Lecture 2: Dynamic Programming

Zhi Wang & Chunlin Chen

Department of Control and Systems Engineering
Nanjing University

Oct. 10th, 2020

Table of Contents

Markov Decision Process (MDP)
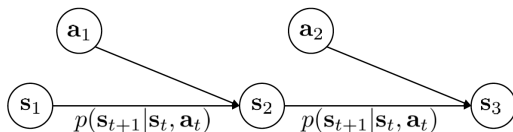
$M = \langle S, A, T, R \rangle$

S: State space   state $s \in S$ (discrete/continuous)

A: Action space   action $a \in A$ (discrete/continuous)

T: Transition operator $T_{i,j,k} = p(s_{t+1} = j | s_t = i, a_t = k)$

R: Reward function $R_{i,j,k} = r(s_{t+1} = j | s_t = i, a_t = k)$

Markov Decision Process (MDP)

- A classical formalization of sequential decision making
  - Choosing different actions in different situations
  - Actions influence not just immediate rewards, but also subsequent situations through future rewards
  - Involve delayed reward and the need to tradeoff immediate and delayed reward
- A mathematically idealized form of the RL problem
  - Precise theoretical statements can be made
  - Key elements of the problem's mathematical structure, such as returns, value functions, Bellman equations, etc
  - A tension between breadth of applicability and mathematical tractability

The Goal of RL

- Find **optimal policies** to maximize cumulative reward

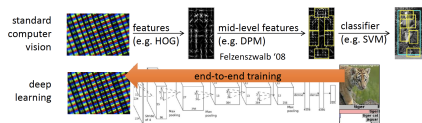$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} r(s_t, a_t) \right]$$

  - In a **trial-and-error** manner
  - A general **optimization** framework for sequential decision-making
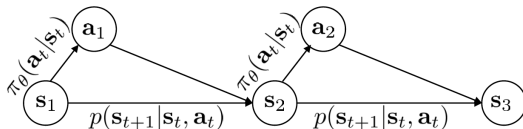
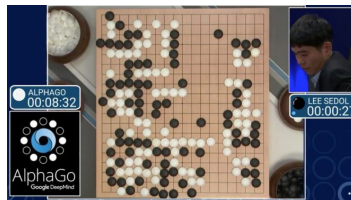# Supervised learning vs. Sequential decision making

Supervised learning

- Samples are independent and identically distributed (i.i.d.)
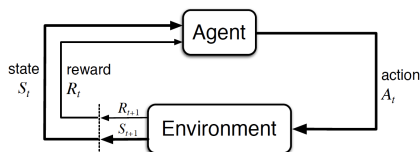- Given an input, map an optimal output

Reinforcement learning

- Samples are not i.i.d., temporally co-related
- Given an initial state, find a sequence of optimal actions

The agent-environment interface

- **Agent:** The learner, decision maker
- **Environment:** The thing it interact with, comprising everything outside the agent

At each time step $t = 0, 1, 2, ...$, the agent...

- receives some representation of the environment's **state**, $S_t \in \mathcal{S}$
- on that basis, selects an **action**, $A_t \in \mathcal{A}(s)$
- one time step later, receives a numerical **reward**, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$
- finds itself in a new state, S_{t+1}, (**transition function**)
- $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3...$

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

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \quad \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

- The probabilities given by p completely characterize the environment's dynamics
- **Markov Property**
  - The probability of each possible value for S_t and R_t depends only on the immediately preceding state S_{t-1} and action A_{t-1}, not at all on earlier states and actions
  - $p(S_t, R_t|S_{t-1}, A_{t-1}) =$
    $p(S_t, R_t|S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, S_{t-3}, A_{t-3}, ...)$
- Recall supervised learning $p(X_i|X_j) = 0$

Dynamics of the MDP $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$

- The probabilities given by p completely characterize the environment's dynamics

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

Incremental Reinforcement Learning With Prioritized Sweeping for Dynamic Environments

Zhi Wang, Chunlin Chen, *Member, IEEE*, Han-Xiong Li, *Fellow, IEEE*, Daoyi Dong, *Senior Member, IEEE*, and Tzyh-Jong Tarn, *Life Fellow, IEEE*

III. INCREMENTAL REINFORCEMENT LEARNING

As an example, imagine a search-and-rescue robot to detect injured people inside damaged buildings. The falling bricks or steels may block the shortest path to the wounded that the robot has already learned. Since relearning the new environment from scratch is too time-consuming, it is reasonable and effective to revise the optimal policy incrementally after learning the changed part first. In this type of problems, incremental learning can be achieved and the learning process can be accelerated to adapt to the dynamic environment. The concept of IRL has been initialized in our conference paper [37] and a formal framework for IRL is presented in detail with related techniques and specific algorithms in this section. In particular, we consider dynamic environments where the reward functions may change over time.

Incremental Reinforcement Learning in Continuous Spaces via Policy Relaxation and Importance Weighting

Zhi Wang, *Student Member, IEEE*, Han-Xiong Li, *Fellow, IEEE*, and Chunlin Chen, *Member, IEEE*

A. Problem Formulation

We consider the dynamic environment as a sequence of stationary tasks on a certain timescale where each task corresponds to the specific type of environment characteristics during the associated time period. Assume there is a space of MDPs, $\mathcal{M}$, and an infinite underlying distribution, $\mathcal{D}$, over time in $\mathcal{M}$. An RL agent interacts with the dynamic environment $\mathcal{D} = \{M_1, \ldots, M_{t-1}, M_t, \ldots\}$, where each $M_t \in \mathcal{M}$ denotes the specific MDP that is stationary during the tth time period. We assume, in this paper, that the environment changes only in the reward and state transition functions, but keeps the same state and action spaces.

Goals and rewards

- **Reward Hypothesis**
  - That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

- The agent's goal: maximize the total amount of reward it receives
  - maximize $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right]$
  - Maximize not immediate reward, but cumulative reward in the long run

Returns and episodes

- The subsequence of the agent-environment interaction, **episodes**
  - $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, ..., S_{T-1}, A_{T-1}, R_T, S_T$
  - Each episode ends in a special state called the terminal state, S_T

- We seek to maximize the **expected return**, G_t, the reward sequence
  - $G_t = R_{t+1} + R_{t+2} + ... + R_T$
  - Episodic tasks

Discount rate $\gamma \in [0, 1]$

- The expected **discounted** return
  - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}... = \sum_{k=1}^{\infty} \gamma^k R_{t+k+1}$
- The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately
- $\gamma \to 0$, the agent is "myopic", only maximizing immediate rewards
  - Akin to supervised learning that maximizes the log-likelihood of each sample, $\log p(y_i|x_i)$
- $\gamma \to 1$, the agent is "farsighted", taking future rewards into account
- Returns at successive time steps are related to each other

$$G_t = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + ...)$$
$$= R_{t+1} + \gamma G_{t+1}$$

Policies and value functions

- **Value functions**: function of states or state-action pairs
    - Estimate how good it is for the agent to be in a given state, or how good it is to perform a given action in a given state
    - "How good": defined in expected future rewards, i.e., expected return
    - Depend on what actions to take, defined w.r.t. particular ways of acting, called **policies**

- **Policy**: $\pi(a|s)$, a mapping from states to probabilities of selecting each possible action
    - $\sum_a \pi(a|s) = 1$
    - e.g., for a given state s_1, four possible actions
      $a_1 = 0.1, a_2 = 0.3, a_3 = 0.2, a_4 = 4$

- RL = estimate value functions + estimate polices + estimate both

Policies and value functions

- v_π, the **state-value function** for policy π

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=1}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right], \forall s \in \mathcal{S}$$
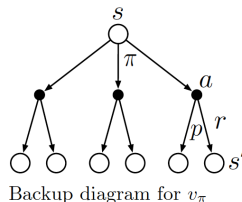
- Q_π, the **action-value function** for policy π

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$
$$= \mathbb{E}_\pi\left[\sum_{k=1}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right], \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

A fundamental property: Bellman equation

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

- Average over all the possibilities, weighting each by its probability of occurring
- Express the relationship between the value of a state and the values of its successor states
- Transfer value information back to a state from its successor states
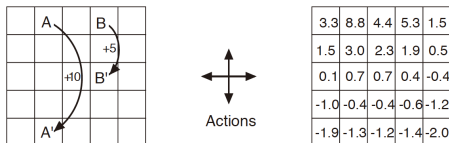

Backup diagram for v_π

Example: Gridworld

Figure 3.3: Gridworld example: exceptional reward dynamics (left) and state-value function for the equiprobable random policy (right).

$$v_\pi(s) = \sum_a \pi(a|s)[r + \gamma v_\pi(s')]$$

- Actions that would take the agent off the grid leave its location unchanged, but also result in $r = -1$, otherwise $r = 0$
- From state A, all four actions yield $r = 10$ and take the agent to A'
- From state B, all actions yield $r = 5$ and the agent to B'

Optimal policies and optimal value functions

- RL tasks: find a policy that achieves a lot of reward over the long run
- Value functions define a partial ordering over policies
  - $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s), \forall s \in \mathcal{S}$
- At least one policy that is better than or equal to all other policies, i.e., **optimal policy**
- Optimal policies, π_*, share the same **optimal value function**

$$v_*(s) = \max_\pi v_\pi(s), \quad \forall s \in \mathcal{S}$$

$$Q_*(s, a) = \max_\pi Q_\pi(s, a), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

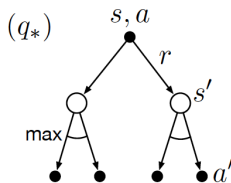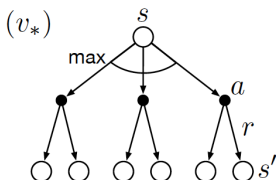$$Q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

Bellman optimality equation

$$
\begin{aligned}
v_*(s) &= \max_a Q_{\pi_*}(s, a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r | s, a)[r + \gamma v_*(s')]
\end{aligned}
$$

- The value of a state under an optimal policy must equal the expected return for the best action from that state

Bellman optimality equation

$$Q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q_*(S_{t+1}, a') | S_t = s, A_t = a]$$

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


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

Determine optimal policies from optimal value functions

- For v_*: a one-step search
  - Actions that appear best after one-step search will be optimal actions

- For Q_*: no need to do a one-step-ahead search
  - $a_* = \arg\max_a Q_*(s, a)$
  - The optimal action-value function allows optimal actions to be selected without having to know anything about possible successor states and their values, i.e., without having to know anything about the environment's dynamics
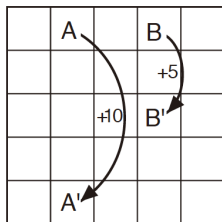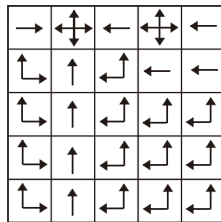
Figure 3.6: Optimal solutions to the gridworld example.

Table of Contents

Dynamic Programming (DP)

- It refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a **recursive** manner.
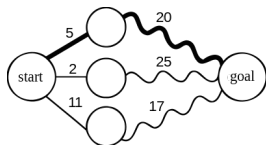
- Finding the shortest path in a graph using optimal substructure
- A straight line: a single edge, a wavy line: a shortest path
- The bold line: the overall shortest path from start to goal

Dynamic Programming (DP)

- A collection of algorithms that can be used to compute optimal policies given a perfect model of the environment (MDP)
  - Of limited utility in RL both because of their assumption of a perfect model and because of their great computational expense
  - Important theoretically, provide an essential foundation for the understanding of RL methods
  - RL methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment

Policy evaluation (Prediction)

- Compute the state-value function v_π for an arbitrary policy π

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

- If the environment's dynamics are completely known
  - In principal, the solution is a straightforward computation

Iterative policy evaluation

- Consider a sequence of approximate value functions $v_0, v_1, v_2, ...$
  - The initial approximation, v_0, is chosen arbitrarily
- Use the **Bellman equation** for v_π as an update rule

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

- $v_k = v_\pi$ is a fixed point for this update rule
  - The sequence $\{v_k\}$ converges to v_π as $k \to \infty$ under the same conditions that guarantee the existence of v_π

Iterative policy evaluation

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

- The updates as being done in a **sweep** through the state space

$$R_t = -1$$
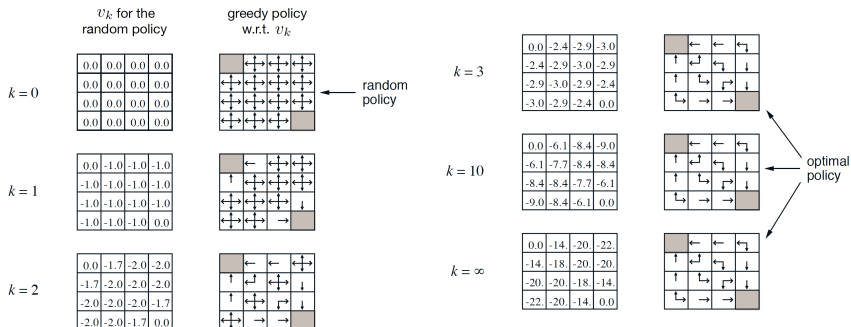on all transitions

actions

- Four actions deterministically case the corresponding state transitions
  - e.g., $p(6, -1|5, right) = 1, p(7, -1|7, right) = 1$
  - $p(10, r|5, right) = 0, \forall r \in \mathcal{R}$

The agent follows the equiprobable random policy

- The final estimate is in fact v_π
  - The negation of the expected number of steps from that state until termination
  - 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

Policy improvement

- Our reason for computing the value function for a policy is to help find better policies
  - We have determined the value function v_π for policy π
  - we would like to know whether or not we should change the policy to deterministically choose an action $a \neq \pi(s)$
  - We know how good it is to follow the current policy from s, e.g., v_π, but would it be better or worse to change to the new policy, π'?

- Consider selecting a in s and thereafter following the existing policy π

$$Q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

  - If $Q_\pi(s, a) \geq v_\pi(s)$?

Policy improvement theorem

- Let π and π' be any pair of deterministic policies such that,

$$Q_\pi(s, \pi'(s)) \geq v_\pi(s), \quad \forall s \in \mathcal{S}.$$

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

Policy improvement theorem

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

Policy improvement

- Consider the new **greedy** policy, π', selecting at each state the action that appears best according to $Q_\pi(s, a)$

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

- The process of making a new policy that improves on an original policy, by making greedy w.r.t. the value function of the original policy, is called **policy improvement**
  - The greedy policy meets the conditions of the policy improvement theorem

Policy improvement

- Suppose the new policy π' is as good as, but not better than, the old policy π, then $v_{\pi'} = v_\pi$

$$v_{\pi'}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1})|S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi'}(s')]$$

  - The sane as the Bellman optimality equation
  - Both π and π' must be optimal policies
- Policy improvement must give us a strictly better policy except when the original policy is already optimal

Table of Contents

Policy iteration

- Using policy improvement theorem, we can obtain a sequence of monotonically improving policies and value functions

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

- This process is guaranteed to converge to an optimal policy and optimal value function in a finite number of iterations
  - Each policy is guaranteed to be a strictly improvement over the previous one unless it is already optimal
  - A finite MDP has only a finite number of policies

Policy iteration

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*
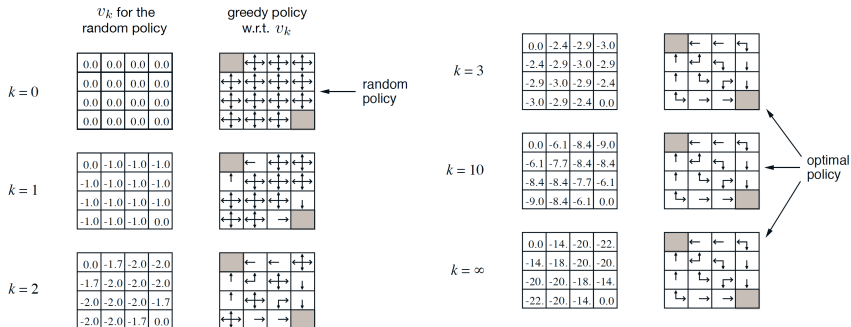   For each $s \in \mathcal{S}$:
       old-action $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r | s, a) \big[r + \gamma V(s') \big]$
       If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy iteration often converges in very few iterations

- The final estimate is in fact v_π
  - The negation of the expected number of steps from that state until termination
  - 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

Value iteration

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
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
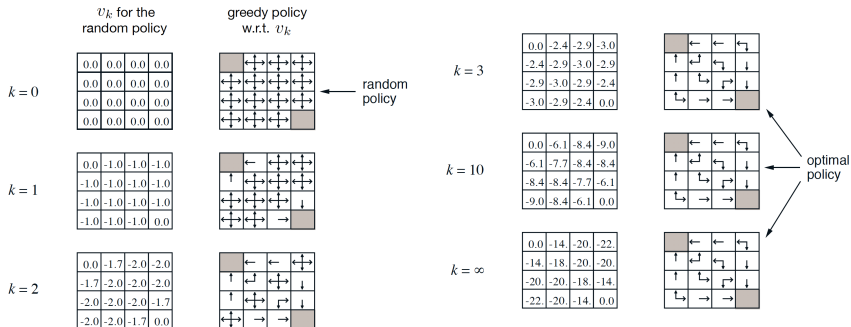 $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

- Each policy iteration involves policy evaluation, which may be a protracted iterative computation **requiring multiple sweeps through the state set**

v_k for the random policy

greedy policy w.r.t. v_k

$k = 0$, $k = 1$, $k = 2$, $k = 3$, $k = 10$, $k = \infty$

random policy

optimal policy

- Policy evaluation iterations beyond the first three have no effect on the corresponding greedy policy

Value iteration = Truncate policy evaluation for one sweep

- In policy iteration, stop policy evaluation after just one sweep

$$v_{k+1}(s) = \sum_{s',r} p(s',r|s,\pi_k(s))[r + \gamma v_k(s')]$$

$$\pi_{k+1}(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_{k+1}(s')]$$

- Combine into one operation, called **value iteration** algorithm

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

  - For arbitrary v_0, the sequence $\{v_k\}$ converges to v_* under the same conditions that guarantee the existence of v_*
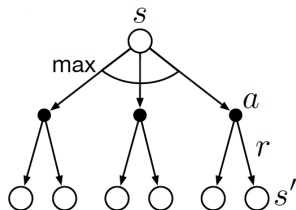
Value iteration

- Bellman optimality equation

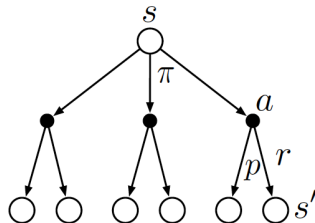$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$$

- Value iteration

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

  - Turn Bellman optimality equation into an update rule
  - Directly approximate the optimal state-value function, v_*

Value iteration vs. policy evaluation

Backup diagram for
value iteration

Backup diagram for
policy evaluation

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

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

- One sweep = one sweep of policy evaluation + one sweep of policy improvement

Properties of dynamic programming

- **Bootstrapping**: Update estimates on the basis of other estimates
    - Estimate the values of states based on estimates of the values of successor states
- **Model-based**: Require the accurate model of the environment
    - The complete probability distributions of all possible transitions, $p(s', r|s, a)$

Algorithms	Bootstrapping?	Model-based?
Dynamic programming	Yes	Yes
Monte Carlo methods	No	No
Temporal-difference learning	Yes	No

Why is DP fundamental and important?

- Important theoretically, provide an essential foundation for the understanding of RL methods
- RL methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment

Why is DP fundamental and important?

Incremental Reinforcement Learning With Prioritized Sweeping for Dynamic Environments

Zhi Wang, Chunlin Chen, *Member, IEEE,* Han-Xiong Li, *Fellow, IEEE,* Daoyi Dong, *Senior Member, IEEE,* and Tzyh-Jong Tarn, *Life Fellow, IEEE*

Algorithm 3: Prioritized Sweeping of Drift Environment.

Input: $Q^*(s, a), \forall (s, a) \in (S, A)$ in E;
     the small threshold θ

Output: $Q_{dn}(s_{dn}, a_{dn}), \forall (s_{dn}, a_{dn}) \in (S_{dn}, A_{dn})$

1   $E_d \leftarrow$ the drift environment using Algorithm 2
2   $E_{dn}^m \leftarrow m$-degree neighbor environment
3   Initialize $E_{dn}^m : Q_{dn}(s_{dn}, a_{dn}) \leftarrow Q^*(s_{dn}, a_{dn})$,
    $\forall (s_{dn}, a_{dn}) \in (S_{dn}, A_{dn}) \bigcap (S, A)$
4   Initialize: $\Delta_{max} \leftarrow \infty$
5   **while** $\Delta_{max} \geq \theta$ **do**
6     $\Delta_{max} \leftarrow 0$
7   **for** $(s_{dn}, a_{dn}) \in (S_{dn}, A_{dn})$ **do**
8        $Q_{temp} \leftarrow \sum_{s'_{dn}} p(s'_{dn} | s_{dn}, a_{dn})[$
        $r_{dn}(s_{dn}, a_{dn}, s'_{dn}) + \gamma \max_{a'_{dn}} Q_{dn}(s'_{dn}, a'_{dn})]$
9        $\Delta \leftarrow |Q_{temp} - Q_{dn}(s_{dn}, a_{dn})|$
10       $Q_{dn}(s_{dn}, a_{dn}) \leftarrow Q_{temp}$
11   **if** $\Delta > \Delta_{max}$ **then**
12         $\Delta_{max} \leftarrow \Delta$
13   **end**
14   **end**
15 **end**

After updating the drift environment, we then update the state-action space of its m-degree neighbor environment with the second priority. It can be viewed as a process of spreading the changes caused by the drift environment to its neighbors and even to the whole state-action space gradually. In this process, the new information is fused into the existing knowledge system starting from the drift environment. The process of updating the state-action space of the drift environment and its neighbor environment with priority by dynamic programming is called as *prioritized sweeping of drift environment,* which is shown as in Algorithm 3.

Fast reinforcement learning with generalized policy updates

André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup

Generalized Policy Updates

From the discussion above, one can see that an important branch of the field of RL depends fundamentally on the notions of policy evaluation and policy improvement. We now discuss generalizations of these operations.

Definition 3. "Generalized policy evaluation" (GPE) is the computation of the value function of a policy π on a set of tasks $\mathcal{R}$.

Definition 4. Given a set of policies Π and a task r, "generalized policy improvement" (GPI) is the definition of a policy π' such that

$$Q_r^{\pi'}(s, a) \geq \sup_{\pi \in \Pi} Q_r^{\pi}(s, a) \; for\, all \; (s, a) \in \mathcal{S} \times \mathcal{A}. \qquad [5]$$

Learning objectives of this lecture

You should be able to...

- Know elements of finite MDPs, the agent-environment interface
- (Discounted) returns and episodes, polices and value functions, Bellman (optimality)

- Understand and be able to use dynamic programming
- Know policy evaluation and policy improvement, policy iteration and value iteration

- Chapter 3: Finite Markov Decision Processes, Chapter 4: Dynamic Programming, *Reinforcement Learning: An Introduction*, 2nd Edition
  - http://202.119.32.195/cache/2/03/incompleteideas.net/5b682cef88335157bc5542267cf3f442/RLbook2018.pdf

THE END