CSCI 1470/2470
Spring 2024

Ritambhara Singh

April 17, 2024
Wednesday

Deep Learning

Reinforcement Learning: Value Iteration
Recap: RL framework

Agent

Action

Environment

State, Reward
Recap: Markov Decision Process (MDP)

- States – set of possible situations in a world, denoted $S$
- Actions – set of different actions an agent can take, denoted $A$
- Transition function – returns the probability of transitioning to state $s'$ after taking action $a$ in state $s$, denoted $T(s, a, s')$
- Reward function – returns the reward received by the agent for transitioning to state $s'$ after taking action $a$ in state $s$, denoted $R(s, a, s')$
Recap: Policy Function

• What action should the agent take in a given state?
• Concretely:
  • \( \pi : S \to A \)
  • Input: state \( s \in S \)
  • Output: action to be chosen in that state
  • \( \pi(s) = a \) means in state \( s \), take action \( a \)
Recap: Goal of RL

- Learn optimal policy $\pi^*$ that maximizes the expected future cumulative reward
  - “Expected” because transitions can be non-deterministic
- Solving MDPs $\leftrightarrow$ find this optimal policy!
Organizing RL problems/algorithms

Know $T$ and $R$
- Value iteration

Don’t know $T$ and $R$
- Q-Learning
- Deep Q-Networks
- REINFORCE
- Actor-Critic

For a more complete taxonomy of RL algorithms, see
https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below
Value Iteration
Value Function

What would motivate us to move from a state $s$ to $s'$?

We assign a "value" to each state.
Value Function

- Function that returns the “value” of each state
Value Function

- Function that returns the “value” of each state
- Value of state $s$ under policy $\pi$ is the expected return when starting in $s$ and following $\pi$
Value Function

- Function that returns the “value” of each state
- Value of state $s$ under policy $\pi$ is the expected return when starting in $s$ and following $\pi$
- “Value of my current state is the total discounted future reward I expect from following a policy $\pi$ from now on”
Value Function

- Function that returns the “value” of each state
- Value of state $s$ under policy $\pi$ is the expected return when starting in $s$ and following $\pi$
- “Value of my current state is the total discounted future reward I expect from following a policy $\pi$ from now on”
- $V_\pi : S \rightarrow \mathbb{R}$
Value Function

- Function that returns the “value” of each state
- Value of state $s$ under policy $\pi$ is the expected return when starting in $s$ and following $\pi$
- “Value of my current state is the total discounted future reward I expect from following a policy $\pi$ from now on”
- $V_\pi : S \rightarrow \mathbb{R}$
- $V_\pi (s) = \mathbb{E} [G_t \mid S_t = s]$ for all $s \in S$, where $G_t$ is the return
Value Function

- Function that returns the “value” of each state
- Value of state $s$ under policy $\pi$ is the expected return when starting in $s$ and following $\pi$
- “Value of my current state is the total discounted future reward I expect from following a policy $\pi$ from now on”
- $V_\pi: S \rightarrow \mathbb{R}$
- $V_\pi(s) = E[G_t \mid S_t = s]$ for all $s \in S$, where $G_t$ is the return
- $= E[R(s, a, s') + \gamma V_\pi(s') \mid S_t = s]$
Value Function

- Function that returns the “value” of each state
  - Value of state $s$ under policy $\pi$ is the expected return when starting in $s$ and following $\pi$
  - “Value of my current state is the total discounted future reward I expect from following a policy $\pi$ from now on”
  - $V_\pi : S \rightarrow \mathbb{R}$
  - $V_\pi(s) = E[G_t | S_t = s]$ for all $s \in S$, where $G_t$ is the return
    - $= E[R(s, a, s') + \gamma V_\pi(s') | S_t = s]$
  - $V_\pi(s) = \sum_{s' \in S} P(s'|s, a) [R(s, a, s') + \gamma V_\pi(s')]$

 expectation across transition probabilities- deals with the potential stochasticity of transitioning to $s'$

 NOTE: recursively defined!
 Literally “reward agent receives now + value of the next state”

Any questions?
Example (made-up) Value Table

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>State #1</td>
<td>0</td>
</tr>
<tr>
<td>State #2</td>
<td>1</td>
</tr>
<tr>
<td>State #3</td>
<td>-1</td>
</tr>
<tr>
<td>State #4</td>
<td>1.9</td>
</tr>
<tr>
<td>State #5</td>
<td>10</td>
</tr>
<tr>
<td>State #6</td>
<td>-10</td>
</tr>
</tbody>
</table>

Which is the favorable state?

“If we transition from state #5 using the (our made-up) policy to other states s’ the expected total discounted future reward is 10”
Q-function

What if we have multiple actions to take from s to s’?

We assign "value" to each action at a given state
Q-function

- \( q_\pi : S \times A \rightarrow \mathbb{R} \)
Q-function

- $q_\pi: S \times A \to \mathbb{R}$
- $q_\pi(s, a) = E[G_t \mid S_t = s, A_t = a]$ for all $s \in S, a \in A$
Q-function

- $q_\pi: S \times A \to \mathbb{R}$
- $q_\pi(s, a) = E[G_t | S_t = s, A_t = a]$ for all $s \in S, a \in A$
- AKA “action-value function”
Q-function

- \( q_\pi : S \times A \to \mathbb{R} \)
- \( q_\pi(s, a) = E[G_t \mid S_t = s, A_t = a] \) for all \( s \in S, a \in A \)
- AKA “action-value function”
- Outputs expected return from taking action \( a \) in state \( s \) and following policy \( \pi \) thereafter
### Q-value Table (made up)

<table>
<thead>
<tr>
<th>State #</th>
<th>Action #1</th>
<th>Action #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>State #1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>State #2</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>State #3</td>
<td>-1</td>
<td>-10</td>
</tr>
<tr>
<td>State #4</td>
<td>0</td>
<td>1.9</td>
</tr>
<tr>
<td>State #5</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>State #6</td>
<td>-10</td>
<td>-10</td>
</tr>
</tbody>
</table>
How to determine policy from Q-function?

Q-value = 9000

Q-value = 10

Any ideas?
How to determine policy from Q-function?

Choose the action that maximizes your Q-value!

\[ \pi(s) = \arg\max_a Q(s, a) \]
What actions to pick for each state for the optimal policy?

<table>
<thead>
<tr>
<th>State #</th>
<th>Action #1</th>
<th>Action #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>State #1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>State #2</td>
<td>0.1</td>
<td>1</td>
</tr>
<tr>
<td>State #3</td>
<td>-1</td>
<td>-10</td>
</tr>
<tr>
<td>State #4</td>
<td>0</td>
<td>1.9</td>
</tr>
<tr>
<td>State #5</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>State #6</td>
<td>-10</td>
<td>-10</td>
</tr>
</tbody>
</table>
Q-function can be expressed in terms of the V-function
Q-function can be expressed in terms of the V-function

- $Q^\pi(s, a) = E[R(s, a, s') + \gamma V^\pi(s')]$
Q-function can be expressed in terms of the V-function

- \( Q^\pi(s, a) = E[R(s, a, s') + \gamma V^\pi(s')] \)
- \( Q^\pi(s, a) = \sum_{s' \in S} P(s'|s, a)[R(s, a, s') + \gamma V^\pi(s')] \)
Q-value and V-value Tables (made up)

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>State #1</td>
<td>0</td>
</tr>
<tr>
<td>State #2</td>
<td>1</td>
</tr>
<tr>
<td>State #3</td>
<td>-1</td>
</tr>
<tr>
<td>State #4</td>
<td>1.9</td>
</tr>
<tr>
<td>State #5</td>
<td>0</td>
</tr>
<tr>
<td>State #6</td>
<td>-10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>State #1</td>
<td>0</td>
</tr>
<tr>
<td>State #2</td>
<td>1</td>
</tr>
<tr>
<td>State #3</td>
<td>-1</td>
</tr>
<tr>
<td>State #4</td>
<td>1.9</td>
</tr>
<tr>
<td>State #5</td>
<td>10</td>
</tr>
<tr>
<td>State #6</td>
<td>-10</td>
</tr>
</tbody>
</table>

Any questions?
Optimal policy and value functions
Optimal policy and value functions

- Goal of RL: find optimal policy, $\pi^*$
Optimal policy and value functions

• Goal of RL: find optimal policy, $\pi^*$
• Approach: learn optimal value functions, $V^*$ and $Q^*$, then define optimal policy from value functions
How do we actually learn $V^*$ and $Q^*$?
Value iteration pseudocode
Value iteration pseudocode

1. For all $s$, set $V(s) := 0$. 
Value iteration pseudocode

1. For all $s$, set $V(s) := 0$.
2. Repeat until convergence:
Value iteration pseudocode

1. For all $s$, set $V(s) := 0$.
2. Repeat until convergence:
   1. For all $s$: 
Value iteration pseudocode

1. For all \( s \), set \( V(s) := 0 \).
2. Repeat until convergence:
   1. For all \( s \):
      1. For all \( a \), set \( Q(s, a) := \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')] \)
Value iteration pseudocode

1. For all $s$, set $V(s) := 0$.

2. Repeat until convergence:
   1. For all $s$:
      1. For all $a$, set $Q(s, a) := \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma V(s')]$
      2. $V(s) := \max_a Q(s, a)$
Value iteration pseudocode

1. For all $s$, set $V(s) := 0$.

2. Repeat until convergence:
   1. For all $s$:
      1. For all $a$, set $Q(s,a) := \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V(s')]$
      2. $V(s) := \max_a Q(s,a)$

3. Return $Q$
Concrete Example:
Frozen Lake Problem
Frozen Lake Problem

- Agent starts in top left corner
- Goal: Reach the bottom right without falling into any of the holes (skulls)
- Game terminates when agent falls into hole or reaches goal
Optimal policy is easy, right?

- Multiple optimal policies, actually
- Solve using shortest path algorithm
Not quite - frozen lakes are slippery!

<table>
<thead>
<tr>
<th></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>

- Agent may not actually move in the direction of the action!
- Yellow arrow indicates the action
- Red arrows indicate where the agent may end up, each with probability 1/3
Can’t “fall off” frozen lake

• Transitioning beyond an edge will keep you in same state
Frozen Lake Problem as an MDP

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="gear.png" alt="Head" /></td>
<td><img src="skull.png" alt="Skull" /></td>
<td><img src="skull.png" alt="Skull" /></td>
<td><img src="skull.png" alt="Skull" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="skull.png" alt="Skull" /></td>
<td><img src="skull.png" alt="Skull" /></td>
<td><img src="skull.png" alt="Skull" /></td>
<td><img src="skull.png" alt="Skull" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="skull.png" alt="Skull" /></td>
<td><img src="skull.png" alt="Skull" /></td>
<td><img src="skull.png" alt="Skull" /></td>
<td><img src="skull.png" alt="Skull" /></td>
</tr>
<tr>
<td>4</td>
<td><img src="skull.png" alt="Skull" /></td>
<td><img src="checkered.png" alt="Checkered" /></td>
<td><img src="checkered.png" alt="Checkered" /></td>
<td><img src="checkered.png" alt="Checkered" /></td>
</tr>
</tbody>
</table>

- **States:** each square - (1, 1), (1, 2), ..., (4, 4)
- **Actions:** left, right, up, down
- **Reward:** +1 when you reach the goal, 0 elsewhere
- **Transition function:** stochastic (because ice is slippery!)
  Equal probability of moving in any direction except chosen action, e.g. if agent is in (1, 3) and action is down:
  - 1/3 chance of moving to (1, 2)
  - 1/3 chance of moving to (2, 3)
  - 1/3 chance of moving to (1, 4)
Frozen Lake - initialization

<table>
<thead>
<tr>
<th>Environment</th>
<th>VALUE TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Frozen Lake – iteration 1: update square (1, 3)

Environment

1 2 3 4

1

2

3

4

Old Value Table

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

New Value Table

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$V((1, 3))$ is still 0, because the adjacent values of (1, 3) are all 0 and no rewards are gained for any possible action taken in (1,3).
Frozen Lake – iteration 1: update square (4, 3)

Environment

Old Value Table

New Value Table

How did we get 0.33?
Update (4, 3) explanation
Update (4, 3) explanation

Update equation:

\[ V(s) = \max_a Q(s, a), \text{ where } Q(s, a) = \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')] \]
Update (4, 3) explanation

Update equation:

\[ V(s) = \max_a Q(s, a), \text{ where } Q(s, a) = \sum_{s' \in S} T(s, a, s') [R(s, a, s') + \gamma V(s')] \]

Finding Q values for all actions:
Update (4, 3) explanation

Update equation:

\[ V(s) = \max_a Q(s, a), \text{ where } Q(s, a) = \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')] \]

Finding Q values for all actions:

- \[ Q((4, 3), \text{right}) = \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((3, 3))) + \frac{1}{3}(0 + \gamma V((4, 3))) = 0.33 \]
Update (4, 3) explanation

Update equation:

$$V(s) = \max_a Q(s, a), \text{ where } Q(s, a) = \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')]$$

Finding Q values for all actions:

- $Q((4, 3), \text{right}) = \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((3, 3))) + \frac{1}{3}(0 + \gamma V((4, 3))) = 0.33$
- $Q((4, 3), \text{up}) = \frac{1}{3}(0 + \gamma V((3, 3))) + \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((4, 2))) = 0.33$
Update (4, 3) explanation

Update equation:

\[ V(s) = \max_a Q(s, a), \text{ where } Q(s, a) = \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')] \]

Finding Q values for all actions:

- \( Q((4, 3), right) = \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((3, 3))) + \frac{1}{3}(0 + \gamma V((4, 3))) = 0.33 \)
- \( Q((4, 3), up) = \frac{1}{3}(0 + \gamma V((3, 3))) + \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((4, 2))) = 0.33 \)
- \( Q((4, 3), down) = \frac{1}{3}(0 + \gamma V((4, 3))) + \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((4, 2))) = 0.33 \)
Update (4, 3) explanation

Update equation:

\[ V(s) = \max_a Q(s, a), \text{ where } Q(s, a) = \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')] \]

Finding Q values for all actions:

- \( Q((4, 3), \text{right}) = \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((3, 3))) + \frac{1}{3}(0 + \gamma V((4, 3))) = 0.33 \)
- \( Q((4, 3), \text{up}) = \frac{1}{3}(0 + \gamma V((3, 3))) + \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((4, 2))) = 0.33 \)
- \( Q((4, 3), \text{down}) = \frac{1}{3}(0 + \gamma V((4, 3))) + \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((4, 2))) = 0.33 \)
- \( Q((4, 3), \text{left}) = \frac{1}{3}(0 + \gamma V((4, 2))) + \frac{1}{3}(0 + \gamma V((4, 3))) + \frac{1}{3}(0 + \gamma V((3, 3))) = 0 \)
Update (4, 3) explanation

Update equation:

\[ V(s) = \max_a Q(s, a), \text{ where } Q(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V(s')] \]

Finding Q values for all actions:

- \( Q((4, 3), \text{right}) = \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((3, 3))) + \frac{1}{3}(0 + \gamma V((4, 3))) = 0.33 \)
- \( Q((4, 3), \text{up}) = \frac{1}{3}(0 + \gamma V((3, 3))) + \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((4, 2))) = 0.33 \)
- \( Q((4, 3), \text{down}) = \frac{1}{3}(0 + \gamma V((4, 3))) + \frac{1}{3}(1 + \gamma V((4, 4))) + \frac{1}{3}(0 + \gamma V((4, 2))) = 0.33 \)
- \( Q((4, 3), \text{left}) = \frac{1}{3}(0 + \gamma V((4, 2))) + \frac{1}{3}(0 + \gamma V((4, 3))) + \frac{1}{3}(0 + \gamma V((3, 3))) = 0 \)

Then,

\[ V((4,3)) = \max_a Q((4,3), a) = 0.33 \]
During this iteration, the value from (4, 3) “backs up” to its adjacent states, (3, 3) and (4, 2).

Value of (4, 3) increases because its adjacent states (3, 3) and (4, 2) have positive values.
Frozen Lake – final value table & optimal policy

Environment

Example Final Value Table

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.068</td>
<td>0.061</td>
<td>0.074</td>
<td>0.055</td>
</tr>
<tr>
<td>2</td>
<td>0.092</td>
<td>0.0</td>
<td>0.112</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>0.145</td>
<td>0.247</td>
<td>0.3</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
<td>0.38</td>
<td>0.639</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Final Policy

Now what?
Frozen Lake – demo

https://colab.research.google.com/drive/1RFFdzJ8VshmpvnbcbLNggwxfwMEBw222?usp=sharing
Frozen Lake – optimal policy in action

Final Policy

1
2
3
4

1
2
3
4

https://towardsdatascience.com/value-iteration-to-solve-openai-gyms-frozenlake-6c5e7bf0a64d
Play more with value iteration!

- https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html
Recap

Value functions
- V function
- Q function (“action-value” function)
- Connection between V and Q

Value Iteration
- Pseudocode
- Frozen Lake Problem
- Demo: Learning Optimal Policy

Environment

This Photo by Unknown Author is licensed under CC BY-SA-NC