Reinforcement Learning and Plan Recognition

Manuela Veloso

Carnegie Mellon University
Computer Science Department

Planning - Fall 2001

Chapter 13 - Machine Learning, Tom Mitchell
Han & Veloso - Behavior HMMs
Reinforcement Learning

• Assume the world is a Markov Decision Process - transition and rewards unknown; states and actions known.

• Two objectives:
  – learning the model
  – converging to the optimal plan.
Reinforcement Learning Problem

Agent

Environment

State Reward Action

$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, where $0 \leq \gamma < 1$

Goal: Learn to choose actions that maximize

Veloso, Carnegie Mellon
Reinforcement Learning

● A variety of successful algorithms
  – Mitchell’s book “Machine Learning” (chapter 13)
  – Sutton and Barto’s book “Reinforcement Learning”
  – Kaelbling, Moore, Littman: JAIR survey

● If we can do reinforcement learning, then:
  – outcome: for every state, optimal action is known
  – **Universal plan!**
Learning Conditions

- Assume world can be modeled as a Markov Decision Process, with rewards as a function of state and action.

- **Markov assumption:**
  New states and rewards are a function only of the current state and action, i.e.,

  - $s_{t+1} = \delta(s_t, a_t)$
  - $r_t = r(s_t, a_t)$

- **Unknown and uncertain environment:**
  Functions $\delta$ and $r$ may be **nondeterministic** and are **not necessarily known** to learner.
Markov Decision Processes

- Finite set of states, $S$
- Finite set of actions, $A$
- Probabilistic state transitions, $\delta(s, a)$
- Reward for each state and action, $R(s, a)$

Model: states, actions, probabilistic transitions, rewards
Control Learning Task

- Execute actions in world,
- Observe state of world,
- Learn action policy $\pi : S \rightarrow A$
  - Maximize expected reward
    \[ E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \]
  - from any starting state in $S$.
  - $0 \leq \gamma < 1$, discount factor for future rewards
Statement of Learning Problem

• We have a target function to learn $\pi : S \rightarrow A$
• We have no training examples of the form $\langle s, a \rangle$
• We have training examples of the form $\langle \langle s, a \rangle, r \rangle$
  (rewards can be any real number)

immediate reward values $r(s, a)$
Policies

Assume deterministic world

- There are many possible policies, of course not necessarily optimal, i.e., with maximum expected reward

- There can be also several OPTIMAL policies.
Value Function

• For each possible policy \( \pi \), define an evaluation function over states

\[
V^\pi(s) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \\
\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}
\]

where \( r_t, r_{t+1}, \ldots \) are generated by following policy \( \pi \) starting at state \( s \)

• Learning task: Learn OPTIMAL policy

\[
\pi^* \equiv \arg\max_{\pi} V^\pi(s), (\forall s)
\]
Learn Value Function

- Learn the evaluation function $V^{\pi^*} - V^*$.
- Select the optimal action from any state $s$, i.e., have an optimal policy, by using $V^*$ with one step lookahead:

$$\pi^*(s) = \arg\max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$
Optimal Value to Optimal Policy

\[ \pi^*(s) = \arg\max_a [r(s, a) + \gamma V^*(\delta(s, a))] \]

A problem:

- This works well if agent knows \( \delta : S \times A \rightarrow S \), and \( r : S \times A \rightarrow \mathbb{R} \)
- When it doesn’t, it can’t choose actions this way
Define new function very similar to $V^*$

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

Learn $Q$ function - $Q$-learning

If agent learns $Q$, it can choose optimal action even without knowing $\delta$ or $r$.

$$\pi^*(s) = \arg\max_a [r(s, a) + \gamma V^*(\delta(s, a))]$$

$$\pi^*(s) = \arg\max_a Q(s, a)$$
Note that $Q$ and $V^*$ are closely related:

$$V^*(s) = \max_{a'} Q(s, a')$$

Which allows us to write $Q$ recursively as

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma V^*(\delta(s_t, a_t))$$
$$= r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a')$$

$Q$-learning actively generates examples. It "processes" examples by updating its $Q$ values. While learning, $Q$ values are approximations.
Training Rule to Learn $Q$

Let $\hat{Q}$ denote current approximation to $Q$. Then Q-learning uses the following training rule:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

where $s'$ is the state resulting from applying action $a$ in state $s$, and $r$ is the reward that is returned.
Example - Updating $\hat{Q}$

$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$

$\leftarrow 0 + 0.9 \max\{63, 81, 100\}$

$\leftarrow 90$
**Q Learning for Deterministic Worlds**

For each $s, a$ initialize table entry $\hat{Q}(s, a) \leftarrow 0$

Observe current state $s$

Do forever:

- Select an action $a$ and execute it
- Receive immediate reward $r$
- Observe the new state $s'$
- Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

- $s \leftarrow s'$

Veloso, Carnegie Mellon

15-889 – Fall 2001
Q Learning Iterations

Starts at bottom left corner - moves clockwise around perimeter;
Initially $Q(s, a) = 0$; $\gamma = 0.8$

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

<table>
<thead>
<tr>
<th>$Q(s1,E)$</th>
<th>$Q(s2,E)$</th>
<th>$Q(s3,S)$</th>
<th>$Q(s4,W)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$r + \gamma \max{Q(s5,\text{loop})} = 10 + 0.8 \cdot 0 = 10$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>$r + \gamma \max{Q(s4,W), Q(s4,N)} = 0 + 0.8 \max{10,0} = 8$</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>$r + \gamma \max{Q(s3,W), Q(s3,S)} = 0 + 0.8 \max{0,8} = 6.4$</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>
How many possible policies are there in this 3-state, 2-action deterministic world?

A robot starts in the state Mild. It moves for 4 steps choosing actions West, East, East, West. The initial values of its Q-table are 0 and the discount factor is $\gamma = 0.5$. 

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HOT</td>
<td>East 0 West 0</td>
<td>East 0 West 0</td>
<td>East 5 West 0</td>
<td>East 5 West 0</td>
</tr>
<tr>
<td>MILD</td>
<td>East 0 West 0</td>
<td>East 0 West 10</td>
<td>East 0 West 10</td>
<td>East 0 West 10</td>
</tr>
<tr>
<td>COLD</td>
<td>East 0 West 0</td>
<td>East 0 West 0</td>
<td>East 0 West 0</td>
<td>East 0 West -5</td>
</tr>
</tbody>
</table>
Why is the policy $\pi(s) = \text{West}$, for all states, better than the policy $\pi(s) = \text{East}$, for all states?

• $\pi_1(s) = \text{West}$, for all states, $\gamma = 0.5$
  \[
  V^{\pi_1}(\text{HOT}) = 10 + \gamma V^{\pi_1}(\text{HOT}) = 20.
  \]

• $\pi_2(s) = \text{East}$, for all states, $\gamma = 0.5$
  - $V^{\pi_2}(\text{COLD}) = -10 + \gamma V^{\pi_2}(\text{COLD}) = -20,$
  - $V^{\pi_2}(\text{MILD}) = 0 + \gamma V^{\pi_2}(\text{COLD}) = -10,$
  - $V^{\pi_2}(\text{HOT}) = 0 + \gamma V^{\pi_2}(\text{MILD}) = -5.$
Another Deterministic Example

\[ r(s, a) \text{ values} \]

\[ Q(s, a) \text{ values} \]

\[ V^*(s) \text{ values} \]

One optimal policy
Nondeterministic Case

What if reward and next state are non-deterministic? We redefine $V, Q$ by taking expected values

$$V^\pi(s) \equiv E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$$

$$\equiv E[\sum_{i=0}^{\infty} \gamma^i r_{t+i}]$$

$$Q(s, a) \equiv E[r(s, a) + \gamma V^*(\delta(s, a))]$$
Nondeterministic Case

Q learning generalizes to nondeterministic worlds

Alter training rule to

\[ \hat{Q}_n(s, a) \leftarrow (1 - \alpha_n) \hat{Q}_{n-1}(s, a) + \]
\[ \alpha_n [r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a')] , \]

where \( \alpha_n = \frac{1}{1 + \text{visits}_n(s, a)} \), and \( s' = \delta(s, a) \).

\( \hat{Q} \) still converges to \( Q^* \) (Watkins and Dayan, 1992)
Nondeterministic Example

S1: Unemployed
S2: Industry
S3: Grad School
S4: Academia

REWARD

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Nondeterministic Example

\[ \pi^*(s) = D, \text{ for any } s = S1, S2, S3, \text{ and } S4, \gamma = 0.9. \]

\[
V^*(S2) = r(S2, D) + 0.9 \cdot 1.0 \cdot V^*(S2)
V^*(S2) = 100 + 0.9 \cdot V^*(S2)
V^*(S2) = 1000.
\]

\[
V^*(S1) = r(S1, D) + 0.9 \cdot (1.0 \cdot V^*(S2))
V^*(S1) = 0 + 0.9 \times 1000
V^*(S1) = 900.
\]

\[
V^*(S3) = r(S3, D) + 0.9 \cdot (0.9 \cdot V^*(S2) + 0.1 \cdot V^*(S3))
V^*(S3) = 0 + 0.9 \cdot (0.9 \times 1000 + 0.1 \cdot V^*(S3))
V^*(S3) = 81000/91.
\]

\[
V^*(S4) = r(S4, D) + 0.9 \cdot (0.9 \cdot V^*(S2) + 0.1 \cdot V^*(S4))
V^*(S4) = 40 + 0.9 \cdot (0.9 \times 1000 + 0.1 \cdot V^*(S4))
V^*(S4) = 85000/91.
\]
What is the Q-value, $Q(S_2,R)$?

$$Q(S_2,R) = r(S_2,R) + 0.9 \left( 0.9 \, V^*(S_1) + 0.1 \, V^*(S_2) \right)$$

$$Q(S_2,R) = 100 + 0.9 \left( 0.9 \times 900 + 0.1 \times 1000 \right)$$

$$Q(S_2,R) = 100 + 0.9 \left( 810 + 100 \right)$$

$$Q(S_2,R) = 100 + 0.9 \times 910$$

$$Q(S_2,R) = 919.$$
Discussion

• How should the learning agent use the intermediate $Q$ values?
  – Exploration
  – Exploitation

• Scaling up in the size of the state space
  – Function approximator (neural net instead of table)
  – Generalization
  – Reuse, use of macros
  – Abstraction, learning substructure
Ongoing Research

- Partially observable state
- Continuous action, state spaces
- Learn state abstractions
- Optimal exploration strategies
- Learn and use $\hat{\delta} : S \times A \rightarrow S$
- Multiple learners - Multi-agent reinforcement learning
Behavior Recognition

If someone is acting according to a Markov Model, then can we follow its “behavior” - state / actions transitions??

• What is known?

• What is observable?

• Real data and abstracted model

• Speech recognition

Hidden Markov Models
Behavior Recognition

State the problem as a behavior membership decision.

- R acts according to a set of behaviors $B(i)$,
- O has a model of the set of possible behaviors,
- O recognizes, which $B(i)$ R is performing.

We can extend HMMs to behavior recognition.
• Observations are defined in terms of state features.
A Behavior HMM

• \( N = \{s_{initial}\} \cup \{s_{intermediate}\} \cup \{s_{accept}\} \cup \{s_{reject}\} \)

• \( M = \{o_i\} \) – the observation space

• \( A = \{a_{ij}\} \) – The state transition matrix, where:

\[
a_{ij} = Pr(S_{t+1} = s_j|S_t = s_i), 1 \leq i, j \leq N
\]

• \( B = \{b_i(o_k)\} \) – The observation probabilities

\[
b_i(o) = Pr(o|S_t = s_i), 1 \leq i \leq N
\]

• \( \pi = \{\pi_i\} \) – The initial state distribution

\[
\pi_i = Pr(S_1 = s_i), 1 \leq i \leq N
\]
Behavior Recognition

How to perform behavior recognition?

- The state is not directly observable.
- We infer the probability of being at state $s_i$,

$$\sum_i Pr(S_t = s_i) = 1, \forall i$$

- This probability gives the likelihood of $s_i$ being the actual behavioral state of the robot.
Multiple Behaviors, Multiple BHMMs

- A different BHMM is used for each type of behavior.

- Several BHMMs may show high accepting probabilities at the same time, as behaviors are not necessarily mutually exclusive.

- Given an observation sequence $O = o_1, o_2, \ldots, o_t$, what is the probability that we are in $s_i$?

$$Pr(S_t = s_i | O = o_1, o_2, \ldots, o_t, \lambda),$$

where $\lambda$ is the Hidden Markov Model parameters.
Orchestrating Multiple BHMMs

\[
Pr(S_t = s_i \mid O = o_1, o_2, \ldots, o_t, \lambda) =
\]

\[
\frac{Pr(S_t = s_i \land O = o_1, o_2, \ldots, o_t, \lambda)}{Pr(O = o_1, o_2, \ldots, o_t, \lambda)}
\]

Let \( \alpha_i(t) = Pr(S_t = s_i \land O = o_1, o_2, \ldots, o_t, \lambda) \), then:

\[
Pr(S_t = s_i \mid O = o_1, o_2, \ldots, o_t, \lambda) = \frac{\alpha_i(t)}{\sum_i \alpha_i(t)}
\]

\( \alpha_i(t) \) is computed recursively:

\[
\alpha_i(t + 1) = \sum_j \alpha_j(t) a_{ji} b_i(o_{t+1})
\]
Behavior Recognition: Issues

- Our assumption requires each behavior to be a sequence of state traversals.

- We assume that each behavior starts from the initial state and completes at the accept state.

- Our Behavior Hidden Markov Model can recognize a single execution of a behavior, provided it is instantiated at the time when the real behavior starts executing.

- Otherwise, it will be “off phase” with the actually behavior and reliable recognition has low guarantees.
Behavior Segmentation and Restart

Instantiate a new recognizer at regular intervals.

- No effort searching for segmentation points between successive executions of two behaviors, and possible fail: we ignore such points.

- **Probabilistic segmentation:** One of the recognizers will have high probability of instantiating at a point in time close to the behavior start time.

- **Granularity:** important, too sparse results in high recognizer’s probability of missing the beginning of the behavior; how frequent?
Two separate schemes to determine the removal of a BHMM.

• A *timeout* is set for each type of behavior.
  – Assumption: each behavior takes some estimated amount of time to execute (e.g. Go-To-Ball 30s)

• A BHMM is removed when it reaches a *high probability* for a reject state.
  – The robot’s behavior is not doing what the BHMM is trying to recognize.
BHMM Recognition Algorithm

- For all behavior recognizer type
  - Instantiate initial copy
  - Record start time and mark this instance as active
- Forever until done
  - Obtain object locations from vision system
  - For each active behavior recognizer
    * Compute current observation from vision data
    * Update current state probabilities using Most Like State update.
    * Find most likely state \(= mls\)
    * If \(mls \in \{\text{accept state}\}\)
      - Signal
      - Continue to next recognizer instance
    * If \(mls \in \{\text{reject state}\}\ AND \ Pr(mls) > \text{reject threshold}\)
      - Signal and mark instance as inactive
      - Continue to next recognizer instance
    * Compute elapse time for this instance. If larger than timeout threshold mark instance as inactive
  - if \((\text{current-time - last-instantiate-time}) > \text{instantiate threshold}\)
    * For all behavior recognizer type
      - Instantiate initial copy
      - Record start time and mark this instance as active
    * last-instantiate-time = current-time
Summary

- Markov model for state/action transitions.
- Markov systems with reward - goal achievement
- Markov decision processes - added actions
- Value, policy iteration
- Q-learning
- Hidden Markov models - observation and recognition.
- Later: POMDPs - Viterbi and Multi/Markov Viterbi