CS294-112 Deep Reinforcement Learning HW5: Exploration

Due November 13th, 11:59 pm

1 Introduction

For this homework, you get to choose among several topics to investigate. Each topic corresponds to a different assignment for HW5. You will implement only **one** of the assignments. You can implement a second assignment as a make-up for a previous homework or for extra credit. Section 2 is for the Exploration assignment.

2 Exploration

Exploration—how agents discover actions that lead to high rewards—is a key component of reinforcement learning. In this homework, you will investigate count-based exploration methods that modify the reward function to encourage exploring novel parts of the state space:

$$\tilde{R}(\mathbf{s}_t) = R(\mathbf{s}_t) + \alpha \cdot \mathcal{B}(N(\mathbf{s}_t)). \tag{1}$$

$N(\mathbf{s}_t)$ represents the number of times the agent has visited the state, and the function f is a monotonically decreasing function of $N(\mathbf{s}_t)$, known as the *exploration bonus*. The intuition is that we would like to encourage the agent to visit novel states. If the state $\mathbf{s}$ is novel or is rarely visited, then $N(\mathbf{s})$ will be low, and $\mathcal{B}(N(\mathbf{s}_t))$ will be high. Conversely, if the state $\mathbf{s}$ is visited often, then $N(\mathbf{s})$ will be high, and $\mathcal{B}(N(\mathbf{s}_t))$ will be low. Therefore, the exploration bonus is an additional term to the reward function that encourages the agent to spend more time visiting novel states. The hyperparameter α indicates how much to reward novel states.

In the discrete case, we can use a histogram to keep track of the number of times the agent visited state $\mathbf{s}$, so the histogram directly gives us $N(\mathbf{s}_t)$. However, when the state space is continuous, the probability of any two states being equal is 0, so we cannot simply tally the number of times we've visited the

state. Instead, we must fit a density model $f_\phi(\mathbf{s}_t)$ over the state space and derive the count $N(\mathbf{s}_t)$ from f_ϕ. Intuitively, if similar states to $\mathbf{s}_t$ have been visited many times, then $f_\phi(\mathbf{s}_t)$ will be high.

Given Eqn. 1, you can then run your standard reinforcement learning algorithms with only a single additional step: computing $\mathcal{B}(N(\mathbf{s}_t))$ as your agent acts in the environment. To do this we need to keep a replay buffer $\mathcal{B}$ that stores the states the agent has visited so far (note that here we only store states, not entire transitions). In the discrete case, the histogram can take place of the replay buffer; in the continuous case, the replay buffer serves as the data distribution with which we will fit the density model $f_\phi(\mathbf{s}_t)$. The algorithm is summarized below:

Algorithm 1 Count-based exploration with reward bonuses

Initialize replay buffer $\mathcal{B}$
while *not done* **do**
  Sample trajectories $\{\tau_j\}$ from policy π_i
  Store the states from $\{\tau_j\}$ into the $\mathcal{B}$
  Fit a histogram or density model to the states in $\mathcal{B}$
  for $s \in \{\tau_j\}$ **do**
    $R'(s,a) = R(s,a) + \alpha\mathcal{B}(N(\mathbf{s}_t))$
  end
  Improve π_i with respect to $R'(s,a)$
end

There are many possible ways to specify $\mathcal{B}(N(\mathbf{s}_t))$. In this homework, for discrete states we will use

$$\mathcal{B}(N(\mathbf{s}_t)) = N(\mathbf{s}_t)^{-\frac{1}{2}}.$$

For continuous states we will use a heuristic bonus

$$\mathcal{B}(N(\mathbf{s}_t)) = -\log f_\phi(\mathbf{s}_t)$$

which skips computing $N(\mathbf{s}_t)$ but is still a function that decreases the more states similar to $\mathbf{s}_t$ have been visited.

2.1 Discrete States

The purpose of this section is to focus on modifying the rewards with the exploration bonus without having to worry about fitting a density model. Therefore we will modify the rewards like so:

$$R'(s,a) = R(s,a) + \alpha N(\mathbf{s}_t)^{-\frac{1}{2}} \tag{2}$$

2.2 Continuous States

Now that we have implemented the framework for Algorithm 1 for discrete states, we will now replace the histogram with a replay buffer and a density model f_ϕ, and our goal is to be able to compute $f_\phi(\mathbf{s})$ for any state $\mathbf{s}$ such that we modify the rewards like so:

$$R'(s,a) = R(s,a) + \alpha\left(-\log f_\phi(s)\right) \tag{3}$$

2.2.1 Non-parametric density estimation: kernel density estimation (KDE)

Kernel density estimation is a non-parametric method that estimates the density model by maintaining a dataset of all encountered states (the replay buffer $\mathcal{B}$ in our case and using a kernel function $K_\phi(\mathbf{s}^1, \mathbf{s}^2)$ to measure the similarity between states.

Using an radial basis function kernel (https://en.wikipedia.org/wiki/Radial_basis_function_kernel), we can to estimate the density of a new datapoint s by plopping a Gaussian distribution centered around each of the datapoints in $\mathcal{B}$, evaluate the probability of s under each of these Gaussians, and average these probabilities together (See https://en.wikipedia.org/wiki/Kernel_density_estimation for some nice intuitive figures). Intuitively, if a lot of the datapoints in $\mathcal{B}$ are close together, then the probability density of a nearby point because each Gaussian contributes to the probability density of that nearby point. In particular, for a given state s, we can estimate its probability density as

$$\begin{aligned} f_\phi(s) &= \frac{1}{|\mathcal{B}|}\sum_{s'\in\mathcal{B}} K_{\text{rbf}}(s,s') \\ &= \frac{1}{|\mathcal{B}|}\sum_{s'\in\mathcal{B}} \exp\left(-\frac{\|s-s'\|^2}{2\sigma^2}\right). \end{aligned}$$

2.2.2 Parametric density estimation: exemplar models

The problem with kernel density estimators is that to every time we evaluate the probability of a point, we have to apply the kernel to every point in the replay buffer, which becomes computationally intensive with a large replay buffer. Alternatively, we can use a parametric density estimator, which does not require a full pass through all the data to compute probabilities, but comes at the cost of training the density model from samples, which introduces another layer of approximation.

One way to estimate the probability density $f_\phi(s)$ is to train a state-conditioned noisy discriminator $D_s(s')$ to output 1 if $s = s'$ and 0 if $s \neq s'$ (note that D_s is a

discriminator conditioned on the *exemplar* s, so D_s and $D_{s'}$ are not the same). The output of the discriminator is the probability that a Bernoulli random variable y takes the value 1: $p(y = 1|s, s') := p(s = s')$. Then we can estimate $f_\phi(s)$ by evaluating D_s on its own state s:

$$f_\phi(s) = \frac{1 - D_s(s)}{D_s(s)}$$

the reasoning behind which you can find here: https://arxiv.org/abs/1703.01260. With this discriminator, we can estimate a probability density model over the states we've seen before (in the replay buffer) by training the discriminator to distinguish between a *exemplar states* s and the states s' from the replay buffer. Intuitively, if $D_s(s')$ is high, then this means that s is easily distinguished from states s' in $\mathcal{B}$, which means the probability that a state similar to s is in $\mathcal{B}$, in which case $f_\phi(s)$ is low. Conversely, if states similar to s are very common in $\mathcal{B}$, then the D_s will have a hard time distinguishing s and s', in which case $D_s(s, s')$ will output a value close to 0.5, which would make $f_\phi(s)$ high.

The discriminator can be viewed as a graphical model decomposed as:

$$p(y|s, s') = p(y|z, z')q(z|s)q(z'|s')$$

where z are latent Gaussian random variables and y is a Bernoulli variable. The z's introduce noise in the discriminator to prevent it from overfitting and encourage it to assign similar probability density to similar states. The discriminator is trained to maximize the following objective:

$$\max_{p_{y|z}, q_{z|x}} \mathbb{E}_{x\sim\tilde{p_x}} \mathbb{E}_{z\sim q_{z|x}} \left[[\log p(y|z)] - \beta D_{KL}\left(q(z|x)||p(z)\right)\right]$$

where $p(z)$ is a multivariate standard Gaussian, β is a weighting coefficient that controls how much the discriminator overfits (tries to maximize the log likelihood more) or underfits (tries to make the latent distribution as close to a standard Gaussian as possible), and $p(\tilde{x})$ is the data distribution the discriminator is trained on, which contains half exemplar states and half replay-buffer states.

2.3 Code

2.3.1 Installation

Obtain the code from https://github.com/berkeleydeeprlcourse/homework/tree/master/hw5/exp. In addition to the installation requirements from previous homeworks, install additional required packages by running: `pip install -r requirements.txt`. Also replace `/gym/envs/mujoco/half_cheetah.py` with the provided `sparse_half_cheetah.py`.

2.3.2 Overview

You will modify the following files:

- `train_ac_exploration_f18.py`
- `density_model.py`
- `exploration.py`

You should also familiarize yourself with the following files:

- `replay.py`
- `pointmass.py`
- `sparse_half_cheetah.py`

All other files are optional to look at.

2.4 Implementation

Problem 1

What you will implement: The reward modification (Eqn. 1), the count-based reward bonus (Eqn. 2), and the histogram density model .

Where in the code to implement: All parts of the code where you find

```
### PROBLEM 1
### YOUR CODE HERE
```

Implementation details are in the code.

How to run: Run the commands under `P1 Hist PointMass` in `run_all.sh` to compare an agent with histogram-based exploration and an agent with no exploration. Then use `plot.py` to plot the returns of the runs.

What will be outputted: A plot with 2 curves comparing an agent with histogram-based exploration and an agent with no exploration.

What will a correct implementation output: The histogram-based exploration should get an average reward of above 90. Average reward for no exploration should get above 65.

Problem 2

What you will implement: The heuristic reward bonus (Eqn. 2), and the kernel density estimator with the radial basis function kernel.

Where in the code to implement: All parts of the code where you find

```
### PROBLEM 2
### YOUR CODE HERE
```

Implementation details are in the code.

How to run: Run the commands under `P2 RBF PointMass` in `run_all.sh` Then use `plot.py` to plot the returns of the runs to compare an agent with KDE-based exploration and an agent with no exploration (the run of which you can reuse from Problem 1)

What will be outputted: A plot with 2 curves comparing an agent with KDE-based exploration and an agent with no exploration.

What will a correct implementation output: The KDE-based exploration should get an average reward of above 70. Average reward for no exploration should get above 65.

Problem 3

What you will implement: The EX2 discriminator.

Where in the code to implement: All parts of the code where you find

```
### PROBLEM 3
### YOUR CODE HERE
```

Implementation details are in the code.

How to run: Run the commands under `P3 EX2 PointMass` in `run_all.sh` Then use `plot.py` to plot the returns of the runs to compare an agent with EX2-based exploration and an agent with no exploration (the run of which you can reuse from Problem 1)

What will be outputted: A plot with 2 curves comparing an agent with EX2-based exploration and an agent with no exploration.

What will a correct implementation output: The EX2-based exploration should get an average reward of about 100. Average reward for no exploration should get above 65.

Problem 4

What you will implement: Nothing! Nothing at all!

How to run: Run the commands under `P4 HalfCheetah` in `run_all.sh`. We have two hyperparameter settings for the EX2-based exploration. One uses the bonus coefficient $\alpha = 0.0001$ and trains the density model for 10000 iterations. The other uses a bonus coefficient $\alpha = 0.001$ and trains the density model for

1000 iterations. Use `plot.py` to plot the returns of the runs to compare the two agents with EX2-based exploration and an agent with no exploration.

What will be outputted: A plot with 3 curves comparing the agents with EX2-based exploration and an agent with no exploration.

What will a correct implementation output: The average return for $\alpha = 0.001$ EX2-based exploration should get up to above 7 before 20 iterations, stay relatively flat, and drop slightly to above 5 iterations. The average return for $\alpha = 0.0001$ EX2-based exploration should get up to above 12 at 30 iterations, fall back down below 10, and reach slightly above 25 after 80 iterations. The agent with no exploration should get an average reward less than 2.5.

Short answer: Compare the two learning curves for EX2 and hypothesize a possible reason for (1) the shape of each learning curve and (2) the difference in performance between the learning curves.

2.5 PDF Deliverable

You can generate all results needed for the deliverables by running:

```
./run_all.sh
```

and then calling `python plot.py` to produce the appropriate plots Please provide the following plots and responses on the specified pages.

Problem 1 (page 1)

(a) A plot with 2 curves comparing an agent with histogram-based exploration and an agent with no exploration for PointMass.

Problem 2 (page 2)

(a) A plot with 2 curves comparing an agent with KDE-based exploration and an agent with no exploration for PointMass.

Problem 3 (page 3)

(a) A plot with 2 curves comparing an agent with EX2-based exploration and an agent with no exploration for PointMass.

Problem 4 (page 4)

(a) A plot with 3 curves comparing an agent with EX2-based exploration and an agent with no exploration for HalfCheetah.

(b) Your short answer response comparing the Ex2 learning curves for HalfCheetah.

2.6 Submission

Turn in both parts of the assignment on Gradescope as one submission. Upload the zip file with your code to **HW5 Code Exploration**, and upload the PDF of your report to **HW5 Exploration**.