Chapter 4: Dynamic Programming

Seungjae Ryan Lee
Dynamic Programming

- Algorithms to compute optimal policies with a **perfect model of environment**
- Use value functions to structure searching for good policies
- Foundation of all methods hereafter
Policy Evaluation (Prediction)

- Compute state-value $v_{\pi}(s)$ for some policy $\pi$
- Use the Bellman Equation:

$$v_{\pi}(s) \doteq \mathbb{E}_{\pi}[G_t \mid S_t = s]$$
$$= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$
$$= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[ r + \gamma \mathbb{E}_{\pi}[G_{t+1} \mid S_{t+1} = s'] \right]$$
$$= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \left[ r + \gamma v_{\pi}(s') \right], \quad \text{for all } s \in S,$$
Iterative Policy Evaluation

- Solving linear systems is tedious → Use iterative methods
- Define sequence of approximate value functions \( v_0, v_1, v_2 \ldots \)
- *Expected update* using the Bellman equation:
  - Update based on *expectation of all possible next states*

\[
v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
= \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a) \left[ r + \gamma v_k(s') \right],
\]
Iterative Policy Evaluation in Practice

- In-place methods usually converge faster than keeping two arrays
- Terminate policy evaluation when $\max_s |v_{k+1}(s) - v_k(s)|$ is sufficiently small

**Iterative Policy Evaluation, for estimating $V \approx v_\pi$**

Input $\pi$, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in S^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in S$:
  $v \leftarrow V(s)$
  $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$
  $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  until $\Delta < \theta$
Gridworld Example

- Deterministic state transition
- Off-the-grid actions leave the state unchanged
- Undiscounted, episodic task

\[ R_t = -1 \]
on all transitions
Policy Evaluation in Gridworld

- Random policy $\pi$

<table>
<thead>
<tr>
<th>$k = 0$</th>
<th>$k = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 0.0 0.0 0.0</td>
<td>0.0 -2.4 -2.9 -3.0</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0</td>
<td>-2.4 -2.9 -3.0 -2.9</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0</td>
<td>-2.9 -3.0 -2.9 -2.4</td>
</tr>
<tr>
<td>0.0 0.0 0.0 0.0</td>
<td>-3.0 -2.9 -2.4 0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k = 1$</th>
<th>$k = 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 -1.0 -1.0 -1.0</td>
<td>0.0 -6.1 -8.4 -9.0</td>
</tr>
<tr>
<td>-1.0 -1.0 -1.0 -1.0</td>
<td>-6.1 -7.7 -8.4 -8.4</td>
</tr>
<tr>
<td>-1.0 -1.0 -1.0 -1.0</td>
<td>-8.4 -8.4 -7.7 -6.1</td>
</tr>
<tr>
<td>-1.0 -1.0 -1.0 0.0</td>
<td>-9.0 -8.4 -6.1 0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k = 2$</th>
<th>$k = \infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 -1.7 -2.0 -2.0</td>
<td>0.0 -14.0 -20.0 -22.0</td>
</tr>
<tr>
<td>-1.7 -2.0 -2.0 -2.0</td>
<td>-14.0 -18.0 -20.0 -20.0</td>
</tr>
<tr>
<td>-2.0 -2.0 -2.0 -1.7</td>
<td>-20.0 -20.0 -18.0 -14.0</td>
</tr>
<tr>
<td>-2.0 -2.0 -1.7 0.0</td>
<td>-22.0 -20.0 -14.0 0.0</td>
</tr>
</tbody>
</table>
Policy Improvement - One state

- Suppose we know $v_\pi$ for some policy $\pi$
- For a state $s$, see if there is a better action $a \neq \pi(s)$
- Check if $q_\pi(s, a) \geq v_\pi(s)$
  - If true, greedily selecting $a$ is better than $\pi(s)$
  - Special case of Policy Improvement Theorem
Policy Improvement Theorem

For policies $\pi, \pi'$, if for all state $s \in S$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

Then, $\pi'$ is at least as good a policy as $\pi$.

$$v_{\pi'}(s) \geq v_\pi(s)$$

(Strict inequality if $q_\pi(s, \pi'(s)) > v_\pi(s)$)
Policy Improvement

- Find better policies with the computed value function
- Use a new greedy policy $\pi'$
- Satisfies the conditions of Policy Improvement Theorem

\[
\pi'(s) = \arg \max_a q_\pi(s, a) = \arg \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] = \arg \max_a \sum_{s', r} p(s', r \mid s, a) \left[ r + \gamma v_\pi(s') \right],
\]
Guarantees of Policy Improvement

- If $v_{\pi} = v_{\pi'}$, then the Bellman Optimality Equation holds.

\[
v_{\pi'}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a] \\
= \max_a \sum_{s', r} p(s', r | s, a) \left[ r + \gamma v_{\pi'}(s') \right].
\]

→ Policy Improvement always returns a better policy unless already optimal
Policy Iteration

- Repeat *Policy Evaluation* and *Policy Improvement*
- Guaranteed improvement for each policy
- Guaranteed convergence in finite number of steps for finite MDPs

\[
\begin{align*}
\pi_0 & \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_* ,
\end{align*}
\]
Policy Iteration in Practice

- Initialize $V_{\pi t+1}$ with $V_{\pi t}$ for quicker policy evaluation
- Often converges in surprisingly few iterations

---

**Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$**

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

2. **Policy Evaluation**
   
   Loop:
   
   \[ \Delta \leftarrow 0 \]
   
   Loop for each $s \in \mathcal{S}$:
   
   \[ v \leftarrow V(s) \]
   
   \[ V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s)) \left[ r + \gamma V(s') \right] \]
   
   \[ \Delta \leftarrow \max(\Delta,|v - V(s)|) \]
   
   until \( \Delta < \theta \) (a small positive number determining the accuracy of estimation)

3. **Policy Improvement**

   $\text{policy-stable} \leftarrow \text{true}$

   For each $s \in \mathcal{S}$:

   \[ \text{old-action} \leftarrow \pi(s) \]
   
   \[ \pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a) \left[ r + \gamma V(s') \right] \]

   If $\text{old-action} \neq \pi(s)$, then $\text{policy-stable} \leftarrow \text{false}$

   If $\text{policy-stable}$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
Value Iteration

- “Truncate” policy evaluation
  - Don’t wait until $\max_s |v_{k+1}(s) - v_k(s)|$ is sufficiently small
  - Update state values **once** for each state

- **Evaluation and improvement** can be simplified to one update operation
  - Bellman optimality equation turned into an update rule

$$
\begin{align*}
v_{k+1}(s) &= \sum_a \pi_k(a, s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')] \\
&= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')]
\end{align*}
$$
Value Iteration in Practice

- Terminate when $\max_s |v_{k+1}(s) - v_k(s)|$ is sufficiently small

---

**Value Iteration, for estimating $\pi \approx \pi_*$**

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in S^+$, arbitrarily except that $V(\text{terminal}) = 0$

Loop:
- $\Delta \leftarrow 0$
- Loop for each $s \in S$:
  - $v \leftarrow V(s)$
  - $V(s) \leftarrow \max_a \sum_{s', r} p(s', r|s, a) [r + \gamma V(s')]$
  - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s', r} p(s', r|s, a) [r + \gamma V(s')]$
Asynchronous Dynamic Programming

- Don’t sweep over the entire state set systematically
  - Some states are updated multiple times before other state is updated once
  - Order/skip states to propagate information efficiently
- Can intermix with real-time interaction
  - Update states according to the agent’s experience
  - Allow focusing updates to relevant states
- To converge, all states must be continuously updated
Generalized Policy Iteration

- Idea of interaction between policy evaluation and policy improvement
  - Policy improved w.r.t. value function
  - Value function updated for new policy
- Describes most RL methods
- Stabilized process guarantees optimal policy
Efficiency of Dynamic Programming

- Polynomial in $|S|$ and $|A|$  
  - Exponentially faster than direct search in policy space $|A|^{|S|}$
- More practical than linear programming methods in larger problems  
  - Asynchronous DP preferred for large state spaces
- Typically converge faster than their worst-case guarantee  
  - Initial values can help faster convergence
Thank you!

Original content from

- Reinforcement Learning: An Introduction by Sutton and Barto

You can find more content in

- github.com/seungjaeryanlee
- www.endtoend.ai