Reinforcement Learning

• Problem
  – Make sequence of decisions (policy) to get to goal / maximize utility

• Search Problems so far
  – Known environment
    • State space
    • Consequences of actions
    • Probability distribution of non-deterministic elements
  – Known utility / cost function
    – First compute the sequence of decisions, then execute (potentially re-compute)

• Real-World Problems
  – Environment is unknown a priori and needs to be explored
  – Utility function unknown – only examples are available for some states
    • No feedback on individual actions
    • Learn to act and to assign blame/credit to individual actions
  – Need to quickly react to unforeseen events (have learned what to do)

Reinforcement Learning

• Issues
  – Agent knows the full environment a priori vs. unknown environment
  – Agent can be passive (watch) or active (explore)
  – Feedback (i.e. rewards) in terminal states only; or a bit of feedback in any state
  – How to measure and estimate the utility of each action
  – Environment fully observable, or partially observable
  – Have model of environment and effects of action…or not

→ Reinforcement Learning will address these issues!

Markov Decision Process

• Representation of Environment:
  – finite set of states $S$
  – set of actions $A$ for each state $s \in S$

• Process
  – At each discrete time step, the agent
    • observes state $s_t \in S$
    • chooses action $a_t \in A$
  – After that, the environment
    • gives agent an immediate reward $r_t$
    • changes state to $s_{t+1}$ (can be probabilistic)

Markov Decision Process

• Model:
  – Initial state: $S_0$
  – Transition function: $T(s,a,s')$
    $T(s,a,s')$ is the probability of moving from state $s$ to $s'$ when executing action $a$.
  – Reward function: $R(s)$
    $R(s)$ is the real valued reward that the agent receives for entering state $s$.

• Assumptions
  – Markov property: $T(s,a,s')$ and $R(s)$ only depend on current state $s$, but not on any states visited earlier.
  – Extension: Function $R$ may be non-deterministic as well

Example

- Move into desired direction with prob 80%
- Move 90 degrees to left with prob 10%
- Move 90 degrees to right with prob 10%

Reward:
- In terminal states reward of +1 / -1 and agent gets “stuck”
- Each other state has a reward of -0.04.
Policy

- **Definition:**
  - A policy \( \pi \) describes which action an agent selects in each state.
  - \( a = \pi(s) \)

- **Utility**
  - For now:
    \[
    U([s_0, \ldots, s_N]) = \sum_i R(s_i)
    \]
  - Let \( P([s_0, \ldots, s_N] | \pi, s_0) \) be the probability of state sequence \([s_0, \ldots, s_N]\) when following policy \( \pi \) from state \( s_0 \)
  - Expected utility:
    \[
    U_\pi(s) = \sum U([s_0, \ldots, s_N]) P([s_0, \ldots, s_N] | \pi, s_0)
    \]
    - measure of quality of policy \( \pi \)
  - Optimal policy \( \pi^* \): Policy with maximal \( U_\pi(s) \) in each state \( s \)

Optimal Policies for Other Rewards

Utility (revisited)

- **Problem:**
  - What happens to utility value when
    - either the state space has no terminal states
    - or the policy never directs the agent to a terminal state
    - Utility becomes infinite
  - Discount factor \( 0 < \gamma < 1 \)
  - closer rewards count more than awards far in the future
  - \( U([s_0, \ldots, s_N]) = \sum_i \gamma^i R(s_i) \)
  - finite utility even for infinite state sequences

How to Compute the Utility for a given Policy?

- **Definition:**
  \( U^\pi(s) = \sum \gamma \sum_{s'} T(s, \pi(s), s') U^\pi(s') \)
- **Recursive computation:**
  \[
  U^\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^\pi(s')
  \]
- Algorithm [Policy Evaluation]:
  - \( i = 0 \);
  - \( U^\pi_i(s) = 0 \) for all \( s \)
  - repeat
    - \( i = i+1 \)
    - for each state \( s \) in \( S \) do
      - \( U^\pi_i(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U^\pi_{i-1}(s') \)
    - endfor
    - until difference between \( U^\pi_i \) and \( U^\pi_{i-1} \) small enough
  - return \( U^\pi_i \)

How to Find the Optimal Policy \( \pi^* \)?

- Is policy \( \pi \) optimal? How can we tell?
  - If \( \pi \) is not optimal, then there exists some state where
    \( \pi(s) \neq \text{argmax}_a \sum \gamma \sum_{s'} T(s, a, s') U^\pi(s') \)
  - How to find the optimal policy \( \pi^* \)?
How to Find the Optimal Policy $\pi^*$?

Algorithm [Policy Iteration]:
- repeat
  - $U^\pi = \text{PolicyEvaluation}(\pi, S, T, R)$
  - for each state $s$ in $S$ do
    - if $\max_a \sum T(s, a, s') U^\pi(s') > \sum T(s, \pi(s), s') U^\pi(s')$ then
      - $\pi(s) = \arg\max_a \sum T(s, a, s') U^\pi(s')$
    - endfor
  - until $\pi$ does not change any more
- return $\pi$

Utility $\Leftrightarrow$ Policy

Equivalence:
- If we know the optimal utility $U(s)$ of each state, we can derive the optimal policy:
  $\pi^*(s) = \arg\max_a \sum T(s, a, s') U(s')$
- If we know the optimal policy $\pi^*$, we can compute the optimal utility of each state:
  $U(s) = R(s) + \gamma \max_a \sum T(s, a, s') U(s')$
  $\Rightarrow$ Necessary and sufficient condition for optimal $U(s)$.

Value Iteration Algorithm

- Algorithm [Value Iteration]:
  - $i = 0$; $U_0(s) = 0$ for all $s$
  - repeat
    - $i = i + 1$
    - for each state $s$ in $S$ do
      - $U_i(s) = R(s) + \gamma \max_a \sum T(s, a, s') U_{i-1}(s')$
    - endfor
  - until difference between $U_i$ and $U_{i-1}$ small enough
  - return $U_i$
  $\Rightarrow$ derive optimal policy via $\pi^*(s) = \arg\max_a \sum T(s, a, s') U(s')$

Convergence of Value Iteration

- Value iteration is guaranteed to converge to optimal $U$ for $0 \leq \gamma < 1$
- Faster convergence for smaller $\gamma$

Reinforcement Learning

Assumptions we made so far:
- Known state space $S$
- Known transition model $T(s, a, s')$
- Known reward function $R(s)$
  $\Rightarrow$ not realistic for many real agents

Reinforcement Learning:
- Learn optimal policy with a priori unknown environment
- Assume fully observable environment (i.e. agent can tell it’s state)
- Agent needs to explore environment (i.e. experimentation)

Passive Reinforcement Learning

Task: Given a policy $\pi$, what is the utility function $U^\pi$?
- Similar to Policy Evaluation, but unknown $T(s, a, s')$ and $R(s)$

Approach: Agent experiments in the environment
- Trials: execute policy from start state until in terminal state.
Direct Utility Estimation

- Data: Trials of the form
  \[(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (4,3)\]

- Idea:
  Average reward over all trials for each state independently

- Why is this less efficient than necessary?
  Ignores dependencies between states

Adaptive Dynamic Programming (ADP)

- Idea:
  Run trials to learn model of environment (i.e. \(T\) and \(R\))
  Memorize \(R(s)\) for all visited states
  Estimate fraction of times action \(a\) from state \(s\) leads to \(s'\)
  Use PolicyEvaluation Algorithm on estimated model

- Problem:
  Can be quite costly for large state spaces
  For example, Backgammon has \(10^9\) states
  Learn and store all transition probabilities and rewards
  PolicyEvaluation needs to solve linear program with \(10^9\) equations and variables.

Temporal Difference (TD) Learning

- Idea:
  Do not learn explicit model of environment!
  Use update rule that implicitly reflects transition probabilities.

- Method:
  Init \(U^\pi(s)\) with \(R(s)\) when first visited
  After each transition, update with
  \[U^\pi(s) = U^\pi(s) + \alpha [R(s) + \gamma \max_{s'} U^\pi(s') - U^\pi(s)]\]

- Properties:
  No need to store model
  Only one update for each action (not full PolicyEvaluation)

Active Reinforcement Learning

- Task: In an a priori unknown environment, find the optimal policy.
  - unknown \(T(s, a, s')\) and \(R(s)\)
  - Agent must experiment with the environment.

- Naïve Approach: “Naïve Active PolicyIteration”
  - Start with some random policy
  - Follow policy to learn model of environment and use ADP to estimate utilities.
  - Update policy using \(\pi(s) \leftarrow \arg\max_a \sum_{s'} T(s, a, s') U^\pi(s')\)

- Problem:
  Can converge to sub-optimal policy!
  By following policy, agent might never learn \(T\) and \(R\) everywhere.
  Need for exploration!

Exploration vs. Exploitation

- Exploration:
  Take actions that explore the environment
  Hope: possibly find areas of the state space of higher reward

- Exploitation:
  Follow current policy
  Guaranteed to get certain expected reward

- Approach:
  Sometimes take random steps
  Bonus reward for states that have not been visited often yet

Q-Learning

- Problem: Agent needs model of environment to select action via
  \[\arg\max_{s'} T(s, a, s') U^\pi(s')\]

- Solution: Learn action utility function \(Q(a, s)\), not state utility function \(U(s)\).
  Define \(Q(a, s)\) as
  \[U(s) = \max_a Q(a, s)\]
  - Bellman equation with \(Q(a, s)\) instead of \(U(s)\)
  - TD-Update with \(Q(a, s)\) instead of \(U(s)\)

- Result: With Q-function, agent can select action without model of environment
  \[\arg\max_a Q(a, s)\]
Function Approximation

- **Problem:**
  - Storing Q or U,T,R for each state in a table is too expensive, if number of states is large
  - Does not exploit “similarity” of states (i.e. agent has to learn separate behavior for each state, even if states are similar)

- **Solution:**
  - Approximate function using parametric representation
  - For example: \( U(s) = \theta \cdot \Phi(s) \)
    - \( \Phi(s) \) is feature vector describing the state
      - “Material values” of board
      - Is the queen threatened?
      - …

Q-Learning Illustration

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>+1</td>
</tr>
<tr>
<td>2</td>
<td>Q(up(1,2))</td>
<td>Q(right(1,2))</td>
<td>Q(down(1,2))</td>
<td>Q(left(1,2))</td>
</tr>
<tr>
<td>1</td>
<td>Q(up(1,1))</td>
<td>Q(right(1,1))</td>
<td>Q(down(1,1))</td>
<td>Q(left(1,1))</td>
</tr>
<tr>
<td>2</td>
<td>Q(up(2,1))</td>
<td>Q(right(2,1))</td>
<td>Q(down(2,1))</td>
<td>Q(left(2,1))</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>-1</td>
</tr>
</tbody>
</table>