Announcements

• Project Milestone 2 due Wednesday, April 5 at 8pm

• Homework 6 released Wednesday, April 5
  • Due Wednesday, April 19 at 8pm
Lecture 21: Reinforcement Learning

CIS 4190/5190
Spring 2023
Reinforcement Learning Problem

• At a high level, we need to specify the following:
  • **State space**: What are the observations the agent may encounter?
  • **Action space**: What are the actions the agent can take?
  • **Transitions/dynamics**: How the state is updated when taking an action
  • **Rewards**: What rewards the agent receives for taking an action in a state

• For most of today, assume state and action spaces are finite
Markov Decision Process (MDP)

• An MDP \((S, A, P, R, \gamma)\) is defined by:
  • Set of states \(s \in S\)
  • Set of actions \(a \in A\)
  • Transition function \(P(s' | s, a)\) (also called “dynamics” or the “model”)
  • Reward function \(R(s, a, s')\)
  • Discount factor \(\gamma < 1\)

• Also assume an initial state distribution \(D(s)\)
  • Often omitted since optimal policy does not depend on \(D\)

Image: https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690
Policy Gradient Algorithm

1. sample \( \{\tau^i\} \) from \( \pi_\theta(a_t|s_t) \) (run the policy)
2. \( \nabla_\theta J(\theta) \approx \sum_i (\sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)) \left( \sum_t r(s_t^i, a_t^i) \right) \)
3. \( \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \)
Reward-to-Go Function

\[ \nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t}|s_{i,t}) \left( \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right) \]

“reward to go” \( \hat{Q}_{i,t} \)

\( \hat{Q}_{i,t} \): estimate of expected reward if we take action \( a_{i,t} \) in state \( s_{i,t} \)

can we get a better estimate?
Reward-to-Go Function

$$
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left( \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right)
$$

“reward to go” $\hat{Q}_{i,t}$

$\hat{Q}_{i,t}$: estimate of expected reward if we take action $a_{i,t}$ in state $s_{i,t}$

can we get a better estimate?

$$
\hat{Q}_{i,t} \approx \sum_{t'=t}^{T} E_{\pi_\theta} [r(s_{t'}, a_{t'})|s_t, a_t]
$$
Reward-to-Go Function

\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left( \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right) \]

“reward to go” \( \hat{Q}_{i,t} \)

\( \hat{Q}_{i,t} \): estimate of expected reward if we take action \( a_{i,t} \) in state \( s_{i,t} \), can we get a better estimate?

\[ Q(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta} [r(s_{t'}, a_{t'})|s_t, a_t] \]: true expected reward-to-go

\[ \hat{Q}_{i,t} \approx \sum_{t'=t}^{T} E_{\pi_\theta} [r(s_{t'}, a_{t'})|s_t, a_t] \]
Space of RL Algorithms

Q-learning  actor-critic learning  REINFORCE
Toy Example

• Grid map with solid/open cells

• **State:** An open grid cell

• **Actions:** Move North, East, South, West

Based on slide by Dan Klein
Toy Example

• **Dynamics**
  • Move in chosen direction, but **not deterministically**!
  • Succeeds 80% of the time
  • 10% of the time, end up 90° off
  • 10% of the time, end up −90° off
  • The agent stays put if it tries to move into a solid cell or outside the world
  • At terminal states, any action ends **episode** (or **rollout**)
Toy Example

• **Rewards**
  • At terminal state, agent receives the specified reward
  • For each timestep outside terminal states, the agent pays a small cost, e.g., a “reward” of $-0.03$
Optimal Policy

- **Optimal policy**: Following $\pi^*$ maximizes total reward received
  - **Discounted**: Future rewards are downweighted
  - **In expectation**: On average across randomness of environment and actions

Based on slide by Dan Klein
Markov Decision Process (MDP)

• **Goal:** Maximize cumulative expected discounted reward:

\[
\pi^* = \max_\pi J(\pi) \quad \text{where} \quad J(\pi) = \mathbb{E}_\zeta \left[ \sum_{t=0}^{\infty} \gamma^t \cdot r_t \right]
\]

• Expectation over episodes \( \zeta = (s_0, a_0, r_0, s_1, \ldots) \), where
  • \( s_0 \sim D \)
  • \( a_t = \pi(s_t) \)
  • \( s_{t+1} \sim P(\cdot|s_t, a_t) \)
  • \( r_t = R(s_t, a_t, s_{t+1}) \)
Policy Value Function

- **Policy Value Function:** Expected reward if we start in \( s \) and use \( \pi \):

  \[
  V^\pi(s) = \mathbb{E} \left( \sum_{t=0}^{\infty} \gamma^t \cdot r_t \mid s_0 = s \right)
  \]

- **Bellman equation:**

  
  \[
  V^\pi(s) = \sum_{s' \in S} P(s' \mid s, \pi(s)) \cdot \left( R(s, \pi(s), s') + \gamma \cdot V^\pi(s') \right)
  \]

  - current value
  - expectation over next state
  - current reward + discounted future reward
Optimal Value Function

• **Optimal value function:** Expected reward if we start in $s$ and use $\pi^*$:

$$V^*(s) = \mathbb{E} \left( \sum_{t=0}^{\infty} \gamma^t \cdot r_t \mid s_0 = s \right)$$

• **Bellman equation:**

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s' \mid s, a) \cdot \left( R(s, a, s') + \gamma \cdot V^*(s') \right)$$

Optimal policy selects action that maximizes future expected reward from state $s$
Optimal Value Function

• Bellman equation:

\[ V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s' | s, a) \cdot (R(s, a, s') + \gamma \cdot V^*(s')) \]

• Do not need to know the optimal policy \( \pi^* \)!

• **Strategy:** Compute \( V^* \) and then use it to compute \( \pi^* \)
  - **Caveat:** Latter step requires knowing \( P \)
Policy Action-Value Function

- **Policy Action-Value Function (or Q function):** Expected reward if we start in $s$, take action $a$, and then use $\pi$ thereafter:

  $$Q^\pi(s, a) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t \cdot r_t \mid s_0 = s, a_0 = a\right)$$

- **Bellman equation:**

  $$Q^\pi(s, a) = \sum_{s' \in S} P(s' \mid s, a) \cdot \left( R(s, a, s') + \gamma \cdot Q^\pi(s', \pi(s')) \right)$$
Optimal Action-Value Function

• Optimal Action-Value Function (or Q function): Expected reward if we start in $s$, take action $a$, and then act optimally thereafter:

$$Q^*(s, a) = \mathbb{E} \left( \sum_{t=0}^{\infty} \gamma^t \cdot r_t \mid s_0 = s, a_0 = a \right)$$

• Bellman equation:

$$Q^*(s, a) = \sum_{s' \in S} P(s' \mid s, a) \cdot \left( R(s, a, s') + \gamma \cdot \max_{a' \in A} Q^*(s', a') \right)$$
Relationship

• We have

\[ V^\pi(s) = Q^\pi(s, \pi(s)) \]

• Similarly, we have

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

• We have

\[ \pi^*(s) = \max_{a \in A} Q^*(s, a) \]

• **Strategy:** Compute \( Q^* \) and then use it to compute \( \pi^* \)
Q Iteration

- Initialize $Q_1(s, a) \leftarrow 0$ for all $s, a$
- For $i \in \{1, 2, \ldots\}$ until convergence:

$$Q_{i+1}(s, a) \leftarrow \sum_{s' \in S} P(s' | s, a) \cdot \left( R(s, a, s') + \gamma \cdot \max_{a' \in A} Q_i(s', a') \right)$$
$Q_{t+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_t(s', a') \right]$
\[ Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] \]

\[
0.8x[0+0.9x1] + 0.1x[0+0] + 0.1x[0+0] = 0.72
\]
\[ Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s' | s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] \]

0.8x[0+0] + 0.1x[0+0.9x1] + 0.1x[0+0] = 0.09
\[ Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] \]

\[
\begin{align*}
0.8x[0+0] &+ 0.1x[0+0.9x1] \\
&+ 0.1x[0+0] = 0.09
\end{align*}
\]
\[ Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] \]
\[ Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] \]

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

\[
0.8x[0+0.9x-1] + 0.1x[0+0] + 0.1x[0+0] = -0.72
\]
\[ Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] \]

\[ 0.8 \times [0+0] + 0.1 \times [0+0] + 0.1 \times [0+0.9 \times -1] = -0.09 \]
$$Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s' \mid s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right]$$

$$0.8x[0+0] + 0.1x[0+0.9x-1] + 0.1x[0+0] = -0.09$$
\[
Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right]
\]

0.8 \times [0+0] + 0.1 \times [0+0] + 0.1 \times [0+0] = 0
Now we have $Q_1(s, a)$ for all $(s, a)$
\[ Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] \]

0.8x[0+0.9x1] 
+ 0.1x[0+0.9x0.72] 
+ 0.1x[0+0] 
= 0.7848
\[ Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] \]

\[
\begin{align*}
0.8\times[0+0] \\
+ 0.1\times[0+0.9\times1] \\
+0.1\times[0+0] \\
=0.09
\end{align*}
\]
After 1000 iterations:

\[ Q_{i+1}(s, a) \leftarrow \sum_{s'} P(s' \mid s, a) \left[ R(s, a, s') + \gamma \max_{a'} Q_i(s', a') \right] \]
Q Iteration

• Information propagates outward from terminal states

• Eventually all state-action pairs converge to correct Q-value estimates
Aside: Value Iteration

• Analogous to Q-Policy iteration but for computing the value function

• Initialize $V_1(s) \leftarrow 0$ for all $s$

• For $i \in \{1, 2, \ldots\}$ until convergence:

$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s' | s, a) \cdot (R(s, a, s') + \gamma \cdot V_i(s'))$$
Example MDP

\[ V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a) [R(s,a,s') + \gamma V_i(s')] \]
Example MDP

$$V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V_i(s')]$$

$$V_2(\langle 4,3 \rangle) \leftarrow 1$$

$$V_2(\langle 4,2 \rangle) \leftarrow -1$$
\[ V_{i+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} P(s'|s, a)[R(s, a, s') + \gamma V_i(s')] \]

Example MDP

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
1 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
1 & 0 & 0 & 0 \\
2 & 0 & -1 & 0 \\
3 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
1 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 \\
\end{array}
\]\n
\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\hline
1 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 \\
\end{array}
\]
Reinforcement Learning

• Q iteration can be used to compute the optimal Q function when \( P \) and \( R \) are known

• How can we adapt it to the setting where these are unknown?
  • **Observation:** Every time you take action \( a \) from state \( s \), you obtain one sample \( s' \sim P(\cdot|s,a) \) and observe \( R(s,a,s') \)
  • Use single sample instead of full \( P \)
Q Learning

• Can we learn $\pi^*$ without explicitly learning $P$ and $R$?

$$Q_{i+1}(s, a) \leftarrow \sum_{s' \in S} P(s' \mid s, a) \cdot \left( R(s, a, s') + \gamma \cdot \max_{a' \in A} Q_i(s', a') \right)$$
Q Learning

• Can we learn $\pi^*$ without explicitly learning $P$ and $R$?

$$Q_{i+1}(s, a) \leftarrow \mathbb{E}_{s' \sim P(\cdot|s, a)} \left[ R(s, a, s') + \gamma \cdot \max_{a' \in A} Q_i(s', a') \right]$$
Q Learning

• Q Learning update:

\[ Q_{i+1}(s, a) \leftarrow R(s, a, s') + \gamma \cdot \max_{a' \in A} Q_i(s', a') \]

• Q Iteration: Update for all \((s, a, s')\) at each step

• Q Learning: Update just for current \((s, a)\), and approximate with the state \(s'\) we actually reached (i.e., a single sample \(s' \sim P(\cdot|s, a)\))
Q Learning

• **Problem:** Forget everything we learned before (i.e., $Q_i(s, a)$)

• **Solution:** Incremental update:

$$Q_{i+1}(s, a) \leftarrow (1 - \alpha) \cdot Q_i(s, a) + \alpha \cdot \left( R(s, a, s') + \gamma \cdot \max_{a' \in A} Q_i(s', a') \right)$$
Sample $R + \gamma \max Q = 0 + 0.9 \times 0.78 = 0.702$

New $Q = 0.09 + 0.1 \times (0.702 - 0.09) = 0.1512$

$Q(s, a) \leftarrow Q(s, a) + \alpha \left( R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$
After 100,000 actions:

\[
Q(s, a) \leftarrow Q(s, a) + \alpha \left( R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)
\]
Policy for Gathering Data

• **Strategy 1:** Randomly explore all \((s, a)\) pairs
  • Not obvious how to do so!
  • E.g., if we act randomly, it may take a very long time to explore states that are difficult to reach

• **Strategy 2:** Use current best policy
  • Can get stuck in local minima
  • E.g., we may never discover a shortcut if it sticks to a known route to the goal
Policy for Gathering Data

• $\epsilon$-greedy:
  • Play current best with probability $1 - \epsilon$ and randomly with probability $\epsilon$
  • Can reduce $\epsilon$ over time
  • Works okay, but exploration is undirected

• Visitation counts:
  • Maintain a count $N(s, a)$ of number of times we tried action $a$ in state $s$
  • Choose $a^* = \arg\max_{a \in A} \left\{ Q(s, a) + \frac{1}{N(s,a)} \right\}$, i.e., inflate less visited states
Summary

• **Q iteration:** Compute optimal Q function when the transitions and rewards are known

• **Q learning:** Compute optimal Q function when the transitions and rewards are unknown

• **Extensions**
  • Various strategies for exploring the state space during learning
  • Handling large or continuous state spaces
Curse of Dimensionality

• How large is the state space?
  • **Gridworld:** One for each of the \( n \) cells
  • **Pacman:** State is \((\text{player, ghost}_1, \ldots, \text{ghost}_k)\), so there are \( n^k \) states!

• **Problem:** Learning in one state does not tell us anything about the other states!

• Many states \( \rightarrow \) learn very slowly
State-Action Features

• Can we learn **across** state-action pairs?

• Yes, use features!
  • \( \phi(s, a) \in \mathbb{R}^d \)
  • Then, learn to predict \( Q^*(s, a) \approx Q_\theta(s, a) = f_\theta(\phi(s, a)) \)
  • Enables generalization to similar states