Planning by Dynamic Programming:

Lecture 3
David Silver

1. introduction, 2 Policy Evaluation. 3- Policy iteration 4 value iteration
5- extensions to dynamic Programming 6. Contraction mapping
مبرهنة النقطة الثابتة.

Dynamic) sequential or temporal component to the problem.

Programming) optimizing a "program" i.e. a policy.

* cF linear programming
* A method for solving complex problems
* By breaking them down into subproblems.
  - Solve the subproblems
  - Combine solutions to subproblems

* Dynamic programming is a very general solution method for problems which have two properties
  * Optimal substructure
    - Principle of optimality applies.
    - optimal solution can be decomposed into subproblems
  * Overlapping subproblems
    - Subproblems recur many times
    - Solutions can be cached and reused

* Markov decision processes satisfy both properties.
  - Bellman equation gives recursive decomposition
  - value function stores and reuses solutions.

- Dynamic programming assumes full knowledge of the MDP.
- It is used for planning in an MDP
- For prediction
  Input MDP: $\langle S, A, P, R, \gamma \rangle$ and policy π

s.a.m

- or MRP $\langle S, P^{\pi}, R^{\pi}, \gamma \rangle$
- output value function V_{π}

- Or for control
  - input MDP $\langle S, A, P, R, \gamma \rangle$
  - output optimal value function V_*
  - and, optimal policy π_*

- Dynamic Programming is used to solve many other problems eg.
  - Scheduling algorithms.
  - String algorithms (sequence alligment)
  - Graph algorithms (shortest path)
  - Graphical models (Viterbi algorithm)
  - Bioinformatics

→ **Problem:** evaluate a given policy π

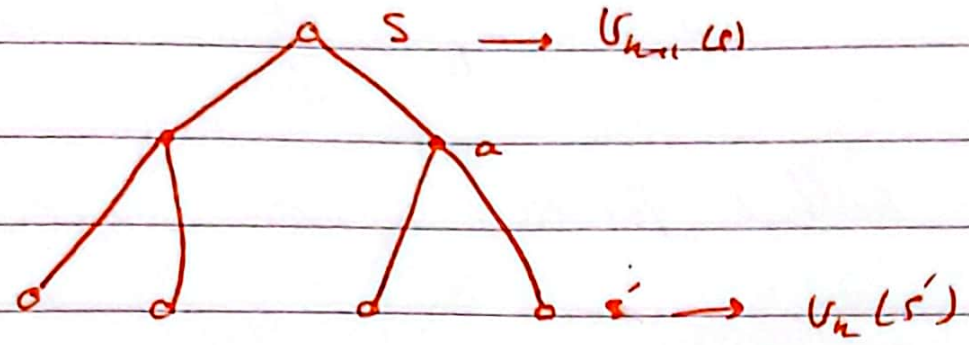
→ **Solution:** iterative application of Bellman expectation backup.

$V_1 \rightarrow V_2 \rightarrow \dots \rightarrow V_{\pi}$

↓

→ arbitrary initial value

- easy synchronous backups
  - At each iteration $k+1$
    - for all states $s \in S$
      - update $V_{k+1}(s)$ from $V_k(s')$
      - where s' is a successor state of s
- we will discuss asynchronous backups later
- Convergence to V_{π} will be proven at the end of the lecture.

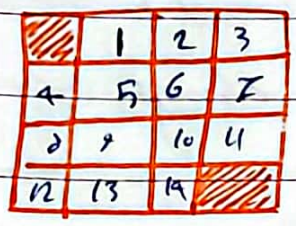


$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

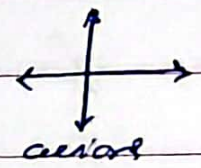
$$v_{k+1} = R^\pi + \gamma P^\pi v^k$$

there is a unique value function that satisfies this for any MDP.

Example: Small Grid world



$r = -1$ for all transitions



- undiscounted episodic MDP ($\gamma = 1$)

Nonterminal states $1, \ldots, 14$

One terminal state (shown twice as shaded squares)

Actions leading out of grid leave state unchanged

Reward is -1 until the terminal state is reached

Agent follows uniform random policy

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

طبق دستور کتاب ست.

از k=3 دیگه به optimal policy رسیدیم تا v تغییر کنه،

Policy iteration.

doing by a feedback process.

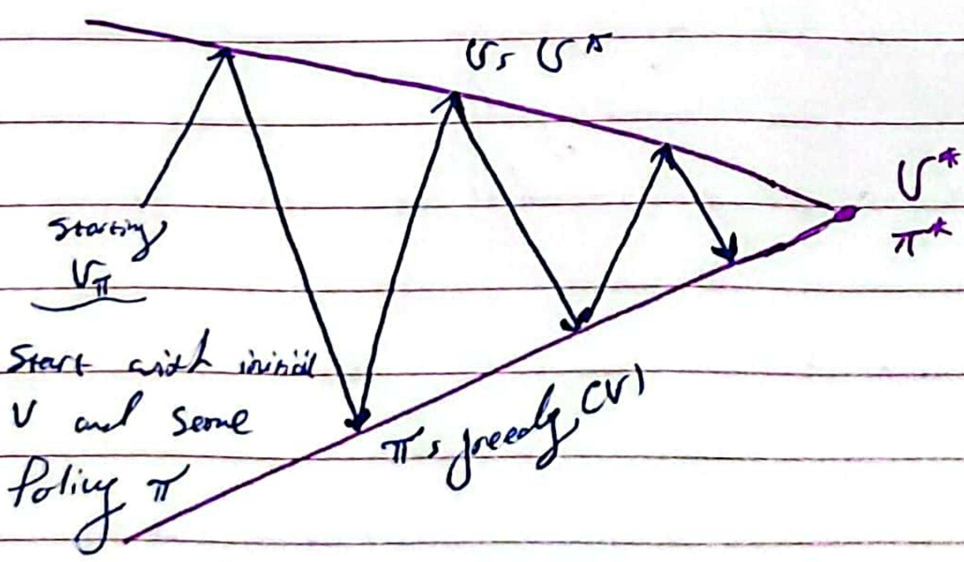
- Given a policy π
  - **Evaluate** the policy π

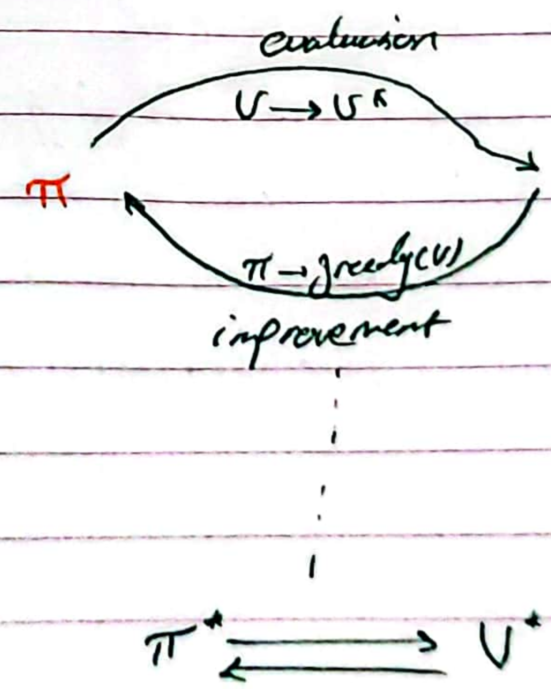
    $v_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s]$
  - **Improve** the policy by acting greedily with respect to v_π

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

In small Grid world improved policy was optimal, $\pi' = \pi^*$.
In general, need more iterations of improvement/evaluation.
But this process of policy iteration always converges to π^*.

★ Recall from last lecture: there is always at least one deterministic optimal policy for any MDP.





Policy evaluation Estimate v_π
Iterative policy evaluation.
Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement

Example:

- States: Two locations, maximum of 20 cars at each.
- Actions: Move up to 5 cars between locations overnight. (from location A to B)
- Reward: $10 for each car rented (must be available)
- Transitions: Transitions: Cars returns and requested randomly:
  - Poisson distribution, n returns/requests with prob $\frac{\lambda^n}{n!} e^{-\lambda}$
  - 1st location: average requests = 3, average returns = 3
  - 2nd " : " " = 4, " " = 2

The Contour Map of Car Rental

Important point:

How sensitive is that the convergence rate to π_0?

A: we'll see later at least for some of the algorithms the convergence rate is independent of π_0, but in practice that doesn't answer the question which is that you might just your π_0 your initial value may be way off and your first policy may be way off and so clearly if you start with the optimal policy you'll get to the optimal policy faster, so it does matter but the convergence rate is independent of the initial value and policy.

a little bit Formally,

- Consider a deterministic policy, $a = \pi(s)$
- we can improve the policy by acting greedily

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

acting greedily means that we basically we look at the value of being in a state and taking a particular action and then following your policy after that.

- This improves the value from any state s over one step.

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

→ if we act greedily that the greedy policy at least improves the value that we're going to get over one step and this immediate one step, it's not worry yet about what happens after that let's just consider one step and see whether we get more value over one step.

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = E_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &\leq E_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\ &\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s] \\ &\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \cdots + \cdots \mid S_t = s] = v_{\pi'}(s) \end{aligned}$$

if improvements stop,

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

Then the Bellman optimality equation has been satisfied

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

Therefore $v_\pi(s) = v_*(s)$ for all $s \in S$

So π is an optimal policy.

Q: Does policy evaluation need to converge to v_π?
Or should we introduce a stopping condition $\overline{v_\pi}$
* eg 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)

Value Iteration

Any optimal policy can be subdivided into two components:
- An optimal first action A_*
- Followed by an optimal policy from successor state s'

Theorem (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if and only if
- For any state s' reachable from s
- π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$

- if we know the solution to subproblems $v_*(s')$
- Then solution $v_*(s)$ can be found by one step lookahead

$$v_*(s) \leftarrow \max_{a \in A} R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

- The idea of value iteration is to apply these updates iteratively, again and again and again
- Intuition: Start with final rewards and work backwards
- Still works with fuzzy, stochastic MDPs

- Problem: find optimal policy
- Solution: iterative application of Bellman optimality backup.
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- Using synchronous backups
  - At each iteration $k+1$

s.a.m

- For all States $s \in S$
- 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

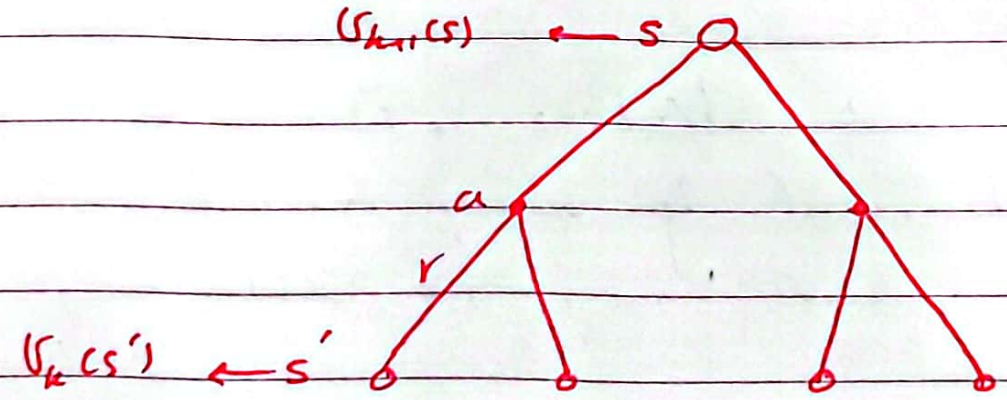
Difference between value iteration and policy iteration

One thing which is apparent is that we're not building an explicit policy at every step we're just working directly in value space.

We kind of used our value function to build a policy, used our policy to build the next value function.

Value iteration goes directly from value function to value function to value function to value function.

↳ may not correspond to particular V_π



$$V_{k+1}(s) = \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s') \right)$$

$$V_{k+1} = \max_{a \in A} \left(R^a + \gamma P^a V_k \right)$$

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	" " " + Greedy policy improvement	policy iteration
Control	Bellman Optimality Equation	value iteration

- ★ Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- ★ Complexity $O(mn^2)$ per iteration, for m actions and n states
- ★ Could also apply to action-value function $q_\pi(s,a)$ or $q_*(s,a)$
- ★ Complexity $O(mn^2n^2)$ per iteration
  high complexity

Extensions:

- DP methods described so far used synchronous backups
- i.e. all states are backed up in parallel
- Asynchronous DP backs up states individually, in any order
- For each selected state apply the appropriate backup
- Can significantly reduce computation
- Guaranteed to converge if all states continue to be selected

Three simple ideas for asynchronous dynamic programming:

- In-place dynamic programming
- Prioritised sweeping
- Real-time dynamic programming

Inplace Dynamic Programming

- Synchronous value iteration stores two copies of value function

for all s in S

$$V_{new}(s) \leftarrow \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_{old}(s') \right)$$

- Inplace value iteration only stores a copy of value function

for all s in S

$$V(s) \leftarrow \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V(s') \right)$$

Prioritised Sweeping

→ how important it is to update any state in your model, which state should you update first.

- Use magnitude of Bellman error to guide state selection, eg

$$\left| \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V(s') \right) - V(s) \right|$$

- Backup the state with the largest remaining Bellman error
- Update Bellman error of affected states after each backup
- Requires knowledge of reverse dynamics (predecessor states)
- Can be implemented efficiently by maintaining a priority queue.

Real-Time Dynamic Programming

→ select the states that the agent actually visits.

Idea: only states that are relevant to agent

Use agent's experience to guide the selection of states.

After each time-step S_t, A_t, R_{t+1} ~~~ collect real samples.

Backup the state S_t

$$V(S_t) \leftarrow \max_{a \in A} \left(R_{S_t}^a + \gamma \sum_{s' \in S} P_{S_t s'}^a V(s') \right)$$

* DP uses full width backups

✓ For each backup (sync or async)

- Every successor state and action is considered
- using knowledge of the MDP transitions and reward function

✓ DP is effective for medium-sized problems

✓ (millions of states)

✗ For large problems DP suffer Bellman's curse of dimensionality

- Number of states $n = |S|$ grows exponentially with number of state variables

✗ Even one backup can be too expensive

s.a.m