

Learning Executable Agent Behaviors from Observation

Andrew Guillory, Hai Nguyen, Tucker Balch, Charles Lee Isbell, Jr.

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

{guillory, haidai, tucker, isbell}@cc.gatech.edu

ABSTRACT

We present a method for learning a human understandable, executable model of an agent's behavior using observations of its interaction with the environment. By *executable* we mean that the model is suitable for direct execution by an agent. Traditional models of behavior used for recognition tasks (*e.g.*, Hidden Markov Models) are insufficient for this problem because they cannot respond to input from the environment. We train an Input/Output Hidden Markov Model where the output distributions are mixtures of learned low level actions and the transition distributions are conditional on features detected by the agent's sensors. We show that the system is able to learn both the behavior and human-understandable structure of a simulated model.

Categories and Subject Descriptors

I.5 [Computing Methodologies]: Pattern Recognition

General Terms

Algorithms, Verification

Keywords

Behavior modeling, Input/Output Hidden Markov Models

1. INTRODUCTION

An ethologist studying the behavior of an animal such as an ant or bee typically starts by defining a set of low-level actions the animal performs. Having defined such low-level actions, the scientist then determines high-level structure; that is, she models what causes the animal to switch between low-level actions. Similarly, a roboticist or agent developer might program a behavior by coding low-level control modes corresponding to primitive actions and combining these low-level control modes together through some kind of high-level action selection mechanism. In this paper we present a method for learning both low-level actions and a high-level

switching model. The result is human-understandable, but also *executable* in the sense that it can be used as a control program in a simulation or as part of an agent.

More formally we assume we are given a model of the agent's perception, a set of perceptions that may cause the agent to switch actions, and example trajectories of the agent interacting in its environment (tracking data), some of which has been labeled by a human expert designating the action the agent is performing at a given time step. We further assume that the agent acts according to a Markov Process with inputs. Our task is then to compute an executable model of the agent's behavior composed of controllers for low-level actions and rules for switching between them.

Our approach is to train an Input/Output Hidden Markov Model (IOHMM) where the output distributions are mixtures of learned low-level actions and the transitions are conditioned on perceptions of the agents. We first learn from labeled data a set of low-level actions that may be assigned to states in the model. Having learned these low-level actions, the problem is then to learn from unlabeled data a mapping between states and low-level actions as well as the transition rules between states. By dividing the problem this way, the human expert can simply label portions of the data where the agent is obviously performing a particular action (not necessarily data in which the agent switches actions).

Our work is in contrast to *activity recognition*. Although models of behavior learned in activity recognition are often generative, they are not usually suitable for re-creation. Hidden Markov Models (HMMs), for example, are generative models but are usually limited in their ability to recreate real activities because they cannot respond to input. Goldberg and Matarić use an HMM based model to learn a robot's behavior patterns online [5]. Our work is more similar in goal to *imitation learning*. The notion of primitives in [3] is similar to our notion of low-level actions. Such techniques generally differ from our work by focusing more on learning and detecting low-level actions while constrained by the kinematics of particular robots. Our work is most similar to that of [4]. The authors approach the problem from the perspective of control theory and recover from unlabeled data both low-level control modes and a high-level control program in the form of a motion description string. Other related work includes [7] where the authors learn decision trees for low-level control modes.

In order to test our learning algorithms, we programmed by hand a model of foraging inspired by the behavior of social insects, using the TeamBots simulation platform and motor schema-based control [1]. Our experimental method

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.
Copyright 2006 ACM 1-59593-303-4/06/0005 ...\$5.00.

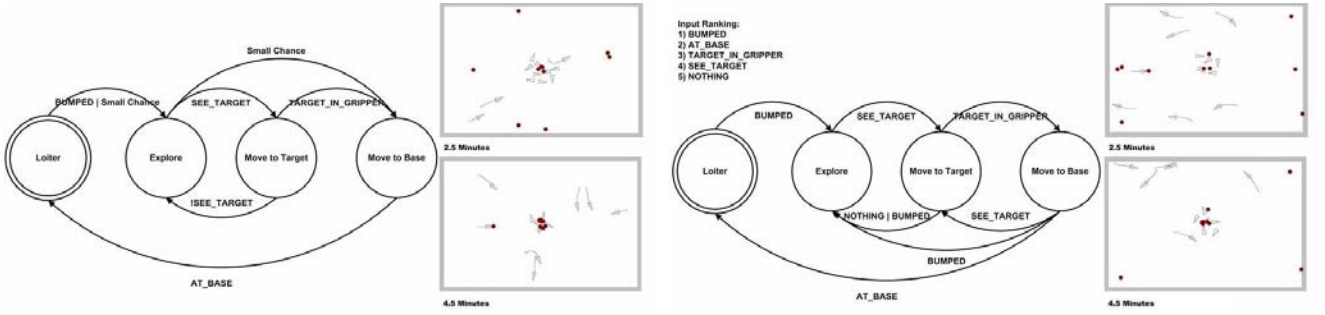


Figure 1: On the left, our original model, including screenshots from the simulation. On the right, a high likelihood learned model, showing the most likely transitions for every input, excluding self transitions.

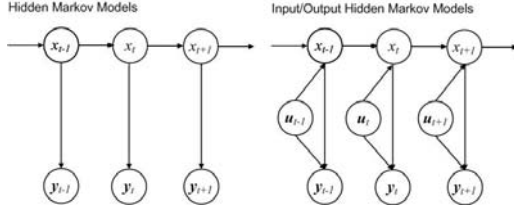


Figure 2: Networks for IOHMMs and HMMs

was to generate data from this known model, then see if we are able to automatically recover the behavior from the data. Figure 1 shows our model. From the simulation we created two data sets, one labeled with the action the agent is performing at each time step and one unlabeled. For both data sets we ran the model in simulation with 12 agents at 33 frames per second, waiting for all of the targets to be carried to base, recording at each frame the positions and orientations of all agents as well as the positions of all targets. Our goal is then to, from the labeled data set, learn the low-level actions of the agent and, from the unlabeled data set, learn the switching behavior.

2. IOHMMS AS EXECUTABLE MODELS

HMMs [6] represent the probability of an observation sequence $P(\mathbf{y}_1^T)$ where $\mathbf{y}_1^T = \mathbf{y}_1 \dots \mathbf{y}_T$ is a sequence of observation vectors. IOHMMs [2] are a generalization of this model that represent the conditional probability of an observation sequence given an input sequence $P(\mathbf{y}_1^T | \mathbf{u}_1^T)$ where $\mathbf{u}_t$ is the input vector for time t . Figure 2 shows the dependency assumptions of the two models, where x_t is the discrete state of the model at time t . The essential difference between the two models, is that in an IOHMM the transition distributions $P(x_t | x_{t-1} = i, \mathbf{u}_t)$ and output distributions $P(\mathbf{y}_t | x_t = i, \mathbf{u}_t)$ are both conditional on the current input vector. The standard algorithm for training IOHMMs is an Expectation Maximization (EM) algorithm that is a straightforward extension of Baum-Welch for HMMs. After training, the model can be interactively executed on an input sequence in a manner similar to a finite state machine.

In our case, the input $\mathbf{u}_t$ is the tracking data at time t , including the position of the agent. The corresponding output $\mathbf{y}_t$ is the next position of the agent after moving. By using the model interactively in a simulator environment we can recreate the global behavior of the agent. Although consis-

tent with previous papers on IOHMMs, our use of variable names and the terms *input* and *output* are different than their use in dynamic systems literature (where input often refers to input into the environment).

We model output distributions as mixtures over already learned low-level actions. Mixture distributions marginalize a distribution over a discrete variable, in this case a_t , the low-level action the agent is performing at time t . Normally, a mixture distribution in an IOHMM would have mixing weights conditioned on $\mathbf{u}_t$ as well as x_t . For our application, input-conditional mixing weights are undesirable because they complicate the meaning of a state—there is not a simple mapping between states and low-level actions. Our approach is to use unconditional mixing weights but also for the purposes of parameter estimation model the output distributions as generating both $\mathbf{u}_t$ and $\mathbf{y}_t$. This allows us to represent information about $\mathbf{u}_t$ without introducing input-conditional mixing weights. We further assume $\mathbf{u}_t$ and $\mathbf{y}_t$ are independent of x_t given a_t (that the low level actions are the same for each state). The overall model is something like an HMM/IOHMM hybrid: the transition distributions are conditional while the output distributions are not. With this model, assuming we have already calculated the values for the mixture components at each time step, which we factor into $P(\mathbf{y}_t | a_t, \mathbf{u}_t)$ and $P(\mathbf{u}_t | a_t)$, only the mixing weights need to be re-estimated during EM training.

We must also represent our transition distributions. In our experiments we avoid some complexity and maintain human understandability by conditioning the transition distributions on a set of binary sensory features calculated from $\mathbf{u}_t$, just as we have done in our simulated domain. We can then represent the transition distributions as look up tables. We experimented with ranking our binary features. We then took as our input the most important triggered feature. This ranking method greatly reduces the number of parameters and constrains the model to resemble hand-made models where only a single binary feature triggers a transition. Unfortunately, choosing a feature ranking requires domain knowledge. We also experimented with a standard binary encoding, allowing for a different transition rule for every possible combination of feature values.

When estimating probabilities in look up tables, it is standard to add a small constant to the counts for each entry to allow some probability mass everywhere. In our case, it is useful to add a small number to only the entries corresponding to self-transitions. In other words, given little evidence, an agent should remain in its current state.

3. LEARNING LOW-LEVEL ACTIONS

To estimate the mixing weights during EM from unlabeled data we need to estimate values for $P(\mathbf{y}_t|a_t, \mathbf{u}_t)$ for each time step and low-level action. To execute the resulting model, we also need to be able to sample from this distribution. Our approach to this problem is to learn a controller for each low-level action from labeled data.

From labeled data, we can calculate sensory features and corresponding motor commands for each low-level action (example inputs and outputs for each controller). In our experiments we solve the function approximation problem by using a modified version of k -nearest neighbor (KNN), randomly choosing from the k nearest neighbors with the probability of each weighted by the point's distance from the query point. By randomly choosing in this way we can model random actions like the Explore action. With the learned controller and a simulation environment, we can sample from $P(\mathbf{y}_t|a_t, \mathbf{u}_t)$ (i.e. simulate a single time step of the agent's motion). From this we can also estimate values for the distribution for unlabeled data using kernel density estimation. For each time step and action we run a simulation many times to produce distributions of points predicting the next position of the agent. We then compare these point distributions to the actual next position to decide which action the agent is performing.

We also learn a model for $P(\mathbf{u}_t|a_t)$ from labeled input vectors. We are not interested in sampling from this distribution as during execution our model does not generate $\mathbf{u}_t$, but we need to estimate its values in order to calculate the mixing weights, since we modeled the output distributions as joint distributions over $\mathbf{u}_t$ and $\mathbf{y}_t$. In our experiments, we again reduce $\mathbf{u}_t$ to a set of binary features, using the same features and encoding as we did for the transition distributions, and learn look up tables.

4. RESULTS

We used k -nearest neighbors on the agent's sense-action pairs to learn the low level controllers. For the sensory features we used distance and angle to the closest obstacle, closest target, closest agent, and the base as well as whether or not the agent is currently carrying a target. For the motor outputs we calculated the distance and angle the agent moved at each time step as well as whether the agent was carrying an object or not. We then tested the learned controllers in simulation using the original forage switching behavior. Our learned low-level controllers were able to recreate the original foraging behavior even on foraging arenas where the position of food items are moved from their original position as well as differently sized arenas with more attractors and more obstacles. Even though the agents were able to recreate the original foraging behavior by successfully gathering all the targets, there were still flaws. For example, the agents would frequently miss targets while moving towards them in the Move to Target controller, especially targets in corners.

With learned low-level actions, we trained 150 models with EM for both ranked and unranked input types, and 40 ranked and 32 unranked input models learned the correct structure of the behavior (that there was one state for each low level action). The likelihood scores for the models that learned the correct structure were all higher than the scores for the models that did not. Their likelihood scores

were also greater than the original model's. Within these models, there were several clusters of models whose likelihood scores were identical within range of our convergence criteria. We tested models from each of these clusters. Figure 1 shows the model from the cluster with the highest likelihood score for the ranked input type. The unranked input type model is similar, except with more complicated conditions on the transitions. 14 ranked and 21 unranked models were within the maximum likelihood clusters.

All of the models that learned the correct structure were able to recreate the forage behavior in simulation when combined with the learned controllers. However, they also all had flaws that affected the behavior to varying degrees. For example, many models did not know what to do when they missed a target or another agent reached a target before them. Some would even remain stuck in Move to Target in this situation, in the worst case then becoming stuck against a wall. Subjectively the models with the maximum likelihood scores seemed to recreate the behavior the best.

5. CONCLUSIONS AND FUTURE WORK

We have presented a method for learning an executable model of behavior from observations. We tested our method on simulated data generated from a hand-constructed model. The learned models recreated the behavior of the original model and exhibited the same basic structure. We hope to next apply these techniques to actual insect tracking data.

6. ACKNOWLEDGMENTS

Thank you to Sridhar Mahadevan, Zia Khan, and Adam Feldman for helpful discussions. This work was supported in part by NSF.

7. REFERENCES

- [1] R. C. Arkin and T. R. Balch. Aura: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, 9(2-3):175–188, April 1997.
- [2] Y. Bengio and P. Frasconi. Input-output HMM's for sequence processing. *IEEE Transactions on Neural Networks*, 7(5):1231–1249, September 1996.
- [3] D. C. Bentivegna, A. Ude, C. G. Atkeson, and G. Cheng. Humanoid robot learning and game playing using pc-based vision. In *2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2449–2454, 2002.
- [4] F. Delmotte and M. Egerstedt. Reconstruction of low-complexity control programs from data. In *43rd IEEE Conference on Decision and Control*, volume 2, pages 1460–1465, December 2004.
- [5] D. Goldberg and M. J. Matarić. Detecting regime changes with a mobile robot using multiple models. In *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS01)*, volume 2, pages 619–624, 2001.
- [6] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, February 1989.
- [7] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *Ninth International Conference on Machine Learning*, 1992.