Tabular Solution Methods
Dynamic Programming

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Fall 2019

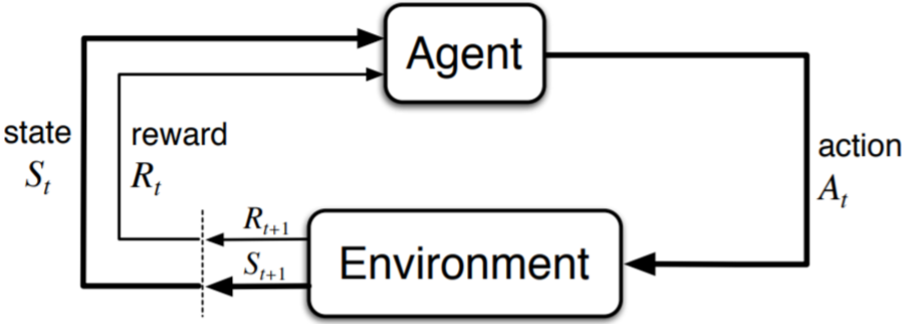
avereshc@buffalo.edu

September 5, 2019

Overview

Table of Contents

Recap: Finite Markov Decision Processes (MDP)



Reward

RL agents learn to **maximize cumulative future reward** (R).

Value Functions

There are two types of value functions:

Value Functions

There are two types of value functions:

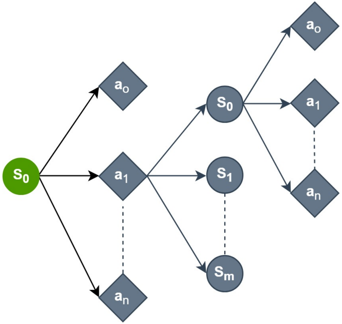
- state value function $V(s)$
- action value function $Q(s, a)$

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 π:

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

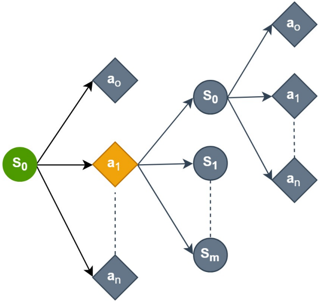


Action value function $Q(s, a)$

Definition

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

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



Policies π

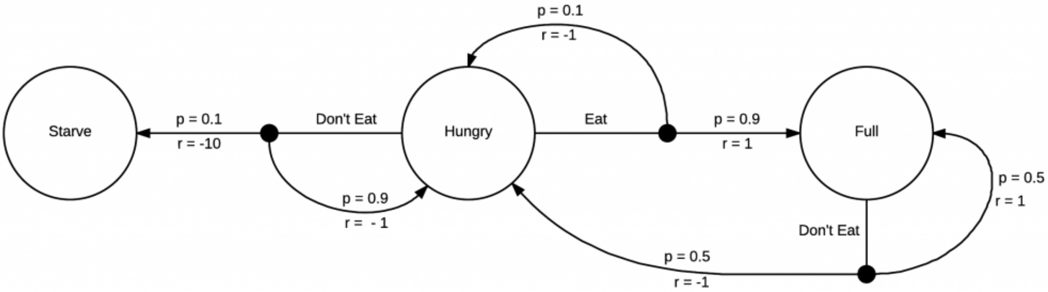
Definition

A *policy* π 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** π^***:** policy that maximizes the expectation of cumulative reward.

Example



Starve
Hungry
Full
p = 0.1
r = -10
Don't Eat
p = 0.9
r = - 1
p = 0.1
r = -1
Eat
p = 0.9
r = 1
p = 0.5
r = 1
Don't Eat
p = 0.5
r = -1

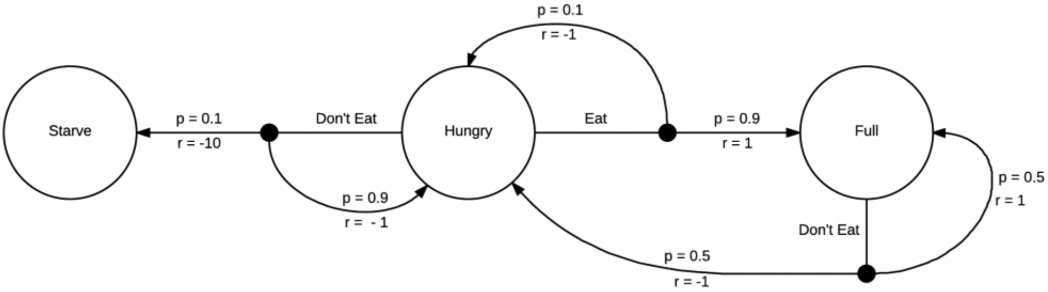
Example

$$P_\pi[A_t = \text{Eat}|S_t = \text{Hungry}] = 0.8$$

$$P_\pi[A_t = \text{Don't Eat}|S_t = \text{Hungry}] = 0.2$$

$$P_\pi[A_t = \text{Eat}|S_t = \text{Full}] = 0$$

$$P_\pi[A_t = \text{Don't Eat}|S_t = \text{Full}] = 1$$



RL Objective

Solving a reinforcement learning task means finding an optimal policy π^* that achieves maximum reward over the long run.
Optimal (greedy) policy π^* chooses an action (a), such that

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

RL Objective

Solving a reinforcement learning task means finding an optimal policy π^* that achieves maximum reward over the long run.

Optimal (greedy) policy π^* chooses an action (a), such that

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

Table of Contents

Model Based

Goal: compute optimal policies given a perfect model of the environment

Optimal Value Functions

$$
\begin{aligned}
V^*(s) &= \max_a \mathbb{E}[r_{t+1} + \gamma V^*(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r | s, a) \big[r + \gamma V^*(s') \big]
\end{aligned}
$$

Optimal Value Functions

$$
\begin{aligned}
V^*(s) &= \max_a \mathbb{E}[r_{t+1} + \gamma V^*(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r | s, a) \big[r + \gamma V^*(s') \big]
\end{aligned}
$$

$$
\begin{aligned}
Q^*(s) &= \mathbb{E}[r_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a] \\
&= \sum_{s',r} p(s', r | s, a) \big[r + \gamma \max_{a'} Q^*(s', a') \big]
\end{aligned}
$$

Dynamic Programming

- Dynamic Programming (DP) is a model-based approach

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

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

Table of Contents

Iterative Policy Evaluation

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

Iterative Policy Evaluation

$$\begin{aligned} V_\pi(s) &\doteq \mathbb{E}_\pi[R_t | S_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | S_t = s] \end{aligned}$$

Iterative Policy Evaluation

$$
\begin{aligned}
V_\pi(s) &\doteq \mathbb{E}_\pi[R_t | S_t = s] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | S_t = s] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s]
\end{aligned}
$$

Iterative Policy Evaluation

$$
\begin{aligned}
V_\pi(s) &\doteq \mathbb{E}_\pi[R_t | S_t = s] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | S_t = s] \\
&= \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) \big[r + \gamma V_\pi(s') \big]
\end{aligned}
$$

Iterative Policy Evaluation

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

Iterative Policy Evaluation

$$
\begin{aligned}
V_{k+1}(s) &\doteq \mathbb{E}_\pi[r_{t+1} + \gamma V_k(S_{t+1}) | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V_k(s')\big]
\end{aligned}
$$

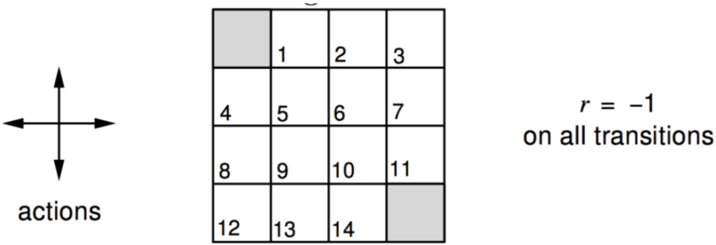
Iterative Policy Evaluation

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

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

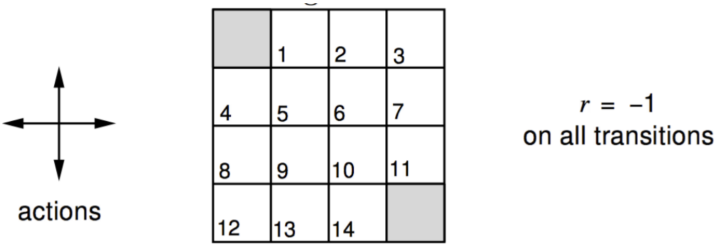
Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

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

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

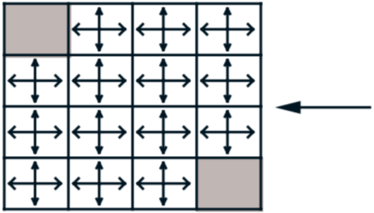
Example: Small Gridworld

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

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	0.0



random policy

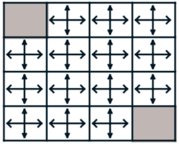
Small Gridworld

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

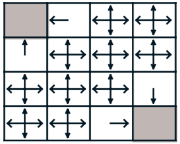
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

$k = 1$

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



Example: Small Gridworld

$k = 1$

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

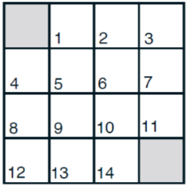
$k = 2$

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

$k = 3$

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

actions



$R_t = -1$
on all transitions

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

Table of Contents

Policy Improvement

New greedy policy π'

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

Policy Improvement

New greedy policy π'

$$\pi'(s) \doteq \arg\max_a Q_\pi(s,a)$$
$$= \arg\max_a \mathbb{E}\left[r_{t+1} + \gamma V_\pi(S_{t+1}) \middle| S_t = s, A_t = a\right]$$

Policy Improvement

New greedy policy π'

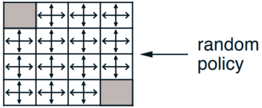
$$\begin{aligned}\pi'(s) &\doteq \arg\max_a Q_\pi(s,a)\\ &= \arg\max_a \mathbb{E}\left[r_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a\right]\\ &= \arg\max_a \sum_{s',r} p(s', r|s,a)\left[r + \gamma V_\pi(s')\right]\end{aligned}$$

Small Gridworld

v_k for the Random Policy — Greedy Policy w.r.t. v_k

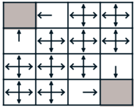
$k = 0$

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	0.0



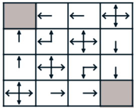
$k = 1$

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



$k = 2$

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



Small Gridworld

$k = 3$

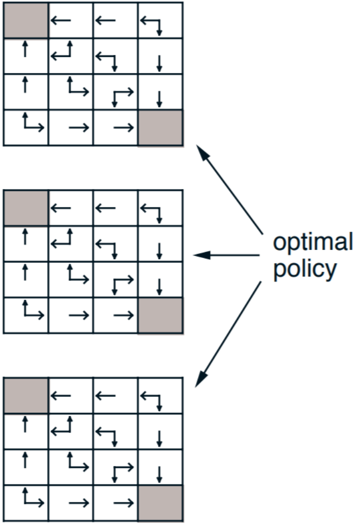
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



How to improve a Policy

- Given a policy π
  - **Evaluate** the policy π

$$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \dots | S_t = s] \tag{1}$$

  - **Improve** the policy by acting greedly with respect to v_π

$$\pi' = greedy(V_\pi) \tag{2}$$

$$\pi_0 \xrightarrow{E} V_{\pi_0}$$

How to improve a Policy

- Given a policy π
  - **Evaluate** the policy π

$$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | S_t = s] \tag{1}$$

  - **Improve** the policy by acting greedily with respect to v_π

$$\pi' = greedy(V_\pi) \tag{2}$$

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1$$

How to improve a Policy

- Given a policy π
  - **Evaluate** the policy π

$$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \dots | S_t = s] \tag{1}$$

  - **Improve** the policy by acting greedly with respect to v_π

$$\pi' = greedy(V_\pi) \tag{2}$$

$$\pi_0 \xrightarrow{\mathrm{E}} V_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} V_{\pi_1}$$

How to improve a Policy

- Given a policy π
  - **Evaluate** the policy π

$$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | S_t = s] \tag{1}$$

  - **Improve** the policy by acting greedly with respect to v_π

$$\pi' = greedy(V_\pi) \tag{2}$$

$$\pi_0 \xrightarrow{\text{E}} V_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} V_{\pi_2} \xrightarrow{\text{E}} \ldots$$

How to improve a Policy

- Given a policy π
  - **Evaluate** the policy π

$$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \ldots | S_t = s] \tag{1}$$

  - **Improve** the policy by acting greedly with respect to v_π

$$\pi' = greedy(V_\pi) \tag{2}$$

$$\pi_0 \xrightarrow{\text{E}} V_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} V_{\pi_2} \xrightarrow{\text{E}} \ldots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} V_*$$

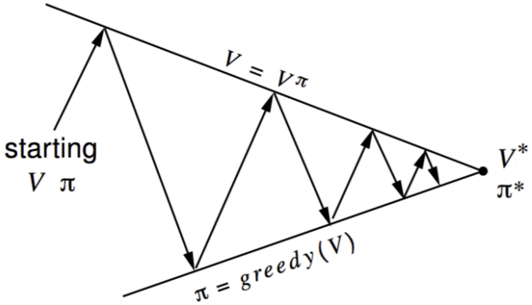
Policy Interaction Algorithm

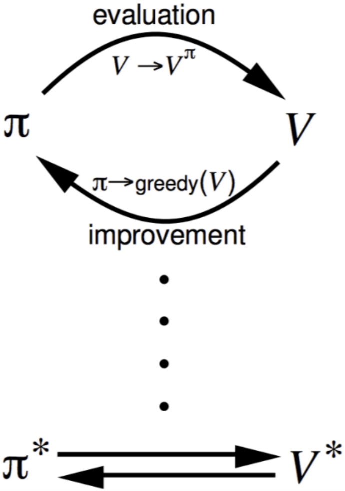
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

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

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

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
   $\quad$ 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 evaluation Estimate v_π

Iterative policy evaluation

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

Greedy policy improvement