Tufts

Class #24: Solving MDPs & Reinforcement Learning

Machine Learning (COMP 135): M. Allen, 15 Apr. 20

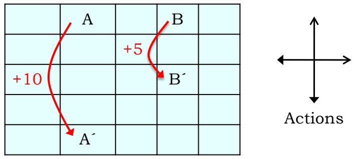
Review: The Bellman Equation

- Richard Bellman (1957), working in Control Theory, was able to show that the utility of any state s, given policy of action π, can be defined recursively in terms of the utility of any states we can get to from s by taking the action that π dictates:

$$U^{\pi}(s) \;=\; \sum_{s'} P(s,\pi(s),s')\;[R(s,\pi(s),s') + \gamma\, U^{\pi}(s')]$$

- Furthermore, he showed how to actually calculate this value using an iterative dynamic programming algorithm

Solving the Bellman Equation



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

- Next, we will see how to ***solve*** the general Bellman Equation for any set of states, probabilities, and rewards, over any time horizon
- Here, we see the solution for a grid with dynamics as follows:
  - Agent policy: ***move randomly*** in one of 4 directions
  - If agent hits a wall, reward is R = -1
  - All other moves are reward R = 0, except for in two special states A and B, where any action takes agent to A´ or B´ with reward indicated
  - Discount factor (gamma) is γ = 0.9

Example from: Sutton & Barto, 1998

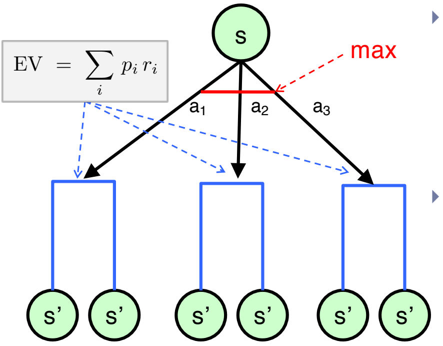
Evaluating a Policy Iteratively

```
function POLICY-EVALUATION($mdp$, $\pi$) returns a value function
    inputs: $mdp$, an MDP, and $\pi$, a policy to be evaluated
    local variables: $\Delta$, maximal amount policy values change per iteration,
                     $\Theta$, a small positive constant

    $\forall s \in S:\ U(S) = 0$
    repeat while $\Delta \geq \Theta$
        $\Delta \leftarrow 0$
        $\forall s \in S:$
            $u \leftarrow U(s)$
            $U(s) \leftarrow \sum_{s'} P(s,\pi(s),s')[R(s,\pi(s),s') + \gamma\, U(s')]$
            $\Delta \leftarrow \max(\Delta,\ |U(s) - u|)$

    return value function $U \approx U^{\pi}$
```

Note: if we set Θ to $\dfrac{\varepsilon\,(1-\gamma)}{\gamma}$ ***approximation error*** is at most ε

- Policy evaluation: given a policy, we calculate the expected value for every state if we follow the policy, iterating until values converge (quit changing very much)

Finding the Optimal Policy (π*)



- Before, to calculate $U^\pi(s)$, we only looked at ***single* set** of actions: those given for each state by policy,

$$a_i = \pi(s_i)$$

- Now, we will consider ***all possible actions***, taking the one that is the best at each state before we sum over the various probabilities and rewards in the system

Bellman Equations

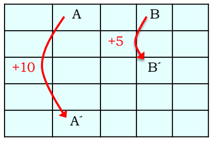
- We have seen that the utility of any state s in a given policy π can be calculated iteratively:

$$U^\pi(s) \;=\; \sum_{s'} P(s,\pi(s),s')\;[R(s,\pi(s),s') + \gamma\, U^\pi(s')]$$

- This same equation can be used to find the value of the ***best possible policy***, simply by calculating what we get if we always take the best action:

$$U^\star(s) = \max_\pi U^\pi(s)$$
$$= \max_a \sum_{s'} P(s,a,s')\,[R(s,a,s') + \gamma\, U^\star(s')]$$

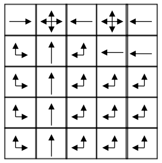
Solving for the Optimal Policy



Problem domain

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

U*



π*

- Before, we looked at the value of the ***purely random*** policy for this particular grid problem
- We can use the Bellman Equation to find the ***optimal policy***
  - Here we see the optimal value function, U*, and the associated optimal policy, π* (where in some cases, multiple actions are all equally good/bad)

Example from: Sutton & Barto, 1998

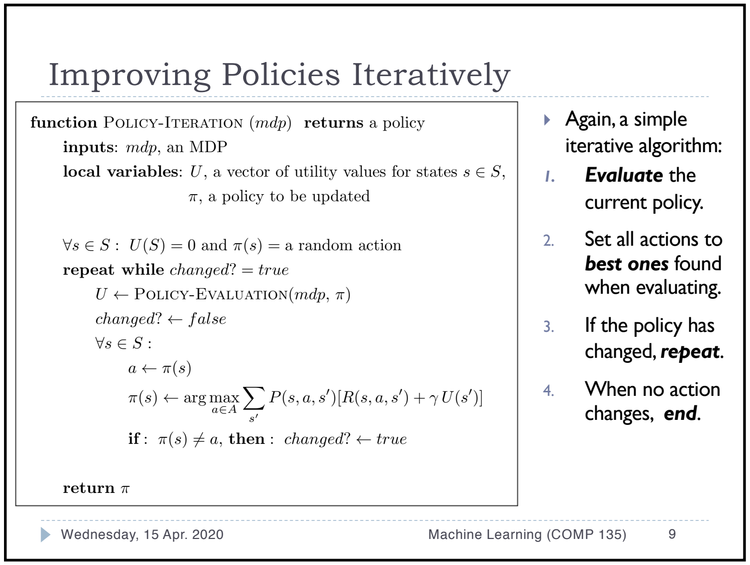
Policy Improvement

- Once we figure out the value for each state under our ***current*** policy, we can choose ***new actions***

$$U^\pi(s) = \sum_{s'} P(s,\pi(s),s')[R(s,\pi(s),s') + \gamma\, U(s')]$$

$$\pi'(s) = \arg\max_a \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma\, U(s')]$$

- Our choice is simple: just set our new policy in a ***greedy way***, choosing the best action available
  - This choice is based on the ***current set*** of values
  - Creates a new policy when we change some action
  - If the policy ***does change***, then we need to update our values again



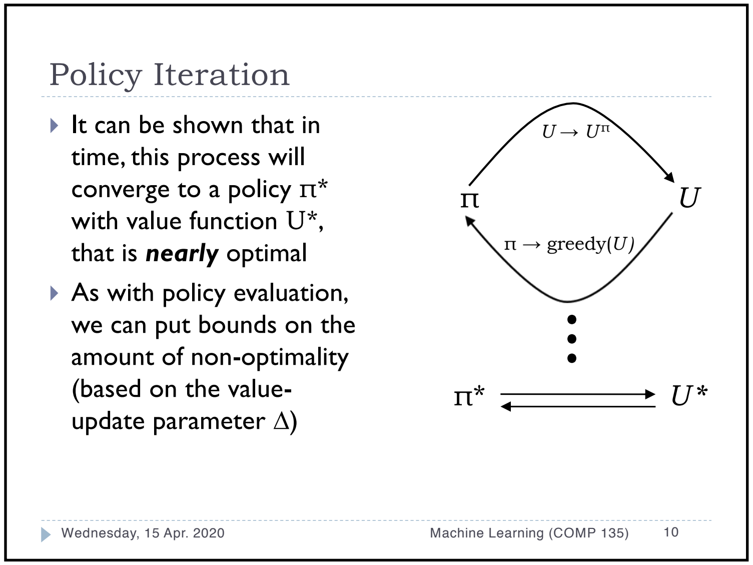
Improving Policies Iteratively

```
function POLICY-ITERATION ($mdp$) returns a policy
    inputs: $mdp$, an MDP
    local variables: $U$, a vector of utility values for states $s \in S$,
                     $\pi$, a policy to be updated

    $\forall s \in S:\ U(S) = 0$ and $\pi(s) =$ a random action
    repeat while $changed? = true$
        $U \leftarrow$ POLICY-EVALUATION($mdp$, $\pi$)
        $changed? \leftarrow false$
        $\forall s \in S:$
            $a \leftarrow \pi(s)$
            $\pi(s) \leftarrow \arg\max_{a \in A} \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma U(s')]$
            if : $\pi(s) \neq a$, then : $changed? \leftarrow true$

    return $\pi$
```

- Again, a simple iterative algorithm:
  1. ***Evaluate*** the current policy.
  2. Set all actions to ***best ones*** found when evaluating.
  3. If the policy has changed, ***repeat***.
  4. When no action changes, ***end***.

Wednesday, 15 Apr. 2020 Machine Learning (COMP 135) 9

9

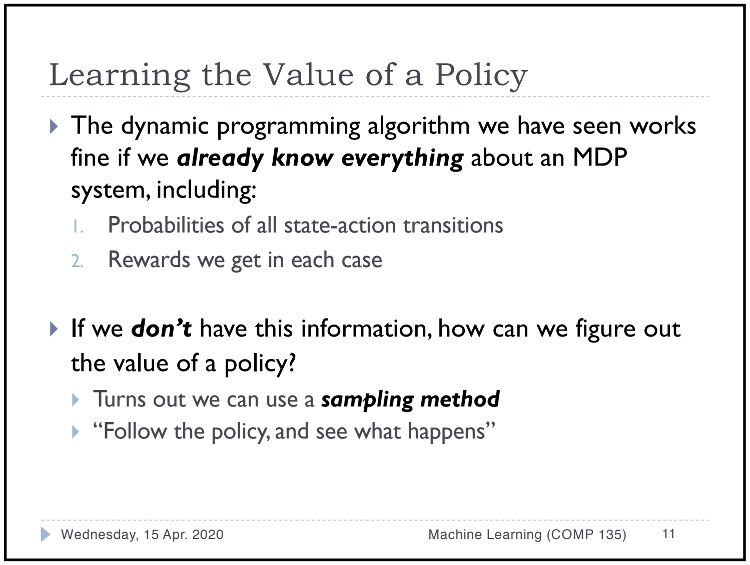


Policy Iteration

- It can be shown that in time, this process will converge to a policy π* with value function U*, that is ***nearly*** optimal
- As with policy evaluation, we can put bounds on the amount of non-optimality (based on the value-update parameter Δ)

Wednesday, 15 Apr. 2020 Machine Learning (COMP 135) 10

10

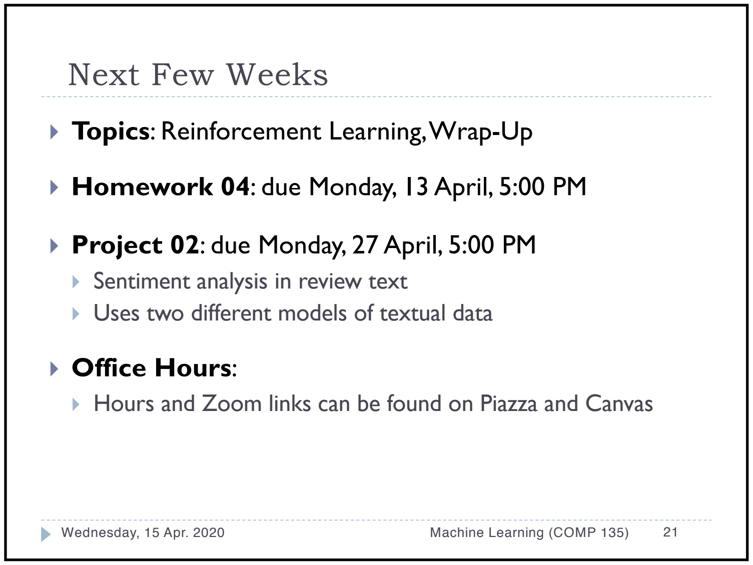


Learning the Value of a Policy

- The dynamic programming algorithm we have seen works fine if we ***already know everything*** about an MDP system, including:
  1. Probabilities of all state-action transitions
  2. Rewards we get in each case
- If we ***don't*** have this information, how can we figure out the value of a policy?
  - Turns out we can use a ***sampling method***
  - "Follow the policy, and see what happens"

Wednesday, 15 Apr. 2020 Machine Learning (COMP 135) 11

11



Next Few Weeks

- **Topics**: Reinforcement Learning, Wrap-Up
- **Homework 04**: due Monday, 13 April, 5:00 PM
- **Project 02**: due Monday, 27 April, 5:00 PM
  - Sentiment analysis in review text
  - Uses two different models of textual data
- **Office Hours**:
  - Hours and Zoom links can be found on Piazza and Canvas

Wednesday, 15 Apr. 2020 Machine Learning (COMP 135) 21

21