CS7180: Special Topic - Artificial Intelligence (Spring 2022) Christopher Amato
Northeastern University Due date seen on Canvas

Exercise 7: Function Approximation

Please remember:

- Exercise due date seen on Canvas.
- Submissions should be made electronically on Canvas. Please ensure that your solutions for both the written and programming parts are present. You can upload multiple files in a single submission, or you can zip them into a single file. You can make as many submissions as you wish, but only the latest one will be considered.
- For **Written** questions, solutions may be handwritten or typeset. If you write your answers by hand and submit images/scans of them, please please ensure legibility and order them correctly in a single PDF file.
- The PDF file should also include the figures from the **Plot** questions.
- For both **Plot** and **Code** questions, submit your source code along with reasonable documentation.
- You are welcome to discuss these problems with other students in the class, but you must understand and write up the solution and code yourself. Also, you *must* list the names of all those (if any) with whom you discussed your answers at the top of your PDF solutions page.
- Each exercise may be handed in up to two days late (24-hour period), penalized by 10% per day late. Submissions later than two days will not be accepted.
- Contact the teaching staff if there are medical or other extenuating circumstances that we should be aware of.

1. **1 point.** (RL2e 10.1) *On-policy Monte-Carlo control with approximation.*
   Written: We have not explicitly considered or given pseudocode for any Monte Carlo methods in this chapter. What would they be like? Why is it reasonable not to give pseudocode for them?
   How would they perform on the Mountain Car task?

2. **1 point.** (RL2e 10.2) *Semi-gradient expected SARSA and Q-learning.*
   Written:
   (a) Give pseudocode for semi-gradient one-step Expected Sarsa for control.
   (b) What changes to your pseudocode are necessary to derive semi-gradient Q-learning?

3. **4 points.** *Four rooms, yet again.*
Let us once again re-visit our favorite domain, Four Rooms, as implemented in Ex4 Q4.
We will first explore function approximation using *state aggregation*.
Since this domain has four discrete actions with significantly different effects, we will only aggregate states; different actions will likely have different Q-values in the same state (or set of states).

   (a) **Written/code:** Design and implement functions for computing features, approximate Q-values, and gradients for state aggregation. Describe your design.

   (b) **Code:** Implement semi-gradient one-step SARSA (or semi-gradient one-step Q-learning).
   Verify that your implementation works by trying it on *tabular* state aggregation, where each state is actually distinct (i.e., only aggregated with itself). Choose appropriate hyperparameters.
   Plot: Plot learning curves with confidence bounds, as in past exercises. 100 episodes should be sufficient.
   Hint: This is equivalent to the tabular setting, so you can compare your results against an implementation of tabular SARSA or Q-learning from Ex 5; the results should be similar.

   (c) **Code/plot:** Try at least three other choices of state aggregation, and plot the learning curves.
   Are you able to find aggregations that do better than tabular? How about worse than tabular?
   Written: Comment on your findings, including any trends and surprising results.

We will now consider the more general case of linear function approximation.
If necessary, adapt your implementation of semi-gradient one-step SARSA (or Q-learning) for linear function approximation; this might not be necessary if you implementation is sufficiently general.

   (d) One natural idea is to use the (x, y) coordinates of the agent's location as features.
   Specifically, use the following three features for state s:

   - The state's x coordinate
   - The state's y coordinate
   - 1 (i.e., the feature value is 1 for any state)

   Code/plot: Use these features for linear function approximation, and plot the learning curves.
   Written: Why is the constant feature necessary? What do you think happens without it?
   Also describe how you incorporated actions into your features.
   Written: 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.

   (e) **Code/plot:** Design and implement at least three more features, and plot the learning curves.
   You may use knowledge about the Four Rooms domain, and possibly knowledge about the goal location.
   Written: Comment on your findings, including any trends and surprising results.
   Will your features work if the goal location is not $(10, 10)$?

   (f) **[Extra credit.]** One of the main benefits of function approximation is that it can handle large state spaces. So far, the Four Rooms domain is still quite small.
   Design successively larger versions of Four Rooms, where each grid cell in the original environment can be subdivided into k cells per side (i.e., the number of states expands by a factor of k^2). You can choose whether or not to expand the walls/doorways.
   Experiment with the various state aggregation and linear features proposed above (or propose more).
   Plot your learning curves, and comment on your findings.