Multiagent Hierarchical Learning from Demonstration

Keith Sullivan
Department of Computer Science, George Mason University
4400 University Drive, MSN 4A5, Fairfax, VA USA

Programming agent behaviors is a tedious task. Typically, behaviors are developed by repeated code, test, debug cycles. The difficulty increases in a multiagent setting due to the increased size of the design space. Density of interactions, the number of agents and the agent's heterogeneity (both capabilities and behaviors) all contribute to the larger design space. This makes *training* the agents rather than programming them highly attractive.

One training approach is *Learning from Demonstration (LfD)* in which agents learn behaviors in real-time based on provided examples from a human demonstrator. The learned behavior maps environmental features to agent action(s), and is constructed from a database of state/action examples supplied by the demonstrator. The database is constructed interactively: initially, the agent is in "training mode," where the demonstrator controls the agent. Every time the demonstrator changes the agent's behavior, the agent saves an example to the database. When the demonstrator is finished collecting examples, the agent learns the behavior, and then enters "testing mode." The demonstrator may offer corrections to the agent based on observation. These corrections add examples to the database, and the behavior is re-learned. LfD is a natural way to train agents since it closely mimics how humans teach each other. Examples include sports, music, and physical therapy.

In my LfD implementation, called Hierarchical Training of Agent Behavior (HITAB), the agents learn behaviors represented as an automaton. HITAB is a supervised machine learning approach which uses a classification algorithm to learn the transitions inside the behavior automaton.

Typically, supervised machine learning requires significant data to learn robust behaviors. This is doubly so in complex, high dimensional design spaces. However, gathering data is potentially expensive since each data point requires an experiment (physically or in simulation) conducted in real-time. HITAB's behavior representation helps reduce the number of required samples by decomposing the task into smaller, less complex tasks. In addition, these smaller tasks might require a reduced set of behaviors and/or features, thus further reducing the design space. Hence, HITAB rapidly learns complex behaviors which are simple from a machine learning perspective. While this places HITAB at the edge of machine learning, it allows novices to train an agent to perform complex behaviors without requiring detailed programming knowledge.

1 Single Agent Model

In HITAB, agents learn a hierarchical finite state automata (HFA) represented as a Moore machine where individual states correspond to agent behaviors or another HFA. An HFA is built iteratively: staring with a behavior library consisting solely of atomic behaviors (e.g., turn, go forward), the demonstrator trains a slightly more complicated behavior, which is then saved to the behavior library. The now expanded behavior library is then used to train an even more complex behavior which is then saved to the library, and so on. This process continues until the desired behavior is trained.

Formally, an HFA is a tuple $\langle S, B, F, T \rangle \in \mathcal{H}$:

- $S = \{S_1, ..., S_n\}$ is the set of *states* in the automaton where S_1 is the *start state*. One state is active at a time, designated S_t.
- $B = \{B_1, ..., B_k\}$ is the set of *basic behaviors*. Each state is associated with either a basic behavior or *another automaton* from $\mathcal{H}$ though recursion is not permitted.
- $F = \{f_1, ..., f_m\}$ is the set of observable *features* in the environment. At any given time each feature has a numerical value. The collective values of F at time t is the environment's *feature vector* $\vec{f_t} = \langle f_1, ..., f_m \rangle$. Features may describe both internal and external (world) conditions, and may be toroidal, continuous, categorical or boolean.
- $T = \vec{f_t} \times S \rightarrow S$ is the *transition function* which maps the current state S_t and the current feature vector $\vec{f_t}$ to a new state S_{t+1}.

Behaviors and features may be optionally assigned one or more parameters: rather than have a behavior called *go to the ball*, we can create a behavior called *goTo(A)*, where *A* is left unspecified. Similarly, a feature might be defined not as *distance to the ball* but as *distanceTo(B)*. If such a behavior or feature is used in an automaton, either its parameter must be bound to a specific *target* (such as "the ball"), or it must be bound to some higher-level parameter *C* of the automaton itself. Thus HFAs may themselves be parameterized.

The agent learns the transition function T of the automaton. The transition function is broken into disjoint transition functions T_j, one for each state in the model. During training, each time a state transition occurs, a tuple $\langle S_t, \vec{f_t}, S_{t+1} \rangle$

is saved into a database, consisting of the current feature vector, and the old and new states. Once enough samples are collected, HITAB builds a classifier for each state S_k based on examples of the form $\langle S_k, \vec{f}_t, S_i \rangle$. Currently, the system uses a version of the C4.5 decision tree algorithm, however, any classification algorithm may be used.

As the agent executes an HFA, the demonstrator may correct its behavior at any time. Correction first switches the agent into training mode, and the demonstrator then collects more samples with the correct behavior. Upon switching to testing mode, the classifiers are re-built, and the agent then executes the new HFA. This observe/correct loop continues until the demonstrator is satisfied with the agent's behavior.

Using this approach, we recently trained a single humanoid robot to perform a simple visual servoing task [Sullivan *et al.*, 2010]. Five computer science graduate students, with no prior experience with the robot nor HITAB, trained the robot to search for, and approach, an orange tennis ball. Four of the five students successfully trained the robots to approach the ball, starting from an arbitrary position. In addition, an expert trained the robot to perform the same behavior, but also to stop when close to the ball. Anecdotal evidence showed that training time for a hierarchical behavior was significantly less than for a monolithic behavior of searching, approaching and stopping.

2 Multiagent Model

In a multiagent setting, the size of the design space grows exponentially due to agent interactions (and any unforeseen emergent phenomena), the number of agents, and the agent's heterogeneity (both hardware and software). As a result, the number of required samples increases dramatically. The notion of an *agent hierarchy*[1] reduces the number of samples required by decomposing the training task into manageable chunks, each requiring a limited number of samples. In an agent hierarchy, agents are organized into a tree where leaf nodes are individual agents, and non-leaf nodes are *coordinator agents.* Coordinator agents are trained in a similar fashion to individual agents. Having no sensors per se, the coordinator agent's "sensor" information consists of statistical information about its children, and/or global information not available to individual agents. Coordinator HFA transitions cause children to transition, regardless of their current state. Membership in a given coordinator agent's group can be dynamic, based on directions from higher level coordinator agents.

We recently conducted a demonstration which organized a group of robots into a *homogeneous* swarm, where robots under the same coordinator robot executed the same HFA [Sullivan and Luke, submitted]. Four Pioneer robots, equipped with cameras and sonars, were trained to perform a patrol-style task: The robots wander randomly searching for an intruder. Upon detecting an intruder, a robot proceeded to "capture" the intruder by approaching it. After capture, the robots returned to a home base, then the behavior restarted with wandering. At any time, if the "boss" entered the environment, the robots would run away.

[1] Not to be confused with the agent's behavior hierarchy.

This work illustrated the need for a coordinator robot: if each robot executed the same HFA without any coordination, the resulting group behavior would be insufficient. For example, if two robots see the intruder and one robot "captures" it, the other robot will search indefinitely for the now non-existent intruder. A coordinator robot provides a solution: once one robot "captures" the intruder, all other robots are informed; thus, everyone correctly changes behavior.

3 Remaining Tasks

There are several remaining tasks. First, I am exploring the idea of *heterogeneous* teams, where agents under a given coordinator agent execute different HFAs. Heterogeneous teams allow coordination between different subgroups within a larger swarm.

The second remaining task is the notion of *unlearning*. During training, it is common for the demonstrator to make a mistake. While immediately deleting the example from the database is relatively straightforward (e.g., an "undo" button), correcting learned behaviors may be more difficult. For example, after training, the human demonstrator notices the agent performing an incorrect behavior. While the human can intervene and issue an additional datapoint, the data which caused the erroneous behavior is still in the database. Identifying this erroneous data is a remaining challenge. In a behavior hierarchy this is especially difficult due to determining not only the incorrect behavior, but the level of the hierarchy which caused the error. A possible solution includes searching the database for the "closest" example, and replacing it with the updated example. However, due to the small number of data samples to begin with, care must be taken to ensure valid or critical outlier data is not deleted.

HITAB currently uses a single decision tree as a classifier which ignores dependencies between features and also assumes linearly separable data. Other classifiers such as oblique decision trees, support vector machines (SVM) or hidden Markov models (HMM) might prove more appropriate due to their ability to account for feature interdependency and to handle non-linearly separable data. Analysis is required to determine how performance changes with different classification algorithms, and to examine potential computational complexity issues.

References

[Sullivan and Luke, submitted] Keith Sullivan and Sean Luke. Hierarchical multi-robot learning from demonstration. In *Proceedings of the Robotics: Science and Systems Conference*, (submitted).

[Sullivan *et al.*, 2010] Keith Sullivan, Sean Luke, and Vittoria Amos Ziparo. Hierarchical learning from demonstration on humanoid robots. In *Proceedings of Humanoid Robots Learning from Human Interaction Workshop*, Nashville, TN, 2010.