APPROXIMATE DYNAMIC PROGRAMMING

LECTURE 3

LECTURE OUTLINE

- Review of discounted DP
- Introduction to approximate DP
- Approximation architectures
- Simulation-based approximate policy iteration
- Approximate policy evaluation
- Some general issues about approximation and simulation

REVIEW

DISCOUNTED PROBLEMS/BOUNDED COST

- Stationary system with arbitrary state space

$$x_{k+1} = f(x_k, u_k, w_k), \qquad k = 0, 1, \ldots$$

- Cost of a policy $\pi = \{\mu_0, \mu_1, \ldots\}$

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

with $\alpha < 1$, and for some M, we have $|g(x, u, w)| \leq M$ for all (x, u, w)

- Shorthand notation for DP mappings (operate on functions of state to produce other functions)

$$(TJ)(x) = \min_{u\in U(x)} \underset{w}{E} \left\{ g(x, u, w) + \alpha J\big(f(x, u, w)\big) \right\}, \ \forall\, x$$

TJ is the optimal cost function for the one-stage problem with stage cost g and terminal cost αJ

- For any stationary policy μ

$$(T_\mu J)(x) = \underset{w}{E} \left\{ g\big(x, \mu(x), w\big) + \alpha J\big(f(x, \mu(x), w)\big) \right\}, \ \forall\, x$$

MDP - TRANSITION PROBABILITY NOTATION

- We will mostly assume the system is an n-state (controlled) Markov chain
- We will often switch to Markov chain notation
  - States $i = 1, \ldots, n$ (instead of x)
  - Transition probabilities $p_{i_k i_{k+1}}(u_k)$ [instead of $x_{k+1} = f(x_k, u_k, w_k)$]
  - Stage cost $g(i_k, u_k, i_{k+1})$ [instead of $g(x_k, u_k, w_k)$]
  - Cost functions $J = \big(J(1), \ldots, J(n)\big)$ (vectors in $\Re^n$)
- Cost of a policy $\pi = \{\mu_0, \mu_1, \ldots\}$

$$J_\pi(i) = \lim_{N\to\infty} \underset{\substack{i_k \\ k=1,2,\ldots}}{E} \left\{ \sum_{k=0}^{N-1} \alpha^k g\big(i_k, \mu_k(i_k), i_{k+1}\big) \mid i_0 = i \right\}$$

- Shorthand notation for DP mappings

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

$$(T_\mu J)(i) = \sum_{j=1}^{n} p_{ij}\big(\mu(i)\big)\big(g\big(i,\mu(i),j\big)+\alpha J(j)\big), \quad i = 1, \ldots, n$$

"SHORTHAND" THEORY – A SUMMARY

- Bellman's equation: $J^* = TJ^*, \quad J_\mu = T_\mu J_\mu$ or

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

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

- Optimality condition:

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

i.e.,

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

THE TWO MAIN ALGORITHMS: VI AND PI

- Value iteration: For any $J \in \Re^n$

$$J^*(i) = \lim_{k \to \infty} (T^k J)(i), \qquad \forall\ i = 1, \ldots, n$$

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

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

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

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

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

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

- Policy evaluation is equivalent to solving an $n \times n$ linear system of equations

- For large n, exact PI is out of the question. We use instead optimistic PI (policy evaluation with a few VIs)

APPROXIMATE DP

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

WHY DO WE USE SIMULATION?

- One reason: Computational complexity advantage in computing sums/expectations involving a very large number of terms
  - Any sum

$$\sum_{i=1}^{n} \boldsymbol{a}_i$$

    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 ξ is any prob. distribution over $\{1, \ldots, n\}$
  - It can be approximated by generating many samples $\{i_1, \ldots, i_k\}$ from $\{1, \ldots, n\}$, according to distribution ξ, and Monte Carlo averaging:

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

- Simulation is also convenient when an analytical model of the system is unavailable, but a simulation/computer model is possible.

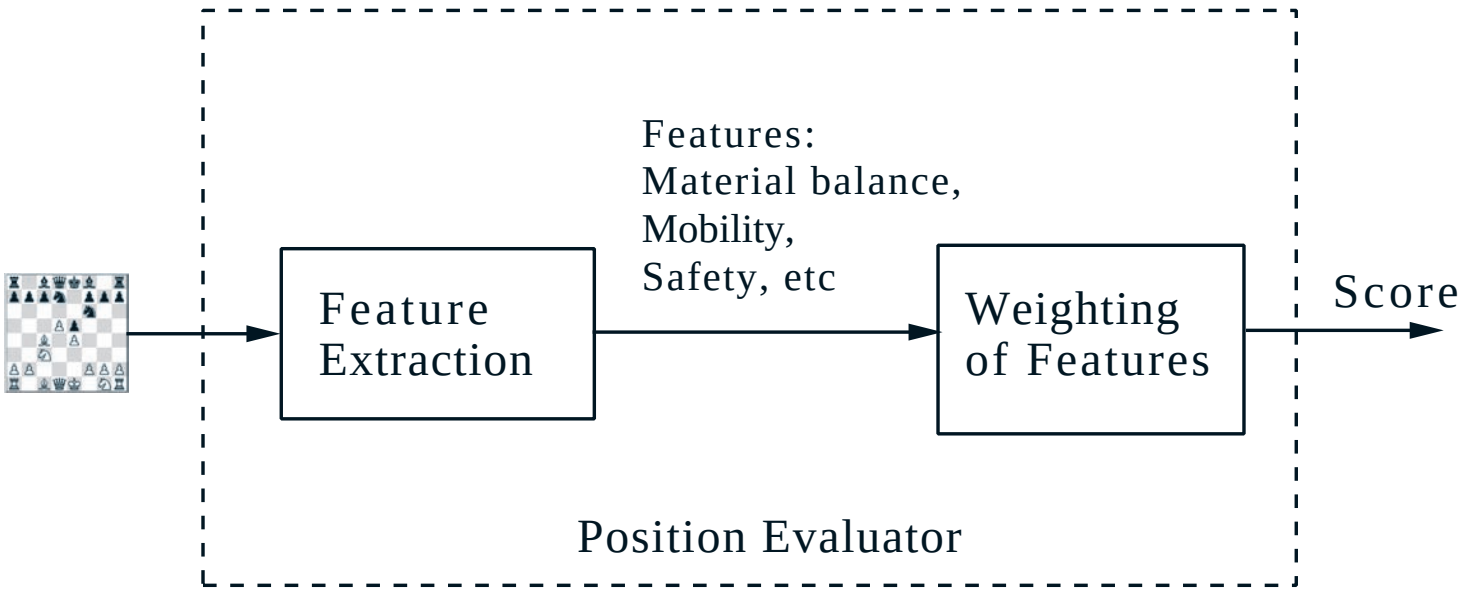
APPROXIMATION IN VALUE AND POLICY SPACE

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
- 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 (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

- 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



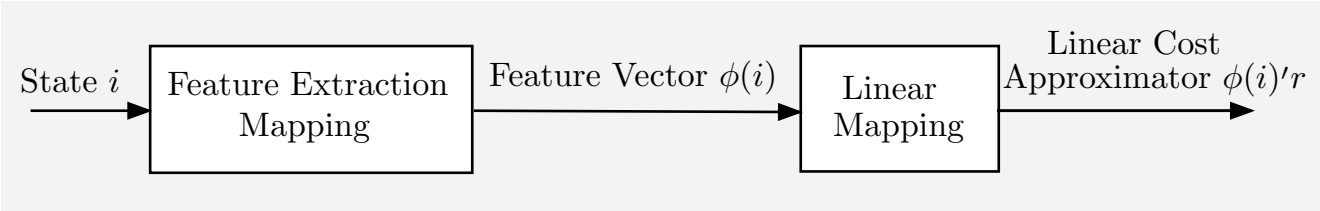
- Relatively few special features and weights, and multistep lookahead

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 (as an extreme example, the ideal feature is the true cost function)
- With well-chosen features, we can use a linear architecture: $\tilde{J}(i;r) = \phi(i)'r$, $i = 1, \ldots, n$, 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 of Φ



- This is approximation on the subspace

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

spanned by the columns of Φ (basis functions)

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

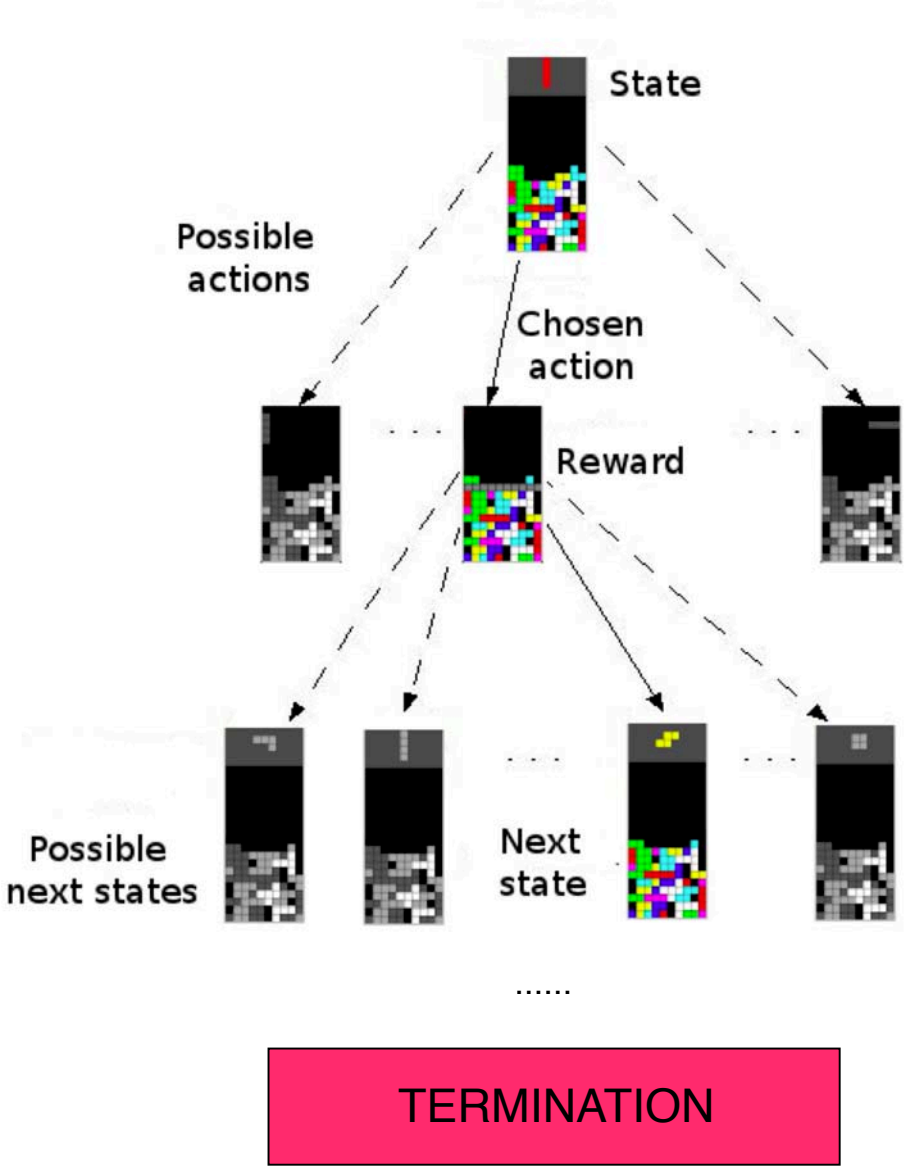
$$\tilde{J}(i; r) = r_i, \qquad i \in I,$$

$\tilde{J}(i; r) =$ 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

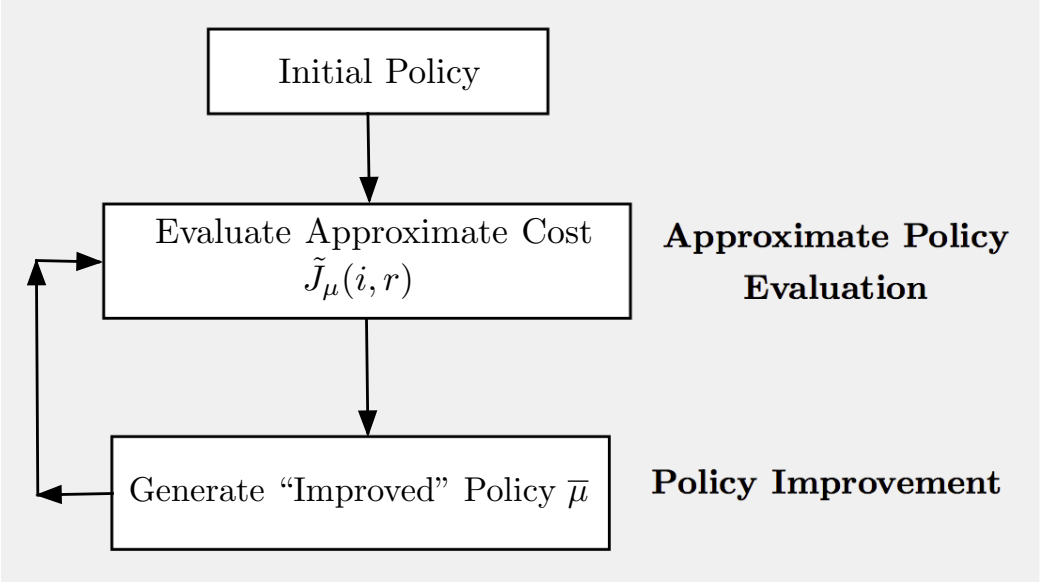
- Tetris game (used as testbed in competitions)



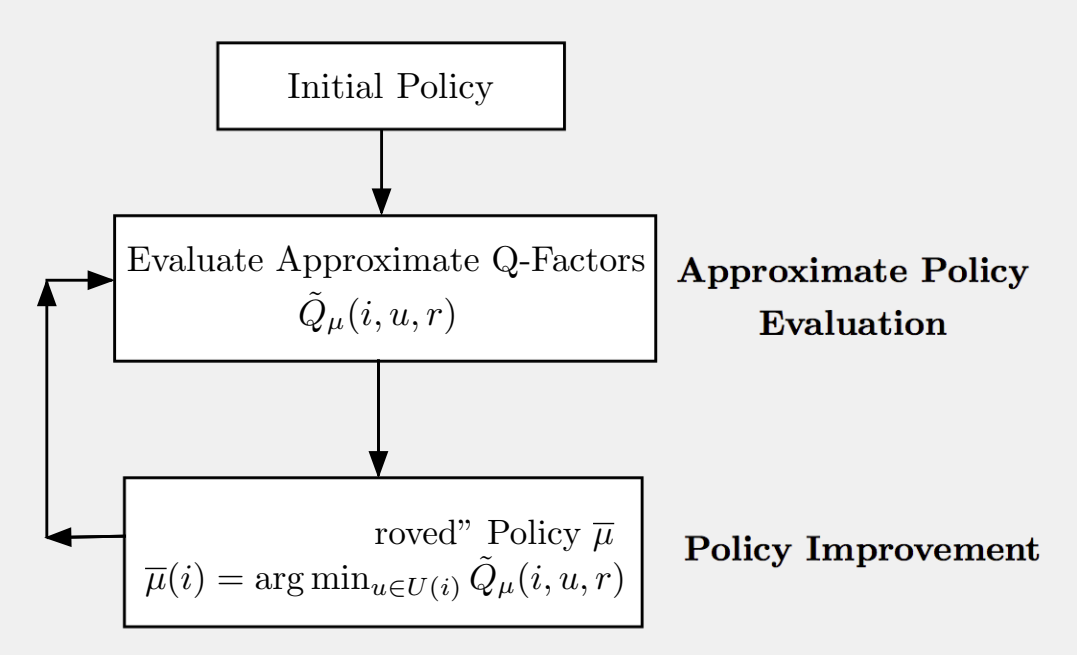
- $J^*(i)$: optimal score starting from position i
- Number of states $> 2^{200}$ (for 10×20 board)
- Success with just 22 features, readily recognized by tetris players as capturing important aspects of the board position (heights of columns, etc)

APPROX. PI - OPTION TO APPROX. J_μ OR Q_μ

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



- Altenatively approximate the Q-factors of μ



APPROXIMATING J^* OR Q^*

- Approximation of the optimal cost function J^*
  - 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

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

    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

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

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

- 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

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

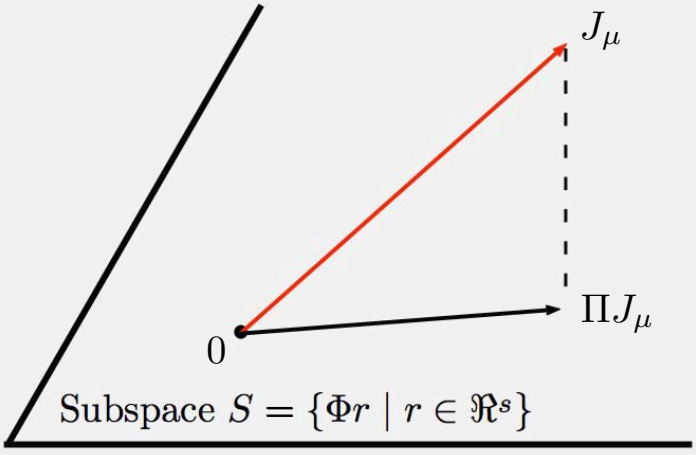
  - Use a random search, gradient, or other method
- A special case: The parameterization of the policies is indirect, through a cost approximation architecture $\hat{J}$, i.e.,

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

APPROXIMATE POLICY EVALUATION METHODS

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



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^*,

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

- 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

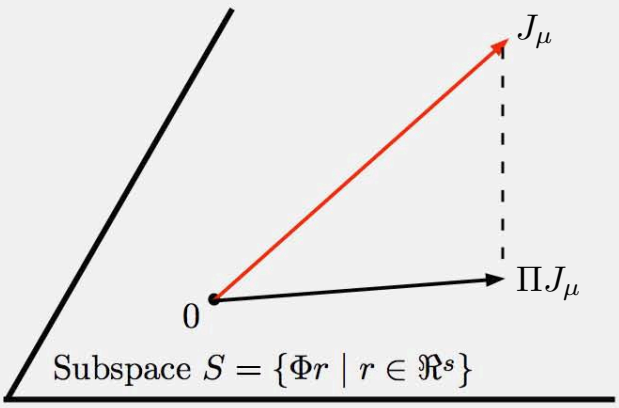
$$\hat{r}_k = \left(\sum_{t=1}^k \phi(i_t)\phi(i_t)'\right)^{-1} \sum_{t=1}^k \phi(i_t) J_\mu(i_t)$$

- Equivalent least squares alternative calculation:

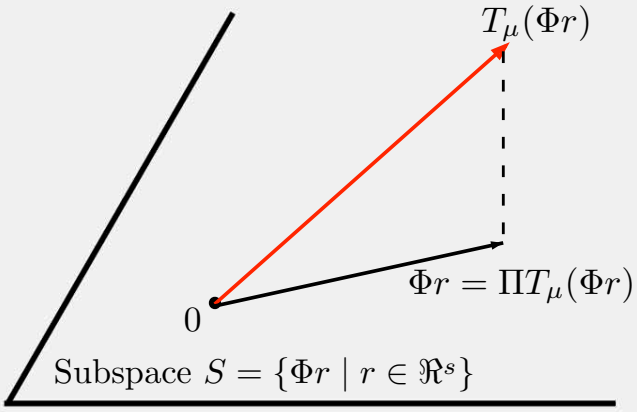
$$\hat{r}_k = \arg\min_{r\in\Re^s} \sum_{t=1}^k \big(\phi(i_t)'r - J_\mu(i_t)\big)^2$$

INDIRECT POLICY EVALUATION

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



Direct Method: Projection of cost vector J_μ



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

- 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

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

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

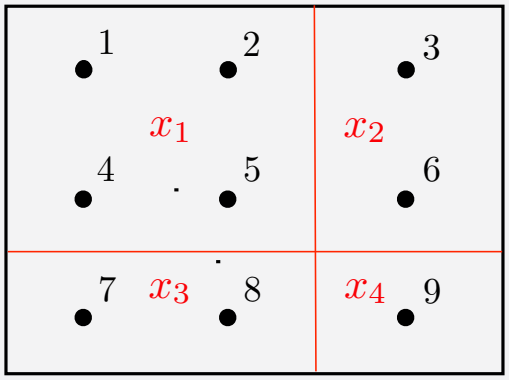
- It is closely related to the projected equation/Galerkin approach (with a special choice of projection norm)

- 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?

ANOTHER INDIRECT METHOD: 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



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

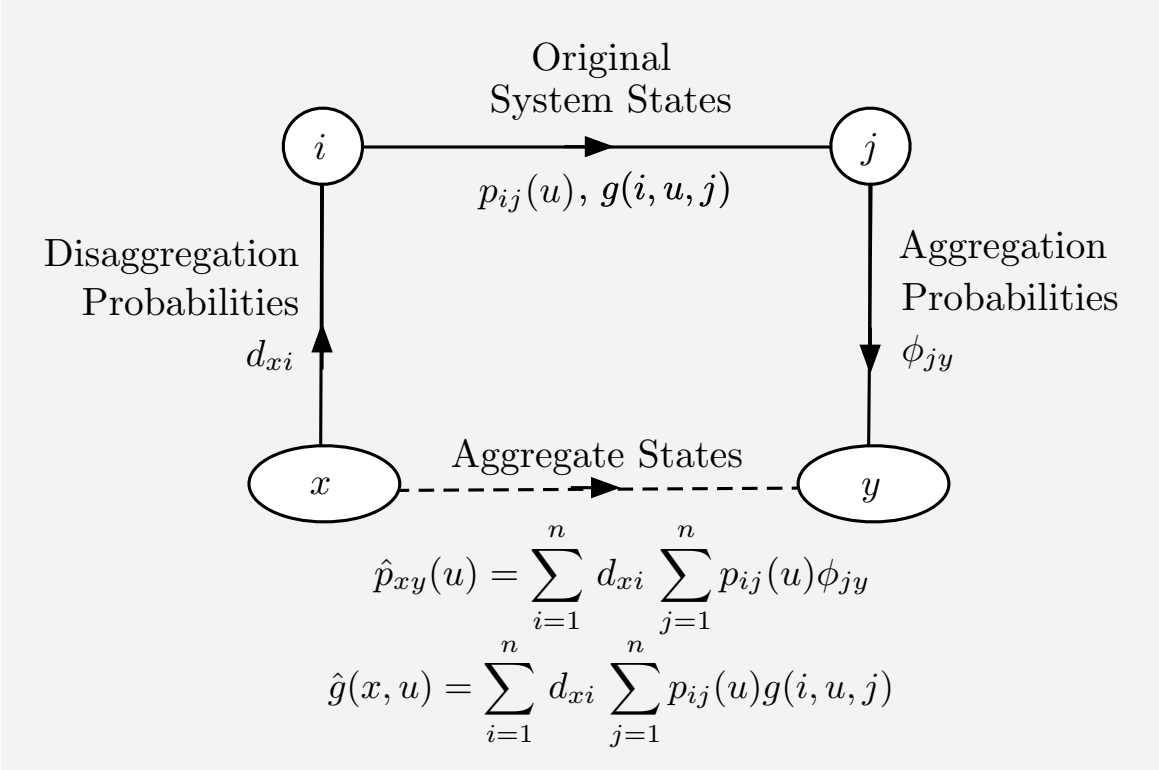
- More general/mathematical view: Solve

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

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 APPROXIMATION



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

APPROXIMATE POLICY ITERATION ISSUES

THEORETICAL BASIS OF APPROXIMATE PI

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

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

- 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

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

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

THE ISSUE OF EXPLORATION

- To evaluate a policy μ, we need to generate cost samples using that policy - this biases the simulation by underrepresenting states that are unlikely to occur under μ
- Cost-to-go estimates of underrepresented states may be highly inaccurate
- This seriously impacts the improved policy $\overline{\mu}$
- 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 μ
  - 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 controls $u \in U(i)$]
- Model-free alternative: Approximate Q-factors

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

and use for policy improvement the minimization

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

- 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

SOME GENERAL ISSUES

STOCHASTIC ALGORITHMS: GENERALITIES

- 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 $k = 1, \ldots, m$

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

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

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

Then solve $x = b_m + A_m x$ by matrix inversion

$$x_m = (1 - A_m)^{-1} b_m$$

or iteratively

- TD(λ) and Q-learning are SA methods

- LSTD(λ) and LSPE(λ) are MCE methods

COSTS OR COST DIFFERENCES?

- Consider the exact policy improvement process. To compare two controls u and u' at x, we need

$$E\ \ g(x,u,w)-g(x,u',w)+\alpha\ \ J_\mu(\overline{x})-J_\mu(\overline{x}')\big)\big\}$$

where $\overline{x}=f(x,u,w)$ and $\overline{x}'=f(x,u',w)$

- Approximate $J_\mu(\overline{x})$ or

$$D_\mu(\overline{x},\overline{x}')=J_\mu(\overline{x})-J_\mu(\overline{x}')?$$

- Approximating $D_\mu(\overline{x},\overline{x}')$ avoids "noise differencing". This can make a big difference

- Important point: D_μ satisfies a Bellman equation for a system with "state" (x,x')

$$D_\mu(x,x')=E\big\{G_\mu(x,x',w)+\alpha D_\mu(\overline{x},\overline{x}')\big\}$$

where $\overline{x}=f\big(x,\mu(x),w\big)$, $\overline{x}'=f\big(x',\mu(x'),w\big)$ and

$$G_\mu(x,x',w)=g\big(x,\mu(x),w\big)-g\big(x',\mu(x'),w\big)$$

- D_μ can be "learned" by the standard methods (TD, LSTD, LSPE, Bellman error, aggregation, etc). This is known as differential training.

AN EXAMPLE (FROM THE NDP TEXT)

- System and cost per stage:

$$x_{k+1} = x_k + \delta u_k, \qquad g(x,u) = \delta(x^2 + u^2)$$

$\delta > 0$ is very small; think of discretization of continuous-time problem involving $dx(t)/dt = u(t)$

- Consider policy $\mu(x) = -2x$. Its cost function is

$$J_\mu(x) = \frac{5x^2}{4}(1+\delta) + O(\delta^2)$$

and its Q-factor is

$$Q_\mu(x,u) = \frac{5x^2}{4} + \delta\left(\frac{9x^2}{4} + u^2 + \frac{5}{2}xu\right) + O(\delta^2)$$

- The important part for policy improvement is

$$\delta\left(u^2 + \frac{5}{2}xu\right)$$

When $J_\mu(x)$ [or $Q_\mu(x,u)$] is approximated by $\tilde{J}_\mu(x;r)$ [or by $\tilde{Q}_\mu(x,u;r)$], it will be dominated by $\frac{5x^2}{4}$ and will be "lost"

6.231 Dynamic Programming and Stochastic Control

Fall 2015