Hierarchical Decision Making

Matthew J. Lewis
Data Exploitation Systems
Michigan Aerospace Corporation
Ann Arbor, MI
mlewis@michaero.com

Abstract—**Decision making must be made within an appropriate context; we contend that such context is best represented by a hierarchy of states. The lowest levels of this hierarchy represent the observed raw data, or specific low-level behaviors and decisions. As we ascend the hierarchy, the states become increasingly abstract, representing higher order tactics, strategies, and over-arching mission goals.**

By representing the hierarchy using probabilistic graphical models, we can readily learn the structure and parameters that define a user's behavior by observing his activities over time—what data they use, how it is visualized, and what decisions are made. Once learned, the resulting mathematical models may be combined with the techniques of reinforcement learning to predict behavior and anticipate the needs of the user, delivering appropriate data, visualizations, and recommending optimal actions.

Keywords—decision making; hierarchical hidden Markov models; reinforcement learning.

I. Introduction

Human operators, particularly in the context of military operations, must quickly make critical decisions. Although these operators have access to unprecedented volumes of diverse data sources involving media reports, financial information, imagery, signals intelligence, and human intelligence, making decisions based on such data is confounded by many factors, including: 1) Limited human and computational resources; 2) difficulty synthesizing a coherent picture from volumes of manifold and (often) irrelevant data; 3) an inability to derive meaning from or detect structure in high dimensional data; and, 4) the randomness and uncertainty intrinsic to the real world. When a human operator is making a decision, much of this data is irrelevant and, worse, confusing.

To reduce the cognitive load of human decision makers and improve the quality of the decisions they make, we can develop frameworks—algorithms, APIs, and user interfaces—that detect what data is relevant and how it should be presented in a manner that is particular to the *decision context*. For our purposes here, a *decision context* specifies: 1) the goal, task, or mission relevant to our decision; 2) the reason the task is important (*i.e.*, a global perspective); 3) any constraints and utilities associated with the decision; 4) previous decisions that were made, as well as likely future ones; and, importantly, 5) the set of candidate decisions available to the decision maker.

In fact, human operators may not explicitly conceive of this decision context, but it nevertheless serves as a useful set of *latent* variables, which help us intelligently aggregate relevant data and determine how to best present it to the human user. We may use machine learning techniques to identify and classify decision contexts, as well as predict which data, in what formats and with what visual representations are most useful.

In this paper we advocate an approach to building flexible frameworks that identify a *decision context* that may help anticipate the types of data the user will find useful, as well as how the data should be represented and visualized. As users interact with the framework to make decisions, entering search terms, selecting data, interacting with tables and plots, and ultimately making decisions, the framework learns and adapts, improving its predictive capabilities and honing its notion of what constitutes a decision context.

II. Decision Context

A. The Importance of a Decision Context

How can we learn the decision context for a particular task or goal? We have, at the lowest level, the measured input associated with how the user is using a system to help make a decision. This can be much more than the keywords associated with a database search. It could include what elements of an interface the user clicks on, heat maps of cursor positions, *how long* a given data source is investigated, *what data* elements a user expands—even, if available via camera interfaces, what information the operator is actually looking at, and for how long. The details we record about how an operator interacts with a user interface (UI)—or, even better, how an entire population of operators interacts with a UI—will yield valuable, predictive insight into what data is useful and what data is quickly discarded by the operator, with respect to the estimated decision context. This information can be gleaned from nearly any system by using external monitoring software packages.

B. Decision Context as a Hierarchy

Making the leap from crude interaction measurements to understanding the intent of the human operator and the decisions he is trying to make is difficult—indeed, a problem at the core of machine learning and artificial intelligence.

One approach to attacking this problem is to understand the decision context as a hierarchy, wherein the lowest layers of the hierarchy represent the raw input signals—the measurements of operator interaction with the UI, and the structure and content of requested data; as we move up in the hierarchy, inputs from the lower layers are mapped to increasingly more abstract ideas—operator intent, operator confusion, mission goal, *etc.*, which together form the decision context. One tool for efficiently representing such hierarchies is the hierarchical hidden Markov model (HHMM) [1].

C. An Example

To take a concrete example, imagine driving a car. At the very lowest level, a driver is taking in visual and auditory information about obstacles and other cars on the road, and making numerous low-level decisions—*press the gas*, *press the brake, turn left*, and so forth. But these decisions are always being made in the context of a higher-order goal—say, navigating to the grocery store. And that goal, in turn, is made in the context of a yet higher order goal—needing to eat. Decisions are made at each level in the hierarchy, and influence the decisions made at the other levels.

Consider Fig. 1, which illustrates a portion of a highly simplified hierarchical hidden Markov model. Each blue node in the network represents a state of the system. These states are arranged into a hierarchy of levels. Suppose you are hungry, so that at the highest level in this hierarchy you are in the *eat* state. There is some probability you will transition to another state at this level — the *sleep* state, for example. However, more likely, because of your hunger, you will transition to a lower level in the hierarchy, perhaps to the node that represents the *go to the grocery store* state. This in turn transitions to yet a lower level, to represent increasingly specific sequences of behavior — *leave the house* followed by *drive to the store*. When behavior at one level of the hierarchy is completed (which happens when one transitions to a black node), control is returned to one level higher in the hierarchy, where it left off. Note that, in general, states transition at lower levels change much more quickly than they do at higher levels: things are changing quickly as we drive down the street, stopping at stop signs, taking right turns, etc. But all the while we are still in the *eat* state—the higher order context for our behaviors has not changed.

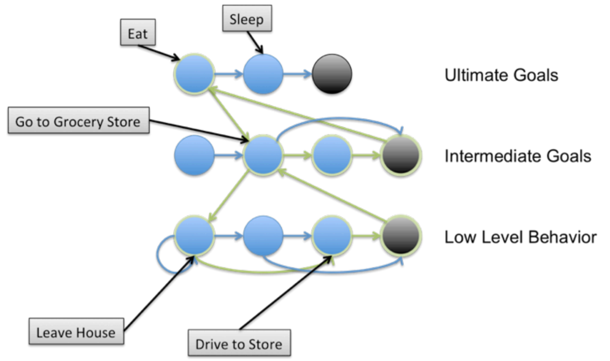

Fig 1: An example of a hierarchical hidden Markov network.

Learning the structure of such a network, as well as the probabilities associated with transitions between states and levels, can be done efficiently and in an unsupervised manner using the mathematics of probabilistic graphical models [2, 3]. We have implemented these techniques in the context of autonomous vehicle control, predictive analytics, and electronic warfare.

Using these mathematical models, we may take low level behavioral inputs and infer the higher order goals that are likely driving this behavior. Conversely, given a higher order goal, we can estimate the behaviors that will likely be used in the context of that goal. This information can be used to optimally configure a user interface, retrieve relevant data, and otherwise support operator decision making. We describe a way to achieve this optimization below.

III. Learning By Interacting with the Environment

A. From States to Actions

We have argued that describing decision context as a hierarchy provides a rich way to describe operator behavior. In a sense, it provides a flexible way of modeling the *state* of the operator — why an operator is doing something, how he is trying to accomplish it, at a strategic level, and what resources he likely needs to support the effort. This representation can determine, at each time step, what the most likely decision an operator is likely to make, based on past behavior. But this representation alone does not provide a mechansim for learning — at a given timestep, the best possible decision.

Consider again the car driving example: if we have an HHMM running, a few observations (our operator is in the car, he is driving toward the grocery store) will quickly establish the decision context (the operator needs to *eat*), and can predict that he will turn left at the upcoming intersection, because he has done so previously, and because it ultimately leads to the grocery store. But it predicts or suggests this decision only because of what has been observed in the past, not because turning left happens to be the fastest way to reach the grocery store. In order to optimize the decision making process, we harness another piece of mathematical technology, known as Reinforcement Learning.

B. Reinforcement Learning

Reinforcement Learning (RL) is a machine learning technique inspired by behavioral psychology, developed to emulate the manner in which humans learn via experience [4]. RL is concerned with teaching an agent how to interact with an environment in order to maximize a *cumulative* reward. RL has been successfully applied to problems across many domains, including industrial planning, autonomous vehicle control, pattern recognition, dynamic channel allocation in the cellular industry, and even the game of chess.

At each time step t of an RL algorithm, the agent finds itself in a state, $s_t \in S$. When in this state, it has available a number of available actions, or decisions, a_t. It selects an action according to a policy, $\pi(s,a)$, which records the probability of selecting action a given the agent is in state s, i.e., $\pi(s,a) = P(a|s)$. Because of this action, the agent transitions to a new state s_{t+1} and receives a scalar reward r_{t+1}.

The reward function is a crucial component. It may be a complex, time dependent function of the state of the agent and his environment; it is used to encode the goals of a decision making task—that is, by optimizing this function, we achieve the planning goal. Surprisingly complex behavior can emerge from the use of even simple scalar reward functions. In this instance, the reward function could be the utility of the final decision, the speed at which the decision is made, or a combination of these and other measures.

But—and this is critical—the goal of RL is not to maximize the reward received at the very next time step, but rather the total cumulative reward over all time, which we call the *return*:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \quad (1)$$

The scalar $\gamma \in [0,1]$ determines the importance of future rewards relative to near-term rewards. If $\gamma = 1$, distant future rewards are as important as near term rewards. For $0 < \gamma < 1$, we refer to R_t as the *discounted return.*

The advantage of this approach is that the reinforcement learning agents are not required to act to maximize short-term gains, but rather learn to act in complex ways to achieve objectives, even if *they must make occasionally suboptimal decisions*.

An object of fundamental interest in RL is the action-value function, which we denote by $Q(s,a)$. The action-value function records the expected discounted return:

$$Q^{\pi}(s,a) = E\{R_t | s_t = s, a_t = a\} \quad (2)$$

The superscript π indicates that the action-value function is relative to the policy π. It tells us the expected value of being in state s and taking action a. The goal of an RL agent is to learn this function by interacting with its environment. Once this function has been learned, determining an optimal policy, π, is straightforward: given we are in state s, we select the action a so that we maximize the expected value—that is, we select the optimal action $a^* = \max_a Q(s,a)$. In fact, we do not always select the optimal action, but sometimes (with probability β) select a suboptimal action. In this way, we manage to avoid the local minima in our reward functions, and may more rapidly adapt as our environment evolves in time.

We learn the Q function iteratively. We begin with an arbitrarily initialized function, $Q(s,a)$. Starting at time t, in state s_t, we select an action a using a policy derived from the function Q—i.e., we select the action with the highest expected value. Because of the action, we find ourselves in state s_{t+1}, and receive reward r_{t+1}. We update the action value function as follows by replacing the value of $Q(s_t, a_t)$ with,

$$Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (3)$$

The scalar $\alpha \in [0,1]$ is a learning rate, which specifies how quickly the system adapts.

After the update, we select another action, and the cycle continues. We can demonstrate (via practical applications and formal mathematical proofs) that this iterative procedure converges to the correct value for the action-value function Q. The policy π is implicit in the action-value function, as emphasized above: when we are in state s we select $a^* = \max_a Q(s,a)$.

Importantly, the RL algorithm does not stop learning after this convergence, for the simple reason that the environment may be changing, and our the agent's behavior may need to adapt accordingly. This occurs naturally in the context of the RL algorithms because there is some nonzero probability (β, as defined above) that we will select a non-optimal action. This ensures that we are trying new things, and although we may not always be making the optimal decision, we can avoid making decidedly poor decisions because our environment has changed from beneath us — there is always, of course, a tradeoff between achieving optimal behavior and responding quickly to a changing environment (i.e., stability vs. maneuverability), but with these methods we can parameterize and quantify the tradeoff.

C. Bringing the Pieces Together

We have discussed two distinct pieces of mathematical equipment that may be used to deal with hierarchical decision making. For the first piece, an HHMM is used to learn the decision contexts relevant to a problem. These decision contexs define a set of *states*. Given a state, and a set of actions that the operator (or the computer) may take, our second piece of technology, Reinforcement Learning, sets out to learn what decisions will lead to the best outcomes, with respect to a set of reward functions.

The fact that these two pieces share a common language—they understand they operator's state and the actions he may take, indicate they may work together effectively. Given the HHMM predicts we're in a particular state, the RL algorithms recommend an action. As a results we transition to a new state, which is estimated by the HHMM, another action is recommended, and the cycle repeats. A human operator may either execute the decision recommended by the RL algorithms, or select another action—and the system learns from the resulting state in either case.

To continue the analogy with the car: As the human operator approaches an intersection, the RL algorithm may understand that the user is in an *eat* state, attempting to go to the grocery store. Typically, at this point, according to the HHMM, the user turns left, but the RL algorithm may recommend right. As a result, the user arrives at the grocery store four minutes faster than usual. This reinforces the RL algorithm's decision to recommend that action, and in the future, it will preferentially recommend it. If at some point something, say construction, renders that driving route unmanageable, the RL algorithms will adapt accordingly, and perhaps again recommend taking a left hand turn at the intersection, instead.

IV. Conclusions

Decision making must be made within the appropriate context, and we contend that *context* is best represented by a hierarchy of states. The lowest levels of this hierarchy represent the observed raw data, or specific low-level behaviors and decisions. As we ascend the hierarchy, the states become increasingly abstract, representing higher order tactics, strategies, and over-arching mission goals.

By representing this hierarchy using probabilistic graphical models, we can readily learn the structure and parameters of a user's behavior by simply observing their activities over time—what data they use, what plots they make and use, etc. Once learned, the resulting mathematical models may be used to intelligently predict behavior, anticipate the needs of the user, and deliver the appropriate data, visualizations, and other resources before the user even knows he wants it.

Furthermore, given this hierarchical representation, we can use the mathematics of reinforcement learning to help the user make the best possible decision (or decisions) with respect to the specified reward functions.

A. Moving Forward

Although Reinforcement Learning methods have been successfully integrated with probabilistic graphical networks, which allow us to build autonomous decision making systems that learn and adapt from experience, and though these technologies have been applied to a number of disparate fields, including autonomous vehicle control, and electronic warfare, more research needs to be done to develop these into a general framework.

In order to produce a flexible framework, we must create a method for easily defining reward functions in terms of the ultimate decision goals of the system, as well as methods for labeling the states of the decision context hierarchy. In addition, for specific uses, we must develop a consistent method for ingesting data so that it may be readily used by these algorithms.

These and other challenges represent future research efforts in this area.

REFERENCES

[1] S. Fine, Y. Singer and N. Tishby, "The Hierarchical Hidden Markov Model: Analysis and Applications", Machine Learning, vol. 32, p. 41–62, 1998

[2] K. Murphy and M. Paskin. "Linear Time Inference in Hierarchical HMMs", NIPS-01 (Neural Info. Proc. Systems).

[3] K. Murphy, "Dynamic Bayesian Networks: Representation, Inference and Learning", UC Berkeley, Computer Science Division, July 2002.

[4] R. Sutton & A. Barto, “Reinforcement Learning: An Introduction”, MIT Press*, Cambridge, MA, 1998.*