Robot Reinforcement Learning
Robot Decision Making and Planning

"His path-planning may be sub-optimal, but it's got flair."
Robot Decision Making and Planning

Robots need to make various decisions and construct different plans, for example:

- Reactive decision making
- Task planning
- Motion planning (e.g., for robotic arms)
- Path planning (e.g., for mobile robotics)
Robot Decision Making and Planning

Robots need to make decisions and construct plans:

- Reactive decision making
- Task planning
- Motion planning (e.g., for robotic arms)
- Path planning (e.g., for mobile robotics)

Perspectives to consider robot planning methods

- Reactive (single-time) decision making versus sequential planning
- Certain and uncertain scenarios
- Observable versus partially observable space

We will focus on sequential decision making
Decision Making/Planning Types

- **Deterministic, fully observable**
  - Agent knows exactly which state it will be in
  - Agent action is executed as expected

- **Dynamic, partially observable**
  - Observations provide new information about current state with uncertainty
  - Robot actions may not be successfully executed

- **Non-observable**
  - Agent may have no idea where it is
Example: Vacuum World

- Observable, start in #5.
  - [Right; Suck]
- Non-observable, start in: 
  \{1; 2, 3; 4; 5; 6; 7; 8\}
  e.g., Right goes to 
  \{2; 4; 6; 8\}
  - [Right; Suck; Left; Suck]
- Partially observable, start in #5, suck can dirty a clean carpet, local sensing only
  - [Right; if dirt then Suck]
- Possible actions:
  left, right, suck
Example: Vacuum World (observable)

- States: cross product of dirtiness and robot locations
- Successor function: Left/Right changes location, Suck changes dirtiness
- Actions: Left, Right, Suck, NoOp
- Goal test: no dirt
- Path cost: 1 per action (0 for NoOp)
  (Also called reward, penalty, or utility)
Example: Vacuum World (non-observable)
Planning with Uncertainty

• How about in an uncertain scenario?
  • Uncertainty in action outcomes
  • Uncertainty in state of knowledge
  • Any combination of the two
Planning with Uncertainty

- Solutions based on decision tree
Planning with Uncertainty

• Utility (i.e., reward or cost) function associates a real-valued utility (reward or cost) with each outcome (state or state-action pair)

• With utilities, we can compute and optimize expected utilities for planning under uncertainty
  • For example, the expected utility of decision $d$ in the state $s$ is defined as

  \[ EU(d) = \sum_{s \in S} \Pr_d(s)U(s) \]

• The principle of *maximum expected utility* states that the optimal decision under uncertainty is the one that has greatest expected utility
Reinforcement Learning for Planning

Two fundamental problems in sequential decision making

• Reinforcement Learning:
  • The environment is initially unknown
  • The agent interacts with the environment
  • The agent improves its policy

• Planning:
  • A model of the environment is known
  • The agent performs computations with its model (without any external interaction)
Reinforcement Learning

• Branches of Machine Learning
Characteristics of Reinforcement Learning

• What makes reinforcement learning different from other machine learning paradigms?
  • There is no supervisor, only a reward signal
  • Feedback is delayed, not instantaneous
  • Time really matters (sequential, non i.i.d data)
  • Agent's actions affect the subsequent data it receives
Applications of Reinforcement Learning
Applications of Reinforcement Learning
Reinforcement Learning

• Reinforcement learning is based on the reward hypothesis

• Definition (Reward Hypothesis): All goals can be described by the maximization of expected cumulative reward
  • A reward $R_t$ is a scalar feedback signal
  • Indicates how well agent is doing at step $t$
  • The agent's job is to maximize cumulative reward

• Actions may have long term consequences, thus reward may be delayed
  • It may be better to sacrifice immediate reward to gain more long-term reward
Agent and Environment

At each step $t$ the agent:
- Executes action $A_t$
- Receives observation $O_t$
- Receives scalar reward $R_t$

The environment:
- Receives action $A_t$
- Emits observation $O_{t+1}$
- Emits scalar reward $R_{t+1}$

$t$ increments at env. step

The slides of RL are from Dr. David Silver: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
History and State

- The **history** is the sequence of observations, actions, rewards
  \[ H_t = O_1, R_1, A_1, ..., A_{t-1}, O_t, R_t \]

- i.e. all observable variables up to time \( t \)
- i.e. the sensorimotor stream of a robot or embodied agent
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards
- **State** is the information used to determine what happens next
- Formally, state is a function of the history:
  \[ S_t = f(H_t) \]
Environment State

- The environment state $S^e_t$ is the environment’s private representation.
- i.e. whatever data the environment uses to pick the next observation/reward.
- The environment state is not usually visible to the agent.
- Even if $S^e_t$ is visible, it may contain irrelevant information.
Agent State

- The agent state $S_t^a$ is the agent’s internal representation
- i.e. whatever information the agent uses to pick the next action
- i.e. it is the information used by reinforcement learning algorithms
- It can be any function of history:

$$S_t^a = f(H_t)$$
Information State

An information state (a.k.a. Markov state) contains all useful information from the history.

Definition

A state $S_t$ is Markov if and only if

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, ..., S_t]$$

- “The future is independent of the past given the present”

$$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$$

- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future
- The environment state $S_t^e$ is Markov
- The history $H_t$ is Markov
Fully Observable Environments

Full observability: agent directly observes environment state

$$O_t = S_t^a = S_t^e$$

- Agent state = environment state = information state
- Formally, this is a Markov decision process (MDP)
Markov Property

“The future is independent of the past given the present”

**Definition**

A state $S_t$ is **Markov** if and only if

$$\mathbb{P} [S_{t+1} \mid S_t] = \mathbb{P} [S_{t+1} \mid S_1, \ldots, S_t]$$

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future
State Transition Matrix

For a Markov state $s$ and successor state $s'$, the state transition probability is defined by

$$ P_{ss'} = \mathbb{P} \left[ S_{t+1} = s' \mid S_t = s \right] $$

State transition matrix $P$ defines transition probabilities from all states $s$ to all successor states $s'$,

$$ P = \begin{bmatrix}
    P_{11} & \cdots & P_{1n} \\
    \vdots \\
    P_{n1} & \cdots & P_{nn}
\end{bmatrix} $$

where each row of the matrix sums to 1.
Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, \ldots$ with the Markov property.

**Definition**

A Markov Process (or Markov Chain) is a tuple $\langle S, \mathcal{P} \rangle$

- $S$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability probability matrix,

$$
\mathcal{P}_{ss'} = \mathbb{P} [S_{t+1} = s' \mid S_t = s]
$$
Markov Process: Example
Markov Process: Example

Sample episodes for Student Markov Chain starting from $S_1 = C_1$

- $S_1, S_2, \ldots, S_T$
- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

\[ P = \begin{bmatrix}
    C1 & C2 & C3 & Pass & Pub & FB & Sleep \\
    C1 & 0.5 & 0.8 & 0.6 & 0.4 & 0.5 & 0.2 \\
    C2 & 0.2 & 0.4 & 0.4 & 0.4 & 0.1 & 1.0 \\
    C3 & 0.1 & 0.4 & 0.4 & 0.9 & 1.0 & 1.0 \\
    Pass & 0.2 & 0.4 & 0.4 & 0.6 & 0.4 & 1.0 \\
    Pub & 0.2 & 0.4 & 0.4 & 0.6 & 0.4 & 1.0 \\
    FB & 0.2 & 0.4 & 0.4 & 0.6 & 0.4 & 1.0 \\
    Sleep & 0.2 & 0.4 & 0.4 & 0.6 & 0.4 & 1.0 \\
\end{bmatrix} \]
A Markov reward process is a Markov chain with values.

**Definition**

A *Markov Reward Process* is a tuple $\langle S, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $S$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- $\gamma$ is a discount factor, $\gamma \in [0, 1]$
Markov Reward Process: Example
Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with decisions. It is an environment in which all states are Markov.

**Definition**

A Markov Decision Process is a tuple \( \langle S, A, P, R, \gamma \rangle \)

- \( S \) is a finite set of states
- \( A \) is a finite set of actions
- \( P \) is a state transition probability matrix,
  \( P_{ss'}^a = P[S_{t+1} = s' \mid S_t = s, A_t = a] \)
- \( R \) is a reward function,
  \( R_s^a = E[R_{t+1} \mid S_t = s, A_t = a] \)
- \( \gamma \) is a discount factor \( \gamma \in [0, 1] \).
Markov Decision Process: Example
MDP and Reinforcement Learning

- Markov decision processes formally describe an environment for reinforcement learning
- Where the environment is fully observable
- i.e. The current state completely characterises the process
- Almost all RL problems can be formalised as MDPs, e.g.
  - Optimal control primarily deals with continuous MDPs
  - Partially observable problems can be converted into MDPs

- An RL agent may include one or more of these components:
  - Policy: agent’s behaviour function
  - Value function: how good is each state and/or action
  - Model: agent’s representation of the environment
## Components of an RL Agent

### Policy

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A policy</strong> $\pi$ is a distribution over actions given states,</td>
</tr>
<tr>
<td>$\pi(a</td>
</tr>
</tbody>
</table>

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent), $A_t \sim \pi(\cdot|S_t), \forall t > 0$
- Deterministic policy: $a = \pi(s)$  \textit{(and greedy)}
- Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$
Components of an RL Agent

Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

\[ v_\pi(s) = \mathbb{E}_\pi \left[ R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right] \]

Model

- A model predicts what the environment will do next
- \( \mathcal{P} \) predicts the next state
- \( \mathcal{R} \) predicts the next (immediate) reward, e.g.

\[ \mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a] \]
\[ \mathcal{R}_s^a = \mathbb{E} [R_{t+1} \mid S_t = s, A_t = a] \]
Maze Example

- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent’s location
Maze Example: Policy

Arrows represent policy $\pi(s)$ for each state $s$
Maze Example: Value Function

Numbers represent value $v_\pi(s)$ of each state $s$
Maze Example: Model

- Agent may have an internal model of the environment
- Dynamics: how actions change the state
- Rewards: how much reward from each state
- The model may be imperfect

- Grid layout represents transition model $P_{ss'}^a$
- Numbers represent immediate reward $R_s^a$ from each state $s$ (same for all $a$)
Model-Based and Model-Free RL

- Model-Free RL
  - No model
  - **Learn** value function (and/or policy) from experience
- Model-Based RL
  - **Learn** a model from experience
  - **Plan** value function (and/or policy) from model

Model-based RL  Model-free RL
Model-Based and Model-Free RL

- **Model-Free RL**
  - **No model**
  - **Learn** value function (and/or policy) from experience

- **Model-Based RL**
  - Learn a model from experience
  - **Plan** value function (and/or policy) from model

[Diagram showing the cycles of Model-based RL and Model-free RL]

**Model-based RL**

**Model-free RL**
Q-Learning: Integrating Learning and Planning

• We’re going to learn model-free RL (though knowing a model also works)
• We will focus on finding a way to estimate the value function directly
  • The value function is not necessary to directly associate with the world and represent the world
• The value function is the **Q-function**
  • A recursive way to approximate a value function
• The process of estimation the Q-function is called **Q-learning**
  • Q-Learning integrates learning and planning
Q-Learning Basics

• Given a sequence of states, actions, and rewards defined by a MDP:

\[ s_0 a_0 r_0 s_1 a_1 r_1 s_2 a_2 r_2 s_3 a_3 r_3 ... s_k a_k r_k ... \]

we define a unit of experience \(< s_k a_k r_k s_{k+1}>\)

• At each step \(s\), choose the action \(a\) which maximizes the Q-function \(Q(s, a)\)
  
  • \(Q\) is the estimated value function
  
  • It tells us how good an action is given a certain state

  • \(Q(s, a) = \text{immediate reward for making an action} + \text{best value (Q) for the resulting (future) state}\)
Q-Learning Formal Definition

• Q-function learning has a recursive definition:

\[
Q(s, a) = r(s, a) + \gamma \max_a Q(s', a')
\]

\[
r(s, a) = \text{Immediate reward}
\]

\[
\gamma = \text{relative value of delayed vs. immediate rewards (0 to 1)}
\]

\[
s' = \text{the new state after action } a
\]

\[
a, a' : \text{actions in states } s \text{ and } s', \text{ respectively}
\]

Selected action:

\[
\pi(s) = \arg\max_a Q(s, a)
\]

• Q-learning is about maintaining and updating the table of Q-values, called Q-table.
  • Only updates Q-values related to the state-action pairs that are visited
Q-Learning Algorithm

• The Q-learning algorithm is also recursive:
  • Consider the unit experience \( <s_k, a_k, r_k, s_{k+1}> \)

For each state-action pair \((s, a)\), initialize the table entry \( \hat{Q}(s, a) \) to zero
Observe the 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:
    \[ \hat{Q}(s, a) = r + \gamma \max_{a'} \hat{Q}(s', a') \]
  --- \( s = s' \)
Q-Learning Example

Example Problem

\[ \gamma = .5, \ r = 100 \text{ if moving into state } s6, \ 0 \text{ otherwise} \]
Q-Learning Example
Q-Learning Example

```
<table>
<thead>
<tr>
<th>State, Action</th>
<th>Q Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1, a12</td>
<td>0</td>
</tr>
<tr>
<td>s1, a14</td>
<td>0</td>
</tr>
<tr>
<td>s2, a21</td>
<td>0</td>
</tr>
<tr>
<td>s2, a23</td>
<td>0</td>
</tr>
<tr>
<td>s2, a25</td>
<td>0</td>
</tr>
<tr>
<td>s3, a32</td>
<td>0</td>
</tr>
<tr>
<td>s3, a36</td>
<td>0</td>
</tr>
<tr>
<td>s4, a41</td>
<td>0</td>
</tr>
<tr>
<td>s4, a45</td>
<td>0</td>
</tr>
<tr>
<td>s5, a54</td>
<td>0</td>
</tr>
<tr>
<td>s5, a52</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Current Position: Red
Available actions: a12, a14
Chose a12
Q-Learning Example

Update $\hat{Q}(s, a)$:
\[
\hat{Q}(s, a) = r + \gamma \max_{a'} \hat{Q}(s', a')
\]

<table>
<thead>
<tr>
<th>s1, a12</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1, a14</td>
<td>0</td>
</tr>
<tr>
<td>s2, a21</td>
<td>0</td>
</tr>
<tr>
<td>s2, a23</td>
<td>0</td>
</tr>
<tr>
<td>s2, a25</td>
<td>0</td>
</tr>
<tr>
<td>s3, a32</td>
<td>0</td>
</tr>
<tr>
<td>s3, a36</td>
<td>0</td>
</tr>
<tr>
<td>s4, a41</td>
<td>0</td>
</tr>
<tr>
<td>s4, a45</td>
<td>0</td>
</tr>
<tr>
<td>s5, a54</td>
<td>0</td>
</tr>
<tr>
<td>s5, a52</td>
<td>0</td>
</tr>
</tbody>
</table>

Current Position: Red
Available actions: a21, a25, a23

Update $\hat{Q}(s1, a12)$:
\[
\hat{Q}(s1, a12) = r + .5 \times \max(Q(s2, a21), Q(s2, a25), Q(s2, a23)) = 0
\]
Q-Learning Example

<table>
<thead>
<tr>
<th>State, Action</th>
<th>Q-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1, a12</td>
<td>0</td>
</tr>
<tr>
<td>s1, a14</td>
<td>0</td>
</tr>
<tr>
<td>s2, a21</td>
<td>0</td>
</tr>
<tr>
<td>s2, a23</td>
<td>0</td>
</tr>
<tr>
<td>s2, a25</td>
<td>0</td>
</tr>
<tr>
<td>s3, a32</td>
<td>0</td>
</tr>
<tr>
<td>s3, a36</td>
<td>0</td>
</tr>
<tr>
<td>s4, a41</td>
<td>0</td>
</tr>
<tr>
<td>s4, a45</td>
<td>0</td>
</tr>
<tr>
<td>s5, a54</td>
<td>0</td>
</tr>
<tr>
<td>s5, a52</td>
<td>0</td>
</tr>
</tbody>
</table>

Current Position: Red

Available actions: a21, a25, a23

Chose a23
**Q-Learning Example**

\[
\hat{Q}(s, a) = r + \gamma \max_a' \hat{Q}(s', a')
\]

**Update Q(s2, a23)**

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1, a12</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s1, a14</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s2, a21</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s2, a23</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s2, a25</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s3, a32</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s3, a36</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s4, a41</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s4, a45</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s5, a54</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s5, a52</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Current Position: Red

Available actions: a32, a36

Update Q(s2, a23):

\[
Q(s2, a23) = r + 0.5 \times \max(Q(s3, a32), Q(s3, a36))
\]

\[
= 0
\]
Q-Learning Example

**Next Move**

<table>
<thead>
<tr>
<th>State, Action</th>
<th>Q-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1, a12</td>
<td>0</td>
</tr>
<tr>
<td>s1, a14</td>
<td>0</td>
</tr>
<tr>
<td>s2, a21</td>
<td>0</td>
</tr>
<tr>
<td>s2, a23</td>
<td>0</td>
</tr>
<tr>
<td>s2, a25</td>
<td>0</td>
</tr>
<tr>
<td>s3, a32</td>
<td>0</td>
</tr>
<tr>
<td>s3, a36</td>
<td>0</td>
</tr>
<tr>
<td>s4, a41</td>
<td>0</td>
</tr>
<tr>
<td>s4, a45</td>
<td>0</td>
</tr>
<tr>
<td>s5, a54</td>
<td>0</td>
</tr>
<tr>
<td>s5, a52</td>
<td>0</td>
</tr>
</tbody>
</table>

**Current Position:** Red

Available actions: a32, a36
Chose a36

**Diagram:**

- States: s1, s2, s3, s4, s5, s6
- Actions: a12, a14, a21, a23, a25, a32, a36, a41, a45, a52, a54, a56
- End state: s6
Q-Learning Example

\[ \hat{Q}(s, a) = r + \gamma \max_{a'} \hat{Q}(s', a') \]

**Update Q(s3, a36)**

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Q Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1, a12</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s1, a14</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s2, a21</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s2, a23</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s2, a25</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s3, a32</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s3, a36</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>s4, a41</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s4, a45</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s5, a54</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s5, a52</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Current Position:** Red

**FINAL STATE!**

Update Q(s3, a36):
\[ Q(s3, a36) = r = 100 \]
Q-Learning Example

New episode

Update $Q(s_1, a_{12})$

<table>
<thead>
<tr>
<th>s1, a12</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1, a14</td>
<td>0</td>
</tr>
<tr>
<td>s2, a21</td>
<td>0</td>
</tr>
<tr>
<td>s2, a23</td>
<td>0</td>
</tr>
<tr>
<td>s2, a25</td>
<td>0</td>
</tr>
<tr>
<td>s3, a32</td>
<td>0</td>
</tr>
<tr>
<td>s3, a36</td>
<td>100</td>
</tr>
<tr>
<td>s4, a41</td>
<td>0</td>
</tr>
<tr>
<td>s4, a45</td>
<td>0</td>
</tr>
<tr>
<td>s5, a54</td>
<td>0</td>
</tr>
<tr>
<td>s5, a52</td>
<td>0</td>
</tr>
</tbody>
</table>

Current Position: Red

Available actions: $a_{21}, a_{25}, a_{23}$

Update $Q(s_1, a_{12})$:

$$Q(s_1, a_{12}) = r + 0.5 \times \max(Q(s_2, a_{21}), Q(s_2, a_{25}), Q(s_2, a_{23}))$$

$$= 0$$
**Q-Learning Example**

### Update Q(s2, a23)

<table>
<thead>
<tr>
<th>State (s)</th>
<th>Action (a)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1, a12</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s1, a14</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s2, a21</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s2, a23</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>s2, a25</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s3, a32</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s3, a36</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>s4, a41</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s4, a45</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s5, a54</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>s5, a52</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Current Position:** Red

**Available actions:** a32, a36

**Update Q(s2, a23):**

\[
\hat{Q}(s, a) = r + \gamma \max_{a'} \hat{Q}(s', a')
\]

- \( Q(s2, a23) = r + .5 \times \max(Q(s3, a32), Q(s3, a36)) \)
- \( = 0 + .5 \times 100 = 50 \)
Q-Learning Example

Final State (after many iterations)

<table>
<thead>
<tr>
<th>States</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1, a12</td>
<td>25</td>
</tr>
<tr>
<td>s1, a14</td>
<td>25</td>
</tr>
<tr>
<td>s2, a21</td>
<td>12.5</td>
</tr>
<tr>
<td>s2, a23</td>
<td>50</td>
</tr>
<tr>
<td>s2, a25</td>
<td>25</td>
</tr>
<tr>
<td>s3, a32</td>
<td>25</td>
</tr>
<tr>
<td>s3, a36</td>
<td>100</td>
</tr>
<tr>
<td>s4, a41</td>
<td>12.5</td>
</tr>
<tr>
<td>s4, a45</td>
<td>50</td>
</tr>
<tr>
<td>s5, a54</td>
<td>25</td>
</tr>
<tr>
<td>s5, a52</td>
<td>25</td>
</tr>
<tr>
<td>s5, a56</td>
<td>100</td>
</tr>
</tbody>
</table>
Q-Learning Example
RL Challenges: State/Action Representation

• Pole-balancing
  • Move car left/right to keep the pole balanced

• State representations
  • Position and velocity of car
  • Angle and angular velocity of pole
  • They are all continuous variables

• Action representations
  • Move the cart left or right with certain speed
  • Speed is also continuous variables

• Straightforward solutions
  • Coarse discretization of 3 state variables: left, center, right
  • Course discretization of speed as well
RL Challenges: Reward Design

• Rewards
  • Indicate what we want to accomplish
  • NOT how we want to accomplish it

• Robot in a maze
  • Episodic task
  • +1 when out, 0 for each step

• Chess
  • GOOD: +1 for winning, -1 losing
  • BAD: +0.25 for taking opponent’s pieces

• It is hard to design rewards for general tasks
  • An active research topic is reward learning