EE290S Lecture Note 1

Fall 2018

1 Administrative

1.1 Class Information

Instructors: Anant Sahai & Vidya Muthukumar
Office Hours: Immediately after lecture, Soda 306/Cory 258.
Course Communication: Primarily through Piazza.

1.2 Prerequisites

Prerequisites of the course are mastery of EE126 and knowledge of EE127 at the level of CS189. These are necessary prerequisites because the course material traditionally leans on much more advanced levels of probability and optimization knowledge. This course will be trying to make the material accessible to those who are meeting prerequisites. There may be struggles to do so — please provide feedback along the way.

1.3 Grading & Assignments

The course is not curved.

40% — Team Project
20% — Midterm (Solo, potentially take-home)
15% — Team HW (roughly every two weeks)
25% — Participation

1.3.1 Project

The format of the project is flexible and expectations will depend on the type of student — the work of a fourth year graduate student will be considered differently than that of an undergraduate student. The most basic project always available is a replication of an existing research paper. The final project includes a **brief presentations** and **poster**.

A **50-60 hour workload per person** is about the amount needed to produce a quality project. You are encouraged to work in groups — 3 people is optimal, and more than 4 is not recommended.

Proposals for the project will be due sometime in October. There will be suggestions for project topics, but hopefully by then you will have ideas growing from your work throughout the course (making homework/demos) or talking to instructors.

It is encouraged that you connect ideas from other courses. Your project may be "double dipped" with any other of your academic endeavors: other research projects/agendas, other course projects (with the agreement of the other professor/s), internship projects, and so on. Consult the instructors if coordination with other faculty is needed.

1.3.2 Midterm

The midterm will likely be a take-home exam on/around October 20. The window to take the exam would be short, so you would need to block out time to do it.

1.3.3 Bi-weekly Homework

Homeworks can be done in groups and are expected to take 5-10 hours per assignment. The release schedule for homework may be irregular.

1.3.4 Participation

While participation is only 25% of the grade, it **can result in you failing the course if minimum participation requirements are not met**. Participation is composed of creating scribe notes of a lecture and making a homework problem/demo, both of which will likely be done in groups.

Scribe notes will require about 15 hours of work per person. Beyond capturing the material presented in lecture, you will have to expand upon the material and incorporate information from other sources. It should be an open loop process in which you will work closely with the instructors to make revisions.

Homework/Demo creation will also require about 15 hours of work per person. You should create the homework/demo for the purpose of delving into course material. Both Coding and/or math problems are okay. Coding must be done using python and the homework must be formatted in latex. Detailed solutions are required.

Grading will not be very fine-grained for participation — putting in an effort to produce quality material will get you around an A-, but exceptional work can get you a higher grade. The assignments mainly there for you to have face-time with the instructors and also gain a deep understanding of at least one topic from the course.

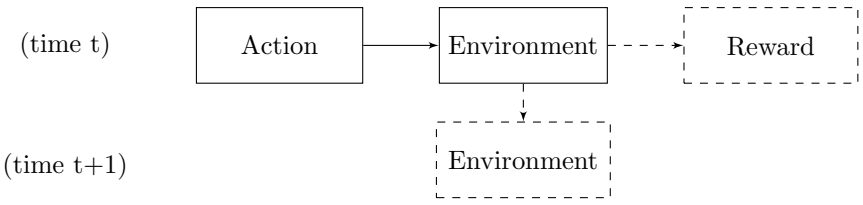
2 Course Overview

In this course, we will explore the challenges of sequential online decision making in non-deterministic, possibly adversarial, environments. We will start with a

simple version of the problem and, as the course progresses, build our way up to a high level of complexity while examining the new interactions and challenges of our changing models.

2.1 Reinforcement learning

A reinforcement learning problem is the modeled by the following situation:



An agent is playing a game over T rounds in some environment. At some round, an action is taken in a particular environment, or state. Based on the state, the action gives the agent some reward and also may change the state of the next round. The goal of the agent is to maximize cumulative reward over all rounds.

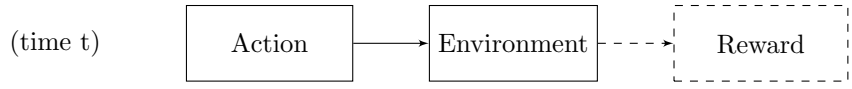
With reinforcement learning, there are several unknowns in the problem which make the problem difficult. We don't know the following:

1. the reward: taking a specific action at some state gives some reward which is not known beforehand.
2. the current environment, or state: we may not have full knowledge about the state we are in.
3. the transition/state dynamics: how the environment, or state, changes due to an action.

In order to maximize cumulative reward, we have to **learn how actions affect the state dynamics**, because future states affect future rewards.

2.2 Contextual Learning

Contextual learning, also contextual bandits, is a simplification of reinforcement learning, where taking an action does not affect state transition dynamics.



An agent is playing a game over T rounds. For each action, dependent on a given context and action taken, an unknown reward will be experienced.

Same as in reinforcement learning, the goal of the agent is to maximize their cumulative reward over all rounds.

In contextual learning, there is complexity with having a large context — you would need many rounds to learn the reward function. Because rounds are limited, there is an **estimation-approximation tradeoff**. You have to balance overfitting to the information you are able to undercover in the few rounds you experience versus having a simple model that has less power to represent a complex underlying pattern.

2.3 Multi-Armed Bandit

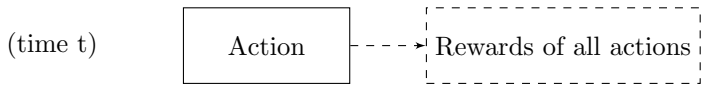
The multi-armed bandit problem is a simplification of contextual learning — the problem no longer depends on context, only the action.

An agent is playing a game over T rounds. Dependent on the action taken in some round, an unknown reward corresponding to the action taken will be experienced. The goal of the agent is still to maximize their cumulative reward over all rounds.

In the multi-armed bandit problem, the main challenge is to balance **exploiting versus exploring**. In any round, there is only one action-reward pair that is experienced — we miss out on information about the actions that are not taken. There will always be the option of taking the best known action or taking a unexplored action which may do better.

2.4 Prediction

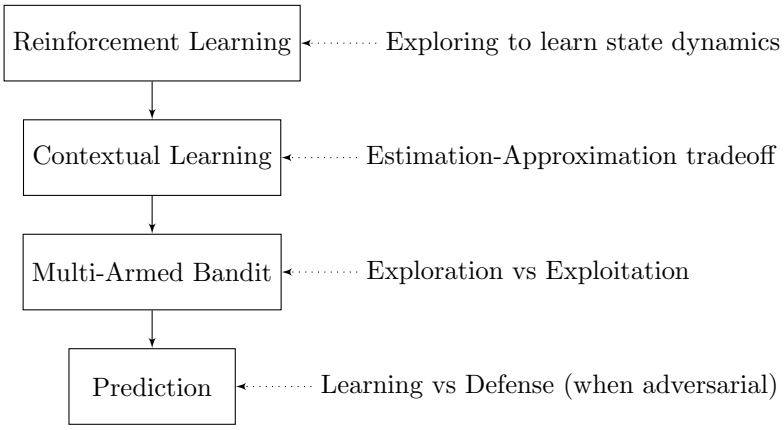
Allowing for the rewards of all actions, taken or not, to be observed for each round turns the multi-armed bandit problem into a prediction problem of the reward process. Again, the goal is to maximize cumulative reward over all rounds.



If the underlying process which determines the rewards is adversarial, the challenge will be balancing **learning** from previous reward patterns versus **defending** from the adversary.

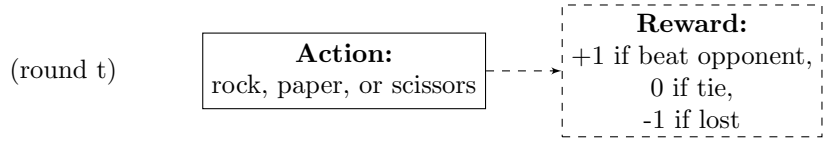
2.5 Summary and Examples

By breaking down the problem we are able to highlight the added difficulties as the complexity is increased.



2.5.1 Prediction Example:

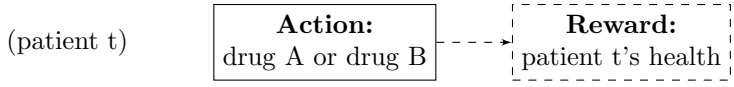
Imagine playing many rounds of rock-paper-scissors against an opponent.



At each turn, you experience only the reward of the action you took and know the reward of all the other actions. To **defend** against your opponent, you will want to be random enough that your opponent will not be able to guess your next move. But you also want to **learn** from any patterns or imperfect strategies of your adversary and capitalize on those.

2.5.2 Multi-Armed Bandit Example:

You are a doctor treating 100 patients with a specific disease. You have two drugs to test: drug A and drug B. You must figure out which experimental drug is most effective, while considering the health of each patients.

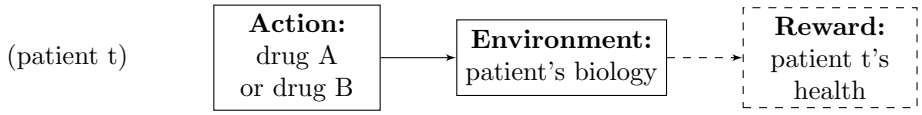


You cannot simply give 50 people drug A and 50 people drug B because what if one drug were ineffective? Ideally, we would adaptively sample our drugs and give very few patients the ineffective drug. In general, as you proceed with each

patient, you can use information from previous decisions and also learn more about the drug you administer. So now suppose that drug A does better than drug B for 3 of your first 5 patients. You have to choose between **exploiting** drug A's efficacy or **exploring** the possibility that drug B could be better.

2.5.3 Contextual Learning Example:

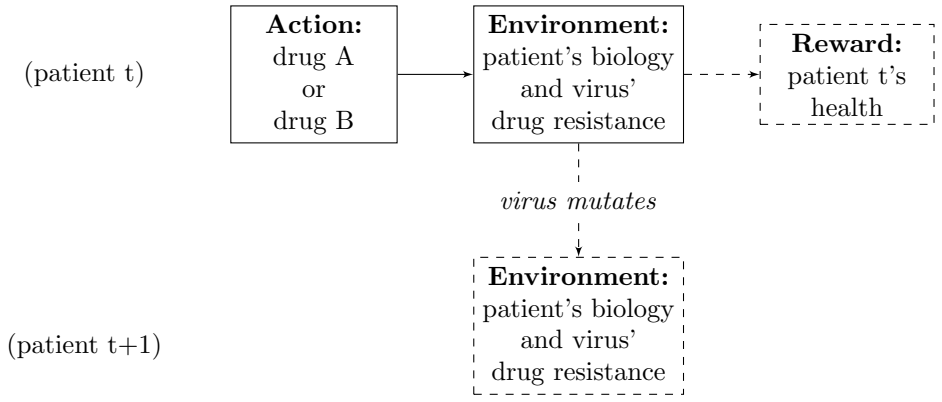
Consider the same situation as in the multi-armed bandit example. There is, however, an added complication. Each drug interacts differently for each patient.



There are so many different combinations of biological factors that could interact with the drugs differently — some factors will be more important than others (this should remind you of feature selection). You only have a small sample of those biological variations in your 100 patients so you must choose a model which balances being able to make good **estimations** outside of your sample with being able to **approximate** whatever information is present in a sample.

2.5.4 Reinforcement Learning Example:

Consider the same situation as in the contextual learning example. Again, there is an added complication. The virus can mutate in response to the drug.



Now you have to also consider how your actions will affect the future. For example, say that administering drug A when it does not work causes the virus to become increasingly resistant to both drug A and drug B. You would need to **explore how the environment is changing in response to your actions** to discover this. And once you do, it would probably make you cautious of using drug A even if it is effective for many people.

3 Machine Learning Highlights

In general, Machine Learning consists of feeding data into an algorithm and extracting some kind of "pattern" which should generalize to do well for unseen data.

In the case of **supervised learning**, we are given labelled data (x_i, y_i), where the x_i are inputs and the y_i are labels. The pattern learned from the data will then predict labels $\hat{y}_i$ given input x_i.

We typically discuss two flavors of supervised learning: regression and classification.

- In a **regression** setting, the labels y_i are continuous values. In regression, we typically define correctness as closeness. $\hat{y}_i$ is correct if it is "close to" y_i, i.e. if for some ϵ : $y_i - \epsilon \leq \hat{y}_i \leq y_i + \epsilon$. An example of a regression problem would be predicting the market value of homes given their characteristics, such as number of bedrooms, bathrooms, etc.
- In a **classification** setting, the labels are discrete values, and classification is correct when the correct label is exactly chosen. An example of a classification task would be determining whether a given picture contains a cat in it or not.

In general, how do Machine Learning algorithms learn?

1. Split data into a training set X_{train} and a validation set X_{val}.
2. Use X_{train} to fit the parameters of a model, using some "intelligent" optimization (such as least squares or gradient descent).
3. Use X_{val} to estimate generalization performance.
4. Repeat the above to set hyper-parameters in a way that maximizes generalization performance. This can be done in brute-force fashion.

This raises the question: what distinguishes parameters from hyper-parameters?

The difference is usually subtle. It boils down to what factors, if left as parameters to be optimized, lead to bad performance due to extreme overfitting. For example, when trying to fit a polynomial onto a set of points from a distribution, if the degree of the polynomial were a parameter, any set of N points could always be perfectly fit with an $N-1$ degree polynomial. However, this would most likely not generalize as well as a lower degree polynomial to unseen points from the distribution which the N points came from.