Planning by Dynamic Programming

Ali Gooya

Reference for slides:

David Silver (Deep Mind)

Learning outcomes

- Understand how Dynamic Programming works for planning in MDPs
- Optimize the policy and value functions of a given MDP using policy and value iterations
- Apply above to sample problems

What is Dynamic Programming (DP)?

- Dynamic refers to sequential or temporal component of the problem
- Programming refers to optimizing
- DP solves for more complex problems by
  - Breaking them down into smaller subproblems (principle of optimality)
  - Solving subproblems
  - Combining solutions to subproblems
- DP is a general method and has been invented by Richard E. Bellman (1950).

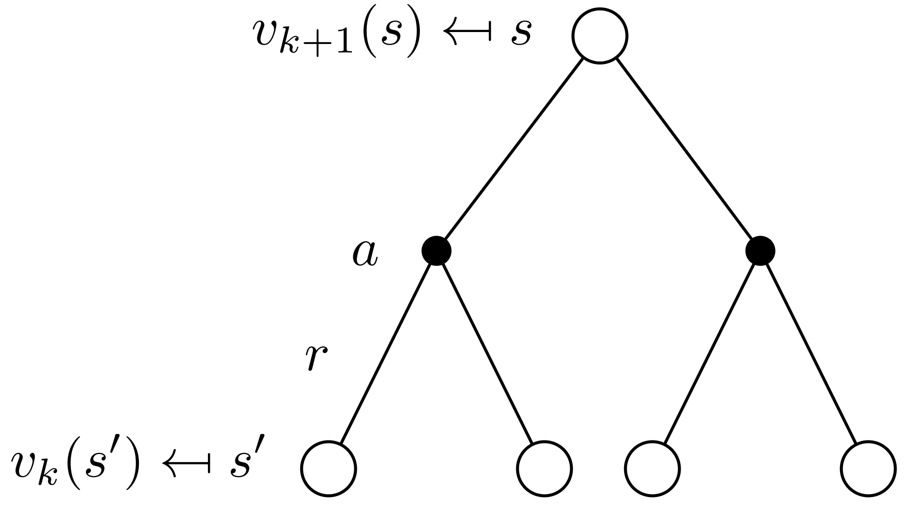
Dynamic Programming applied to MDP

- Dynamic programming requires full knowledge of the MDP (states, actions, transition probabilities, rewards, discount factor).
- It is a model based and solves the problem of *planning:*
  - Prediction:
    - Input - an MDP $< S, A, P, R, \gamma >$ and policy π
    - Output - value function v_π
  - Control:
    - Input - MDP $< S, A, P, R, \gamma >$
    - Output - Optimal value function v_* and optimal policy π_*

Iterative Policy Evaluation (i)

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$
- Using *synchronous* backups,
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$
  - where s' is a successor state of s
- This solves for medium sized MDPs, where one step solution to Bellman equation is not feasible.

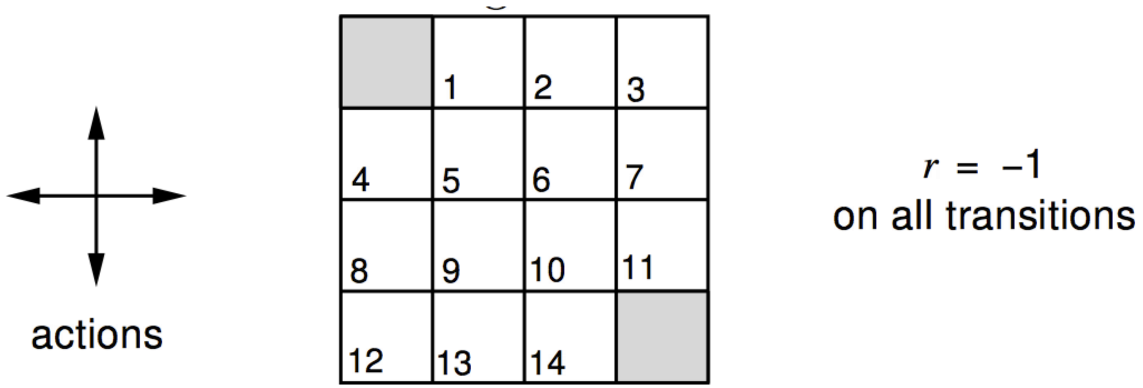
Iterative Policy Evaluation (ii)



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\boldsymbol{\pi}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{\pi}} \mathbf{v}^k$$

Evaluating random policy in a small grid world



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, \ldots, 14$
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

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

Iterative Policy Evaluation in Small Grid world (i)

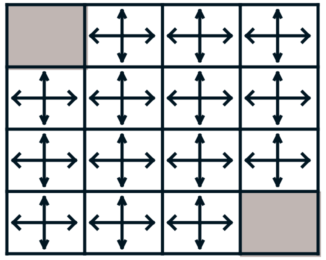
$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

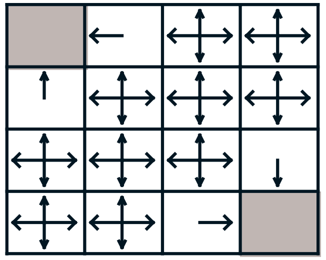
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

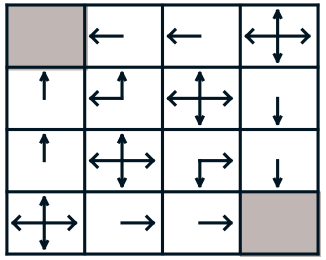
$k = 1$

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



$k = 2$

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



Iterative Policy Evaluation in Small Grid world (ii)

$k = 3$

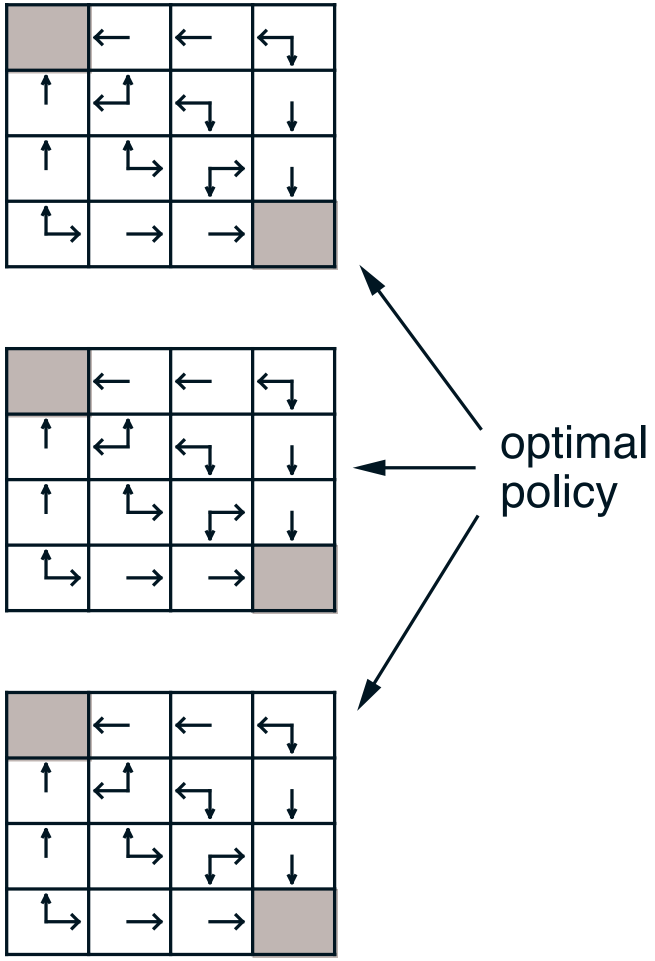
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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 policy

- Given a policy π
  - Evaluate the policy π

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ...|S_t = s\right]$$

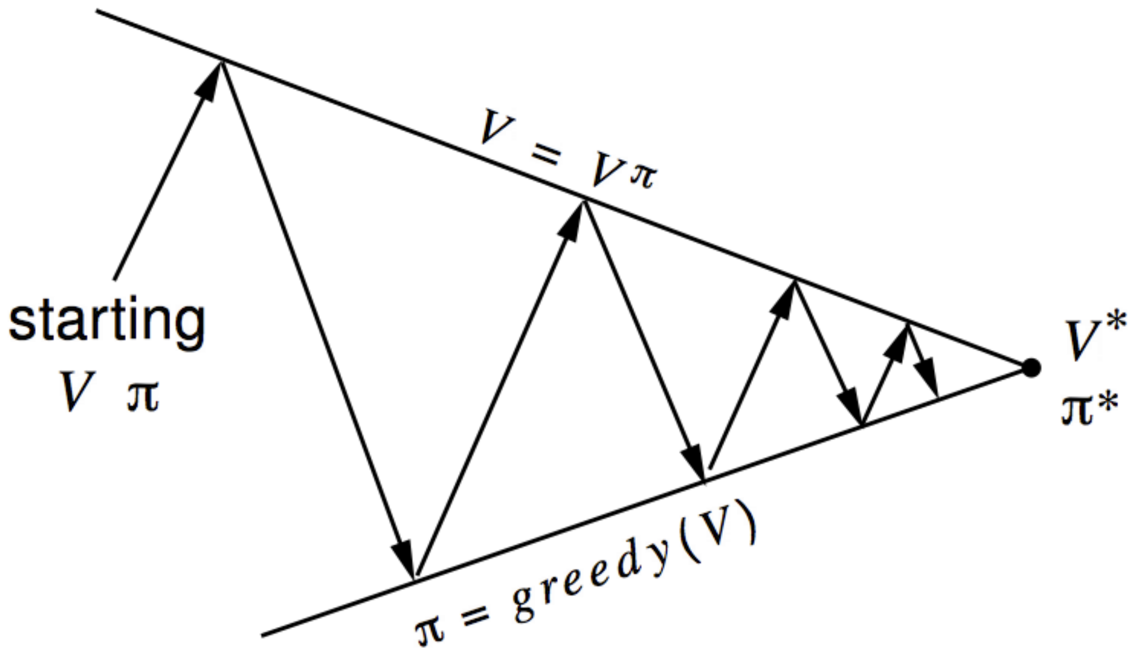
  - Improve the policy by acting greedily with respect to v_π

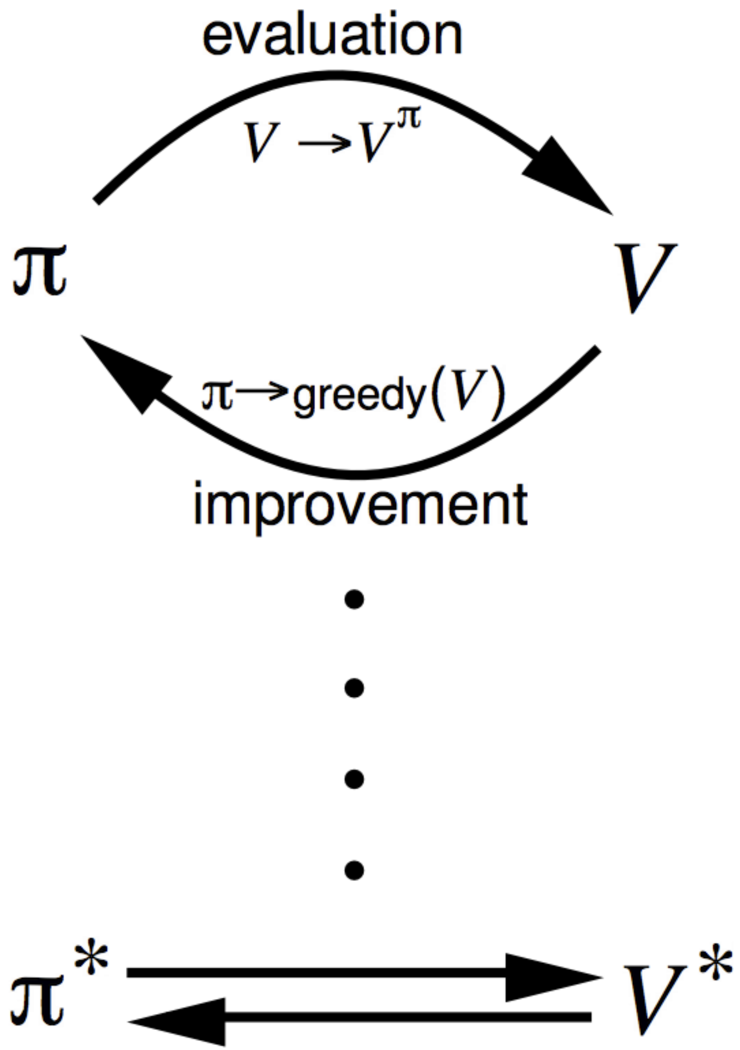
$$\pi' = \text{greedy}(v_\pi)$$

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

- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to $\pi*$

Policy Iteration





Policy evaluation Estimate v_π

Iterative policy evaluation

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

Greedy policy improvement

Why values improve?

- Consider a deterministic existing policy: $a = \pi(s)$
- With the new greedy policy

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ q_\pi(s, a)$$

- The value of any state s is improved by one step following π'

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

Why values improve?

- Hence:

$$\begin{aligned} v_\pi(s) &\le q_\pi(s, \pi'(s)) \\ &= R_s^{\pi'(s)} + \gamma \sum_{s'} P_{ss'}^{\pi'(s)} q_\pi(s', \pi(s')) \\ &\le R_s^{\pi'(s)} + \gamma \sum_{s'} P_{ss'}^{\pi'(s)} q_\pi(s', \pi'(s')) \end{aligned}$$

- So if we follow π' in the successor states we only make the right hand side larger until a goal achieved.

- At that time we have essentially followed π' from state s to the end which means: $v_\pi(s) \le v_{\pi'}(s)$

Policy improvement

- If improvements stop,

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

- Then the Bellman optimality equation has been satisfied

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

- Therefore $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$
- so π is an optimal policy

Modified Policy Iteration

- Does policy evaluation need to converge to v_π?
- Or should we introduce a stopping condition
  - e.g. ϵ-convergence of value function
- Or simply stop after k iterations of iterative policy evaluation?
- For example, in the small gridworld $k = 3$ 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)

Deterministic Value Iteration

- Intuition: start from final rewards (goal states) and work backwards
- The idea of value iteration is to apply iterative update Bellman optimality equations

$$v_*(s) = \max_a \{\mathcal{R}_s^a + \gamma \sum_{s'} P_{ss'}^a v_*(s')\}$$

- If we know the solution of $v_*(s')$ from previous iteration, we can update the values.

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_*$
- Using synchronous backups
  - At each iteration $k + 1$
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$
- Convergence to v_* will be proven later
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

Example: Shortest path

- The value at the goal (length of the path) is always zero, allowing us to update the values of other nodes
- Actions: n, s, w, e (no change of state in the edge)

$$v_*(s) = \max_a \{\mathcal{R}_s^a + \gamma \sum_{s'} P_{ss'}^a v_*(s')\}$$

- Can be rewritten as

$$v_{k+1}(s) \leftarrow -1 + \max_{s'} v_k(s')$$

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Conclusions

- Dynamic Programming performs planning in the form of prediction and control
- Prediction is equivalent to policy evaluation
- Control is equivalent to policy optimization
- Planning is not learning and assumes a known MDP (environment)
- This will be addressed in the next lecture.