Multi Agent Systems

- Lab 6 -

N-Step Bootstrapping

Recap: state-value prediction

In estimating the value of a policy v_π there are two extremes:

- The ***return G_t*** of a state reward sequence is
  - $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-t-1} R_T$
  - ***G_t*** is the *target* in MC updates
  - $V^\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s] = E_\pi[G_t | S_t = s]$

- In *one-step* updates (Value-Iteration; Q-Learning, SARSA – based on TD learning) the *target* is first reward + discounted estimate for value of next state
  - $G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$

State-value prediction generalization

- The ***n*-step return $G_{t:t+n}$** of a state reward sequence is
  - $G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$
  - $V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha [G_{t:t+n} - V_{t+n-1}(S_t)], \quad 0 \leq t < T$

n-step TD for estimating $V \approx v_\pi$

Initialize $V(s)$ arbitrarily, $s \in \mathcal{S}$
Parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t and R_t) can take their index mod n

Repeat (for each episode):
    Initialize and store $S_0 \neq$ terminal
    $T \leftarrow \infty$
    For $t = 0, 1, 2, \ldots$:
    | If $t < T$, then:
    | Take an action according to $\pi(\cdot|S_t)$
    | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
    | If S_{t+1} is terminal, then $T \leftarrow t + 1$
    | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
    | If $\tau \geq 0$:
    | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
    | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
    | $V(S_\tau) \leftarrow V(S_\tau) + \alpha [G - V(S_\tau)]$
    Until $\tau = T - 1$

N-step SARSA

- Use of N-step methods for control as well, besides prediction
- N-step method + SARSA → on-policy TD control method
  - $G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^{n} Q_{t+n-1}(S_{t+n}, A_{t+n}), \quad n \geq 1, 0 \leq t < T-n$
  - $Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha [G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], \quad 0 \leq t < T$

N-step SARSA

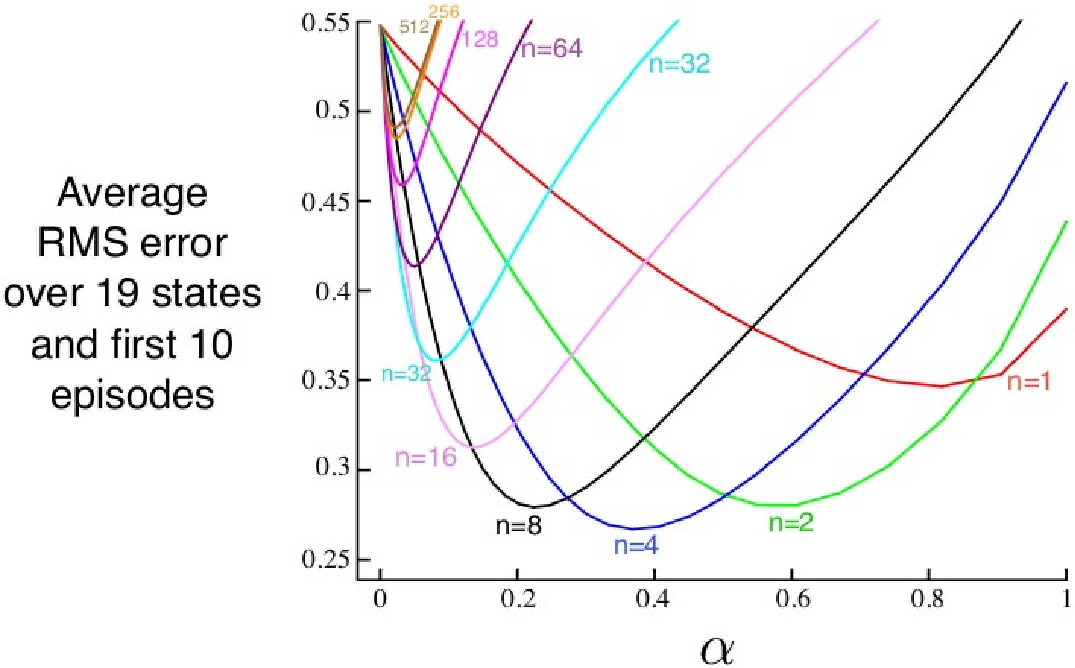
n-step Sarsa for estimating $Q \approx q_*$, or $Q \approx q_\pi$ for a given π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod n

Repeat (for each episode):
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  For $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\,[G - Q(S_\tau, A_\tau)]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

Test environment and Task

- FrozenLake-4x4 environment in OpenAI Gym
- **Task:** analyze the state-value prediction accuracy of TD(0) methods (Q-Learning, SARSA) and TD(n) methods (n-step SARSA)



Task steps

- Implement **n-step SARSA** agent for the **Frozen Lake - small (4x4)** environments
- **Run Value Iteration** on the environment to compute the *ground truth* state-value function – obtain **V*(s)**
- For **alpha = [0.0, 0.1, 0.3, 0.5, 07, 1.0]**
  - For **20 repetitions** with the chosen **alpha**
    - Initialize Q values to 0
    - Run **Q-Learning** and **SARSA** for **8000** episodes
    - Run **n-step SARSA** for 8000 epsiodes, where n = 2, 3 and 4
    - At the end of each episode compute the RMSE (root mean squared error) between V*(s) and $\max_a Q(s, a)$
  - After learning for the 8000 episodes in each repetition, average MSE over the **20 repetitions**
- **On a same graph**, plot RMSE errors for Q-Learning, SARSA and n-step SARSA (for each value of n): **x-axis = alpha values**, **y-axis = RMSE**

Task steps

- **Important Notes**
  - Run each algorithm using the following hyper-parameter setup:
    - $\gamma = 0.99$
    - **Linear decay procedure** for **ε** (in ε-greedy action selection)
      - $\varepsilon_{start} = 0.75$
      - $\varepsilon_{end} = 0.001$
      - Decay linearly from episode 0 until 0.3 x total number of episodes
      - After decay continue with ε_{end}