CMPSCI 687: Reinforcement Learning
Fall 2018 Class Syllabus, Notes, and Assignments

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

In Fall 2018 I taught a course on reinforcement learning using the whiteboard. Because I used the whiteboard, there were no slides that I could provide students to use when studying. I decided to type up my own notes describing what we covered in class at night after each lecture (lectures ended at 6:45pm). Here you will find these (often hastily written) notes. There are likely errors and this document is not intended to stand on its own (it is a supplement for the lectures).
-Phil

Table of Contents

1 Syllabus

1.1 Class

Class will be held on Tuesdays and Thursdays from 5:30pm–6:45pm in ELab 2, 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_Fall2018.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 9am–10am in his office, room 346 of the computer science building. The location of his office hours may move during the semester, in which case this document will be updated (and it will be announced in class).

1.5 Teaching Assistants and Office Hours

The *teaching assistants* (TAs) this semester will be Nicholas (Nick) Jacek (njacek@cs.umass.edu) and James Kostas (jekostas@cs.umass.edu) Nick will have office hours on Wednesdays and Fridays from 2:00–3:00 in CS Building room 207. James will have office hours on Tuesdays and Thursdays from 2:30–3:30 in CS Building room 207. The TAs will extend their office hours by up to a half hour if students are present and asking questions.

1.6 Office Hours Summary

- **Mondays**: 9am–10am, Philip Thomas, room 346.
- **Tuesdays**: 2:30pm–3:30pm, James Kostas, room 207.
- **Wednesdays**: 2:00pm–3:00pm, Nick Jacek, room 207.
- **Thursdays**: 2:30pm–3:30pm, James Kostas, room 207.
- **Fridays**: 2:00pm–3:00pm, Nick Jacek, room 207.

1.7 Grading

Your grade will have four components:

1. **Homework Assignments** (60%): 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** (10%): There will be a midterm exam on Thursday October 18, from 7pm–9pm in room 135 of the Integrated Science Building (ISB).

4. **Final Exam** (15%): There will be a final exam on Tuesday December 18 from 6pm–8pm in Marcus Hall room 131.

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-).

1.8 Late Policy

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

1.9 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.

1.10 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.

1.11 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.

1.12 Questions About Course Material

We encourage you to ask questions in office hours. Questions may also be asked on Piazza, here. We will try to maintain a maximum time-to-answer of at most 24 hours Monday-Friday. Questions asked on the weekend or during holidays may not be answered until the next work day.

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 round brackets, 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

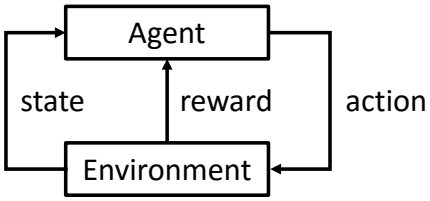


Figure 1: Agent-environment diagram.

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.

2.3 687-Gridworld: A Simple Environment

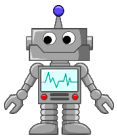
Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	**Goal** State 23

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 $\gamma = 0.9$.

End of Lecture 1, September 4, 2018

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'.

- 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

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

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.

- 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.

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 $t \in \mathbb{N}_{\geq 0}$,

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

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

	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.

deterministic policies (policies for which $\pi(s, a) \in \{0, 1\}$ for all s and a). We denote the set of all policies by Π. Figure 3 presents an example of a policy for 687-Gridworld. We will sometimes write $\pi : \mathcal{S} \to \mathcal{A}$ to denote a deterministic policy, where $A_t = \pi(S_t)$.

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.

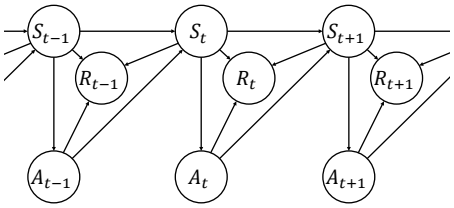


Figure 4: Bayesian network depicted the running of an MDP.

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 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.

[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.

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.*

We will prove Property 1 later.

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?*

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 2, September 6, 2018———————————————

CMPSCI 687 Pop Quiz 1
September 11, 2018

Instructions: You have 10 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 (7 and 8 should be real numbers as your answers, not math expressions):

[Answers in blue.]

1. $P(s, a, s') = \Pr(S_{t+1} = s' | S_t = s, A_t = a)$
2. $R(s, a) = \mathbf{E}[R_t | S_t = s, A_t = a]$
3. $\pi(s, a) = \Pr(A_t = a | S_t = s)$
4. $d_0(s) = \Pr(S_0 = s)$
5. $J(\pi) = \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t R_t | \pi]$
6. $\pi^\star \in \arg\max_{\pi \in \Pi} J(\pi)$
7. $P(s_\infty, a, s_\infty) = 1$
8. $\max_{a \in \mathcal{A}} R(s_\infty, a) = 0$
9. (True or False) an MDP is a mathematical formalization of an agent.
10. (True or False) RL problems are problems where only evaluative feedback is available, not instructive feedback. (Due to valid student argument, false is accepted for this problem.)

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:

$$H_t \coloneqq (S_0, A_0, R_0, S_1, A_t, R_1, S_2, \ldots, S_t, A_t, R_t). \tag{19}$$

- 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 5.

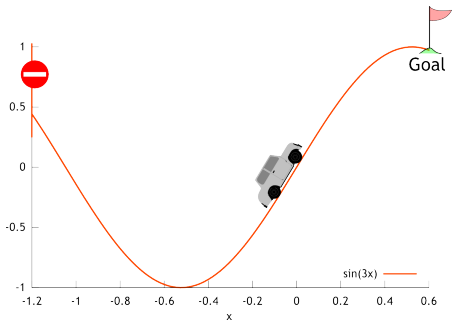


Figure 5: Diagram of the mountain car domain.

- **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: $a \in \{0, 1, 2\}$.
- **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:

$$v_{t+1} = v_t + 0.001a - 0.0025\cos(3x) \tag{20}$$

$$x_{t+1} = x_t + v_{t+1}. \tag{21}$$

  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 5. *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 5. *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:

$$P(h, s, a, s') \coloneqq \Pr(S_{t+1} = s' | H_{t-1} = h, S_t = s, A_t = a). \tag{22}$$

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:[2]

$$\Pr(S_{t+1} = s' | H_{t-1} = h, S_t = s, A_t = a) = \Pr(S_{t+1} = s' | S_t = s, A_t = a) \tag{23}$$

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 (22). 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 6. *Can you give examples of Markovian and non-Markovian environments?*

Question 7. *Is the Markov property a property of the problem being formulated as an MDP or a property of the state representation used when 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.

[2]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 $s \in \mathcal{S}$".

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:

$$\Pr(S_{t+1} = s' | S_t = s, A_t = a) = \Pr(S_{i+1} = s' | S_i = s, A_i = a). \tag{24}$$

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. 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*.

End of Lecture 3, September 11, 2018

CMPSCI 687 Homework 1

Due September 20, 2018, 11:55pm Eastern Time

Instructions: This homework assignment consists of a written portion and a programming portion. 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 on Moodle as a .zip (.gz, .tar.gz, etc.) file containing your answers in a .pdf file and a folder with your source code. Include with your source code instructions for how to run your code. You may not use any reinforcement learning or machine learning specific libraries in your code (you may use libraries like C++ Eigen and numpy though). If you are unsure whether you can use a library, ask on Piazza. If you submit by September 25, you will not lose any credit. The automated system will not accept assignments after 11:55pm on September 25.

Part One: Written (65 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:

$$\Pr(A_0 = a) = \sum_{s \in \mathcal{S}} d_0(s)\pi(s, a). \tag{25}$$

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:** $\Pr(a|b) = \Pr(b|a)\Pr(a)/\Pr(b)$. 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 easy to deal with the 3 terms in $\Pr(A_0 = a_0|S_0 = s_0)\Pr(S_0 = s_0)/\Pr(A_0 = a_0)$.
- **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 $s \in \mathcal{S}$.

$$\begin{aligned}
\Pr(S_1 = s) &= \sum_{s_0 \in \mathcal{S}} \Pr(S_0 = s_0)\Pr(S_1 = s|S_0 = s_0) && (26)\\
&= \sum_{s_0 \in \mathcal{S}} d_0(s_0)\Pr(S_1 = s|S_0 = s_0) && (27)\\
&= \sum_{s_0 \in \mathcal{S}} d_0(s_0) \sum_{a_0 \in \mathcal{A}} \Pr(A_0 = a_0|S_0 = s_0) && (28)\\
&\quad \times \Pr(S_1 = s|S_0 = s_0, A_0 = a_0) && (29)\\
&= \sum_{s_0 \in \mathcal{S}} d_0(s_0) \sum_{a_0 \in \mathcal{A}} \pi(s_0, a_0)P(s_0, a_0, s). && (30)
\end{aligned}$$

Problems:

- The probability that the state at time $t = 3$ is either $s \in \mathcal{S}$ or $s' \in \mathcal{S}$.

$$\Pr(S_3 = s \cup S_3 = s') = \sum_{s_0} d_0(s_0) \sum_{a_0} \pi(s_0, a_0) \sum_{s_1} P(s_0, a_0, s_1) \tag{31}$$

$$\times \sum_{a_1} \pi(s_1, a_1) \sum_{s_2} P(s_1, a_1, s_2) \tag{32}$$

$$\times \sum_{a_2} \pi\big(s_2, a_2)(P(s_2, a_2, s) + P(s_2, a_2, s')\big) \tag{33}$$

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

$$\Pr(A_{16} = a' | S_{14} = s, A_{15} = a) \tag{34}$$

$$= \frac{\Pr(A_{16} = a', A_{15} = a | S_{14} = s)}{\Pr(A_{15} = a | S_{14} = s)} \tag{35}$$

$$= \frac{\sum_{a_{14}} \pi(s, a_{14}) \sum_{s_{15}} P(s, a_{14}, s_{15}) \pi(s_{15}, a) \sum_{s_{16}} P(s_{15}, a, s_{16}) \pi(s_{16}, a')}{\sum_{a_{14}} \pi(s, a_{14}) \sum_{s_{15}} P(s, a_{14}, s_{15}) \pi(s_{15}, a)}. \tag{36}$$

- The expected reward at time $t = 6$ given that the action at time $t = 3$ is $a \in \mathcal{A}$, and the state at time $t = 5$ is $s \in \mathcal{S}$.

$$\mathbf{E}\left[R_6 | A_3 = a, S_5 = s\right] = \sum_{a_5} \pi(s, a_5) \sum_{s_6} P(s, a_5, s_6) \sum_{a_6} \pi(s_6, a_6) R(s_6, a_6) \tag{37}$$

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

$$\Pr(S_0 = s | S_1 = s') = \frac{\Pr(S_0 = s, S_1 = s')}{\Pr(S_1 = s')} \tag{38}$$

$$= \frac{d_0(s) \sum_{a_0} \pi(s, a_0) P(s, a_0, s')}{\sum_{s_0} d_0(s_0) \sum_{a_0} \pi(s, a_0) P(s_0, a_0, s')} \tag{39}$$

- The probability that the action at time $t = 5$ is $a \in \mathcal{A}$ given that the initial state is $s \in \mathcal{S}$, the state at time $t = 5$ is $s' \in \mathcal{S}$, and the action at time $t = 6$ is $a' \in \mathcal{A}$.

$$\Pr(A_5 = a | S_0 = s, S_5 = s', A_6 = a') = \Pr(A_5 = a | s_5 = s', A_6 = a') \tag{40}$$

$$= \frac{\Pr(A_5 = a, A_6 = a' | s_5 = s')}{\Pr(A_6 = a' | S_5 = s')} \tag{41}$$

$$= \frac{\pi(s', a) \sum_{s_6} P(s', a, s_6) \pi(s_6, a')}{\sum_{a_5} \pi(s', a_5) \sum_{s_6} P(s', a_5, s_6) \pi(s_6, a')} \tag{42}$$

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 -1, 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 $S_t = 17, A_t = \text{AL}, S_{t+1} = 17$.

$$\mathbf{E}[R_t | S_t = 17, A_t = \text{AL}, S_{t+1} = 17] = \Pr(\text{HitWall} | 17\text{AL}17) \mathbf{E}[R_t | 17\text{AL}17, \text{HitWall}] \tag{43}$$

$$+ \Pr(\neg\text{HitWall} | 17\text{AL}17) \mathbf{E}[R_t | 17\text{AL}17, \neg\text{HitWall}] \tag{44}$$

$$= \Pr(\text{HitWall} | 17\text{AL}17)(-1) + \Pr(\neg\text{HitWall} | 17\text{AL}17)(0) \tag{45}$$

$$= -\Pr(\text{HitWall} | 17\text{AL}17). \tag{46}$$

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.

$$\Pr(\text{HitWall}|17\text{AL}17) = \frac{\Pr(\text{HitWall}, S_{t+1} = 17|S_t = 17, A_t = \text{AL})}{\Pr(S_{t+1} = 17|S_t = 17, A_t = \text{AL})} \tag{47}$$

$$= \frac{\Pr(\text{HitWall}|S_t = 17, A_t = \text{AL})}{\Pr(S_{t+1} = 17|S_t = 17, A_t = \text{AL})} \tag{48}$$

$$= \frac{0.8}{0.8 + 0.1} \tag{49}$$

$$= \frac{0.8}{0.9}. \tag{50}$$

So,

$$\mathbf{E}[R_t|S_t = 17, A_t = \text{AL}, S_{t+1} = 17] = -\frac{0.8}{0.9}. \tag{51}$$

4. (2 Points) How many stochastic 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 $|\mathcal{A}|$).

   Infinite.

5. (5 Points) Create an MDP (which may not have finite state or action sets) that does *not* have an optimal policy. The rewards for your MDP must be bounded, and it must use $\gamma < 1$.

   Student answers may vary. One example is 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.

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. One example is a robot learning to balance. The set of states is the set of all possible configurations of the robot. The set of actions is each possible amount of power that could be given to each motor, P captures the dynamics described by physics, R might be 0 if the force sensors indicate that the robot has fallen over, and +1 otherwise, d_0 might be the initial distribution of positions that results from me trying to set up the robot to a near-standing position, and $\gamma = 0.99$. This example is partially observable because the sensors on the robot will not allow it to precisely measure the true state of the robot—the agent will not fully observe the state.

9. (5 Points) Create an MDP for which there exist at least two optimal policies that have different variance of their returns. Describe the two optimal policies and derive the expected value and variance of their returns.

   Student answers may vary. One example is an MDP with one state that always transitions to s_∞ and which has two actions. The first actions results in $R_t \sim N(0, 1)$ and the second results in $R_t \sim N(0, 2)$.

10. (2 Points) Create an MDP that always terminates, but which has no terminal states other than s_∞.

    Student answers may vary. One example is an MDP with a single state that always transitions to s_∞ with probability 0.5.

11. (2 Points) Read the Wikipedia page on Markov chains. 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,

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

    An MDP with a fixed policy results in state transitions that can be described as a Markov chain. Create an MDP and a policy that result in a state having a period of 3.

    Student answers may vary. One example is an MDP with three states, where s_1 always transitions to s_2, which always transitions to s_3, which always transitions back to s_1.

12. (5 Points) 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 deterministic policy is irredicible. A Markov chain is *aperiodic* if the period of every state is $k = 1$. 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. A Markov chain is *ergodic* if it is *aperiodic* and *positive recurrent*. An MDP is *ergodic* if the Markov chain associated with every deterministic policy is ergodic. Create an MDP that is ergodic, but *not* irreducible.

    Student answers may vary. One example is an MDP with two states, s_1 and s_2, where s_1 always transitions back to s_1, and s_2 always transitions back to s_2.

13. (3 Points) Create an MDP that is not ergodic.

    Student answers may vary. One example is 687-Gridworld.

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}.

    Student answers may vary. One example is digital marketing, as described in class. The state might be a vector of features that describe a person, the action might be the advertisement shown to the user, and the reward might be +1 if the user clicks on the advertisement, and 0 otherwise. The next state might always be s_∞ if we view the problem as a bandit problem. For this problem, whether the person clicks the advertisement or not is not a deterministic function of the feature vector describing the person and the advertisement shown.

15. (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 might 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 is S_t is a state that is not considered to be "fallen over".

16. (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+1}.

    Student answers may vary. One example is our example from the previous question, but where we choose to define R_t to be +1 if S_{t+1} is a state that is not considered to be "fallen over".

17. (2 Points) Describe a real-world problem and how it can be reasonably modeled as an MDP where the reward function, R, would be known.

    Student answers may vary. The examples over a robot learning not to fall over again applies.

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.

    Student answers may vary. The digital marketing example is one possible example. We do not know the distribution over whether or not a person will click on an advertisement.

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-understoon. We can view these as being problems where the transition dynamics are known.

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 behavior—how 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)

Implement the 687-Gridworld domain described in class and in the class notes. Have the agent select actions uniformly randomly.

- (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.

  Sample correct answers (correct answers will vary slightly): mean: -.55, standard deviation: 2.18, min: -36.59, max: 4.78

- (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.

$$\begin{bmatrix} right & right & right & down & down \\ right & right & right & down & down \\ up & up & XXXXX & down & down \\ up & up & XXXXX & down & down \\ up & up & right & right & up \end{bmatrix} \tag{52}$$

- (10 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.

- (5 Points) Using simulations, empirically estimate the probability that $S_{19} = 21$ (the state with water) 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).

$$\sim 0.02 \tag{53}$$

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 6. 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: $\mathcal{A} = \{\text{left}, \text{right}\}$.

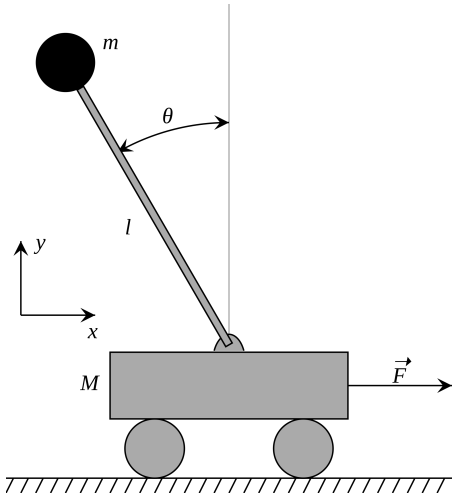


Figure 6: Cart-Pole Domain.

- $R_t = 1$ 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, when the pole falls down, or when the cart hits the end of the track, which spans from -3 to 3. Time is simulated with time steps of $\Delta t = 0.02$ seconds.

Question 8. *Is the optimal policy for cart-pole unique?*

Answer 8. *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 9. *Is the state representation Markovian?*

Answer 9. *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

$$\forall t \geq L, \Pr(S_t = s_\infty) = 1. \tag{54}$$

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 10. *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?*

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 11. *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:

$$\arg\max_{x \in \mathbb{R}^n} f(x), \tag{55}$$

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:

$$\arg\max_{\pi \in \Pi} J(\pi). \tag{56}$$

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:

$$\hat{J}(\pi) \coloneqq \frac{1}{N}\sum_{i=1}^{N} G^i \tag{57}$$

$$= \frac{1}{n}\sum_{i=1}^{N}\sum_{t=0}^{\infty} \gamma^t R_t^i, \tag{58}$$

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:

$$\pi(s,a) \coloneqq \frac{e^{\sigma p(s,a)}}{\sum_{a'} e^{\sigma p(s,a')}}, \tag{59}$$

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 $\sigma = 1$, and note that uses of the symbol σ in future sections of the notes often refer to other hyperparameters (e.g., later in the first-choice hill-climbing algorithm, σ is a different parameter). 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 $s \in \mathcal{S}$:

$$\sum_{a\in\mathcal{A}} \pi(s,a) = \sum_{a\in\mathcal{A}} \frac{e^{\sigma\pi(s,a)}}{\sum_{a'\in\mathcal{A}} e^{\sigma p(s,a')}} \tag{60}$$

$$= \frac{\sum_{a\in\mathcal{A}} e^{\sigma\pi(s,a)}}{\sum_{a'\in\mathcal{A}} e^{\sigma p(s,a')}} \tag{61}$$

$$= 1. \tag{62}$$

Second, for all s and a, $e^{\sigma p(s,a)} > 0$, and so all terms in the numerator and denominator of (59) 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:

$$\pi(s,a,\theta) \coloneqq \frac{e^{\sigma\theta_{s,a}}}{\sum_{a'} e^{\sigma\theta_{s,a'}}}. \tag{63}$$

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

$$J(\theta) \coloneqq \mathbf{E}[G|\theta], \tag{64}$$

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:

$$\theta^\star \in \arg\max_{\theta \in \mathbb{R}^n} J(\theta). \tag{65}$$

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:

$$\text{injection size} = \frac{\text{current blood glucose} - \text{target blood glucose}}{\theta_1} + \frac{\text{meal size}}{\theta_2}, \tag{66}$$

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).

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 (55), 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.

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 $\theta_1, \ldots, \theta_K$ 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 [for example, σI, for some $\sigma \in \mathbb{R}$]
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**
    $\theta_k \sim N(\theta, \Sigma)$;
    $\hat{J}_k = \texttt{evaluate}(\theta_k, N)$;
  $\texttt{sort}((\theta_1, \hat{J}_1), (\theta_2, \hat{J}_2), \ldots, (\theta_K, \hat{J}_K), \text{descending})$;
  $\theta = \frac{1}{K_e} \sum_{k=1}^{K_e} \theta_k$;
  $\Sigma = \frac{1}{\epsilon + K_e} \left(\epsilon I + \sum_{k=1}^{K_e} (\theta_k - \theta)(\theta_k - \theta)^\intercal \right)$;

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, $G^1, G^2, \ldots, G^N$, where $G^i = \sum_{t=0}^{\infty} \gamma^t R_t^i$;
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, $N = 10$]

$\hat{J} = \texttt{evaluate}(\theta, N)$;
while `true` **do**
  $\theta' \sim N(\theta, \sigma I)$;
  $\hat{J}' = \texttt{evaluate}(\theta', N)$;
  if $\hat{J}' > \hat{J}$ **then**
    $\theta = \theta'$;
    $\hat{J} = \hat{J}'$;

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 7.

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

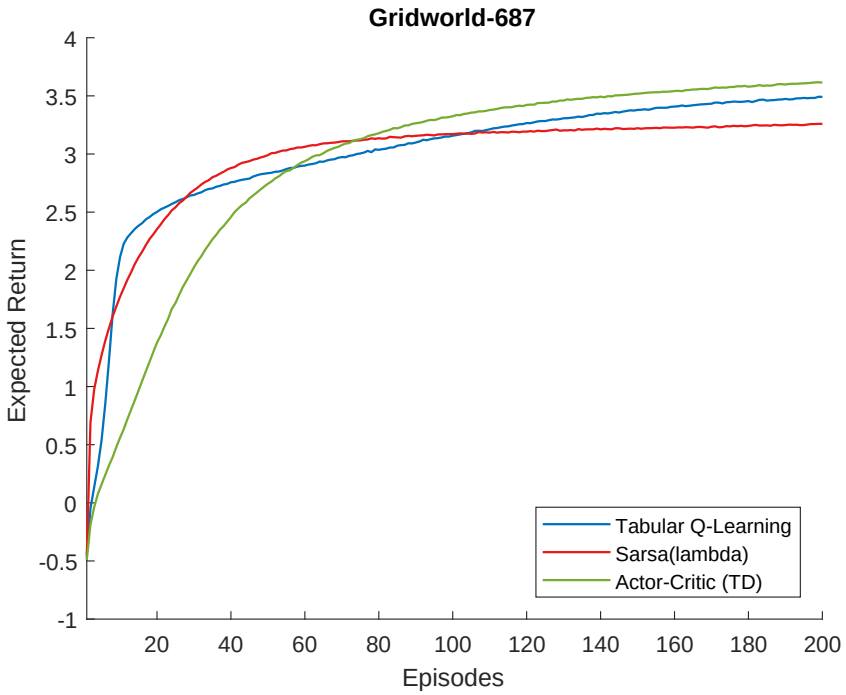


Figure 7: Example learning curve. The horizontal axis shows the number of episodes that the agent has been learning for, and the vertical axis shows the mean return. This plot is often confusing to students, so we will go through it very precisely. Consider running an algorithm for 200 episodes. During the first episode run, there was some resulting return, which we plot at the 1-mark on the horizontal axis. The second episode resulted in a return, which we plot on the 2-mark, etc. This process would result in a learning curve, and we refer to this process as a *trial*. Imagine if we did this entire process a second time—we ran a second trial. This would result in a *different* learning curve due to the random nature of how policies sample actions, as well as how states transition and how rewards are generated. To get a good idea of how the algorithm performs on average, we run a large number of trials and plot the mean learning curve—the point at the horizontal hash mark at 20 episodes is the average return during the 20th episode during all of the trials that were run. To show how much variance there is in performance, we include error bars showing the standard deviation or standard error of returns. This plot was averaged over 100,000 trials. Notice that these trials all use the same hyperparameters—this plot does not show the sensitivity of the algorithm to its hyperparameters. 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.

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

———————————End of Lecture 4, September 18, 2018———————————

CMPSCI 687 Pop Quiz 2
September 20, 2018

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, and the real-number answers for 9 and 10:

[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. $d_0(s) = \Pr(S_0 = s)$
5. $J(\pi) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t | \pi\right]$
6. $G = \sum_{t=0}^{\infty} \gamma^t R_t$
7. $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$
8. $\pi^\star \in \arg\max_{\pi \in \Pi} J(\pi)$
9. $\min_{s \in \mathcal{S}} \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') = 1$
10. $\max_{a \in \mathcal{A}} R(s_\infty, a) = 0$

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:

$$v^\pi(s) \coloneqq \mathbf{E}\left[G_t | S_t = s, \pi\right]. \tag{67}$$

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 (67), even though it does not appear on the left side. This is because the right side takes the same value for all t, and thus the following definitions are equivalent:

$$v^\pi(s) \coloneqq \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, \pi\right] \tag{68}$$

$$v^\pi(s) \coloneqq \mathbf{E}\left[G | S_0 = s, \pi\right] \tag{69}$$

$$v^\pi(s) \coloneqq \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| S_0 = s, \pi\right]. \tag{70}$$

You will prove that these latter two definitions are equivalent to the first two definitions in Homework 2.

As an example, consider the MDP depicted in Figure 8. For this MDP:

$$v^{\pi_1}(s_1) = 0 \tag{71}$$

$$v^{\pi_1}(s_2) = 12\gamma^0 = 12 \tag{72}$$

$$v^{\pi_1}(s_3) = 0\gamma^0 + 12\gamma^1 = 6 \tag{73}$$

$$v^{\pi_1}(s_4) = 0\gamma^0 + 0\gamma^1 + 12\gamma^2 = 3 \tag{74}$$

$$v^{\pi_1}(s_5) = 0\gamma^0 + 0\gamma^1 + 0\gamma^2 + 12\gamma^3 = 1.5 \tag{75}$$

$$v^{\pi_1}(s_6) = 0 \tag{76}$$

$$v^{\pi_2}(s_1) = 0 \tag{77}$$

$$v^{\pi_2}(s_2) = 0\gamma^0 + 0\gamma^1 + 0\gamma^2 + 2\gamma^3 = 1/4 \tag{78}$$

$$v^{\pi_2}(s_3) = 0\gamma^0 + 0\gamma^1 + 2\gamma^2 = 1/2 \tag{79}$$

$$v^{\pi_2}(s_4) = 0\gamma^0 + 2\gamma^1 = 1 \tag{80}$$

$$v^{\pi_2}(s_5) = 2\gamma^0 = 2 \tag{81}$$

$$v^{\pi_2}(s_6) = 0. \tag{82}$$

4.2 Action-Value Function

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

$$q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \tag{83}$$

$$q^\pi(s, a) \coloneqq \mathbf{E}[G_t | S_t = s, A_t = a, \pi], \tag{84}$$

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

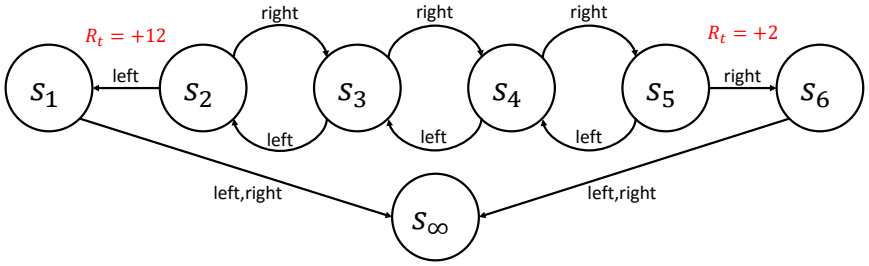


Figure 8: A simple MDP that we will call the "chain" MDP. There are many "chain" MDPs used in the RL literature—this is not a standard one. In each state the agent can choose to move left (L) or right (R), and the transition function is deterministic in implementing these transitions. In states s_1 and 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.

discounted return if the agent takes action a in state s and follows the policy π thereafter. Equivalent definitions of q^π are:

$$q^\pi(s,a) \coloneqq \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, A_t = a, \pi\right] \tag{85}$$

$$q^\pi(s,a) \coloneqq \mathbf{E}\left[G | S_0 = s, A_0 = a, \pi\right] \tag{86}$$

$$q^\pi(s,a) \coloneqq \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| S_0 = s, A_0 = a, \pi\right]. \tag{87}$$

For the chain MDP depicted in Figure 8:

$$q^{\pi_1}(s_1, L) = 0 \tag{88}$$

$$q^{\pi_1}(s_1, R) = 0 \tag{89}$$

$$q^{\pi_1}(s_2, L) = 12\gamma^0 = 12 \tag{90}$$

$$q^{\pi_1}(s_2, R) = 0\gamma^0 + 0\gamma^1 + 12\gamma^2 = 3 \tag{91}$$

$$q^{\pi_1}(s_3, L) = 0\gamma^0 + 12\gamma^1 = 6 \tag{92}$$

$$q^{\pi_1}(s_3, R) = 0\gamma^0 + 0\gamma^1 + 0\gamma^2 + 12\gamma^3 = 1.5. \tag{93}$$

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 (91)—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:

$$v^\pi(s) := \mathbf{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k}\middle| S_t = s, \pi\right] \tag{94}$$

$$= \mathbf{E}\left[R_t + \sum_{k=1}^{\infty}\gamma^k R_{t+k}\middle| S_t = s, \pi\right] \tag{95}$$

$$= \mathbf{E}\left[R_t + \gamma\sum_{k=1}^{\infty}\gamma^{k-1} R_{t+k}\middle| S_t = s, \pi\right] \tag{96}$$

$$\overset{(a)}{=} \sum_{a\in\mathcal{A}}\pi(s,a)R(s,a) + \mathbf{E}\left[\gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle| S_t = s, \pi\right] \tag{97}$$

$$\overset{(b)}{=} \sum_{a\in\mathcal{A}}\pi(s,a)R(s,a) + \sum_{a\in\mathcal{A}}\pi(s,a)\sum_{s'\in\mathcal{S}}P(s,a,s')\mathbf{E}\left[\gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle| S_t = s, A_t = a, S_{t+1} = s', \pi\right] \tag{98}$$

$$\overset{(c)}{=} \sum_{a\in\mathcal{A}}\pi(s,a)R(s,a) + \sum_{a\in\mathcal{A}}\pi(s,a)\sum_{s'\in\mathcal{S}}P(s,a,s')\gamma\mathbf{E}\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle| S_{t+1} = s', \pi\right] \tag{99}$$

$$\overset{(d)}{=} \sum_{a\in\mathcal{A}}\pi(s,a)R(s,a) + \sum_{a\in\mathcal{A}}\pi(s,a)\sum_{s'\in\mathcal{S}}P(s,a,s')\gamma v^\pi(s') \tag{100}$$

$$= \sum_{a\in\mathcal{A}}\pi(s,a)\sum_{s'\in\mathcal{S}}P(s,a,s')\left(R(s,a) + \gamma v^\pi(s')\right), \tag{101}$$

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 (67)).

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

$$v^\pi(s) = \sum_{a\in\mathcal{A}}\pi(s,a)\sum_{s'\in\mathcal{S}}P(s,a,s')\left(R(s,a) + \gamma v^\pi(s')\right). \tag{102}$$

End of Lecture 5, September 20, 2018

CMPSCI 687 Homework 2

Due October 8, 2018, 11:55pm Eastern Time

Instructions: This homework assignment consists of a written portion and a programming portion. 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 on Moodle as a .zip (.gz, .tar.gz, etc.) file containing your answers in a .pdf file and a folder with your source code. Include with your source code instructions for how to run your code. You may not use any reinforcement learning or machine learning specific libraries in your code (you may use libraries like C++ Eigen and numpy though). If you are unsure whether you can use a library, ask on Piazza. If you submit by October 14, you will not lose any credit. The automated system will not accept assignments after 11:55pm on October 14.

Part One: Written (25 Points Total)

1. (5 Points) Prove that the following two definitions of the state-value function are equivalent:

$$v^\pi(s) \coloneqq \mathbf{E}\left[G_t | S_t = s, \pi\right] \tag{103}$$
$$v^\pi(s) \coloneqq \mathbf{E}\left[G | S_0 = s, \pi\right]. \tag{104}$$

Let us denote the first definition as $v_t^\pi(s)$ and the second as $v_0^\pi(s)$.

$$\begin{aligned}
v_t^\pi(s) =& \mathbf{E}[G_t | S_t = s, \pi] \\
=& \sum_{k=0}^{\infty} \gamma^k \mathbf{E}[R_{t+k} | S_t = s, \pi] \\
=& \sum_{a\in\mathcal{A}} \pi(s,a)(R(s,a) + \sum_{k=1}^{\infty} \gamma^k \mathbf{E}[R_{t+k} | S_t = s, \pi]) \\
=& \sum_{a\in\mathcal{A}} \pi(s,a)\left[R(s,a) + \sum_{s'\in\mathcal{S}} P(s,a,s') \sum_{a'\in\mathcal{A}} \pi(s',a')(\gamma^1 R(s',a') + \sum_{k=2}^{\infty} \gamma^k \mathbf{E}[R_{t+k} | S_t = s, \pi])\right] \\
=& \gamma^0 \sum_{a\in\mathcal{A}} \pi(s,a) R(s,a) + \gamma^1 \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s') \sum_{a'\in\mathcal{A}} \pi(s',a') R(s',a') \\
& + \gamma^2 \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s') \sum_{a'\in\mathcal{A}} \pi(s',a') \sum_{s''\in\mathcal{S}} P(s',a',s'') \sum_{a''\in\mathcal{A}} \pi(s'',a'') R(s'',a'') + \dots \\
=& \gamma^0 \sum_{a\in\mathcal{A}} \Pr(A_0 = a | S_0 = s) R(s,a) \\
& + \gamma^1 \sum_{a\in\mathcal{A}} \Pr(A_0 = a | S_0 = s) \sum_{s'\in\mathcal{S}} \Pr(S_1 = s' | A_0 = a, S =_0 = s) \sum_{a'\in\mathcal{A}} \Pr(A_1 = a' | S_1 = s') R(s',a') + \dots \\
=& \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, \pi] \\
=& \mathbf{E}[G | S_0 = s, \pi] \\
=& v_0^\pi(s).
\end{aligned}$$

We will accept less thorough/more trivial answers for full credit. For example:

- Plugging in $t = 0$ for v_t.
- Only expanding one step and then doing a correct, if not rigorous, switch.
- Even just a correct English explanation using the Markov Property.

2. (5 Points) Prove that the following two definitions of the action-value function are equivalent:

$$q^\pi(s,a) \coloneqq \mathbf{E}\left[G_t | S_t = s, A_t = a, \pi\right] \tag{105}$$
$$q^\pi(s,a) \coloneqq \mathbf{E}\left[G | S_0 = s, A_0 = a, \pi\right]. \tag{106}$$

Very similar to question 1; use q instead of v, so no need to sum over the first action.

3. (5 Points) Let M and M' be two MDPs that are identical except for their initial state distributions. Prove that v^π is the same for both MDPs. You may write v^π_M and $v^\pi_{M'}$ to denote the state-value functions for π on M and M' respectively.

Let us write the first MDP's transition and reward functions as P and R, and the second's as P' and R'. Note that $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}, \forall s' \in \mathcal{S}, P(s,a,s') = P'(s,a,s')$ and $R(s,a) = R'(s,a)$.

$$
\begin{aligned}
v^\pi_M(s) =& \mathbf{E}[G_t | S_t = s, \pi] \\
=& \sum_{k=0}^{\infty} \gamma^k \mathbf{E}[R_{t+k} | S_t = s, \pi] \\
=& \sum_{a \in \mathcal{A}} \pi(s,a)(R(s,a) + \sum_{k=1}^{\infty} \gamma^k \mathbf{E}[R_{t+k} | S_t = s, \pi]) \\
=& \sum_{a \in \mathcal{A}} \pi(s,a) \left[R(s,a) + \sum_{s' \in \mathcal{S}} P(s,a,s') \sum_{a' \in \mathcal{A}} \pi(s',a')(\gamma^1 R(s',a') + \sum_{k=2}^{\infty} \gamma^k \mathbf{E}[R_{t+k} | S_t = s, \pi]) \right] \\
=& \gamma^0 \sum_{a \in \mathcal{A}} \pi(s,a) R(s,a) + \gamma^1 \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s') \sum_{a' \in \mathcal{A}} \pi(s',a') R(s',a') \\
& + \gamma^2 \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s') \sum_{a' \in \mathcal{A}} \pi(s',a') \sum_{s'' \in \mathcal{S}} P(s',a',s'') \sum_{a'' \in \mathcal{A}} \pi(s'',a'') R(s'',a'') + \dots \\
=& \gamma^0 \sum_{a \in \mathcal{A}} \pi(s,a) R'(s,a) + \gamma^1 \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P'(s,a,s') \sum_{a' \in \mathcal{A}} \pi(s',a') R'(s',a') \\
& + \gamma^2 \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P'(s,a,s') \sum_{a' \in \mathcal{A}} \pi(s',a') \sum_{s'' \in \mathcal{S}} P'(s',a',s'') \sum_{a'' \in \mathcal{A}} \pi(s'',a'') R'(s'',a'') + \dots \\
\overset{\textbf{(a)}}{=}& v^\pi_{M'}(s).
\end{aligned}
$$

Where **(a)** follows from reversing the first 4 steps.
Like questions 1 and 2, we will accept more trivial or less rigorous answers for full credit as long as they are basically correct.

4. (2 Points) Prove that multiplying all rewards (of a finite MDP with bounded rewards and $\gamma < 1$) by a positive scalar does not change which policies are optimal.

Consider a finite MDP, M, with bounded rewards. If $\pi^\star$ is an optimal policy for M, then we will show that it is an optimal policy for all MDPs M_α that are identical to M, except that their rewards, R'_t are a positive constant times the rewards of M, i.e., $R'_t = \alpha R_t$ for some $\alpha > 0$.

Let $J_\alpha(\pi) \coloneqq \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \alpha R_t\right]$. That is, J_α is the objective function for the MDP M_α, and $J_1 = J$ (recall that J is the objective function for M, as defined in class). If $\pi^\star$ is an optimal policy for M, then for all $\pi \in \Pi$:

$$J(\pi^\star) \geq J(\pi) \tag{107}$$

$$\mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| \pi^\star\right] \geq \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| \pi\right]. \tag{108}$$

Multiplying both sides by α we have that for all $\pi \in \Pi$:

$$\alpha \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| \pi^\star\right] \geq \alpha \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| \pi\right] \tag{109}$$

$$\mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \alpha R_t \middle| \pi^\star\right] \geq \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \alpha R_t \middle| \pi\right] \tag{110}$$

$$J_\alpha(\pi^\star) \geq J_\alpha(\pi). \tag{111}$$

Thus, $\pi^\star$ is a maximizer of J_α and therefore an optimal policy for M_α.

5. (2 Points) Prove that adding a positive constant to all rewards (of a finite MDP with bounded rewards and $\gamma < 1$) can change which policies are optimal.

Consider the MDP shown in Figure 9. S0 is the start state. π_1 denotes the policy that takes action 1, and π_2 denotes the policy that takes action 2. Initially, let every transition give a deterministic reward of -1. Then, $J(\pi_1) = -1$ and $J(\pi_2) = -2$, so π_1 is the optimal policy. Next, add 2 to the reward given by every transition, so every transition now gives a reward of 1. In this case, $J(\pi_1) = 1$ and $J(\pi_2) = 2$. Now, π_2 is the optimal policy; this shows that adding a constant to all rewards can change which policies are optimal.

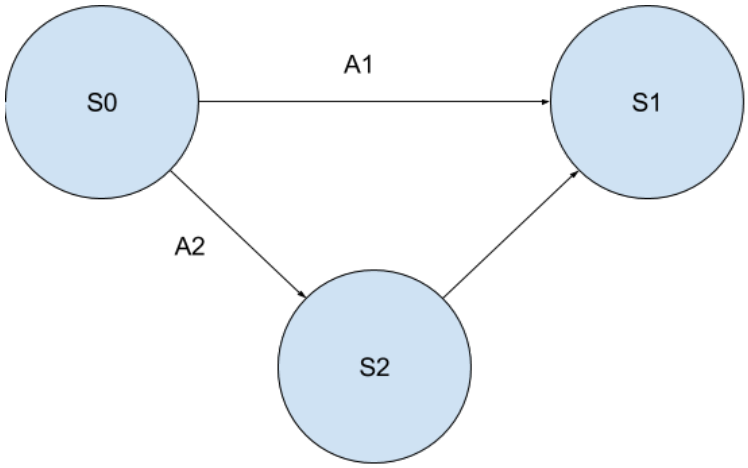


Figure 9: MDP for Question 5

6. (1 Point) Your boss asked you to estimate the state-value function associated with a known policy, π, for a specific MDP. You misheard and instead estimated the action-value function. This estimation was very expensive, and so you do not want to do it again. Explain how you could easily retrieve the value of any state given what you have already computed.

   Use the property: $v^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) q^\pi(s,a)$. That is, when asked for $v^\pi(s)$, compute the right side of this equation, which uses q.

7. (5) Consider a finite MDP with bounded rewards, where all rewards are negative. That is, $R_t < 0$ always. Let $\gamma = 1$. The MDP is finite horizon, with horizon L, and also has a deterministic transition function and initial state distribution (rewards may be stochastic). Let $H_\infty = (S_0, A_0, R_0, S_1, A_1, R_1, \ldots, S_{L-1}, A_{L-1}, R_{L-1})$ be any history that can be generated by a deterministic policy, π. Prove that the sequence $v^\pi(S_0), v^\pi(S_1), \ldots, v^\pi(S_{L-1})$ is strictly increasing.

$$v^\pi(S_t) \overset{(a)}{=} v^\pi(S_t) \tag{112}$$
$$\overset{(b)}{=} v^\pi(s_t) \tag{113}$$
$$\overset{(c)}{=} \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s_t, \pi\right] \tag{114}$$
$$\overset{(d)}{=} \sum_{k=0}^{\infty} \mathbf{E}\left[R_{t+k} | S_t = s_t, \pi\right] \tag{115}$$
$$\overset{(e)}{=} \sum_{k=0}^{\infty} \mathbf{E}\left[R_{t+k} | \pi\right] \tag{116}$$
$$= \mathbf{E}\left[R_t | \pi^\star\right] + \sum_{k=0}^{\infty} \mathbf{E}\left[R_{t+k+1} | \pi\right] \tag{117}$$
$$\overset{(f)}{=} \mathbf{E}\left[R_t | \pi\right] + \sum_{k=0}^{\infty} \mathbf{E}\left[R_{t+k+1} | S_{t+1} = s_{t+1}, \pi\right] \tag{118}$$
$$\overset{(g)}{=} \mathbf{E}\left[R_t | \pi\right] + v^\pi(s_{t+1}) \tag{119}$$
$$\overset{(h)}{=} \mathbf{E}\left[R_t | \pi\right] + v^\pi(S_{t+1}) \tag{120}$$
$$\overset{(i)}{\leq} v^\pi(S_{t+1}), \tag{121}$$

where **(a)** holds because $v^\star$ is the state-value function associated with all optimal policies, $\pi^\star$, as discussed in class, **(b)** holds because $S_t = s_t$ (for some $s_t \in \mathcal{S}$) deterministically due to the transition function and initial state distributions being deterministic, **(c)** comes from the definition of the state-value function, **(d)** holds because $\gamma = 1$, **(e)** and **(f)** hold because the sequence of states is deterministic and so conditioning on $S_t = s_t$ or $S_{t+1} = s_{t+1}$ is conditioning on an event that always happens, **(g)** holds by the definition of $v^{\pi^\star}$, **(h)** holds because $S_{t+1} = s_{t+1}$ always, and **(i)** holds because $R_t < 0$ (this was specified in the problem statement).

Part Two: Programming (45 Points Total)

Implement the 687-Gridworld domain described in class and in the class notes, and the cart-pole domain using the (frictionless) dynamics described by Florian (2007, Equations 23 and 24). When implementing Cart-Pole, the state should include the position of the cart, velocity of the cart, angle of the pole, and angular velocity of the pole. You **must** use a forward Euler approximation of the dynamics. If the cart hits the boundary of the track (which runs from -3 to 3), 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 from scratch.** Four questions ask for a plot. You may 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 and a curve corresponding to first-choice hill-climbing.

Use the following values for the Cart-Pole constants:

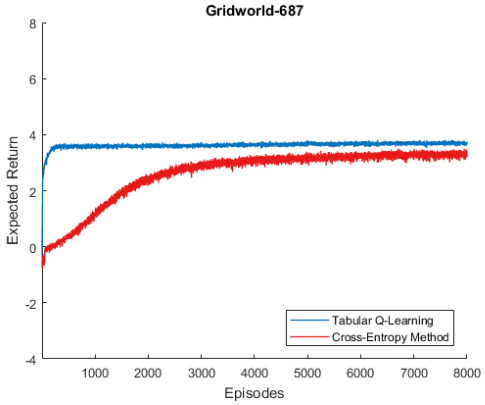
- Fail angle $= \pi/2$. (If it exceeds this value or its negative, the episode ends in failure.)
- Motor Force F $= 10.0$ (force on cart in Newtons).
- Gravitational constant g is 9.8.
- Cart mass $= 1.0$.
- Pole mass $= 0.1$.
- Pole half-length $l = 0.5$.
- $\Delta t = 0.02$ seconds (time step).
- Max time before end of episode $= 20$ seconds $+ 10\Delta t = 20.2$ seconds.
- The cart's position may be bounded in $[-3, 3]$, the cart's velocity in $[-10, 10]$, and the angular velocity in $[-\pi, \pi]$.

Problems:

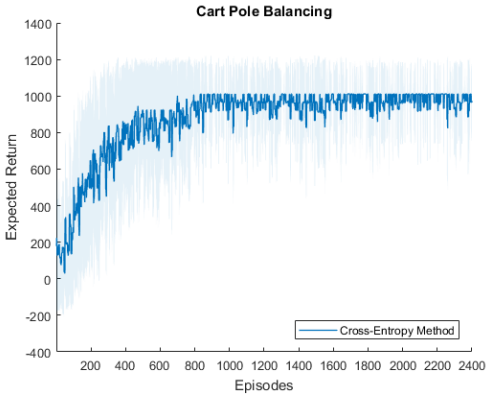
- (10 Points) Implement the cross-entropy method as described in the class notes and apply it to the 687-Gridworld. Use a tabular softmax policy. Search the space of hyperparameters for hyperparameters that work well. Report how you searched the hyperparameters, what hyperparameters you found worked best, and present a learning curve plot using these hyperparameters, as described in Figure 7. 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 error (or standard deviation) error bars (say which error bar variant you used).
- (10 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. Report the same quantities, as well as how you parameterized the policy.
- (10 Points) Repeat the previous question, but using first-choice hill-climbing on the 687-Gridworld domain. Report the same quantities.
- (10 Points) Repeat the previous question, but using first-choice hill-climbing (as described earlier in these notes) on the cart-pole domain. Report the same quantities and how the policy was parameterized.
- (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 687-Gridworld problem. Did it take more or fewer episodes than you expected? Why do you think this happened?

Be very generous with the grading; half-decent results should receive full credit. Not for grading, but useful for giving office hours hints: Gridworld CEM hyperparameters: pop 10, 4 elite, 1 ep per policy, sigma 1, init with 0's. This gives a smooth curve, (not stepwise).

Below is an example plot using the cross entropy method for 687-Gridworld, a population of 10, 4 elite, 5 episodes run to evaluate each policy, and the initial covariance matrix equal to $5I$, where I is the identity matrix.



Below is an exmaple plot using the cross entropy method for Cart-Pole, a population of 10, 5 elite, 5 episodes run to evaluate each policy, and the initial covariance matrix equal to $2I$.



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

$$v^\pi(s) = \mathbf{E}\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+1} + \ldots \middle| S_t = s, \pi\right] \tag{122}$$

$$= \mathbf{E}\left[R_t + \gamma\left(R_{t+1} + \gamma R_{t+1} + \ldots\right) | S_t = s, \pi\right] \tag{123}$$

$$= \mathbf{E}\left[R_t + \gamma v^\pi(S_{t+1}) | S_t = s, \pi\right] \tag{124}$$

$$= \sum_{a\in\mathcal{A}} \pi(s,a) \sum_{s'\in\mathcal{S}} P(s,a,s')\left(R(s,a) + \gamma v^\pi(s')\right). \tag{125}$$

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):

$$v^\pi(s_2) = R_2 + \gamma R_3 = 10 + \gamma 0 = 10. \tag{126}$$

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

$$v^\pi(s_2) = R_2 + \gamma v^\pi(s_3) = 10 + \gamma 0 = 10. \tag{127}$$

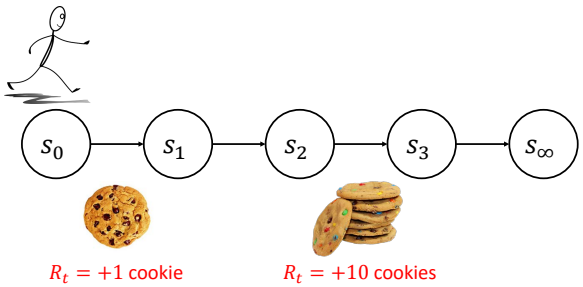


Figure 10: The "Cookie MDP". Under some policy, π, the agent always begins in s_0, and walks down the line of states. The agent receives a reward of $+1$ when transitioning from state s_0 to s_1 and a reward of $+10$ when transitioning from s_2 to s_3. All other transitions result in a reward of zero.

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

$$v^\pi(s_1) = R_1 + \gamma R_2 + \gamma^2 R_3 = 0 + \gamma 10 + \gamma^2 0 = \gamma 10. \tag{128}$$

$$v^\pi(s_1) = R_1 + \gamma v^\pi(s_2) = 0 + \gamma 10. \tag{129}$$

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

$$v^\pi(s_0) = R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 R_3 = 1 + \gamma 0 + \gamma^2 10 + \gamma^3 0 = 1 + \gamma^2 10. \tag{130}$$

$$v^\pi(s_0) = R_0 + \gamma v^\pi(s_1) = 1 + \gamma\gamma 10 = 1 + \gamma^2 10. \tag{131}$$

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,

$$v^\pi(s) = \mathbf{E}\left[\underbrace{R(s,A_t)}_{\text{immediate reward}} + \gamma \underbrace{v^\pi(S_{t+1})}_{\text{value of next state}} \middle| S_t = s, \pi\right]. \tag{132}$$

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:

$$q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s,a,s') \sum_{a' \in \mathcal{A}} \pi(s',a') q^\pi(s',a'). \tag{133}$$

Question 12. *Can you derive the Bellman equation for $q^\pi(s,a)$ from the definition of q^π?*

Answer 11.

4.5 Optimal Value Functions

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

$$v^*(s) \coloneqq \max_{\pi \in \Pi} v^\pi(s). \tag{142}$$

Notice that for each state v^* "uses" the policy, π, that maximizes $v^\pi(s)$. From this definition, we can immediately see taht 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. Hereafter we show that $v^* = v^{\pi^*}$ for all optimal policies π^*—notice that this does not follow immediately from the definition of v^*.

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

$$\pi \geq \pi' \text{ if and only if } \forall s \in \mathcal{S},\ v^\pi(s) \geq v^{\pi'}(s). \tag{143}$$

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 13. *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

$$\forall \pi \in \Pi,\ \pi^* \geq \pi. \tag{144}$$

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 (142), 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 (144) is common in theoretical reinforcement learning papers and in papers that emphasize the use of value functions. 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.

Question 14. *Which definition of an optimal policy is stricter, the definition in (17) or the definition in (144)?*

Answer 13. The definition in (144) is stricter. That is, every optimal policy according to (144) is an optimal policy according to (17), but every optimal policy according to (17) is not necessarily optimal according to (144). 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 (144) requires an optimal policy to act to maximize the expected return if were to ever be placed in these unreachable states.

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

$$q^*(s,a) \coloneqq \max_{\pi \in \Pi} q^\pi(s,a). \tag{145}$$

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

Question 15. *Given v^*, can you compute π^* if you do not know P and R?*

Answer 14. No. Any action in $\arg\max_{a \in \mathcal{A}} \sum_{s'} P(s,a,s')[R(s,a) + \gamma v^*(s')]$ (146) is an optimal action in state s. Computing these actions requires knowledge of P and R.

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

Answer 15. Yes. Any action in $\arg\max_{a \in \mathcal{A}} q^*(s,a)$ (147) 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 can derive it beginning from the Bellman equation for π^*:

$$v^*(s) = \sum_{a \in \mathcal{A}} \pi^*(s,a) \underbrace{\sum_{s' \in \mathcal{S}} P(s,a,s') \left[R(s,a) + \gamma v^*(s')\right]}_{q^*(s,a)}. \tag{148}$$

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 (148) to be maximized. Thus,

$$v^*(s) = \max_{a \in \mathcal{A}} \underbrace{\sum_{s' \in \mathcal{S}} P(s,a,s') \left[R(s,a) + \gamma v^*(s') \right]}_{q^*(s,a)}. \tag{149}$$

The above equation is the Bellman optimality equation for v^*. Similarly, the Bellman optimality equation for q^* is:

$$q^*(s,a) = \sum_{s' \in \mathcal{S}} P(s,a,s') \left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} q^*(s',a') \right]. \tag{150}$$

Question 17. *Derive the Bellman optimality equation for q^* starting with the Bellman equation for q^{π^*}.*

An obvious question is whether the Bellman optimality equations can hold for value functions that are not v^* or q^*. That is, if π is not an optimal policy, can it be true that for all s,

$$v^\pi(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')[R(s,a) + \gamma v^\pi(s')]? \tag{151}$$

Later we will show that this *cannot* happen—any policy whose value function satisfies the Bellman optimality equation in necessarily an optimal policy.

End of Lecture 6, September 25, 2018

Name:

CMPSCI 687 Pop Quiz 3
September 27, 2018

Instructions: You have 10 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:

$$v^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s') \big(R(s,a) + \gamma v^\pi(s')\big) \tag{152}$$

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

$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s') \big(R(s,a) + \gamma v^*(s')\big) \tag{153}$$

5. (True or False) Although there may be multiple optimal policies, there is only one optimal value function for an MDP.

   Any answer for question 5 accepted since we did not specify that the MDP has bounded rewards or is finite.

5 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.

5.1 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, $v^\pi(s_1), v^\pi(s_2), \ldots, v^\pi(s_{|\mathcal{S}|}$. These equations are:

$$v^\pi(s_1) = \sum_{a\in\mathcal{A}} \pi(s_1, a) \sum_{s'\in\mathcal{S}} P(s_1, a, s') \left(R(s_1, a) + \gamma v^\pi(s')\right) \tag{154}$$

$$v^\pi(s_2) = \sum_{a\in\mathcal{A}} \pi(s_2, a) \sum_{s'\in\mathcal{S}} P(s_2, a, s') \left(R(s_2, a) + \gamma v^\pi(s')\right) \tag{155}$$

$$\ldots$$

$$v^\pi(s_{|\mathcal{S}|}) = \sum_{a\in\mathcal{A}} \pi(s_{|\mathcal{S}|}, a) \sum_{s'\in\mathcal{S}} P(s_{|\mathcal{S}|}, a, s') \left(R(s_{|\mathcal{S}|}, a) + \gamma v^\pi(s')\right). \tag{156}$$

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 (67), 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 the current 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 $v_0, v_1, v_2, \ldots$ denote a sequence of functions where each $v_i : \mathcal{S} \to \mathbb{R}$. 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:

$$v^\pi(s) = \sum_{a\in\mathcal{A}} \pi(s, a) \sum_{s'\in\mathcal{S}} P(s, a, s') \left(R(s, a) + \gamma v^\pi(s')\right). \tag{157}$$

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

$$v_{i+1}(s) = \sum_{a\in\mathcal{A}} \pi(s, a) \sum_{s'\in\mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_i(s')\big). \tag{158}$$

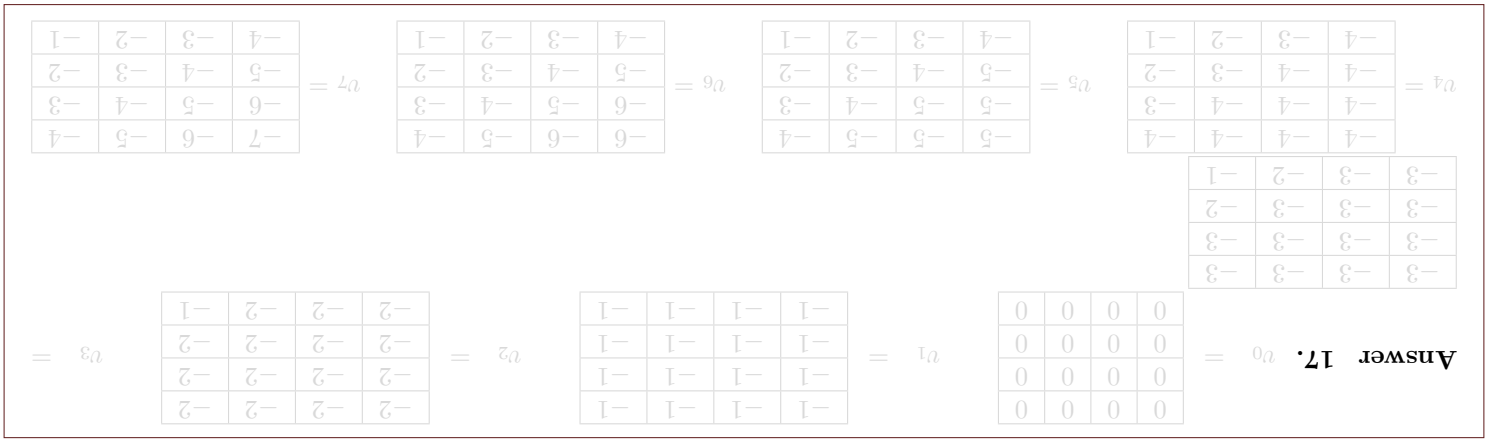
Applying this update to compute $v_{i+1}(s)$ for ever state s given v_i, is called a *full backup*. Applying the update to a compute $v_{i+1}(s)$ for a single state, s, is called a *backup*.

To see why this is a dynamic programming approach, consider what this algorithm does if we stored $v_0, v_1, \ldots$ 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 18. *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 $v_1, v_2, \ldots, v_7$.*

Answer 17.



Notice that in Question 18 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 18, 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.

Question 19. *What is the dynamic programming policy evaluation update for estimating the action-value function for a policy?*

Answer 18.

$$q_{i+1}(s,a) = \sum_{s' \in \mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma \sum_{a' \in \mathcal{A}} \pi(s',a') q_i(s',a')\big). \tag{159}$$

5.2 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

$$\pi'(s) \in \arg\max_{a \in \mathcal{A}} q^\pi(s,a). \tag{160}$$

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 (143)). This result is described by the *policy improvement theorem*, Theorem 1.

Theorem 1 (Policy Improvement Theorem)**.** *For any policy π, if π' is a deterministic policy such that $\forall s \in \mathcal{S}$,*

$$q^{\pi}(s, \pi'(s)) \geq v^{\pi}(s), \tag{161}$$

then $\pi' \geq \pi$.

Proof.

$$\begin{aligned}
v^{\pi}(s) \leq & q^{\pi}(s, \pi'(s)) && (162)\\
= & \mathbf{E}\left[R_t + \gamma v^{\pi}(S_{t+1}) | S_t = s, \pi'\right] && (163)\\
\leq & \mathbf{E}\left[R_t + \gamma q^{\pi}(S_{t+1}, \pi'(S_{t+1})) | S_t = s, \pi'\right] && (164)\\
= & \mathbf{E}\left[R_t + \gamma \mathbf{E}\left[R_{t+1} + \gamma v^{\pi}(S_{t+2}) | S_t = s, \pi'\right] | S_t = s, \pi'\right] && (165)\\
= & \mathbf{E}\left[R_t + \gamma R_{t+1} + \gamma^2 v^{\pi}(S_{t+2}) \middle| S_t = s, \pi'\right] && (166)\\
\leq & \mathbf{E}\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 v^{\pi}(S_{t+3}) \middle| S_t = s, \pi'\right] && (167)\\
& \dots && (168)\\
\leq & \mathbf{E}\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^4 R_{t+3} + \cdots \middle| S_t = s, \pi'\right] && (169)\\
= & v^{\pi'}(s), && (170)
\end{aligned}$$

where each step follows from the previous using mixtures of the definitions of value functions and the assumption within the theorem statement. Notice that (163) 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 (165) 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 (165) 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 2, for which we do not provide a proof.

Theorem 2 (Policy Improvement Theorem for Stochastic Policies)**.** *For any policy π, if π' satisfies*

$$\sum_{a \in \mathcal{A}} \pi'(s,a) q^{\pi}(s,a) \geq v^{\pi}(s), \tag{171}$$

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

End of Lecture 7, September 27, 2018

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 11, and presented using pseudocode in Algorithm 6.

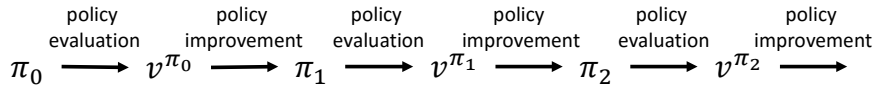


Figure 11: Diagram of the policy iteration algorithm. It begins with an arbitrary policy, π_0, and evaluates it using the dynamic programming approach described previously to produce v^{π_0}. It then performs greedy policy improvement to select a new deterministic policy, π_1, that is at least as good 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 6: 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 $s \in \mathcal{S}$:

$$v_{k+1}(s) = \sum_{s' \in \mathcal{S}} P(s, \pi_i(s), s') \left(R(s, \pi_i(s)) + \gamma v_k(s')\right) \tag{172}$$

    if $v_{k+1} = v_k$ **then**
      $v^{\pi_i} = v_k$;
      break;
  /* Policy Improvement */
  Compute π_{i+1} such that for all s,

$$\pi_{i+1}(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \Big(R(s, a) + \gamma v^{\pi_i}(s')\Big), \tag{173}$$

  with ties broken by selecting actions according to some strict total order on $\mathcal{A}$;
  /* Check for Termination */
  if $\pi_{i+1} = \pi_i$ **then**
    break;

Because the number of deterministic policies for a finite MDP is finite, policy iteration terminates after a finite number of iterations (if rewards are bounded and $\gamma < 1$). When this process terminates, we have that for all $s \in \mathcal{S}$,

$$v^{\pi_{i+1}}(s) \overset{(a)}{=} \sum_{a \in \mathcal{A}} \pi_{i+1}(s, a) \sum_{s' \in \mathcal{S}} P(s, a, s')(R(s, a) + \gamma v^{\pi_{i+1}}(s')) \tag{174}$$

$$\overset{(b)}{=} \sum_{a \in \mathcal{A}} \pi_{i+1}(s, a) \sum_{s' \in \mathcal{S}} P(s, a, s')(R(s, a) + \gamma v^{\pi_i}(s')) \tag{175}$$

$$\overset{(c)}{=} \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s')(R(s, a) + \gamma v^{\pi_i}(s')) \tag{176}$$

$$\overset{(b)}{=} \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s')(R(s, a) + \gamma v^{\pi_{i+1}}(s')), \tag{177}$$

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 (177) 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.

5.3 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 7.

Algorithm 7: 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 $s \in \mathcal{S}$:

$$v_{i+1}(s) = \sum_{s' \in \mathcal{S}} P(s, \pi_i(s), s') \left(R(s, \pi_i(s)) + \gamma v_i(s') \right) \tag{178}$$

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

$$\pi_{i+1}(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \Big(R(s, a) + \gamma v_{i+1}(s') \Big). \tag{179}$$

If we were to following the pseudocode in Algorithm 7, we would obtain the following sequence of policies and value functions:

$$v_0 : \text{arbitrary} \tag{180}$$

$$\pi_0 : \text{arbitrary} \tag{181}$$

$$v_1 : \forall s, v_1(s) = \sum_{s' \in \mathcal{S}} P(s, \pi_0(s), s') \big(R(s, \pi_0(s)) + \gamma v_0(s') \big) \tag{182}$$

$$\pi_1 : \forall s, \pi_1(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_1(s') \big) \tag{183}$$

$$v_2 : \forall s, v_2(s) = \sum_{s' \in \mathcal{S}} P(s, \pi_1(s), s') \big(R(s, \pi_1(s)) + \gamma v_1(s') \big) \tag{184}$$

$$\pi_2 : \forall s, \pi_2(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_2(s') \big) \tag{185}$$

$$v_3 : \forall s, v_3(s) = \sum_{s' \in \mathcal{S}} P(s, \pi_2(s), s') \big(R(s, \pi_2(s)) + \gamma v_2(s') \big) \tag{186}$$

$$\pi_3 : \forall s, \pi_3(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_3(s') \big) \tag{187}$$

$$\dots \tag{188}$$

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 (183). This expression is the same expression as that in the right side of (184). Thus, the expression for $v_2(s)$ can be written as:

$$v_2(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_1(s') \big). \tag{189}$$

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:

$$v_{i+1}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_i(s') \big). \tag{190}$$

Notice that, while the policy evaluation algorithm is an iterative form for the Bellman equation, the value iteration update in (190) is an iterative form for the Bellman optimality equation. Pseudocode for the value iteration algorithm using this more efficient update is presented in Algorithm 8.

Algorithm 8: Value Iteration.

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

$$v_{i+1}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_i(s') \big). \tag{191}$$

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

5.4 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

$$\mathcal{T}(v_i) \coloneqq v_{i+1}, \tag{192}$$

where the sequence of value function approximations, $v_0, v_1, \ldots$, is as defined by (190). That is,

$$\mathcal{T}(v_i) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s, a, s') \big(R(s, a) + \gamma v_i(s') \big). \tag{193}$$

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 12 presents a diagram that may assist in understanding the definition of a contraction mapping.

Question 20. *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 19. *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:

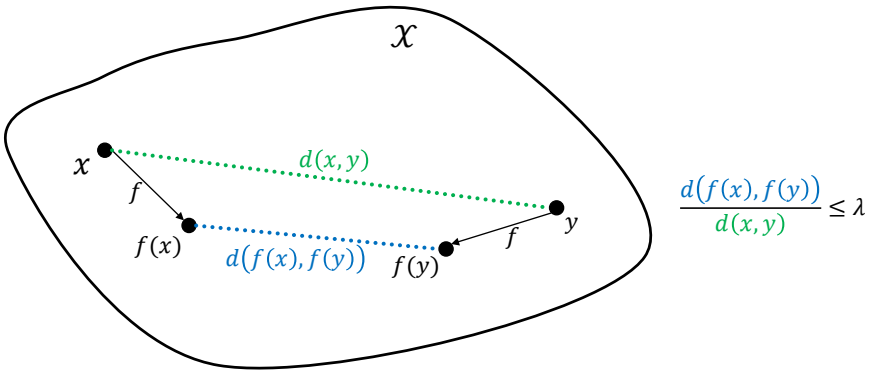


Figure 12: Diagram to assist with interpreting the definition of a contraction mapping. Here x and y denote two points in the space $\mathcal{X}$. The function, f, maps x to $f(x)$, and y to $f(y)$. If f is a contraction mapping, then for every possible x and y, the distance between x and y (the length 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.

Theorem 3 (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.[3] We must also show that the Bellman operator is a contraction mapping—we show this in Theorem 4.

Theorem 4 (The Bellman operator is a contraction mapping)**.** *The Bellman operator is a contraction mapping on $\mathbb{R}^{|\mathcal{S}|}$ with $d(v, v') \coloneqq \max_{s\in\mathcal{S}} |v(s) - v'(s)|$.*

End of Lecture 8, October 2, 2018

Proof.

$$\|Tv - Tv'\| = \max_{s\in\mathcal{S}} |Tv(s) - Tv'(s)| \tag{194}$$

$$= \max_{s\in\mathcal{S}} \left| \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v(s')\big) - \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v'(s')\big) \right|, \tag{195}$$

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:

$$\forall x, f(x) - g(x) \leq |f(x) - g(x)| \tag{196}$$

$$\forall x, f(x) \leq |f(x) - g(x)| + g(x) \tag{197}$$

$$\max_{x\in\mathcal{X}} f(x) \leq \max_{x\in\mathcal{X}} |f(x) - g(x)| + g(x) \tag{198}$$

$$\max_{x\in\mathcal{X}} f(x) \leq \max_{x\in\mathcal{X}} |f(x) - g(x)| + \max_{x\in\mathcal{X}} g(x) \tag{199}$$

$$\max_{x\in\mathcal{X}} f(x) - \max_{x\in\mathcal{X}} g(x) \leq \max_{x\in\mathcal{X}} |f(x) - g(x)|. \tag{200}$$

If $\max_{x\in\mathcal{X}} f(x) - \max_{x\in\mathcal{X}} g(x) \geq 0$, then it follows from (200) that

$$\left| \max_{x\in\mathcal{X}} f(x) - \max_{x\in\mathcal{X}} g(x) \right| \leq \max_{x\in\mathcal{X}} |f(x) - g(x)|. \tag{201}$$

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

If $\max_{x\in\mathcal{X}} f(x) - \max_{x\in\mathcal{X}} g(x) < 0$, then we have from (200) that:

$$\max_{x\in\mathcal{X}} g(x) - \max_{x\in\mathcal{X}} f(x) \leq \max_{x\in\mathcal{X}} |g(x) - f(x)|, \tag{202}$$

which also implies (201), 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 (201) to (195), we obtain:

$$\begin{aligned}
\|Tv - Tv'\| \leq & \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \left| \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v(s')\big) - \sum_{s'\in\mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v'(s')\big) \right| && (203)\\
= & \gamma \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \left| \sum_{s'\in\mathcal{S}} P(s,a,s')\big(v(s') - v'(s')\big) \right| && (204)\\
= & \gamma \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s') \left|\big(v(s') - v'(s')\big)\right| && (205)\\
\leq & \gamma \max_{s\in\mathcal{S}} \max_{a\in\mathcal{A}} \max_{s'\in\mathcal{S}} \left|\big(v(s') - v'(s')\big)\right| && (206)\\
= & \gamma \max_{s'\in\mathcal{S}} |v(s') - v'(s')| && (207)\\
= & \gamma \|v - v'\|. && (208)
\end{aligned}$$

□

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^∞. 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.

Because v^∞ is a fixed-point of the Bellman operator, we have that for all $s \in \mathcal{S}$:

$$v^\infty(s) = \max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^\infty(s)), \tag{209}$$

which is the Bellman optimality equation. Consider any deterministic policy, π^∞, that is greedy with respect to v^∞. We have that for all $s \in \mathcal{S}$:

$$\pi^\infty(s) \in \arg\max_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^\infty(s)). \tag{210}$$

We would like to conclude that π^∞ is an optimal policy, which would prove that an optimal deterministic policy always exists (for MDPs with finite states and actions, bounded rewards, and $\gamma \in [0,1)$). It is tempting to conclude this property because the fixed point of Value Iteration is the Bellman optimality equation, but we derived the Bellman optimality equation beginning with the assumption that π^* exists, so we cannot rely on the Bellman optimality equation to prove that an optimal policy exists.

It is further tempting to argue that for all policies, π, if we begin value iteration with v^π, it still converges to v^∞. Further, by the policy improvement theorem each iteration of value iteration results in a policy that is at least as good as the preceding policy, and thus the policy that is greedy with respect to v^∞ is at least as good as every other policy, and thus optimal. However, this is a misuse of the policy improvement theorem, which describes iterations of *policy* iteration, not *value* iteration (and we have not proven that v^∞ is the unique fixed point of policy iteration).

We *can* conclude that any policy, π, that is not greedy with respect to v^∞ is not an optimal policy. This is clear because value iteration beginning with v^π can proceed in two ways. First, $v^\pi = v^\infty$, in which case π is greedy with respect to v^∞, which is a contradiction. Second, $v^\pi \neq v^\infty$, and so during the iteration starting from v^π the value function changes, which means that the policy changes. The policy change is performed by greedy policy improvement to produce a new policy, π'. Here, by the policy improvement theorem we have that $\pi' \geq \pi$: that $\forall s, v^{\pi'}(s) \geq v^\pi(s)$. Further, since $v^\pi \neq v^{\pi'}$ (if it was, we are at a fixed point, and thus $v^\pi = v^\infty$), we have that there exists a state $s \in \mathcal{S}$ for which $v^{\pi'}(s) > v^\pi(s)$, and thus π cannot be an optimal policy. However, this does not establish that π^∞ is an optimal policy, nor that $v^\infty = v^*$ as defined in (142).

Question 21. *How can one prove that an optimal deterministic policy exists for all MDPs with finite states, finite actions, bounded rewards, and $\gamma \in [0,1)$?*

Midterm Extra Credit
Assigned: October 4, 2018
Due: October 11, 2018

Instructions: This extra credit assignment should be solved individually—do not collaborate with others. There is no partial credit on this: you either obtain full credit (+5% on your midterm grade) or no credit. You may submit your answers on Moodle.

1. (+5% on the midterm) Prove that for all MDPs with finite state set, finite action set, bounded rewards, and $\gamma \in [0, 1)$ there exists at least one optimal deterministic policy. You may use results that we have *proven* in the notes above (e.g., that the Bellman operator is a contraction, or the policy improvement theorem), but not results that we have asserted (e.g., that policy iteration converges to an optimal policy, or that the Bellman optimality equation only holds for the value function of optimal policies). Your proof must be formal and precise. You *may* assume that the space of value functions is a complete normed vector space in order to apply the Banach fixed point theorem.

Answer: The proof is beyond the scope of these notes. For a complete proof, see the work of Puterman (1994, Proposition 4.4.3, page 90).

6 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:

$$\pi \approx 4 \frac{\text{number of darts inside the unit square and the circle}}{\text{number of darts inside the unit square}}. \tag{211}$$

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

6.1 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 22. *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 9, October 4, 2018

Name:

CMPSCI 687 Pop Quiz 4
October 11, 2018

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:

$$v^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v^{\pi}(s')\big) \tag{212}$$

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

$$v^{*}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v^{*}(s')\big) \tag{213}$$

3. The value iteration update is:

$$v_{i+1} = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')\big(R(s,a) + \gamma v_i(s')\big) \tag{214}$$

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 $\pi_{i+1} \geq \pi_i$.

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 the reward be +1 for self-transitions, and 0 for transitioning to s_∞. The value of s_0 in the example is 2, 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. The expected return for intermediate visits (not the first, and not the last) is between zero and two.

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 9.

Algorithm 9: 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, $\text{Var}(v_k(s)) \propto \frac{1}{k}$ since (using the fact that G_i are i.i.d.):

$$\text{Var}(v_k(s)) = \text{Var}\left(\frac{1}{k}\sum_{i=1}^k G_i\right) \tag{215}$$

$$= \frac{1}{k^2}\text{Var}\left(\sum_{i=1}^k G_i\right) \tag{216}$$

$$= \frac{1}{k^2} k\,\text{Var}\left(G_i\right) \tag{217}$$

$$= \frac{1}{k}\text{Var}\left(G_i\right). \tag{218}$$

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 10. 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 10: 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.[4] One way to fix this problem is 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.

End of Lecture 10, October 11, 2018

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 6. 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:

$$\pi_{i+1}(s) \in \arg\max_{a \in \mathcal{A}} q^{\pi_i}(s,a). \tag{219}$$

Unlike line 173 of Algorithm 6, 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 the current policy.

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

Pseudocode for this algorithm is presented in Algorithm 11.

Algorithm 11: Monte Carlo - Exploring Starts.

```
1  for all s ∈ S and a ∈ A do
2      q(s, a) ← arbitrary;
3      π(s) ← arbitrary;
4      Returns(s, a) ← 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 ← return following the first occurrence of (s, a);
9          Append G to Returns(s, a);
10         q(s, a) ← average(Returns(s, a));
11     for each s in the episode do
12         π(s) ← arg max_{a∈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, $a^* \leftarrow \arg\max_{a\in\mathcal{A}} q(s,a)$, and then defining the ϵ-greedy policy (a stochastic policy):

$$\pi(s,a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|} & \text{if } a = a* \\ \frac{\epsilon}{|\mathcal{A}|} & \text{otherwise.} \end{cases} \tag{220}$$

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)

6.2 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:

$$v(S_t) \leftarrow v(S_t) + \alpha(G_t - V(S_t)). \tag{221}$$

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

$$f(x) \leftarrow f(x) + \alpha(g(x) - f(x)). \tag{222}$$

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):

$$\text{MSVE}(v) \coloneqq \mathbf{E}\left[\frac{1}{2}(v^\pi(S) - v(S))^2\right], \tag{223}$$

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:

$$v \leftarrow v - \alpha\frac{\partial\,\text{MSVE}(v)}{\partial v} \tag{224}$$

$$= v - \alpha\frac{\partial}{\partial v}\mathbf{E}\left[\frac{1}{2}(v^\pi(S) - v(S))^2\right] \tag{225}$$

$$= v + \alpha\mathbf{E}\left[(v^\pi(S) - v(S))\frac{\partial v(S)}{\partial v}\right]. \tag{226}$$

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:

$$v \leftarrow v + \alpha(G_t - V(S_t))\frac{\partial v(S_t)}{\partial v}. \tag{227}$$

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 (221).

Thus, because (221) 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 (221) as the stochastic gradient descent update for MSVE, we can obtain the equivalent update using function approximation:

$$w \leftarrow w + \alpha(G_t - v_w(S_t))\frac{\partial v_w(S_t)}{\partial w}. \tag{228}$$

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 (221) is a special case of (228), where v_w stores one number per state.

———————————End of Lecture 11, October 16, 2018———————————

Name: _____

CMPSCI 687 Midterm

October 18, 2018

Instructions: This midterm is **closed** notes. Do not use any notes or electronic devices. You have until 9pm to complete this exam.

1. Recall the gridworld described in class:

Start $s = 1$	$s = 2$	$s = 3$	$s = 4$	$s = 5$
$s = 6$	$s = 7$	$s = 8$	$s = 9$	$s = 10$
$s = 11$	$s = 12$		$s = 13$	$s = 14$
$s = 15$	$s = 16$		$s = 17$	$s = 18$
$s = 19$	$s = 20$	$R = -10$ $s = 21$	$s = 22$	$R = +10$ $s = 23$ Goal

Actions:
attempt_up
attempt_down
attempt_left
attempt_right

When the agent attempts to move in a direction:
The agent succeeds, $p = 0.8$
The agent veers 90° right, $p = 0.05$
The agent veers 90° left, $p = 0.05$
The agent stays in place, $p = 0.1$

If the agent would ever hit a wall, it stays in its current position.

All unspecified rewards are zero.
All specified rewards are for entering the state

Note: If the agent is in a state, s, takes an action, a, and transitions back to state s, we considering that "entering" the state s. Also, for this problem, use $\gamma = 1$.
You may write your answers on the next page.

(A) [4%] What is $P(18, attempt\_down, 18)$?

0.15

(B) [8%] Consider an agent that begins with an initial state-value function approximation, $v(s) = 0$ for all $s \in \mathcal{S}$. During the first episode, the agent visits the following states in the following order: 1, 2, 3, 8, 7, 12, 16, 20, 21, 22, 21, 22, 17, 18, 23. If the agent uses every-visit Monte Carlo to estimate the state-value function, what will the resulting value function be after this first episode? Draw the value function as a 5×5 matrix.

−10	−10	−10	0	0
0	−10	−10	0	0
0	−10	N/A	0	0
0	−10	N/A	10	10
0	−10	5	5	0

(C) [8%] Repeat part (B), but if the agent used First-Visit Monte Carlo policy evaluation.

−10	−10	−10	0	0
0	−10	−10	0	0
0	−10	N/A	0	0
0	−10	N/A	10	10
0	−10	0	0	0

2. Define the following (using math, and the notation in class):

- [2%] $q^\pi(s,a) = \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, A_t = a, \pi\right]$
- [2%] $J(\pi) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t | \pi\right]$
- [2%] $Tv(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a,s') + \gamma v(s'))$
- [2%] $P(s,a,s') = \Pr(S_{t+1} = s' | S_t = s, A_t = a)$
- [2%] $d_0(s) = \Pr(S_0 = s)$

3. [5%] Let v^π be the value function for the policy π. Write an expression for the greedy policy with respect to v^π: A greedy deterministic policy, π_{greedy} is given by:

$$\pi_{\text{greedy}}(s) \in \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^\pi(s')). \tag{229}$$

4. [5%] Write the Bellman optimality equation (using state-values, not action-values):

$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^*(s')). \tag{230}$$

5. [5%] Write the Bellman equation (using state-values, not action-values):

$$v^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^\pi(s')). \tag{231}$$

6. [2% each] For each of the following, circle either true or false:

   - (True or False) Every finite MDP with bounded rewards and $\gamma \in [0, 1)$ has at least one optimal value function.
   - (True or False) For all finite MDPs with bounded rewards and $\gamma \in [0, 1)$, for all policies π,

$$\forall s \in \mathcal{S},\ \max_{a \in \mathcal{A}} q^\pi(s, a) \geq v^\pi(s). \tag{232}$$

   - (True or False) Let $v_1, v_2, \ldots$ be the value function approximations created by running the value iteration algorithm. Let π_i be the greedy policy with respect to v_i. It is always true that $\pi_{i+1} \geq \pi_i$.
   - (True or False) Because the Bellman operator is a contraction mapping (for finite MDPs with bounded rewards and $\gamma \in [0, 1)$), if the Bellman operator is applied to a value function $v : \mathcal{S} \to \mathbb{R}$ that is not v^*, then $Tv(s)$ will be closer to $v^*(s)$ for all $s \in \mathcal{S}$. That is, for all $s \in \mathcal{S}$, $|Tv(s) - v^*(s)| < |v(s) - v^*(s)|$, where $|\cdot|$ denotes absolute value.
   - (True or False) The optimal policy (defined as any policy π^* such that for all other policies, π, $\pi^* \geq \pi$) does not depend on the initial state distribution.

7. [5%] Write an expression (as in Homework 1—in terms of π and symbols defined in an MDP) for the probability that the state at time 0 is s given that the state at time 1 is s' and the action at time 0 is a. Recall Bayes' Theorem:

$$\Pr(A = a | B = b) = \frac{\Pr(B = b | A = a) \Pr(A = a)}{\Pr(B = b)}, \tag{233}$$

   and the definition of conditional probabilities:

$$\Pr(A = a | B = b) = \frac{\Pr(B = b, A = a)}{\Pr(B = b)}. \tag{234}$$

$$\Pr(S_0 = s | A_0 = a, S_1 = s') \overset{(a)}{=} \frac{\Pr(S_0 = s, A_0 = a, S_1 = s')}{\Pr(S_1 = s', A_0 = a)} \tag{235}$$

$$= \frac{d_0(s)\pi(s, a)P(s, a, s')}{\sum_{s_0} \Pr(S_0 = s_0) \Pr(S_1 = s', A_0 = a | S_0 = s_0)} \tag{236}$$

$$= \frac{d_0(s)\pi(s, a)P(s, a, s')}{\sum_{s_0} d_0(s_0)\pi(s_0, a)P(s_0, a, s')}, \tag{237}$$

   where **(a)** comes from the definition of conditional probability.

8. [10%] Prove that last-visit Monte Carlo (for estimating a state-value function, v^π) is not guaranteed to converge almost surely to v^π for all finite MDPs with bounded rewards and $\gamma \in [0, 1)$. You may reference Khintchine's Strong Law of Large Numbers:

[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., $\frac{1}{n}\sum_{i=1}^n X_i \xrightarrow{\text{a.s.}} \mathbf{E}[X_1]$.

Consider applying last visit Monte Carlo to the MDP with $\mathcal{S} = \{1, s_\infty\}$, $\mathcal{A} = \{1\}$, $P(1, 1, 1) = 0.5$, $P(1, 1, s_\infty) = 0.5$, $R_t = 1$ if $S_{t+1} = 1$ and $R_t = 0$ if $S_{t+1} = s_\infty$, $d_0(1) = 1$, and $\gamma = 0.5$. Let π be the only policy, which deterministically selects action 1. Notice that $v^\pi(1) = 1$.

Last visit Monte Carlo computes returns from state 1 using only the last visit, so all returns from state 1 will be for transitions to s_∞, for which the reward (and return) are zero. Hence, the last visit Monte Carlo estimate for $v^\pi(1)$ after n episodes will be:

$$\frac{1}{n}\sum_{i=1}^n 0, \tag{238}$$

and so

$$\lim_{n\to\infty} \hat{v}^\pi(1) = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^n 0 = 0. \tag{239}$$

Hence,

$$\Pr\left(\lim_{n\to\infty} \hat{v}^\pi(1) = v^\pi(1)\right) = \Pr\left(\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^n 0 = 1\right) \tag{240}$$

$$= 0. \tag{241}$$

9. [15%] Someone proposes reversing the Bellman equation, and writing the value of a state in terms of the values of previous states. They come up with the expression:

$$v(s') = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} P(s, a, s') \pi(s, a) \left[\frac{v^{\pi}(s) - R(s, a)}{\gamma} \right]. \tag{242}$$

Is this expression correct when $\gamma \in (0, 1)$? Prove your answer (prove that this expression is or is not correct).

No. Consider an MDP with a state, s', that cannot be reached from any other state, but such that $v^{\pi}(s') \neq 0$. This new equation would say that $v^{\pi}(s') = 0$ because every $P(s, a, s')$ term would be zero, and so this equation cannot be correct.

10. [15%] Consider a finite MDP, $M = (\mathcal{S}, \mathcal{A}, P, d_R, d_0, \gamma)$, with bounded rewards and $\gamma \in [0, 1)$. Let π^* be a deterministic optimal policy for this MDP. Let $M' = (\mathcal{S}', \mathcal{A}', P', d'_R, d'_0, \gamma')$ be a new MDP that is the same as M, except that a positive constant, c, is subtracted from R_t if A_t is not the action that π^* would select. Is π^* necessarily always an optimal policy for M'? Prove your answer (if it is not, prove that it is not, and if it is, prove that it is).

Note: this last problem was intentionally made to be particularly difficult.

First, notice that for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, $R(s,a) \geq R'(s,a)$, where R' is the reward function of M', because $R(s,a) = R'(s,a)$ if π^* would choose action a in state s, and $R(s,a) > R'(s,a)$ otherwise. We write v_M^π to denote the value function for policy π on MDP M. Notice that for all $s \in \mathcal{S}$:

$$v_M^{\pi^*}(s) = \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, \pi^*, M\right] \tag{243}$$

$$\overset{(a)}{=} \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, \pi^*, M'\right] \tag{244}$$

$$= v_{M'}^{\pi^*}(s), \tag{245}$$

where **(a)** holds because $A_t \sim \pi^*$ means that R_t will be unchanged. Notice also that for all π and all $s \in \mathcal{S}$,

$$v_M^{\pi}(s) = \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s, \pi, M\right] \tag{246}$$

$$= \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R(S_{t+k}, A_{t+k}) \middle| S_t = s, \pi, M\right] \tag{247}$$

$$\overset{(a)}{\geq} \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R'(S_{t+k}, A_{t+k}) \middle| S_t = s, \pi, M'\right] \tag{248}$$

$$= v_{M'}^{\pi}(s), \tag{249}$$

where **(a)** holds because $R(s,a) \geq R'(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. Lastly, notice that, because π^* is optimal for M, we have that for all π and all $s \in \mathcal{S}$,

$$v_M^{\pi^*}(s) \geq v_M^{\pi}(s). \tag{250}$$

Combining (245), (249), and (250), we have that for all π and $s \in \mathcal{S}$,

$$v_{M'}^{\pi^*}(s) = v_M^{\pi^*}(s) \geq v_M^{\pi}(s) \geq V_{M'}^{\pi}(s), \tag{251}$$

and so $\pi^* \geq \pi$ for all policies π, and therefore π^* is an optimal policy for M'.

End of Midterm

7 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:

$$v(s) \leftarrow v(s) + \alpha(r + \gamma v(s') - v(s)). \tag{252}$$

Using other notation it can be defined equivalently as:

$$v(S_t) \leftarrow v(S_t) + \alpha(R_t + \gamma v(S_{t+1}) - v(S_t)). \tag{253}$$

This is very much like the Gradient-Based Monte Carlo Algorithm from Section 6.2, 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:

$$\delta_t = R_t + \gamma v(S_{t+1}) - v(S_t), \tag{254}$$

and allows us to write the TD update as:

$$v(S_t) \leftarrow v(S_t) + \alpha\delta. \tag{255}$$

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 23. *What is* $\mathbf{E}[\delta_t|S_t = s]$ *if* δ_t *uses the true state-value function,* v^π*?*

Answer 22.

$$\mathbf{E}[\delta_t|S_t = s] = \mathbf{E}[R_t + \gamma v^\pi(S_{t+1}) - v^\pi(S_t)|S_t = s] \tag{256}$$
$$= \mathbf{E}[R_t + \gamma v^\pi(S_{t+1})|S_t = s] - v^\pi(s) \tag{257}$$
$$\overset{(a)}{=} v^\pi(s) - v^\pi(s) \tag{258}$$
$$= 0. \tag{259}$$

where (a) comes from the Bellman equation.

Question 24. *What is* $\mathbf{E}[\delta_t|S_t = s, A_t = a]$ *if* δ_t *uses the true state-value function,* v^π*?*

Answer 23.

$$\mathbf{E}[\delta_t|S_t = s, A_t = a] = \mathbf{E}[R_t + \gamma v^\pi(S_{t+1}) - v^\pi(S_t)|S_t = s, A_t = a] \tag{260}$$
$$= \mathbf{E}[R_t + \gamma v^\pi(S_{t+1})|S_t = s, A_t = a] - v^\pi(s) \tag{261}$$
$$\overset{(a)}{=} q^\pi(s,a) - v^\pi(s) \tag{262}$$
$$= A(s,a), \tag{263}$$

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:

$$w \leftarrow w + \alpha(R_t + \gamma v_w(S_{t+1}) - v_w(S_t)) \frac{\partial v_w(S_t)}{\partial w}. \tag{264}$$

Question 25. *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 (253). Prove this.*

One might think that the TD algorithm is a gradient algorithm, much like the Gradient-Based Monte Carlo Algorithm from Section 6.2, 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:

$$L(w) = \mathbf{E}\left[\frac{1}{2}(R_t + \gamma v_w(S_{t+1}) - v_w(S_t))^2\right] \tag{265}$$

$$= \mathbf{E}[\frac{1}{2}\delta_t^2]. \tag{266}$$

We then compute the gradient:

$$\frac{\partial}{\partial w} L(w) = \frac{\partial}{\partial w} \mathbf{E}\left[\frac{1}{2}(R_t + \gamma v_w(S_{t+1}) - v_w(S_t))^2\right] \tag{267}$$

$$= \mathbf{E}\left[\delta_t \left(\gamma \frac{\partial v_w(S_{t+1})}{\partial w} - \frac{\partial v_w(S_t)}{\partial w}\right)\right] \tag{268}$$

$$= \mathbf{E}\left[-\delta_t \left(\frac{\partial v_w(S_t)}{\partial w} - \gamma \frac{\partial v_w(S_{t+1})}{\partial w}\right)\right], \tag{269}$$

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):

$$w \leftarrow w + \alpha\delta_t \left(\frac{\partial v_w(S_t)}{\partial w} - \gamma \frac{\partial v_w(S_{t+1})}{\partial w}\right). \tag{270}$$

Notice that **1)** due to subtle reasons, (270) is *not* an unbiased estimator of (269) (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 (270) using tabular function approximation can be written as:

$$v(S_t) \leftarrow v(S_t) + \alpha\delta_t \tag{271}$$

$$v(S_{t+1}) \leftarrow v(S_{t+1}) - \alpha\gamma\delta_t. \tag{272}$$

This alternate algorithm is *not residual gradient* (Baird, 1995), but is similar.[5]

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

[5]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).

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,

$$\mathbf{E}\left[\delta_t \frac{\partial v_w(S_t)}{\partial w}\right], \tag{273}$$

would be the gradient of a function. We can show that this is not the case: (273) 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 (273) 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 (273). This term is:

$$\frac{\partial}{\partial w_i}\delta_t \frac{\partial v_w(S_t)}{\partial w_j} = \delta_t \frac{\partial^2 v_w(S_t)}{\partial w_i \partial w_j} + \frac{\partial v_w(S_t)}{\partial w_j}\frac{\partial}{\partial w_i}\left(R_t + \gamma v_w(S_{t+1}) - v(S_t)\right) \tag{274}$$

$$= \underbrace{\delta_t \frac{\partial^2 v_w(S_t)}{\partial w_i \partial w_j}}_{(a)} + \underbrace{\frac{\partial v_w(S_t)}{\partial w_j}\left(\gamma \frac{\partial v_w(S_{t+1})}{\partial w_i} - \frac{\partial v(S_t)}{\partial w_i}\right)}_{(b)}. \tag{275}$$

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^*:

$$w^* \in \arg\min_w \mathbf{E}[(v_w(S_t) - v^\pi(S_t))^2]. \tag{276}$$

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):

$$\mathbf{E}[(v_{w_\infty}(S_t) - v^\pi(S_t))^2] \leq \frac{1}{1-\gamma}\mathbf{E}[(v_{w^*}(S_t) - v^\pi(S_t))^2]. \tag{277}$$

End of Lecture 12, October 23, 2018

CMPSCI 687 Pop Quiz 5
October 25, 2018

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.

1. What is the definition of the TD error (in math)?

$$\delta_t = R_t + \gamma v(S_{t+1}) - v(S_t), \tag{278}$$

or

$$\delta = r + \gamma v(s') - v(s). \tag{279}$$

2. If your value function estimate is correct and you are not expecting to get a cake, but you get a cake, what sign would your TD error have? (Assume that you like cake)

   It will be positive.

3. (True or False) TD is stocastic gradient descent on the mean squared value error (MSVE).

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:

$$\text{MSE}(X) \coloneqq \mathbf{E}[(X - \theta)^2]. \tag{280}$$

The MSE can be decomposed into two components: the squared bias and the variance:

$$\text{MSE}(X) = \text{Bias}(X)^2 + \text{Var}(X), \tag{281}$$

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.

8 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:

$$v_w(s) = w^\intercal \phi(s), \tag{282}$$

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:

$$\phi(s) = \begin{bmatrix} 1 \\ s \\ s^2 \\ \vdots \\ s^k \end{bmatrix}. \tag{283}$$

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.

8.1 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 (277) 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

$$\phi(s) = [\min\{5, \max\{1, x + \eta\}\}, y]^\intercal. \tag{284}$$

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 (277)) when using linear function approximation, and even if the state observations are noisy and/or incomplete.

———————————End of Lecture 13, October 25, 2018———————————

CMPSCI 687 Homework 3

Due November 6, 2018, 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 on Moodle as a .zip (.gz, .tar.gz, etc.) file containing your answers in a .pdf file and a folder with your source code. Include with your source code instructions for how to run your code. You may not use any reinforcement learning or machine learning specific libraries in your code (you may use libraries like C++ Eigen and numpy though). If you are unsure whether you can use a library, ask on Piazza. If you submit by November 13, you will not lose any credit. The automated system will not accept assignments after 11:55pm on November 13.

Programming: (60 Points)

This assignment only has a programming component. As in Homework 2, **you may not use existing RL code for this problem—you must implement the agent and environment entirely on your own and from scratch.** The exception to this is that you may re-use the code that you wrote for the previous homework assignments.

Implement TD for the 687-Gridworld and Cart-Pole domains using fixed step-sizes. Use TD to estimate the value of the uniform random policy—the policy that always selects actions from the discrete uniform distribution. For Cart-Pole, use the third- and fifth-order Fourier basis (Konidaris et al., 2011b) (apply the Fourier basis as though time were not part of the state). For both domains, initialize the value functions to be zero (all entries in the tabular representation for the gridworld are zero, and all weights for the linear representation used for Cart-Pole are zero). Create plots where the horizontal axis is the step size (using a logarithmic scale, and any reasonable range that includes 0.1, 0.01, and 0.001), and the vertical axis is the mean squared TD-error of the value function output by TD after 100 episodes. To estimate this mean squared TD-error, stop updating the weights after 100 episodes, and run an additional 100 episodes. During these extra 100 episodes, compute the TD error at each time step, square it to obtain the squared TD error, and report the average value of these TD errors. Submit one plot with three curves: one for 687-Gridworld and two for Cart-Pole (one using the third-order Fourier basis, and the other using the fifth-order Fourier basis).

After presenting this plot, write a reflection on this process. What details did you not notice until implementing the methods? Is the range of step sizes that works well bigger or smaller than you expected? Answer questions of this sort. Your reflection should be at least half a page, and no more than two pages.

CMPSCI 687 Pop Quiz 6
October 30, 2018

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.

1. When using linear function approximation, what is $\frac{\partial v_w(s)}{\partial w}$ equal to?

$$\frac{\partial v_w(s)}{\partial w} = \phi(s). \tag{285}$$

2. The TD error is defined as:

$$\delta_t = R_t + \gamma v^\pi(S_{t+1}) - v^\pi(S_t) \tag{286}$$

3. Write the TD update (tabular, linear, or using arbitrary function approximation—your choice):

$$v(S_t) = v(S_t) + \alpha(R_t + \gamma v(S_{t+1}) - v(S_t)) \tag{287}$$

8.2 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:

$$\hat{P}(s, a, s') = \frac{\#(s, a, s')}{\#(s, a)} \tag{288}$$

$$\hat{R}(s, a) = \text{mean}(r|s, a), \tag{289}$$

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.

9 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:

$$q(s, a) \leftarrow q(s, a) + \alpha(r + \gamma q(s', a') - q(s, a)), \tag{290}$$

and the TD update for q_w using arbitrary function approximation is:

$$w \leftarrow w + \alpha(r + \gamma q_w(s', a') - q_w(s, a))\frac{\partial q_w(s, a)}{\partial w}. \tag{291}$$

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:

$$\delta_t = R_t + \gamma q_w(S_{t+1}, A_{t+1}) - q_w(S_t, A_t) \tag{292}$$

$$w_{t+1} \leftarrow w_t + \alpha\delta_t\frac{\partial q_w(S_t, A_t)}{\partial w}. \tag{293}$$

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:

$$v(x) = \mathbf{E}[G_t|X_t = x] \tag{294}$$

$$[\text{for } M] = \mathbf{E}[G_t|S_t = s, A_t = a] \tag{295}$$

$$[\text{for } M] = q^\pi(s, a). \tag{296}$$

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 (290). 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 12, and Algorithm 13 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 12: 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 13: 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;
    $w \leftarrow w + \alpha(r + \gamma q_w(s', a') - q_w(s, a)) \frac{\partial q_w(s,a)}{\partial w}$;
    $s \leftarrow s'$;
    $a \leftarrow a'$;

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 26. *What happens if actions are chosen greedily with respect to q rather than nearly greedily?*

Answer 25. *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 27. *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 26. *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

10 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):

$$q(s,a) = q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} q(s', a') - q(s,a)), \tag{297}$$

or with arbitrary function approximation:

$$w = w + \alpha(r + \gamma \max_{a' \in \mathcal{A}} q_w(s', a') - q_w(s,a)) \frac{\partial q_w(s,a)}{\partial w}. \tag{298}$$

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 14 and 15

Algorithm 14: Tabular Q-Learning

```
Initialize q(s, a) arbitrarily;
for each episode do
    s ~ 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) ← q(s, a) + α(r + γ max_{a'∈A} q(s', a') − q(s, a));
        s ← s';
```

Algorithm 15: Q-Learning

```
Initialize w arbitrarily;
for each episode do
    s ~ 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 ← w + α(r + γ max_{a'∈A} q_w(s', a') − q_w(s, a)) ∂q_w(s,a)/∂w;
        s ← s';
```

The convergence properties of TD, Sasra, and Q-Learning are presented in Table 1.

11 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

	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.

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:

$$G_t^{(n)} = \left(\sum_{k=0}^{n-1} \gamma^k R_{t+k}\right) + \gamma^n v(S_{t+n}). \tag{299}$$

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:

$$G_t^\lambda := (1-\lambda)\sum_{i=0}^{\infty} \lambda^i G_t^{(i+1)}. \tag{300}$$

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.

End of Lecture 15, November 1, 2018

Name:

CMPSCI 687 Pop Quiz 7
November 6, 2018

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 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:

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

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

$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s,a,s') \big(R(s,a) + \gamma v^*(s')\big) \tag{302}$$

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

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 $(1-\lambda)\lambda^2, \ldots$, 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:

$$G_t^\lambda := (1-\lambda)\sum_{i=0}^{\infty}\lambda^i G_t^{(i+1)} \tag{303}$$

$$= (1-\lambda)\left(\sum_{i=0}^{L-2}\lambda^i G_t^{(i+1)}\right) + (1-\lambda)\sum_{i=L-1}^{\infty}\lambda^i G_t^{\text{MC}} \tag{304}$$

$$= (1-\lambda)\left(\sum_{i=0}^{L-2}\lambda^i G_t^{(i+1)}\right) + (1-\lambda)\left(\sum_{i=L-1}^{\infty}\lambda^i\right) G_t^{\text{MC}}. \tag{305}$$

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 5. *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: $\text{Var}(G_t^{(i)}) = \beta/\lambda^i$, 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 $G_t^{(1)}, G_t^{(2)}, G_t^{(3)}, \ldots$ is

$$\mathcal{L}(x|G_t^{(1)}, G_t^{(2)}, G_t^{(3)}, \ldots) = \Pr\left(G_t^{(1)}, G_t^{(2)}, G_t^{(3)}, \ldots \middle| v^\pi(S_t) = x\right) \tag{306}$$

$$= \prod_{i=1}^{\infty}\Pr(G_t^{(i)}|v^\pi(S_t) = x), \tag{307}$$

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:

$$\arg\max_{x\in\mathbb{R}} \mathcal{L}(x|G_t^{(1)}, G_t^{(2)}, G_t^{(3)}, \dots) = \arg\max_{x\in\mathbb{R}} \prod_{i=1}^{\infty} \Pr(G_t^{(i)}|v^\pi(S_t) = x) \tag{308}$$

$$= \arg\max_{x\in\mathbb{R}} \ln\left(\prod_{i=1}^{\infty} \Pr(G_t^{(i)}\Big|v^\pi(S_t) = x)\right) \tag{309}$$

$$= \arg\max_{x\in\mathbb{R}} \sum_{i=1}^{\infty} \ln\left(\Pr(G_t^{(i)}\Big|v^\pi(S_t) = x)\right) \tag{310}$$

$$\overset{(a)}{=} \arg\max_{x\in\mathbb{R}} \sum_{i=1}^{\infty} \ln\left(\frac{1}{\sqrt{2\pi\beta/\lambda^i}} \exp\frac{-(G_t^{(i)} - x)^2}{2\beta/\lambda^i}\right) \tag{311}$$

$$= \arg\max_{x\in\mathbb{R}} \sum_{i=1}^{\infty} \ln\left(\frac{1}{\sqrt{2\pi\beta/\lambda^i}}\right) + \ln\left(\exp\frac{-(G_t^{(i)} - x)^2}{2\beta/\lambda^i}\right) \tag{312}$$

$$\overset{(b)}{=} \arg\max_{x\in\mathbb{R}} \sum_{i=1}^{\infty} \ln\left(\exp\frac{-(G_t^{(i)} - x)^2}{2\beta/\lambda^i}\right) \tag{313}$$

$$= \arg\max_{x\in\mathbb{R}} \sum_{i=1}^{\infty} \frac{-(G_t^{(i)} - x)^2}{2\beta/\lambda^i}, \tag{314}$$

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:

$$\frac{\partial}{\partial x} \sum_{i=1}^{\infty} \frac{-(G_t^{(i)} - x)^2}{2\beta/\lambda^i} = 0 \tag{315}$$

$$\iff \sum_{i=1}^{\infty} \frac{\partial}{\partial x} \frac{-\lambda^i(G_t^{(i)} - x)^2}{2\beta} = 0 \tag{316}$$

$$\iff \sum_{i=1}^{\infty} \frac{\lambda^i(G_t^{(i)} - x)}{\beta} = 0 \tag{317}$$

$$\iff \sum_{i=1}^{\infty} \lambda^i G_t^{(i)} = \sum_{i=1}^{\infty} \lambda^i x \tag{318}$$

$$\iff \sum_{i=1}^{\infty} \lambda^i G_t^{(i)} = \frac{\lambda}{1-\lambda} x \tag{319}$$

$$\iff x = \frac{1-\lambda}{\lambda} \sum_{i=1}^{\infty} \lambda^i G_t^{(i)} \tag{320}$$

$$\iff x = \frac{1-\lambda}{\lambda} \sum_{i=0}^{\infty} \lambda^{i+1} G_t^{(i+1)} \tag{321}$$

$$\iff x = (1-\lambda) \sum_{i=0}^{\infty} \lambda^i G_t^{(i+1)} \tag{322}$$

$$\iff x = G_t^\lambda. \tag{323}$$

□

Notice that the conditions required by Theorem 5 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).[6] 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.

———————————————End of Lecture 16, November 6, 2018———————————————

[6]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.

CMPSCI 687 Homework 4

Due November 16, 2018, 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 on Moodle as a .zip (.gz, .tar.gz, etc.) file containing your answers in a .pdf file and a folder with your source code. **Include with your source code instructions for how to run your code to produce the results you report below**. You may not use any reinforcement learning or machine learning specific libraries in your code (you may use libraries like C++ Eigen and numpy though). If you are unsure whether you can use a library, ask on Piazza. If you submit by November 25, you will not lose any credit. The automated system will not accept assignments after 11:55pm on November 25.

We will not be answering question on Piazza over Thanksgiving break, which begins at the end of the day on November 16. (We *may* answer some, but do not expect any answers). As in Homework 3, **you may not use existing RL code for this problem—you must implement the agent and environment entirely on your own and from scratch.** The exception to this is that you may re-use the code that you wrote for the previous homework assignments.

Programming: (75 Points)

This assignment only has a programming component. Implement Q-learning and Sarsa using both a tabular estimate of the q-function and linear function approximation and ϵ-greedy action selection. Apply these two algorithms to the 687-Gridworld and Cart-Pole problems. You may optimize hyperparameters (step size, ϵ, order of the Fourier basis) manually or using an automated search (e.g., a grid-search or random search over the space of hyperparameters).

1. **(30 Points)** For each domain (the gridworld and Cart-Pole) present a learning curve, where the horizontal axis spans 100 episodes. Include curves for both Sarsa and Q-learning. Average over at least 100 trials, and include standard error bars.

2. **(30 Points)** Try using a different function approximator for Cart-Pole, and compare the performances of both Sarsa and Q-learning with this alternate (possibly non-linear) function approximator to the performance you observed using the linear approximator and the Fourier basis. Your comparison should include text discussion of how difficult it was to tune each method (or empirical results if you used an automated hyperparameter optimization), as well as learning curves using the best found hyperparameters. Your alternate function approximator *must* be either *tile coding* (see the course book by Sutton and Barto (1998)), linear function approximation using the *polynomial basis* (see the class notes), *normalized radial basis functions* (see, for example, the appendix of the paper by (Doya, 2000)), or an *artificial neural network* (for example, a feed-forward fully-connected network with two layers and Sigmoid activation functions, although other networks are allowed). You need only select one of these alternate function approximators, and if you are looking to put in the minimal effort, I suggest using the polynomial basis. You may use code libraries that implement neural networks and backpropagation as long as they do not include any reinforcement learning code. If you want to use a function approximator not listed here, confirm that your alternate choice is ok with the instructors (and they will add it to the assignment). Discuss how difficult it was to tune these methods with the alternate function approximator relative to the Fourier basis. Discuss any noticeable differences in final performance and why you think these differences occur.

3. **(15 Points)** Reflect back on your implementation of the CEM algorithm for Homework 2. How do Sarsa and Q-learning compare? Create one plot for the gridworld and one plot for cart-pole that includes learning curves for CEM (use your results from Homework 2) and Sarsa and Q-learning (using any or all of the function approximators you tested). Which method was harder to tune? For other MDPs, when do you expect that CEM will be superior, and when do you expect that Sarsa and Q-learning will be superior?

4. **(10 Points Extra Credit)** Implement softmax action selection (see (59) in the class notes) for both methods, and optimize the temperature parameter, σ. Include in your plots and discussion the results using both ϵ-greedy and softmax action selection.

5. **(10 Points Extra Credit)** Implement the Mountain Car domain, apply Sarsa and Q-learning, and report your results.

12 Polynomial Basis

This material was not covered in class, but is being presented to simplify Homework 4. The polynomial basis can be constructed using the same vectors, c_i, used by the Fourier basis. Whereas the i^{th} feature using the Fourier basis is $\cos(\pi c_i \cdot \phi(s))$, the i^{th} feature using the polynomial basis is

$$\prod_{j=1}^{n} (s_j)^{c_{i,j}}, \tag{324}$$

where $c_{i,j}$ is the j^{th} element in the vector c_i and $s = (s_1, s_2, \ldots, s_n)$. That is, where the state is n-dimensional and s_j is the j^{th} component of the state (a real number, normalized to $[0, 1]$ or $[-1, 1]$).

12.1 λ-Return Algorithm

We can use the λ-return as a target to obtain a policy evaluation algorithm. The resulting update is:

$$v(S_t) \leftarrow v(S_t) + \alpha(G_t^\lambda - v(S_t)). \tag{325}$$

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:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq S_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{otherwise.} \end{cases} \tag{326}$$

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:

$$\delta_t = R_t + \gamma v(S_{t+1}) - v(S_t) \tag{327}$$

$$\forall s \in \mathcal{S},\ e(s) = \gamma\lambda e(s) \tag{328}$$

$$e(S_t) = e(S_t) + 1 \tag{329}$$

$$\forall s \in \mathcal{S},\ v(s) = v(s) + \alpha\delta_t e(s). \tag{330}$$

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 (221).

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 (325) and the updates in (327), the resulting value function estimates will be (approximately) equal. To see this, we begin by establishing some notation.

First, recall that

$$\delta_t = R_t + \gamma v_t(S_{t+1}) - v_t(S_t), \tag{331}$$

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:

$$e_t(s) = \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathcal{I}_{s,S_k}. \tag{332}$$

Unlike (326), 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,

$$\Delta v_t^F(s) = \alpha(G_t^\lambda - v_t(S_t)), \tag{333}$$

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:

$$\Delta v_t^B(s) = \alpha\delta_t e_t(s). \tag{334}$$

In Theorem 6 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 6. *For all $s \in \mathcal{S}$,*

$$\sum_{t=0}^{L} \Delta v_t^B(s) \approx \sum_{t=0}^{L} \Delta v_t^F(s) \mathcal{I}_{s,S_t}. \tag{335}$$

Proof. We begin with the left side:

$$\sum_{t=0}^{L} \Delta v_t^B(s) \overset{\text{(a)}}{=} \sum_{t=0}^{L} \alpha \delta_t e_t(s) \tag{336}$$

$$\overset{\text{(b)}}{=} \sum_{t=0}^{L} \alpha \delta_t \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathcal{I}_{s,S_k} \tag{337}$$

$$\overset{\text{(c)}}{=} \sum_{t=0}^{L} \alpha \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathcal{I}_{s,S_k} \delta_t \tag{338}$$

$$\overset{\text{(d)}}{=} \sum_{k=0}^{L} \alpha \sum_{t=0}^{k} (\gamma\lambda)^{k-t} \mathcal{I}_{s,S_t} \delta_k, \tag{339}$$

where **(a)** comes from (334), **(b)** comes from (332), **(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:

$$\sum_{t=0}^{L} \Delta v_t^B(s) = \sum_{t=0}^{L} \alpha \sum_{k=t}^{L} (\gamma\lambda)^{k-t} \mathcal{I}_{s,S_t} \delta_k \tag{340}$$

$$\overset{\text{(a)}}{=} \sum_{t=0}^{L} \alpha \mathcal{I}_{s,S_t} \sum_{k=t}^{L} (\gamma\lambda)^{k-t} \delta_k. \tag{341}$$

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:

$$\sum_{t=0}^{L} \Delta v_t^B(s) = \sum_{t=0}^{L} \alpha \mathcal{I}_{s,S_t} \sum_{k=t}^{L} (\gamma\lambda)^{k-t} \delta_k. \tag{342}$$

We now turn to the right hand side of (335), and consider the update at a single time step:

$$\Delta v_t^F(S_t) = \alpha(G_t^\lambda - v_t(S_t)). \tag{343}$$

Dividing both sides be α, we obtain:

$$\frac{1}{\alpha} \Delta v_t^F(S_t) = G_t^\lambda - v_t(S_t) \tag{344}$$

$$= \quad - v_t(S_t) + (1-\lambda)\lambda^0 (R_t + \gamma v_t(S_{t+1})) \tag{345}$$

$$+ (1-\lambda)\lambda^1 (R_t + \gamma R_{t+1} \quad + \gamma^2 v_t(S_{t+2})) \tag{346}$$

$$+ (1-\lambda)\lambda^1 (R_t + \gamma R_{t+1} \quad + \gamma^2 R_{t+2} \quad + \gamma^3 v_t(S_{t+3})) \tag{347}$$

$$\vdots \tag{348}$$

Consider all of the R_t terms:

$$\sum_{i=0}^{\infty} (1-\lambda)\lambda^i R_t = \frac{1-\lambda}{1-\lambda} R_t \tag{349}$$

$$= R_t. \tag{350}$$

Now consider all of the R_{t+1} terms:

$$\sum_{i=1}^{\infty}(1-\lambda)\lambda^i\gamma R_{t+1} =(1-\lambda)(\gamma\lambda)\sum_{i=0}^{\infty}\lambda^i R_{t+1} \tag{351}$$

$$=(\gamma\lambda)R_{t+1}. \tag{352}$$

In general, all of the R_{t+k} terms combine to $(\gamma\lambda)^k R_{t+k}$. We now rewrite (348), but combining all of the R_t terms, R_{t+1} terms, etc.

$$\frac{1}{\alpha}\Delta v_t^F(S_t) = - v_t(S_t) \tag{353}$$

$$+ R_t \qquad + (1-\lambda)\gamma v_t(S_{t+1}) \tag{354}$$

$$+ (\gamma\lambda)R_{t+1} \quad + (1-\lambda)(\gamma\lambda)\gamma v_t(S_{t+2}) \tag{355}$$

$$+ (\gamma\lambda)^2 R_{t+2} + (1-\lambda)(\gamma\lambda)^2\gamma v_t(S_{t+3}) \tag{356}$$

$$\vdots \tag{357}$$

Pulling out a $(\gamma\lambda)^i$ from each row and expanding the $(1-\lambda)$ term, we obtain:

$$\frac{1}{\alpha}\Delta v_t^F(S_t) = - v_t(S_t) \tag{358}$$

$$+ (\gamma\lambda)^0\left(R_t + \gamma v_t(S_{t+1}) - \gamma\lambda v_t(S_{t+1})\right) \tag{359}$$

$$+ (\gamma\lambda)^1\left(R_{t+1} + \gamma v_t(S_{t+2}) - \gamma\lambda v_t(S_{t+2})\right) \tag{360}$$

$$+ (\gamma\lambda)^2\left(R_{t+2} + \gamma v_t(S_{t+3}) - \gamma\lambda v_t(S_{t+3})\right) \tag{361}$$

$$\vdots \tag{362}$$

Shifting the right-most v_t terms all down one line, and plugging the $-v_t(S_t)$ from (358) into (359), we obtain:

$$\frac{1}{\alpha}\Delta v_t^F(S_t) =(\gamma\lambda)^0\left(R_t + \gamma v_t(S_{t+1}) - v_t(S_t)\right) \tag{363}$$

$$+ (\gamma\lambda)^1\left(R_{t+1} + \gamma v_t(S_{t+2}) - v_t(S_{t+1})\right) \tag{364}$$

$$+ (\gamma\lambda)^2\left(R_{t+2} + \gamma v_t(S_{t+3}) - v_t(S_{t+2})\right) \tag{365}$$

$$\vdots \tag{366}$$

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:

$$\frac{1}{\alpha}\Delta v_t^F(S_t) \approx \sum_{k=0}^{\infty}(\gamma\lambda)^k\delta_{t+k} \tag{367}$$

$$= \sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k \tag{368}$$

$$= \sum_{k=t}^{L}(\gamma\lambda)^{k-t}\delta_k, \tag{369}$$

since $\delta_k = 0$ if $k > L$. So, returning to the right side of (335),

$$\sum_{t=0}^{L}\Delta v_t^F(S_t)\mathcal{I}_{s,S_t} \approx \sum_{t=0}^{L}\alpha\sum_{k=t}^{L}(\gamma\lambda)^{k-t}\delta_k\mathcal{I}_{s,S_t} \tag{370}$$

$$= \sum_{t=0}^{L}\alpha\mathcal{I}_{s,S_t}\sum_{k=t}^{L}(\gamma\lambda)^{k-t}\delta_k, \tag{371}$$

which is the same as the left side, as expressed in (342). □

Name:

CMPSCI 687 Pop Quiz 8
November 13, 2018

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. A positive TD error means that:

   (a) The actual outcome was better than expected.

   (b) The actual outcome was worse than expected.

   (c) A terminal state has been reached.

2. (True or False) Although derived differently (via the forwards and backwards views), γ and λ are approximately equivalent—increasing one is equivalent to increasing the other.

3. The TD error is defined as:

   $$\delta_t = R_t + \gamma v(S_{t+1}) - v(S_t). \tag{372}$$

   We will accept other forms (using v^π, q^π, q, and (s, a, r, s', a') or $(S_t, A_t, R_t, S_{t+1}, A_{t+1})$.

4. (True or False) If a real-world problem has small amounts of noise in the sensors making observations of the state, then reinforcement learning methods should not be applied.

5. (True or False) TD is stochastic gradient descent on the mean-squared TD-error.

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). The same goes for true online Sarsa(λ) (the control form of true online TD(λ)).

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 16, and pseudocode for $Q(\lambda)$ is provided in Algorithm 17. In both algorithms, e is the vector of eligibility traces—one real-valued trace per weight.

Algorithm 16: 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 17: $Q(\lambda)$

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 28. *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 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:

$$\delta_t \leftarrow R_t + \gamma v(S_{t+1}) - v(S_t) \tag{373}$$

$$v(S_t) \leftarrow v(s) + \alpha\delta_t. \tag{374}$$

We could then use a softmax policy, parameterized as in (59). To capture our intuition, we can then use the actor update rule:

$$p(S_t, A_t) \leftarrow P(S_t, A_t) + \alpha\delta_t. \tag{375}$$

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.

14.1 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:

$$J(\theta) = \mathbf{E}\left[G|\theta\right], \tag{376}$$

and then we can perform gradient ascent to search for policy parameters that maximize the expected discounted return:

$$\theta \leftarrow \theta + \alpha\nabla J(\theta). \tag{377}$$

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?

———————————End of Lecture 17, November 13, 2018———————————

14.2 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 7 (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,*

$$\nabla J(\theta) = \sum_{s \in \mathcal{S}} d^\theta(s) \sum_{a \in \mathcal{A}} q^\pi(s,a) \frac{\partial \pi(s,a,\theta)}{\partial \theta}, \tag{378}$$

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

$$\frac{\partial \ln(f(x))}{\partial x} = \frac{1}{x} \frac{\partial f(x)}{\partial x}, \tag{379}$$

and so

$$\frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta} = \frac{1}{\pi(s,a,\theta)} \frac{\partial \pi(s,a,\theta)}{\partial \theta}. \tag{380}$$

Thus, we can rewrite (378) as:

$$\nabla J(\theta) = \sum_{s \in \mathcal{S}} d^\theta(s) \sum_{a \in \mathcal{A}} \pi(s,a,\theta) q^\pi(s,a) \frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta}. \tag{381}$$

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:

$$\nabla J(\theta) = \mathbf{E}\left[q^\pi(S,A) \frac{\partial \ln(\pi(S,A,\theta))}{\partial \theta} \middle| \theta \right], \tag{382}$$

where $S \sim d^\theta$ and $A \sim \pi(S,\cdot,\theta)$.

To obtain an intuitive understanding of (378), 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:

$$\nabla J(\theta) = \underbrace{\sum_{s \in \mathcal{S}} d^\theta(s)}_{\text{average over states}} \underbrace{\sum_{a \in \mathcal{A}} \pi(s,a,\theta)}_{\text{average over actions}} \underbrace{q^\pi(s,a)}_{\text{how good is } a \text{ in } s?} \underbrace{\frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta}}_{\text{How to change } \theta \text{ to make } a \text{ more likely in } s}. \tag{383}$$

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:

$$\nabla J(\theta) \propto \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t q^\pi(S_t, A_t) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \middle| \theta \right], \tag{384}$$

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 (384). For further discussion of this term (and why it is usually ignored), see the work of Thomas and Barto (2012); Thomas (2014).

14.3 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,

$$J(\theta) = \mathbf{E}[G|\theta] \tag{385}$$

$$= v^\theta(s_0). \tag{386}$$

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 $s \in \mathcal{S}$:

$$\frac{\partial v^\theta(s)}{\partial \theta} = \frac{\partial}{\partial \theta} \sum_{a \in \mathcal{A}} \pi(s,a,\theta) q^\theta(s,a) \tag{387}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \pi(s,a,\theta) \frac{\partial q^\theta(s,a)}{\partial \theta} \tag{388}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \sum_{a \in \mathcal{A}} \pi(s,a,\theta) \frac{\partial}{\partial \theta} \sum_{s' \in \mathcal{S}} P(s,a,s') \left(R(s,a) + \gamma v^\theta(s') \right) \tag{389}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \frac{\partial v^\theta(s')}{\partial \theta} \tag{390}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \frac{\partial}{\partial \theta} \left(\sum_{a' \in \mathcal{A}} \pi(s',a',\theta) q^\theta(s',a') \right) \tag{391}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \left(\sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a') + \pi(s',a',\theta) \frac{\partial q^\theta(s',a')}{\partial \theta} \right) \tag{392}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a') \tag{393}$$

$$+ \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \sum_{a' \in \mathcal{A}} \pi(s',a',\theta) \frac{\partial q^\theta(s',a')}{\partial \theta} \tag{394}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a') \tag{395}$$

$$+ \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \sum_{a' \in \mathcal{A}} \pi(s',a',\theta) \frac{\partial}{\partial \theta} \left(\sum_{s'' \in \mathcal{S}} P(s',a',s'') (R(s',a') + \gamma v^\theta(s'')) \right) \tag{396}$$

$$= \sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a) + \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a') \tag{397}$$

$$+ \gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \sum_{a' \in \mathcal{A}} \pi(s',a',\theta) \sum_{s'' \in \mathcal{S}} P(s',a',s'') \gamma \frac{\partial v^\theta(s'')}{\partial \theta} \tag{398}$$

$$= \underbrace{\sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a)}_{\text{first term}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a')}_{\text{second term}} \tag{399}$$

$$+ \gamma \sum_{s'' \in \mathcal{S}} \Pr(S_{t+2} = s'' | S_t = s, \theta) \gamma \frac{\partial v^\theta(s'')}{\partial \theta}. \tag{400}$$

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:

$$\frac{\partial v^\theta(s)}{\partial \theta} = \underbrace{\sum_{a \in \mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a)}_{\text{first term}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} \Pr(S_{t+1} = s' | S_t = s, \theta) \sum_{a' \in \mathcal{A}} \frac{\partial \pi(s',a',\theta)}{\partial \theta} q^\theta(s',a')}_{\text{second term}} \tag{401}$$

$$+ \underbrace{\gamma^2 \sum_{s'' \in \mathcal{S}} \Pr(S_{t+2} = s'' | S_t = s, \theta) \sum_{a'' \in \mathcal{A}} \frac{\partial \pi(s'',a'',\theta)}{\partial \theta} q^\theta(s'',a'')}_{\text{third term}} \tag{402}$$

$$+ \gamma^2 \sum_{s''' \in \mathcal{S}} \Pr(S_{t+r} = s''' | S_t = s, \theta) \gamma \frac{\partial v^\theta(s''')}{\partial \theta}. \tag{403}$$

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 $\ldots$ to denote that we could continue to unravel the expression):

$$\frac{\partial v^\theta(s)}{\partial \theta} = \underbrace{\sum_{a\in\mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a)}_{\text{first term}} + \underbrace{\gamma \sum_{x\in\mathcal{S}} \Pr(S_{t+1}=x|S_t=s,\theta) \sum_{a\in\mathcal{A}} \frac{\partial \pi(x,a,\theta)}{\partial \theta} q^\theta(x,a)}_{\text{second term}} \tag{404}$$

$$+ \underbrace{\gamma^2 \sum_{x\in\mathcal{S}} \Pr(S_{t+2}=x|S_t=s,\theta) \sum_{a\in\mathcal{A}} \frac{\partial \pi(x,a,\theta)}{\partial \theta} q^\theta(x,a)}_{\text{third term}} + \ldots \tag{405}$$

We now index each term by k, with the first term being $k=0$, the second $k=1$, etc., which results in the expression:

$$\frac{\partial v^\theta(s)}{\partial \theta} = \sum_{k=0}^{\infty} \sum_{x\in\mathcal{S}} \Pr(S_{t+k}=x|S_t=s,\theta) \sum_{a\in\mathcal{A}} \gamma^k \frac{\partial \pi(x,a,\theta)}{\partial \theta} q^\theta(x,a). \tag{406}$$

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 $\Pr(s\to x,k,\pi)$.

———————————End of Lecture 18, November 15, 2018———————————

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:

$$\begin{aligned}
\nabla J(\theta) =& \frac{\partial}{\partial \theta} J(\theta) && (407)\\
=& \frac{\partial}{\partial \theta} \mathbf{E}\left[G|\theta\right] && (408)\\
=& \frac{\partial}{\partial \theta} \mathbf{E}\left[G_t|S_t=s_0,\theta\right] && (409)\\
=& \frac{\partial}{\partial \theta} v^\theta(s_0) && (410)\\
=& \sum_{k=0}^{\infty} \sum_{x\in\mathcal{S}} \Pr(S_{t+k}=x|S_t=s_0,\theta) \sum_{a\in\mathcal{A}} \gamma^k \frac{\partial \pi(x,a,\theta)}{\partial \theta} q^\theta(x,a) && (411)\\
=& \sum_{x\in\mathcal{S}} \underbrace{\sum_{k=0}^{\infty} \gamma^k \Pr(S_{t+k}=x|S_t=s_0,\theta)}_{=d^\theta(x)} \sum_{a\in\mathcal{A}} \frac{\partial \pi(x,a,\theta)}{\partial \theta} q^\theta(x,a) && (412)\\
=& \sum_{s\in\mathcal{S}} d^\theta(s) \sum_{a\in\mathcal{A}} \frac{\partial \pi(s,a,\theta)}{\partial \theta} q^\theta(s,a), && (413)
\end{aligned}$$

where the last term comes from replacing the symbol x with the symbol s. This completes the proof of the policy gradient theorem.

14.4 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:

$$\nabla J(\theta) \propto \mathbf{E}\left[\gamma^t q^\theta(S_t,A_t) \frac{\partial \ln \pi(S_t,A_t,\theta)}{\partial \theta} \middle| \theta\right], \tag{414}$$

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:

$$\nabla J(\theta) \propto \mathbf{E}\left[\sum_{t=0}^{L} \gamma^t q^\theta(S_t, A_t) \frac{\partial \ln \pi(S_t, A_t, \theta)}{\partial \theta} \middle| \theta\right]. \tag{415}$$

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 18 we present the unbiased REINFORCE algorithm—true stochastic gradient ascent on J.

Algorithm 18: 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, ..., S_{L-1}, A_{L-1}, R_{L-1} using policy parameters θ.;
    ∇̂J(θ) = Σ_{t=0}^{L-1} γ^t G_t ∂ln(π(S_t, A_t, θ))/∂θ;
    θ ← θ + α ∇̂J(θ);
```

Question 29. *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 28. *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 $\Delta \ln \pi(A_t|S_t, \theta)$ 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 $\hat{mu} = X$. This estimator is unbiased: $\mathbf{E}[\hat{\mu}] = \mathbf{E}[X] = \mu$, and has variance $\mathrm{Var}(\hat{\mu}) = \mathrm{Var}(X)$.

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:

$$\begin{aligned}
\mathbf{E}[\hat{\mu}] &= \mathbf{E}[X - Y + \mathbf{E}[Y]] && (416)\\
&= \mathbf{E}[X] - \mathbf{E}[Y] + \mathbf{E}[Y] && (417)\\
&= \mathbf{E}[X] && (418)\\
&= \mu. && (419)
\end{aligned}$$

However, its variance is:

$$\begin{aligned}
\mathrm{Var}(\hat{\mu}) &= \mathrm{Var}(X - Y + \mathbf{E}[Y]) && (420)\\
&= \mathrm{Var}(X - Y) && (421)\\
&= \mathrm{Var}(X) + \mathrm{Var}(Y) - 2\,\mathrm{Cov}(X, Y). && (422)
\end{aligned}$$

This variance is lower than the variance of the original estimator when:

$$\mathrm{Var}(X) + \mathrm{Var}(Y) - 2\,\mathrm{Cov}(X, Y) < \mathrm{Var}(X), \tag{423}$$

or equivalently, when

$$\text{Var}(Y) < 2\,\text{Cov}(X, Y). \tag{424}$$

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:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{L-1} \gamma^t (G_t - b(S_t)) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}, \tag{425}$$

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:

$$\theta \leftarrow \theta + \underbrace{\alpha \sum_{t=0}^{L-1} \gamma^t G_t \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}}_{X} - \underbrace{\alpha \sum_{t=0}^{L-1} \gamma^t b(S_t) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}}_{Y}, \tag{426}$$

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:

$$\begin{aligned}
\mathbf{E}\left[\alpha \sum_{t=0}^{L-1} \gamma^t b(S_t) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \middle| \theta\right] &= \alpha \sum_{t=0}^{L-1} \gamma^t \mathbf{E}\left[b(S_t) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \middle| \theta\right] && (427)\\
&= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s|\theta) \sum_{a \in \mathcal{A}} \Pr(A_t = a|S_t = s, \theta) b(s) \frac{\partial \ln(\pi(s, a, \theta))}{\partial \theta} && (428)\\
&= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s|\theta) b(s) \sum_{a \in \mathcal{A}} \pi(s, a, \theta) \frac{\partial \ln(\pi(s, a, \theta))}{\partial \theta} && (429)\\
&= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s|\theta) b(s) \sum_{a \in \mathcal{A}} \frac{\partial \pi(s, a, \theta)}{\partial \theta} && (430)\\
&= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s|\theta) b(s) \frac{\partial}{\partial \theta} \sum_{a \in \mathcal{A}} \pi(s, a, \theta) && (431)\\
&= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s|\theta) b(s) \frac{\partial}{\partial \theta} 1 && (432)\\
&= \alpha \sum_{t=0}^{L-1} \gamma^t \sum_{s \in \mathcal{S}} \Pr(S_t = s|\theta) b(s) 0 && (433)\\
&= 0. && (434)
\end{aligned}$$

So, inserting the $b(s)$ control variate in (425) 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.[7]

[7] 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 19.

Algorithm 19: 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.

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:

$$\widehat{\nabla J(\theta)} = \widehat{\nabla J(\theta)} + \gamma^t (R_t + \gamma v_w(S_{t+1}) - v_w(S_t)) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}. \tag{435}$$

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 20.

Algorithm 20: 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 23 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:

$$\theta \leftarrow \theta + \alpha \gamma^t \mathbf{E}\left[(q^\theta(S_t, A_t) - v^\theta(S_t)) \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \middle| \theta \right], \tag{436}$$

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 21.

Algorithm 21: 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 21 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 $n = |\mathcal{A}|$).

End of Lecture 19, November 27, 2018

CMPSCI 687 Homework 5

Due December 11, 2018, 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 on Moodle as a .zip (.gz, .tar.gz, etc.) file containing your answers in a .pdf file and a folder with your source code. **Include with your source code instructions for how to run your code to produce the results you report below**. You may not use any reinforcement learning or machine learning specific libraries in your code (you may use libraries like C++ Eigen and numpy 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 December 11.

As in Homework 3, **you may not use existing RL code for this problem—you must implement the agent and environment entirely on your own and from scratch.** The exception to this is that you may re-use the code that you wrote for the previous homework assignments.

Written: (20 Points Total)

1. (5 Points) If A_t is sampled from a normal distribution with mean $\theta^\intercal \phi(S_t)$ and variance σ^2, what is $\frac{\partial}{\partial \theta} \ln(\pi(s,a,\theta))$ (here $\pi(s,\cdot,\theta)$ is a probability density function)? If σ is considered a tunable policy parameter, what is $\frac{\partial}{\partial \sigma} \ln(\pi(s,a,\theta))$?

$$\begin{aligned}
\frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta} &= \frac{1}{\pi(s,a,\theta)} \frac{\partial}{\partial \theta} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(a-\theta^\intercal\phi(s))^2}{2\sigma^2}\right) && (437)\\
&= \frac{1}{\pi(s,a,\theta)} \pi(s,a,\theta) \frac{\partial}{\partial \theta} \frac{(a-\theta^\intercal\phi(s))^2}{2\sigma^2} && (438)\\
&= \frac{1}{2\sigma^2} 2(a-\theta^\intercal\phi(s))(-\phi(s)) && (439)\\
&= \frac{-1}{\sigma^2}(a-\theta^\intercal\phi(s))\phi(s). && (440)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \ln(\pi(s,a,\theta))}{\partial \sigma} &= \frac{1}{\pi(s,a,\theta)} \frac{\partial}{\partial \sigma} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(a-\theta^\intercal\phi(s))^2}{2\sigma^2}\right) && (441)\\
&= \frac{1}{\pi(s,a,\theta)} \left(\frac{1}{\sqrt{2\pi\sigma^2}} \frac{\partial}{\partial \sigma} \exp\left(\frac{(a-\theta^\intercal\phi(s))^2}{2\sigma^2}\right) + \exp\left(\frac{(a-\theta^\intercal\phi(s))^2}{2\sigma^2}\right) \frac{\partial}{\partial \theta}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)\right) && (442)\\
&= \frac{\partial}{\partial \theta}\left(\frac{1}{2\sigma^2}(a-\theta^\intercal\phi(s))^2\right) + \frac{1}{\pi(s,a,\theta)} \exp\left(\frac{(a-\theta^\intercal\phi(s))^2}{2\sigma^2}\right) \frac{1}{\sqrt{2\pi}} \frac{\partial}{\partial \theta} \frac{1}{\sqrt{\sigma^2}} && (443)\\
&= \frac{-(a-\theta^\intercal\phi(s))^2}{\sigma^3} + \frac{1}{\pi(s,a,\theta)} \exp\left(\frac{(a-\theta^\intercal\phi(s))^2}{2\sigma^2}\right) \frac{1}{\sqrt{2\pi}} \left(\frac{-\sigma}{|\sigma|^3}\right) && (444)\\
&= \frac{-(a-\theta^\intercal\phi(s))^2}{\sigma^3} - \frac{\sigma}{|\sigma|}. && (445)
\end{aligned}$$

2. (15 Points) Consider evaluating a fixed policy, π. The fixed-point of the TD update using linear function approximation occurs when the expected TD update is zero—when $\mathbf{E}[(R_t + \gamma w^\intercal \phi(S_{t+1}) - w^\intercal \phi(S_t))\phi(S_t)] = 0$. This can be equivalently written as:

$$\mathbf{E}[R_t \phi(S_t)] + \mathbf{E}\left[\phi(S_t)\big(\gamma\phi(S_{t+1}) - \phi(S_t)\big)^\intercal\right] w = 0, \tag{446}$$

or

$$\underbrace{\mathbf{E}\left[\phi(S_t)\big(\phi(S_t) - \gamma\phi(S_{t+1})\big)^\intercal\right]}_{A} w = \underbrace{\mathbf{E}[R_t\phi(S_t)]}_{b}. \tag{447}$$

Consider the problem of estimating the state-value function for a policy, π, using a fixed batch of data containing many (s,a,r,s') tuples generated from running the policy. One way to do this would be to run linear TD over and over on the data until it converges. Another would be do directly solve for the fixed-point of TD by solving the least-squares problem in (447), which has the form $Aw = b$, where the data could be used to construct unbiased estimates of A and b, and the system of linear equations solved to obtain the fixed-point for w. This algorithm is called Least Squared Temporal Difference Learning (LSTD), and you can read about it in the work of Boyan (1999), or the original paper by Bradtke and Barto

(1996). For this problem, derive the corresponding least-squares algorithm for the residual-gradient-like algorithm (see (271))[8] that uses the update:[9]

$$w \leftarrow w + \alpha(R_t + \gamma w^\intercal \phi(S_{t+1}) - w^\intercal \phi(S_t))(\phi(S_t) - \gamma\phi(S_{t+1})). \tag{448}$$

Show the derivation of the corresponding least squares problem for this residual-gradient-like update and present pseudocode for your least squares residual-gradient-like algorithm.

We want to solve for the weights, w, such that the expected update is zero:

$$\mathbf{E}\left[(R_t + \gamma w^\intercal \phi(S_{t+1}) - w^\intercal \phi(S_t))(\phi(S_t) - \gamma\phi(S_{t+1}))\right] = 0. \tag{449}$$

We begin by putting this into the form $Aw = b$, with new definitions of A and b.

$$\begin{aligned}
\mathbf{E}\left[(\gamma w^\intercal \phi(S_{t+1}) - w^\intercal \phi(S_t))(\phi(S_t) - \gamma\phi(S_{t+1}))\right] &= \mathbf{E}\left[R_t(\gamma\phi(S_{t+1}) - \phi(S_t))\right] && (450)\\
\mathbf{E}\left[\left(w^\intercal(\gamma\phi(S_{t+1}) - \phi(S_t))\right)(\phi(S_t) - \gamma\phi(S_{t+1}))\right] &= \mathbf{E}\left[R_t(\gamma\phi(S_{t+1}) - \phi(S_t))\right] && (451)\\
\mathbf{E}\left[(\phi(S_t) - \gamma\phi(S_{t+1}))(\gamma\phi(S_{t+1}) - \phi(S_t))^\intercal\right] &= \mathbf{E}\left[R_t(\gamma\phi(S_{t+1}) - \phi(S_t))\right] && (452)\\
\mathbf{E}\left[(\phi(S_t) - \gamma\phi(S_{t+1}))(\gamma\phi(S_{t+1}) - \phi(S_t))^\intercal\right] w &= \mathbf{E}\left[R_t(\gamma\phi(S_{t+1}) - \phi(S_t))\right]. && (453)
\end{aligned}$$

For full credit, pseudocode similar to that of Bradtke and Barto (1996) should be provided as well.

Programming: (50 Points)

1. (20 Points) Convert your Q-learning and Sarsa implementations into implementations of Q(λ) and Sarsa(λ), and apply these algorithms to the gridworld and to mountain car (using the function approximator of your choice). Present learning curves for Q-learning, Sarsa, Q(λ), and Sarsa(λ), all using the best parameters that you found for each method, and with one plot for the gridworld and one plot for Mountain Car. Run at least 100 trials and include error bars. Comment on this experience—did you observe a big difference between the λ and non-λ variants of these algorithms?

2. (30 Points) Implement the actor-critic algorithm presented in Algorithm 21. Use the function approximator of your choice. Apply the algorithm to the gridworld and mountain car domains. Present learning curves using the best parameters that you found, compare to your results from the previous problem (you may include all methods in two plots—one per domain—you do not need separate plots for each problem). Describe the function approximator that you used, as well as the best hyperparameters that you found. Write a brief description of this process: how hard was it to tune the actor-critic? How well did you find it worked in comparison to the other algorithms?

3. (15 Points Extra Credit) Implement REINFORCE with or without the baseline. Apply it to the gridworld problem and present the resulting learning curve, which should obtain a mean return of at least 1.55. Describe the policy representation you used and any hyperparameter settings. *Try* to get REINFORCE working on Mountain Car, and report the best results that you find and the hyperparameters that you used. Describe this experience: how hard was it to tune REINFORCE and how well did it work?

[8] This link will not work in the homework assignment on its own, but will work in the class notes

[9] This algorithm is stochastic gradient descent on the squared TD error, not the squared Bellman error, which, as you saw in a previous assignment, is *not* what we actually want to minimize.

15 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:

$$F(\theta) = \sum_x d_\theta(x) \frac{\partial d_\theta(x)}{\partial \theta} \frac{\partial d_\theta(x)}{\partial \theta}^\intercal, \tag{454}$$

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 reprameterization" 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:

$$L(w) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi(s, a, \theta) \left(q^\pi(s, a) - q_w(s, a) \right)^2, \tag{455}$$

where

$$q_w(s, a) = w^\intercal \frac{\partial \ln(\pi(s, a, \theta))}{\partial \theta}, \tag{456}$$

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. (2012a, 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.[10] 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 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

[10] 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.

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).

———————————————End of Lecture 20, November 29, 2018———————————————

Andy Barto gave a guest lecture on designing reward functions. His slides are available as a .key or .pdf.

———————————————End of Lecture 21, December 4, 2018———————————————

Andy Barto gave a guest lecture on neuroscience and psychology (and their relation to reinforcement learning). His slides are available as a .key or .pdf.

———————————————End of Lecture 22, December 6, 2018———————————————

16 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).

16.1 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).

16.2 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).

16.3 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 $f_1, \ldots, f_n$ are n objective functions, then the Pareto frontier is the set

$$P \coloneqq \left\{\theta \in \Theta : \forall \theta' \in \Theta, \left(\exists i \in \{1, \ldots, n\},\ f_i(\theta') > f_i(\theta) \implies \exists j \in \{1, \ldots, n\},\ f_j(\theta') < f_j(\theta)\right)\right\}. \tag{457}$$

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).

16.4 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).

16.5 Off-Policy Policy Evaluation

As you have seen throughout the assignments for this course, reinforcement learning algorithms almost never work on the first try. If any of the hyperparameters is not set properly for the problem at hand, or if the wrong representation is used, the algorithm might fail to learn (or diverge). In some cases, no settings of the hyperparameters will allow for convergence (c.f., REINFORCE for Mountain Car, or Q(0) for Cart-Pole), unless the algorithm is modified in some way. How then can an RL algorithm be applied responsibly to a high-risk problem (e.g., medical applications, autonomous vehicle applications, etc.)?

Off-policy policy evaluation algorithms aim to give users confidence that a newly proposed policy will work well if it were to be used. The goal is to evaluate a new policy using data collected from executing some current policy. For a review of off-policy policy evaluation methods, begin with the work of Precup et al. (2000), which presents *importance sampling*, as a way to estimate the performance of one policy using data from another. Then, see the work of Thomas et al. (2015b), for a description of how we can construct confidence intervals around these predictions, and then the work of Thomas et al. (2015c) for examples of how these tools can all be combined to create RL algorithms that guarantee that they will only change the current policy if it will result in an increased expected return (with probability at least $1 - \delta$). A further example of a "safe RL" algorithm of this sort is presented as an example in the work of Thomas et al. (2017).

16.6 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 23, December 11, 2018———————————————

Name: _____

CMPSCI 687 Final Exam

December 18, 2018

Instructions: This exam is **closed** notes. Do not use any notes or electronic devices. You have until 8:00pm to complete this exam.

Problem	Possible Points	Points Obtained	Comments
1 (total)	15		
1a	3		
1b	3		
1c	3		
1d	3		
1e	3		
2 (total)	5		
3 (total)	5		
4 (total)	15		
4a	3		
4b	3		
4c	3		
4d	3		
4e	3		
5 (total)	15		
6 (total)	15		
7 (total)	15		
7a	5		
7b	4		
7c	6		
8 (total)	15		
8a	3		
8b	6		
8c	6		

Exam Total Score (out of 100): _____

1. [15%] Define the following (using math, and the notation used in class):

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

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

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

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

   - Using a state-value function estimate, v, the TD error is defined as

$$\delta_t = R_t + \gamma v(S_{t+1}) - v(S_t)$$

2. [5%] Let v^π be the value function for the policy π. Write an expression for the greedy action with respect to v^π in state s:

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

3. [5%] Write the Bellman equation (using state-values, not action-values):

$$v^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \sum_{s' \in \mathcal{S}} P(s,a,s')(R(s,a) + \gamma v^\pi(s')). \tag{458}$$

4. [15%] For each of the following, circle either true or false:

   - (True or False) Every finite MDP with bounded rewards and $\gamma \in [0, 1)$ has at least one optimal policy.
   - (True or False) All policy gradient algorithms are actor-critic algorithms, but not all actor-critic algorithms are policy gradient algorithms.
   - (True or False) Using smaller values of λ in TD(λ) is one way to further discount rewards (i.e., similar to decreasing γ).
   - (True or False) In class, Andy Barto discussed research suggesting that the firing of some dopamine-producing neurons in human brains corresponds to a negative TD error.
   - (True or False) For an MDP with horizon $L > 0$, the $2L$-step return from S_t, $G_t^{(2L)}$, is always equal to the Monte Carlo return, G_t.

5. [15%] For this problem you will derive an expression for the policy gradient for a new objective function, J'. This new objective is *similar* to one used in off-policy actor-critics (Degris et al., 2012b). Assume that there is a fixed policy π_b. Let

$$d'(s) = \sum_{t=0}^{L-1} \Pr(S_t = s|\pi_b). \tag{459}$$

   The objective function J' is given by:

$$J'(\theta) = \sum_{s \in \mathcal{S}} d'(s) \mathbf{E}[R_t|S_t = s, \theta]. \tag{460}$$

   Derive an expression for the policy gradient for this objective. We will only grade the math written in the sequence of equations below. In the last line, fill in the terms inside the expectation. These terms should only be terms that we defined when defining an MDP (including the reward function), and the only terms that include differentiation with respect to θ should be $\partial\pi(\dots)/\partial\theta$ or $\partial \ln(\pi(\dots))/\partial\theta$. You do not have to use all of the lines provided. Your equations should fit easily in the space provided—if you are trying to write particularly small, you are doing something wrong. There is a blank page after this page that you can use to draft your answer before writing your final answer here. During the exam we stated that θ are *not* the parameters of π_b, but the parameters of some other policy, π, and that your final answer should not have d' terms.

$$\begin{aligned}
\frac{\partial J'(\theta)}{\partial \theta} &= \frac{\partial}{\partial \theta} \sum_{s \in \mathcal{S}} d'(s) \sum_{a \in \mathcal{A}} \pi(s, a, \theta) \mathbf{E}[R_t|S_t = s, A_t = a, \theta] \\
&= \sum_{s \in \mathcal{S}} \sum_{t=0}^{L-1} \Pr(S_t = s|\pi_b) \sum_{a \in \mathcal{A}} \frac{\partial}{\partial \theta} \pi(s, a, \theta) R(s, a) \\
&= \sum_{t=0}^{L-1} \sum_{s \in \mathcal{S}} \Pr(S_t = s|\pi_b) \sum_{a \in \mathcal{A}} R(s, a) \frac{\partial \pi(s, a, \theta)}{\partial \theta} \\
&= \sum_{t=0}^{L-1} \mathbf{E}\left[\sum_{a \in \mathcal{A}} R(S_t, a) \frac{\partial \pi(S_t, a, \theta)}{\partial \theta} \middle| \pi_b \right] \\
&= \mathbf{E}\left[\sum_{t=0}^{L-1} \sum_{a \in \mathcal{A}} R(S_t, a) \frac{\partial \pi(S_t, a, \theta)}{\partial \theta} \middle| \pi_b \right].
\end{aligned}$$

This page left intentionally blank.

6. [15%] Let π and μ be two different stochastic policies. Let $H = (S_0, A_0, R_0, S_1, \ldots, S_{L-1}, A_{L-1}, R_{L-1})$ be a history. For now, assume that rewards are sampled from a finite set, and $R(s, a, s', r) = \Pr(R_t = r | S_t = s, A_t = a, S_{t+1} = s')$. Show how the term below can be simplified so that it only contains terms from the definition of an MDP. Furthermore, ensure that your final expression is one that could be computed by an agent that does not know the transition function, reward distribution, or reward function. (After the exam, you can see how this result is used by Precup et al. (2000)).

$$\begin{aligned}
\frac{\Pr(H|\pi)}{\Pr(H|\mu)} &= \frac{d_0(S_0)\pi(S_0, A_0)P(S_0, A_0, S_1)R(S_0, A_0, S_1, R_0)\pi(S_1, A_1)P(S_1, A_1, S_2)R(S_1, A_1, S_2, R_1)\pi(S_2, A_2)\ldots}{d_0(S_0)\mu(S_0, A_0)P(S_0, A_0, S_1)R(S_0, A_0, S_1, R_0)\mu(S_1, A_1)P(S_1, A_1, S_2)R(S_1, A_1, S_2, R_1)\mu(S_2, A_2)\ldots} \\
&= \frac{\pi(S_0, A_0)\pi(S_1, A_1)\pi(S_2, A_2)\ldots}{\mu(S_0, A_0)\mu(S_1, A_1)\mu(S_2, A_2)\ldots} \\
&= \prod_{t=0}^{L-1} \frac{\pi(S_t, A_t)}{\mu(S_t, A_t)}.
\end{aligned}$$

7. [15%] Some natural actor-critic algorithms estimate the action-value function with a linear approximator $w^\intercal \frac{\partial \ln(\pi(s,a,\theta))}{\partial \theta} + v^\intercal \phi(s)$, where w and v are two weight vectors that together form a combined weight vector that parameterizes the q-approximation for a fixed policy, π. In this question we will derive an important property about estimating the action-value function in this way.

- (5%) Below, write out the TD(λ) update equations using this function approximator. Write e_v to denote the e-traces for the parameters v and e_w to denote the e-traces for the parameters w.

$$\delta \leftarrow R_t + \gamma \left(w^\intercal \frac{\partial \ln(\pi(S_{t+1}, A_{t+1}, \theta))}{\partial \theta} + v^\intercal \phi(S_{t+1}) \right) - w^\intercal \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} - v^\intercal \phi(S_t) \tag{461}$$

$$e_v \leftarrow \gamma \lambda e_v + \phi(S_t) \tag{462}$$

$$e_w \leftarrow \gamma \lambda e_w + \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \tag{463}$$

$$v \leftarrow v + \alpha \delta e_v \tag{464}$$

$$w \leftarrow w + \alpha \delta e_w \tag{465}$$

- (4%) Create updates for an equivalent algorithm, but using the provided update for δ. To get the right answer, you should include in the other updates corrections necessary to make up for the terms missing from the provided update for δ.

$$\delta \leftarrow R_t + \gamma v^\intercal \phi(S_{t+1}) - v^\intercal \phi(S_t) \tag{466}$$

$$e_v \leftarrow \gamma \lambda e_v + \phi(S_t) \tag{467}$$

$$e_w \leftarrow \gamma \lambda e_w + \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \tag{468}$$

$$v \leftarrow v + \alpha \left(\delta + \gamma w^\intercal \frac{\partial \ln(\pi(S_{t+1}, A_{t+1}, \theta))}{\partial \theta} - w^\intercal \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \right) e_v \tag{469}$$

$$w \leftarrow w + \alpha \left(\delta + \gamma w^\intercal \frac{\partial \ln(\pi(S_{t+1}, A_{t+1}, \theta))}{\partial \theta} - w^\intercal \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta} \right) e_w \tag{470}$$

- (6%) For stochastic gradient updates, adding mean-zero noise to gradient updates is not usually beneficial—it does not change the expected updates but increases the variance of the updates. Although TD(λ) is not a gradient algorithm, we usually do not want mean-zero terms in our updates. The updates above contain at least one mean-zero term that could be removed without changing the expected update. This term is easier to identify in the second form. Identify one term that is mean-zero noise in the update to v (and which depends on S_t), and prove that its expected value is zero. Use the equation blocks below. Only the math that you write will be graded. [Note: If you made a mistake in the previous questions, you may necessarily miss all points on this problem—double check your previous answers.]

$$\mathbf{E}\Bigg[\underbrace{\alpha\left(w^\intercal \frac{\partial \ln(\pi(S_t, A_t, \theta))}{\partial \theta}\right) e_v}_{\text{Your term here}} \Bigg| \pi\Bigg]$$

$$= \sum_{s\in\mathcal{S}} \Pr(S_t = s|\pi)\mathbf{E}\Bigg[\underbrace{\alpha\left(w^\intercal \frac{\partial \ln(\pi(s, A_t, \theta))}{\partial \theta}\right) e_v}_{\text{Your term here, replacing } S_t \text{ with } s} \Bigg| S_t = s, \pi\Bigg]$$

$$= \sum_{s\in\mathcal{S}} \Pr(S_t = s|\pi) \sum_{a\in\mathcal{A}} \pi(s, a, \theta)\alpha\left(w^\intercal \frac{\partial \ln(\pi(s, a, \theta))}{\partial \theta}\right)\phi(s)$$

$$= \sum_{s\in\mathcal{S}} \Pr(S_t = s|\pi)\alpha\left(w^\intercal \sum_{a\in\mathcal{A}} \pi(s, a, \theta)\frac{\partial \ln(\pi(s, a, \theta))}{\partial \theta}\right)\phi(s)$$

$$= \sum_{s\in\mathcal{S}} \Pr(S_t = s|\pi)\alpha\left(w^\intercal \sum_{a\in\mathcal{A}} \frac{\partial \pi(s, a, \theta)}{\partial \theta}\right)\phi(s)$$

$$= \sum_{s\in\mathcal{S}} \Pr(S_t = s|\pi)\alpha\left(w^\intercal \frac{\partial}{\partial \theta} \sum_{a\in\mathcal{A}} \pi(s, a, \theta)\right)\phi(s)$$

$$= \sum_{s\in\mathcal{S}} \Pr(S_t = s|\pi)\alpha\left(w^\intercal \underbrace{\frac{\partial}{\partial \theta}1}_{=0}\right)\phi(s)$$

$$= 0.$$

8. [15%] The TD(0) update for estimating the state-value function with the estimator v_w can be written in the form:

$$w \leftarrow w + \alpha \Delta w, \tag{471}$$

where Δw is the change to w at time step t due to the transition (S_t, A_t, R_t, S_{t+1}). To show that the TD update is not a gradient algorithm, we argue that, if it was, then Δw must be the gradient of the objective function, i.e., $\Delta w = \partial f(w)/\partial w$ for some function f. The gradient of a function, $\partial f(w)/\partial w$, is called the *Jacobian*, and sometimes written as $\nabla f(w)$. The *Hessian* of a function is $\partial^2 f(w)/\partial w^2$. The Hessian of all functions (with some smoothness assumptions) is symmetric. To show that TD is not a gradient algorithm, we will compute what the Hessian of $f(w)$ would be if the TD update was gradient ascent/descent on $f(w)$, and will argue that it is not always symmetric.

- (3%) Write the expression for Δw (this will be a vector):

$$\Delta w = (R_t + \gamma v_w(S_{t+1}) - v_w(S_t)) \frac{\partial v_w(S_t)}{\partial w} \tag{472}$$

- (6%) Write an expression for the entry in the i^{th} row and j^{th} column of the Hessian of f (this will be a scalar, while the Hessian is a matrix). Read the next question and ensure that your final expression allows you to answer the next question.
  During the exam, we said that here you could assume linear function approximation if you want. We give the answer for arbitrary function approximation. Note also the typo below where the order of i and j is reversed—this is inconsequential in the end—we've just solved for $\partial^2 f(w)/\partial w_j \partial w_i$.

$$\begin{aligned}
\frac{\partial^2 f(w)}{\partial w_i \partial w_j} &= \frac{\partial}{\partial w_j} \frac{\partial}{\partial w_i} f(w) && (473)\\
&= \frac{\partial}{\partial w_j} \Delta w_i && (474)\\
&= \frac{\partial}{\partial w_j} (R_t + \gamma v_w(S_{t+1}) - v_w(S_t)) \frac{\partial v_w(S_t)}{\partial w_i} && (475)\\
&= (R_t + \gamma v_w(S_{t+1}) - v_w(S_t)) \frac{\partial^2 v_w(S_t)}{\partial w_i \partial w_j} + \left(\frac{\partial}{\partial w_j} (R_t + \gamma v_w(S_{t+1}) - v_w(S_t)) \right)^\intercal \frac{\partial v_w(S_t)}{\partial w_i} && (476)\\
&= (R_t + \gamma v_w(S_{t+1}) - v_w(S_t)) \frac{\partial^2 v_w(S_t)}{\partial w_i \partial w_j} + \left(\gamma \frac{\partial v_w(S_{t+1})}{\partial w_j} - \frac{\partial v_w(S_t)}{\partial w_j} \right)^\intercal \frac{\partial v_w(S_t)}{\partial w_i} && (477)\\
& && (478)
\end{aligned}$$

  where Δw_i denotes the i^{th} entry in Δw.

- (6%) Explain why your final expression above implies that the Hessian would not be a symmetric matrix. Draft your answer elsewhere (e.g., the back of this page), and write your final answer clearly on this page. Your answer to this question may be a clear English description—it does not have to be formal math.

The first term, $(R_t + \gamma v_w(S_{t+1}) - v_w(S_t)) \frac{\partial^2 v_w(S_t)}{\partial w_i \partial w_j}$ is the same if i and j are swapped, since the Hessian of $v_w(S_t)$ is typically symmetric. However, the second term, $\left(\gamma \frac{\partial v_w(S_{t+1})}{\partial w_j} - \frac{\partial v_w(S_t)}{\partial w_j}\right)^\intercal \frac{\partial v_w(S_t)}{\partial w_i}$, is not necessarily the same if i and j are swapped. Specifically, consider the tabular setting where there is one weight per state and a case where $S_{t+1} \neq S_t$. If w_i is the weight for S_t and w_j is the weight for S_{t+1}, then the second term will be γ. However, swapping i and j so that w_i is not the weight for S_{t+1}, the second term is necessarily zero, since $\frac{\partial v_w(S_t)}{\partial w_i} = 0$.

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.

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.

J. Boyan. Least-squares temporal difference learning. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56, 1999.

S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

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*, 2012a.

T. Degris, M. White, and R. S. Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012b.

K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 2000.

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. *The Journal of Machine Learning Research*, 17(1):8055–8105, 2016.

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.

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_γ: 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.

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.

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, R. S. Sutton, and S. Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17th International Conference on Machine Learning*, pages 759–766, 2000.

M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

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.

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. 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 *Advances in Neural Information Processing Systems 29*, 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, B. C. da Silva, A. G. Barto, and E. Brunskill. On ensuring that intelligent machines are well-behaved. *arXiv preprint arXiv:1708.05448*, 2017.

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.