- **Policy Gradient Methods**
  - Adjust the policy directly using policy gradient, rather than calculating VF
  - **Recall:** Two types of value function

    $V^{\pi}(s)$

    $Q^{\pi}(s, a)$

    } Ground truth approximated by function approximation

    ↳ Policy not represented explicitly, rather actions are selected on Q
  - This time, we parameterize policy

    ↳ We control parameters that affect distribution of actions specified by π

    $\pi_\theta(s, a) = P[a \mid s, \theta]$
  - This lecture will focus on model-free RL

    ↳ This lets us scale to large, complicated MDPs
  - How do we use gradient ascent to improve policy
  - **Policy vs. Value-based Methods**

    ↳ Policy-based method is often more efficient when policy is simpler to represent than value function

    ↳ Policy may be more compact than VF

    ↳ Policy-based methods tend to be more stable, more likely to converge (guaranteed to converge at least to local optima)

    * Policy gradient methods effective over high-dimensional or continuous action space since it does not require calculation of max over every step.

    ↳ Policy-based methods can learn stochastic methods.

    ↳ **Disadvantages:** Typically converge to local rather than global optimum

    Sometimes VF is more compact than Policy.

- Sometimes you want a Stochastic policy
  - ↳ Deterministic policies can be exploited (e.g. Rock, Paper, Scissors)
  - ↳ Stochastic policy works better when Markov Property breaks down (or partially observed)
    - ↳ For aliased states (indistinguishable states at different locations), a stochastic policy can be better than a deterministic policy
    - ↳ Partially observed ⇔ State aliasing ⇔ Not MP

- Policy Objective Function:
  - Goal: Given policy $\pi_\theta(s,a)$ with parameters θ, find best θ
    - ↳ How do we measure quality of π_θ?
      1. In episodic case, maximize value function of start state
      2. Average value function over all states
      3. Average reward per timestep

      → Policy gradient is the same for all.
  - Policy Optimization: Find θ that maximizes objective $J(\theta)$
    - ↳ Some approaches that do not use gradients:
      1. Hill Climbing
      2. Simplex
      3. Genetic Algorithms
    - ↳ We will focus on gradient descent
      - ↳ This method exploits sequential structure present through trajectory
  - Finite Difference Policy Gradient
    - ↳ look for local max of objective $J(\theta)$ by moving up gradient

      $\Delta\theta = \alpha \nabla_\theta J(\theta)$

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

↳ Naive, Numerical approach

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

(u_k: unit vector with 1 in k^{th} dimension, 0 elsewhere)

↳ Independently perturb $J(\theta)$ in each dimension

↳ Useful when $J(\theta)$ is not known or not differentiable

↳ Does not work for large dimensions

- Analytical Policy Gradient: Compute policy gradient analytically

↳ Assume we know $\nabla_\theta \pi_\theta(s,a)$

↳ Likelihood ratio exploits the likelihood ratio

$$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$

$$= \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)$$

Score function is $\nabla_\theta \log \pi_\theta(s,a)$

↳ Score function for softmax policy (linear)

↳ Weight actions using linear combination of features: $\phi(s,a)^T \theta$

↳ Probability of action proportional to exponential weight:

$$\pi_\theta(s,a) \propto \exp\left[\phi(s,a)^T \theta\right]$$

∴ Score function given by:

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}\left[\phi(s,\cdot)\right]$$

Feature of taken action ↑ Average feature over all actions ↑

↳ If feature occurs more than usual and gets more reward, want to increase frequency of that feature

↳ If action selected from a Gaussian Policy, score function is:

$$\nabla_\theta \log \pi_\theta(s,a) = \frac{(a - \mu(s))\,\phi(s)}{\sigma^2}$$

- One-Step MDPs:
  - start state: $s \sim d(s)$
  - Take one time step with $r = R_{s,a}$
  - Use likelihood ratio to get policy gradient
    - ↳ See slide for derivation

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\underbrace{\nabla_\theta \log \pi_\theta(s,a)}_{\text{Likelihood}}\, r\right] \quad \text{(r: Sample reward)}$$

- Policy Gradient Theorem:
  - ↳ For differentiable policy $\pi_\theta(s,a)$, for any objective function J, policy gradient is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, Q^{\pi_\theta}(s,a)\right]$$

- Monte Carlo Policy Gradient (REINFORCE)

$$\Delta\theta_t = \alpha\, \nabla_\theta \log \pi_\theta(s_t, a_t)\, v_t$$

↳ Update after each episode

- Policy-based reinforcement learning yields smooth error terms

↳ Policy-based MC is very slow.

- **Actor-Critic Methods**: Reducing variance from MC using a critic

↳ Instead of using a return to estimate VF, we estimate action value function using critic

$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$

↑ Critic-based action-value function using function approx

↳ Critic updates weights w in each step

$J_\theta(\theta)$ in slides.

↳ Critic's $Q_w(s,a)$ is a policy evaluation method.

- **Practical Example**: Action-Value Actor-Critic

↳ Update Critic using TD(0) to improve w (which affects Q)

↳ Update Actor using policy gradient to improve θ (which affects policy)

- Pseudo-code in slide

Initialize s, θ
Sample $a \sim \pi_\theta$
for each step do:

Sample reward $r = R_s^a$; sample transition $s' \sim P_s^a$

...

We're using model free approach. I'm assuming these come from environment

- Actor picking actions, critic evaluating things, actor moves in direction of critic

- Trick to improve performance:

  → Subtract baseline to reduce variance

$$\mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, B(s)\right] = 0$$

($B(s)$: Baseline Function)

  → Because this is zero, we can add or subtract it to Q to control variance

  → Use baseline as the State-Value Function

Advantage Function is Action-Value function − State value function

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$

(Using baseline reduces variance)

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, A^{\pi_\theta}(s,a)\right]$$

- Estimating Advantage Function Using Critic:

① Have critic learn both Q and V using function approx with separate sets of parameters

② (Better idea) TD Error is a sample of advantage function

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

↳ TD error is an unbiased sample of advantage function

$$\mathbb{E}\left[\delta^{\pi_\theta} \mid s,a\right] = \dots$$

$$= A^{\pi_\theta}(s,a)$$

↳ Because TD error is an estimate of advantage function, you could estimate policy gradient using TD error

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, \delta^{\pi_\theta}\right]$$

↳ In practice, you can estimate TD error by only estimating V in critic, not Q.
∴ One set of parameters

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

- Critics at different timescales using eligibility traces (Actor-Critic using TD(λ))

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$e_t = \gamma \lambda e_{t-1} + \phi(s_t)$$

$$\Delta\theta = \alpha\, \delta_t\, e_t$$

* Continue from 1:18:50