We begin a lecture series on approximate DP.

Reading: Chapters 6 and 7, Vol. 2 of the text.

Today we discuss some general issues about approximation and simulation.

We classify/overview the main approaches:

- **Approximation in policy space** (policy parameterization, gradient methods, random search)
- **Approximation in value space** (approximate PI, approximate VI, Q-Learning, Bellman error approach, approximate LP)
- **Rollout/Simulation-based single policy iteration** (will not discuss this further)
- **Approximation in value space using problem approximation** (simplification - forms of aggregation - limited lookahead) - will not discuss much
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)

- 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 and 1-step lookahead
  - Simulation-based approaches (we will focus on these)

- Simulation-based methods are of three types:
  - Rollout (we will not discuss further)
  - Approximation in policy space
  - Approximation in value space
WHY DO WE USE SIMULATION?

• One reason: **Computational complexity advantage** in computing expected values and sums/inner products involving a very large number of terms
  – **Speeds up linear algebra:** Any sum $\sum_{i=1}^{n} a_i$ can be written as an expected value

\[
\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} \xi_i \frac{a_i}{\xi_i} = E_{\xi} \left\{ \frac{a_i}{\xi_i} \right\},
\]

where $\xi$ is any prob. distribution over $\{1, \ldots, n\}$

– It is approximated by generating many samples $\{i_1, \ldots, i_k\}$ from $\{1, \ldots, n\}$, according to $\xi$, and Monte Carlo averaging:

\[
\sum_{i=1}^{n} a_i = E_{\xi} \left\{ \frac{a_i}{\xi_i} \right\} \approx \frac{1}{k} \sum_{t=1}^{k} \frac{a_{i_t}}{\xi_{i_t}}
\]

– Choice of $\xi$ makes a difference. **Importance sampling** methodology.

• Simulation is also convenient when an analytical model of the system is unavailable, but a simulation/computer model is possible.
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., $\mu(i; r) = r_1 + r_2 \cdot i + r_3 \cdot i^2$
  - Multi-warehouse inventory system: $\mu(i; r)$ is threshold policy with thresholds $r = (r_1, \ldots, r_s)$
- 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.,
  \[
  \mu(i; r) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)(g(i, u, j) + \alpha \hat{J}(j; r))
  \]
APPROXIMATION IN VALUE SPACE

• Approximate $J^*$ or $J_\mu$ 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 $J_\mu$ in various algorithms and computations (VI, PI, LP).

• Role of $r$: By adjusting $r$ we can change the “shape” of $\tilde{J}$ so that it is “close” to $J^*$ or $J_\mu$.

• 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.

• A simulator may be used, particularly when there is no mathematical model of the system.

• We will focus on simulation, but this is not the only possibility.

• We may also use parametric approximation for $Q$-factors.
 APPROXIMATION ARCHITECTURES

- 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

- Often, the features encode much of the nonlinearity inherent in the cost function approximated.
- Then the approximation may be quite accurate without a complicated architecture. (Extreme example: The ideal feature is the true cost function)
- With well-chosen features, we can use a linear architecture:

\[
\tilde{J}(i; r) = \phi(i)'r, \quad \forall \ i \quad \text{or} \quad \tilde{J}(r) = \Phi r = \sum_{j=1}^{s} \Phi_j r_j
\]

\[\Phi: \text{the matrix whose rows are } \phi(i)', \ i = 1, \ldots, n, \ \Phi_j \text{ is the } j\text{th column of } \Phi\]

- This is approximation on the subspace

\[S = \{\Phi r \mid r \in \mathbb{R}^s\}\]

spanned by the columns of \(\Phi\) (basis functions)

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

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

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

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

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

For example, piecewise constant, piecewise linear, more general polynomial interpolations.
A DOMAIN SPECIFIC EXAMPLE

- **Tetris game** (used as testbed in competitions)

\[ J^*(i): \text{optimal score starting from position } i \]

- **Number of states** $> 2^{200}$ (for $10 \times 20$ board)

- Success with just 22 features, readily recognized by tetris players as capturing important aspects of the board position (heights of columns, etc)
Approx. Pi - Option to Approx. $J_\mu$ or $Q_\mu$

- Use simulation to approximate the cost $J_\mu$ of the current policy $\mu$.
- Generate “improved” policy $\bar{\mu}$ by minimizing in (approx.) Bellman equation.

Alternatively approximate the $Q$-factors of $\mu$.

DIRECTLY APPROXIMATING $J^*$ OR $Q^*$

• Approximation of the optimal cost function $J^*$ directly (without PI)
  – $Q$-Learning: Use a simulation algorithm to approximate the $Q$-factors

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

and the optimal costs

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

– Bellman Error approach: Find $r$ to

$$\min_r E_i \left\{ (\tilde{J}(i; r) - (T\tilde{J})(i; r))^2 \right\}$$

where $E_i \{ \cdot \}$ is taken with respect to some distribution over the states

– Approximate Linear Programming (we will not discuss here)

• $Q$-learning can also be used with approximations

• $Q$-learning and Bellman error approach can also be used for policy evaluation
DIRECT POLICY EVALUATION

- Can be combined with regular and optimistic policy iteration
- Find $r$ that minimizes $\| J_\mu - \tilde{J}(\cdot, r) \|_2^2$, i.e.,
  \[ \sum_{i=1}^{n} \xi_i (J_\mu(i) - \tilde{J}(i, r))^2, \quad \xi_i: \text{some pos. weights} \]
- Nonlinear architectures may be used
- The linear architecture case: Amounts to projection of $J_\mu$ onto the approximation subspace

![Diagram](image)

- Solution by linear least squares methods
POLICY EVALUATION BY SIMULATION

- Projection by Monte Carlo Simulation: Compute the projection $\Pi J_\mu$ of $J_\mu$ on subspace $S = \{ \Phi r \mid r \in \mathbb{R}^s \}$, with respect to a weighted Euclidean norm $\| \cdot \|_\xi$

- Equivalently, find $\Phi r^*$, where

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

- Setting to 0 the gradient at $r^*$,

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

- Generate samples $\{ (i_1, J_\mu(i_1)), \ldots, (i_k, J_\mu(i_k)) \}$ using distribution $\xi$

- 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:

$$\hat{r}_k = \arg\min_{r \in \mathbb{R}^s} \sum_{t=1}^k (\phi(i_t)'r - J_\mu(i_t))^2$$
INDIRECT POLICY EVALUATION

• An example: Solve the projected equation $\Phi r = \Pi T_\mu(\Phi r)$ where $\Pi$ is projection w/ respect to a suitable weighted Euclidean norm (Galerkin approx).

Direct Method: Projection of cost vector $J_\mu$

Indirect Method: Solving a projected form of Bellman’s equation

• Solution methods that use simulation (to manage the calculation of $\Pi$)
  
  − $\text{TD}(\lambda)$: Stochastic iterative algorithm for solving $\Phi r = \Pi T_\mu(\Phi r)$
  
  − $\text{LSTD}(\lambda)$: Solves a simulation-based approximation w/ a standard solver
  
  − $\text{LSPE}(\lambda)$: A simulation-based form of projected value iteration; essentially
    
    $$\Phi r_{k+1} = \Pi T_\mu(\Phi r_k) + \text{simulation noise}$$
BELLMAN EQUATION ERROR METHODS

• Another example of indirect approximate policy evaluation:

$$\min_r \| \Phi r - T\mu(\Phi r) \|_\xi^2$$  

(*)

where $\| \cdot \|_\xi$ is Euclidean norm, weighted with respect to some distribution $\xi$

• It is closely related to the projected equation 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 $\xi$

  – Generating many samples of transitions $(i_k, j_k)$ using the policy $\mu$

  – 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?
ANOTHER INDIRECT METHOD: AGGREGATION

- An example: Group similar states together into “aggregate states” $x_1, \ldots, x_s$; assign a common cost $r_i$ to each group $x_i$. A linear architecture called hard aggregation.

- Solve an “aggregate” DP problem to obtain $r = (r_1, \ldots, r_s)$.

- More general/mathematical view: Solve

$$\Phi r = \Phi DT_\mu(\Phi r)$$

where the rows of $D$ and $\Phi$ are prob. distributions (e.g., $D$ and $\Phi$ “aggregate” rows and columns of the linear system $J = T_\mu J$)

- Compare with projected equation $\Phi r = \Pi T_\mu(\Phi r)$. Note: $\Phi D$ is a projection in some interesting cases
AGGREGATION AS PROBLEM APPROXIMATION

- Aggregation can be viewed as a systematic approach for problem approx. 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

- Because an exact PI algorithm is used to solve the approximate/aggregate problem the method behaves more regularly than the projected equation approach
THEORETICAL BASIS OF APPROXIMATE PI

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

\[ \max_i |\tilde{J}(i, r_k) - J_{\mu^k}(i)| \leq \delta, \quad k = 0, 1, \ldots \]

- If policy improvement is also approximate,

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

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

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

- **Typical practical behavior:** The method makes steady progress up to a point and then the iterates \( J_{\mu^k} \) oscillate within a neighborhood of \( J^* \).

- Oscillations are quite unpredictable.
  - Bad examples of oscillations are known.
  - In practice oscillations between policies is probably not the major concern.
  - In aggregation case, there are no oscillations.
THE ISSUE OF EXPLORATION

- To evaluate a policy $\mu$, we need to generate cost samples using that policy - this biases the simulation by underrepresenting states that are unlikely to occur under $\mu$

- Cost-to-go estimates of underrepresented states may be highly inaccurate

- This seriously impacts the improved policy $\bar{\mu}$

- This is known as inadequate exploration - a particularly acute difficulty when the randomness embodied in the transition probabilities is “relatively small” (e.g., a deterministic system)

- Some remedies:
  - Frequently restart the simulation and ensure that the initial states employed form a rich and representative subset
  - Occasionally generate transitions that use a randomly selected control rather than the one dictated by the policy $\mu$
  - Other methods: Use two Markov chains (one is the chain of the policy and is used to generate the transition sequence, the other is used to generate the state sequence).
APPROXIMATING Q-FACTORS

• Given $\tilde{J}(i; r)$, policy improvement requires a model [knowledge of $p_{ij}(u)$ for all $u \in U(i)$]

• Model-free alternative: Approximate $Q$-factors

$$\tilde{Q}(i, u; r) \approx \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J_{\mu}(j) \right)$$

and use for policy improvement the minimization

$$\bar{\mu}(i) \in \arg \min_{u \in U(i)} \tilde{Q}(i, u; r)$$

• $r$ is an adjustable parameter vector and $\tilde{Q}(i, u; r)$ is a parametric architecture, such as

$$\tilde{Q}(i, u; r) = \sum_{m=1}^{s} r_{m} \phi_{m}(i, u)$$

• We can adapt any of the cost approximation approaches, e.g., projected equations, aggregation

• Use the Markov chain with states $(i, u)$, so $p_{ij}(\mu(i))$ is the transition prob. to $(j, \mu(i))$, 0 to other $(j, u')$

• Major concern: Acutely diminished exploration
Consider solution of a linear equation \( x = b + Ax \) by using \( m \) simulation samples \( b + w_k \) and \( A + W_k, k = 1, \ldots, m \), where \( w_k, W_k \) are random, e.g., “simulation noise”

Think of \( x = b + Ax \) as approximate policy evaluation (projected or aggregation equations)

Stoch. approx. (SA) approach: For \( k = 1, \ldots, m \)

\[
x_{k+1} = (1 - \gamma_k)x_k + \gamma_k((b + w_k) + (A + W_k)x_k)
\]

Monte Carlo estimation (MCE) approach: Form Monte Carlo estimates of \( b \) and \( A \)

\[
b_m = \frac{1}{m} \sum_{k=1}^{m} (b + w_k), \quad A_m = \frac{1}{m} \sum_{k=1}^{m} (A + W_k)
\]

Then solve \( x = b_m + A_m x \) by matrix inversion

\[
x_m = (1 - A_m)^{-1}b_m
\]

or iteratively

TD(\( \lambda \)) and Q-learning are SA methods

LSTD(\( \lambda \)) and LSPE(\( \lambda \)) are MCE methods