CME 241: Foundations of Reinforcement Learning with Applications in Finance

Ashwin Rao

ICME, Stanford University

Winter 2025

Meet your Instructor

- Joined Stanford ICME as Adjunct Professor in Fall 2018
- Research Interests: A.I. for Dynamic Decisioning under Uncertainty
- Leading ICME's *Mathematical and Computational Finance* program
- Technical mentor to ICME students
- Teaching faculty at *HAI Institute*
- Educational background: Algorithms Theory & Abstract Algebra
- 10 years at Goldman Sachs (NY) *Rates/Mortgage Derivatives* Trading
- 4 years at Morgan Stanley as *Managing Director - Market Modeling*
- 6 years at Target, leading it's AI/Data Science capabilities
- Currently Chief AI Officer at QXO (distribution of building materials)
- Teaching experience: Pure & Applied Math, CompSci, Finance, Mgmt

Requirements and Setup

- Prerequisites:
  - Undergraduate-level background in Applied Mathematics (Multivariate Analysis, Linear Algebra, Probability, Optimization)
  - Background in data structures/algorithms, fluency with numpy
  - Basic familiarity with Pricing, Portfolio Mgmt and Algo Trading, but we will do an overview of the requisite Finance/Economics
  - No background required in MDP, DP, RL (we will cover from scratch)
- Here's a sample exam to get a sense of course difficulty
- Register for the course on Ed Discussion
- Install Python 3 and supporting IDE/tools (eg: VS Code, Jupyter)
- Install LaTeX/Markdown and supporting editor for tech writing
- Get the course textbook or download the PDF version
- Assignments and code in the textbook based on this open-source code
- *Fork* this repo and get set up to use this code in assignments (Assignment 0)
- Create separate directories for each assignment for CA Neel Narayan to review - send Neel your forked repo URL and *git push* by due dates

Housekeeping

- Lectures: Wed & Fri 4:30pm-5:50pm in 160-124.
- Office Hours:
    1. Ashwin: 2:30pm-4:00pm Fri (or by appointment) in ICME Mezzanine, room M05 (within Huang Engg Bldg)
    2. Neel: poll will be sent out on Ed (or by appointment)
- Course Web Site: cme241.stanford.edu
- Ask Questions and engage in Discussions on Ed Discussion
- Note: This course is listed as CME 241 as well as MS&E 346
- Any accommodations: Please reach out to me or Neel as soon as possible

Resources and Success Tips

- Course based on my [RL For Finance book](#)
- Slides available for each lecture ("guided tour" of respective chapter)
- Code in my book based on [this open-source code](#)
- Reading this code as important as reading the theory
- All resources organized on the [course web site](#) ("source of truth")
- Today: Overview of course topics and getting a general flavor
- How to succeed in this course:
  - **Attend lectures**
  - Lectures' **Guided Tours** are more efficient for learning (than book)
  - **Ask questions** during class, on Ed, and during office hours
  - Learn by **writing** (code and math)
  - Learn by **observing code design** of modules in book's git repo
  - Investments in **first few weeks** vital (foundations for 2nd half)

Grading

- Breakdown:
  - 45% *Group Assignments* (Mix of Math and Programming)
  - 30% *Exam* (Theory, Modeling, and Programming)
  - 20% *Project* (Idea Creativity, Proof-of-Concept, Presentation)
  - 5% *Attendance* (18/20 Minimum Lecture Attendance)
- Assignments
  - 3 assignments total, each worth 15% and graded based on accuracy
  - Can be completed in groups of up to 3 (single repository)
  - 3 free late days, -10% for each late day thereafter
- Exam
  - Individual take-home exam in week 8, covering topics from weeks 1-7
  - 48 hour window (tentatively 8PM 2/28 to 8 PM 3/2)
  - Practice exams from prior years available
- Project
  - Open-ended project, groups of up to 3
  - Choose topics of interest, a list of ideas will be released by Week 5
  - Deliverables will be a proposal, presentation, and code

# Stanford Honor Code - Assignments versus Exams

- *Assignments*: You can discuss approaches among your group members, not with other students outside of your group
  - Each group member should be responsible for a subset of questions (indicate which question you do on the assignment)
  - Writing answers/code should be your own (don't copy/paste)
  - You can invoke the core modules we have written (as instructed), and can use LLMS (ChatGPT for example)
- *Exams*: You cannot engage in any conversation with other students
  - Write to the CA if a question is unclear
  - Exams are graded on correctness and completeness
  - Open-internet Exams: Search for concepts, not answers to exam questions
  - If you accidentally run into a strong hint/answer, state it honestly

A.I. for Dynamic Decisioning under Uncertainty

- Let's browse some terms used to characterize this branch of A.I.
- *Stochastic*: Uncertainty in key quantities, evolving over time
- *Optimization*: A well-defined metric to be maximized ("The Goal")
- *Dynamic*: Decisions need to be a function of the changing situations
- *Control*: Overpower uncertainty by persistent steering towards goal
- Jargon overload due to confluence of Control Theory, O.R. and A.I.
- For language clarity, let's just refer to this area as *Stochastic Control*
- The core framework is called *Markov Decision Processes* (MDP)
- *Reinforcement Learning* is a class of algorithms to solve MDPs

The MDP Framework
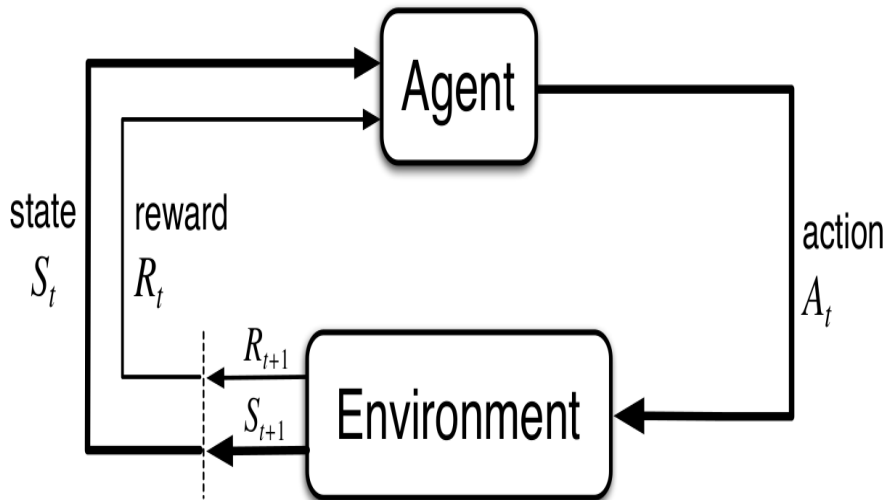
Components of the MDP Framework

- The *Agent* and the *Environment* interact in a time-sequenced loop
- *Agent* responds to [*State*, *Reward*] by taking an *Action*
- *Environment* responds by producing next step's (random) *State*
- *Environment* also produces a (random) scalar denoted as *Reward*
- Each *State* is assumed to have the *Markov Property*, meaning:
    - Next *State/Reward* depends only on Current *State* (for a given *Action*)
    - Current *State* captures all relevant information from *History*
    - Current *State* is a sufficient statistic of the future (for a given *Action*)
- Goal of *Agent* is to maximize *Expected Sum* of all future *Reward*s
- By controlling the (*Policy* : *State* → *Action*) function
- This is a dynamic (time-sequenced control) system under uncertainty

Formal MDP Framework

The following notation is for discrete time steps. Continuous-time formulation is analogous (often involving Stochastic Calculus)

- Time steps denoted as $t = 1, 2, 3, \ldots$
- Markov States $S_t \in \mathcal{S}$ where $\mathcal{S}$ is the State Space
- Actions $A_t \in \mathcal{A}$ where $\mathcal{A}$ is the Action Space
- Rewards $R_t \in \mathbb{R}$ denoting numerical feedback
- Transitions $p(r, s'|s, a) = \mathbb{P}[(R_{t+1} = r, S_{t+1} = s')|S_t = s, A_t = a]$
- $\gamma \in [0, 1]$ is the Discount Factor for Reward when defining *Return*
- Return $G_t = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \ldots$
- Policy $\pi(a|s)$ is probability that Agent takes action a in states s
- The goal is find a policy that maximizes $\mathbb{E}[G_t|S_t = s]$ for all $s \in \mathcal{S}$

How a baby learns to walk
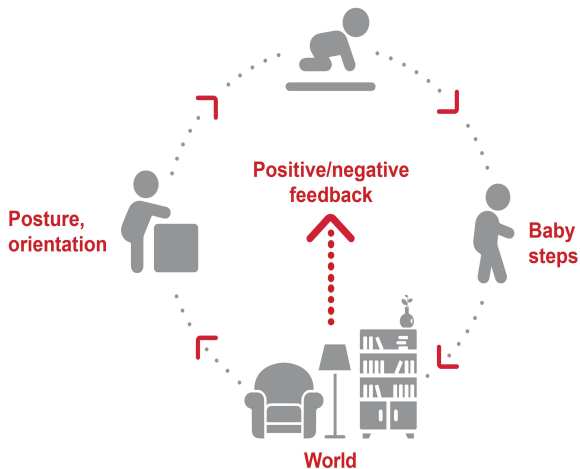
Many real-world problems fit this MDP framework

- Self-driving vehicle (speed/steering to optimize safety/time)
- Game of Chess (Boolean *Reward* at end of game)
- Complex Logistical Operations (eg: movements in a Warehouse)
- Make a humanoid robot walk/run on difficult terrains
- Manage an investment portfolio
- Control a power station
- Optimal decisions during a football game
- Strategy to win an election (high-complexity MDP)

Self-Driving Vehicle

Ride quality, arrival time

Location, velocity, sensor data

Steering, acceleration, braking

Traffic, signals, road conditions, routes

Why are these problems hard?

- *State* space can be large or complex (involving many variables)
- Sometimes, *Action* space is also large or complex
- No direct feedback on "correct" *Actions* (only feedback is *Reward*)
- Time-sequenced complexity (*Actions* influence future *States/Actions*)
- *Action*s can have delayed consequences (late *Reward*s)
- *Agent* often doesn't know the *Model* of the *Environment*
- "Model" refers to probabilities of state-transitions and rewards
- So, *Agent* has to learn the *Model* AND solve for the Optimal *Policy*
- *Agent Action*s need to tradeoff between "explore" and "exploit"

Value Function and Bellman Equations

- Value function (under policy π) $V^\pi(s) = \mathbb{E}[G_t | S_t = s]$ for all $s \in \mathcal{S}$

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(r,s'|s,a) \cdot (r + \gamma V^\pi(s')) \text{ for all } s \in \mathcal{S}$$

- Optimal Value Function $V^*(s) = \max_\pi V^\pi(s)$ for all $s \in \mathcal{S}$

$$V^*(s) = \max_a \sum_{r,s'} p(r,s'|s,a) \cdot (r + \gamma V^*(s')) \text{ for all } s \in \mathcal{S}$$

- *There exists an Optimal Policy π^* achieving $V^*(s)$ for all $s \in \mathcal{S}$*
- Determining $V^\pi(s)$ known as *Prediction*, and $V^*(s)$ known as *Control*
- The above recursive equations are called *Bellman equations*
- In continuous time, refered to as *Hamilton-Jacobi-Bellman (HJB)*
- The algorithms based on Bellman equations are broadly classified as:
    - Dynamic Programming
    - Reinforcement Learning

Dynamic Programming

- When Probabilities Model is known $\Rightarrow$ *Dynamic Programming* (DP)
- DP Algorithms take advantage of knowledge of probabilities
- So, DP Algorithms do not require interaction with the environment
- In the Language of AI, DP is a type of *Planning Algorithm*
- DP algorithms are iterative algorithms based on Fixed-Point Theorem
- Finding a *Fixed Point* of Operator based on Bellman Equation
- Why is DP not effective in practice?
  - Curse of Dimensionality
  - Curse of Modeling
- Curse of Dimensionality can be partially cured with Approximate DP
- To resolve both curses effectively, we need RL

Reinforcement Learning

- Typically in real-world, we don't have access to a Probabilities Model
- All we have is access to an environment serving individual transitions
- Even if MDP model is available, model updates can be challenging
- Often real-world models end up being too large or too complex
- Sometimes estimating a *sampling model* is much more feasible
- RL interacts with either *actual* or *simulated* environment
- Either way, we receive *individual transitions* to next state and reward
- RL is a "trial-and-error" approach linking *Actions* to *Returns*
- Try different actions & learn what works, what doesn't
- This is hard because actions have overlapping reward sequences
- Also, sometimes Actions result in *delayed Rewards*

RL: Learning Value Function Approximation from Samples

- RL incrementally learns the Value Function from transitions data
- Appropriate Approximation of Value Function is key to success
- Deep Neural Networks are typically used for function approximation
- Big Picture: Sampling and Function Approximation come together
- RL algorithms are clever about balancing "explore" versus "exploit"
- Most RL Algorithms are founded on the Bellman Equations
- **Promise of modern A.I. is based on success of RL algorithms**
- Potential for automated decision-making in many industries
- RL Agents: Bots that act or behave more optimal than humans
- RL already solves various low-complexity real-world problems
- RL might soon be the most-desired skill in the technical job-market
- Possibilities in Finance are endless (we cover 5 important problems)
- Studying RL is a lot of fun! (interesting in theory as well as coding)
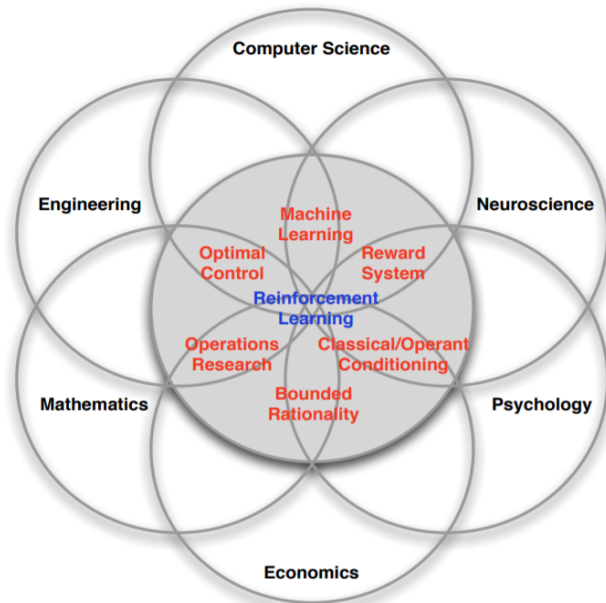
Vague (but in-vogue) Classification of Machine Learning
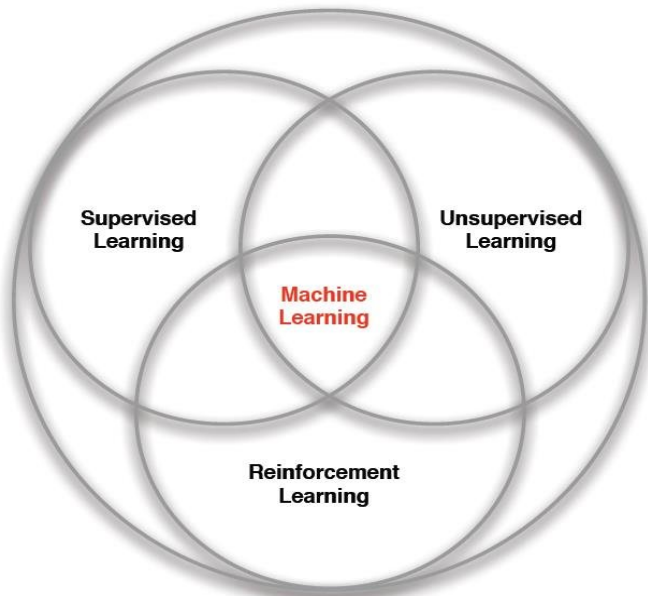
Overview of the Course

- Theory of Markov Decision Processes (MDPs)
- Dynamic Programming (DP) Algorithms
- Approximate DP and Backward Induction Algorithms
- Reinforcement Learning (RL) Algorithms
- Plenty of Python implementations of models and algorithms
- Apply these algorithms to 5 Financial/Trading problems:
  - (Dynamic) Asset-Allocation to maximize Utility of Consumption
  - Pricing and Hedging of Derivatives in an Incomplete Market
  - Optimal Exercise/Stopping of Path-dependent American Options
  - Optimal Trade Order Execution (managing Price Impact)
  - Optimal Market-Making (Bids and Asks managing Inventory Risk)
- By treating each of the problems as MDPs (i.e., Stochastic Control)
- We will go over classical/analytical solutions to these problems
- Then introduce real-world considerations, and tackle with RL (or DP)
- Course blends Theory/Math, Algorithms/Coding, Real-World Finance

Optimal Asset Allocation to Maximize Consumption Utility

- You can invest in (allocate wealth to) a collection of assets
- Investment horizon is a fixed length of time
- Each risky asset characterized by a probability distribution of returns
- Periodically, you can re-allocate your wealth across the various assets
- Transaction Costs & Constraints on trading hours/quantities/shorting
- Allowed to consume a fraction of your wealth at specific times
- Dynamic Decision: Time-Sequenced Allocation & Consumption
- To maximize horizon-aggregated *Risk-Adjusted Consumption*
- *Risk-Adjustment* involves a study of *Utility Theory*

MDP for Optimal Asset Allocation problem

- *State* is [Current Time, Current Holdings, Current Prices]
- *Action* is [Allocation Quantities, Consumption Quantity]
- *Action*s limited by various real-world trading constraints
- *Reward* is Utility of Consumption less Transaction Costs
- *State*-transitions governed by risky asset movements

Derivatives Pricing and Hedging in an Incomplete Market

- Classical Pricing/Hedging Theory assumes "frictionless market"
- Technically, refered to as arbitrage-free and complete market
- *Complete market* means derivatives can be perfectly replicated
- But real world has transaction costs and trading constraints
- So real markets are incomplete where classical theory doesn't fit
- How to price and hedge in an *Incomplete Market*?
- Maximize "risk-adjusted-return" of the derivative plus hedges
- Similar to Asset Allocation, this is a stochastic control problem
- Deep Reinforcement Learning helps solve when framed as an MDP

MDP for Pricing/Hedging in an Incomplete Market

- *State* is [Current Time, PnL, Hedge Qtys, Hedge Prices]
- *Action* is Units of Hedges to be traded at each time step
- *Reward* only at termination, equal to Utility of terminal PnL
- *State*-transitions governed by evolution of hedge prices
- Optimal Policy $\Rightarrow$ Derivative Hedging Strategy
- Optimal Value Function $\Rightarrow$ Derivative Price

Optimal Exercise of Path-dependent American Options

- An American option can be exercised anytime before option maturity
- Key decision at any time is to exercise or continue
- The default algorithm is Backward Induction on a tree/grid
- But it doesn't work for American options with complex payoffs
- Also, it's not feasible when state dimension is large
- Industry-Standard: Longstaff-Schwartz's simulation-based algorithm
- RL is an attractive alternative to Longstaff-Schwartz
- RL is straightforward once Optimal Exercise is modeled as an MDP

MDP for Optimal American Options Exercise

- *State* is [Current Time, History of Underlying Security Prices]
- *Action* is Boolean: Exercise (i.e., Payoff and Stop) or Continue
- *Reward* always 0, except upon Exercise ($=$ Payoff)
- *State*-transitions governed by Underlying Prices' Stochastic Process
- Optimal Policy $\Rightarrow$ Optimal Stopping $\Rightarrow$ Option Price
- Can be generalized to other Optimal Stopping problems

Optimal Trade Order Execution (controlling Price Impact)

- You are tasked with selling a large qty of a (relatively less-liquid) stock
- You have a fixed horizon over which to complete the sale
- Goal is to maximize aggregate sales proceeds over horizon
- If you sell too fast, *Price Impact* will result in poor sales proceeds
- If you sell too slow, you risk running out of time
- We need to model temporary and permanent *Price Impact*s
- Objective should incorporate penalty for variance of sales proceeds
- Again, this amounts to maximizing Utility of sales proceeds

MDP for Optimal Trade Order Execution

- *State* is [Time Remaining, Stock Remaining to be Sold, Market Info]
- *Action* is Quantity of Stock to Sell at current time
- *Reward* is Utility of Sales Proceeds (i.e., Variance-adjusted-Proceeds)
- *Reward* & *State*-transitions governed by *Price Impact Model*
- Real-world *Model* can be quite complex (*Order Book Dynamics*)

Optimal Market-Making (controlling Inventory Buildup)

- Market-maker's job is to submit bid and ask prices (and sizes)
- On the Trading *Order Book* (which moves due to other players)
- Market-maker needs to adjust bid/ask prices/sizes appropriately
- By anticipating the *Order Book Dynamics*
- Goal is to maximize *Utility of Gains* at the end of a suitable horizon
- If Buy/Sell LOs are too narrow, more frequent but small gains
- If Buy/Sell LOs are too wide, less frequent but large gains
- Market-maker also needs to manage potential unfavorable inventory (long or short) buildup and consequent unfavorable liquidation
- This is a classical stochastic control problem

MDP for Optimal Market-Making

- *State* is [Current Time, Mid-Price, PnL, Inventory of Stock Held]
- *Action* is Bid & Ask Prices & Sizes at each time step
- *Reward* is Utility of Gains at termination
- *State*-transitions governed by probabilities of hitting/lifting Bid/Ask
- Also governed by Order Book Dynamics (can be quite complex)

Other RL Courses at Stanford

- CS 234 (Emma Brunskill)
- CS 332 (Emma Brunskill)
- CS 224R (Chelsea Finn & Karol Hausman)
- CS 422 (Nick Haber)
- AA 203 (Marco Pavone)
- AA 228/CS 238 (Mykel Kochenderfer)
- MS&E 338 (Ben Van Roy)
- MS&E 237/EE 277 (Ben Van Roy)

Salient/Distinguishing features of this Course

- Emphasis on Foundations and Core Concepts
- More about *why* and *how*, versus *what*
- Balance between mathematical precision and intuitive understanding
- Coding from scratch, avoiding standard packages
- Encourages *Creator/Builder* mindset, versus *User* mindset
- Emphasis on code design driven by mathematical concepts/structures
- Key purpose of coding: Enables long-term retention of key learnings
- Several financial trading applications (and a couple from Retail)
- Coverage of continuous-time versions (elegant, analytical)
- I will dispel some common myths about industry versus academia