CMPSCI 687: Reinforcement Learning
Fall 2019 Class Syllabus, Notes, and Assignments

Professor Philip S. Thomas
University of Massachusetts Amherst
pthomas@cs.umass.edu

Table of Contents

1 Syllabus

1.1 Class

Class will be held on Tuesdays and Thursdays from 4:00pm–5:15pm in Engineering Lab II, Room 119. Lectures will be given primarily on the whiteboard, with typed notes provided below and updated throughout the semester as additional material is covered. These notes are *not* a complete summary of all material that students are responsible for—you are responsible for all material covered in class, even if it is not present in these notes.

1.2 Website

The class website is `https://people.cs.umass.edu/~pthomas/courses/CMPSCI_687_Fall2019.html`. All homework assignments, due dates, and notes will be posted there.

1.3 Book

The start of the course will be roughly based on the *first* edition of Sutton and Barto's book, *Reinforcement Learning: An Introduction*. It can be found on Amazon here. It is also available for free online here. Although the book is a fantastic introduction to the topic (and I encourage purchasing a copy if you plan to study reinforcement learning), owning the book is not a requirement.

1.4 Office Hours

Prof. Thomas' office hours will be Mondays from 1:30pm–3:00pm in his office, room 346 of the computer science building. Office hours will follow the academic calendar: they will be offered from 1:30-3:00pm on (and only on) all days that are a Monday schedule.

1.5 Teaching Assistants and Office Hours

The *teaching assistants* (TAs) this semester will be Blossom Metevier (bmetevier@umass.edu) and Scott Jordan (sjordan@cs.umass.edu). Blossom will have office hours on Fridays from 10:00am–11:00am in CS Building room 207. Scott will have office hours on Tuesdays from 2:45pm–4pm in CS Building room 207.

1.6 Piazza

The course will use Piazza as a forum where you can ask questions. Every afternoon (at some time between 1pm and 6pm), Prof. Thomas or one of the TAs will go through and answer all of the questions on Piazza. If you asked a question before 1pm on a weekday that is not a holiday, and did not get a response by 6pm, please e-mail Prof. Thomas directly (pthomas@cs.umass.edu), as this should not occur.

1.7 Support Summary

Office hours:

- **Mondays**: 1:30pm-3:00pm, Prof. Thomas' office hours, CS room 346.
- **Tuesdays**: 2:45pm–4:00pm, Scott Jordan's office hours, CS room 207.
- **Friday**: 10:00am–11:00am, Blossom Metevier's office hours, CS room 207.

Piazza question answering (some time between 1pm and 6pm):

- **Mondays**: Blossom Metevier
- **Tuesdays**: Scott Jordan
- **Wednesdays**: Blossom Metevier
- **Thursdays**: Prof. Thomas
- **Fridays**: Scott Jordan

1.8 Grading

Your grade will have four components:

1. **Homework Assignments (**50%**)**: There will be *roughly* seven homework assignments. Each problem in an assignment will specify its point value, and not all homework assignments will necessarily have the same point value (i.e., some homework assignments may be smaller and worth less than others). All assignments will have total point values of at most 100 (thus, the last assignment will not have a point value so high that previous assignments are irrelevant).
2. **Pop Quizzes (**15%**)**: There will be pop-quizzes given in class without prior announcement. They will typically take about 10 minutes to complete and will be given at the start of class. If you know in advance that you will miss class, please e-mail both TAs, and you may be excused from any quizzes that occur that day.
3. **Midterm Exam** (15%): There will be a midterm exam in class (the regular room and time) on Tuesday November 19.
4. **Project** (20%): There will be a course project. The details of the project will be announced later in the semester, and may depend on how much content is covered.

A cumulative grade in $[90\% - 100\%]$ will be an A- or A, $[80\%, 90\%)$ will be a B-, B, or B+, and $[70\%, 80\%)$ will be a C-, C, or C+. Course grades will be curved only in students' favor (that is, these thresholds may be lowered, but a grade of 90% will not be lower than an A-).

1.9 Late Policy

Late homework assignments will not be accepted. An assignment submitted one minutes late is late, and will not be accepted. I recommend submitting homework well in advance of the due date and time.

1.10 Missing Class / Assignments

If you are going to miss class, e-mail the TAs (not Prof. Thomas) *before* the start of class letting them know. You will then be excused from any pop-quizzes that occur on that day (your grade will be computed as though that quiz did not occur).

Sometimes things come up that prevent you from completing an assignment well or at all. To handle this, **your homework assignment with the lowest score will be dropped**. To avoid encouraging skipping the final assignment, if you perform consistently on *all* assignments without any clearly low outliers, Prof. Thomas will consider this when assigning grades (it may bump you up if you're near a boundary).

1.11 Disability Services

If you have a disability and require accommodations, please let me know as soon as possible. You will need to register with Disability Services (161 Whitemore Administration Building; phone (413) 545–0892). Information on services and materials for registering are also available on their website: www.umass.edu/disability.

1.12 Cheating

Cheating will not be tolerated. Each assignment includes instructions about what forms of collaboration are allowed. Copying answers or code from online sources or from solutions to assignments from previous years is always considered cheating. All instances of cheating will be reported to the university's Academic Honesty Board, and will result in a failing grade letter grade for the course.

1.13 LaTeX

Your homework submissions must be typed using LaTeX. If you have not used LaTeX before, you may want to complete an online tutorial now. Also, the instructor and TAs are prepared to help you learn about LaTeX during their office hours. Note: The formatting of math using editors like Microsoft Word is *not* as clear as LaTeX. Assignments created using other editors will not be accepted.

2 Introduction

2.1 Notation

When possible, sets will be denoted by calligraphic capital letters (e.g., $\mathcal{X}$), elements of sets by lowercase letters (e.g., $x \in \mathcal{X}$), random variables by capital letters (e.g., X), and functions by lowercase letters (e.g., f). This will not always be possible, so keep an eye out for exceptions (e.g., later P will be a function).

We write $f : \mathcal{X} \to \mathcal{Y}$ to denote that f is a function with domain $\mathcal{X}$ and range $\mathcal{Y}$. That is, it takes as input an element of the set $\mathcal{X}$ and produces as output an element of $\mathcal{Y}$. We write $|\mathcal{X}|$ to denote the cardinality of the set $\mathcal{X}$—the number of elements in $\mathcal{X}$, and $|x|$ to denote the absolute value of x (thus the meaning of $|\cdot|$ depends on context).

We typically use capital letters for matrices (e.g., A) and lowercase letters for vectors (e.g., b). We write $A^\intercal$ to denote the transpose of A. Vectors are assumed to be column vectors. Unless otherwise specified, $\|b\|$ denotes the l^2-norm (Euclidean norm) of the vector v.

We write $\mathbb{N}_{>0}$ to denote the natural numbers *not* including zero, and $\mathbb{N}_{\geq 0}$ to denote the natural numbers including zero.

We write $\coloneqq$ to denote *is defined to be*. In lecture we may write $\triangleq$ rather than $\coloneqq$ since the triangle is easier to see when reading handwriting from the back of the room.

If $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ for any sets $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$, then we write $f(\cdot, y)$ to denote a function, $g : \mathcal{X} \to \mathcal{Z}$, such that $g(x) = f(x, y)$ for all $x \in \mathcal{X}$.

We denote sets using brackets, e.g., $\{1, 2, 3\}$, and sequences and tuples using parentheses, e.g., $(x_1, x_2, \dots)$.

The notation that we use is *not* the same as that of the book or other sources (papers and books often use different notations, and there is no agreed-upon standard). Our notation is a mix between the notations of the first and second editions of Sutton and Barto's book.

2.2 What is Reinforcement Learning (RL)?

> *Reinforcement learning is an area of machine learning, inspired by behaviorist psychology, concerned with how an agent can learn from interactions with an environment.*

–Wikipedia, Sutton and Barto (1998), Phil

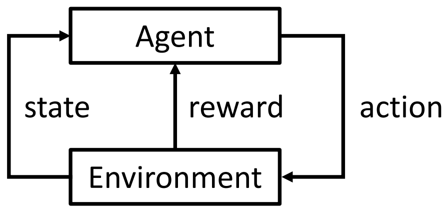


Figure 1: Agent-environment diagram.

Agent: Child, dog, robot, program, etc.

Environment: World, lab, software environment, etc.

Evaluative Feedback: Rewards convey how "good" an agent's actions are, not what the best actions would have been. If the agent was given instructive feedback (what action it should have taken) this would be a *supervised learning* problem, not a reinforcement learning problem.

Sequential: The entire sequence of actions must be optimized to maximize the "total" reward the agent obtains. This might require forgoing immediate rewards to obtain larger rewards later. Also, the way that the agent makes decisions (selects actions) changes the distribution of states that it sees. This means that RL problems aren't provided as fixed data sets like in supervised learning, but instead as code or descriptions of the entire environment.

Question 1. *If the agent-environment diagram describes a child learning to walk, what exactly is the "Agent" block? Is it the child's brain, and its body is part of the environment? Is the agent the entire physical child? If the diagram describes a robot, are its sensors part of the environment or the agent?*

Neuroscience and *psychology* ask how animals learn. It is the study of some examples of learning and intelligence. Reinforcement learning asks how we can make an agent that learns. It is the study of learning and intelligence in general (animal, computer, match-boxes, purely theoretical, etc.). In this course we may discuss the relationship between RL and computational neuroscience in one lecture, but in general will *not* concern ourselves with how animals learn (other than, perhaps, for intuition and motivation).

There are many other fields that are similar and related to RL. Separate research fields often do not communicate much, resulting in different language and approaches. Other notable fields related to RL include operations research and control (classical, adaptive, etc.). Although these fields are similar to RL, there are often subtle but impactful differences between the problems studied in these other fields and in RL. Examples include whether the dynamics of the environment are known to the agent *a priori* (they are not in RL), and whether the dynamics of the environment will be estimated by the agent (many, but not all, RL agents do not directly estimate the dynamics of the environment). There are also many less-impactful differences, like differences in notation (in control, the environment is called the *plant*, the agent the *controller*, the reward the (negative) *cost*, the state the *feedback*, etc.).

A common misconception is that RL is an alternative to supervised learning—that one might take a supervised learning problem and convert it into an RL problem in order to apply sophisticated RL methods. For example, one might treat the state as the input to a classifier, the action as a label, and the reward as -1 if the label is correct and 1 otherwise. Although this is technically possible and a valid use of RL, it *should not be done*. In a sense, RL should be a last resort—the tool that you use when supervised learning algorithms cannot solve the problem you are interested in. If you have labels for your data, do *not* discard them and convert the feedback from instructive feedback (telling the agent what label it should have given) to evaluative feedback (telling the agent if it was right or wrong). The RL methods will likely be far worse than standard supervised learning algorithms. However, if you have a sequential problem or a problem where only evaluative feedback is available (or both!), then you cannot apply supervised learning methods and you should use RL.

Question 2. ***[Puzzle]*** *There are 100 pirates. They have* 10,000 *gold pieces. These pirates are ranked from most fearsome (1) to least fearsome (100). To divide the gold, the most fearsome pirate comes up with a method (e.g., split it evenly, or I get half and the second most fearsome gets the other half). The pirates then vote on this plan. If* 50% *or more vote in favor of the plan, then that is how the gold is divided. If* $>$ 50% *vote against the plan, the most fearsome pirate is thrown off the boat and the next most fearsome comes up with a plan, etc.* The pirates are perfectly rational. *You are the most fearsome pirate. How much of the gold can you get? How?*

Answer 2. *You should be able to keep* 9,951 *pieces of gold.*

If you solved the above puzzle, you very likely did so by first solving easier versions. What if there were only two pirates? What if there were three? This is what we will do in this course. We will study and understand an easier version of the problem and then will build up to more complex and interesting cases over the semester.

End of Lecture 1, September 3, 2019

2.3 687-Gridworld: A Simple Environment

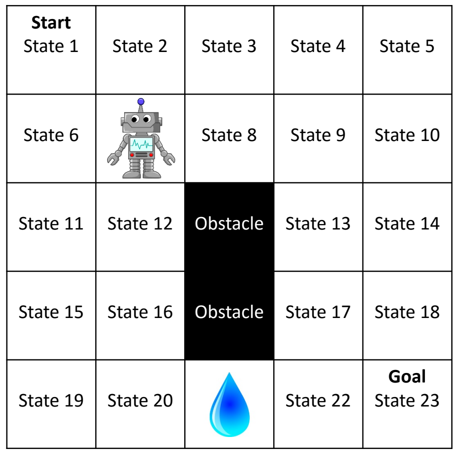


Figure 2: 687-Gridworld, a simple example environment we will reference often.

State: Position of robot. The robot does not have a direction that it is facing.

Actions: `Attempt_Up`, `Attempt_Down`, `Attempt_Left`, `Attempt_Right`. We abbreviate these as: AU, AD, AL, AR.

Environment Dynamics: With probability 0.8 the robot moves in the specified direction. With probability 0.05 it gets confused and veers to the right—moves $+90°$ from where it attempted to move (that is, AU results in the robot moving right, AL results in the robot moving up, etc.). With probability 0.05 it gets confused and veers to the left—moves $-90°$ from where it attempted to move (that is, AU results in the robot moving left, AL results in the robot moving down, etc.). With probability 0.1 the robot temporarily breaks and does not move at all. If the movement defined by these dynamics would cause the agent to exit the grid (e.g., move up from state 2) or hit an obstacle (e.g., move right from state 12), then the agent does not move. The robot starts in state 1, and the process ends when the robot reaches state 23.

Rewards: The agent receives a reward of -10 for entering the state with the water and a reward of $+10$ for entering the goal state. Entering any other state results in a reward of zero. If the agent is in the state with the water (state 21) and stays in state 21 for any reason (hitting a wall, temporarily breaking), it counts as "entering" the water state again and results in an additional reward of -10. We use a reward discount parameter (the purpose of which is described later) of $\gamma = 0.9$.

2.4 Describing the Agent and Environment Mathematically

In order to reason about learning, we will describe the environment (and soon the agent) using math. Of the many different mathematical models that can be used to describe the environment (POMDPs, DEC-POMDPs, SMDPs, etc.), we will initially focus on *Markov decision processes* (MDPs). Despite their apparent simplicity, we will see that they capture a wide range of real and interesting problems, including problems that might at first appear to be outside their scope (e.g., problems where the agent makes observations about the state using sensors that might be incomplete and noisy descriptions of the state). Also, a common misconception is that RL is only about MDPs. This is not the case: MDPs are just one way of formalizing the environment of an RL problem.

- An MDP is a mathematical specification of both the environment and what we want the agent to learn.
- Let $t \in \mathbb{N}_{\geq 0}$ be the *time step* (iteration of the agent-environment loop).
- Let S_t be the state of the environment at time t.
- Let A_t be the action taken by the agent at time t.
- Let $R_t \in \mathbb{R}$ be the reward received by the agent at time t. That is, when the state of the environment is S_t, the agent takes action A_t, and the environment transitions to state S_{t+1}, the agent receives the reward R_t. This differs from some other sources wherein this reward is called R_{t+1}.

Formally, a finite MDP is a tuple, $(\mathcal{S}, \mathcal{A}, P, d_R, d_0, \gamma)$, where:

- $\mathcal{S}$ is the set of all possible states of the environment. The state at time t, S_t, always takes values in $\mathcal{S}$. For now we will assume that $|\mathcal{S}| < \infty$—that the set of states is finite.
- $\mathcal{A}$ is the set of all possible actions the agent can take. The action at time t, A_t, always takes values in $\mathcal{A}$. For now we will assume that $|\mathcal{A}| < \infty$.
- P is called the *transition function*, and it describes how the state of the environment changes.

  $$P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]. \tag{1}$$

  For all $s \in \mathcal{S}$, $a \in \mathcal{A}$, $s' \in \mathcal{S}$, and $t \in \mathbb{N}_{\geq 0}$:

  $$P(s, a, s') \coloneqq \Pr(S_{t+1} = s' | S_t = s, A_t = a). \tag{2}$$

  Hereafter we suppress the sets when writing quantifiers (like $\exists$ and $\forall$)—these should be clear from context. We say that the transition function is *deterministic* if $P(s, a, s') \in \{0, 1\}$ for all s, a, and s'. Recall that we will use lower-case letters to denote functions when possible—notice that P is an exception to this rule (due to historical usage and to avoid using p, a commonly used symbol otherwise).

- d_R describes how rewards are generated. Intuitively, it is a conditional distribution over R_t given S_t, A_t, and S_{t+1}. For now we assume that the rewards are bounded—that $|R_t| \leq R_{\max}$ always, for all $t \in \mathbb{N}_{\geq 0}$ and some constant $R_{\max} \in \mathbb{R}$.[1]

- R is a function called the *reward function*, which is implicitly defined by d_R. Other sources often define an MDP to contain R rather than d_R. Formally

$$R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \tag{3}$$

  and

$$R(s, a) := \mathbf{E}\left[R_t | S_t = s, A_t = a\right], \tag{4}$$

  for all s, a, and t. Although the reward function, R, does not precisely define how the rewards, R_t, are generated (and thus a definition of an MDP with R in place of d_R would in a way be incomplete), it is often all that is necessary to reason about how an agent should act. Like P, notice that R is a function despite being a capital letter. This is also due to a long history of this notation, and also because we will use r to denote a particular reward, e.g., when writing (s, a, r, s', a') later.

- d_0 is the *initial state distribution*:

$$d_0 : \mathcal{S} \to [0, 1], \tag{5}$$

  and for all s:

$$d_0(s) = \Pr(S_0 = s). \tag{6}$$

- $\gamma \in [0, 1]$ is a parameter called the *reward discount parameter*, and which we discuss later.

——————————End of Lecture 2, September 5, 2019——————————

Just as we have defined the environment mathematically, we now define the agent mathematically. A *policy* is a decision rule—a way that the agent can select actions. Formally, a policy, π, is a function:

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1], \tag{7}$$

and for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $t \in \mathbb{N}_{\geq 0}$,

$$\pi(s, a) := \Pr(A_t = a | S_t = s). \tag{8}$$

Thus, a policy is the conditional distribution over actions given the state. That is, π is not a distribution, but a collection of distributions over the action set—one per state. There are an infinite number of possible policies, but a finite number of *deterministic* policies (policies for which $\pi(s, a) \in \{0, 1\}$ for all s and a). We

	AU	AD	AL	AR
1	0	0.1	0.3	0.6
2	0.8	0	0	0.2
3	0.1	0.1	0.5	0.3
4	0.25	0.25	0.25	0.25
5	0.25	0.25	0.5	0
6	0.2	0.3	0.5	0
…	…	…	…	…

Figure 3: Example of a tabular policy. Each cell denotes the probability of the action (specified by the column) in each state (specified by the row). In this format, Π is the set of all $|\mathcal{S}| \times |\mathcal{A}|$ matrices with non-negative entries and rows that all sum to one.

denote the set of all policies by Π. Figure 3 presents an example of a policy for 687-Gridworld.

To summarize so far, the interaction between the agent and environment proceeds as follows (where $R_t \sim d_R(S_t, A_t, S_{t+1}, \cdot)$ denotes that R_t is sampled according to d_R):

$$S_0 \sim d_0 \tag{9}$$
$$A_0 \sim \pi(S_0, \cdot) \tag{10}$$
$$S_1 \sim P(S_0, A_0, \cdot) \tag{11}$$
$$R_0 \sim d_R(S_0, A_0, S_1, \cdot) \tag{12}$$
$$A_1 \sim \pi(S_1, \cdot) \tag{13}$$
$$S_2 \sim P(S_1, A_1, \cdot) \tag{14}$$
$$\dots \tag{15}$$

In pseudocode:

Algorithm 1: General flow of agent-environment interaction.

```
S_0 ~ d_0;
for t = 0 to ∞ do
    A_t ~ π(S_t, ·);
    S_{t+1} ~ P(S_t, A_t, ·);
    R_t ~ d_R(S_t, A_t, S_{t+1}, ·);
```

The running of an MDP is also presented as a Bayesian network in Figure 4.

Notice that we have defined rewards so that R_0 is the first reward, while Sutton and Barto (1998) define rewards such that R_1 is the first reward. We do this because S_0, A_0, and $t = 0$ are the first state, action, and time, and so

[1] Sometimes d_R will place probabilities on a small number of rewards (often the reward may be a deterministic function of S_t, A_t, and S_{t+1}). Sometimes d_R will characterize a continuous distribution. Defining d_R properly therefore requires the use of measure theory for probability. To keep things simple, we will not do this—we will not define d_R more formally.

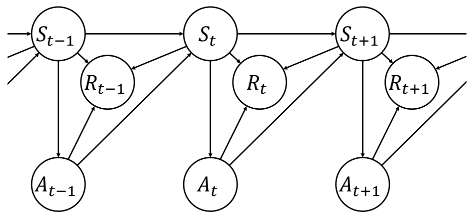


Figure 4: Bayesian network depicted the running of an MDP.

having R_1 be the first reward would be inconsistent. Furthermore, this causes indices to align better later on. However, when comparing notes from the course to the book, be sure to account for this notational discrepancy.

Agent's goal: Find a policy, $\pi^\star$, called an *optimal policy*. Intuitively, an optimal policy maximizes the expected total amount of reward that the agent will obtain.

Objective function: $J : \Pi \to \mathbb{R}$, where for all $\pi \in \Pi$,

$$J(\pi) \coloneqq \mathbf{E}\left[\sum_{t=0}^{\infty} R_t \middle| \pi\right]. \tag{16}$$

Note: Later we will revise this definition—if you are skimming looking for the correct definition of J, it is in (18).

Note: Expectations and probabilities can be conditioned on *events*. A policy, π, is not an event. Conditioning on π, e.g., when we wrote $|\pi$ in the definition of J above, denotes that all actions (the distributions or values of which are not otherwise explicitly specified) are sampled according to π. That is, for all $t \in \mathbb{N}_{\geq 0}$, $A_t \sim \pi(S_t, \cdot)$.

Optimal Policy: An optimal policy, $\pi^\star$, is any policy that satisfies:

$$\pi^\star \in \arg\max_{\pi \in \Pi} J(\pi). \tag{17}$$

Note: Much later we will define an optimal policy in a different and more strict way.

Property 1 (Existence of an optimal policy)**.** *If $|\mathcal{S}| < \infty$, $|\mathcal{A}| < \infty$, $R_{max} < \infty$, and $\gamma < 1$, then an optimal policy exists.*[2]

We will prove Property 1 later.

[2]The restriction of γ is not necessary, but is present because our proof of this property will rely on the assumption that $\gamma < 1$.

Question 3. *Is the optimal policy always unique when it exists?*

Answer 3. *No. For example, in 687-Gridworld (if actions always succeeded), then AD and AR would both be equally "good" in state 1, and so any optimal policy could be modified my shifting probability from AD to AR (or vice versa) in state 1 and the resulting policy would also be optimal.*

Reward Discounting: If you could have one cookie today or two cookies on the last day of class, which would you pick? Many people pick one cookie today when actually presented with these options. This suggests that rewards that are obtained in the distant future are worth less to us than rewards in the near future. The reward discount parameter, γ, allows us to encode, within the objective function, this discounting of rewards based on how distant in the future they occur.

Recall that $\gamma \in [0, 1]$. We redefine the objective function, J, as:

$$J(\pi) \coloneqq \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| \pi\right], \tag{18}$$

for all $\pi \in \Pi$. So, $\gamma < 1$ means that rewards that occur later are worth less to the agent—the utility of a reward, r, t time steps in the future is $\gamma^t r$. Including γ also ensures that $J(\pi)$ is bounded, and later we will see that smaller values of γ make the MDP easier to solve (*solving* an MDP refers to finding or approximating an optimal policy).

To summarize, the agent's goal is to find (or approximate) an optimal policy, $\pi^\star$, as defined in (17), using the definition of J that includes reward discounting—(18).

Question 4. *What is an optimal policy for 687-Gridworld? Is it unique? How does the optimal action in state 20 change if we were to change the value of γ?*

Question 5. *Consider two MDPs that are identical, except for their initial state distributions, d_0. Let π^* and μ^* be optimal policies for the first and second MDP, respectively. Let $s^\star \in \mathcal{S}$ be a state that has a non-zero probability of occurring when using π^* on the first MDP and a non-zero probability of occurring when using μ^* on the second MDP. Consider a new policy, π' such that $\pi'(s, a) = \pi^\star(s, a)$ for all $s \in \mathcal{S} \setminus \{s^\star\}$ and $a \in \mathcal{A}$ and $\pi'(s^\star, a) = \mu^*(s^\star, a)$ for all $a \in \mathcal{A}$. Is π' an optimal policy for the first MDP?*

Answer 5. *Yes! Later we will have the mathematical tools to discuss this more formally. For now, notice that how the agent entered state s^* does not impact what action it should take in state s^* to get as much reward as possible in the future. In this way, optimal behavior in a state is independent of how the agent got to the state, and thus independent of the initial state distribution, d_0, if every state is reachable under every policy. Later we will present a different definition of an optimal policy that is completely independent of d_0 without this reachability condition.*

When we introduced 687-Gridworld, we said that the agent-environment interactions terminate when the agent reaches state 23, which we called the goal. This notion of a *terminal state* can be encoded using our definition of an MDP above. Specifically, we define a terminal state to be any state that *always* transitions to a special state, s_∞, called the *terminal absorbing state*. Once in s_∞, the agent can never leave (s_∞ is *absorbing*)—the agent will forever continue to transition from s_∞ back into s_∞. Transitioning from s_∞ to s_∞ always results in a reward of zero. Effectively, when the agent enters a terminal state the process ends. There are no more decisions to make (since all actions have the same outcome) or rewards to collect. Thus, an episode *terminates* when the agent enters s_∞. Notice that terminal states are optional—MDPs need not have any terminal states. Also, there may be states that only sometimes transition to s_∞, and we do not call these terminal states. Notice also that s_∞ is an element of $\mathcal{S}$. Lastly, although terminal states are defined, *goal states* are *not* defined—the notion of a goal in 687-Gridworld is simply for our own intuition.

When the agent reaches s_∞, the current trial, called an *episode* ends and a new one begins. This means that t is reset to zero, the initial state, S_0, is sampled from d_0, and the next episode begins (the agent selects A_0, gets reward R_0, and transitions to state S_1). The agent is notified that this has occurred, since this reset may change its behavior (e.g., it might clear some sort of short-term memory).

——————End of Lecture 3, September 10, 2019——————

CMPSCI 687 Homework 1

Due September 19, 2019, 11:55pm Eastern Time

Instructions: This homework assignment consists of a written portion and a programming portion. While you may discuss problems with your peers (e.g., to discuss high-level approaches), you must answer the questions on your own. Submissions must be typed (hand written and scanned submissions will not be accepted). You must use LaTeX. The assignment should be submitted on Gradescope as PDF with marked answers via the Gradescope interface. The source code should be submitted via the Gradescope programming assignment as a .zip file. Include with your source code instructions for how to run your code. You **must** use Python 3 for your homework code. You may not use any reinforcement learning or machine learning specific libraries in your code, e.g., TensorFlow or PyTorch (you may use libraries like numpy and matplotlib though). The automated system will not accept assignments after 11:55pm on September 19. The tex file for this homework can be found here.

Part One: Written (62 Points Total)

1. (Your grade will be a zero on this assignment if this question is not answered correctly) Read the class syllabus carefully, including the academic honesty policy. To affirm that you have read the syllabus, type your name as the answer to this problem.

2. (15 Points) Given an MDP $M = (\mathcal{S}, \mathcal{A}, P, d_R, d_0, \gamma)$ and a fixed policy, π, the probability that the action at time $t = 0$ is $a \in \mathcal{A}$ is:

$$\Pr(A_0 = a) = \sum_{s \in \mathcal{S}} d_0(s)\pi(s, a). \tag{19}$$

   Write similar expressions (using only $\mathcal{S}, \mathcal{A}, P, R, d_0, \gamma$, and π) for the following problems.

   Hints and Probability Review:

   - **Write Probabilities of Events:** In some of the probability hints below that are not specific to RL, we use expressions like $\Pr(a|b)$, where a and b are events. Remember that in the RL notation used for this class, the values of $\Pr(s_0)$, $\Pr(a_0)$, $\Pr(A_0)$, or $\Pr(A_0|S_0)$ are all undefined, since those are simply states, actions, or random variables (not events). Instead, we must write about the probabilities of events. For example: $\Pr(A_0 = a_0)$ or $\Pr(A_0 = a_0|S_0 = s_0)$.
   - **Bayes' Theorem:** $\Pr(a|b) = \frac{\Pr(b|a)\Pr(a)}{\Pr(b)}$. This is useful for dealing with conditional probabilities $\Pr(a|b)$, where event a occurs before event b. For example, it is often difficult to work with an expression like $\Pr(S_0 = s_0|A_0 = a_0)$, but much easier to deal with the 3 terms in $\frac{\Pr(A_0=a_0|S_0=s_0)\Pr(S_0=s_0)}{\Pr(A_0=a_0)}$.

- **The law of total probability:** For event a, and a set of events $\mathcal{B}$,

$$\Pr(a) = \sum_{b \in \mathcal{B}} \Pr(b) \Pr(a|b)$$

See the example below for several useful applications of this property.

- **"Extra" given terms:** Remember that when applying laws of probability, any "extra" given terms stay in the result. For example, applying the law of total probability:

$$\Pr(a|c,d) = \sum_{b \in \mathcal{B}} \Pr(b|c,d) \Pr(a|b,c,d)$$

- **Example problem:** The probability that the state at time $t = 1$ is $s \in \mathcal{S}$.

$$\begin{aligned}
\Pr(S_1 = s) &= \sum_{s_0 \in \mathcal{S}} \Pr(S_0 = s_0) \Pr(S_1 = s|S_0 = s_0) && (20)\\
&= \sum_{s_0 \in \mathcal{S}} d_0(s_0) \Pr(S_1 = s|S_0 = s_0) && (21)\\
&= \sum_{s_0 \in \mathcal{S}} d_0(s_0) \sum_{a_0 \in \mathcal{A}} \Pr(A_0 = a_0|S_0 = s_0) && (22)\\
&\quad \times \Pr(S_1 = s|S_0 = s_0, A_0 = a_0) && (23)\\
&= \sum_{s_0 \in \mathcal{S}} d_0(s_0) \sum_{a_0 \in \mathcal{A}} \pi(s_0, a_0) P(s_0, a_0, s). && (24)
\end{aligned}$$

Problems:

A The probability that the action at time $t = 3$ is either $a \in \mathcal{A}$ or $a' \in \mathcal{A}$, with $a \neq a'$.

$$\Pr(A_3 = a \cup A_3 = a') = \sum_{s_3} \Pr(A_3 = a \cup A_3 = a' | S_3 = s_3) \Pr(S_3 = s_3) \tag{25}$$

$$= \sum_{s_3} \Pr(S_3 = s_3)(\pi(s_3, a) + \pi(s_3, a')) \tag{26}$$

$$= \sum_{s_2, a_2} \Pr(S_2 = s_2) \Pr(A_2 = a_2 | S_2 = s_2) \tag{27}$$

$$\times \sum_{s_3} \Pr(S_3 = s_3 | S_2 = s_2, A_2 = a_2)(\pi(s_3, a) + \pi(s_3, a')) \tag{28}$$

$$= \sum_{s_2} \Pr(S_2 = s_2) \sum_{a_2} \pi(s_2, a_2) \times \sum_{s_3} P(s_2, a_2, s_3)(\pi(s_3, a) + \pi(s_3, a')) \tag{29}$$

$$= \sum_{s_0} \Pr(S_0 = s_0) \sum_{a_0} \Pr(A_0 = a_0 | S_0 = s_0) \sum_{s_1} \Pr(S_1 = s_1 | S_0 = s_0, A_0 = a_0) \tag{30}$$

$$\times \sum_{a_1} \Pr(A_1 = a_1 | S_1 = s_1) \sum_{s_2} \Pr(S_2 = s_2 | S_1 = s_1, A_1 = a_1) \tag{31}$$

$$\times \sum_{a_2} \pi(s_2, a_2) \sum_{s_3} P(s_2, a_2, s_3)(\pi(s_3, a) + \pi(s_3, a')) \tag{32}$$

$$= \sum_{s_0} d_0(s_0) \sum_{a_0} \pi(s_0, a_0) \sum_{s_1} P(s_0, a_0, s_1) \tag{33}$$

$$\times \sum_{a_1} \pi(s_1, a_1) \sum_{s_2} P(s_1, a_1, s_2) \tag{34}$$

$$\times \sum_{a_2} \pi(s_2, a_2) \sum_{s_3} P(s_2, a_2, s_3)(\pi(s_3, a) + \pi(s_3, a')) \tag{35}$$

B The expected reward at time $t = 6$ given that the action at time $t = 5$

is $a \in \mathcal{A}$ and the state at time $t = 4$ is $s \in \mathcal{S}$.

$$\mathbf{E}\left[R_6 | A_5 = a, S_4 = s\right] = \sum_{s_6, a_6} R(s_6, a_6) \Pr(S_6 = s_6, A_6 = a_6 | S_4 = s, A_5 = a) \tag{36}$$

$$= \sum_{s_6, a_6} R(s_6, a_6) \frac{\Pr(S_6 = s_6, A_6 = a_6, A_5 = a | S_4 = s)}{\Pr(A_5 = a | S_4 = s)} \tag{37}$$

$$= \sum_{a_4} \pi(s, a_4) \sum_{s_5} P(s, a_4, s_5) \pi(s_5, a) \sum_{s_6, a_6} R(s_6, a_6) \frac{P(s_5, a, s_6)\pi(s_6, a_6)}{\Pr(A_5 = a | S_4 = s)} \tag{38}$$

$$= \sum_{a_4} \pi(s, a_4) \sum_{s_5} P(s, a_4, s_5) \pi(s_5, a) \sum_{s_6, a_6} R(s_6, a_6) \frac{P(s_5, a, s_6)\pi(s_6, a_6)}{\sum_{a_4} \pi(s, a_4) \sum_{s_5} P(s, a_4, s_5) \pi(s_5, a)} \tag{39}$$

$$= \frac{\sum_{a_4} \pi(s, a_4) \sum_{s_5} P(s, a_4, s_5) \pi(s_5, a) \sum_{s_6, a_6} P(s_5, a, s_6) \pi(s_6, a_6) R(s_6, a_6)}{\sum_{a_4} \pi(s, a_4) \sum_{s_5} P(s, a_4, s_5) \pi(s_5, a)} \tag{40}$$

C The probability that the action at time $t = 16$ is $a' \in \mathcal{A}$ given that the action at time $t = 14$ is $a \in \mathcal{A}$, and the state at time $t = 15$ is $s \in \mathcal{S}$.

$$\Pr(A_{16} = a' | S_{15} = s, A_{14} = a) = \Pr(A_{16} = a' | S_{15} = s) \tag{41}$$

$$= \sum_{a_{15}} \Pr(A_{15} = a_{15} | S_{15} = s) \sum_{s_{16}} \Pr(S_{16} = s_{16} | S_{15} = s, A_{15} = a_{15}) \Pr(A_{16} = a' | S_{16} = s_{16}) \tag{42}$$

$$= \sum_{a_{15}} \pi(s, a_{15}) \sum_{s_{16}} P(s, a_{15}, s_{16}) \pi(s_{16}, a') \tag{43}$$

D The probability that the initial state was $s \in \mathcal{S}$ given that the action at time $t = 1$ is $a' \in \mathcal{A}$.

$$\Pr(S_0 = s | A_1 = a') = \frac{\Pr(S_0 = s, A_1 = a')}{\Pr(A_1 = a')} \tag{44}$$

$$\frac{\Pr(S_0 = s) \sum_{a_0} \Pr(A_0 = a_0 | S_0 = s) \sum_{s_1} \Pr(S_1 = s_1 | S_0 = s, A_0 = a_0) \Pr(A_1 = a' | S_1 = s_1)}{\sum_{s_0} \Pr(S_0 = s_0) \sum_{a_0} \Pr(A_0 = a_0 | S_0 = s_0) \sum_{s_1} \Pr(S_1 = s_1 | S_0 = s_0, A_0 = a_0) \Pr(A_1 = a' | S_1 = s_1)} \tag{45}$$

$$= \frac{d_0(s) \sum_{a_0} \pi(s, a_0) \sum_{s_1} P(s, a_0, s_1) \pi(s_1, a')}{\sum_{s_0} d_0(s_0) \sum_{a_0} \pi(s_0, a_0) \sum_{s_1} P(s_0, a_0, s_1) \pi(s_1, a')} \tag{46}$$

E This question has been removed. You do not need to submit an answer.

3. (3 Points) In 687-Gridworld, if we changed how rewards are generated so that hitting a wall (i.e., when the agent would enter an obstacle state, and is placed back where it started) results in a reward of -10, then what is $\mathbf{E}[R_t|S_t = 17, A_t = \text{AL}, S_{t+1} = 17]$?

   Consider the event HitWall, which occurs if the agent hits the obstacle to the left of state 17, and does not occur if the agent does not hit the obstacle to the left of state 17. For brevity, we write 17AL17 to denote $S_t = 17, A_t = \text{AL}, S_{t+1} = 17$.

$$
\begin{aligned}
&\mathbf{E}[R_t|S_t = 17, A_t = \text{AL}, S_{t+1} = 17] && (47)\\
=&\Pr(\text{HitWall}|\text{17AL17})\mathbf{E}[R_t|\text{17AL17}, \text{HitWall}] && (48)\\
&+\Pr(\neg\text{HitWall}|\text{17AL17})\mathbf{E}[R_t|\text{17AL17}, \neg\text{HitWall}] && (49)\\
=&\Pr(\text{HitWall}|\text{17AL17}) \times -10 && (50)\\
&+\Pr(\neg\text{HitWall}|\text{17AL17}) \times 0 && (51)\\
=&-10\Pr(\text{HitWall}|\text{17AL17}) && (52)\\
& && (53)
\end{aligned}
$$

   To conclude, we must compute the probability that the agent hit the wall if it started in state 17, took action AL, and ended in state 17.

$$
\begin{aligned}
\Pr(\text{HitWall}|\text{17AL17}) =&\frac{\Pr(\text{HitWall}, S_{t+1} = 17|S_t = 17, A_t = \text{AL})}{\Pr(S_{t+1} = 17|S_t = 17, A_t = \text{AL})} && (54)\\
=&\frac{\Pr(\text{HitWall}|S_t = 17, A_t = \text{AL})}{\Pr(S_{t+1} = 17|S_t = 17, A_t = \text{AL})} && (55)\\
=&\frac{0.8}{0.8 + 0.1} && (56)\\
=&\frac{0.8}{0.9}. && (57)
\end{aligned}
$$

   So,

$$
\mathbf{E}[R_t|S_t = 17, A_t = \text{AL}, S_{t+1} = 17] = -10\frac{0.8}{0.9}. \qquad (58)
$$

4. (2 Points) How many deterministic policies are there for an MDP with $|\mathcal{S}| < \infty$ and $|\mathcal{A}| < \infty$? (You may write your answer in terms of $|\mathcal{S}|$ and $|\mathcal{A}|$).

   $|A|^{|S|}$

5. (2 Points) Give an example of an MDP with $|\mathcal{S}| < \infty, |\mathcal{A}| = \infty$, and $\gamma < 1$ such that an optimal policy *does not* exist. Give an example of an MDP with $|\mathcal{S}| = \infty, |\mathcal{A}| < \infty$, and $\gamma < 1$ such that an optimal policy exists.

   Student answers may vary. Example of nonoptimal: An MDP with one state that always transitions to s_∞, $\mathcal{A} = [0, 1)$, and $R_t = A_t$. Since the action set is open, the maximum element in $\mathcal{A}$ does not exist, and so no optimal policy exists. Example of optimal: mountain car domain.

6. (3 Points) Read about the Pendulum domain, described in Section 5.1 of this paper (Reinforcement Learning in Continuous Time and Space by Kenji Doya). Consider a variant where the initial state has the pendulum hanging down with zero angular velocity always (a deterministic initial state where the pendulum is hanging straight down with no velocity) and a variant where the initial angle is chosen uniformly randomly in $[-\pi, \pi]$ and the initial velocity is zero. Which variant do you expect an agent to require more episodes to solve? Why? Note: We did not talk about the complexity of solving MDPs in class yet—we want you to provide your best guess here.

   Any reasonable answer is ok, but it must have an explanation that makes some sense.

7. (1 Point) How many episodes do you expect an agent should need in order to find near-optimal policies for the gridworld and pendulum domains? Note: We did not talk about the complexity of solving MDPs in class yet—we want you to provide your best guess here.
   Any reasonable answer is ok, but it must have an explanation that makes some sense.

8. (5 Points) Select a problem that we have not talked about in class, where the agent does not make Markovian observations about the world around it. Describe how the environment for this problem can be formulated as an MDP by specifying $(\mathcal{S}, \mathcal{A}, P, [d_r \text{ or } R], d_0, \gamma)$ (your specifications of these terms may use English rather than math, but be precise).
   Answers may vary. The key property should be that the states are Markovian and correspond to states, not observations.

9. (5 Points) We refer to the discounted sum of rewards, $\sum_{t=0}^{\infty} \gamma^t R_t$, as the *return*. Let an MDP exist such that it has two optimal policies. Can the expected value of their returns differ? If so, give an example. If not, explain why. Can the variance of their returns differ? If so, give an example. If not, explain why.

   By the definition of an optimal policy described in class, the expected value of the returns of the policies cannot differ. However, the variance of the policies can differ. Answers may vary for the example. One example is an MDP with one state that always transitions to s_∞ and which has two actions. The first actions results in $R_t \sim \mathcal{N}(0, 1)$ and the second results in $R_t \sim \mathcal{N}(0, 2)$.

10. (2 Points) Consider the one state MDP wherein s_0 there are three actions, a_0, a_1, a_2, and all actions transition to s_∞ with probability 0.5 and stay in s_0 otherwise. The reward for taking actions a_0, a_1 are drawn from the uniform distribution on $[0, 1]$ and the normal distribution $\mathcal{N}(0.5, 1)$, respectively. The reward for a_3 is always 0.25. What are all the optimal policies of this MDP?

    The optimal policies are all policies such that $\pi(s_0, a_0) + \pi(s_0, a_1) = 1$

11. (2 Points) Read the Wikipedia page on Markov chains. A state in a *Markov chain* is *irreducible* if it is possible to get to any state from any state. An MDP is *irreducible* if the Markov chain associated with every policy is irreducible. A state in a *Markov chain* has *period* k if every return to the state must occur in multiples of k time steps. More formally,

$$k = \gcd\{t > 0 : \Pr(S_t = s|S_0 = s) > 0\}.$$

A Markov chain is *aperiodic* if the period of every state is $k = 1$. An MDP is aperiodic if the Markov chain associated with every deterministic policy is aperiodic (or equivalently, if the Markov chain associated with every policy is aperiodic). Can an MDP be aperiodic and *not* irreducible? If so, give an example. If not, explain your reasoning.

Yes, an MDP can be aperiodic and not irreducible. Student answers may vary for the example. Imagine an MDP with two states that always transition to themselves.

12. (5 Points) The state of a Markov chain is *positive recurrent* if the expected time until the state recurs is finite. A Markov chain is *positive recurrent* if all states are positive recurrent. Give an example of an MDP with $|S| > 2$ states that is positive recurrent and aperiodic. For any number of states $|S|$, can you think of a simple way of defining state transitions such that the MDP is positive recurrent and aperiodic? Explain your methodology (a picture might be useful).

Student answers may vary. An example is an MDP with states that always transition to themselves will be positive recurrent and aperiodic.

13. (1 point) Let a tabular policy representation be used to represent stochastic policies for an MDP with n states and m actions. What is the sum of every element in the matrix representation of this policy? Why?

If we represent a tabular policy as a matrix with rows being the states and columns being the actions, each row should sum to one. If there are n states, then the sum of each element is $1 \times n = n$

14. (2 Points) Describe a real-world problem and how it can be reasonably modeled as an MDP where R_t is *not* a deterministic function of S_t, A_t, and S_{t+1}, and is not a bandit problem.

Student answers may vary. Example: Consider scanning space for extraterrestrial or other interesting radio signals. The state might be a vector representing the current spatial location and frequency the radio is scanning. The action space is the direction to point the radio and what frequency to listen to next. The reward would be +1 for finding an interesting signal and 0 otherwise. The signal may not always be received even when the radio focuses in the right direction/frequency due to noise in the signal, so it is not a deterministic function of the state, action, or next state.

15. (2 Points) If you know $(\mathcal{S}, \mathcal{A}, P, R, d_0, \gamma)$, can you derive d_R? Prove your answer is correct.

    You cannot derive d_R given R. To prove this, we provide two MDPs that have the same $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$, d_0, and γ, but different d_R. Let d_R^1 be $\mathcal{U}(0,1)$ for all s, a, s' in the first MDP, and let d_R^2 be $\mathcal{N}(0.5, 1.0)$ for all s, a, s' in the second MDP. Then for any state s and action a, $R^1(s,a) = \mathbf{E}[R_t|S_t = s, A_t = a, d_R^1]$, $R^2(s,a) = \mathbf{E}[R_t|S_t = s, A_t = a, d_R^2]$ and $R^1(s,a) = R^2(s,a)$. Therefore, given only R such that $R = R^1$ or $R = R^2$ it is not possible to know if the rewards come a distribution corresponding to d_R^1 or d_R^2.

16. (2 Points) If you know $(\mathcal{S}, \mathcal{A}, P, d_R, d_0, \gamma)$, can you derive R? Prove your answer is correct.

    Yes, you can derive R. If the expectation $\mathbf{E}[R_t|S_t = s, A_t = a, S_{t+1} = s', d_R]$ exists for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$, then R can be derived.

$$R(s,a) = \mathbf{E}[R_t|S_t = s, A_t = a] \tag{59}$$

$$= \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s'|S_t = s, A_t = a)\mathbf{E}[R_t|S_t = s, A_t = a, S_{t+1} = s'] \tag{60}$$

$$= \sum_{s' \in \mathcal{S}} P(s,a,s')\mathbf{E}[R_t|R_t \sim d_R(s,a,s',\cdot)], \tag{61}$$

    where $d_R(s,a,s',\cdot)$ is the probability distribution of R_t given $S_t = s$, $A_t = a$, and $S_{t+1} = s'$.

17. (2 Points) Describe a real-world problem and how it can be reasonably modeled as an MDP where R_t is a deterministic function of S_t.

    Student answers may vary. One example is a robot learning to balance. The reward mght be +1 for every time step where the robot has not fallen over (and zero when the robot falls over, entering s_∞). Here we could define the reward to be +1 if S_t is a state that is not considered to be "fallen over".

18. (2 Points) Describe a real-world problem and how it can be reasonably modeled as an MDP where the reward function, R, would *not* be known.

    Consider a the problem of a robot cooking for a human. The reward could be +1 for the human being satisfied, −1 for disgusted, and 0 otherwise. Human preferences are dependent on many non-observable factors and change over time. This means that given s, a the expectation, $R(s,a)$, would still be unknown.

19. (2 Points) Describe a real-world problem and how it can be reasonably modeled as an MDP where the transition function, P, would be known.

    Student answers may vary. Robotics applications sometimes involve physical systems, the dynamics of which are well-understood. We can view these as being problems where the transition dynamics are known. Another example is digital marketing, where the state is a description of a person's interests (perhaps based only on external information, not information gleaned from an episode) along with a count of how many times each advertisement was shown. The action is a choice about which ad to show, and the reward is +1 if the user clicks the ad and zero otherwise. In this case, the transition function is known (the counter for the shown ad will be incremented), but the reward function is unknown.

20. (2 Points) Describe a real-world problem and how it can be reasonably modeled as an MDP where the transition function, P, would *not* be known.

    Student answers may vary. Consider a tutoring system, wherein the states correspond to what the student currently understands, and the actions correspond to decisions about what topic to teach next. The transition function for this problem includes human behaviorhow a student learns about a topic. Modeleing human learning is challenging, and an active research topic. Current models are not reliable enough to be viewed as known transition functions.

Part Two: Programming (25 Points Total)

More-Watery 687-Gridworld. For this assignment, we will be working with a slightly modified version of the 687-Gridworld domain described in class and in the class notes. In this new Gridworld, called More-Watery Gridworld, there are two extra water states located in state 7 and state 17, as shown in Figure 5. Implement More-Watery Gridworld.

Codebase. We have provided a template for programming the homework on the github repository for this class located here. You do *not* need to use this template for the assignment. After the due date for this assignment, an example will be posted on this site.

A (5 Points) Have the agent uniformly randomly select actions. Run 10,000 episodes. Report the mean, standard deviation, maximum, and minimum of the observed discounted returns.

These values can vary, e.g., the actual maximum value is the maximum return for any policy.

policy	mean	std	min	max
uniform	-9.2600	7.2887	-45.984	4.3047

B (5 Points) Find an optimal policy (you may do this any way you choose, including by reasoning through the problem yourself). Report the optimal policy here. Comment on whether it is unique.

The optimal policy is not unique. The action in the terminal state, s_{23}, can be anything. If the terminal state is not represented in the policy,

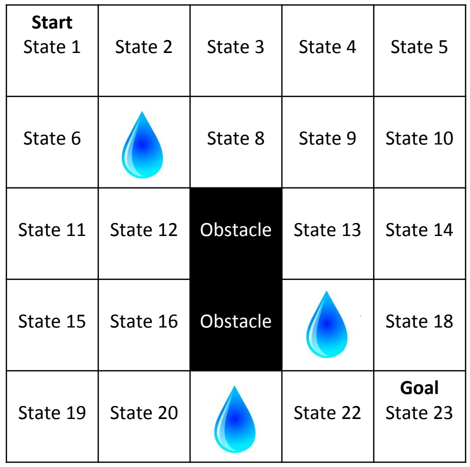
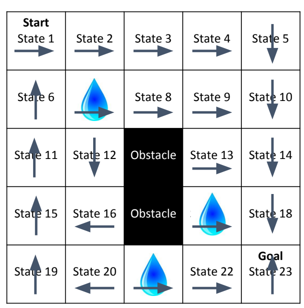


Figure 5: A more watery version of 687-Gridworld. Water states are located in state 7, state 17, and state 21.

then the optimal policy is unique. However, since this can be challenging to ascertain without the tools that we will teach you later, a reasonable argument for why two different policies would be optimal (particularly in states where the expected return is very similar when the two actions are taken) *may* be accepted for full credit.



C (5 Points) Run the optimal policy that you found in the previous question for 10,000 episodes. Report the mean, standard deviation, maximum, and minimum of the observed discounted returns.

D (5 Points) The distribution of returns is often not normal, thus it cannot

policy	mean	std	min	max
optimal	3.0176	2.9235	-27.4706	4.7830

be fully characterized by its mean and standard deviation. To provide more information about the performance, the empirical distribution of returns can be reported.

For a random variable, X, its *cumulative distribution function* (CDF), F_X, is defined as $F_X(x) \coloneqq \Pr(X \leq x)$. The empirical CDF, $\hat{F}$, for a sequence of n samples of X, $X_1, \ldots, X_n$ is given by the function

$$\hat{F}_n(x) \coloneqq \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{X_i \leq x},$$

where X_i is the i^{th} sample of X and $\mathbf{1}_A$ is the indicator function of an event A, i.e., $\mathbf{1}_A = 1$ if A is true and 0 otherwise.

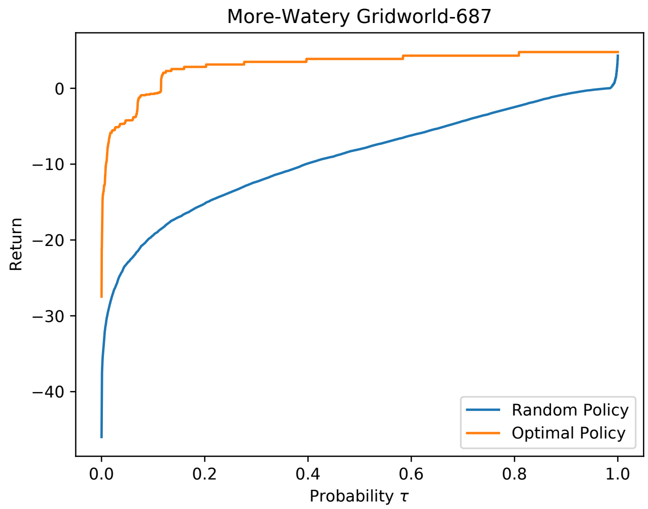
The quantile function, also referred to as the inverse CDF, is the function $Q(\tau) \coloneqq \inf\{x \in \mathbb{R} \colon \tau \leq F_X(x)\}$ for $\tau \in (0, 1)$. The empirical quantile function, $\hat{Q}$, can be constructed by considering the order statistics, Z_i, the sorted samples of X_i such that $Z_1 \leq Z_2 \leq \ldots \leq Z_n$. The empirical quantile function is given by

$$\hat{Q}(\tau) \coloneqq Z_{\lfloor (n+1)\tau \rfloor}.$$

Both the CDF and quantile functions capture all the information about a random variable, but for plotting purposes the quantile function is often preferred. This is because we are interested in maximizing returns, so the quantile function has a more natural interpretation as higher is better.

Plot the distribution of returns for both the random policy and the optimal policy using 10,000 trials each. You must clearly label each line and axis. Additionally, report the random seed used for the experiments.

Random seed used was 4321 and was initialized before all experiments.



E (5 Points) Using simulations, empirically estimate the probability that $S_{19} = 21$ given that $S_8 = 18$ (the state above the goal) when running the uniform random policy. Describe how you estimated this quantity (there is *not* a typo in this problem, nor an oversight).

After using 100,000 trials, $\Pr(S_{19} = s_{21}|S_8 = s_{18}) \approx 0.01858$ and is in $[0.01776, 0.01944]$ with 95% confidence according to Wilson Score interval. Each trial started with $S_0 = s_{18}$ and was executed until $t = 11$ or the terminal absorbing state was reached.

CMPSCI 687 Pop Quiz 1
September 12, 2019

Instructions: You have 5 minutes to complete this quiz. This quiz is **closed** notes—do not use your notes or a laptop. Do not discuss problems with your neighbors until after everyone has handed in their quiz.

Use the notation from class. Don't forget to capitalize your random variables. Presenting equalities that are true is not enough—you must provide the *definitions* of the symbols on the left.

Fill in the definitions for the following terms (your answer to 5 should be a real number, not math expressions):

[Answers in blue.]

1. $P(s,a,s') = \Pr(S_{t+1}=s'|S_t=s, A_t=a)$

2. $R(s,a) = \mathbf{E}\left[R_t|S_t=s, A_t=a\right]$

3. $\pi(s,a) = \Pr(A_t=a|S_t=s)$

4. $J(\pi) = \mathbf{E}\left[\sum_{t=0}^{\infty}\gamma^t R_t|\pi\right]$

5. $\min_{a\in\mathcal{A}} R(s_\infty,a) = 0$

Consider again the definition of reinforcement learning. Notice the segment "learn from *interactions* with the environment." If P and R (or d_R) are known, then the agent does not need to interact with the environment. E.g., an agent solving 687-Gridworld can plan in its head, work out an optimal policy and execute this optimal policy from this start. This is *not* reinforcement learning—this is *planning*. More concretely, in planning problems P and R are known, while in reinforcement learning problems at least P (and usually R) is not known by the agent. Instead, the agent must learn by interacting with the environment—taking different actions and seeing what happens. Most reinforcement learning algorithms will *not* estimate P. The environment is often too complex to model well, and small errors in an estimate of P compound over multiple time steps making plans built from estimates of P unreliable. We will discuss this more later.

2.5 Additional Terminology, Notation, and Assumptions

- A *history*, H_t, is a recording of what has happened up to time t in an episode:
$$H_t \coloneqq (S_0, A_0, R_0, S_1, A_t, R_1, S_2, \dots, S_t, A_t, R_t). \tag{62}$$
- A *trajectory* is the history of an entire episode: H_∞.
- The *return* or *discounted return* of a trajectory is the discounted sum of rewards: $G \coloneqq \sum_{t=0}^{\infty} \gamma^t R_t$. So, the objective, J, is the *expected return* or *expected discounted return*, and can be written as $J(\pi) \coloneqq \mathbf{E}[G|\pi]$.
- The *return from time t* or *discounted return from time t*, G_t, is the discounted sum of rewards starting from time t:
$$G_t \coloneqq \sum_{k=0}^{\infty} \gamma^k R_{t+k}.$$

2.5.1 Example Domain: Mountain Car

Environments studied in RL are often called *domains*. One of the most common domains is *mountain car*, wherein the agent is driving a crude approximation of a car. The car is stuck in a valley, and the agent wants to get to the top of the hill in front of the car. However, the car does not have enough power to drive straight up the hill in front, and so it must learn to reverse up the hill behind it before accelerating forwards to climb the hill in front. A diagram of the mountain car environment is depicted in Figure 6.

- **State**: $s = (x, v)$, where $x \in \mathbb{R}$ is the position of the car and $v \in \mathbb{R}$ is the velocity.
- **Actions**: $a \in \{\texttt{forward}, \texttt{neutral}, \texttt{reverse}\}$. These actions can be renamed to be less unwieldy: $a \in \{0, 1, 2\}$.

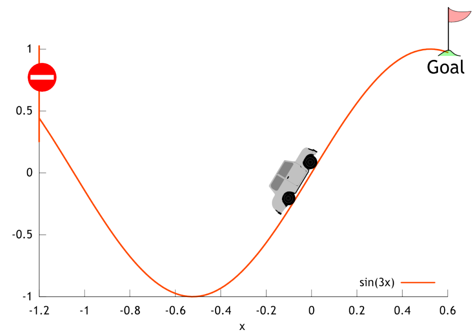


Figure 6: Diagram of the mountain car domain.

- **Dynamics**: The dynamics are *deterministic*—taking action a in state s always produces the same state, s'. Thus, $P(s, a, s') \in \{0, 1\}$. The dynamics are characterized by:

$$v_{t+1} = v_t + 0.001a - 0.0025\cos(3x)x_{t+1} = x_t + v_{t+1}. \tag{63}$$

  If these equations would cause $x_{t+1} < -1.2$ then instead $x_{t+1} = -1.2$ and the velocity is set to zero: $v_{t+1} = 0$. Similarly, if these equations would cause $x_{t+1} > 0.5$, then $x_{t+1} = 0.5$ and the velocity is set to zero: $v_{t+1} = 0$. This simulates inelastic collisions with walls at -1.2 and 0.5.

- **Terminal States**: If $x_t = 0.5$, then the state is terminal (it always transitions to s_∞).

- **Rewards**: $R_t = -1$ always, except when transitioning to s_∞ (from s_∞ or from a terminal state), in which case $R_t = 0$.

- **Discount**: $\gamma = 1.0$.

- **Initial State**: $S_0 = (-0.5, 0)$ deterministically (i.e., $\Pr(S_0 = (-0.5, 0)) = 1$).

Question 6. *For this problem, what is an English description of the meaning behind a return? What is an episode? What is an optimal policy? How long can an episode be? What is the English meaning of $J(\pi)$?*

Answer 6. *The return is negative the number of time steps for the car to reach the goal. An episode corresponds to the car starting near the bottom of the valley and the agent driving it until it reaches the top of the hill in front of the car. An optimal policy reverses up the hill behind the car until some specific point is reached, at which point the car accelerates forward until it reaches the goal. There is no limit on how long an episode can be. $J(\pi)$ is the expected number of time steps for the agent to reach the goal when it uses policy π.*

2.5.2 Markov Property

A seemingly more general non-Markovian formulation for the transition function might be:

$$P(h, s, a, s') \coloneqq \Pr(S_{t+1} = s' | H_{t-1} = h, S_t = s, A_t = a). \tag{64}$$

The *Markov assumption* is the assumption that S_{t+1} is conditionally independent of H_{t-1} given S_t. That is, for all h, s, a, s', t:[3]

$$\Pr(S_{t+1} = s' | H_{t-1} = h, S_t = s, A_t = a) = \Pr(S_{t+1} = s' | S_t = s, A_t = a) \tag{65}$$

Since we make this Markov assumption, P as defined earlier completely captures the transition dynamics of the environment, and there is no need for the alternate definition in (64). The Markov assumption is sometimes referred to as the *Markov property* (for example one would usually say that a domain has the Markov property, not that the domain satisfies the Markov assumption). It can also be stated colloquially as: the future is independent of the past given the present.

We also assume that the rewards are Markovian—R_t is conditionally independent of H_{t-1} given S_t (since A_t depends only on S_t, this is equivalent to assuming that R_t is conditionally independent of H_{t-1} given both S_t and A_t). While the previous Markov assumptions apply to the environment (and are inherent assumptions in the MDP formulation of the environment), we make an additional Markov assumption about the agent: the agent's policy is Markovian. That is, A_t is conditionally independent of H_{t-1} given S_t.

Question 7. *Can you give examples of Markovian and non-Markovian environments?*

Question 8. *Is the Markov property a property of the problem being formulated as an MDP or a property of the state representation used when*

[3]For brevity, hereafter we omit the sets that elements are in when it should be clear from context, e.g., we say "for all s" rather than "for all $s \in \mathcal{S}$".

formulating the problem?

To answer this second question, consider whether state transitions are Markovian in mountain car. It should be clear that they are as the domain has been described. What about if the state was $s = (x)$ rather than $s = (x, v)$? You could deduce v_t from the previous state, x_{t-1} and current state, x_t, but that would require part of the history before s_t. Thus, using $s = (x)$ mountain car is *not* Markovian. So, the Markov property is really a property of the state representation, not the problem being formulated as an MDP.

Notice that one can always define a Markovian state representation. Let S_t be a non-Markovian state representation. Then (S_t, H_{t-1}) is a Markovian state representation. That is, we can include the history within the states in order to enforce the Markov property. This is typically undesirable because the size of the state set grows exponentially with the maximum episode length (a term discussed more later). This trick of adding information into the state is called *state augmentation*.

There is often confusion about terminology surrounding states, state representations, and the Markov property. The *state* of an MDP (and every other similar formulation, like POMDPs, DEC-POMDPs, SMDPs, etc.) should *always* be defined so that the Markov property is satisfied. Later we will reason about *state representations* that are not Markovian, in order to model situations where the agent might only be able to make partial or noisy observations about the state of the environment.

——————End of Lecture 4, September 12, 2019——————

2.5.3 Stationary vs. Nonstationary

We assume that the dynamics of the environment are *stationary*. This means that the dynamics of the environment do not change between episodes, and also that the transition function does not change within episodes. That is, $\Pr(S_0 = s)$ is the same for all episodes, and also for all s, a, t, and i:

$$\Pr(S_{t+1} = s' | S_t = s, A_t = a) = \Pr(S_{i+1} = s' | S_i = s, A_i = a). \tag{66}$$

Importantly, here t and i can be time steps from different episodes.

This is one of the assumptions that is most often *not* true for real problems. For example, when using RL to control a car, we might not account for how ware on the parts (e.g., tires) causes the dynamics of the car to change across drives. When using RL to optimize the selection of advertisements, the day of the week, time of day, and even season can have a large impact on how advertisements are received by people. Depending on how the problem is modeled as an RL problem, this may manifest as nonstationarity.

Although this assumption is almost always made, and is almost always false, we usually justify it by saying that some assumption of this sort is necessary.

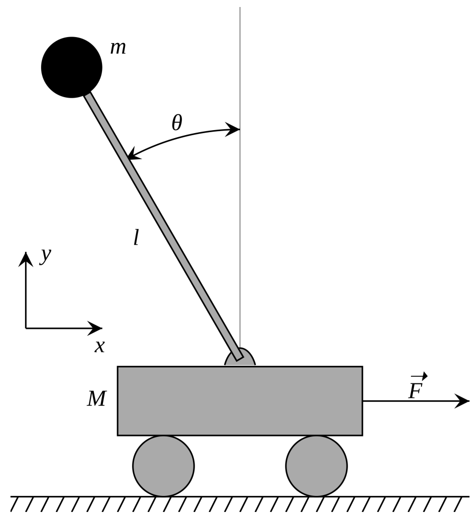


Figure 7: Cart-Pole Domain.

This assumption is what allows us to use data from the past to inform how we make decisions about the future. Without some assumption saying that the future resembles the past, we would be unable to leverage data to improve future decision making. Still, there exist weaker assumptions than requiring stationarity (e.g., a small amount of work focuses on how to handle a small finite number of jumps in system dynamics or slow and continuous shifts in dynamics).

We also assume that the rewards are stationary (that the distribution over rewards that result from taking action a in state s and transitioning to state s' does not depend on the time step or episode number). However, usually we assume that π is *nonstationary*. This is because learning corresponds to changing (ideally, improving) the policy both within an episode and across episodes. A stationary policy is sometimes called a *fixed policy*.

2.5.4 Cart-Pole Balancing

Also called *pole balancing*, *cart-pole*, and *inverted pendulum*.

This domain models a pole balancing on a cart, as depicted in Figure 7. The agent must learn to move the cart forwards and backwards to keep the pole from falling.

- State: $s = (x, v, \theta, \dot{\theta})$, where x is the horizontal position of the cart, θ is the angle of the pole, and $\dot{\theta}$ is the angular velocity of the pole.
- Actions: $\mathcal{A} = \{\text{left}, \text{right}\}$.
- $R_t = 1$ always.
- $\gamma = 1$.

- $S_0 = (0, 0, 0, 0)$ always.
- Dynamics = physics of the system. See the work of Florian (2007) for the derivation of the correct dynamics. The domain was originally presented by Barto et al. (1983). However, this original work presents the dynamics with gravity reversed—pulling the pole up rather than down. Andy says that they did use the correct direction for gravity in their code though.
- Episodes terminate after 20 seconds, or when the pole falls down. Time is simulated with time steps of $\Delta t = 0.02$ seconds.

Question 9. *Is the optimal policy for cart-pole unique?*

Answer 9. *No. An action might cause the pole to move away from vertical, but as long as it does not fall this is not penalized by the reward function.*

Question 10. *Is the state representation Markovian?*

Answer 10. *No. In order for the transition function to cause a transition to s_∞ after twenty seconds, the state must encode the current time step.*

2.5.5 Finite-Horizon MDPs

The *horizon*, L, of an MDP is the smallest integer such that

$$\forall t \geq L, \ \Pr(S_t = s_\infty) = 1. \tag{67}$$

If $L < \infty$ for all policies, then we say that the MDP is *finite horizon*. If $L = \infty$ then the domain may be *indefinite horizon* or *infinite horizon*. An MDP with *indefinite horizon* is one for which $L = \infty$, but where the agent will always enter s_∞. One example of an indefinite horizon MDP is one where the agent transitions to s_∞ with probability 0.5 from every state. An *infinite horizon* MDP is an MDP where the agent may never enter s_∞.

For the cart-pole domain, how can we implement within P that a transition to s_∞ must occur after 20 seconds? We achieve this by augmenting the state to include the current time. That is, the state is (s, t), where s is what we preivously defined to be the state for cart-pole and t is the current time step. P increments t at each time step and causes transitions to S_∞ when t is incremented to $20/\Delta t = 1000$. So, the state for cart-pole is really $s = (x, v, \theta, \dot{\theta}, t)$. Often the dependence on t is implicit—we write $s = (x, v, \theta, \dot{\theta})$ and say that the domain is finite horizon.

2.5.6 Partial Observability

For many problems of interest, the agent does not know the state—it only makes observations about the state. These observations may be noisy and/or incomplete. We will discuss this later in the course.

2.5.7 Other Domains

See if you can model the following problems as MDPs:

- Type 1 Diabetes treatment, where the goal is to determine how much insulin an insulin pump should inject in order to keep a person's blood glucose levels near optimum levels (Bastani, 2014).
- Digital marketing, where the goal is to present ads on a website that are likely to be clicked by the user. Assume that we have some knowledge about the user, like their age and gender.
- Playing video games to maximize score.
- Automation of methods for determining the composition of rocks using spectroscopy. For this task the agent observes spectra collected from a rock and makes a decision about what minerals it believes are present. There are data sets containing examples of spectra of rocks with known compositions, but these data sets are small.
- Functional electrical stimulation, where the goal is to stimulate the muscles in a paralyzed person's arm in order to cause it to move to a desired position (Thomas et al., 2009).
- Garbage collection systems, where the agent must optimize the choice of when to garbage collect.
- Intelligent tutoring systems, where the goal is to decide how to order topics presented to a student in order to maximize their score on a quiz after the tutorial (Woolf, 2010; Mandel et al., 2014).
- Elevator scheduling, where the agent must decide where to send an elevator to minimize wait times.

Question 11. *All but one of the above problems are cases where supervision is not available. After the agent takes an action, we do not know what the optimal action would have been. One example is of a supervised learning problem that likely should* not *be solved using RL methods. Which one?*

——————End of Lecture 5, September 17, 2019——————

---

3 Black-Box Optimization for RL

3.1 Hello Environment!

In this lecture we will describe how you can create your first RL agent. Agents can be viewed as objects in an object-oriented programming language that have functions for getting an action from the agent, telling an agent about what states and rewards it obtains, and for telling the agent when a new episode has occurs. High level pseudocode for an agent interacting with an environment may look like:

Algorithm 2: Pseudocode for an agent interacting with an environment.

```
for episode = 0, 1, 2, ... do
    s ~ d_0;
    for t = 0 to ∞ do
        a = agent.getAction(s);
        s' ~ P(s, a, ·);
        r ~ d_R(s, a, s', ·);
        agent.train(s, a, r, s');
        if s' == s_∞ then
            break;          // Exit out of loop over time, t
        s = s';
    agent.newEpisode();
```

Here the agent has three functions. The first, `getAction`, which samples an action, a, given the current state s, and using the agent's current policy. The second function, `train`, alerts the agent to the transition that just ocurred, from s to s', due to action a, resulting in reward r. This function might update the agent's current policy. The third function, `newEpisode`, alerts the agent that the episode has ended and it should prepare for the next episode. Notice that the agent object might have memory. This allows it to, for example, store the states, actions, and rewards from an entire episode during calls to `train`, and then update its policy when `newEpisode` is called using all of the data from the episode (or from multiple episodes).

Question 12. *How might you create this agent object so that it improves its policy? This might be your last chance to think about this problem in your own creative way, before we corrupt your mind with the standard solutions used by the RL community.*

3.2 Black-Box Optimization (BBO) for Policy Search

Black-box optimization (BBO) algorithms are generic optimization algorithms (i.e., not specific to RL) that solve problems of the form:

$$\arg\max_{x\in\mathbb{R}^n} f(x), \tag{68}$$

where $f:\mathbb{R}^n \to \mathbb{R}$. Different BBO algorithms make different assumptions about f, like that it is smooth or bounded. Furthermore, different BBO algorithms make different assumptions about what information is known about f. Some algorithm might assume that the optimization algorithm can compute $f(x)$ for any $x \in \mathbb{R}^n$, while others assume that the agent can compute the gradient, $\nabla f(x)$, at any point x. BBO algorithms are called *black-box* because they treat f as a black-box—they do not look "inside" of f to leverage knowledge about its structure (or knowledge of an analytic form) in order to speed up the optimization process. For RL, this means that BBO algorithms will not leverage the knowledge that the environment can be modeled as an MDP.

Here, we will consider BBO algorithms that assume the estimates of $f(x)$ can be produced for any $x \in \mathbb{R}^n$, but that the precise value of $f(x)$ is not known, and the gradient, $\nabla f(x)$, is also not known. Examples of BBO algorithms that can be used for problems of this sort include (first-choice) hill-climbing search, simulated annealing, and genetic algorithms (Russell et al., 2003). To apply these algorithms to RL, we will use them to optimize the objective function. That is, we will use them to solve the problem:

$$\arg\max_{\pi\in\Pi} J(\pi). \tag{69}$$

In order to apply these algorithms to the above problem, we must determine how we can estimate $J(\pi)$, and also how we can represent each policy, π, as a vector in $\mathbb{R}^n$.

3.2.1 How to estimate the objective function?

We can estimate $J(\pi)$ by running the policy π for N episodes and then averaging the observed returns. That is, we can use the following estimator, $\hat{J}$, of J:

$$\begin{aligned}\hat{J}(\pi) \coloneqq& \frac{1}{N}\sum_{i=1}^N G^i \qquad (70)\\ =& \frac{1}{n}\sum_{i=1}^N\sum_{t=0}^\infty \gamma^t R_t^i, \qquad (71)\end{aligned}$$

where G^i denotes the return of the i^{th} episode and R_t^i denotes the reward at time t during episode i. Hereafter, we will use superscripts on random variables to denote the episode during which they occurred (but will omit these superscripts when the relevant episode should be clear from context).

3.2.2 Parameterized Policies

Recall from Figure 3 that we can represent policies as $|\mathcal{S}| \times |\mathcal{A}|$ matrices with non-negative entries and rows that sum to one. When numbers are used to define a policy, such that using different numbers results in different policies, we refer to the numbers as *policy parameters*. Unlike other areas of machine learning, often we will discuss different functions and/or distributions that can be parameterized in this way, and so it is important to refer to these parameters as *policy parameters* and not just *parameters* (unless it is exceedingly obvious which parameters we are referring to). We refer to the representation described in Figure 3 as a *tabular* policy, since the policy is stored in a table, with one entry per state-action pair. Notice that we can view this table as a vector by appending the rows or columns into a vector in $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$.

Although this parameterization of the policy is simple, it requires constraints on the solution vectors that standard BBO algorithms are not designed to handle—they require the rows in the tabular policy to sum to one and the entries to always be positive. Although BBO algorithms might be adapted to work with this policy representation, it is usually easier to change how we represent the policy. That is, we want to store π as a $|\mathcal{S}| \times |\mathcal{A}|$ matrix, p, that has no constraints on its entries. Furthermore, increasing $p(s,a)$ (the entry in the s'th row and a'th column) should increase $\pi(s,a)$. Notice that, using this notation, $p(s,a)$ is a policy parameter for each (s,a) pair.

One common way to achieve this is to use *softmax action selection*. Softmax action selection defines the policy in terms of $p(s,a)$ as:

$$\pi(s,a) \coloneqq \frac{e^{\sigma p(s,a)}}{\sum_{a'} e^{\sigma p(s,a')}}, \tag{72}$$

where $\sigma > 0$ is a hyperparameter that scales how differences in values of $p(s,a)$ and $p(s,a')$ change the probabilities of the actions a and a'. For now, assume that $\sigma = 1$. To see that this is a valid definition of π, we must show that $\pi(s,\cdot)$ is a probability distribution over $\mathcal{A}$ for all $s \in \mathcal{S}$. That is, $\sum_{a\in\mathcal{A}} \pi(s,a) = 1$ for all s and $\pi(s,a) \geq 0$ for all s and a. We now show these two properties. First, for all $s \in \mathcal{S}$:

$$\sum_{a\in\mathcal{A}} \pi(s,a) = \sum_{a\in\mathcal{A}} \frac{e^{\sigma\pi(s,a)}}{\sum_{a'\in\mathcal{A}} e^{\sigma p(s,a')}} \tag{73}$$

$$= \frac{\sum_{a\in\mathcal{A}} e^{\sigma\pi(s,a)}}{\sum_{a'\in\mathcal{A}} e^{\sigma p(s,a')}} \tag{74}$$

$$= 1. \tag{75}$$

Second, for all s and a, $e^{\sigma p(s,a)} > 0$, and so all terms in the numerator and denominator of (72) are non-negative, and thus $\pi(s,a)$ is non-negative.

This distribution is also known as the Boltzman distribution or Gibbs distribution. A drawback of using this distribution is that it cannot exactly represent deterministic policies without letting $p(s,a) = \pm\infty$.

We typically denote the parameters of a policy by θ, not p, and we define a *parameterized policy* to be a function $\pi : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \to [0, 1]$ such that $\pi(s, a, \theta) = \Pr(A_t = a | S_t = s, \theta)$. Thus, changing the policy parameter vector θ results in the parameterized policy being a different policy. So, you might see the equivalent definition of a tabular softmax policy:

$$\pi(s, a, \theta) \coloneqq \frac{e^{\sigma \theta_{s,a}}}{\sum_{a'} e^{\sigma \theta_{s,a'}}}. \tag{76}$$

Tabular softmax policies are just one way that the policy might be parameterized. In general, you should pick the policy parameterization (the way that policy parameter vectors map to stochastic policies) to be one that you think will work well for the problem you want to solve. This is more of an art than a science—it is something that you will learn from experience.

Now that we have represented the policy as a vector in $\mathbb{R}^n$ (with $n = |\mathcal{S}||\mathcal{A}|$ when using a tabular softmax policy), we must redefine our objective function to be a function of policy parameter vectors rather than policies. That is, let

$$J(\theta) \coloneqq \mathbf{E}[G|\theta], \tag{77}$$

where conditioning on θ denotes that $A_t \sim \pi(S_t, \cdot, \theta)$. Often a policy parameterization will be used that cannot represent all policies. In these cases, the goal is to find the best policy that can be represented, i.e., the optimal policy parameter vector:

$$\theta^\star \in \operatorname*{arg\,max}_{\theta \in \mathbb{R}^n} J(\theta). \tag{78}$$

Examples of other parameterized policies include deep neural networks, where the input to the network is the state, s, and the network has one output per action. One can then use softmax action selection over the outputs of the network. If the actions are continuous, one might assume that the action, A_t, should be normally distributed given S_t, where the mean is parameterized by a neural network with parameters θ, and the variance is a fixed constant (or another parameter of the policy parameter vector).

One might also choose to represent deterministic policies, where the input is a state and the output is the action that is always chosen in that state. For example, consider the problem of deciding how much insulin an insulin pump should inject prior to a person eating a meal. One common (and particularly simple) policy parameterization is:

$$\text{injection size} = \frac{\text{current blood glucose} - \text{target blood glucose}}{\theta_1} + \frac{\text{meal size}}{\theta_2}, \tag{79}$$

where $\theta = [\theta_1, \theta_2]^\intercal \in \mathbb{R}^2$ is the policy parameter vector, the current blood glucose and meal size form the state, the target blood glucose is a constant value specified by a diabetologist, and the injection size is the action (Bastani, 2014). Notice that this representation results in a small number of policy parameters—far fewer than if we were to use a neural network. Similarly, many control problems can use policies parameterized using few parameters (Schaal et al., 2005) (see also PID and PD controllers).

Another common policy representation (typically used when states are not discrete) is *softmax action selection with linear function approximation*. Later we will discuss linear function approximation in general and in more detail. Here we will consider a single specific use. Let $\phi : \mathcal{S} \to \mathbb{R}^m$ be a function that takes a state as input and returns a vector of features as output. Notice that $\phi(s)$ can be a short vector even when $\mathcal{S}$ is continuous. For example, if $\mathcal{S} = \mathbb{R}$, we could define $\phi(s) = [s, s^2, s^3, ..., s^10]^\intercal$. For now we define $\phi(s)$ vector to be of length m because we use n to denote the total number of policy parameters. Later we may use n to denote the number of features when using linear function approximation to represent structures other than the policy (e.g., when we discuss linear temporal difference learning, we will use n to denote the number of features).

To use linear function approximation for softmax action selection, we store a different vector, θ_a, for each action $a \in \mathcal{A}$. We then compute $p(s, a)$ from (72) as $\theta_a^\intercal \phi(s)$, where $v^\intercal$ denotes the transpose of the vector v. That is,

$$p(s, a) = \theta_a^\intercal \phi(s) \tag{80}$$

$$= \theta_a \cdot \phi(s) \tag{81}$$

$$= \sum_{i=1}^{m} \theta_{a,i} \phi_i(s), \tag{82}$$

where $\theta_{a,i}$ is the i^{th} entry in the vector θ_a and $\phi_i(s)$ is the i^{th} entry in the vector $\phi(s)$. Hence, in softmax action selection using linear function approximation:

$$\pi(s, a, \theta) = \frac{e^{\sigma \theta_a^\intercal \phi(s)}}{\sum_{a' \in \mathcal{A}} e^{\sigma \theta^\intercal_{a'} \phi(s)}}, \tag{83}$$

where θ is one big vector containing $m|\mathcal{A}|$ parameters: one vector $\theta_a \in \mathbb{R}^m$ for each of the $|\mathcal{A}$ actions. Although you might think of θ as being a matrix, I encourage you to think of θ as a vector (the matrix with all of the columns stacked into one big column)—this will simplify later math. Also, we refer to this as "linear" because $\theta_a^\intercal \phi(s)$ is linear in *feature* space, even though it may not be a linear function of s due to nonlinearities in ϕ.

The question remains: how should we define ϕ? Which features allow the agent to represent a good policy? You can view a neural network as learning these features (the output of the second to last layer is $\phi(s)$). However, there are also known choices for $\phi(s)$ that tend to result in strong performance (Konidaris et al., 2011b). We will discuss some of these in more detail later.

Selecting policy representations with small numbers of parameters often speeds up learning. This is because the space of policy parameter vectors that an algorithm must search is lower dimensional—when the problem is phrased as in (68), n is the number of policy parameters, and so fewer policy parameters results in a smaller search space. Thus, deep neural networks should usually be a last resort when selecting a policy representation, since they often have thousands, if not millions of policy parameters.

——————————End of Lecture 6, September 19, 2019——————————

3.3 Cross-Entropy Method

The *cross-entropy* (CE) method for policy search is a simple BBO algorithm that has achieved remarkable performance on domains like playing Tetris (Szita and Lörincz, 2006). We present a variant of CE based on the work of Stulp and Sigaud (2012). Intuitively, CE starts with a multivariate Gaussian distribution over policy parameter vectors. This distribution has mean θ and covariance matrix Σ. It then samples some fixed number, K, of policy parameter vectors from this distribution. Let $\theta_1, \dots, \theta_K$ denote these samples. It evaluates these K sampled policies by running each one for N episodes and averaging the resulting returns. It then picks the K_e best performing policy parameter vectors (for some constant K_e) and fits a multivariate Gaussian to these parameter vectors. The mean and covariance matrix for this fit are stored in θ and Σ and this process is repeated. We present pseudocode for CE in Algorithm 3, which uses the `evaluate` function defined in Algorithm 4.

Algorithm 3: `Cross-Entropy` (CE) for Policy Search.
Input:
1) Initial mean policy parameter vector, $\theta \in \mathbb{R}^n$
2) Initial $n \times n$ covariance matrix, Σ
3) Population, $K \in \mathbb{N}_{>1}$ [for example, $K = 20$]
4) Elite population, $K_e \in \mathbb{N}_{>0}$, where $K_e < K$ [for example, $K_e = 10$]
5) Number of episodes to sample per policy, $N \in \mathbb{N}_{>0}$ [for example, $N = 10$]
6) Small numerical stability parameter $\epsilon \in \mathbb{R}$ [for example, $\epsilon = 0.0001$]

while *true* **do**
  for $k = 1$ **to** K **do**
    $\theta_k \sim N(\theta, \Sigma)$;
    $\hat{J}_k = \texttt{evaluate}(\theta_k, N)$;
  $\texttt{sort}((\theta_1, \hat{J}_1), (\theta_2, \hat{J}_2), \dots, (\theta_K, \hat{J}_K), \text{descending})$;
  $\theta = \frac{1}{K_e} \sum_{k=1}^{K_e} \theta_k$;
  $\Sigma = \frac{1}{\epsilon + K_e} \left(\epsilon I + \sum_{k=1}^{K_e} (\theta_k - \theta)(\theta_k - \theta)^\intercal \right)$;

Algorithm 4: `evaluate`
Input:
1) Policy parameter vector, $\theta \in \mathbb{R}^n$
2) Number of episodes to sample, $N \in \mathbb{N}_{>0}$ [for example, $N = 10$]

Run the parameterized policy using policy parameters θ for N episodes;
Compute the resulting N returns, $G^1, G^2, \dots, G^N$, where $G^i = \sum_{t=0}^{\infty} R_t^i$.;
Return $\frac{1}{N} \sum_{i=1}^{N} G^i$;

3.4 First-Choice Hill-Climbing

Another simple BBO algorithm is *First-Choice Hill-Climbing* (FCHC) (Russell et al., 2003). CE, presented above, has several hyperparameters that must be set properly for the algorithm to work well. FCHC has remarkably few hyperparameters, which makes it easy to apply. Pseudocode for a variant of FCHC, created for this class, is presented in Algorithm 5.

Algorithm 5: `First-Choice Hill-Climbing` (FCHC) for Policy Search.

Input:

1) Initial mean policy parameter vector, $\theta \in \mathbb{R}^n$

2) Exploration parameter, $\sigma \in \mathbb{R}$

3) Number of episodes to sample per policy, $N \in \mathbb{N}_{>0}$ [for example, $N = 10$]

```
Ĵ = evaluate(θ, N);
while true do
    θ' ~ N(θ, σI);
    Ĵ' = evaluate(θ', N);
    if Ĵ' > Ĵ then
        θ = θ';
        Ĵ = Ĵ';
```

3.5 Evaluating RL Algorithms

There are many different ways that researchers report the performance of their algorithms. The most common method is for researchers to optimize all of the hyperparameters for all of the algorithms, and then plot *learning curves* for each algorithm, which show how quickly the agent learns when using each RL algorithm. The details of this process are *not* standardized across the community. An example of a learning curve is depicted in Figure 8.

There is no agreed upon standard for how the optimization of hyperparameters should be performed. Some researchers use grid searches and others random searches. Research on hyperparameter optimization suggests that you should at least use a random search (Bergstra and Bengio, 2012) rather than a grid search. You might consider using a BBO algorithm to optimize the hyperparameters of the algorithm. There is also no agreed upon objective for the hyperparameter optimization: some authors choose the hyperparameters that maximize the area under the learning curve (the sum of returns over a fixed number of episodes, averaged over several trials), while others optimize for the performance of the final policy after a fixed number of episodes.

Notice that this approach for reporting the performance of RL algorithms does not capture how difficult it is to find good hyperparameters for each algorithm. This problem is not new—there has been a push for many years now to report the sensitivity of algorithms to their hyperparameters, and increasingly many papers provide plots showing hyperparameter sensitivity. Notice also that some

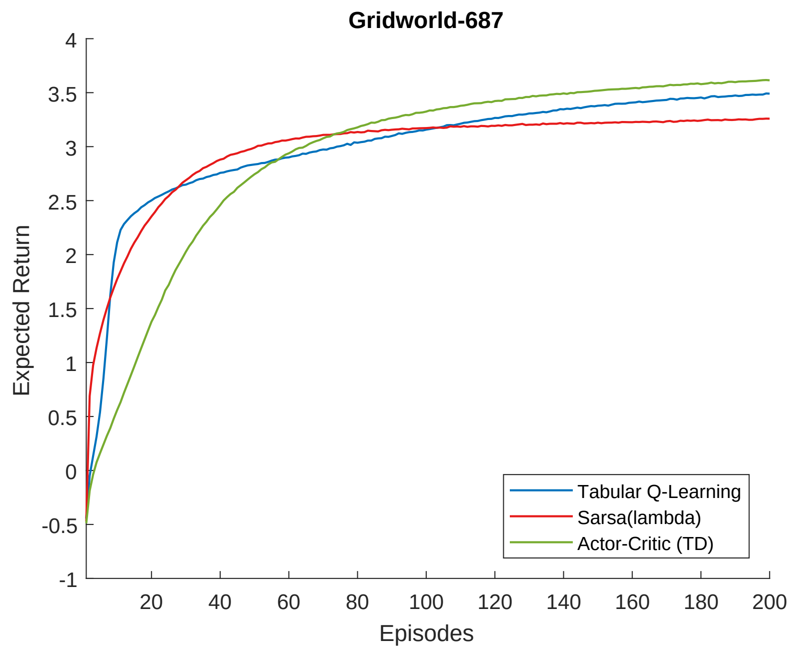


Figure 8: Example learning curve. The horizontal axis shows the number of episodes that the agent has been learning for, and the vertical axis shows the mean return. Here the "expected return" is averaged over many trials—many lifetimes of the agent. This plot was averaged over 100,000 trials. That is, 100,000 of each type of agent (Sarsa, Q-learning, and Actor-Critic) were created and trained from scratch for 200 episodes each. The discounted returns of these 100,000 agents on the 20th episode were averaged to create the points at the 20th episode mark on the horizontal axis. Standard error error-bars are included, but are too small to be seen. It is not uncommon for researchers to provide standard *deviation* error bars. Researchers also sometimes plot total time-steps on the horizontal axis (particularly when episode lengths vary significantly with policy performance), and cumulative reward (total reward since the first step of the first episode) on the vertical axis.

papers present similar looking plots, but report statistics other than expected return (Johns, 2010, Page 80).

——————————End of Lecture 7, September 24, 2019——————————

CMPSCI 687 Homework 2

Due October 3, 2019, 11:55pm Eastern Time

Instructions: This homework consists of a programming portion only. Collaboration is not allowed on any part of this assignment. Submissions must be typed (hand written and scanned submissions will not be accepted). You must use LaTeX. The coding portion of the assignment should be submitted on Gradescope as a .zip file containing your code (see details below). The written response answers should be submitted to Gradescope via a .pdf file and a tagged with the relevant pages. You **must** use the template we have provided, which is located at the github repository here. You may not use any reinforcement learning or machine learning specific libraries in your code, e.g., TensorFlow or PyTorch (you may use libraries like numpy and matplotlib though). If you are unsure whether you can use a library, ask on Piazza. The automated system will not accept assignments after 11:55pm on October 3. The tex file for this homework can be found here. The number of points for this assignment is 100.

Implement Cart-Pole

In this assignment you will implement the cart-pole domain using the (frictionless) dynamics described by Florian (2007, Equations 23 and 24), we provide the expressions below. You **must** use a forward Euler approximation of the dynamics. If the cart hits the boundary of the track, the pole falls below a fail angle, or the time limit is exceeded, then terminate the episode. **You may not use existing RL code for this problem—you must implement the agent and environment entirely on your own and using the class template.**

The Cart-pole environments consists of two interacting bodies: a cart with position x and velocity $\dot{x}$, and a pole with angle θ and angular velocity $\dot{\theta}$. An expression for the pole's angular acceleration is

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta \left(\frac{-F - m_p l \dot{\theta}^2 \sin\theta}{m_c + m_p} \right)}{l \left(\frac{4}{3} - \frac{m_p \cos^2\theta}{m_c + m_p} \right)}, \tag{84}$$

where g is the acceleration due to gravity, F is the force applied to the cart, m_p is the mass of the pole, m_c is the mass of the cart, and l is half the length of the pole. An expression for acceleration of the cart is

$$\ddot{x} = \frac{F + m_p l (\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta)}{m_c + m_p}. \tag{85}$$

The system state is described by the state vector $\mathbf{x} := [x, v, \theta, \omega]$, where $v = \dot{x}$ and $\omega = \dot{\theta}$.

The state space equations of motion are:

$$\dot{x}_t = v_t \tag{86}$$

$$\dot{v}_t = \frac{F + m_p l(\dot{\theta}_t^2 \sin\theta_t - \dot{\omega}_t \cos\theta_t)}{m_c + m_p} \tag{87}$$

$$\dot{\theta}_t = \omega_t \tag{88}$$

$$\dot{\omega}_t = \frac{g\sin\theta_t + \cos\theta_t\left(\frac{-F - m_p l \omega_t^2 \sin\theta_t}{m_c + m_p}\right)}{l\left(\frac{4}{3} - \frac{m_p \cos^2\theta_t}{m_c + m_p}\right)}. \tag{89}$$

Using the Euler approximation the system evolves as:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \Delta t \dot{\mathbf{x}}_t, \tag{90}$$

where $\dot{\mathbf{x}}_t = [\dot{x}_t, \dot{v}_t, \dot{\theta}_t, \dot{\omega}_t]$ and Δt is the time in seconds between updates. In this environment, an agent's action specifies a force $F \in [-F_{\text{mag}}, F_{\text{mag}}]$ on the cart. Traditionally, in RL, the cart pole environment is specified with two discrete actions, where $a_0 = -F_{\text{mag}}$ and $a_1 = F_{\text{mag}}$. You must use this action space in creating the environment.

Use the following values for the cart-pole constants:

- Fail angle $= \pi/12$. (If it exceeds this value or its negative, the episode ends in failure.)
- Cart boundaries are at $x = -3m$ and $x = 3m$.
- Max motor force magnitude $F_{\text{mag}} = 10.0$ (force on cart in Newtons).
- Gravitational constant g is 9.8.
- Cart mass $m_c = 1.0$ kg.
- Pole mass $m_p = 0.1$ kg.
- Pole half-length $l = 0.5m$.
- $\Delta t = 0.02$ seconds (time step).
- Max time before end of episode = 20 seconds.

Genetic Algorithms

Genetic algorithms (GAs) are a family of biologically inspired BBO algorithms that are empirically competitive with deep reinforcement learning methods (Such et al., 2017). Intuitively, the GA starts with a population of *candidate solutions* (in our case, parameterized policies θ) and iteratively modifies, or evolves, the population. We call the number of iterations over which a population is evolved the number of *generations*.

At every generation, the GA uses three main methods, called *genetic operators*, to create the next generation. These genetic operators are known as parent selection, mutation operators, and crossover operators. *Parent selection* chooses which candidate solutions in the current generation will be used as parents for creating children in the next generation. *Mutation operators* apply random changes to parents to create children for the next generation. *Crossover operators* combine two or more parents to create children for the next generation.

Example pseudocode for a simple GA is below. Pseudocode for the functions `get_parents` and `get_children` are purposefully not provided. At each generation, every candidate solution, θ_i, in the population, $P = \{\theta_i\}_{i=0}^{K}$, is evaluated and a *fitness score* for θ_i is produced. The top K_p candidate solutions are then chosen to become parents of the next generation (this parent selection method is known as truncation selection). Children are created by selecting a parent uniformly at random to mutate by applying additive Gaussian noise to the parent: $\theta_{\text{child}} = \theta_{\text{parent}} + \alpha\epsilon$, where $\epsilon \sim \mathcal{N}(0,1)$ and α is a hyperparameter. K_e candidate solutions in the new generation are unmodified copies of the top K_e candidate solutions from the previous generation. This is known as *elitism*.

Algorithm 6: `Genetic Algorithm` (GA) for Policy Search.
Input:
1) Number of generations, $G \in \mathbb{N}_{>1}$ [for example, $G = 300$]
2) Initial population, $P = \{\theta_i\}_{i=0}^{K}$
3) Population size, $K \in \mathbb{N}_{>1}$ [for example, $K = 20$]
4) Truncation index, $K_p \in \mathbb{N}_{>0}$, where $K_p < K$ [for example, $K_p = 10$]
5) Elite population, $K_e \in \mathbb{N}_{>0}$, where $K_e < K$ [for example, $K_e = 10$]
6) Number of episodes to sample per policy, $N \in \mathbb{N}_{>0}$ [for example, $N = 10$]
7) Learning parameter $\alpha \in \mathbb{R}$ [for example, $\alpha = 2.5$]

for $g = 1$ **to** G **do**
  for $k = 1$ **to** K **do**
    $\hat{J}_k = \texttt{evaluate}(\theta_k, N)$;
  $\texttt{sorted} = \texttt{sort}((\theta_1, \hat{J}_1), (\theta_2, \hat{J}_2), \ldots, (\theta_K, \hat{J}_K), \text{descending})$;
  $\texttt{parents} = \texttt{get\_parents}(K_p, \texttt{sorted})$;
  $\texttt{next\_gen} = \texttt{sorted}[1 : K_e].\text{append}(\texttt{get\_children}(\alpha, \texttt{parents}))$;

Algorithm 7: `evaluate`
Input:
1) Policy parameter vector, $\theta \in \mathbb{R}^n$
2) Number of episodes to sample, $N \in \mathbb{N}_{>0}$ [for example, $N = 10$]

Run the parameterized policy using policy parameters θ for N episodes;
Compute the resulting N returns, $G^1, G^2, \ldots, G^N$, where $G^i = \sum_{t=0}^{\infty} R_t^i$.;
Return $\frac{1}{N}\sum_{i=1}^{N} G^i$;

Note that we do not provide pseudocode for the `get_parents` and `get_children` methods. It will be up to you to define these methods.

Problems

Six questions ask for a plot. You must report two plots: one for cart-pole and one for 687-Gridworld, where each of these two plots has a curve corresponding to the cross-entropy method, a curve corresponding to the GA method, and a curve corresponding to first-choice hill-climbing. In the coding problems your solutions will be graded via an autograder on Gradescope. To submit assignments you must zip the rl687 folder and its structure must be the same as the template. You must use the same files provided in the template and fill in the missing function bodies. Correct solutions can be fit within the provided files, but you can create additional files as necessary for the assignment. However, if any errors occur in using them the questions will be marked as zero. The functions will need to match our outputs exactly so pay attention to types (we use np.float64 and np.int for arrays, int, float, and bool else where). The autograder will give results before the assignment is due so you can check your implementations for correctness.

Coding Problems:

1. (15 Points) Implement the cart-pole environment as specified above.
2. (5 Points) Implement the tabular softmax policy.
3. (10 Points) Implement the cross-entropy method as described in the class notes.
4. (10 Points) Implement the First-choice hill-climbing algorithm as described in the class notes.
5. (10 Points) Implement the GA algorithm described earlier in this assignment.

Written Response:

1. (7.5 Points) Apply the CEM algorithm to the More-Watery 687-Gridworld. Use a tabular softmax policy. Search the space of hyperparameters for hyperparameters that work well. Report how you searched the hyperparameters, what hyperparameters you found worked best, and present a learning curve plot using these hyperparameters, as described in class. This plot may be over any number of episodes, but should show convergence to a nearly optimal policy. The plot should average over at least 500 trials and should include standard deviation error bars.
2. (7.5 Points) Repeat the previous question, but using first-choice hill-climbing on the More-Watery 687-Gridworld domain. Report the same quantities.
3. (7.5 Points) Repeat the previous question, but using the GA (as described earlier in this assignment) on the More-Watery 687-Gridworld domain. Report the same quantities.

4. (7.5 Points) Repeat the previous question, but using the cross-entropy method on the cart-pole domain. Notice that the state is not discrete, and so you cannot directly apply a tabular softmax policy. It is up to you to create a representation for the policy for this problem. Consider using the softmax action selection using linear function approximation as described in the notes. For descriptions of common choices of ϕ, see the work of Konidaris et al. (2011b). Report the same quantities, as well as how you parameterized the policy.

5. (7.5 Points) Repeat the previous question, but using first-choice hill-climbing (as described in class) on the cart-pole domain. Report the same quantities and how the policy was parameterized.

6. (7.5 Points) Repeat the previous question, but using the GA (as described earlier in this homework) on the cart-pole domain. Report the same quantities and how the policy was parameterized.

7. (5 Points) Reflect on this problem. Was it easier or harder than you expected to get these methods working? In the previous assignment you hypothesized how long it would take an agent to solve the More-Watery 687-Gridworld problem. Did it take more or fewer episodes than you expected? Why do you think this happened?

4 Value Functions

So far we have described the problem we want to solve mathematically, and have described how BBO methods can be applied. These BBO algorithms do not leverage the assumed MDP structure of the environment. We will now present *value functions*, which are a tool that we will use to leverage the MDP structure of the environment. Notice that value function are *not* a complete agent on their own.

4.1 State-Value Function

The *state-value function*, $v^\pi : \mathcal{S} \to \mathbb{R}$, is defined as follows, for all s:

$$v^\pi(s) \coloneqq \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, \pi\right]. \tag{91}$$

Using the G_t notation we have the equivalent definition (you may provide this as the definition when asked):

$$v^\pi(s) \coloneqq \mathbf{E}\left[G_t | S_t = s, \pi\right]. \tag{92}$$

In English, $v^\pi(s)$ is the expected discounted return if the agent follows policy π from state s. Notice that this quantity depends on the policy, π. More informally, $v^\pi(s)$, is a measure of how "good" it is for the agent to be in state s when using policy π. We call $v^\pi(s)$ the *value of state s*.

Notice that we use t on the right side of (91), even though it does not appear on the left side. This is because the right side takes the same value for all t. We can show this as follows:

$$
\begin{aligned}
v^\pi(s) :=& \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, \pi\right] && (93)\\
=& \sum_{k=0}^{\infty} \gamma^k \mathbf{E}\left[R_{t+k} | S_t = s, \pi\right] && (94)\\
=& \mathbf{E}[R_t|S_t = s, \pi] + \gamma \mathbf{E}[R_{t+1}|S_t = s, \pi] + \gamma^2 \mathbf{E}[R_{t+2}|S_t = s, \pi] + \cdots && (95)\\
=& \sum_{a \in \mathcal{A}} \Pr(A_t = a|S_t = s, \pi) \mathbf{E}[R_t|S_t = s, A_t = a, \pi] && (96)\\
& + \gamma \sum_{a \in \mathcal{A}} \Pr(A_t = a|S_t = s, \pi) \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s'|S_t = s, A_t = a, \pi) && (97)\\
& \times \sum_{a' \in \mathcal{A}} \Pr(A_{t+1} = a'|S_{t+1} = s', S_t = s, A_t = a, \pi) && (98)\\
& \times \mathbf{E}[R_{t+1}|S_{t+1} = s', A_{t+1} = a', S_t = s, A_t = a, \pi] && (99)\\
& \ldots && (100)\\
=& \sum_{a \in \mathcal{A}} \pi(s, a) R(s, a) && (101)\\
& + \gamma \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} P(s, a, s') \sum_{a' \in \mathcal{A}} \pi(s', a') R(s', a') && (102)\\
& + \gamma^2 \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} \pi(s', a') \sum_{s'' \in \mathcal{S}} P(s', a', s'') \sum_{a'' \in \mathcal{A}} \pi(s'', a'') R(s'', a'') && (103)\\
& \ldots, && (104)
\end{aligned}
$$

where $\times$ denotes scalar multiplication split across two lines. Notice that t does not show up in any terms in the last expression, and so regardless of the value of t, $v^\pi(s)$ takes the same value. Hence, the following definitions are equivalent (do not provide these when asked for the definition of the state value function):

$$
\begin{aligned}
v^\pi(s) :=& \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, \pi\right] && (105)\\
v^\pi(s) :=& \mathbf{E}\left[G | S_0 = s, \pi\right] && (106)\\
v^\pi(s) :=& \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| S_0 = s, \pi\right]. && (107)
\end{aligned}
$$

As an example, consider the MDP depicted in Figure 9. For this MDP:

$$v^{\pi_1}(s_1) = 0 \tag{108}$$
$$v^{\pi_1}(s_2) = 12\gamma^0 = 12 \tag{109}$$
$$v^{\pi_1}(s_3) = 0\gamma^0 + 12\gamma^1 = 6 \tag{110}$$
$$v^{\pi_1}(s_4) = 0\gamma^0 + 0\gamma^1 + 12\gamma^2 = 3 \tag{111}$$
$$v^{\pi_1}(s_5) = 0\gamma^0 + 0\gamma^1 + 0\gamma^2 + 12\gamma^3 = 1.5 \tag{112}$$
$$v^{\pi_1}(s_6) = 0 \tag{113}$$

$$v^{\pi_2}(s_1) = 0 \tag{114}$$
$$v^{\pi_2}(s_2) = 0\gamma^0 + 0\gamma^1 + 0\gamma^2 + 2\gamma^3 = 1/4 \tag{115}$$
$$v^{\pi_2}(s_3) = 0\gamma^0 + 0\gamma^1 + 2\gamma^2 = 1/2 \tag{116}$$
$$v^{\pi_2}(s_4) = 0\gamma^0 + 2\gamma^1 = 1 \tag{117}$$
$$v^{\pi_2}(s_5) = 2\gamma^0 = 2 \tag{118}$$
$$v^{\pi_2}(s_6) = 0. \tag{119}$$

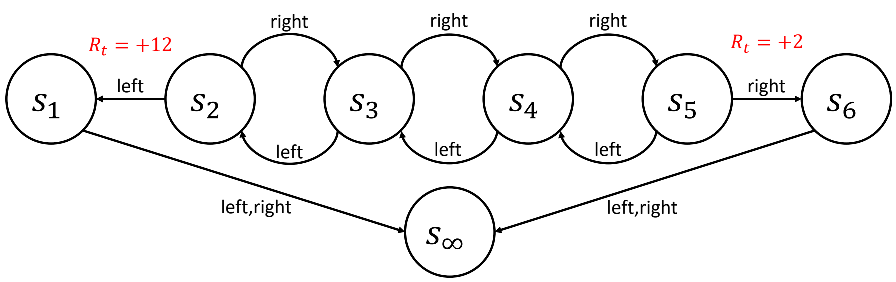


Figure 9: A simple MDP that we will call the "chain" MDP. There are many "chain" MDPs used in the RL literature—this is not a standard one. In each state the agent can choose to move left (L) or right (R), and the transition function is deterministic in implementing these transitions. In states s_1 and s_6, both actions cause a transition to s_∞. The rewards are always zero, except for when the agent transitions from s_2 to s_1, in which case the reward is $+12$, or when the agent transitions from s_5 to s_6, in which case the reward is $+2$. The initial state distribution is not specified. For simplicity, let $\gamma = 0.5$. We will consider two policies for this MDP. The first, π_1, always selects the left action, while the second, π_2, always selects the right action.

4.2 Action-Value Function

The *action-value function*, also called the *state-action value function* or *Q-function*, is defined as:

$$q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \tag{120}$$

$$q^\pi(s,a) := \mathbf{E}[G_t | S_t = s, A_t = a, \pi], \tag{121}$$

for all s and a. Recall that conditioning on π denotes that $A_t \sim \pi(S_t, \cdot)$ for all times, t, where A_t has not otherwise been specified. Here A_t has been specified, and so conditioning on π only applies to time steps other than t. That is, $q^\pi(s,a)$ is the expected discounted return if the agent takes action a in state s and follows the policy π thereafter. Equivalent definitions of q^π are:

$$q^\pi(s,a) := \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, A_t = a, \pi\right] \tag{122}$$

$$q^\pi(s,a) := \mathbf{E}\left[G | S_0 = s, A_0 = a, \pi\right] \tag{123}$$

$$q^\pi(s,a) := \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| S_0 = s, A_0 = a, \pi\right]. \tag{124}$$

For the chain MDP depicted in Figure 9:

$$q^{\pi_1}(s_1, L) = 0 \tag{125}$$

$$q^{\pi_1}(s_1, R) = 0 \tag{126}$$

$$q^{\pi_1}(s_2, L) = 12\gamma^0 = 12 \tag{127}$$

$$q^{\pi_1}(s_2, R) = 0\gamma^0 + 0\gamma^1 + 12\gamma^2 = 3 \tag{128}$$

$$q^{\pi_1}(s_3, L) = 0\gamma^0 + 12\gamma^1 = 6 \tag{129}$$

$$q^{\pi_1}(s_3, R) = 0\gamma^0 + 0\gamma^1 + 0\gamma^2 + 12\gamma^3 = 1.5. \tag{130}$$

Notice that $q^\pi(s,a)$ and $v^\pi(s)$ are both always zero if s is a terminal state. Also, take particular note of (128)—it shows a nuance of q-values that is often missed. That is, the agent begins in s_2 and takes the action to go right. It then takes the action to go left, bringing it back to s_2. Now when it is again in s_2, it takes the action to go left. In this sense, the q-function considers the behavior of an agent that is not following a fixed policy. For more on this topic, see the work of Bellemare et al. (2016) for further discussion of this "inconsistency".

4.3 The Bellman Equation for v^π

The *Bellman equation* is a recursive expression for the value function—a sort of consistency condition that the value function satisfies. Specifically, the Bellman equation for the state-value function can be derived from the definition of the state-value function:

$$
\begin{aligned}
v^\pi(s) :=& \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, \pi\right] && (131)\\
=& \mathbf{E}\left[R_t + \sum_{k=1}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, \pi\right] && (132)\\
=& \mathbf{E}\left[R_t + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k} \middle| S_t = s, \pi\right] && (133)\\
\overset{(a)}{=}& \sum_{a\in\mathcal{A}} \pi(s,a) R(s,a) + \mathbf{E}\left[\gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, \pi\right] && (134)\\
\overset{(b)}{=}& \sum_{a\in\mathcal{A}} \pi(s,a) R(s,a) + \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s') && (135)\\
& \times \mathbf{E}\left[\gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a, S_{t+1} = s', \pi\right] && (136)\\
\overset{(c)}{=}& \sum_{a\in\mathcal{A}} \pi(s,a) R(s,a) + \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s') && (137)\\
& \times \gamma \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_{t+1} = s', \pi\right] && (138)\\
\overset{(d)}{=}& \sum_{a\in\mathcal{A}} \pi(s,a) R(s,a) + \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s') \gamma v^\pi(s') && (139)\\
=& \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s') \left(R(s,a) + \gamma v^\pi(s')\right), && (140)
\end{aligned}
$$

where $\times$ denotes scalar multiplication split across two lines, **(a)** comes from modifying the indexing of the sum to start at zero instead of one, but changes all uses of k within the sum to $k+1$, **(b)** comes from marginalizing over A_t and S_{t+1}, **(c)** follows from the Markov property, and **(d)** comes from the definition of the state-value function (see (91)).

This final expression gives the Bellman equation for v^π:

$$
v^\pi(s) = \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s') \left(R(s,a) + \gamma v^\pi(s')\right). \tag{141}
$$

Another way to understand the Bellman equation for v^π is to consider the expansion:

$$
\begin{aligned}
v^\pi(s) =& \mathbf{E}\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots \middle| S_t = s, \pi\right] && (142)\\
=& \mathbf{E}\left[R_t + \gamma\left(R_{t+1} + \gamma R_{t+2} + \dots\right) | S_t = s, \pi\right] && (143)\\
=& \mathbf{E}\left[R_t + \gamma v^\pi(S_{t+1}) | S_t = s, \pi\right] && (144)\\
=& \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')\left(R(s,a) + \gamma v^\pi(s')\right). && (145)
\end{aligned}
$$

Consider the "Cookie MDP" depicted in Figure 10, which was created for this course (it is not a standard domain). Clearly $v^\pi(s_3) = 0$. We can then compute $v^\pi(s_2)$ in two different ways. The first is to use the definition of the value function (and the property that state transitions are deterministic in this case):

$$v^\pi(s_2) = R_2 + \gamma R_3 = 10 + \gamma 0 = 10. \tag{146}$$

The second approach is to use the Bellman equation (and the property that state transitions are deterministic in this case):

$$v^\pi(s_2) = R_2 + \gamma v^\pi(s_3) = 10 + \gamma 0 = 10. \tag{147}$$

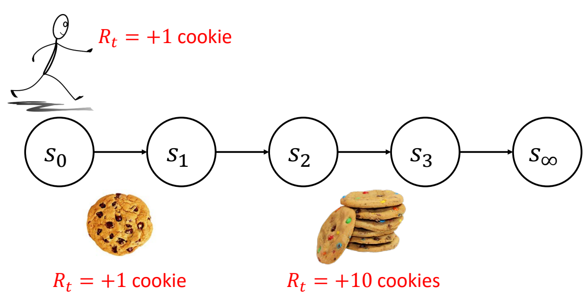


Figure 10: The "Cookie MDP". Under some policy, π, the agent always begins in s_0, and walks down the line of states. The agent receives a reward of +1 when transitioning from state s_0 to s_1 and a reward of +10 when transitioning from s_2 to s_3. All other transitions result in a reward of zero.

Similarly, we can compute $v^\pi(s_1)$ using the definition of the value function or the Bellman equation:

$$
\begin{aligned}
v^\pi(s_1) =& R_1 + \gamma R_2 + \gamma^2 R_3 = 0 + \gamma 10 + \gamma^2 0 = \gamma 10. && (148)\\
v^\pi(s_1) =& R_1 + \gamma v^\pi(s_2) = 0 + \gamma 10. && (149)
\end{aligned}
$$

We can also compute $v^\pi(s_0)$ both ways:

$$
\begin{aligned}
v^\pi(s_1) =& R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 = 0 + \gamma 0 + \gamma^2 10 + \gamma^3 0 = \gamma^2 10. && (150)\\
v^\pi(s_1) =& R_0 + \gamma v^\pi(s_1) = 0 + \gamma\gamma 10 = \gamma^2 10. && (151)
\end{aligned}
$$

Notice that already it is becoming easier to compute the value of states using the Bellman equation than it is to compute the values of states from the definition of the value function. This is because the Bellman equation only needs to look forward one time step into the future, while the definition of the value function must consider the entire sequence of states that will occur until the end of the episode. When considering larger problems, and particularly problems with stochastic policies, transition functions, and rewards, the Bellman equation will be increasingly useful, since computing state-values from the definition of the value function would require reasoning about every possible sequence of events that could occur from the occurrence of that state until the end of the episode.

For more intuition about the Bellman equation, imagine that the current state is s. We can view the Bellman equation as breaking the expected return that will occur into two parts: the reward that we will obtain during the next time step, and the value of the next state that we end up in. That is,

$$v^\pi(s) = \mathbf{E}\left[\underbrace{R(s, A_t)}_{\text{immediate reward}} + \gamma \underbrace{v^\pi(S_{t+1})}_{\text{value of next state}} \middle| S_t = s, \pi\right]. \tag{152}$$

This should make intuitive sense, because the value of the next state is the expected discounted sum of rewards that we will obtain from the next state, and so summing the expected immediate reward and the expected discounted sum of rewards thereafter gives the expected discounted sum of rewards from the current state.

4.4 The Bellman Equation for q^π

While the Bellman equation for v^π is a recurrent expression for v^π, the Bellman equation for q^π is a recurrent expression for q^π. Specifically:

$$q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s, a, s') \sum_{a' \in \mathcal{A}} \pi(s', a') q^\pi(s', a'). \tag{153}$$

Question 13. *Can you derive the Bellman equation for $q^\pi(s, a)$ from the definition of q^π?*

Answer 13.

$$
\begin{aligned}
q^\pi(s,a) :=& \mathbf{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k}\middle| S_t=s, A_t=a,\pi\right] && (154)\\
=&\mathbf{E}[R_t|S_t=s,A_t=a,\pi] + \mathbf{E}\left[\sum_{k=1}^{\infty}\gamma^k R_{t+k}\middle| S_t=s, A_t=a,\pi\right] && (155)\\
=&\mathbf{E}[R_t|S_t=s,A_t=a,\pi] + \gamma\mathbf{E}\left[\sum_{k=1}^{\infty}\gamma^{k-1} R_{t+k}\middle| S_t=s, A_t=a,\pi\right] && (156)\\
=&\mathbf{E}[R_t|S_t=s,A_t=a,\pi] + \gamma\mathbf{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle| S_t=s, A_t=a,\pi\right] && (157)\\
=&\mathbf{E}[R_t|S_t=s,A_t=a,\pi] + \gamma\sum_{s'\in\mathcal{S}}\Pr(S_{t+1}=s'|S_t=s,A_t=a,\pi) && (158)\\
&\times\sum_{a'\in\mathcal{A}}\Pr(A_{t+1}=a'|S_{t+1}=s',S_t=s,A_t=a,\pi) && (159)\\
&\times\mathbf{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle| S_t=s, A_t=a,S_{t+1}=s',A_{t+1}=a'\pi\right] && (160)\\
=&R(s,a)+\gamma\sum_{s'\in\mathcal{S}}P(s,a,s')\sum_{a'\in\mathcal{A}}\pi(s',a')q^\pi(s',a'). && (161)
\end{aligned}
$$

4.5 Optimal Value Functions

The *optimal value function*, v^*, is a function $v^*:\mathcal{S}\to\mathbb{R}$ defined by:

$$v^*(s) \coloneqq \max_{\pi\in\Pi} v^\pi(s). \tag{162}$$

Notice that for each state v^* "uses" the policy, π, that maximizes $v^\pi(s)$. Thus, v^* is not necessarily associated with a particular policy—$v^*(s)$ can be the value function associated with different policies depending on which state, s, it is evaluated on.

Consider the relation $\geq$ for policies defined as:

$$\pi \geq \pi' \text{ if and only if } \forall s\in\mathcal{S},\ v^\pi(s)\geq v^{\pi'}(s). \tag{163}$$

Notice that this relation produces a *partial ordering* on the set of policies. This is not a total order on the set of policies because there can exist two policies, π and π', such that both $\pi \not\geq \pi'$ and $\pi' \not\geq \pi$.

Question 14. *Give an example MDP for which there exist policies π and π' such that $\pi \not\geq \pi'$ and $\pi' \not\geq \pi$.*

We now present a definition of an *optimal policy* that differs from our earlier definition. Specifically, an optimal policy, π^* is any policy that is at least as good as all other policies. That is, π^* is an optimal policy if and only if

$$\forall \pi \in \Pi,\, \pi^* \geq \pi. \tag{164}$$

Particularly given that $\geq$ only produces a partial ordering, at this point it may not be clear than such an optimal policy exists.

Later we will prove that for all MDPs where $|\mathcal{S}| < \infty$, $|\mathcal{A}| < \infty$, $R_{\max} < \infty$, and $\gamma < 1$, there exists at least one optimal policy, π^* under this definition of an optimal policy. That is, Property 1 holds for this definition of an optimal policy as well as the definition of an optimal policy in (17). From the definition of v^* in (162), it follows that $v^* = v^{\pi^*}$ for all optimal policies, π^*.

We now have two *different* definitions of an optimal policy. Both definitions are standard in RL research. The definition presented in (17) is common in papers that focus on policy optimization, like BBO algorithms and algorithms that compute the gradient of $J(\theta)$ (we will talk more about these *policy gradient* algorithms later). The definition presented in (164) is common in theoretical reinforcement learning papers and in papers that emphasize the use of value functions.

Question 15. *Which definition of an optimal policy is stricter, the definition in (17) or the definition in (164)?*

Answer 15. *The definition in (164) is stricter. That is, every optimal policy according to (164) is an optimal policy according to (17), but every optimal policy according to (17) is not necessarily optimal according to (164). The difference between these two definitions stems from their requirements for states, s, that are not reachable (from any state in the support of the initial state distribution). The definition in (17) does not place any requirements on the behavior of an optimal policy for these unreachable states, while the definition in (164) requires an optimal policy to act to maximize the expected return if were to ever be placed in these unreachable states.*

Also, notice that even when π^* is not unique, the optimal value function, v^* is unique—all optimal policies share the same state-value function.

———End of Lecture 8, September 26, 2019———

CMPSCI 687 Pop Quiz 2
October 1, 2019

Instructions: You have 5 minutes to complete this quiz. This quiz is **closed** notes—do not use your notes or a laptop. Do not discuss problems with your neighbors until after everyone has handed in their quiz.

Use the notation from class. Don't forget to capitalize your random variables. Presenting equalities that are true is not enough—you must provide the *math definitions* of the symbols on the left.

Fill in the definitions for the following terms:

1. $R(s,a) = \mathbf{E}\left[R_t | S_t = s, A_t = a\right]$

2. $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$

3. $v^\pi(s) = \mathbf{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s, \pi]$

4. $q^\pi(s,a) = \mathbf{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s, A_t = a, \pi]$

5. The Bellman equation says that for all states $s \in \mathcal{S}$,

   $v^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')[R(s,a) + \gamma v^\pi(s')]$

Just as we defined an optimal state-value function, we can define the optimal action-value function, $q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where

$$q^*(s, a) \coloneqq \max_{\pi \in \Pi} q^\pi(s, a). \tag{165}$$

Also like the optimal state-value function, $q^* = q^{\pi^*}$ for all optimal policies, π.

Question 16. *Given v^*, can you compute π^* if you do not know P and R?*

Answer 16. *No. Any action in*

$$\arg\max_{a \in \mathcal{A}} \sum_{s'} P(s, a, s')[R(s, a) + \gamma v^*(s')] \tag{166}$$

is an optimal action in state s. Computing these actions requires knowledge of P and R.

Question 17. *Given q^*, can you compute π^* if you do not know P and R?*

Answer 17. *Yes. Any action in*

$$\arg\max_{a \in \mathcal{A}} q^*(s, a) \tag{167}$$

is an optimal action in state s.

4.6 Bellman Optimality Equation for v^*

The *Bellman optimality equation for v^** is a recurrent expression for v^*. We will show later that it holds for optimal policies, and only for optimal policies.

To try to get a recurrent expression for v^π, we can imagine what would happen if there was an optimal policy π^*, with value function v^*. We can begin with the Bellman equation for this policy π^*:

$$v^*(s) = \sum_{a \in \mathcal{A}} \pi^*(s, a) \underbrace{\sum_{s' \in \mathcal{S}} P(s, a, s') \left[R(s, a) + \gamma v^*(s')\right]}_{q^*(s,a)}. \tag{168}$$

Notice that in state s, π^* will pick the action that maximizes $q^*(s, a)$. So, we do not need to consider all possible actions, a—we only need to consider those that cause the $q^*(s,a)$ term in (168) to be maximized. Thus,

$$v^*(s) = \max_{a \in \mathcal{A}} \underbrace{\sum_{s' \in \mathcal{S}} P(s,a,s') \left[R(s,a) + \gamma v^*(s')\right]}_{q^*(s,a)}. \tag{169}$$

The above equation is the Bellman optimality equation for v^*. We say that a policy, π, *satisfies the Bellman optimality equation* if for all states $s \in \mathcal{S}$:

$$v^\pi(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s') \left[R(s,a) + \gamma v^\pi(s')\right]. \tag{170}$$

A possible misconception is that we have "derived" the Bellman optimality equation formally. We have not—we have not established that there actually exists a policy π^* such that $v^{\pi^*} = v^*$, a property that we used when introducing the Bellman optimality equation. Rather, one should view the Bellman optimality equation at this point as an equation that policies may, or may not satisfy. The Bellman optimality equation will be useful to us because we will establish that 1) if a policy π satisfies the Bellman optimality equation, then it is an optimal policy, and 2) if the state and action sets are finite, rewards are bounded, and $\gamma < 1$, then there exists a policy π that satisfies the Bellman optimality equation. With these two results, we will have established the existence of an optimal policy, π^*.

Once we have established these results, we will have that all optimal policies satisfy the Bellman optimality equation, there exists at least one optimal policy, and all optimal policies π have value functions v^π that are equal to $v^\star$.

Like the Bellman optimality equation for v^*, we can define Bellman optimality equation for q^* as:

$$q^*(s,a) = \sum_{s' \in \mathcal{S}} P(s,a,s') \left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} q^*(s',a')\right]. \tag{171}$$

Question 18. *Derive the Bellman optimality equation for q^* starting with the Bellman equation for q^{π^*}.*

We now establish the first link in our line of reasoning that will allow us to establish the existence of an optimal policy:

Theorem 1. *If a policy π satisfies the Bellman optimality equation, then π is an optimal policy.*

Proof. By the assumption that π satisfies the Bellman optimality equation, we have that for all states s:

$$v^\pi(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')[R(s,a) + \gamma v^\pi(s')]. \tag{172}$$

Applying the Bellman optimality equation again, this time for $v^\pi(s')$, we can expand the above equation:

$$v^\pi(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')\left[R(s,a) + \gamma\left(\max_{a'\in\mathcal{A}} \sum_{s''} P(s',a',s'')(R(s',a') + \gamma v^\pi(s''))\right)\right] \tag{173}$$

We could continue this process, constantly replacing the final v^π term using the Bellman optimality equation. If we could do this infinitely often, we would eliminate π from the expression entirely. Consider what this expression would become if s corresponds to the state at time t. $R(s,a)$ is the first reward, and $R(s',a')$ is the second reward (which is discounted by γ). We would eventually obtain an $R(s'',a'')$ term, which would be discounted by γ^2. The P terms are capturing state transition dynamics. So, ignoring for a moment how actions are selected, this expression is the expected discounted sum of rewards. Now, when computing this expected discounted sum of rewards, which actions are being chosen? At each time, t, the action is chosen that maximizes the expected discounted sum of future rewards (given that in the future the actions are chosen to also maximize the discounted sum of future rewards).

Consider now any policy π'. What would happen if we replaced each $\max_{a\in\mathcal{A}}$ with $\sum_{a\in\mathcal{A}} \pi'(s,a)$? Would the expression become bigger or smaller? We argue that the expression could not become bigger. That is, for any policy π':

$$v^\pi(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')\left[R(s,a) + \gamma\left(\max_{a'\in\mathcal{A}} \sum_{s''} P(s',a',s'')(R(s',a') + \gamma \dots\right)\right] \tag{174}$$

$$\geq \sum_{a\in\mathcal{A}} \pi'(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s')\left[R(s,a) + \gamma\left(\sum_{a'\in\mathcal{A}} \pi'(s',a') \sum_{s''} P(s',a',s'')(R(s',a') + \gamma \dots\right)\right]. \tag{175}$$

If you do not see why this is true, consider any finite set $\mathcal{X}$, any distribution μ over the set X, and any function $f : \mathcal{X} \to \mathbb{R}$. Convince yourself that the following property holds:

$$\max_{x\in\mathcal{X}} f(x) \geq \sum_{x\in\mathcal{X}} \mu(x) f(X). \tag{176}$$

We have simply applied this property repeatedly, where $\mathcal{X}$ is $\mathcal{A}$, μ is $\pi(s,\cdot)$, x is a, and f is the remainder of the expression as a function of a.

Continuing the proof, given that the above holds for all policies π', we have that for all states $s \in \mathcal{S}$ and all policies $\pi' \in \Pi$:

$$
\begin{aligned}
v^\pi(s) =& \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s') \left[R(s,a) + \gamma \left(\max_{a' \in \mathcal{A}} \sum_{s''} P(s',a',s'')(R(s',a') + \gamma \dots \right) \right] && (177)\\
\geq& \mathbf{E}[G_t | S_t = s, \pi'] && (178)\\
=& v^{\pi'}(s). && (179)
\end{aligned}
$$

Hence, for all states $s \in \mathcal{S}$, and all policies $\pi' \in \Pi$, $v^\pi(s) \geq v^{\pi'}(s)$, i.e., for all policies $\pi' \in \Pi$, we have that $\pi \geq \pi'$, and hence π is an optimal policy. $\square$

Notice that this theorem has not established the existence of an optimal policy because we did not show that there exists a policy that satisfies the Bellman optimality equation.

——————————End of Lecture 9, October 1, 2019——————————

5 Reward Design [Guest Lecture by Yash Chandak]

So far in this course, we have seen how a reward function is essential to specify the objective a researcher wants an agent to optimize for. However, in many cases, it is not simple to design a reward function that incentivizes an agent to find a policy that aligns with the researcher's goal for the agent. This section first discusses how some heuristically designed reward functions can *distract* the agent and result in sub-optimal policies. Subsequently, two techniques: (a) Optimal reward functions, and (b) Reward shaping, are discussed which can help in mitigating this problem.

This section also has an associated set of powerpoint slides, which are available here. While not a part of the main syllabus, these slides also contain information related to designing reward functions from languages that can be encoded as a regular expression. Additionally, some methods for teaching robots through active human interaction or demonstrations are also briefly reviewed in the presentation.

5.1 Examples of Bad Reward Functions

There are several noteworthy instances where researchers have unknowingly designed reward functions that did not align with what they wanted. For example, consider the example discussed in this blog post, where the researchers wanted to train a robotic arm simulator to grasp and move a cube. To make the agent do so, they designed a reward function that rewarded the agent based on how much it ultimately moves the cube. Instantly, the agent exploited this reward function and learned to smack the cube hard such that it rolls away the

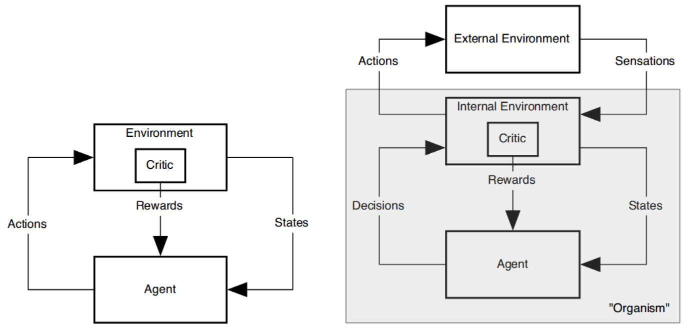


Figure 11

maximum distance. On further training, it learned to grasp the cube and toss it in the air such that it goes even farther!

In another blogpost, the author recalls an incident where an agent was trained to make a simulated *half-cheetah* run. Similar to the previous example, a positive reward was given when the half-cheetah moved a certain distance. Surprisingly, the agent discovered a unique gait to move by flipping on its back and kicking in the air! While this did help it cover the distance and optimize for the total reward, it certainly did not align with what the researchers had aimed to make the agent learn.

5.2 Optimal Reward Function

The foundations of the reinforcement learning framework are motivated by how animals learn and interact with their surroundings. A popular computational abstraction of this interaction process is provided in Figure 11 (left), where the agent observes a state and executes an action that results in a reward and the next state. From the point of view of this abstraction, these rewards, which drive the agent's behavior, are generated completely by the environment. However, Singh et al. (2009a) argue that such an abstraction fails to capture an important aspect of internal motivation in animal learning.

Figure 11 (right) presents an alternate abstraction, where the reward functions that drive the animal behavior is generated by the animal itself. Here, Singh et al. (2009b) argue that the environment only provides the evolutionary basis for the animal (eg., survival, happiness, etc.), and the animal sets its own goals and objectives that help it maximize the evolutionary *fitness*. This viewpoint has a key advantage. It allows different animals to have different goals even though the final evolutionary objective is the same. The internal reward can depend on the way an animal interacts with the environment and can thus possibly help the animal overcome its limitations to better optimize for the evolutionary objective.

We now elaborate on this idea of internal rewards from a more computational point of view. In the following paragraphs, we provide an example that demon-

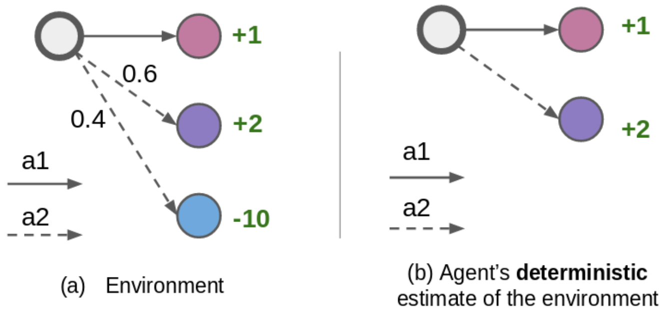


Figure 12

strates how internal rewards can be useful. Then we define what an optimal reward function is for an agent and provide a simple algorithm to find it.

The following example succinctly highlights how an internal reward function can be beneficial for the agent. Consider the four-state MDP in the Figure 12(a). The agent starts in the grey state and has two available actions. Action a1 takes the agent to the red state and rewards +1. Action a2 taken the agent to purple state with probability 0.6 and the blue state with probability 0.4, yielding rewards of +2 and −10, respectively. The optimal action is $a1$.

Now consider an agent that enumerates several random trajectories, estimates a model of the environment using those trajectories, and then acts according to the estimated model. Additionally, due to certain constraints, it can only represent *deterministic* transitions. Therefore, the agent estimates a model where action a2 transitions deterministically to the purple state (Figure 12(b)). According to this model, unfortunately, the agent will consider action a2 to be optimal when using the original reward functions. However, if there is an alternate reward function that assigns the red state a reward of +3 then the agent can learn the optimal policy despite its model error.

While this particular example might seem contrived, it captures the essence of the problem. The original reward function does not depend on the limitations (eg., restricted expressibility of function approximators, computational and memory constraints, modeling errors, etc.) an agent might have while interacting with the environment. But an alternate, agent dependent, reward function can be designed that can help mitigate the negative consequences of these limitations.

Let $J : \mathbb{A} \times \mathcal{M} \times \mathcal{R} \rightarrow \mathbb{R}$ be an objective function that maps the performance of any agent $A \in \mathbb{A}$, in an MDP $M \in \mathcal{M}$, trained using a reward function $r \in \mathcal{R}$. Then the optimal reward function for an agent A in MDP M is,

$$r^* \in \underset{r \in \mathcal{R}}{\arg\max}\, J(A, M, r). \tag{180}$$

An outline of a black-box algorithm for searching for the optimal reward function is given below,

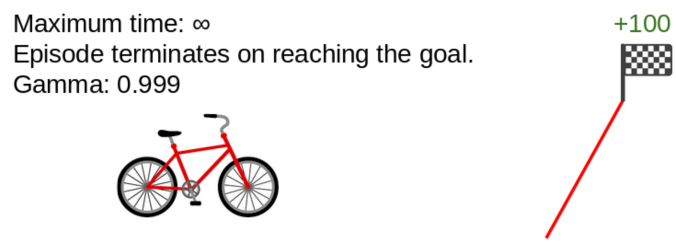


Figure 13

- Draw a sample of candidates for optimal reward function.
- For each candidate reward function, train an agent for a fixed number of episodes.
- For each agent, sample a trajectory after training.
- Evaluate the return obtained in those trajectories on the original reward function.
- Update the set of optimal reward function candidates using the above evaluations.
- Repeat.

An alternative to the black box search for finding the optimal reward function is by doing online gradient descent (Sorg et al., 2010). However, it requires the policy gradient theorem, which will be covered in the latter half of the course.

5.3 Reward Shaping

One of the major challenges for a reinforcement learning agent is the availability of only sparse rewards. This makes finding the optimal trajectory extremely hard as there is no intermediate help during exploration. For example, if an agent has four actions available at any state and the horizon length is 20, then the number of possible trajectories is in order of 4^{20}. Among them, if there exists only one trajectory that yields the goal reward then random exploration is unlikely to stumble across that trajectory even once.

Consider the problem in Figure 13, where the task for an agent is to ride a bicycle and reach the finish line. The episode ends when the agent reaches the goal line, otherwise, there is no time limit for solving the problem. On reaching the goal, the agent receives a reward of +100 and 0 otherwise. This domain also suffers from sparse rewards problem. One natural heuristic to help the agent reach the goal is by awarding a +1 *helping reward* whenever the agent moves towards the finish line. However, this heuristic has a major drawback: As the episode never ends until the agent reaches the finish line, the agent can keep moving to and fro to accumulate the helping reward whose net worth is more than the goal reward.

While there are other possible heuristics that will not have this problem (eg., additionally penalizing the agent with -1 when it moves away from the finish line), it shows that heuristics can often fail. As the domains become increasingly more complex, fixing this heuristic can become difficult. In the following paragraphs, we discuss the underlying problem with such heuristics and present a principled way to design helping rewards.

The core reason behind the problem in the bicycle domain can be reduced to the fact that the agent can find a loop that has a net positive return from the helping reward (Note: A net positive return from the actual reward function is fine). This permanently *distracts* the agent from the main task as the agent can exploit this loop to obtain a return higher than what it would have by actually solving the desired task. To mitigate this problem, the helping reward should be designed such that it neither positively nor negatively distracts the agent permanently. That is, while it would be beneficial to have a helping reward that can help the agent quickly find the desired trajectory, its net impact should be zero.

Let $R(s,a,s')$ be the original reward function and $F(s,a,s')$ be the helping reward function, then the new reward function $R'(s,a,s') \coloneqq R(s,a,s') + F(s,a,s')$ should be such that for any trajectory,

$$\sum_{t=0}^{\infty} \gamma^t R'(s_t, a_t, s_{t+1}) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}). \tag{181}$$

This would imply that $\sum_{t=0}^{\infty} \gamma^t F(s_t, a_t, s_{t+1}) = 0$. Ng et al. (1999) showed that using *any* function $\Phi(s) : \mathcal{S} \to \mathbb{R}$, if $F \coloneqq \gamma\Phi(s') - \Phi(s)$ then (181) can be satisfied. To see this intuitively, consider unrolling (181) using the F as defined above,

$$\begin{aligned}
\sum_{t=0}^{\infty} \gamma^t R'(s_t, a_t, s_{t+1}) =& R(s_0, a_0, s_1) + \gamma\Phi(s_1) - \Phi(s_0) && (182)\\
&+ \gamma \left(R(s_1, a_1, s_2) + \gamma\Phi(s_2) - \Phi(s_1)\right) && (183)\\
&... && (184)\\
&+ \gamma^t \left(R(s_t, a_t, s_{t+1}) + \gamma\Phi(s_{t+1}) - \Phi(s_t)\right) && (185)\\
&... && (186)\\
=& - \Phi(s_0) + \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) && (187)
\end{aligned}$$

It can be noticed that in (186), all the $\Phi(s_t)$ terms cancel each other out, except $\Phi(s_0)$ and $\gamma^{t+1}\Phi(s_{t+1})$. For continuing tasks, as $t \to \infty$, $\gamma^{t+1} \to 0$, therefore $\gamma^{t+1}\Phi(s_{t+1}) \to 0$. For episodic tasks, s_{t+1} corresponds to the absorbing state and $\Phi(s_{t+1})$ can be easily defined to be 0. Regarding the starting state, to ensure (181) holds with exact equality, $\Phi(s_0)$ is also be defined to be 0. Ng et al. (1999)

refer to such a F as the *potential-based* shaping reward.[4] The following theorem formally establishes this intuition.

Theorem 2 (Ng et al. (1999))**.** *For any $\Phi : \mathcal{S} \to \mathbb{R}$, if $F(s, a, s') \coloneqq \gamma\Phi(s') - \Phi(s)$, then any optimal policy for the MDP $M \coloneqq (\mathcal{S}, \mathcal{A}, P, \gamma, d_0, R)$ is also an optimal policy for the MDP $M' \coloneqq (\mathcal{S}, \mathcal{A}, P, \gamma, d_0, R + F)$, and vice-versa.*

Proof. Consider the Bellman optimality equation for M,

$$q^*_M(s, a) = \sum_{s' \in \mathcal{S}} P(s, a, s') \left[R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} q^*_M(s', a') \right]. \tag{188}$$

Note that (188) can also be expressed as,

$$q^*_M(s, a) - \Phi(s) = \sum_{s' \in \mathcal{S}} P(s, a, s') \left[R(s, a, s') + \gamma\Phi(s') - \Phi(s) + \gamma \max_{a' \in \mathcal{A}} \left(q^*_M(s', a') - \Phi(s')\right) \right]. \tag{189}$$

Let, $\hat{q}_{M'}(s, a) \coloneqq q^*_M(s, a) - \Phi(s)$, then (189) is,

$$\hat{q}_{M'}(s, a) = \sum_{s' \in \mathcal{S}} P(s, a, s') \left[R(s, a, s') + F(s, a, s') + \gamma \max_{a' \in \mathcal{A}} \hat{q}_{M'}(s', a') \right] \tag{190}$$

$$= \sum_{s' \in \mathcal{S}} P(s, a, s') \left[R'(s, a, s') + \gamma \max_{a' \in \mathcal{A}} \hat{q}_{M'}(s', a') \right]. \tag{191}$$

Notice that (191) is the Bellman optimality equation for M'. Therefore,

$$\pi^*_{M'}(s) \in \arg\max_{a \in \mathcal{A}} \; \hat{q}_{M'}(s, a) \tag{192}$$

$$= \arg\max_{a \in \mathcal{A}} \; q^*_M(s, a) - \Phi(s) \tag{193}$$

$$= \arg\max_{a \in \mathcal{A}} \; q^*_M(s, a) \tag{194}$$

$$= \pi^*_M(s). \tag{195}$$

□

Theorem 2 shows that $F(s, a, s') \coloneqq \gamma\Phi(s') - \Phi(s)$ is sufficient to preserve the optimal policy. Ng et al. (1999) in their work also prove that this is not only sufficient but also the necessary condition. In other words, there is no other way of defining $F(s, a, s')$ that ensures this property, unless some other assumptions are made. The proof of this is beyond the scope of this lecture and interested readers can refer appendix in the work by Ng et al. (1999).

[4] My guess is that this terminology has its root in physics, where traditionally potential functions are used when either the net work done in moving a particle around a loop is 0, or else it only depends on the start and the final position of the particle.

Question 19. *How is the optimal policy affected if* $\Phi(s_0) \neq 0$?

Answer 19. *It will not impact on the optimal policy as reward in the starting state does not impact any decision making steps.*

Question 20. *What is the optimal policy when only a potential-based shaping reward is provided to the agent?*

Question 21. *What is the impact of reward shaping on policies which are not optimal? Are the partial orderings preserved?*

———————End of Lecture 10, October 3, 2019———————

6 Policy Iteration and Value Iteration

So far we have defined the problem that we would like to solve, discussed BBO algorithms, which ignore the MDP structure of the problem, and defined value functions that more sophisticated algorithms will use to leverage the MDP structure of the environment. In this section we will show how value functions can be used to efficiently solve for the optimal policies of finite MDPs *when the transition function and reward function are known*. That is, in this section we will present standard *planning* algorithms. We present these algorithms because the later RL algorithms (which do not require P and R to be known) are closely related to these algorithms (they can be viewed as stochastic approximations to these planning algorithms). We begin with the question of how the value function for a policy can be computed.

6.1 Policy Evaluation

Here we consider the question: given a policy, π, how can we efficiently compute v^π? Notice that the Bellman equation provides us with $|\mathcal{S}|$ equations and $|\mathcal{S}|$ unknown variables, $v^\pi(s_1), v^\pi(s_2), \dots, v^\pi(s_{|\mathcal{S}|}$. These equations are:

$$v^\pi(s_1) = \sum_{a \in \mathcal{A}} \pi(s_1, a) \sum_{s' \in \mathcal{S}} P(s_1, a, s') \left(R(s_1, a) + \gamma v^\pi(s')\right) \tag{196}$$

$$v^\pi(s_2) = \sum_{a \in \mathcal{A}} \pi(s_2, a) \sum_{s' \in \mathcal{S}} P(s_2, a, s') \left(R(s_2, a) + \gamma v^\pi(s')\right) \tag{197}$$

$$\dots$$

$$v^\pi(s_{|\mathcal{S}|}) = \sum_{a \in \mathcal{A}} \pi(s_{|\mathcal{S}|}, a) \sum_{s' \in \mathcal{S}} P(s_{|\mathcal{S}|}, a, s') \left(R(s_{|\mathcal{S}|}, a) + \gamma v^\pi(s')\right). \tag{198}$$

Notice that this is a system of linear equations for which we know there is a unique solution (the value function—we know this is unique because these equations were derived from the definition of the value function in (91), which is clearly unique). This system can be solved in $O(|\mathcal{S}|^3)$ operations (in general this problem requires $\Omega(|\mathcal{S}|^2)$ operations, and in Fall 2018 the algorithm with the best asymptotic runtime is that of Coppersmith and Winograd (1987), which requires $O(n^{2.736})$ operations.).

An alternative approach is to use dynamic programming. Although not necessarily more efficient, this dynamic programming approach will later allow us to efficiently interleave steps of evaluating the current policy and then improving the current policy. Here, when we talk about *evaluating a policy* or *policy evaluation*, we refer to estimating the state-value or action-value function associated with the policy. Later we will discuss a different form of *policy evaluation* wherein the goal is to estimate $J(\pi)$, not the entire value function, v^π.

Let $v_0, v_1, v_2, \dots$ denote a sequence of functions where each $v_i : \mathcal{S} \to \mathbb{R}$. Note that some people like to include hat symbols on estimators. In the notes we will continue by writing v_i to denote the i^{th} estimator of v^π, but you are free in your assignments to write $\hat{v}_i$ (as we are doing in lecture) in place of v_i. Intuitively, this sequence of functions represents the sequence of estimates of v^π produced by an algorithm estimating v^π in an incremental manner. This has been a point of confusion for students in the past, who did not recognize that v_i and v^π are different. So, to say it again, notice that v_i is our i^{th} approximation of v^π. It is *not* necessarily v^π. Also, although we typically "know" v_i (we have an analytic form for it, or code to evaluate it), we often do not precisely know v^π (if we did, we would have no need to approximate it!).

Consider the setting where v_0 is chosen arbitrarily. One way to improve our estimate is to try to make the two sides of the Bellman equation equal. Recall that the Bellman equation for v^π is:

$$v^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s') \left(R(s,a) + \gamma v^\pi(s')\right). \tag{199}$$

We can therefore try to make the two sides of the Bellman equation equal with the following update:

$$v_{i+1}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v_i(s')\big). \tag{200}$$

Applying this update to compute $v_{i+1}(s)$ for ever state s given v_i, is called a *full backup*. Applying the update to a compute $v_{i+1}(s)$ for a single state, s, is called a *backup*.

To see why this is a dynamic programming approach, consider what this algorithm does if we stored $v_0, v_1, \dots$ as a matrix with v_0 as the first column, v_1 and the second column, etc. This update rule fills this matrix from the left to the right, where the values for entries depend on previously computed values for the previous column.

From our derivation of the Bellman equation, it should be clear that v^π is a fixed point of this iterative procedure (that is, if $v_i = v^\pi$, then $v_{i+1} = v^\pi$ as well). Less obviously, as $i \to \infty$, $v_i \to v^\pi$. We will not prove this property (this update is a stepping stone to the *value iteration* update that we present later, and we will focus on proving the convergence of the value iteration update).

Question 22. *Consider a 4×4 gridworld where the agent starts in the top left, the bottom right state is terminal, rewards are always -1, $\gamma = 1$, and state transitions are deterministic. Consider the policy that always chooses the action to move down, except when it is on the bottom row, at which point it chooses the action to move right. Starting with $v_0(s) = 0$ for all s, compute $v_1, v_2, \dots, v_7$. For this problem, assume that the reward for transitioning from the bottom right state to s_∞ is also -1 (also think about how this could be implemented in our notation where the reward for entering s_∞ is always zero—can an MDP of this sort not exist in our notation?)*

Answer 22. $v_0 = \begin{array}{|c|c|c|c|}\hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline\end{array}$ $v_1 = \begin{array}{|c|c|c|c|}\hline -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 \\ \hline\end{array}$ $v_2 = \begin{array}{|c|c|c|c|}\hline -2 & -2 & -2 & -2 \\ \hline -2 & -2 & -2 & -2 \\ \hline -2 & -2 & -2 & -2 \\ \hline -2 & -2 & -2 & -1 \\ \hline\end{array}$ $v_3 = \begin{array}{|c|c|c|c|}\hline -3 & -3 & -3 & -3 \\ \hline -3 & -3 & -3 & -3 \\ \hline -3 & -3 & -3 & -2 \\ \hline -3 & -3 & -2 & -1 \\ \hline\end{array}$

$v_4 = \begin{array}{|c|c|c|c|}\hline -4 & -4 & -4 & -4 \\ \hline -4 & -4 & -4 & -3 \\ \hline -4 & -4 & -3 & -2 \\ \hline -4 & -3 & -2 & -1 \\ \hline\end{array}$ $v_5 = \begin{array}{|c|c|c|c|}\hline -5 & -5 & -5 & -4 \\ \hline -5 & -5 & -4 & -3 \\ \hline -5 & -4 & -3 & -2 \\ \hline -4 & -3 & -2 & -1 \\ \hline\end{array}$ $v_6 = \begin{array}{|c|c|c|c|}\hline -6 & -6 & -5 & -4 \\ \hline -6 & -5 & -4 & -3 \\ \hline -5 & -4 & -3 & -2 \\ \hline -4 & -3 & -2 & -1 \\ \hline\end{array}$ $v_7 = \begin{array}{|c|c|c|c|}\hline -7 & -6 & -5 & -4 \\ \hline -6 & -5 & -4 & -3 \\ \hline -5 & -4 & -3 & -2 \\ \hline -4 & -3 & -2 & -1 \\ \hline\end{array}$

Allowing rewards when transitioning to s_∞ from other states is ok. We can handle this by introducing a new state that occurs just before s_∞. The reward can be non-zero when transitioning into this new state, and this new state always transitions to s_∞ with a reward of zero.

Notice that in Question 22 information appears to flow backwards across state transitions. Also notice that in this example the process has reached its fixed point after only seven iterations. In general, this policy evaluation algorithm is only guaranteed to converge in the limit, and so practical implementations might halt the process when all changes to the current state-value approximation are smaller than some predefined constant value.

The dynamic programming algorithm for policy evaluation that we have described thus far can be implemented by storing $2|\mathcal{S}|$ values—by storing v_i only until v_{i+1} has been computed. When a new estimate of $v^\pi(s)$ has been computed, it is placed in $v_{i+1}(s)$ in order to not overwrite the value stored in $v_i(s)$, which might be used when computing the next values for other states. An alternative *in-place* implementation keeps only a single table, performs individual state backups (rather than full backups) and stores updated state-value approximations directly in the same table from which they were computed. This variant has also been shown to converge, even if states are updated in any order or if some states are updated more frequently than others (as long as every state is updated infinitely often). Notice that in these in-place variants, the order that states are updated in matters. In Question 22, updating states from the bottom left to the top right can result in a single sweep of the state space being sufficient for the algorithm to reach its fixed point, while updating states from the top left to bottom right will take many sweeps before convergence.

6.2 Policy Improvement

Once we have estimated v^π or q^π for some initial policy, π, how can we find a new policy that is at least as good as π? Notice that if we have v^π, we can easily compute $q^\pi(s,a)$ for any (s,a) by the equation $q^\pi(s,a) = \sum_{s' \in \mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v^\pi(s')\big)$. So, for now consider how we could find a policy π' that is always at least as good as π if we have already computed q^π.

Consider a greedy approach, where we define π' to be a deterministic policy that selects the action that maximizes $q^\pi(s,\cdot)$ when in state s. That is, $\pi' : \mathcal{S} \to \mathcal{A}$ is a deterministic policy defined such that

$$\pi'(s) \in \arg\max_{a \in \mathcal{A}} q^\pi(s,a). \tag{201}$$

This policy is the *greedy policy with respect to* q^π. It is greedy because it optimizes for the immediate future without considering long-term ramifications. Recall that $q^\pi(s,a)$ is the expected discounted return if the agent takes action a in state s and follows the policy π thereafter. So, when π' chooses actions, a, that maximize $q^\pi(s,a)$, it *not* necessarily choosing actions that cause $q^{\pi'}(s,a)$ or $v^{\pi'}(s)$ to be maximized. It is choosing the actions that maximize the expected discounted return if the action is chosen at the current step, and then afterwards the policy π (not π'!) is used. Can this greedy update to the policy, which only considers using π' to take a single action, cause π' to be worse than π?

Perhaps surprisingly, the greedy policy with respect to q^π is always at least as good as π, i.e., $\pi' \geq \pi$ (recall the definition of $\geq$ for policies, presented in (163)). This result is described by the *policy improvement theorem*, Theorem 3.

Theorem 3 (Policy Improvement Theorem). *For any policy π, if π' is a deterministic policy such that $\forall s \in \mathcal{S}$,*

$$q^\pi(s, \pi'(s)) \geq v^\pi(s), \tag{202}$$

then $\pi' \geq \pi$.

Proof.

$$\begin{aligned}
v^\pi(s) \leq & q^\pi(s,\pi'(s)) && (203)\\
=&\mathbf{E}\left[R_t + \gamma v^\pi(S_{t+1}) | S_t = s, \pi'\right] && (204)\\
\leq&\mathbf{E}\left[R_t + \gamma q^\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s, \pi'\right] && (205)\\
=&\mathbf{E}\left[R_t + \gamma \mathbf{E}\left[R_{t+1} + \gamma v^\pi(S_{t+2}) | S_t = s, \pi'\right] | S_t = s, \pi'\right] && (206)\\
=&\mathbf{E}\left[R_t + \gamma R_{t+1} + \gamma^2 v^\pi(S_{t+2}) \middle| S_t = s, \pi'\right] && (207)\\
\leq&\mathbf{E}\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 v^\pi(S_{t+3}) \middle| S_t = s, \pi'\right] && (208)\\
&\ldots && (209)\\
\leq&\mathbf{E}\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^4 R_{t+3} + \cdots \middle| S_t = s, \pi'\right] && (210)\\
=&v^{\pi'}(s), && (211)
\end{aligned}$$

where each step follows from the previous using mixtures of the definitions of value functions and the assumption within the theorem statement. Notice that (204) conditions on π', not π. This is because the only action that this conditioning impacts is A_t—all subsequent actions are captured by the $v^\pi(S_{t+1})$ term, which uses the policy π. Notice also that the inner expectation in (206) does not condition on S_{t+1} taking a particular value, which one might expect given that it is an expansion of $q^\pi(S_{t+1}, \pi'(S_{t+1}))$. This is because the state, S_{t+1}, that $q^\pi(S_{t+1}, \pi'(S_{t+1}))$ takes as its first argument is a random variable. So, the condition that one might expect in (206) when expanding $q^\pi(S_{t+1}, \pi'(S_{t+1}))$ is that $S_{t+1} = S_{t+1}$. This condition is a tautology, and so it can be ignored. ☐

The policy improvement theorem also holds for stochastic greedy policies, as described in Theorem 4, for which we do not provide a proof.

Theorem 4 (Policy Improvement Theorem for Stochastic Policies)**.** *For any policy π, if π' satisfies*

$$\sum_{a \in \mathcal{A}} \pi'(s,a) q^\pi(s,a) \geq v^\pi(s), \tag{212}$$

for all $s \in \mathcal{S}$, then $\pi' \geq \pi$.

——————————End of Lecture 11, October 8, 2019——————————

We now have the components to create our first planning algorithm, *policy iteration*. The policy iteration algorithm interleaves policy evaluation steps using the dynamic programming approach and policy improvement steps. This process is depicted in Figure 14, and presented using pseudocode in Algorithm 8.

$$\pi_0 \xrightarrow{\text{policy evaluation}} v^{\pi_0} \xrightarrow{\text{policy improvement}} \pi_1 \xrightarrow{\text{policy evaluation}} v^{\pi_1} \xrightarrow{\text{policy improvement}} \pi_2 \xrightarrow{\text{policy evaluation}} v^{\pi_2} \xrightarrow{\text{policy improvement}}$$

Figure 14: Diagram of the policy iteration algorithm. It begins with an arbitrary policy, π_0, and evaluates it using the dynamic programming approach described previously to produce v^{π_0}. It then performs greedy policy improvement to select a new deterministic policy, π_1, that is at least as good as π_0. It then repeats this process, evaluating π_1, and using the resulting value function to obtain a new policy, π_2, etc.

Algorithm 8: Policy iteration. This pseudocode assumes that policies are deterministic.

Initialize π_0 arbitrarily;
for $i = 0$ **to** ∞ **do**
    `/* Policy Evaluation */`
    Initialize v_0 arbitrarily;
    for $k = 0$ **to** ∞ **do**
        For all $s \in \mathcal{S}$:

$$v_{k+1}(s) = \sum_{s' \in \mathcal{S}} P(s, \pi_i(s), s')\left(R(s, \pi_i(s)) + \gamma v_k(s')\right) \tag{213}$$

        if $v_{k+1} = v_k$ **then**
            $v^{\pi_i} = v_k$;
            break;
    `/* Check for Termination */`
    if $\forall s \in \mathcal{S},\ \pi_i(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s')\Big(R(s, a) + \gamma v^{\pi_i}(s')\Big)$ **then**
        terminate;
    `/* Policy Improvement */`
    Compute π_{i+1} such that for all s,

$$\pi_{i+1}(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s')\Big(R(s, a) + \gamma v^{\pi_i}(s')\Big), \tag{214}$$

    with ties broken by selecting actions according to some strict total order on $\mathcal{A}$;

Notice that the number of deterministic policies for a finite MDP is finite. So, either policy iteration terminates after a finite number of iterations (if rewards are bounded and $\gamma < 1$) or some policy must occur at least twice (there is a cycle in the sequence of policies). We now show that there cannot be a cycle of policies, so we can conclude that policy iteration terminates after a finite number of iterations.

Theorem 5. *It cannot occur that $\pi_j = \pi_k$ for $j \neq k$ when using the policy iteration algorithm.*

Proof. This was not covered in lecture, and you are not responsible for it. We provide it for completeness.

We assume without loss of generality that $j < k$. We have from the policy improvement theorem that $\pi_j \leq \pi_{j+1} \leq \cdots \leq \pi_k$. Since $\pi_j = \pi_k$, and thus $v^{\pi_j} = v^{\pi_k}$, we therefore have that $v^{\pi_j} = v^{\pi_{j+1}} = v^{\pi_k}$. So (recall that the policies are deterministic policies):

$$
\begin{aligned}
v^{\pi_j}(s) =& v^{\pi_{j+1}}(s) && (215)\\
\overset{\textbf{(a)}}{=}& R(s, \pi_{j+1}(s)) + \sum_{s \in \mathcal{S}} P(s, \pi_{j+1}(s), s')\gamma v^{\pi_{j+1}}(s') && (216)\\
\overset{\textbf{(b)}}{=}& R(s, \pi_{j+1}(s)) + \sum_{s \in \mathcal{S}} P(s, \pi_{j+1}(s), s')\gamma v^{\pi_j}(s') && (217)\\
\overset{\textbf{(c)}}{=}& \max_{a \in \mathcal{A}} R(s, a) + \sum_{s \in \mathcal{S}} P(s, a, s')\gamma v^{\pi_j}(s'), && (218)
\end{aligned}
$$

where **(a)** comes from the Bellman equation, **(b)** holds because $v^{\pi_j} = v^{\pi_{j+1}}$, and **(c)** holds by the definition of π_{j+1}. Furthermore, by the Bellman equation for v^{π_j} we have that:

$$
v^{\pi_j}(s) = R(s, \pi_j(s)) + \sum_{s \in \mathcal{S}} P(s, \pi_j(s), s')\gamma v^{\pi_j}(s'). \tag{219}
$$

For (219) and (218) to hold simultaneously, we have that

$$
\max_{a \in \mathcal{A}} R(s, a) + \sum_{s \in \mathcal{S}} P(s, a, s')\gamma v^{\pi_j}(s') = R(s, \pi_j(s)) + \sum_{s \in \mathcal{S}} P(s, \pi_j(s), s')\gamma v^{\pi_j}(s'), \tag{220}
$$

and hence that

$$
\pi_j(s) \in \arg\max_{a \in \mathcal{A}} R(s, a) + \sum_{s \in \mathcal{S}} P(s, a, s')\gamma v^{\pi_j}(s'). \tag{221}
$$

However, this means that the termination condition for policy iteration would have been satisfied with π_j. $\square$

Now that we know policy iteration terminates, consider what we know about the policy that it converges to. When this it terminates, we have that for all

$s \in \mathcal{S}$,

$$v^{\pi_{i+1}}(s) \overset{\text{(a)}}{=} \sum_{a \in \mathcal{A}} \pi_{i+1}(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^{\pi_{i+1}}(s')) \tag{222}$$

$$\overset{\text{(b)}}{=} \sum_{a \in \mathcal{A}} \pi_{i+1}(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^{\pi_i}(s')) \tag{223}$$

$$\overset{\text{(c)}}{=} \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^{\pi_i}(s')) \tag{224}$$

$$\overset{\text{(b)}}{=} \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^{\pi_{i+1}}(s')), \tag{225}$$

where **(a)** is the Bellman equation (and where we view π_{i+1} as a distribution rather than a mapping from states to actions), **(b)** both follow from the starting assumption that the process has terminated, and so $\pi_{i+1} = \pi_i$, and **(c)** comes from the fact that π_{i+1} is greedy with respect to v^{π_i}. Since (225) is the Bellman optimality equation, π_{i+1} is an optimal policy—when policy iteration stops, the policy is optimal.

Notice also that the policy evaluation algorithm could terminate when $v^{\pi_{i+1}} = v^{\pi_i}$. Using this termination condition would not require π_{i+1} to break ties in any particular order and is equivalent, but makes the analysis of the final policy less straightforward.

6.3 Value Iteration

Notice that the policy iteration algorithm is not efficient. Even though policy evaluation using dynamic programming is guaranteed to converge to v^π, it is not guaranteed to *reach* v^π, except in the limit as the number of iterations of policy evaluation goes to infinity. Thus, each iteration of the outer loop in policy iteration (the loop over i), may require an infinite amount of computation. An obvious question is whether or not the policy evaluation algorithm can be stopped early—when $v_{k+1} \neq v_k$, but perhaps after some fixed number of iterations (i.e., the loop over k goes from 0 to K, for some constant K).

If policy evaluation is stopped early, then the estimate of v^{π_i} will have error, and so the policy that is greedy with respect to the estimate of v^{π_i} may not be an improvement over the current policy. However, perhaps surprisingly, this process *does* still converge to an optimal policy. A particularly popular variant of this algorithm is *value iteration*, which uses $K = 1$—it performs a *single* iteration of policy evaluation between policy improvement steps. Importantly, each iteration of policy evaluation begins with the value function estimate used in the previous step (rather than a random initial value function. Pseudocode for value iteration is presented in Algorithm 9.

Algorithm 9: Value Iteration. This pseudocode is an inefficient implementation that we use as a stepping-stone to the common pseudocode.

Initialize π_0 and v_0 arbitrarily;
for $i = 0$ **to** ∞ **do**
    `/* Policy Evaluation */`
    For all $s \in \mathcal{S}$:

$$v_{i+1}(s) = \sum_{s' \in \mathcal{S}} P(s, \pi_i(s), s') \left(R(s, \pi_i(s)) + \gamma v_i(s') \right) \tag{226}$$

    `/* Check for Termination */`
    if $v_{i+1} = v_i$ **then**
        terminate;
    `/* Policy Improvement */`
    Compute π_{i+1} such that for all s,

$$\pi_{i+1}(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \Big(R(s, a) + \gamma v_{i+1}(s') \Big). \tag{227}$$

If we were to following the pseudocode in Algorithm 9, we would obtain the following sequence of policies and value functions:

$$v_0 : \text{arbitrary} \tag{228}$$

$$\pi_0 : \text{arbitrary} \tag{229}$$

$$v_1 : \forall s, v_1(s) = \sum_{s' \in \mathcal{S}} P(s, \pi_0(s), s') \big(R(s, \pi_0(s)) + \gamma v_0(s') \big) \tag{230}$$

$$\pi_1 : \forall s, \pi_1(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_1(s') \big) \tag{231}$$

$$v_2 : \forall s, v_2(s) = \sum_{s' \in \mathcal{S}} P(s, \pi_1(s), s') \big(R(s, \pi_1(s)) + \gamma v_1(s') \big) \tag{232}$$

$$\pi_2 : \forall s, \pi_2(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_2(s') \big) \tag{233}$$

$$v_3 : \forall s, v_3(s) = \sum_{s' \in \mathcal{S}} P(s, \pi_2(s), s') \big(R(s, \pi_2(s)) + \gamma v_2(s') \big) \tag{234}$$

$$\pi_3 : \forall s, \pi_3(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_3(s') \big) \tag{235}$$

$$\dots \tag{236}$$

Notice the similarity between the updates to the policy and the value function estimate. When computing $v_2(s)$ we use $\pi_1(s)$, which is the action, a, that maximizes the expression on the right side of (231). This expression is the same expression as that in the right side of (232). Thus, the expression for $v_2(s)$ can be written as:

$$v_2(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s')\big(R(s, a) + \gamma v_1(s')\big). \tag{237}$$

This same trend holds for v_3 and v_2. In general, we can compute v_{i+1} directly from v_i without explicitly computing π_i. This results in the more efficient form for the value iteration algorithm:

$$v_{i+1}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s')\big(R(s, a) + \gamma v_i(s')\big). \tag{238}$$

Notice that, while the policy evaluation algorithm is an iterative form for the Bellman equation, the value iteration update in (238) is an iterative form for the Bellman optimality equation. Pseudocode for the value iteration algorithm using this more efficient update is presented in Algorithm 10.

Algorithm 10: Value Iteration.

```
1 Initialize v_0 arbitrarily;
2 for i = 0 to ∞ do
      /* Policy Evaluation                                   */
3     For all s ∈ S:
```

$$v_{i+1}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s')\big(R(s, a) + \gamma v_i(s')\big). \tag{239}$$

```
      /* Check for Termination                               */
4     if v_{i+1} = v_i then
5         terminate;
```

6.4 The Bellman Operator and Convergence of Value Iteration

In this subsection we prove that value iteration converges to a single unique value function. We then argue that this result implies all of the claims that we previously stated we would prove later: a deterministic optimal policy exists for all finite MDPs with bounded rewards and $\gamma < 1$, and the Bellman optimality equation only holds for v^{π^*}, where π^* is an optimal policy.

Before we present the main theorem of this subsection, we will establish additional notation. First notice that, for finite MDPs, we can view value function estimates as vectors in $\mathbb{R}^{|\mathcal{S}|}$, where each element in the vector corresponds to the value of one state. Also, recall that an *operator* is a function that takes elements of a space as input and produces elements of the same space as output. Let $\mathcal{T} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ be an operator that we call the *Bellman operator*, which takes value function estimates as input and produces as output new value function estimates, and such that

$$\mathcal{T}(v_i) \coloneqq v_{i+1}, \tag{240}$$

where the sequence of value function approximations, $v_0, v_1, \ldots$, is as defined by (238). That is,

$$\mathcal{T}(v_i) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s')\big(R(s, a) + \gamma v_i(s')\big). \tag{241}$$

That is, the Bellman operator is the operator that encodes a single iteration of value iteration. We will abuse notation and omit parenthesis, writing $\mathcal{T} v_i = v_{i+1}$, and further assume that the order of operations prioritizes evaluation of the Bellman operator over evaluation of the value function approximation, so that $\mathcal{T} v(s)$ denotes $\mathcal{T}(v)$ evaluated at s.

An operator is a *contraction mapping* if there exists a $\lambda \in [0, 1)$ such that $\forall x \in \mathcal{X}, \forall y \in \mathcal{Y}$, $d(f(x), f(y)) \leq \lambda d(x, y)$, where d is a distance function. Figure 15 presents a diagram that may assist in understanding the definition of a contraction mapping.

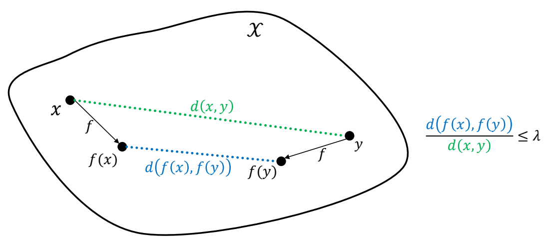


Figure 15: Diagram to assist with interpreting the definition of a contraction mapping. Here x and y denote two points in the space $\mathcal{X}$. The function, f, maps x to $f(x)$, and y to $f(y)$. If f is a contraction mapping, then for every possible x and y, the distance between x and y (the length of the green dotted line) must be greater than the distance between $f(x)$ and $f(y)$ (the length of the dotted blue line). Moreover, the ratio of these distances must be at most λ. For example, if $\lambda = 0.5$, then every application of f to two points, x and y must at least halve the distance between x and y.

Question 23. *If f is a contraction mapping, then is the sequence $x_{i+1} = f(x_i)$ guaranteed to converge? Is it guaranteed to converge to a unique point within $\mathcal{X}$?*

Answer 23. *Under certain conditions, this sequence will converge to a unique fixed-point within $\mathcal{X}$. This should be intuitively clear, since if the process were to be started at any two points in $\mathcal{X}$, the distance between the i^{th} point in each sequence will decrease at a rate of λ during each iteration. Furthermore, the fixed point must be unique, since otherwise defining x to be one fixed point and y to be the other fixed point would result in $d(f(x), f(y)) = d(x, y) \not\leq \lambda d(x, y)$. This intuition is captured by the Banach fixed point theorem, presented below.*

As described in the answer to the above question, if f is a contraction mapping then it is guaranteed to converge to a unique fixed point. This intuition is formalized by the Banach fixed-point theorem:

Theorem 6 (Banach Fixed-Point Theorem)**.** *If f is a contraction mapping on a non-empty complete normed vector space, then f has a unique fixed point, x^*, and the sequence defined by $x_{k+1} = f(x_k)$, with x_0 chosen arbitrarily, converges to x^*.*

Proof. We do not provide a proof in this course. A proof can be found on Wikipedia. □

We will apply the Banach fixed-point theorem where $f \leftarrow \mathcal{T}$, $x \in \mathbb{R}^{|\mathcal{S}|}$, and $d(v, v') \coloneqq \max_{s \in \mathcal{S}} |v(s) - v'(s)|$. That is, we will consider the *max norm*, $\|v - v'\|_\infty = \max_{s \in \mathcal{S}} |v(s) - v'(s)|$. Recall that the max norm is the p-norm, with $p = \infty$. In order to apply the Banach fixed-point theorem, first notice that $\mathbb{R}^{|\mathcal{S}|}$ is complete under the max-norm.[5] We must also show that the Bellman operator is a contraction mapping—we show this in Theorem 7.

——————End of Lecture 12, October 10, 2019——————

[5]This follows from the Riesz-Fisher theorem, which implies that L^p space is complete for $1 \leq p \leq \infty$.

Name:

CMPSCI 687 Pop Quiz 3
October 17, 2019

Instructions: You have 5 minutes to complete this quiz. This quiz is **closed** notes—do not use your notes or a laptop. Do not discuss problems with your neighbors until after everyone has handed in their quiz.

Use the notation from class. Don't forget to capitalize your random variables.

1. The definition of the state-value function is:

   $v^\pi(s) \coloneqq \mathbf{E}[G_t|S_t = s, \pi]$

2. The definition of the action-value function is:

   $q^\pi(s,a) \coloneqq \mathbf{E}[G_t|S_t = s, A_t = a, \pi]$

3. The Bellman equation states that for all s:

   $v^\pi(s) = \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v^\pi(s')\big)$

4. The Bellman optimality equation states that for all s:

   $v^*(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v^*(s')\big)$

5. (True or False) Although there may be multiple optimal policies, there is only one optimal value function for a finite MDP with bounded rewards.

Theorem 7 (The Bellman operator is a contraction mapping)**.** *The Bellman operator is a contraction mapping on* $\mathbb{R}^{|\mathcal{S}|}$ *with* $d(v, v') \coloneqq \max_{s\in\mathcal{S}} |v(s) - v'(s)|$.

Proof.

$$\|Tv - Tv'\| = \max_{s\in\mathcal{S}} |Tv(s) - Tv'(s)| \tag{242}$$

$$= \max_{s\in\mathcal{S}} \left| \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v(s')\big) - \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v'(s')\big) \right|, \tag{243}$$

by the definition of the Bellman operator. To continue, we derive a relevant property of arbitrary functions, $f : \mathcal{X} \to \mathbb{R}$ and $g : \mathcal{S} \to \mathbb{R}$, for arbitrary sets, $\mathcal{X}$. We begin with a simple expression and then list inequalities implied by the preceding inequalities to obtain the desired expression:

$$\forall x, f(x) - g(x) \leq |f(x) - g(x)| \tag{244}$$

$$\forall x, f(x) \leq |f(x) - g(x)| + g(x) \tag{245}$$

$$\max_{x\in\mathcal{X}} f(x) \leq \max_{x\in\mathcal{X}} |f(x) - g(x)| + g(x) \tag{246}$$

$$\max_{x\in\mathcal{X}} f(x) \leq \max_{x\in\mathcal{X}} |f(x) - g(x)| + \max_{x\in\mathcal{X}} g(x) \tag{247}$$

$$\max_{x\in\mathcal{X}} f(x) - \max_{x\in\mathcal{X}} g(x) \leq \max_{x\in\mathcal{X}} |f(x) - g(x)|. \tag{248}$$

If $\max_{x\in\mathcal{X}} f(x) - \max_{x\in\mathcal{X}} g(x) \geq 0$, then it follows from (248) that

$$\left| \max_{x\in\mathcal{X}} f(x) - \max_{x\in\mathcal{X}} g(x) \right| \leq \max_{x\in\mathcal{X}} |f(x) - g(x)|. \tag{249}$$

If $\max_{x\in\mathcal{X}} f(x) - \max_{x\in\mathcal{X}} g(x) < 0$, then we have from (248) that:

$$\max_{x\in\mathcal{X}} g(x) - \max_{x\in\mathcal{X}} f(x) \leq \max_{x\in\mathcal{X}} |g(x) - f(x)|, \tag{250}$$

which also implies (249), since $\max_{x\in\mathcal{X}} f(x) - \max_{x\in\mathcal{X}} g(x) \geq 0$ and $|f(x) - g(x)| = |g(x) - f(x)|$. Applying (249) to (243), we obtain:

$$\|Tv - Tv'\| \leq \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \left| \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v(s')\big) - \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v'(s')\big) \right| \tag{251}$$

$$= \gamma \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \left| \sum_{s'\in\mathcal{S}} P(s,a,s')\big(v(s') - v'(s')\big) \right| \tag{252}$$

$$\leq \gamma \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s') \left| \big(v(s') - v'(s')\big) \right| \tag{253}$$

$$\leq \gamma \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \max_{s'\in\mathcal{S}} \left| \big(v(s') - v'(s')\big) \right| \tag{254}$$

$$= \gamma \max_{s'\in\mathcal{S}} |v(s') - v'(s')| \tag{255}$$

$$= \gamma \|v - v'\|. \tag{256}$$

□

Thus, we have that the Bellman operator is a contraction mapping, and so by the Banach fixed point theorem it follows that the value iteration algorithm converges to a unique fixed point, which we denote here by v^∞.

Theorem 8. *Value iteration converges to a unique fixed point v^∞ for all MDPs with finite state and action sets, bounded rewards, and $\gamma < 1$.*

Proof. This follows from the Banach fixed point theorem (Theorem 6) and the fact that the Bellman operator (which encodes the value iteration update) is a contraction (Theorem 7). □

Although we do not provide a proof, policy iteration and value iteration both converge in a number of iterations that is polynomial in $|\mathcal{S}|$ and $|\mathcal{A}|$. Notice also that the Bellman operator is a contract with parameter γ—as γ approaches one the speed of convergence slows, while small values for γ speed up convergence. This is intuitive because small γ mean that events that occur in the distant future are of little importance, and so the value function will become accurate after fewer backups.

We can now establish that the existence of deterministic optimal policies:

Theorem 9. *All MDPs with finite state and action sets, bounded rewards, and $\gamma < 1$ have at least one optimal policy.*

Proof. By Theorem 8 we have that value iteration converges to a unique fixed point v^∞. Consider any deterministic policy π^∞ satisfying:

$$\pi^\infty(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^\infty(s)). \tag{257}$$

At least one such policy exists, since $\mathcal{A}$ is a finite set. Recall that value iteration corresponds to one iteration of policy iteration, but where policy evaluation only conducts a single full backup. This π^∞ is the greedy policy with respect to v^∞. Since v^∞ is a fixed point of value iteration, performing one full backup of policy evaluation for π^∞ results in v^∞ again. This means that v^∞ is a fixed-point of policy evaluation for π^∞. That is:

$$v^\infty(s) = \sum_{s' \in \mathcal{S}} P(s, \pi^\infty(s), s')(R(s, \pi^\infty(s)) + \gamma v^\infty(s')). \tag{258}$$

As this is the Bellman equation, we have that v^∞ is the state-value function for π^∞. Next, since v^∞ is a fixed point of the value iteration algorithm, we have that for all $s \in \mathcal{S}$:

$$v^\infty(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^\infty(s')), \tag{259}$$

which is the Bellman optimality equation. Since v^∞ is the value function for π^∞, we therefore have that π^∞ satisfies the Bellman optimality equation. We showed in Theorem 1 that any policy satisfying the Bellman optimality equation is an optimal policy, and so we have that π^∞ is an optimal policy. □

7 Monte Carlo Methods

Monte Carlo algorithms, which have a history worth reading about, use randomness to solve problems that are deterministic in principle. A classical example is the estimation of π. Consider the unit circle, drawn with its center at the bottom left corner of a unit square. The percent of the area inside the square that is also inside the circle is $\pi/4$. Hence, once can estimate π by throwing darts at the unit square (such that the darts land with a uniform distribution over the square). An estimate of π is then given by:

$$\pi \approx 4 \frac{\text{number of darts inside the unit square and the circle}}{\text{number of darts inside the unit square}}. \tag{260}$$

In this example, the random dart throws are used to provide an (unbiased) estimate of the deterministic value, π.

7.1 Monte Carlo Policy Evaluation

Consider using a Monte Carlo approach to estimate the state-value function for a policy π. In this case, a "dart throw" corresponds to sampling history (running an episode) using the policy, π. More specifically, consider the task of estimating the value of a state, $s \in \mathcal{S}$. If we generate a history, $H = (S_0, A_0, R_0, S_1, A_1, R_1, \dots)$, starting from $S_0 = s$, how can we construct an unbiased estimator of $v^\pi(s)$?

Question 24. *Which of the following three estimators of $v^\pi(s)$ is an unbiased estimator?*

1. $\sum_{k=0}^{\infty} \gamma^k R_k$.
2. $\sum_{k=0}^{\infty} \gamma^k R_{t_{last}+k}$, *where t_{last} is the last time step where the state was s.*
3. *Other estimators that average the returns from each occurrence of s within the history.*

———————End of Lecture 13, October 17, 2019———————

CMPSCI 687 Homework 3

Due October 29, 2019, 11:55pm Eastern Time

Instructions: Collaboration is not allowed on any part of this assignment. Submissions must be typed (hand written and scanned submissions will not be accepted). You must use LaTeX. The assignment should be submitted as two documents: a .pdf with your written answers and a single .cpp file as described in the programming portion.

Part One: Written (50 Points Total)

1. (2 Points) One day while working in the engineering department of the Starship Enterprise, your friend Geordi comes to you with an idea. He points out that the warp core (engine) uses a reinforcement learning algorithm to regulate its temperature. He hypothesizes the the value function that it uses would be easier to represent and/or faster to approximate in two distinct parts: one that estimates the value of a state given that the next state is safe (within desirable thresholds), and another that estimates the value of a state given that the next state is not safe. Working with Geordi, who of course uses the notation from this class, you decide to define $\mathcal{X}$ to be the set of safe states, and $\mathcal{X}^\complement$ to be the set of unsafe states, i.e., $\mathcal{X}^\complement = \mathcal{S} \setminus \mathcal{X}$.[6] In order to continue, you and Geordi decide to establish some notation. Specifically, you want to define $v^\pi_\mathcal{Y}(s)$ to be the expected discounted return given that the agent begins in state s, follows policy π, and the next state (but not necessarily the states after the next state) happens to be in $\mathcal{Y}$. Give a mathematical definition for $v^\pi_\mathcal{Y}$ like our definition for v^π:
$$v^\pi_\mathcal{Y}(s) \coloneqq \mathbf{E}\left[G_t | S_t = s, S_{t+1} \in \mathcal{Y}, \pi\right]. \tag{261}$$

2. (5 Points) Having defined $v^\pi_\mathcal{Y}$, you decide to relate your new value functions, $v^\pi_\mathcal{X}$ and $v^\pi_{\mathcal{X}^\complement}$, to the standard value function, v^π. Derive an expression for $v^\pi(s)$ that *only* uses the following terms: $\pi, \mathcal{X}^\complement, P, \mathcal{A}, \mathcal{S}, \mathcal{X}, v^\pi_\mathcal{X}$ and $v^\pi_{\mathcal{X}^\complement}$. Note: You may introduce variables when summing over sets, e.g., x in $\sum_{x\in\mathcal{X}}$. Your final answer should not include expectations or any random variables like S_t or R_t. You should begin with the definition of $v^\pi(s)$ and end with an expression that only contains the allowed terms. Show your work (show the steps, don't just jump to your final answer). You may want to derive some properties before proceeding with the derivation for $v^\pi(s)$—that is allowed.

[6] In latex, here we are using the symbols \complement and \setminus for $\complement$ and $\setminus$ respectively.

First notice that for any set $\mathcal{Y} \subseteq \mathcal{S}$:

$$\Pr(S_{t+1} \in \mathcal{Y}|S_t = s, \pi) = \sum_{s' \in \mathcal{Y}} \Pr(S_{t+1} = s'|S_t = s, \pi) \Pr(S_{t+1} \in \mathcal{Y}|S_t = s, S_{t+1} = s', \pi) \tag{262}$$

$$= \sum_{s' \in \mathcal{Y}} \Pr(S_{t+1} = s'|S_t = s, \pi) \tag{263}$$

$$= \sum_{s' \in \mathcal{Y}} \sum_{a \in \mathcal{A}} \Pr(A_t = a|S_t = s, \pi) \Pr(S_{t+1} = s'|S_t = s, A_t = a, \pi) \tag{264}$$

$$= \sum_{s' \in \mathcal{Y}} \sum_{a \in \mathcal{A}} \pi(s, a) P(s, a, s'). \tag{265}$$

We now answer the question:

$$v^\pi(s) = \mathbf{E}\left[G_t|S_t = s, \pi\right] \tag{266}$$

$$= \Pr(S_{t+1} \in \mathcal{X}|S_t = s, \pi)\mathbf{E}\left[G_t|S_t = s, S_{t+1} \in \mathcal{X}, \pi\right] \tag{267}$$

$$+ \Pr(S_{t+1} \in \mathcal{X}^\complement|S_t = s, \pi)\mathbf{E}\left[G_t\middle|S_t = s, S_{t+1} \in \mathcal{X}^\complement, \pi\right] \tag{268}$$

$$= \Pr(S_{t+1} \in \mathcal{X}|S_t = s, \pi)v^\pi_{\mathcal{X}}(s) + \Pr(S_{t+1} \in \mathcal{X}^\complement|S_t = s, \pi)v^\pi_{\mathcal{X}^\complement}(s) \tag{269}$$

$$= \sum_{s' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(s, a)P(s, a, s')v^\pi_{\mathcal{X}}(s) + \sum_{s' \in \mathcal{X}^\complement} \sum_{a \in \mathcal{A}} \pi(s, a)P(s, a, s')v^\pi_{\mathcal{X}^\complement}(s), \tag{270}$$

where the final line follows from the property we derived first. Note that the terms $v^\pi_{\mathcal{X}}(s)$ and $v^\pi_{\mathcal{X}^\complement}(s)$ could be moved before the summations:

$$v^\pi(s) = v^\pi_{\mathcal{X}}(s) \sum_{s' \in \mathcal{X}} \sum_{a \in \mathcal{A}} \pi(s, a)P(s, a, s') + v^\pi_{\mathcal{X}^\complement}(s) \sum_{s' \in \mathcal{X}^\complement} \sum_{a \in \mathcal{A}} \pi(s, a)P(s, a, s'). \tag{271}$$

3. (13 Points) Having related your new value functions to the standard value function, you now talk to Geordi about what to do next to design a reinforcement learning algorithm using your new value functions. Another friend named Data loads the course notes from CMPSCI 687 in Fall 2019. He finds that the next step towards developing an algorithm with this value function may be to write out a new Bellman equation for $v^\pi_{\mathcal{X}}$. Derive a Bellman-like equation for this new value function. You should begin with the definition of $v^\pi_{\mathcal{X}}$ according to your answer to the first question, and should end with a recursive expression for $v^\pi_{\mathcal{X}}$ that is written only in terms of $\mathcal{S}, \mathcal{A}, \mathcal{P}, R, d_0, \gamma, \pi, \mathcal{X}$, and $\mathcal{X}^\complement$ (you may choose not to use some of these terms, but you cannot use others). Your definition may also be in terms of $v^\pi_{\mathcal{X}}$ and $v^\pi_{\mathcal{X}^\complement}$, but should be in the form of a Bellman-like equation (e.g., don't write $v^\pi_{\mathcal{X}}(s) = v^\pi_{\mathcal{X}}(s)$). For this problem, use an alternate definition of R: $R(s, a, s') \coloneqq \mathbf{E}[R_t|S_t = s, A_t = a, S_{t+1} = s']$. (Hint: Using font size "tiny", our answer spans two lines—do not expect a short answer).

$$v^{\pi}_{\mathcal{X}}(s) = \mathbf{E}\left[G_t \middle| S_t = s, S_{t+1} \in \mathcal{X}, \pi\right] \tag{272}$$

$$= \sum_{a \in \mathcal{A}} \Pr(A_t = a | S_t = s, S_{t+1} \in \mathcal{X}, \pi) \mathbf{E}\left[G_t \middle| S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, \pi\right] \tag{273}$$

$$\overset{(\mathbf{a})}{=} \sum_{a \in \mathcal{A}} \frac{\Pr(A_t = a, S_{t+1} \in \mathcal{X} | S_t = s, \pi)}{\Pr(S_{t+1} \in \mathcal{X} | S_t = s, \pi)} \mathbf{E}\left[G_t \middle| S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, \pi\right] \tag{274}$$

$$\overset{(\mathbf{b})}{=} \sum_{a \in \mathcal{A}} \frac{\Pr(A_t = a | S_t = s, \pi) \Pr(S_{t+1} \in \mathcal{X} | S_t = s, A_t = a, \pi)}{\Pr(S_{t+1} \in \mathcal{X} | S_t = s, \pi)} \mathbf{E}\left[G_t \middle| S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, \pi\right] \tag{275}$$

$$\overset{(\mathbf{c})}{=} \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} \Pr(S_{t+1} = s' | S_t = s, A_t = a, \pi)}{\Pr(S_{t+1} \in \mathcal{X} | S_t = s, \pi)} \mathbf{E}\left[G_t \middle| S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, \pi\right] \tag{276}$$

$$= \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} P(s, a, s')}{\Pr(S_{t+1} \in \mathcal{X} | S_t = s, \pi)} \mathbf{E}\left[G_t \middle| S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, \pi\right] \tag{277}$$

$$= \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} P(s, a, s')}{\sum_{b \in \mathcal{A}} \Pr(A_t = b | S_t = s, \pi) \Pr(S_{t+1} \in \mathcal{X} | A_t = b, S_t = s, \pi)} \mathbf{E}\left[G_t \middle| S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, \pi\right] \tag{278}$$

$$= \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} P(s, a, s')}{\sum_{b \in \mathcal{A}} \pi(s, b) \sum_{x \in \mathcal{X}} P(s, b, x)} \mathbf{E}\left[G_t \middle| S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, \pi\right] \tag{279}$$

$$= \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} P(s, a, s')}{\sum_{b \in \mathcal{A}} \pi(s, b) \sum_{x \in \mathcal{X}} P(s, b, x)} \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, \pi) \mathbf{E}\left[G_t \middle| S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, S_{t+1} = s', \pi\right] \tag{280}$$

$$\overset{(\mathbf{d})}{=} \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} P(s, a, s')}{\sum_{b \in \mathcal{A}} \pi(s, b) \sum_{x \in \mathcal{X}} P(s, b, x)} \sum_{s' \in \mathcal{X}} \Pr(S_{t+1} = s' | S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, \pi) \mathbf{E}\left[G_t \middle| S_t = s, S_{t+1} \in \mathcal{X}, A_t = a, S_{t+1} = s', \pi\right] \tag{281}$$

$$= \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} P(s, a, s')}{\sum_{b \in \mathcal{A}} \pi(s, b) \sum_{x \in \mathcal{X}} P(s, b, x)} \sum_{s' \in \mathcal{X}} \Pr(S_{t+1} = s' | S_t = s, A_t = a, \pi) \mathbf{E}\left[G_t \middle| S_t = s, A_t = a, S_{t+1} = s', \pi\right] \tag{282}$$

$$= \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} P(s, a, s')}{\sum_{b \in \mathcal{A}} \pi(s, b) \sum_{x \in \mathcal{X}} P(s, b, x)} \sum_{s' \in \mathcal{X}} P(s, a, s') \mathbf{E}\left[R_t + \gamma G_{t+1} \middle| S_t = s, A_t = a, S_{t+1} = s', \pi\right] \tag{283}$$

$$= \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} P(s, a, s')}{\sum_{b \in \mathcal{A}} \pi(s, b) \sum_{x \in \mathcal{X}} P(s, b, x)} \sum_{s' \in \mathcal{X}} P(s, a, s') \left(R(s, a, s') + \gamma \mathbf{E}\left[G_{t+1} \middle| S_{t+1} = s', \pi\right]\right) \tag{284}$$

$$= \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} P(s, a, s')}{\sum_{b \in \mathcal{A}} \pi(s, b) \sum_{x \in \mathcal{X}} P(s, b, x)} \sum_{s' \in \mathcal{X}} P(s, a, s') \left(R(s, a, s') + \gamma v^{\pi}(s')\right) \tag{285}$$

$$= \sum_{a \in \mathcal{A}} \frac{\pi(s, a) \sum_{s' \in \mathcal{X}} P(s, a, s')}{\sum_{b \in \mathcal{A}} \pi(s, b) \sum_{x \in \mathcal{X}} P(s, b, x)} \sum_{s' \in \mathcal{X}} P(s, a, s') \tag{286}$$

$$\times \left(R(s, a, s') + \gamma \left(v^{\pi}_{\mathcal{X}}(s') \sum_{s'' \in \mathcal{X}} \sum_{a' \in \mathcal{A}} \pi(s', a') P(s', a', s'') + v^{\pi}_{\mathcal{X}^{\complement}}(s') \sum_{s'' \in \mathcal{X}^{\complement}} \sum_{a' \in \mathcal{A}} \pi(s', a') P(s', a', s'')\right)\right), \tag{287}$$

where **(a)** and **(b)** follow from the definition of conditional probability, **(c)** comes from the property derived in the answer to the previous question, **(d)** holds because the probability is zero for states not in $\mathcal{X}$, and **(e)** holds because conditioning on $S_{t+1} \in \mathcal{X}$ is redundant because we already condition on $S_{t+1} = s'$, where $s' \in \mathcal{X}$. Also the final line comes from the previous answer giving an expression for $v^{\pi}(s)$ in terms of $v^{\pi}_{\mathcal{X}}$ and $v^{\pi}_{\mathcal{X}^{\complement}}$. **Note, that** (282) **is not correct. We will be lenient in grading due to this error.**

4. (5 Points) Consider the following definition of an optimal policy:

For any finite MDP with $\gamma < 1$ and precisely two actions, a_1 and a_2, for any two policies π and π', $\pi \geq \pi'$ iff $\forall s \in \mathcal{S}$, $q^\pi(s, a_1) \geq q^{\pi'}(s, a_1)$. A policy π is optimal iff $\pi \geq \pi'$ for all policies π'.

Is this definition equivalent to the definition from Section 4.5 in the course notes? Prove that your answer is correct.

These definitions are not equivalent. Consider an MDP with $\gamma = 0$, one state s, two actions a_1 and a_2, where $R(s, a_1) < R(s, a_2)$. Let π always select action a_1 and π' always select a_2. Since $q^\pi(s, a)$ does not depend on π due to $\gamma = 0$, we have that $q^\pi(s, a_1) \geq q^{\pi'}(s, a_1)$ for all π', and hence π is optimal under this new definition. However, since $R(s, a_1) < R(s, a_2)$, under the definition of an optimal policy from the class notes, π is not optimal, as $v^\pi(s) = R(s, a_1) < R(s, a_2) = v^{\pi'}(s)$.

5. (5 Points) Consider a different definition of $\geq$ for policies: $\pi \geq \pi'$ iff $\sum_{s\in\mathcal{S}} d_0(s) v^\pi(s) \geq \sum_{s\in\mathcal{S}} d_0(s) v^{\pi'}(s)$. Using this modified version of $\geq$, we can still define an optimal policy to be any policy π such that $\pi \geq \pi'$ for all π'. Prove that using this definition of an optimal policy is equivalent to using our first definition:

$$\pi^* \in \arg\max_{\pi \in \Pi} J(\pi). \tag{288}$$

Notice that

$$
\begin{aligned}
J(\pi) =& \mathbf{E}[G|\pi] && (289)\\
=& \sum_{s\in\mathcal{S}} \Pr(S_0 = s|\pi) \mathbf{E}[G|S_0 = s, \pi] && (290)\\
=& \sum_{s\in\mathcal{S}} d_0(s) \mathbf{E}[G_0|S_0 = s, \pi] && (291)\\
=& \sum_{s\in\mathcal{S}} d_0(s) v^\pi(s). && (292)
\end{aligned}
$$

First we will show that any policy π that is optimal under the definition in (288) is optimal under our new definition:

$$
\begin{aligned}
& \pi \in \arg\max_{\pi\in\Pi} J(\pi) && (293)\\
\implies & \forall \pi' \in \Pi, J(\pi) \geq J(\pi') && (294)\\
\implies & \forall \pi' \in \Pi, \sum_{s\in\mathcal{S}} d_0(s) v^\pi(s) \geq \sum_{s\in\mathcal{S}} d_0(s) v^{\pi'}(s), && (295)
\end{aligned}
$$

by the property that we derived first. Continuing, we have:

$$\pi \in \arg\max_{\pi\in\Pi} J(\pi) \implies \forall \pi' \in \Pi, \pi \geq \pi', \tag{296}$$

which means that π' is optimal under our new definition.

Next we show the other way: that if a policy is optimal under the new definition, then it is optimal under (288). If a policy π^* is optimal under our new definition, then

$$\forall \pi' \in \Pi, \pi^* \geq \pi' \tag{297}$$

$$\implies \forall \pi' \in \Pi, \sum_{s \in \mathcal{S}} d_0(s) v^{\pi^*}(s) \geq \sum_{s \in \mathcal{S}} d_0(s) v^{\pi'}(s) \tag{298}$$

$$\implies \forall \pi' \in \Pi, J(\pi^*) \geq J(\pi'), \tag{299}$$

by the property that we derived first. So,

$$\forall \pi' \in \Pi, \pi^* \geq \pi' \implies \pi^* \in \arg\max_{\pi \in \Pi} J(\pi), \tag{300}$$

which means that π is optimal under (288).

6. (20 Points) In class we proved that the Bellman operator is a contraction, and used this to show that value iteration converges to a unique fixed point. In this problem you will prove that the dynamic programming policy evaluation operator is a contraction, and so the policy evaluation algorithm converges to a unique fixed-point. (From the Bellman equation, it should then be clear that this fixed point is v^π, establishing that our dynamic programming policy evaluation algorithm converges to v^π.) Let f denote the dynamic programming policy evaluation operator (this is (200), viewed as an operator on value function approximations):

$$fv(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v(s')\big). \tag{301}$$

Notice that the definition of f relies on a specific policy π—this is the policy being evaluated by the policy evaluation algorithm. Prove that f is a contraction under the L^∞ norm (the same max norm used in our proof that the Bellman operator is a contraction).

We begin with a property. For any functions $f : \mathcal{X} \to \mathbb{R}$ and $g : \mathcal{X} \to \mathbb{R}$ and any probability distribution p on $\mathcal{X}$,

$$\left| \sum_{x \in \mathcal{X}} p(x) f(x) - \sum_{x \in \mathcal{X}} p(x) g(x) \right| \leq \sum_{x \in \mathcal{X}} p(x) |f(x) - g(x)|. \tag{302}$$

We show that this property holds before continuing with our proof that f is a contraction.

$$\forall x, f(x) - g(x) \leq |f(x) - g(x)| \tag{303}$$

$$\sum_{x \in \mathcal{X}} p(x)(f(x) - g(x)) \leq \sum_{x \in \mathcal{X}} p(x) |f(x) - g(x)| \tag{304}$$

$$\sum_{x \in \mathcal{X}} p(x) f(x) - \sum_{x \in \mathcal{X}} p(x) g(x) \leq \sum_{x \in \mathcal{X}} p(x) |f(x) - g(x)|. \tag{305}$$

If $\sum_{x \in \mathcal{X}} p(x)f(x) - \sum_{x \in \mathcal{X}} p(x)g(x) \geq 0$, then (305) implies (302). If $\sum_{x \in \mathcal{X}} p(x)f(x) - \sum_{x \in \mathcal{X}} p(x)g(x) \geq 0$, then (swapping f and g) $\sum_{x \in \mathcal{X}} p(x)g(x) - \sum_{x \in \mathcal{X}} p(x)f(x) \geq 0$, and so by the sequence of inequalities above with f and g reversed

$$\sum_{x \in \mathcal{X}} p(x)g(x) - \sum_{x \in \mathcal{X}} p(x)f(x) \leq \sum_{x \in \mathcal{X}} p(x)|g(x) - f(x)|. \tag{306}$$

Since $\sum_{x \in \mathcal{X}} p(x)g(x) - \sum_{x \in \mathcal{X}} p(x)f(x) > 0$, taking its absolute value makes no difference and so:

$$\left| \sum_{x \in \mathcal{X}} p(x)g(x) - \sum_{x \in \mathcal{X}} p(x)f(x) \right| \leq \sum_{x \in \mathcal{X}} p(x)|g(x) - f(x)|. \tag{307}$$

Due to the absolute values, the order of f and g make no difference in the above inequality, and so we again can conclude that (302) holds. We now show that f is a contraction, using the above property to obtain the second line:

$$\begin{aligned}
\|fv - fv'\|_\infty &= \max_{s \in \mathcal{S}} \left| \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v(s')) - \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v'(s')) \right| && (308) \\
&\leq \max_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \pi(s,a) \left| \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v(s')) - \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v'(s')) \right| && (309) \\
&= \gamma \max_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \pi(s,a) \left| \sum_{s' \in \mathcal{S}} P(s,a,s')(v(s') - v'(s')) \right| && (310) \\
&\leq \gamma \max_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s') \left|(v(s') - v'(s'))\right| && (311) \\
&\leq \gamma \max_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \pi(s,a) \max_{s' \in \mathcal{S}} \left|(v(s') - v'(s'))\right| && (312) \\
&= \gamma \max_{s' \in \mathcal{S}} \left|(v(s') - v'(s'))\right| && (313) \\
&= \gamma \|v - v'\|_\infty. && (314)
\end{aligned}$$

Part Two: Programming (25 Points Total)

For this part of the assignment, you will implement value iteration (modified to terminate when the value function estimate has not changed significantly between two iterations). Your program will read an MDP from a file, run value iteration on the MDP, and output the final estimate of the optimal value function and the policies that are greedy with respect to this value function. As a soft introduction to C++, we are providing you with most of the code here: your job is to fill in the missing lines in the function `valueIteration`, marked with a comment saying "TODO". Do not change the code logic outside of the valueIteration function (you may add new functions if you like, but do not modify any of the other functions in your final submission or it may fail to run as expected in our auto-grader). You may introduce additional include statements, but should only use common libraries (all of the C++ standard libraries are allowed).

You are free to use any IDE or toolchain you would like to program in C++. If you are not familiar with C++, we have provided two different systems for opening and working with this C++ code. If you are using Windows, you should download Microsoft Visual Studio. The community version is perfectly sufficient, and is free online (in my opinion, this is the best C++ experience out there). Clicking on the .sln file in HW3/build/VisualStudio will open the project. On the left you should see main.cpp—open this file to see all of the code for this assignment. If you are using Mac or Linux, we have provided a CLion project. CLion is free for students. To open this project, select "Open" when launching CLion. Select the file HW3/build/CLion/CMakeLists.txt. When prompted, select "Open as Project". If main.cpp does not immediately open, on the left click on HW3/main.cpp.

This assignment is your chance to begin to familiarize yourself with C++. Please look over all of the provided code, and feel free to ask if you have questions about what some portion of the code is doing. Also, take this opportunity to familiarize yourself with the debugger in your IDE—developing simple programs in C++ is a breeze when you are familiar with how to use the different capabilities of your debugger.

We have provided you (within the provided code) with 687Gridworld.txt, a text file containing the MDP we have been using in class. We will evaluate your program on other MDPs that we are not providing to you. You are welcome to create your own test MDPs, but do not share these with others.

You must submit your main.cpp file. A correct implementation is worth 20 points. *Any* incorrect output (beyond numerical issues) will result in 0/20 points. In the .pdf that you submit, answer the following questions.

1. (2 Points) Did your final code compile on your machine? (Yes or no).

2. (3 Points) Comment on your experience with this problem. Did your first implementation work, or did you introduce a bug at first? Was there anything we could do to smooth your introduction to C++? Did you implement any additional test MDPs (you do not have to in order to get full credit). Did the number of iterations required by value iteration surprise you? Do you have any other comments on this problem?

We can construct an unbiased estimator of $v^\pi(s)$ by computing the discounted return starting from the first occurrence of s within a history sampled using the policy, π. If we were to compute the return from the last visit to the state, s, it would not necessarily produce an unbiased estimate. To see why, consider an MDP with a single state, s_0, that self-transitions with probability 0.5, and transitions to s_∞ with probability 0.5. Let $\gamma = 1$. Let the reward be $+1$ for self-transitions, and 0 for transitioning to s_∞. The value of s_0 in the example is 1, and the expected return from the first visit to s_0 is 1. However, if we compute the expected return from the *last* visit to s_0, it is zero.

Next consider the expected return if we only consider returns from the second time that s is visited in each trajectory. By the Markov property, what happened prior to entering state s will not change the expected return, and so this remains an unbiased estimator of $v^\pi(s)$.

However, consider what happens if we average the expected return from every occurrence of s. This is not necessarily an unbiased estimator. Consider our example above. The expected value of the every-visit estimator can be computed as follows, where the 0.5^k terms compute the probability that S_{k+1} is the first occurrence of s_∞, and the values in parentheses after these terms are the corresponding average return from all visits to s (that is, $\frac{1}{k}\sum_{i=0}^{k-1} i = (k-1)k/2k = (k-1)/2$).

$$0.5(0) + 0.5^2\left(\frac{1}{2}\right) + 0.5^3\left(\frac{3}{3}\right) + 0.5^4\left(\frac{3}{2}\right) + 0.5^5\left(2\right) + \ldots + 0.5^k\left(\frac{k-1}{2}\right)\ldots \tag{315}$$

$$=\sum_{k=1}^{\infty} 0.5^k\left(\frac{k-1}{2}\right) \tag{316}$$

$$=0.5. \tag{317}$$

Since $v^\pi(s) = 1$, this means that averaging the values from every visit provides a biased estimate (an estimate with expected value 0.5).

This suggests a simple algorithm for estimating v^π: generate many episodes of data, and for each state, average the discounted returns after it was visited for the first time in each episode. This algorithm is called *First-Visit Monte Carlo*, pseudocode for which is provided in Algorithm 11.

Algorithm 11: First-Visit Monte Carlo
Input:
1) Policy, π, whose state-value function will be approximated
2) Initial state-value function estimate, v (e.g., initialized to zero)

```
Returns(s) ← an empty list, for all s ∈ S. while true do
    Generate an episode using π;
    for each state, s, appearing in the episode do
        t ← time of first occurrence of s in the episode;
        G ← Σ_{k=0}^∞ γ^k R_{t+k};
        Append G to Returns(s);
        v(s) ← average(Returns(s));
```

If every state is visited infinitely often, then (for any finite MDP with bounded rewards and $\gamma < 1$) the state-value function estimate, v, in First-Visit Monte Carlo converges almost surely to v^π. The proof of this property stems from the Khitchine strong law of large numbers:

Property 2 (Khintchine Strong Law of Large Numbers)**.** *Let $\{X_i\}_{i=1}^\infty$ be independent and identically distributed random variables. Then $(\frac{1}{n}\sum_{i=1}^n X_i)_{n=1}^\infty$ is a sequence of random variables that converges almost surely to $\mathbf{E}[X_1]$, i.e., $\frac{1}{n}\sum_{i=1}^n X_i \xrightarrow{a.s.} \mathbf{E}[X_1]$.*

Proof. See the work of Sen and Singer (1993, Theorem 2.3.13). □

For a review of almost sure convergence, see Wikipedia. Another useful law of large numbers, which we will not use here, is the Kolmogorov trong law of large numbers:

Property 3 (Kolmogorov Strong Law of Large Numbers)**.** *Let $\{X_i\}_{i=1}^\infty$ be independent (not necessarily identically distributed) random variables. If all X_i have the same mean and bounded variance, then $(\frac{1}{n}\sum_{i=1}^n X_i)_{n=1}^\infty$ is a sequence of random variables that converges almost surely to $\mathbf{E}[X_1]$.*

Proof. See the work of Sen and Singer (1993, Theorem 2.3.10 with Proposition 2.3.10). □

To see that First-Visit Monte Carlo converges almost surely to v^π, consider the sequence of estimates, $v_k(s)$ for a particular state, s. We have that $v_k(s) = \frac{1}{k}\sum_{i=1}^k G_i$, where G_i is the i^{th} return in Returns(s). Notice that $\mathbf{E}[G_i] = v^\pi(s)$ and that the G_i are i.i.d. because they are sampled from independent episodes. Hence, by Khintchine's strong law of large numbers we have that $v_k(s) \xrightarrow{\text{a.s.}} v^\pi(s)$. Furthermore, because the number of states is finite, we have that this convergence is uniform, not just pointwise. That is, not only does the value of each state converge almost surely to the correct value, but the entire value function estimate converges to the true state-value function.

Furthermore, $\operatorname{Var}(v_k(s)) \propto \frac{1}{k}$ since (using the fact that G_i are i.i.d.):

$$
\begin{aligned}
\operatorname{Var}(v_k(s)) =& \operatorname{Var}\left(\frac{1}{k}\sum_{i=1}^{k} G_i\right) && (318)\\
=& \frac{1}{k^2}\operatorname{Var}\left(\sum_{i=1}^{k} G_i\right) && (319)\\
=& \frac{1}{k^2} k \operatorname{Var}\left(G_i\right) && (320)\\
=& \frac{1}{k}\operatorname{Var}\left(G_i\right). && (321)
\end{aligned}
$$

An alternative to First-Visit Monte Carlo is *Every-Visit Monte Carlo*, which uses the return from *every* visit to state s during an episode. Pseudocode for this algorithm is provided in Algorithm 12. Notice that the return estimates used by Every-Visit Monte Carlo are *not* all unbiased estimators of $v^\pi(s)$. Furthermore, these estimators are *not* all independent, since two returns computed from the same episode may be correlated. Hence, our argument using Khintchine's strong law of large numbers does not apply, and we also cannot directly use Kolmogorov's strong law, since the G_i will not be independent. However, it can be shown that if every state is visited infinitely often, then (for any finite MDP with bounded rewards and $\gamma < 1$) the state-value approximation of Every-Visit Monte Carlo also converges almost surely to v^π.

Algorithm 12: Every-Visit Monte Carlo
Input:
1) Policy, π, whose state-value function will be approximated
2) Initial state-value function estimate, v (e.g., initialized to zero)

```
Returns(s) ← an empty list, for all s ∈ S. while true do
    Generate an episode using π;
    for each state, s, appearing in the episode and each time, t, that it occurred do
        G ← Σ_{k=0}^∞ γ^k R_{t+k};
        Append G to Returns(s);
        v(s) ← average(Returns(s));
```

Notice that we can also use Monte Carlo methods to estimate action values. The idea is the same: our estimate, $\hat{q}(s, a)$ will be the average return from the first time that action, a, was taken in state s in each episode. This raises a problem: what if the policy, π, never takes an action, a? If we are going to use Monte Carlo approximation to estimate $q^\pi(s, a)$ within policy iteration, we need to compute the action-values for all actions, not just the actions that π takes (this is particularly true because policy iteration typically uses deterministic policies). That is, we need to estimate the value of *all* actions at each state, not just the actino that we currently favor.[7] One way to fix this problem is

[7]Interestingly, this problem comes up in more modern reinforcement learning results as

to use *exploring starts*: to randomize S_0 and A_0 such that every state-action pair has non-zero probability of being the initial state and action. Although effective, this is not always possible (some systems cannot be reset to arbitrary states—you can reset a chess board to a different state, but you cannot reset a student interacting with a tutoring system to a particular state). Another solution is to use a stochastic policy—to ensure that the policies being evaluated have non-zero probability for every action in every state.

Using these Monte Carlo evaluation algorithms we can create a Monte Carlo control algorithm. Recall that we refer to algorithms as *control* algorithms if they search for an optimal policy, and *evaluation* algorithms if they estimate the value function associated with a policy. In order to use Monte Carlo evaluation within the policy iteration algorithm, we must estimate q^π rather than v^π. This is because, now that we are assuming P and R are *not* known, we could not perform the greedy policy improvement step in Algorithm 8. However, if we estimate the action-value function we can: the greedy policy, π_{i+1} with respect to the current policy, π_i, satisfies the following for all s:

$$\pi_{i+1}(s) \in \arg\max_{a \in \mathcal{A}} q^{\pi_i}(s, a). \tag{322}$$

Unlike line 214 of Algorithm 8, this can be computed without knowledge of P or R.

Policy iteration, using Monte Carlo evaluation instead of dynamic programming policy evaluation (and estimating q^π instead of v^π), has the same properties as standard policy iteration if Monte Carlo evaluation is guaranteed to converge to q^π (e.g., if using exploring starts).

This algorithm remains impractical because it calls for an infinite number of episodes to be run in order to compute one iteration (to evaluate a policy). We can create a Monte Carlo control algorithm similar to value iteration, which avoids this infinite number of episodes in the evaluation step by terminating evaluation after one episode. This algorithm accumulates returns over all episodes—it uses all past returns to evaluate the current policy, not just the returns generated by

well. For example, Silver et al. (2014) present the deterministic policy gradient theorem, which considers a deterministic policy, π, but requires estimates of the action-value function for actions that the deterministic policy will never take. The solution they propose is the same as one we use here: sampling using a stochastic policy.

the current policy. Pseudocode for this algorithm is presented in Algorithm 13.

Algorithm 13: Monte Carlo - Exploring Starts.

1 **for** *all* $s \in \mathcal{S}$ *and* $a \in \mathcal{A}$ **do**
2     $q(s,a) \leftarrow$ arbitrary;
3     $\pi(s) \leftarrow$ arbitrary;
4   Returns$(s,a) \leftarrow$ empty list;
5 **for** $i = 0$ **to** ∞ **do**
6   Generate an episode using exploring starts and π;
7   **for** *each* (s,a) *appearing in the episode* **do**
8         $G \leftarrow$ return following the first occurrence of (s,a);
9   Append G to Returns(s,a);
10         $q(s,a) \leftarrow$ average(Returns(s,a));
11   **for** *each* s *in the episode* **do**
12         $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} q(s,a)$;

Notice that this algorithm cannot converge to a sub-optimal policy. If it did, then q would converge to q^π (by the convergence of first-visit Monte Carlo for policy evaluation), and π would be improved (if not, then π is a fixed-point of the Bellman operator, and so it is an optimal policy). Notice also that we can avoid exploring starts by removing the exploring starts and changing line 11 to compute the greedy action, $a^* \leftarrow \arg\max_{a \in \mathcal{A}} q(s,a)$ (if more than one action is optimal, select one of the optimal actions arbitrarily), and then defining the ϵ-greedy policy (a stochastic policy):

$$\pi(s,a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|} & \text{if } a = a* \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise.} \end{cases} \tag{323}$$

By variations on the policy improvement theorem, policy iteration using ϵ-greedy policies converges to an optimal ϵ-greedy policy. For more details on this topic, see the work of Sutton and Barto (1998, Section 5.4)

———End of Lecture 14, October 22, 2019———

Name:

CMPSCI 687 Pop Quiz 4
October 24, 2019

Instructions: You have 7 minutes to complete this quiz. This quiz is **closed** notes—do not use your notes or a laptop. Do not discuss problems with your neighbors until after everyone has handed in their quiz.

Use the notation from class. Don't forget to capitalize your random variables.

1. The Bellman equation states that for all s:

$$v^\pi(s) = \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v^\pi(s')\big) \tag{324}$$

2. The Bellman optimality equation states that for all s:

$$v^*(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v^*(s')\big) \tag{325}$$

3. The value iteration update is:

$$v_{i+1} = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v_i(s')\big) \tag{326}$$

4. (True or False) The Bellman operator is a contraction mapping for all finite MDPs with bounded rewards and $\gamma < 1$.

5. (True or False) Consider the value functions v_i and v_{i+1} from two iterations of value iteration. Let π_i and π_{i+1} be the policies that are greedy with respect to these value functions. It is always true (for finite MDPs with bounded rewards and $\gamma < 1$) that $\pi_{i+1} \geq \pi_i$.

7.2 A Gradient-Based Monte Carlo Algorithm

Consider another Monte Carlo algorithm. It begins with a value function estimate, $v \in \mathbb{R}^{|\mathcal{S}|}$. At time t it performs the update:

$$v(S_t) \leftarrow v(S_t) + \alpha(G_t - V(S_t)). \tag{327}$$

We will see many updates of this form. The general form is:

$$f(x) \leftarrow f(x) + \alpha(g(x) - f(x)). \tag{328}$$

Here we refer to $g(x)$ as the *target* for $f(x)$, and this update changes $f(x)$ to be more similar to $g(x)$.

We now show that this update can be viewed as the gradient descent update on a loss function called the *mean squared value error* (MSVE):

$$\operatorname{MSVE}(v) \coloneqq \mathbf{E}\left[\frac{1}{2}(v^\pi(S) - v(S))^2\right], \tag{329}$$

where the expectation is over states, S. The precise distribution of states can be found in the second edition of Sutton and Barto's book—here we use the intuition that this distribution is the "observed distribution of states when the policy π is executed.

The gradient descent algorithm on MSVE uses the update:

$$\begin{aligned}
v \leftarrow & v - \alpha\frac{\partial \operatorname{MSVE}(v)}{\partial v} && (330)\\
=& v - \alpha\frac{\partial}{\partial v}\mathbf{E}\left[\frac{1}{2}(v^\pi(S) - v(S))^2\right] && (331)\\
=& v + \alpha\mathbf{E}\left[(v^\pi(S) - v(S))\frac{\partial v(S)}{\partial v}\right]. && (332)
\end{aligned}$$

Because we do not know v^π, nor the distribution over states that results from running π, we cannot compute this gradient update. Instead we can perform *stochastic gradient descent*, wherein an unbiased estimator of the gradient is used. We can obtain an unbiased estimate by sampling the state, S, and using the Monte Carlo return, G, in place of $v^\pi(S)$. Thus we obtain the stochastic gradient descent update:

$$v \leftarrow v + \alpha(G_t - V(S_t))\frac{\partial v(S_t)}{\partial v}. \tag{333}$$

Consider now the term $\partial v(S)/\partial v$. This term is a vector with $|\mathcal{S}|$ entries, all of which are zero, except for the S^{th}, which is one. Hence, the update update to the entire vector v can be written as an update to only the S^{th} term, since all other updates are set to zero by multiplication with $\partial v(S)/\partial v$, giving the update in (327).

Thus, because (327) is an instance of the stochastic gradient descent algorithm, its convergence properties have been well-studied (Bertsekas and Tsitsiklis, 2000).

That is, with sufficient smoothness assumptions, it will converge to a locally optimal solution. Furthermore, because MSVE is a quadratic function of v, it is convex, and the local optimum is the global minimum—stochastic gradient descent will converge to v^π (with different forms of convergence given different smoothness assumptions and assumptions on the step size sequence).

Notice that this algorithm can easily be adapted to work with continuous states. Let v_w be a function parameterized by the vector $w \in \mathbb{R}^n$ (here n is not related to any n previously discussed—it is an arbitrary integer). That is, different vectors, w, result in v being a different function, but for all $w \in \mathbb{R}^n$, $v_w : \mathcal{S} \to \mathbb{R}$. In reinforcement learning literature, we refer to v_w as a *function approximator*. A common example of a function approximator is an artificial neural network, wherein w are the weights of the network.

Following our derivation of (327) as the stochastic gradient descent update for MSVE, we can obtain the equivalent update using function approximation:

$$w \leftarrow w + \alpha(G_t - v_w(S_t))\frac{\partial v_w(S_t)}{\partial w}. \tag{334}$$

Again, because this algorithm is stochastic gradient descent, it will converge almost surely to a locally optimal solution given the appropriate assumptions. However, because v_w may not be a linear function of w, this solution may not be a global optimum. Furthermore, the global optimum may not be v^π, if this true state-value function is not representable by the chosen function approximator. Lastly, notice that (327) is a special case of (334), where v_w stores one number per state.

8 Temporal Difference (TD) Learning

Temporal difference learning, introduced by (Sutton, 1988a), is a policy evaluation algorithm. Like Monte-Carlo algorithms, it learns from experiences (by sampling—choosing actions using π and seeing what happens) rather than requiring knowledge about P and R. However, like the dynamic programming methods it produces estimates based on other estimates—it *bootstraps*. This latter property means that it can perform its updates before the end of an episode (a requirement of the Monte Carlo methods).

Like previous algorithms, TD begins with an initial value function estimate, v. As an *evaluation* algorithm rather than a *control* algorithm, it estimates v^π (as opposed to obtaining π^* by estimating q^*). The TD update given that the agent was in state s, took action a, transitioned to state s', and obtained reward r is:

$$v(s) \leftarrow v(s) + \alpha(r + \gamma v(s') - v(s)). \tag{335}$$

Using other notation it can be defined equivalently as:

$$v(S_t) \leftarrow v(S_t) + \alpha(R_t + \gamma v(S_{t+1}) - v(S_t)). \tag{336}$$

This is very much like the Gradient-Based Monte Carlo Algorithm from Section 7.2, except that instead of using G_t as the target, it uses $R_t + \gamma v(S_{t+1})$.

The *temporal difference error* (TD error), δ_t is defined as:

$$\delta_t = R_t + \gamma v(S_{t+1}) - v(S_t), \tag{337}$$

and allows us to write the TD update as:

$$v(S_t) \leftarrow v(S_t) + \alpha\delta. \tag{338}$$

Notice that a positive TD error means that the observed outcome (the reward, R_t, plus the value, $v(S_{t+1})$, of the resulting state) was better than what was expected (i.e., the value, $v(S_t)$, of the current state). Also, note that the TD error can refer to different terms: it can use the current value estimate, v, or it could use the true state-value function, v^π. In both cases δ_t is referred to as the TD error.

Question 25. *What is* $\mathbf{E}[\delta_t|S_t=s]$ *if* δ_t *uses the true state-value function,* v^π*?*

Answer 25.

$$\mathbf{E}[\delta_t|S_t=s] = \mathbf{E}[R_t + \gamma v^\pi(S_{t+1}) - v^\pi(S_t)|S_t=s] \tag{339}$$
$$= \mathbf{E}[R_t + \gamma v^\pi(S_{t+1})|S_t=s] - v^\pi(s) \tag{340}$$
$$\underset{(a)}{=} v^\pi(s) - v^\pi(s) \tag{341}$$
$$= 0. \tag{342}$$

where (a) comes from the Bellman equation.

Question 26. *What is* $\mathbf{E}[\delta_t|S_t=s, A_t=a]$ *if* δ_t *uses the true state-value function,* v^π*?*

Answer 26.

$$\mathbf{E}[\delta_t|S_t=s, A_t=a] = \mathbf{E}[R_t + \gamma v^\pi(S_{t+1}) - v^\pi(S_t)|S_t=s, A_t=a] \tag{343}$$
$$= \mathbf{E}[R_t + \gamma v^\pi(S_{t+1})|S_t=s, A_t=a] - v^\pi(s) \tag{344}$$
$$\underset{(a)}{=} q^\pi(s,a) - v^\pi(s) \tag{345}$$
$$= A(s,a), \tag{346}$$

where here A is a function we have not yet discussed called the advantage function, and $A: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, $A(s,a) := q^\pi(s,a) - v^\pi(s)$. Note that this definition of the advantage function, although popular recently and in policy gradient algorithms, differs from the original definition presented and studied by Baird (1993).

Consider further all of the possible causes for a positive TD error. A positive TD error might occur because **1)** $v(s)$ was too small, **2)** the random nature of rewards and state transitions, combined with luck, and/or **3)** $v(s')$ was too large. If $v = v^\pi$, then the TD errors are due to #2, but will average out to be mean-zero updates (this follows from the Bellman equation). If $v \neq v^\pi$, then the TD update attempts to correct for #1, but does not make corrections due to #3. This is because, by the Markov property, we know that $v^\pi(s')$ does not depend on how we got to state s', and yet the TD error is due to the transition (s, a, r, s')—i.e., the TD error describes events prior to reaching s', and should not impact our estimates of the value of state s'.

Notice that the TD update can also be viewed as converting the Bellman equation into an update rule, just like with policy evaluation using dynamic programming. However, whereas we could exactly compute the right side of the Bellman equation when using dynamic programming (because we assumed P and R are known), the TD algorithm does not assume P and R are known and so instead uses *sampling*—it samples A_t, R_t, and S_{t+1} according to π, P, and R.

Notice that we can write the TD update with function approximation as:

$$w \leftarrow w + \alpha(R_t + \gamma v_w(S_{t+1}) - v_w(S_t))\frac{\partial v_w(S_t)}{\partial w}. \tag{347}$$

Question 27. *Consider the function approximator v_w that is defined such that $|w| = |\mathcal{S}|$ and the i^{th} element of w is $v_w(s)$. This* tabular representation *causes the update using function approximation to be equivalent to the update in (336). Prove this.*

One might think that the TD algorithm is a gradient algorithm, much like the Gradient-Based Monte Carlo Algorithm from Section 7.2, except with the target replaced with $R_t + \gamma v(S_{t+1})$. However, this is not the case. Consider how one might try to derive TD as a gradient algorithm. We begin by defining our loss function:

$$L(w) = \mathbf{E}\left[\frac{1}{2}(R_t + \gamma v_w(S_{t+1}) - v_w(S_t))^2\right] \tag{348}$$

$$= \mathbf{E}[\frac{1}{2}\delta_t^2]. \tag{349}$$

We then compute the gradient:

$$\frac{\partial}{\partial w}L(w) = \frac{\partial}{\partial w}\mathbf{E}\left[\frac{1}{2}(R_t + \gamma v_w(S_{t+1}) - v_w(S_t))^2\right] \tag{350}$$

$$= \mathbf{E}\left[\delta_t\left(\gamma\frac{\partial v_w(S_{t+1})}{\partial w} - \frac{\partial v_w(S_t)}{\partial w}\right)\right] \tag{351}$$

$$= \mathbf{E}\left[-\delta_t\left(\frac{\partial v_w(S_t)}{\partial w} - \gamma\frac{\partial v_w(S_{t+1})}{\partial w}\right)\right], \tag{352}$$

where the sign change in the last term is to obtain a standard form. This suggests a stochastic gradient descent update (notice that the negative from this being a descent algorithm cancels with the negative before the δ_t):

$$w \leftarrow w + \alpha\delta_t \left(\frac{\partial v_w(S_t)}{\partial w} - \gamma \frac{\partial v_w(S_{t+1})}{\partial w} \right). \tag{353}$$

Notice that **1)** due to subtle reasons, (353) is *not* an unbiased estimator of (352) (Baird, 1995), and **2)** this update is *not* the same as the TD update. Consider what this update does when the TD error is positive: it changes w to increase $v_w(S_t)$ and to decrease $v_w(S_{t+1})$, whereas the TD update only increases $v_w(S_t)$. To make this more clear, notice that (353) using tabular function approximation can be written as:

$$v(S_t) \leftarrow v(S_t) + \alpha\delta_t \tag{354}$$

$$v(S_{t+1}) \leftarrow v(S_{t+1}) - \alpha\gamma\delta_t. \tag{355}$$

This alternate algorithm is *not residual gradient* (Baird, 1995), but is similar.[8]

Just because we tried to derive the TD algorithm as the gradient of a loss function and obtained a different algorithm does *not* mean that the TD algorithm is not a gradient algorithm—it just means it is not the (stochastic) gradient of L as we defined it. However, it can be shown that the TD algorithm is not a stochastic gradient algorithm for *any* objective function. If it were, then the expected TD update must be the gradient of a loss function (the gradient of an objective function). That is,

$$\mathbf{E}\left[\delta_t \frac{\partial v_w(S_t)}{\partial w} \right], \tag{356}$$

would be the gradient of a function. We can show that this is not the case: (356) is not the gradient of any loss function (with continuous second derivatives). More precisely, recall that for any function L that has continuous second partial derivatives at w, the Hessian, $\partial^2 L(w)/\partial w^2$, must be symmetric (see Schwarz's theorem). If (356) were the gradient of the function, then its derivative would be the Hessian. Rather than compute the complete derivative, let us compute what $\partial^2/\partial w_i \partial w_j$ would be for the loss function—that is, the partial derivative with respect to w_i of the j^{th} element of (356). This term is:

$$\frac{\partial}{\partial w_i} \delta_t \frac{\partial v_w(S_t)}{\partial w_j} = \delta_t \frac{\partial^2 v_w(S_t)}{\partial w_i \partial w_j} + \frac{\partial v_w(S_t)}{\partial w_j} \frac{\partial}{\partial w_i} \left(R_t + \gamma v_w(S_{t+1}) - v(S_t) \right) \tag{357}$$

$$= \underbrace{\delta_t \frac{\partial^2 v_w(S_t)}{\partial w_i \partial w_j}}_{(a)} + \underbrace{\frac{\partial v_w(S_t)}{\partial w_j} \left(\gamma \frac{\partial v_w(S_{t+1})}{\partial w_i} - \frac{\partial v(S_t)}{\partial w_i} \right)}_{\text{(b)}}. \tag{358}$$

[8] Residual gradient takes the gradient of the mean squared Bellman error, $\mathbf{E}[\delta_t]^2$, rather than the mean squared TD error, $\mathbf{E}[\delta_t^2]$. However, in doing so, it requires *double sampling* to get an unbiased gradient estimate (Baird, 1995).

Notice that, although the term **(a)** is symmetric–it is the same if i and j are flipped, assuming that v_w has continuous second derivatives, the term **(b)** is *not* symmetric—flipping i and j does change its value. To see why, consider using tabular function approximation and the case where w_j is the weight for S_t and w_i is the weight for S_{t+1}, and $S_t \neq S_{t+1}$—the **(b)** term will not necessarily be zero, but if w_j were the weight for S_{t+1}, then this term would be zero. Hence, the derivative of the expected TD update is not symmetric, and so the TD update cannot be a stochastic gradient update for a loss function that has continuous second partial derivatives.

Despite the TD algorithm not being a gradient algorithm, it does have desirable convergence properties. When using a tabular representation for the value function approximation, TD converges with probability one to v^π given standard assumptions and decaying step sizes (Dayan and Sejnowski, 1994; Jaakkola et al., 1994), and it converges in mean to v^π if the step size is sufficiently small (Sutton, 1988b). When using linear function approximation—when $v_w(s)$ can be written as $v_w(s) = w^\intercal \phi(s)$ for some function $\pi : \mathcal{S} \to \mathbb{R}^n$ (for some n)—TD converges with probability one to some weight vector, w_∞, given standard assumptions (Tsitsiklis and Van Roy, 1997). If there exists a weight vector such that $v_w = v^\pi$, then $v_{w_\infty} = v^\pi$—TD will converge to weights that cause v_w to be v^π. If, however, there is no weight vector w such that $v_w = v^\pi$ (if the state-value function cannot be precisely represented given the class of functions that can be produced by v_w for various w), then the weight vector that TD converges to (with probability one) is *not* necessarily the "best" possible weight vector, w^*:

$$w^* \in \arg\min_w \mathbf{E}[(v_w(S_t) - v^\pi(S_t))^2]. \tag{359}$$

However, w_∞ satisfies the following inequality that ensures that the weights that TD converges to with probability 1 will not be "too far" away from these optimal weights (Tsitsiklis and Van Roy, 1997, Theorem 1):

$$\mathbf{E}[(v_{w_\infty}(S_t) - v^\pi(S_t))^2] \leq \frac{1}{1-\gamma}\mathbf{E}[(v_{w^*}(S_t) - v^\pi(S_t))^2]. \tag{360}$$

————————End of Lecture 15, October 24, 2019————————

When using non-linear function approximation, TD can diverge.

What makes for a better target, the Monte-Carlo return, G_t, or the target used by TD, $R_t + \gamma v(S_{t+1})$? Each is an *estimator* of $v^\pi(S_t)$. The *mean squared error* (MSE) is a common measurement of how "bad" an estimator is. Let a random variable, X, be an estimator of $\theta \in \mathbb{R}$. The MSE of X is defined as:

$$\text{MSE}(X) \coloneqq \mathbf{E}[(X - \theta)^2]. \tag{361}$$

The MSE can be decomposed into two components: the squared bias and the variance:

$$\text{MSE}(X) = \text{Bias}(X)^2 + \text{Var}(X), \tag{362}$$

where $\text{Bias}(X) = \mathbf{E}[X - \theta]$ and $\text{Var}(X)$ is the variance of X. Consider again the two possible targets, each of which is an estimator of v^π—which is a "better" estimator?

The Monte-Carlo return is unbiased, and so it has zero bias. However, it often has high variance because it depends on all of the rewards that occur during an episode. The TD target can be biased if $v \neq v^\pi$, since it replaces all of the rewards in the Monte Carlo return, except for the first, with a biased estimate, $v(S_{t+1})$ (this is biased because $v \neq v^\pi$). However, it can have much lower variance because it only looks forward a single time-step: both R_t and S_{t+1} (the only random terms in the TD target) can be computed after a single time step. Hence, TD and Monte Carlo are on opposite extremes: the Monte Carlo target has high variance but no bias, and the TD target has low variance but high bias. Later we will discuss ways to create estimators that can provide a better trade-off of bias and variance in order to obtain targets that are "better" estimates of v^π.

9 Function Approximation

Before continuing, it is worth discussing function approximation in more detail. First, notice that we say that v_w is a *linear* function approximator if it is a linear function of w. This does *not* mean that v_w must be a linear function of the states. Furthermore, we typically write linear function approximators as:

$$v_w(s) = w^\intercal \phi(s), \tag{363}$$

where $\phi : \mathcal{S} \to \mathbb{R}^n$ maps states to vectors of features.

One possible choice for ϕ is the *polynomial basis*, which assumes that s is real-valued (it can be extended to the multivariate polynomial basis, in which case s is a real vector). Most bases have a parameter that we call the *order*, which controls how many features will be produced. The k^{th} order polynomial basis is:

$$\phi(s) = \begin{bmatrix} 1 \\ s \\ s^2 \\ \vdots \\ s^k \end{bmatrix}. \tag{364}$$

By the Stone-Weierstrass Theorem, any continuous function can be approximated to any desired level of accuracy given a high enough order.

The Fourier basis for value function approximation is a common linear function approximator that works very well for most of the standard benchmark RL problems. The paper presenting the Fourier basis (Konidaris et al., 2011b) should be an easy read at this point, and can be found here. Please read it. Note that the states should be normalized prior to applying the multivariate Fourier basis.

9.1 Function Approximation and Partial Observability

A common misconception about RL is that RL methods designed to solve MDPs cannot handle noise or partial observability. This is *not* the case: RL algorithms often work well even in the presence of noise and partial observability. Although this is not often discussed, function approximation and noise/partial observability are equivalent.

Notice that, when discussing how TD converges when using linear function approximation, we did *not* require $\phi(s)$ to be a Markovian state representation. Hence, once can view $\phi(s)$ as being the *observation* of state s if an agent does not have access to the full state, s. We have from (360) that TD using a linear approximator (with possibly non-Markovian features) will converge, and will converge to weights that are not much worse than the best possible weights. If the features, $\phi(s)$, contain so little information about the state that the best possible weight vector results in a poor value function, then TD will not work well. However, if $\phi(s)$ contains enough information to get a good estimate of the state-value function, then TD will converge, and will converge to weights that result in accurate estimates.

Consider for example our gridworld, but where $s = [x, y]^\intercal$, where x is the horizontal position of the agent ant y is the vertical position. If $\phi(s) = [1, x]^\intercal$, then TD will converge, although it will converge to weights that are not much worse than the best possible weights. In this case, the best possible weights are not very good—all weight vectors make the same predictions for states that have the same x-position but different y-positions.

Consider another example: estimating the value function associated with different medical treatments. The true state to determine how well a drug will work might be all information about a person, down to an atomic level. However, the features, $\phi(s)$, might be a detailed enough description of the person (age, height, weight, temperature, blood glucose, etc.) to accurately predict how well a drug will work, in which case TD may work well.

However, note that ϕ is a deterministic function—if it sees a state, s, it will always produce the exact same features, $\phi(s)$. How then is function approximation equivalent to *noise*? To model a problem with noisy state observations, construct a new MDP where the state is augmented to include noise. For example, in our gridworld we might augment the state to be $s = (x, y, \eta)$, where η is uniformly sampled from $\{-1, 0, 1\}$. We could then define

$$\phi(s) = [\min\{5, \max\{1, x + \eta\}\}, y]^\intercal. \tag{365}$$

This new MDP models the original MDP, but where our sensors have noise added to the observations of the x-position of the agent. Applying TD to the original MDP where the sensors have noise when producing observations of the x-position is equivalent to applying TD to this new MDP with linear function approximation. Hence, TD converges to weights not far from the best possible weights (as described in (360)) when using linear function approximation, and even if the state observations are noisy and/or incomplete.

9.2 Maximum Likelihood Model of an MDP versus Temporal Difference Learning

If we have data regarding many transitions, (s, a, r, s'), we can use this data to estimate P and d_R. Notice that we can do this regardless of how this data is generated—by running complete episodes, by randomly sampling s, etc. For now, we assume that a is sampled according to π. One common estimator is the *maximum likelihood model*—the estimates of P and d_R that maximize the probability that we would see the data we have. The maximum likelihood model for an MDP with finite states and actions is exactly what one might expect. That is:

$$\hat{P}(s, a, s') = \frac{\#(s, a, s')}{\#(s, a)} \tag{366}$$

$$\hat{R}(s, a) = \text{mean}(r|s, a), \tag{367}$$

where $\#(s, a, s')$ is the number of occurrences of (s, a, s') in our data, $\#(s, a)$ is the number of occurrences of (s, a), and $\text{mean}(r|s, a)$ is the average value of r in the samples where action a is taken in state s.

Once we have our estimates, $\hat{P}$ and $\hat{R}$, we could use dynamic programming evaluation methods to solve for what v^π would be if these were the true transition function and reward function. An interesting question is then: how does this value function estimate compare to what TD would produce if it were run on the same data over and over until convergence? Perhaps surprisingly, TD converges to exactly this same value function estimate (if every state is observed at least once or more). So, one might view TD as being an efficient way to compute the value function that would result if we built the maximum likelihood model and then solved for the value function for π.

In practice, TD will be far more useful. Notice that estimating the model requires storing at least $|\mathcal{S}|^2|\mathcal{A}|$ numbers, while TD only requires storing $|\mathcal{S}|$ numbers. Also, estimating P (and R) is difficult when the states are continuous, while estimating value functions using function approximators (like neural networks) is straightforward. This is because P is a distribution, and so estimating P is more than just regression—it requires density estimation.

10 Sarsa: Using TD for Control

Idea: We can use TD to estimate q^π, and simultaneously change π to be (nearly) greedy with respect to q^π. First, we must determine how to use TD to estimate the action-value function rather than the state-value function. The tabular TD update for q given a transition (s, a, r, s', a') is:

$$q(s, a) \leftarrow q(s, a) + \alpha(r + \gamma q(s', a') - q(s, a)), \tag{368}$$

and the TD update for q_w using arbitrary function approximation is:

$$w \leftarrow w + \alpha(r + \gamma q_w(s', a') - q_w(s, a))\frac{\partial q_w(s, a)}{\partial w}. \tag{369}$$

We refer to the term $r + \gamma q(s', a') - q(s, a)$ as the TD error. However, if someone refers to the TD error, they typically mean the TD error using the state-value function.

In terms of the TD error (using the action value function), and using random variables for states, actions, and rewards, we can write the TD update using arbitrary function approximation as:

$$\delta_t = R_t + \gamma q_w(S_{t+1}, A_{t+1}) - q_w(S_t, A_t) \tag{370}$$

$$w_{t+1} \leftarrow w_t + \alpha \delta_t \frac{\partial q_w(S_t, A_t)}{\partial w}. \tag{371}$$

One can view the TD update for the action-value function as being equivalent to the TD update for the state-value function on a different MDP where the state is augmented to include the action chosen according to the policy being evaluated. That is, consider an MDP M. We can construct a new MDP, M', the states of which are $x = (s, a)$, where s is a state in M and a is an action in M. The transition function for M' causes s to transition as in M, and selects actions a according to the policy, π, that is to be evaluated. The actions in M' are irrelevant—we assume that $|\mathcal{A}| = 1$ so that there is only one action. If we apply TD to estimate $v(x)$ for this new MDP (there is only one policy, so we omit the policy-superscript), we obtain:

$$v(x) = \mathbf{E}[G_t | X_t = x] \tag{372}$$

$$[\text{for } M] = \mathbf{E}[G_t | S_t = s, A_t = a] \tag{373}$$

$$[\text{for } M] = q^\pi(s, a). \tag{374}$$

where X_t is the state of M' at time t. Furthermore, writing out the TD update for M' in terms of states, x, we obtain the TD update for q in (368). Hence, applying TD to learn the action-value function is equivalent to applying TD to learn the state-value function for a different MDP, and thus it inherits exactly the same convergence properties.

We can now use the TD algorithm to estimate q^π, and we can then act greedily with respect to q^π. This can be viewed as a sort of approximate form of value iteration, or a generalized policy iteration algorithm. This algorithm is called *Sarsa* because the data used for an update is (s, a, r, s', a'). Pseudocode for tabular Sarsa is presented in Algorithm 14, and Algorithm 15 presents Sarsa using arbitrary function approximation. Note: you do not need to store an estimate for $q(s_\infty, a)$—we know that it is zero, and there is no need to learn this value.

Algorithm 14: Tabular Sarsa

Initialize $q(s, a)$ arbitrarily;
for *each episode* **do**
  $s \sim d_0$;
  Choose a from s using a policy derived from q (e.g., ϵ-greedy or softmax);
  for *each time step, until s is the terminal absorbing state* **do**
    Take action a and observe r and s';
    Choose a' from s' using a policy derived from q;
    $q(s, a) \leftarrow q(s, a) + \alpha(r + \gamma q(s', a') - q(s, a))$;
    $s \leftarrow s'$;
    $a \leftarrow a'$;

Algorithm 15: Sarsa

Initialize w arbitrarily;
for *each episode* **do**
  $s \sim d_0$;
  Choose a from s using a policy derived from q (e.g., ϵ-greedy or softmax);
  for *each time step, until s is the terminal absorbing state* **do**
    Take action a and observe r and s';
    Choose a' from s' using a policy derived from q;
    $w \leftarrow w + \alpha(r + \gamma q_w(s', a') - q_w(s, a))\frac{\partial q_w(s,a)}{\partial w}$;
    $s \leftarrow s'$;
    $a \leftarrow a'$;

———End of Lecture 16, October 29, 2019———

CMPSCI 687 Pop Quiz 5
October 31, 2019

1. What is the TD-error?

$$\delta_t = R_t + \gamma v(S_{t+1}) - v(S_t), \tag{375}$$

or

$$\delta = r + \gamma v(s') - v(s). \tag{376}$$

We will also take the q-forms, or the above using v^π rather than v.

2. What is the Bellman equation?

For all $s \in \mathcal{S}$, $v^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^\pi(s'))$.

3. What is the Bellman optimality equation?

For all $s \in \mathcal{S}$, $v^\star(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^\star(s'))$.

For Sarsa to be guaranteed to converge almost surely to the optimal action-value function, we require the normal assumptions (finite states, finite actions, bounded rewards, $\gamma \in [0, 1)$, step sizes decayed appropriately), as well as two additional assumptions. First, all state action pairs must be visited infinitely often. Second, the policy must converge in the limit to a greedy policy (e.g., $\epsilon_t = \frac{1}{t}$). These two additional assumptions are sometimes called the GLIE assumption: **g**reedy in the **l**imit with **i**nfinite **e**xploration.

Question 28. *What happens if actions are chosen greedily with respect to q rather than nearly greedily?*

Answer 28. *The agent can get stuck assuming that some actions are worse than the one it is currently taking. It will not retry these other actions, and so it cannot learn that these other actions are actually better.*

Question 29. *What happens if the q estimate is initialized optimistically (too large)? What if it is optimized pessimistically? What if $\epsilon = 0$ in these two cases?*

Answer 29. *Optimistic initialization results in exploration. The agent tries one action and finds that it is worse than expected. The next time it visits the same state, it will try a different action. The estimates of action values slowly (assuming a small step size) come down to their correct values. Often Sarsa using $\epsilon = 0$ can work quite well when the value function is initialized optimistically. The opposite happens when the value function is pessimistic—the agent fixates on the first action it chose and explores little. In general, you are encouraged to use optimistic initialization of the value function. However, do not go overboard—you should try to keep your initialization close to the magnitude you expect of the true value function (do not initialize the q-function to 10,000 for all s and a for the 687-Gridworld.*

We refer to Sarsa as an *on-policy* algorithm. This is because it estimates the q-function for the current policy at each time step. Next we will consider a similar algorithm that estimates the q-function for a policy that differs from the one currently being executed.

——————End of Lecture 14, October 30, 2018——————

11 Q-Learning: Off-Policy TD-Control

While the Sarsa update can be viewed as changing the Bellman equation into an update rule, Q-learning can be viewed as changing the Bellman optimality equation into an update rule. The Q-learning update based on a transition (s, a, r, s') is (Watkins, 1989):

$$q(s,a) = q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} q(s',a') - q(s,a)), \tag{377}$$

or with arbitrary function approximation:

$$w = w + \alpha(r + \gamma \max_{a' \in \mathcal{A}} q_w(s',a') - q_w(s,a)) \frac{\partial q_w(s,a)}{\partial w}. \tag{378}$$

Thus, whereas Sarsa changes q towards an estimate of q^π at each step (where π is the policy generating actions), Q-learning changes q towards an estimate of q^* at each step, regardless of which policy is used to generate actions. In this sense it is *off-policy*—it estimates q^* regardless of the policy used to generate data. Notice also that Q-learning can update as soon as S_{t+1} is sampled—before A_{t+1} is sampled, while Sarsa must wait until A_{t+1} was sampled. Q-learning converges to q^* under the standard assumptions if all (s, a) pairs are seen infinitely often. This means that Q-learning does not require the GLIE assumption—the sampling policy does not need to become greedy in the limit.

Because Q-learning does not require A_{t+1} to update, its pseudocode is not just Sarsa with a modified q-update. Pseudocode for Q-learning is provided in Algorithms 16 and 17

Algorithm 16: Tabular Q-Learning

Initialize $q(s,a)$ arbitrarily;
for *each episode* **do**
  $s \sim d_0$;
  for *each time step, until s is the terminal absorbing state* **do**
    Choose a from s using a policy derived from q;
    Take action a and observe r and s';
    $q(s,a) \leftarrow q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} q(s',a') - q(s,a))$;
    $s \leftarrow s'$;

Algorithm 17: Q-Learning

Initialize w arbitrarily;
for *each episode* **do**
  $s \sim d_0$;
  for *each time step, until s is the terminal absorbing state* **do**
    Choose a from s using a policy derived from q;
    Take action a and observe r and s';
    $w \leftarrow w + \alpha(r + \gamma \max_{a' \in \mathcal{A}} q_w(s',a') - q_w(s,a)) \frac{\partial q_w(s,a)}{\partial w}$;
    $s \leftarrow s'$;

The convergence properties of TD, Sasra, and Q-Learning are presented in Table 1.

	TD	**Sarsa**	**Q-learning**
Tabular	**Converges to v^π** (**Tsitsiklis and Van Roy, 1997**)	**Converges to q^*** (**Singh et al., 2000**)	**Converges to q^*** (**Watkins and Dayan, 1992**) (**Tsitsiklis, 1994**)
Linear	**Converges to a policy "close" to v^π** (**Tsitsiklis and Van Roy, 1997**)	**Converges** (**Perkins and Precup, 2003**)	**Can diverge** (**Wiering, 2004; Baird, 1995**)
Non-Linear	Can diverge	Can diverge	Can Diverge

Table 1: Convergence properties of TD, Sarsa, and Q-learning. For complete details (including types of convergence and necessary assumptions), see the provided references. These references are not the first proofs.

———————End of Lecture 17, October 31, 2019———————

CMPSCI 687 Homework 4

Due November 14, 2019, 11:55pm Eastern Time

Instructions: Collaboration is not allowed on any part of this assignment. Submissions must be typed (hand written and scanned submissions will not be accepted). You must use LaTeX. The assignment should be submitted as five documents: a .pdf with your written answers, two .hpp files, and two .cpp files as described in the programming portion.

Programming (75 Points Total)

In this assignment, you will implement Sarsa and Q-learning, and will apply them to a gridworld (not the one from the course notes), mountain car, acrobot, and cart-pole. Begin with the source code provided here (see the previous assignments for instructions regarding opening the project). Look at main.cpp, starting with the function main. Look through how this code functions: it applies Sarsa and Q-learning to various MDPs in sequence. Hyperparameters (not good ones!) are specified for each environment in main.cpp.[9] The code for Sarsa should be in Sarsa.hpp (a header file, that defines the Sarsa class) and Sarsa.cpp (the source file, that includes the actual code for all of the functions that a Sarsa object requires). Similarly the code for Q-learning is split across QLearning.hpp and QLearning.cpp. You should fill code into Sarsa.hpp, Sarsa.cpp, QLearning.hpp, and QLearning.cpp, and these four files are the four that you should submit with your assignment.

To be clear, your edits should be: 1) changing the hyperparameters specified in main.cpp (you do not have to submit these, but will report hyper-parameter values in your write-up), 2) adding code to the train function in QLearning.cpp (you may change QLearning.hpp and other functions in QLearning.cpp, but this is not necessary), 3) adding code to Sarsa.hpp (private member variables) and Sarsa.cpp (likely all of the functions except for "getAction" will have code that you add).

After reading through main.cpp to see what it does, look through Gridworld.hpp and Gridworld.cpp. Gridworld.hpp and Gridworld.cpp have been commented more heavily than the files for the other MDPs. These provide an example of a class in C++. The .hpp file contains the definition of the Gridworld object, and the .cpp file implements the functions that it requires. This also shows how the environments all work. Notice, for example, that the getState() function normalizes the state for you – it returns the state as a vector, each element of which is in the interval $[0, 1]$. Notice also that this code is set up to work well with linear function approximation, as the state is a vector of floating

[9] For this assignment, you may view the iOrder and dOrder hyperparameters as both being the order of the Fourier basis, and you may always set them to the same value.

point numbers (not yet features!) that can be provided to the FourierBasis class to convert to features.

Now that you've read through main.cpp and Gridworld.cpp, look at QLearning.hpp and QLearning.cpp. QLearning.hpp and QLearning.cpp have been commented more heavily than the files for Sarsa. Most of this class is implemented for you. The "train" function has not been implemented fully – you must fill this in. Notice some useful functions have been provided in MathUtils.hpp, like "dot". Also, note that this time we are not using the Eigen library. Don't be afraid to use for loops though, as these are very efficient in C++. The computational bottleneck in this code is usually computing the cosines in the FourierBasis object. This is why we compute and store the features for state s' in an (s, a, r, s') tuple, so that we can re-use them at the next iteration for state s. We could be even more efficient by not recomputing features whenever the agent is asked for an action (right now, QLearning will compute features for state s twice, once in the train function and once in the getAction function). For this assignment, this inefficiency is ok.

Once you have implemented the train function in QLearning.cpp, try setting the hyperparemeters in main.cpp to get results similar to those in the provided file "plots.xlsx". If, after running your code, you copy the contents of the other .csv files over the entries in plots.xlsx, it should update to show the plots we want. You are welcome to use your own system (e.g., write your own python code) to make plots from the output .xlsx files. Hint: For both Q-Learning and Sarsa, set $q(s', a') = 0$ when computing the TD-error if s' is a terminal state, since we know this action-value and therefore do not need to approximate it.

Next, look at Sarsa.hpp and Sarsa.cpp. These are left more empty for you to fill in. Importantly, we're making this harder for you that just putting in the pseudocode. Look back at Section 3.1. The pseudocode there works well for Q-learning, but not as well for Sarsa, since Sarsa requires the action a' to update. Notice that main.cpp implements the pseudocode from Section 3.1. So, you must write Sarsa in a way that works with this setup. Hint: you will want the agent to have some memory, perhaps remembering which states, actions, and/or rewards it saw previously.

Point allocations for this assignment will be determined at the time of grading, based on which mistakes and issues are common.

1. Describe the process of implementing Q-Learning. Did everything work immediately? Did you have bugs that you had to fix? What were they?

2. Describe the process of implementing Sarsa. Did everything work immediately? Did you have bugs that you had to fix? What were they?

3. Describe the process of optimizing the hyperparameters for Q-Learning for MountainCar, and report the final hyperparameters that you found.

4. Describe the process of optimizing the hyperparameters for Q-Learning for CartPole, and report the final hyperparameters that you found.

5. Describe the process of optimizing the hyperparameters for Q-Learning for Acrobot, and report the final hyperparameters that you found.

6. Describe the process of optimizing the hyperparameters for Q-Learning for Gridworld, and report the final hyperparameters that you found.

7. Describe the process of optimizing the hyperparameters for Sarsa for MountainCar, and report the final hyperparameters that you found.

8. Describe the process of optimizing the hyperparameters for Sarsa for CartPole, and report the final hyperparameters that you found.

9. Describe the process of optimizing the hyperparameters for Sarsa for Acrobot, and report the final hyperparameters that you found.

10. Describe the process of optimizing the hyperparameters for Sarsa for Gridworld, and report the final hyperparameters that you found.

11. Provide four plots, one per environment, showing the learning curves for Sarsa and Q-learning with the best hyperparameters that you found. Keep the number of trials, number of episodes, and maxEpisodeLength terms from the provided main.cpp. Include error bars showing one standard deviation. These plots can be created using any plotting software of your choice.

12. Compare your experiences with Q-Learning and Sarsa to your experiences with BBO. Which did you find easier to get working? Which algorithms learned fastest on Cart-Pole (in HW2, you implemented BOO algorithms for Cart-Pole)?

13. Be sure to submit your QLearning.hpp (even if it is unchanged, as recommended), QLearning.cpp, Sarsa.hpp, and Sarsa.cpp files with your write-up.

Note: This code is written to be relatively simple, as many of you are new to C++. This does not represent best coding practices (e.g., we could make use of subclasses).

12 TD(λ)

Notice that, like dynamic programming policy evaluation, TD is slow. Consider using TD to estimate the value function for a policy on 687-Gridworld, starting with initial estimates $v(s) = 0$, and if the first episode happens to reach the terminal state without entering the water state. After his first episode, the only state with non-zero value estimate will be the state that transitioned to the goal. In a sense, the reward at the goal has only propagated backwards a single time step. By contrast, if we updated using the Monte Carlo target, every state along the path to the goal would have had its value updated.

As an intuitive example, imagine that you received an extra \$1,000 in your paycheck one week. If this was unexpected, you might get a positive TD-error. The Sarsa and TD algorithms attribute this TD-error to the most recent state and action: they declare that whatever actions you took just before receiving the check were responsible for the TD-error. This seems a bit absurd: it was likely a combination of actions over the past week or two that resulted in this TD-error. The Monte Carlo algorithm has a similar flaw: if $\gamma \approx 1$, it will assign credit to the states and actions from the distant past. That is, it will conclude that the action value for eating a sandwich five years before should be increased.

In this section we consider trying to find a mix between Monte Carlo Methods and TD methods to try to get the best of both of these approaches. This algorithm, called TD(λ) assigns a "credit" to each state or state-action pair. This credit is discounted over time, and the updates to states are weighted by this credit. We will present this algorithm from two different points of view: the forwards view and the backwards view, and we will will show that these two views are *approximately* equivalent.

The key to mixing Monte Carlo methods with temporal difference methods is the *n-step return*. We sometimes refer to n-step returns as i-step returns. The n-step return is a target that could be used in place of the Monte Carlo target or TD target. Formally, the n-step return is:

$$G_t^{(n)} = \left(\sum_{k=0}^{n-1} \gamma^k R_{t+k} \right) + \gamma^n v(S_{t+n}). \tag{379}$$

Notice that $G_t^{(1)}$ is the TD target, and is sometimes called G_t^{TD}, or the *TD return*, and that $G_t^{(\infty)}$ is the Monte Carlo return, G_t, and sometimes also called G_t^{MC}. Notice that longer returns (larger n) results in higher variance in the target, but lower bias, as discussed previously. We might try to select a value for n that works well for our problem. Intuitively, the n-step return assigns credit to all of the n most recent states (to see this, consider again what would happen when running TD on 687-Gridworld starting with the initial value function equal to zero everywhere, but using n-step returns rather than Monte Carlo returns or TD returns).

Instead of simply selecting one n, we will take a weighted average of all of the different n-step returns. We call this new return a *complex return* because it combines different length returns. We also choose a weighting that depends on a parameter $\lambda \in [0,1]$. We refer to this weighted return as the λ-return, and define it as:

$$G_t^\lambda \coloneqq (1-\lambda)\sum_{i=0}^\infty \lambda^i G_t^{(i+1)}. \tag{380}$$

Notice that if $\lambda = 0$ the λ-return is the TD return. In the limit as $\lambda \to 1$, the λ-return is the Monte Carlo return. When $\lambda = 1$, G_t^λ as defined above is not necessarily defined, since it could become infinity times zero. Hence we explicitly re-define G_t^λ to be the limit as $\lambda \to 1$, i.e., the Monte Carlo return.

To better understand what the λ-return is doing, consider the weights that would be placed on the different length returns for an MDP with finite horizon, $L = 10$. The weight placed on the 1-step return would be $(1-\lambda)$, the weight on the 2-step return would be $(1-\lambda)\lambda$, the weight on the 3-step return would be $(1-\lambda)\lambda^2, \dots,$ the weight on the 10-step return would be $(1-\lambda)\lambda^9$, the weight on the 11-step return would be $(1-\lambda)\lambda^{10}$, etc. Notice that the 10-step return is the Monte Carlo return, since the horizon is $L = 10$, which means that $S_{10} = s_\infty$ and so $R_t = 0$ for $t > 10$. Shorter returns may also be the Monte Carlo return if the agent happened to enter s_∞ earlier, but we know that at some point, before the L-step return, the return from any state will be the Monte Carlo return. Hence, the λ-return can be written as:

$$\begin{aligned}
G_t^\lambda \coloneqq& (1-\lambda)\sum_{i=0}^\infty \lambda^i G_t^{(i+1)} && (381)\\
=& (1-\lambda)\left(\sum_{i=0}^{L-2} \lambda^i G_t^{(i+1)}\right) + (1-\lambda)\sum_{i=L-1}^\infty \lambda^i G_t^{\text{MC}} && (382)\\
=& (1-\lambda)\left(\sum_{i=0}^{L-2} \lambda^i G_t^{(i+1)}\right) + (1-\lambda)\left(\sum_{i=L-1}^\infty \lambda^i\right) G_t^{\text{MC}}. && (383)
\end{aligned}$$

That is, all of the weight placed on returns of length at least L is placed on the Monte-Carlo return. So, although the weights are generally decreasing as the return length increases, a large weight is often placed on the Monte Carlo return. Furthermore, since the first weight is $(1-\lambda)$, as $\lambda \to 1$ the sum of the first L weights decreases, and so the weight on the Monte Carlo term increases.

A common question is: why this geometric series of weights? Is the choice of weighting used in the λ-return in some way a statistically principled choice? The original reason for this weighting scheme is that it will make our subsequent math work out (more specifically, it is not clear how to make a "backwards view" with other weighting schemes—in the next lecture we will describe what this backwards view is). Konidaris et al. (2011a) investigated conditions under which the λ-return is statistically principled. Below we will review their findings (not the alternative to the λ-return that they propose, but their analysis of the λ-return). These findings show a set of conditions under which the λ-return could be derived as a principled estimator of $v^\pi(s)$. Other conditions may exist under which the λ-return is a reasonable weighting scheme, but this is the only example that I am aware of today.

A common point of confusion here is about whether the returns come from the same episode. They do. We are consider an agent that is currently at S_t, the current time step is t, and we are deciding what target the agent should use—what value it should change its estimate of $v^\pi(S_t)$ to be closer to. For now, we are ignoring the fact that the agent must wait until the end of an episode to compute some of these longer returns, and asking: if we had all of the data from now until the end of the episode, what should our target be? One answer is the TD target, $G_t^{(1)}$, while another is the Monte-Carlo target, G_t. The one we're considering here is the λ-return, which blends together targets between the Monte-Carlo and TD targets. Also, note that each of these targets is an estimator of $v^\pi(S_t)$.

Theorem 10. *If*

1. *The i-step returns are all statistically independent,*
2. *The i-step returns are all normally distributed,*
3. *The variance of the i-step returns grows with i according to:* $\operatorname{Var}(G_t^{(i)}) = \beta/\lambda^i$, *for some constant β,*
4. $\mathbf{E}[G_t^{(i)}] = v^\pi(S_t)$ *for all i, i.e., the i-step returns are all unbiased estimators of $v^\pi(S_t)$,*

then the maximum likelihood estimator of $v^\pi(S_t)$ is the λ-return, G_t^λ.

Proof. The likelihood that $v^\pi(S_t) = x$ given the estimators $G_t^{(1)}, G_t^{(2)}, G_t^{(3)}, \dots$ is

$$\mathcal{L}(x|G_t^{(1)}, G_t^{(2)}, G_t^{(3)}, \dots) = \Pr\left(G_t^{(1)}, G_t^{(2)}, G_t^{(3)}, \dots \middle| v^\pi(S_t) = x\right) \tag{384}$$

$$= \prod_{i=1}^{\infty} \Pr(G_t^{(i)} | v^\pi(S_t) = x), \tag{385}$$

by the assumption that the different length returns are independent, and where in this proof, L denotes the *likelihood function*, not the horizon of an MDP. To find the maximum likelihood estimator, we must search for the value of x that maximizes the likelihood:

$$\arg\max_{x\in\mathbb{R}} \mathcal{L}(x|G_t^{(1)}, G_t^{(2)}, G_t^{(3)}, \dots) = \arg\max_{x\in\mathbb{R}} \prod_{i=1}^{\infty} \Pr(G_t^{(i)}|v^\pi(S_t)=x) \tag{386}$$

$$= \arg\max_{x\in\mathbb{R}} \ln\left(\prod_{i=1}^{\infty} \Pr(G_t^{(i)}\middle| v^\pi(S_t)=x)\right) \tag{387}$$

$$= \arg\max_{x\in\mathbb{R}} \sum_{i=1}^{\infty} \ln\left(\Pr(G_t^{(i)}\middle| v^\pi(S_t)=x)\right) \tag{388}$$

$$\overset{\text{(a)}}{=} \arg\max_{x\in\mathbb{R}} \sum_{i=1}^{\infty} \ln\left(\frac{1}{\sqrt{2\pi\beta/\lambda^i}} \exp\frac{-(G_t^{(i)}-x)^2}{2\beta/\lambda^i}\right) \tag{389}$$

$$= \arg\max_{x\in\mathbb{R}} \sum_{i=1}^{\infty} \ln\left(\frac{1}{\sqrt{2\pi\beta/\lambda^i}}\right) + \ln\left(\exp\frac{-(G_t^{(i)}-x)^2}{2\beta/\lambda^i}\right) \tag{390}$$

$$\overset{\text{(b)}}{=} \arg\max_{x\in\mathbb{R}} \sum_{i=1}^{\infty} \ln\left(\exp\frac{-(G_t^{(i)}-x)^2}{2\beta/\lambda^i}\right) \tag{391}$$

$$= \arg\max_{x\in\mathbb{R}} \sum_{i=1}^{\infty} \frac{-(G_t^{(i)}-x)^2}{2\beta/\lambda^i}, \tag{392}$$

where **(a)** comes from the assumptions that each $G_t^{(i)}$ is normally distributed with mean $v^\pi(S_t)$ and variance β/λ^i and **(b)** holds because the dropped term is not a function of x, and so does not impact the result (due to the $\arg\max_{x\in\mathbb{R}}$).

Solving for the critical points, we have that any critical point must satisfy:

$$\frac{\partial}{\partial x}\sum_{i=1}^{\infty}\frac{-(G_t^{(i)}-x)^2}{2\beta/\lambda^i}=0 \tag{393}$$

$$\iff \sum_{i=1}^{\infty}\frac{\partial}{\partial x}\frac{-\lambda^i(G_t^{(i)}-x)^2}{2\beta}=0 \tag{394}$$

$$\iff \sum_{i=1}^{\infty}\frac{\lambda^i(G_t^{(i)}-x)}{\beta}=0 \tag{395}$$

$$\iff \sum_{i=1}^{\infty}\lambda^i G_t^{(i)}=\sum_{i=1}^{\infty}\lambda^i x \tag{396}$$

$$\iff \sum_{i=1}^{\infty}\lambda^i G_t^{(i)}=\frac{\lambda}{1-\lambda}x \tag{397}$$

$$\iff x=\frac{1-\lambda}{\lambda}\sum_{i=1}^{\infty}\lambda^i G_t^{(i)} \tag{398}$$

$$\iff x=\frac{1-\lambda}{\lambda}\sum_{i=0}^{\infty}\lambda^{i+1} G_t^{(i+1)} \tag{399}$$

$$\iff x=(1-\lambda)\sum_{i=0}^{\infty}\lambda^{i} G_t^{(i+1)} \tag{400}$$

$$\iff x=G_t^{\lambda}. \tag{401}$$

□

Notice that the conditions required by Theorem 10 are egregious. The i-step returns are *not* statistically independent (the 98-step and 99-step returns with small γ are nearly identical), they often are not normally distributed, the variance does not grow proportional to $1/\lambda^i$ (particularly for longer returns, the discounting makes returns almost identical, and thus they have almost identical variance), and only the Monte-Carlo return is unbiased (the bias of the TD return was one of the main motivations for creating the λ-return in the first place!). So, right now you might be thinking "this seems like a very bad estimator for us to use!" Perhaps, and there is research into producing better estimators (Konidaris et al., 2011a; Thomas et al., 2015a; Thomas and Brunskill, 2016).[10] However, for now we will proceed using the λ-return. Note that the λ-return *is* the current standard in reinforcement learning research, despite its lack of a principled derivation.

[10]Because it is not clear how to create backwards views for these more advanced estimators, they are not practical for use in place of the λ-return. Making these estimators practical would be a significant advancement. Since they cannot replace the λ-return (because they do not have a backwards form), these latter papers considered a setting where they are applicable: off-policy policy evaluation. Hence, the latter two papers begin targeting this alternate problem, but their real underlying motivation was an effort to improve and replace the λ-return.

12.1 λ-Return Algorithm

We can use the λ-return as a target to obtain a policy evaluation algorithm. The resulting update is:

$$v(S_t) \leftarrow v(S_t) + \alpha(G_t^\lambda - v(S_t)). \tag{402}$$

This algorithm is called the *forwards view* because, when updating $v(S_t)$, we look forward in time to see what states and rewards occur in the future. Also, after updating $v(s)$, we never look at $v(s)$ again until it is visited another time. The drawback of the forward view is that we must wait until the end of an episode in order to compute the update to $v(S_t)$. We therefore call this algorithm *offline*, as opposed to *online* algorithms, which update at each time step.

13 Backwards View of TD(λ)

In the backwards view, at time t the agent looks back to all of the states that have occurred up until time t, and determines how these previous states should be updated based on the newly observed state and reward. We store an additional variable for each state, called an *eligibility trace*. We write $e_t(s)$ to denote the eligibility of state s at time t. We sometimes refer to eligibility traces as e-traces. An e-trace quantifies how much $v(s)$ should be updated if there is a TD error at the current time step, t. At each time step, all e-traces will be decayed by $\gamma\lambda$, and the e-trace for the current state is incremented. That is:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq S_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{otherwise.} \end{cases} \tag{403}$$

This type of e-trace is called an *accumulating trace* because the traces can accumulate to be larger than one. Other alternatives exist, like replacing traces, wherein the eligibility of the current state is set to one rather than incremented by one. We will focus on accumulating traces.

Note: the eligibility traces start equal to zero for all states *and should be reset to zero at the start of every episode.*

The TD(λ) algorithm updates all states at each time step in proportion to their eligibility:

$$\delta_t = R_t + \gamma v(S_{t+1}) - v(S_t) \tag{404}$$
$$\forall s \in \mathcal{S},\ e(s) = \gamma\lambda e(s) \tag{405}$$
$$e(S_t) = e(S_t) + 1 \tag{406}$$
$$\forall s \in \mathcal{S},\ v(s) = v(s) + \alpha\delta_t e(s). \tag{407}$$

At each time step, this algorithm looks backwards and asks "which states should have their values updated due to the current TD error?" Notice that if $\lambda = 0$, this algorithm is clearly identical to TD. Perhaps less obviously, if $\lambda = 1$, this algorithm is equivalent to the Monte Carlo algorithm in (327).

———————End of Lecture 18, November 5, 2019———————

The forwards and backwards updates are (approximately) equivalent. That is, if we start with the same value function, after running a complete episode using the update in (402) and the updates in (404), the resulting value function estimates will be (approximately) equal. To see this, we begin by establishing some notation.

First, recall that

$$\delta_t = R_t + \gamma v_t(S_{t+1}) - v_t(S_t), \tag{408}$$

where v_t denotes the value function estimate at time t. Let $\mathcal{I}_{s,S_t} = 1$ if $S_t = s$ and $\mathcal{I}_{s,S_t} = 0$ otherwise. With this indicator function, we can write an expression for the eligibility trace at time t that is *not* recurrent:

$$e_t(s) = \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathcal{I}_{s,S_k}. \tag{409}$$

Unlike (403), which computed the eligiblities recurrently, this equation looks back from time t at all previous time steps, k, and adds the contribution to the e-trace at time t that is due to the state at time k. If the state at time k is not s, then there is no contribution from time k to the eligibility of state s at time t. If the state at time k was s, then at time t this contributes $(\gamma\lambda)^{t-k}$ to the eligibility of state s at time t.

Next, let $\Delta v_t^F(s)$ denote the update at time t to $v_t(s)$ according to the forward view. That is,

$$\Delta v_t^F(s) = \alpha(G_t^\lambda - v_t(S_t)), \tag{410}$$

if $S_t = s$, and $\Delta v_t^F(s) = 0$ otherwise. We do not express this by including a $\mathcal{I}_{s,S_t}$ term on the right side in order to simplify the use of this term later. Similarly, let $\Delta v_t^B(s)$ denote the update at time t to $v_t(s)$ according to the backwards view:

$$\Delta v_t^B(s) = \alpha \delta_t e_t(s). \tag{411}$$

In Theorem 11 we show that the forwards and backwards updates result in the roughly same change to the value function at the end of an episode. After the proof we discuss the step that makes this equivalence approximate.

Theorem 11. *For all $s \in \mathcal{S}$,*

$$\sum_{t=0}^{L} \Delta v_t^B(s) \approx \sum_{t=0}^{L} \Delta v_t^F(s) \mathcal{I}_{s,S_t}. \tag{412}$$

Proof. We begin with the left side:

$$\sum_{t=0}^{L} \Delta v_t^B(s) \overset{\text{(a)}}{=} \sum_{t=0}^{L} \alpha \delta_t e_t(s) \tag{413}$$

$$\overset{\text{(b)}}{=} \sum_{t=0}^{L} \alpha \delta_t \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathcal{I}_{s,S_k} \tag{414}$$

$$\overset{\text{(c)}}{=} \sum_{t=0}^{L} \alpha \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathcal{I}_{s,S_k} \delta_t \tag{415}$$

$$\overset{\text{(d)}}{=} \sum_{k=0}^{L} \alpha \sum_{t=0}^{k} (\gamma\lambda)^{k-t} \mathcal{I}_{s,S_t} \delta_k, \tag{416}$$

where **(a)** comes from (411), **(b)** comes from (409), **(c)** comes from moving the δ_t term inside the sum over k, and **(d)** comes from swapping the variable names t and k (this is a name change only - no terms have changed). Notice that $\sum_{i=0}^{n} \sum_{j=0}^{i} f(i,j) = \sum_{j=0}^{n} \sum_{i=j}^{n} f(i,j)$, since all of the same pairs of i and j are included. Using this property, we reverse the order of the sums to obtain:

$$\sum_{t=0}^{L} \Delta v_t^B(s) = \sum_{t=0}^{L} \alpha \sum_{k=t}^{L} (\gamma\lambda)^{k-t} \mathcal{I}_{s,S_t} \delta_k \tag{417}$$

$$\overset{\text{(a)}}{=} \sum_{t=0}^{L} \alpha \mathcal{I}_{s,S_t} \sum_{k=t}^{L} (\gamma\lambda)^{k-t} \delta_k. \tag{418}$$

where **(a)** comes from moving the $\mathcal{I}_{s,S_t}$ term outside of the sum over k, since it does not depend on k. Thus, on one line, we have that:

$$\sum_{t=0}^{L} \Delta v_t^B(s) = \sum_{t=0}^{L} \alpha \mathcal{I}_{s,S_t} \sum_{k=t}^{L} (\gamma\lambda)^{k-t} \delta_k. \tag{419}$$

We now turn to the right hand side of (412), and consider the update at a single time step:

$$\Delta v_t^F(S_t) = \alpha(G_t^\lambda - v_t(S_t)). \tag{420}$$

Dividing both sides be α, we obtain:

$$\frac{1}{\alpha} \Delta v_t^F(S_t) = G_t^\lambda - v_t(S_t) \tag{421}$$

$$= \quad -v_t(S_t) + (1-\lambda)\lambda^0(R_t + \gamma v_t(S_{t+1})) \tag{422}$$

$$+ (1-\lambda)\lambda^1(R_t + \gamma R_{t+1} \quad + \gamma^2 v_t(S_{t+2})) \tag{423}$$

$$+ (1-\lambda)\lambda^1(R_t + \gamma R_{t+1} \quad + \gamma^2 R_{t+2} \quad + \gamma^3 v_t(S_{t+3})) \tag{424}$$

$$\vdots \tag{425}$$

Consider all of the R_t terms:

$$\sum_{i=0}^{\infty}(1-\lambda)\lambda^i R_t = \frac{1-\lambda}{1-\lambda}R_t \tag{426}$$
$$= R_t. \tag{427}$$

Now consider all of the R_{t+1} terms:

$$\sum_{i=1}^{\infty}(1-\lambda)\lambda^i \gamma R_{t+1} = (1-\lambda)(\gamma\lambda)\sum_{i=0}^{\infty}\lambda^i R_{t+1} \tag{428}$$
$$= (\gamma\lambda)R_{t+1}. \tag{429}$$

In general, all of the R_{t+k} terms combine to $(\gamma\lambda)^k R_{t+k}$. We now rewrite (425), but combining all of the R_t terms, R_{t+1} terms, etc.

$$\frac{1}{\alpha}\Delta v_t^F(S_t) = -v_t(S_t) \tag{430}$$
$$+ R_t \quad + (1-\lambda)\gamma v_t(S_{t+1}) \tag{431}$$
$$+ (\gamma\lambda)R_{t+1} + (1-\lambda)(\gamma\lambda)\gamma v_t(S_{t+2}) \tag{432}$$
$$+ (\gamma\lambda)^2 R_{t+2} + (1-\lambda)(\gamma\lambda)^2\gamma v_t(S_{t+3}) \tag{433}$$
$$\vdots \tag{434}$$

Pulling out a $(\gamma\lambda)^i$ from each row and expanding the $(1-\lambda)$ term, we obtain:

$$\frac{1}{\alpha}\Delta v_t^F(S_t) = -v_t(S_t) \tag{435}$$
$$+ (\gamma\lambda)^0\left(R_t + \gamma v_t(S_{t+1}) - \gamma\lambda v_t(S_{t+1})\right) \tag{436}$$
$$+ (\gamma\lambda)^1\left(R_{t+1} + \gamma v_t(S_{t+2}) - \gamma\lambda v_t(S_{t+2})\right) \tag{437}$$
$$+ (\gamma\lambda)^2\left(R_{t+2} + \gamma v_t(S_{t+3}) - \gamma\lambda v_t(S_{t+3})\right) \tag{438}$$
$$\vdots \tag{439}$$

Shifting the right-most v_t terms all down one line, and plugging the $-v_t(S_t)$ from (435) into (436), we obtain:

$$\frac{1}{\alpha}\Delta v_t^F(S_t) = (\gamma\lambda)^0\left(R_t + \gamma v_t(S_{t+1}) - v_t(S_t)\right) \tag{440}$$
$$+ (\gamma\lambda)^1\left(R_{t+1} + \gamma v_t(S_{t+2}) - v_t(S_{t+1})\right) \tag{441}$$
$$+ (\gamma\lambda)^2\left(R_{t+2} + \gamma v_t(S_{t+3}) - v_t(S_{t+2})\right) \tag{442}$$
$$\vdots \tag{443}$$

Consider the term $R_{t+k} + \gamma v_t(S_{t+k+1}) - v_t(S_{t+k})$. This term resembles the TD-error that occurs k steps in the future from time t, that is δ_{t+k}. However, it is *not* precisely δ_{t+k}, since δ_{t+k} (when computed using the backwards algorithm—the way we defined TD-errors) will use v_{t+k} when computing δ_{t+k}, not v_t. That is, this is the TD-error k steps in the future if we were to use our value function from time t to compute the TD error. If the step size is small, then the change to the value function should not have been significant within an episode, and so $R_{t+k} + \gamma v_t(S_{t+k+1}) - v_t(S_{t+k}) \approx \delta_{t+k}$. Using this approximation, we obtain:

$$\frac{1}{\alpha}\Delta v_t^F(S_t) \approx \sum_{k=0}^{\infty}(\gamma\lambda)^k \delta_{t+k} \tag{444}$$

$$= \sum_{k=t}^{\infty}(\gamma\lambda)^{k-t} \delta_k \tag{445}$$

$$= \sum_{k=t}^{L}(\gamma\lambda)^{k-t} \delta_k, \tag{446}$$

since $\delta_k = 0$ if $k > L$. So, returning to the right side of (412),

$$\sum_{t=0}^{L}\Delta v_t^F(S_t)\mathcal{I}_{s,S_t} \approx \sum_{t=0}^{L}\alpha\sum_{k=t}^{L}(\gamma\lambda)^{k-t}\delta_k\mathcal{I}_{s,S_t} \tag{447}$$

$$= \sum_{t=0}^{L}\alpha\mathcal{I}_{s,S_t}\sum_{k=t}^{L}(\gamma\lambda)^{k-t}\delta_k, \tag{448}$$

which is the same as the left side, as expressed in (419). □

13.1 True Online Temporal Difference Learning

The equivalence of the forwards and backwards views of TD(λ) is only approximate. Seijen and Sutton (2014) showed how the TD(λ) algorithm can be modified so that the modified backwards view is actually equivalent to the forwards view (Van Seijen et al., 2016). This *true online TD*(λ) algorithm is only designed for the tabular and linear function approximation settings—it is not applicable with non-linear function approximation. In practice, true online TD(λ) is more robust to the step size parameter than ordinary TD(λ) (with an improperly tuned step size, α, this can appear to be faster learning). The same goes for true online Sarsa(λ) (the control form of true online TD(λ)).

——————————End of Lecture 19, November 7, 2019——————————

13.2 Sarsa(λ) and Q(λ)

We can use TD(λ) for control, just as we used TD to create the Sarsa and Q-learning algorithms. The resulting algorithms are called Sarsa(λ) and $Q(\lambda)$, respectively. Pseudocode for Sarsa(λ) is provided in Algorithm 18, and pseudocode for $Q(\lambda)$ is provided in Algorithm 19. In both algorithms, e is the vector of eligibility traces—one real-valued trace per weight.

Algorithm 18: Sarsa(λ)

Initialize w arbitrarily;
for *each episode* **do**
  $s \sim d_0$;
  $e \leftarrow 0$;
  Choose a from s using a policy derived from q (e.g., ϵ-greedy or softmax);
  for *each time step, until s is the terminal absorbing state* **do**
    Take action a and observe r and s';
    Choose a' from s' using a policy derived from q;
    $e \leftarrow \gamma\lambda e + \frac{\partial q_w(s,a)}{\partial w}$;
    $\delta \leftarrow r + \gamma q_w(s', a') - q_w(s,a)$;
    $w \leftarrow w + \alpha\delta e$;
    $s \leftarrow s'$;
    $a \leftarrow a'$;

Algorithm 19: $Q(\lambda)$

Initialize w arbitrarily;
for *each episode* **do**
  $s \sim d_0$;
  $e \leftarrow 0$;
  for *each time step, until s is the terminal absorbing state* **do**
    Choose a from s using a policy derived from q;
    Take action a and observe r and s';
    $e \leftarrow \gamma\lambda e + \frac{\partial q_w(s,a)}{\partial w}$;
    $\delta \leftarrow r + \gamma \max_{a' \in \mathcal{A}} q_w(s', a') - q_w(s,a)$;
    $w \leftarrow w + \alpha\delta e$;
    $s \leftarrow s'$;

Question 30. *If we store one weight per state-action pair (i.e., if we use a tabular representation) and always sample actions from a fixed policy, π, that does not depend on q_w, confirm that the Sarsa(λ) algorithm is equivalent to TD(λ) for estimating the action-value function.*

There are other $Q(\lambda)$ variants—particularly ones that use different eligibility traces, like replacing traces. The one that we present here is the most simple, and perhaps the most common. If someone refers to Q-learning, they are typically referring to this variant of the $Q(\lambda)$ algorithm.

14 High-Confidence Policy Improvement

As you've seen from the homework assignments, RL algorithms usually don't work when you first try to use them. It takes a lot of tuning before they produce decent performance. When you are first tuning the algorithm, you are deploying the algorithm to the environment, and it is producing poor performance, even diverging. For real-world applications, this would mean deploying an agent that produces policies that are often worse than an initial policy. That is fine for video games—we can lose as many games of Pacman as we want with little cost. However, that is not fine for real-world applications where deploying a bad policy can be costly or dangerous. So, how then can we deploy RL algorithms to any real-world problem?

Let's begin to answer this by studying a simplified setting: *batch policy improvement*. In this setting we assume that we have collected n trajectories of data from running some current policy (or some sequence of past policies). For simplicity here, let's assume that there is a single current policy, π_b, which we will call the *behavior policy*. From running n trajectories we obtain data $D = \{H_1, H_2, \dots, H_n\}$—a set of n histories. Here each $H_i = (S_0^i, A_0^i, R_0^i, S_1^i, A_1^i, \dots)$. That is, we use superscripts to indicate which episode a state, action, or reward is from. Given the data D, we want to find a new policy π that is as good as possible. That is, we want to maximize $J(\pi)$, where π is computed from the data D.

This batch setting models many real problems, where some current solution is being used, and we would like to use an RL algorithm to improve the policy. What properties would we want of an RL algorithm before we would be comfortable applying it to a real problem? First, we know that designing reward functions is challenging, and so we would want to ensure that the reward function aligns with our goals. For now, let's assume that this is the case. Next, it might be that expected return is not what we want to optimize, since we might care about the variance of returns as well. We can discuss this later, as it is a current area of research (for example, we can get guarantees on the expected return given that we only consider returns less than some constant, but we do not yet know how to get guarantees on the CVaR of returns). For now, let's assume that the expected return does capture the quantity we care about.

In this case, we might be comfortable using an RL algorithm that guaranteed that $J(\pi) \geq J(\pi_b)$. Unfortunately, this is not possible to guarantee. The data that we collected could be a random fluke that causes us to draw incorrect conclusions about the performance of π or π_b—in our gridworld, the agent's "attempt" actions might have just happened to always fail, causing the agent to have incorrect beliefs about what the actions do. The core problem here is that the available data is random. Hence, rather than guaranteeing improvement with certainty, we will guarantee improvement with high probability:

$$\Pr(J(\pi) \geq J(\pi_b)) \geq 1 - \delta, \tag{449}$$

where δ is some small probability that the user of the batch RL algorithm will get to select. The above equation can be confusing—what term is random inside of the probability? Why is this probability not necessarily zero or one depending on which policy is better? The random term here is π, since π is computed from the data D. To make this explicit, let $\mathcal{D}$ be the set of all possible data sets, and let our algorithm be a function $a : \mathcal{D} \to \Pi$. Hence, $a(D)$ is the solution returned by our algorithm when run on data D. We can now write the guarantee that we want as:

$$\Pr(J(a(D)) \geq J(\pi_b)) \geq 1 - \delta, \tag{450}$$

where now it is clear that the source of randomness is the data set, D.

Unfortunately, this too is impossible! What happens if I give you two trajectories from some stochastic MDP. No algorithm can have any hope of giving any high-probability guarantees given so little data. The fix to this is to allow the algorithm to say "I can't do that." More formally, we say that the algorithm returns "No Solution Found" (NSF). In reality, this would correspond to saying: "keep running π_b, because I am not able to improve performance with high-probability given the data that I have." To make this fit with our expression above, we define $J(\text{NSF}) \geq \pi_b$ so that the algorithm can always return NSF.

Finally, to simplify our math, let's replace $J(\pi_b)$ with some user-selected constant. This constant could be a high-confidence upper bound on $J(\pi_b)$, it could be the observed performance of π_b in the past, it could be a 10% improvement on the observed performance of π_b, or it could be 80% of the performance of π_b. This gives our final goal—to create an algorithm a that guarantees that:

$$\Pr(J(a(D)) \geq c) \geq 1 - \delta, \tag{451}$$

where the user of the algorithm selects c and δ.

Note that with the value c here, we must define $J(\text{NSF}) \geq c$ for it to be possible to create such an algorithm a. This is to ensure that, when it has insufficient data (or if it is tasked with the impossible, due to c being larger than the expected return of any policy), the algorithm can return NSF.

Notice also that there is a naive solution: have the algorithm a always output NSF. This is because (451) is a safety constraint, it is not the primary objective. The primary objective remains unchanged: maximize the expected discounted return. (451) is merely a constraint that we must guarantee we satisfy while trying to maximize expected return.

Consider how we might create an algorithm a that satisfies (451). At some point it will consider returning a policy π. However, it must determine whether it has high-confidence that $J(\pi) \geq c$. Forgetting high-confidence for a moment, this means we at least need to be able to estimate $J(\pi)$. Normally we would do this by running the policy π, but here we aren't allowed to do that. That would correspond to running some policy that has no safety guarantees, which could be dangerous or costly. Instead, we need a way to estimate $J(\pi)$ using the data D, generated by running the policy π_b. This is what we will present next. Once we have that method, we will discuss how we can obtain confidence intervals around this estimate, which will eventually allow us to create our algorithm a that satisfies (451).

14.1 Off-Policy Policy Evaluation

The goal in *off-policy policy evaluation* (OPE) is to estimate $J(\pi)$ given data D generated by $\pi_b \neq \pi$. In this area, the policy π is typically denoted by π_e, and called the *evaluation policy.* Recall that π_b is called the behavior policy.

First, notice that we can view policies as distributions over trajectories, H. With slightly sloppy notation, we can write $H \sim \pi_e$ or $H \sim \pi_b$ to denote a trajectory sampled from running π_e or π_b. Also, let $g(H) \coloneqq \sum_{t=0}^{\infty} \gamma^t R_t$. We want to estimate $\mathbf{E}[g(H)|H \sim \pi_e]$ given our data, which contains trajectories $H \sim \pi_b$. We will use an approach called *importance sampling.* First, let's review importance sampling in general (not in the context of RL).

Let p and q be two distributions and f be some function. Our goal is to estimate $\mathbf{E}[f(X)|X \sim p]$ given samples of $X \sim q$. At a high-level, we will take a weighted average of the observed values of $f(x)$. That is, for each sample, x, we ask "would this have been more likely under the distribution p?" If so, we will give it a large weight (a weight bigger than one) to pretend that we saw that sample more often. If not (if the sample would have been more likely under q than under p), we give it a weight smaller than one to pretend that we saw that sample less often.

More precisely, the importance sampling estimator is:

$$\text{IS}(x, p, q) = \frac{p(x)}{q(x)} f(x). \tag{452}$$

Next lecture, we will begin by proving that this estimator is an unbiased estimator—that $\mathbf{E}[\text{IS}(X, p, q)|X \sim q] = \mathbf{E}[f(x)|X \sim p]$. We will then show how to use it in our RL setting.

——————————End of Lecture 20, November 12, 2019——————————

CMPSCI 687 Pop Quiz 6 or 7
November 14, 2019

1. (True or False) The hyperparameter λ scales how much a Sarsa(λ) agent explores.

2. (True or False) The Monte Carlo return typically has higher variance than the TD target.

3. (True or False) Q-learning is an *on-policy* algorithm.

4. (True or False) TD(λ) is a control algorithm, while TD is an evaluation algorithm.

5. (True or False) The difference between Sarsa and Q-learning is that Sarsa learns the state-value function v^π, while Q-learning learns the action value function q^π.

Recall that $\text{supp}(p) = \{x : p(x) > 0\}$. For the importance sampling estimator to be unbiased, we need some assumptions related to the support of p and q. For example:

Theorem 12. *If* $\text{supp}(p) \subseteq \text{supp}(q)$, *then*

$$\mathbf{E}[\text{IS}(X, p, q)|X \sim q] = \mathbf{E}[f(X)|X \sim p]. \tag{453}$$

Proof.

$$\mathbf{E}[\text{IS}(X, p, q)|X \sim q] = \mathbf{E}\left[\frac{p(X)}{q(X)} f(X) \middle| X \sim q\right] \tag{454}$$

$$= \sum_{x \in \text{supp}(q)} q(x) \frac{p(X)}{q(X)} f(X) \tag{455}$$

$$= \sum_{x \in \text{supp}(q)} p(x) f(x) \tag{456}$$

$$= \sum_{x \in \text{supp}(q) \cap \text{supp}(p)} p(x) f(x) + \sum_{x \in \text{supp}(q) \setminus \text{supp}(p)} p(x) f(x). \tag{457}$$

Notice that the second summation only includes values x that are not in $\text{supp}(p)$, meaning samples such that $p(x) = 0$. Hence, this second term is zero. For the first term, notice that by our assumption $\text{supp}(p) \subseteq \text{supp}(q)$. Hence, $\text{supp}(q) \cap \text{supp}(p) = \text{supp}(p)$. So:

$$\mathbf{E}[\text{IS}(X, p, q)|X \sim q] = \sum_{x \in \text{supp}(p)} p(x) f(x) \tag{458}$$

$$= \mathbf{E}[f(X)|X \sim p]. \tag{459}$$

□

Note: if the support assumption is not ensured, one can (in some cases) ensure that the bias is strictly positive or negative (Thomas et al., 2015b).

To apply this for OPE we will use $X \leftarrow H$, $f \leftarrow g$, $p \leftarrow \pi_e$, and $q \leftarrow \pi_b$. Hence, the IS estimator for OPE is:

$$\text{IS}(H, \pi_e, \pi_b) = \frac{\pi_e(H)}{\pi_b(H)} g(H). \tag{460}$$

Since this estimator is unbiased we have from Theorem 12 that if $\text{supp}(\pi_e) \subseteq \text{supp}(\pi_b)$, then

$$\mathbf{E}\left[\text{IS}(H, \pi_e, \pi_b)|H \sim \pi_b\right] = \mathbf{E}[g(H)|H \sim \pi_e] \tag{461}$$

$$= J(\pi_e). \tag{462}$$

That is, the importance sampling estimator is an unbiased estimator of the performance of π_e.

Given our entire dataset D, we define:

$$\text{IS}(D, \pi_e, \pi_b) = \frac{1}{n} \sum_{i=1}^{n} \text{IS}(H_i, \pi_e, \pi_b). \tag{463}$$

That is, the IS estimate from the entire dataset is the average IS estimate from each trajectory.

Although we do not show it here, the importance sampling estimator is also a strongly consistent estimator of $J(\pi_e)$, meaning that in the limit as the amount of data goes to infinity, $\text{IS}(D, \pi_e, \pi_b) \xrightarrow{\text{a.s.}} J(\pi_e)$.

However, how can we compute the importance sampling estimator? It included the term $\pi_e(H)$, which really means $\Pr(H|\pi_e)$. This probably depends on the transition and reward functions, which we assume are unknown. (Precup, 2000) showed that we *can* compute the importance sampling estimator without knowing the transition and reward functions because these terms cancel out when computing the ratio $\pi_e(H)/\pi_b(H)$.

That is:

$$\frac{\pi_e(H)}{\pi_b(H)} = \frac{\Pr(H|\pi_e)}{\Pr(H|\pi_b)} \tag{464}$$

$$= \frac{d_0(S_0)\pi_e(S_0, A_0)P(S_0, A_0, S_1)d_R(S_0, A_0, S_1, R_0)\pi_e(S_1, A_1)P(S_1, A_1, S_2)\ldots}{d_0(S_0)\pi_b(S_0, A_0)P(S_0, A_0, S_1)d_R(S_0, A_0, S_1, R_0)\pi_b(S_1, A_1)P(S_1, A_1, S_2)\ldots}. \tag{465}$$

Notice that all of the terms in the numerator and denominator are the same except for the policy-terms, and so all of the terms but the policy-terms cancel out to give:

$$\frac{\pi_e(H)}{\pi_b(H)} = \frac{\pi_e(S_0, A_0)\pi_e(S_1, A_1)\pi_e(S_2, A_2)\cdots}{\pi_b(S_0, A_0)\pi_b(S_1, A_1)\pi_b(S_2, A_2)\cdots} \tag{466}$$

$$= \prod_{t=0}^{L-1} \frac{\pi_e(S_t, A_t)}{\pi_b(S_t, A_t)}. \tag{467}$$

We refer to the term above as an *importance weight*. Hence, the IS estimator can be written as:

$$\text{IS}(H, \pi_e, \pi_b) = \left(\prod_{t=0}^{L-1} \frac{\pi_e(S_t, A_t)}{\pi_b(S_t, A_t)} \right) \sum_{t=0}^{\infty} \gamma^t R_t. \tag{468}$$

We refer to the right hand side of the above equation as an *importance weighted return*,

The IS estimator for an entire dataset is simply the average IS estimate from each trajectory:

$$\text{IS}(D, \pi_e, \pi_b) = \frac{1}{n} \sum_{i=1}^{n} \text{IS}(H_i, \pi_e, \pi_b). \tag{469}$$

Notice that the IS estimator can have high variance. If H happens to be a trajectory that is more likely under π_e, for example, if each action is twice as likely under π_e, then the importance weight can be as large as 2^L—exponential in the horizon. In practice, often importance weights are near zero, with massive importance weights occurring occasionally such that, in expectation, the estimator is correct. Reducing the variance of importance sampling estimators is an active area of research (Jiang and Li, 2015).

Here we discuss one straightforward improvement. Rather than use importance sampling to estimate the expected return, let us use importance sampling to estimate $\mathbf{E}[R_t|\pi_e]$. This means defining $f(H) = R_t$ instead of using g for f. This importance sampling estimator is:

$$\left(\prod_{j=0}^{L-1} \frac{\pi_e(S_j, A_j)}{\pi_b(S_j, A_j)}\right) R_t. \tag{470}$$

However, notice that the actions taken after R_t occurs do not influence R_t. As a result, the product over time in the importance weight can stop at time t, giving:

$$\left(\prod_{j=0}^{t} \frac{\pi_e(S_j, A_j)}{\pi_b(S_j, A_j)}\right) R_t. \tag{471}$$

Now, the get an estimate of $\mathbf{E}[g(H)|H \sim \pi_e]$, we need to take the discounted sum of the per-reward estimates. This estimator is called the *per-decision importance sampling* (PDIS) estimator:

$$\text{PDIS}(H, \pi_e, \pi_b) = \sum_{t=0}^{L-1} \gamma^t \left(\prod_{j=0}^{t} \frac{\pi_e(S_j, A_j)}{\pi_b(S_j, A_j)}\right) R_t. \tag{472}$$

Similarly:

$$\text{PDIS}(D, \pi_e, \pi_b) = \frac{1}{n} \sum_{i=1}^{n} \text{PDIS}(H_i, \pi_e, \pi_b). \tag{473}$$

Both IS and PDIS give unbiased and strongly consistent estimates of $J(\pi_e)$ (Thomas, 2009). For further reading on other importance sampling estimators, I encourage you to read about *weighted importance sampling* estimators, discussed in my dissertation (Thomas, 2009).

14.2 High-Confidence Off-Policy Evaluation (HCOPE)

It's not enough for us to estimate the performance of a new policy, as these estimates are not always perfectly correct. To trust our estimates of $J(\pi_e)$, we need a confidence interval around our prediction, or at least one side of a confidence interval. That is, we want a value B such that:

$$\Pr(J(\pi_e) \geq B) \geq 1 - \delta, \tag{474}$$

where δ is some small probability. To obtain such a confidence interval, one might use Hoeffding's inequality (Hoeffding, 1963), a simplified form of which is:

Theorem 13 (Variant of Hoeffding's Inequality)**.** *If $X_1, \dots, X_n$ are n i.i.d. random variables and $\Pr(X_i \in [a,b]) = 1$, then*

$$\Pr\left(\mathbf{E}[X_1] \geq \frac{1}{n}\sum_{i=1}^{n} X_i - (b-a)\sqrt{\frac{\ln(1/\delta)}{2n}}\right) \geq 1-\delta. \tag{475}$$

However, recall that the range of the importance sampling estimates can grow exponentially with the horizon of the MDP, meaning that $(b-a)$ will be large, making our high-confidence lower bound on $J(\pi_e)$ a loose bound. Although there exist better concentration inequalities for obtaining high-confidence lower bounds (Thomas et al., 2015b; Maurer and Pontil, 2009; Anderson, 1969), in practice we often use a variant of Student's t-test, which makes a reasonable but false assumption:

Theorem 14. *If $X_1, \dots, X_n$ are n i.i.d. random variables and $\sum_{i=1}^n X_i$ is normally distributed, then:*

$$\Pr\left(\mathbf{E}[X_1] \geq \frac{1}{n}\sum_{i=1}^{n} X_i - \frac{\sqrt{\frac{1}{n-1}\sum_{i=1}^n \left(X_i - \bar{X}\right)^2}}{\sqrt{n}} t_{1-\delta,n-1}\right) \geq 1-\delta, \tag{476}$$

where $\bar{X} = \frac{1}{n}\sum_{i=1}^n X_i$ and $t_{1-\delta,\nu}$ is the $100(1-\delta)$ percentile of the Student t-distribution with ν degrees of freedom, i.e., tinv$(1-\delta,\nu)$ in Matlab.

Student's t-test assumes that $\bar{X}$ is normally distributed, which typically is not true. However, by the central limit theorem, as $n \to \infty$, it becomes normally distributed regardless of the distribution of X_i. Often this happens quickly, and so Student's t-test tends to hold with probability approximated $1-\delta$ if n is reasonably large. Here the definition of "reasonably large" depends on how non-normal the distribution of X_i is. In practice, I recommend that you use Student's t-test for obtaining confidence intervals, as it provides a nice trade-off of tightness, reasonable assumptions, and computational efficiency.

We use Hoeffding's inequaliy or Student's t-test to obtain a high-confidence lower bound on $J(\pi_e)$ by applying them with $X_i = \text{PDIS}(H_i, \pi_e, \pi_b)$. Since these importance sampling estimates are unbiased estimators of $J(\pi_e)$, we obtain (for Student's t-test):

$$\Pr\left(J(\pi_e) \geq \text{PDIS}(D, \pi_e, \pi_b) - \frac{\hat{\sigma}}{\sqrt{n}} t_{1-\delta,n-1}\right) \geq 1-\delta, \tag{477}$$

where $\hat{\sigma}$ is the sample standard deviation:

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(\text{PDIS}(H_i, \pi_e, \pi_b) - \overline{\text{PDIS}}\right)^2}, \tag{478}$$

where $\overline{\text{PDIS}} = \frac{1}{n}\sum_{i=1}^n \text{PDIS}(H_i, \pi_e, \pi_b)$.

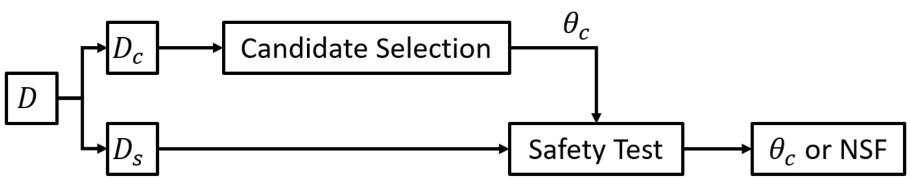


Figure 16: Diagram of high-confidence policy improvement algorithm.

14.3 High-Confidence Policy Improvement

We now have the necessary components to create our batch RL algorithm that ensures that $\Pr(J(a(D)) \geq c) \geq 1 - \delta$. An outline of the algorithm is presented in Figure 16.

At a high level, this algorithm partitions the available data into two sets, D_c (called the *candidate data*) and D_s (called the *safety data*). One might put half of the data in each set, 60% in D_c and 40% in D_s, or even 80% in D_c and 20% in D_s, though I recommend not placing less data in D_c. The candidate data is then used to pick a single solution, θ_c, called the *candidate solution*, that the algorithm considers returning. This candidate solution is then given to the *safety test*, which uses the held-out data, D_s, to check whether θ_c is safe to return. If it is, it returns θ_c, otherwise it returns NSF (No Solution Found).

As we provide more detail, let us begin with the safety test. Regardless of how θ_c is computed, we can use an HCOPE method to get a high-confidence lower bound on $J(\theta_c)$. If this high-confidence lower bound is at least c, then we return θ_c, otherwise we return NSF. That is, if

$$\underbrace{\text{PDIS}(D_s, \pi_e, \pi_b) - \frac{\hat{\sigma}_s}{\sqrt{|D_s|}} t_{1-\delta, |D_s|-1}}_{B} \geq c, \tag{479}$$

then return θ_c, otherwise return NSF, where we write $\hat{\sigma}_s$ to denote that the sample standard deviation is computed here using D_s and we denote a term by B to refer to later. When will we make an error? We will only make an error if $J(\pi_e) < c$ (the policy is actually not good enough to return) and $B > c$ (we would return the policy). However, by transitivity this means that $B \geq J(\pi_e)$, which from (477) we know will happen with probability at most δ. Hence, the probability we make a mistake is at most δ, and so we have ensured that (451) holds.

Now consider candidate selection. We could choose the candidate solution that we predict will perform best:

$$\theta_c \in \arg\max_\theta \text{PDIS}(D_c, \theta, \pi_b). \tag{480}$$

This will not work well, because the policy that we predict will perform the best is not necessarily one that is likely to pass the safety test. For example, this policy is often one with such high-variance PDIS estimates that the confidence interval from Student's t-test will be too wide to guarantee performance imporvement with high probability. The fix is to constrain the search to solutions that are predicted to pass the safety test:

$$\theta_c \in \arg\max_\theta \text{PDIS}(D_c, \theta, \pi_b) \tag{481}$$

$$\text{s.t. } \theta \text{ predicted to pass the safety test.} \tag{482}$$

How should we go about predicting which solutions will pass the safety test? Let us begin by plugging in the actual safety test:

$$\theta_c \in \arg\max_\theta \text{PDIS}(D_c, \theta, \pi_b) \tag{483}$$

$$\text{s.t. PDIS}(D_s, \pi_e, \pi_b) - \frac{\hat{\sigma}_s}{\sqrt{|D_s|}} t_{1-\delta,|D_s|-1} \geq c. \tag{484}$$

Using the actual safety test like this would result in solutions often passing the safety test – in fact, they would pass the safety test too often. Essentially, the candidate selection mechanism is cheating by looking at the data that will be used in the safety test, and then over-fitting to this data. Technically, this results in the random variables provided to Student's t-test (or Hoeffding's inequality) not being statistically independent.

So, within the candidate selection mechanism we must predict the outcome of the safety test *without looking at the safety data*. We *can* look at the number of episodes in the safety set, $|D_s|$, though we should not look at the contents of the data. So, we can use:

$$\theta_c \in \arg\max_\theta \text{PDIS}(D_c, \theta, \pi_b) \tag{485}$$

$$\text{s.t. PDIS}(D_c, \pi_e, \pi_b) - \frac{\hat{\sigma}_c}{\sqrt{|D_s|}} t_{1-\delta,|D_s|-1} \geq c. \tag{486}$$

Here we have replaced the PDIS estimate (and its sample standard deviation) with the estimates from the candidate data, but will still use the size of the safety data set when computing other terms. This will work in some cases, but it can result in the candidate selection mechanism over-fitting to the candidate data. This results in it often predicting that solutions will pass the safety test when in reality they will not. The fix is a hack—to double the confidence interval in candidate selection in order to increase the chance that the solution will actually pass the safety test. This gives us the actual expression used for finding θ_c:

$$\theta_c \in \arg\max_\theta \text{PDIS}(D_c, \theta, \pi_b) \tag{487}$$

$$\text{s.t. PDIS}(D_c, \pi_e, \pi_b) - 2\frac{\hat{\sigma}_c}{\sqrt{|D_s|}} t_{1-\delta,|D_s|-1} \geq c. \tag{488}$$

To summarize the algorithm: split D into D_c and D_s. Use D_c to select θ_c according to (487). Next, run the safety test. That is, if (479) holds, return θ_c, and otherwise return NSF.

The search for the candidate objective function can be performed using the optimization method of your choice, for example CMA-ES with some variant of a barrier function for the constraint.

For further details regarding high-confidence policy improvement algorithms (including alternate importance sampling estimators, concentration inequalities, and other improvements) see the relevant literature (Thomas et al., 2015b,c; Thomas, 2009).

——————————End of Lecture 20, November 14, 2019——————————

Name: ____________________________

CMPSCI 687 Midterm Exam

November 19, 2019

Instructions: This exam is **closed** notes. Do not use any notes or electronic devices. You have until the end of class to complete this exam. Complete this exam on your own—do not collaborate with anyone else. When you finish the exam, you may bring it to the front of the room and then leave. If you talk to anyone else, it will be considered cheating if either of you has not completed the exam. If both of you have completed the exam, **I reserve the right to change your grade to a zero on the exam**—this is to prevent the room from becoming loud near the end when some people are trying to finish.

Problem	**Possible Points**	**Points Obtained**	**Comments**
1 (total)	60		
2 (total)	20		
3 (total)	10		
4 (total)	10		

Exam Total Score (out of 100): ____________________________

1. ($60\%, 3\%$ each) Define the following (using math, and the notation used in class):

   - $P(s, a, s') = \Pr(S_{t=1} = s' | S_t = s, A_t = a)$

   - $R(s, a) = \mathbf{E}[R_t | S_t = s, A_t = a]$

   - $d_0(s) = \Pr(S_0 = s)$

   - $J(\pi) = \mathbf{E}[G|\pi]$ or $\mathbf{E}[\sum_{t=0}^{\infty} \gamma^k R_t | \pi]$

   - $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$

   - $v^\pi(s) = \mathbf{E}[G_t | S_t = s, \pi]$ or $\mathbf{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s, \pi]$

   - $q^\pi(s, a) = \mathbf{E}[G_t | S_t = s, A_t = a, \pi]$ or $\mathbf{E}[\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s, A_t = a, \pi]$

   - $Tv(s) =$
   $$\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s')(R(s, a) + \gamma v(s'))$$
   or
   $$\max_{a \in \mathcal{A}} (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s, a, s') v(s'))$$

   - Using the state-value function and general function approximation, $\delta_t = R_t + \gamma v_w(S_{t+1}) - v_w(S_t)$.

   - The Sarsa(λ) updates after observing S_t, A_t, R_t, S_{t+1}, and A_{t+1}, and using general (possibly non-linear function approximation) are as follows: The course notes were not clear about the time subscripts for this problem. We will not take of points for systematic off-by-one errors in the time subscripts in this problem and the next problem.

$$e_{t+1} = \gamma\lambda e_t + \frac{\partial q_w(S_t, A_t)}{\partial w} \tag{489}$$

$$\delta_t = R_t + \gamma q_w(S_{t+1}, A_{t+1}) - q_w(S_t, A_t) \tag{490}$$

$$w_{t+1} = w_t + \alpha\delta_t e_{t+1} \tag{491}$$

- The TD(λ) updates after observing S_t, A_t, R_t, and S_{t+1}, and using linear function approximation with features $\phi(s)$ for each state s, are as follows:

$$e_{t+1} = \gamma\lambda e_t + \phi(S_t) \tag{492}$$

$$\delta_t = R_t + \gamma v_w(S_{t+1}) - v_w(S_t) \text{ OR } R_t + \gamma w^\intercal \phi(S_{t+1}) - w^\intercal \phi(S_t) \tag{493}$$

$$w_{t+1} = w_t + \alpha\delta_t e_{t+1} \tag{494}$$

- The Bellman equation for v^π is defined as follows for all states s and policies π:

$$v^\pi(s) = \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^\pi(s')) \tag{495}$$

OR $\sum_{a\in\mathcal{A}} \pi(s,a)(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s,a,s') v^\pi(s'))$.

- The Bellman optimality equation is defined as follows for all states s:

$$v^*(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^*(s')) \tag{496}$$

OR $\max_{a\in\mathcal{A}}(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s,a,s') v^*(s'))$.

- $\text{IS}(H, \pi_e, \pi_b) = \left(\prod_{t=0}^{L-1} \frac{\pi_e(S_t,A_t)}{\pi_b(S_t,A_t)}\right) G$ or $\frac{\pi_e(H)}{\pi_b(H)} G$ or $\frac{\Pr(H|\pi_e)}{\Pr(H|\pi_b)} G$, or any of the previous answers with $\sum_{t=0}^{L-1} \gamma^t R_t$ instead of G.

- $G_t^{(n)} = \left(\sum_{k=0}^{n-1} \gamma^k R_{t+k}\right) + \gamma^n v(S_{t+n})$

- $G_t^\lambda = (1-\lambda)\sum_{i=0}^{\infty} \lambda^i G_t^{(i+1)}$.

2. (20%, 2% each) Indicate whether each of the statements below is true or false by circling either true or false.

   - (True or False) All MDPs with finite states, finite actions, finite horizon, and bounded reward have at least one optimal deterministic policy. During the exam we announced the additional constraint that $\gamma < 1$.
   - (True or False) We say that $\pi \geq \pi'$ if and only if $J(\pi) \geq J(\pi')$. We only defined the $\geq$ operator once for policies, in (163). That definition and this definition are not equivalent.
   - (True or False) Value iteration is an evaluation algorithm that converges to v^π. It is neither an evaluation algorithm, nor does it converge to v^π—it converges to v^*.
   - (True or False) Given that $S_t = s$, the Monte Carlo return G_t, generated by running policy π, is an unbiased estimator of $v^\pi(s)$.
   - (True or False) The parameter λ scales how much rewards are discounted.
   - (True or False) The Bellman operator is a contraction mapping when applied to finite MDPs with bounded rewards and $\gamma < 1$.
   - (True or False) Sarsa(λ) with linear function approximation will converge when run on any finite MDP with bounded rewards and $\gamma < 1$, even if it is only provided with observations about the state that could be missing some state information any even if these observations are noisy.
   - (True or False) Sarsa is an on-policy algorithm.
   - (True or False) Q-learning with linear function approximation is guaranteed to converge, though not to the optimal weights. It is not guaranteed to converge with linear function approximation—it can diverge.
   - (True or False) Pessimistic initial value functions encourage agents to explore. *Optimistic* initial value functions encourage agents to explore, not pessimistic.

3. (10 Points) Another researcher proposes a new importance sampling estimator, *shifted importance sampling* (SIS):

$$\text{SIS}(H, \pi_e, \pi_b) = \left(\prod_{t=0}^{L-1} \frac{\pi_e(S_t, A_t)}{\pi_b(S_t, A_t)}(G - b)\right) + b, \tag{497}$$

where $b \in \mathbb{R}$ is a constant called a *baseline*. Specifically, b will be their best-guess of $J(\pi_e)$, created without looking at the history H. Prove that if $\text{supp}(\pi_e) \subseteq \text{supp}(\pi_b)$, then $\text{SIS}(H, \pi_e, \pi_b)$ is an unbiased estimator of $J(\pi_e)$ if $H \sim \pi_b$. You may use the results $\mathbf{E}[\text{IS}(H, \pi_e, \pi_b)|H \sim \pi_b] = J(\pi_e)$ and $\prod_{t=0}^{L-1} \frac{\pi_e(S_t,A_t)}{\pi_b(S_t,A_t)} = \frac{\Pr(H|\pi_e)}{\Pr(H|\pi_b)}$ without proving them. You may assume that set, $\mathcal{H}$, of possible trajectories, H, is finite. During the exam we announced that the parentheses should be:

$$\text{SIS}(H, \pi_e, \pi_b) = \left(\prod_{t=0}^{L-1} \frac{\pi_e(S_t, A_t)}{\pi_b(S_t, A_t)}\right)(G - b) + b, \tag{498}$$

$$\mathbf{E}[\text{SIS}(H, \pi_e, \pi_b)|H \sim \pi_b] = \mathbf{E}\left[\left(\prod_{t=0}^{L-1} \frac{\pi_e(S_t, A_t)}{\pi_b(S_t, A_t)}\right)(G - b) + b \middle| H \sim \pi_b\right] \tag{499}$$

$$=\mathbf{E}\left[\left(\frac{\Pr(H|\pi_e)}{\Pr(H|\pi_b)}(G - b)\right) + b \middle| H \sim \pi_b\right] \tag{500}$$

$$= \sum_{H \in \text{supp}(\pi_b)} \Pr(H|\pi_b)\left(\frac{\Pr(H|\pi_e)}{\Pr(H|\pi_b)}(G - b) + b\right) \tag{501}$$

$$= \sum_{H \in \text{supp}(\pi_b)} \Pr(H|\pi_e)(G - b) + \Pr(H|\pi_b)b \tag{502}$$

$$= \sum_{H \in \text{supp}(\pi_b)} \Pr(H|\pi_e)G - \sum_{H \in \text{supp}(\pi_b)} \Pr(H|\pi_e)b + \sum_{H \in \text{supp}(\pi_b)} \Pr(H|\pi_b)b \tag{503}$$

$$= \sum_{H \in \text{supp}(\pi_b)} \Pr(H|\pi_e)G - b\underbrace{\sum_{H \in \text{supp}(\pi_b)} \Pr(H|\pi_e)}_{=1} + b\underbrace{\sum_{H \in \text{supp}(\pi_b)} \Pr(H|\pi_b)}_{=1} \tag{504}$$

$$\overset{\text{(a)}}{=} \sum_{H \in \text{supp}(\pi_e)} \Pr(H|\pi_e)G \tag{505}$$

$$=\mathbf{E}[G|H \sim \pi_e] \tag{506}$$

$$=J(\pi_e), \tag{507}$$

where **(a)** holds because $\text{supp}(\pi_b) \subseteq \text{supp}(\pi_e)$.

4. (10 Points) Consider a new Bellman-like operator M, defined as:

$$Mv(s) \coloneqq \sum_{a \in \mathcal{A}} \pi(s,a) \max_{s' \in \mathcal{S}} \left(R(s,a) + \gamma v(s')\right). \tag{508}$$

Does the iteration $v_{t+1} \leftarrow Mv_t$ converge (for all MDPs with finite state and action sets, bounded rewards, $\gamma < 1$, and v_0 initialized to the zero-vector)? If not, prove that it does not. If it does, prove that it does and explain whether or not it converges to v^π for finite-horizon MDPs where $R_t \leq 0$ always. You may use the following results without proving them:

$$\left|\max_{x \in \mathcal{X}} f(x) - \max_{x \in \mathcal{X}} g(x)\right| \leq \max_{x \in \mathcal{X}} |f(x) - g(x)|. \tag{509}$$

and for any probability distribution p on $\mathcal{X}$:

$$\left|\sum_{x \in \mathcal{X}} p(x) f(x)\right| \leq \sum_{x \in \mathcal{X}} p(x) \left|f(x)\right|. \tag{510}$$

It does converge. Like the Bellman operator, we will show that it converges by showing that it is a contraction under the max-norm—convergence then follows from the Banach fixed-point theorem.

$$\begin{aligned}
\|Mv - Mv'\| = &\max_{s \in \mathcal{S}} \Big| \sum_{a \in \mathcal{A}} \pi(s,a) \max_{s' \in \mathcal{S}} \left(R(s,a) + \gamma v(s')\right) && (511)\\
&- \sum_{a \in \mathcal{A}} \pi(s,a) \max_{s' \in \mathcal{S}} \left(R(s,a) + \gamma v'(s')\right) \Big| && (512)\\
=&\gamma \max_{s \in \mathcal{S}} \left| \sum_{a \in \mathcal{A}} \pi(s,a) \max_{s' \in \mathcal{S}} \left(v(s') - v'(s')\right)\right| && (513)\\
=&\gamma \underbrace{\max_{s \in \mathcal{S}}}_{\text{(a)}} \left| \max_{s' \in \mathcal{S}} \left(v(s') - v'(s')\right) \underbrace{\sum_{a \in \mathcal{A}} \pi(s,a)}_{=1} \right| && (514)\\
=&\gamma \left| \max_{s' \in \mathcal{S}} \left(v(s') - v'(s')\right)\right| && (515)\\
\leq&\gamma \max_{s' \in \mathcal{S}} |v(s') - v'(s')| && (516)\\
=&\gamma \|v - v'\|, && (517)
\end{aligned}$$

where **(a)** is dropped because the remainder of the function does not depend on s. Hence, the iteration converges to a unique fixed point. We now solve for this fixed-point when $R_t \leq 0$ always.

Note that $v_t(s)$ will always be less than or equal to zero since we assumed $R_t \leq 0$ and $v_0(s) = 0$. Also, note that $v_t(s_\infty)$ will never change from its initial value of zero. So, the $\max_{s' \in \mathcal{S}}$ will always select $s' = s_\infty$, and

$v_t(s_\infty) = 0$, so the update becomes $v_{t+1}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) R(s,a)$. As the right side does not depend on v_t, this update converges in one step, with the fixed point $v_1(s) = \mathbf{E}[R_t | S_t = s, \pi]$. This is usually not the state-value function (though it is when $\gamma = 0$). As announced during the exam, the latter part of this question asked for an "explanation" not a proof—we do not require a formal proof that this operator does not converge to v^π always.

This page intentionally left blank.

15 Guest Lecture

In this guest lecture, Andy Barto presented on the psychology and neuroscience chapters of the second edition of his book. For further information, we refer you to these chapters.

———————————End of Lecture 21, November 21, 2019———————————

CMPSCI 687 Course Project

Due December 10, 2019

Due Date: This course project is due on the last day of class, December 10. There will be no penalty for submitting late as long as you submit by midnight at the end of December 13.

Overview: This course project simulates the application of RL to a real problem, like a medical application or business application, where deploying a policy worse than the current one would be dangerous or costly. To simulate this, **1)** we will provide you with data, but not the underlying MDP that created the data and **2)** deploying a policy that is worse than the policy we used to generate the data will be costly for your grade—your grade will be the percent of policies that you provide us that are better than the policy we used to generate the data. That is, if your policy is π and the policy we used to generated data is π_b, then your policy is better if and only if $J(\pi) > J(\pi_b)$.

Tools and Collaboration: You can use *any* language and code libraries you find online. You can discuss the project with other students, including the approaches you plan to take. **However, you must write your code on your own.** You will also be submitting your code with compilation instructions, though your code will not influence your grade other than to confirm that it produces the policies you submit.

Details: We will provide you with one file, data.csv, and you will provide us with 100 policies (1.csv, 2.csv, etc.) and the code you used to generate these policies. Your policies must be different from the policy we used to generate the data (the details of this policy are included in data.csv), though they do not need to be different from each other—you could submit the same policy for each of the 100 policies you submit, though then your grade will be a zero if it is worse (but 100 if it is better). You can use *any* approach you want to compute the policies that you submit. We will use $\gamma = 1$, and you may assume that $R_t \in [-10, 10]$ always.

The file data.csv will contain the following values, where each item below indicates a row in the CSV file.

1. An integer m indicating the number of state features.
2. An integer $|\mathcal{A}|$ indicating the number of discrete actions. That is $\mathcal{A} = \{0, 1, 2, \ldots, |\mathcal{A}| - 1\}$.
3. An integer k, indicating the Fourier basis order used by π_b, the policy that generated the data we are providing.
4. θ_b, the parameters of the policy π_b. The following describes precisely how θ_b parameterizes π_b: in short, it is a softmax policy using the Fourier basis of order k, and assuming that states are already normalized when they are

provided (they will be). There will be $|\mathcal{A}|(k+1)^m$ real numbers on this row. Let θ_b^i denote the i^{th} block of $(k+1)^m$ numbers on this row, as a column. Then, $\theta_b = [(\theta_b^1)^\intercal, (\theta_b^2)^\intercal, \dots, (\theta_b^{|\mathcal{A}|})^\intercal]^\intercal$. The policy π_b is defined as:

$$\pi_b(s,a) = \frac{e^{\phi(s)^\intercal \theta_b^a}}{\sum_{a' \in \mathcal{A}} e^{\phi(s)^\intercal \theta_b^{a'}}}, \tag{518}$$

where $\phi(s) \in \mathbb{R}^{(k+1)^m}$, the i^{th} element of which is: $\phi_i(s) = \cos(\pi c_i^\intercal s)$, where $c_i \in \mathbb{R}^{|\mathcal{S}|}$ is the number i written in base $k+1$ using little-endian (100 is one rather than 001), where each digit corresponds to one element of the vector c_i.

5. An integer, n, that indicates the number of episodes of data that will follow.

6. The next n rows each correspond to one episode of data. The row will contain $S_0, A_0, R_0, S_1, A_1, R_1, \dots, S_{T-1}, A_{T-1}, R_{T-1}$, where T is the length of the episode (which may vary across episodes). Here the states S_t will each be sequences of m real numbers, A_t will be a single integer, and R_t will be a real number. The total number of entries on this row is therefore $T(m+2)$. The states will really be observations—measurements about the world that the agent is interacting with—and so they may not be Markovian. Assume that $S_T = s_\infty$. The provided states will be normalized so that all values are in the range $[0,1]$.

7. Let T_1 be the length of the first episode of data above. This line contains T_1 real numbers, that correspond to $\pi_b(S_t, A_t)$ for $t \in \{0, 1, 2, \dots, T-1\}$. This row is provided to give you a way to test whether your policy representation matches ours.

What to submit: You will submit a .zip folder that contains files 1.csv, 2.csv, ..., 100.csv. If you want to submit a single policy, you still need to have all of these files, though they might be identical. These files each contain one row corresponding to new policy parameters to use in place of θ_b using the same policy parameterization described for π_b. There should also be a directory called "source" that contains source code for producing these policies. Inside of the "source" directory there should be a file called "readme.txt" that includes instructions for running your source code.

Hint: You probably do not want to write code, run it on our data first, and submit the policies it produces. If you do this, how confident are you that your code is correct and that the method you implemented will succeed? We recommend that you create your own MDPs, generate data, and see if your method succeeds on those tests. Running your code on our provided data should just be the final step right before submitting. We will ensure that particularly advanced methods that we did not cover in class are not necessary (e.g., using the high-confidence policy improvement algorithm from class with t-test and per-decision importance sampling can be enough to obtain a 90%).

Data availability: We will provide a link to the data here on December 3. To help you create your example MDPs (though we suggest starting tabular and building up!), the MDP that we use will likely have a very small value of m (likely 1), k will be small, and episode lengths will be short (likely < 10).

16 Actor-Critic Methods

Actor-critic algorithms are algorithms that have an actor, which selects actions, and a critic, that critiques the actions chosen by the actor. That is, the actor stores π, and the critic estimates v^π. Intuitively, the critic watches what the actor does, and passes the TD errors to the actor. If the actor receives a positive TD error, it means that the action it took turned out better than it expected, and so the probability of the action should be increased. If the actor receives a negative TD error, it means that the action it took turned out worse than it expected, and so the probability of the action should be decreased.

We could construct a simple actor-critic by using TD to update a tabular critic:

$$\delta_t \leftarrow R_t + \gamma v(S_{t+1}) - v(S_t) \tag{519}$$

$$v(S_t) \leftarrow v(s) + \alpha\delta_t. \tag{520}$$

We could then use a softmax policy, parameterized as in (72). To capture our intuition, we can then use the actor update rule:

$$p(S_t, A_t) \leftarrow P(S_t, A_t) + \alpha\delta_t. \tag{521}$$

Notice that the actor and critic updates are very similar—they both add $\alpha\delta_t$ to a stored variable. However, this update causes the actor to tend towards an optimal policy, and the critic to tend towards v^π. This different behavior is due to the different conditioning of the updates: the expected update to the critic is the expected TD-error given the current state, while the expected update to the actor is the expected TD-error given the current state and action.

16.1 Policy Gradient Algorithms

Policy gradient is not just one algorithm—it is a class of algorithms. Many policy gradient algorithms are actor-critics, but not all. Similarly, actor-critic is not a single algorithm, but a class of algorithms, and many (but not all) actor-critics are policy gradient algorithms.

For example (referencing algorithms we will describe later), our simple actor-critic is an actor-critic, but is not a policy gradient algorithm. REINFORCE (Williams, 1992) is a policy gradient algorithm, but it doesn't have a critic and therefore is not an actor-critic. So, although most policy gradient algorithms will also be actor-critics, and most actor-critic algorithms are policy gradient algorithms, these two terms are not interchangeable.

The idea underlying policy gradient algorithms is that we can use a parameterized policy, with parameters $\theta \in \mathbb{R}^n$, we can define an objective function:

$$J(\theta) = \mathbf{E}\left[G|\theta\right], \tag{522}$$

and then we can perform gradient ascent to search for policy parameters that maximize the expected discounted return:

$$\theta \leftarrow \theta + \alpha\nabla J(\theta). \tag{523}$$

Policy gradient methods have several benefits over value function based methods like Sarsa and Q-learning. First, policy gradient methods work well with continuous actions (we can easily parameterize a continuous distribution), while Q-learning and Sarsa often do not (and solving for the action that maximizes $q(s, a)$ when a is continuous can be computationally expensive) (Baird and Klopf, 1993). Second, since they are (stochastic) gradient algorithms, policy gradient algorithms tend to have convergence guarantees when value-function based methods do not (e.g., using non-linear policy parameterizations is not a problem for policy gradient methods). Furthermore, whereas value-based methods approximately optimize an objective (minimizing some notion of expected Bellman error), this objective is merely a surrogate for the primary objective: maximizing expected return. Policy gradient methods take a more direct approach and directly maximize the expected return.

Notice however, that Q-learning and Sarsa are global algorithms in that they are guaranteed to converge to *globally* optimal policies (when using a tabular representation), whereas gradient methods can often become stuck in local minima. This common argument is flawed: when using tabular representations for finite MDPs, the objective function has no local optima (Thomas, 2014). The proof presented in this citation is not complete because it does not address the fact that global optima also do not exist, since weights will tend to $\pm\infty$. A complete proof showing convergence to an optimal policy is in-progress.

The crucial question that we will address in future lectures is: how can we estimate $\nabla J(\theta)$ when we do not know the transition or reward function?

16.2 Policy Gradient Theorem

How can we efficiently compute $\nabla J(\theta)$? One option is to use finite difference methods, which approximate the gradients of functions by evaluating them at different points. However, these algorithms do not take advantage of the known structure of the problem: that the objective function corresponds to expected returns for an MDP. One might also use automatic differentiation tools, but these require knowledge of the transition function and reward function.

The *policy gradient theorem* gives an analytic expression for $\nabla J(\theta)$ that consists of terms that are known or which can be approximated. Here we will follow the presentation of Sutton et al. (2000). The policy gradient theorem states:

Theorem 15 (Policy Gradient Theorem)**.** *If $\frac{\partial \pi(s,a,\theta)}{\partial \theta}$ exists for all s, a, and θ, then for all finite finite MDPs with bounded rewards, $\gamma \in [0, 1)$, and unique deterministic initial state s_0,*

$$\nabla J(\theta) = \sum_{s \in \mathcal{S}} d^\theta(s) \sum_{a \in \mathcal{A}} q^\pi(s, a) \frac{\partial \pi(s, a, \theta)}{\partial \theta}, \tag{524}$$

where $d^\theta(s) = \sum_{t=0}^{\infty} \gamma^t \Pr(S_t = s | \theta)$.

Although we present the policy gradient theorem here for finite MDPs, extensions hold for MDPs with continuous states and/or actions, and even hybrid (mixtures of discrete and continuous) states and actions. Extensions to MDPs without unique deterministic initial states, and to the average reward setting also exist. Recall that

$$\frac{\partial \ln(f(x))}{\partial x} = \frac{1}{x}\frac{\partial f(x)}{\partial x}, \tag{525}$$

and so

$$\frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta} = \frac{1}{\pi(s,a,\theta)}\frac{\partial \pi(s,a,\theta)}{\partial \theta}. \tag{526}$$

Thus, we can rewrite (524) as:

$$\nabla J(\theta) = \sum_{s\in\mathcal{S}} d^\theta(s) \sum_{a\in\mathcal{A}} \pi(s,a,\theta) q^\pi(s,a) \frac{\partial \ln\left(\pi(s,a,\theta)\right)}{\partial \theta}. \tag{527}$$

Also, if we were to view d^θ as a distribution over the states (it is *not*, as we will discuss shortly), then we could write the policy gradient theorem as:

$$\nabla J(\theta) = \mathbf{E}\left[q^\pi(S,A)\frac{\partial \ln(\pi(S,A,\theta))}{\partial \theta}\middle| \theta\right], \tag{528}$$

where $S \sim d^\theta$ and $A \sim \pi(S,\cdot,\theta)$.

To obtain an intuitive understanding of (524), recall that $\frac{\partial f(x,y)}{\partial y}$ is the direction of change to y taht most quickly increases $f(x,y)$. That is, it is the direction of steepest ascent of $f(x,\cdot)$ at y. So, if we consider each term:

$$\nabla J(\theta) = \underbrace{\sum_{s\in\mathcal{S}} d^\theta(s)}_{\text{average over states}} \underbrace{\sum_{a\in\mathcal{A}} \pi(s,a,\theta)}_{\text{average over actions}} \underbrace{q^\pi(s,a)}_{\text{how good is } a \text{ in } s?} \underbrace{\frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta}}_{\text{How to change } \theta \text{ to make } a \text{ more likely in } s}. \tag{529}$$

Consider d^θ in more detail. This term is sometimes called the *discounted state distribution*. However, notice that it is *not* a probability distribution. Since $\sum_{t=0}^\infty \gamma^t = \frac{1}{1-\gamma}$, we could make d^θ into a distribution by multiplying it by $1-\gamma$. Intuitively, what d^θ does is it averages over state distributions at different times, giving less weight to later state distributions. So, $J(\theta)$ favors changes to the policy that increase the expected return at earlier time steps.

The policy gradient theorem can be written as:

$$\nabla J(\theta) \propto \mathbf{E}\left[\sum_{t=0}^\infty \gamma^t q^\pi(S_t,A_t)\frac{\partial \ln(\pi(S_t,A_t,\theta))}{\partial \theta}\middle|\theta\right], \tag{530}$$

where here states and actions are generated by running the policy with parameters θ—not by sampling from $(1-\gamma)d^\theta$.

In practice, most algorithms ignore the γ^t term preceding $q^\pi(S_t,A_t)$ in (530). For further discussion of this term (and why it is usually ignored), see the work of Thomas and Barto (2012); Thomas (2014).

——————————End of Lecture 22, December 3, 2019——————————

Name:

Student ID:

CMPSCI 687 Pop Quiz 7 or 8
December 5, 2019

Instructions: You have 5 minutes to complete this quiz. This quiz is **closed** notes—do not use your notes or a laptop. Do not discuss problems with your neighbors until after everyone has handed in their quiz.

Circle either true or false below.

1. (True or False) Policy gradient algorithms change the policy parameters using gradient descent on the mean squared Bellman error.
2. (True or False) All policy gradient algorithms are actor-critic algorithms.
3. (True or False) All actor-critic algorithms are policy gradient algorithms.
4. (True or False) When not equal to zero, $\partial \ln(\pi(s,a,\theta))/\partial\theta$ is a direction of change to θ that increases $\pi(s,a,\theta)$.
5. (True or False) If one were to estimate the action value function with $q_w(s,a) = w^\intercal \frac{\partial \ln(\pi(s,a,\theta))}{\partial\theta}$, this would be a case of **linear** function approximation.

16.3 Proof of the Policy Gradient Theorem

In this section we may switch freely between using π and θ to denote a policy. Writing π emphasizes parameters. A more precise notation might be to always write $\pi(\cdot, \cdot, \theta)$, but this is too verbose. So, for example, we may switch between writing q^π and q^θ to both denote the action-value function when when using the parameterized policy π, with parameters θ. We will tend towards using the θ notation to make it clear which terms depend on the policy parameters. Note that, since $S_0 = s$ always,

$$J(\theta) = \mathbf{E}\left[G|\theta\right] \tag{531}$$

$$= v^\theta(s_0). \tag{532}$$

So, to obtain an expression for the policy gradient we will obtain an expression for the derivative of the value of a state with respect to the policy parameters.

We begin by showing this for all states, not just s_0. That is, for all states $s \in \mathcal{S}$:

$$\frac{\partial v^\theta(s)}{\partial \theta} \tag{533}$$

$$= \frac{\partial}{\partial \theta} \sum_{a \in \mathcal{A}} \pi(s,a,\theta) q^\theta(s,a) \tag{534}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \pi(s,a,\theta) \frac{\partial q^\theta(s,a)}{\partial \theta} \tag{535}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \sum_{a \in \mathcal{A}} \pi(s,a,\theta) \frac{\partial}{\partial \theta} \sum_{s' \in \mathcal{S}} P(s,a,s') \left(R(s,a) + \gamma v^\theta(s') \right) \tag{536}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \frac{\partial v^\theta(s')}{\partial \theta} \tag{537}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \frac{\partial}{\partial \theta} \left(\sum_{a' \in \mathcal{A}} \pi(s',a',\theta) q^\theta(s',a') \right) \tag{538}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \left(\sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a') + \pi(s',a',\theta) \frac{\partial q^\theta(s',a')}{\partial \theta} \right) \tag{539}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a') \tag{540}$$

$$+ \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \sum_{a' \in \mathcal{A}} \pi(s',a',\theta) \frac{\partial q^\theta(s',a')}{\partial \theta} \tag{541}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a') \tag{542}$$

$$+ \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \sum_{a' \in \mathcal{A}} \pi(s',a',\theta) \frac{\partial}{\partial \theta} \left(\sum_{s'' \in \mathcal{S}} P(s',a',s'')(R(s',a') + \gamma v^\theta(s'')) \right) \tag{543}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a') \tag{544}$$

$$+ \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \sum_{a' \in \mathcal{A}} \pi(s',a',\theta) \sum_{s'' \in \mathcal{S}} P(s',a',s'') \gamma \frac{\partial v^\theta(s'')}{\partial \theta} \tag{545}$$

$$= \underbrace{\sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a)}_{\text{first term}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a')}_{\text{second term}} \tag{546}$$

$$+ \gamma \sum_{s'' \in \mathcal{S}} \Pr(S_{t+2}=s''|S_t=s,\theta) \gamma \frac{\partial v^\theta(s'')}{\partial \theta}. \tag{547}$$

Notice what we have been doing: we are expanding the state value function by looking forward one time step, and writing the value function in terms of the value of the next state, and then repeating this process. Above we have "unravelled" two times, resulting in two terms, marked in the final expression. If we were to unravel the expression one more time, by expanding $\partial v^\theta(s'')/\partial\theta$ and then differentiating, we would obtain:

$$\frac{\partial v^\theta(s)}{\partial\theta} = \underbrace{\sum_{a\in\mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial\theta} q^\theta(s,a)}_{\text{first term}} + \underbrace{\gamma \sum_{s'\in\mathcal{S}} \Pr(S_{t+1}=s'|S_t=s,\theta) \sum_{a'\in\mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial\theta} q^\theta(s',a')}_{\text{second term}} \tag{548}$$

$$+ \underbrace{\gamma^2 \sum_{s''\in\mathcal{S}} \Pr(S_{t+2}=s''|S_t=s,\theta) \sum_{a''\in\mathcal{A}} \frac{\partial \pi(s'',a'',\theta)}{\partial\theta} q^\theta(s'',a'')}_{\text{third term}} \tag{549}$$

$$+ \gamma^2 \sum_{s'''\in\mathcal{S}} \Pr(S_{t+r}=s'''|S_t=s,\theta) \gamma \frac{\partial v^\theta(s''')}{\partial\theta}. \tag{550}$$

Notice that in each term the symbol used for the state and action does not matter, and we can write x for the state and a for the action (we also replace the final term with ... to denote that we could continue to unravel the expression):

$$\frac{\partial v^\theta(s)}{\partial\theta} = \underbrace{\sum_{a\in\mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial\theta} q^\theta(s,a)}_{\text{first term}} + \underbrace{\gamma \sum_{x\in\mathcal{S}} \Pr(S_{t+1}=x|S_t=s,\theta) \sum_{a\in\mathcal{A}} \frac{\partial \pi(x,a,\theta)}{\partial\theta} q^\theta(x,a)}_{\text{second term}} \tag{551}$$

$$+ \underbrace{\gamma^2 \sum_{x\in\mathcal{S}} \Pr(S_{t+2}=x|S_t=s,\theta) \sum_{a\in\mathcal{A}} \frac{\partial \pi(x,a,\theta)}{\partial\theta} q^\theta(x,a)}_{\text{third term}} + \dots \tag{552}$$

We now index each term by k, with the first term being $k=0$, the second $k=1$, etc., which results in the expression:

$$\frac{\partial v^\theta(s)}{\partial\theta} = \sum_{k=0}^{\infty} \sum_{x\in\mathcal{S}} \Pr(S_{t+k}=x|S_t=s,\theta) \sum_{a\in\mathcal{A}} \gamma^k \frac{\partial \pi(x,a,\theta)}{\partial\theta} q^\theta(x,a). \tag{553}$$

Notice that we have modified the first term by including a sum over states. This is not a change because when $k=0$, the only state, x, where $\Pr(S_{t+0}=x|S_t=s,\theta)$ is not zero will be when $x=s$ (at which point this probability is one).

Notice that, in the notation used by Sutton et al. (2000), $\Pr(S_{t+k}=x|S_t=s,\theta)$ is denoted by $\Pr(s\to x,k,\pi)$.

With this expression for the value derivative of the value of any state with respect to the policy parameters, we turn to computing the policy gradient in the start-state setting:

$$
\begin{aligned}
\nabla J(\theta) =& \frac{\partial}{\partial\theta} J(\theta) && (554)\\
=& \frac{\partial}{\partial\theta} \mathbf{E}\left[G|\theta\right] && (555)\\
=& \frac{\partial}{\partial\theta} \mathbf{E}\left[G_t|S_t = s_0, \theta\right] && (556)\\
=& \frac{\partial}{\partial\theta} v^\theta(s_0) && (557)\\
=& \sum_{k=0}^{\infty} \sum_{x\in\mathcal{S}} \Pr(S_{t+k} = x|S_t = s_0, \theta) \sum_{a\in\mathcal{A}} \gamma^k \frac{\partial\pi(x,a,\theta)}{\partial\theta} q^\theta(x,a) && (558)\\
=& \sum_{x\in\mathcal{S}} \underbrace{\sum_{k=0}^{\infty} \gamma^k \Pr(S_{t+k} = x|S_t = s_0, \theta)}_{=d^\theta(x)} \sum_{a\in\mathcal{A}} \frac{\partial\pi(x,a,\theta)}{\partial\theta} q^\theta(x,a) && (559)\\
=& \sum_{s\in\mathcal{S}} d^\theta(s) \sum_{a\in\mathcal{A}} \frac{\partial\pi(s,a,\theta)}{\partial\theta} q^\theta(s,a), && (560)
\end{aligned}
$$

where the last term comes from replacing the symbol x with the symbol s. This completes the proof of the policy gradient theorem.

16.4 REINFORCE

The REINFORCE algorithm (Williams, 1992) uses unbiased estimates of the policy gradient to perform stochastic gradient ascent on J. To obtain stochastic gradient estimates, notice that the policy gradient can be written as:

$$
\nabla J(\theta) \propto \mathbf{E}\left[\gamma^t q^\theta(S_t, A_t) \frac{\partial \ln \pi(S_t, A_t, \theta)}{\partial\theta} \middle| \theta\right], \tag{561}
$$

where $\propto$ is due to the dropped missing $(1-\gamma)$ term necessary to make d^θ a distribution, and where S_t and A_t are sampled according to the on-policy distribution (by running the policy with parameters θ and observing the resulting states and actions), and where t is uniformly distributed from 0 to $L-1$. Alternatively, we can avoid the uniform distribution of t by summing over time steps in the episode:

$$
\nabla J(\theta) \propto \mathbf{E}\left[\sum_{t=0}^{L} \gamma^t q^\theta(S_t, A_t) \frac{\partial \ln \pi(S_t, A_t, \theta)}{\partial\theta} \middle| \theta\right]. \tag{562}
$$

We can obtain unbiased estimates of this gradient by sampling running an episode to obtain samples of S_t and A_t, and using G_t as an unbiased estimate of

$q^\theta(S_t, A_t)$. In Algorithm 20 we present the unbiased REINFORCE algorithm—true stochastic gradient ascent on J.

Algorithm 20: Stochastic Gradient Ascent on J (REINFORCE)

Initialize θ arbitrarily;
for *each episode* **do**
  Generate an episode $S_0, A_0, R_0, S_1, A_1, R_1, \ldots, S_{L-1}, A_{L-1}, R_{L-1}$ using policy parameters θ.;
  $\widehat{\nabla J(\theta)} = \sum_{t=0}^{L-1} \gamma^t G_t \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}$;
  $\theta \leftarrow \theta + \alpha \widehat{\nabla J(\theta)}$;

Question 31. *Consider the REINFORCE algorithm presented on page 328 of the second edition of Sutton and Barto's book (Sutton and Barto, 2018). Compare their algorithm to the one presented above. Notice that they are* not *equivalent. Are they both true stochastic gradient ascent algorithms? Is one not?*

Answer 31. *The algorithm we have presented is a true stochastic gradient ascent algorithm. The algorithm Sutton and Barto presented is approximately stochastic gradient ascent. It is only approximately stochastic gradient ascent because they change the parameters at each time step of the episode. So, (using their notation) the $\nabla \ln \pi(A_t|S_t, \theta)$ term in their update will be computed using parameters θ, for which G is not an unbiased estimator of $q^\theta(S_t, A_t)$, since G was produced using previous policy parameters. Notice that Williams (1992) explicitly states, just previous to his equation 11, that "At the conclusion of each episode, each parameter w_{ij} is incremented by [...]," and so it is proper to view REINFORCE as updating at the end of episodes, not during episodes.*

In practice, the γ^t term appearing in the REINFORCE pseudocode is ignored. This term came from the discounted state distribution, and results in a discounting of the updates that degrades performance empirically. This theory surrounding the removal of this γ^t term has been discussed by Thomas (2014)—at present, it is not known whether this algorithm without the γ^t term is even a true stochastic gradient function (just for a different objective). Still, in your implementations, you should likely drop this γ^t term.

Since REINFORCE is using a Monte Carlo estimator, G_t, of $q^\theta(S_t, A_t)$, it has high variance. Consider how we might reduce the variance of this update. One approach is to use a *control variate*. Consider a general problem unrelated to reinforcement learning: estimating $\mu = \mathbf{E}[X]$ for some random variable X. Consider doing so given a single sample, X. The obvious estimator of μ is $\hat{m}u = X$. This estimator is unbiased: $\mathbf{E}[\hat{\mu}] = \mathbf{E}[X] = \mu$, and has variance $\text{Var}(\hat{\mu}) = \text{Var}(X)$.

Now consider estimating μ given a single sample, X, as well as a sample, Y of another variable whose expectation, $\mathbf{E}[Y]$ is known. Can we somehow create an estimator of μ that is better? One approach is to use the estimator $\hat{\mu} = X - Y + \mathbf{E}[Y]$. This estimator is still unbiased:

$$\mathbf{E}[\hat{\mu}] = \mathbf{E}[X - Y + \mathbf{E}[Y]] \tag{563}$$

$$= \mathbf{E}[X] - \mathbf{E}[Y] + \mathbf{E}[Y] \tag{564}$$

$$= \mathbf{E}[X] \tag{565}$$

$$= \mu. \tag{566}$$

However, its variance is:

$$\operatorname{Var}(\hat{\mu}) = \operatorname{Var}(X - Y + \mathbf{E}[Y]) \tag{567}$$

$$= \operatorname{Var}(X - Y) \tag{568}$$

$$= \operatorname{Var}(X) + \operatorname{Var}(Y) - 2\operatorname{Cov}(X, Y). \tag{569}$$

This variance is lower than the variance of the original estimator when:

$$\operatorname{Var}(X) + \operatorname{Var}(Y) - 2\operatorname{Cov}(X, Y) < \operatorname{Var}(X), \tag{570}$$

or equivalently, when

$$\operatorname{Var}(Y) < 2\operatorname{Cov}(X, Y). \tag{571}$$

We refer to Y as a *control variate*.

So, if Y is similar to X—if X and Y have positive covariance, then subtracting Y from X can reduce the variance of the estimate. However, even if Y and X have positive covariance, if Y is very noisy, the additional variance introduced by adding Y can result in a net increase in variance. So, Y helps if it is low variance, yet similar to X. In some cases, Y might be an estimate of a random process X, built from a model that has error (see the discussion of the doubly robust estimator in the appendix of the paper by Thomas and Brunskill (2016)). This provides a way to use a model that has error to reduce the variance of Monte Carlo estimates while preserving the unbiased nature of the estimate. In more general cases, if you know something about the randomness in X, but you don't know X precisely, you can subtract off your random estimate, Y, of X, and add back in the expected value of the amount that you are subtracting off, and this will often reduce variance.

Consider again the REINFORCE update. We will insert a control variate to get the update:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{L-1} \gamma^t (G_t - b(S_t)) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}, \tag{572}$$

where $b : \mathcal{S} \to \mathbb{R}$ is any function of the state. To see this in the form of a control variate, we can rewrite it as:

$$\theta \leftarrow \theta + \underbrace{\alpha \sum_{t=0}^{L-1} \gamma^t G_t \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}}_{X} - \underbrace{\alpha \sum_{t=0}^{L-1} \gamma^t b(S_t) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}}_{Y}, \tag{573}$$

where X is our unbiased gradient estimate (ignoring the $(1-\gamma)$ normalization term) and Y is the control variate for this estimate. The $\mathbf{E}[Y]$ term is not present because, in this case, it is always zero (regardless of the choice of b). That is:

$$\mathbf{E}\left[\alpha \sum_{t=0}^{L-1} \gamma^t b(S_t) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \middle| \theta \right] = \alpha \sum_{t=0}^{L-1} \gamma^t \mathbf{E}\left[b(S_t) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \middle| \theta \right] \tag{574}$$

$$= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s | \theta) \sum_{a \in \mathcal{A}} \Pr(A_t = a | S_t = s, \theta) b(s) \frac{\partial \ln(\pi(s, a, \theta))}{\partial \theta} \tag{575}$$

$$= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s | \theta) b(s) \sum_{a \in \mathcal{A}} \pi(s, a, \theta) \frac{\partial \ln(\pi(s, a, \theta))}{\partial \theta} \tag{576}$$

$$= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s | \theta) b(s) \sum_{a \in \mathcal{A}} \frac{\partial \pi(s, a, \theta)}{\partial \theta} \tag{577}$$

$$= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s | \theta) b(s) \frac{\partial}{\partial \theta} \sum_{a \in \mathcal{A}} \pi(s, a, \theta) \tag{578}$$

$$= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s | \theta) b(s) \frac{\partial}{\partial \theta} 1 \tag{579}$$

$$= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s | \theta) b(s) 0 \tag{580}$$

$$= 0. \tag{581}$$

So, inserting the $b(s)$ control variate in (572) did not change the expected value of the update—we still obtain unbiased estimates of the policy gradient. This raises the question: what should we use for b? A common choice is the state-value function: v^θ. This is because we expect $v^\theta(S_t)$ to be similar to G_t, and thus the covariance term when computing the benefit of the control variate to be positive. Hereafter we will use the state-value function as the baseline. Bhatnagar et al. (2009, Lemma 2) showed that the optimal baseline in the average-reward setting is the state-value function, while Weaver and Tao (2001) showed that the optimal constant (state-independent) baseline is the average reward. Although I cannot find it at the moment, I recall hearing that the optimal baseline (minimal-variance baseline) in the discounted start-state setting is *not* exactly the state-value function, but something similar.[11]

[11] If anyone discovers the appropriate reference and posts it on Piazza, I will update this document.

We can estimate the baseline using the TD(λ) algorithm to obtain Algorithm 21.

Algorithm 21: Stochastic Gradient Ascent on J (REINFORCE) including a baseline (control variate). Here α and β are step sizes.

Initialize θ and w arbitrarily;
for *each episode* **do**
  Generate an episode $S_0, A_0, R_0, S_1, A_1, R_1, \ldots, S_{L-1}, A_{L-1}, R_{L-1}$ using policy parameters θ.;
  $\widehat{\nabla J(\theta)} = 0$;
  $e \leftarrow 0$;
  for $t = 0$ *to* $L - 1$ **do**
    $\widehat{\nabla J(\theta)} = \widehat{\nabla J(\theta)} + \gamma^t (G_t - v_w(S_t)) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}$;
    $e \leftarrow \gamma \lambda e + \frac{\partial v_w(S_t)}{\partial w}$;
    $\delta \leftarrow R_t + \gamma v_w(S_{t+1}) - v_w(S_t)$;
    $w \leftarrow w + \alpha \delta e$;
  $\theta \leftarrow \theta + \beta \widehat{\nabla J(\theta)}$;

As in the REINFORCE algorithm without the baseline, you should ignore the γ^t term in the policy update. Also notice that the update using $v_w(S_t)$ as the baseline occurs before w is updated based on data that occurred after S_t. This is to ensure that w is not changed based on A_t, which would in turn make $v_w(S_t)$ depend on A_t, and thus would result in the control variate (baseline) not being mean-zero.

———————End of Lecture 23, December 5, 2019———————

CMPSCI 687 Homework 5

Due December 10, 2019

Late submissions have no penalty until Midnight December 14.

Instructions: This assignment is worth 50 points, and is available through Moodle. Instructions regarding how this assignment will be graded are included on Moodle.

Although REINFORCE with the baseline term is an improvement upon REINFORCE, it still uses a Monte Carlo return, G_t. If we are willing to introduce bias into our gradient estimates in an effort to reduce their variance, then we can replace the Monte Carlo return, G_t, with the TD return, $R_t + \gamma v_w(S_{t+1})$. This results in the update to the gradient estimate:

$$\widehat{\nabla J(\theta)} = \widehat{\nabla J(\theta)} + \gamma^t (R_t + \gamma v_w(S_{t+1}) - v_w(S_t)) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}. \tag{582}$$

If we further reverse the order of the updates so that the TD-error is computed before the gradient estimate is updated, and if we introduce bias by updating the policy parameters at every time step (as Sutton and Barto did in their REINFORCE update), we obtain an Actor-Critic that follows *biased* estimates of the gradient, as presented in Algorithm 22.

Algorithm 22: Basic Actor-Critic

Initialize θ and w arbitrarily;
for *each episode* **do**
  $s \sim d_0$;
  $e \leftarrow 0$;
  for *each time step, until s is the terminal absorbing state* **do**
    `/* Act using the actor */`
    $a \sim \pi(s, \cdot)$;
    Take action a and observe r and s';
    `/* Critic update using TD(λ) */`
    $e \leftarrow \gamma \lambda e + \frac{\partial v_w(s)}{\partial w}$;
    $\delta \leftarrow r + \gamma v_w(s') - v_w(s)$;
    $w \leftarrow w + \alpha \delta e$;
    `/* Actor update */`
    $\theta \leftarrow \theta + \alpha \gamma^t \delta \frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta}$;
    `/* Prepare for next episode */`
    $s \leftarrow s'$;

As in the previous algorithms, the γ^t term in the actor update should not be included in real implementations (Thomas, 2014). Another way to see why this basic actor-critic uses a reasonable update is to consider what would happen if δ were to use v^π rather than an estimate thereof. Specifically, recall from question 25 that $\mathbf{E}[\delta_t | S_t = s, A_t = a, \theta] = q^\theta(s,a) - v^\theta(s)$. Thus, if $v_w = v^\pi$, then the basic actor-critic's update would be, in expectation:

$$\theta \leftarrow \theta + \alpha \gamma^t \mathbf{E}\left[(q^\theta(S_t, A_t) - v^\theta(S_t)) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \middle| \theta \right], \tag{583}$$

which is the policy gradient (with $v^\theta(S_t)$ as a baseline, and if we ignore the fact that changing θ within an episode will change the state distribution).

The basic actor-critic presented above has also been presented with eligiblity traces added to the actor. To the best of my knowledge, there is no principled reason to do so. I believe that this change is similar in effect to modifying the objective function to emphasize obtaining a good policy for states that occur later in an episode, but at this point this is an educated guess. Still, this alternate actor-critic algorithm performs remarkably well. Pseudocode for this actor-critic algorithm, with the pesky γ^t term also removed (this algorithm is so unprincipled, there's no need to pretend we're going for unbiased estimates or a real gradient algorithm—we're going for good performance here) is provided in Algorithm 23.

Algorithm 23: Actor-Critic that looks like a policy gradient algorithm if you squint hard enough (ACTLLAPGAIYSHE)

Initialize θ and w arbitrarily;
for *each episode* **do**
    $s \sim d_0$;
    $e_v \leftarrow 0$;
    $e_\theta \leftarrow 0$;
    for *each time step, until s is the terminal absorbing state* **do**
        /* Act using the actor */
        $a \sim \pi(s, \cdot)$;
        Take action a and observe r and s';
        /* Critic update using TD(λ) */
        $e_v \leftarrow \gamma\lambda e_v + \frac{\partial v_w(s)}{\partial w}$;
        $\delta \leftarrow r + \gamma v_w(s') - v_w(s)$;
        $w \leftarrow w + \alpha\delta e_v$;
        /* Actor update */
        $e_\theta \leftarrow \gamma\lambda e_\theta + \frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta}$;
        $\theta \leftarrow \theta + \beta\delta e_\theta$;
        /* Prepare for next episode */
        $s \leftarrow s'$;

To be clear, Algorithm 23 is not a true policy gradient algorithm because:

1. It ignores the γ^t term that came from the discounted state "distribution".

2. It includes eligiblity traces in the actor update (I am unaware of any analysis of what these traces do theoretically).

3. It uses a value function estimate in place of the true value function.

4. The policy parameters are updated at every time step, and so the resulting state distribution is not d^θ or the on-policy state distribution for any particular policy—it comes from running a mixture of policies.

Still, this algorithm is often referred to as a policy gradient algorithm. This same algorithm (with the γ^t terms implemented via the variable I), appears on page 332 of the second edition of Sutton and Barto's book (Sutton and Barto, 2018).

Note: Assume that the softmax policy's weights take the form of a vector, and that the weights for action 1 are followed by the weights for action 2, the weights for action 2 are followed by the weights for action 3, etc. The derivative of the natural log of this softmax policy is:

$$\frac{\partial \ln \pi(s, a_k, \theta)}{\partial \theta} = \begin{bmatrix} -\pi(s, a_1, \theta)\phi(s) \\ -\pi(s, a_2, \theta)\phi(s) \\ \vdots \\ [1 - \pi(s, a_k, \theta)]\phi(s) \\ \vdots \\ -\pi(s, a_n, \theta)\phi(s) \end{bmatrix}.$$

Note that each line represents $|\phi(s)|$ elements of the vector. This results in a vector of length $n(|\phi(s)|)$ (where $n = |\mathcal{A}|$).

17 Natural Gradient

As mentioned in class, you are not responsible for this lecture (it will not be on any exams, quizzes, or assignments). Also, notes will not be typed up in full (we covered a lot!). Instead, I will give a brief summary and references.

Natural gradients were popularized by Amari in 1998 in two papers (Amari, 1998; Amari and Douglas, 1998). He argued that, when optimizing a function $f(x)$, where x is a vector, you may not want to assume that x lies in Euclidean space. If you want to measure distances differently between inputs, x, natural gradients give you a way to do so. Specifically, if the distance between x and $x+\Delta$ is $\sqrt{\Delta^\intercal G(x)\Delta}$, where $G(x)$ is a positive definite matrix, then the direction of steepest ascent is $G(x)^{-1}\nabla f(x)$. This direction is called the *natural gradient*, and is often denoted by $\widetilde{\nabla} f(x)$. Note that $G(x)$ can be a function of x—we can measure distances differently around different points, x.

This raises the question: what should $G(x)$ be? If the function being optimized is a loss function of a parameterized distribution, e.g., $f(d_\theta)$, where f is the loss or objective function and d is a parameterized distribution, parameterized by vector θ, then Amari argued that the *Fisher information matrix* (FIM), $F(\theta)$, of the parameterized distribution d is a good choice for G. The FIM is defined as:

$$F(\theta) = \sum_x d_\theta(x) \frac{\partial d_\theta(x)}{\partial \theta} \frac{\partial d_\theta(x)}{\partial \theta}^\intercal, \tag{584}$$

where $d_\theta(x)$ denotes the probability of event x under the distribution with parameters θ (e.g., θ could be the mean and variance of a normal distribution).

I do not know who was first to show it, but it has been shown that the Fisher information matrix results in using a second order Taylor approximation of the KL-divergence as the notion of squared distance. A review of these results so far can be found in the introduction to my paper (Thomas et al., 2016), and the appendix includes a derivation of the Fisher information matrix from KL divergence. The introduction to another of my papers (not the remainder of the paper) (Thomas et al., 2018) also provides a clear example of why the "invariance to repraemeterization" or "covariance" property of natural gradient algorithms is desirable.

After Amari introduced the idea of natural gradients, Kakade (2002) showed how it could be used for reinforcement learning. Specifically, he showed that, when using compatible function approximation (this is also not covered in the class notes, but first appears in the paper by Sutton et al. (2000)), the natural gradient is $\widetilde{\nabla} J(\theta) = w$.

That is, if you solve for weights w that are a local minimizer of the loss function:

$$L(w) = \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi(s,a,\theta) \left(q^\pi(s,a) - q_w(s,a)\right)^2, \tag{585}$$

where

$$q_w(s,a) = w^\intercal \frac{\partial \ln(\pi(s,a,\theta))}{\partial\theta}, \tag{586}$$

then the natural gradient is w.

Kakade did not promote a particular algorithm, but soon after many natural actor-critic algorithms were created. These algorithms use different policy-evaluation algorithms and baselines to estimate $q^\pi(s,a)$ with $q_w(s,a)$, and then use the update $w \leftarrow w + \alpha w$. Popular examples include Morimura's linear time *NTD* algorithm (Morimura et al., 2005), which was later reinvented separately (with slight tweaks) by Degris et al. (2012, INAC) and Thomas and Barto (2012, NAC-S) (neither of us knew of Morimura's work at the time). Perhaps the most popular natural actor-critic was that of (Peters and Schaal, 2008), previous variants of which they published earlier (Peters and Schaal, 2006), and which uses *least squares temporal difference* (LSTD), discussed in the last homework assignment, to approximate the value function.[12] Average-reward natural actor-critic algorithms were also created (Bhatnagar et al., 2009), and the TRPO algorithm is a natural actor-critic (the "trust region" part denotes that the step size is measured in terms of the KL-divergence between the policy before and after the step, but the direction of the step is just the natural gradient) (Schulman et al., 2015). The idea of natural gradients has also been applied to value function based methods, resulting in natural variants of q-learning and Sarsa (Choi and Van Roy, 2006; Dabney and Thomas, 2014).

On the theory side, shortly after the derivation of natural gradients for RL by Kakade, Bagnell and Schneider (2003) showed that the Kakade's guess as to what the Fisher information matrix should be is correct. The Fisher information matrix is defined for parameterized distributions, and a policy is one distribution

[12] If you implement their algorithm, note that one version of the paper has an error in the pseudocode (I believe v and w are reversed), and be sure to clear the eligibility trace vector between episodes. You can also use WIS-LSTD (Mahmood et al., 2014) in place of LSTD to better handle the off-policy nature of old data.

per state. Kakade averaged these Fisher information matrices, weighted by the state distribution d^π. Bagnell showed that this is the Fisher information matrix that you get if you view policies as distributions over trajectories, and also proved that natural gradient ascent using the Fisher information matrix is invariant to reparameterization. A connection between natural gradient methods and mirror descent (a convex optimization algorithm) has also been established (Thomas et al., 2013; Raskutti and Mukherjee, 2015). For a discussion of the relation to Newton's method, see the works of Furmston et al. (2016) and Martens (2014).

18 Other Topics

In this lecture we very briefly discussed other topics in reinforcement learning. We began by watching the first 14 minutes of this fantastic TED talk by Gero Miesenboeck, which describes work by Claridge-Chang et al. (2009).

18.1 Hierarchical Reinforcement Learning

For many problems, learning at the level of primitive actions is not sufficient. A human brain implementing ϵ-greedy Q-learning would never learn to play chess if the primitive actions correspond to muscle twitches. Some learning must occur at a higher level—at the level of deciding which skills to apply next. Here skills might correspond to picking up an object, standing up, changing lanes while driving, etc. *Hierarchical reinforcement learning* (HRL) aims to create RL agents that learn a hierarchy of reusable skills, while also learning when these skills should be applied. For the chess example, we might want to learn a skill to grasp an object, a skill to move our arm to a position, a skill that uses these two to move a particular piece, and then a policy that uses all of these skills to play chess. This top-level policy would be learning (and exploring) at the level of moves in a game of chess rather than muscle twitches. Although there are several different frameworks for HRL, one of the most popular is the *options framework*, introduced by Sutton et al. (1999). If you plan on studying reinforcement learning in the future, you should absolutely read their paper in detail.

An open problem in reinforcement learning is determining automatically which skills are worth learning. Should a baby learn a skill to balance a book on its head while standing on one foot during a solar eclipse? Or, would it be more useful for it to learn a skill to walk? How can an RL agent autonomously determine that the skill to walk is useful, while the other is not?

Several heuristic solutions to this problem have been proposed, with notable examples including the work by Simsek and Barto (2008) and Machado et al. (2017). A perhaps more principled approach, which involves solving for the gradient of the expected return with respect to parameterized skills, was proposed recently by Bacon et al. (2017).

18.2 Experience Replay

Q-learning only uses samples once. This is wasteful because some experiences may be rare or costly to obtain. Lin and Mitchell (1992) suggested that an agent might store "experiences" as tuples, (s, a, r, s'), which can then be repeatedly presented to a learning algorithm as if the agent experiences these experiences again. This *experience replay* can improve the data efficiency of algorithms (make them learn faster) and help to avoid *forgetting*. Forgetting occurs when an agent uses function approximation, and some states occur infrequently. If updates to one state change the value for other states (due to the use of function approximation), the updates to infrequent states may be small in comparison to the updates that occur as a side-effect of updates for other more frequent states. Thus, and agent can *forget* what it has learned about states (or state-action pairs) if they are not revisited sufficiently often. Experience replay helps to mitigate this forgetting.

However, experience replay in its standard form is not compatible with eligiblity traces, and so usually experience replay is only used when $\lambda = 0$. This is not necessarily desirable—the DQN algorithm's lack of eligibility traces is not a feature, but an unfortunate consequence of using experience replay (Mnih et al., 2015).

18.3 Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) involves a set of agents acting in the same environment, where the actions of one agent can impact the states and rewards as seen by other agents. Research has studied both cooperative problems, wherein all of the agents obtain the same rewards, and thus work together, as well as more game theoretic problems wherein the agents obtain different rewards, and so some agents might actively work to decrease the expected return for other agents so as to increse their own expected returns. For a review of MARL, see the work of Busoniu et al. (2008).

A common concept in MARL research, and multi-objective machine learning research in general, is the idea of a the *Pareto frontier*. A solution θ, is on the Pareto frontier for a multi-objective problem if there does not exist another solution θ', that causes any of the objectives to increase without decreasing at least one of the other objectives (assuming that larger values are better for all objectives). Formally, if $f_1, \dots, f_n$ are n objective functions, then the Pareto frontier is the set

$$P \coloneqq \left\{ \theta \in \Theta : \forall \theta' \in \Theta, \Big(\exists i \in \{1, \dots, n\},\ f_i(\theta') > f_i(\theta) \implies \exists j \in \{1, \dots, n\},\ f_j(\theta') < f_j(\theta) \Big) \right\}. \tag{587}$$

Solutions on the Pareto frontier provide a balance between the different objectives, and an algorithm should ideally return a solution on the Pareto frontier since any other solution could be improved with respect to at least one objective function without hindering performance with respect to any of the other objective functions. In the context of MARL, the Pareto frontier is a set of joint-policies (a set, where each element contains a policy for all of the agents), such that increasing the expected return for one agent necessarily means that another agent's expected return must decrease. Some MARL research deals with the question of solving for this Pareto frontier (Pirotta et al., 2014).

18.4 Reinforcement Learning Theory

An RL algorithm is said to be *Probably Approximately Correct in Markov Decision Processes* (PAC-MDP) if, with probability at least $1-\delta$, after executing a fixed number of time steps less than some polynomial function of $|\mathcal{S}|$, $|\mathcal{A}|$, $1/\delta$, and $1/(1-\gamma)$, it returns a policy whose expected return is within ϵ of $J(\pi^*)$. The *sample complexity* of an algorithm is this polynomial function. For discussion of PAC-MDP algorithms, see the works of Kearns and Singh (2002), Kakade (2003), and Strehl et al. (2006). Other researchers focus on other theoretical notions of data efficiency, including *regret* (Azar et al., 2017). Although the algorithms that are backed by strong theory in terms of regret and PAC bounds might seem like obvious choices to use in practice, they tend to perform extremely poorly relative to the algorithms mentioned previously when applied to typical problems. An active area of research is the push to make PAC and low-regret algorithms practical (Osband et al., 2016).

18.5 Deep Reinforcement Learning

Deep learning and reinforcement learning are largely orthogonal questions. Deep learning provides a function approximator, and reinforcement learning algorithms describe how to train the weights of an arbitrary function approximator for sequential decision problems (MDPs). That is, deep networks are, from the point of view of reinforcement learning algorithms, simply a non-linear function approximator.

However, there are some special considerations that become important when using deep neural networks to estimate value functions or represent policies. For example, the large number of weights means that linear time algorithms are particularly important. For example, the NAC-LSTD algorithm (Peters and Schaal, 2008), although useful for problems using linear function approximation with a small number of features, is completely impractical for policies with millions of weights due to its quadratic to cubic per-time-step time complexity (as a function of the number of parameters of the policy). Furthermore, the high computation time associated with training deep neural networks has resulted in increased interest in methods for parallization (Mnih et al., 2016).

Also, a notable paper worth reading is that of Mnih et al. (2015), and the follow-up papers by Liang et al. (2016) (which shows that the same results are obtainable using linear function approximation), and Such et al. (2017) (which shows that random search outperforms RL algorithms like DQN for playing Atari 2600 games).

——————End of Lecture 24, December 10, 2019——————

References

S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10:251–276, 1998.

S. Amari and S. Douglas. Why natural gradient? In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 1213–1216, 1998.

T. W. Anderson. Confidence limits for the value of an arbitrary bounded random variable with a continuous distribution function. *Bulletin of The International and Statistical Institute*, 43:249–251, 1969.

M. G. Azar, I. Osband, and R. Munos. Minimax regret bounds for reinforcement learning. *arXiv preprint arXiv:1703.05449*, 2017.

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *AAAI*, pages 1726–1734, 2017.

J. A. Bagnell and J. Schneider. Covariant policy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1019–1024, 2003.

L. Baird. Residual algorithms: reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

L. C. Baird. Advantage updating. Technical Report WL-TR-93-1146, Wright-Patterson Air Force Base, 1993.

L. C. Baird and A. H. Klopf. Reinforcement learning with high-dimensional, continuous actions. Technical Report WL-TR-93-1147, Wright-Patterson Air Force Base, 1993.

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

M. Bastani. Model-free intelligent diabetes management using machine learning. Master's thesis, Department of Computing Science, University of Alberta, 2014.

M. G. Bellemare, G. Ostrovski, A. Guez, P. S. Thomas, and R. Munos. Increasing the action gap: New operators for reinforcement learning. In *AAAI*, pages 1476–1483, 2016.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

D. P. Bertsekas and J. N. Tsitsiklis. Gradient convergence in gradient methods. *SIAM J. Optim.*, 10:627–642, 2000.

S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

L. Busoniu, R. Babuska, and B. D. Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, 2008.

D. Choi and B. Van Roy. A generalized kalman filter for fixed point approximation and efficient temporal-difference learning. *Discrete Event Dynamic Systems*, 16(2):207–239, 2006.

A. Claridge-Chang, R. Roorda, E. Vrontou, L. Sjulson, H. Li, J. Hirsh, and G. Miesenbock. Writing memories with light-addressable reinforcement circuitry. *Cell*, 193(2):405–415, 2009.

D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6. ACM, 1987.

W. Dabney and P. S. Thomas. Natural temporal difference learning. In *AAAI*, pages 1767–1773, 2014.

P. Dayan and T. J. Sejnowski. Td (λ) converges with probability 1. *Machine Learning*, 14(3):295–301, 1994.

T. Degris, P. M. Pilarski, and R. S. Sutton. Model-free reinforcement learning with continuous action in practice. In *Proceedings of the 2012 American Control Conference*, 2012.

R. V. Florian. Correct equations for the dynamics of the cart-pole system. Center for Cognitive and Neural Studies, viewed January 2012, 2007.

T. Furmston, G. Lever, and D. Barber. Approximate Newton methods for policy search in Markov decision processes. *Journal of Machine Learning Research*, 17(227):1–51, 2016.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

T. Jaakkola, M. I. Jordan, and S. P. Singh. Convergence of stochastic iterative dynamic programming algorithms. In *Advances in neural information processing systems*, pages 703–710, 1994.

N. Jiang and L. Li. Doubly robust off-policy evaluation for reinforcement learning. *arXiv preprint*, arXiv:1511.03722v1, 2015.

J. T. Johns. Basis construction and utilization for markov decision processes using graphs. 2010.

S. Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems*, volume 14, pages 1531–1538, 2002.

S. Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University College London, 2003.

M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49(2-3):209–232, 2002.

G. D. Konidaris, S. Niekum, and P. S. Thomas. TD$_\gamma$: Re-evaluating complex backups in temporal difference learning. In *Advances in Neural Information Processing Systems 24*, pages 2402–2410. 2011a.

G. D. Konidaris, S. Osentoski, and P. S. Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence*, pages 380–395, 2011b.

Y. Liang, M. C. Machado, E. Talvitie, and M. Bowling. State of the art control of atari games using shallow reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 485–493. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

L. Lin and T. M. Mitchell. Memory approaches to reinforcement learning in non-Markovian domains. Technical Report CMU-CS-92-138, May 1992.

M. C. Machado, M. G. Bellemare, and M. Bowling. A Laplacian framework for option discovery in reinforcement learning. *arXiv preprint arXiv:1703.00956*, 2017.

A. R. Mahmood, H. Hasselt, and R. S. Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems 27*, 2014.

T. Mandel, Y. Liu, S. Levine, E. Brunskill, and Z. Popović. Offline policy evaluation across representations with applications to educational games. In *Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems*, 2014.

J. Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

A. Maurer and M. Pontil. Empirical Bernstein bounds and sample variance penalization. In *Proceedings of the Twenty-Second Annual Conference on Learning Theory*, pages 115–124, 2009.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

T. Morimura, E. Uchibe, and K. Doya. Utilizing the natural gradient in temporal difference reinforcement learning with eligibility traces. In *International Symposium on Information Geometry and its Application*, pages 256–263, 2005.

A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

T. J. Perkins and D. Precup. A convergent form of approximate policy iteration. In *Advances in neural information processing systems*, pages 1627–1634, 2003.

J. Peters and S. Schaal. Policy gradient methods for robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71:1180–1190, 2008.

M. Pirotta, S. Parisi, and M. Restelli. Multi-objective reinforcement learning with continuous pareto frontier approximation supplementary material. *arXiv preprint arXiv:1406.3497*, 2014.

D. Precup. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, 2000.

G. Raskutti and S. Mukherjee. The information geometry of mirror descent. *IEEE Transactions on Information Theory*, 61(3):1451–1457, 2015.

S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial Intelligence: A Modern Approach*, volume 2. Prentice hall Upper Saddle River, 2003.

S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In *Robotics research. the eleventh international symposium*, pages 561–572. Springer, 2005.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

H. Seijen and R. Sutton. True online td (lambda). In *International Conference on Machine Learning*, pages 692–700, 2014.

P. K. Sen and J. M. Singer. *Large Sample Methods in Statistics An Introduction With Applications*. Chapman & Hall, 1993.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

O. Simsek and A. Barto. Skill characterization based on betweenness. In *Proceedings of the 22nd Annual Conference on Neural Information Processing Systems*, Vancouver, B.C, Canada, December 2008.

S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3):287–308, 2000.

S. Singh, R. Lewis, and A. Barto. Where do rewards come from? In *Proceedings of the 31st Annual Conference of the Cognitive Science Society*, Austin, TX, 2009a.

S. Singh, R. L. Lewis, and A. G. Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pages 2601–2606. Cognitive Science Society, 2009b.

J. Sorg, R. L. Lewis, and S. P. Singh. Reward design via online gradient ascent. In *Advances in Neural Information Processing Systems*, pages 2190–2198, 2010.

A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 881–888. ACM, 2006.

F. Stulp and O. Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. hal-00738463, 2012. URL http://hal.archives-ouvertes.fr/hal-00738463.

F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988a.

R. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112: 181–211, 1999.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988b.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 2 edition, 2018.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063, 2000.

I. Szita and A. Lörincz. Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–2941, 2006.

P. Thomas, B. C. Silva, C. Dann, and E. Brunskill. Energetic natural gradient descent. In *International Conference on Machine Learning*, pages 2887–2895, 2016.

P. S. Thomas. A reinforcement learning controller for functional electrical stimulation of a human arm. Master's thesis, Department of Electrical Engineering and Computer Science, Case Western Reserve University, August 2009.

P. S. Thomas. Bias in natural actor-critic algorithms. In *Proceedings of the Thirty-First International Conference on Machine Learning*, 2014.

P. S. Thomas and A. G. Barto. Motor primitive discovery. In *Procedings of the IEEE Conference on Development and Learning and Epigenetic Robotics*, pages 1–8, 2012.

P. S. Thomas and E. Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, 2016.

P. S. Thomas, M. S. Branicky, A. J. van den Bogert, and K. M. Jagodnik. Application of the actor-critic architecture to functional electrical stimulation control of a human arm. In *Proceedings of the Twenty-First Innovative Applications of Artificial Intelligence*, pages 165–172, 2009.

P. S. Thomas, W. Dabney, S. Mahadevan, and S. Giguere. Projected natural actor-critic. In *Advances in Neural Information Processing Systems 26*, 2013.

P. S. Thomas, S. Niekum, G. Theocharous, and G. D. Konidaris. Policy evaluation using the Ω-return. In *In submission*, 2015a.

P. S. Thomas, G. Theocharous, and M. Ghavamzadeh. High confidence off-policy evaluation. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence*, 2015b.

P. S. Thomas, G. Theocharous, and M. Ghavamzadeh. High confidence policy improvement. In *International Conference on Machine Learning*, 2015c.

P. S. Thomas, C. Dann, and E. Brunskill. Decoupling learning rules from representations. *ICML*, 2018.

J. N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3):185–202, 1994.

J. N. Tsitsiklis and B. Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

H. Van Seijen, A. R. Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton. True online temporal-difference learning. *The Journal of Machine Learning Research*, 17(1):5057–5096, 2016.

C. Watkins. *Learning From Delayed Rewards*. PhD thesis, University of Cambridge, England, 1989.

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

L. Weaver and N. Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 538–545. Morgan Kaufmann Publishers Inc., 2001.

M. A. Wiering. Convergence and divergence in standard and averaging reinforcement learning. In *European Conference on Machine Learning*, pages 477–488. Springer, 2004.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

B. P. Woolf. *Building Intelligent Interactive Tutors: Student-centered strategies for revolutionizing e-learning*. Morgan Kaufmann Publishers, 2010.