XCS234 Assignment 3

Due Sunday, 9 January at 11:59pm PT.

Guidelines

1. If you have a question about this homework, we encourage you to post your question on our Slack channel, at http://xcs234-scpd.slack.com/
2. Familiarize yourself with the collaboration and honor code policy before starting work.
3. For the coding problems, you must use the packages specified in the provided environment description. Since the autograder uses this environment, we will not be able to grade any submissions which import unexpected libraries.

Submission Instructions

Written Submission: Some questions in this assignment require a written response. For these questions, you should submit a PDF with your solutions online in the online student portal. As long as the PDF is legible and organized, the course staff has no preference between a handwritten and a typeset LaTeX submission. If you wish to typeset your submission and are new to LaTeX, you can get started with the following:

- Type responses only in `submission.tex`.
- Submit the compiled PDF, **not** `submission.tex`.
- Use the commented instructions within the `Makefile` and `README.md` to get started.

Coding Submission: Some questions in this assignment require a coding response. For these questions, you should submit only the `src/submission.py` file in the online student portal. For further details, see Writing Code and Running the Autograder below.

Online Submission: Some questions in this assignment require responses to be submitted interactively within a Gradescope online assessment. For these questions, you should consult your Gradescope dashboard for the availability of this assessment.

Honor code

We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions. More information regarding the Stanford honor code can be found at https://communitystandards.stanford.edu/policies-and-guidance/honor-code.

Writing Code and Running the Autograder

All your code should be entered into `src/submission.py`. When editing `src/submission.py`, please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do not make changes to files other than `src/submission.py`.

The unit tests in `src/grader.py` (the autograder) will be used to verify a correct submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
$ python grader.py
```

There are two types of unit tests used by the autograder:

- `basic`: These tests are provided to make sure that your inputs and outputs are on the right track, and that the hidden evaluation tests will be able to execute.
- `hidden`: These unit tests are the evaluated elements of the assignment, and run your code with more complex inputs and corner cases. Just because your code passed the basic local tests does not necessarily mean that they will pass all of the hidden tests. These evaluative hidden tests will be run when you submit your code to the Gradescope autograder via the online student portal, and will provide feedback on how many points you have earned.

For debugging purposes, you can run a single unit test locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
$ python grader.py 3a-0-basic
```

Before beginning this course, please walk through the Anaconda Setup for XCS Courses to familiarize yourself with the coding environment. Use the env defined in `src/environment.yml` to run your code. This is the same environment used by the online autograder.

Test Cases

The autograder is a thin wrapper over the python `unittest` framework. It can be run either locally (on your computer) or remotely (on SCPD servers). The following description demonstrates what test results will look like for both local and remote execution. For the sake of example, we will consider two generic tests: `1a-0-basic` and `1a-1-hidden`.

Local Execution - Hidden Tests

All hidden tests rely on files that are not provided to students. Therefore, the tests can only be run remotely. When a hidden test like `1a-1-hidden` is executed locally, it will produce the following result:

```
----- START 1a-1-hidden:  Test multiple instances of the same word in a sentence.
----- END 1a-1-hidden [took 0:00:00.011989 (max allowed 1 seconds), ???/3 points] (hidden test ungraded)
```

Local Execution - Basic Tests

When a basic test like `1a-0-basic` passes locally, the autograder will indicate success:

```
----- START 1a-0-basic:  Basic test case.
----- END 1a-0-basic [took 0:00:00.000062 (max allowed 1 seconds), 2/2 points]
```

When a basic test like `1a-0-basic` fails locally, the error is printed to the terminal, along with a stack trace indicating where the error occurred:

```
----- START 1a-0-basic:  Basic test case.
<class 'AssertionError'>
{'a': 2, 'b': 1} != None
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/Users/grinch/Local_Documents/SCPD/XCS221/A1/src/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/Users/grinch/Local_Documents/Software/anaconda3/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
----- END 1a-0-basic [took 0:00:00.003809 (max allowed 1 seconds), 0/2 points]
```

This error caused the test to fail.

In this case, start your debugging in line 23 of grader.py.

Remote Execution

Basic and hidden tests are treated the same by the remote autograder. Here are screenshots of failed basic and hidden tests. Notice that the same information (error and stack trace) is provided as the in local autograder, now for both basic and hidden tests.

1a-0-basic) Basic test case. (0.0/2.0)

```
<class 'AssertionError'>:  {'a': 2, 'b': 1} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 23, in test_0
    submission.extractWordFeatures("a b a"))
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

Just like in the local autograder, this error caused the test to fail.

Just like in the local autograder, start your debugging in line 23 of grader.py.

1a-1-hidden) Test multiple instances of the same word in a sentence. (0.0/3.0)

```
<class 'AssertionError'>:  {'a': 23, 'ab': 22, 'aa': 24, 'c': 16, 'b': 15} != None
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 59, in testPartExecutor
    yield
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 605, in run
    testMethod()
  File "/autograder/source/graderUtil.py", line 54, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/graderUtil.py", line 83, in wrapper
    result = func(*args, **kwargs)
  File "/autograder/source/grader.py", line 31, in test_1
    self.compare_with_solution_or_wait(submission, 'extractWordFeatures', lambda f: f(sentence))
  File "/autograder/source/graderUtil.py", line 183, in compare_with_solution_or_wait
    self.assertEqual(ans1, ans2)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 829, in assertEqual
    assertion_func(first, second, msg=msg)
  File "/autograder/source/miniconda/envs/XCS221/lib/python3.6/unittest/case.py", line 822, in _baseAssertEqual
    raise self.failureException(msg)
```

This error caused the test to fail.

Start your debugging in line 31 of grader.py.

Finally, here is what it looks like when basic and hidden tests pass in the remote autograder.

1a-0-basic) Basic test case. (2.0/2.0)

1a-1-hidden) Test multiple instances of the same word in a sentence. (3.0/3.0)

0 Introduction

In this assignment, we will begin implementing policy gradient methods on environments with discrete or continuous actions spaces. We will also explore variance reduction methods to allow us to improve upon the performance of the REINFORCE algorithm.

By the end of this assignment, you should have an understanding of how to implement some of the fundamental policy gradient algorithms. You should also be able to apply variance reduction techniques to improve upon the performance of REINFORCE and to derive/discuss variance reduction in policy gradient methods.

Advice

- It will take time to train your policy gradient implementation across all environments for various seeds. It is worthwhile budgeting at least 12 hours for all training jobs to run to completion.
- In this assignment, you will train your policy gradient implementation on your local machine using MuJoCo (physics engine) to power the simulation environment. To accomplish this you need to configure your local environment appropriately, we recommend using Docker to streamline the environment setup. We have provided a guide MuJoCo Installation Guide for how to setup your environment using Docker and otherwise.
- Be sure to check that you have access to the MuJoCo license key file `mjkey.txt`. You will need this file to setup your environment and run simulations using MuJoCo.

Coding Deliverables

For this assignment, please submit the following files to gradescope to recieve points for coding questions:

- `src/submission/__init__.py`
- `src/submission/baseline_network.py`
- `src/submission/mlp.py`
- `src/submission/policy.py`
- `src/submission/policy_gradient.py`
- `src/submission/CartPole-v0-no-baseline-model-weights.pt`
- `src/submission/CartPole-v0-no-baseline-scores.npy`
- `src/submission/CartPole-v0-baseline-model-weights.pt`
- `src/submission/CartPole-v0-baseline-scores.npy`
- `src/submission/InvertedPendulum-v2-no-baseline-model-weights.pt`
- `src/submission/InvertedPendulum-v2-no-baseline-scores.npy`
- `src/submission/InvertedPendulum-v2-baseline-model-weights.pt`
- `src/submission/InvertedPendulum-v2-baseline-scores.npy`
- `src/submission/HalfCheetah-v2-no-baseline-model-weights.pt`
- `src/submission/HalfCheetah-v2-no-baseline-scores.npy`
- `src/submission/HalfCheetah-v2-baseline-model-weights.pt`
- `src/submission/HalfCheetah-v2-baseline-scores.npy`

1 Policy Gradient Methods

The goal of this problem is to experiment with policy gradient and its variants, including variance reduction methods. Your goal will be to set up policy gradient for both continuous and discrete environments, and implement a neural network baseline for variance reduction. The script for running the policy gradient algorithm is setup in `run.py`, and everything that you need to implement is in the files `baseline_network.py`, `mlp.py`, `policy.py` and `policy_gradient.py` within the submission folder. Each submission script has detailed instructions for each implementation task. We have also provided an overview of key steps in the algorithm below:

REINFORCE

Recall the policy gradient theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q^{\pi_\theta}(s,a)\right]$$

REINFORCE is a Monte Carlo policy gradient algorithm, so we will be using the sampled returns G_t as unbiased estimates of $Q^{\pi_\theta}(s,a)$. The REINFORCE estimator can be expressed as the gradient of the following objective function:

$$J(\theta) = \frac{1}{\sum T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} \log(\pi_\theta(a_t^i \mid s_t^i)) G_t^i$$

where D is the set of all trajectories collected by policy π_θ, and $\tau^i = (s_0^i, a_0^i, r_0^i, s_1^i, \ldots, s_{T_i}^i, a_{T_i}^i, r_{T_i}^i)$ is trajectory i.

Baseline

One difficulty of training with the REINFORCE algorithm is that the Monte Carlo sampled return(s) G_t can have high variance. To reduce variance, we subtract a baseline $b_\phi(s)$ from the estimated returns when computing the policy gradient. A good baseline is the state value function, $V^{\pi_\theta}(s)$, which requires a training update to ϕ to minimize the following mean-squared error loss:

$$L_{\text{MSE}}(\phi) = \frac{1}{\sum T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} (G_t^i - b_\phi(s_t^i))^2$$

Advantage Normalization

After subtracting the baseline, we get the following new objective function:

$$J(\theta) = \frac{1}{\sum T_i} \sum_{i=1}^{|D|} \sum_{t=1}^{T_i} \log(\pi_\theta(a_t^i \mid s_t^i)) \hat{A}_t^i$$

where

$$\hat{A}_t^i = G_t^i - b_\phi(s_t^i)$$

A second variance reduction technique is to normalize the computed advantages, $\hat{A}_t^i$, so that they have mean 0 and standard deviation 1. From a theoretical perspective, we can consider centering the advantages to be simply adjusting the advantages by a constant baseline, which does not change the policy gradient. Likewise, rescaling the advantages effectively changes the learning rate by a factor of $1/\sigma$, where σ is the standard deviation of the empirical advantages.

Note: for the following coding questions some scripts contain member functions of different classes with the same name. In order to distinguish which function we refer to in each question we use the following syntax `class::function`*.*

(a) **[5 points (Coding)]**

Implement `build_mlp` within `submission/mlp.py` which will be used to construct a multi layer perceptron based on input argument values.

(b) **[6 points (Coding)]**

Implement the following functions within `submission/baseline_network.py` to create the baseline network for our policy gradient implementation:

- `BaselineNetwork::__init__`
- `BaselineNetwork::forward`
- `BaselineNetwork::calculate_advantage`
- `BaselineNetwork::updated_baseline`

(c) **[4 points (Coding)]**

Implement `PolicyGradient::init_policy` within `policy_gradient.py` in order to initialize a network and optimizer for our implementation of policy gradient.

(d) **[3 points (Online)]**

Please refer to question 1 of the Gradescope online assessment `A3 (Quiz)`.

(e) **[6 points (Coding)]**

Implement `PolicyGradient::get_returns` within `policy_gradient.py` in order to calculate returns G_t given data about specific trajectories.

(f) **[12 points (Coding)]**

In this question, we will define the conditional probability distribution over actions for both discrete and continuous environments. Implement the following functions in `submission/policy.py` in order to define these distributions and how we sample from them:

- `BasePolicy::act`
- `CategoricalPolicy::action_distribution`
- `GaussianPolicy::__init__`
- `GaussianPolicy::std`
- `GaussianPolicy::action_distribution`

(g) **[6 points (Coding)]**

Implement the following functions in `submission/policy_gradient.py`:

- `PolicyGradient::update_policy`
- `PolicyGradient::normalize_advantage`

This will complete the `PolicyGradient` class by providing a network update rule as well as the option to normalize advantages we have calculated.

(h) **[9 points (Coding)]**

In this question, you will train your policy gradient implementation on three different environments.

- CartPole-v0
- InvertedPendulum-v2
- HalfCheetah-v2

For each of the 3 environments, choose 3 random seeds and run your policy gradient implementation both with and without a baseline.

Setting up your Environment

For this question we need to setup your local environment to work with MuJoCo. In order to run, each of the simulation environments you will need to ensure that you have the MuJoCo physics engine installed and authorized via a license key. To streamline this setup process we have created a Dockerfile with the necessary setup `src/Dockerfile`. We recommend using this in order to speed up the installation process.

You may find a full guide on how to setup your environment and further details about Docker through our MuJoCo Installation Guide.

Training Policy Gradient

The general form for running your policy gradient implementation is as follows (where config_filename is the name of a yaml file in the `src/config` folder with your training configuration):

```
$ python run.py --config_filename config_filename
```

Depending on the configuration file details you provide this command may train models for more than one seed which can take a while (especially for *HalfCheetah-v2*). As a result, you may wish to run these jobs as background processes in which case you may run the command as follows:

```
$ nohup python run.py --config_filename config_filename &
```

This will prevent the python process from ending when you close your terminal window as well as running the command in the background. Additionally, in the directory where you run the command a `nohup.out` file is created and contains the standard output from the process that you are running. Feel free to make use of this form of the command for the longer running processes.

Altering the Training Configuration

We have provided you with sample configuration files in the assignment `config` folder for you to use such as `config/cartpole_baseline.yml`. These configurations will be suitable to use directly in training your agent without altering the config file.

However, you may optionally alter the training configuration files directly to run your policy gradient implementation with different settings. Below we have briefly described some of the changes you could apply:

- To run your implementation with/without a baseline you may change the `use_baseline` option to either `true` or `false`.
- You may also choose to train multiple seeds under the one python process through specifying more than one seed in the list `seed` (equally you may specify a single seed in this list if you want to run just one).
- You may also wish to qualitatively observe the performance of your agent. To do so, you can record a single episode of the trained agent through changing the `record` option to `true` in the training configuration file for your run. Once your training run is complete you should see a video recording outputted in the `results` folder for the given run.

Results & Plotting

Once training is complete you should observe the creation of the following folder `src/results` which contains the results of your training runs. In addition, you should note that some files based on model evaluation have been created in the submission folder. This contains the weights and scores for one of your training runs for each environment and baseline/no baseline configuration (this will only populate once one of your training runs achieves evaluation scores above a certain threshold). You will need to upload these weights with your submission to receive full credit for this question. To plot your results for certain seeds run:

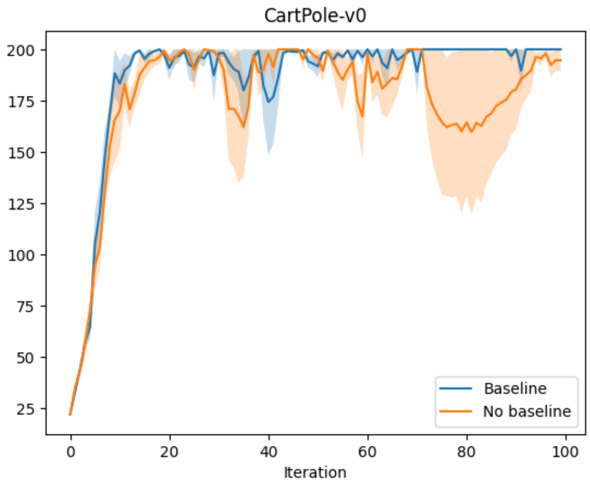
```
$ python run.py --plot_config_filename plot_config_filename
```

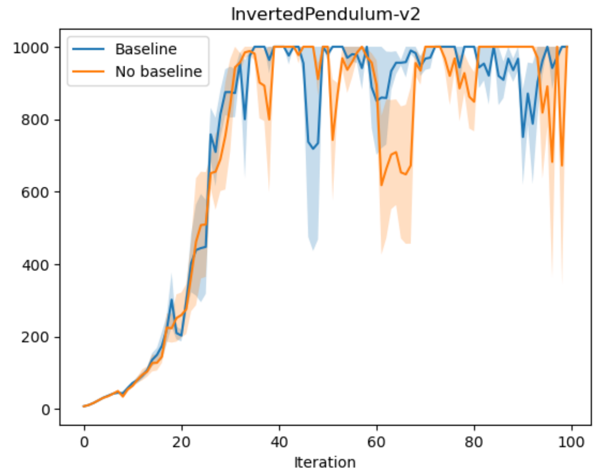
where plot_config_filename is the name of a yaml file in the `src/config` folder containing information on the seeds we wish to plot and for which environments. Please consult `config/plot.yml` for an example of the structure of this configuration file.

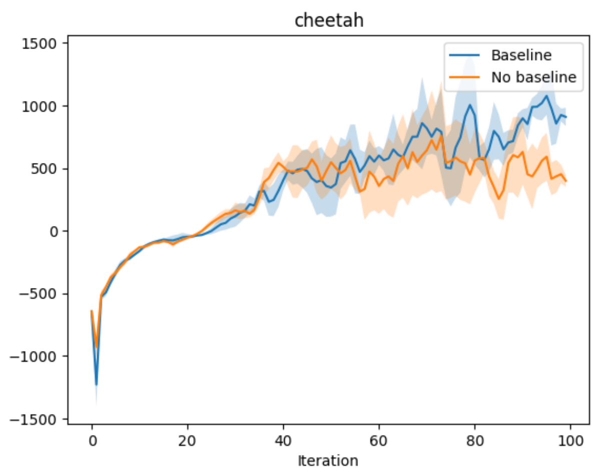
On the following page we provide sample outputs and an outline of the performance you can expect for the correct implementation.

Expected Results

We expect your plots to look similar to the plots which we have included:







The following performance benchmarks need to be achieved by your implementation in order to recieve full credit for this question:

- CartPole-v0: agent should reach the maximum return of 200 with and without baseline (although it may not stay there)
- InvertedPendulum-v2: agent should reach the maximum return of 1000 with and without baseline (although it may not stay there)
- HalfCheetah-v2: agent should reach at least a score of 200 with and without baseline (in general it can achieve a much higher score e.g. 900)

2 Reducing Variance in Policy Gradient Methods

In class, we explored REINFORCE as a policy gradient method with no bias but high variance. In this problem, we will explore methods to dramatically reduce variance in policy gradient methods, potentially at the cost of increased bias.

Let us consider an infinite horizon MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$. Let us define

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

An approximation to the policy gradient is defined as

$$g = \mathbb{E}_{\substack{s_{0:\infty} \\ a_{0:\infty}}}[\sum_{t=0}^{\infty} A^\pi(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t, s_t)]$$

where the colon notation $a : b$ represents the range $[a, a+1, a+2, ...b]$ inclusive of both ends.

(a) **[3 points (Online)]**

Please refer to question 2 of the Gradescope online assessment `A3 (Quiz)`.

(b) **[3 points (Written)]**

Prove that $\text{Var}(R_{t+1}) \geq \text{Var}(R_t)$ is true if we assume that r_{t+1} is, on average, correlated with the previous rewards, i.e. $\frac{1}{t+1}\sum_{i=0}^{t} \text{Cov}(r_i, r_{t+1}) > 0$.

You may find the following properties helpful in proving this result:

$$Var(X + Y) = Var(X) + Var(Y) + 2Cov(X, Y)$$

$$Cov(X + Y, Z) = Cov(X, Z) + Cov(Y, Z)$$

Hint: Try to write the expression for the return at a given timestep as a sum of rewards.

(c) **[5 points (Written)]**

In practice, we do not have access to the true function $A^\pi(s_t, a_t)$, so we would like to obtain an estimate instead. We will consider the general form of an estimator $\hat{A}_t(s_{0:\infty}, a_{0:\infty})$ that can be a function of the entire trajectory.

Consider the following setup:

$$\hat{A}_t(s_{0:\infty}, a_{0:\infty}) = \hat{Q}_t(s_{t:\infty}, a_{t:\infty}) - b_t(s_{0:t}, a_{0:t-1})$$

where,

$$\mathbb{E}_{\substack{s_{t+1:\infty} \\ a_{t+1:\infty}}}[\hat{Q}_t(s_{t:\infty}, a_{t:\infty})] = Q^\pi(s_t, a_t)$$

which indicates that for all s_t, a_t, we have that $\hat{Q}_t$ is an unbiased estimator of the true Q^π.

Also note, that b_t is an arbitrary function of the actions and states sampled before a_t. Prove that by using this estimate of $\hat{A}_t$, we obtain an unbiased estimate of the policy gradient g. In other words, prove the following identity:

$$\mathbb{E}_{\substack{s_{0:\infty} \\ a_{0:\infty}}}[\sum_{t=0}^{\infty} \hat{A}_t(s_{0:\infty}, a_{0:\infty}) \nabla_\theta \log \pi_\theta(a_t, s_t)] = g$$

Note: Recall the following result from class:

$$\mathbb{E}_\tau[(b_t(s_{0:t}, a_{0:t-1})) \nabla_\theta \log \pi_\theta(a_t, s_t)] = 0$$

You may cite this result without proof in your answer. Please consult the a.1 for further details on how we derived this result.

We have also provided you with the first few lines of a proof for this questions to help you get started.

$$\mathbb{E}_{\substack{s_{0:\infty} \\ a_{0:\infty}}}[\sum_{t=0}^{\infty} \hat{A}_t(s_{0:\infty}, a_{0:\infty})\nabla_\theta \ \mathsf{log}\ \pi_\theta(a_t, s_t)]$$
$$= \mathbb{E}_{\substack{s_{0:\infty} \\ a_{0:\infty}}}[\sum_{t=0}^{\infty}(\hat{Q}_t(s_{t:\infty}, a_{t:\infty}) - b_t(s_{0:t}, a_{0:t-1}))\nabla_\theta \ \mathsf{log}\ \pi_\theta(a_t, s_t)]$$
$$= \mathbb{E}_{\substack{s_{0:\infty} \\ a_{0:\infty}}}[\sum_{t=0}^{\infty}(\hat{Q}_t(s_{t:\infty}, a_{t:\infty}))\nabla_\theta \ \mathsf{log}\ \pi_\theta(a_t, s_t)] - \mathbb{E}_{\substack{s_{0:\infty} \\ a_{0:\infty}}}[\sum_{t=0}^{\infty}(b_t(s_{0:t}, a_{0:t-1}))\nabla_\theta \ \mathsf{log}\ \pi_\theta(a_t, s_t)]$$

(d) **[3 points (Written)]**

We will now look at a few different variants of $\hat{A}_t$. Recall the TD error $\delta_t^{\hat{V}}(s_t, a_t) = r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)$. If $\hat{V} = V^\pi$, prove that $\delta_t^{\hat{V}}$ is an unbiased estimate of A^π.

Note: Recall that an estimator $\hat{\theta}$ is an unbiased estimator of θ if $\mathbb{E}[\hat{\theta}] = \theta$.

(e) **[3 points (Written)]**

Let us define $\hat{A}_t^{(k)} = \sum_{i=0}^{k-1} \gamma^i \delta_{t+i}^{\hat{V}}$. Show that $\hat{A}_t^{(k)} = -\hat{V}(s_t) + \gamma^k \hat{V}(s_{t+k}) + \sum_{i=0}^{k-1} \gamma^i r_{t+i}$.

In general, how does bias and variance change as k increases? A few sentences of justification will suffice when describing the changes in variance and bias as we increase k, no formal proof is necessary.

Hint: if you expand the expression for $\hat{A}_t^{(k)}$ you should observe a telescoping sum which can help simplify your proof for the first part of this question.

(f) **[3 points (Written)]**

Show that $\hat{A}_t^{(\infty)} = \sum_{i=0}^{\infty} \gamma^i r_{t+i} - \hat{V}(s_t)$.

Note: you may assume that $0 \leq \gamma < 1$.

3 Appendix

a.1

Statement: $\mathbb{E}_\tau[(b_t(s_{0:t}, a_{0:t-1}))\nabla_\theta \log \pi_\theta(a_t, s_t)] = 0$
Proof:

$$\mathbb{E}_\tau[\nabla_\theta log(\pi(a_t|s_t;\theta))b(s_t)]$$

$$= \mathbb{E}_{s_{0:t},a_0:t-1}[\mathbb{E}_{s_{t+1:T},a_t:T-1}[\nabla_\theta log(\pi(a_t|s_t;\theta))b(s_t)]]$$

$$= \mathbb{E}_{s_{0:t},a_0:t-1}[b(s_t)\mathbb{E}_{s_{t+1:T},a_t:T-1}[\nabla_\theta log(\pi(a_t|s_t;\theta))]]$$

$$= \mathbb{E}_{s_{0:t},a_0:t-1}[b(s_t)\mathbb{E}_{a_t}[\nabla_\theta log(\pi(a_t|s_t;\theta))]]$$

$$= \mathbb{E}_{s_{0:t},a_0:t-1}[b(s_t)\sum_a \pi_\theta(a_t|s_t)\frac{\nabla_\theta \pi(a_t|s_t;\theta)}{\pi_\theta(a_t|s_t)}]$$

$$= \mathbb{E}_{s_{0:t},a_0:t-1}[b(s_t)\sum_a \nabla_\theta \pi(a_t|s_t;\theta)]$$

$$= \mathbb{E}_{s_{0:t},a_0:t-1}[b(s_t)\nabla_\theta \sum_a \pi(a_t|s_t;\theta)]$$

$$= \mathbb{E}_{s_{0:t},a_0:t-1}[b(s_t)\nabla_\theta 1]$$

$$= 0$$

This handout includes space for every question that requires a written response. Please feel free to use it to handwrite your solutions (legibly, please). If you choose to typeset your solutions, the —README.md— for this assignment includes instructions to regenerate this handout with your typeset LaTeX solutions.

2.b

2.c

$$
\begin{aligned}
&\mathbb{E}_{\substack{s_{0:\infty}\\a_{0:\infty}}}[\sum_{t=0}^{\infty}\hat{A}_t(s_{0:\infty},a_{0:\infty})\nabla_\theta \ \mathsf{log}\ \pi_\theta(a_t,s_t)]\\
&=\mathbb{E}_{\substack{s_{0:\infty}\\a_{0:\infty}}}[\sum_{t=0}^{\infty}(\hat{Q}_t(s_{t:\infty},a_{t:\infty})-b_t(s_{0:t},a_{0:t-1}))\nabla_\theta \ \mathsf{log}\ \pi_\theta(a_t,s_t)]\\
&=\mathbb{E}_{\substack{s_{0:\infty}\\a_{0:\infty}}}[\sum_{t=0}^{\infty}(\hat{Q}_t(s_{t:\infty},a_{t:\infty}))\nabla_\theta \ \mathsf{log}\ \pi_\theta(a_t,s_t)]-\mathbb{E}_{\substack{s_{0:\infty}\\a_{0:\infty}}}[\sum_{t=0}^{\infty}(b_t(s_{0:t},a_{0:t-1}))\nabla_\theta \ \mathsf{log}\ \pi_\theta(a_t,s_t)]
\end{aligned}
$$

2.d

2.e

2.f