Lab 5

Due: Friday 11/21/2025 @ 11:59pm EST

The purpose of labs is to practice the concepts that we learn in class. This will be the first and only lab where we do **NOT** use Sepia. I wrote this entirely from scratch. This lab will be the reverse of your second programming assignment. In pa2, I implemented a lot of infrastructure for training a q-function and expect you to train it. This time, I will provide the model and training settings, and it will be up to you to complete the infrastructure. Hopefully after this lab you have an appreciation for the machinery that goes into training arbitrary reinforcement learning agents; not just q-functions.

This lab is all about the CartPole environment. CartPole is a famous control theory toy problem that you will train a q-function to learn. CartPole is implemented for RL in the `gymnasium` python package, and I have ported it over to java for you. The idea is thus: there is a cart that is allowed to slide along a horizontal rail. Attached to this cart is a pole which can pivot around the joint its attached to the cart with. The goal is to move the cart either to the left or the right to keep the pole from falling over. Real world demonstrations of this typically put a glass of water on the top of the pole, and the goal is to keep any water from spilling. here is a short youtube video of the CartPole-python being controlled (successfully) via a much more advanced version of Reinforcement Learning called Promixal Policy Optimization (PPO).

1. Copy Files

Please, copy the files from the downloaded lab directory to your cs440 directory. You can just drag and drop them in your file explorer.

- Copy `Downloads/lab5/lib/cp.jar` to `cs440/lib/cp.jar`.
  This file is the custom jarfile that I created for you.
- Copy `Downloads/lab5/src/labs` to `cs444/src/labs`.
  This directory contains our source code `.java` files.
- Copy `Downloads/lab5/cp.srcs` to `cs440/cp.srcs`.
  This file contains the paths to the `.java` files we are working with in this lab. Just like last lab, files like these are used to speed up the compilation process by preventing you from listing all source files you want to compile manually.
- Copy `Downloads/lab5/doc/labs` to `cs440/doc/labs`. This is the documentation generated from `cp.jar` and will be extremely useful in this assignment. After copying, if you double-click on `cs440/doc/labs/cp/index.html`, the documentation should open in your browser.

2. Test run

If your setup is correct, you should be able to compile and execute the given code. Your code will crash. Don't worry, this is fine it will work when the implementation is complete.

```
# Mac, Linux. Run from the cs440 directory.
javac -cp "./lib/*:." @cp.srcs
java -cp "./lib/*:." src.labs.cp.Main -p 500 -t 1000 -v 500 -g 0.99 -n 1e-4

# Windows. Run from the cs440 directory.
javac -cp "./lib/*;." @cp.srcs
java -cp "./lib/*;." src.labs.cp.Main -p 500 -t 1000 -v 500 -g 0.99 -n 1e-4
```

Task 1: ReplayBuffer (20 points)

In this task, I want you to finish the implementation of the replay buffer in `./src/labs/cp/ReplayBuffer.java`. In particular, there are two methods you need to finish: `addSample` and `getGroundTruth`. All of the details for how to complete these methods are left as comments in the (empty) method bodies.

Task 2: Main (20 points)

In this task, I want you to finish the implementation of the training logic in `./src/labs/cp/Main.java`. This file has a `main` method, so after compiling, you can run this file directly. The purpose of this file is to execute the main training loop for reinforcement learning, which roughly goes as follows:

- Play a bunch of games. If you are using a replay buffer, populate the replay buffer from these training games. If you are not using a replay buffer, your model should be updating as the training games are played.
- If you are using a replay buffer, convert your replay buffer into a supervised learning dataset, and fit your model to this data.
- Save your model to disk. This should happen regardless of whether or not you are using a replay buffer.
- Play a bunch of evaluation games. During these games, your model is fixed, and you are not recording any data from these games. The purpose of these games are to directly measure the average trajectory utility, so we can check on the progress of learning (i.e. plot a learning curve from these samples).

I have implemented the main training loop for you, but I have left some of the pieces for you to implement. Most notably, you will have to complete the `train` static method, in which you are reponsible for playing a bunch of games (the exact number you can lookup from the `Namespace`). Since we are using a replay buffer, you should record individual transitions from these games in your replay buffer. Don't forget to add a transition after the game ends to record the terminal transition!

Train the Model (10 points)

Once you have your implementation working, you will need to actually train a model. Like pa2, your model is automatically saved after every cycle. If you do not provide a `-o` or `--outFile` argument, then these files will appear at `./params/qFunctionXX.model` where `XX` is the index of the cycle that model was saved during. I used the following settings, which I know work:

```
java -cp ./lib/*:.  src.labs.cp.Main -p 500 -t 1000 -v 500 -g 0.99 -n 1e-4
--numUpdates 3 -b 50000
```

CartPole is not a difficult problem, so there are many other settings that will work, so don't feel beholden to using these (you may not have enough RAM to make a replay buffer of 50k).

Train your model. Your goal is to produce a model that can keep the CartPole from falling over for at least 250 turns of the game. Once you have a model that you're happy with, rename that `.model` file to `params.model`, and include it in your submission. The autograder will look specifically for `params.model`, so be sure to name it correctly.

Task 2: Submitting your lab

Please submit all four files: `ReplayBuffer.java`, `Dataset.java`, `Main.java` and `params.model` on gradescope (just drag and drop in the file).