Reinforcement Learning and Optimal Control
A Selective Overview

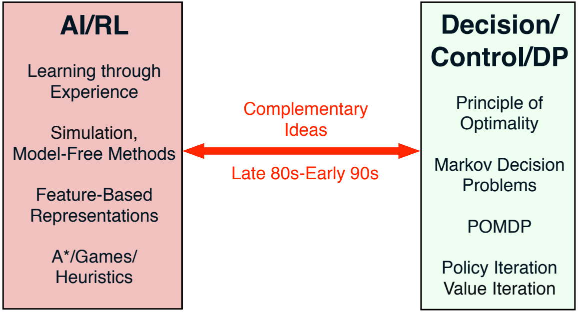
Dimitri P. Bertsekas

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

2018 CDC

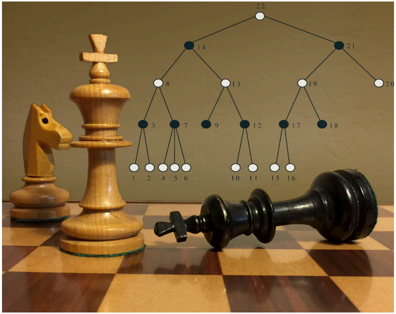
December 2018

Reinforcement Learning (RL): A Happy Union of AI and Decision/Control Ideas



Historical highlights

- Exact DP, optimal control (Bellman, Shannon, 1950s ...)
- 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)



AlphaZero

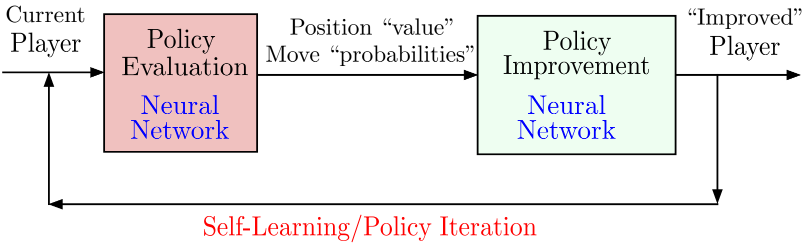
Plays much better than all chess programs

Plays different!

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

Same algorithm learned multiple games (Go, Shogi)

AlphaZero was Trained Using Self-Generated Data



- The "current" player plays games that are used to "train" an "improved" player
- At a given position, the "move probabilities" and the "value" of a position are approximated by a deep neural net (NN)
- Successive NNs are trained using self-generated data and a form of regression
- A form of randomized policy improvement Monte-Carlo Tree Search (MCTS) generates move probabilities
- 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

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 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

Approximation in Value Space

Central Idea: Lookahead with an approximate cost

- Compute an approximation $\tilde{J}$ to the optimal cost function J^*
- At current state, apply control that attains the minimum in

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

Multistep lookahead extension

- At current state solve an ℓ-step DP problem using terminal cost $\tilde{J}$
- Apply the first control in the optimal policy for the ℓ-step problem

Example approaches to compute $\tilde{J}$:

- Problem approximation: Use as $\tilde{J}$ the optimal cost function of a simpler problem
- Rollout and model predictive control: Use a single policy iteration, with cost evaluated on-line by simulation or limited optimization
- Self-learning/approximate policy iteration (API): Use as $\tilde{J}$ an approximation to the cost function of the final policy obtained through a policy iteration process
- Role of neural networks: "Learn" the cost functions of policies in the context of API; "learn" policies obtained by value space approximation

Aims and References of this Talk

The purpose of this talk

To selectively review some of the methods, and bring out some of the AI-DP connections

References

- Quite a few Exact DP books (1950s-present starting with Bellman; my latest book “Abstract DP" came out earlier this year)
- Quite a few DP/Approximate DP/RL/Neural Nets books (1996-Present)
  - Bertsekas and Tsitsiklis, Neuro-Dynamic Programming, 1996
  - Sutton and Barto, 1998, Reinforcement Learning (new edition 2019, Draft on-line)
  - NEW DRAFT BOOK: Bertsekas, Reinforcement Learning and Optimal Control, 2019, on-line
- Many surveys on all aspects of the subject; Tesauro’s papers on computer backgammon, and Silver, et al., papers on AlphaZero

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 = Control.
- Environment = Dynamic system.

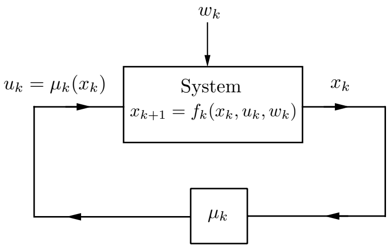
Methods terminology

- 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.

Outline

1. Approximation in Value Space
2. Problem Approximation
3. Rollout and Model Predictive Control
4. Parametric Approximation - Neural Networks
5. Neural Networks and Approximation in Value Space
6. Model-free DP in Terms of Q-Factors
7. Policy Iteration - Self-Learning

Finite Horizon Problem - Exact DP



- System

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

where x_k: State, u_k: Control, w_k: Random disturbance

- Cost function:

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

- Perfect state information: u_k is applied with (exact) knowledge of x_k
- Optimization over feedback policies $\{\mu_0, \ldots, \mu_{N-1}\}$: Rules that specify the control $\mu_k(x_k)$ to apply at each possible state x_k that can occur

The DP Algorithm and Approximation in Value Space

Go backwards, $k = N-1, \ldots, 0$, using

$$J_N(x_N) = g_N(x_N)$$

$$J_k(x_k) = \min_{u_k} E_{w_k}\Big\{g_k(x_k, u_k, w_k) + J_{k+1}\big(f_k(x_k, u_k, w_k)\big)\Big\}$$

$J_k(x_k)$: Optimal cost-to-go starting from state x_k

Approximate DP is motivated by the ENORMOUS computational demands of exact DP

Approximation in value space: Use an approximate cost-to-go function $\tilde{J}_{k+1}$

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k} E_{w_k}\Big\{g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big)\Big\}$$

There is also a multistep lookahead version

At state x_k solve an ℓ-step DP problem with terminal cost function approximation $\tilde{J}_{k+\ell}$.
Use the first control in the optimal ℓ-step sequence.

One-step case at state x_k:

Approximate minimization

First Step — "Future"

$$\min_{u_k} E\Big\{g_k(x_k,u_k,w_k) + \tilde J_{k+1}(x_{k+1})\Big\}$$

Approximations:

Simplify $E\{\cdot\}$
(certainty equivalence)
Adaptive simulation

Computation of $\tilde J_{k+1}$:

Problem approximation
Rollout
Model Predictive Control
Parametric approximation
Aggregation

Multistep case at state x_k:

DP minimization

First ℓ Steps — "Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{g_k(x_k,u_k,w_k) + \sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big) + \tilde J_{k+\ell}(x_{k+\ell})\right\}$$

Lookahead Minimization

Cost-to-go Approximation

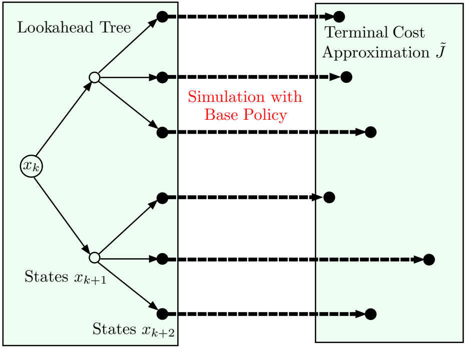
Problem Approximation: Simplify the Tail Problem and Solve it Exactly

Use as cost-to-go approximation $\tilde{J}_{k+1}$ the exact cost-to-go of a simpler problem

Many problem-dependent possibilities:

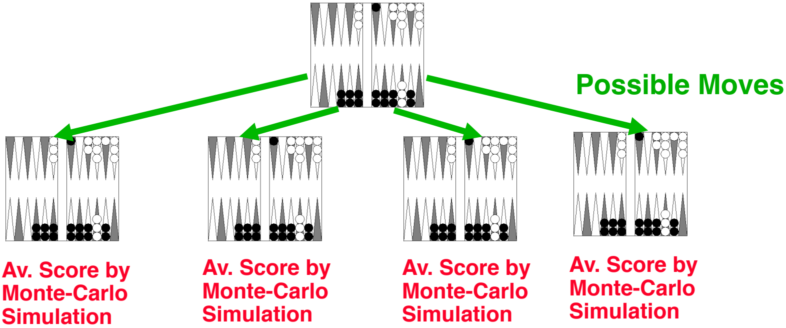
- Probabilistic approximation
  - Certainty equivalence: Replace stochastic quantities by deterministic ones (makes the lookahead minimization deterministic)
  - Approximate expected values by limited simulation
  - Partial versions of certainty equivalence
- Enforced decomposition of coupled subsystems
  - One-subsystem-at-a-time optimization
  - Constraint decomposition
  - Lagrangian relaxation
- Aggregation: Group states together and view the groups as aggregate states
  - Hard aggregation: $\tilde{J}_{k+1}$ is a piecewise constant approximation to J_{k+1}
  - Feature-based aggregation: The aggregate states are defined by “features" of the original states
  - Biased hard aggregation: $\tilde{J}_{k+1}$ is a piecewise constant local correction to some other approximation $\hat{J}_{k+1}$, e.g., one provided by a neural net

Rollout: On-Line Simulation-Based Approximation in Value Space



- The base policy can be any suboptimal policy (obtained by another method)
- One-step or multistep lookahead; exact minimization or a “randomized form of lookahead" that involves “adaptive" simulation and Monte Carlo tree search
- With or without terminal cost approximation (obtained by another method)
- Some forms of model predictive control can be viewed as special cases (base policy is a short-term deterministic optimization)
- Important theoretical fact: With exact lookahead and no terminal cost approximation, the rollout policy improves over the base policy

Example of Rollout: Backgammon (Tesauro, 1996)



- Base policy was a backgammon player developed by a different RL method [TD(λ) trained with a neural network]; was also used for terminal cost approximation
- The best backgammon players are based on rollout ... but are too slow for real-time play (MC simulation takes too long)

AlphaGo has similar structure to backgammon

The base policy and terminal cost approximation are obtained with a deep neural net. In AlphaZero the rollout-with-base-policy part was dropped (long lookahead suffices)

Lookahead Minimization | **Cost-to-go Approximation**

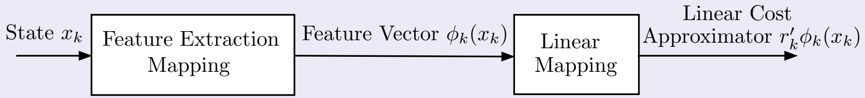
First ℓ Steps | "Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{ g_k(x_k,u_k,w_k) + \sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big) + \tilde J_{k+\ell}(x_{k+\ell}) \right\}$$

Parametric approximation

$\tilde J_k$ comes from a class of functions $\tilde J_k(x_k, r_k)$, where r_k is a tunable parameter vector

Feature-based architectures: The linear case



Training with Fitted Value Iteration

This is just DP with intermediate approximation at each step

- Start with $\tilde{J}_N = g_N$ and sequentially train going backwards, until $k = 0$
- Given $\tilde{J}_{k+1}$, we construct a number of samples (x_k^s, β_k^s), $s = 1, \ldots, q$,

$$\beta_k^s = \min_u E\Big\{g(x_k^s, u, w_k) + \tilde{J}_{k+1}\big(f_k(x_k^s, u, w_k), r_{k+1}\big)\Big\}, \qquad s = 1, \ldots, q$$

- We “train" $\tilde{J}_k$ on the set of samples (x_k^s, β_k^s), $s = 1, \ldots, q$

Training by least squares/regression

- We minimize over r_k

$$\sum_{s=1}^{q} \big(\tilde{J}_k(x_k^s, r_k) - \beta^s\big)^2 + \gamma \|r_k - \bar{r}\|^2$$

where $\bar{r}$ is an initial guess for r_k and $\gamma > 0$ is a regularization parameter

Major fact about neural networks

They automatically construct features to be used in a linear architecture

- Neural nets are approximation architectures of the form

$$\tilde{J}(x, v, r) = \sum_{i=1}^{m} r_i \phi_i(x, v) = r' \phi(x, v)$$

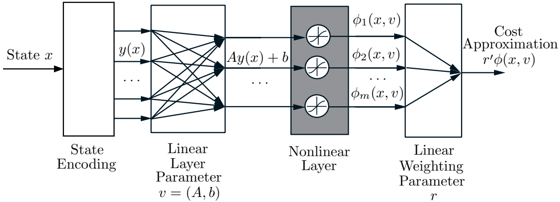
involving two parameter vectors r and v with different roles

- View $\phi(x, v)$ as a feature vector
- View r as a vector of linear weights for $\phi(x, v)$
- By training v jointly with r, we obtain automatically generated features!

Neural nets can be used in the fitted value iteration scheme

Train the stage k neural net (i.e., compute $\tilde{J}_k$) using a training set generated with the stage $k+1$ neural net (which defines $\tilde{J}_{k+1}$)

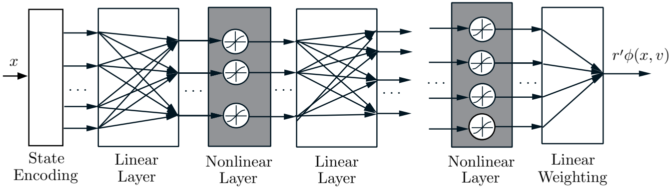
Neural Network with a Single Nonlinear Layer



- State encoding (could be the identity, could include special features of the state)
- Linear layer $Ay(x)+b$ [parameters to be determined: $v=(A,b)$]
- Nonlinear layer produces m outputs $\phi_i(x,v)=\sigma\Big(\big(Ay(x)+b\big)_i\Big)$, $i=1,\ldots,m$
- σ is a scalar nonlinear differentiable function; several types have been used (hyperbolic tangent, logistic, rectified linear unit)
- Training problem is to use the training set (x^s,β^s), $s=1,\ldots,q$, for

$$\min_{v,r}\sum_{s=1}^{q}\left(\sum_{i=1}^{m}r_i\phi_i(x^s,v)-\beta^s\right)^2+\text{(Regularization Term)}$$

- Solved often with incremental gradient methods (known as backpropagation)
- Universal approximation theorem: With sufficiently large number of parameters, “arbitrarily" complex functions can be closely approximated



- More complex NNs are formed by concatenation of multiple layers
- The outputs of each nonlinear layer become the inputs of the next linear layer
- A hierarchy of features
- Considerable success has been achieved in major contexts

Possible reasons for the success

- With more complex features, the number of parameters in the linear layers may be drastically decreased
- We may use matrices *A* with a special structure that encodes special linear operations such as convolution

Q-Factors - Model-Free RL

- The Q-factor of a state-control pair (x_k, u_k) at time k is defined by

$$Q_k(x_k, u_k) = E\Big\{g_k(x_k, u_k, w_k) + J_{k+1}(x_{k+1})\Big\}$$

where J_{k+1} is the optimal cost-to-go function for stage $k+1$

- Note that

$$J_k(x_k) = \min_{u \in U_k(x_k)} Q_k(x_k, u_k)$$

so the DP algorithm is written in terms of Q_k

$$Q_k(x_k, u_k) = E\Big\{g_k(x_k, u_k, w_k) + \min_{u \in U_{k+1}(x_{k+1})} Q_{k+1}(x_{k+1}, u)\Big\}$$

- We can approximate Q-factors instead of costs

Fitted Value Iteration for Q-Factors: Model-Free Approximate DP

- Consider fitted value iteration of Q-factor parametric approximations

$$\tilde{Q}_k(x_k, u_k, r_k) \approx E\Big\{g_k(x_k, u_k, w_k) + \min_{u \in U_{k+1}(x_{k+1})} \tilde{Q}_{k+1}(x_{k+1}, u, r_{k+1})\Big\}$$

(Note a mathematical magic: The order of $E\{\cdot\}$ and min have been reversed.)

- We obtain $\tilde{Q}_k(x_k, u_k, r_k)$ by training with many pairs $\big((x_k^s, u_k^s), \beta_k^s\big)$, where β_k^s is a sample of the approximate Q-factor of (x_k^s, u_k^s). No need to compute $E\{\cdot\}$
- No need for a model to obtain β_k^s. Sufficient to have a simulator that generates random samples of state-control-cost-next state

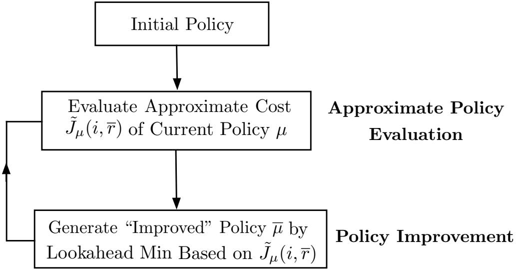
$$\big((x_k, u_k), (g_k(x_k, u_k, w_k), x_{k+1})\big)$$

- Having computed r_k, the one-step lookahead control is obtained on-line as

$$\overline{\mu}_k(x_k) = \arg \min_{u \in U_k(x_k)} \tilde{Q}_k(x_k, u, r_k)$$

without the need of a model or expected value calculations

- Also the on-line calculation of the control is simplified



- Most popular setting: Stationary finite-state system, stationary policies, discounting or termination state
- Policy iteration (PI) method generates a sequence of policies
  - The current policy μ is evaluated using a parametric architecture: $\tilde J_\mu(x,\overline r)$
  - An "improved" policy $\overline\mu$ is obtained by one-step lookahead using $\tilde J_\mu(x,\overline r)$
- The architecture is trained using simulation data with μ
- Thus the system "observes itself" under μ and uses the data to "learn" the improved policy $\overline\mu$ - "self-learning"
- Exact PI converges to an optimal policy; approximate PI "converges" to within an "error zone" of the optimal, then oscillates
- TD-Gammon, AlphaGo, and AlphaZero, all use forms of approximate PI for training

A Few Topics we did not Cover in this Talk

- Infinite horizon extensions: Approximate value and policy iteration methods, error bounds, model-based and model-free methods
- Temporal difference methods: A class of methods for policy evaluation in infinite horizon problems with a rich theory, issues of variance-bias tradeoff
- Sampling for exploration, in the context of policy iteration
- Monte Carlo tree search, and related methods
- Aggregation methods, synergism with other approximate DP methods
- Approximation in policy space, actor-critic methods, policy gradient and cross-entropy methods
- Special aspects of imperfect state information problems, connections with traditional control schemes
- Infinite spaces optimal control, connections with aggregation schemes
- Special aspects of deterministic problems: Shortest paths and their use in approximate DP
- A broad view of using simulation for large-scale computations: Methods for large systems of equations and linear programs, connection to proximal algorithms

Concluding Remarks

Some words of caution

- There are challenging implementation issues in all approaches, and no fool-proof methods
- Problem approximation and feature selection require domain-specific knowledge
- Training algorithms are not as reliable as you might think by reading the literature
- Approximate PI involves oscillations
- Recognizing success or failure can be a challenge!
- The RL successes in game contexts are spectacular, but they have benefited from perfectly known and stable models and small number of controls (per state)
- Problems with partial state observation remain a big challenge

On the positive side

- Massive computational power together with distributed computation are a source of hope
- Silver lining: We can begin to address practical problems of unimaginable difficulty!
- There is an exciting journey ahead!

Thank you!