Infinite-Horizon Dynamic Programming

http://bicmr.pku.edu.cn/~wenzw/bigdata2018.html

Acknowledgement: this slides is based on Prof. Mengdi Wang's and Prof. Dimitri Bertsekas' lecture notes

作业

1) 阅读如下章节： Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, http://incompleteideas.net/book/the-book-2nd.html

- Chapter 2: Multi-armed Bandits
- Chapter 3: Finite Markov Decision Processes
- Chapter 4: Dynamic Programming
- Chapter 5: Monte Carlo Methods
- Chapter 6: Temporal-Difference Learning
- Chapter 9: On-policy Prediction with Approximation
- Chapter 10: On-policy Control with Approximation
- Chapter 13: Policy Gradient Methods

2) 至少看懂每章的三个Example。如果有程序，测试或实现其程序。

Outline

Infinite-Horizon Discounted Problems/Bounded Cost

- Stationary system

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

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

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

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

- Optimal cost function is defined as

$$J^*(x) = \min_\pi \quad J_\pi(x)$$

Infinite-Horizon Discounted Problems/Bounded Cost

- Boundedness of g guarantees that all costs are well-defined and bounded:
$$|J_\pi(x)| \leq \frac{M}{1-\alpha}$$
- All spaces are arbitrary - only boundedness of g is important (there are math fine points, e.g. measurability, but they don't matter in practice)
- Important special case with finite space: Markovian Decision Problem
- All algorithms ultimately work with a finite spaces MDP approximating the original problem

Shorthand notation for DP mappings

- For any function J of x, denote

$$(TJ)(x) = \min_{u \in U(x)} \mathbf{E}_w \{ g(x,u,w) + \alpha J(f(x,u,w)) \}, \quad \forall x$$

- TJ is the optimal cost function for the one-stage problem with stage cost g and terminal cost function αJ.
- T operates on bounded functions of x to produce other bounded functions of x
- For any stationary policy μ, denote

$$(T_\mu J)(x) = \mathbf{E}_w \{ g(x,\mu(x),w) + \alpha J(f(x,\mu(x),w)) \}, \ \forall x$$

- The critical structure of the problem is captured in T and T_μ
- The entire theory of discounted problems can be developed in shorthand using T and T_μ
- True for many other DP problems.
- T and T_μ provide a powerful unifying framework for DP. This is the essence of the book “Abstract Dynamic Programming”

Express Finite-Horizon Cost using T

- Consider an N-stage policy $\pi_0^N = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$ with a terminal cost J and $\pi_1^N = \{\mu_1, \mu_2, \ldots, \mu_{N-1}\}$:

$$
\begin{aligned}
J_{\pi_0^N}(x_0) &= \mathbf{E}\left[\alpha^N J(x_k) + \sum_{\ell=0}^{N-1} \alpha^\ell g(x_\ell, \mu_\ell(x_\ell), w_\ell)\right] \\
&= \mathbf{E}\left[g(x_0, \mu_0(x_0), w_0) + \alpha J_{\pi_1^N}(x_1)\right] \\
&= (T_{\mu_0} J_{\pi_1^N})(x_0)
\end{aligned}
$$

- By induction, we have $J_{\pi_0^N}(x_0) = (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_{N-1}} J)(x), \ \forall x$
- For a stationary policy μ the N-stage cost function (with terminal cost J) is $J_{\pi_0^N} = T_\mu^N J$, where T_μ^N is the N-fold composition of T_μ
- Similarly the optimal N-stage cost function (with terminal cost J) is $T^N J$
- $T^N J = T(T^{N-1} J)$ is just the DP algorithm

Markov Chain

- A Markov chain is a random process that takes values on the state space $\{1, \dots, n\}$.
- The process evolves according to a certain transition probability matrix $P \in \mathbb{R}^{n\times n}$ where

$$P(i_{k+1} = j \mid i_k, i_{k-1}, \dots, i_0) = P(i_{k+1} = j \mid i_k = i) = P_{ij}$$

- Markov chain is memoryless, i.e., further evolvements are independent with past trajectory conditioned on the current state.
- The "memoryless" property is equivalent to "Markov."
- A state i is recurrent if it will be visited infinitely many times with probability 1.
- A Markov chain is said to be irreducible if its state space is a single communicating class; in other words, if it is possible to get to any state from any state.
- When states are modeled appropriately, all stochastic processes are Markov.

Markovian Decision Problem

We will mostly assume the system is an n-state (controlled) Markov chain

- 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 function $J = (J(1), \ldots, J(n))$ (vectors in $\mathbb{R}^n$)
- cost of a policy $\pi = \{\mu_0, \mu_1, \ldots\}$

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

- MDP is the most important problem in infinite-horizon DP

Markovian Decision Problem

- Shorthand notation for DP mappings

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

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

- Vector form of DP mappings

$$TJ = \min_{\mu}\{g_\mu + \alpha P_\mu J\}$$

and

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

where

$$g_\mu(i) = \sum_{j=1}^{n} p_{ij}(\mu(i)) g(i,\mu(i),j), \quad P_\mu(i,j) = p_{ij}(\mu(i))$$

Two Key properties

- Monotonicity property: For any J and J' such that $J(x) \leq J'(x)$ for all x, and any μ

$$(TJ)(x) \leq (TJ')(x), \quad (T_\mu J)(x) \leq (T_\mu J')(x), \quad \forall x$$

- Constant Shift property: For any J, any scalar r, and any μ

$$\big(T(J+re)\big)(x) = (TJ)(x) + ar, \quad \big(T_\mu(J+re)\big)(x) = (T_\mu J)(x) + ar, \ \forall x$$

  where e is the unit function [$e(x) \equiv 1$].

- Monotonicity is present in all DP models (undiscounted, etc)
- Constant shift is special to discounted models
- Discounted problems have another property of major importance: T and T_μ are contraction mappings (we will show this later)

Convergence of Value Iteration

Theorem

For all bounded J_0, we have $J^*(x) = \lim_{k\to\infty}(T^k J_0)(x)$, for all x

Proof. For simplicity we give the proof for $J_0 \equiv 0$. For any initial state x_0, and policy $\pi = \{\mu_0, \mu_1, \ldots\}$,

$$\begin{aligned} J_\pi(x_0) &= \mathbf{E}\left[\sum_{\ell=0}^{\infty} \alpha^\ell g(x_\ell, \mu_\ell(x_\ell), w_\ell)\right] \\ &= \mathbf{E}\left[\sum_{\ell=0}^{k-1} \alpha^\ell g(x_\ell, \mu_\ell(x_\ell), w_\ell)\right] + \mathbf{E}\left[\sum_{\ell=k}^{\infty} \alpha^\ell g(x_\ell, \mu_\ell(x_\ell), w_\ell)\right] \end{aligned}$$

The tail portion satisfies

$$\left|\mathbf{E}\left[\sum_{\ell=k}^{\infty} \alpha^\ell g(x_\ell, \mu_\ell(x_\ell), w_\ell)\right]\right| \leq \frac{\alpha^k M}{1-\alpha}$$

where $M \geq |g(x,u,w)|$. Take min over π of both sides, then $\lim$ as $k \to \infty$

Proof of Bellman's equation

Theorem

The optimal cost function J^* is a solution of Bellman's equation, $J^* = TJ^*$, i.e., for all x,

$$J^*(x) = \min_{u \in U(x)} \mathbf{E}_w \{ g(x,u,w) + \alpha J^*(f(x,u,w)) \}$$

Proof. For all x and k, $J^*(x) - \frac{\alpha^k M}{1-\alpha} \leq (T^k J_0)(x) \leq J^*(x) + \frac{\alpha^k M}{1-\alpha}$ where $J_0(x) \equiv 0$ and $M \geq |g(x,u,w)|$. Applying T to this relation, and using Monotonicity and Constant Shift,

$$(TJ^*)(x) - \frac{\alpha^{k+1} M}{1-\alpha} \leq (T^{k+1} J_0)(x) \leq (TJ^*)(x) + \frac{\alpha^{k+1} M}{1-\alpha}$$

Taking the limit as $k \to \infty$ and using the fact $\lim_{k \to \infty} (T^{k+1} J_0)(x) = J^*(x)$ we obtain $J^* = TJ^*$.

The Contraction Property

Contraction property: For any bounded functions J and J', and any μ,

$$\begin{aligned} \max_x |(TJ)(x) - (TJ')(x)| &\leq \alpha \max_x |J(x) - J'(x)|, \\ \max_x |(T_\mu J)(x) - (T_\mu J')(x)| &\leq \alpha \max_x |J(x) - J'(x)| \end{aligned}$$

Proof. Denote $c = \max_{x \in S} |J(x) - J'(x)|$. Then

$$J(x) - c \leq J'(x) \leq J(x) + c, \quad \forall x$$

Apply T to both sides, and use the Monotonicity and Constant Shift properties:

$$(TJ)(x) - \alpha c \leq (TJ')(x) \leq (TJ)(x) + \alpha c, \ \forall x$$

Hence, $|(TJ)(x) - (TJ')(x)| \leq \alpha c, \ \forall x$.
This implies that T, T_μ have unique fixed points. Then J^* is the unique solution of $J^* = TJ^*$, and J_μ is the unique solution of $J_\mu = T_\mu J_\mu$

Necessary and Sufficient Optimality Condition

Theorem

A stationary policy μ is optimal if and only if $\mu(x)$ attains the minimum in Bellman's equation for each x; i.e.,

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

or, equivalently, for all x,

$$\mu(x) \in \arg \min_{u \in U(x)} \mathbf{E}_w \{ g(x, u, w) + \alpha J^*(f(x, u, w)) \}$$

Proof of Optimality Condition

Proof: We have two directions.

- If $TJ^* = T_\mu J^*$, then using Bellman's equation $(J^* = TJ^*)$, we have $J^* = T_\mu J^*$, so by uniqueness of the fixed point of T_μ, we obtain $J^* = J_\mu$; i.e., μ is optimal.
- Conversely, if the stationary policy μ is optimal, we have $J^* = J_\mu$, so $J^* = T_\mu J^*$. Combining this with Bellman's Eq. $(J^* = TJ^*)$, we obtain $TJ^* = T_\mu J^*$.

Two Main Algorithms

Value Iteration

Solve the Bellman equation $J^* = TJ^*$ by iterating on the value functions:

$$J_{k+1} = TJ_k,$$

or

$$J_{k+1}(i) = \min_u \sum_{j=1}^{n} p_{ij}(u)(g(i,u,j) + \alpha J_k(j))$$

for $i = 1, \ldots, n$.

- The program only needs to memorize the current value function J_k.
- We have shown that $J_k \to J^*$ as $k \to \infty$.

Policy Iteration

Solve the Bellman equation $J^* = TJ^*$ by iterating on the policies

Policy Iteration (PI)

Given μ_k, the k-th policy iteration has two steps

- Policy evaluation: Find J_{μ^k} by $J_{\mu^k} = T_{\mu^k} J_{\mu^k}$ or solving

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

- Policy improvement: Let μ_{k+1} be such that $T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}$ or

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

Policy iteration is a method that updates the policy instead of the value function.

Policy Iteration (PI)

More abstractly, the k-th policy iteration has two steps

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

$$J_{\mu^k} = T_{\mu^k} J_{\mu^k} = g_{\mu^k} + \alpha P_{\mu^k} J_{\mu^k}$$

- Policy improvement: Let μ^{k+1} be such that $T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}$

Comments:

- Policy evaluation is equivalent to solving an $n \times n$ linear system of equations
- Policy improvement is equivalent to 1-step lookahead using the evaluated value function
- For large n, exact PI is out of the question . We use instead optimistic PI (policy evaluation with a few VIs)

Convergence of Policy Iteration

Theorem

Assume that the state and action spaces are finite. The policy iteration generates μ_k that converges to the optimal policy μ^* in a finite number of steps.

Proof. We show that $J_{\mu^k} \geq J_{\mu^{k+1}}$ for all k. For given k, we have

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

Using the monotonicity property of DP,

$$J_{\mu^k} \geq T_{\mu^{k+1}} J_{\mu^k} \geq T^2_{\mu^{k+1}} J_{\mu^k} \geq \ldots \geq \lim_{N \to \infty} T^N_{\mu^{k+1}} J_{\mu^k}$$

Since $\lim_{N \to \infty} T^N_{\mu^{k+1}} J_{\mu^k} = J_{\mu^{k+1}}$, we have $J_{\mu^k} \geq J_{\mu^{k+1}}$.
If $J_{\mu^k} = J_{\mu^{k+1}}$, all above inequalities hold as equations, so J_{μ^k} solves Bellman's equation. Hence $J_{\mu^k} = J^*$. Thus at iteration k either the algorithm generates a strictly improved policy or it finds an optimal policy. For a finite spaces MDP, the algorithm terminates with an optimal policy.

“Shorthand” Theory - A Summary

- 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 $J^* = TJ^*$, $J_\mu = T_\mu J_\mu$
- Optimality condition: μ is optimal iff $T_\mu J^* = TJ^*$
- Value iteration: For any (bounded) J:

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

- Policy iteration: given μ^k,
  - Policy evaluation: Find J_{μ^k} by solving $J_{\mu^k} = T_{\mu^k}J_{\mu^k}$
  - Policy improvement: Let μ^{k+1} be such that $T_{\mu^{k+1}}J_{\mu^k} = TJ_{\mu^k}$

Q-Factors

- Optimal Q-factor of (x, u) :

$$Q^*(x, u) = \mathbf{E}\{g(x, u, w) + \alpha J^*(f(x, u, w))\}$$

- It is the cost of starting at x, applying u in the 1st stage, and an optimal policy after the 1st stage
- The value function is equivalent to

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

- Q-factors are costs in an "augmented" problem where states are (x, u)
- Here (x, u) is a post-decision state.

VI in Q-factors

- We can equivalently write the VI method as

$$J_{k+1}(x) = \min_{u \in U(x)} Q_{k+1}(x, u), \quad \forall x$$

where Q_{k+1} is generated by

$$Q_{k+1}(x, u) = \mathbf{E}\left[g(x, u, w) + \alpha \min_{v \in U(\bar{x})} Q_k(f(x, u, w), v)\right]$$

- VI converges for Q-factors

Q-factors

- VI and PI for Q-factors are mathematically equivalent to VI and PI for costs
- They require equal amount of computation ... they just need more storage
- Having optimal Q-factors is convenient when implementing an optimal policy on-line by

$$\mu^*(x) = \arg \min_{u \in U(x)} Q^*(x, u)$$

- Once $Q^*(x, u)$ are known, the model [g and $E\{\cdot\}$] is not needed. Model-free operation
- Q-Learning (to be discussed later) is a sampling method that calculates $Q^*(x, u)$ using a simulator of the system (no model needed)

MDP and Q-Factors

Optimal Q-factors - the function $Q(i,u)$ that satisfies the following Bellman equation

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u) \left(g(i,u,j) + \alpha \min_{v \in U(j)} Q^*(j,v) \right)$$

or in short $Q^* = FQ^*$.

Interpretation Q-factors can be viewed as J values by considering (i,u) as the post-decision state

DP Algorithm for Q-values instead of J-values

- Value Iteration: $Q_{k+1} = FQ_k$
- Policy evaluation: $Q_{\mu_k} = F_{\mu_k} Q_{\mu_k}$
- Policy improvement: $F_{\mu_{k+1}} Q_{\mu_k} = FQ_{\mu_k}$
- VI and PI are convergent for Q-values
- Model-free.

Other DP Models

- We have looked so far at the (discrete or continuous spaces) discounted models for which the analysis is simplest and results are most powerful
- Other DP models include:
- Undiscounted problems ($\alpha = 1$): They may include a special termination state (stochastic shortest path problems)
- Continuous-time finite-state MDP : The time between transitions is random and state-and-control-dependent (typical in queueing systems, called Semi-Markov MDP). These can be viewed as discounted problems with state-and-control-dependent discount factors
- Continuous-time, continuous-space models : Classical automatic control, process control, robotics
- Substantial differences from discrete-time
- Mathematically more complex theory (particularly for stochastic problems)
- Deterministic versions can be analyzed using classical optimal control theory

Outline

(Optional) Formalism: MDP

- **Markov Decision Processes** (MDPs). An MDP is a 5-tuple, $\langle S, A, R, P, \rho_0 \rangle$, where
- S is the set of all valid states,
- A is the set of all valid actions,
- $R : S \times A \times S \to \mathbb{R}$ is the reward function, with $r_t = R(s_t, a_t, s_{t+1})$,
- $P : S \times A \to \mathcal{P}(S)$ is the transition probability function, with $P(s'|s, a)$ being the probability of transitioning into state s' if you start in state s and take action a,
- and ρ_0 is the starting state distribution.
- The name Markov Decision Process refers to the fact that the system obeys the 'Markov property': transitions only depend on the most recent state and action, and no prior history. https://en.wikipedia.org/wiki/Markov_property

Look at the Bellman Equation Again

Consider a MDP model with

- States $i = 1, \ldots, n$
- Probability transition matrix under policy μ is $P_\mu \in \mathbb{R}^{n \times n}$
- Reward of transition is $g_\mu \in \mathbb{R}^n$

The Bellman equation is

$$J = \min_{\mu} \quad g_\mu + \alpha P_\mu J$$

This is a nonlinear system of equations.

Note: The righthandside is the infimum of a number of linear mappings of J!

DP is a special case of LP

Theorem

Every finite-state DP problem is an LP problem.

Let $c \geq 0$. We construct the following LP

$$\begin{aligned} \max \quad & c_1 J(1) + \ldots + c_n J(n) \\ \text{s.t.} \quad & J(i) \leq \sum_{j=1}^{n} p_{ij}(u) g_{iju} + \alpha \sum_{j=1}^{n} p_{ij}(u) J(j), \forall u \in A \end{aligned}$$

or more compactly

$$\begin{aligned} \max \quad & c' J \\ \text{s.t.} \quad & J \leq g_\mu + \alpha P_\mu J, \forall u \in A \end{aligned}$$

- The variables are $J(i)$ where $i = 1, ..., n$.
- For each state action pair (i, u), there is an inequality constraint.

DP is a special case of LP

If $J \leq TJ$, then $J \leq J^*$. If $J \geq TJ$, then $J \geq J^*$.

- Suppose that $J \leq TJ$. Applying operator T on both sides $k-1$ times, and by the monotonicity of T, we have

$$J \leq TJ \leq T^2 J \leq \ldots \leq T^k J.$$

Note that $\lim_{k\to\infty} T^k J = J^*$. Hence, we have $J \leq J^*$.

DP is a special case of LP

Theorem

This solution to the constructed LP

$$\begin{aligned} \max \quad & c'J \\ \text{s.t.} \quad & J \leq g_\mu + \alpha P_\mu J, \forall u \in A \end{aligned}$$

is exactly the solution to the Bellman's equation

$$J = \min_\mu g_\mu + P_\mu J$$

Proof: The solution J^* to the Bellman equation is obviously a feasible solution to the LP. If the LP solution $\bar{J}$ is different from J^*, it must solve the Bellman equation at the same time. Since the Bellman equation has a unique solution, $J^* = \bar{J}$.

ADP via Approximate Linear Programming

The constructed LP is of huge scale.

$$\begin{aligned} \max \quad & c'J \\ \text{s.t.} \quad & J \leq g_\mu + \alpha P_\mu J, \forall u \in A \end{aligned}$$

Approximate LP:

- We may approximate J by adding the constraint $J = \Phi\sigma$, so the variable dimension becomes smaller.
- We may sample a subset of all constraints, so the constraint dimension becomes smaller.
- LP and Approximate LP can be solved by simulation/online.

Outline

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.
- 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 $\sum_{i=1}^{n} a_i$ can be written as an expected value:

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} \xi_i \frac{a_i}{\xi_i} = \mathbf{E}_\xi \left[\frac{a_i}{\xi_i} \right]$$

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

$$\sum_{i=1}^{n} a_i = \mathbf{E}_\xi \left[\frac{a}{\xi} \right] \approx \frac{1}{k} \sum_{t=1}^{k} \frac{a_{i_t}}{\xi_{i_t}}$$

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

Solve DP via Simulation

- Ideally, VI and PI solve the fixed equation: finding J^* such that

$$J^* = \min_{\mu}\{g_\mu + \alpha P_\mu J^*\}$$

- Practically, we often wish to solve Bellman's equation without knowing P_μ, g_μ.
- What we do have: a simulator that starts from state i, given action a, generate random samples of transition costs and future state $g(i, i_{\text{next}}, a), i_{\text{next}}$

Example: Optimize a trading policy to maximize profit

- Current transaction has unknown market impact
- Use current order book as states/features

Example: stochastic games, Tetris, hundreds of millions of states, captured using 22 features

Outline

Approximation in value space

- Approximate J^* or J_μ from a parametric class $\tilde{J}(i;\sigma)$ where i is the current state and $\sigma = (\sigma_1, \ldots, \sigma_m)$ is a vector of "tunable" scalars weights
- Use $\tilde{J}$ in place of J^* or J_μ in various algorithms and computations
- Role of σ : By adjusting σ 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:

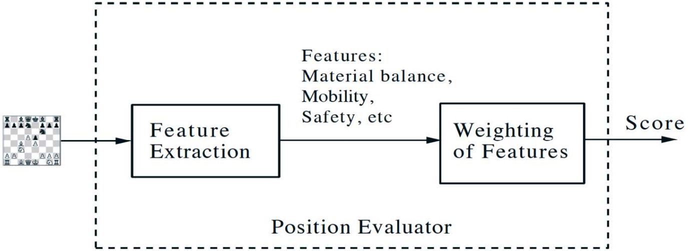
- The choice of parametric class $\tilde{J}(i;\sigma)$ (the approximation architecture)
- Method for tuning the weights ("training" the architecture)

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 $\tilde{J}(i;\sigma)$ on σ]
- 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:

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

or

$$\tilde{J}\sigma = \Phi\sigma = \sum_{j=1}^{s} \Phi_j \sigma_j$$

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

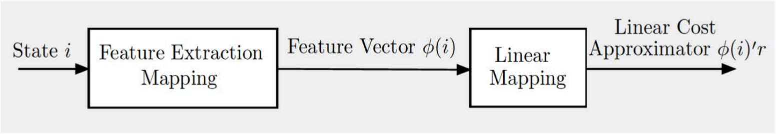
- This is approximation on the subspace

$$S = \{\Phi\sigma \mid \sigma \in \mathbb{R}^s\}$$

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



- 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

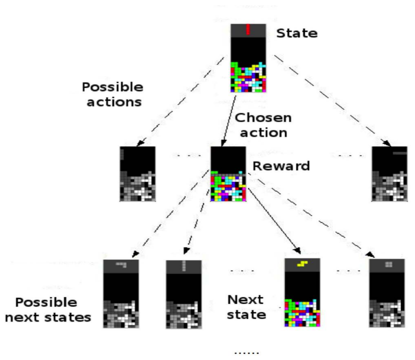
$$\phi_0(i) = 1, \phi_k(i) = i_k, \phi_{km}(i) = i_k i_m, k, m = 1, \ldots, q$$

  Linear approximation architecture :

$$\tilde{J}(i; \sigma) = \sigma_0 + \sum_{k=1}^{q} \sigma_k i_k + \sum_{k=1}^{q} \sum_{m=k}^{q} \sigma_{km} i_k i_m,$$

  where σ has components σ_0, σ_k, and σ_{km}.
- Interpolation : A subset I of special/representative states is selected, and the parameter vector σ has one component σ_i per state $i \in I$. The approximating function is $\tilde{J}(i; \sigma) = \sigma_i, i \in I$. $\tilde{J}(i; \sigma)$ is the interpolation using the values at $i \in I, i \notin I$. 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 $\mu(i;\sigma)$ of policies with a vector $\sigma = (\sigma_1, \ldots, \sigma_s)$.

Examples:

- Polynomial, e.g., $\mu(i;\sigma) = \sigma_1 + \sigma_2 \cdot i + \sigma_3 \cdot i^2$
- Linear feature-based

$$\mu(i;\sigma) = \phi_1(i) \cdot \sigma_1 + \phi_2(i) \cdot \sigma_2$$

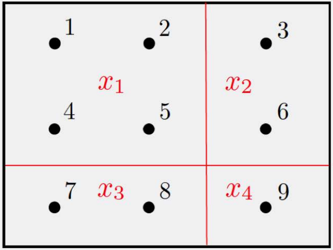
Approximation in Policy Space

- Optimize the cost over σ. For example:
- Each value of σ defines a stationary policy, with cost starting at state i denoted by $\tilde{J}(i;\sigma)$.
- Let $(p_1,\ldots,p_n)$ be some probability distribution over the states, and minimize over σ: $\sum_{i=1}^n p_i\tilde{J}(i;\sigma)$
- Use a random search, gradient, or other method
- A special case: The parameterization of the policies is indirect, through a cost approximation architecture $\tilde{J}$, i.e.,

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

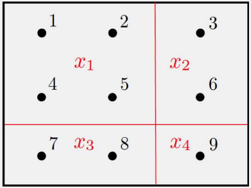
Aggregation

- A first idea : Group similar states together into “aggregate states” $x_1, \ldots, x_s$; assign a common cost value σ_i to each group x_i.
- Solve an “aggregate” DP problem , involving the aggregate states, to obtain $\sigma = (\sigma_1, ..., \sigma_s)$. 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



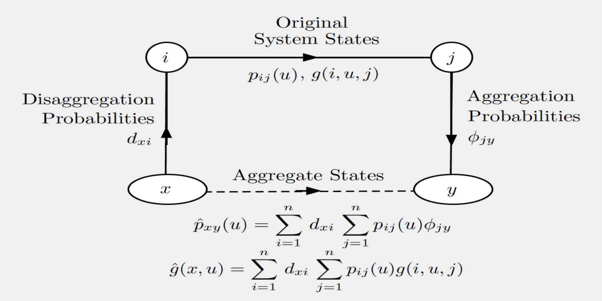
$$\Phi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- More general/mathematical view : Solve

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

  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 $\Phi\sigma = \Pi T_\mu(\Phi\sigma)$. Note: ΦD is a projection in some interesting cases

Aggregation as Problem Abstraction



- 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

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

  Bellman's equation for the aggregate problem.

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

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

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

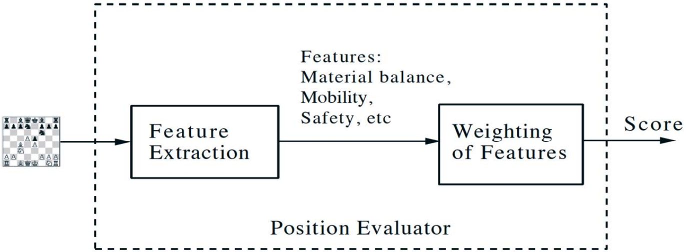
Example III: Representative States/Coarse Grid

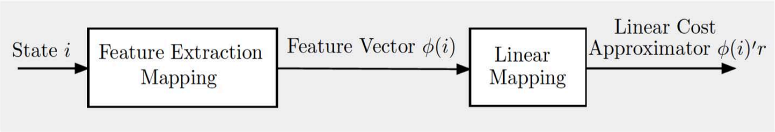
- Choose a collection of "representative" original system states, and associate each one of them with an aggregate state
- Disaggregation probabilities are $d_{xi} = 1$ if i is equal to representative state x.
- Aggregation probabilities associate original system states with convex combinations of representative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

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

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

Feature Extraction is Linear Approximation of High-d Cost Vector

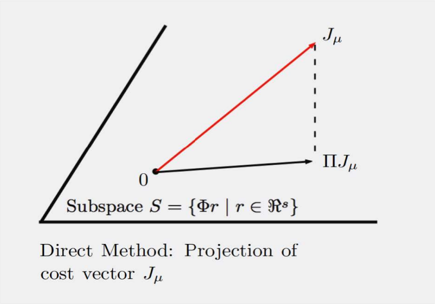




Outline

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



- 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 $S = \{\Phi\sigma \mid \sigma \in \Re^s\}$, with respect to a weighted Euclidean norm $\|\cdot\|_\xi$
- Equivalently, find $\Phi\sigma^*$, where

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

- Setting to 0 the gradient at σ^*,

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

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

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

- Equivalent least squares alternative calculation:

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

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

$$\sigma_k \xrightarrow{a.s.} \sigma^* = \Pi_S J_\mu$$

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

Indirect policy evaluation

- Solve the projected equation

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

  where Π is projection with 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 $\Phi\sigma = \Pi T_\mu(\Phi\sigma)$
- LSTD(λ): Solves a simulation-based approximation with a standard solver
- LSPE(λ): A simulation-based form of projected value iteration ; essentially

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

- Almost sure convergence guarantee

Bellman Error Minimization

- Another example of indirect approximate policy evaluation:

$$\min_{\sigma} \|\Phi\sigma - T_{\mu}(\Phi\sigma)\|_{\xi}^{2} \qquad (*)$$

  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 σ^* 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

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

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

Matrix Form of Projected Equation

- The solution $\Phi\sigma^*$ satisfies the orthogonality condition: The error

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

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

- This is written as

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

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

- Equivalently, $C\sigma^* = d$, where

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

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

Simulation-Based Implementations

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

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

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

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

  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, σ_k converges a.s. to the solution of projected Bellman equation.

Approximate PI via On-Policy Learning

Outer Loop (Off-Policy RL):

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

$$J_{\mu_t} \approx \Phi\sigma_t$$

  Inner Loop (On-Policy RL):

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

- Update the policy by

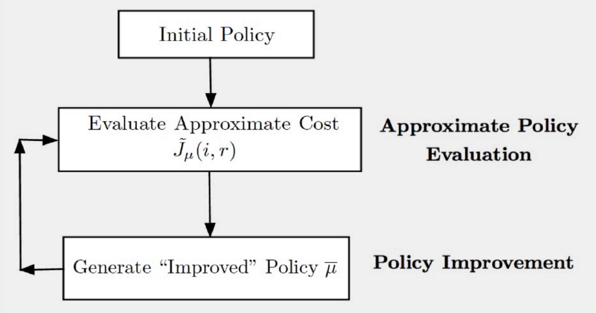
$$\mu_{t+1}(i) = \arg\min_a \sum_j p_{ij}(\alpha)(g(i,\alpha,j) + \phi(j)'\sigma_t), \quad \forall i$$

Comments:

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

Approximate PI via On-Policy Learning

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



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

$$\max_i |\tilde{J}(i, \sigma_k) - J_{\mu^k}(i)| \le d, \quad k = 0, 1, \ldots,$$

- If policy improvement is also approximate,

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

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

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

- 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.