Introduction to Approximate Dynamic Programming

Dan Zhang
Leeds School of Business
University of Colorado at Boulder

Key References

- Bertsekas, D.P. 2011. Chapter 6, Approximate Dynamic Programming, Dynamic Programming and Optimal Control, 3rd Edition, Volume II. Available online at http://web.mit.edu/dimitrib/www/dpchapter.pdf.
- Bertsekas, D.P., J.N. Tsitsiklis. 1996. Neuro-Dynamic Programming. Athena Scientific, Belmont, MA.
- Powell, W. 2007. Approximate Dynamic Programming: Solving the Curses of Dimensionality. Wiley-Interscience.

Outline

- Setup: infinite horizon discounted MDPs
- Policy evaluation via Monte Carlo simultion
- Q-Learning
- Linear programming based approximate dynamic programming
- Approximate policy iteration
- Rollout policies
- State aggregation

Setup

- Infinite-horizon discounted MDP
- Transition probability $p_{ij}(u)$
- Cost: $g(i, u, j)$ with $|g(i, u, j)| < \infty$
- Discounting: $\alpha \in [0, 1)$.
- Discrete state space: S is finite
- Action space U.

- Let $\pi \in \Pi^{MD}$. States visited under policy π follows a Markov chain with transition probability $P^{\pi} = \{p_{ij}(\pi(i)) : i, j \in S\}$.
- Total discounted cost

$$J^{\pi}(i) = \lim_{T \to \infty} \mathrm{E}\left\{ \sum_{t=0}^{T} \alpha^t g(i_t^{\pi}, \pi(i_t^{\pi}), i_{t+1}^{\pi}) \right\}.$$

- An optimal policy can be computed by solving the optimality equations:

$$J(i) = \min_{u \in U(i)} \left\{ \sum_{j \in S} p_{ij}(u)[g(i,u,j) + \alpha J(j)] \right\}.$$

Value Iteration

- The optimality equation can be written as

$$J = TJ,$$

where

$$[TJ](i) = \min_{u \in U(i)} \left\{ \sum_{j \in S} p_{ij}(u)[g(i,u,j) + \alpha J(j)] \right\}.$$

- The value function J^* is a fixed point of the operator T.
  - Under suitable technical conditions, $J^* = \lim_{k \to \infty} T^k J^1$ for an initial vector J^1.

Policy Iteration

- Define

$$[T_\pi J](i) = \sum_{j \in S} p_{ij}(\pi(i))[g(i, \pi(i), j) + \alpha J(j)].$$

It can be shown that the discounted cost J^π incurred by policy π is a fixed point of the operator T_π; i.e., $J^\pi = T_\pi J^\pi$.

Policy Iteration (Continued)

- Policy evaluation: Compute the discounted cost J^{π^k} incurred by policy π^k, possibly by solving the system of linear equations $J^{\pi^k} = g^{\pi^k} + \alpha P^{\pi^k} J^{\pi^k}$.
- Policy improvement: For all $i \in S$, let policy π^{k+1} be defined such that we have

$$\pi^{k+1}(i) = \underset{u \in U(i)}{\operatorname{argmin}} \sum_{j \in S} p_{ij}(u)[g(i,u,j) + \alpha J^{\pi^k}(j)].$$

- Stop if $\pi^k = \pi^{k+1}$.

Three Curses of Dimensionality (Powell, 2007)

- State space is large: Computing and storing the value function $v(\cdot)$ can be difficult for both value iteration and policy iteration algorithms
- Action space is large: Both value iteration and policy evaluation become difficult
- Computing expectations with respect to the transition probability matrix can be difficult when the system dynamics is complex

Policy Evaluation with Monte Carlo Simulation

- Policy evaluation requires solving linear equations of the form $J^\pi = g^\pi + \alpha P^\pi J^\pi$.
  - Difficult when P is large or unknown

- Idea: Simulate the sequence of state $\{i_0, i_1, \dots\}$ by following a particular policy π. An approximation J of J^π can be updated as follows:

$$J(i_k) \leftarrow (1-\gamma)J(i_k) + \gamma[g(i_k, \pi(i_k), i_{k+1}) + \alpha J(i_{k+1})].$$

Policy Evaluation with Monte Carlo Simulation

- Policy evaluation requires solving linear equations of the form $J^\pi = g^\pi + \alpha P^\pi J^\pi$.
  - Difficult when P is large or unknown

- Idea: Simulate the sequence of state $\{i_0, i_1, \dots\}$ by following a particular policy π. An approximation J of J^π can be updated as follows:

$$J(i_k) \leftarrow (1-\gamma)J(i_k) + \gamma[g(i_k, \pi(i_k), i_{k+1}) + \alpha J(i_{k+1})].$$

- Why does this make sense?

Policy Evaluation with Monte Carlo Simulation (continued)

- The procedure requires storing J, which can be problematic when the state space S is large.
- Idea: Use a parameterized approximation architecture:

$$\tilde{J}(i, r) = \sum_{l=1}^{L} r_l \phi_l(i),$$

where $\phi_l(\cdot)$'s are pre-specified functions ("feature functions") and r is a vector of adjustable parameters.

Policy Evaluation with Monte Carlo Simulation (continued)

- The procedure requires storing J, which can be problematic when the state space S is large.
- Idea: Use a parameterized approximation architecture:

$$\tilde{J}(i, r) = \sum_{l=1}^{L} r_l \phi_l(i),$$

where $\phi_l(\cdot)$'s are pre-specified functions ("feature functions") and r is a vector of adjustable parameters.

- How to update r?

Q-Learning

- Q-learning is based on an alternative representation of the optimality equation:

$$J(i) = \min_{u \in U(i)} Q(i,u),$$

$$Q(i,u) = \sum_{j \in S} p_{ij}(u)[g(i,u,j) + \alpha J(j)].$$

Q-Learning

- Q-learning is based on an alternative representation of the optimality equation:

$$J(i) = \min_{u \in U(i)} Q(i,u),$$

$$Q(i,u) = \sum_{j \in S} p_{ij}(u)[g(i,u,j) + \alpha J(j)].$$

- What is the interpretation of $Q(i,u)$?

Q-Learning (Continued)

- Key idea: Evaluate $Q(i, u)$ by using a stochastic approximation iteration:

$$Q(i_k, u_k) \leftarrow (1-\gamma)Q(i_k, u_k)+\gamma[g(i_k, u_k, s_k)+\alpha \min_{v \in U(s_k)} Q(s_k, v)],$$

  where the successor state s_k is sampled according to the probabilities $\{p_{i_k,j}(u_k) : j \in S\}$.

Q-Learning (Continued)

- Key idea: Evaluate $Q(i,u)$ by using a stochastic approximation iteration:

$$Q(i_k,u_k) \leftarrow (1-\gamma)Q(i_k,u_k)+\gamma[g(i_k,u_k,s_k)+\alpha \min_{v\in U(s_k)} Q(s_k,v)],$$

  where the successor state s_k is sampled according to the probabilities $\{p_{i_k,j}(u_k) : j \in S\}$.

- What is the benefit of Q-learning?

Linear Programming Approach

- Let $\theta(s)$ be positive scalars such that $\sum_{s \in S} \theta(s) = 1$.
- The linear programming formulation is given by

$$\max_J \sum_{i \in S} \theta(i) J(i)$$

$$J(i) - \sum_{j \in S} \alpha P_{ij}(u) J(j) \leq \sum_{j \in S} P_{ij}(u) g(i,u,j), \qquad \forall i \in S, u \in U(i).$$

- Dual linear program is given by

$$\min_x \sum_{i \in S} \sum_{u \in U(i)} \sum_{j \in S} p_{ij}(u) g(i,u,j) x(i,u)$$

$$\sum_{u \in U(i)} x(i,u) - \sum_{j \in S} \sum_{u \in U(j)} \alpha p_{ji}(u) x(j,u) = \theta(i), \quad \forall i \in S,$$

$$x(i,u) \geq 0, \qquad \forall i \in S, u \in U(i).$$

Linear Programming Approach (continued)

- The size of LP can be reduced by using a parameterized approximation architecture (Schweitzer and Seidmann, 1985):

$$\tilde{J}(i,r) = \sum_{l=1}^{L} r_l \phi_l(i),$$

- Two solution approaches:
  - Simulation-based approach (de Farias and Van Roy, 2003; also Powell, 2007)
  - Mathematical programming-based approach (Adelman, 2003; Adelman, 2007)

Approximate Policy Iteration

- Policy evaluation algorithm can be difficult when the problem is "large".
- Idea: Carry out policy iteration approximately.

Approximate Policy Iteration (Continued)

- Approximate policy evaluation: Simulate a sequence of states $\{i_0, i_1, \dots\}$ by following policy π^k. Let $C_l = \sum_{t=l}^{\infty} \alpha^{t-l} g(i_t, \pi^k(i_t), i_{t+1})$ for all $l = 0, 1, \dots$. Let r^k be the solution to the regression problem

$$\min_r \left\{ \sum_{t=0}^{\infty} [\tilde{J}(i_t, r) - C_t]^2 \right\} = \min_r \left\{ \sum_{t=0}^{\infty} \left[\sum_{l=1}^{L} r_l \phi_l(i_t) - C_t \right]^2 \right\}.$$

- Approximate policy improvement: For all $i \in S$, let policy π^{k+1} be defined such that we have

$$\pi^{k+1}(i) = \operatorname*{argmin}_{u \in U(i)} \sum_{j \in S} p_{ij}(u)[g(i, u, j) + \alpha \tilde{J}(j, r^k)].$$

Rollout Policy

- Idea: Improve the performance of a given policy.
- Given policy π, let π' be defined such that for each $i \in S$,

$$\pi'(i) = \underset{u \in U(i)}{\operatorname{argmin}} \sum_{j \in S} p_{ij}(u)[g(i,u,j) + \alpha J^{\pi}(j)].$$

  Then it can be shown that π' is at least as good as π.

- Simulation may be involved to estimate J^{π}.

State Aggregation

- Idea: Partition the state space into a number of subsets and assume the value function is constant over each partition.
- State aggregation can be combined with other approximation methods.

Discussion

- ADP aims at alleviating computational effort required to solve large-scale dynamic programs
  - The area is still in its infancy
  - A commonly accepted definition of ADP does not seem to exist
- Open problem: How to specify "feature functions"?
  - Research to date usually assume they are fixed in advance.
  - Recent advances: Klabjan and Adelman (2007)
- Develop efficient solution approaches for practical applications could be valuable.