Assignement: ex7

Ian Steenstra

November 2024

1 Problem 1

- Instead of using bootstrapped estimates (like n-step Sarsa or TD(0)), they would use the complete return from each episode, G_t, as the target: $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[G_t - \hat{q}(S_t, A_t, \mathbf{w}_t)]\nabla\hat{q}(S_t, A_t, \mathbf{w}_t)$
- Since they already established the framework for semi-gradient Sarsa, including function approximation, we can easily modify it to use Monte Carlo returns. The only change needed is in the target of the update rule (as shown above).
- Mountain Car episodes, particularly initially, tend to be quite long. This poses a problem for Monte Carlo methods because their reliance on complete returns introduces significant variance. Sarsa, on the other hand, bootstraps its estimates, allowing it to learn from shorter sequences and generally leading to more stable and efficient learning in this environment.

2 Problem 2

- **(a):** Pseudocode for semi-gradient one-step Expected Sarsa for control
  - **Input:** Differentiable action-value function parameterization $\hat{q}(s, a, \mathbf{w})$
  - **Input:** Policy $\pi(a|s)$ (if estimating q_π)
  - **Input:** Step size $\alpha > 0$, small $\epsilon > 0$
  - Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
  - **For each episode:**
    - Initialize S
    - **While** S is not terminal:
    - Choose $A \sim \pi(\cdot|S)$ or ϵ-greedy w.r.t. $\hat{q}(S, \cdot, \mathbf{w})$
    - Take action A, observe R, S'
    - **If** S' is terminal: $G \leftarrow R$
    - **Else**: Calculate expected value of next action: $E \leftarrow \sum_{a'} \pi(a'|S')\hat{q}(S', a', \mathbf{w})$ $G \leftarrow R + \gamma E$

* $\mathbf{w} \leftarrow \mathbf{w} + \alpha[G - \hat{q}(S, A, \mathbf{w})]\nabla\hat{q}(S, A, \mathbf{w})$
* $S \leftarrow S'$

- **(b):** The essential difference between Expected Sarsa (part (a)) and Q-learning lies in how the target value (G) is computed. While Expected Sarsa uses the expected value under the target policy, Q-learning would use the maximum Q-value over all possible next actions.

3 Problem 3

- **(a):** See code and Figure 1.

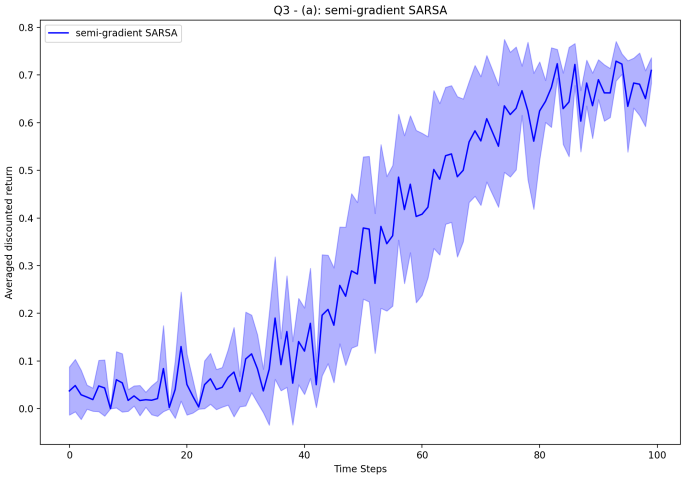

Figure 1: Problem 3(a): Semi-Gradient SARSA

- **(b):** See code and Figure 2.
  - **Is aggregating the states helping to boost the learning speed? Shall we always aggregate the states as much as possible to learn better policy?** Aggregating states can indeed speed up learning, but only up to a point. The 2x2 tiling approach seems to strike a good balance, generalizing effectively without oversimplifying. However, the room-based aggregation is too simplistic; it throws away too much information about the environment's structure, hindering learning.
- **(c):** N/A

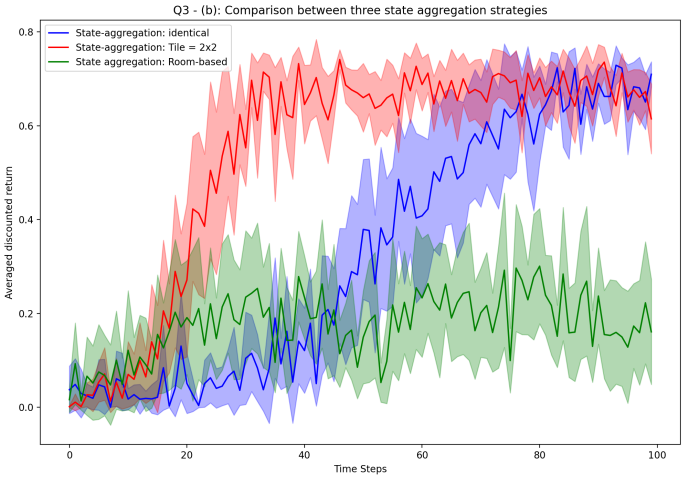

Figure 2: Problem 3(b): Comparison between three state aggregation strategies

- **(d):** See code and Figure 3. **Why is the constant feature necessary? What do you think happens without it? Also describe how you incorporated actions into your features.** The bias term (the constant feature usually set to 1) is essential. It provides an offset, allowing my model to learn a baseline Q-value independent of the state or action. This is crucial because a zero state does not necessarily imply a zero Q-value; there might be potential for future rewards even from a seemingly neutral starting point. Activations are incorporated into the feature vector by creating separate features for each possible action. This is equivalent to having independent sets of weights for each action. So, for a state (x, y), the feature vector would have components specific to each action—for instance, [x, y, 1] for action 1, another [x, y, 1] for action 2, and so on. When evaluating or updating the Q-value for a particular action, only the weights corresponding to that action's features are used. This action-specific feature representation allows the model to learn different Q-values for each action in each state. Compared to state aggregation, linear function approximation with carefully designed features offers more flexibility. State aggregation can be too simplistic, potentially overlooking important distinctions between similar states. Feature-based approaches, on the other hand, can capture finer-grained differences in the state-action space, leading to more accurate Q-value estimates and more effective learning. **How do your results with the above features compare with state aggregation? If there is a significant performance difference, try to come up with explanations for it.** My initial results with linear function approximation were poorer than

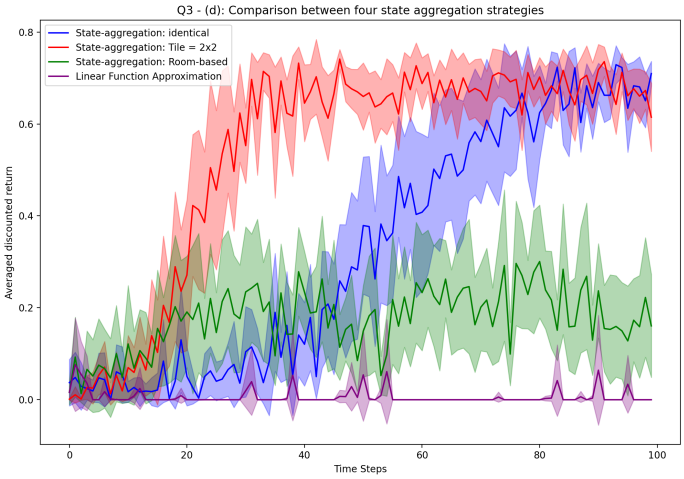

Figure 3: Problem 3(d): Comparison between four state aggregation strategies

those with state aggregation. This is likely due to the need to perform a more careful tuning of the learning rate (alpha) with linear function approximation. The increased number of weights and the more complex interaction between features can make the learning process more sensitive to the learning rate. I experimented with lower values of alpha and I expect that further tuning could lead to performance comparable to or better than state aggregation. It is also possible that my initial feature choices were not optimal; as I know that feature engineering is often crucial for linear function approximation and machine learning in general.

- **(e):** See code and Figure 4. **Comment on your findings, including any trends and surprising results.** Both the normalized state coordinates ([x/10, y/10, 1]) and distance-to-goal features ([d, x, y, 1]) significantly improved performance compared to the simpler features in part (d). Both sets of features led to a much faster initial increase in returns. However, normalized coordinates ultimately yielded better average returns after 100 episodes, contrary to my expectation that distance would be the stronger signal. This suggests that scaling input features can be more effective than explicitly providing distance information; normalized features may enable the agent to learn more nuanced state-action relationships without overemphasizing distance. The distance-based features also showed greater instability in later episodes, possibly indicating sensitivity to learning rate or exploration parameters that could benefit from further tuning.

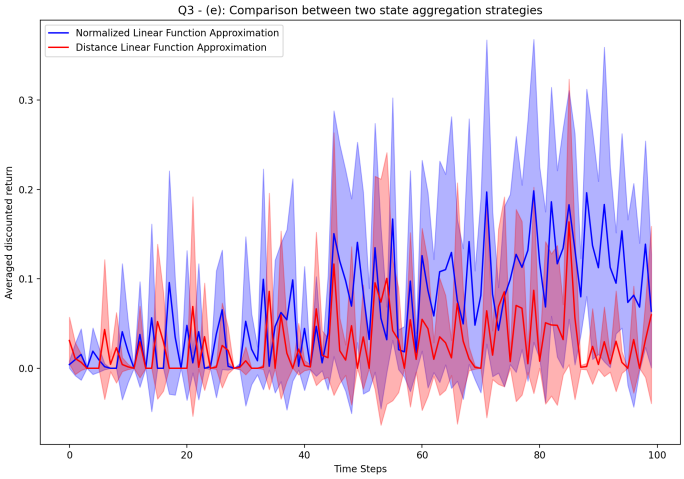

Figure 4: Problem 3(e): Comparison between two state aggregation strategies

4 Problem 4

See code and Figures 5 and 6.

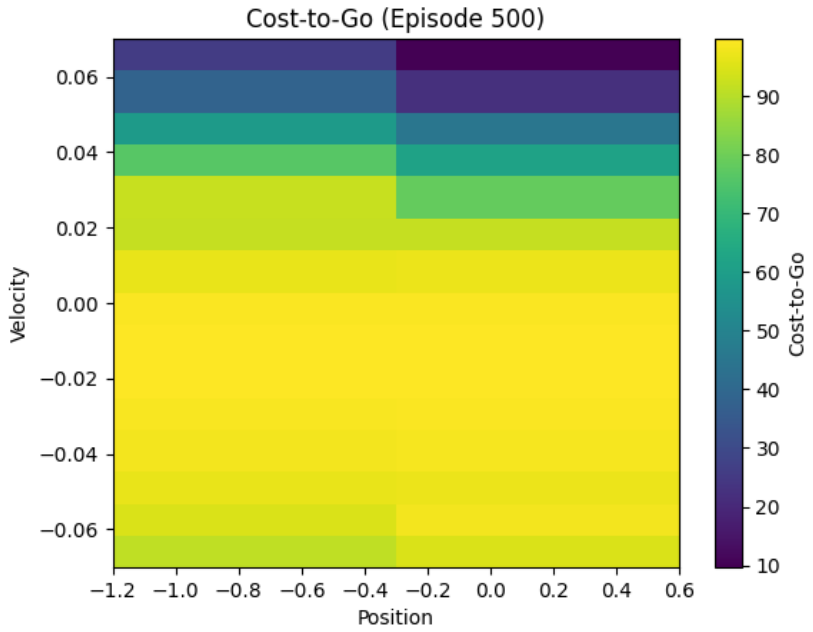

Figure 5: Problem 4: Cost-to-Go (Episode 500)

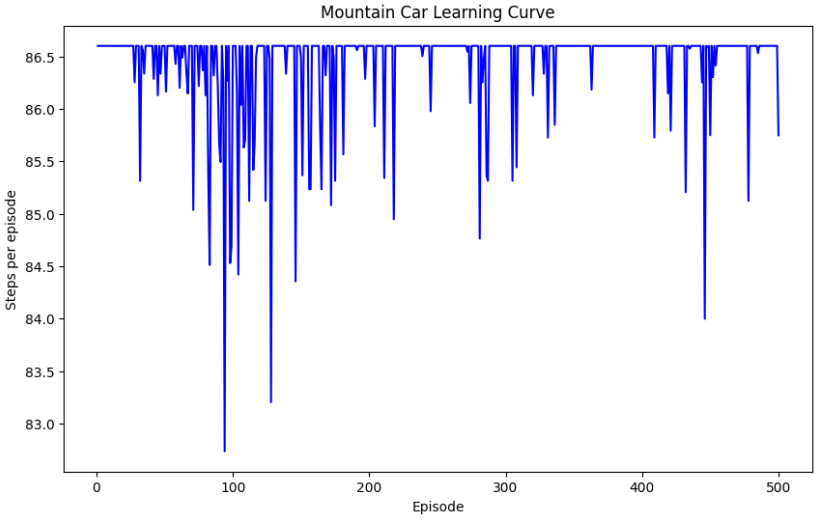

Figure 6: Problem 4: Mountain Car Learning Curve