Reinforcement Learning HW6

Hadi Askari

24th May 2023

1 Description

The code takes approximately 30 seconds to run and plots the required functions for the 1000 random walk problem. Their are 1000 states, and the person can take a step anywhere left a 100 steps or anywhere right a 100 steps. Each episode begins at the center state 500. The reward for reaching the first state is -1, the reward for reaching the last state is 1 and all other rewards are 0. The algorithms required were:

1.1 Policy Evaluation

This method was used to generate the True Values. This is coded in the get_true_value() function. Policy Evaluation solves the following equation:

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$$

where the probability of each action is 1/200. We kept on updating the true values till the change was below a certain threshold.

1.2 Gradient Monte Carlo Algorithm

This was coded in the get_monte_carlo() function and implements the following algorithm:

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t, \mathbf{w})\big]\nabla\hat{v}(S_t, \mathbf{w})$

(a) Gradient Monte Carlo Algorithm

This function estimates the true state values by computing weights for groups of states collected by playing out an episode.

The algorithm plays outs episodes and estimates the true state values by computing weight vectors for groups of states (we have groups of 100 so 10 weight vectors). The final reward helps back propagate the states that led to the reward being achieved.

1.3 Semi-Gradient TD(0)

This was coded in the get_td0() function and implements the following algorithm:

The critical difference here was that the true state values are estimated by updates to the weight vectors during the episodes themselves as opposed to at the end of the episodes.

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

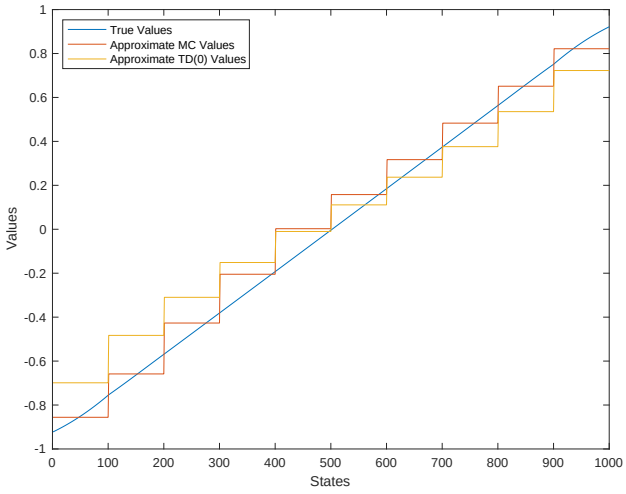
Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S is terminal

(a) Semi-Gradient TD(0)

2 Results

The following are the reproduced results as required by the assignment. They follow the pattern in the book. MC is closer to the true values than TD.



(a)