Multi-Agent Systems of Inverse Reinforcement Learners in Complex Games

Dave Mobley
University of Kentucky, dave.mobley@uky.edu

Abstract

Reinforcement Learning (RL) allows an agent to discover a suitable policy to achieve a goal. However, interesting problems for RL become complex extremely fast, as a function of the number of features that compose the state space. The proposed research is to decompose a core problem into tasks with only the features required to solve the task. The core agent then uses the reward for the task, without knowing the underlying task model. This paper discusses task-based RL and Inverse Reinforcement Learning to train the tasks.

1 Introduction

We want to use machine intelligence to solve real-world problems such as driving a car efficiently and safely, or managing a retail business. Real-world problems exhibit a few defining criteria that make them challenging, such as resource and task management. Knowing how much fuel is in a car or how to deal with approaching sirens are important to the overall goal of driving. For a restaurant you worry about how much to staff tonight, or what to do if a shipment of arrives late, all while trying to keep the store operating. Each of these tasks requires many features that describe the problem and state of the system. As the number of features grows, the problem becomes nearly intractable. Furthermore, the agent may not know how to quantify a reward for each task nor for the ultimate goal. Do we value getting to the next destination as fast as possible regardless of safety or with absolute safety but at a walking pace? Is the purpose of a restaurant about making a profit or employing people in the community? Another set of complex problems is computer games. Games exhibit overarching goals with many small subproblems. I am exploring tiered Reinforcement Learning techniques as a starting point for satisfactory baseline results when learning to play role-playing games. I use existing expert policies as a starting point to learn how to play and extrapolate ideal goals and rewards.

1.1 Modeling Problems as MDPs

Problems as complex as driving a car or managing a restaurant can be modeled as *Markov Decision Processes (MDPs)*. An MDP is a collection of states where each state exhibits the Markov Property, an action can be chosen to transition states, and a reward is given on state change. The *Markov Property* says that future states can be determined from the current state and chosen actions, independent of prior states. Transitions between states are determined by actions. Each transition provides a reward. A policy is a mapping from states to actions or distributions over actions. The value for a policy can be estimated as the current reward for an action plus the sum of expected rewards for following the rest of the policy.

1.2 Reinforcement Learning

In Reinforcement Learning (RL), the goal is to devise a policy based on the current model of the problem, given state space S and action space A and rewards R associated with those state/action transitions. Dynamic programming techniques, e.g., value iteration and policy iteration learn through step-wise improvements on a given policy. Off-policy algorithms such as Q-Learning update Q values, the state-action pair values. These algorithms are off-policy because they allow the learner to learn while following another policy, even a random policy [Sutton and Barto, 2016], updating expected rewards from an action taken from a given state. The result is that a set of known rewards can be mapped to a policy.

When the rewards are not known, a policy cannot be calculated. Exploring the state space and actions and determining expected rewards is called Q-learning. In complex problems, there may be no way of evaluating a reward for a single action, but having a working policy and mapping that policy to rewards can give an approximation of the rewards. This technique is called *Inverse Reinforcement Learning (IRL).*

1.3 Big Problems Means Big Data

Complex problems require many features to define the myriad possible states. When the number of features grows, the problem size grows exponentially. Another source of complexity is the "Curse of Dimensionality:" more features used implies more data required to efficiently train a system. If you have a training set that trains each state only once, then adding one features doubles the size of the training set required since the extra feature doubles the state space. Thus, the data set must grow exponentially with the number of features. Most interesting problems exhibit this curse; optimization quickly becomes intractable. One cannot come up with enough data

(or time) to begin to significantly test a problem with a large number of variables. However, there are ways to get around the sparsity of data.

1.4 Learning by Example

There are tasks, like driving a car or managing a restaurant, that are considered hard, even by human standards. To look at teaching a computer to understand the goals, and not just mimic a teacher, we must use a system that can reward finding effective policies and improving upon them [Abbeel and Ng, 2004]. Abbeel has shown that by using IRL, computers can be taught through apprenticeship, using a teacher and inferring their goals. Examples currently demonstrated are auto driving in a simulator, and teaching a system to perform aerobatics with a helicopter [Abbeel *et al.*, 2007]. Many complex problems can be decomposed into smaller problems. Though a system can be modeled to learn from all features composing each state, subproblems, like braking a car, can be treated as an independent action with a probability and reward. The braking problem itself can be modeled as an MDP and trained through RL or IRL as well.

Using an approach that breaks out features into subproblems should ease the need for data to train the system. Subproblems will only need enough data to sufficiently train them, and results can be funneled back to the primary learner. Splitting problems and training the individual pieces is typically faster than trying to train the whole of the problem monolithically. Finally, the outcome for complex problems shouldn't need to be optimal, but rather a system that starts at human level competency as a lower boundary and can be continually trained to improve over time. In the next section, I present proposed benchmark problems, then describe my proposed research.

2 Description of Games

Role Playing Games (RPGs) are some of the hardest games for players to learn to play well. They have a set of overarching goals, but have many smaller tasks that contribute to different parts of this goal. Overarching goals may be as simple as completing the game, not dying, or improving the player avatar. Tasks can include fighting in combat, performing a sub-game to receive rewards, or just walking around collecting goodies. In such a game it's hard to understand and quantize complex, intertwined rewards. Another disadvantage of these games is that players might not exhibit the policy choices for rare game states.

I am looking at two types of games. The first is a text-based RPG, as the states and actions are easy to teach to a system. This type of RPG, called a MUD (Multi-User Dungeon) is a gridworld-like game with a collection of quests alongside primary goals of staying alive and improving a character. I will use expert trainers to teach individual learners how to perform complex actions such as combat, map navigation, and healing, while allowing a higher level learner to focus on bigger goals such as deciding the value of having a combat (chance of death vs. growth), quest completion, and avatar advancement trajectory.

The second type of game is a graphical RPG, Pirate101, a first-person RPG produced by KingsIsle. It has similar tasks of combat, map navigation, healing, and quests. Being graphical, the environment and possible actions are more varied and nuanced. Also the world-space is much larger, making it similar to the text RPG, but with more complexity.

3 Novelty of Research

RPGs can be modeled as factored MDPs. There is a set of variables that defines the current state of the game, and a set of available actions. To teach a system to play an RPG, two levels of learning must occur. The top level, which uses IRL, learns the general goals of the expert and uses those rewards to train the game playing agent. For sub-tasks, learners are taught using a combination of learner types depending on the sub-task. For example, a Deep IRL learner trains to handle combat by an expert. Another type of learner can learn to navigate using a Deep Q Network, as rewards can be created without an expert to teach the system. The fact that the subtasks are played in a competitive space adds complexity. To compensate for that, adversary actions also must be considered as part of the state/action transition. Littman [Littman, 1994] suggests a minimax strategy using Q-learning that helps account for the action of the other participant.

To properly model complex problems, I'm looking at using Hierarchical Task Networks (HTNs) in combination with IRL tasks to show that the MDPs can be decomposed into smaller tasks and solved independently from the main problem. The results, and expected reward, should flow up to the primary learner so it doesn't have to manage all of the features for each sub-task as part of its learning.

I am looking at ways to teach learners to perform as well as humans because they are trained by human experts. I assume that this training forms a lower bound on the ability for the learners, meaning that, though they may not be optimal, they are at least satisfactory in performance. Training systems using multiple experts gives a sample set that provides expectations of rewards for different aspects of the system, but also allows the establishment of what the goals should be based on consensus. This new interpolated group goal is the starting point that a new reinforcement learner can be trained. By cycling between expert training and estimating a more refined final goal, we expect to monotonically improve the rewards estimates and goals, ultimately converging on knowledge of goals and rewards.

References

[Abbeel and Ng, 2004] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[Abbeel *et al.*, 2007] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1, 2007.

[Littman, 1994] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. ICML*, volume 157, pages 157–163, 1994.

[Sutton and Barto, 2016] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. A Bradford book. Cambridge University Press, 2016.