CMPUT 365: Dynamic Programming

Rupam Mahmood

Jan 31, 2022

Admin: Last module of Coursera course 1

Due dates C1M4:

- Practice quiz: Tues Feb 1
- Graded quiz: Sat Feb 5

Midterm:

- Based on worksheet questions, book reading, and lectures
- Start working on worksheet questions now
- Discuss over slack
- Meet TAs during their office hours

Chapter 3 vs. Chapter 4

- In Chapter 3, we didn’t learn anything about the mechanisms the agent can use
- Value functions are essential concepts when talking about an MDP
- And Bellman equations are mathematical facts, not algorithms
- It was not mentioned in Chapter 3 how agent functions accept that it maintains a policy
- Starting from Chapter 4, we start to talk about mechanisms the agent can use

Who knows what?

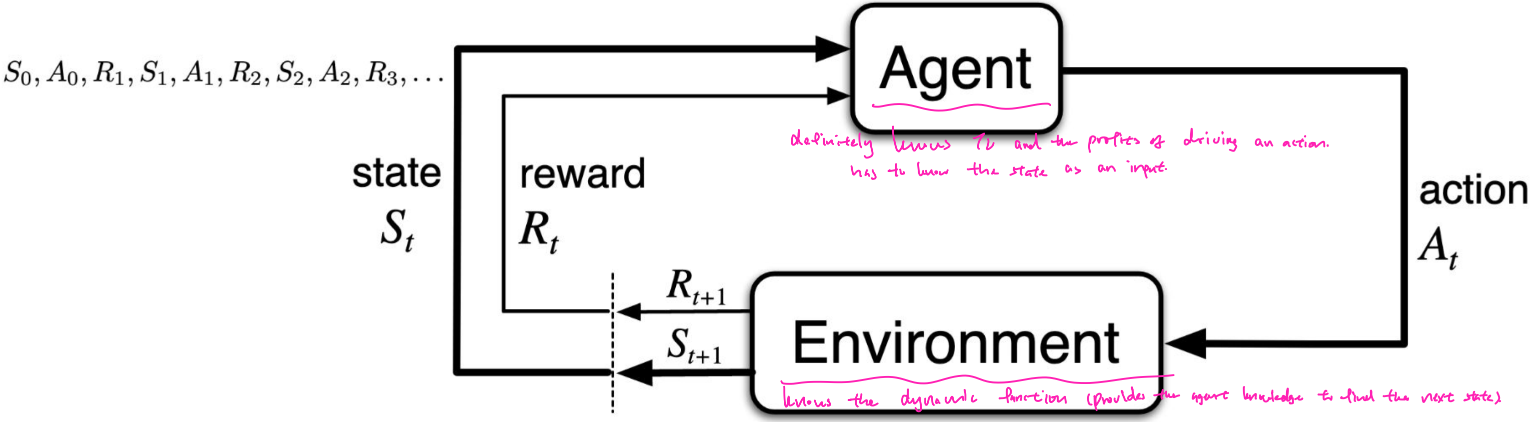
Questions to ask:

- Who definitely knows these
- Who may have been given the knowledge ⇒ problem dependent.
- Who definitely doesn't know these or do not need to know these

Learning and other improvement mechanisms

Estimates like *V* or *Q*

$\pi(a|s)$ finding an optimal policy. (the goal).



definitely knows π and the profits of driving an action.
has to know the state as an input.

knows the dynamic function (provides the agent knowledge to find the next state)

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

⇒ know how the world works = how the state change.
(doesn't mean we know the best way).

not depend on the policy

$r(s, a)$ $\quad v_\pi(s)$ $\quad q_\pi(s, a)$ $\quad \pi_*$

expected reward

value function
(agent doesn't know).
environment needs to know.

optimal policy.

Dynamic Programming

- Dynamic programming (DP) is $\boxed{\text{a way of knowing values and optimal policies}}$ when the model of the environment's dynamics is given
- If in a problem the model is given to the agent, it can use DP
- Often the agent is only given an estimate of the model if at all or…
- The agent estimates the model through experience (model learning)
- Or a designer tests their agent by comparing agent's performance against true values / optimal policies in a toy MDP
- Generally, mechanisms that use a given model as opposed to experience to improve performance are known as planning methods

dynamic programming. (use model to improve policy).

Coursera Video review: Policy Evaluation vs. Control

- Control is the task of improving a policy

- By prediction, then, we usually mean tasks where the policy doesn't improve. policy is fixed and the agent has to estimate the consequence of that policy, for example, in terms of a value estimate

- We can turn Bellman equations into update rules

- These update rules will require the knowledge of the environment's dynamics

Iterative Policy Evaluation ⇒ convert Bellman equation into Bellman update.

Bellman equation

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

mathematical equation

the value function.

$$v_{k+1}(s) \doteq \sum_a \pi(a\,|\,s) \sum_{s',r} p(s',r\,|\,s,a)[r + \gamma v_k(s')]$$

definition.

defing an update rule.

Bellman update

用于estimate v_π.

for all $s \in \mathcal{S}$

the estimate of the value function of the k^{th} iteration.

implementation of

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 $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $\underbrace{V(s)}_{V_{k+1}(s)} \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r\,|\,s,a)\big[r + \gamma \underbrace{V(s')}_{V_k(s')}\big]$ ⇒ keep updating.
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

- These are also called expected updates
- Updates are done in a sweep through the state space ⇒ 将所有state全部过一遍.

Coursera Video review: Policy Improvement

the action chosen by π'

choose all action following π'

If for a policy π' we have $q_\pi(s, \pi'(s)) \geq v_\pi(s) \forall s$, **then we also have** $v_{\pi'}(s) \geq v_\pi(s), \forall s$

action value

state value 遵循 policy π.

(only take the first action follow π', 其他都遵循 policy π).

A greedy policy w.r.t. q_π is such a policy

$$\max_a q_\pi(s, a) \geq v_\pi(s) \forall s$$

Greedy policy: ⇒ how do you chose π'?

$$\begin{aligned} \pi'(s) &\doteq \underset{a}{\arg\max}\, q_\pi(s, a) \\ &= \underset{a}{\arg\max}\, \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \underset{a}{\arg\max} \sum_{s', r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S} \end{aligned}$$

⇒ make a new policy π' always take the max value of action value q_π

the greedy policy with respect to policy π.

- If a policy π is not optimal already, then the greedy policy π' w.r.t the action value q_π will improve the value at least in one state
- There π' is strictly better than π if π isn't optimal already

Policy iteration

evaluation step, 求出 V_{π_0}

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

improvement step

converge to the optimal policy.

⇒ 每次都 strictly improve the policy.
∴ 保证可以 converge to the optimal policy.

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 $s \in \mathcal{S}$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s',r \mid s,\pi(s))\big[r + \gamma V(s')\big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

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

Vova Selin对所有人说 下午1:51

V Is it possible for this greedy policy to overlook a better policy from greed? For example pi' might be better for S1, though moves away from a greater reward in S2

Oh I see, so it doesn't matter in the end once we convert to the optimal policy

Mohamed Ahmed对所有人说 下午1:12

MA what's the difference between greedifying w.r.t. value function and greedifying w.r.t. action value function?

They are the same thing.

Vova Selin对所有人说 下午1:13

V Nothing should happen right, we'll greedify the policy and it'll remain the same ⇒ Yes!!

Policy iteration

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

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 $s \in \mathcal{S}$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[r + \gamma V(s')\big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

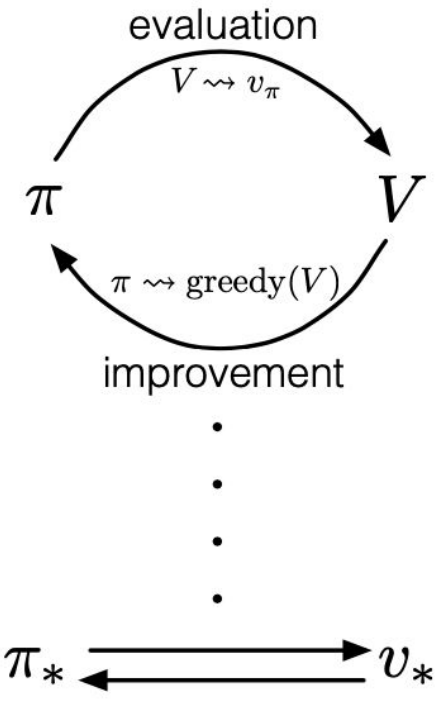
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

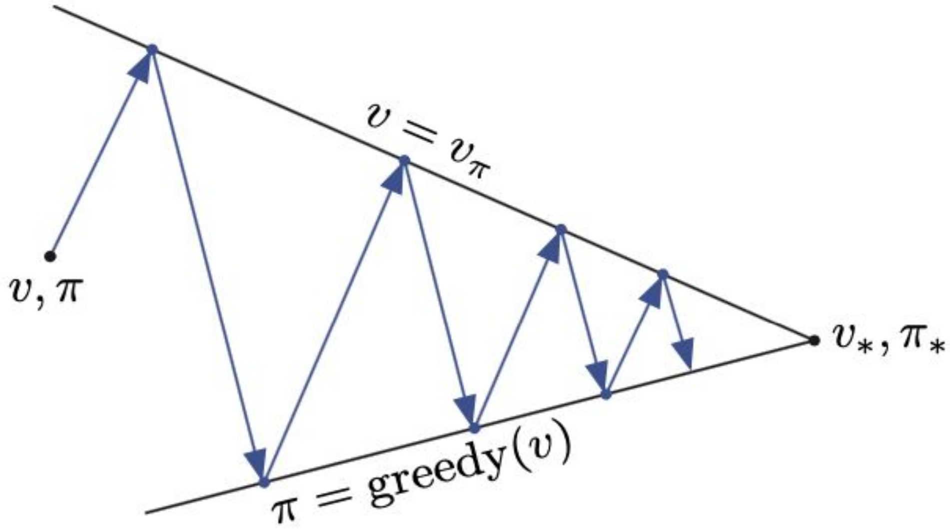
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   - *old-action* $\leftarrow \pi(s)$
   - $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
   - If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*

   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

If the program halts, what happens in the last two iterations?

Generalized Policy Iteration





Value Iteration

$$v_* = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

Not the same

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

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 $V(terminal) = 0$

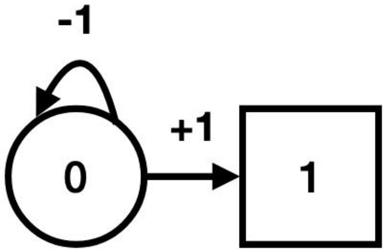
Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

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

Asynchronous Dynamic Programming

- Asynchronous DP algorithms are in-place iterative DP algorithms that are not organized in terms of systematic sweeps of the state set
- The values of some states may be updated several times before the values of others are updated once
- An asynchronous algorithm must continue to update the values of all the states
- Of course, avoiding sweeps does not necessarily mean that we can get away with less computation. It just means that an algorithm does not need to get locked into any hopelessly long sweep before it can make progress improving a policy.
- We can try to take advantage of this flexibility by selecting the states to which we apply updates so as to improve the algorithm's rate of progress
- Asynchronous algorithms also make it easier to intermix computation with real-time interaction

Demo: MC vs. Iterative policy evaluation



$$v_\pi(0) \doteq E_\pi\left[G_0 \mid S_0 = 0\right] \approx \sum_{e=1}^{N} \frac{G_{0,e}}{N}$$

Return of eth episode

- [colab]()
-

$$\left.\begin{array}{c}\pi' \\ \downarrow E \\ v_{\pi'} \\ \downarrow I \\ \pi^* \end{array}\right\}\text{iteration}$$

have one more iteration to have both v^* and π^*.

$$\left\{\begin{array}{c}\pi^* \\ \downarrow E \\ v_{\pi^*} \\ \downarrow I \\ \pi^* \end{array}\right.$$

$\text{greedify}(v_{\pi^*}) = \bar{\pi}$

where $v_{\bar{\pi}} = v_{\pi^*}$

policy π: probability of chose action 1 or 2.