Lecture 7 - Dynamic Programming (DP)

Instructor: GuangBing Yang, PhD

Introduction to DP

- What is Dynamic Programming (DP)?
  - Dynamic stand for sequential or temporal component to the problem.
  - Programming means optimizing a "program", i.e., a policy
- DP is a collection of algorithms for solving the problems of optimal policies in MDPs and RL.
- It is a set of algorithms for solving complex problems
- It splits the complex problems into subproblems, then solving the subproblems and combine solutions of subproblems.

Requirements for DP

- To apply a DP solution, a problem shall have two properties:
  - Optimal substructure
    - Principle of optimality applies
    - Optimal solution can be decomposed into subproblems
  - Overlapping subproblems
    - Subproblems repeat many times
    - solutions can be cached and reused
- Make decision processes satisfy both properties
  - Bellman equation gives recursive decomposition (subproblems)
  - value function stores and reuses solutions

DP Applications

- DP assumes full knowledge of the MDP, and the environment be a finite MDP, which means S, A, and R are finite.
- Its dynamics are defined as a set of probabilities p(s', r | s, a).
- For prediction:
  - input: MDP $<\mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma>$ and policy π, or: MRP $<\mathbb{S}, \mathbb{P}^{\pi}, \mathbb{R}^{\pi}, \gamma>$
  - output: value function v_{π}
- For control:
  - input: MDP $<\mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma>$
  - output: optimal value function v_* and optimal policy π_*

Other Applications of DP

- Dynamic programming is used for many other problems, e.g.,
  - Scheduling algorithms
  - String algorithms (e.g., sequence alignment, edit distance, LCS)
  - Graph algorithms (e.g., shorts path algorithms, such as Dijkstra's algorithm)
  - Graphical models (e.g., Viterbi algorithm,)
  - Bioinformatics (e.g., lattice models)

Iterative Policy Evaluation

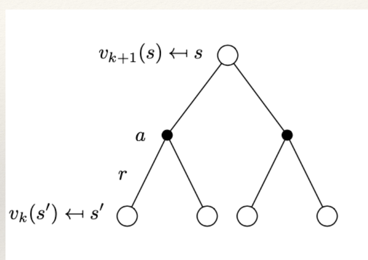
- Problem: how to compute the state-value function v_π for an arbitrary policy π.
- Approach: iterative application of Bellman optimality equation
- $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$
- Synchronous evaluation,
  - for all states $s \in \mathbb{S}$, at each iteration k+1
    - update $v_{k+1}(s)$ from $v_k(s')$, where s' is a successor state of s
- Asynchronous approach follows

Iterative Policy Evaluation (2)

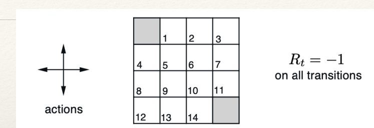
- Backup diagram:

$$v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \,|\, S_t = s]$$

- $$= \sum_{a \in A} \pi(a \,|\, s) \sum_{s',r} p(s', r \,|\, s, a)[r + \gamma v_k(s')]$$
  - where $\pi(a \,|\, s)$ is the probability of taking action a in state s under policy π.



Example - a random policy in a small grid

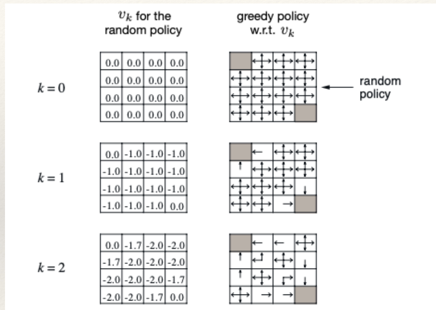


A 4 x 4 gridworld

- Follow uniform random policy: four actions: [north, east, south, west], $\pi(n \,|\, .) = \pi(e \,|\, .) = \pi(s \,|\, .) = \pi(w \,|\, .) = 0.25$
- Un-discounted episodic MDP, $\gamma = 1$
- Non-terminal states 1, …, 14
- one terminal state (show twice)
- reward is -1 for all actions until the terminal state is reached, +1
- actions leading out of the terminal leave state unchanged

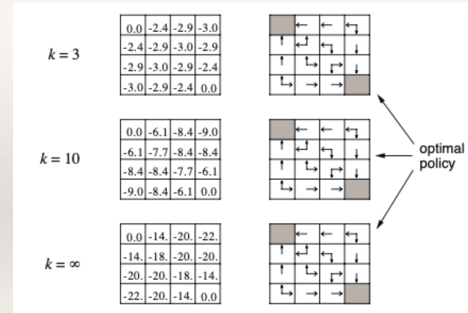
Example - a random policy in a small grid

- Convergence of iterative policy evaluation on a small grid world.
- The left column is the sequence of approximations of the state-value function for the random policy (all actions equally likely).
- The right column is the sequence of greedy policies corresponding to the value function estimates.
- Arrows are shown for all actions achieving the maximum, and the numbers shown are rounded to two significant digits.



Example - a random policy in a small grid

- Convergence of iterative policy evaluation on a small grid world.
- The last policy is guaranteed only to be an improvement over the random policy, but in this case it, and all policies after the third iteration, are optimal.

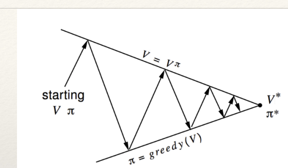


Improve a Policy

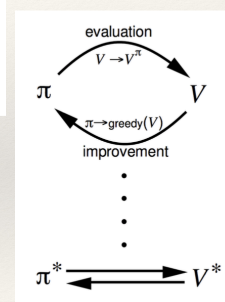
- Given a policy π
  - Evaluate the policy π using $v_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s]$
  - Improve the policy by acting greedily w.r.t. $v_\pi : \pi' = \text{greedy}(v_\pi)$
- In small grid world improved policy was optimal $\pi' = \pi^*$
- This process of policy iteration always converges to π^*

Policy Iteration

- Policy evaluation — estimate v_π by iterative policy evaluation
- Policy improvement — update $\pi' \geq \pi$ by greedy policy improvement



Interaction between evaluation and improvement



Generalized policy iteration

Policy Improvement

- Goal: for a better policy
- $$q_\pi(s,a) = E[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ = \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]$$
  - The key criterion is:
    - when $q_\pi(s,a) \geq v_\pi(s)$, the new policy is better than the old one,
    - otherwise, keep the old policy.

Policy Improvement

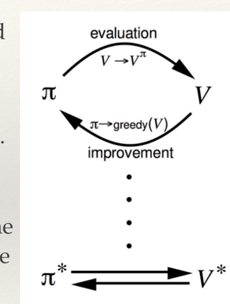
- This is policy improvement theorem.
- It can be expressed as: , $q_\pi(s, \pi'(s)) \geq v_\pi(s), \forall s \in S$
- the policy π' must be as good as, or better than, π, that is $v_{\pi'}(s) \geq v_\pi(s), \forall s \in S$
- Consider a deterministic policy, $a = \pi(s)$, improve the policy greedily, $\pi'(s) = \text{argmax}_{a \in A} q_\pi(s,a)$
- This improves the value from any state s over one step: $q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s,a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$

Policy Improvement

- This therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s), \forall s \in S$
  $$v_\pi(s) \leq q_\pi(s, \pi'(s)) = E_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ \leq E_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\ \leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s] \\ \leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s], \forall s \in S$$
- When converged, we have, $q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s,a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$
- This satisfies the Bellman optimality equation: $v_\pi(s) = \max_{a \in A} q_\pi(s,a)$, and
- $v_\pi(s) = v_*(s), \forall s \in S$, so π is an optimal policy

Generalized Policy Iteration (GPI)

- GPI refers to the general idea of letting policy-evaluation and policy-improvement processes interact and interleave repeatedly until convergence.
- Almost all reinforcement learning methods are a kind of GPI.
- All have identifiable policies and value functions.
- The policy always improve w.r.t. to the value function and the value function always drive toward the value function for the policy, as suggested by the diagram to the right.



Generalized policy iteration

Principle of Optimality

- Any optimal policy can be subdivided into two components:
  - An optimal first action A_*
  - Followed by an optimal policy from successor state s'
- The theorem is:
  - A policy $\pi(a \mid s)$ achieves the optimal values from state s, $v_\pi(s) = v_*(s), \forall s \in S$, that means,
    - for any state s' in S, π achieves the optimal value from state s'

Deterministic Value Iteration

- In deterministic case, if the solution to the subproblems $v_*(s')$ is known, then
- the solution $v_*(s)$ can be found by one-step lookahead approach:
  - $v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1} \mid S_t = s, A_t = a]$, or
  - $v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')], \forall s \in S$
- The intuition behind the value iteration is the Bellman optimality equation.
- Value iteration is obtained simply by turning the Bellman optimality equation into an update rule.
- The same approach works for stochastic MDPs.

Value Iteration Algorithm

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

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma V(s')\right]$

Value iteration effectively combines policy evaluation and policy improvement iterativly. Faster convergence is achieved by interposing multiple policy evaluation between each policy improvement process.

Asynchronous Dynamic Programming

- DP methods described so far used synchronous backups
- For example, all states are backed up in parallel
- A drawback of DP is the operations over entire set of MDPs.
- Asynchronous DP algorithms are in-place iterative DP that are not organized in terms of systematic process of the state set.
- It backs up states individually in any order.
- To converge correctly, an asynchronous algorithm must continue to update the values of all the states.
- In this case, it can't ignore any state after some point in the computation.
- Asynchronous DP algorithms allow great flexibility in selecting states to update.

Asynchronous Dynamic Programming

- Three simple ideas for asynchronous dynamic programming:
  - In-place dynamic programming
  - Prioritized sweeping
  - Real-time dynamic programming

In-place dynamic programming

- Synchronous value iteration stores two copies of value function:
  - $v_{\text{new}}(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{\text{old}}(s')], \forall s \in S,$
  - then, $v_{\text{old}} \leftarrow v_{\text{new}}$
- In-place value iteration only stores one copy of value function
  - $v(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v(s')], \forall s \in S$

Prioritized Sweeping

- Use magnitude of Bellman error to guide state selection, e.g.
  - $\left| \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v(s')] - v(s) \right|$
- 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

Recap

- Knowing the basic ideas and algorithms of dynamic programming as they relate to solving finite MDPs.
- *Policy evaluation* is about the iterative computation of the value functions for a given policy.
- *Policy improvement* is about the computation of an improved policy given the value function for that policy.
- Policy iteration and value iteration put these two computations together. They are used to compute the optimal policy in MDPs.

Recap

- Classical DP methods operate in sweeps through the state set, performing an expected update operation on each state.
- Each such operation updates the value of one state based on the values of all possible successor states and probabilities of occurring.
- Optimal policy is found when the DP operation converged, corresponding the Bellman optimality equations for value functions: $v_\pi(s), q_\pi(s), v_*(s), q_*(s)$
- Generalized policy iteration (GPI) is the general idea of two interacting processes revolving around an approximate policy and an approximate value function.

Recap

- One process takes the policy as given and performs some form of policy evaluation, changing the value function to be more like the true value function for the policy.
- The other process takes the value function as given and performs some form of policy improvement, changing the policy to make it better, assuming that the value function is its value function.
- Although each process changes the basis for the other, overall they work together to find a joint solution: a policy and value function that are unchanged by either process and, consequently, are optimal.
- Asynchronous DP methods are in-place iterative methods that update states in an arbitrary order, perhaps stochastically determined and using out-of-date information.
- DP methods update estimates of the values of states based on estimates of the values of successor states. This process is called bootstrapping.

Questions and Lab