Dynamic Programming - Value Iteration and Policy Iteration

Reinforcement learning – LM Artificial Iintelligence
(2022-23)

Alberto Castellini
University of Verona

Summary

- Introduction
- Policy Evaluation (Prediction)
- Policy Improvement
- Policy Iteration
- Value Iteration
- Asynchronous Dynamic Programming
- Generalized Policy Iteration

Categorization of RL algorithms (recap)

Value based

- Value function: **yes**
- Policy: **no (implicit)**

Policy based

- Value function: **no**
- Policy: **yes**

Actor-critic

- Value function: **yes**
- Policy: **yes**

Model based

- Dynamics model (i.e., transition and reward): **yes**

Model free

- Dynamics model (i.e., transition and reward): **no**

Introduction

Introduction

- **Dynamic Programming (DP)** in RL refers to a collection of **algorithms** used to **compute optimal policies**
- **Assumption**: complete knowledge of the dynamics model (**limit** to their **applicability**)
- Drawback: **high computational expense**
- DP methods are **important theoretically**
- DP methods are applied to **finite MDP**. A common way of obtaining approximate solutions for tasks with **continuous states and actions** is to **quantize** the state and action spaces
- **DP can be used to compute the value function using the Bellman equation in an iterative way to improve value approximations**

Policy Evaluation (Prediction)

(Iterative) Policy evaluation

- How to compute the value function v_* for an arbitrary policy π ?

- Let's recall the Bellman equation (system of |S| equations):

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s') \Big] \end{aligned}$$

- For our purpose **iterative solution methods** are most suitable

- We use the **Bellman equation as an update rule** (k is the iteration):

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_k(s') \Big] \end{aligned}$$

(Iterative) Policy evaluation

- Given an arbitrary initial approximation v_0 we obtain a sequence of approximate value functions $v_0, v_1, v_2, \ldots$ in which $v_k = v_\pi$ is a fixed point.

- The sequence can be shown to **converge** to v_π as $k \to \infty$ under the conditions that guarantee the existence of v_π ($\gamma < 1$ or termination)

Algorithm (in place update)

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

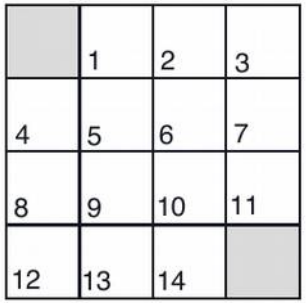
Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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

Bellman Equation

Complexity: O(|S|² * |A|)

Example: Gridworld



$R_t = -1$
on all transitions

- Nonterminal states: $\mathcal{S} = \{1, 2, \ldots, 14\}$
- Actions: $\mathcal{A} = \{\texttt{up}, \texttt{down}, \texttt{right}, \texttt{left}\}$
- Deterministic transition model
- Undiscounted episodic task

	v_k for the random policy	greedy policy w.r.t. v_k		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		$k = 3$	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	
$k = 1$	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		$k = 10$	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	
$k = 2$	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		$k = \infty$	0.0 -14. -20. -22. -14. -18. -20. -20. -20. -20. -18. -14. -22. -20. -14. 0.0	

Optimal policies in this case, but only guaranteed to be improvements over the random policy

Example: Gridworld

Homework: use the code of the policy evaluation algorithm developed in the next lab to evaluate the random policy on the gridworld problem of the **previous lecture (lecture 3)**

Policy Improvement

Policy improvement

- Given the value function $v_\pi(s)$ for a policy π we would like to know whether we can **get a better policy** **by choosing a different action** *a* in a specific state *s*

- **Idea**: consider selecting the **new action** *a* in *s* and then following the existing policy π. The **value** for the **state-action pair** in that case is:

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma v_\pi(s')\right]. \end{aligned}$$

- **Criterion: If $q_\pi(s,a) \geq v_\pi(s)$ than the action improves the policy.**

- In this case action *a* should always be **substituted** to action $\pi(s)$ in state *s*

Policy improvement theorem

Theorem: Let π and π' be any pair of deterministic policies such that **for all** $s \in S$

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

Then the policy π' must be as good as, or better than, π. That is,

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

In particular, if the first inequality is **strict at any state**, then there must be a strict inequality also in the second one in at least one state.

Proof (idea):

$v_\pi(s) \leq q_\pi(s, \pi'(s))$ →

$v_\pi(s) \leq q_\pi(s, \pi'(s))$ →

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s] \\
&= v_{\pi'}(s).
\end{aligned}$$

Policy Improvement

- **Policy improvement:** We extend the improvement to **all states and actions,** selecting at each state the action that appears best according to $q_\pi(s,a)$, i.e., new **greedy** policy π' given by

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

- The **greedy policy** takes the **action** that looks **best** after **one step of lookahead** according to v_π

- By construction this policy **meets** the conditions of the **policy improvement theorem**, hence it is as good as, or better than, the original policy

Policy Improvement

- **Optimality: If $v_\pi = v_{\pi'}$ than $v_\pi = v_*$ and both π and π' are optimal** and their value is

$$\begin{aligned} v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_{\pi'}(s') \Big]. \end{aligned}$$

- This theory can be extended to stochastic policies $\pi(a|s)$

Policy Iteration

Policy Iteration

- The **Policy Iteration algorithm** repeats **policy evaluation** and **policy improvement** obtaining a sequence of **monotonically improving policies** and **value functions**, until convergence to an **optimal policy and optimal value function**.

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

- The **convergence** in a **finite number of iterations** is ensured in finite MDPs by the finite number of policies available.

- **Notice:** each **policy evaluation** step is **started** with the value function for the **previous** policy (with a great increase in the speed of convergence)

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

   → Deterministic policy assumed

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
   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

Value Iteration

Value Iteration

- The **Value Iteration Algorithm** can be seen as a version of Policy Iteration in which the **policy evaluation step (generally iterative) is stopped after a single step**.

- The **update rule** that combines policy improvement and single-step policy evaluation is:

$$
\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_k(s') \Big],
\end{aligned}
$$

- The policy evaluation step of policy iteration can be truncated in several ways **without losing the convergence guarantees** of policy iteration. The strategy used by value iteration is an example.

- The **Gridworld example** seen before provides an idea about why policy evaluation can be truncated in advance. Iterations after the third have no effect on the greedy policy in that example.

Value Iteration

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

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

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

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

Complexity: O(|S|^2 * |A|)

Value Iteration

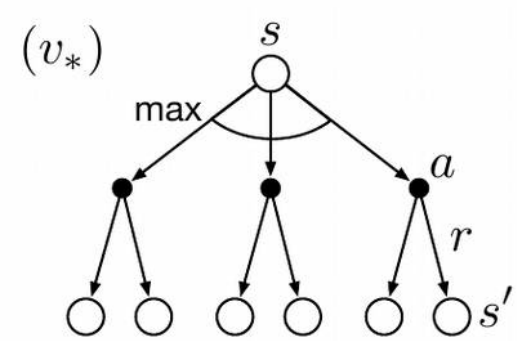
- Value iteration is obtained by turning the **Bellman optimality equation** in an **update rule**.

- **Notice:** Value iteration **does not explicitly represent the policy** and it **does not explicitly improves it**. The policy is implicitly extracted from the value function at the end of the algorithm (see last line).

- **Notice:** the **value iteration** update is **identical** to the **policy evaluation** update **exept** that it requires the **maximum** to be taken overall actions.

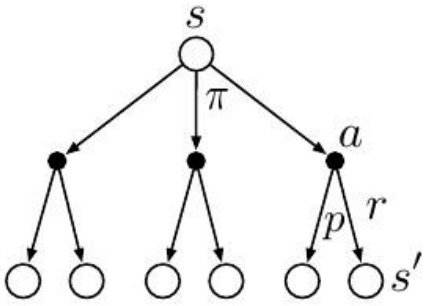
- Backup diagrams

Value iteration



(see Bellman optimality equation for v_*)

Policy evaluation



(see Bellman equation for v_π)

Asyncronous Dynamic Programming

Asynchronous Dynamic Programming

- The main idea of Asynchronous DP is to **order the policy/value-function updates** to let value information propagate from state to state in an efficient way.

- Some **ideas**:
  - Some states may not need their values updated as often as others
  - Skip updating some states entirely if they are not relevant to optimal behaviour (see Chapter 8 of SutBar)
  - Agent's experinece can be used to determine the states to which the DP algorithm applies its updates (e.g., update states as the agent visits them)
  - Focus the DP algorithm's update onto parts of the state set that are most relevant

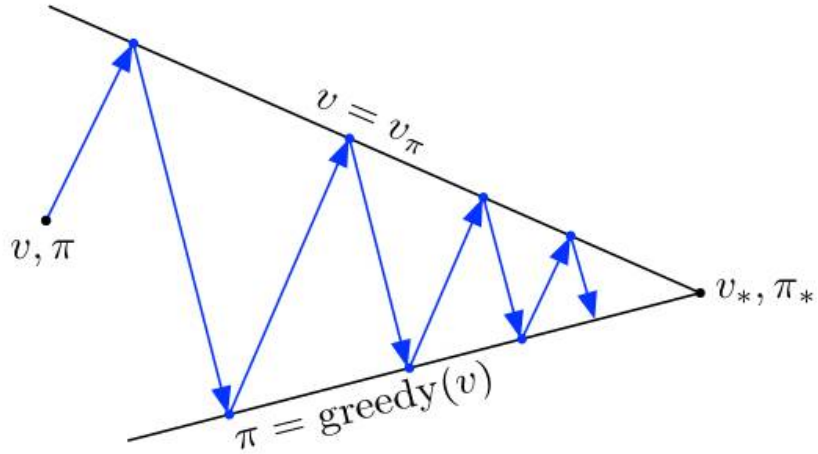
- This can **improve performance** in general and allow DP algorithms to be applied to **domains with large state space**

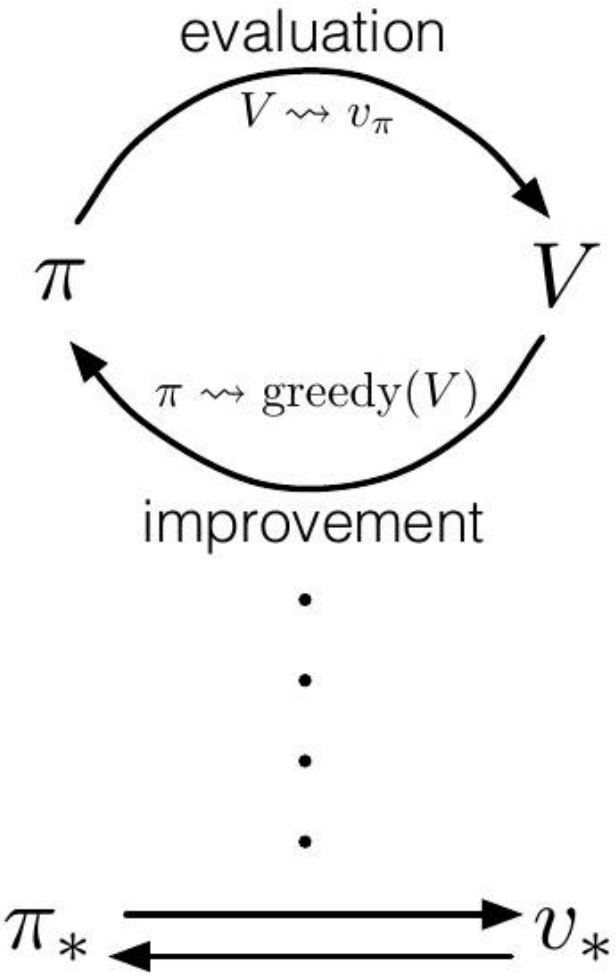
Generalized Policy Iteration

Generalized Policy Iteration

- **Policy iteration**: **two** interacting (competing/cooperating) **processes**:
  - Policy evaluation
  - Policy improvement

- This **schema**, called **Generalized Policy Iteration (GPI)**, is common to several RL algorithms, such as,
  - Value iteration
  - Asynchronous DP methods





Efficiency of Dynamic Programming

Efficiency of Dynamic Programming

- **DP** may not be practical for very large problems but it is **efficient**

- DP methods are **polynomial in the number of states and actions**
  - They take a number of computational operations less than some polynomial function of |S| and |A|
  - Although the total numeber of (deterministic) policies to search is $|A|^{|S|}$

- **Linear programming methods** become **impractical** at a much smaller number of states than do DP methods

- With today's computer, **DP methods can be used to solve MDPs with millions of states** → **Policy and Value Iterations are used in practice**

- It is not clear which of the two algorithms is better in general

- They are faster than their theoretical worst-case run times if started with good initial value function or policy

Important Observation about DP methods

- **DP methods update estimates** of the values of states **based** on **estimates** of the **values** of **successor states**, i.e., they update estimates on the basis of other estimates.

  → **DP methods perform bootstrapping**

- Lecture 5 (Monte Carlo Methods): methods that do **not** require a **model** and **do not bootstrap**

- Lecture 6 (Temporal-Difference Learning): methods that do **not** require a **model** and **do bootstrap**

References

- R. S. Sutton, A. G. Barto. Reinforcement learning, An Introduction. Second edition. Chapter 4