Topics in Reinforcement Learning: Rollout and Approximate Policy Iteration

ASU, CSE 691, Spring 2021

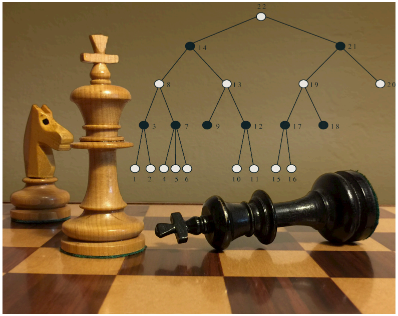
Links to Class Notes, Videolectures, and Slides at
http://web.mit.edu/dimitrib/www/RLbook.html

Dimitri P. Bertsekas
dbertsek@asu.edu

Lecture 1
Course Overview

Outline

AlphaGo (2016) and AlphaZero (2017)



AlphaZero

Plays much better than all chess programs

Plays different!

Learned from scratch ... with 4 hours of training!

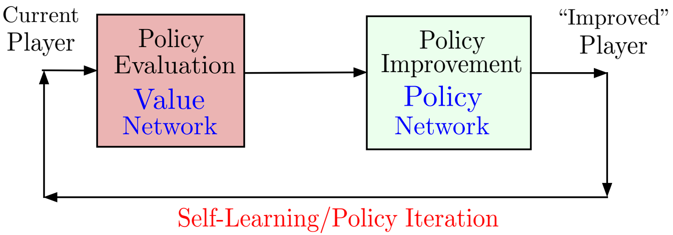
Same algorithm learned multiple games (Go, Shogi)

AlphaZero is not just playing better, it has discovered a new way to play!

The methodology is based on the principal ideas of this course:

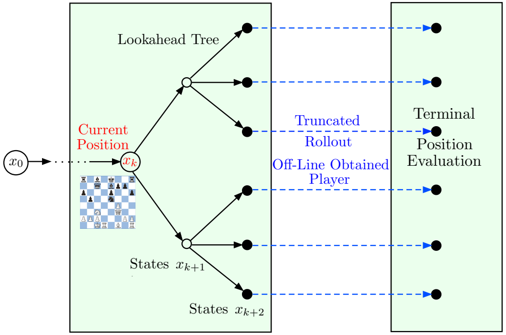
- Off-line training/policy iteration - Self learning
- Approximations with value and policy NN approximations
- Parallel computation
- On-line play by multistep lookahead and cost function approximations

AlphaZero Off-Line Training by Policy Iteration Using Self-Generated Data



- The "current" player plays games that are used to "train" an "improved" player
- At a given position, the "value" of a position and the "move probabilities" (the player) are approximated by a deep neural nets (NN)
- Successive NNs are trained using self-generated data and a form of regression
- A form of randomized policy improvement is used: Monte-Carlo Tree Search (MCTS)
- AlphaZero bears similarity to earlier works, e.g., TD-Gammon (Tesauro,1992), but is more complicated because of the MCTS and the deep NN
- The success of AlphaZero is due to a skillful implementation/integration of known ideas, and awesome computational power

AlphaZero/AlphaGo On-Line Play by Approximation in Value Space: Multistep Lookahead, Rollout, and Cost Approximation



- Off-line training yields a “value network" and a “policy network" that provide a position evaluation function and a default/base policy to play
- On-line play improves the default/base policy by:
  - Searching forward for several moves
  - Simulating the base policy for some more moves
  - Approximating the effect of future moves by using the terminal position evaluation

Backgammon programs by Tesauro (mid 1990s) use a similar methodology

Our Objectives in this Course

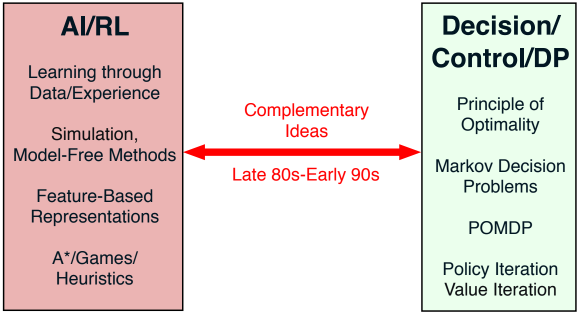
Develop the AlphaZero/AlphaGo methodology, but more generally for

- (Approximate) Dynamic Programming, including Operations Research/planning type problems
- (Approximately) Optimal Control, including Model Predictive Control and Robotics
- Multiagent versions of Dynamic Programming and Optimal Control
- Challenging/intractable discrete and combinatorial optimization problems
- Decision and control in a changing environment (adaptive control).

We will:

- Discuss off-line training methods of approximate policy iteration and Q-learning to construct “value networks" and “policy networks"
- Discuss on-line control methods based on approximation in value space, one-step or multistep lookahead, rollout, etc
- Develop the underlying theory relying on intuition and some math analysis
- Develop the methodology in a unified framework that allows dealing with deterministic and stochastic, discrete and continuous problems

Evolution of Approximate DP/RL



Historical highlights

- Exact DP, optimal control (Bellman, Shannon, and others 1950s ...)
- AI/RL and Decision/Control/DP ideas meet (late 80s-early 90s)
- First major successes: Backgammon programs (Tesauro, 1992, 1996)
- Algorithmic progress, analysis, applications, first books (mid 90s ...)
- Machine Learning, BIG Data, Robotics, Deep Neural Networks (mid 2000s ...)
- AlphaGo and AlphaZero (DeepMind, 2016, 2017)

Approximate DP/RL Methodology is now Ambitious and Universal

Exact DP applies (in principle) to a very broad range of optimization problems

- Deterministic <—-> Stochastic
- Combinatorial optimization <—-> Optimal control w/ infinite state/control spaces
- One decision maker <—-> Two player games
- ... BUT is plagued by the curse of dimensionality and need for a math model

Approximate DP/RL overcomes the difficulties of exact DP by:

- Approximation (use neural nets and other architectures to reduce dimension)
- Simulation (use a computer model in place of a math model)

State of the art:

- Broadly applicable methodology: Can address a very broad range of challenging problems. Deterministic-stochastic-dynamic, discrete-continuous, games, etc
- There are no methods that are guaranteed to work for all or even most problems
- There are enough methods to try with a reasonable chance of success for most types of optimization problems
- Role of the theory: Guide the art, delineate the sound ideas

A Relevant Quotation from 25 Years Ago

From preface of Neuro-Dynamic Programming, Bertsekas and Tsitsiklis, 1996

A few years ago our curiosity was aroused by reports on new methods in reinforcement learning, a field that was developed primarily within the artificial intelligence community, starting a few decades ago. These methods were aiming to provide effective suboptimal solutions to complex problems of planning and sequential decision making under uncertainty, that for a long time were thought to be intractable.

Our first impression was that the new methods were ambitious, overly optimistic, and lacked firm foundation. Yet there were claims of impressive successes and indications of a solid core to the modern developments in reinforcement learning, suggesting that the correct approach to their understanding was through dynamic programming.

Three years later, after a lot of study, analysis, and experimentation, we believe that our initial impressions were largely correct. This is indeed an ambitious, often ad hoc, methodology, but for reasons that we now understand much better, it does have the potential of success with important and challenging problems.

This assessment still holds true!

References of this Course

This course is research-oriented. It aims:

- To explore the state of the art of approximate DP/RL at a graduate level
- To explore in depth some special research topics (rollout, policy iteration)
- To provide the opportunity for you to explore research in the area

Main references:

- Bertsekas, Reinforcement Learning and Optimal Control, Athena Scientific, 2019
- Bertsekas, Rollout, Policy Iteration, and Distributed Reinforcement Learning, Athena Scientific, 2020
- Bertsekas: Class notes based on the above, and focused on our special RL topics.
- Slides, papers, and videos from the 2019 ASU course; check my web site

Supplementary references

- Exact DP: Bertsekas, Dynamic Programming and Optimal Control, Vol. I (2017), Vol. II (2012) (also contains approximate DP material)
- Bertsekas and Tsitsiklis, Neuro-Dynamic Programming, 1996
- Sutton and Barto, 1998, Reinforcement Learning (new edition 2018, on-line)

Terminology in RL/AI and DP/Control

RL uses Max/Value, DP uses Min/Cost

- Reward of a stage = (Opposite of) Cost of a stage.
- State value = (Opposite of) State cost.
- Value (or state-value) function = (Opposite of) Cost function.

Controlled system terminology

- Agent = Decision maker or controller.
- Action = Decision or control.
- Environment = Dynamic system.

Methods terminology

- Learning = Solving a DP-related problem using simulation.
- Self-learning (or self-play in the context of games) = Solving a DP problem using simulation-based policy iteration.
- Planning vs Learning distinction = Solving a DP problem with model-based vs model-free simulation.

Notation in RL/AI and DP/Control

- Reinforcement learning uses transition probability notation $p(s, a, s')$ (s, s' are states, a is action), which is standard in finite-state Markovian Decision Problems (MDP)
- Control theory uses discrete-time system equation $x_{k+1} = f(x_k, u_k, w_k)$, which is standard in continuous spaces control problems
- Operations research uses both notations [typically $p_{ij}(u)$ for transition probabilities]

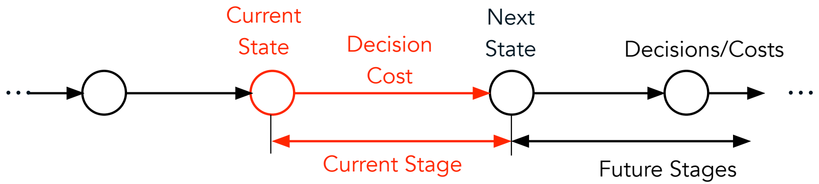
These two notational systems are mathematically equivalent but:

- Transition probabilities are cumbersome for deterministic problems and continuous spaces problems
- System equations are cumbersome for finite-state MDP

We use both notational systems:

- For the first 3/4 of the course we use system equations
- For the last 1/4 of the course we use transition probabilities

A Key Idea: Sequential Decisions w/ Approximation in Value Space



Exact DP: Making optimal decisions in stages (deterministic state transitions)

- On-line: At current state, apply decision that minimizes

$$\text{Current Stage Cost} + J^*(\text{Next State})$$

  where $J^*(\text{Next State})$ is the optimal future cost (computed off-line).
- This defines an optimal policy (an optimal control to apply at each state and stage)

Approximate DP: Use approximate cost $\tilde{J}$ instead of J^*

- On-line: At current state, apply decision that minimizes (perhaps approximately)

$$\text{Current Stage Cost} + \tilde{J}(\text{Next State})$$

- This defines a suboptimal policy

Major Approaches to Compute the Approximate Cost Function $\tilde{J}$

Problem approximation

Use as $\tilde{J}$ the optimal cost function of a related problem (computed by exact DP)

Rollout and model predictive control

Use as $\tilde{J}$ the cost function of some policy (computed somehow, perhaps according to some simplified optimization process)

Use of neural networks and other feature-based architectures

They serve as function approximators (usually obtained through off-line training)

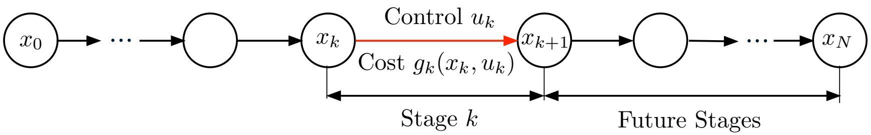
Use of simulation to generate data to “train" the architectures

Approximation architectures involve parameters that are “optimized" using data

Policy iteration/self-learning, repeated policy changes

Multiple policies are sequentially generated; each is used to provide the data to train the next

Finite Horizon Deterministic Problem



- System

$$x_{k+1} = f_k(x_k, u_k), \qquad k = 0, 1, \ldots, N-1$$

where x_k: State, u_k: Control chosen from some set $U_k(x_k)$

- Cost function:

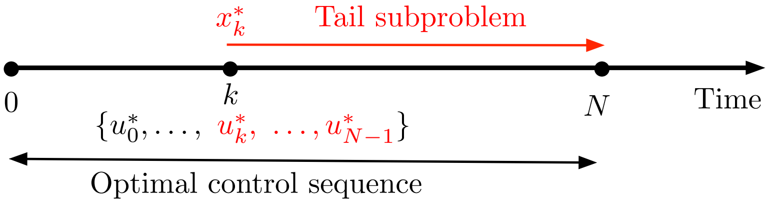
$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$$

- For given initial state x_0, minimize over control sequences $\{u_0, \ldots, u_{N-1}\}$

$$J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$$

- Optimal cost function $J^*(x_0) = \min_{\substack{u_k \in U_k(x_k) \\ k=0,\ldots,N-1}} J(x_0; u_0, \ldots, u_{N-1})$

Principle of Optimality: A Very Simple Idea



Principle of Optimality

THE TAIL OF AN OPTIMAL SEQUENCE IS OPTIMAL FOR THE TAIL SUBPROBLEM

Let $\{u_0^*, \dots, u_{N-1}^*\}$ be an optimal control sequence with corresponding state sequence $\{x_1^*, \dots, x_N^*\}$. Consider the tail subproblem that starts at x_k^* at time k and minimizes over $\{u_k, \dots, u_{N-1}\}$ the "cost-to-go" from k to N,

$$g_k(x_k^*, u_k) + \sum_{m=k+1}^{N-1} g_m(x_m, u_m) + g_N(x_N).$$

Then the tail optimal control sequence $\{u_k^*, \dots, u_{N-1}^*\}$ is optimal for the tail subproblem.

DP Algorithm: Solves All Tail Subproblems Using the Principle of Optimality

Idea of the DP algorithm

Solve all the tail subproblems of a given time length using the solution of all the tail subproblems of shorter time length

By the principle of optimality: To solve the tail subproblem that starts at x_k

- Consider every possible u_k and solve the tail subproblem that starts at next state $x_{k+1} = f_k(x_k, u_k)$. This gives the "cost of u_k"
- Optimize over all possible u_k

DP Algorithm: Produces the optimal costs $J_k^*(x_k)$ of the x_k-tail subproblems

Start with

$$J_N^*(x_N) = g_N(x_N), \qquad \text{for all } x_N,$$

and for $k = 0, \ldots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + J_{k+1}^*\big(f_k(x_k, u_k)\big) \Big], \qquad \text{for all } x_k.$$

The optimal cost $J^*(x_0)$ is obtained at the last step: $J_0(x_0) = J^*(x_0)$.

Construction of Optimal Control Sequence $\{u_0^*, \ldots, u_{N-1}^*\}$

Start with

$$u_0^* \in \arg \min_{u_0 \in U_0(x_0)} \Big[g_0(x_0, u_0) + J_1^*\big(f_0(x_0, u_0)\big) \Big],$$

and

$$x_1^* = f_0(x_0, u_0^*).$$

Sequentially, going forward, for $k = 1, 2, \ldots, N-1$, set

$$u_k^* \in \arg \min_{u_k \in U_k(x_k^*)} \Big[g_k(x_k^*, u_k) + J_{k+1}^*\big(f_k(x_k^*, u_k)\big) \Big], \qquad x_{k+1}^* = f_k(x_k^*, u_k^*).$$

Approximation in Value Space - Use Some $\tilde{J}_k$ in Place of J_k^*

Start with

$$\tilde{u}_0 \in \arg \min_{u_0 \in U_0(x_0)} \Big[g_0(x_0, u_0) + \tilde{J}_1\big(f_0(x_0, u_0)\big) \Big],$$

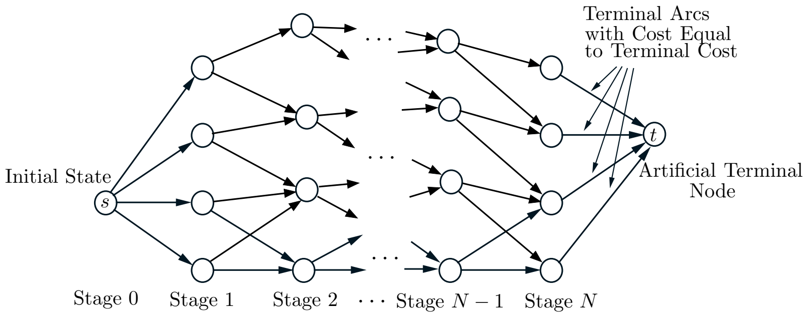
and set

$$\tilde{x}_1 = f_0(x_0, \tilde{u}_0).$$

Sequentially, going forward, for $k = 1, 2, \ldots, N-1$, set

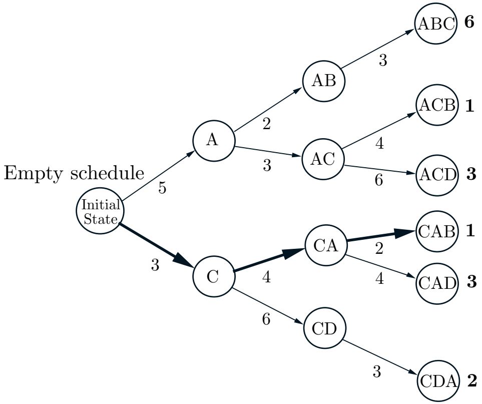
$$\tilde{u}_k \in \arg \min_{u_k \in U_k(\tilde{x}_k)} \Big[g_k(\tilde{x}_k, u_k) + \tilde{J}_{k+1}\big(f_k(\tilde{x}_k, u_k)\big) \Big], \qquad \tilde{x}_{k+1} = f_k(\tilde{x}_k, \tilde{u}_k).$$

Finite-State Problems: Shortest Path View



- Nodes correspond to states x_k
- Arcs correspond to state-control pairs (x_k, u_k)
- An arc (x_k, u_k) has start and end nodes x_k and $x_{k+1} = f_k(x_k, u_k)$
- An arc (x_k, u_k) has a cost $g_k(x_k, u_k)$. The cost to optimize is the sum of the arc costs from the initial node s to the terminal node t.
- The problem is equivalent to finding a minimum cost/shortest path from s to t.

Discrete-State Deterministic Scheduling Example

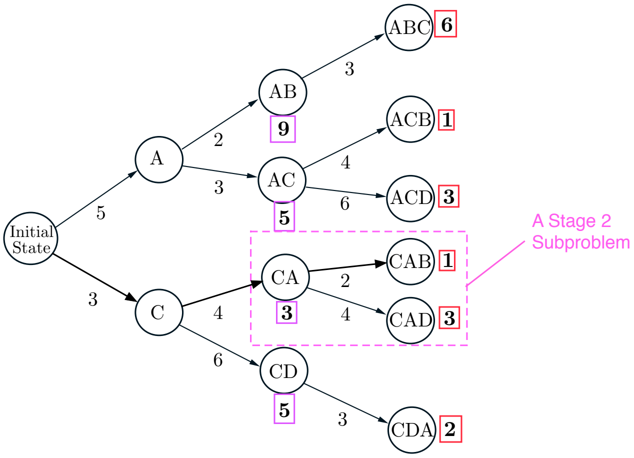


Find optimal sequence of operations A, B, C, D (A must precede B and C must precede D)

DP Problem Formulation

- States: Partial schedules; Controls: Stage 0, 1, and 2 decisions; Cost data shown along the arcs
- Recall the DP idea: Break down the problem into smaller pieces (tail subproblems)
- Start from the last decision and go backwards

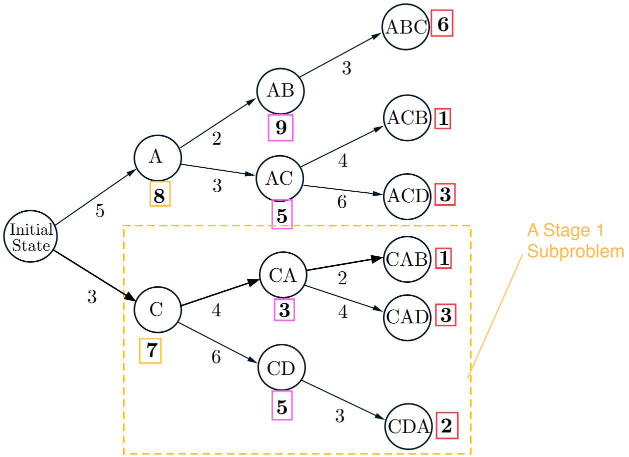
DP Algorithm: Stage 2 Tail Subproblems



Solve the stage 2 subproblems (using the terminal costs - in red)

At each state of stage 2, we record the optimal cost-to-go and the optimal decision

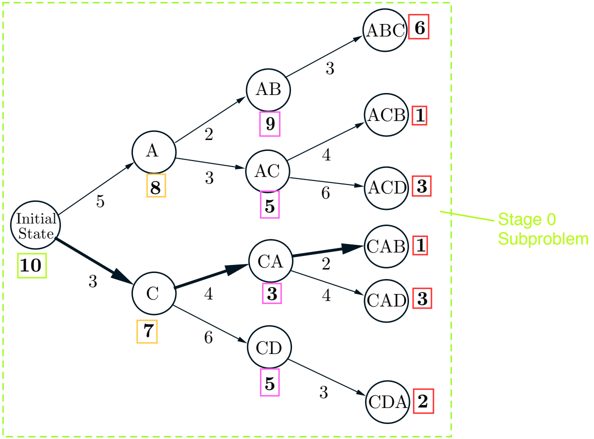
DP Algorithm: Stage 1 Tail Subproblems



Solve the stage 1 subproblems (using the optimal costs of stage 2 subproblems - in purple)

At each state of stage 1, we record the optimal cost-to-go and the optimal decision

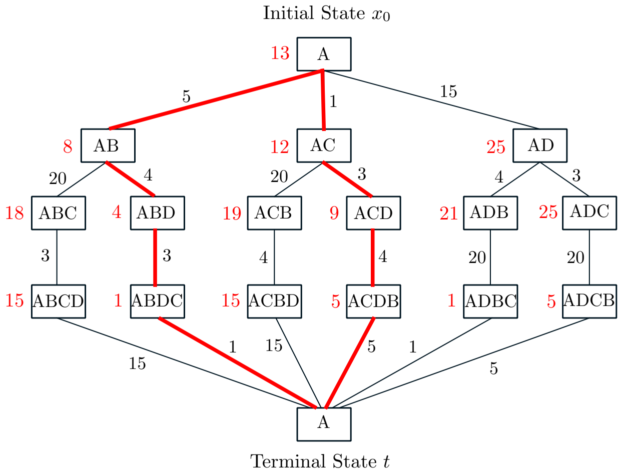
DP Algorithm: Stage 0 Tail Subproblems



Solve the stage 0 subproblem (using the optimal costs of stage 1 subproblems - in orange)

- The stage 0 subproblem is the entire problem
- The optimal value of the stage 0 subproblem is the optimal cost J^*(initial state)
- Construct the optimal sequence going forward

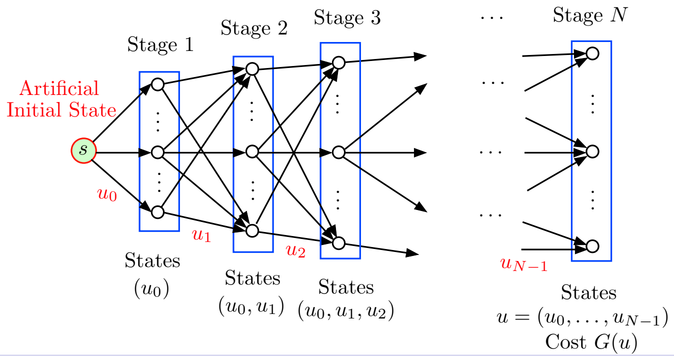
Discrete Optimization: Traveling Salesman Example; Cities A,B,C,D



Matrix of Intercity Travel Costs

■	5	1	15
5	■	20	4
1	20	■	3
15	4	3	■

General Discrete Optimization



Minimize $G(u)$ subject to $u \in U$

- Assume that each solution u has N components: $u = (u_0, \ldots, u_{N-1})$
- View the components as the controls of N stages
- Define $x_k = (u_0, \ldots, u_{k-1})$, $k = 1, \ldots, N$, and introduce artificial start state $x_0 = s$
- Define just terminal cost as $G(u)$; all other costs are 0

This formulation typically makes little sense for exact DP, but often makes a lot of sense for approximate DP/approximation in value space

Extensions

Stochastic finite horizon problems

The next state x_{k+1} is also affected by a random parameter (in addition to x_k and u_k)

Infinite horizon problems

The exact DP theory is mathematically more complex

Stochastic partial state information problems

We will convert them to problems of perfect state information, and then apply DP. Very hard to solve even approximately ... but offer great promise for applications

Minimax/game problems

The exact DP theory is substantially more complex ... but the most spectacular successes of RL involve games. We will treat lightly.

Course Aims and Requirements

Our principal aim: Think about how RL may apply to your research interests

Requirements: Course is Pass-Fail

- Homework (30%): A total of 3-4
- Research-oriented term paper (70%). A choice of:
  - A mini-research project. You may work in teams of 1-3 persons. You are encouraged to try. Selected class presentations at the end.
  - A read-and-report term paper based on 2-3 research publications (chosen by you in consultation with me)

Notation: People in AI/RL, Control Theory, and Operations Research focus on different problems and use different notations

- AI/RL and OR focus on discrete/finite-state problems which are stochastic [Markovian Decision Problems (MDP)]. Use transition probabilities $p_{ij}(u)$ to describe the uncertainty.
- Control theorists use system equation notation $x_{k+1} = f_k(x_k, u_k, w_k)$. This notation is well-suited for continuous-state problems and deterministic problems.
- You are strongly encouraged to use the notation and terminology of the course.

Syllabus (Approximate)

- Lecture 1 (this lecture): Introduction, finite horizon deterministic exact DP
- Lecture 2: Stochastic exact DP, examples of problem formulation
- Lecture 3: Examples of problem formulation
- Lecture 4: Approximation in value space, introduction to rollout (start from a policy, get a better policy)
- Lecture 5: Rollout, Monte Carlo tree search, variance reduction
- Lecture 6: Rollout with an expert, multiagent rollout, constrained rollout
- Lecture 7: Rollout and model predictive control
- Lecture 8: Multiagent systems and rollout
- Lecture 9: Combinatorial optimization and rollout
- Lecture 10: Parametric approximation architectures, feature-based architectures, (deep) neural nets, training with incremental/stochastic gradient methods
- Lecture 11: Infinite horizon problems
- Lecture 12: Variants of policy iteration: Optimistic, multistep, multiagent
- Lecture 13: Value and policy networks; use in approximate DP; perpetual rollout
- Lecture 14: Partitioned architectures and distributed asynchronous policy iteration
- Lecture 15: Project presentations

About Machine Learning and Math

Math requirements for this course are modest

Calculus, elementary probability, minimal use of vector-matrix algebra. Our objective is to use math to the extent needed to develop insight into the mechanism of various methods, and to be able to start research.

However a math framework is critically important

Human insight can only develop within some structure of human thought ... math reasoning is most suitable for this purpose

On machine learning (from NY Times Article, Dec. 2018)

“What is frustrating about machine learning is that the algorithms can’t articulate what they’re thinking. We don’t know why they work, so we don’t know if they can be trusted ... As human beings, we want more than answers. We want insight. This is going to be a source of tension in our interactions with computers from now on."

About the Next Lecture

We will cover:

- Stochastic DP algorithm
- DP algorithm for Q-factors
- Approximation in value space
- Examples of discrete and continuous DP problems

PLEASE READ AS MUCH OF CLASS NOTES AS YOU CAN

Watch the videos of Lectures 1 and 2 of the 2019 offering of the class at my web site
http://web.mit.edu/dimitrib/www/RLbook.html