CS 229 Lecture on MDPs and Value / Policy Iteration:

Notes taken from Lecture 17, CS 229, Autumn 2018, Prof. Ng, Stanford Online.

- MDP Comprises:
  1. States
  2. Actions
  3. State transition Probabilities
  4. Discount Factor
  5. Rewards

$s_0, a_0, s_1 \sim P_{s_0,a_0}, a_1, s_2 \sim P_{s_1,a_1}, \ldots$

$V(s_0) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots$

Policy: $S \longmapsto a$

- How do you compute the optimal policy?
  - ↳ Exponentially large # of combinations given # of States.
  - ↳ Define: V^π, V^*, π^*

For a policy π, $V^\pi : S \to \mathbb{R}$ is such that $V^\pi(s)$ is the expected total payoff for starting in state s and executing π.

$$V^\pi(s) = \mathbb{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots \mid \pi, s_0 = s \right]$$

↳ "Value function for policy π"

  - ↳ Bellman's Equation:

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P_{s,\pi(s)}(s') V^\pi(s')$$

$$V^\pi(s_0) = \left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots \mid \pi, s_0 = s \right]$$

($R(s_0)$: Immediate reward; $\gamma R(s_1)$, $\gamma^2 R(s_2)$: For deterministic policy)

$$= \left[R(s_0) + \gamma \underbrace{\left(R(s_1) + \gamma R(s_2) + \ldots \right)}_{V^\pi(s_1)} \mid \pi, s_0 = s \right]$$

($V^\pi(s_1)$: Expected Future reward)

$$V^{\pi}(s) = R(s) + \gamma V^{\pi}(s') \quad \text{for deterministic policy}$$

↳ What is the distribution s' is drawn from?

$$s' \sim P_{s, \pi(s)}$$

↳ In state s, take action $a = \pi(s)$

∴ In general form, you want <u>expected</u> value of future states

$$\therefore V^{\pi}(s) = R(s) + \gamma \sum_{s'} P_{s,\pi(s)}(s') V^{\pi}(s')$$

Bellman Equation

$V^{\pi}(s)$: Value of future state

$P_{s,\pi(s)}(s')$: Probability of transition to future state

$\sum_{s'}$: All possible future states

↳ This can be set up as a linear system of equations to solve for Value functions

↳ Given π, get a linear system of equations in terms of $V^{\pi}(s)$

↳ For each state s, you can write a linear equation for $V^{\pi}(s)$ in terms of $P_{ss'}$ $V^{\pi}(s')$ where s' any/all other states.

$$\begin{bmatrix} V_{\pi}(s_1) \\ \vdots \\ V_{\pi}(s_n) \end{bmatrix} = \begin{bmatrix} R(s_1) \\ \vdots \\ R(s_n) \end{bmatrix} + \begin{bmatrix} P_{11}^{\pi} & \cdots & P_{1n}^{\pi} \\ \vdots & \ddots & \vdots \\ P_{n1}^{\pi} & \cdots & P_{nn}^{\pi} \end{bmatrix} \begin{bmatrix} V_{\pi}(s_1) \\ \vdots \\ V_{\pi}(s_n) \end{bmatrix}$$

- V^{*} is the <u>optimal value function</u>.

$$V^{*}(s) = \max_{\pi} V^{\pi}(s)$$

↳ Look across all possible policies you could implement, what is the best value you could get for state s?

↳ There is a different set of Bellman Equations for optimal value function:

↳ Bellman Optimality Equations:

$$V^*(s) = R(s) + \gamma \max_a \sum_{s'} P_{s,a}(s') V^*(s')$$

($P_{s,a}(s')$: if you take action a from state s)

↳ You want to take action a which maximizes Expected accumulated future value function

↳ Best payoff at s is reward at s plus maximum expected future payoff over all possible actions.

$$\therefore \pi^*(s) = \arg\max_a \sum_{s'} P_{sa}(s') V^*(s')$$

(π^*: Optimal policy)

↳ Whatever policy gives the action that maximizes total payoff.

- Bellman optimality equations give way to compute optimal policy.

↳ Claim: $V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$

↳ Strategy for finding optimal policy:

① Find V^*

② Use argmax equation to find π^*

↳ How do you compute V^*?

↳ Value Iteration:

① Initialize $V(s) := 0$ for every s

② For each s, update:

$$V(s) := R(s) + \max_a \gamma \sum_{s'} P_{s,a}(s') V(s')$$

↳ i.e. Update using optimality equation

↳ To implement this, create an n-dimensional vector representing value function of n initial states. Initialize vector to zeros. Repeatedly update $V(s)$ using Bellman Optimality Equations.

Value iteration works with both, but synchronous update more efficient because vectorization

* ↳ Synchronous update: Compute $V(s)$ for all states, then overwrite vector for $V(s)$ of all states at once

↳ Asynchronous Update: Compute each $V(s)$ and update it immediately before computing next $V(s)$

- If using Synchronous update, you use old estimates to compute new estimates.

↳ One-step often called Bellman backup operator

$$V := B(V)$$

↳ Proof of convergence (not included here)

↳ V will converge to V^*

↳ Converges quickly

↳ Error reduces by factor of discount factor at each iterations.

↳ Check this

- If you run value iteration on an MDP, you get $V^*(s)$ for every state. The actions that got you to V^* in the last iteration gives you π^*

↳ Given an MDP, solve for V^* and use this to get π^*

↳ So far, π of states is finite.

↳ For continuous MDP's often you discretize into finite MDPs

- **Policy Iteration**: Starting at 50:00 timepoint.

① Initialize π randomly

② Repeat:

Set $V := V^\pi$ (i.e. Solve the Bellman Equation to get V^π)

Set $\pi(s) := \arg\max_a \sum_{s'} P_{sa}(s') V(s')$

↳ Recall in Value iteration, our focus is on solving for V^*

↳ In policy iteration, focus is on π

V^π is linear system of equations with n equations, n unknowns ← for V^π

↳ Solved efficiently w/ linear solver

Pretend V^π is V^* and update $\pi(s)$ using Bellman optimality equation.

↳ i.e. $\pi(s) := \arg\max_a \sum_{s'} P_{sa}(s') V(s')$

↳ Select action which maximizes expected future reward.

Policy evaluation can also be done numerically by iteratively applying policy, right?

- $\pi(s)$ will eventually converge.

- In value iteration, you wait until the end to get $\pi(s)$.
  In policy iteration, you get a new $\pi(s)$ at each iteration.

- Pros and Cons of Policy vs. Value iteration

  - For small state space, linear system of equations easy to solve

    ∴ policy iteration works quickly and is preferred

  - For large set of states, linear system of equations slower to solve.

    ∴ Value iteration is preferred.

  - More academic:

    $V \to V^*$ in value iteration, but never quite gets to exactly V^*

    $V = V^*$ eventually in policy iteration (if using linear systems)

  - Value iteration used most often in practice due to large state spaces.

- What if you don't know P_{sa} ??

  ↳ State transition probabilities are often not known in advance

  ↳ In practice, estimated from data / simulation

  ↳ Execute some policy and estimate P_{sa} from data

  $$P_{sa}(s') = \frac{\#\text{ of times you took action } a \text{ in state } s \text{ and got to } s'}{\#\text{ of times you took action } a \text{ in state } s}$$

  ↳ Maximum likelihood estimate

  (or $\frac{1}{|S|}$ if above is $\frac{0}{0}$)

  ↳ Heuristic is initialize to uniform probability

  ↳ Can use Laplace smoothing, but you don't have to.

  ↳ Solves for $0/0$ problem

↳ But MDP solvers are not sensitive to zero probabilities.

- **Recap:**
  - If given robot and asked to implement MDP solver:

Repeat {

① Take actions wrt π to get experience in MDP

② Update estimates of P_{sa} (and possibly R)

③ Solve Bellman Equation using value iteration to get V

④ Update $\pi(s) := \underset{a}{\operatorname{argmax}} \sum_{s'} P_{sa}(s') V(s')$

}

- Reward function may be unknown, may be a little random (random function of environment/state)

- **Exploration vs. Exploitation**

↳ If during training, robot happens to find locally optimal outcome, it may greedily stay near local optimal and never find global optimal.

↳ How greedy should you be when acting on MDP at just taking actions to maximize reward.

↳ Previous algorithms are greedy

↳ Explore = take actions that are purposely non-optimal to explore new space

↳ Exploration vs. Exploitation tradeoff

↳ Major consideration in advertising.

↳ Explore user interests.

- Modification to Recap which enables exploitation:

  ↳ Instead of just taking actions w.r.t. π, give some chance of taking actions randomly.

    ↳ Exploration is called "ϵ greedy" ("epsilon greedy exploration")

    ϵ is chance of taking random move

    A little bit of a misnomer. More literally $(1-\epsilon)$ greedy.

    ↳ With ϵ greedy exploration will eventually converge to optimal policy for any MDP.

    ↳ Kind of like simulated annealing

    ↳ Another type of exploration is called Boltzmann Exploration

      ↳ ϵ greedy is most common

- Intrinsic Motivation = giving reward for finding new states

- How many actions to take before updating π in recap?

  ↳ As frequently as possible!