CME 241: Reinforcement Learning for Stochastic Control Problems in Finance

Ashwin Rao

ICME, Stanford University

Winter 2021

Meet your Instructor

- Joined Stanford ICME as Adjunct Faculty in Fall 2018
- Research Interests: A.I. for Dynamic Decisioning under Uncertainty
- Technical mentor to ICME students, partnerships with industry
- 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*
- Founded Tech Startup *ZLemma*, Acquired by hired.com in 2015
- One of our products was algorithmic jobs/career guidance for students
- Teaching experience: Pure & Applied Math, CompSci, Finance, Mgmt
- Current Industry Job: V.P. of A.I. at Target (the Retail company)

Requirements and Setup

- Pre-requisites:
  - 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 last year's final exam to get a sense of course difficulty
- Register for the course on Piazza
- Install Python 3 and supporting IDE/tools (eg: PyCharm, Jupyter)
- Install LaTeX/Markdown and supporting editor for tech writing
- Assignments and code in my book based on this open-source code
- *Fork* this repo and get set up to use this code in assignments
- Create separate directories for each assignment for CA (Sven Lerner) to review - send Sven your forked repo URL and *git push* often

Housekeeping

- Grade based on:
  - 30% 48-hour Mid-Term Exam (on Theory, Modeling, Programming)
  - 40% 48-hour Final Exam (on Theory, Modeling, Programming)
  - 30% *Assignments*: Technical Writing and Programming
- Lectures (on Zoom): Wed & Fri 4:00pm - 5:20pm, Jan 13 - Mar 19
- Office Hours 1-4pm Fri (or by appointment) on Zoom
- Course Web Site: cme241.stanford.edu
- Ask Questions and engage in Discussions on Piazza
- My e-mail: ashwin.rao@stanford.edu

Purpose and Grading of *Assignments*

- Assignments shouldn't be treated as "tests" with right/wrong answer
- Rather, they should be treated as part of your learning experience
- You will *truly* understand ideas/models/algorithms only when you *write down* the Mathematics and the Code precisely
- Simply reading Math/Code gives you a false sense of understanding
- Take the initiative to make up your own assignments
- Especially on topics you feel you don't quite understand
- Individual assignments won't get a grade and there are no due dates
- The CA will review once every 2 weeks and provide feedback
- It will be graded less on correctness and completeness, and more on:
  - Coding and Technical Writing style that is clear and modular
  - Demonstration of curiosity and commitment to learning through the overall body of assignments work
  - Engagement in asking questions and seeking feedback for improvements

Resources

- Course based on the (incomplete) book I am currently writing
- Supplementary/Optional reading: Sutton-Barto's RL book
- I prepare slides for each lecture ("guided tour" of respective chapter)
- A couple of lecture slides are from David Silver's RL course
- Code in my book based on this open-source code
- Reading this code as important as the reading of the theory
- We will go over some classical papers on the Finance applications
- Some supplementary/optional papers from Finance/RL
- All resources organized on the course web site ("source of truth")

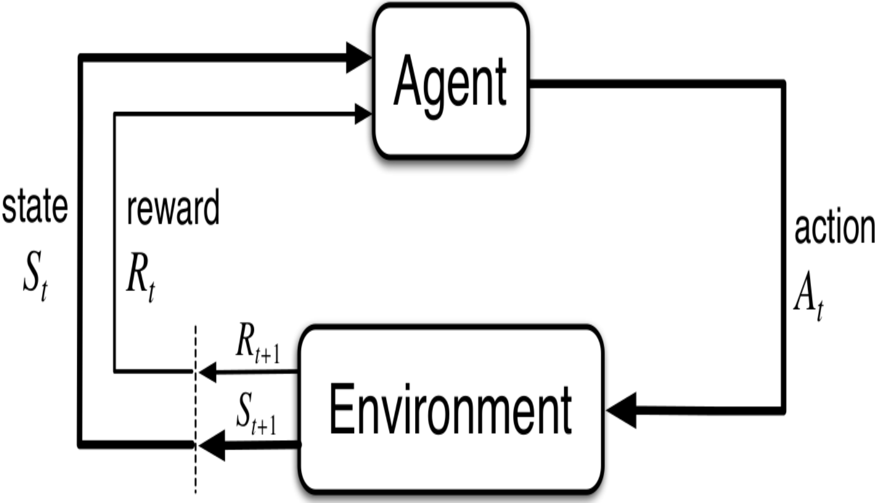
Stanford Honor Code - For Assignments versus Exams

- *Assignments*: You can discuss solution approaches with other students
- Because assignments are graded more for effort than correctness
- Writing (answers/code) should be your own (don't copy/paste)
- You can invoke the core modules I have written (as instructed)
- *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
- So *don't ask for help* on how to solve exam questions
- Open-internet Exams: Search for concepts, not answers to exam Qs
- 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

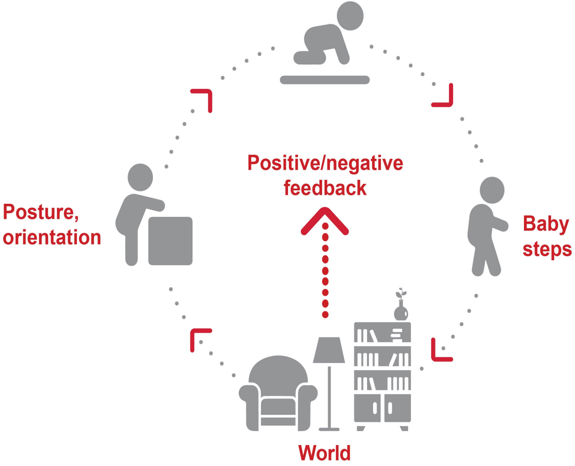
- The *Agent* and the *Environment* interact in a time-sequenced loop
- *Agent* responds to [*State*, *Reward*] by taking an *Action*
- *Environment* responds by producing next step's (random) *State*
- *Environment* also produces a (random) scalar denoted as *Reward*
- Each *State* is assumed to have 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 ($Policy : State \rightarrow Action$) 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 $p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$
- $\gamma \in [0, 1]$ 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 $s \in \mathcal{S}$

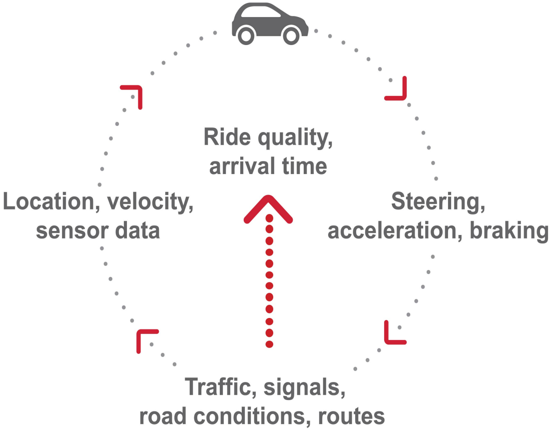
How a baby learns to walk



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



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 $s \in \mathcal{S}$

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

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

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

- *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

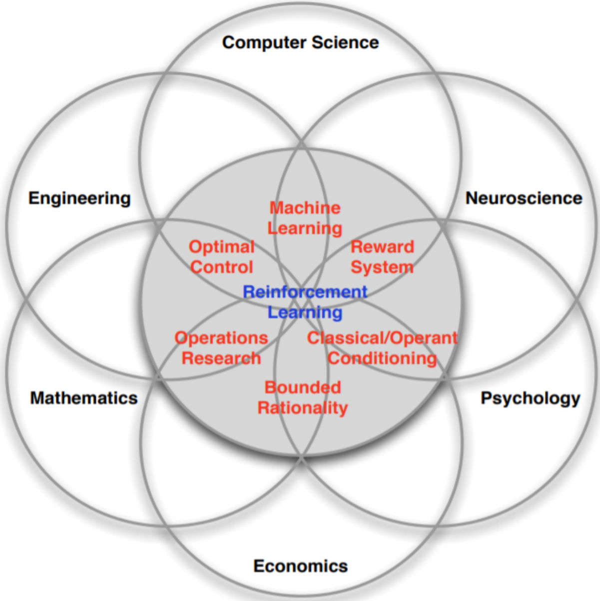
Dynamic Programming versus Reinforcement Learning

- 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 A.I, DP is a type of *Planning Algorithm*
- When Probabilities Model unknown $\Rightarrow$ *Reinforcement Learning* (RL)
- RL Algorithms interact with the Environment and incrementally learn
- Environment interaction could be *real* or *simulated* interaction
- RL approach: Try different actions & learn what works, what doesn't
- RL Algorithms' key challenge is to tradeoff "explore" versus "exploit"
- DP or RL, Good approximation of Value Function is vital to success
- Deep Neural Networks are typically used for function approximation

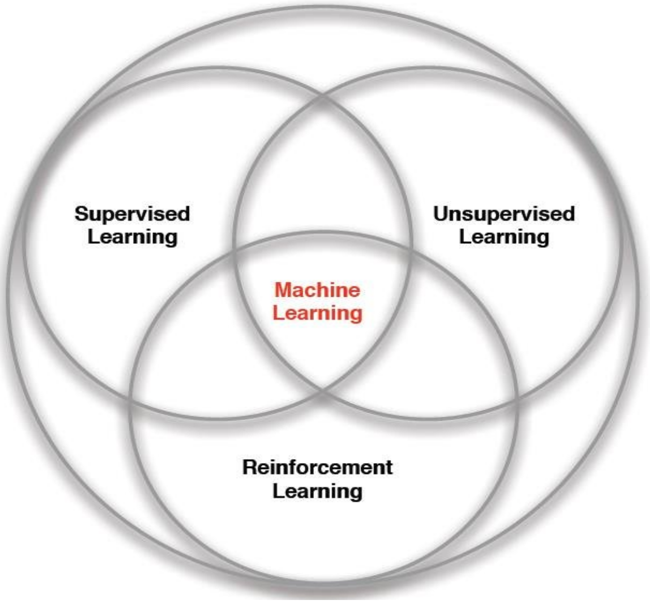
Why is RL interesting/useful to learn about?

- RL solves MDP problem when *Environment Probabilities* are unknown
- This is typical in real-world problems (complex/unknown probabilities)
- RL interacts with *Actual Environment* or with *Simulated Environment*
- **Promise of modern A.I. is based on success of RL algorithms**
- Potential for automated decision-making in many industries
- In 10-20 years: 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)
- Learning RL is a lot of fun! (interesting in theory as well as coding)

Many Faces of Reinforcement Learning



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 are re-allocate your wealth to 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 Utility of Consumption
- Utility function represents degree of risk-aversion
- So, we effectively maximize aggregate Risk-Adjusted Consumption

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
- In an incomplete market, we need to "choose" a risk-neutral measure
- This amounts to specifying a Utility function
- Maximizing "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 path-dependent options
- 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
- Which is equivalent to maximizing aggregate 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 (Limit Order Book Dynamics)

Optimal Market-Making (controlling Inventory Buildup)

- Market-maker's job is to submit bid and ask prices (and sizes)
- On the Limit Order Book (which moves due to other players)
- Market-maker needs to adjust bid/ask prizes/sizes appropriately
- By anticipating the Limit 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 Limit Order Book Dynamics (can be quite complex)

Week by Week (Tentative) Schedule

- W1: Markov Decision Processes
- W2: Bellman Equations & Dynamic Programming Algorithms
- W3: Backward Induction and Approximate DP Algorithms
- W4: Optimal Asset Allocation & Derivatives Pricing/Hedging
- W5: Options Exercise, Order Execution, Market-Making
- Mid-Term Exam
- W6: RL For Prediction (MC, TD, TD(λ))
- W7: RL for Control (SARSA, Q-Learning)
- W8: Batch Methods (DQN, LSTD/LSPI) and Gradient TD
- W9: Policy Gradient and Actor-Critic Algorithms
- W10: Model-based RL and Explore v/s Exploit
- Final Exam

Getting a sense of the style and content of the lectures

A sampling of lectures to browse through and get a sense ...

- Understanding Risk-Aversion through Utility Theory
- HJB Equation and Merton's Portfolio Problem
- Derivatives Pricing and Hedging with Deep Reinforcement Learning
- Stochastic Control for Optimal Market-Making
- Policy Gradient Theorem and Compatible Approximation Theorem
- Value Function Geometry and Gradient TD
- Adapative Multistage Sampling Algorithm (Origins of MCTS)

Some Landmark Papers we cover in this course

- Merton's solution for Optimal Portfolio Allocation/Consumption
- Longstaff-Schwartz Algorithm for Pricing American Options
- Almgren-Chriss paper on Optimal Order Execution
- Avellaneda-Stoikov paper on Optimal Market-Making
- Original DQN paper and Nature DQN paper
- Lagoudakis-Parr paper on Least Squares Policy Iteration
- Sutton, McAllester, Singh, Mansour's Policy Gradient Theorem
- Chang, Fu, Hu, Marcus' AMS origins of Monte Carlo Tree Search

Similar Courses offered at Stanford

- AA 228/CS 238 (Mykel Kochenderfer)
- CS 234 (Emma Brunskill)
- CS 332 (Emma Brunskill)
- MS&E 338 (Ben Van Roy)
- EE 277 (Ben Van Roy)
- MS&E 251 (Edison Tse)
- MS&E 348 (Gerd Infanger)
- MS&E 351 (Ben Van Roy)
- MS&E 339 (Ben Van Roy)