Exercise 7
Gregory Attra
03.28.2022
CS 7180 - Prof. Amato
Code Setup:

1. Unzipped the 'ex7.zip' file
2. 'cd' into the 'code' directory
3. run 'source ./init' to setup the pyenv

Questions:

1. (a)
   input parameterized Q-value function $\mathcal{Q}(s, a|\mathcal{W})$
   initialize weights $\mathcal{W}$
   $\mathcal{E}$ = number of episodes
   for e in $\mathcal{E}$ **do**
   $\mathcal{T}$ = generate episode($\mathcal{W}$)
   $\mathcal{G} = 0$
   for $(s, a, s', r$ in $\mathcal{T}$ **do**
   $\mathcal{G} \leftarrow r + \gamma\mathcal{G}$
   $\mathcal{W} \leftarrow \mathcal{W} + \alpha[\mathcal{G} - \mathcal{Q}(s, a|\mathcal{W})]\nabla\mathcal{Q}(s, a|\mathcal{W})$
   end for
   end for

   (b) The pseudocode is not discussed in chapter 11 because the pseudocode for Monte Carl control with function approximation is the same as the pseudocode for Monte Carlo prediction with function approximation, which is already provided in chapter 10.

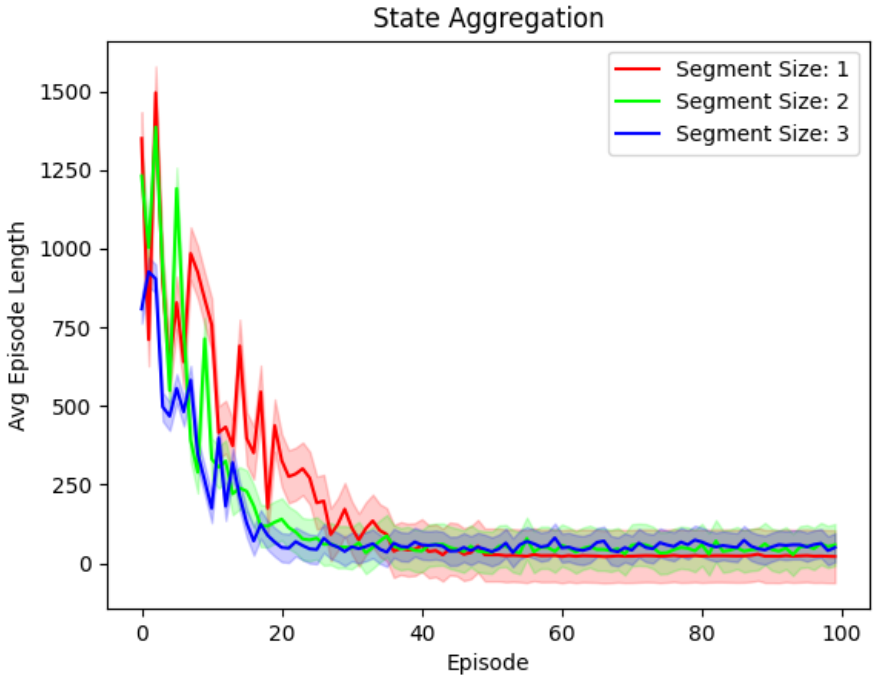
2. (a) The pseudocode for semi-gradient one-step Expected SARSA for control is identical to regular semi-gradient one-step SARSA for control with the following modification to the weight update rule:
   $\mathcal{W} \leftarrow \mathcal{W}+\alpha[r+\gamma\sum_{a'\in\mathcal{A}}\pi(a', s')\mathcal{Q}(s', a'|\mathcal{W})-\mathcal{Q}(s, a|\mathcal{W})]\nabla\mathcal{Q}(s, a|\mathcal{W})$

   (b) Again the only change need to derive semi-gradient Q-Learning is in the weight update rule:
   $\mathcal{W} \leftarrow \mathcal{W} + \alpha[r + \gamma \max_{a'\in\mathcal{A}} \mathcal{Q}(s', a'|\mathcal{W}) - \mathcal{Q}(s, a|\mathcal{W})]\nabla\mathcal{Q}(s, a|\mathcal{W})$

3. (a) For state aggregation, I implement an aggregation function $segment(s, N)$ which buckets the states into segments of size N. It does this by dividing state s by N.
   The update rule for the Q-function weights is:
   $\mathcal{W} \leftarrow \mathcal{W} + \alpha[r + \gamma\mathcal{Q}(s', \pi(s')|\mathcal{W}) - \mathcal{Q}(s, a|\mathcal{W})]\nabla\mathcal{Q}(s, a|\mathcal{W})$
   The gradient update $\nabla\mathcal{Q}(s, a|\mathcal{W})$ equals $X(s)$ where $X(s)$ is the feature set of state s

   (b) Using various segment sizes, I got the following results (plotted below). With a segment size of 1 (effectively tabular state aggregation), we get the best long-term performance but slightly slower learning

than segments of size 2 or 3 which learn a little faster but achieve slightly worse long-term results.

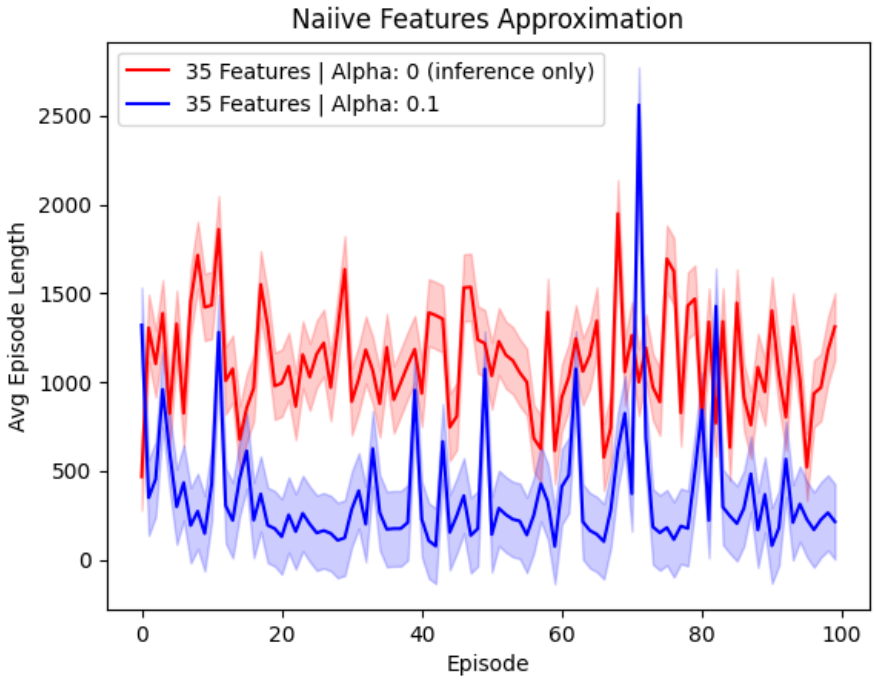


(c) Applying the same learning algorithm to naiive features for function approximation, my results were significantly worse. When using the feature set $(x, y, 1)$, the agent failed to learn at all and performed worse than a random policy. If the learning rate α was too high, I got exploding gradients. If it was too low, the agent got stuck in a loop of bad action selection and the episode never terminated.
I do not have a plot for these results as the agent almost never finished a signle episode and would be stuck in an infinite loop of bad action-selection.

(d) I achieved slightly better success by adding many more features to the feature set. The following features were added to produce the results plotted below:

- Normalized x and y coordinates (so that the values fell between 0 and 1 to avoid exploding gradients)
- The y and x distances from the goal position, also normalized
- A binary flag if the goal was one step up from the current position
- A binary flag if the goal was one step right of the current position
- A binary flag if a wall was one step above the agent
- A binary flag if a wall was one step below the agent

- A binary flag if a wall was one step to the right of the agent
- A binary flag if a wall was one step to the left of the agent
- A binary flag if a door was one step above the agent
- A binary flag if a door was one step below the agent
- A binary flag if a door was one step to the right of the agent
- A binary flag if a door was one step to the left of the agent
- A one-hot encoding of the doors in the map, where each entry was 0 by default, and 1 if the agent was within one step of that door
- Similarly, another one-hot encoding of the doors where the value was 1 if the agent was "in" the corresponding door
- A one-hot encoding of the rooms in the map where the value was 1 if the agent was in that room



We can see that very little learning occurs against a baseline of a random policy ($\alpha = 0$). Still there is some improvement from the random policy. But the agent fails to learn a near-optimal policy.