Sample Based Inverse Reinforcement Learning For Quantifying Human Reward Seeking / Risk Avoidant Behavior

Christopher Gagne and Daniel (Danny) Weitekamp
CS294-112

Abstract

The use of computational approaches in psychiatry has recently become popular (Montegue and Dayan 2012). As part of this trend, many studies have modeled dysfunctional behavior using reinforcement learning models with flexible parameters such as learning rate. For example, this has been done with depressed individuals, showing a reduced learning about rewards relative to controls (Huys 2013). However, often these models assume that people are aiming to maximize the same rewards - those given by the instructions or experimenter. This is unlikely the case, as we often characterize other aspects of psychopathology, such as anhedonia, as differences in an underlying reward function.

In this project, we aim to explore whether techniques from inverse reinforcement learning (IRL - Abbeel and Ng 2004) might allow us to identify and quantify differences in reward functions underlying aberrant behavior. Specifically, our aim was to measure individual differences in the relative valuation of obtaining rewards vs. avoiding threat, which is thought to be the core distinction between depression and anxiety. Therefore, we found a game that had the dual objectives of obtaining crystals and avoiding crashing into aliens or asteroids. We created both a javascript version of the game, to host on AWS and collect demonstration data from, and a python version to run the RL and IRL algorithms on.

We were unable to obtain CPHS approval from Berkeley in time to collect demonstrations from individuals with anxiety or depression. Therefore, for an intermediate goal, we generated data from different reward functions and recovered them using our IRL algorithm. We simulated data from a reward function designed to give rise to a risk seeking agent (one with high relative value for collecting crystals vs. crashing), and a second reward function designed to give rise to risk avoidant agent (one with low relative value for collection crystals vs. crashing). We used deep Q-learning (DQN - Minh 2013) with a small convolutional neural network to learn a soft-optimal policy under each reward function given images from the game as observations. The two reward functions gave rise to distinct soft-optimal behavior. We then used a sample-based version of maximum entropy IRL (Finn 2016) combined with DQN to recover the two ground truth reward functions from two ‘expert’ demonstrations. This approach worked relatively well, though aspects of the game dynamics made it difficult to infer the reward functions perfectly. We discuss these limitations and how IRL might be of use to psychology and neuroscience at the end.

Background

Reinforcement Learning in Psychiatry

Applying computational methods to psychiatry has recently become popular (Montegue and Dayan 2012). As part of this trend, studies have fit human learning behavior using simple reinforcement learning models with flexible parameters. This has been done, for example, in depressed individuals, revealing a reduced learning about rewards relative to controls (Huys 2013). However to date, most analyses have fit reinforcement learning models by adjusting algorithmic parameters, such as learning rate and exploration noise. Few, if any, have explored looking at differences in underlying reward functions. Inverse reinforcement learning is one way to do this.

Inverse Reinforcement Learning

In standard reinforcement learning (RL), the goal is to solve a markov decision process (MDP), where you are given a set of states S, actions A, and a reward function R, and the environment is governed by a transition distribution T. The solution is defined by learning an optimal policy pi(a|s) that maximizes the sum of discounted rewards received from the environment. In the inverse reinforcement learning (IRL), you are given demonstrations of state-action (s,a) pairs (or trajectories) from an optimal policy, often a human-expert. However, you are not given the reward function or policy that generated these examples. The solution consists of using these demonstrations to infer both a reward function and a corresponding policy that is optimal with respect to that reward function.

One motivation for formulating the IRL problem is that it is often easier for humans to demonstrate desirable behavior than it is for them to specify a reward function that would give rise to that behavior. For example, it might be easier for us to demonstrate reaching for a cup rather than specifying the rewards associated with various joint angles etc. that are necessary for that action. A second motivation is that if an agent learns the same reward function as humans, it might generalize to situations where no demonstrations existed or outperform humans in what they were trying to accomplish. Our motivation for using IRL, however, is to infer reward functions for the sake of interpreting them psychologically.

Solution to Inverse Reinforcement Learning

Early work on solving the IRL problem focused on discrete state-spaces and linear reward functions, where reward was defined as a linear combination of features of the states. Under these condition, Abbeel and Ng 2004 showed that solving the IRL problem amounted to matching the feature counts of the expert demonstrations to expectations from an optimal policy. However, this was only part of the solution, because many different reward functions could give rise to the same feature counts.

To constrain the solution space further, you can assume a probabilistic model for the expert demonstrations where the probability for observing different actions given states is proportional

to the exponential of their reward. Maximizing the likelihood of this model then provides a unique solution to the IRL problem. This approach became known as maximum entropy, or MaxEnt, IRL (Ziebart 2008). Maximization can be accomplished by iteratively (i) solving the MDP to find an optimal policy given a temporary reward function and (ii) updating that reward function using the mismatch in expert feature counts and expected feature counts under that current policy. Although in principle the approach is simplistic, it was only tractable in its 2008 formulation for small MDPs and tabular algorithms that allowed exact calculation of the expected feature counts.

To scale MaxEnt IRL to complex MDPs and approximate solutions, Finn 2016 introduced a sample-based approach. Unlike the original MaxEnt approach which solves the MDP completely to calculate feature expectations for each update to the reward function, the sample-based approach only incrementally improves the policy and uses samples to approximate these feature expectations for each reward update. Each iteration of the algorithm makes a small improvement to the policy with respect to the current reward function and then updates the reward function based on the mismatch between samples from that policy and the expert demonstrations.

Methods

The long-term aim of our project was to apply sample-based MaxEnt IRL (Finn 2016) to human game play in order to quantify and understand differences in how individuals make valuations. Specifically, our long-term aim was to measure individual differences in the relative valuation of obtaining rewards vs. avoiding threat, which is thought to be the core distinction between depression and anxiety. However, human data collection was not possible because the project has not yet been approved by CPHS. Therefore, our intermediate goal for this class project was to validate our IRL approach on simulated data. We generated data from different reward functions using deep Q-learning with a small convolutional neural network on our game and recovered them using our IRL algorithm.

Web version of game and data collection
To eventually measure individual differences in the relative valuation of obtaining rewards vs. avoiding threat, we borrowed a game, “crystal quest”, where the aim was to collect crystals while avoiding crashing into aliens or asteroids. We found a JavaScript version of this game, and hosted it on Elastic Beanstalk on AWS. Gameplay data (e.g. ship position/velocity, crystals locations, etc) was recorded every 100ms and stored in an AWS SQL database. Games last 60 seconds before the environment resets. We were able to record our own game-play and recreate videos of our trajectories in our gym environment. The game demo.

Game adjustments for IRL
We then adjusted the game so that it would be better for both RL and IRL. We reduced the different types of crystals and aliens, and removed the ability of the player to shoot aliens. Crucially, we also removed explicit points, and instead changed the instructions to be simply:

“obtain crystals while avoiding crashing into asteroids and aliens”. We hoped that this vague objective would encourage players to bring their own valuations into the game for how much they preferred to collect crystals relative to crashing.

In order for IRL to work well, different reward functions need to give rise to different optimal behavior. We based our reward functions on collecting crystals vs. avoiding crashes, so in order to encourage different optimal behavior, we introduced spatial correlation between crystals and asteroids, and introduced noise into the transition function (ie. actions sometimes fail to change the ship’s direction). This successfully allowed for either risky or safe behavior depending on the reward function, as will be discussed below.

Python version of game

We also implemented a corresponding gym environment to run RL and IRL on. This allowed for relatively easy testing and video recording of episodes. The game corresponded to a 35x25 grid with a variable number of asteroids and crystals and other parameters that were necessary for testing the algorithm.

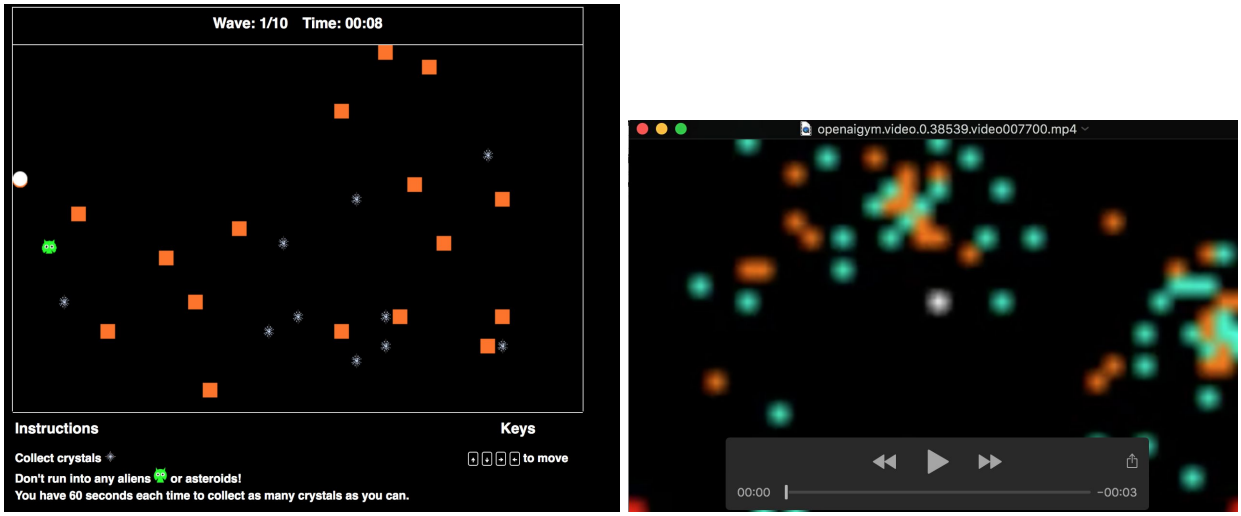

Figure 1: Web version of the game for collecting human data (left) and python version of the game for running algorithms (right). On the right, crystals are in green, asteroids are in orange, the ship is in white, and the aliens are in red (bottom corners).

Implementing DQN

To solve the MDP, we used deep Q-learning with a convolutional network using 4 consecutive 35x25x3 images of the game as observations (Mnih 2013). The motivation for using DQN was that we wanted the approach to scale to more complicated games with unknown dynamics and we thought off-policy learning might have some benefits for IRL.

We used various sized convolutional networks to solve the game. The largest network contained 3 convolutional layers (32,64,64 filters) and a hidden layer of 512 units. The smallest

network that performed well used 2 convolutional layers (8,16 filters) and a 64-unit hidden layer. The final simulations reported here use this small network. A crucial aspect of getting DQN to work quickly and well on this game was to reorient the images from the game so that they were in a ship-centric reference frame, as opposed to absolute image of the game.

In order to run a MaxEnt IRL algorithm, we needed to use a soft-optimality version of Q-learning (Haarnoja 2017). This required two adjustments to the standard DQN. The first is to use Boltzmann exploration instead of epsilon greedy - that is to choose actions in proportion to the exponential of their Q-value:

$$\pi(s|a)\ propto\ exp(Q(s,a))$$

The second adjustment was to do Boltzmann backups, using the softmax instead of the max for the next state's values:

$$V(s') = log \int_a exp(Q(s',a'))da$$

To implement this target value calculation, we first used Tensorflow's built in function reduce_logsumexp(), which avoids underflow and overflow. However, we still found that using this estimate for target value increased all Q-values perpetually. They increased to from around 1 to >100, which caused issues with overflow in the softmax exploration. In the final simulations, we ended up using regular max backups which kept Q-values <10.

Implementing sample-based MaxEnt Inverse Reinforcement Learning

For the IRL, we implemented our own version of the sample-based MaxEnt inverse reinforcement learning (Finn 2016). The overall procedure can be seen in Algorithm 1. The Q-function used the same convolutional network and the Q-learning updates were the same as just described. The reward function was a linear function of the number of crystals collected, asteroids avoided and aliens avoided:

$$R(s,a) = \phi_1 \times crystals + \phi_2 \times asteroid\ collisions + \phi_3 \times alien\ crashes$$

The loss function for IRL was taken from (Finn 2016 and IRL slides):

$$grad\ L = \sum_i^N grad\ r_\psi(\tau_i) - \frac{1}{\sum_j w_j} \sum_j^M w_j\ grad\ r_\psi(\tau_j)$$

The importance weights used for the function were:

$$w_j = \frac{exp(\sum_t r_\psi(s_t,a_t))}{\prod_t \pi(a_t|s_t)}$$

These should correct for the fact that our sample estimate for the feature expectations are coming from a non-optimal policy during each iteration of the IRL algorithm (ie. they are from the wrong distribution). As the policy becomes closer to optimal, these weights should approach 1. For our implementation, the product of the probabilities underflowed for most trajectories. So, we took the log of this expression, as a hack, to use as weights.

Algorithm 1 Sample-based IRL with DQN and Linear Reward

Start with human samples D_h and feature counts F_h
Initialize reward function $r_\phi(s, a)$
Initialize q function $Q_\theta(s, a)$
for $i = 1$ to I: **do**
    Use π_i to sample S times from env with r_i; store in replay buffer
    Calculate features counts F_s and importance weights w
    Update $r_\phi(s, a)$ using K steps with samples from F_h, F_s
    Update $Q_\theta(s, a)$ using J steps with samples from replay buffer
    Update target $Q_\theta(s, a)$ and π

Simulations to test IRL implementation

To test our IRL, we generated data from known reward functions using DQN and then ran the IRL algorithm on the feature counts obtained to recover them.

We specified two different known reward functions:

$$\phi_{risky} = \{1, -0.1, -0.1\}$$
$$\phi_{safe} = \{1, -10, -10\}$$

The first reward function values obtaining crystals more highly than it devalues crashing. The optimal policy for this should be to collect crystals in high density areas where there is also the chance of crashing. The second reward function has a much higher cost for crashes, so its optimal policy should give low probability to these high density areas.

We ran DQN to find the soft-optimal policy with respect to each reward function. The MDP was specified to have 40 crystals, 30 asteroids, 15% stochasticity in actions, and 2 high-density areas of crystals and asteroids. An example with these specifications can be seen in Figure 1. The DQN algorithm used a replay buffer of 500,000 and updated its target network every 5000 Q-value updates. Optimization was performed using Adam with a learning rate of 0.004. The two forward runs took approximately 1hr each.

For each of the two forward runs of DQN, we recorded feature counts, number of crystals and collisions, per episode. We used the last 1000 episodes, where the policies seemed to be stable, as ‘expert demonstrations’ for the IRL.

Each IRL iteration consisted first of taking S=20,000 new samples from the game and adding them to the replay buffer. Feature counts and importance weights used to update the reward function we calculated from these new 2000 episodes. Next, K=200 gradient steps were taken with mini-batches of 25 to update the reward function. Then, J=4000 randomly selected (s,a,s’,r) experiences were sampled from the replay buffer. Finally, the target Q-function was replaced with the updated Q-function before the next IRL iteration. IRL was repeated for 200 iterations taking approximately 2.5hrs.

Code:
Can be found on Github: https://github.com/crgagne/irl_project

Results

Simulation - DQN policies from two different reward functions
DQN was run on ϕ_{risky} and ϕ_{safe}. These runs achieved a steady policy around 8000 episodes. The ϕ_{risky} parameters yielded an average of 20 crystals, 2.1 asteroid collisions and 1 alien collision which can be seen in Figure 2. The ϕ_{safe} parameters yielded an average of 17 crystals, 1 asteroid collision and 1 alien collisions.

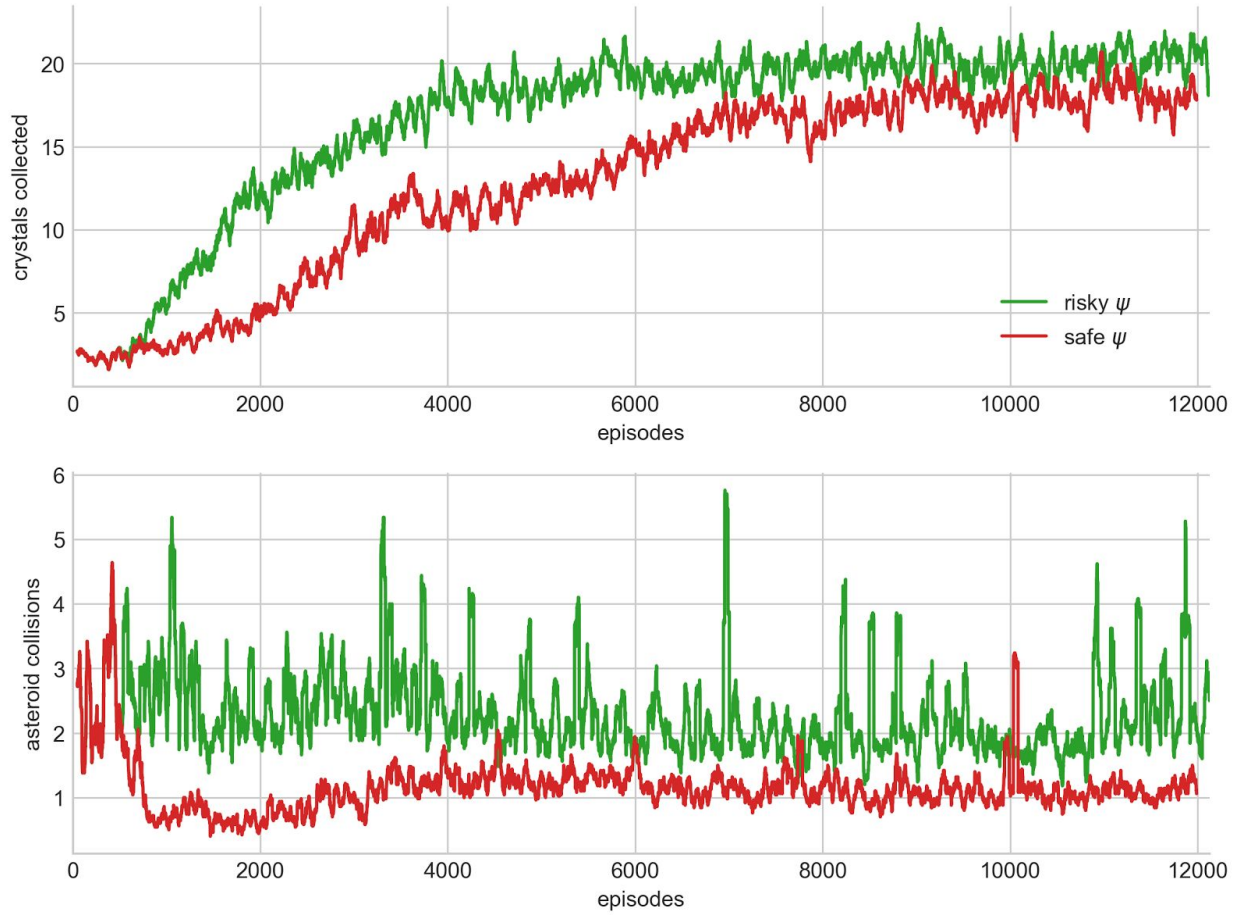

Figure 2: Feature counts from running DQN on our game under two different reward functions (risky and safe).

Simulation - recovery of reward function parameters with IRL

For each of the two IRL runs, we started with random parameter values for ϕ .

After several iterations of the algorithm, the estimates for ϕ_1 (for crystals collected) were extremely high due to the large mismatch between crystals obtained by expert and the number obtained by the DQN policies at that point. Eventually, these estimates decreased to around 1, the true reward weight.

The estimates for ϕ_2 (for asteroid collisions) started positive and quickly became negative. The estimate for the risky reward function stayed around the true value of -0.1, whereas the estimate for safe reward function became more negative. However, it never quite reaching the true value of -10. This is likely because -2 (the estimate) and -10, both give rise to an average of 1 asteroid collision per episode.

Although they are not plotted, we obtained final estimates for ϕ_3 (for alien collisions) of 0.79 and 0.43. These were not close to the true parameters, but we did not expect accurate estimates of these given that both forward runs had the same number of alien collisions.

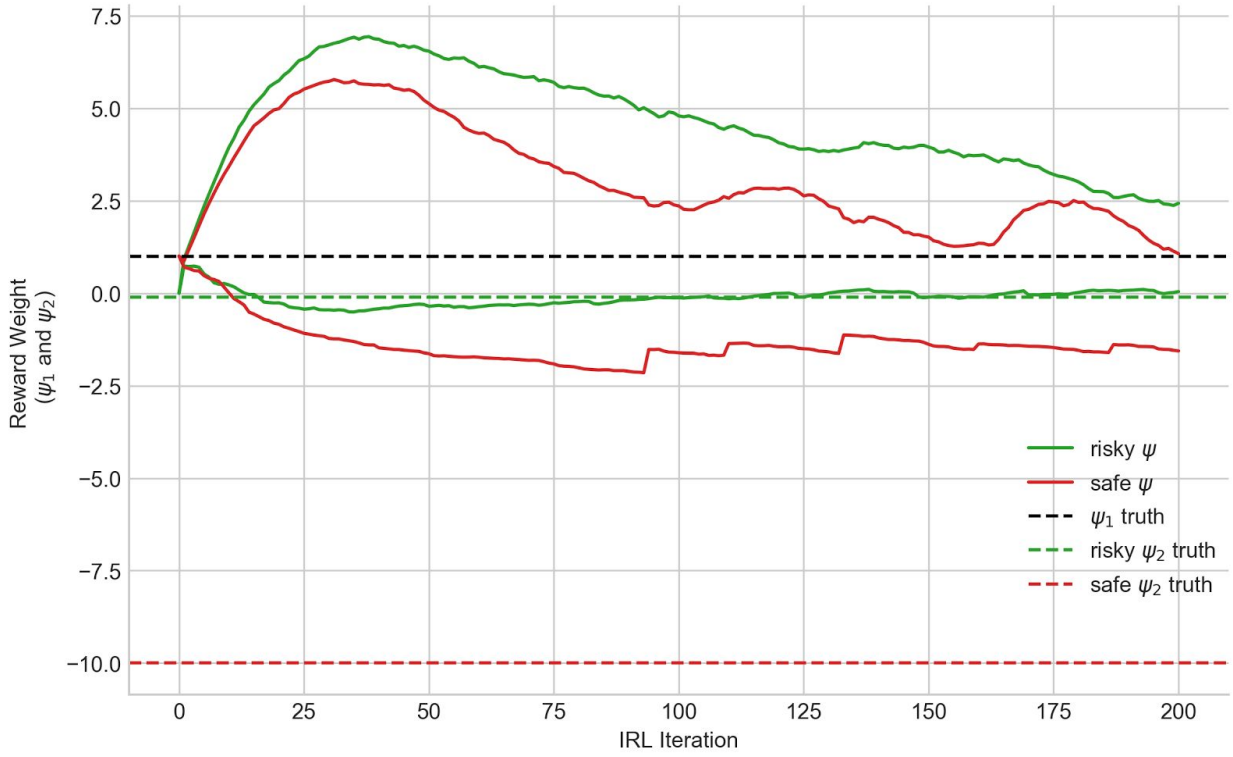

Figure 3: Results of IRL simulation to recover two different reward functions (risky and safe). The top solid lines correspond to estimates for ϕ_1, whereas the bottom solid lines correspond to estimates for ϕ_2. The dashed lines correspond the the true parameters.

Limitations

Hyperparameter tuning

The time it took to get reasonable behavior in our game using DQN with convolutional nets was very dependent on these hyper parameters: size of the reward (e.g. 10 vs. 1), the learning rate (0.001 v 0.0001), the amount and type (boltzmann or epsilon greedy) of exploration, and the number of time-steps in the environment. These hyperparameters had to be roughly correct before IRL could be conducted. The IRL algorithm itself had a number of hyper-parameters like how frequently to update the reward function vs. the q-function. Thus, our method is far from being out-of-the-box. However, we hope to make improvements to this in the future.

Limited identifiability of reward function in our game

One issue with the simple reward function that we chose was that differences in it led to very similar optimal behavior in terms of feature counts. One way around this was to make the

reward function dependent on more features of the environment to force separate optimal behavior. Another approach, which we took, was to tweak the game (stochastic actions and correlated rewards/punishments) until differences in behavior naturally arose from the simple reward function. Before we use our game to collect actual human demonstrations, we will adjust our game further so that differences in reward function yield even more disparate behavior.

Discussion: the use of IRL for Psychology and Neuroscience

What does the reward function buy us above feature counts?
In our game, differences in reward function parameters lead to obvious differences in feature counts. If we had been analyzing human data, couldn't we just have used the relative feature to see how people's valuations differed? What does estimating a reward function add?

We think there are two related benefits to estimating a reward function. The first is that the feature counts provide only a rough estimate of the relative value of features (and perhaps only an ordinal ranking), while the reward function provides a more precise quantification. For example, observing 80 crystals and 10 crashes could imply a relative valuation of 8:1 or 1.5:1, depending on the dynamics of the game (e.g. if the first 60 crystals could be obtained with no risk of crashing). The second related reason is that in other environments, differences in reward function may give rise to differences in feature counts through long, convoluted trajectories. These differences might only really be inferrable through IRL, because it solves the forward sequential decision problem.

What do we choose as reward functions?
Another interesting consideration for psychologists using IRL is what features should we include in our reward functions? In our project, we considered adding distance to asteroids and distance to crystals as features in the reward function to get more dissociable optimal behaviors. However, keeping a distance from asteroids seems like a natural consequence of a negative reward associated with crashing. So it seemed redundant as a psychological model to say that there is a negative cost of being near asteroids in addition to that associated with the risk of crashing. Instead, we added more stochasticity into the actions so that distance to asteroids would arise as a natural consequence of the negative value of crashing.

More generally, one route could be exclude features in IRL that we don't hypothesize have a explicit psychological or neural representation and only use linear models. This could potentially allow IRL to pick up signatures of explicitly represented neural quantities from complex behavior. However, this route is limited in how well it might match or predict behavior because IRL only allows us to match behavior on these features. The other route, however, is to consider IRL reward functions purely as concise descriptions of behavior, without any ontological status, in the way that economists treat utility. We could then use nonlinear reward functions and include any features that might predict behavior well.

References:

Montague, P. R., Dolan, R. J., Friston, K. J., & Dayan, P. (2012). Computational psychiatry. Trends in cognitive sciences, 16(1), 72-80.

Huys, Quentin JM, et al. "Mapping anhedonia onto reinforcement learning: a behavioural meta-analysis." Biology of mood & anxiety disorders 3.1 (2013): 12.

Abbeel, Pieter, and Andrew Y. Ng. "Apprenticeship learning via inverse reinforcement learning." Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.

Ziebart, Brian D., et al. "Maximum Entropy Inverse Reinforcement Learning." AAAI. Vol. 8. 2008.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Finn, Chelsea, Sergey Levine, and Pieter Abbeel. "Guided cost learning: Deep inverse optimal control via policy optimization." International Conference on Machine Learning. 2016.

Haarnoja, Tuomas, et al. "Reinforcement Learning with Deep Energy-Based Policies." arXiv preprint arXiv:1702.08165 (2017).