CS-747

Foundations of Intelligent and Learning Agents

Adarsh Raj - 190050004

Assignment-3

November 2021

Task - 1

Tabular SARSA

Implementation:

For task 1, I have added few instances in the constructor itself, which are `self.states_T1`, with value of (18×29). This is for the discretisation of the continuous domain, where x is divided into 18 bins and v is divided into 29 bins. Also, `self.actions = 3` was added for the number of actions available.

- In `get_table_features()`, the bin value for `obs`, i.e observation (x, v) and accordingly indexed in a numpy array of zeros and then that value was set to 1, to result in one hot features array of size (18*29).
- In `choose_action()`, epsilon greedy with respect to the Q values is done. If the random sampled value is less than epsilon, then the action is chosen randomly, else action is chosen using `argmax` from Q values.

  Assumption: Since, we are using `argmax`, the tie-breaking rule is assumed as the first value for which $Q(s, a)$ is maximum.
- In `sarsa_update()`, for the given action, the weights are updated according to the `Sarsa(0)` update rule, and the updated weights are returned.

Initialization:

Weights for task 1 were initialised as numpy array of zeros of size `(num_actions, 18*29)`, and for different values of epsilon and learning rate, the agent was trained. The best reward value observed was for the case `learning rate = 0.9` and `epsilon = 0`. According to the book by Sutton and Barto, we can take the value of epsilon close to 0 to minimize exploration and optimize training. Using this configuration, we got the average reward value `-145.25` for test part of the task.

Two of the multiple training plots are added with different `epsilon` (0, `1e-6`)and `learning rate` values (0.9, 0.8) respectively.

Comments: Upon increasing epsilon, it was observed that the reward value in plot fluctuates for as we indirectly increase the exploration. Also, on increasing learning rate, the weights are updated more heavily, hence, the optimal training was observed with low epsilon and high learning rate (0 and 0.9 respectively).

The below two plots are the example of what I have stated above, in the second plot epsilon is `1e-6`, hence more fluctuations are observed, also learning rate is 0.8 which is less than first plot ($lr = 0.9$). The average reward observed for the second case on testing is `-151.65`.

- `Epsilon = 0` and `Lr = 0.9`
  Average Reward on Testing = -145.25

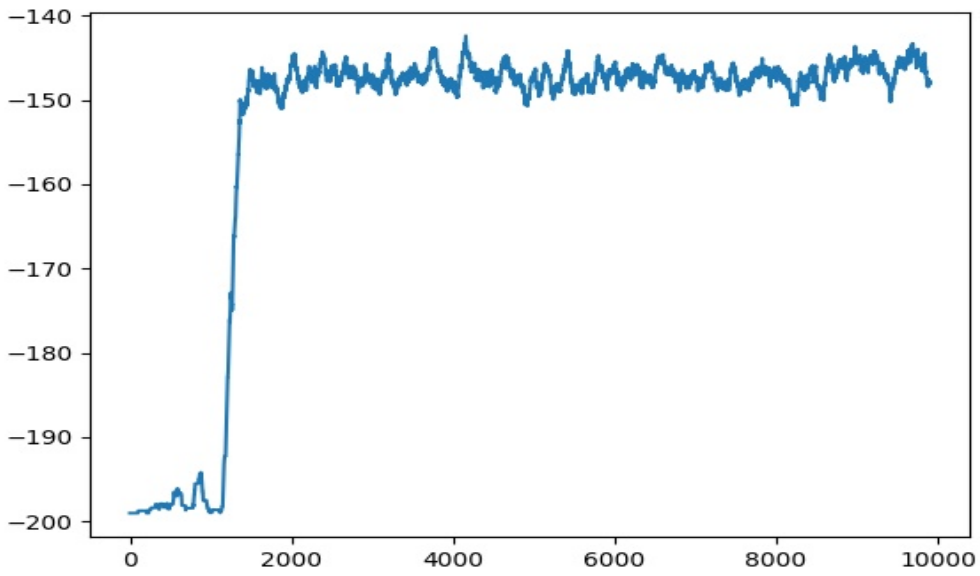

Figure 1: Task 1 Training Plot for `epsilon = 0 and lr = 0.9`

- `Epsilon = 1e-6` and `Lr = 0.8`
  Average Reward on Testing = -151.65

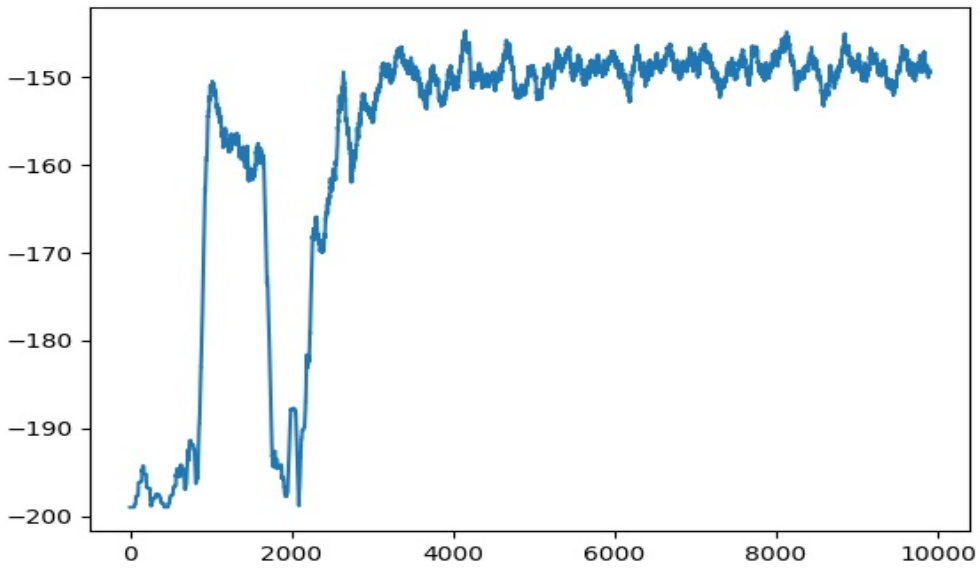

Figure 2: Task 1 Training Plot for `epsilon = 1e-6 and lr = 0.8`

Task - 2

SARSA with Linear Func Approximation

Implementation:

Approach used: Tile Coding

For task 2, I have added an instance in the constructor itself, which is `self.numTilings`, with value of 8. This is for the number of tilings. Also the same discretisation of the continuous domain is done here as in task 1, where x is divided into 18 bins and v is divided into 29 bins for each tile.

In function `get_better_features()`, a feature vector of size `numTilings*18*29`, was initialised, so that there is no need to change the other functions due to change in dimensionality of feature vector. Now, for each tile except first, offset was added to the bins and then the indices for (x, v) were calculated accordingly. The offsets were set as `(0.1/numTilings)` and `(0.005/numTilings)` for x and v, respectively. The values corresponding to the indices were all made one, for each tile and the resulting feature vector was returned.

Note: The functions `choose_action()` and `sarsa_update()` are same for task 2. Only difference is the size of weights, different initialisations, and the function for generating feature vector.

Initialization:

Weights for task 2 for each actions were initialised as numpy array of zeros of size `(num_actions, numTilings*18*29)`, and for different values of epsilon and learning rate, the agent was trained. The best reward value observed was for the case `learning rate = 0.1` and `epsilon = 0`. Using this configuration, we got the average reward value `-108.03` for test part of the task.

Comments: Two plots are added below for `(epsilon, lr)` values `(0, 0.1)` and `(1e-5, 0.1)`. Same trend is observed here as well, as on increasing the epsilon (exploration), reward value fluctuates on a few points more than the first run and overall the hyper params perform slightly poorly than the first run.

- `Epsilon = 0` and `Lr = 0.1`
  Average Reward on Testing = -108.03

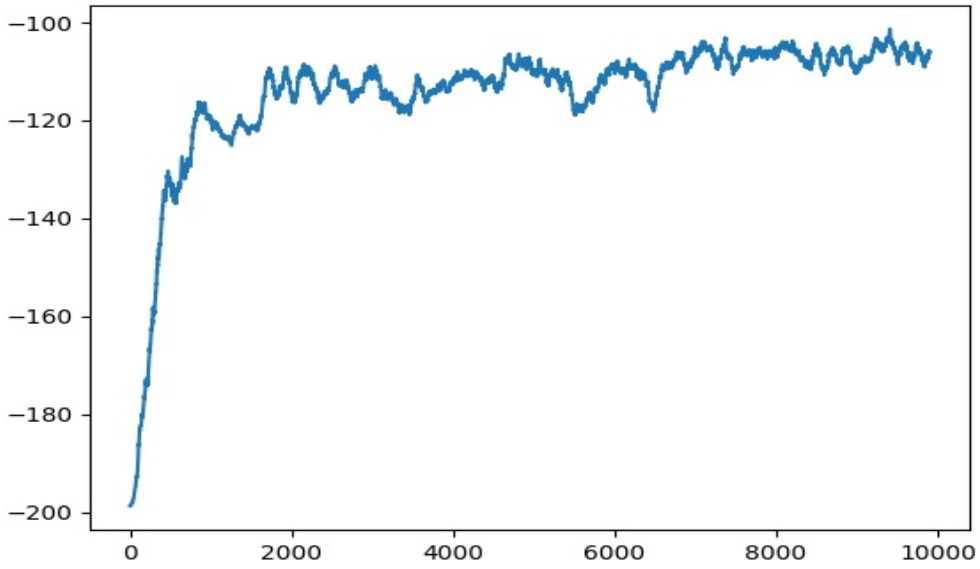

Figure 3: Task 1 Training Plot for `epsilon = 0 and lr = 0.1`

- `Epsilon = 1e-5` and `Lr = 0.1`
  Average Reward on Testing = -113.5

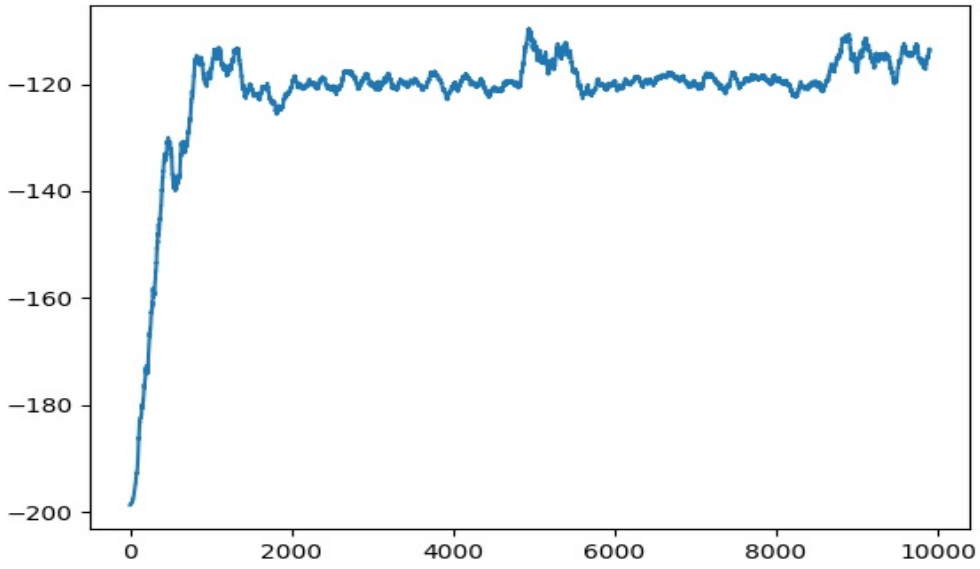

Figure 4: Task 1 Training Plot for `epsilon = 1e-5 and lr = 0.1`

Submission

- For Task 1, the plots and final weights added in submission directory corresponds to parameters `epsilon = 0 and lr = 0.9`.
- For Task 2, the plots and final weights added in submission directory corresponds to parameters `epsilon = 0 and lr = 0.1`.

Issues while Experimenting

- For Task 1, for high values of epsilon there was much exploration that the update rule can't keep up and hence, the resulting reward values were less than -160. Finding the right parameters was difficult, due to the run time of train (which is approx 5 minutes on my system).
- For Task 2, same issue as task 1, although an additional issue was finding the right number of tilings and value of offsets, which provide good weights.