Programming Assignment 3: CS 747

Richeek Das : 190260036

2nd November 2021

Contents

1 `Task 1: "Tabular" Sarsa` **2**

2 `Task 2: Sarsa with Linear Function Approximation` **3**

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

2021-2022

1 Task 1: "Tabular" Sarsa

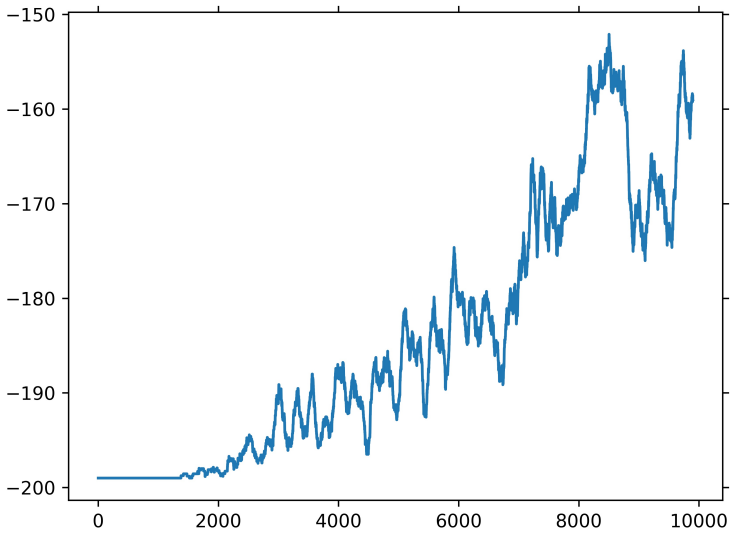

Figure 1: Tabular Sarsa(0) with $\epsilon = 0.08$ and $\eta_{lr} = 0.18$, i.e. it explores with ϵ probability and exploits with $1 - \epsilon$. In this case to discretize the state space we divide the entire range of positions and velocities into 50 uniform partitions. We return a **one-hot encoding** of the enabled partition index in the `get_table_features()` function.

Observations: The idea of discretization in Tabular Sarsa(0) is pretty naive. Dividing the state space into non-intersecting blocks doesn't carry a lot of information. It takes around **164s** to converge and we receive a mean test reward of $\mathbf{-141.22}$ over 100 episodes. We also found this strategy to be very sensitive to the hyper-parameters ϵ and η_{lr}.

2 Task 2: Sarsa with Linear Function Approximation

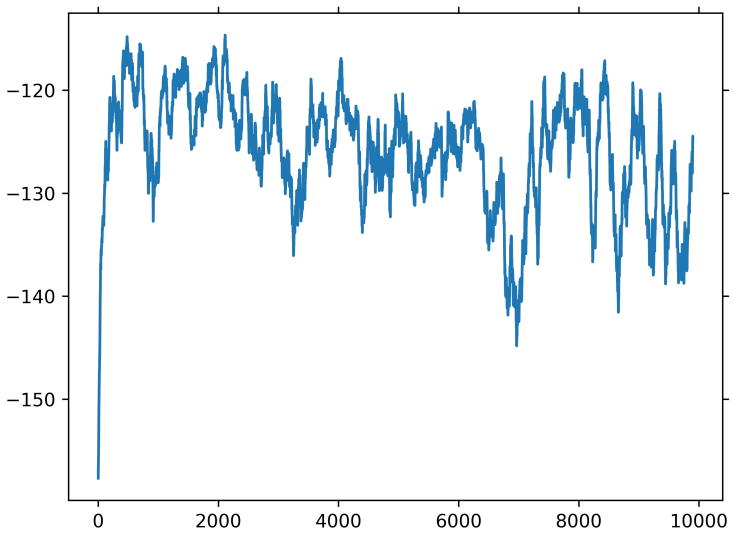

Figure 2: Sarsa(0) linear function approximation with RBF kernel. $\epsilon = 0.05$ and $\eta_{lr} = 0.18$. In this case we discretize the space with the same idea as before, i.e. 20 uniform partitions of the entire range of position and velocity. In this case instead of returning a one-hot encoding of the enabled partition index, we apply an **RBF** kernel on it. It is intuitive to provide some importance to the neighbouring $(\mathbf{x}, \mathbf{v})$ blocks (since we may want to assume, there won't be any drastic changes if the position or velocity of the car is changed slightly!). This sort of gives us a smoothened version of the one-hot encoding, which we return from the `get_better_features()` function.

Say from a one-hot encoding like this:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

To this:

$$\begin{bmatrix} 0.08 & 0.37 & 0.61 \\ 0.14 & 0.61 & 1 \\ 0.08 & 0.37 & 0.61 \end{bmatrix} \longrightarrow \begin{bmatrix} 0.08 & 0.37 & 0.61 & 0.14 & 0.61 & 1 & 0.08 & 0.37 & 0.61 \end{bmatrix}$$

Observations: This idea generalises very well for this problem setting! Our idea that there won't be any drastic changes if the position or velocity of the car is changed slightly, fits well in this setting. Upon careful vectorization of the RBF kernel operation, we observe faster convergence (particularly because of faster termination of the episodic task) than Tabular Sarsa(0) at **141s** with a mean test reward of $\mathbf{-104.03}$ over 100 episodes. If we take a look at the graph, its easy to observe that even below **1000** training iterations this method achieves much better results than Tabular Sarsa(0) at 10000 iterations.