Reinforcement Learning and Optimal Control

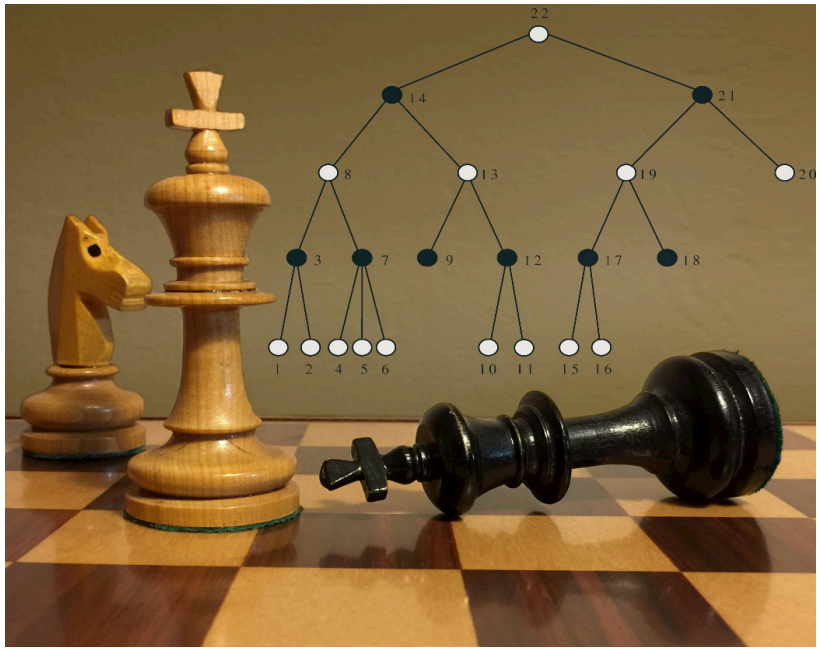
ASU, CSE 691, Winter 2019

Dimitri P. Bertsekas
dimitrib@mit.edu

Lecture 1

Outline

AlphaGo (2016) and AlphaZero (2017)



AlphaZero

Plays much better than all chess programs

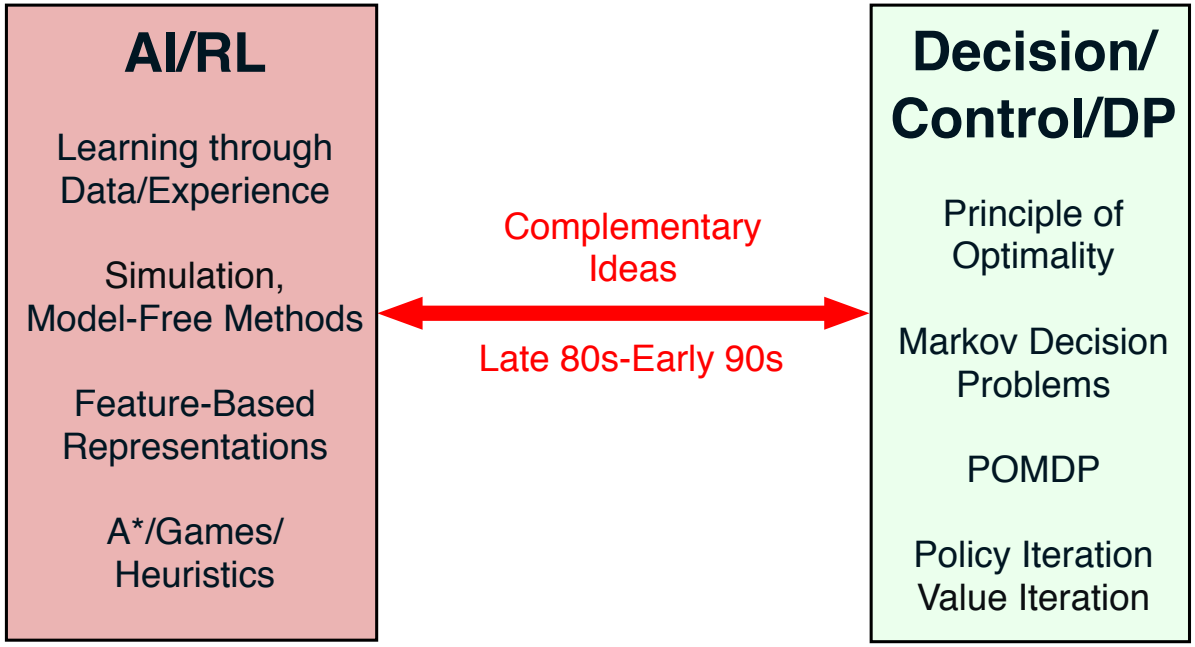
Plays different!

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

Same algorithm learned multiple games (Go, Shogi)

A technological “miracle" couched in sequential decision making methodology!

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

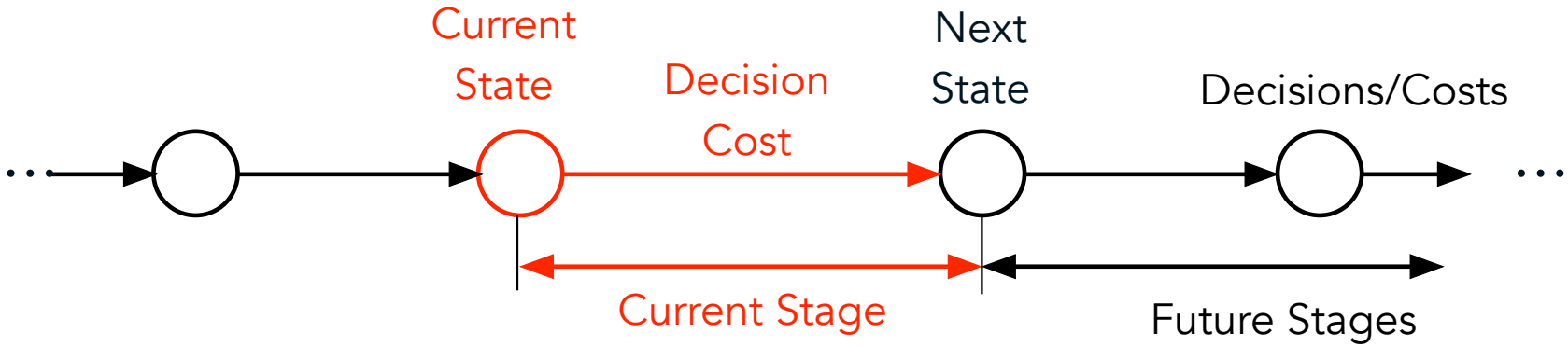
Approximate DP/RL overcomes the difficulties of exact DP by:

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



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

- 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, starting from the next state.
- 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^*

- At current state, apply decision that minimizes

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

- This defines a suboptimal policy

Major Approaches/Ideas 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 by some optimization, simulation, and approximation)

Use of neural networks and other feature-based architectures

They serve as function approximators

Use of simulation 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

Aims and References of this Course

Purpose of this course

- To explore the state of the art of approximate DP/RL at a graduate level
- To explore the common boundary between AI and optimal control
- To provide a bridge that workers with background in either field find it accessible (modest math)

Textbook: Will be followed closely

NEW DRAFT BOOK: Bertsekas, Reinforcement Learning and Optimal Control, 2019, on-line from my website

Supplementary references

- Exact DP: Bertsekas, Dynamic Programming and Optimal Control, Vol. I (2017), Vol. II (2012) (also contains approximate DP material)
- Approximate DP/RL
  - Bertsekas and Tsitsiklis, Neuro-Dynamic Programming, 1996
  - Sutton and Barto, 1998, Reinforcement Learning (new edition 2018, on-line)
  - Powell, Approximate Dynamic Programming, 2011

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.

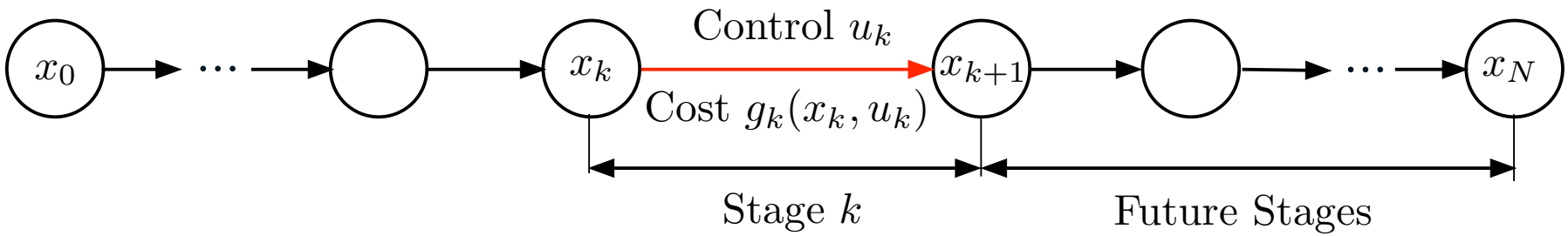
Controlled system terminology

- 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.

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:

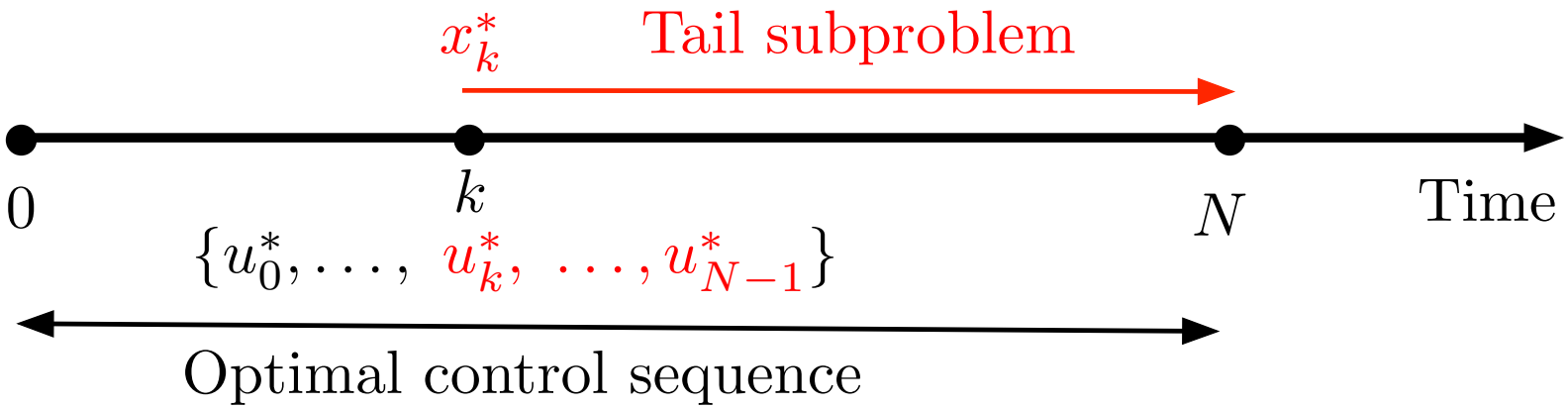
$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$$

- 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

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

THE TAIL OF AN OPTIMAL SEQUENCE 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 $x_{k+1} = f_k(x_k, u_k)$
- Optimize over all 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

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

Then 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

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

Course Requirements

- Pass-Fail
- Homework (70%): Roughly every two weeks
- Participation (30%):
  - Attend class
  - A voluntary project ($\leq$ 10 selected projects will be presented to the class at the end of the term)
  - Please point out typos, and make suggestions for clarity and improvement of my book

Our TA: Shushmita Bhattacharya, sbhatt55@asu.edu, Office hours: To be announced

Syllabus

- Lecture 1: Introduction, finite horizon deterministic exact DP
- Lecture 2: Stochastic exact DP, examples of problem formulations
- Lecture 3: Approximation in value space, problem approximation
- Lecture 4: Rollout, Monte Carlo tree search, model predictive control
- Lecture 5: Parametric approximation architectures, feature-based architectures, neural nets, training with incremental/stochastic gradient methods
- Lecture 6: Model-based and model-free parametric approximate DP
- Lecture 7: Infinite horizon discounted and stochastic shortest path problems
- Lecture 8: Forms of model-based and model-free policy iteration, self-learning
- Lecture 9: Parametric approximation in policy space, policy gradient methods, cross-entropy method
- Lecture 10: Additional methods, temporal difference methods
- Lecture 11: Problem approximation by aggregation
- Lecture 12: Feature-based and biased aggregation

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.

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:

- Examples of discrete and continuous deterministic DP problems
- Stochastic DP algorithm
- DP algorithm for Q-factors
- Partial information problems

PLEASE READ AS MUCH OF CHAPTER 1 AS YOU CAN