Lecture 11: Reinforcement Learning

Shen Shen
April 26, 2024
Outline

- Recap: Markov Decision Processes
- Reinforcement Learning Setup
  - What's changed from MDP?
- Model-based methods
- Model-free methods
  - (tabular) Q-learning
    - $\epsilon$-greedy action selection
    - exploration vs. exploitation
  - (neural network) Q-learning
- RL setup again
  - What's changed from supervised learning?
MDP Definition and Goal

- $S$: state space, contains all possible states $s$.
- $A$: action space, contains all possible actions $a$.
- $T(s, a, s')$: the probability of transition from state $s$ to $s'$ when action $a$ is taken.
- $R(s, a)$: a function that takes in the (state, action) and returns a reward.
- $\gamma \in [0, 1]$: discount factor, a scalar.

- $\pi(s)$: policy, takes in a state and returns an action.

Ultimate goal of an MDP: Find the "best" policy $\pi$. 
a trajectory (aka an experience or rollout) \( \tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots) \)

how "good" is a trajectory?

\[ R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \gamma^3 R(s_3, a_3) + \gamma^4 R(s_4, a_4) + \gamma^5 R(s_5, a_5) + \gamma^6 R(s_6, a_6) + \gamma^7 R(s_7, a_7) \ldots \]
Running example: Mario in a grid-world

- 9 possible states
- 4 possible actions: \{Up ↑, Down ↓, Left ←, Right →\}
- almost all transitions are deterministic:
  - Normally, actions take Mario to the “intended” state.
    - E.g., in state (7), action “↑” gets to state (4)
  - If an action would've taken us out of this world, stay put
    - E.g., in state (9), action “→” gets back to state (9)
  - except, in state (6), action “↑” leads to two possibilities:
    - 20% chance ends in (2)
    - 80% chance ends in (3)
example cont'd

- (state, action) pair can get Mario rewards:
  - In state (3), any action gets reward $+1$
  - In state (6), any action gets reward $-10$
  - Any other (state, action) pairs get reward $0$

actions: {Up ↑, Down ↓, Left ←, Right →}

goal is to find a gameplay policy strategy for Mario, to get maximum expected sum of discounted rewards, with a discount factor $\gamma = 0.9$
Recall:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>20%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>20%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>80%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( \pi(s) = \text{“} \uparrow \text{”, } \forall s \)

\( R(3, \uparrow) = 1 \)

\( R(6, \uparrow) = -10 \)

\( \gamma = 0.9 \)

Now, let's think about \( V^3_\pi(6) \)

\[
V^3_\pi(6) = \begin{bmatrix} R(6, \uparrow) \end{bmatrix} + 20\% \left[ \gamma \begin{bmatrix} R(2, \uparrow) \end{bmatrix} + \gamma^2 \begin{bmatrix} R(2, \uparrow) \end{bmatrix} \right] + 80\% \left[ \gamma \begin{bmatrix} R(3, \uparrow) \end{bmatrix} + \gamma^2 \begin{bmatrix} R(3, \uparrow) \end{bmatrix} \right]
\]

\[
= \begin{bmatrix} R(6, \uparrow) \end{bmatrix} + 20\% \gamma \begin{bmatrix} R(2, \uparrow) \end{bmatrix} + 80\% \gamma \begin{bmatrix} R(3, \uparrow) \end{bmatrix}
\]

\[
= \begin{bmatrix} R(6, \uparrow) \end{bmatrix} + 20\% \gamma \begin{bmatrix} V^2_\pi(2) \end{bmatrix} + 80\% \gamma \begin{bmatrix} V^2_\pi(3) \end{bmatrix}
\]
finite-horizon policy evaluation

For a given policy $\pi(s)$, the finite-horizon horizon-$h$ (state) value functions are:

$$V^h_\pi(s) := \mathbb{E} \left[ \sum_{t=0}^{h-1} \gamma^t R(s_t, \pi(s_t)) \mid s_0 = s, \pi \right], \forall s$$

Bellman recursion

$$V^h_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V^{h-1}_\pi(s'), \forall s$$

infinite-horizon policy evaluation

For any given policy $\pi(s)$, the infinite-horizon (state) value functions are

$$V_\pi(s) := \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \mid s_0 = s, \pi \right], \forall s$$

$\gamma$ is now necessarily $<1$ for convergence too in general

Bellman equation

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_\pi(s'), \forall s$$

- $|S|$ many linear equations
Recall: \( \gamma = 0.9 \)

States and one special transition:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>20%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>80%</td>
</tr>
</tbody>
</table>

Example: recursively finding \( Q^h(s, a) \)

\( Q^h(s, a) \) is the expected sum of discounted rewards for

- starting in state \( s \),
- take action \( a \), for one step
- act **optimally** there afterwards for the remaining \((h - 1)\) steps
$Q^h(s, a)$ is the expected sum of discounted rewards for

- starting in state $s$,
- take action $a$, for one step
- act optimally there afterwards for the remaining $(h - 1)$ steps

Recall: $\gamma = 0.9$

States and one special transition:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>20%</td>
<td>5</td>
<td>80%</td>
</tr>
</tbody>
</table>

Let's consider $Q^2(6, \uparrow) = R(6, \uparrow) + \gamma \cdot 0.2 \cdot \max_{a'} Q^1(2, a') + 0.8 \cdot \max_{a'} Q^1(3, a')$)

- receive $R(6, \uparrow)$
  - act optimally for one more timestep, at the next state $s'$
    - 20% chance, $s' = 2$, act optimally, receive $\max_{a'} Q^1(2, a')$
    - 80% chance, $s' = 3$, act optimally, receive $\max_{a'} Q^1(3, a')$

$$= -10 + 0.9 \cdot [0.2 \cdot 0 + 0.8 \cdot 1] = -9.28$$
$Q^h(s, a)$ is the expected sum of discounted rewards for starting in state $s$, taking action $a$, for one step and act optimally there afterwards for the remaining $(h - 1)$ steps.

Recall: $\gamma = 0.9$

States and one special transition:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>20%</td>
<td></td>
<td>80%</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

$Q^1(s, a) = R(s, a)$

$Q^2(s, a)$

$Q^2(6, \uparrow) = R(6, \uparrow) + \gamma[.2 \max_{a'} Q^1(2, a') + .8 \max_{a'} Q^1(3, a')]$

in general $Q^h(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q^{h-1}(s', a'), \forall s, a$
$Q^h(s, a)$ is the expected sum of discounted rewards for

- starting in state $s$,
- take action $a$, for one step
- act **optimally** there afterwards for the remaining $(h-1)$ steps

Recall: $\gamma = 0.9$

<table>
<thead>
<tr>
<th>States and one special transition:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

what's the optimal action in state 3, with horizon 2, given by $\pi^*_2(3) =$?

in general $\pi^*_h(s) = \arg\max_a Q^h(s, a), \forall s, h$

either up or right
Given the finite horizon recursion

\[ Q^h(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q^{h-1}(s', a') \]

We should easily be convinced of the infinite horizon equation

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

**Infinite-horizon Value Iteration**

1. **for** \( s \in S, a \in A \):
2. \( Q_{\text{old}}(s, a) = 0 \)
3. **while** True:

   4. **for** \( s \in S, a \in A \):
   5. \( Q_{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a') \)
   6. **if** \( \max_{s, a} |Q_{\text{old}}(s, a) - Q_{\text{new}}(s, a)| < \epsilon \):
   7. \( \text{return } Q_{\text{new}} \)
   8. \( Q_{\text{old}} \leftarrow Q_{\text{new}} \)
Outline

- Recap: Markov Decision Processes
- Reinforcement Learning Setup
  - What's changed from MDP?
- Model-based methods
- Model-free methods
  - (tabular) Q-learning
    - $\epsilon$-greedy action selection
    - exploration vs. exploitation
  - (neural network) Q-learning
- RL setup again
  - What's changed from supervised learning?
Running example: Mario in a grid-world (the Reinforcement-Learning Setup)

- 9 possible states
- 4 possible actions: \{Up $\uparrow$, Down $\downarrow$, Left $\leftarrow$, Right $\rightarrow$\}
- all transitions probabilities are unknown.

- (state, action) pair gets Mario unknown rewards.
- goal is to find a gameplay policy strategy for Mario, to get maximum expected sum of discounted rewards, with a discount factor $\gamma = 0.9$
RL  Definition and Goal

- \( \mathcal{S} \): state space, contains all possible states \( s \).
- \( \mathcal{A} \): action space, contains all possible actions \( a \).
- \( T(s, a, s') \): the probability of transition from state \( s \) to \( s' \) when action \( a \) is taken.
- \( R(s, a) \): a function that takes in the (state, action) and returns a reward.
- \( \gamma \in [0, 1] \): discount factor, a scalar.

- \( \pi(s) \): policy, takes in a state and returns an action.

Ultimate goal of an RL: Find the "best" policy \( \pi \).
Outline

- Recap: Markov Decision Processes
- Reinforcement Learning Setup
  - What's changed from MDP?
- Model-based methods
- Model-free methods
  - (tabular) Q-learning
    - $\epsilon$-greedy action selection
    - exploration vs. exploitation
  - (neural network) Q-learning
- RL setup again
  - What's changed from supervised learning?
(MDP)-Model-Based Methods (for solving RL)

Keep playing the game to approximate the unknown rewards and transitions.

Rewards are particularly easy:

  e.g. by observing what reward \( r \) received from being in state 6 and take \( \uparrow \) action, we know \( R(6, \uparrow) \)

Transitions are a bit more involved but still simple:

  e.g. play the game 1000 times, count the # of times (we started in state 6, take \( \uparrow \) action, end in state 2), then, roughly, \( T(6, \uparrow, 2) = \text{(that count/1000)} \)

Now, with \( R, T \) estimated, we're back in MDP setting.

In Reinforcement Learning:

  - *Model* typically means MDP tuple \( \langle S, A, T, R, \gamma \rangle \)
  - The learning objective is not referred to as *hypothesis* explicitly, we simply just call it the *policy*. 

Outline

- Recap: Markov Decision Processes
- Reinforcement Learning Setup
  - What's changed from MDP?
- Model-based methods
- Model-free methods
  - (tabular) Q-learning
    - $\epsilon$-greedy action selection
    - exploration vs. exploitation
  - (neural network) Q-learning
- RL setup again
  - What's changed from supervised learning?
How do we learn a good policy without learning transition or rewards explicitly?

We kinda already know a way: Q functions!

2.4) (Recall from MDP lab)

We switch to an infinite-horizon scenario. For our stochastic machine, here is the infinite-horizon $Q$ function (computed via value iteration) for $\gamma$ near 1.

<table>
<thead>
<tr>
<th>State</th>
<th>wash</th>
<th>paint</th>
<th>eject</th>
</tr>
</thead>
<tbody>
<tr>
<td>dirty</td>
<td>$\begin{bmatrix} 2.32274541 , -0.70048204 , 0. \end{bmatrix}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clean</td>
<td>$\begin{bmatrix} 2.32274541 , 5.71581775 , 0. \end{bmatrix}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>painted</td>
<td>$\begin{bmatrix} 2.32274541 , 6.9 , 10. \end{bmatrix}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ejected</td>
<td>$\begin{bmatrix} 0. , 0. , 0. \end{bmatrix}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What is the optimal thing to do with a clean object?

What will you do if it becomes dirty?

Does this optimal policy make intuitive sense?

So once we have "good" Q values, we can find optimal policy easily.
But didn't we calculate this Q-table via value iteration using transition and rewards explicitly?
Indeed, recall that, in MDP:

- Finite horizon recursion
  \[ Q^h(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q^{h-1}(s', a') \]

- Infinite horizon equation
  \[ Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a') \]

- Infinite-horizon Value Iteration
  1. for \( s \in S, a \in A \):
     2. \( Q_{\text{old}}(s, a) = 0 \)
     3. while True:
        4. for \( s \in S, a \in A \):
           5. \( Q_{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}}(s', a') \)
           6. if \( \max_{s,a} |Q_{\text{old}}(s, a) - Q_{\text{new}}(s, a)| < \epsilon \):
              7. return \( Q_{\text{new}} \)
           8. \( Q_{\text{old}} \leftarrow Q_{\text{new}} \)
• value iteration relied on having full access to \( R \) and \( T \)

\[
Q_{\text{new}}(s, a) \leftarrow R(s, a) + \gamma \sum T(s, a, s') \max_{a'} Q_{\text{old}}(s', a')
\]

• hmm... perhaps, we could simulate \( s' \), observe \( r \) and \( s' \), and just use

\[
r + \gamma \max_{a'} Q_{\text{old}}(s', a')
\]

as the proxy for the r.h.s. assignment?

• BUT, this is basically saying the realized \( s' \) is the only possible next state; pretty rough! We'd override any previous "learned" Q values.

\[
\text{e.g. } Q_{\text{new}}(6, \uparrow) \leftarrow -10 + \gamma \max_{a'} Q_{\text{old}}(2, a')
\]

\[
Q_{\text{new}}(6, \uparrow) \leftarrow -10 + \gamma \max_{a'} Q_{\text{old}}(3, a')
\]

• **better** way is to smoothly keep track of what's our old belief with new evidence:

\[
Q_{\text{new}}(s, a) \leftarrow (1 - \alpha) Q_{\text{old}}(s, a) + \alpha \left( r + \gamma \max_{a'} Q_{\text{old}}(s', a') \right)
\]
VALUE-ITERATION \((S, A, T, R, \gamma, \epsilon)\)

1. for \(s \in S, a \in A:\)
2. \(Q_{\text{old}} (s, a) = 0\)
3. while True:
4. for \(s \in S, a \in A:\)
5. \(Q_{\text{new}} (s, a) \leftarrow R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{\text{old}} (s', a')\)
6. if \(\max_{s,a} |Q_{\text{old}} (s, a) - Q_{\text{new}} (s, a)| < \epsilon:\)
7. return \(Q_{\text{new}}\)
8. \(Q_{\text{old}} \leftarrow Q_{\text{new}}\)

"calculating"

Q-LEARNING \((S, A, \gamma, s_0, \alpha)\)

1. for \(s \in S, a \in A:\)
2. \(Q_{\text{old}} (s, a) = 0\)
3. \(s \leftarrow s_0\)
4. while True:
5. \(a \leftarrow \text{select_action} (s, Q_{\text{old}} (s, a))\)
6. \(r, s' \leftarrow \text{execute}(a)\)
7. \(Q_{\text{new}} (s, a) \leftarrow (1 - \alpha) Q_{\text{old}} (s, a) + \alpha (r + \gamma \max_{a'} Q_{\text{old}} (s', a'))\)
8. \(s \leftarrow s'\)
9. if \(\max_{s,a} |Q_{\text{old}} (s, a) - Q_{\text{new}} (s, a)| < \epsilon:\)
10. return \(Q_{\text{new}}\)
11. \(Q_{\text{old}} \leftarrow Q_{\text{new}}\)

"estimating"
Q-LEARNING \((\mathcal{S}, \mathcal{A}, \gamma, s_0, \alpha)\)

1. for \(s \in \mathcal{S}, a \in \mathcal{A}\):
2. \(Q_{\text{old}}(s, a) = 0\)
3. \(s \leftarrow s_0\)
4. while True:
5. \(a \leftarrow \text{select_action}(s, Q_{\text{old}}(s, a))\)
6. \(r, s' = \text{execute}(a)\)
7. \(Q_{\text{new}}(s, a) \leftarrow (1 - \alpha)Q_{\text{old}}(s, a) + \alpha(r + \gamma \max_{a'} Q_{\text{old}}(s', a'))\)
8. \(s \leftarrow s'\)
9. if \(\max_{s,a} |Q_{\text{old}}(s, a) - Q_{\text{new}}(s, a)| < \epsilon\):
10. \(\text{return } Q_{\text{new}}\)
11. \(Q_{\text{old}} \leftarrow Q_{\text{new}}\)

- Remarkably, still can converge. (So long \(\mathcal{S}, \mathcal{A}\) are finite; we visit every state and action infinity-many times; and \(\alpha\) decays.

  - Line 7:
  
  \[
  Q_{\text{new}}(s, a) \leftarrow (1 - \alpha)Q_{\text{old}}(s, a) + \alpha(r + \gamma \max_{a'} Q_{\text{old}}(s', a'))
  \]

  is equivalently:

  \[
  Q_{\text{new}}(s, a) \leftarrow Q_{\text{old}}(s, a) + \alpha([r + \gamma \max_{a'} Q_{\text{old}}(s', a')] - Q_{\text{old}}(s, a))
  \]

  \[
  \text{old belief} + \text{learning rate} \left(\text{target} - \text{old belief}\right)
  \]

  pretty similar to SGD.

  - Line 5, a sub-routine.
• If our $Q$ values are estimated quite accurately (nearly converged to the true $Q$ values), then should act greedily
  • $\arg\max_{a} Q^{h}(s, a)$, as we did in MDP.

• During learning, especially in early stages, we'd like to explore.

• $\epsilon$-greedy action selection strategy:
  • with probability $1 - \epsilon$, choose $\arg\max_{a} Q(s, a)$
  • with probability $\epsilon$, choose an action $a \in A$ uniformly at random

• Benefit: get to observe more diverse $(s, a)$ consequences.

• *exploration vs. exploitation.*
Outline

- Recap: Markov Decision Processes
- Reinforcement Learning Setup
  - What's changed from MDP?
- Model-based methods
- Model-free methods
  - (tabular) Q-learning
    - ε-greedy action selection
    - exploration vs. exploitation
  - (neural network) Q-learning
- RL setup again
  - What's changed from supervised learning?
• Q-learning only is kinda sensible for tabular setting.

• What do we do if $S$ and/or $A$ are large (or continuous)?

• Recall from Q-learning algorithm, key line 7:

$$Q_{\text{new}}(s, a) \leftarrow (1 - \alpha)Q_{\text{old}}(s, a) + \alpha \left( r + \gamma \max_{a'} Q_{\text{old}}(s', a') \right)$$

is equivalently:

$$Q_{\text{new}}(s, a) \leftarrow Q_{\text{old}}(s, a) + \alpha \left( [r + \gamma \max_{a'} Q_{\text{old}}(s', a')] - Q_{\text{old}}(s, a) \right)$$

old belief + learning rate ( target − old belief )

• Can be interpreted as we're minimizing:

$$(Q(s, a) - (r + \gamma \max_{a'} Q(s', a')))^2$$

via gradient method!
Outline

- Recap: Markov Decision Processes
- Reinforcement Learning Setup
  - What's changed from MDP?
- Model-based methods
- Model-free methods
  - (tabular) Q-learning
    - $\epsilon$-greedy action selection
    - exploration vs. exploitation
  - (neural network) Q-learning
- RL setup again
  - What's changed from supervised learning?
Supervised learning

Prediction \( \hat{y} \)

\[ f_\theta : X \to \mathbb{R}^K \]

dolphin

cat

grizzly bear

angel fish

chameleon

clown fish

iguana

elephant

\vdots

Ground truth label \( y \)

\[
\sum_{k=1}^{K} y_k \log \hat{y}_k
\]

dolphin

cat

grizzly bear

angel fish

chameleon

clown fish

iguana

elephant

\vdots

0 1

0 1

0
- If explicit “good” state-action pair is given, also supervised learning.
- Behavior cloning or imitation learning.
- But what if no explicit guide?

[Adapted from Andrej Karpathy: http://karpathy.github.io/2016/05/31/rl/]
- If no direct supervision is available?
- Strictly RL setting. Interact, observe and get data. Use rewards/value as "coy" supervision signal.

[Adapted from Andrej Karpathy: http://karpathy.github.io/2016/05/31/rl/]
We'd appreciate your feedback on the lecture.

Thanks!