XCS234 Assignment 1

Due Sunday, December 3 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. For SCPD classes, it is also important that students avoid opening pull requests containing their solution code on the shared assignment repositories. 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, you will be exploring tabular solution methods. In particular, you will apply your understanding of MDPs to solve for the optimal paths an agent can take in a mobile game called *Flappy Karel*. You will then learn to apply the performance difference lemma, which is a useful result used in convergence analysis of policy search methods. Afterwards you will have the opportunity to apply your knowledge of the Bellman operator to a time dependent variation of this operator. By the end of the assignment, you will implement both value iteration and policy iteration to solve for optimal value functions and policies in OpenAI's Frozen Lake gym environment.

Advice for this assignment:

- Remember to activate the conda environment `XCS234_Default` defined in `src/environment.yml` before developing and testing your code.
- In question 1, when considering a unique optimal policy do not count actions from states with a wall directly to the right of them or terminal states.

1 Flappy Karel MDP

There is a hot new mobile game on the market called Flappy Karel, where Karel the robot must dodge the red pillars of doom and flap its way to the green pasture. Karel needs your help in finding the best path to take in different scenarios. Below we have described Flappy World and Karel's capabilities in more detail:

Flappy World

- Each square on the Flappy World grid represents a single state which Karel may occupy
- Squares shaded in red represent terminal states, taking an action from these squares will provide Karel with a reward of r_r and end the episode
- Squares shaded in green represent the goal state and are also terminal, taking an action from these squares will provide Karel with a reward of r_g and end the episode.
- Squares left unshaded represent non-terminal states, taking an action from these squares will provide Karel with a reward of r_s

Karel's Movement

- Karel can move right and up (e.g. starting from square 2 Karel can move to square 8)
- Karel can move right and down (e.g. starting from square 2 Karel can move to square 10)
- There are walls in each version of Flappy World represented by thicker edges. If a move taken by Karel runs into a wall this will result in Karel falling down one square (e.g. going in any direction from square 30 results in falling to square 31)
- Actions are deterministic and always succeed unless they cause Karel to run into a wall

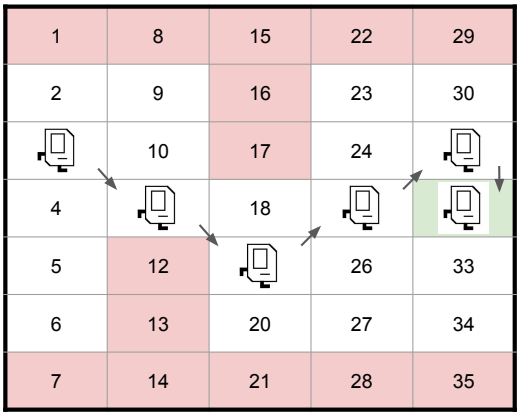
Assumptions (unless we have specified otherwise)

- Discount factor $\gamma = 0.9$
- $r_g = +5$
- $r_r = -5$

We will consider the following two variations of the described environment in the upcoming questions (Flappy World 1 and Flappy World 2).

1	8	15	22	29
2	9	16	23	30
3	10	17	24	31
4	11	18	25	32
5	12	19	26	33
6	13	20	27	34
7	14	21	28	35

(a) Flappy World 1



(b) A successful run by Karel in Flappy World 1

Figure 1

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

Let $r_s \in \{-4, -1, 0, 1\}$. Starting in **square 2**, for each of the possible values of r_s briefly explain what the optimal policy would be in Flappy World 1. In each case is the optimal policy unique and does the optimal policy depend on the value of the discount factor $\gamma \in [0, 1]$? Explain your answer.

Hint: The fact that actions are deterministic should reduce the overall amount of calculation required.

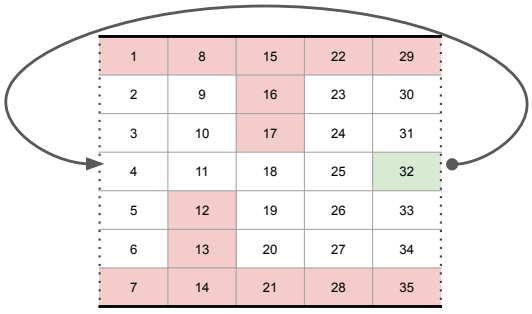
(b) **[5 points (Written)]**

Consider different possible grids and grid shading (with walls at the border similar to Flappy World 1) in which the green target square is reacheable from the starting square.

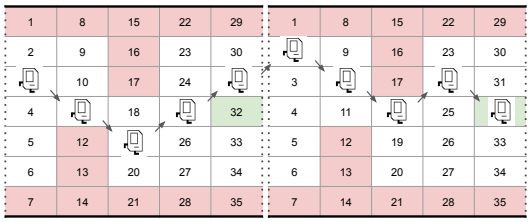
- What value of r_s from part (a) would cause the optimal policy to return the shortest path to the green target square in all cases?
- Find the optimal value function for each square in Flappy World 1 using this value of r_s? *i.e.* show the value functions for each square.
- What is the optimal action from square 27?

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

Now consider Flappy World 2. It is the same as Flappy World 1, except there are no walls on the right and left sides. Going past the right end of Flappy World 2 simply loops you to left hand side. Take a look at Figure 2b for a successful run by Karel in Flappy World 2.



(a) Flappy World 2



(b) A successful run by Karel in Flappy World 2

Figure 2

Let $r_s \in \{-4, -1, 0, 1\}$. Starting in **square 3**, for each of the possible values of r_s:

- Briefly explain what the optimal policy would be in Flappy World 2?

Once again consider different grids and grid shading (without walls at either end similar to Flappy World 2) in which the green target square is reacheable from the starting square.

- What is the value of r_s that would cause the optimal policy to return the shortest path to the green target square for all cases?
- Find the optimal value for each square in Flappy World 2 using the value of r_s, that would cause the optimal policy to return the shortest path to the green target square for all cases?
- What is the optimal action from square 27?

Hint: There are three possible long-term behaviours, terminate through a red square, terminate through a green square and do not ever terminate. Consider these cases when formulating the optimal policy for each value of r_s.

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

Consider a general MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ and in this case assume that the horizon is infinite (so there is no termination). A policy π in this MDP induces a value function V^π (lets refer to this as V^π_{old}). Now suppose we have the same MDP where all rewards have a constant c added to them and then have been scaled by a

constant a (i.e. $r_{\text{new}} = a(c+r)$). Can you come up with an expression for the new value function V^{π}_{new} induced by π in this second MDP in terms of $V^{\pi}_{\text{old}}, c, a$, and γ?

You can start this question with the expression for the original value function and consider how this expression would change for scaled rewards:

$$V^{\pi}_{\text{old}}(s) = \mathbb{E}_{\pi}[G_{\text{old},t} \mid x_t = s]$$

Where the return is defined as the discounted sum of rewards:

$$G_{\text{old},t} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Now consider how we would rewrite the following expression in terms of the original:

$$V^{\pi}_{\text{new}}(s) = \mathbb{E}_{\pi}[G_{\text{new},t} \mid x_t = s]$$

Where $G_{\text{new},t}$ is comprised of the rewards which have been translated and scaled.

(e) **[5 points (Online)]**

Please refer to question 1.1 of the Gradescope online assessment `A1 (Quiz)`.

2 Applications of the Performance Difference Lemma

In many situations such as healthcare or education, we cannot run any arbitrary policy and collect data from running those policies for evaluation. In these cases, we may need to take data collected from following one policy and use it to evaluate the value of a different policy. The equality used in the following exercise can be an important tool for achieving this.

The purpose of this exercise is to get familiar with how to compare the value of different policies, π_1 and π_2, on a fixed horizon MDP. A fixed horizon MDP is an MDP where the agent's state is reset after H timesteps; H is called the *horizon* of the MDP. There is no discount (i.e. $\gamma = 1$) and policies are allowed to be non-stationary, i.e., the action identified by a policy depends on the timestep in addition to the state.

Let $x_t \sim \pi$ denote the distribution over states at timestep t (for $1 \leq t \leq H$) upon following policy π and $V_t^\pi(x_t)$ denote the value function of policy π in state x_t and timestep t, and $Q_t^\pi(x_t, a)$ denote the corresponding Q value associated to action a. As a clarifying example, we denote $\mathbb{E}_{x_t \sim \pi_1}[V_t(x_t)]$ to represent the average value of the value function $V_t(\cdot)$ over the states at timestep t encountered upon following policy π_1. The following equality is called *performance difference lemma* :

$$V_1^{\pi_1}(x_1) - V_1^{\pi_2}(x_1) = \sum_{t=1}^{H} \mathbb{E}_{x_t \sim \pi_2}\Big(Q_t^{\pi_1}(x_t, \pi_1(x_t, t)) - Q_t^{\pi_1}(x_t, \pi_2(x_t, t))\Big) \tag{1}$$

Intuition: The above expression can be interpreted in the following way. For concreteness, assume that π_1 is the better policy, i.e., achieving $V_1^{\pi_1}(x_1) \geq V_1^{\pi_2}(x_1)$. Suppose you're following policy π_2 and you are at timestep t in state x_t. You have the option to follow π_1 (the better policy) until the end of the episode, totalling $Q_t^{\pi_1}(x_t, \pi_1(x_t, t))$ return from the current state-timestep; or you have the option to follow π_2 for one timestep and then follow π_1 instead until the end of the episode (you can follow many other policies of course). This would give you a "loss" of $Q_t^{\pi_1}(x_t, \pi_1(x_t, t)) - Q_t^{\pi_1}(x_t, \pi_2(x_t, t))$ that originates from following the worse policy π_2 instead of π_1 in that timestep. Then the equation above means that the value difference of the two policies is the sum of all the losses induced by following the suboptimal policy for every timestep, weighted by the expected trajectory of the policy you're following.

(a) **[10 points (Written)]**

You will use the performance difference lemma to solve this problem. Consider an MDP where the state space $\mathcal{S}$ is partitioned into two sets of states $\mathcal{S}^+$ and its complement $\overline{\mathcal{S}}^+$.

$$\mathcal{S} = \mathcal{S}^+ \cup \overline{\mathcal{S}}^+$$
$$\mathcal{S}^+ \cap \overline{\mathcal{S}}^+ = \emptyset.$$

In every state $s \in \mathcal{S}^+$ there exists an action a^+ that leads to the same state with probability 1 and gives a unitary reward:

$$p(s_{t+1} = s \mid s_t = s, a_t = a^+) = 1, \qquad p(s_{t+1} \neq s \mid s_t = s, a_t = a^+) = 0$$

The reward function is always positive. In $\mathcal{S}^+$ the reward function equals 1 upon playing a^+ and H (where H is a contant representing the horizon of the MDP) upon playing any action $a \neq a^+$. Therefore in $\mathcal{S}^+$

$$r(s, a^+) = 1, \quad r(s, a) = H, \quad a \neq a^+$$

Conversely, in any state $s \notin S^+$, the reward function is in $[0, 1]$ ($\forall s \notin S^+ \quad \forall a \;\; r(s, a) \in [0, 1]$).

Consider a deterministic policy π and define a policy π^+ that takes action a^+ in any state $\mathcal{S}^+$ and is otherwise equal to π:

$$\pi^+(s) = a^+ \text{ if } s \in \mathcal{S}^+, \qquad \pi^+(s) = \pi(s) \text{ if } s \notin \mathcal{S}^+$$

Intuitively, π accumulates higher return than π^+: in any state in $\mathcal{S}^+$ the policy π^+ chooses to take a unitary reward forever instead of a reward of H and then maybe more. Using the performance difference lemma show

that for any arbitrary initial state s_{start}

$$V_1^{\pi}(s_{\text{start}}) \geq V_1^{\pi^+}(s_{\text{start}}).$$

Through rearranging terms you can think of this problem as proving the following identity $V_1^{\pi^+}(s_{\text{start}}) - V_1^{\pi}(s_{\text{start}}) \leq 0$. From this point you may use the Performance Difference Lemma to understand the LHS of the given inequality.

3 Nonstationary Discount Factor γ

In this problem you will consider a variable discount factor γ. In lectures, we proved that the Bellman backup is a contraction for $\gamma < 1$ in the infinity norm.

In this problem we consider having a non-stationary discount factor and assume you want to run K iterations of value iteration. Let V_K and V'_K be any two arbitrary initial value functions (at timestep K). The time-dependent Bellman backup operator B_k for an arbitrary timestep k is defined as

$$V_{k-1} \stackrel{def}{=} B_k V_k = \max_a [R(s,a) + \gamma_k \sum_{s' \in S} p(s' \mid s,a) V_k(s')] \tag{2}$$

where

$$\gamma_k = 1 - \frac{1}{k+1} \tag{3}$$

Notice that the value function index is decreasing: $K, K-1, \ldots, 2, 1$

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

Similarly to what you've done in class, show that the Bellman operator with non-stationary discount factor at time step k is still a contraction, i.e.,

$$\|B_k V - B_k V'\|_\infty \leq \gamma_k \|V_k - V'_k\|_\infty$$

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

Using the above inequality prove that:

$$\|B_1 B_2 \cdots B_K V_K - B_1 B_2 \cdots B_K V'_K\|_\infty \leq \gamma_1 \gamma_2 \cdots \gamma_K \|V_K - V'_K\|_\infty$$

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

Unfortunately $\gamma_k \approx 1$ when k is large so you cannot conclude that the convergence occurs exponentially fast. However, the error still shrinks. Show that:

$$\gamma_1 \gamma_2 \cdots \gamma_K = \frac{1}{K+1}$$

which allows you to write

$$\|B_1 B_2 \cdots B_K V_K - B_1 B_2 \cdots B_K V'_K\|_\infty \leq \frac{1}{K+1} \|V_K - V'_K\|_\infty$$

and ensure convergence, albeit at a slower rate.

4 Frozen Lake MDP

Now you will implement value iteration and policy iteration for the Frozen Lake environment from OpenAI Gym. We have provided custom versions of this environment in the starter code.

For each of the following questions you may assume that the stopping tolerance (defined as $\max_s \mid V_{old}(s) - V_{new}(s) \mid$) is tol $= 10^{-3}$ and $\gamma = 0.9$.

For this question, the state value function in Eqn. 4 could be used since the state transition probability p is stochastic

$$V_k^\pi(s) = \sum_{s',r} p(s', r|s, a)[r + \gamma V_{k-1}^\pi(s')], \qquad \forall s \in S \tag{4}$$

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

Read through `submission.py` and implement `policy_evaluation`.

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

Read through `submission.py` and implement `policy_improvement`.

(c) **[7 points (Coding)]**

Read through `submission.py` and implement `policy_iteration`.

(d) **[8 points (Coding)]**

Read through `submission.py` and implement `value_iteration`.

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

Please refer to question 2.1 of the Gradescope online assessment `A1 (Quiz)`.

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.

1.a

1.b

1.c

1.d

2.a

3.a

3.b

3.c