A Proposal for Behavior Prediction via Estimating Agents' Evaluation Functions Using Prior Observations of Behavior

Robert Loftin
North Carolina State University
rtloftin@ncsu.edu

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

Abstract

In this work we present a theoretical approach (not currently implemented), to the problem of predicting agent behavior. The ultimate goal of this work is to learn models that can be used to predict the future actions of intelligent agents, based on previously recorded data on those agents' behavior. We believe that we can improve the predictive accuracy of our models by assuming that an agent reasons about the actions it takes, and trying to explicitly model that reasoning process. Here, we model an agent's reasoning process as a form of Monte-Carlo search, and attempt to learn a state evaluation function that, when used with this planning algorithm, yields a similar distribution of actions given the current state of the world as we observe in the data. While it is simple to simulate Monte-Carlo search given an evaluation function, it is much more difficult to determine an evaluation function that will generate a certain behavior. Here we will use Expectation-Maximization to find a maximum likelihood estimate of the parameters of the evaluation function, treating the actual steps taken in planning each action as unobserved data.

Introduction

The ability to accurately predict the actions of an intelligent agent is useful for a wide variety of problems, such as computer security (Qin and Lee 2004) and automated driver assistance (Dagli and Reichardt 2002). In this work we are specifically interested in sampling from a distribution over actions, conditioned on the current state of the world, that is as close as possible to the distribution of actions that a given agent would take in the same state. This work is a detailed proposal for an approach to behavior prediction which learns parametric models that can be used to predict the behavior of agents, based on previously recorded data on those agents' behaviors. As this approach has yet to be implemented and experimentally validated, we attempt to provide theoretical justification for our approach, and discuss multiple possible design choices that could be made when implementing the approach for a specific domain.

As opposed to modeling a distribution over actions directly using existing techniques, we believe that we can improve the predictive accuracy of our models by assuming that an agent reasons about the actions it takes, and trying to explicitly model that reasoning process. We can reasonably assume that any intelligent agent will have some goals or motivations, some states of the world they prefer over others, and that they follow some procedure to choose actions based on these motivations. We do not need know exactly how the agent reasons about their actions, as many different processes may lead to the same observed behavior. Instead, this assumption serves as an *inductive bias* for our prediction approach, a bias towards actions that are reasonable (though not necessarily optimal) with respect to some motivations.

We specifically assume, however, that while agents attempt to take optimal action, they can only reason within certain computational bounds, and so may act sub-optimally. While assuming optimal behavior can be helpful in some domains, such as game playing, ensuring an opponent can do no better than some outcome, in many domains it will lead to less accurate predictions of behavior. While different reasoning models might ultimately lead to the same optimal behavior, if those models are not allowed to run until they find truly optimal solutions, then they may yield potentially very different approximate solutions.

Here, we model an agent's reasoning process as being similar to Monte-Carlo planning (Kocsis and Szepesvári 2006), and attempt to learn a state evaluation function that, when used with this planning algorithm, yields a similar distribution of actions given the current state of the world as we observe in the data. The structure of this type of planning algorithm lends itself to learning the state evaluation function efficiently. It is easy to simulate Monte-Carlo search given an evaluation function, but it is much more difficult to determine an evaluation function that will generate a certain behavior. Here we will use Expectation-Maximization to find a maximum likelihood estimate of the parameters of the evaluation function, treating the actual steps taken in planning as hidden data that must be marginalized over.

Related Work

A number of techniques have been applied to the problem of behavior prediction, including supervised learning and probabilistic modeling (Davison and Hirsh 1998). While not restricted to the problem of behavior prediction, work on the problem of *inverse reinforcement learning* (Ng, Russell, and others 2000; Ramachandran and Amir 2007) or IRL is most

closely related to the approach described here. Inverse reinforcement learning is the problem of identifying a reward function for an otherwise known Markov Decision Process such that a policy or observed set of actions is optimal with respect to that reward function. IRL could be seen as a more general form of goal recognition, where the reward function can represent a preference over possible goals, and over ways of reaching those goals (for example, by avoiding certain bad states). IRL has been used to allow agents to learn to perform behaviors based on partial demonstrations (not covering all possible states) of those demonstrations by another agent (Abbeel and Ng 2004), a problem that is very similar to that of learning to predict another agents behavior in previously unseen states.

Most closely related to our work is the use of maximum-entropy IRL to predict the paths of pedestrians so as to allow a mobile robot to avoid them (Ziebart et al. 2009). In that work, IRL was applied to learn a reward function used to predict the movement of humans walking in an environment. The significance of that work to our approach is that it assumed a degree of sub-optimality in the humans' choice of path, allowing for a more accurate prediction of their actual behavior. It did not, however explicitly consider the nature of the underlying reasoning process used to choose those paths, instead assuming that the sub-optimal policy resulted from a sub-optimal value function.

While analogous to our approach, IRL algorithms generally work only with relatively small state spaces, or with value functions that can be linearly approximated in some set of state features, and do not scale well to many real-world domains. Because our algorithm explicitly models the planning done by the agent or agents in the training data, it does not need to find truly optimal solutions to a MDP or planning problem. Our algorithm can also use classes of evaluation functions which are nonlinear in their parameters.

Motivation

Our ultimate goal is to be able to predict the action that an agent will take in a given state of the world, or more specifically, to sample from the distribution over actions that is as close as possible to the actual action distribution. To do this, we will learn a parametric model of this distribution from data on previous actions of the target agent, or of agents which we assume choose actions in a similar way to the target agent. For simplicity, we assume that the state representation is sufficiently detailed such that we only need to condition our learned action distribution on the current state. Instead of simply modeling a function or probability distribution over the agent's actions given the current state, our algorithm learns a parametric model of the agent's reasoning process, and simulates that process to predict the agent's actions in states not seen in the training data.

We assume that the agent chooses its next action by executing a form of Monte-Carlo search. Unlike standard Monte-Carlo search algorithms (Browne et al. 2012), we also assume that that the agent has some *evaluation function* which it uses to asses the value of different states reached during that search. It is this evaluation function that will actually be learned, and used to simulate the agent's reasoning. The relationship between the evaluation function, and the probability of a particular action being selected, is highly complex, such that directly optimizing the evaluation function with respect to the accuracy of the action distribution will be intractably hard. Given the actual sequence of states explored during a *planning run*, the optimal evaluation function is much easier to estimate. As the details of the planning run for each data point (state-action) pair are unknown, we will compute a *maximum likelihood* estimate of the parameters of the evaluation function model, attempting to marginalize over all possible planning runs, using the Expectation-Maximization (EM) algorithm (Dempster, Laird, and Rubin 1977).

We believe that this approach will allow for better generalization, that is, it will allow a sufficiently accurate model to be learned with less data, than could be achieved by learning a function or probability distribution mapping states to actions. Like inverse reinforcement learning, there are two ways in which this approach can improve generalization. Firstly, we assume that the state evaluation function (and the transition dynamics) are simpler to represent, and can be learned from fewer examples, than a function or probability distribution directly mapping state to actions, that is, it is simpler to represent the goal than it is to represent the set of actions needed to reach it. Secondly, by learning the evaluation function itself, and assuming we know or can learn the transition dynamics of the domain, we can use planning to identify a likely action for a previously unseen state.

Data Representation

We assume that our training data takes the form of a sequence (or set of sequences) of state-action pairs. The domain will use a *STRIPS* planning-like representation (Fikes and Nilsson 1972), where actions will be represented as functions parameterized by some set of entities (entities which may be different from entities in subsequent test data), and that the state will consist of a set of grounded first order predicates over the same set of entities. In this work, S will be the set of all possible *sets* of grounded state predicates, while A is the set of all grounded actions. These groundings are all with respect to the entities present in the training data. Some domain specification may also be provided, giving preconditions for certain actions, which can reduce the size of the action set available in a given state, and may also specify which predicates are added or removed from the state after each action.

As an example, consider a state-action pair that might arise in a computer game. The state takes the form

$$at(Character\_A, Location\_C)$$
$$at(Character\_B, Location\_C)$$
$$damaged(Character\_B),$$

while a logical action in this state (assuming $Character\_A$ is the agent taking an action at this time step) might be

$$attack(Character\_B).$$

As can be seen in this example, the choice of action depends on which agent is actually taking the action. We can assume

that this information is provided for each action. However, we wish to learn an evaluation function over states, one that is independent of the specific set of entities present, and can be used to model the behavior of multiple agents by computing the value of a state relative to the agent actually choosing the action. We must therefore add information to the state representation such that the agent currently acting is apparent. In the simplest case, this could consist of a single predicate, $self(Character\_A)$, where $Character\_A$ is the agent taking the next action, such that the agent can distinguish between itself and all other agents. There will be some actions in the data which do not correspond to the actions of any agent (environmental events), which can be removed for training. We may also wish to ignore the actions of some agents whose behavior we do not wish to model.

Monte-Carlo Planning

Here we will use a form of Monte-Carlo planning (Kocsis and Szepesvári 2006), where the agent simulates the results of a sequence of actions (which we will call a *roll-out*), and then computes the value of the evaluation function at the final state reached after that action sequence. The value of the final state in a roll-out will be propagated back to every action taken in every state encountered during the roll-out. The values propagated back to a state are used to select the actions for subsequent roll-outs that reach that state. We will assume that the agent always does K roll-outs of T actions, with K and T being parameters of the algorithm.

Action selection is done according to some exploration strategy, and here we will use a simple *SoftMax* strategy (Vermorel and Mohri 2005), where actions are selected according to a Boltzmann distribution based on there average value so far. Let $q_i(s, a)$ be the average return of all roll-outs that took action a in state s, up to but not including roll-out i. The probability of taking action a in state s at step i of the planning process would be

$$p(a|s, i) = \frac{1}{Z} e^{\frac{q_i(s,a)}{\alpha_i}},$$

where Z is a normalization constant, and α_i is a temperature parameter which can be gradually decreased to make action selection greedier. While not always the most efficient action selection rule, it has the advantage of having action probabilities which are differentiable with respect to $q_i(s, a)$ (which we will see is differentiable with respect to the evaluation function). Because the SoftMax strategy is stochastic, it will also lead to a likelihood function that is smoother with respect to the evaluation function parameters, which will make optimizing the model parameters easier. Intuitively, any sequence of actions is at least possible under SoftMax, if not probable, whereas with a deterministic strategy and sequence would either be possible or impossible, leading to a highly discontinuous likelihood function.

It should be noted that this approach does not require that an agent actually use this planning algorithm to be able to accurately predict the actions that that agent will take. The evaluation function learned will will be that which maximizes the probability of the agent taking the actions it did, given that it used this algorithm, and so it may not actually represent the true evaluation function used by the agent.

State Transition Model

To actually simulate the agent's planning runs, we will need a model of how the world evolves in time in response to the agent's actions, that is a transition distribution $p(s'|s, a)$, conditioned on the current state and action. We will assume that the model used by the agent is relatively accurate, and so will learn this model directly from the training data, prior to attempting to learn the agent's evaluation function. Therefore, our choice of transition model will not consider how the model affects the agent's choice of actions, only how accurately it models their effects.

This model could be deterministic, based on the planning representation, such that each subsequent state is determined by the set of predicates that are added or deleted from the previous state by the previous action. For simplicity, we will assume that the effects of any action are deterministic. State transitions, however, may still be stochastic from the agent's perspective. Between each action taken by the agent, other events will occur, either actions taken by other agents, or events occurring beyond the control of any agent. The agent will likely choose whether to select a new action in a stochastic manner as well.

In the general case, an accurate transition model will therefore require two components, an event distribution conditioned on the current state, and a binary distribution giving the probability that the agent will take an action in the current state, conditioned on the time since the last action was taken, and possibly on the current state. There are a number of existing algorithms and representations that could be used to learn and sample from these distributions, such as Markov Logic Networks (Richardson and Domingos 2006). The best choice of model will depend on the specific domain.

Evaluation Function Model

What will actually be learned is a state evaluation function, that is, a function $f_\theta : S \mapsto \Re$ represented by parameters θ. This function is meant to encode the near term goals of an agent, such that the actions generated by the resulting action selection model will as often as possible match the actions taken in the training data.

As a concrete example, we can use a multilayer perceptron (Attali and Pagès 1997) as a parametric model of the evaluation function. We can construct by hand (based on domain knowledge) a set of k binary state features $c(s)$, which take the form of existentially or universally quantified first order clauses. Our evaluation function will then consist of a multilayer perceptron, with a single hidden layer of size l, that takes these binary features as inputs, and uses a logistic activation function. The weight for the jth input to the ith hidden node will be w_{ij}, the bias for the ith hidden node will be b_i, and the weight of the output of the ith hidden node will be o_i, such that $\theta = \langle w, b, o \rangle$. Therefore, the value of the evaluation function for state s will be:

$$f_\theta(s) = \sum_i^l o_i \left[\frac{1}{1 + e^{b_i + \Sigma_j^k w_{ij} c_j(s)}} \right].$$

As this model is differentiable with respect to θ, and the expected log-likelihood to be maximized at each step of the EM algorithm is differentiable with respect to the evaluation function, we can optimize θ via gradient ascent.

Expectation-Maximization for Learning Evaluation Functions

Under our model, the action selected by the agent in a given state depends not only on the evaluation function, but also on the sequence of state-action roll-outs generated from that state during the planning run. This sequence of roll-outs is not observable, and is not fully determined by the parameters of the evaluation, due to the stochastic nature of the transition simulation, and potentially of the evaluation function and the exploration rule. As a result, we need to marginalize over the set of possible planning runs when computing the likelihood. Directly marginalizing over possible planning runs, while simultaneously maximizing the parameters of the evaluation function, would be intractable. Fortunately, the problem can be addressed using the Expectation-Maximization algorithm (Dempster, Laird, and Rubin 1977). The basic EM algorithm iteratively computes estimates of the true model parameters, with the $(n+1)$th estimate being

$$\theta_{n+1} = \operatorname*{argmax}_{\theta} \mathrm{E}_{H\sim D,\theta_n} \left[\ln(p(H, D|\theta))\right]. \quad (1)$$

Where each θ_n is a maximum of a progressively tighter lower bound on the log likelihood of the true parameters. This update is typically divided into an expectation step, where the expectation (given θ_n) over the log-likelihood of θ is estimated, followed by a maximization step in which the expectation is maximized with respect to θ. Updates can continue until the θ parameters converge, or until the action distribution they generate is sufficiently close to the distribution observed in the training data set.

Expectation Step

The unknown data H over which we must compute the expectation is the set of planning runs (where each planning run is a sequence of roll-outs) done by the agent for each of the data points (state-action pairs) in the training data set D. It is unlikely that we will be able to tractably marginalize over all possible values for H, and so we will need to estimate the expectation by sampling roll-out sequences from the distribution $p(\tau_i|a_i, s_i, \theta_n)$, where τ_i is a random variable representing the planing run associated with the ith state-action pair in D. As planning runs for each state are independent, we can rewrite the expectation such that we can generate samples of τ_i for each state action pair separately. The expectation at the nth EM iteration is

$$\begin{aligned}\mathrm{E}_{H\sim D,\theta_n} \left[\ln(p(H, D|\theta))\right] &= \mathrm{E}_{H\sim D,\theta_n} \left[\sum_{i=1}^{|D|} \ln(p(\tau_i, D_i|\theta))\right] \\ &= \sum_{i=1}^{|D|} \mathrm{E}_{\tau_i\sim D_i,\theta_n} \left[\ln(p(\tau_i, D_i|\theta))\right].\end{aligned}$$

We will generate κ samples of each τ_i, with κ being a parameter of the algorithm that must be selected based on the domain and on available computational resources.

For a single state-action pair $\langle s, a\rangle$ in the data set, we need to sample from the posterior distribution $p(\tau|s, a, \theta_n)$. Sampling from $p(\tau|\theta_n)$ simply requires running the planning algorithm for N roll-outs, using the evaluation function represented by θ. We can therefore use rejection sampling (Flury 1990) to sample from the posterior, by noting that

$$\begin{aligned}p(\tau|s, a, \theta_n) &= \frac{p(\tau, a|s, \theta_n)}{p(a|s, \theta_n)} \\ &= p(\tau|s, \theta_n) \frac{p(a|\tau, s, \theta_n)}{\sum_{\tau'} p(a|\tau', s, \theta_n)}.\end{aligned}$$

Using $p(\tau|\theta_n)$ as the proposal distribution, we can accept each sample with probability

$$\frac{1}{C} p(a|\tau, s, \theta_n),$$

where C is defined such that

$$C \geq \frac{p(a|\tau, s, \theta_n)}{\sum_{\tau'} p(a|\tau', s, \theta_n)}, \forall \tau.$$

Note that we drop the normalization constant $\sum_{\tau'} p(a|\tau', s, \theta_n)$ as it can be be assumed to cancel out with C.

We assume that the action a is also chosen with the Soft-Max rule, with temperature β, using the final action value estimates $q_N(s, \ldots)$. To minimize the rejection rate, we choose the smallest value of C possible, which is

$$\begin{aligned}C &= \max_{\tau} p(a|\tau, s, \theta_n) \\ &= \max_{q(s,\ldots)} \frac{e^{q(s,a)/\beta}}{\sum_{a'\in A} e^{q(s,a')/\beta}}.\end{aligned}$$

Knowing that our evaluation function is bounded to the range $[c, d], \forall s \in S$, we see that

$$\begin{aligned}C &= \frac{e^{d/\beta}}{e^{d/\beta} + \sum_{a'\neq a\in A} e^{c/\beta}} \\ &= \frac{1}{1 + (|A| - 1)e^{\frac{c-d}{\beta}}}.\end{aligned}$$

And so we accept τ with probability

$$\frac{e^{q(s,a)/\beta}}{\sum_{a'\in A} e^{q(s,a')/\beta}} \left[1 + (|A| - 1)e^{\frac{c-d}{\beta}}\right],$$

where c and d will depend on θ_n.

Initially, with θ_0 chosen at random, we might expect the average returns $q(s, a)$ to be similar for all a, and so $p(a|\tau, s, \theta_0) \sim \frac{1}{|A_s|}$, though for a particularly poor initialization of θ the probability of the correct action being taken could be much lower. At the same time, we would need to set the temperature β low enough such that the probability of selecting the optimal action under a good estimate of evaluation function f_θ will be close to 1, meaning that M will also be close to 1. As the action space A will be quite large under our representation, a large majority of samples would likely be rejected, and to generate κ final samples, we might have to generate $\kappa|A|$ samples from the proposal distribution.

One way to avoid, or at least reduce, this problem is to allow the temperature β to vary, similar to the approach used in simulated tempering (Marinari and Parisi 1992). Starting out with a high temperature, a much larger number of samples will be accepted. As the the estimate of the evaluation function improves, and the sampled planning runs are more heavily biased towards the correct actions, the temperature

can be reduced while still having a relatively high acceptance rate. To update the temperature in a way that maintains the convergence of the EM algorithm, we can include β in the set of parameters being estimated, along with the θ parameters. We will show that β can be updated using gradient ascent, in the same way that θ is updated.

A good initial value of β will be one that is large enough to ensure a reasonable acceptance probability for all samples. If we assume that, on average (across all data points), the probability of a sample τ leading to the correct action is approximately $\frac{1}{|A|}$, and we want an average acceptance probability of α, then we will want β_0 such that

$$\begin{aligned} \alpha &= \frac{1}{|A|}\left[1 + (|A|-1)e^{\frac{c-d}{\beta_0}}\right] \\ \frac{\alpha|A|-1}{|A|-1} &= e^{\frac{c-d}{\beta}} \\ \frac{c-d}{\ln(\alpha|A|-1) - \ln(|A|-1)} &= \beta_0. \end{aligned}$$

In addition to reducing the number of proposal samples which need to be generated, allowing β to vary may improve the accuracy of the learned model. For example, there may be similar (or even identical) states in the data set with different associated actions, such that choosing θ to account for those data points may be difficult or impossible. A less greedy action selection distribution (higher β) may better account for these cases, and even if some value of θ could account for them with greedy action selection, that function might not generalize as well to states not observed in the data set. Therefore, a variable temperature may help to reduce overfitting of the evaluation function.

We define $\bar{H}$ to be the set of all accepted samples τ for all data points in D. Regardless of how we choose to generate these sample planning runs, we can write the sample estimate of the expectation as

$$\frac{1}{\kappa}\sum_{\tau\in\bar{H}} \ln\left[p(a_\tau, \tau|\theta)\right], \tag{2}$$

where κ is again the desired number of samples for each state-action pair in the data set, and a_τ is the action from the state-action pair for which sample τ was generated.

Maximization Step

The evaluation function parameters θ will be trained via gradient ascent on the expected log-probability (Equation 2). Note that we can restrict the number of gradient ascent updates performed per maximization step to some fixed constant, as the EM algorithm only requires that θ_{n+1} be an improvement over θ_n (Neal and Hinton 1998). At each update, we can compute the gradient separately for each sampled planning run $\tau \in \bar{H}$, such that the full gradient is

$$\frac{1}{\kappa}\sum_{\tau\in\bar{H}} \nabla_\theta \ln\left[p(a_\tau, \tau|\theta)\right].$$

To derive the gradient of a single sample τ, we will define a sequence, from 0 to $KT+1$, of state-action-state triples. $\langle s_i, a_i, s'_i\rangle$ will represent the ith action selection made during planning run τ (essentially concatenating the sequence of K roll-outs), choosing action a_i in state s_i, and transitioning to state s'_i. We then have

$$\begin{aligned} \nabla_\theta \ln\left[p(\tau, a|\theta)\right] &= \sum_{i=0}^{KT} \ln\left[p(s'_i, a_i|s_i, \theta)\right] && (3)\\ &= \sum_{i=0}^{KT} \ln\left[p(s'_i|a_i, s_i)p(a_i|s_i, \theta)\right] && (4)\\ &= \sum_{i=0}^{KT} \ln\left[p(s'_i|a_i, s_i)\right] + \sum_{i=0}^{KT}\left[p(a_i|s_i, \theta)\right]. && (5) \end{aligned}$$

As the state transition probabilities $p(s'|s,a)$ are independent of θ, we can drop the second term of Equation 5, and can derive the gradient in terms of the gradient of the values $q^i_\theta(a)$, which we define as the average return (up to time i), for taking a in state s_i:

$$\begin{aligned} \sum_{i=0}^{KT} \nabla_\theta \ln\left[p(a_i|q^i_\theta, s_i)\right] &= \sum_{i=0}^{KT} \nabla_\theta \ln\left[\frac{e^{q^i_\theta(a_i)/\alpha_i}}{\sum_{a\in A} e^{q^i_\theta(a)/\alpha_i}}\right] && (6)\\ &= \sum_{i=0}^{KT}\left[\frac{\nabla_\theta q^i_\theta(a_i)}{\alpha_i} - \nabla_\theta \ln\left(\sum_{a\in A} e^{\frac{q^i_\theta(a)}{\alpha_i}}\right)\right] && (7)\\ &= \sum_{i=0}^{KT}\left[\frac{\nabla_\theta q^i_\theta(a_i)}{\alpha_i} - \frac{\sum_{a\in A}\left(e^{q^i_\theta(a)}\frac{\nabla_\theta q^i_\theta(a)}{\alpha_i}\right)}{\sum_{a\in A} e^{q^i_\theta(a)/\alpha_i}}\right]. && (8) \end{aligned}$$

We note that the q^i_θ values are simply the average return of all roll-outs prior to i that have taken action a_i in state s_i. We define $\mu_i(a, s')$ to be the number of roll-outs prior to action selection step i that have taken a in state s_i and have terminated in state s'. Therefore

$$q^i_\theta(a) = \frac{\sum_{s'\in S}\mu_i(a,s')f_\theta(s')}{\sum_{s'\in S}\mu_i(a,s')}, \tag{9}$$

with $\mu_i(a, s')$ equal to 0 for most states s'. Therefore

$$\nabla_\theta q^i_\theta(a) = \frac{\sum_{s'\in S}\mu_i(a,s')\nabla_\theta f_\theta(s')}{\sum_{s'\in S}\mu_i(a,s')}. \tag{10}$$

Plugging this into Equation 8, we get the gradient of the log probability of sample τ with respect to θ.

We can similarly find the derivative of $\ln[p(\tau|\theta)]$ with respect to the temperature parameter β, which depends only on the final state action pair in planning run τ, $\langle s_{KT}, a_{KT}\rangle$, with $\alpha_{KT} = \beta$.

$$\begin{aligned} \frac{\partial}{\partial\beta}\ln\left[p(a_{KT}|q^{KT}_\theta, s_{KT})\right] &= \frac{\partial}{\partial\beta}\ln\left[\frac{e^{q^{KT}_\theta(s_{KT},a_{KT})/\beta}}{\sum_{a\in A} e^{q^{KT}_\theta(s_{KT},a)/\beta}}\right] \\ &= -\frac{q^{KT}_\theta(s_{KT},a_{KT})}{\beta^2} - \frac{\partial}{\partial\beta}\ln\left(\sum_{a\in A} e^{\frac{q^{KT}_\theta(s_{KT},a)}{\beta}}\right) \\ &= \frac{\sum_{a\in A}\left(e^{1/\beta}\frac{q^{KT}_\theta(s_{KT},a)}{\beta^2}\right)}{\sum_{a\in A} e^{q^{KT}_\theta(s_{KT},a)/\beta}} - \frac{q^{KT}_\theta(s_{KT},a_{KT})}{\beta^2}. \end{aligned}$$

Note that we do not have to update β with the same gradient step size, nor with the same number of iterations as we do θ, as we only need an improvement, not a true maximum, for the EM algorithm to work. It may improve sampling performance if β is changed more slowly.

Efficiently Computing the Gradient It can be seen in this form that computing the gradient for each sample will take time proportional to $KT + 1$, the total number of action selection steps in the sample. While the gradient of the log probability is linear in the gradient of the evaluation function, it is highly non-linear in the parameters θ, due to the normalization constants. As we will do multiple gradient ascent steps per maximization step, the maximization step may end up representing the bulk of the cost of the algorithm, depending on the sample rejection rate of the expectation step.

There are, however, aspects of the problem that we can take advantage of to make computing the gradient somewhat less expensive. For one, a large fraction of action selection steps will occur in states that have not been previously encountered during the planning run, and so have no dependence on the evaluation function. These selection steps (which include the entire initial roll-out, and deeper segments of many subsequent roll-outs) can therefore be ignored when computing the gradient.

We also note that the first term of the gradient (Equation 8) is linear in the gradient of the evaluation function, and does not depend on the parameters of the evaluation function in any other way. This term could therefore be replaced by a linear combination of the gradients for each state

$$\sum_{s' \in S} w_{s'} \nabla_\theta f_\theta(s'),$$

with weights

$$w_{s'} = \sum_{i=0}^{KT} \frac{\sum_{s' \in S} \mu_i(a, s')}{\alpha_i \sum_{s' \in S} \mu_i(a, s')},$$

where $\mu_i(a, s')$ is defined as in Equation 9. These weights would be zero for any state that was not the terminal state of some sampled roll-out.

We can further reduce the effort required by noting that while each normalization constant must be recomputed for each action selection step, for each new set of θ parameters, at each gradient update, the constants for some selection steps may be equivalent. This would occur when there are two action selection steps i and j such that there exists a bijection $m : A \mapsto A$ (a permutation of actions) such that $\forall s \in S, a \in A, \mu_i(a, s) = \mu_j(m(a), s)$. In this case, regardless of the parameters of the evaluation function, the normalization constants for steps i and j would be equal. By assigning a total order over μ functions, a look-up table could be used to keep track of the number of occurrences of each unique μ function, and compute the second term of Equation 8 once for each μ.

Future Directions

Future work will involve implementing the algorithm described here, and applying them to real-world data. We will evaluate the accuracy and generality of the learned models, as well as the computational efficiency of the approach, in a number of domains. The accuracy and efficiency of this algorithm is highly dependent on the configuration parameters (search depth, number of roll-outs) and the choice of transition and evaluation models used, and so efforts will be made to find good values and models for different domains, and if possible to automate parameter selection to some degree. Future work will also consider how the models learned by this algorithm could be used to simulate the behavior of a collection of agents in some domain, for the purpose of estimating the probability of events of interest occurring in that domain, given some initial state.

There are a number of ways in which this approach could be extended to improve accuracy or computational efficiency. For example, for states that have not previously been visited during a planning run, the agent could be assumed to use the value of some heuristic function to select actions. We might also assume that the agent assigns return values not only to the final state of each roll-out, but to intermediate states as well, possibly to avoid undesirable states. These heuristic or return functions could be modeled using the same representation as used for the evaluation function, and similarly trained, though at potentially greater computational cost. We can also consider learning more abstract state representations that the agent might use. Learning to group states together, or ignore state features, in such a way that the resulting action selection more closely matches the training data, could in fact lead to reduced computational complexity when generating, or optimizing over, planning runs.

Conclusion

Here we have presented an approach to the problem of predicting the future actions of intelligent agents. Our proposed approach explicitly models the reasoning process such agents follow when choosing their actions, and learns parameters of that reasoning process (the state evaluation function) which yield behavior similar to that observed in training data. Because this algorithm models the entire reasoning procedure, it should be able to generalize more effectively to novel situations than algorithms that directly model an agent's actions. By working with a more complex planning representation, and modeling reasoning as a more efficient search procedure, this algorithm can be applied to much larger state and action spaces capable of representing real-world problems. Our overall approach is independent of many domain specific details such as the choice of state transition model and evaluation function representation, and so should be applicable to a wide range of domains. Ideally, the algorithm described here could be combined with other approaches, so as to enable the simulation and prediction of the evolution of complex, multi-agent environments.

Acknowledgments

This material is based upon work supported in whole or in part with funding from the Laboratory for Analytic Sciences (LAS). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the LAS and/or any agency or entity of the United States Government.

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, 1–8. ACM.

Attali, J.-G., and Pagès, G. 1997. Approximations of functions by a multilayer perceptron: a new approach. *Neural networks* 10(6):1069–1081.

Browne, C.; Powley, E.; Whitehouse, D.; Lucas, S.; Cowling, P.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on* 4(1):1–43.

Dagli, I., and Reichardt, D. 2002. Motivation-based approach to behavior prediction. In *Intelligent Vehicle Symposium, 2002. IEEE*, volume 1, 227–233 vol.1.

Davison, B., and Hirsh, H. 1998. Probabilistic online action prediction. In *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, 148–154.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)* 39(1):pp. 1–38.

Fikes, R. E., and Nilsson, N. J. 1972. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3):189–208.

Flury, B. D. 1990. Acceptance-rejection sampling made easy. *SIAM Review* 32(3):pp. 474–476.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*. Springer. 282–293.

Marinari, E., and Parisi, G. 1992. Simulated tempering: a new monte carlo scheme. *EPL (Europhysics Letters)* 19(6):451.

Neal, R. M., and Hinton, G. E. 1998. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*. Springer. 355–368.

Ng, A. Y.; Russell, S. J.; et al. 2000. Algorithms for inverse reinforcement learning. In *Icml*, 663–670.

Qin, X., and Lee, W. 2004. Attack plan recognition and prediction using causal networks. In *Computer Security Applications Conference, 2004. 20th Annual*, 370–379. IEEE.

Ramachandran, D., and Amir, E. 2007. Bayesian inverse reinforcement learning. *Urbana* 51:61801.

Richardson, M., and Domingos, P. 2006. Markov logic networks. *Machine learning* 62(1-2):107–136.

Vermorel, J., and Mohri, M. 2005. Multi-armed bandit algorithms and empirical evaluation. In *Machine Learning: ECML 2005*. Springer. 437–448.

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 *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 3931–3936. IEEE.