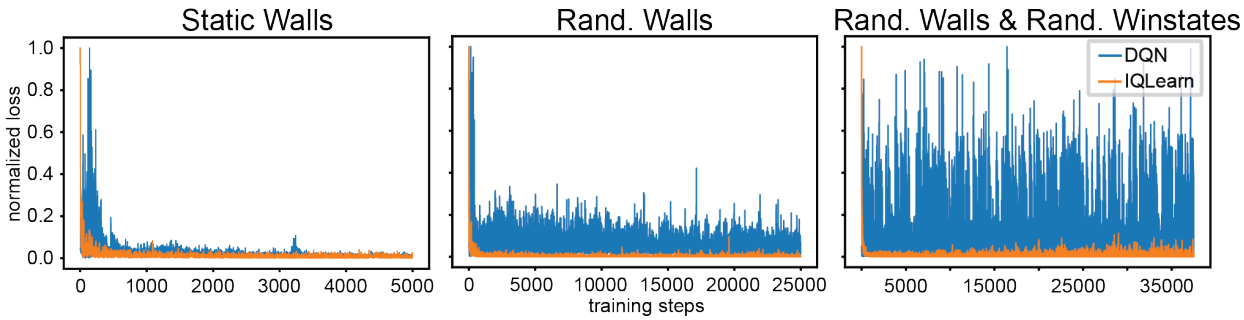

Figure 2: Loss by training step for the DQN and IQ-Learn models for the three grid types. The networks had matching hyperparameters during training based on initial experiments: $\gamma = 0.9$, learning rate = 0.001, batch size = 32, testing frequency = 4, $\epsilon_{init} = 1.0$, $\epsilon_{decay} = 0.97$, $\epsilon_{floor} = 0.1$. To match the difficulty levels of the three grid types, the networks were trained for 20,000, 100,000, and 150,000 timesteps in the static walls, random walls, and random win states grid types respectively.

3 right) to compare their average distance ratios to the expert datasets and the average distance ratios achieved by a random agent in each condition (marked with black 'x's). It seems that for all three conditions, the DQN and IQ-Learn networks have equivalent average performance that is worse than the human expert baseline. However, for the static walls and random walls grid types, the performance is markedly better than a random agent. The variability in gameplay performance scales with the average distance ratio. Importantly, this indicates that the policies that the networks are following in the more challenging random grid configurations lead to different performance under different conditions. This is opposed to the case where a policy is systematically biased as a whole, which would lead to consistently bad performance. Of course, given the random placement of the agent at the start of each game, it is possible that the policy is systematically biased in specific regions of the state space but in other regions is optimal. It is possible that this reflects a common pitfall in imitation learning, where an agent fails in states that are not well represented in its training data. These errors compound and can lead to drastically poor performance [2].

To further investigate each networks' representation of the state-space after training, we plotted state-value maps for each grid type for the DQNs (Fig. 4) and the IQ-Learn networks (Fig. 5).

The state-value maps were estimated by passing every possible state in an empty Grid-World (zero walls) through each network with a fixed win state position. The maximum Q-values returned by the network are assigned to each position on the map. For the static walls and random walls grid types, the DQNs each yielded value maps that seemed to value positions adjacent to the win state more than more distant positions. This smooth gradation in value, though less pronounced in the static walls case, is exactly what we would expect for the expected return for each state. However, for the most challenging random win state grid type, the value map did not have a structure related to the expected return. A closer look at the greedy policies derived from these value maps indicated that the actions that correspond with these estimated returns, in some regions, corresponded to the direction toward the win state, except for the random win states condition.

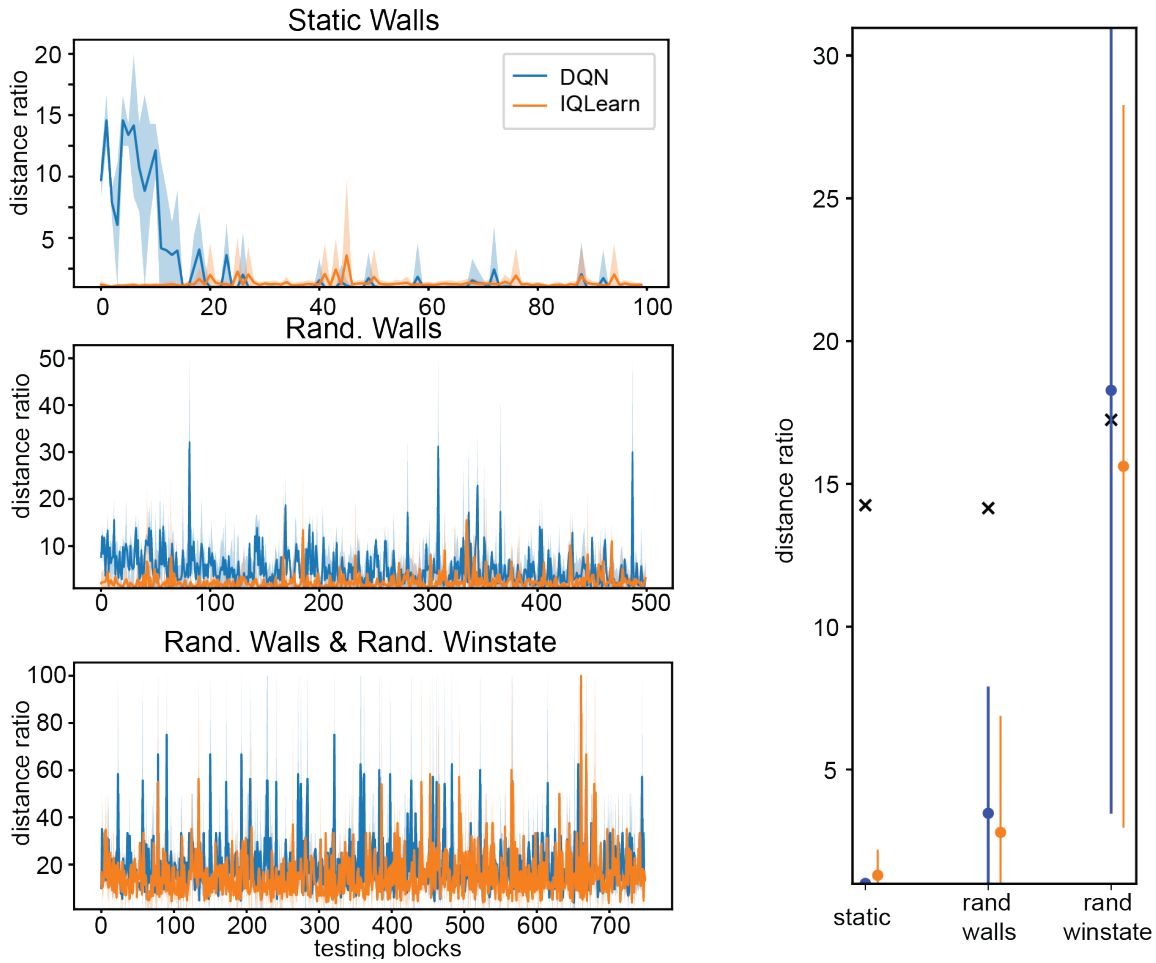

Figure 3: The performance of the DQN and IQ-Learn models during and after training for the three grid types, represented as the average distance ratio over testing blocks. The right plot shows the average distance ratio over 5000 game steps after training for each model and grid type. The black 'x' symbols depict the mean distance ratio of an agent making random moves as a comparison. Shaded regions and error bars represent +/- 1 standard deviation.

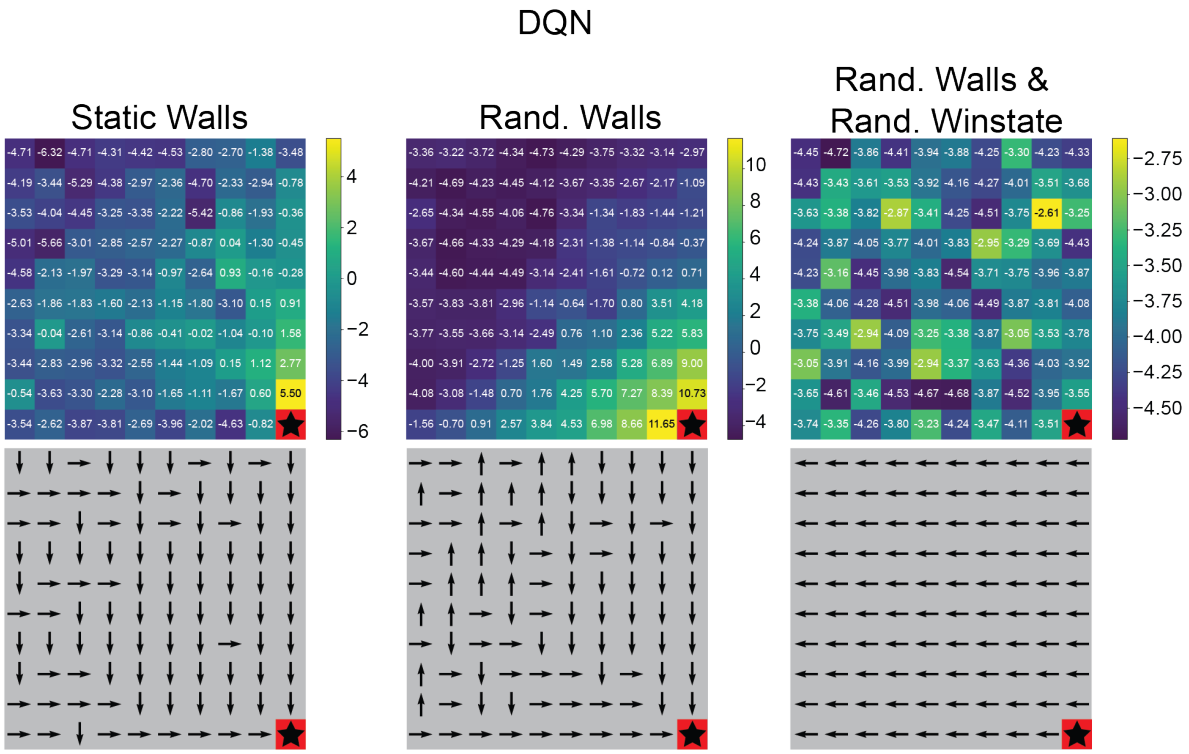

Figure 4: Value maps after training for the DQNs trained on the three grid types. Under each value map is a policy map reporting the action associated with the value for each state. Win states are marked by a black star.

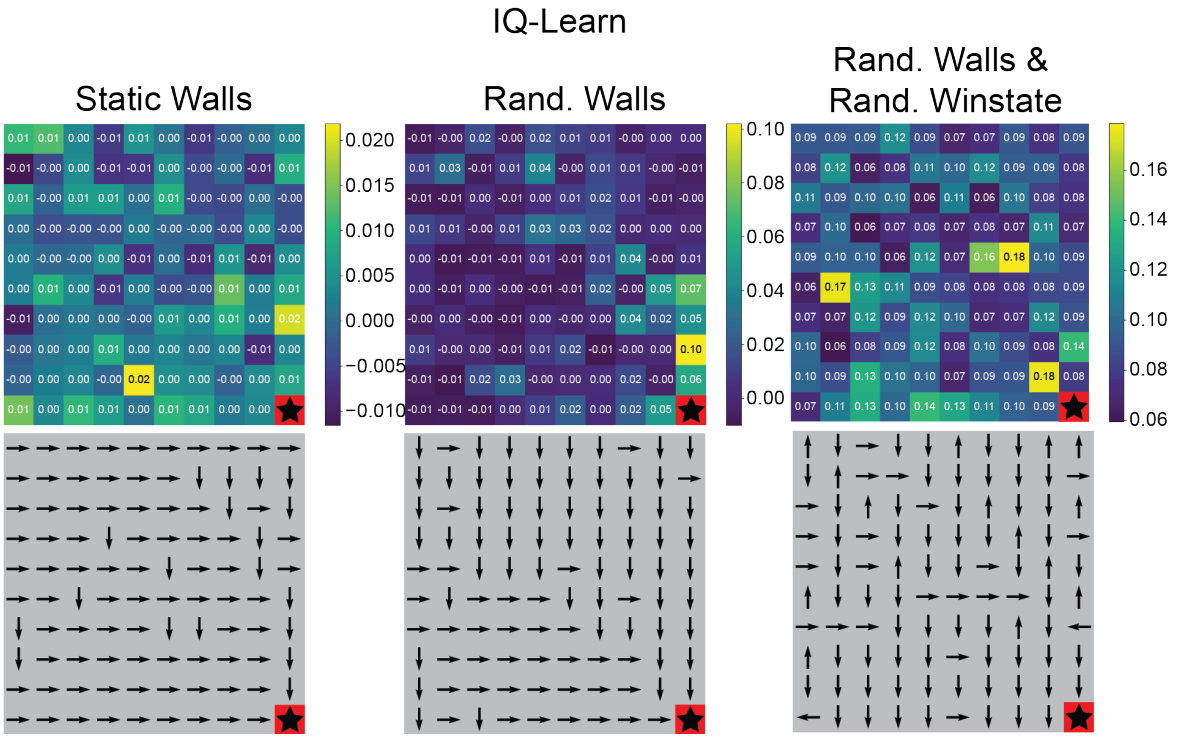

Figure 5: Value maps after training for the IQ-Learn networks trained on the three grid types. Under each value map is a policy map reporting the action associated with the largest Q-value for each state. Win states are marked by a black star.

The state-value maps for IQ-Learn did not appear to have a structure related to the position of the win state. This was an unexpected result, given the approximately equivalent performance of the IQ-Learn networks to the DQNs. However, this may be explained by the policy maps, which show that the actions associated with the values for each grid position do have a consistent structure related to the position of the win state. We ran a further test to see if the policy maps for the IQ-Learn network trained in the random win states condition would reflect varying win state positions reliably (see Fig. 6. Strikingly, an informal observation of the policy maps indicates that the general path to the win state is captured by each state value, even if the range of values across states is not consistent with this pattern. These tests did not include wall obstacles, so it is not clear to what extent the network would produce policies that avoided walls effectively, as would be required to achieve good performance.

While IQ-Learn does not have access to reward data during training, the Q-function it estimates should correlate with environment reward, as shown by Garg et al. (2022). We also assessed whether the recovered rewards from the IQ-Learn networks qualitatively matched the ground truth environment rewards (Fig. 7). It seems that for all grid types, the rewards recovered from the IQ-Learn networks were proportional to the environment rewards. We also depict the rewards recovered from the DQNs. They are expected to recover rewards well, given their direct access to reward information during training. Here, we use them as a reference for the expected variability in reward estimation introduced through the sampling of training data.

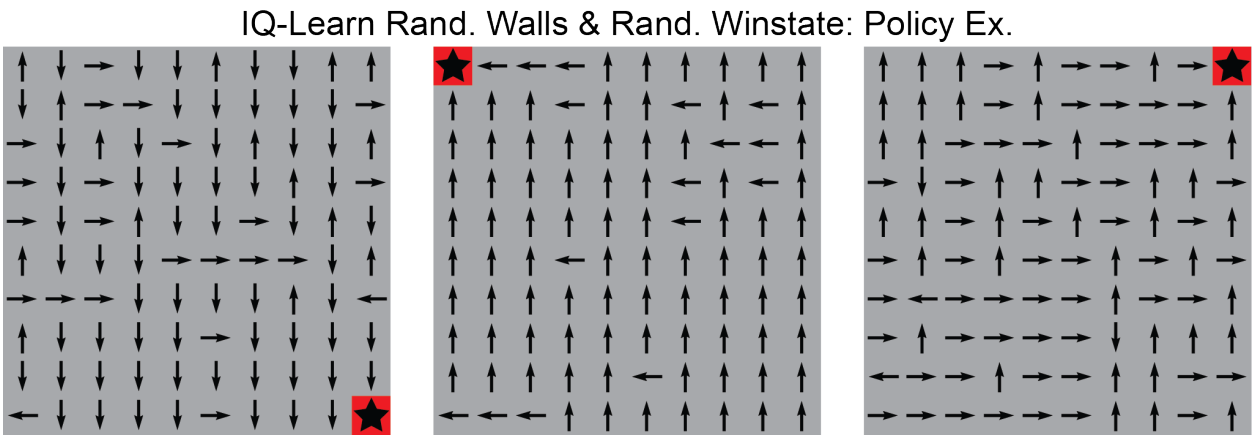

Figure 6: Greedy policy maps for the random walls and random win states trained IQ-Learn network made from empty grids with different win state positions, marked by a black star.

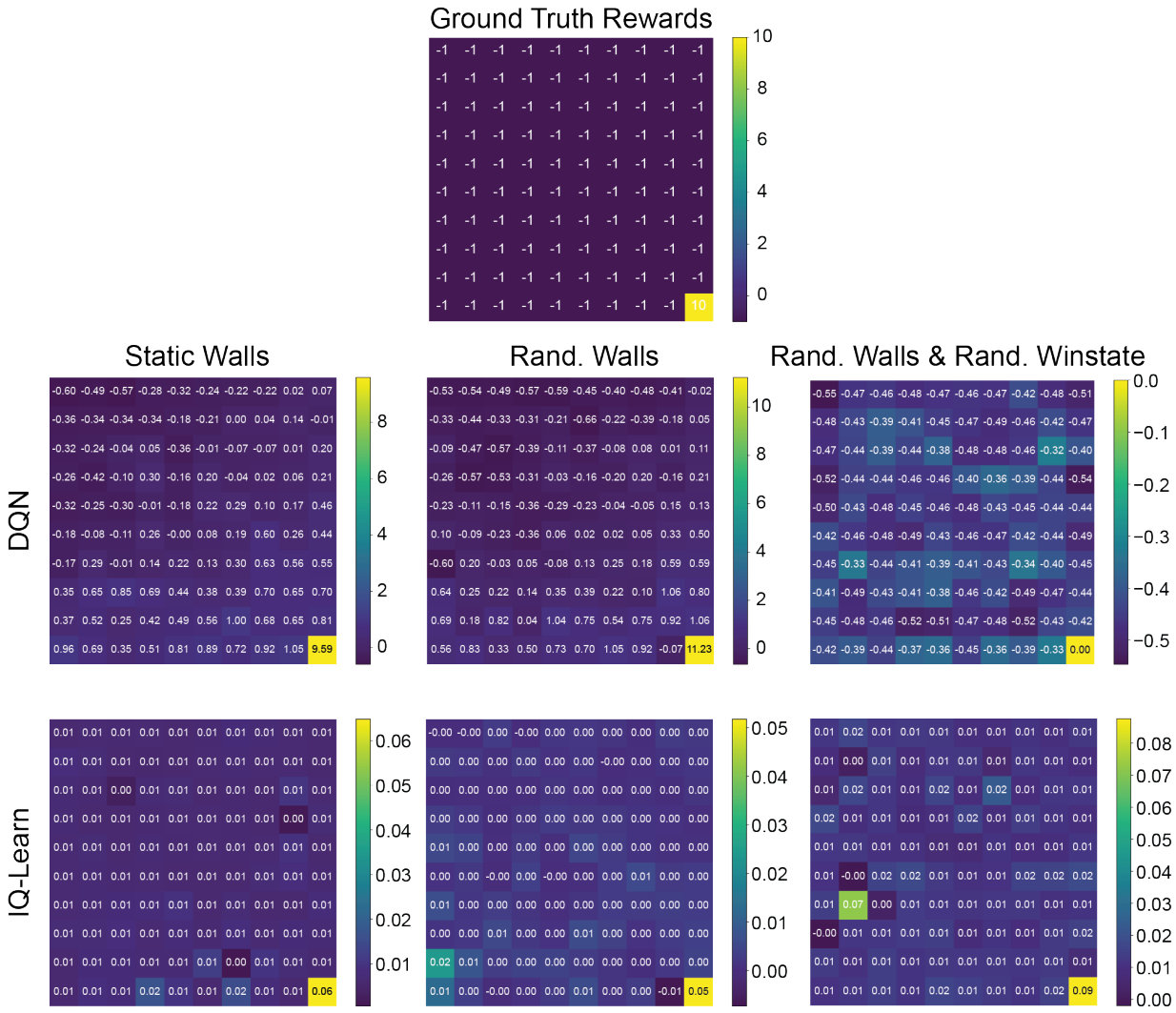

Figure 7: Recovered rewards for each model and grid type.

Discussion

We use two measures of performance to track how quickly the networks learn to solve the GridWorld and the optimality of their solutions:

- Normalized loss as a function of training steps, where the DQN uses mean squared error loss, and the IQ-Learn agent uses IQ Loss.
- Distance ratios comparing the moves taken by the agent to the minimum number of grid tiles separating its start state from its win state

We found that by both measures, the IQ-Learn agent was capable of learning the expected behavior within a far shorter period than the DQN. However, we observed that although the IQ-Learn agent achieves better distance ratios than the DQN in the early testing blocks, its performance is inconsistent and occasionally suboptimal through the entirety of testing. This suggests that it may only be partially learning what the expert knows, but continues to be challenged in some situations.

We looked underneath the performance of the two models and inspected the state values that dictate their policies. The extracted state value maps of both agents illustrate very different learned value functions. Whereas the DQN shows a smooth gradient of increasing value toward the win state, IQ-Learn lacks any discernible pattern in its map. Despite this incongruence between the value maps, their gameplay performance is similar. Both methods require approximately three times the number of actions than the expert to solve the GridWorld. Visualizing the DQN and IQ-Learn gameplay showed that the poor performance may be explained by goal-directed trajectories that were obstructed by unrecognized walls.

Future Work

The current state of the project leaves several potential lines of questioning that could be pursued. For example, the absolute values for the states do not resemble each other between the two methods. The DQN has values in the (-10, 10) range because of its incorporation of explicit rewards, whereas the IQ-Learn agent has values in the (-0.1, 0.1) range because of its reward inference. One path forward would be to further replicate some of the work from the original paper, namely the implementation and testing of various statistical distances that measure divergence between the policy and expert distributions. These statistical distances are functions over the inferred reward, which is a component of the loss function. The authors note that Total Variation, the statistical distance that we used, constrains the reward values at $\frac{1}{2}$, which may partially account for the small range of reward values we uncovered for IQ-Learn (Fig. 7).

Another worthwhile follow-up to our work is measuring the performance of a convolutional neural network which may be better suited to handle the GridWorld problem because of the spatial nature of its states. It is possible that the multi-layer perceptron architecture in our model reached a performance ceiling when facing more spatially complex environments.

Conclusion

We've shown in our work limited success in replicating the achievements of IQ-Learn's core algorithm [3]. Our IQ-Learn agent achieved near-expert level performance in the static walls grid type and matched the DQN performance for the random walls grid type, without access to environment rewards. Though it did not beat random agent performance for the random win state grid type, model inspection suggests that it may encode the position of the win state but fail to avoid wall obstacles (Fig. 6). Our implementation of IQ-Learn also recovered the pattern of the environmental rewards, though it did not capture the ground truth values. Overall, this work demonstrates a simple method for inverse reinforcement learning and imitation learning that is more efficient than data-hungry end-to-end methods like traditional Q-Learning.

References

[1] Arora, S., and Doshi, P. A survey of inverse reinforcement learning: Challenges, methods and progress, 2020.

[2] Bohg, J., Pavone, M., and Sadigh, D. Imitation learning. online, 2023.

[3] Garg, D., Chakraborty, S., Cundy, C., Song, J., Geist, M., and Ermon, S. Iq-learn: Inverse soft-q learning for imitation, 2022.

[4] Malik, S., Anwar, U., Aghasi, A., and Ahmed, A. Inverse constrained reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning* (18–24 Jul 2021), M. Meila and T. Zhang, Eds., vol. 139 of *Proceedings of Machine Learning Research*, PMLR, pp. 7390–7399.

[5] Ng, A. Y., Russell, S., et al. Algorithms for inverse reinforcement learning. In *Icml* (2000), vol. 1, p. 2.