Markov Decision Processes (2)

Lecture 4, CMU 10-403

Katerina Fragkiadaki
Used Materials

• **Disclaimer**: Some material and slides for this lecture were borrowed from Rich Sutton’s class and David Silver’s class on Reinforcement Learning.
An operator $F$ on a normed vector space $\mathcal{X}$ is a $\gamma$-contraction, for $0 < \gamma < 1$, provided for all $x, y \in \mathcal{X}$

$$||T(x) - T(y)|| \leq \gamma ||x - y||$$

**Theorem (Contraction mapping)**
For a $\gamma$-contraction $F$ in a complete normed vector space $\mathcal{X}$

- $F$ converges to a unique fixed point in $\mathcal{X}$
- at a linear convergence rate $\gamma$

**Remark.** In general we only need metric (vs normed) space
Value Function Sapce

- Consider the vector space $V$ over value functions
- There are $|\mathcal{S}|$ dimensions
- Each point in this space fully specifies a value function $v(s)$
- Bellman backup brings value functions closer in this space
- And therefore the backup must converge to a unique solution
We will measure distance between state-value functions $u$ and $v$ by the $\infty$-norm.

i.e. the largest difference between state values,

$$||u - v||_\infty = \max_{s \in S} |u(s) - v(s)|$$

$$||u||_\infty = \max_{s \in S} |u(s)|$$
Bellman Expectation Backup is a Contraction

- Define the Bellman expectation backup operator
  \[
  F^\pi(v) = r^\pi + \gamma T^\pi v
  \]
- This operator is a \( \gamma \)-contraction, i.e. it makes value functions closer by at least \( \gamma \),

\[
\|F^\pi(u) - F^\pi(v)\|_\infty = \|(r^\pi + \gamma T^\pi u) - (r^\pi + \gamma T^\pi v)\|_\infty \\
= \|\gamma T^\pi (u - v)\|_\infty \\
\leq \|\gamma T^\pi (1)\|(u - v)\|_\infty\|_\infty \\
= \|\gamma (T^\pi 1)\| u - v\|_\infty\|_\infty \\
= \|\gamma 1\| u - v\|_\infty\|_\infty \\
= \gamma \|u - v\|_\infty
\]
The Bellman expectation operator $F^\pi$ has a unique fixed point.

$v_\pi$ is a fixed point of $F^\pi$ (by Bellman expectation equation).

By contraction mapping theorem.

Iterative policy evaluation converges to $v_\pi$. 

Convergence of Iter. Policy Evaluation and Policy Iteration
Policy Improvement

- Suppose we have computed $v_\pi$ for a deterministic policy $\pi$.
- For a given state $s$, would it be better to do an action $a \neq \pi(s)$?
- It is better to switch to action $a$ for state $s$ if and only if
  \[ q_\pi(s, a) > v_\pi(s) \]
- And we can compute $q_\pi(s, a)$ from $v_\pi$ by:
  \[ q_\pi(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) v_\pi(s') \]
Do this for all states to get a new policy $\pi' \geq \pi$ that is greedy with respect to $V_\pi$:

$$\pi'(s) = \arg \max_a q_\pi(s, a)$$

$$= \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(s')|S_t = s, A_t = a]$$

$$= \arg \max r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) v_\pi(s')$$
Do this for all states to get a new policy $\pi' \geq \pi$ that is greedy with respect to $v_\pi$:

$$\pi'(s) = \arg \max_a q_\pi(s, a)$$

$$= \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(s') | S_t = s, A_t = a]$$

$$= \arg \max_a r(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) v_\pi(s')$$

After policy update it holds that:

$$q_{\pi_k}(s, \pi_{k+1}(s)) = q_{\pi_k}(s, \arg \max_a q_{\pi_k}(s, a))$$

$$= \max_a q_{\pi_k}(s, a)$$

$$\geq q_{\pi_k}(s, \pi_k(s))$$

$$\geq v_{\pi_k}(s).$$

$$v_\pi(s) = \sum_{a \in A} \pi(a | s) q_\pi(s, a)$$
After policy update it holds that:

$$v_{\pi_k}(s) \leq q_{\pi_k}(s, \pi_{k+1}(s))$$

We have indeed improved the policy (or ended up on an equally good policy)

\[
v_\pi(s) \leq q_\pi(s, \pi'(s)) \\
= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
= \mathbb{E}_\pi'[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
\leq \mathbb{E}_\pi'[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
= \mathbb{E}_\pi'[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
= \mathbb{E}_\pi'[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s] \\
\leq \mathbb{E}_\pi'[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) | S_t = s] \\
\vdots \\
\leq \mathbb{E}_\pi'[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots | S_t = s] \\
= v_{\pi'}(s).
\]
If policy is unchanged after the greedification step, this means that:

\[ v_\pi(s) = \max_{a \in \mathcal{A}} \left( r(s, a) + \gamma \sum_{s'} p(s' | s, a) v_\pi(s') \right) \]

\[ v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a) \]

But this is the Bellman optimality Equation. So \( v_{\pi}=v^* \) and \( \pi \) is optimal.
Policy Iteration

\[\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_*\]

- policy evaluation
- policy improvement
- “greedification”
Policy Iteration

1. Initialization
   \( V(s) \in \mathbb{R} \) and \( \pi(s) \in \mathcal{A}(s) \) arbitrarily for all \( s \in S \)

2. Policy Evaluation
   Repeat
   \[ \Delta \leftarrow 0 \]
   For each \( s \in S \):
   \[ v \leftarrow V(s) \]
   \[ V(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s) (r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s')) \]
   \[ \Delta \leftarrow \max(\Delta, |v - V(s)|) \]
   until \( \Delta < \theta \) (a small positive number)

3. Policy Improvement
   \( policy-stable \leftarrow true \)
   For each \( s \in S \):
   \[ a \leftarrow \pi(s) \]
   \[ \pi(s) \leftarrow \text{arg max } r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v_{\pi}(s') \]
   If \( a \neq \pi(s) \), then \( policy-stable \leftarrow false \)
   If \( policy-stable \), then stop and return \( V \) and \( \pi \); else go to 2
Generalized Policy Iteration

• Does policy evaluation need to converge to $v_\pi$?
• Or should we introduce a stopping condition
  • e.g. $\epsilon$-convergence of value function
• Or simply stop after $k$ iterations of iterative policy evaluation?
• For example, in the small grid world $k = 3$ was sufficient to achieve optimal policy
• **Why not update policy every iteration?** i.e. stop after $k = 1$
  • This is equivalent to value iteration (next section)
Generalized Policy Iteration (GPI): any interleaving of policy evaluation and policy improvement, independent of their granularity.

A geometric metaphor for convergence of GPI:
Principle of Optimality

- Any optimal policy can be subdivided into two components:
  - An optimal first action $A_*$
  - Followed by an optimal policy from successor state $S'$

Theorem (Principle of Optimality)

- A policy $\pi(a|s)$ achieves the optimal value from state $s$, $v_\pi(s) = v_*(s)$, if and only if
- For any state $S'$ reachable from $s$, $\pi$ achieves the optimal value from state $S'$, $v_\pi(S') = v_*(S')$
• Problem: find optimal policy $\pi$
• Solution: iterative application of Bellman optimality backup
• $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_*$
• Using synchronous backups
  • At each iteration $k + 1$
  • For all states $s \in S$
  • Update $v_{k+1}(s)$ from $v_k(s')$
Value Iteration (2)

\[ v_{k+1}(s) \leftarrow s \]

\[ v_k(s') \leftarrow s' \]

\[ v_{[k+1]}(s) = \max_{a \in \mathcal{A}} \left( r(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' | s, a) v_{[k]}(s') \right), \forall s \]

\[ v_{k+1} = \max_{a \in \mathcal{A}} r(a) + \gamma p(a) v_k \]
Bellman Optimality Backup is a Contraction

- Define the Bellman optimality backup operator $F^*$

$$F^*(v) = \max_{a \in A} r(a) + \gamma p(a)v$$

- This operator is a $\gamma$-contraction, i.e. it makes value functions closer by at least $\gamma$ (similar to previous proof)

$$||F^*(u) - F^*(v)||_\infty \leq \gamma ||u - v||_\infty$$
Convergence of Value Iteration

- The Bellman optimality operator $F^*$ has a unique fixed point
- $v_*$ is a fixed point of $F^*$ (by Bellman optimality equation)
- By contraction mapping theorem
- Value iteration converges on $v_*$
Synchronous Dynamic Programming Algorithms

<table>
<thead>
<tr>
<th>Problem</th>
<th>Bellman Equation</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction</td>
<td>Bellman Expectation Equation</td>
<td>Iterative Policy Evaluation</td>
</tr>
<tr>
<td>Control</td>
<td>Bellman Expectation Equation + Greedy Policy Improvement</td>
<td>Policy Iteration</td>
</tr>
<tr>
<td>Control</td>
<td>Bellman Optimality Equation</td>
<td>Value Iteration</td>
</tr>
</tbody>
</table>

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for $m$ actions and $n$ states
- Could also apply to action-value function $q_\pi(s, a)$ or $q_*(s, a)$
- Complexity $O(m^2n^2)$ per iteration
• We are investigating finite MDPs: finite sets of actions and states

• We explained why value functions are important

• We discussed two ways to compute optimal policies: policy iteration and value iteration

• We saw that value iteration and policy evaluation converge to \( v^* \) and \( v_{\pi_i} \) and that policy iteration converges to the optimal policy and optimal value function \( (\pi^*, v^*) \)

• We have understood that exhaustive state sweeps (synchronous dynamic programming) are hopeless…

Can we change that?
Efficiency of DP

- To find an optimal policy is polynomial in the number of states...

- BUT, the number of states is often astronomical, e.g., often growing exponentially with the number of state variables (what Bellman called “the curse of dimensionality”).

- In practice, classical DP can be applied to problems with a few millions of states.
Asynchronous DP

- All the DP methods described so far require exhaustive sweeps of the entire state set.

- Asynchronous DP does not use sweeps. Instead it works like this:
  - Repeat until convergence criterion is met:
    - Sample a state at random and apply the appropriate backup

- Still need lots of computation, but does not get locked into hopelessly long sweeps

- Guaranteed to converge if all states continue to be selected
Asynchronous Dynamic Programming

Three simple ideas for asynchronous dynamic programming:

- In-place dynamic programming
- Prioritized sweeping
- Real-time dynamic programming
In-Place Dynamic Programming

- Synchronous value iteration stores two copies of value function
  - for all $s$ in $S$
    $$v_{new}(s) \leftarrow \max_{a \in A} \left( r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)v_{old}(s') \right)$$
    
  $v_{old} \leftarrow v_{new}$
  - In-place value iteration only stores one copy of value function
  - for all $s$ in $S$
    $$v(s) \leftarrow \max_{a \in A} \left( r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a)v(s') \right)$$
Prioritized Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.

\[
\max_{a \in A} \left( r(s, a) + \gamma \sum_{s' \in S} P(s' | s, a)v(s') \right) - v(s)
\]

- Backup the state with the largest remaining Bellman error
- Update Bellman poor of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue
Example: Shortest Path

<table>
<thead>
<tr>
<th>Problem</th>
<th>V₁</th>
<th>V₂</th>
<th>V₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0 -1 -1 -1</td>
<td>0 -1 -1 -1</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>-1 -1 -1 -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>-2 -2 -2 -2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V₄</th>
<th>V₅</th>
<th>V₆</th>
<th>V₇</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -1 -2 -3</td>
<td>0 -1 -2 -3</td>
<td>0 -1 -2 -3</td>
<td>0 -1 -2 -3</td>
</tr>
<tr>
<td>-1 -2 -3 -3</td>
<td>-1 -2 -3 -4</td>
<td>-1 -2 -3 -4</td>
<td>-1 -2 -3 -4</td>
</tr>
<tr>
<td>-2 -3 -3 -3</td>
<td>-2 -3 -4 -4</td>
<td>-2 -3 -4 -5</td>
<td>-2 -3 -4 -5</td>
</tr>
<tr>
<td>-3 -3 -3 -3</td>
<td>-3 -4 -4 -4</td>
<td>-3 -4 -5 -5</td>
<td>-3 -4 -5 -6</td>
</tr>
</tbody>
</table>
Real-time Dynamic Programming

• Idea: only states that are relevant to agent
• Use agent’s experience to guide the selection of states
• After each time-step $S_t, A_t, r_{t+1}$
• Backup the state $S_t$

$$v(S_t) \leftarrow \max_{a \in A} \left( r(S_t, a) + \gamma \sum_{s' \in S} p(s' | S_t, a)v(s') \right)$$