ME 599 Homework 1 Notes: Reinforcement Learning

Eric Mei

NOTE: I was only able to get the code to run properly via the command line and a .py file, sorry.

The purpose of this assignment is to become more familiar with reinforcement learning (RL) by exploring the gymnasium package. RL is not something I generally think of using in my field (climate sciences), since performing actions and making observations of the climate state are generally very expensive (in the case of general circulation models) or difficult (on the actual climate system). I'm curious if these exercises can help me think more creatively about implementations of RL in my work.

To-do list for assignment

0. Read up on documentation and make some quick notes
1. Run example code & take notes about variables and how they fit into various aspects of RL (reward, action, state, etc.)
2. Search online for fun tutorials with gymnasium to get some ideas about fun environemnts
3. Try training an agent on a particular environment

0. Notes from basic usage guide

- `make("render_mode")` defines specific environment and how it should be visualized in the PyGame window.
- `env.reset()` resets and makes an initial observation (state) of the environment, needs to be called initially or when the game has terminated (e.g., when `env.step()` returns `True` for `terminated` or `truncated`).
- `env.step(action)` causes the agent to perform an action. Needs to be looped (?) for making additional actions. Observations (states) and rewards are returned as well.
- can use `env.action_space` and `env.observation_space` to view all available actions and observations available to an environment. Calling `env.action_space.sample()` performs a random action in the action space, but a specific policy can be designed as well.
- Wrappers can enable implemention of boilerplate code without needing to write it myself. I'm not too clear about what these do yet, but I'll see if I can play around with them. Further documentation about wrappers here.

1. Example code notes

- kernel crashed immediately when running cell beginning `gym.make(...)`. edit: it's not crashing, the PyGame window just isn't closing after the cell terminates (when `env.close()` is called). edit2: took me a *significant* time to figure out, but this is just a product of me using jupyter notebooks (on mac). Running the example from the command line (`python RL_example.py`) fixes the issue entirely. This is suboptimal because it'll slow me down and I can't keep my figures in the notebook, but I'll use this fix for now.
- `for _ in range(10):` I usually use loops to index an array, so I've actually never seen this syntax but it makes sense.
- this example is very similar to the one in the basic usage guide in section 0 (above), so I'll only make notes in addition to the ones I've already made in that section.
- running 'CartPole-v1' for 10 steps is super short. Stick to 100 or 1000. This example went a little fast for me to understand exactly what the actions, observations, and rewards were. `action` is an integer that perhaps dictates the direction of the cartpole, `observation` is a vector that perhaps describes the position and velocity of the cartpole, and `reward` seems to be awarded at every step.
- For the 'FrozenLake-v1' example, `observation` is an integer that represents the position of the player, `action` is an integer that represents where the player moves, and `reward` is 0 until the player reaches the end, in which the reward becomes 1. It seems there's no RL in these examples, as there is neither a governing policy (it's random) nor an optimization strategy to improve the policy.

2. Fun environments

- taxi looks like a fun, more complicated alternative to the frozen lake environment. In running the environment briefly, the action and observations look similar to the frozen lake environment, albeit with a lot larger of a state space. The rewards are also more detailed as well, with -1 penalizing each step, -10 penalizing improper moves in the pickup locations, and (allegedly from the docs) +20 for successfully picking up and dropping off the passenger. I never received the larger reward with the short simulation times I played around with.
- mountain car for getting on top of a hill.

3. Training an agent

I'm going to first follow the tutorial to train an agent on the frozenlake example (with a few modifications). If I finish that, I'll move onto doing the same thing with the taxi environment.

- never used `class Params(NamedTuple)` before, but it looks super useful for some of my other research projects. I'll likely retrofit my existing research code after this.
- found some resources for Q-learning and ϵ-greedy. As a reminder, Q is a combo of the policy and value functions, which describes the 'quality' of action-value pairs. You can get stuck in local minima by always optimizing for the best action-value pairs from a given initializiation, so ϵ-greedy optimization takes random actions at ϵ probability to get out of local minima.
- the above bullet point describes what is happening in lines 146-149, in which the `explorer` of the `EpsilonGreedy` class takes the optimal action as defined by the `learner` (of the `Qlearning` class) 90% of the time. 10% of the actions are random within the action space.
- The existing implementation of the `Qlearning` class uses a TD(0) algorithm to update the Q-table. It could be fun trying to code up TD(1+) algorithms.
- The plots of the policies from the Q-tables are the most interesting things to me - it'll be interesting to find out how to visualize policies for enviroments where there is stochasticity in the states (e.g., taxi or flappy bird). I'm most interested in how these could be used to understand what the trained RL model has learned.

I didn't get far enough to edit the tutorial code and play around on my own, much less try this myself in a different environment. I think I have enough notes to pick up where I left off, though.

Important resources

- Eigensteve videos + textbook
- tic-tac-toe RL
- Hugging Face Q-learning and epsilon greedy
- Baeldung Q-learning and epsilon greedy