IRL with Suboptimal Experts

Parsa Saadatpanah, Steven Schwarcz, Chinmaya Devaraj

Introduction

In this project we were trying to find a feasible solution for the Inverse Reinforcement Learning in a setting where the expert trajectories are not completely optimal. [Russell, 1998] defines Inverse Reinforcement Learning as: **Given** 1) measurements of an agent's behavior over time in a variety of circumstances, 2) if needed, measurements of the sensory inputs of that agent; 3) if available, a model of the environment, **determine** the reward function being optimized.

The problem definition does not talk about how well behaved the experts are, and if they know the exact model of the environment when they are taking actions. Therefore a general solution to this problem also should not include any assumptions as to whether the expert trajectories are optimal or not. However, existing works in this field either explicitly assume the expert trajectories are all optimal, or their algorithms tend to work poorly for sub-optimal expert trajectories. We propose a new algorithm which is much more resilient to sub-optimal trajectories.

A very important thing to consider here is the notion of "sub-optimality". It is not clear what would be a good definition of sub-optimality. We know that in real world examples, there is no expert that always behaves optimally. For example when a human wants to go from point A to point B, he may try to minimize a reward function that includes many factors like: travel time, gas usage, accident possibility, etc. The path chosen by a human may not be best solution, but it should have very close to the best reward. Other than this similarity to the best trajectory, we don't know much about how the trajectory was chosen. This needs to be studied further in future studies. In this project, we simulate sub-optimality by masking some reward values across all the states. During a single generated trajectory, each state loses all of its rewards (negative or positive) with equal chance. We believe this is a decent assumption about the real world. For example, returning the driving scenario, a driver might not be aware of an accident on some road, so he would have no idea about the reward function in that state.

Approach

Many of the previous works Subramanian[1], NG[2] for this problem assume that the expert behavior is optimal and is in fact maximizing an unknown optimal reward function. The most important thing left out in these algorithms is the notion that there is no optimal expert in the real world, and that all experts are suboptimal to some extent. This leads all of these algorithms to learn a "suboptimal" reward function rather than the true reward function of the world.
We modify the approach discussed in Ziebart[3], where the original ''Maximum Entropy IRL algorithm'' does not make any assumptions as to the quality of the expert trajectories, or

explicitly compare the similarities of different trajectories. In response, we present a variant of Maximum Entropy IRL which tries to fix this. Our modification accounts for some notion of consensus among experts. This is motivated by the belief that suboptimal experts will tend to disagree on actions that are suboptimal, but will likely agree on optimal actions. In short, we are assuming that there is no structural bias among experts towards any specific suboptimal policy. We therefore want the optimizer to pay more attention to the actions that all experts agree on, and less attention to the actions only a few experts take. Our proposed algorithm (presented with the same notation as the original Maximum Entropy IRL paper[3]) makes use of the following equations:

$$P(\varsigma_i|\theta) = \frac{1}{Z(\theta)} e^{\Sigma_{s_j \in \varsigma_i} \theta^T f_{s_j} C(s_j)}$$

$$C(s_j) = \Sigma_{\varsigma_i \in \varsigma} \textit{number of times state } s_j \textit{ appears in path } \varsigma_i.$$

Here, f_{s_j} denotes the features at state s_j, and $Z(\theta)$ represents the partition function.

In this case, the summation over all $C(s_j)$ adds a strong preferential bias towards states that appear more often within any given trajectory.

Our objective is then to solve for:

$$\theta^* = argmax_\theta L(\theta) = argmax_\theta \Sigma_{examples} \log P(\varsigma|\theta).$$

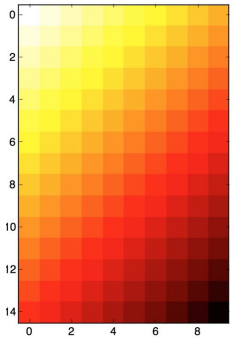
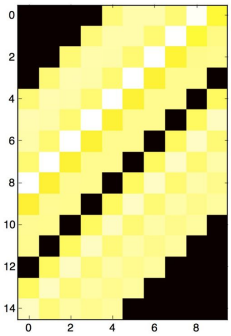
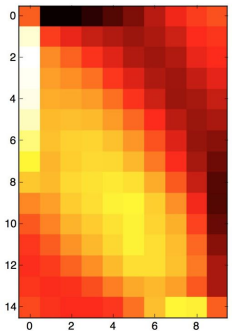
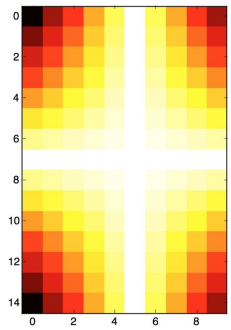
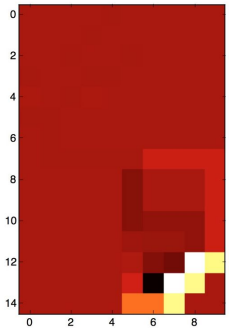
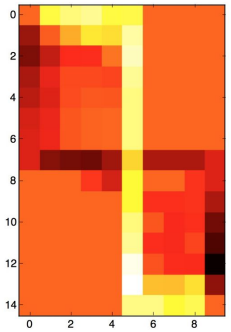
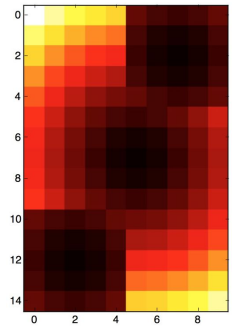
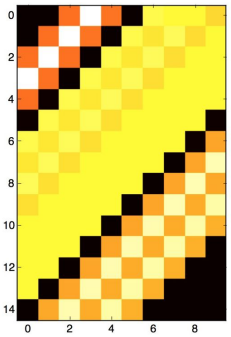
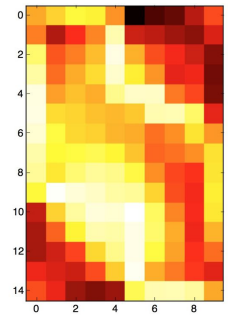
This is a convex function and can be optimized using:

$$\nabla L(\theta) = \tilde{f} - \Sigma_\varsigma P(\varsigma|\theta) f_\varsigma.$$

Test Results

We compared our algorithm to the original algorithm in Ziebart[3] in multiple scenarios. In all the scenarios we used a 15x10 grid with known rewards at each state. The feature vector was simply a one-hot vector that with a 1 in the index that corresponded to the associated state in the grid.

We then generated 1000 suboptimal trajectories on this grid. We found that using more than 1000 trajectories did not yield a noticeable difference in results. We simulated suboptimality according to the scheme mentioned above, wherein we removed the reward information for %10 of the states on the grid. This means that before generating any given trajectory, we replaced 10% of the rewards with 0, selected uniformly at random, and then used Value Iteration to learn a trajectory for this suboptimal grid. Then, we fed these trajectories to both our algorithm and Ziebart's[3] algorithm. The following images show the results. At each row, the image on the left is the actual reward function of the grid, the image in the middle is what Ziebart[3] learns, and the image on the right is what our algorithm learns.

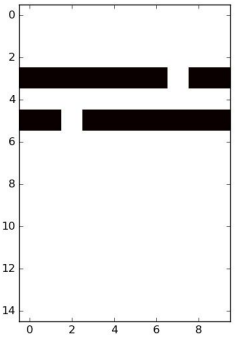
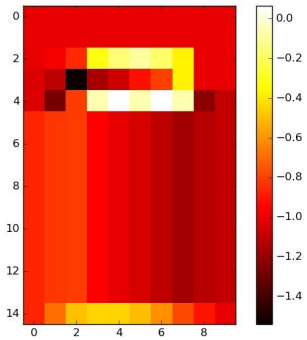
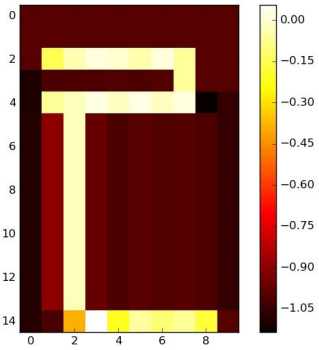
Real Reward	Ziebart[3]	Our Algorithm
		
		
		

In the figure above, hotter colors indicate higher reward and darker spaces indicate lower reward. Red and black generally indicate negative reward. In all cases, agents begin in the upper left corner and are tasked with reaching the lower right, while collected the maximum possible reward along the way.

It can be seen that in all 3 cases, our algorithm learns the reward function very similar to the real function. Many of the differences can be attributed to the fact that experts, even when suboptimal, still explore only a small portion of the grid. It's important to note, however, that our algorithm tends to attribute high reward to the places in the grid that tend to fall along the optimal path for any given reward scheme.

For these examples where the reward function changes smoothly and the optimal policy is often very close to a straight line, the algorithm presented in Ziebart[3] often breaks down, whereas our algorithm remains robust. Using the rewards generated by Ziebart[3] in these cases would not produce desirable behavior.

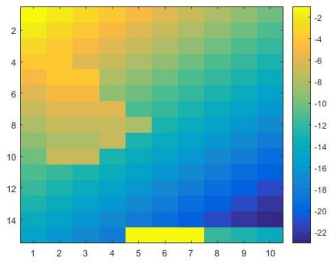
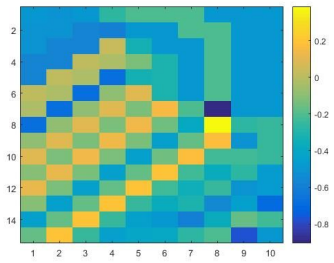
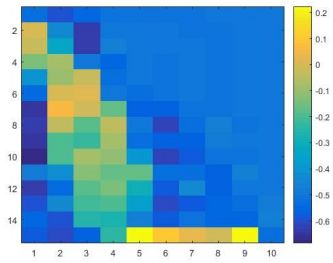
In order to further test the algorithm, we tested our algorithm in situations where the reward function was significantly less smooth. The results of this experiment are shown below.

Real Reward	Ziebart[3]	Our Algorithm
		

In the true grid, we have a uniform small negative reward of -1 in most of the grid, with two strips that have a strongly negative reward of -100. The only way around these strong negative rewards is by travelling through the small corridor generated between them. In the generated trajectories, not every expert avoided the low reward, and many took short cuts through it as a result of their suboptimality. However, because most experts took different shortcuts, the average behavior was still generally optimal.

Both algorithms successfully identify the corridor as high reward, and running value iteration on either resultant reward would not achieve egregious behavior. An interesting difference between the results, however, is that our algorithm also puts high reward on the optimal path that exists outside the corridor. In this particular case, our algorithm has an advantage, in that it assigns non-negative reward to the space (4, 2). Since Ziebart[3] does not, the resulting Value Iteration turns south too soon, and ultimately cuts through the extremely negative reward region as a result.

It's important to note as well that in this case all rewards generated are negative, with the possible exception of a handful of rewards in the corridor. If this had not been the case, then the behavior induced by value iteration on this reward function may very well have cut through the regions of extremely negative reward.

Real Reward	Ziebart[3]	Our Algorithm
		

Above figures show another test case where our algorithm is performing well.
Something we noticed among all the test cases, is that optimization of our algorithm, converges about 10 times faster than Ziebart[3].

For future work, it may be interesting to expand these tests to situations in which all experts do not have the same goals and starting positions.

Conclusion

Our algorithm proposes a change to the Maximum Entropy IRL algorithm presented in Ziebart[3], wherein we introduce a notion of consensus to overcome sub-optimality in trajectories. We find that, in the controlled case where all experts are attempting to perform the same actions on a grid with imperfect knowledge, introducing consensus is sufficient to produce more effective reward functions. Ziebart[3]This can be seen in all of the examples we have presented, wherein our algorithm outperforms the original presented by Ziebart[3], in some cases very significantly.

References

1. Subramanian , Isbell Jr, Thomaz.(2015) Exploration from Demonstration for Interactive Reinforcement Learning
2. Andrew Ng , Russell (2000). Algorithms for inverse reinforcement learning
3. Brian D. Ziebart, Andrew Maas, J.Andrew Bagnell, and Anind K. Dey (2010) .Maximum Entropy Inverse Reinforcement Learning