Markov Decision Processes

Part 1

28 September 2020
Reading journals

On pause

Programming Assignment 1: Search

No due date, but make sure you’re working on it
This Friday makes two weeks

Programming Assignment 2: Multi-Agent Pacman

Coming out soon, covering adversarial search / games

Exam 1

On the horizon
Nondeterministic search
Example: Grid World

A maze-like problem

The agent lives in a grid
Walls block the agent’s path
Noisy movement: actions do not always go as planned.

80% of the time, the action *North* takes the agent *North* (if there is no wall there).
10% of the time, *North* takes the agent *West*; 10% *East*.
If there’s a wall in the direction the agent would have been taken, the agent stays put.
Noisy movement: actions do not always go as planned.

80% of the time, the action *North* takes the agent *North* (if there is no wall there).
10% of the time, *North* takes the agent *West*; 10% *East*.

If there's a wall in the direction the agent would have been taken, the agent stays put.
The agent receives rewards each time step.

Small “living” reward each step (can be negative).

Big rewards come at the end (good or bad).

**Goal**: Maximize sum of rewards
Grid World actions

Deterministic Grid World
Grid World actions

Deterministic Grid World

Stochastic Grid World
A **Markov decision process** is defined by:

- A set of **states** \( s \in S \)
- A set of **actions** \( a \in A \)
- A **transition function** \( T(s, a, s') \)
  - Probability that \( a \) from \( s \) leads to \( s' \), i.e., \( P(s' | s, a) \)
  - Also called the **model** or the **dynamics**
- A **reward function** \( R(s, a, s') \)
  - In the textbook, it’s just \( R(s') \)
- A **start state**
- Maybe a **terminal state**
MDPs are nondeterministic search problems.

One way to solve them is with expectimax search.

We’ll have a new tool soon!
[python3 gridworld.py -d 1.0 -r -0.1 -m]
What’s Markov about MDPs?

“Markov” generally means that given the present state, the future and the past are independent

For Markov decision processes, “Markov” means action outcomes depend only on the current state:

\[
P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0) = P(S_{t+1} = s' \mid S_t = s_t, A_t = a_t)
\]

This is just like search, where the successor function could only depend on the current state (not the history).
In deterministic single-agent search problems, we wanted an optimal \textit{plan}, or sequence of actions, from start to a goal.

For MDPs, we want an optimal \textit{policy} $\pi^*: S \rightarrow A$

A policy $\pi$ gives an action for each state.

An optimal policy is one that maximizes expected utility if followed.

An explicit policy defines a reflex agent.

\textit{Optimal policy when} $R(s, a, s') = -0.03$ \textit{for all non-terminals} $s$
Optimal policies

\[ R(s) = -0.01 \]
Optimal policies

\[ R(s) = -0.01 \]

\[ R(s) = -0.03 \]
Optimal policies

\[ R(s) = -0.4 \]

\[ R(s) = -0.03 \]

\[ R(s) = -0.01 \]
Optimal policies

Optimal policies for different reward values:

- $R(s) = -2.0$
- $R(s) = -0.4$
- $R(s) = -0.03$
- $R(s) = -0.01$
Utilities of sequences
What preferences should an agent have over reward sequences?
What preferences should an agent have over reward sequences?
More or less?

\[ [1, 2, 2] \text{ or } [2, 3, 4] \]

Now or later?
What preferences should an agent have over reward sequences?

More or less?

\[ [1, 2, 2] \text{ or } [2, 3, 4] \]

Now or later?

\[ [0, 0, 1] \text{ or } [1, 0, 0] \]
Discounting

It’s reasonable to maximize the sum of rewards.

It’s also reasonable to prefer rewards now to rewards later.

One solution: values of rewards decay exponentially.

Discount factor $\gamma$ between 0 and 1
Discounting

It’s reasonable to maximize the sum of rewards.

It’s also reasonable to prefer rewards *now* to rewards *later*.

One solution: values of rewards decay *exponentially*.

Discount factor $\gamma$ between 0 and 1
Discounting

It’s reasonable to maximize the sum of rewards.

It’s also reasonable to prefer rewards now to rewards later.

One solution: values of rewards decay exponentially.

Discount factor $\gamma$ between 0 and 1
Discounting

It’s reasonable to maximize the sum of rewards.

It’s also reasonable to prefer rewards *now* to rewards *later*.

One solution: values of rewards decay *exponentially*.

Discount factor $\gamma$ between 0 and 1
How to discount?

Each time we descend a level, we multiply in the discount once.

Why discount?

Sooner rewards probably do have higher utility than later rewards.
Also helps our algorithms converge!

Example: discount of 0.5

\[ U([1, 2, 3]) = 1 \cdot 1 + 0.5 \cdot 2 + 0.25 \cdot 3 \]
How to discount?

Each time we descend a level, we multiply in the discount once.

Why discount?

Sooner rewards probably do have higher utility than later rewards.
Also helps our algorithms converge!

**Example:** discount of 0.5

\[ U([1, 2, 3]) = 1 \cdot 1 + 0.5 \cdot 2 + 0.25 \cdot 3 \]

\[ U([1, 2, 3]) < U([3, 2, 1]) \]
THEOREM: If we assume stationary preferences,

\[ [a_1, a_2, \ldots] > [b_1, b_2, \ldots] \]
\[ \iff \]
\[ [r, a_1, a_2, \ldots] > [r, b_1, b_2, \ldots] \]

then there are only two ways to define utilities:

*Additive utility:*

\[ U([r_0, r_1, r_2, \ldots]) = r_0 + r_1 + r_2 + \cdots \]

*Discounted utility:*

\[ U([r_0, r_1, r_2, \ldots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \]
Quiz: Discounting

Given:

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

*Actions:* East, West, and Exit (only available in exit states a, e)

*Transitions:* deterministic

1: For $\gamma = 1$, what is the optimal policy?

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

2: For $\gamma = 0.1$, what is the optimal policy?

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

3: For which $\gamma$ are West and East equally good when in state d?
Quiz: Discounting

Given:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

*Actions*: *East*, *West*, and *Exit* (only available in exit states a, e)

*Transitions*: deterministic

1: For $\gamma = 1$, what is the optimal policy?

```
10 ← ← ← 1
```

2: For $\gamma = 0.1$, what is the optimal policy?

```
10   1
```

3: For which $\gamma$ are *West* and *East* equally good when in state $d$?
Quiz: Discounting

Given: 

<table>
<thead>
<tr>
<th>10</th>
<th></th>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
</tbody>
</table>

*Actions: East, West, and Exit* (only available in exit states a, e)

*Transitions:* deterministic

1: For $\gamma = 1$, what is the optimal policy?

```
10 ← ← ← 1
```

2: For $\gamma = 0.1$, what is the optimal policy?

```
10 ← ← → 1
```

3: For which $\gamma$ are West and East equally good when in state d?
Quiz: Discounting

Given:

\[
\begin{array}{cccc}
10 & & & 1 \\
\hline
a & b & c & d & e
\end{array}
\]

*Actions:* East, West, and Exit (only available in exit states a, e)

*Transitions:* deterministic

1: For $\gamma = 1$, what is the optimal policy?

\[
10 \leftarrow \leftarrow \leftarrow 1
\]

2: For $\gamma = 0.1$, what is the optimal policy?

\[
10 \leftarrow \leftarrow \rightarrow 1
\]

3: For which $\gamma$ are West and East equally good when in state d?

\[
1\gamma = 10\gamma^3
\]
Infinite utilities?!

**Problem:** What if the game lasts forever? Do we get infinite rewards?

**Solutions:**

\[
U([r_0, \ldots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{\text{max}}}{1 - \gamma}
\]
Infinite utilities?!

*Problem:* What if the game lasts forever? Do we get infinite rewards?

*Solutions:*

*Finite horizon:* (similar to depth-limited search)

\[
U([r_0, \ldots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{\text{max}}}{1 - \gamma}
\]
Infinite utilities?!

**Problem:** What if the game lasts forever? Do we get infinite rewards?

**Solutions:**

*Finite horizon:* (similar to depth-limited search)
- Terminate episodes after a fixed $T$ steps (e.g., life)
- Gives non-stationary policies ($\pi$ depends on time left)

*Discounting:* Use $0 < \gamma < 1$:

$$U([r_0, \ldots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{\text{max}}}{1 - \gamma}$$
Infinite utilities?!

**Problem**: What if the game lasts forever? Do we get infinite rewards?

**Solutions**:

*Finite horizon*: (similar to depth-limited search)
- Terminate episodes after a fixed $T$ steps (e.g., life)
- Gives non-stationary policies ($\pi$ depends on time left)

*Discounting*: Use $0 < \gamma < 1$:

$$ U([r_0, \ldots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{\text{max}}}{1 - \gamma} $$

Smaller $\gamma$ means smaller “horizon” – shorter term focus.

*Absorbing state*: Guarantee that for every policy, a terminal state will eventually be reached.
Example: Racing
A robot car wants to travel far, quickly.
A robot car wants to travel far, quickly.

Three states: Cool, Warm, Overheated
A robot car wants to travel far, quickly.

Three states: **Cool**, **Warm**, **Overheated**

Two actions: **Slow**, **Fast**
A robot car wants to travel far, quickly.

Three states: **Cool**, **Warm**, **Overheated**

Two actions: **Slow**, **Fast**

Going faster gets double reward
Racing search tree
Racing search tree
Racing search tree
Racing search tree
Racing search tree
Each MDP state projects an expectimax-like search tree

(s, a) is a q-state

(s, a, s′) called a transition

\[ T(s, a, s′) = P(s′ | s, a) \]

\[ R(s, a, s′) \]
Recap: Defining MDPs

Markov decision processes:

Set of **states** $S$

**Start state** $s_0$

Set of **actions** $A$

**Transitions** $P(s' | s, a)$ or $T(s, a, s')$

**Rewards** $R(s, a, s')$ and discount $\gamma$

MDP quantities so far:

**Policy**: Choice of action for each state

**Utility**: Sum of (discounted) rewards
Solving MDPs
Optimal quantities

The value (utility) of a state $s$:

\[ V^*(s) = \text{expected utility starting in } s \text{ and acting optimally} \]

The value (utility) of a q-state $(s, a)$:

\[ Q^*(s, a) = \text{expected utility starting out having taken action } a \text{ from state } s \text{ and (thereafter) acting optimally} \]

The optimal policy:

\[ \pi^*(s) = \text{optimal action from state } s \]
Grid World V values

Noise = 0.2
Discount = 0.9
Living reward = 0
Grid World Q values

Noise = 0.2
Discount = 0.9
Living reward = 0
Values of states

Recursive definition of value:

$$V^*(s) = \max_a Q^*(s, a)$$

How likely is this outcome?

Expectimax:

$$Q^*(s, a) = \sum_{s'} T(s, a, s') V^*(s')$$

Adding up to average

Future reward
Values of states

Recursive definition of value:

\[ V^*(s) = \max_a Q^*(s, a) \]

How likely is this outcome?  
Future reward

Adding up to average

Immediate reward

Discount for \( s' \) being one time step in the future

\[ Q^*(s, a) = \sum_{s'} T(s, a, s') R(s, a, s') + \gamma V^*(s') \]
Values of states

Recursive definition of value:

\[ V^*(s) = \max_a Q^*(s, a) \]

\[ Q^*(s, a) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')] \]

Bellman Equation:

\[ V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma V^*(s')] \]
Racing search tree
Racing search tree
Racing search tree
Racing search tree

We’re doing way too much work with expectimax!

**Problem**: States are repeated

**Idea**: Only compute needed quantities once
Racing search tree

We’re doing way too much work with expectimax!

**Problem:** States are repeated

*Idea:* Only compute needed quantities once

**Problem:** Tree goes on forever

*Idea:* Do a depth-limited computation, but with increasing depths until change is small

*Note:* Deep parts of the tree eventually don’t matter if $\gamma < 1$
To be continued!
Acknowledgments

The lecture incorporates material from:

Pieter Abbeel, Dan Klein, et al., University of California, Berkeley;
ai.berkeley.edu
Ketrina Yim (illustrations)