6.231 DYNAMIC PROGRAMMING

LECTURE 21

LECTURE OUTLINE

- Discounted problems - Approximate policy evaluation/policy improvement
- Direct approach - Least squares
- Batch and incremental gradient methods
- Implementation using TD
- Optimistic policy iteration
- Exploration issues

THEORETICAL BASIS

- If policies are approximately evaluated using an approximation architecture:

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

- If policy improvement is also approximate,

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

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

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

- 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^*.

SIMULATION-BASED POLICY EVALUATION

- Suppose we can implement in a simulator the improved policy $\overline{\mu}$, and want to calculate $J_{\overline{\mu}}$ by simulation.

- Generate by simulation sample costs. Then:

$$J_{\overline{\mu}}(i) \approx \frac{1}{M_i} \sum_{m=1}^{M_i} c(i,m)$$

$c(i,m)$: mth (noisy) sample cost starting from state i

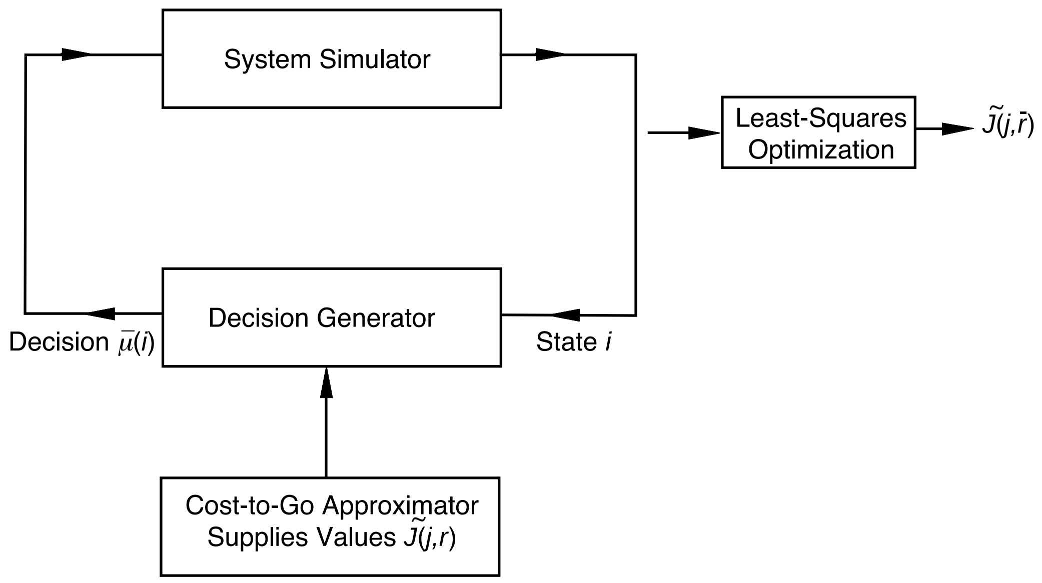
- Approximating well each $J_{\overline{\mu}}(i)$ is impractical for a large state space. Instead, a "compact representation" $\tilde{J}_{\overline{\mu}}(i,r)$ is used, where r is a tunable parameter vector.

- Direct approach: Calculate an optimal value r^* of r by a least squares fit

$$r^* = \arg\min_r \sum_{i=1}^{n} \sum_{m=1}^{M_i} \left| c(i,m) - \tilde{J}_{\overline{\mu}}(i,r) \right|^2$$

- Note that this is much easier when the architecture is linear - but this is not a requirement.

SIMULATION-BASED DIRECT APPROACH



- Simulator: Given a state-control pair (i, u), generates the next state j using system's transition probabilities under policy $\overline{\mu}$ currently evaluated

- Decision generator: Generates the control $\overline{\mu}(i)$ of the evaluated policy at the current state i

- Cost-to-go approximator: $\tilde{J}(j, r)$ used by the decision generator and corresponding to preceding policy (already evaluated in preceding iteration)

- Least squares optimizer: Uses cost samples $c(i, m)$ produced by the simulator and solves a least squares problem to approximate $\tilde{J}_{\overline{\mu}}(\cdot, \overline{r})$

BATCH GRADIENT METHOD I

- Focus on a batch: an N-transition portion $(i_0,\ldots,i_N)$ of a simulated trajectory
- We view the numbers

$$\sum_{t=k}^{N-1} \alpha^{t-k} g\big(i_t,\overline{\mu}(i_t),i_{t+1}\big), \qquad k=0,\ldots,N-1,$$

as cost samples, one per initial state $i_0,\ldots,i_{N-1}$

- Least squares problem

$$\min_{\overline{r}} \frac{1}{2}\sum_{k=0}^{N-1}\left(\tilde{J}(i_k,\overline{r}) - \sum_{t=k}^{N-1} \alpha^{t-k} g\big(i_t,\overline{\mu}(i_t),i_{t+1}\big)\right)^2$$

- Gradient iteration

$$\overline{r} := \overline{r} - \gamma \sum_{k=0}^{N-1} \nabla \tilde{J}(i_k,\overline{r}) \left(\tilde{J}(i_k,\overline{r}) - \sum_{t=k}^{N-1} \alpha^{t-k} g\big(i_t,\overline{\mu}(i_t),i_{t+1}\big)\right)$$

BATCH GRADIENT METHOD II

- Important tradeoff:
  - In order to reduce simulation error and cost samples for a representatively large subset of states, we must use a large N
  - To keep the work per gradient iteration small, we must use a small N
- To address the issue of size of N, small batches may be used and changed after one or more iterations
- Then the method becomes susceptible to simulation noise - requires a diminishing stepsize for convergence
- This slows down the convergence (which can be very slow for a gradient method even without noise)
- Theoretical convergence is guaranteed (with a diminishing stepsize) under reasonable conditions, but in practice this is not much of a guarantee

INCREMENTAL GRADIENT METHOD I

- Again focus on an N-transition portion $(i_0, \ldots, i_N)$ of a simulated trajectory.
- The batch gradient method processes the N transitions all at once, and updates $\overline{r}$ using the gradient iteration.
- The incremental method updates $\overline{r}$ a total of N times, once after each transition.
- After each transition (i_k, i_{k+1}) it uses only the portion of the gradient affected by that transition:
  - Evaluate the (single-term) gradient $\nabla \tilde{J}(i_k, \overline{r})$ at the current value of $\overline{r}$ (call it r_k).
  - Sum all the terms that involve the transition (i_k, i_{k+1}), and update r_k by making a correction along their sum:

$$\begin{aligned} r_{k+1} = & r_k - \gamma \Bigg(\nabla \tilde{J}(i_k, r_k) \tilde{J}(i_k, r_k) \\ & - \left(\sum_{t=0}^{k} \alpha^{k-t} \nabla \tilde{J}(i_t, r_t) \right) g\big(i_k, \overline{\mu}(i_k), i_{k+1}\big) \Bigg) \end{aligned}$$

INCREMENTAL GRADIENT METHOD II

- After N transitions, all the component gradient terms of the batch iteration are accumulated.
- BIG difference:
  - In the incremental method, $\overline{r}$ is changed while processing the batch – the (single-term) gradient $\nabla\tilde{J}(i_t, \overline{r})$ is evaluated at the most recent value of $\overline{r}$ [after the transition (i_t, i_{t+1})].
  - In the batch version these gradients are evaluated at the value of $\overline{r}$ prevailing at the beginning of the batch.
- Because $\overline{r}$ is updated at intermediate transitions within a batch (rather than at the end of the batch), the location of the end of the batch becomes less relevant.
- Can have very long batches - can have a single very long simulated trajectory and a single batch.
- The incremental version can be implemented more flexibly, converges much faster in practice.
- Interesting convergence analysis (beyond our scope - see Bertsekas and Tsitsiklis, NDP book, also paper in SIAM J. on Optimization, 2000)

TEMPORAL DIFFERENCES - TD(1)

- A mathematically equivalent implementation of the incremental method.
- It uses *temporal difference* (TD for short)

$$d_k = g\big(i_k, \overline{\mu}(i_k), i_{k+1}\big) + \alpha \tilde{J}(i_{k+1}, \overline{r}) - \tilde{J}(i_k, \overline{r}), \ \ k \le N-2,$$

$$d_{N-1} = g\big(i_{N-1}, \overline{\mu}(i_{N-1}), i_N\big) - \tilde{J}(i_{N-1}, \overline{r})$$

- Following the transition (i_k, i_{k+1}), set

$$r_{k+1} = r_k + \gamma_k d_k \sum_{t=0}^{k} \alpha^{k-t} \nabla \tilde{J}(i_t, r_t)$$

- This algorithm is known as TD(1). In the important linear case $\tilde{J}(i,r) = \phi(i)'r$, it becomes

$$r_{k+1} = r_k + \gamma_k d_k \sum_{t=0}^{k} \alpha^{k-t} \phi(i_t)$$

- A variant of TD(1) is TD(λ), $\lambda \in [0,1]$. It sets

$$r_{k+1} = r_k + \gamma_k d_k \sum_{t=0}^{k} (\alpha\lambda)^{k-t} \phi(i_t)$$

OPTIMISTIC POLICY ITERATION

- We have assumed so far is that the least squares optimization must be solved completely for $\overline{r}$.

- An alternative, known as *optimistic policy iteration*, is to solve this problem approximately and replace policy μ with policy $\overline{\mu}$ after only a few simulation samples.

- Extreme possibility is to replace μ with $\overline{\mu}$ at the end of each state transition: After state transition (i_k, i_{k+1}), set

$$r_{k+1} = r_k + \gamma_k d_k \sum_{t=0}^{k} (\alpha\lambda)^{k-t} \nabla \tilde{J}(i_t, r_t),$$

and simulate next transition (i_{k+1}, i_{k+2}) using $\overline{\mu}(i_{k+1})$, the control of the new policy.

- For $\lambda = 0$, we obtain (the popular) optimistic TD(0), which has the simple form

$$r_{k+1} = r_k + \gamma_k d_k \nabla \tilde{J}(i_k, r_k)$$

- Optimistic policy iteration can exhibit fascinating and counterintuitive behavior (see the NDP book by Bertsekas and Tsitsiklis, Section 6.4.2).

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.
- As a result, the cost-to-go estimates of these underrepresented states may be highly inaccurate.
- This can cause serious errors in the calculation of the improved control 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).
- One possibility to guarantee adequate exploration: Frequently restart the simulation and ensure that the initial states employed form a rich and representative subset.
- Another possibility is to artificially introduce some extra randomization in the simulation, by occasionally generating transitions that use a randomly selected control rather than the one dictated by the policy μ.

APPROXIMATING Q-FACTORS

- The approach described so far for policy evaluation requires calculating expected values for all controls $u \in U(i)$ (and knowledge of $p_{ij}(u)$).

- Model-free alternative: Approximate Q-factors

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

and use for policy improvement the minimization

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

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

$$\tilde{Q}(i,u,r) = \sum_{k=1}^{m} r_k \phi_k(i,u)$$

- Can use any method for constructing cost approximations, e.g., TD(λ).

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

- Major concern: Acutely diminished exploration.