Chapter 10: On-policy Control with Approximation

Seungjae Ryan Lee
Episodic 1-step semi-gradient Sarsa

- Approximate action values (instead of state values)

\[ w_{t+1} = w_t + \alpha \left[ U_t - \hat{q}(S_t, A_t, w_t) \right] \nabla \hat{q}(S_t, A_t, w_t). \]

- Use Sarsa to define target

\[ w_{t+1} = w_t + \alpha \left[ R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w_t) - \hat{q}(S_t, A_t, w_t) \right] \nabla \hat{q}(S_t, A_t, w_t). \]

- Converges the same ways as TD(0) with same error bound

\[ \overline{VE}(w_{TD}) \leq \frac{1}{1 - \gamma} \min_w \overline{VE}(w). \]
Control with Episodic 1-step semi-gradient Sarsa

- Select action and improve policy using an $\varepsilon$-greedy action w.r.t. $\hat{q}(S_t, a, w_t)$

---

**Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$**

Input: a differentiable action-value function parameterization $\hat{q} : S \times A \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $w \in \mathbb{R}^d$ arbitrarily (e.g., $w = 0$)

Loop for each episode:
- $S, A \leftarrow$ initial state and action of episode (e.g., $\varepsilon$-greedy)

Loop for each step of episode:
- Take action $A$, observe $R, S'$
- If $S'$ is terminal:
  - $w \leftarrow w + \alpha [R - \hat{q}(S, A, w)] \nabla \hat{q}(S, A, w)$
  - Go to next episode
- Choose $A'$ as a function of $\hat{q}(S', \cdot, w)$ (e.g., $\varepsilon$-greedy)
  - $w \leftarrow w + \alpha [R + \gamma \hat{q}(S', A', w) - \hat{q}(S, A, w)] \nabla \hat{q}(S, A, w)$
  - $S \leftarrow S'$
  - $A \leftarrow A'$
Mountain Car Example

- Task: Drive an underpowered car up a steep mountain road
  - Gravity is stronger than car’s engine
  - Must swing back and forth to build enough inertia
- State: position $x_t$, velocity $\dot{x}_t$
- Actions: Forward (+1), Reverse (-1), No-op (0)
- Reward: -1 until the goal is reached
Approximation for Mountain Car

- *Tile coding* used to select binary features (8 tiles)

\[
\hat{q}(s, a, \mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s, a) = \sum_{i=1}^{d} w_i \cdot x_i(s, a),
\]
Results of Mountain Car

- Plot the cost-to-go function: $-\max_a \hat{q}(s, a, w)$
- Initial action values set to 0
  - Very optimistic
Results of Mountain Car

Mountain Car
Steps per episode
log scale
averaged over 100 runs

α = 0.1/8
α = 0.2/8
α = 0.5/8

Episode
Episodic n-step Semi-gradient Sarsa

- Use n-step return $G_{t:t+n}$ as the update target

\[
\mathbf{w}_{t+1} ≜ \mathbf{w}_t + \alpha \left[ R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t) \right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t).
\]

\[
\mathbf{w}_{t+n} ≜ \mathbf{w}_{t+n-1} + \alpha \left[ G_{t:t+n} - \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1}) \right] \nabla \hat{q}(S_t, A_t, \mathbf{w}_{t+n-1}),
\]
Episodic n-step Semi-gradient Sarsa in Practice

Episodic semi-gradient n-step Sarsa for estimating $\hat{q} \approx q_\pi$ or $q_\pi$

Input: a differentiable action-value function parameterization $\hat{q} : S \times A \times \mathbb{R}^d \to \mathbb{R}$
Input: a policy $\pi$ (if estimating $q_\pi$)
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$, a positive integer $n$
Initialize value-function weights $w \in \mathbb{R}^d$ arbitrarily (e.g., $w = 0$)
All store and access operations ($S_t$, $A_t$, and $R_t$) can take their index mod $n + 1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$ or $\varepsilon$-greedy wrt $\hat{q}(S_0, \cdot, w)$
  $T \leftarrow \infty$
  Loop 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 $A_{t+1} \sim \pi(\cdot|S_{t+1})$ or $\varepsilon$-greedy wrt $\hat{q}(S_{t+1}, \cdot, w)$
        $\tau \leftarrow t - n + 1$  ($\tau$ 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 $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n}, w)$
          $w \leftarrow w + \alpha \left| G - \hat{q}(S_t, A_t, w) \right| \nabla \hat{q}(S_t, A_t, w)$  $(G_{\tau:T+n})$
    Until $\tau = T - 1$
Episodic n-step Semi-gradient Sarsa Results

- Faster learning
- Better asymptotic performance

Mountain Car
Steps per episode
log scale
averaged over 100 runs
Episodic n-step Semi-gradient Sarsa Results

- Best performance for intermediate values of n-step

Mountain Car
Steps per episode averaged over first 50 episodes and 100 runs
Average Reward Setting

- Quality $r(\pi)$ of policy $\pi$ defined by the average reward following policy $\pi$
- Continuing tasks without discounting

\[
r(\pi) = \lim_{h \to \infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi] = \lim_{t \to \infty} \mathbb{E}[R_t \mid S_0, A_{0:t-1} \sim \pi],
\]

\[
= \sum_s \mu_\pi(s) \sum_a \pi(a \mid s) \sum_{s', r} p(s', r \mid s, a)r,
\]
Differential Return and Value Functions

Differential Return: differences between rewards and average reward

\[ G_t = R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \ldots \]

Differential Value Functions: Expected differential returns

\[ v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \]
\[ q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \]
Bellman Equations

- Remove all $\gamma$
- Replace rewards with difference of rewards

\[
v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(s', r|s,a) \left[ r - r(\pi) + v_\pi(s') \right],
\]

\[
q_\pi(s, a) = \sum_{r,s'} p(s', r|s,a) \left[ r - r(\pi) + \sum_{a'} \pi(a'|s') q_\pi(s', a') \right],
\]

\[
v_*(s) = \max_a \sum_{r,s'} p(s', r|s,a) \left[ r - \max_\pi r(\pi) + v_*(s') \right], \quad \text{and}
\]

\[
q_*(s, a) = \sum_{r,s'} p(s', r|s,a) \left[ r - \max_\pi r(\pi) + \max_{a'} q_*(s', a') \right]
\]
Differential semi-gradient Sarsa

- Same update rule: $w_{t+1} = w_t + \alpha \delta_t \nabla \hat{q}(S_t, A_t, w_t)$, with differential TD error
- Original TD error:
  \[ \delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w_t) - \hat{q}(S_t, A_t, w_t) \]
- Differential TD error:
  \[ \delta_t = R_{t+1} - \bar{R}_{t+1} + \hat{q}(S_{t+1}, A_{t+1}, w_t) - \hat{q}(S_t, A_t, w_t) \]
Differential semi-gradient Sarsa

Differential semi-gradient Sarsa for estimating $\hat{q} \approx q^*$

Input: a differentiable action-value function parameterization $\hat{q} : S \times A \times \mathbb{R}^d \to \mathbb{R}$

Algorithm parameters: step sizes $\alpha, \beta > 0$

Initialize value-function weights $w \in \mathbb{R}^d$ arbitrarily (e.g., $w = 0$)

Initialize average reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)

Initialize state $S$, and action $A$

Loop for each step:
- Take action $A$, observe $R, S'$
- Choose $A'$ as a function of $\hat{q}(S', \cdot, w)$ (e.g., $\varepsilon$-greedy)
- $\delta \leftarrow R - \bar{R} + \hat{q}(S', A', w) - \hat{q}(S, A, w)$
- $\bar{R} \leftarrow \bar{R} + \beta \delta$
- $w \leftarrow w + \alpha \delta \nabla \hat{q}(S, A, w)$
- $S \leftarrow S'$
- $A \leftarrow A'$
Access-Control Queuing Example

- Agent can grant access to 10 servers
  - Agent can accept or reject customers
- Customers arrive at a single queue
  - Customers have 4 different priorities, randomly distributed
  - Pay a reward of 1, 2, 4, or 8 when granted access to a server
- A busy server is freed with some probability
Access-Control Queuing Results

- Tabular solution with differential semi-gradient Sarsa
n-step Semi-gradient Sarsa

- Use n-step return
  \[
  G_{t:t+n} = R_{t+1} - \bar{R}_{t+1} + R_{t+2} - \bar{R}_{t+2} + \ldots + R_{t+n} - \bar{R}_{t+n} + \hat{q}(S_{t+n}, A_{t+n}, w_{t+n-1})
  \]

  \[
  \delta_t = R_{t+1} - \bar{R}_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w_t) - \hat{q}(S_t, A_t, w_t)
  \]

  \[
  G_{t:t+1} = \hat{q}(S_t, A_t, w)
  \]

  \[
  \delta_t = G_{t:t+n} - \hat{q}(S_t, A_t, w)
  \]
Thank you!

Original content from

- Reinforcement Learning: An Introduction by Sutton and Barto

You can find more content in

- github.com/seungjaeryanlee
- www.endtoend.ai