- Feb. 24 Notes: Reinforcement Learning for Control

  - Ben Vanroy course gives mathematical depth of RL

  - DP / ADP assume probability model knowledge, which we do not have.

    ↳ Environment (actual or simulated) serves individual experiences in RL

  - For control, goal is to find optimal value function / policy.

  - Start with Tabular RL:

    - Concept: Generalized Policy Iteration (GPI)

      ↳ You don't need to be tied to DP method for policy evaluation or improvement.

      ↳ Do some kind of policy evaluation or improvement.

    - GPI with Tabular MC Evaluation:

      ↳ Monte Carlo approach to policy evaluation instead of DP

      ↳ But greedy policy improvement requires an MDP

      ↳ If we work with Action value function estimated from samples:

$$\pi_0'(s) \longleftarrow \underset{a \in A}{\text{Argmax}} \; Q^{\pi}(s, a)$$

See Slide 5 for creating approach to getting Q^{π} from episodes.

      ↳ Different ways to evaluate Q value function

        ↳ A full policy evaluation with MC takes too long

        ∴ Update policy after each episode

↳ GPI will be aggressive

↳ But we may not have full Q value function for all actions. If we update after every episode, we will not have Q for all actions

⇒ Erroneous Argmax

↳ Locked into suboptimal actions. Not giving opportunities for other actions

∴ Do not do greedy policy improvement

↳ Same as Explore vs. Exploit dilemma of Multi-Armed Bandit Problem.

↳ You want to prioritize options with good returns (exploit), but you want to try all actions (explore).

↳ Do ϵ-Greedy Policy Improvement

↳ Try all actions with non-zero probability.

↳ Probability of greedy action is assigned $1 - \epsilon$

↳ Stochastic Policy $\pi(s, a) = \begin{cases} \frac{\epsilon}{|A|} + 1 - \epsilon & \text{for greedy action} \\ \frac{\epsilon}{|A|} & \text{for non-greedy action} \end{cases}$

↳ ϵ is dynamic (more to come in week 7)

- <u>ϵ-Greedy Improves policy π:</u>
  - Theorem on slide 7. (No textbook chapter)

Apply $B^{\pi'}$ repeatedly starting at V^{π} to get $V^{\pi'}$

↑ Bellman policy operator

Goal: Show by induction that for all i, $V^{\pi'} \geq V^{\pi}$

↳ See proof on Slide 8

↳ Base case is the hardest to evaluate.

* Key Step: Maximum action Q is greater or equal to weighted average of Q over all actions.

↳ Induction Step relies on monotonicity property of Bellman operators

- Greedy in the Limit for Infinite Exploration (GLIE):

  - All State - action pairs are explored infinitely many times

$$\therefore \lim_{k \to \infty} N_k(s,a) = \infty$$

($N_k(s,a)$: Count of State - action pair)

  - The policy obtained by ϵ - greedy policy converges to absolute greedy policy

$$\lim_{k \to \infty} \pi_k(s,a) = \Pi_{a = \operatorname{argmax}_{b \in A} Q(s,b)}$$

- ϵ - greedy can be made GLIE if ϵ is reduced as $\epsilon_k = \frac{1}{k}$

- GLIE Tabular MC Control: See Slide 11

  ↳ At the end of each episode, reduce $\epsilon \leftarrow \frac{1}{k}$ and adjust policy $\pi \leftarrow \epsilon\text{-greedy}(Q)$

- Theorem: GLIE Tabular MC Policy converges on optimal policy

- Assignment: Implement GLIE Tabular MC Policy Improvement

- **TD Control:**
  - ϵ-greedy policy updates automatically after each atomic experience
  - Focus on updating Q after atomic experience and π can be automatically updated.
  - Tabular SARSA Algorithm
    - "State, action, reward, next state, next action"
  - See slide 13 for update algorithm
  - A_t generated from S_t based on ϵ-greedy policy
  - Unlike MC Control, these experiences are incrementally generated
    - ↳ Action requires a live experience policy.
  - **Theorem:** Tabular SARSA converges under conditions listed on slide 15.
  - Tabular n-step SARSA looks n steps ahead.
  - Tabular SARSA(λ) is basically TD(λ) used for control
    - Eligibility traces for state-action pairs

- **Off-Policy Learning:** Estimate VF for target policy π while following experience policy μ

$$\{S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T\} \sim \mu$$

  - Why is this important:
    - ↳ Learn improved policy by observing other agents

↳ Reuse experiences from old policies

↳ Learn about optimal policies while following exploratory policy

↳ Learn multiple policies by following one policy

- One way to do off-policy sampling is <u>Importance Sampling</u>

  ↳ See slide 20

    ↳ Expectation of $f(x)$ over P = Expectation of $\frac{Q}{P} f(x)$ over Q

      ↳ Check this on slide 20

  ↳ See slide 21 for importance sampling of off-policy MC

    ↳ Ratio of probability for every step of trace

    ↳ Biggest problem is very high variance

      ∴ Not at all useful, but we can use the same idea for TD

- Importance Sampling for off-policy TD:

  Weight TD target with importance sampling

- <u>Q-Learning</u>:

  - Off-policy learning of action-values $Q(S, A)$, but no importance sampling is required.
  - Train experience generated from experience policy μ
  - But consider alternative successor action $A' \sim \pi(S_t, \cdot)$

    ↑ Target policy

    ↳ See update rule slide 23

- Standard Q-learning Algorithm:
  - Allow both behavior and target policies to improve
  - Experience policy μ is exploratory / stochastic
  - Target policy is 100% greedy
  - See slide 24

Q-learning simplified to:

$$R_{t+1} + \gamma \max_{a' \in A} Q(S_{t+1}, a')$$

↳ See update rule for adjustments after each atomic experience

↳ Similar to SARSA, but you keep track of two policies

- So far we have learned tabular control RL Algorithms.
  - ↳ These algorithms generalize to function approximation
  - ↳ See Q-value update at the end of every policy on slide 26
  - ↳ See slide 27 for update after every atomic experience using SARSA

- Slide 28 gives a good summary of DP vs. TD algorithms

- Slide 30: ✓ means convergence, ✗ means not
  - "Deadly Triad": off policy, Bootstrapping, and function approximation does not work together in RL
    - ↳ Google "Deadly Triad in RL"
    - ↳ RL is about using two of the three tools.

- Slide 33: (ε) means goes close to optimal value, then bounces off.