Tabular Solution Methods
Dynamic Programming

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Spring 2020

avereshc@buffalo.edu

February 5, 2020
Overview

1. Recap

2. Bellman Equation

3. Model Based

4. Iterative Policy Evaluation

5. Policy Improvement
Table of Contents

1 Recap

2 Bellman Equation

3 Model Based

4 Iterative Policy Evaluation

5 Policy Improvement
Recap: Finite Markov Decision Processes (MDP)
RL agents learn to maximize cumulative future reward ($R$).
There are two types of value functions:
Value Functions

There are two types of value functions:

- state value function $V(s)$
- action value function $Q(s, a)$
State value function $V(s)$

**Definition**

State value function describes the value of a state when following a policy. It is the expected return when starting from state $s$ acting according to our policy $\pi$:

$$V^\pi(s) = E_\pi[R_t | S_t = s]$$
**Action value function** $Q(s, a)$

**Definition**

*Action value function* tells us the value of taking an action $a$ in state $s$ when following a certain policy $\pi$. It is the expected return given the state and action under $\pi$:

$$Q^\pi(s, a) = \mathbb{E}_\pi[R_t | S_t = s, A_t = a]$$
Policies $\pi$

**Definition**

A *policy* $\pi$ is a distribution over actions given states. It defines the agent’s behaviour. It can be either deterministic or stochastic:

- **Deterministic**: $\pi(s) = a$
- **Stochastic**: $\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- **Optimal policy** $\pi^*$: policy that maximizes the expectation of cumulative reward.
Solving a reinforcement learning task means finding an optimal policy $\pi^*$ that achieves maximum reward over the long run.

**Optimal (greedy) policy** $\pi^*$ chooses an action ($a$), such that

$$a = \arg \max_a [E_\pi [r_t + \gamma V^*(S_{t+1}|S_t = s, A_t = a)]]$$
Solving a reinforcement learning task means finding an optimal policy $\pi^*$ that achieves maximum reward over the long run.

**Optimal (greedy) policy** $\pi^*$ chooses an action ($a$), such that

1. $a = \arg\max_a \left[ \mathbb{E}_{\pi} [r_t + \gamma V^*(S_{t+1}|S_t = s, A_t = a)] \right]$

2. $a = \arg\max_a [Q^*_\pi(s, a)]$
Table of Contents

1 Recap

2 Bellman Equation

3 Model Based

4 Iterative Policy Evaluation

5 Policy Improvement
Richard Bellman was an American applied mathematician who derived the following equations which allow us to start solving these MDPs. The Bellman equations are ubiquitous in RL and are necessary to understand how RL algorithms work.
Bellman equation for the state value function

\[ V^\pi(s) = \mathbb{E}_\pi[R_t | S_t = s] \]
Bellman equation for the state value function

\[ V^\pi(s) = \mathbb{E}_\pi[R_t | S_t = s] \]

\[ = \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots | S_t = s] \]
Bellman equation for the state value function

\[ V^\pi(s) = \mathbb{E}_\pi[R_t | S_t = s] \]
\[ = \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots | S_t = s] \]
\[ = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s] \]
Bellman equation for the state value function

\[ V^\pi(s) = \mathbb{E}_\pi[R_t | S_t = s] \]

\[ = \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots | S_t = s] \]

\[ = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s] \]

\[ = \mathbb{E}_\pi[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | S_t = s] \]
The expectation here describes what we expect the return to be if we continue from state $s$ following policy $\pi$. The expectation can be written explicitly by summing over all possible actions and all possible returned states.

$$E_{\pi}[r_{t+1}|s_t = s] = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a R_{ss'}^a$$

$$E_{\pi}[\gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|S_t = s] = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a \gamma E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|S_{t+1} = s'\right]$$
Bellman equation for the state value function

By distributing the expectation between these two parts, we can then manipulate our equation into the form:

\[
V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a \left[ r_{t+1} + \gamma \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right] \right]
\]
Bellman equation for the state value function

By distributing the expectation between these two parts, we can then manipulate our equation into the form:

\[
V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P^a_{ss'} \left[ r_{t+1} + \gamma \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right] \right]
\]

\[
= \sum_a \pi(s, a) \sum_{s'} P^a_{ss'} \left[ r_{t+1} + \gamma V^\pi(s') \right]
\]
Bellman equations is that they let us express values of states as values of other states. This means that if we know the value of $s_{t+1}$, we can very easily calculate the value of $s_t$.

Bellman equations is a foundation for iterative approaches to solve reinforcement learning task.
Goal: compute optimal policies given a perfect model of the environment
Intuitively: value of a state under an optimal policy must equal the expected return for the best action from that state
Optimal Value Functions

\[ V^*(s) = \max_a \mathbb{E}[r_{t+1} + \gamma V^*(S_{t+1}) | S_t = s, A_t = a] \]

\[ = \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V^*(s')] \]

Intuitively: value of a state under an optimal policy must equal the expected return for the best action from that state

\[ Q^*(s, a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a] \]

\[ = \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} Q^*(s', a')] \]
Backup Diagrams
Backup Diagrams (a) V* (b) Q*
Estimating $V_\pi$ or $Q_\pi$ is called policy evaluation or control.

Estimating $V^*$ or $Q^*$ is called control, because these can be used for policy optimization.
There are four main Bellman equations:

\[ V_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \]

\[ V^*(s) = \max_a \mathbb{E}[r_{t+1} + \gamma V^*(S_{t+1}) | S_t = s, A_t = a] \]

\[ Q_\pi(s, a) = \mathbb{E}_\pi[r_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \]

\[ Q^*(s, a) = \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a] \]

\[ V^*(s) = \max_\pi V_\pi(s), \forall s \]
Dynamic Programming

- Dynamic Programming (DP) is a model-based approach
- Dynamic Programming (DP) is a model-based approach
- A method for solving complex problems, by breaking them down into sub-problems:
  - Solve the subproblems
Dynamic Programming

- Dynamic Programming (DP) is a model-based approach
- A method for solving complex problems, by breaking them down into sub-problems:
  - Solve the subproblems
  - Combine solutions to subproblems
Dynamic Programming

- Dynamic Programming (DP) is a model-based approach
- A method for solving complex problems, by breaking them down into sub-problems:
  - Solve the subproblems
  - Combine solutions to subproblems
- Iteratively evaluates value functions and improving policy following Bellman equations.
Dynamic Programming

- Dynamic Programming (DP) is a model-based approach
- A method for solving complex problems, by breaking them down into sub-problems:
  - Solve the subproblems
  - Combine solutions to subproblems
- Iteratively evaluates value functions and improving policy following Bellman equations.
- Assumes full knowledge of the MDP
Dynamic Programming

- Dynamic Programming (DP) is a model-based approach
- A method for solving complex problems, by breaking them down into sub-problems:
  - Solve the subproblems
  - Combine solutions to subproblems
- Iteratively evaluates value functions and improving policy following Bellman equations.
- Assumes full knowledge of the MDP
- Bellman equations give recursive decomposition
Iterative Policy Evaluation

\[ V_\pi(s) \doteq \mathbb{E}_\pi[R_t | S_t = s] \]
Iterative Policy Evaluation

\[
V_\pi(s) \triangleq \mathbb{E}_\pi [R_t | S_t = s] = \mathbb{E}_\pi [r_{t+1} + \gamma R_{t+1} | S_t = s]
\]
\[ V_\pi(s) \triangleq \mathbb{E}_\pi[R_t | S_t = s] \]
\[ = \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | S_t = s] \]
\[ = \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \]
Iterative Policy Evaluation

\[ V_\pi(s) = \mathbb{E}_\pi[R_t | S_t = s] \]
\[ = \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | S_t = s] \]
\[ = \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \]
\[ = \sum_a \pi(a | s) \sum_{s', r} p(s', r | s, a) [r + \gamma V_\pi(s')] \]
Iterative Policy Evaluation

\[ V_{k+1}(s) = E_{\pi}[r_{t+1} + \gamma V_k(S_{t+1}) | S_t = s] \]
Iterative Policy Evaluation

\[ V_{k+1}(s) = \mathbb{E}_{\pi}[r_{t+1} + \gamma V_k(S_{t+1})|S_t = s] \]

\[ = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma V_k(s')] \]
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 $V(\text{terminal}) = 0$

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 $\Delta < \theta$
Example: Small Gridworld

- Undiscounted episodic MDP ($\gamma = 1$)
- Non-terminal states $1, \ldots, 14$
- $r(s, a, s') = -1$ - reward is $-1$ until the terminal state is reached
- Agent follows uniform random policy
Example: Small Gridworld

- Undiscounted episodic MDP \((\gamma = 1)\)
- Non-terminal states \(1, \ldots, 14\)
- \(r(s, a, s') = -1\) - reward is \(-1\) until the terminal state is reached
- Agent follows uniform random policy

\[
\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25
\]
Example: Small Gridworld

$\mathcal{V}_k$: for the Random Policy

$k = 0$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Greedy Policy w.r.t. $\mathcal{V}_k$

random policy
Small Gridworld

$k = 0$

$\mathcal{U}_k$ for the Random Policy

$\begin{array}{cccc}
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
\end{array}$

Greedy Policy w.r.t. $\mathcal{U}_k$

$k = 1$

$\begin{array}{cccc}
0.0 & -1.0 & -1.0 & -1.0 \\
-1.0 & -1.0 & -1.0 & -1.0 \\
-1.0 & -1.0 & -1.0 & -1.0 \\
-1.0 & -1.0 & -1.0 & 0.0 \\
\end{array}$
Example: Small Gridworld

### $k = 1$

\[
\begin{array}{cccc}
0.0 & -1.0 & -1.0 & -1.0 \\
-1.0 & -1.0 & -1.0 & -1.0 \\
-1.0 & -1.0 & -1.0 & -1.0 \\
-1.0 & -1.0 & -1.0 & 0.0 \\
\end{array}
\]

### $k = 2$

\[
\begin{array}{cccc}
0.0 & -1.7 & -2.0 & -2.0 \\
-1.7 & -2.0 & -2.0 & -2.0 \\
-2.0 & -2.0 & -2.0 & -1.7 \\
-2.0 & -2.0 & -1.7 & 0.0 \\
\end{array}
\]

### $k = 3$

\[
\begin{array}{cccc}
0.0 & -2.4 & -2.9 & -3.0 \\
-2.4 & -2.9 & -3.0 & -2.9 \\
-2.9 & -3.0 & -2.9 & -2.4 \\
-3.0 & -2.9 & -2.4 & 0.0 \\
\end{array}
\]

Assume the policy is uniform

\[
V(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s', r} p(s', r|s, a) \left[ r + \gamma V(s') \right]
\]

$R_t = -1$ on all transitions
Table of Contents

1 Recap
2 Bellman Equation
3 Model Based
4 Iterative Policy Evaluation
5 Policy Improvement
New greedy policy $\pi'$

$$\pi'(s) \equiv \arg \max_a Q_\pi(s, a)$$
New greedy policy $\pi'$

$$\pi'(s) \doteq \arg\max_a Q_\pi(s, a)$$

$$= \arg\max_a \mathbb{E}[r_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a]$$
New greedy policy \( \pi' \)

\[
\pi'(s) = \arg \max_a Q_{\pi}(s, a) \\
= \arg \max_a \mathbb{E}[r_{t+1} + \gamma V_{\pi}(S_{t+1}) | S_t = s, A_t = a] \\
= \arg \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma V_{\pi}(s')] 
\]
Small Gridworld

$k = 0$

$\mathcal{U}_k$ for the Random Policy

$\begin{array}{cccc}
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 \\
\end{array}$

Greedy Policy w.r.t. $\mathcal{U}_k$

$k = 1$

$\begin{array}{cccc}
0.0 & -1.0 & -1.0 & -1.0 \\
-1.0 & -1.0 & -1.0 & -1.0 \\
-1.0 & -1.0 & -1.0 & -1.0 \\
-1.0 & -1.0 & -1.0 & 0.0 \\
\end{array}$

$k = 2$

$\begin{array}{cccc}
0.0 & -1.7 & 2.0 & 2.0 \\
-1.7 & -2.0 & -2.0 & -2.0 \\
-2.0 & -2.0 & -2.0 & -1.7 \\
-2.0 & -2.0 & -1.7 & 0.0 \\
\end{array}$
Small Gridworld

$k = 3$

$$
\begin{array}{cccc}
0.0 & -2.4 & -2.9 & -3.0 \\
-2.4 & -2.9 & -3.0 & -2.9 \\
-2.9 & -3.0 & -2.9 & -2.4 \\
-3.0 & -2.9 & -2.4 & 0.0 \\
\end{array}
$$

$k = 10$

$$
\begin{array}{cccc}
0.0 & -6.1 & -8.4 & -9.0 \\
-6.1 & -7.7 & -8.4 & -8.4 \\
-8.4 & -8.4 & -7.7 & -6.1 \\
-9.0 & -8.4 & -6.1 & 0.0 \\
\end{array}
$$

$k = \infty$

$$
\begin{array}{cccc}
0.0 & -14.0 & -20.0 & -22.0 \\
-14.0 & -18.0 & -20.0 & -20.0 \\
-20.0 & -20.0 & -18.0 & -14.0 \\
-22.0 & -20.0 & -14.0 & 0.0 \\
\end{array}
$$
How to improve a Policy

- Given a policy $\pi$
  - **Evaluate** the policy $\pi$
    \[
    V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | S_t = s]
    \]  
    (1)
  - **Improve** the policy by acting greedily with respect to $v_\pi$
    \[
    \pi' = \text{greedy}(V_\pi)
    \]  
    (2)

$\pi_0 \xrightarrow{E} V_{\pi_0}$
How to improve a Policy

- Given a policy $\pi$
  - Evaluate the policy $\pi$
    $$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | S_t = s]$$  \hspace{1cm} (1)
  - Improve the policy by acting greedily with respect to $v_\pi$
    $$\pi' = greedy(V_\pi)$$  \hspace{1cm} (2)

$$\pi_0 \xrightarrow{\mathbb{E}} V_{\pi_0} \xrightarrow{I} \pi_1$$
How to improve a Policy

- Given a policy $\pi$
  - Evaluate the policy $\pi$
    \[
    V_{\pi}(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | S_t = s]
    \]  
    (1)
  - Improve the policy by acting greedily with respect to $v_{\pi}$
    \[
    \pi' = \text{greedy}(V_{\pi})
    \]  
    (2)
How to improve a Policy

- **Given a policy** $\pi$
  - **Evaluate** the policy $\pi$
    \[
    V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | S_t = s]
    \]  
    \[\text{(1)}\]
  - **Improve** the policy by acting greedily with respect to $\nu_\pi$
    \[
    \pi' = \text{greedy}(V_\pi)
    \]
    \[\text{(2)}\]

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} V_{\pi_2} \xrightarrow{I} \ldots$$
How to improve a Policy

- Given a policy $\pi$
  - Evaluate the policy $\pi$
    \[
    V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | S_t = s] \tag{1}
    \]
  - Improve the policy by acting greedily with respect to $v_\pi$
    \[
    \pi' = \text{greedy}(V_\pi) \tag{2}
    \]
Policy Interaction Algorithm

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 $s \in S$

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

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

Policy evaluation
Estimate $v_{\pi}$
Iterative policy evaluation

Policy improvement
Generate $\pi' \geq \pi$
Greedy policy improvement