Reinforcement Learning

Supervised Learning:
- Training examples: \((x,y)\)
- Direct feedback \(y\) for each input \(x\)

Reinforcement Learning
- Sequence of decisions with eventual feedback
- No teacher that critiques individual actions
- Learn to act and to assign blame/credit to individual actions
- Examples
  - when playing a game, only after many actions final result: win, loss, or draw.
  - Robot fetching bagels from bakery
  - Navigating the Web for collecting all CS pages
  - Control problems (reactor control)

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

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 \(s_t \in S\) and then
  - 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)

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 a 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

Each other state has a reward of -0.04.

- move into desired direction with prob 80%
- move 90 degrees to left with prob 10%
- move 90 degrees to right with prob 10%
**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
  - Solution
    - Discount factor \( 0 < \gamma < 1 \)
    - \( U([s_0, \ldots, s_N]) = \sum_i \gamma^i R(s_i) \)
    - Finite utility even for infinite state sequences

**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

- Solution
  - Discount factor \( 0 < \gamma < 1 \)
  - \( U([s_0, \ldots, s_N]) = \sum_i \gamma^i R(s_i) \)
  - Closer rewards count more than awards far in the future
  - Finite utility even for infinite state sequences

**Bellman Update (for fixed \( \pi \))**

- **Goal:** Solve set of \( n = |S| \) equations (one for each state)
  - \( U_\pi(s_0) = R(s_0) + \gamma \sum \gamma \ T(s_0, \pi(s), s') U_\pi(s') \)
  - \( \ldots \)
  - \( U_\pi(s) = R(s) + \gamma \sum \gamma \ T(s, \pi(s), s') U_\pi(s') \)

- **Algorithm [Policy Evaluation]:**
  - \( i = 0 \); \( U_\pi^0(s) = 0 \) for all \( s \)
  - repeat
    - \( i = i + 1 \)
    - for each state \( s \) in \( S \) do
      - \( U_\pi^i(s) = R(s) + \gamma \sum \gamma \ T(s, \pi(s), s') U_\pi^{i-1}(s') \)
    - 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
    \( \pi(s) \neq \arg \max_a \sum \gamma \ T(s, a, s') U_\pi(s') \)
  - How to find the optimal policy \( \pi^* \)?

<table>
<thead>
<tr>
<th>State</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>←</td>
<td>←</td>
<td>←</td>
<td>←</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>←</td>
<td>←</td>
<td>←</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>←</td>
<td>←</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>←</td>
</tr>
</tbody>
</table>
How to Find the Optimal Policy $\pi^*$?

Algorithm [Policy Iteration]:
- repeat
  - $U^\pi = \text{PolicyEvaluation}(\pi, S, T, R)$
  - for each state $s$ in $S$
    - If $\max_a \sum_{s'} T(s, a, s') U^\pi(s') > \sum_{s'} T(s, \pi(s), s') U^\pi(s')$ then
      - $\pi(s) = \arg\max_a \sum_{s'} T(s, a, s') U^\pi(s')$
    - 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:
  $\pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s') U(s')$.
- If we know the optimal policy $\pi^*$, we can compute the optimal utility of each state:
  $U(s) = R(s) + \gamma \sum_{s'} T(s, a, s') U(s')$.

Bellman Equation:
$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s')$.

$\Rightarrow$ 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 $S$
      - $U^i(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U^{i-1}(s')$
    - endfor
  - until difference between $U^i$ and $U^{i-1}$ small enough
  - return $U^i$

derive optimal policy via $\pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s') U(s')$.

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 its 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) -0.04 
  - (1,2) -0.04 
  - (1,3) -0.04
  - (1,2) -0.04 
  - (1,3) -0.04 
  - (2,3) -0.04 
  - (3,3) -0.04 
  - (3,2) -0.04 
  - (3,3) -0.04 
  - (4,3) 1.0
  - (1,1) -0.04 
  - (1,2) -0.04 
  - (1,3) -0.04 
  - (2,3) -0.04 
  - (3,3) -0.04 
  - (3,2) -0.04 
  - (3,3) -0.04 
  - (4,3) 1.0 
  - (1,1) -0.04 
  - (2,1) -0.04 
  - (3,1) -0.04 
  - (3,2) -0.04 
  - (4,2) 1.0

- **Idea:**
  - Average reward over all trials for each state independently
  - Supervised Learning Problem

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

\[ U_\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U_\pi(s') \]

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 $10^{50}$ states
  - PolicyEvaluation needs to solve linear program with $10^{50}$ 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
    \[ U_\pi(s) = U_\pi(s) + \alpha [R(s) + \gamma U_\pi(s') - U_\pi(s)] \]
  - $\alpha$ is learning rate. $\alpha$ should decrease slowly over time, so that estimates stabilize eventually.

- **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 $\pi(s) \leftarrow \text{argmax}_a \sum_{s'} T(s, a, s') U_\pi(s')$

- **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 in the state space of higher reward
  - Problem: possibly take suboptimal steps

- **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
  \[ \text{argmax}_a \sum_{s'} T(s, a, s') U_\pi(s') \]

- **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
  \[ \text{argmax}_a Q(a,s) \]
### 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: $U(s) = \psi^T \cdot \Phi(s)$
    - $\Phi(s)$ is feature vector describing the state
    - “Material values” of board
    - Is the queen threatened?
    - …