Technische Hochschule Ingolstadt

Principles of Autonomy and Decision Making

(AI_PrincAutonomy_2808)

Week 9: Deep Dive into Reinforcement Learning and Q-Learning

Team:

- **Prof. Dr. rer. nat. Lenz Belzner**
- **Chidvilas Karpenahalli Ramakrishna, M.Eng.**
- **Adithya Mohan, M.Eng.**
- **Zahra Zeinaly, Ph.D.**

Announcements

- Assignment 1: Build your own environment
  - **Deadline**: Monday, 10 June 2024, 23:59
  - **Rules**:
    - Please do not use Dynamic Environments!
    - **If you are not using images**: You should submit one *.py* file with the *"env_<student user id>.py"* naming convention. For example, *"env_chk3541.py"*
    - **If you are using images**: Please submit the assignment as one *.zip* file with the naming convention *"env_<student user id>.zip"*

- Assignment 2: MDPs and Value Iteration
  - **Deadline**: Monday, 10 June 2024, 23:59

- Assignment 3: Build and train a Q-Learning agent
  - Coming soon!
  - **Open date**: Thursday, 06.06.2024, 18:00
  - **Deadline**: Sunday, 30.06.2024, 23:59

- Final Presentation:
  - **Tentative dates**: 01.07.2024 to 10.07.2024
  - Slots will be available soon on Moodle. Each student can only book one slot.
  - Slots fixed and are on First-Come-First-Serve basis
  - *Slots are binding! Cancellations are not allowed!*

Recap

Complexity

Uninformed Search

Depth-First Search (DFS)
- No cost consideration

Breadth-First Search (BFS)
- No cost consideration

Uniform Cost Search (UCS)
- Cost consideration

Informed Search

Greedy Search
- Heuristics

A* algorithm
- Cost + Heuristics

MDP
- Uncertainty consideration
  - $T(s, a, s')$
- Reward consideration
  - $R(s, a, s')$

Value Iteration

Policy Iteration

POMDP
- Uncertainty consideration
  - $T(s, a, s')$
  - $O(o, s', a)$
- Reward consideration
  - $R(s, a, s')$

MC Methods and Particle Filtering

Question:
- Do we have access to $T(s, a, s')$, $R(s, a, s')$ and $O(o, s', a)$?

Why Reinforcement Learning?

- MDP Bellman equation:

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

- POMDP Bellman equation:

$$V(b) = max_{a \in A} \left[\sum_{s \in S} b(s) \left\{ \sum_{s' \in S} P(s'|s,a) \left\langle R(s,a,s') + \gamma \sum_{o \in O} P(o|s',a) V(b') \right\rangle \right\} \right]$$

- In reality:
  - We may not have access to the set of states in the environment i.e. $S = \{s_1, s_2, \ldots, s_n\}$
  - We may not have access to $P(s'|a,s)$
  - We may not have access to $R(s,a,s')$
  - This would result in…

$V^*(s) = max_a \sum_{s'}$ 🙈 [🙈 $+ \; \gamma \; V^*(s')$] i.e. Unknown MDP

  - How would you solve it?

$\Rightarrow$ Reinforcement Learning (RL) 😁

- Solution:
  - We have to explore the environment to find states
  - We have to interact with the environment to learn $P(s'|a,s)$ and $R(s,a,s')$

Model-based and Model-free approach

- Example:
  - Estimate the average test score of PADM students
- Model-based approach:
  - I have data from previous tests showing how different study habits that influence the test scores like hours spent studying, types of study materials used etc
  - I create a model to predict the test scores based on these study habits
  - I use this model to predict the expected test score for each student and calculate the average score
- Model-Free Approach:
  - I don't have data on study habits or a predictive model
  - I collect the scores of a sample of students who take a mock test
  - I use the frequency of these scores to calculate the average score
- In the current lecture, we consider model-free approaches

Unknown MDP

- Known MDP:
  - $T(s, a, s') = 70\%$ in intended direction
  - $T(s, a, s') = 10\%$ in each of the unintended direction

- Unknown MDP:
  - $T(s, a, s') = ?$
  - We will gather experiences and predict the transition probability i.e. $\hat{T}(s, a, s')$
  - $T(s, a, s')$ is built into the sampling process

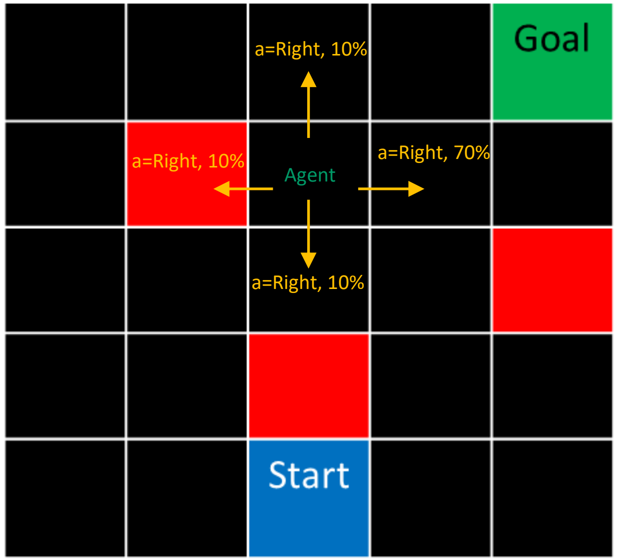

Fig. 1 A grid-world with known MDP

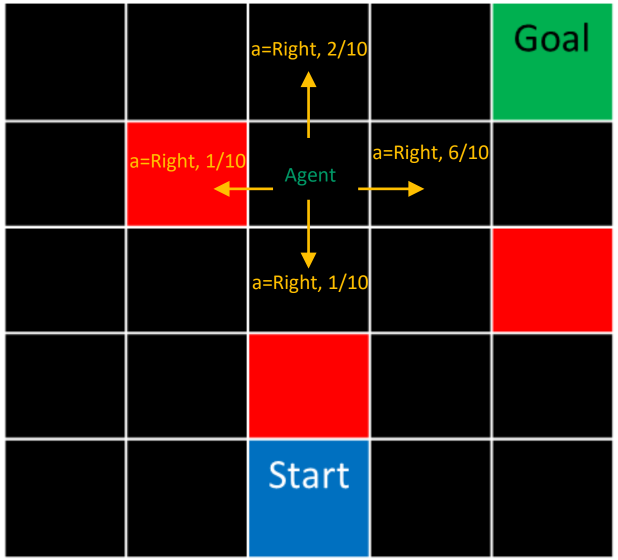

Fig. 2 A grid-world with unknown MDP

Outcome $\hat{T}(s, a, s') = 60\%$ in intended direction

In RL, we learn the MDP

Background: Moving average

- We have to update our knowledge as we experience new things

- Cumulative Moving Average (CMA):
  - CMA considers all the data points up to the current point

$$CMA_t = \frac{1}{t}\sum_{i=1}^{t} x_i$$

- Simple Moving Average (SMA):
  - SMA is calculated by taking the arithmetic mean of a fixed number of recent values in the dataset

$$SMA_t = \frac{1}{n}\sum_{i=t-n+1}^{t} x_i$$

- Exponential moving average (EMA):
  - EMA gives more weight to recent data points, making it more responsive to new information compared to the SMA

$$EMA_t = (1-\alpha).EMA_{t-1} + \alpha.x_t$$

Q-Learning

- Value-Iteration:

$$V_k(s) = max_a \sum_{s'} P(s'|a,s) [\ R(s,a,s') + \gamma\ V_{k-1}(s')\ ]$$

- Q-value iteration:
  - We are more interested in the Q-values
  - Initialize all Q-values to 0 i.e. $Q_k(s,a) = 0$
  - Updating is similar to value-iteration

$$Q_k(s,a) = \sum_{s'} P(s'|a,s) [\ R(s,a,s') + \gamma\ max_{a'} Q_{k-1}(s',a')\ ]$$

  - However, we do not know $R(s,a,s')$ and $P(s'|a,s)$
  - Solution:
    - Q-learning

Q-Learning

- Q-learning:
  - Use moving averages i.e. compute averages as we go
  - Gather a sample by interacting with the environment i.e. (s, a, s', r)
    - $T(s, a, s')$ governs this transition
  - $sample = [R(s, a, s') + \gamma \, max_{a'} Q_{k-1}(s', a')]$ by considering the old Q-values i.e. $Q_{k-1}(s', a')$
  - $Q_k(s, a) = (1 - \alpha) Q_{k-1}(s, a) + \alpha.(sample)$

  - After substitution and rearrangement:

$$\underbrace{Q_k(s,a)}_{\text{New Q-value}} = \underbrace{Q_{k-1}(s,a)}_{\text{Old Q-value}} + \underbrace{\alpha}_{\text{Learning rate}}.[\underbrace{R(s,a,s')}_{\text{Immediate reward for transitioning into } s'} + \underbrace{\gamma}_{\text{Discount factor}} \underbrace{max_{a'} Q_{k-1}(s',a')}_{\text{Maximum expected future reward}} - \underbrace{Q_{k-1}(s,a)}_{\text{Old Q-value}}] \Longrightarrow \text{Q-learning update rule}$$

Features of Q-Learning

- Q-learning is a “model-free” algorithm
  - Does not require the model of the environment
  - It just needs the ability to sample transitions and rewards

- Q-learning is an “off-policy” algorithm
  - Q-learning is called off-policy because it decouples the learning of the optimal policy from the behaviour policy used to explore the environment
  - This allows Q-learning to effectively learn the optimal action-value function, even when the agent's actions are driven by an exploratory or suboptimal policy

- Q-learning convergence to optimal policy
  - Conditions:
    1. The agent must “sufficiently” explore
    2. The learning rate has to decay over time
    3. The environment must remain static/ stationary i.e. $T(s, a, s')$ and $R(s, a, s')$ should remain same

Exploration vs. Exploitation

- Exploration:
  - This involves selecting an action at random, which allows the agent to discover new states and potentially learn more about the environment, even if it means potentially receiving a lower immediate reward
- Exploitation:
  - This involves selecting the action that has the highest known Q-value for a given state, effectively leveraging the knowledge already gathered to maximize immediate rewards
- Q-learning uses an ε-greedy policy for exploration
  - $\varepsilon \in [0, 1]$
  - $\varepsilon = 0$ means the agent always exploits and $\varepsilon = 1$ means the agent only explores
  - So, the agent chooses exploration with a probability of ε and chooses the action with the highest Q-value for the given state with a probability of $(1 - \varepsilon)$

Q-table

- A Q-table is used to store and update Q-values as shown in Fig. 1
- New rows are added as the agent discovers new states
- The Q-values of each state-action pair is updated as per the Q-learning update rule
- Once training is done, when the agent visits the state s_n, it chooses action based on the maximum Q-value
- Can you see the drawbacks of Q-learning?

States \ Actions	Action 1 „Up“	Action 2 „Down“	Action 3 „Left“	Action 4 „Right“
State 1 s_1	$Q(s_1, U)$	$Q(s_1, D)$	$Q(s_1, L)$	$Q(s_1 R)$
State 2 s_2	$Q(s_2, U)$	$Q(s_2, D)$	$Q(s_2, L)$	$Q(s_2, R)$
⋮	⋮	⋮	⋮	⋮
State n s_n	$Q(s_n, U)$	$Q(s_n, D)$	$Q(s_n, L)$	$Q(s_n, R)$

Fig. 1 Q-table with max Q-values highlighted in green

Drawbacks of Q-learning

- Scalability issues:
  - Not suitable for high dimensional state spaces – more rows
  - Not suitable for high dimensional action spaces – more columns
  - Not suitable for continuous action and state spaces

- Non-static environment: Cannot handle complex state spaces i.e. no approximation capabilities
  - Do not use a Q-learning to solve a Dynamic environment
  - If you wish to use Q-learning for a Dynamic environment, you need „Approximate Q-learning“
  - „Approximate Q-learning“ is out of scope of the current lecture