Learning Visual Routines with Reinforcement Learning

Andrew Kachites McCallum
Department of Computer Science
University of Rochester
Rochester, NY 14627-0226
(716) 275-2527
FAX: (716) 461-2018
mccallum@cs.rochester.edu

Abstract

Reinforcement learning is an ideal framework to learn visual routines since the routines are made up of sequences of actions. However, such algorithms must be able to handle the hidden state (perceptual aliasing) that results from visual routine's purposefully narrowed attention.

The *U-Tree* algorithm successfully learns visual routines for a complex driving task in which the agent makes eye movements and executes deictic actions in order to weave in and out of traffic on a four-laned highway. The task involves hidden state, time pressure, stochasticity, a large world state space, and a large perceptual state space.

U-Tree uses a tree-structured representation, and is related to work on Prediction Suffix Trees (Ron, Singer, & Tishby 1994), Parti-game (Moore 1993), G-algorithm (Chapman & Kaelbling 1991), and Variable Resolution Dynamic Programming (Moore 1991). U-Tree is a direct descendant of Utile Suffix Memory (McCallum 1995c), which used short-term memory, but not selective perception. Unlike Whitehead's Lion algorithm, the algorithm handles noise, large state spaces, and uses short-term memory to uncover hidden state.

Introduction

Work in animate vision (*e.g.* (Aloimonos, Bandopadhay, & Weiss 1988; Ballard & Brown 1992)) and research on human eye movements (*e.g.* (Yarbus 1967; Ballard *et al.* 1996)) has shown that vision is not a passive process, but one in which the viewer actively shifts attention to different parts of the scene. Furthermore, indications are that the viewer chooses to focus attention exactly on those parts of the environment that are relevant to the task at hand. The study of Visual Routines (Ullman 1984) and Deictic Actions (Agre & Chapman 1987) has been an effort to formalize this approach.

This active approach to vision can greatly reduce the computational burdens of perception, but is also raises new problems. Perception becomes sequential process, instead of parallel. The agent must choose a sequence of perceptual actions to redirect its purposefully narrowed attention to the relevant parts of the environment. At each step, the agent must choose where to move the eyes, where to focus covert attention, and which perceptual computations will be performed on the available sensory data. Learning a strategy for perceptual actions becomes an integral part of learning to perform the task.

Reinforcement learning (RL) is an ideal framework in which to learn these sequences of attentional shifts because RL's chief strength is its ability to address the issues of choosing *sequences* of actions. This match between RL and visual routines has been recognized in past work (Chapman 1990; Whitehead 1992), but has only been applied with limited success.

One of the key difficulties of applying RL to visual routines, (not addressed by this previous work), is the hidden state problem (*aka* perceptual aliasing) that results from visual routine's purposefully narrowed attention. On the one hand, choosing to focus on only a specialized few features in the current "perceptual snapshot," active vision greatly reduces perceptual computation burdens; on the other hand, since a "perceptual snapshot" in isolation does not provide enough information to choose the next action, the agent must use context over time (short-term memory) in order to select its next action.

Although hidden state can appear without selective perception, hidden state is *especially* likely to occur with selective perception because the defining aspect of selective perception is directable, focused (i.e. purposefully narrowed or specialized, and thus limited) attention.

The paper presents U-Tree, a reinforcement learning algorithm that handles hidden state well, and has been used to successfully learn tasks with visual routines.

The U-Tree Algorithm

U-Tree is a new reinforcement learning algorithm that dynamically builds the agent's internal state representation. It uses a robust statistical test on reward, called a Utile Distinction test, to determine which short-term memories and state features should be used because they are relevant to the current task, and which should be ignored because they are not relevant to reward.

A key feature of the algorithm is that it can simultaneously handle both "too much sensory data" and "too little sensory data." The algorithm uses selective attention to prune an unnecessarily large perceptual state space, and it also uses short-term memory to augment a perceptual state space that, due to hidden state, is missing crucial features.

The algorithm can be understood as an effort to combine the advantages of (1) instance-based (or "memory-based") learning, from Parti-game (Moore 1993) and Nearest Sequence Memory (McCallum 1995b); (2) a utile distinction test, from Utile Distinction Memory (McCallum 1993); (3) the ability to represent variable amounts of short-term memory in different parts of state space, from Prediction Suffix Tree Learning (Ron, Singer, & Tishby 1994); and (4) the ability to select individual perceptual features, from the G-algorithm (Chapman 1989).

Like Parti-game and Nearest Sequence Memory, the algorithm makes efficient use of raw experience in order to learn quickly and discover multi-term feature conjunctions easily. Like Utile Distinction Memory, the algorithm uses a robust statistical technique in order to separate noise from task structure and build a task-dependent state space. Like Prediction Suffix Tree Learning, the algorithm uses a tree to represent short-term memory in a way that is much easier to learn than Hidden Markov models or partially observable Markov decision processes. Like the G-algorithm, the agent can select which individual features, or "dimensions of perception," to attend to.

U-Tree is a direct descendant of Utile Suffix Memory (USM) (McCallum 1995c), which also incorporates the first three techniques mentioned above; the new feature of U-Tree is it ability to divide a percept into components, choose to ignore some of those components, and thus to perform covert selective attention.

How it Works

U-Tree uses two key structures: a time-ordered chain of raw-experience-representing "instances," and a tree in which those instances are organized.

The leaves of the tree represent the internal states of the reinforcement learning agent. That is, the agent's utility estimates (Q-values) are stored in the leaves. When the agent receives an observation, it determines its current internal state by beginning at the root of the tree, and successively falling down nested tree branches until it reaches a leaf. At each non-leaf node, it chooses to fall down the branch labeled by a feature that matches its observations—working in much the same way as an exemplar is classified by a decision tree. When it reaches a leaf, the agent examines the utility estimates found in the leaf in order to choose its next action.

The key mechanism of U-Tree is the way in which it grows this tree on-line to learn a task-relevant state space. Deep parts of the tree correspond to finely distinguished parts of state space where many details and short-term memories are significant; shallow parts of the tree correspond to only loosely distinguish parts of state space where only a few features or memories are needed in order to determine what action to choose next. Another key feature of the algorithm is that both feature selection (selective perception) and short-term memory (dealing with hidden state) are represented uniformly in the same tree structure—thus capturing the inherent similarity in addressing these two problems.

The Chain and the Tree

Like all instance-based algorithms, U-Tree records each of its raw experiences. In reinforcement learning, a raw experience is a transition consisting of action-percept-reward triple, connected to its previous and successive transitions in a time-ordered chain. The linear graph at the base of Figure 1 shows an example of such a chain of instances.

U-Tree organizes, or "clusters," these instances in such a way as to explicitly control how many of their features, and how much history, to consider significant in making state-distinctions. All the instances in a cluster are used together to calculate utility estimates for that state. The structure that controls this clustering is a modified version of a finite state machine called Prediction Suffix Tree (PST) (Ron, Singer, & Tishby 1994). A PST can be thought of as an order-n Markov model, with varying n in different parts of state space. U-Tree modifies this structure to allow for the consideration of individual dimensions of perception in isolation; it also adds to PST's a notion of actions and rewards.

A leaf of the tree acts as a bucket, holding a cluster of instances that share certain features in common. The features associated with a certain leaf are determined by the nodes on the path from that leaf to the root of the tree. Each interior node of the tree introduces a new distinction; the distinctions are based on two parameters: (1) a "perceptual dimension," indicating

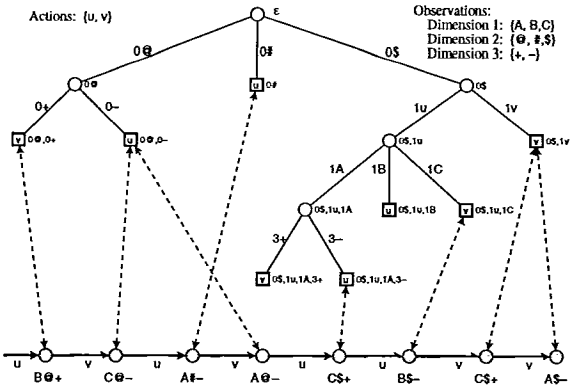

Figure 1: A U-Tree instance-chain and tree. The instance-chain is at the bottom; each instance circle is labeled by the observation received at that time step, each instance transition is labeled by the action taken to make that transition. In the tree, each branch is labeled by it's history count and perceptual dimension label; beside each node is the conjunction of features the node represents. The tree nodes drawn as squares are the agent's internal states; each contains a Q-value for each action, although the figure only shows the policy action. The dashed-arrows show which agent internal state holds each instance.

which individual feature (or dimension of perception) will be examined in order to determine which of the node's branches an instance should fall down, and (2) a "history index," indicating at how many time steps backward from the current time this feature will be examined. By using a non-zero history index, tree branches can distinguish between instances based on past features, and can thus represent short-term memory.

Figure 1 depicts a tree and instance chain based on a simple abstract task in which the agent can execute two actions (labeled 'u' and 'v'), and receives observations comprised of three features (labeled 'Dimension 1', '2', and '3'). For example, in a simple navigation task, actions 'u' and 'v' may correspond to turning left or right; 'Dimension 1' may correspond to a range sensor facing right, with 'A' indicating 'one foot', 'B' indicating 'three feet' and 'C' indicating 'far'; 'Dimension 2' may correspond to a range sensor facing left, and 'Dimension 3' may correspond to a bump sensor, with '+' indicating touching and '–' indicating open.

Consider the moment at which the agent has just experienced the sequence leading up to the starred instance, (observation 'B$–'). In order to find the leaf node in which the instance belongs, we begin at the root of the tree, and see that it has branches labeled '0@', '0#' and '0$'. Thus it adds a distinction based on the value of 'Dimension 2' at zero time steps backward. Since our instance has a '$' for 'Dimension 2', we fall down the right-hand branch. The node at the bottom of the right-hand branch has child branches labeled '1u' and '1v', and thus adds a distinction based on the action taken one time step ago. Since, one time step ago, the agent executed action 'u', (the transition coming into the starred instance is labeled with action 'u'), we fall down the left-hand branch. The node at the bottom of that left-hand branch has child branches labeled '1A', '1B' and '1C', and thus adds a distinction based on the value of 'Dimension 1' one time step ago. Since, one time step ago, the agent observation was 'C$+', (that is the label on the instance to the left of the starred instance), we fall down the right-hand branch, which is labeled '1C'. Here we are at a leaf, and we deposit the instance. Inside that leaf the agent will also find the policy action to be executed—in this case 'u'. The dashed line shows the association between the instance and the leaf.

Building the Tree

The agent constructs this tree on-line during training—beginning with no state distinctions (a single agent internal state, the root node), and selectively adding branches only where additional distinctions are needed. In order to calculate statistics about the value of an additional distinction, the agent builds a "fringe," or additional branches below what we normally consider the leaves of the tree. The instances in the fringe nodes are tested for statistically significant differences in expected future discounted reward. If the Kolmogorov-Smirnov test indicates that the instances came from different distributions, then we have determined that using this distinction will help the agent predict reward (that is, in the history case, we have found a violation of the Markov property), and these fringe nodes become "official" leaves, and the fringe is extended below them, (thus, in the history case,

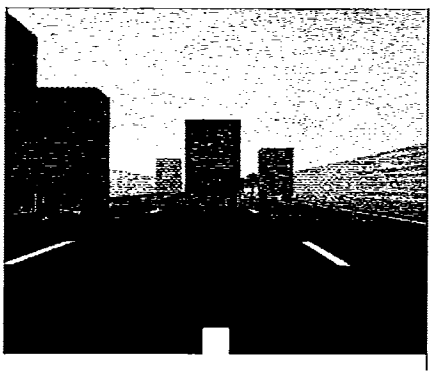

Figure 2: The perception from the driver's point of view, showing the driver's hood, the highway, and trucks.

we have turn a non-Markov state into several Markov states).

When deeper nodes are added, they are populated with instances from their parent node—the instances being properly distributed among the new children according to the additional distinction. The depth of the fringe is a configurable parameter. By using a fringe-depth greater than one, U-Tree can successfully discover useful epistatic combinations of features, in the form of multi-term conjunctions.

Further details on the algorithm are omitted here for lack of space, but are discussed in (McCallum 1995a) and (McCallum 1996).

U-Tree Learning Visual Routines for Highway Driving

U-Tree has performed well in a difficult driving task in which the agent weaves in and out of highway traffic, using actions and perceptions based on visual routines. The agent shares the road with both faster and slower trucks. The agent has control of its steering, but no control of its speed. I call the task "New York Driving."

The agent navigates using five actions, which are based on visual routines and deictic actions (Ullman 1984; Agre & Chapman 1987; Ballard *et al.* 1996). These actions are: gaze-forward-left, gaze-forward-center, gaze-forward-right, gaze-backward, and shift-to-gaze-lane. The gaze-forward visual routines begin by positioning the agent's gaze immediately in front of the agent in the left, center or right lane (relative the agent's lane), then the routines trace the gaze forward along that lane until either an truck or the horizon is reached. The gaze-backward action performs the same lane-tracing behavior, except it begins immediately behind the agent and traces backward; the lane in which the agent traces backwards is the same lane (left, center or right) in which the agent was looking previously. (Thus, for example, there is no way to shift gaze from forward right to backwards left with a single action. The agent must execute gaze-forward-left, then gaze-backward.) The shift-to-gaze-lane action is a deictic action that causes the agent's car to shift into the lane at which the agent is currently looking. This action works whether the agent is looking forward or backward. As the last step of executing the shift-to-gaze-lane action, the agent's gaze is placed at the forward center of its new lane, as if it had executed an implicit gaze-forward-center.

The agent's sensory system delivers seven dimensions of sensor information; these are: hear-horn, gaze-object, gaze-side, gaze-direction, gaze-speed, gaze-distance, gaze-refined-distance, and gaze-color. Hear-horn is one of two values, indicating whether or not a fast truck is on the agent's tail, beeping its horn. Gaze-object indicates whether the agent's gaze is pointed at a truck, the road or the shoulder. The only situations in which the agent's gaze will fall on the road is when it looks into a lane with no trucks in the specified direction. Gaze-side indicates whether the agent's gaze is left, center or right of the agent car position. Gaze-direction indicates forward or backward. Gaze-speed indicates whether the object the agent is looking at is looming or receding. When looking forward, the road and shoulder are always looming; slow trucks also loom; fast trucks recede. Gaze distance gives a rough-grained indication of the distance to the gaze point, indicating far, near or "nose," which is closer than near. "Nose" covers distances from 0 to 8 meters, "near" distances from 8 meters to 12 meters, and "far" all the further distances. Gaze-refined-distance provides the agent with a finer grain distance measure, by dividing each of the gaze-distance regions into two. Because it is in a separate "dimension," the agent can choose to have finer grain resolution in some situations, but only rough-grained resolution in others. Gaze-color is one of six values—either red, blue, yellow, white, gray or tan if looking at a truck, and yellow, white, gray or tan if looking at the road or shoulder.

The reward function delivers one of three possible rewards at each time step. Whenever the agent scrapes by a slower truck, it receives a negative reward of -10.0; whenever a fast truck is honking its horn at the agent, the agent receives a negative reward of -1.0; otherwise, the agent receives positive reward of 0.1 for making clear forward progress.

The environment has over 21,000 world states and over 2,500 sensor states; and there is also much utile

non-Markov hidden state. In order to perform this task well, a short-term memory of length three is required in some contexts—thus, a naive fixed-sized history-window approach to memory would result in an agent-internal state space of size 2500^3. Performance on this task strongly benefits from the selection of Utile Distinctions. U-Tree's solution to this task results in an agent-internal state space of only 143 states—quite a bit more manageable. It also performs better than a hand-coded policy written in two hours by an expert.

The details of the task, the sensory-motor system, and the agent's performance are discussed in in (McCallum 1995a) and (McCallum 1996).

Acknowledgments

This work has benefited from discussions with many colleagues, including: Dana Ballard, Andrew Moore, and Jonas Karlsson. This material is based upon work supported by NSF under Grant no. IRI-8903582 and by NIH/PHS under Grant no. 1 R24 RR06853-02.

References

Agre, P. E., and Chapman, D. 1987. Pengi: an implementation of a theory of activity. In *AAAI*, 268–272.

Aloimonos, J.; Bandopadhay, A.; and Weiss, I. 1988. Active vision. *International Journal of Computer Vision* 1 (4):333–356.

Ballard, D. H., and Brown, C. M. 1992. Principles of animate vision. *Computer Vision, Graphics, and Image Processing: Image Understanding* 56(1):3–21.

Ballard, D. H.; Hayhoe, M. M.; Pook, P. K.; and Rao, R. 1996. Deictic codes for the embodiment of cognition. *Brain and Behavioral Sciences.* [To appear – earlier version available as National Resource Laboratory for the study of Brain and Behavior TR95.1, January 1995, U. of Rochester].

Chapman, D., and Kaelbling, L. P. 1991. Learning from delayed reinforcement in a complex domain. In *Twelfth International Joint Conference on Artificial Intelligence.*

Chapman, D. 1989. Penguins can make cake. *AI Magazine* 10(4):45–50.

Chapman, D. 1990. *Vision, Instruction, and Action.* Ph.D. Dissertation, MIT Artificial Intelligence Laboratory.

McCallum, R. A. 1993. Overcoming incomplete perception with utile distinction memory. In *The Proceedings of the Tenth International Machine Learning Conference.* Morgan Kaufmann Publishers, Inc.

McCallum, A. K. 1995a. *Reinforcement Learning with Selective Perception and Hidden State.* Ph.D. Dissertation, Department of Computer Science, University of Rochester.

McCallum, R. A. 1995b. Instance-based state identification for reinforcement learning. In *Advances of Neural Information Processing Systems (NIPS 7)*, 377–384.

McCallum, R. A. 1995c. Instance-based utile distinctions for reinforcement learning. In *The Proceedings of the Twelfth International Machine Learning Conference.* Morgan Kaufmann Publishers, Inc.

McCallum, A. K. 1996. Learning to use selective attention and short-term memory. In *From Animals to Animats:Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior.* MIT Press.

Moore, A. W. 1991. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. *Proceedings of the Eighth International Workshop on Machine Learning* 333–337.

Moore, A. W. 1993. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. In *Advances of Neural Information Processing Systems (NIPS 6)*, 711–718. Morgan Kaufmann.

Ron, D.; Singer, Y.; and Tishby, N. 1994. Learning probabilistic automata with variable memory length. In *Proceedings Computational Learning Theory.* ACM Press.

Ullman, S. 1984. Visual routines. *Cognition* 18:97–159. (Also in: Visual Cognition, S. Pinker ed., 1985).

Whitehead, S. D. 1992. *Reinforcement Learning for the Adaptive Control of Perception and Action.* Ph.D. Dissertation, Department of Computer Science, University of Rochester.

Yarbus, A. 1967. *Eye Movements and Vision.* Plenum Press.