Dynamic Programming and Reinforcement Learning

Lecture 4: Dimension Reduction in DP and RL

Mengdi Wang

Operations Research and Financial Engineering
Princeton University

July 25-29, 2016

Today

Infinite-Horizon DP: Theory

- Infinite horizon cost function expressions [with $J_0(x) \equiv 0$]

$$J_\pi(x) = \lim_{N\to\infty} (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_N} J_0)(x), \quad J_\mu(x) = \lim_{N\to\infty} (T_\mu^N J_0)(x)$$

- Bellman's equation:

$$J^* = TJ^*, \qquad J_\mu = T_\mu J_\mu$$

- Optimality condition:

$$\mu\text{: optimal} \quad <==> \quad T_\mu J^* = TJ^*$$

Infinite-Horizon DP: Algorithms

- Value iteration: For any (bounded) J

$$J^*(x) = \lim_{k\to\infty} (T^k J)(x), \qquad \forall\ x$$

- Policy iteration: Given μ^k,
  - Policy evaluation: Find J_{μ^k} by solving

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k}$$

  - Policy improvement : Find μ^{k+1} such that

$$T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}$$

Practical Difficulties of DP

The curse of dimensionality

- Exponential growth of the computational and storage requirements as the number of state variables and control variables increases
- Quick explosion of the number of states in combinatorial problems

The curse of modeling

- Sometimes a simulator of the system is easier to construct than a model
- There may be real-time solution constraints
- A family of problems may be addressed. The data of the problem to be solved is given with little advance notice
- The problem data may change as the system is controlled – need for on-line replanning

All of the above are motivations for approximation and simulation

General Orientation to ADP

ADP (late 80s - present) is a breakthrough methodology that allows the application of DP to problems with many or infinite number of states .

Other names for ADP are:

- "reinforcement learning" (RL).
- "neuro-dynamic programming" (NDP).
- "adaptive dynamic programming" (ADP).

We will mainly adopt an n-state discounted model (the easiest case - but think of HUGE n).

- Extensions to other DP models (continuous space, continuous-time, not discounted) are possible (but more quirky). We will set aside for later.

There are many approaches:

- Problem approximation
- Simulation-based approaches (we will focus on these)

Simulation-based methods are of three types:

- Rollout (we will not discuss further)
- Approximation in value space
- Approximation in policy space

Approximation in value space

- Approximate J^* or J_μ from a parametric class $\tilde{J}(i;r)$ where i is the current state and $r = (r_1, \ldots, r_m)$ is a vector of “tunable” scalars weights
- Use $\tilde{J}$ in place of J^* or J_μ in various algorithms and computations
- Role of r : By adjusting r we can change the “shape” of $\tilde{J}$ so that it is “close” to J^* or J_μ
- A simulator may be used, particularly when there is no mathematical model of the system (but there is a computer model)
- We will focus on simulation , but this is not the only possibility
- We may also use parametric approximation for Q-factors or cost function differences

Approximation Architectures

Two key issues:

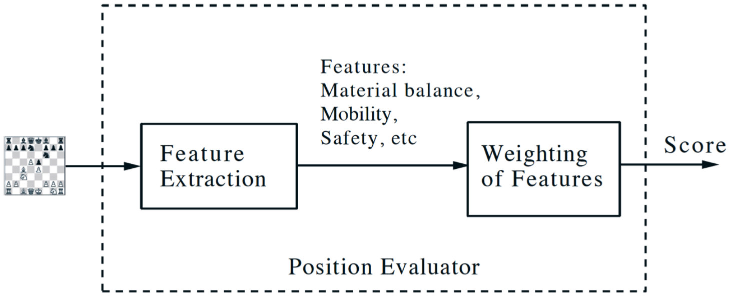
- The choice of parametric class $\tilde{J}(i; r)$ (the approximation architecture)
- Method for tuning the weights ("training" the architecture)

Success depends strongly on how these issues are handled ... also on insight about the problem

- Divided in linear and nonlinear [i.e., linear or nonlinear dependence of $\tilde{J}(i; r)$ on r]
- Linear architectures are easier to train, but nonlinear ones (e.g., neural networks) are richer

Computer chess example

- Think of board position as state and move as control
- Uses a feature-based position evaluator that assigns a score (or approximate Q-factor) to each position/move



- Relatively few special features and weights, and multistep lookahead

Linear Approximation Architectures

- With well-chosen features, we can use a linear architecture:

$$\tilde{J}(i;r) = \phi(i)'r, \qquad i = 1, \ldots, n,$$

  or

$$\tilde{J}(r) = \Phi r = \sum_{j=1}^{s} \Phi_j r_j$$

  Φ: the matrix whose rows are $\phi(i)'$, $i = 1, \ldots, n$, Φ_j is the jth column

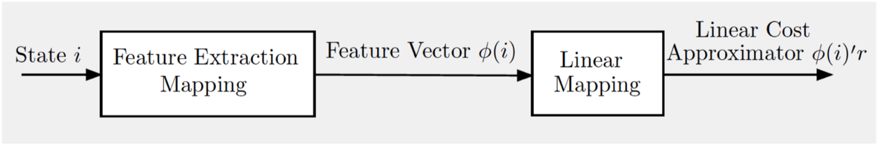
- This is approximation on the subspace

$$S = \{\Phi r \mid r \in \Re^s\}$$

  spanned by the columns of Φ (basis functions)

Linear Approximation Architectures

Often, the features encode much of the nonlinearity inherent in the cost function approximated


State i
Feature Extraction Mapping
Feature Vector $\phi(i)$
Linear Mapping
Linear Cost Approximator $\phi(i)'r$

- Many examples of feature types: Polynomial approximation, radial basis functions, etc

Example: Polynomial type

- Polynomial Approximation , e.g., a quadratic approximating function. Let the state be $i = (i_1, \ldots, i_q)$ (i.e., have q "dimensions") and define

$$\phi_0(i) = 1, \quad \phi_k(i) = i_k, \quad \phi_{km}(i) = i_k i_m, \qquad k, m = 1, \ldots, q$$

  Linear approximation architecture :

$$\tilde{J}(i; r) = r_0 + \sum_{k=1}^{q} r_k i_k + \sum_{k=1}^{q} \sum_{m=k}^{q} r_{km} i_k i_m,$$

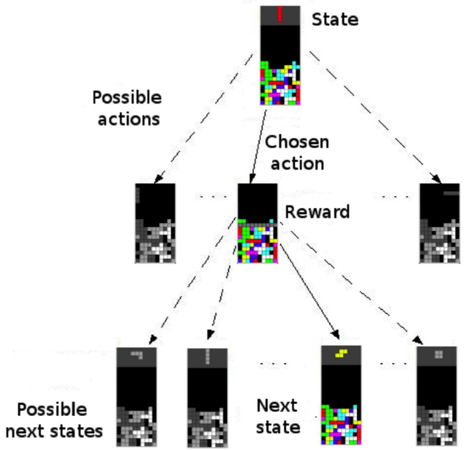
  where r has components r_0, r_k, and r_{km}.
- Interpolation : A subset I of special/representative states is selected, and the parameter vector r has one component r_i per state $i \in I$. The approximating function is

$$\tilde{J}(i; r) = r_i, \qquad i \in I,$$

$$\tilde{J}(i; r) = \text{interpolation using the values at } i \in I, \quad i \notin I$$

  For example, piecewise constant, piecewise linear, more general polynomial interpolations .

Another Example



- $J^*(i)$: optimal score starting from position i
- Number of states $> 2^{200}$ (for 10×20 board)
- Success with just 22 features, readily recognized by tetris players as capturing important aspects of the board position (heights of columns, etc)

Approximation in Policy Space

- A brief discussion; we will return to it later.
- Use parametrization $\mu(i; r)$ of policies with a vector $r = (r_1, \ldots, r_s)$. Examples:
- Polynomial, e.g., $\mu(i; r) = r_1 + r_2 \cdot i + r_3 \cdot i^2$
- Linear feature-based

$$\mu(i; r) = \phi_1(i) \cdot r_1 + \phi_2(i) \cdot r_2$$

Approximation in Policy Space

- Optimize the cost over r . For example:
- Each value of r defines a stationary policy, with cost starting at state i denoted by $\tilde{J}(i;r)$.
- Let $(p_1, \ldots, p_n)$ be some probability distribution over the states, and minimize over r

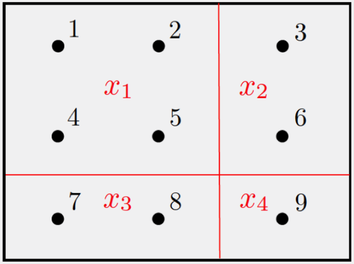
$$\sum_{i=1}^{n} p_i \tilde{J}(i;r)$$

- Use a random search, gradient, or other method
- A special case : The parameterization of the policies is indirect, through a cost approximation architecture $\hat{J}$, i.e.,

$$\mu(i;r) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \hat{J}(j;r)\big)$$

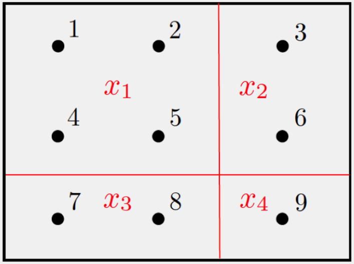
Aggregation

- A first idea : Group similar states together into "aggregate states" $x_1, \ldots, x_s$; assign a common cost value r_i to each group x_i.
- Solve an "aggregate" DP problem , involving the aggregate states, to obtain $r = (r_1, \ldots, r_s)$. This is called hard aggregation



$$\Phi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Aggregation



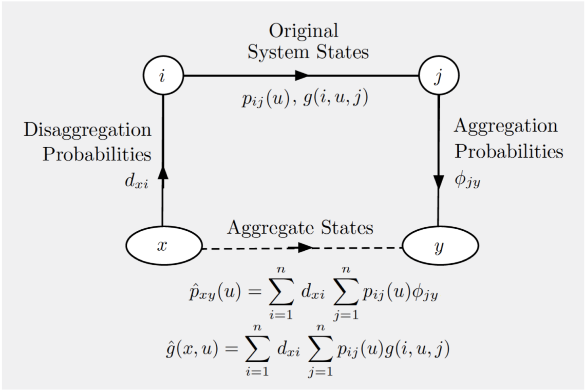
$$\Phi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- More general/mathematical view : Solve

$$\Phi r = \Phi D T_\mu(\Phi r)$$

  where the rows of D and Φ are prob. distributions (e.g., D and Φ "aggregate" rows and columns of the linear system $J = T_\mu J$)
- Compare with projected equation $\Phi r = \Pi T_\mu(\Phi r)$. Note: ΦD is a projection in some interesting cases

Aggregation as Problem Abstraction



$$\hat{p}_{xy}(u) = \sum_{i=1}^{n} d_{xi} \sum_{j=1}^{n} p_{ij}(u)\phi_{jy}$$

$$\hat{g}(x,u) = \sum_{i=1}^{n} d_{xi} \sum_{j=1}^{n} p_{ij}(u) g(i,u,j)$$

- Aggregation can be viewed as a systematic approach for problem approximation. Main elements:
- Solve (exactly or approximately) the "aggregate" problem by any kind of VI or PI method (including simulation-based methods)
- Use the optimal cost of the aggregate problem to approximate the optimal cost of the original problem

Aggregate System Description

- The transition probability from aggregate state x to aggregate state y under control u

$$\hat{p}_{xy}(u) = \sum_{i=1}^{n} d_{xi} \sum_{j=1}^{n} p_{ij}(u)\phi_{jy}, \quad \text{or } \hat{P}(u) = DP(u)\Phi$$

  where the rows of D and Φ are the disaggregation and aggregation probs.

- The expected transition cost is

$$\hat{g}(x,u) = \sum_{i=1}^{n} d_{xi} \sum_{j=1}^{n} p_{ij}(u)g(i,u,j), \quad \text{or } \hat{g} = DP(u)g$$

Aggregate Bellman's Equation

- The optimal cost function of the aggregate problem, denoted $\hat{R}$, is

$$\hat{R}(x) = \min_{u \in U} \left[\hat{g}(x,u) + \alpha \sum_{y} \hat{p}_{xy}(u) \hat{R}(y) \right], \qquad \forall\ x$$

  Bellman's equation for the aggregate problem.

- The optimal cost function J^* of the original problem is approximated by $\tilde{J}$ given by

$$\tilde{J}(j) = \sum_{y} \phi_{jy} \hat{R}(y), \qquad \forall\ j$$

Example I: Hard Aggregation

- Group the original system states into subsets, and view each subset as an aggregate state
- Aggregation probs.: $\phi_{jy} = 1$ if j belongs to aggregate state y.
- Disaggregation probs.: There are many possibilities, e.g., all states i within aggregate state x have equal prob. d_{xi}.
- If optimal cost vector J^* is piecewise constant over the aggregate states/subsets, hard aggregation is exact. Suggests grouping states with "roughly equal" cost into aggregates.
- A variant: Soft aggregation (provides "soft boundaries" between aggregate states).

Example II: Feature-Based Aggregation

- Important question: How do we group states together?
- If we know good features, it makes sense to group together states that have "similar features"

- A general approach for passing from a feature-based state representation to a hard aggregation-based architecture
- Essentially discretize the features and generate a corresponding piecewise constant approximation to the optimal cost function
- Aggregation-based architecture is more powerful (it is nonlinear in the features)
- ... but may require many more aggregate states to reach the same level of performance as the corresponding linear feature-based architecture

Example III: Rep. States/Coarse Grid

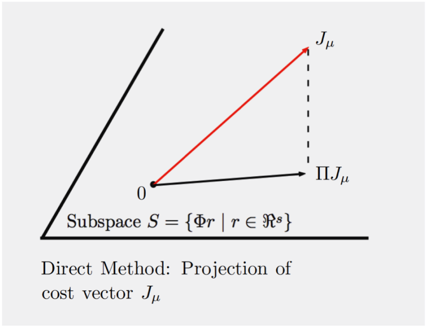
- Choose a collection of “representative” original system states, and associate each one of them with an aggregate state
- Disaggregation probabilities are $d_{xi} = 1$ if i is equal to representative state x.
- Aggregation probabilities associate original system states with convex combinations of representative states

$$j \sim \sum_{y \in \mathcal{A}} \phi_{jy} y$$

- Well-suited for Euclidean space discretization
- Extends nicely to continuous state space, including belief space of POMDP

Direct Policy evaluation

- Approximate the cost of the current policy by using least squares and simulation-generated cost samples
- Amounts to projection of J_μ onto the approximation subspace


J_μ
ΠJ_μ
0
Subspace $S = \{\Phi r \mid r \in \Re^s\}$
Direct Method: Projection of cost vector J_μ

- Solution by least squares methods
- Regular and optimistic policy iteration
- Nonlinear approximation architectures may also be used

Direct Evaluation by Simulation

- Projection by Monte Carlo Simulation: Compute the projection ΠJ_μ of J_μ on subspace $S = \{\Phi r \mid r \in \Re^s\}$, with respect to a weighted Euclidean norm $\|\cdot\|_\xi$
- Equivalently, find Φr^*, where

$$r^* = \arg \min_{r \in \Re^s} \|\Phi r - J_\mu\|_\xi^2 = \arg \min_{r \in \Re^s} \sum_{i=1}^{n} \xi_i \big(\phi(i)' r - J_\mu(i)\big)^2$$

- Setting to 0 the gradient at r^*,

$$r^* = \left(\sum_{i=1}^{n} \xi_i \phi(i) \phi(i)' \right)^{-1} \sum_{i=1}^{n} \xi_i \phi(i) J_\mu(i)$$

Direct Evaluation by Simulation

- Generate samples $\{(i_1, J_\mu(i_1)), \ldots, (i_k, J_\mu(i_k))\}$ using distribution ξ
- Approximate by Monte Carlo the two "expected values" with low-dimensional calculations

$$\hat{r}_k = \left(\sum_{t=1}^{k} \phi(i_t)\phi(i_t)' \right)^{-1} \sum_{t=1}^{k} \phi(i_t) J_\mu(i_t)$$

- Equivalent least squares alternative calculation:

$$\hat{r}_k = \arg\min_{r \in \Re^s} \sum_{t=1}^{k} \big(\phi(i_t)' r - J_\mu(i_t) \big)^2$$

Convergence of Evaluated Policy

- By law of large numbers, we have

$$\frac{1}{k}\sum_{t=1}^{k}\phi(i_t)\phi(i_t)' \xrightarrow{a.s.} \frac{1}{n}\sum_{i=1}^{n}\xi_i\phi(i)\phi(i)'$$

  and

$$\frac{1}{k}\sum_{t=1}^{k}\phi(i_t)J_\mu(i_t) \xrightarrow{a.s.} \frac{1}{n}\sum_{i=1}^{n}\xi_i\phi(i)J_\mu(i)$$

- We have

$$r_k \xrightarrow{a.s.} r^* = \Pi_S J_\mu.$$

- As the number of samples increases, the estimated low-dim cost r_k converges almost surely to the projected J_μ.

Indirect policy evaluation

- Solve the projected equation

$$\Phi r = \Pi T_\mu(\Phi r)$$

where Π is projection w/ respect to a suitable weighted Euclidean norm
- Solution methods that use simulation (to manage the calculation of Π)
- TD(λ): Stochastic iterative algorithm for solving

$$\Phi r = \Pi T_\mu(\Phi r)$$

- LSTD(λ): Solves a simulation-based approximation w/ a standard solver
- LSPE(λ): A simulation-based form of projected value iteration ; essentially

$$\Phi r_{k+1} = \Pi T_\mu(\Phi r_k) + \text{ simulation noise}$$

- Almost sure convergence guarantee

Bellman equation error methods

- Another example of indirect approximate policy evaluation:

$$\min_r \|\Phi r - T_\mu(\Phi r)\|_\xi^2 \qquad (*)$$

  where $\|\cdot\|_\xi$ is Euclidean norm, weighted with respect to some distribution ξ
- It is closely related to the projected equation/Galerkin approach (with a special choice of projection norm)
- Several ways to implement projected equation and Bellman error methods by simulation . They involve:
  - Generating many random samples of states i_k using the distribution ξ
  - Generating many samples of transitions (i_k, j_k) using the policy μ
  - Form a simulation-based approximation of the optimality condition for projection problem or problem (*) (use sample averages in place of inner products)
  - Solve the Monte-Carlo approximation of the optimality condition
- Issues for indirect methods: How to generate the samples ? How to calculate r^* efficiently ?

Cost Function Approximation via Projected Equations

Ideally, we want to solve the Bellman equation (for a fixed policy μ)

$$J = T_\mu J$$

In MDP, the equation is $n \times n$:

$$J = g_\mu + \alpha P_\mu J.$$

We solve a projected version of the high-dim equation

$$J = \Pi(g_\mu + \alpha P_\mu J)$$

Since the projection Π is onto the space spanned by Φ, the projected equation is equivalent to

$$\Phi r = \Pi(g_\mu + \alpha P_\mu \Phi r)$$

We fix the policy μ from now on, and omit mentioning it.

Matrix Form of Projected Equation

- The solution Φr^* satisfies the orthogonality condition : The error

$$\Phi r^* - (g + \alpha P \Phi r^*)$$

  is "orthogonal" to the subspace spanned by the columns of Φ.

- This is written as

$$\Phi' \Xi \big(\Phi r^* - (g + \alpha P \Phi r^*)\big) = 0,$$

  where Ξ is the diagonal matrix with the steady-state probabilities $\xi_1, \ldots, \xi_n$ along the diagonal.

- Equivalently, $Cr^* = d$, where

$$C = \Phi' \Xi (I - \alpha P) \Phi, \qquad d = \Phi' \Xi g$$

  but computing C and d is HARD (high-dimensional inner products) .

Simulation-Based Implementations

- Key idea: Calculate simulation-based approximations based on k samples

$$C_k \approx C, \qquad d_k \approx d$$

- Matrix inversion $r^* = C^{-1}d$ is approximated by

$$\hat{r}_k = C_k^{-1} d_k$$

  This is the LSTD (Least Squares Temporal Differences) Method.

- Key fact: C_k, d_k can be computed with low-dimensional linear algebra (of order s; the number of basis functions).

Simulation Mechanics

- We generate an infinitely long trajectory $(i_0, i_1, \ldots)$ of the Markov chain, so states i and transitions (i,j) appear with long-term frequencies ξ_i and p_{ij}.
- After generating each transition (i_t, i_{t+1}), we compute the row $\phi(i_t)'$ of Φ and the cost component $g(i_t, i_{t+1})$.
- We form

$$d_k = \frac{1}{k+1} \sum_{t=0}^{k} \phi(i_t) g(i_t, i_{t+1}) \approx \sum_{i,j} \xi_i p_{ij} \phi(i) g(i,j) = \Phi' \Xi g = d$$

$$C_k = \frac{1}{k+1} \sum_{t=0}^{k} \phi(i_t) \big(\phi(i_t) - \alpha \phi(i_{t+1})\big)' \approx \Phi' \Xi (I - \alpha P) \Phi = C$$

- Convergence based on law of large numbers: $C_k \xrightarrow{a.s.} C, d_k \xrightarrow{a.s.} d$. As sample size increases, r_k converges a.s. to the solution of projected Bellman equation.

Approx. PI via On-Policy Learning

Outer Loop (Off-Policy RL):

- Estimate the value function of the current policy μ_t using linear features:

$$J_{\mu_t} \approx \Phi r_t.$$

  Inner Loop (On-Policy RL):

  - Generate state trajectories ...
  - Estimate r_t via Bellman error minimization (or direct projection, or projected equation approach)

- Update the policy by

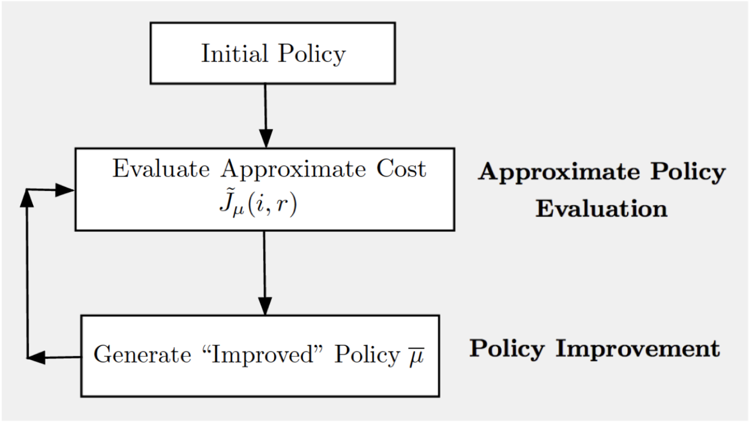
$$\mu_{t+1}(i) = \text{argmin}_a \sum_j p_{ij}(a)(g(i,a,j) + \phi(j)' r_t), \qquad \forall i.$$

Comments:

- Requires knowledge of p_{ij} (suitable for computer games with known transitions)
- The policy μ_{t+1} is parameterized by r_t.

Approx. PI via On-Policy Learning

- Use simulation to approximate the cost J_μ of the current policy μ
- Generate "improved" policy $\overline{\mu}$ by minimizing in (approx.) Bellman equation


Initial Policy
Evaluate Approximate Cost $\tilde{J}_\mu(i, r)$
Approximate Policy Evaluation
Generate "Improved" Policy $\overline{\mu}$
Policy Improvement

Alternatively we can approximate the Q-factors of μ

Theoretical Basis of Approximate PI

- If policies are approximately evaluated using an approximation architecture such that

$$\max_i |\tilde{J}(i, r_k) - J_{\mu^k}(i)| \leq d, \qquad k = 0, 1, \ldots$$

- If policy improvement is also approximate,

$$\max_i |(T_{\mu^{k+1}}\tilde{J})(i, r_k) - (T\tilde{J})(i, r_k)| \leq \epsilon, \qquad k = 0, 1, \ldots$$

- Error bound: The sequence $\{\mu^k\}$ generated by approximate policy iteration satisfies

$$\limsup_{k\to\infty} \max_i \big(J_{\mu^k}(i) - J^*(i)\big) \leq \frac{\epsilon + 2\alpha d}{(1-\alpha)^2}$$

- Typical practical behavior: The method makes steady progress up to a point and then the iterates J_{μ^k} oscillate within a neighborhood of J^*.
- In practice oscillations between policies is probably not the major concern.