CS 378: Autonomous Intelligent Robotics FRI-II

Instructor: Jivko Sinapov
http://www.cs.utexas.edu/~jsinapov/teaching/cs378_fall2016/
Reinforcement Learning (Part 2)

A maze represents the environment, with an internal state model and a learning agent making actions and receiving rewards. Parameters such as learning rate $\alpha$, inverse temperature $\beta$, and discount rate $\gamma$ are mentioned.
Announcements
Volunteers needed for robot study

Sign up sheet here:

https://docs.google.com/spreadsheets/d/1Gr2GqlPt8kdTJwlZ3FerxU0J8oIoGt0pEbeR37iCHqY/edit#gid=0

Further details will be made available on Canvas via an announcement
FAI Talk this Friday

“Turning Assistive Machines into Assistive Robots”

Brenna Argall
Northwestern University

Friday, Sept. 9th, 11 am @ GDC 6.302

[https://www.cs.utexas.edu/~ai-lab/fai/] or google “fai ut cs”
Robotics Seminar Series Talk

“Learning from and about humans using an autonomous multi-robot mobile platform”

Jivko Sinapov

UT Austin

Wed., Sept. 7th, 3 pm @ GDC 5.302
Robot Training

- Sign up for a robot training session next week at: https://docs.google.com/spreadsheets/d/1kz6QMPa-xkdFQNyV0Biif913JKf73GIUGx9bVSLo_R8/edit?usp=sharing

- Link will be posted as Announcement on Canvas
Preliminary Project “Presentations”

- Date: **September 13\(^{th}\)**
- Form groups of 2-3 prior to the date
- Be prepared to talk about 2-3 project ideas for 5-10 minutes
- Email me your group info, i.e., who is in it
Project Ideas
Project Idea: Improve the robot's grasping ability

- Currently, the robot does not “remember” which grasps succeeded and which failed
- If the robot were to log the context of the grasp (e.g., the position of the gripper relative to the object's point cloud) and the outcome, it could incrementally learn a model to predict the outcome given the context
- The robot's current grasping software is described in [http://wiki.ros.org/agile_grasp](http://wiki.ros.org/agile_grasp)
Project Idea: Object Handover

- Currently, the arm can let go of an object or close its fingers upon sufficient contact using haptic feedback.
- Can you make it so that it can move towards an object held by a human and grasp it based on visual *and* haptic feedback?
Project Idea: Learning about objects from humans

- The robot is currently able to grasp an object from a table and navigate to an office.
- Can we use the GUI to ask humans questions about objects and store this information in a database that can be used for learning recognition models?
Project Idea: 
Large-Scale 3D object mapping

- Can we combine 3D Plane detection and Clustering to detect and map objects in the environment?
Project Idea: Learning an object manipulation skill

- Example: Pressing a button
Project Idea: Enhance Virtour

www.cs.utexas.edu/~larg/bwi_virtour
Markov Decision Process (MDP)
The reward and state-transition observed at time $t$ after picking action $a$ in state $s$ is independent of anything that happened before time $t$.
Maze World

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

[slide credit: David Silver]
Maze World

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

State Representation: **Factored vs. Tabula Rasa**

[slide credit: David Silver]
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: Policy

- Arrows represent policy $\pi(s)$ for 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 $P_{ss'}$
- Numbers represent immediate reward $R_s^a$ from each state $s$ (same for all $a$)

[slide credit: David Silver]
Sparse vs. Dense Reward
Notation and Problem Formulation

- Overview of notation in TEXPLORE paper
Notation

Set of States:  \( S \)
Set of Actions:  \( A \)
Transition Function:
\[ \mathcal{P} : S \times A \rightarrow \Pi(S) \]
Reward Function:
\[ \mathcal{R} : S \times A \rightarrow \mathbb{R} \]
Action-Value Function

\[ Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a') \]
The value of taking action $a$ in state $s$ is given by the action-value function $Q^*(s, a)$, which is defined as:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q^*(s', a')$$

- $R(s, a)$: The reward received after taking action $a$ in state $s$.
- $P(s' | s, a)$: Probability of going to state $s'$ from $s$ after $a$.
- $\gamma$: Discount factor (between 0 and 1).
- $a'$ is the action with the highest action-value in state $s'$. 
- $Q^*$: The value of taking action $a$ in state $s$. 

The discount factor is a weight that determines how much future rewards are discounted compared to immediate ones.
Action-Value Function

\[ Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a') \]

Common algorithms to learn the action-value function include Q-Learning and SARSA.

The policy consists of always taking the action that maximize the action-value function.
Q-Learning Grid World Example

https://www-s.acm.illinois.edu/sigart/docs/QLearning.pdf
**Algorithm 1 Sequential Model-Based Architecture**

1: **Input:** $S, A$
2: Initialize $M$ to empty model
3: Initialize policy $\pi$ randomly
4: Initialize $s$ to a starting state in the MDP
5: loop
6:   Choose $a \leftarrow \pi(s)$
7:   Take action $a$, observe $r, s'$
8:   $M \Rightarrow \text{UPDATE-MODEL}(\langle s, a, s', r \rangle)$  ▶ Update model $M$ with new experience
9:   $\pi \leftarrow \text{PLAN-POLICY}(M)$  ▶ Exact planning on updated model
10: $s \leftarrow s'$
11: end loop

▷ $S$: state space, $A$: action space
RL in a nutshell

1. **Update Model with Experiences**
2. **Plan Exactly on Updated Model**
3. **Return Action From Policy**

Agent

State $s$, Reward $r$

Environment

Action $a$
Q-Learning

• Guest Slides
Pac Man Example
Linear Function Approximator of $Q^*$

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} \mathcal{P}(s'|s, a) \max_{a'} Q^*(s', a')$$

$\Phi(s,a) = \mathbf{x}$ where $\mathbf{x}$ is an $n$-dimensional feature vector

$$Q^*(\Phi(s,a)) = w_1 * x_1 + w_2 * x_2 + \ldots + w_n * x_n$$
How does Pac-Man “see” the world?
How does Pac-Man “see” the world?
How does Pac-Man “see” the world?
Linear Function Approximator of $Q^*$

\[ Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q^*(s', a') \]

\[ \Phi(s,a) = \mathbf{x} \text{ where } \mathbf{x} \text{ is an } n\text{-dimensional feature vector} \]

\[ Q^*(\Phi(s,a)) = \sum_{i=1}^{n} w_i \cdot x_i \]

The task now is to find the optimal weight vector $w$
Can RL learn directly from images?

• Yes it can:

• http://karpathy.github.io/2016/05/31/rl/
Video on Updating a NN's Weights

Neural Networks Demystified [Part 3: Gradient Descent]

https://www.youtube.com/watch?v=5u0jaA3qAGk
Video of TAMER

http://labcast.media.mit.edu/?p=300
Using RL: Essential Steps

1) Specify the state space or the state-action space
   - Are the states and/or actions discrete or continuous?
2) Specify the reward function
   - If you have control over this, dense reward is better than sparse reward
3) Specify the environment (e.g., a simulator or perhaps the real world)
4) Pick your favorite RL algorithm that can handle the state and action representation
Resources

- BURLAP: Java RL Library: http://burlap.cs.brown.edu/
- Reinforcement Learning: An Introduction