Towards Behavior-Aware Model Learning from Human-Generated Trajectories

Robert Loftin
North Carolina State University
rtloftin@ncsu.edu

James MacGlashan
Brown University
james_macglashan@brown.edu

Bei Peng
Washington State University
bei.peng@wsu.edu

Matthew E. Taylor
Washington State University
taylorm@eecs.wsu.edu

Michael L. Littman
Brown University
mlittman@brown.edu

David L. Roberts
North Carolina State University
robertsd@csc.ncsu.edu

Abstract

Inverse reinforcement learning algorithms recover an unknown reward function for a Markov decision process, based on observations of user behaviors that optimize this reward function. Here we consider the complementary problem of learning the unknown transition dynamics of an MDP based on such observations. We describe the *behavior-aware modeling* (BAM) algorithm, which learns models of transition dynamics from user generated state-action trajectories. BAM makes assumptions about how users select their actions that are similar to those used in inverse reinforcement learning, and searches for a model that maximizes the probability of the observed actions. The BAM algorithm is based on *policy gradient* algorithms, essentially reversing the roles of the policy and transition distribution in those algorithms. As a result, BAM is highly flexible, and can be applied to continuous state spaces using a wide variety of model representations. In this preliminary work, we discuss why the model learning problem is interesting, describe algorithms to solve this problem, and discuss directions for future work.

Introduction

Inverse reinforcement learning (IRL) can be described as the problem of learning an unknown reward function of a Markov decision process based on observations of a behavior that is optimal or nearly optimal under that reward function (Ng and Russell 2000). In this work, we consider a complementary problem; identifying the transition dynamics of an MDP based on observations of user behavior, under known reward functions. We show that by making similar assumptions to IRL about how a user selects actions, we can learn unknown parameters of a transition model based on their behavior. Our goal in addressing this problem is to allow an agent to learn a model of its environment based on human-generated data, and use that model to choose solutions to tasks we wish the agent to perform.

As an example, we consider the problem of a robot learning to navigate in a two dimensional environment (e.g., an office) similar to Ziebart et al. (2009). If we represent this problem as a Markov decision process, then an unknown map of the environment corresponds to incomplete knowledge of the transition dynamics of the MDP. Instead of having the robot map the environment on its own, we assume that the robot is under the control of a human operator who is performing some task, and not actively helping the robot build its map. The robot's sensors may allow it to build a partial map, but only of areas that it passes through under user control. The user is likely to avoid obstacles and dead ends, and so the robot may never observe them.

Without considering why a user avoids certain paths, the robot's map may lead it to follow impossible trajectories. For example, a pair of open doors may suggest that a shorter path exists between two points than the one followed by the user, but the fact that the user takes a longer path suggests that there is a hidden obstacle (e.g., the doors lead to different rooms) that would make a path through those rooms impossible, or at least longer than the alternative. By combining direct observations of the environment with indirect knowledge based on user behavior, we can construct a more complete map than could be learned from direct observations alone, allowing for more accurate path planning.

We describe the *behavior-aware modeling* (BAM) algorithm, which learns transition models from user-generated data. As in Ziebart et al. (2008), BAM computes a maximum likelihood estimate of the model parameters via gradient ascent. To compute the likelihood gradient with respect to the model parameters, we take an approach similar to *policy gradient* methods (Peters and Schaal 2006), but treating the action distribution as being fixed. While this work is preliminary, it derives a framework for learning transition models based on user-generated data, and describes model-learning algorithms for discrete and continuous domains. In addition, we experimentally validate that the BAM algorithm can incorporate indirect knowledge, learning transition models based on synthetic data in a discrete, 2D navigation domain.

Related Work

Inverse reinforcement learning is generally used either to have an agent learn to perform a task by observing a human demonstrating that behavior, or to learn a global (task independent) cost function encoding human preferences over solutions to different tasks. Abbeel and Ng (2004) look specifically at replicating a human's behavior by learning the reward function the user is optimizing. Ziebart et al. (2008) learn a global cost function over road networks, allowing for more accurate route planning, while Ziebart et al. (2009) learn a cost function for predicting the motion of

pedestrians. Similarly, our work focuses on learning a task-independent model, which can be used to find solutions to any tasks the agent might be asked to perform.

IRL methods differ in the assumptions they make about how the actions are selected. Ng and Russell (2000) and Abbeel and Ng (2004) assume that the observed policy is optimal under the reward function, while Ramachandran and Amir (2007) assume that users sample actions from a softmax distribution based on their expected return given that subsequent actions are optimal. Ziebart et al. (2008) use a less greedy assumption such that the total reward roughly follows a softmax distribution. We use these same assumptions about how user actions are selected to infer the model.

Motivation

The transition dynamics of an environment are often much more complex than the reward function, and so learning an accurate transition model may require more data than learning a reward function. We therefore need to ask what advantage there is to learning an explicit transition model. In cases such as 2D mapping where transition dynamics are only partially known, one could encode the unknown information as a global cost function (e.g., assign high cost to blocked states). The limitation of this approach is apparent when we want to plan under different reward functions. The cost function may simply overwhelm the reward function, leading to undesirable behaviors such as remaining in the same location instead of passing through a high-cost area to reach a goal. Ziebart (2010) overcome this problem by imposing task-specific constraints on the behavior generated by the agent, such as requiring that all trajectories end in the goal state. When such constraints cannot be easily described, do not exist, or make planning difficult, a global cost function alone may not be able to capture the way in which each task-specific reward function affects the choice of policy.

Another potential advantage of learning transition models is that we generally have more prior information about the model than we do about the reward function. In IRL, the reward function is treated as being internal to the user, and thus can only be recovered from the user's actions. The transition dynamics, on the other hand, are observable directly from transition data, without considering the user's policy. We may also have additional information, such as sensor data indicating the presence of obstacles. Finally, we note that IRL generally requires that the transition dynamics are known or have already been learned. BAM can therefore complement IRL by improving the quality of the learned model.

Preliminaries

We represent the environment in which the agent operates as a Markov Decision Process (MDP) defined by $\{S, A, T\}$. S and A are the state and action spaces, while $T(s, a, s')$ is the (partially unknown) state transition distribution (or density for continuous state spaces). A single task is defined by a reward function R, and we wish to learn an approximation of T which will allow the agent to select a policy for any given R. This model is learned from training data D generated by a user who we assume knows the true transition dynamics T. Each of the k entries $D_i \in D$ consists of a state-action trajectory $\zeta_i = \{s_1, a_1 \dots s_{\tau_i}, a_{\tau_i}, s_{\tau_i+1}\}$, and a complete reward function R_i and discount factor γ_i under which that trajectory was generated. In our experiments, this reward function takes a value of 1 at the known goal state $s_i^\star$, and 0 everywhere else. We are given a parametric space of transition models $T_\theta(s, a, s')$, and attempt to maximize the likelihood of the parameters θ given D, that is

$$\theta^\star = \arg\max_\theta p(D|T = T_\theta) \tag{1}$$

The choice of model spaces may encode a significant amount of domain knowledge. In the case of 2D navigation, we assume that the dynamics in empty space are known, and what needs to be learned are the locations of unobserved obstacles. We consider two different assumptions about how actions are selected by the user. The first is based on the assumption used in bayesian inverse reinforcement learning (Ramachandran and Amir 2007). In that work, it is assumed that actions are selected according to

$$\pi(s, a) = p(a|s) \propto e^{\beta Q^\star(s,a)}, \tag{2}$$

where $Q^\star(s, a)$ is the expected return starting in state s taking action a, and then following the optimal policy $\pi^\star$. The alternative assumption, based on the maximum-entropy IRL approach (Ziebart et al. 2008), is that

$$\pi(s, a) \propto e^{\beta Q^{\approx}(s,a)}, \tag{3}$$

where $Q^{\approx}(s, a)$ is the solution to the soft Bellman equations,

$$Q^{\approx}(s, a) = R(s) + \gamma \sum_{s' \in S} T(s, a, s') V^{\approx}(s') \tag{4}$$

$$V^{\approx}(s) = \sum_{a \in A} Q^{\approx}(s, a) \frac{1}{Z(s)} e^{\beta Q^{\approx}(s,a)}. \tag{5}$$

Where $Z(s)$ is a normalization term for the action distribution for s. When the transition dynamics are deterministic, this assumption is equivalent to assuming that state-action trajectories have a probability proportional to the exponent of their total reward. Under either of these assumptions, BAM finds a maximum likelihood estimate of the transition model parameters via gradient ascent.

Behavior-Aware Modeling

The log-likelihood of θ given trajectory data D is $L(\theta; D) = \sum_{i=1}^{k} L(\theta; \zeta_i, R_i, \gamma_i)$. The gradient of $L(\theta; \zeta, R, \gamma)$ is

$$\begin{aligned} \nabla_\theta L(\theta; \zeta, R, \gamma) = \sum_{i=1}^{\tau} [&\beta \nabla_\theta Q(s_i, a_i) \\ &- \beta \nabla_\theta V(s_i) + \nabla_\theta \ln T_\theta(s_i, a_i, s_{i+1})], \end{aligned} \tag{6}$$

where Q can be either $Q^\star$ or $Q^{\approx}$. This gradient combines the probability of each observed transition with the probability of each action. Since we assume the user chooses their actions based on the expected return under the true model, we have to consider how changes to the model will affect their choice of actions. For both $Q^\star$ and $Q^{\approx}$

$$\begin{aligned} \nabla_\theta Q(s, a) = \gamma \mathrm{E}_{s' \sim T_\theta(s,a,*)} \big[&\nabla_\theta V(s') \\ &+ V(s') \nabla_\theta \ln T_\theta(s, a, s')\big]. \end{aligned} \tag{7}$$

Where actions are assumed to be selected as in Equation 3, we have $Q = Q^{\approx}$, the gradient of which is

$$\nabla_\theta V^{\approx}(s) = \mathrm{E}_{a\sim\pi^{\approx}(s,*)}\left[\nabla_\theta Q^{\approx}(s,a) + \beta\nabla_\theta Q^{\approx}(s,a)\left(Q^{\approx}(s,a) - V^{\approx}(s)\right)\right], \tag{8}$$

with $\pi^{\approx}$ defined such that the expected return $Q^{\pi^{\approx}}$ following $\pi^{\approx}$ is equal to $Q^{\approx}$, that is $\pi^{\approx}(s,a) \propto \exp\{\beta Q^{\approx}(s,a)\}$. The exact gradient takes into account the fact that the change in the transition model also leads to a change in the action distribution $\pi(s,a)$. Maximum entropy IRL uses the approximation that the action distribution is fixed, in which case

$$\nabla_\theta V^{\approx}(s) = \mathrm{E}_{a\sim\pi^{\approx}(s,*)}\left[\nabla_\theta Q^{\approx}(s,a)\right], \tag{9}$$

which will generally be of smaller magnitude. We use this approximation in our experiments. Assuming that actions are selected based on the optimal value function, this fixed-policy approximation is required, as the policy is not differentiable. Under that assumption, we have

$$\nabla_\theta V^{\star}(s) = \mathrm{E}_{a\sim\pi^{\star}(s,*)}\left[\nabla_\theta Q^{\star}(s,a)\right]. \tag{10}$$

For discrete state spaces, we can compute Q and V, using value iteration, and then compute the gradients via dynamic programming. The BAM algorithm follows the gradient in Equation 6, which combines the action probabilities with the probabilities of the observed transitions.

Continuous State Spaces To extend the BAM approach to continuous state spaces, we note that the gradient is an expectation under the state transition and action distributions. We assume that the user selects actions based on a τ-step approximation of Q. For a fixed policy π, we have

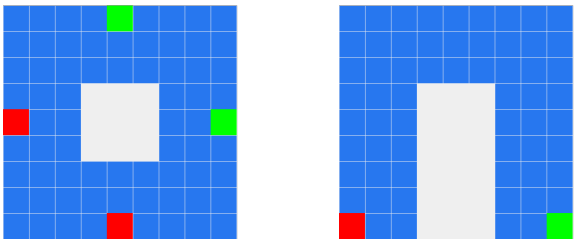
$$\nabla_\theta Q^{\pi}(s,a) = \mathrm{E}\left[\sum_{i=0}^{\tau-1}\nabla_\theta \ln T_\theta(s_i,a_i,s_{i+1})\sum_{j=i}^{n}\gamma^i R(s_j)\right], \tag{11}$$

where the expectation is over trajectories starting with state s and action a. The gradient can therefore be estimated by generating sample trajectories under the current model T_θ. Generating these samples requires first computing the policy π, which in the case of $Q = Q^\star$ is simply the optimal policy, which can itself be found via a policy gradient algorithm.

The case where $Q = Q^{\approx}$ is more difficult, but we can find an approximation of $\pi^{\approx}$ using a *variational* form of policy gradient learning. We define a policy π_λ (with parameters λ) such that $\pi_\lambda(s,a) \propto \exp\{q_\lambda(s,a)\}$, with q_λ satisfying $\sum_{a\in A}\pi_\lambda(s,a)q_\lambda(s,a) = 0, \forall s \in S, \lambda$, such that q_λ is an advantage function. We then seek λ which minimizes the KL divergence between the trajectory distributions $p(\zeta|T_\theta,\pi_\lambda)$ and $p(\zeta|T_\theta,\pi^{\approx})$. For the second distribution we can substitute the approximation from Ziebart et al. (2008) $p^{\approx}(\zeta|T_\theta) \propto \exp\{R(\zeta)\}\prod_{i=0}^{\tau-1}T_\theta(s_i,a_i,s_{i+1})$. The gradient of the KL divergence is then

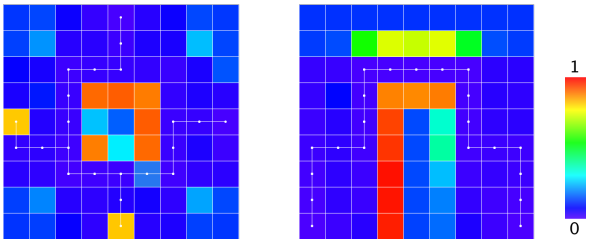
$$\nabla_\lambda \mathrm{D}_{\mathrm{KL}}(p(\zeta|T_\theta,\pi_\lambda)\|p^{\approx}(\zeta|T_\theta)) = \sum_{i=1}^{\tau}\mathrm{E}\left[\nabla_\lambda \ln\pi_\lambda(s_i,a_i)\left(q_\lambda(s_i,a_i) - \sum_{j=i}^{\tau}R(s_j)\right)\right], \tag{12}$$

where the expectation is over trajectories drawn from $p(\zeta|T_\theta,\pi_\lambda)$. We can therefore apply BAM to continuous state spaces under either action selection assumption.



(a) block environment (b) wall environment

Figure 1: Maps of the two grid environments. White squares indicate obstacles. Red squares are the initial states of the training trajectories, while green squares are the goal states.



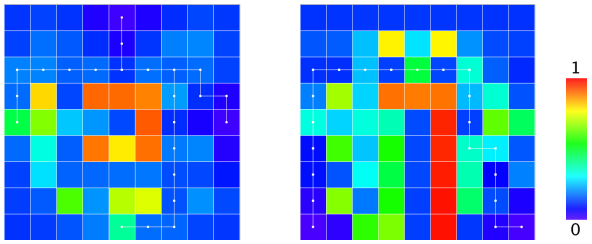
(a) block environment (b) wall environment

Figure 2: Models learned with the BAM algorithm, with paths planned under these models. The color represents the probability (from 0 to 1) of an obstacle in each state.

Experiments

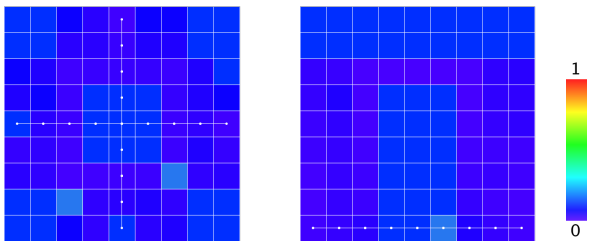
To demonstrate how BAM can learn transition models for unobserved states and actions, we conducted a set of experiments using synthetic trajectory data. We use the two 9x9 grid world environments shown in Figures 1a and 1b in which certain states are blocked by obstacles. State transitions are deterministic, and the agent can move up, down, left or right. Actions that would lead to a blocked state simply maintain the current state. For each start state/goal pair, we generated 10 optimal state-action trajectories, and used these to learn a transition model of the environment. In these environments, there are multiple optimal paths to the goal states, and so trajectories are generated by selecting randomly from the optimal actions for a state. Because the trajectories are optimal, the training set contains no actions that would result in a collision, and thus the presence of obstacles must be inferred from the user's actions.

We learn a nondeterministic model of the deterministic transition dynamics. The parametric model assigns a parameter $\theta(s)$ to each state which gives the probability of that state being blocked by an obstacle. Under this model, if s' is the adjacent state given current state s and action a, the probability of remaining in state s is $T_\theta(s,a,s) = 1/(1+\exp\{-\theta(s')\})$. We use a squared penalty term $\frac{1}{a}(\theta(s)-b)^2$ for each parameter, corresponding to a normal prior $\theta(s) \sim \mathcal{N}(b,a)$ for each parameter. We set $b = -2$ to bias the model towards the absence of obstacles. In these experiments, BAM assumes that actions are based on the softmax return $Q^{\approx}(s,a)$, as defined in Equation 4.



(a) block environment (b) wall environment

Figure 3: Models learned with the BAM algorithm, but without considering the likelihoods of the observed transitions, with paths planned under these models. The color represents the probability (from 0 to 1) of an obstacle in each state.



(a) block environment (b) wall environment

Figure 4: Models learned with maximum likelihood, using only transitions and not using BAM, with paths planned under these models. The color represents the probability (from 0 to 1) of an obstacle in each state.

Figures 2a and 2b are visualizations of models of each environment learned with the BAM algorithm, along with paths planned under those models. They show that most of the states which are blocked in the true environment have been given a high probability of being blocked in the learned model. We note that even though the learned model is not a perfect reconstruction of the true environment, it captures enough information about the obstacles to be useful for planning paths to new goals. Figures 3a and 3b show the models learned by the BAM algorithm when we remove the component of the gradient in Equation 6 that incorporates the probability of the observed transitions themselves. While the paths are still correct, we can see higher obstacle probabilities in empty states, demonstrating how these models have not taken into account the fact that some of the observed trajectories actually passed through these states.

Simply building a model based on the observed transitions is ineffective in this setting. Figures 4a and 4b show the models learned by maximizing the likelihood of the observed transitions, without considering the likelihood of the actions themselves. These learned models do not indicate the presence of any obstacles because no transitions that collide with an obstacle are ever observed. These simple examples therefore illustrate the potential value of the BAM algorithm in learning from real users. Even when users avoid parts of the state space, the BAM algorithm can recover information about the transition dynamics for those states. These results do suggest some limitations of the BAM algorithm, such as overfitting, where the model contains non-existent obstacles which nonetheless help explain the observed behaviors. We argue however that these problems can be alleviated by a better choice of model representation, and the use of a more informative prior distribution over models.

Conclusion

In this work we have described the *behavior-aware modeling* (BAM) algorithm, an approach to the problem of learning transition models that takes into account the fact that data is generated by a user attempting to accomplish a specific task. This preliminary work has demonstrated, in principle, that by using assumptions about how users select their actions BAM can learn about the dynamics of parts of the state/action space that are never observed in the training data. We believe that BAM can be applied to a wide variety of domains in which we need to learn by observing humans or human-controlled agents. Future work will focus on using the BAM algorithm to learn models from data generated by real human users, in more realistic simulated domains and on robotic platforms. Future work will also examine how well BAM handles continuous state spaces, and what model representations are most effective.

Acknowledgments

This work was supported in part by NSF grants IIS1319412 and IIS1643614.

References

Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, 1. ACM.

Ng, A. Y., and Russell, S. J. 2000. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 663–670. Morgan Kaufmann Publishers Inc.

Peters, J., and Schaal, S. 2006. Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2219–2225. IEEE.

Ramachandran, D., and Amir, E. 2007. Bayesian inverse reinforcement learning. In *in 20th Int. Joint Conf. Artificial Intelligence*.

Ziebart, B. D.; Maas, A. L.; Bagnell, J. A.; and Dey, A. K. 2008. Maximum entropy inverse reinforcement learning. In *AAAI*, 1433–1438.

Ziebart, B. D.; Ratliff, N.; Gallagher, G.; Mertz, C.; Peterson, K.; Bagnell, J. A.; Hebert, M.; Dey, A. K.; and Srinivasa, S. 2009. Planning-based prediction for pedestrians. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3931–3936. IEEE.

Ziebart, B. D. 2010. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Ph.D. Dissertation, University of Washington.