Spinning Up Workshop: Hackathon Project Recommendations

During the free-hacking period, you're welcome to work on anything you like! But we strongly encourage you to work on something related to AI or deep reinforcement learning, to help you take what you've learned from the lecture materials and put it into practice. To help you get started, we've compiled a few recommendations for things to hack on!

1. Spinning Up in Deep RL Exercises

Spinning Up ships with a set of warm-up exercises, and this could be a great way to ease into hacking on deep RL. While they default to using MuJoCo (which requires a license), you can easily retool them to use equivalent (free) Roboschool environments. To see one example of such retooling, check out this pull request on Spinning Up.

2. Design an RL Environment

To make a problem tractable for reinforcement learning, you first have to formalize it as an RL environment: that means

- deciding how the observations work,
- deciding what the space of actions will be,
- deciding termination criteria (that is: when does an episode end?),
- selecting a reward function which, when optimized, will yield optimal behavior,
- and creating a software interface that allows the agent to send an action to the environment and step the environment forward in time.

A good project in RL is to take a problem that you care about, formalize it as an RL environment, and build the software interface. Possible problems you might consider include:

- a robot interacting with a physical world,
- multiple agents interacting with each other in some kind of game,
- natural language conversation with a human (what would the reward function for this even be?).

Note: reward design is really hard, so even if you get everything else working, it may be really hard to train an agent to do what you want it to do. But don't be afraid of this! Lean in---these kinds of problems are worth solving!

3. Reimplement a Core Deep RL Algorithm from Scratch

For instance, try writing your own DQN or PPO implementation. If those are easy, try building more sophisticated algorithms, like Rainbow. Make sure to follow the recommendations from the Spinning Up essay for how to build and test your RL implementation.

4. Implement Random Network Distillation

Exploration is a hard problem in reinforcement learning when rewards are sparse--that is, when rewards are usually zero, and only very rarely does the agent randomly find a state with a helpful reward. One way to repair this issue is with **intrinsic motivation**: you give the agent a reward which encourages it to seek out new and unfamiliar states. This makes it more likely that it will find real rewards (**extrinsic rewards**).

One algorithm for intrinsic motivation is called Random Network Distillation (RND). It works by keeping track of two neural networks: a randomly-initialized **target** network (which never changes), and a different randomly-initialized **predictor** network. Both the target and the predictor networks map from observations to vectors of some size, and these two networks may have the same or different architectures as long as they have the same inputs and outputs. Throughout the RL training, the predictor network is trained to output the same thing as the target network. The predictor network error is used as an intrinsic reward signal for the agent: that is,

$$r_{int}(o_t) = \|\hat{f}(o_t) - f(o_t)\|$$

where o_t is an observation, r_{int} is the intrinsic reward at that observation, $\hat{f}$ is the current predictor network, and f is the current target network.

Try implementing this algorithm (as a subroutine of a standard core RL algorithm, like the Spinning Up PPO implementation), and then seeing if it helps exploration in the Atari-Ram environments from the OpenAI Gym.

5. Implement Hindsight Experience Replay in GoalGridWorld

Set up an environment for testing goal-oriented policy learning, which is just a simple fully-observed gridworld plus goals (eg a 1-hot the same size as the grid, with a 1 where the goal state is and 0s everywhere else). The reward should be 0 if the agent attains the goal (at which point the episode ends), otherwise the reward is -1. The size of the gridworld should be an adjustable parameter (you should be able to run experiments with small gridworlds or large gridworlds).

Train agents with DQN in this GoalGridWorld env. Then, train agents with Hindsight Experience Replay (HER). What happens to DQN and HER as you increase the size of the gridworld? Probably should see DQN begin to fail, while HER continues to perform well.

6. Implement Hindsight Experience Replay's BitFlip Experiment

Reimplement the bitflip experiment in the HER paper (section 3.1) and duplicate the results. Show that DQN fails as described in the paper.

7. Implement Diversity is All You Need in GridWorld

8. Extend Spinning Up's PPO Implementation to Support Recurrence