# CMPSCI 687: Reinforcement Learning

Fall 2019 Class Syllabus, Notes, and Assignments

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

## Table of Contents

1 Syllabus  
   1.1 Class ........................................... 2  
   1.2 Website ....................................... 2  
   1.3 Book .......................................... 2  
   1.4 Office Hours ................................... 2  
   1.5 Teaching Assistants and Office Hours .......... 2  
   1.6 Piazza ......................................... 2  
   1.7 Support Summary ................................ 3  
   1.8 Grading ....................................... 3  
   1.9 Late Policy .................................... 4  
   1.10 Missing Class / Assignments .................. 4  
   1.11 Disability Services ......................... 4  
   1.12 Cheating .................................... 4  
   1.13 LaTeX ........................................ 4  

2 Introduction ....................................... 5  
   2.1 Notation ...................................... 5  
   2.2 What is Reinforcement Learning (RL)? ........ 5  
   2.3 687-Gridworld: A Simple Environment .......... 9  
   2.4 Describing the Agent and Environment Mathematically ... 10
1 Syllabus

1.1 Class

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

1.2 Website

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

1.3 Book

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

1.4 Office Hours

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

1.5 Teaching Assistants and Office Hours

The teaching assistants (TAs) this semester will be Blossom Metevier (bmete-vier@umass.edu) and Scott Jordan (sjordan@cs.umass.edu). Blossom will have office hours on Thursdays from 8:55am–9:55am in CS Building room 207. Scott will have office hours on Tuesdays from 3pm–4pm in CS Building room 207.

1.6 Piazza

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

1.7 Support Summary

Office hours:

- **Mondays**: 1:30pm-3:00pm, Prof. Thomas’ office hours, CS room 346.
- **Tuesdays**: 3:00pm–4:00pm, Scott Jordan’s office hours, CS room 207.
- **Thursdays**: 8:55am–9:55am, Blossom Metevier’s office hours, CS room 207.

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

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

1.8 Grading

Your grade will have four components:

1. **Homework Assignments** (50%): There will be *roughly* seven homework assignments. Each problem in an assignment will specify its point value, and not all homework assignments will necessarily have the same point value (i.e., some homework assignments may be smaller and worth less than others). All assignments will have total point values of at most 100 (thus, the last assignment will not have a point value so high that previous assignments are irrelevant).

2. **Pop Quizzes** (15%): There will be pop-quizzes given in class without prior announcement. They will typically take about 10 minutes to complete and will be given at the start of class. If you know in advance that you will miss class, please e-mail both TAs, and you may be excused from any quizzes that occur that day.

3. **Midterm Exam** (15%): There will be a midterm exam relatively late in the semester (precise date TBD).

4. **Project** (20%): There will be a course project. The details of the project will be announced later in the semester, and may depend on how much content is covered.
A cumulative grade in $[90\% - 100\%]$ will be an A- or A, $[80\%, 90\%)$ will be a B-, B, or B+, and $[70\%, 80\%)$ will be a C-, C, or C+. Course grades will be curved only in students’ favor (that is, these thresholds may be lowered, but a grade of 90\% will not be lower than an A-).

1.9 Late Policy

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

1.10 Missing Class / Assignments

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

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

1.11 Disability Services

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

1.12 Cheating

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

1.13 \LaTeX

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

2.1 Notation

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

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

We typically use capital letters for matrices (e.g., \(A\)) and lowercase letters
for vectors (e.g., \(b\)). We write \(A^\top\) 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 \textit{not} including zero, and \(\mathbb{N}_{\geq 0}\)
to denote the natural numbers including zero.

We write := to denote \textit{is defined to be}. In lecture we may write \(\triangleq\) rather
than := 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 \(\mathcal{X}, \mathcal{Y}, \mathcal{Z}\), then we write \(f(\cdot, y)\) to
denote a function, \(g : \mathcal{X} \to \mathcal{Z}\), such that \(g(x) = f(x, y)\) for all \(x \in \mathcal{X}\).

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

The notation that we use is \textit{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.
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 \textit{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 \textit{plant}, the agent the \textit{controller}, the reward the (negative) \textit{cost}, the state the \textit{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 \textit{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 \textit{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.

\textbf{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?

\textbf{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

![687-Gridworld](image)

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

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

2.4 Describing the Agent and Environment Mathematically

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

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

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

- \( S \) is the set of all possible states of the environment. The state at time \( t \), \( S_t \), always takes values in \( S \). For now we will assume that \(|S| < \infty|\) — that the set of states is finite.

- \( A \) is the set of all possible actions the agent can take. The action at time \( t \), \( A_t \), always takes values in \( A \). For now we will assume that \(|A| < \infty|\).

- \( P \) is called the transition function, and it describes how the state of the environment changes.

\[
P : S \times A \times S \rightarrow [0, 1],
\]

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

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

Hereafter we suppress the sets when writing quantifiers (like \( \exists \) and \( \forall \)) — these should be clear from context. We say that the transition function is deterministic if \( P(s, a, s') \in \{0, 1\} \) for all \( s, a, \) and \( s' \). Recall that we will use lower-case letters to denote functions when possible — notice that \( P \) is an exception to this rule (due to historical usage and to avoid using \( p \), a commonly used symbol otherwise).
• $d_R$ describes how rewards are generated. Intuitively, it is a conditional distribution over $R_t$ given $S_t, A_t,$ and $S_{t+1}$. For now we assume that the rewards are bounded—that $|R_t| \leq R_{\text{max}}$ always, for all $t \in \mathbb{N} \geq 0$ and some constant $R_{\text{max}} \in \mathbb{R}$.\footnote{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.}

• $R$ 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

$$R : S \times A \rightarrow \mathbb{R},$$

and

$$R(s, a) := \mathbb{E}[R_t | S_t = s, A_t = a],$$

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

• $d_0$ is the initial state distribution:

$$d_0 : \mathcal{S} \rightarrow [0,1],$$

and for all $s$:

$$d_0(s) = \Pr(S_0 = s).$$

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

References