AI 533 **Intelligent Agents and Decision Making** Winter 2025

OREGON STATE UNIVERSITY
School of Electrical Engineering and Computer Science
Instructor: Sandhya Saisubramanian
Name: Hyuntaek Oh
Email: ohhyun@oregonstate.edu

Total points: 100 Assignment 2: Monte Carlo, TD methods, and Actor-Critic Due date: March 6, 2025

**Instructions**: Collaboration is not allowed on any part of this assignment. It is acceptable to discuss concepts or clarify questions but you must document who you worked with for this assignment. Copying answers or reusing solutions from individuals/the internet is unacceptable and will be dealt with strictly. Solutions must be typed (hand written and scanned submissions will not be accepted) and saved as a .pdf file.

1. **(10 points)** After applying SARSA, can we estimate $V^\pi$ upon its termination, based on the information available to the agent? If yes, write the equation to estimate $V^\pi$. If no, explain why.

   *We can estimate $V^\pi(s)$ upon its termination.*

   *SARSA approximates the Bellman equation by using sampled transitions rather than computing full expectations, meaning that it samples one transition $(s', a')$ at a time and one action $a'$ from policy $\pi$.*

   *The state-value function $V^\pi(s)$ is defined as the expected return when starting in state $s$ and following policy $\pi$:*

   $$V^\pi(s) = E_\pi[G_t|S_t = s], where\ G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ...$$

   *Expanding the expectation:*

   $$V^\pi(s) = E_\pi[R_t + \gamma G_{t+1}|S_t = s]$$

   *The expectation conditioned on taking action $a$ defines the action-value function $Q^\pi(s, a)$:*

   $$Q^\pi(s, a) = E[R_t + \gamma V^\pi(S_{t+1})|S_t = s, A_t = a]$$

   *By expanding the expected return recursively, it would be:*

   $$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} T(s, a, s')[R(s, a) + \gamma V^\pi(s')]$$

   *Since $V^\pi(s)$ is derived from $Q^\pi(s, a)$, $V^\pi(s)$ in stochastic policy can be computed as:*

   $$V^\pi(s) = \sum_a \pi(s, a) Q^\pi(s, a)$$

   *Or if the policy is deterministic, meaning $\pi(s, a) = 1$, $V^\pi(s)$ would be:*

   $$V^\pi(s) = Q^\pi(s, \pi(s))$$

   *Thus, the equation above follows directly from the Bellman equation and is the correct way to estimate $V^\pi(s)$ after SARSA has learned $Q^\pi(s, a)$.*

2. **(10 points)** Does SARSA update equation change when we modify the reward notations ($R(s)$, $R(s,a)$, or $R(s,a,s')$). Why or why not?

*No, different reward notations do not change SARSA update equation, meaning that SARSA works correctly regardless of whether the reward is state-based, action-based, or transition-based since reward itself is just a value. The general form of SARSA update equation is:*

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)],$$

*where $R_t$ is the reward received after taking action $a_t$ in state $s_t$.*

*If state-based reward $R(s_t)$ is used, the equation is:*

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

*This equation does not change the original SARSA update equation, but the agent cannot learn which actions lead to better outcomes since $R(s_t)$ does not depend on the action $a_t$.*

*If action-based reward $R(s_t, a_t)$ is used, the equation is:*

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

*It is better than $R(s_t)$ and does not change SARSA update equation as well because SARSA can distinguish between different actions taken in the same state.*

*If transition-based reward $R(s_t, a_t, s_{t+1})$, the equation is:*

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R(s_t, a_t, s_{t+1}) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

*The equation above does not change the equation. Moreover, the agent can learn not just which actions are good but also how different resulting states affect the outcome.*

*Therefore, only the specific reward value $R_t$ changes depending on the notation used, and the SARSA update equation remains structurally the same.*

3. **(10 points)** Policy iteration algorithm consists of a policy evaluation phase and a policy improvement phase. If we replace dynamic programming policy evaluation with First Visit Monte Carlo prediction for policy evaluation, will policy iteration converge to an optimal policy in a finite number of iterations? Why or why not? Assume the agent has access to $T$ and $R$.

*Policy iteration with First-visit Monte Carlo for policy evaluation does not necessarily guarantee convergence to the optimal in a finite number of iterations. There are some problems when replacing with first-visit Monte Carlo algorithm: relying on sample-based estimates, stochasticity in returns, and exploration issues.*

*In dynamic programming policy evaluation, the Bellman equation is solved iteratively, ensuring exact values of $V^{\pi}(s)$ after a finite number of iterations. Unlike the dynamic programming policy evaluation, first-visit Monte Carlo relies on sample-based estimation, which is to estimate $V^{\pi}(s)$ by averaging returns from sampled episodes. This needs a large number of episodes to produce accurate estimation.*

*Due to the stochasticity, the return values have variance and may not be accurate in a finite number of iterations. Since each episode samples a different trajectory, the individual values of $G_i$ will fluctuate, leading to variance in the estimated $V(s)$. This makes it difficult to determine when to stop policy iteration.*

*Furthermore, if certain states are not frequently visited under the current policy, it is hard to estimate their values well, probably leading to suboptimal policy improvements.*

4. **(20 points)** Consider actor-critic methods in tabular format. (i) We will replace the TD-critic module with that of an Oracle critic: the critic has an MDP model (unknown to the actor) and can calculate $V(s)$ accurately using the Bellman equation. How does this affect the actor updates and convergence of the algorithm? (ii) We will now replace the TD-critic module with that of a lazy critic: it updates the values $V(s)$ in every other iteration (instead of every iteration). How does this affect the actor updates and convergence of the algorithm?

*In the architecture of Actor-critic (AC), actor updates the policy based on the value function, which is derived from the estimation of the critic, and critic estimates the value function $V(s)$ or action-value function $Q(s,a)$, which helps the actor improve. According to the lecture note, the critic updates its estimate of the state-value function using temporal difference (TD) error:*

$$\delta_t = r_t + \hat{V}_w(s_{t+1}) - \hat{V}_w(s_t), \text{ where } \gamma = 1$$

$$w \leftarrow w + \beta \cdot \delta_t \cdot \nabla_w \hat{V}_w(s_t), \text{ where } w \text{ is value function parameter}$$

*Also, the actor's update rule using TD is:*

$$\theta \leftarrow \theta + \alpha \cdot \delta_t \cdot \nabla_\theta log\pi_\theta(s_t, a_t), \text{ where } \theta \text{ is policy parameter}$$

*As can be seen above, the value function in TD error is in both update rules, implying the importance of value function.*

*(i) The impact of Oracle critic on actor updates and convergence*

*The oracle critic has an MDP model and can compute the exact value function $V(s)$ accurately using the Bellman equation. If the critic is oracle-level, the Bellman equation would be:*

$$V^*(s) = E[r_t + \gamma V^*(s_{t+1})]$$

*Based on the equation, the TD error becomes:*

$$\delta_t^{oracle} = r_t + \gamma V^*(s_{t+1}) - V^*(s_t)$$

*Due to the exact values of $V(s)$, the policy gradient estimates in the actor update rule will be more accurate (the gradient estimate becomes deterministic), and the policy can improve more stable with significantly low variance. This is based on reliable value estimates that the actor optimizes its updates.*

*The convergence would be faster and more stable since the oracle critic provides the exact $V(s)$. Unlike TD-learning, which can be affected by sampling errors, oracle critic has no bias in value function estimation. Based on the Policy Gradient Theorem in the lecture note, the actor updates are effectively guided by the oracle critic's exact values, so the policy gradient update would be:*

$$\nabla_\theta \rho(\theta) = E_{\pi_\theta}[\delta_t^{oracle} \nabla_\theta log\pi(S_t, A_t)]$$

$$= E_{\pi_\theta}[(r_t + \gamma V^*(S_{t+1}) - V^*(S_t)) \nabla_\theta log\pi(S_t, A_t)]$$

*Thus, oracle critic brings out faster, more stable convergence compared to traditional TD-learning (But it is not practical for real-world where the MDP model is unavailable).*

*(ii) The impact of Lazy critic on actor updates and convergence*

*Unlike Oracle critic, Lazy critic only updates every other iteration (updates after one step instead of every step). For example, every even-numbered iteration $t = 0, 2, 4, ...$ updates $V(s)$ whereas every odd-numbered iterations $t = 1, 3, 5, ...$ does not update $V(s)$, and the previous estimate is used for it.*

*The update rule of the lazy value function is:*

$$V_{lazy}(s_t) = V(s_{t-k}), \text{ where } k \text{ is the delay in updates}$$

*Based on the equation above, the lazy TD error would be:*

$$\delta_t^{lazy} = r_t + \gamma V_{lazy}(s_t) - V_{lazy}(s_{t-1})$$

*Since the Lazy critic does not update at every step, the actor relies on outdated value function estimates. This delayed feedback causes the actor's gradient updates to learn inaccurate or outdated value estimates. Lazy actor update rule is:*

$$\theta \leftarrow \theta + \alpha \cdot \delta_t^{lazy} \cdot \nabla_\theta log \pi_\theta(s_t, a_t)$$

$$\theta \leftarrow \theta + \alpha \cdot (r_t + \gamma V_{lazy}(s_t) - V_{lazy}(s_{t-1})) \cdot \nabla_\theta log \pi_\theta(s_t, a_t)$$

*The key difference is that Lazy critic uses $V(s_{t-1})$ instead of $V(s_t)$, meaning that actor will update based on stale information. Due to this slow learning, the policy updates may become less responsive to recent experience, which can slow down learning.*

*For convergence, the learning lags behind reality, making convergence slower. The delay in updating the value function can cause the policy to oscillate since it learns from previous estimate, not current one. Moreover, the policy gradient is not pointing in the exact correct direction, leading to higher bias.*

*Thus, Lazy critic delays policy updates, causing slow learning, more oscillations, and higher bias.*

5. **(25 points)** Consider the following episodes of an MDP with five states and five actions. Each entry in the episode includes the state id, action id, and the corresponding reward observed. Discount factor $\gamma = 0.9$. (i) For each episode, calculate the returns of each state using *first-visit Monte Carlo* algorithm and calculate the value of each state $V(s)$ using both the episodes. (ii) For each episode, calculate the returns of each state using e*very-visit Monte Carlo* algorithm and calculate the value of each state $V(s)$ using both the episodes. Report your values as a table to facilitate faster grading.

   - Episode 1: $\{(s_1, a_1, 1), (s_1, a_1, 5), (s_2, a_4, 8), (s_3, a_5, 5), (s_4, a_1, 10)\}$
   - Episode 2: $\{(s_1, a_1, 5), (s_2, a_4, -1), (s_3, a_5, -2), (s_5, a_2, 1), (s_4, a_1, 10)\}$

   *(i) Returns and V(s) of each state using first-visit Monte Carlo algorithm*

| state | Episode 1 returns | Episode 2 returns |
|---|---|---|
| $s_1$ | $1 + (0.9) * 5 + (0.9)^2 * 8 + (0.9)^3 * 5 + (0.9)^4 * 10$ $= 22.186$ | $5 + (0.9) * (-1) + (0.9)^2 * (-2) + (0.9)^3 * 1 + (0.9)^4 * 10$ $= 9.77$ |
| $s_2$ | $8 + (0.9) * 5 + (0.9)^2 * 10$ $= 20.6$ | $-1 + (0.9) * (-2) + (0.9)^2 * 1 + (0.9)^3 * 10$ $= 5.3$ |
| $s_3$ | $5 + (0.9) * 10$ $= 14$ | $-2 + (0.9) * 1 + (0.9)^2 * 10$ $= 7$ |
| $s_4$ | $10 + (0.9) * 0$ $= 10$ | $10 + (0.9) * 0$ $= 10$ |
| $s_5$ | None | $1 + (0.9) * 10$ $= 10$ |

Table 1: Returns for each state (first-visit Monte Carlo algorithm)

| state | V(s) |
|---|---|
| $s_1$ | $\frac{22.186+9.77}{2} = 15.987$ |
| $s_2$ | $\frac{20.6+5.3}{2} = 12.95$ |
| $s_3$ | $\frac{14+7}{2} = 10.5$ |
| $s_4$ | $\frac{10+10}{2} = 10$ |
| $s_5$ | $\frac{10}{1} = 10$ |

Table 2: V(s) for each state (first-visit Monte Carlo algorithm)

*(ii) Returns and V(s) of each state using every-visit Monte Carlo algorithm*

| state | Episode 1 returns | Episode 2 returns |
|---|---|---|
| $s_1$ | $1+(0.9)*5+(0.9)^2*8+(0.9)^3*5+(0.9)^4*10$ $=22.186$ $5+(0.9)*8+(0.9)^2*5+(0.9)^3*10$ $=23.54$ | $5+(0.9)*(-1)+(0.9)^2*(-2)+(0.9)^3*1+(0.9)^4*10$ $=9.77$ |
| $s_2$ | $8+(0.9)*5+(0.9)^2*10$ $=20.6$ | $-1+(0.9)*(-2)+(0.9)^2*1+(0.9)^3*10$ $=5.3$ |
| $s_3$ | $5+(0.9)*10$ $=14$ | $-2+(0.9)*1+(0.9)^2*10$ $=7$ |
| $s_4$ | $10+(0.9)*0$ $=10$ | $10+(0.9)*0$ $=10$ |
| $s_5$ | None | $1+(0.9)*10$ $=10$ |

Table 3: Returns for each state (every-visit Monte Carlo algorithm)

| state | V(s) |
|---|---|
| $s_1$ | $\frac{22.186+23.54+9.77}{3}=18.499$ |
| $s_2$ | $\frac{20.6+5.3}{2}=12.95$ |
| $s_3$ | $\frac{14+7}{2}=10.5$ |
| $s_4$ | $\frac{10+10}{2}=10$ |
| $s_5$ | $\frac{10}{1}=10$ |

Table 4: V(s) for each state (every-visit Monte Carlo algorithm)

6. **(25 points)** Consider two MDPs $\mathcal{M} = \langle S, A, T, R \rangle$ and $\mathcal{M}' = \langle S, A, T, R' \rangle$. The MDPs have the same state space, action space, transition functions, and discuount factor $\gamma$ but differ in their reward functions such that $R'(s, a, s') = R(s, a, s') + F(s, a, s')$ where $F(s, a, s')$ is a bonus reward that could help speed up the learning process (also referred to as reward shaping). In our case, $F(s, a, s') = \gamma\phi(s') - \phi(s)$ for some arbitrary function $\phi : S \to \mathbb{R}$.

   Consider running tabular Q-learning in each MDP $\mathcal{M}$ and $\mathcal{M}'$, with initial values $Q^0_{\mathcal{M}}(s, a) = q_{\text{init}} + \phi(s)$ and $Q^0_{\mathcal{M}'}(s, a) = q_{\text{init}}$, $\forall(s, a) \in S \times A$ and $q_{\text{init}} \in \mathbb{R}$. At any moment in time, the current Q-value of any state-action pair is always equal to its initial value plus some $\Delta$ value denoting the total change in the Q-value across all updates:

$$Q_{\mathcal{M}}(s, a) = Q^0_{\mathcal{M}}(s, a) + \Delta Q_{\mathcal{M}}(s, a) \tag{1}$$
$$Q_{\mathcal{M}'}(s, a) = Q^0_{\mathcal{M}'}(s, a) + \Delta Q_{\mathcal{M}'}(s, a) \tag{2}$$

   **Show that if $\Delta Q_{\mathcal{M}}(s, a) = \Delta Q_{\mathcal{M}'}(s, a)$ for all $(s, a) \in S \times A$, then these two Q-learning agents yield identical updates for any state-action pair**. That is, both agents will converge to policies that behave identically, and the offset $\phi(s)$ will not affect action selection.

   Hints:

   - Equations (1) and (2) indicate that $\Delta Q_{\mathcal{M}}(s, a) = \alpha_M \delta_M$ and $\Delta Q_{\mathcal{M}'}(s, a) = \alpha_{M'} \delta_{M'}$ where $\alpha_M$ and $\alpha_{M'}$ denote the learning rates or step sizes and $\delta_M$ and $\delta_{M'}$ denotes the corresponding TD-errors.
   - For ease, you can assume that $\alpha_M = \alpha_{M'}$.
   - What you then need to show is $\delta_M = \delta_{M'}$. To do so, use Equations (1) and (2) to rewrite $Q_{\mathcal{M}}(s, a)$ and $r_M$ in terms of $Q_{\mathcal{M}'}(s, a)$ and $r_{M'}$ or vice versa.

*(i) Check the relationship between $Q_{\mathcal{M}}(s, a)$ and $Q_{\mathcal{M}'}(s, a)$*

*From the given initial values, $Q^0_{\mathcal{M}}(s, a) = q_{init} + \phi(s)$ and $Q^0_{\mathcal{M}'}(s, a) = q_{init}$, we get:*

$$Q^0_{\mathcal{M}}(s, a) = Q^0_{\mathcal{M}'}(s, a) + \phi(s)$$

*Then, we need to check the relationship between $Q_{\mathcal{M}}(s, a)$ and $Q_{\mathcal{M}'}(s, a)$ remains valid after the first Q-learning update due to its generalization.*

*For ease, $\alpha_{\mathcal{M}} = \alpha_{\mathcal{M}'}$, meaning that both can be expressed as just $\alpha$, and Q-value update equations for $\mathcal{M}$ and $\mathcal{M}'$ can be expressed as:*

$$Q^1_{\mathcal{M}}(s, a) = Q^0_{\mathcal{M}}(s, a) + \alpha\delta_{\mathcal{M}}$$
$$Q^1_{\mathcal{M}'}(s, a) = Q^0_{\mathcal{M}'}(s, a) + \alpha\delta_{\mathcal{M}'}$$

*, where $\delta_{\mathcal{M}}$ is:*

$$\delta_M = r_{\mathcal{M}}(s, a, s') + \gamma \max_{a'} Q^0_{\mathcal{M}}(s', a') - Q^0_{\mathcal{M}}(s, a)$$
$$= r_{\mathcal{M}}(s, a, s') + \gamma \max_{a'}(q_{init} + \phi(s')) - (q_{init} + \phi(s))$$
$$= r_{\mathcal{M}}(s, a, s') + \gamma q_{init} + \gamma\phi(s') - q_{init} - \phi(s)$$
$$= r_{\mathcal{M}}(s, a, s') + \gamma\phi(s') - \phi(s) + (\gamma - 1)q_{init}$$

*The update Q-value, $Q^1_{\mathcal{M}}(s,a)$, is:*

$$Q^1_{\mathcal{M}}(s,a) = q_{init} + \phi(s) + \alpha[r_{\mathcal{M}}(s,a,s') + \gamma\phi(s') - \phi(s) + (\gamma-1)q_{init}]$$

*Similarly, The update Q-value, $Q^1_{\mathcal{M}'}(s,a)$, is:*

$$Q^1_{\mathcal{M}'}(s,a) = q_{init} + \alpha[r_{\mathcal{M}'}(s,a,s') + \gamma\phi(s') - \phi(s) + (\gamma-1)q_{init}]$$

*From these values, we get:*

$$Q^1_{\mathcal{M}}(s,a) = Q^1_{\mathcal{M}'}(s,a) + \phi(s)$$

*Thus, the relationship holds after the first update:*

$$Q_{\mathcal{M}}(s,a) = Q_{\mathcal{M}'}(s,a) + \phi(s)$$

*(ii) Show $\delta_M = \delta_{M'}$*

*The TD-error $\delta_M$ and $\delta_{M'}$ are:*

$$\delta_M = R(s,a,s') + \gamma \max_{a'} Q_{\mathcal{M}}(s',a') - Q_{\mathcal{M}}(s,a)$$

$$\delta_{M'} = R'(s,a,s') + \gamma \max_{a'} Q_{\mathcal{M}'}(s',a') - Q_{\mathcal{M}'}(s,a)$$

*Since $R'(s,a,s') = R(s,a,s') + \gamma\phi(s') - \phi(s)$:*

$$\delta_{M'} = R(s,a,s') + \gamma\phi(s') - \phi(s) + \gamma \max_{a'} Q_{\mathcal{M}'}(s',a') - Q_{\mathcal{M}'}(s,a)$$

*From (i), we take the next-step max Q-value:*

$$\max_{a'} Q_{\mathcal{M}}(s',a') = \max_{a'}(Q_{\mathcal{M}'}(s',a') + \phi(s'))$$

*Since $\phi(s')$ is independent of action, it simplifies to:*

$$\max_{a'} Q_{\mathcal{M}}(s',a') = \max_{a'} Q_{\mathcal{M}'}(s',a') + \phi(s')$$

*With the relationship above, we can rewrite $\delta_{M'}$ to:*

$$\begin{aligned}\delta_{M'} &= R(s,a,s') + \gamma \max_{a'} Q_{\mathcal{M}}(s',a') - \phi(s') - (Q_{\mathcal{M}}(s,a) - \phi(s)) \\ &= R(s,a,s') + \gamma \max_{a'} Q_{\mathcal{M}}(s',a') - Q_{\mathcal{M}}(s,a) \\ &= \delta_M\end{aligned}$$

*Since $\delta_{M'} = \delta_M$, and assuming $\alpha_M = \alpha_{M'}$, we get:*

$$\boldsymbol{\Delta Q_{\mathcal{M}}(s,a) = \Delta Q_{\mathcal{M}'}(s,a)}$$

*Thus, both Q-learning agents converge to the identical Q-values, and the offset $\phi(s)$ does not affect action selection.*