10-703 Fall 2025
Recitation 2

MDPs, Policy Gradient

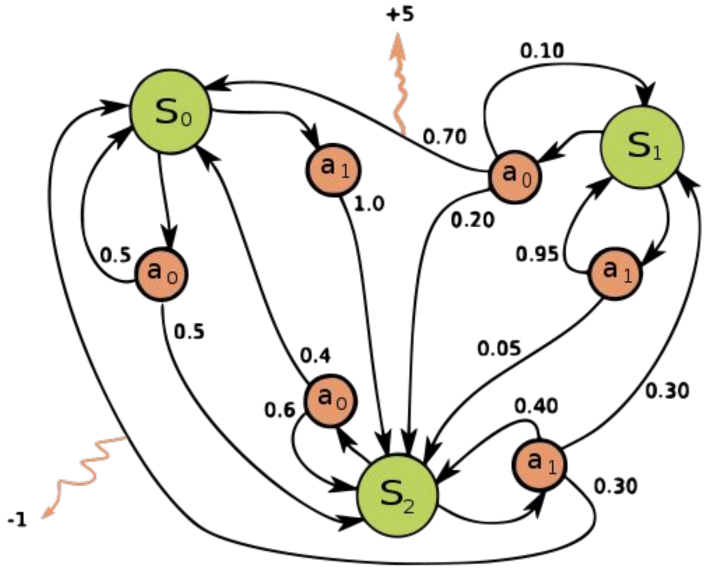
Vansh Kapoor

Markov Decision Process (MDP)

- Simple way to represent the idea of “state”
  - The actions you take will result in new places with new reward/transition functions
- Represented oftentimes by 5 main items
  - State Space (S)
  - Action Space (A)
  - Transition Function ($T: S \times A \times S \rightarrow$ Real)
    - Represents $P(s'|s, a)$
  - Reward Function ($R: S \times A \rightarrow$ Real)
  - Discount Factor (γ)
- The above formulation can also represent
  - Reward functions of the form:

    $R : S \times A \times S \rightarrow$ Real, i.e., $R(s, a, s')$

MDP Example



+5
0.10
S0
S1
a1
a0
0.70
1.0
0.20
0.5
a0
0.95
a1
0.5
0.4
0.05
a0
0.6
0.40
0.30
a1
S2
0.30
-1

Markov Decision Process (MDP)

- Markov Property
  - The idea of state encapsulating all past information (trajectory)

$$\mathbb{P}\left[R_{t+1} = r, S_{t+1} = s' \,|\, S_0, A_0, R_1, \ldots, A_{t-1}, R_t, S_t, A_t\right] = \mathbb{P}\left[R_{t+1} = r, S_{t+1} = s' \,|\, S_t, A_t\right]$$

<table>
<tr><th colspan="2" rowspan="2">Markov Models</th><th colspan="2">Do we have control over the state transitions?</th></tr>
<tr><th>NO</th><th>YES</th></tr>
<tr><th rowspan="2">Are the states completely observable?</th><th>YES</th><td>Markov Chain</td><td>MDP
Markov Decision Process</td></tr>
<tr><th>NO</th><td>HMM
Hidden Markov Model</td><td>POMDP
Partially Observable Markov Decision Process</td></tr>
</table>

MDP - What can we do?

- Goal - Learn policy $\pi: S \rightarrow A$
  - Maximize discounted reward
- Discounting rewards
  - Deals with infinite trajectories
  - Incentivizes early action
- Discount factor γ

MDP - Learning the Policy

- Value function (V: S → Real)
  - Represents the estimated value of being in a state while following policy
- Q-value (Q: SxA → Real)
  - Represents the estimated value of being in a state and taking an action, then following policy

Where does the REINFORCE/A2C equation come from?

Derivatives of the policy objective

$$\max_{\theta} . \; U(\theta) = \boxed{\mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right]}$$

We want to find the policy that maximizes expected reward when rolled out

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right] \\
&= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau) \\
&= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau) \\
&= \sum_\tau P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} R(\tau) \\
&= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau) \\
&= \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\nabla_\theta \log P_\theta(\tau) R(\tau) \right]
\end{aligned}$$

Derivatives of the policy objective

$$\max_{\theta} . \; U(\theta) = \boxed{\mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right]}$$

We want to find the policy that maximizes expected reward of generated trajectories

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right] \\
&= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau) \\
&= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau) \\
&= \sum_\tau P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} R(\tau) \\
&= \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau) \\
&= \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\nabla_\theta \log P_\theta(\tau) R(\tau) \right]
\end{aligned}$$

Derivatives of the policy objective

$$\max_{\theta} . \ U(\theta) = \boxed{\mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right]}$$

We want to find the policy that maximizes expected reward of generated trajectories

$$\boxed{\nabla_\theta U(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[R(\tau) \right]}$$

Let's find the gradient, and run gradient *Ascent*!

$$= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau)$$

$$= \sum_\tau \nabla_\theta P_\theta(\tau) R(\tau)$$

$$= \sum_\tau P_\theta(\tau) \frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)} R(\tau)$$

$$= \boxed{\sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) R(\tau)}$$

d/dx(log f(x)) = f'(x) / f(x)

$$= \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\nabla_\theta \log P_\theta(\tau) R(\tau) \right]$$

Policy Gradient Theorem

$$\text{(REINFORCE)} : \nabla_\theta U(\theta) = \mathbb{E}_{\tau\sim P_\theta(\tau)}\left(\sum_{h=0}^{H-1} \nabla_\theta \log(\pi_\theta(a_h|s_h)) \cdot R(\tau)\right).$$

$$\text{(Q-Version)} : \nabla_\theta U(\theta) = \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{h=0}^{H-1} \nabla_\theta \log(\pi_\theta(a_h|s_h)) Q_h^{\pi_\theta}(s_h, a_h)\right].$$

Proof:

$$\begin{aligned}
&\sum_{h=0}^{H-1} \nabla \log \pi_\theta(a_h|s_h) \cdot \left(\sum_{h=0}^{H-1} r_h\right) \\
&= \sum_{h=0}^{H-1} \nabla \log \pi_\theta(a_h|s_h) \cdot \left(\sum_{h=0}^{H-1} r_h\right) \\
&= \sum_{h=0}^{H-1} \nabla \log \pi_\theta(a_h|s_h) \cdot \left(\underbrace{\sum_{h'=0}^{h-1} r_{h'}}_{A} + \underbrace{\sum_{h'=h}^{H-1} r_{h'}}_{B}\right).
\end{aligned}$$

The term A is unaffected by a_h; when we take the expectation outside, the term A evaluates to zero. Term B in expectation is simply $Q_h^{\pi_\theta}(s_h, a_h)$.

Policy Gradient Theorem

$$\text{(REINFORCE)} : \nabla_\theta U(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left(\sum_{h=0}^{H-1} \nabla_\theta \log(\pi_\theta(a_h|s_h)) \cdot R(\tau) \right).$$

$$\text{(Q-Version)} : \nabla_\theta U(\theta) = \mathbb{E}_{\tau \sim P_\theta} \left[\sum_{h=0}^{H-1} \nabla_\theta \log(\pi_\theta(a_h|s_h)) Q_h^{\pi_\theta}(s_h, a_h) \right].$$

However, since we are in a learning (RL) setup, we **cannot calculate this expectation** explicitly.
We therefore use Monte Carlo estimation:

1. We first roll out N trajectories

$$\{\tau_i = (s_0^{(i)}, a_0^{(i)}, r_0^{(i)}, \cdots, s_{H-1}^{(i)}, a_{H-1}^{(i)}, r_{H-1}^{(i)}, s_H^{(i)})\}_{i \in [N]}$$

   following the policy π_θ.

2. The gradient estimate for REINFORCE is given by

$$\nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{h} \nabla_\theta \log \pi_\theta(a_h^{(i)}|s_h^{(i)}) \cdot R(\tau_i).$$

3. Similarly, the gradient estimate for the Q-version is given by

$$\nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{h} \nabla_\theta \log \pi_\theta(a_h^{(i)}|s_h^{(i)}) \cdot \hat{Q}_h^{(i)},$$

   where $\hat{Q}_h^{(i)} = \sum_{h'=h}^{H-1} r_{h'}^{(i)}$.

However, the Monte-Carlo sampling method often results in a gradient estimate with high variance.

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

For any function $b(s)$ that only depends on the state, we have

$$\begin{aligned}
\mathbb{E}_{a\sim\pi(\cdot|s)}\left[b(s)\cdot\nabla_\theta \log\pi(a\mid s)\right] &= \sum_a b(s)\cdot\pi(a\mid s)\cdot\frac{1}{\pi(a\mid s)}\nabla_\theta\pi(a\mid s)\\
&= b(s)\sum_a \nabla_\theta\pi_\theta(a\mid s)\\
&= b(s)\,\nabla_\theta\sum_a \pi_\theta(a\mid s)\\
&= 0.
\end{aligned}$$

The Advantage version is obtained by subtracting the **baseline** $\mathbf{b_h(s) = V_h(s_h)}$ (independent of a_h):

$$\text{(Advantage-Version)}\quad \nabla_\theta U(\theta) = \mathbb{E}_{\tau\sim P_\theta(\tau)}\left[\sum_{h=0}^{H-1} A_h^{\pi_\theta}(s_h,a_h)\,\nabla_\theta\log\pi(a_h\mid s_h)\right],$$

where

$$A_h^{\pi_\theta}(s_h,a_h) = Q_h^{\pi_\theta}(s_h,a_h) - V_h^{\pi_\theta}(s_h).$$

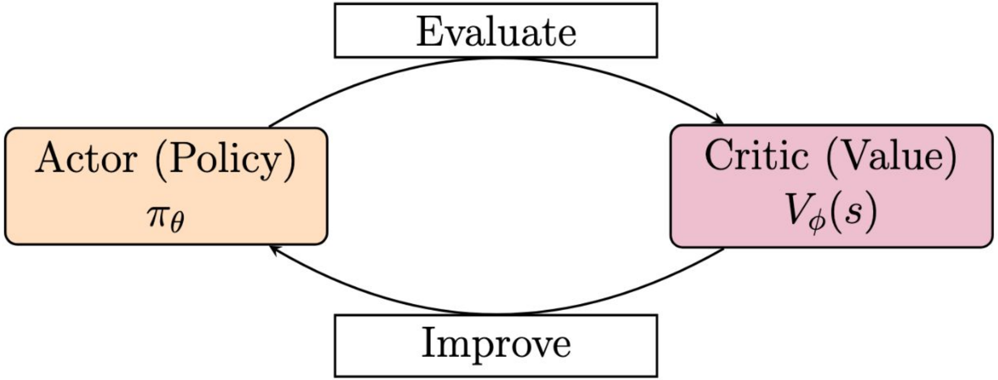
REINFORCE with Baseline (episodic), for estimating $\pi_{\theta} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ $\qquad (G_t)$
        $\delta \leftarrow \boxed{G - \hat{v}(S_t,\mathbf{w})}$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

To reduce variance, we replace G with G - v(S_t, w) (this is called the advantage function)
Note: Actions achieving a higher score than the value function have a positive advantage, while actions with a lower score than the value function have a negative advantage. (and adding a baseline doesn't change expected gradient !)

Advantage Actor-Critic with TD(0)



We define the actor as π_θ and the critic as $V_\phi(s)$. The Advantage Actor-Critic, estimates the gradient as,

$$\nabla_\theta U(\theta) \approx \sum_{h=0}^{H-1} \nabla_\theta \log \pi(a_h \mid s_h) \cdot \left(\underbrace{r_h + V_\phi(s_{h+1}) - V_\phi(s_h)}_{\text{Temporal Difference Error}} \right),$$

where the **T**emporal **D**ifference (TD) error is an estimate of $A_h^{\pi_\theta}(s_h, a_h)$.

Advantage Actor-Critic

0. Initialize policy parameters θ and critic parameters ϕ .

1. Sample trajectories $\{\tau_i = \{s_t^i, a_t^i\}_{t=0}^T\}$ by deploying the current policy $\pi_\theta(a_t \mid s_t)$.

2. Fit value function $V_\phi^\pi(s)$ by MC or TD estimation (update ϕ)

3. Compute action advantages $A^\pi(s_t^i, a_t^i) = R(s_t^i, a_t^i) + \gamma V_\phi^\pi(s_{t+1}^i) - V_\phi^\pi(s_t^i)$

4. $\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\alpha_t^i \mid s_t^i) A^\pi(s_t^i, a_t^i)$

5. $\theta \leftarrow \theta + \alpha \nabla_\theta U(\theta)$

Some Questions to Think About

1) Can we use policy gradient methods to solve the planning problem in MDPs, i.e., to find the optimal policy?

2) What could be the possible drawbacks?

   (Hint: Think about the drawbacks of gradient descent.)

A2C/Reinforce/Reinforce with Baseline

Actor outputs probability of picking each action given a state

Critic outputs the predicted value for a given state

Hint 1: Actor and Critic networks should be the same except for input dimension, output dimension, and activation (probability distribution versus value function)

```
gamma = 0.99

for i in tqdm.tqdm(range(num_seeds)):
    reward_means = []

    # TODO: create networks and setup reinforce/a2c



    for m in range(num_episodes):
        A2C_net.train(env, gamma=gamma)
        if m % 100 == 0:
```

Generate Episode: run the actor/policy, and save the list of states, actions, and rewards generated (needed for training)

Evaluate Policy: run the actor/policy, and save the undiscounted sum of rewards/length of trajectory (needed for plot creation/testing)

Train: you should have different cases for reinforce, reinforce with baseline, and A2C

Define the losses for each of these and run training (optimizer.zero_grad(), loss.backward(), optimizer.step() etc)

Run: Within each episode, call env.reset() and step through for max_steps. If terminated, then break and move on to the next episode (make sure to update the states, actions and rewards you've seen in that episode)

```
def __init__(self, actor, actor_lr, N, nA, critic, critic_lr, bas
    # Note: baseline is true if we use reinforce with baseline
    #       a2c is true if we use a2c else reinforce
    # TODO: Initializes A2C.
    self.type = None  # Pick one of: "A2C", "Baseline", "Reinforc
    assert self.type is not None
    pass

def evaluate_policy(self, env):
    # TODO: Compute Accumulative trajectory reward(set a trajecto
    pass

def generate_episode(self, env, render=False):
    # Generates an episode by executing the current policy in the
    # Returns:
    # - a list of states, indexed by time step
    # - a list of actions, indexed by time step
    # - a list of rewards, indexed by time step
    # TODO: Implement this method.
    pass

def train(self, env, gamma=0.99, n=10):
    # Trains the model on a single episode using REINFORCE or A2C
    # TODO: Implement this method. It may be helpful to call the
    #       method generate_episode() to generate training data.
    pass
```

Questions?

References

Some of the slides are heavily inspired by the notes from the class

17-740: Algorithmic Foundations of Interactive Learning,

and some are directly taken from Sutton & Barto’s Reinforcement Learning: An Introduction