Topics in Reinforcement Learning: Lessons from AlphaZero for (Sub)Optimal Control and Discrete Optimization

Arizona State University
Course CSE 691, Spring 2023


Dimitri P. Bertsekas
dbertsek@asu.edu

Lecture 2
Stochastic Finite and Infinite Horizon DP
Review - Finite Horizon Deterministic Problem

- System

\[ x_{k+1} = f_k(x_k, u_k), \quad k = 0, 1, \ldots, N - 1 \]

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

- Arbitrary state and control spaces

- 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 \( J^*(x_0) = \min_{u_k \in U_k(x_k)} J(x_0; u_0, \ldots, u_{N-1}) \)
Go backward to compute the optimal costs $J_k^*(x_k)$ of the $x_k$-tail subproblems (off-line training - involves lots of computation)

Start with

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

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

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

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

Go forward to construct optimal control sequence $\{u_0^*, \ldots, u_{N-1}^*\}$ (on-line play)

Start with

$$u_0^* \in \arg \min_{u_0 \in U_0(x_0)} \left[ g_0(x_0, u_0) + J_1^*(f_0(x_0, u_0)) \right], \quad x_1^* = f_0(x_0, u_0^*).$$

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

$$u_k^* \in \arg \min_{u_k \in U_k(x_k^*)} \left[ g_k(x_k^*, u_k) + J_{k+1}^*(f_k(x_k^*, u_k)) \right], \quad x_{k+1}^* = f_k(x_k^*, u_k^*).$$
An alternative (and equivalent) form of the DP algorithm

Generates the optimal Q-factors, defined for all \((x_k, u_k)\) and \(k\) by

\[
Q^*_k(x_k, u_k) = g_k(x_k, u_k) + J^*_{k+1}(f_k(x_k, u_k))
\]

The optimal cost function \(J^*_k\) can be recovered from the optimal Q-factor \(Q^*_k\)

\[
J^*_k(x_k) = \min_{u_k \in U_k(x_k)} Q^*_k(x_k, u_k)
\]

The DP algorithm can be written in terms of Q-factors

\[
Q^*_k(x_k, u_k) = g_k(x_k, u_k) + \min_{u_{k+1} \in U_{k+1}(f_k(x_k, u_k))} Q^*_k(f_k(x_k, u_k), u_{k+1})
\]

Exact and approximate forms of this and other related algorithms, form an important class of RL methods known as Q-learning.
We replace $J_k^*$ with an approximation $\tilde{J}_k$ during on-line play.

- **Start with**
  \[
  \tilde{u}_0 \in \arg \min_{u_0 \in U_0(x_0)} \left[ g_0(x_0, u_0) + \tilde{J}_1(f_0(x_0, u_0)) \right]
  \]

- **Set** $\tilde{x}_1 = f_0(x_0, \tilde{u}_0)$

- **Sequentially,** going forward, for $k = 1, 2, \ldots, N-1$, set
  \[
  \tilde{u}_k \in \arg \min_{u_k \in U_k(\tilde{x}_k)} \left[ g_k(\tilde{x}_k, u_k) + \tilde{J}_{k+1}(f_k(\tilde{x}_k, u_k)) \right], \quad \tilde{x}_{k+1} = f_k(\tilde{x}_k, \tilde{u}_k)
  \]

**How do we compute $\tilde{J}_{k+1}(x_{k+1})$?** This is one of the principal issues in RL.

- **Off-line problem approximation:** Use as $\tilde{J}_{k+1}$ the optimal cost function of a simpler problem, computed off-line by exact DP.

- **On-line approximate optimization,** e.g., solve on-line a shorter horizon problem by multistep lookahead minimization and simple terminal cost (often done in MPC).

- **Parametric cost approximation:** Obtain $\tilde{J}_{k+1}(x_{k+1})$ from a parametric class of functions $J(x_{k+1}, r)$, where $r$ is a parameter, e.g., training using data and a NN.

- **Rollout with a heuristic:** We will focus on this for the moment.
Rollout for Finite-State Deterministic Problems

Cost approximation by running a heuristic from states of interest

We generate a single system trajectory \( \{x_0, x_1, \ldots, x_N\} \) by on-line play

- Upon reaching \( x_k \), we compute for all \( u_k \in U_k(x_k) \), the corresponding next states \( x_{k+1} = f_k(x_k, u_k) \)
- From each of the next states \( x_{k+1} \) we run the heuristic and compute the heuristic cost \( H_{k+1}(x_{k+1}) \)
- We apply \( \tilde{u}_k \) that minimizes over \( u_k \in U_k(x_k) \), the (heuristic) Q-factor
  \[ g_k(x_k, u_k) + H_{k+1}(x_{k+1}) \]
- We generate the next state \( x_{k+1} = f_k(x_k, \tilde{u}_k) \) and repeat
Stochastic DP Problems - Perfect State Observation (We Know $x_k$)

- System $x_{k+1} = f_k(x_k, u_k, w_k)$ with random "disturbance" $w_k$ (e.g., physical noise, market uncertainties, demand for inventory, unpredictable breakdowns, etc).
- Cost function: $E \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \right\}$
- Policies $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, where $\mu_k$ is a "closed-loop control law" or "feedback policy"/a function of $x_k$. A "lookup table" for the control $u_k = \mu_k(x_k)$ to apply at $x_k$.
- An important point: Using feedback (i.e., choosing controls with knowledge of the state) is beneficial in view of the stochastic nature of the problem.
- For given initial state $x_0$, minimize over all $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$ the cost

$$J_\pi(x_0) = E \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right\}$$

- Optimal cost function: $J^*(x_0) = \min_\pi J_\pi(x_0)$. Optimal policy: $J_{\pi^*}(x_0) = J^*(x_0)$
The Stochastic DP Algorithm

**Produces the optimal costs** \( J_k^*(x_k) \) **of the tail subproblems that start at** \( x_k \)

Start with \( J_N^*(x_N) = g_N(x_N) \), and for \( k = 0, \ldots, N - 1 \), let

\[
J_k^*(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k} \left\{ g_k(x_k, u_k, w_k) + J_{k+1}^*(f_k(x_k, u_k, w_k)) \right\}, \quad \text{for all } x_k.
\]

- The optimal cost \( J^*(x_0) \) is obtained at the last step: \( J_0^*(x_0) = J^*(x_0) \).
- The optimal policy component \( \mu_k^* \) can be constructed simultaneously with \( J_k^* \), and consists of the minimizing \( u_k^* = \mu_k^*(x_k) \) above.

**Alternative on-line implementation of the optimal policy, given** \( J_1^*, \ldots, J_{N-1}^* \)

Sequentially, going forward, for \( k = 0, 1, \ldots, N - 1 \), observe \( x_k \) and apply

\[
u_k^* \in \arg \min_{u_k \in U_k(x_k)} E_{w_k} \left\{ g_k(x_k, u_k, w_k) + J_{k+1}^*(f_k(x_k, u_k, w_k)) \right\}.
\]

**Issues:** Need to know \( J_{k+1}^* \), compute \( E_{w_k} \{ \cdot \} \) for each \( u_k \), minimize over all \( u_k \).
### A Very Favorable Case: Linear-Quadratic Problems

#### One-dimensional linear-quadratic problem

- **System**: \( x_{k+1} = ax_k + bu_k + w_k \) (\( a \) and \( b \) are given scalars)
- **Cost**: over \( N \) stages: \( qx_N^2 + \sum_{k=0}^{N-1} (qx_k^2 + ru_k^2) \), where \( q > 0 \) and \( r > 0 \) are given scalars
- **DP algorithm**: starts with \( J^*_N(x_N) = qx_N^2 \), and generates \( J^*_k \) according to
  \[
  J^*_k(x_k) = \min_{u_k} E_{w_k} \left\{ qx_k^2 + ru_k^2 + J^*_{k+1}(ax_k + bu_k + w_k) \right\}, \quad k = 0, \ldots, N - 1
  \]
- **DP algorithm**: can be carried out in closed form to yield
  \[
  J^*_k(x_k) = K_k x_k^2 + \text{const}, \quad \mu^*_k(x_k) = L_k x_k \] 
  \( K_k \) and \( L_k \) can be explicitly computed
- **Certainty Equivalence**: \( \mu^*_k(x_k) \) does not depend on the distribution of \( w_k \) as long as it has 0 mean:

#### These results generalize to multidimensional linear-quadratic problems

- \( x_k \in \mathbb{R}^n, u_k \in \mathbb{R}^m \); the scalars \( a, b, q, r \) are replaced by matrices \( A, B, Q, R \)
Derivation - DP Algorithm starting from Terminal Cost $J^*_N(x_N) = qx_N^2$

\[
J^*_{N-1}(x_{N-1}) = \min_{u_{N-1}} E\left\{ qx_{N-1}^2 + ru_{N-1}^2 + J^*_N(ax_{N-1} + bu_{N-1} + w_{N-1}) \right\}
\]

\[
= \min_{u_{N-1}} E\left\{ qx_{N-1}^2 + ru_{N-1}^2 + q(ax_{N-1} + bu_{N-1} + w_{N-1})^2 \right\}
\]

\[
= \min_{u_{N-1}} \left[ qx_{N-1}^2 + ru_{N-1}^2 + q(ax_{N-1} + bu_{N-1})^2 + 2q E\{w_{N-1}\}(ax_{N-1} + bu_{N-1}) + q E\{w_{N-1}^2\} \right]
\]

\[
= qx_{N-1}^2 + \min_{u_{N-1}} \left[ ru_{N-1}^2 + q(ax_{N-1} + bu_{N-1})^2 \right] + q\sigma^2
\]

Minimize by setting to zero the derivative: $0 = 2ru_{N-1} + 2qb(ax_{N-1} + bu_{N-1})$, to obtain

\[
\mu^*_{N-1}(x_{N-1}) = L_{N-1}x_{N-1} \quad \text{with} \quad L_{N-1} = -\frac{abq}{r + b^2q}
\]

and by substitution, $J^*_{N-1}(x_{N-1}) = K_{N-1}x_{N-1}^2 + q\sigma^2$, where $K_{N-1} = \frac{a^2rq}{r + b^2q} + q$

Similarly, going backwards (starting with $K_N = q$), we obtain for all $k$:

\[
J^*_k(x_k) = K_k x_k^2 + \sigma^2 \sum_{m=k}^{N-1} K_{m+1}, \quad \mu^*_k(x_k) = L_k x_k, \quad K_k = \frac{a^2rK_{k+1}}{r + b^2K_{k+1}} + q, \quad L_k = -\frac{abK_{k+1}}{r + b^2K_{k+1}}
\]
Observations and generalizations

- The solution does not depend on the distribution of $w_k$, only on the mean (which is 0), i.e., we have **certainty equivalence** (the stochastic problem can be replaced by a deterministic problem)
- Generalization to **multidimensional problems**, nonzero mean disturbances, etc
- Generalization to **infinite horizon**
- Generalization to problems where the **state is observed partially through linear measurements**: Optimal policy involves an extended form of certainty equivalence

$$L_k E\{x_k \mid \text{measurements}\}$$

where $E\{x_k \mid \text{measurements}\}$ is provided by an estimator (e.g., Kalman filter)

- Linear systems and quadratic cost are a starting point for other lines of investigations and approximations:
  - Problems with safety/state constraints [Model Predictive Control (MPC)]
  - Problems with control constraints (MPC)
  - Unknown or changing system parameters (adaptive control)
**Approximation in Value Space - The Three Approximations**

**Simplified minimization**

At $x_k$

\[
\min_{u_k} E\left\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(x_{k+1}) \right\}
\]

Expected value approximation

Cost-to-go approximation

**First Step**

**“Future”**

**“On-Line Play”**

**Important variants:** Use **multistep lookahead**, use **multiagent rollout** (for multicomponent control problems)

**Multistep lookahead (performance - computational overhead tradeoff)**

**At State** $x_k$

**DP minimization**

\[
\min_{u_k, \mu_{k+1}, \ldots, \mu_{k+\ell-1}} E\left\{ g_k(x_k, u_k, w_k) + \sum_{m=k+1}^{k+\ell-1} g_m(x_m, \mu_m(x_m), w_m) + \tilde{J}_{k+\ell}(x_{k+\ell}) \right\}
\]

**First $\ell$ Steps**

**“Future”**

**Lookahead Minimization**

**Cost-to-go Approximation**
Constructing Approximations

Approximate Min
Discretization
Selective Minimization

At \( x_k \)
\[
\min_{u_k} E\left\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(x_{k+1}) \right\}
\]

Approximate \( E\{\cdot\} \)
Adaptive simulation
Monte Carlo tree search

First Step

“Future”

Approximate Cost-to-Go \( \tilde{J}_{k+1} \)
Certainty equivalence
Problem approximation
Rollout, Model Predictive Control
Parametric approximation
Neural nets
Aggregation

An example: Truncated rollout with base policy and terminal cost approximation (however obtained, e.g., off-line training)
Approximation in Policy Space: The Major Alternative to Approximation in Value Space

Control
\[ u_k = \tilde{\mu}_k(x_k, r_k) \]

System
Environment
State \( x_k \)

Controller
\[ \tilde{\mu}_k(\cdot, r_k) \]

Training Data

- **Idea:** Select the policy by optimization over a suitably restricted class of policies
- The restricted class is usually a parametric family of policies \( \tilde{\mu}_k(x_k, r_k) \), \( k = 0, \ldots, N - 1 \), of some form, where \( r_k \) is a parameter (e.g., a neural net)
- Methods used for optimization/off-line training: Random search, policy gradient, classification (to be discussed later)
- **Important advantage once the parameters \( r_k \) are computed:** The on-line computation of controls is often much faster ... at state \( x_k \) apply \( u_k = \tilde{\mu}_k(x_k, r_k) \)
- **Important disadvantage:** It does not allow for on-line replanning ... no Newton step
An Important Conceptual Difference Between Approximation in Value and in Policy Space

Approximation in value space is primarily an “on-line play" method
with off-line training used optionally to construct cost function approximations for one-step or multistep lookahead

Approximation in policy space is primarily an “off-line training" method
which may be used optionally to provide a policy for on-line rollout
The approximate cost-to-go functions $\tilde{J}_{k+1}$ define a suboptimal policy $\tilde{\mu}_k$ through one-step or multistep lookahead minimization

- Given functions $\tilde{J}_{k+1}$, how do we simplify the computation of $\tilde{\mu}_k$?
- **Idea:** Use approximation in policy space to represent $\tilde{\mu}_k$: Approximate $\tilde{\mu}_k$ using a training set of a large number $q$ of sample pairs $(x^s_k, u^s_k)$, $s = 1, \ldots, q$, where $u^s_k = \tilde{\mu}_k(x^s_k)$:

$$u^s_k \in \arg \min_{u \in U_k(x_k)} E \left\{ g_k(x^s_k, u, w_k) + \tilde{J}_{k+1}(f_k(x^s_k, u, w_k)) \right\}$$

- **Example:** Introduce a parametric family of randomized policies $\mu_k(x_k, r_k)$, $k = 0, \ldots, N - 1$, of some form (e.g., a neural net), where $r_k$ is a parameter. Then estimate the parameters $r_k$ by least squares fit:

$$r_k \in \arg \min_r \sum_{s=1}^{q} \left\| u^s_k - \mu_k(x^s_k, r) \right\|^2$$

- Relation to classification methods ... policy $\leftrightarrow$ classifier; more on this later.
All our lectures will have a 15-minute break, somewhere in the middle
Catch our breath and think about issues relating to the first half of the lecture.
A short discussion/questions/answers period will follow each break.
Infinite Horizon Problems

System $x_{k+1} = f(x_k, u_k, w_k)$ with state, control, and random disturbance.

Policies $\pi = \{\mu_0, \mu_1, \ldots\}$ with $\mu_k(x) \in U(x)$ for all $x$ and $k$.

Cost of stage $k$: $\alpha^k g(x_k, \mu_k(x_k), w_k)$.

$0 < \alpha \leq 1$ is the discount factor. If $\alpha < 1$ the problem is called discounted.

Cost of a policy $\pi = \{\mu_0, \mu_1, \ldots\}$: The limit as $N \to \infty$ of the $N$-stage costs

$$J_\pi(x_0) = \lim_{N \to \infty} E_{w_k} \left\{ \sum_{k=0}^{N-1} \alpha^k g(x_k, \mu_k(x_k), w_k) \right\}$$

Optimal cost function $J^*(x_0) = \min_\pi J_\pi(x_0)$.

Problems with $\alpha = 1$ typically include a special cost-free termination state $t$. The objective is to reach (or approach) $t$ at minimum expected cost.
\( k \)-stages opt. cost \( \rightarrow \) Infinite horizon opt. cost as \( k \rightarrow \infty \)

- We have \( J^*(x) = \lim_{k \rightarrow \infty} J_k(x) \), for all \( x \), where for any \( k \), \( J_k(x) = k \)-stages optimal cost starting from \( x \), and is generated by

\[
J_k(x) = \min_{u \in U(x)} E_w \left\{ g(x, u, w) + \alpha J_{k-1}(f(x, u, w)) \right\}, \quad J_0(x) \equiv 0 \tag{VI}
\]

- Derivation using DP: Let \( V_{N-k}(x) \) be the optimal cost-to-go starting at \( x \) with \( k \) stages to go,

\[
V_{N-k}(x) = \min_{u \in U(x)} E_w \left\{ \alpha^{N-k} g(x, u, w) + V_{N-k+1}(f(x, u, w)) \right\}, \quad V_N(x) \equiv 0
\]

- Define \( J_k(x) = V_{N-k}(x)/\alpha^{N-k} \) to obtain Eq. (VI)

\( J^* \) satisfies Bellman’s equation: Take the limit in Eq. (VI)

\[
J^*(x) = \min_{u \in U(x)} E_w \left\{ g(x, u, w) + \alpha J^*(f(x, u, w)) \right\}, \quad \text{for all } x
\]

Optimality condition: Let \( \mu^*(x) \) attain the min in the Bellman equation for all \( x \)

The policy \( \{\mu^*, \mu^*, \ldots\} \) is optimal. (This type of policy is called stationary.)
### Infinite Horizon Problems - The Two Algorithms

**Value iteration (VI):** Generates finite horizon opt. cost function sequence \( \{J_k\} \)

\[
J_k(x) = \min_{u \in U(x)} E_w \left\{ g(x, u, w) + \alpha J_{k-1}(f(x, u, w)) \right\}, \quad J_0 \text{ is "arbitrary" (?) }
\]

**Policy Iteration (PI):** Generates sequences of policies \( \{\mu^k\} \) and their cost functions \( \{J_{\mu^k}\} \); \( \mu^0 \) is "arbitrary"

The typical iteration starts with a policy \( \mu \) and generates a new policy \( \bar{\mu} \) in two steps:

- **Policy evaluation step**, which computes \( J_{\mu} \) the cost function of the (base) policy \( \mu \)
- **Policy improvement step**, which computes the improved (rollout) policy \( \bar{\mu} \) using the one-step lookahead minimization

\[
\bar{\mu}(x) \in \arg \min_{u \in U(x)} E_w \left\{ g(x, u, w) + \alpha J_{\mu}(f(x, u, w)) \right\}
\]

**There are several options for policy evaluation to compute** \( J_{\mu} \)

- Solve Bellman’s equation for \( \mu \) \[ J_{\mu}(x) = E\{g(x, \mu(x), w) + \alpha J_{\mu}(f(x, \mu(x), w))\} \] by using VI or other method (it is linear in \( J_{\mu} \))
- Use simulation (on-line Monte-Carlo, Temporal Difference (TD) methods)
Important facts (to be discussed later):

- PI yields in the limit an optimal policy (\( J^* \))
- PI is faster than VI; can be viewed as Newton’s method for solving Bellman’s Eq.
- PI can be implemented approximately, with a value and (perhaps) a policy network
Deterministic Linear Quadratic Problem - Infinite Horizon, Undiscounted

Linear system \( x_{k+1} = ax_k + bu_k \); quadratic cost per stage \( g(x, u) = qx^2 + ru^2 \)

Bellman equation: \( J(x) = \min_u \{ qx^2 + ru^2 + J(ax + bu) \} \)

Take the limit as \( N \to \infty \) in the \( N \)-step horizon results: \( K_k \to K^*, L_k \to L^* \)

- \( J^*(x) = K^* x^2 \) where \( K^* \) is some positive scalar
- The optimal policy has the form \( \mu^*(x) = L^* x \) where \( L^* \) is some scalar
- To characterize \( K^* \) and \( L^* \), we plug \( J(x) = Kx^2 \) into the Bellman equation
  \[ Kx^2 = \min_u \{ qx^2 + ru^2 + K(ax + bu)^2 \} = \cdots = F(K)x^2 \]
  where \( F(K) = \frac{a^2 rK + q}{r + b^2 K} \) with the minimizing \( u \) being equal to \(-\frac{abK}{r + b^2 K}x\)
- Thus the Bellman equation is solved by \( J^*(x) = K^* x^2 \), with \( K^* \) being a solution of the Riccati equation
  \[ K^* = F(K^*) = \frac{a^2 rK^*}{r + b^2 K^*} + q \]
  and the optimal policy is linear:
  \[ \mu^*(x) = L^* x \quad \text{with} \quad L^* = -\frac{abK^*}{r + b^2 K^*} \]
Graphical Solution of the Riccati Equation

\[ F(K) = \frac{a^2 r K}{r + b^2 K} + q \]

Riccati Equation: \( K = F(K) \)

from

Bellman Equation on
Space of Quadratic Functions

\[ J(x) = K x^2 \]
Visualization of VI

\[ F(K) = \frac{a^2 r K}{r + b^2 K} + q \]

Value Iteration:
\[ K_{k+1} = F(K_k) \]
from
\[ J_{k+1}(x) = K_{k+1} x^2 = F(K_k) x^2 \]
About the Next Lecture

Linear quadratic problems and Newton step interpretations
- Approximation in value space as a Newton step for solving the Riccati equation
- Rollout as a Newton step starting from the cost of the base policy
- Policy Iteration as repeated Newton steps

Problem formulations and reformulations
- How do we formulate DP models for practical problems?
- Problems involving a terminal state (stochastic shortest path problems)
- Problem reformulation by state augmentation (dealing with delays, correlations, forecasts, etc)
- Problems involving imperfect state observation (POMDP)
- Multiagent problems - Nonclassical information patterns
- Systems with unknown or changing parameters - Adaptive control

PLEASE READ SECTIONS 1.5-1.6 OF THE CLASS NOTES (as much as you can)

1ST HOMEWORK (DUE IN ONE WEEK): Exercise 1.1 of the Class Notes