Inverse Reinforcement Learning for Cell Shaping

Vikramank Singh

August 5, 2019

1 Apprenticeship learning using Inverse RL

Introduction. Reinforcement learning (RL) is is the very basic and most intuitive form of trial and error learning. Often referred to as learning by exploration, that is by taking random actions initially and then slowly figuring out the actions which lead to the desired motion. In any RL setting, to make the agent perform different behaviors, it is the reward structure that one must modify/exploit. But assume we only have the knowledge of the behavior of the expert with us, then how do we estimate the reward structure given a particular behavior in the environment? This is the problem of **Inverse Reinforcement Learning (IRL)**, where given the optimal/sub-optimal expert policy, we wish to determine the underlying reward structure.

Setup. In our case, we assume the actions taken by the humans are sub-optimal and we need the RL agent to take better actions. **Apprenticeship Learning** via IRL will try to inder the goal of a teacher. It will learn a reward function from observations, which can then be used for reinforcement learning. If it discovers that the goal is to hit a nail with a hammer, it will ignore blinks and scratches from the teacher, as they are irrelevant to the goal.

Agent. An agent is the algorithm which will make the decisions given a set of observations.

Sensors. An agent will be equipped with sensing capabilities to collect raw data (409 in our case).

State Space. The state of the agent consists of 409 observable features.

Rewards. The reward after every decision is calculated as a weighted linear combination of the feature values observed in that frame. Here the reward r_t in the t^{th} frame, is calculated by the dot product of the weight vector w with the vector of feature values in t^{th} frame, that is the state vector ϕ_t. Such that $r_t = w^T * \phi_t$.

Inverse RL.

- The **features** or the **basis functions** ϕ_i which are basically observable in the state. We define $\phi(s_t)$ to be the sum of all the feature expectations ϕ_i such that: $phi(s_t) = \phi_1 + \phi_2 + ...\phi_n$.
- **Rewards** r_t - linear combination of these feature values observed at each state s_t. $r(s, a, s') = w_1\phi_1 + w_2\phi_2 + ... + w_n\phi_n = w^T * \phi(s_t)$.
- **Feature expectations** $\mu(\pi)$ of a policy π is the sum of discounted feature values $\phi(s_t)$.
  $\mu(\pi) = \sum_{t=0}^{\infty} \gamma^t \phi(s_t)$.
  The feature expectations of a policy are independent of weights, they only depend on the state visited during the run (according to the policy) and on the discount factor γ a number between 0 and 1 (e.g. 0.9 in our case). To obtain the feature expectations of a policy we have to execute the policy in real time with the agent and record the states visited and the feature values obtained.

Initialization.

- Expert policy feature expectations or the **expert's feature expectations** $\mu(\pi_E)$ are obtained by the actions that are taken according to the expert behavior. We basically execute this policy and get the feature expectations as we do with any other policy. The expert feature expectations are given to the IRL algorithm to find the weights such that the reward funciton corresponding to the weights resemebles the underlying reward function that the expert is trying to maximize (in usual RL language).
- **Random policy feature expectations** - execute a random policy and use the feature expectations obtained to initialize IRL.

The Algorithm.

- Maintain a list of the policy feature expectations that we obtain after every iteration.
- At the very start we have only π^1, the random policy feature expectations.
- Find the first set of weights of w^1 by convex optimization, the problem is similar to an SVM classifier which tries to give a +1 label to the expert feature expec. and a -1 label to all the other policy feature expecs -

  $$min||w||_2^2$$

  such that,

  $$w^T\mu_E >= 1$$
  $$-w^T\mu_{\pi(i)} >= 1$$

  Termination condition:

  $$w^T(\mu_E - \mu_{\pi(i)} <= \epsilon$$

- Now, once we get the weights after one iteration of optimization, that is once we get a new reward function, we have to learn the policy which this reward function gives rise to. This is same as saying, find a policy that tries to maximize this obtained reward function. To find this new policy we have to train the Reinforcement Learning algorithm with this new reward function, and train it until the Q-values converge, to get a proper estimate of the policy.
- After we have learned a new policy we have to test this policy online, in order to get the feature expectations corresponding to this new policy. Then we add these new feature expectaitons to our list of feature expectations and carry on with out IRL algorithm's next iteration until convergence.

References.

- Andrew Ng and Stuart Russell, 2000 - Algorithms for Inverse Reinforcement Learning
- Pieter Abbeel and Andrew Ng, 2004 - Apprenticeship Learning via Inverse Reinforcement Learning