Reinforcement Learning: Chapter 7 - n-step Bootstrapping

Based on Sutton and Barto's Book

Flávio Codeço Coelho

September 25, 2024

Outline

Introduction

- Overview of n-step Bootstrapping
- Importance in the context of Reinforcement Learning
- Main objectives and learning outcomes

n-step Bootstrapping

- Balances between sampling and bootstrapping
- Extends one-step TD prediction to use multiple steps
- n-step return balances immediate reward and future value
- Evaluates the policy over multiple steps

n-step TD Prediction

- Extends one-step TD prediction to use multiple steps
- Balance between bootstrapping and sampling
- Mathematical formulation:

$$V(s) \leftarrow V(s) + \alpha[G_t^{(n)} - V(s)]$$

- $(G_t^{(n)})$: n-step return

n-step Return

- Definition of n-step return:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

- Balances immediate reward and future value

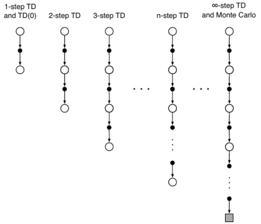

Figure 7.1: The backup diagrams of n-step methods. These methods form a spectrum ranging from one-step TD methods to Monte Carlo methods.

Full return, truncated after n steps and then corrected for the remaining missing terms by $V_{t+n-1}(S_{t+n})$

n-step TD Prediction Algorithm

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

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

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop 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: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

n-step SARSA

- Extends TD prediction to control using an on-policy method
- SARSA: State-Action-Reward-State-Action
- Update rule:

$$Q_{t+n}(s, a) = Q_{t+n-1}(s, a) + \alpha \left[G_{t:t+n} - Q_{t+n-1}(s, a) \right]$$

n-step Return for SARSA

- Definition of n-step return for SARSA:

$$
\begin{aligned}
G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \dots \\
+ \gamma^{n-1} R_{t+n} + \gamma^{n} Q_{t+n-1}(S_{t+n}, A_{t+n})
\end{aligned}
$$

n-step SARSA Algorithm

n-step Sarsa for estimating $Q \approx q_*$ or q_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
Algorithm 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 $n+1$

Loop for each episode:
    Initialize and store $S_0 \neq$ terminal
    Select and store an action $A_0 \sim \pi(\cdot|S_0)$
    $T \leftarrow \infty$
    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 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 $\tau = T - 1$

n-step Off-Policy Learning

- Learn from a target policy different from the behavior policy
- Importance sampling needed to correct for distribution mismatch
- Update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \rho_t \left[G_t^{(n)} - Q(s, a) \right]$$

- (ρ_t): Importance sampling ratio

n-step Off-Policy Learning Algorithm

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations (for 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 $A_0 \sim b(\cdot|S_0)$
    $T \leftarrow \infty$
    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 an action $A_{t+1} \sim b(\cdot|S_{t+1})$
    | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
    | If $\tau \geq 0$:
    | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1, T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ $(\rho_{\tau+1:t+n-1})$
    | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} R_i$
    | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
    | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\,[G - Q(S_\tau, A_\tau)]$
    | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
    Until $\tau = T - 1$

Per-decision Methods with Control Variates

- Reduce variance of updates using control variates
- Per-decision updates enhance learning stability
- Control variates help in adjusting the n-step returns

n-step Tree Backup Algorithm

- Avoids the use of importance sampling ratios
- Constructs updates considering multiple future steps
- Tree backup algorithm uses:

$$G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n \sum_a \pi(a|S_{t+n}) Q(S_{t+n}, a)$$

A Unifying Algorithm: n-step Q(σ)

- Combines aspects of both n-step SARSA and n-step Tree Backup methods
- The parameter σ controls the degree of sampling vs. bootstrapping
- Update rule integrates both on-policy and off-policy learning:

$$G_t^{(n)} = (1-\sigma)\sum_{k=0}^{n-1}\gamma^k R_{t+k+1} + \sigma\sum_{k=0}^{n-1}\gamma^k Q(S_{t+k}, A_{t+k})$$

- The (σ)-weighted combination adjusts the algorithm's behavior
- Generalizes both the SARSA()-forward view and the Tree Backup algorithm