Reinforcement Learning and Optimal Control

ASU, CSE 691, Winter 2019

Dimitri P. Bertsekas
dimitrib@mit.edu

Lecture 9
Outline

1. Review of Exact and Approximate Policy Iteration
2. Approximate PI with Parametric Approximation
3. Q-Learning
4. Least Squares Training and Simulation-Based Projection
Main Results - Discounted Problems

Infinite horizon discounted problems: States $i$, controls $u \in U(i)$, transition probs $p_{ij}(u)$, cost per stage $g(i, u, j)$, discount factor $\alpha < 1$

Bellman’s equation for optimal cost $J^*$ and policy cost $J_\mu$

$$J^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) (g(i, u, j) + \alpha J^*(j)),$$

$$J_\mu(i) = \sum_{j=1}^{n} p_{ij} (\mu(i)) \left( g(i, \mu(i), j) + \alpha J_\mu(j) \right)$$

Value iteration convergence for optimal cost and policy cost

$$J_{k+1}(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) (g(i, u, j) + \alpha J_k(j)), \quad J_k \to J^*$$

$$J_{k+1}(i) = \sum_{j=1}^{n} p_{ij} (\mu(i)) \left( g(i, \mu(i), j) + \alpha J_k(j) \right), \quad J_k \to J_\mu$$

Optimality condition

$\mu$ is optimal if and only if it attains the min in Bellman’s equation
Given the current policy $\mu^k$, a PI consists of two phases:

- **Policy evaluation** computes $J_{\mu^k}(i)$, $i = 1, \ldots, n$, as the solution of the (linear) Bellman equation system

$$J_{\mu^k}(i) = \sum_{j=1}^{n} p_{ij}(\mu^k(i)) \left( g(i, \mu^k(i), j) + \alpha J_{\mu^k}(j) \right), \quad i = 1, \ldots, n$$

- **Policy improvement** then computes a new policy $\mu^{k+1}$ as

$$\mu^{k+1}(i) \in \arg \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J_{\mu^k}(j) \right), \quad i = 1, \ldots, n$$

**Optimistic PI**: Like standard PI, but policy evaluation uses a finite number of VI.
Approximation in Value Space for Infinite Horizon Problems

Approximate minimization

\[
\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}(j))
\]

First Step "Future"

Approximations:
- Replace \( E\{\cdot\} \) with nominal values (certainty equivalence)
- Adaptive simulation
- Monte Carlo tree search

Computation of \( \tilde{J} \):
- Problem approximation
- Rollout
- Approximate PI
- Parametric approximation
- Aggregation

We focus on rollout, and particularly on approximate PI schemes, which operate as follows:

- Several policies \( \mu^0, \mu^1, \ldots, \mu^m \) are generated, starting with an initial policy \( \mu^0 \).
- Each policy \( \mu^k \) is evaluated approximately, with a cost function \( \tilde{J}_{\mu^k} \), often with the use of a parametric approximation/neural network approach.
- The next policy \( \mu^{k+1} \) is generated by policy improvement based on \( \tilde{J}_{\mu^k} \).
- The approximate evaluation \( \tilde{J}_{\mu^m} \) of the last policy in the sequence is used as the lookahead approximation \( \tilde{J} \) in a one-step or multistep lookahead minimization.
-step lookahead, truncated rollout, terminal cost approximation

- *-step lookahead, then rollout with policy $\mu$ for a limited number of steps, and finally a terminal cost approximation $\tilde{J}$.

Without terminal approximation, this is a single PI combined with multistep lookahead.

With a terminal approximation, this is a single optimistic PI combined with multistep lookahead.

Performance bounds: They improve as $\ell$ increases and $\tilde{J} \approx J^*$ (within a constant shift).
Approximate (Nonoptimistic) Policy Iteration - Performance Bound

Typical Behavior: Oscillations within an error zone

- "Size" of the zone depends on the "approximation quality" of policy evaluation ($\delta$) and policy improvement ($\epsilon$).
- When the generated policies converge, the performance bound is better.
Introduce a differentiable parametric architecture $\tilde{J}_\mu(i, r)$ for policy evaluation

- **Examples**: A linear feature-based architecture or a neural net.
- **Approximate policy evaluation/training**: Generate state-cost pairs $(i^s, \beta^s)$, where $\beta^s$ is a sample cost corresponding to $i^s$. Use least squares/regression:

  $$\bar{r} \in \arg \min_r \sum_{s=1}^q (\tilde{J}_\mu(i^s, r) - \beta^s)^2$$

- $\beta^s$ is generated by simulating a trajectory that starts at $i^s$, using $\mu$ for some number $N$ of stages, accumulating the corresponding discounted costs, and adding a terminal cost approximation $\alpha^N \tilde{J}(i_N)$.
The training problem

\[
\bar{r} \in \arg \min \sum_{s=1}^{q} \left( \tilde{J}_\mu(i^s, r) - \beta^s \right)^2
\]

is well-suited for incremental gradient:

\[
r^{k+1} = r^k - \gamma^k \nabla \tilde{J}_\mu(i^{s_k}, r^k) \left( \tilde{J}_\mu(i^{s_k}, r^k) - \beta^{s_k} \right)
\]

where \((i^{s_k}, \beta^{s_k})\) is the state-cost sample pair that is used at the \(k\)th iteration.

- **Trajectory reuse**: Given a long trajectory \((i_0, i_1, \ldots, i_N)\), we can obtain cost samples for all the states \(i_0, i_1, i_2, \ldots\), by using the tail portions of the trajectory.

- **Exploration**: When evaluating \(\mu\) with trajectory reuse, we generate many cost samples that start from states frequently visited by \(\mu\). Then the cost of underrepresented states may be estimated inaccurately, causing potentially serious errors in the calculation of the improved policy \(\bar{\mu}\).

- **Bias-variance tradeoff**: As the trajectory length \(N\) increases, the cost samples \(\beta^s\) become more accurate but also more “noisy.”

- **Cost shaping**: Replace \(g(i, u, j)\) with \(\hat{g}(i, u, j) = g(i, u, j) + \alpha V(j) - V(i)\), to approximate \(J_\mu - V\) rather than \(J_\mu\). Suboptimal policies depend on \(V\), and \(V\) can capture much of the “nonlinearity” in \(J_\mu\). Allows the use of enhanced approximations.
A Working Break: Think About Exploration in Approximate PI

How would you introduce both exploration and trajectory reuse into policy evaluation?

- What kind of schemes would diversify the cost samples of a given policy $\mu$?
- How would they work for deterministic problems?
- How would they work if we estimate Q-factors?

Answer: Many starting states, short trajectories, terminal cost approximation, use of an "off-policy".
Approximate PI Schemes with Q-Factors

![Diagram of approximate PI scheme]

Introduce a parametric architecture $\tilde{Q}_\mu(i, u, r)$ for Q-factor evaluation

- **Approximate policy evaluation/training**: Generate training triplets $(i^s, u^s, \beta^s)$, where $\beta^s$ is a sample Q-factor corresponding to $(i^s, u^s)$. Use least squares/regression:

  $$\bar{r} \in \arg \min_r \sum_{s=1}^{q} (\tilde{Q}_\mu(i^s, u^s, r) - \beta^s)^2$$

- $\beta^s$ is generated by simulating a trajectory that starts at $(i^s, u^s)$, using $\mu$ for some number $N$ of stages, accumulating the corresponding discounted costs, and adding a terminal cost approximation $\alpha^N \hat{J}(i_N)$. 
Approximate PI with Approximation in Policy Space on Top of Approximation in Value Space

Trajectory Reuse and Exploration Issues

- Trajectory reuse is more problematic in Q-factor evaluation than in cost evaluation; each trajectory generates state-control pairs of the special form \((i, \mu(i))\) at every stage after the first, so pairs \((i, u)\) with \(u \neq \mu(i)\) are not adequately explored.
- For this reason, it is necessary to make an effort to include in the samples a rich enough set of trajectories that start at pairs \((i, u)\) with \(u \neq \mu(i)\).
- An alternative approach: First compute in model-free fashion a cost function approximation \(\tilde{J}_\mu(j, \bar{r})\), and then use a second sampling process and regression to approximate further the (already approximate) Q-factor

\[
\sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \tilde{J}_\mu(j, \bar{r})),
\]

with some \(\tilde{Q}_\mu(i, u, \bar{r})\) possibly obtained with a policy approximation architecture.
- This is model-free approximate PI that is based on approximation in policy space on top of approximation in value space. It is more complex, but allows trajectory reuse and thus deals better with the exploration issue.
Q-Learning with Lookup Table Representation

Recall the VI Algorithm for Q-Factors $Q_{k+1} = FQ_k$ where $F$ is the operator

$$(FQ)(i, u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \min_{v \in U(j)} Q(j, v) \right), \quad \text{for all } (i, u)$$

$F$ is a contraction with modulus $\alpha$, so VI converges to $Q^*$.

Q-Learning is a simulation-based VI algorithm for Q-factors, based on “asynchronous DP” ideas [iterate on a single pair $(i, u)$ at a time]

- An infinitely long sequence of state-control pairs $\{(i^k, u^k)\}$ is generated according to some (essentially arbitrary) probabilistic mechanism.
- For each pair $(i^k, u^k)$, a state $j^k$ is generated according to the probabilities $p_{ikj}(u^k)$.
- Then only the Q-factor of $(i^k, u^k)$ is updated using a stepsize $\gamma^k \in (0, 1]$; all other Q-factors are left unchanged:

$$Q_{k+1}(i, u) = (1 - \gamma^k)Q_k(i, u) + \gamma^k(F_kQ_k)(i, u), \quad \text{for all } (i, u),$$

where $(F_kQ_k)(i, u) = Q_k(i, u)$ if $(i, u) \neq (i^k, u^k)$, and

$$(F_kQ_k)(i^k, u^k) = g(i^k, u^k, j^k) + \alpha \min_{v \in U(j^k)} Q_k(j^k, v) \quad \text{if } (i, u) = (i^k, u^k)$$

- To guarantee convergence some technical conditions are needed, e.g., $\gamma^k \to 0$. 
Introduce a linear parametric architecture \( \tilde{Q}(i, u, r) = \phi(i, u)'r \), and iterate on \( r \). Each value of \( r \) defines a policy, which generates controls. As \( r \) is iterated on the policy changes.

**SARSA:** At iteration \( k \), we have \( r^k, i^k \), and we have chosen a control \( u^k \)

- We simulate the next transition \((i^k, i^{k+1})\) using the transition probabilities \( p_{i^k j}(u^k) \).
- We generate \( u^{k+1} \) with the minimization \( u^{k+1} \in \arg\min_{u \in U(i^{k+1})} \tilde{Q}(i^{k+1}, u, r^k) \) [In some schemes, \( u^{k+1} \) is chosen with a small probability to be a different element of \( U(i^{k+1}) \) to enhance exploration.]
- We update the parameter vector via

\[
    r^{k+1} = r^k - \gamma^k \phi(i^k, u^k) q_k,
\]

where \( \gamma^k \) is a positive stepsize, and \( q_k \) is given by

\[
    q_k = \phi(i^k, u^k)'r^k - \alpha \phi(i^{k+1}, u^{k+1})'r^k - g(i^k, u^k, i^{k+1})
\]

- The vector \( \phi(i^k, u^k)q_k \) can be interpreted as an approximate gradient direction, and \( q_k \) is referred to as a temporal difference.
A Projection View of Approximate Policy Evaluation

- Approximation of solution of Bellman’s equation $J_\mu = T_\mu J_\mu$ with a parametric architecture amounts to replacing $J_\mu$ with a vector in
  \[ \mathcal{M} = \left\{ (\tilde{J}(1, r), \ldots, \tilde{J}(n, r)) \mid \text{all parameter vectors } r \right\} \]

- A common approach uses projection onto $\mathcal{M}$:
  \[ \Pi(J) \in \arg\min_{V \in \mathcal{M}} \| J - V \|^2 \]
  where
  \[ \| J \|^2 = \sum_{i=1}^{n} \xi_i (J(i))^2, \]
  where $J(i)$ are the components of $J$, and $\xi_i$ are some positive weights.

Three general approaches for approximation of $J_\mu$ using projection

- Project $J_\mu$ onto $\mathcal{M}$ to obtain $\Pi(J_\mu)$, which is used in place of $J_\mu$.
- Start with some approximation $\hat{J}$ of $J_\mu$, perform $N$ VIs to obtain $T_\mu^N \hat{J}$, and project onto $\mathcal{M}$ to obtain $\Pi(T_\mu^N \hat{J})$. We then use $\Pi(T_\mu^N \hat{J})$ in place of $J_\mu$.
- Solve a projected equation $J_\mu = \Pi(T_\mu J_\mu)$, and use the solution in place of $J_\mu$. 
Approximate Projection by Monte-Carlo Simulation

- We focus on the case where the manifold \( \mathcal{M} \) is a subspace \( \mathcal{M} = \{ \Phi r \mid r \in \mathbb{R}^m \} \) where \( \Phi \) is an \( n \times m \) matrix with rows denoted by \( \phi(i)' \), \( i = 1, \ldots, n \).

- The projection \( \Pi(J) \) is of the form \( \Phi r^* \), where

\[
 r^* \in \arg \min_{r \in \mathbb{R}^m} \| \Phi r - J \|_2^2 = \arg \min_{r \in \mathbb{R}^m} \sum_{i=1}^{n} \xi_i (\phi(i)' r - J(i))^2
\]

- This minimization can be done in closed form,

\[
 r^* = \left( \sum_{i=1}^{n} \xi_i \phi(i)\phi(i)' \right)^{-1} \sum_{i=1}^{n} \xi_i \phi(i)J(i)
\]

View the two terms as expectations and approximate them by MC simulation

- Generate samples \( i^s, s = 1, \ldots, q \), according to \( \xi \), and form the estimates

\[
 \frac{1}{q} \sum_{s=1}^{q} \phi(i^s)\phi(i^s)' \approx \sum_{i=1}^{n} \xi_i \phi(i)\phi(i)', \quad \frac{1}{q} \sum_{t=1}^{q} \phi(i^s)\beta^s \approx \sum_{i=1}^{n} \xi_i \phi(i)J(i)
\]

where \( \beta^s \) is a sample of \( J(i^s) \) plus a “zero mean noise” term \( n(i^s) \) (see the text).

- Estimate \( r^* \) by

\[
 \bar{r} = \left( \sum_{t=1}^{q} \phi(i^s)\phi(i^s)' \right)^{-1} \sum_{t=1}^{q} \phi(i^s)\beta^s
\]
Connection with Least Squares

The solution of the simulation-based approximate projection

\[ \bar{r} = \left( \sum_{t=1}^{q} \phi(i^s)\phi(i^s)' \right)^{-1} \sum_{t=1}^{q} \phi(i^s)\beta^s \]

is also obtained by the least squares minimization

\[ \bar{r} \in \arg \min_{r \in \mathbb{R}^m} \sum_{s=1}^{q} (\phi(i^s)'r - \beta^s)^2 \]

Thus simulation-based projection can be implemented in two equivalent ways

- Replacing expected values in the exact projection formula by simulation-based estimates.
- Replacing the exact least squares/projection problem with a simulation-based least squares approximation.

It is not necessary that the simulation produces independent samples.
It is sufficient that the long term empirical frequencies by which the indices \( i \) appear in the simulation sequence are consistent with the probabilities \( \xi_i \).
We do not need the probabilities \( \xi_i \) (the simulation determines them implicitly).
About the Next Lecture

We will cover:

- More on parametric approximation methods
- Exact and approximate linear programming
- Approximation in policy space

PLEASE READ AS MUCH OF SECTIONS 4.9-4.11 AS YOU CAN
PLEASE DOWNLOAD THE LATEST VERSIONS FROM MY WEBSITE