Topics in Reinforcement Learning: AlphaZero, ChatGPT, Neuro-Dynamic Programming, Model Predictive Control, Discrete Optimization

Arizona State University
Course CSE 691, Spring 2024

Links to Class Notes, Videolectures, and Slides at
http://web.mit.edu/dimitrib/www/RLbook.html

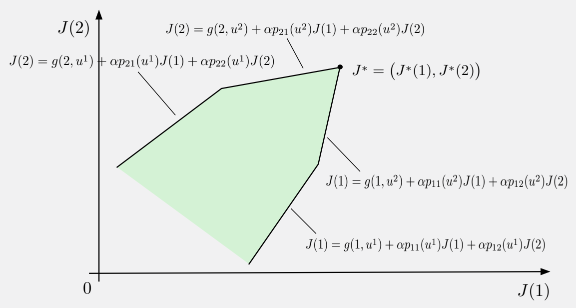
Dimitri Bertsekas dbertsek@asu.edu, Jamison Weber jwweber@asu.edu

Lecture 13
Approximate Linear Programming;
Policy Gradient and Random Search Methods

Outline

1. Linear Programming: Another Approach to Approximation in Value Space
2. Approximation in Policy Space: Motivation
3. Training of Policies by Cost Optimization - Random Search
4. Training of Policies by Cost Optimization - Policy Gradient Methods
5. Implementation Issues of Policy Gradient Methods

Exact Solution of Discounted DP by Linear Programming



Key idea: J^* is the "largest" J that satisfies the constraint

$$J(i) \le \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha J(j)\big), \qquad \text{for all } i = 1,\ldots,n \text{ and } u \in U(i),$$

so that $J^* = \big(J^*(1),\ldots,J^*(n)\big)$ maximizes $\sum_{i=1}^{n} J(i)$ subject to the above constraint.

Proof: Generate sequence $\{J_k\}$ with VI, starting from any $J = J_0$ satisfying the constraint, which implies that $J_0 \le J_1$. Since $J_k = T^k J_0$ and T is monotone, we have $J = J_0 \le J_k \le J_{k+1} \to J^*$. So any J satisfying the constraint also satisfies $J \le J^*$.

Linear Programming with Approximation in Value Space

Difficulty of the exact LP algorithm for large problems

Too many variables (n) and too many constraints (the # of state-control pairs).

Introduce a linear feature-based architecture $J^*(i) \approx \tilde{J}(i,r) = \sum_{\ell=1}^{m} r_\ell \phi_\ell(i)$

- Replace $J(i)$ with $\tilde{J}(i,r)$ to reduce the number of variables.
- Introduce constraint sampling to reduce the number of constraints.
- Maximize $\sum_{i \in \tilde{I}} \tilde{J}(i,r)$ subject to

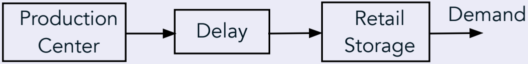
$$\tilde{J}(i,r) \leq \sum_{i=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}(j,r)\big), \quad i \in \tilde{I}, \; u \in \tilde{U}(i)$$

  This is a linear program.
- $\tilde{I}$ is a set of "representative states", $\tilde{U}(i)$ is a set of "representative controls".
- Sampling with some known suboptimal policies is typically used to select a subset of the constraints to enforce; progressively enrich the subset as necessary.
- The approach has not been used widely, but has been successful on substantive test problems (see Van Roy and De Farias' works, among others).
- Capitalizes on the reliability of large-scale LP software.

General Framework for Approximation in Policy Space

- Parametrize stationary policies with a parameter vector r; denote them by $\tilde{\mu}(r)$, with components $\tilde{\mu}(i, r)$, $i = 1, \dots, n$. Each r defines a policy.
- The parametrization may be problem-specific, or feature-based, or may involve a neural network.
- The idea is to optimize some measure of performance with respect to r.

An example of problem-specific/natural parametrization: Supply chains, inventory control



- Retail center places orders to the production center, depending on current stock; there may be orders in transit; demand and delays can be stochastic.
- State is (current stock, orders in transit, ++). Can be formulated by DP but can be very difficult to solve exactly.
- Intuitively, a near-optimal policy is of the form: When the retail inventory goes below level r_1, order an amount r_2. Optimize over the parameter vector $r = (r_1, r_2)$.
- Extensions to a network of production/retail centers, multiple products, etc.

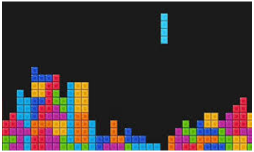
Another Example: Policy Parametrization Through Value Parametrization

Indirect parametrization of policies through cost features

- Suppose $\tilde{J}(i,r)$ is a cost function parametric approximation.
- $\tilde{J}$ may be a linear feature-based architecture that is natural for the given problem.
- Define

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

- This is useful when we know a good parametrization in value space, but we want to use a method that works well in policy space, and results in an easily implementable policy.



Tetris example: There are good linear parametrizations through features. Great success has been achieved by indirect approximation in policy space.

Working Break: When Would you Use Approximation in Policy Space?

Think about at least six contexts where approximation in policy space is either essential or is helpful

- Problems with natural policy parametrizations (like the supply chain problem)
- Problems with natural value parametrizations (like the tetris problem), where a good policy training method works well.
- Approximation in policy space on top of approximation in value space.
- Learning from a software or human expert.
- Unconventional information structures (limited memory, etc) - Conventional DP breaks down.
- Multiagent systems with local information (not shared with other agents).

Policy Approximation on Top of Value Approximation

- Compute approximate cost-to-go function $\tilde{J}$ using an approximation in value space scheme.
- This defines the corresponding suboptimal policy $\hat{\mu}$ through one-step lookahead,

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

  or a multistep lookahead version.
- Approximate $\hat{\mu}$ using a training set consisting of a large number q of sample pairs $\big(i^s, u^s\big)$, $s = 1, \ldots, q$, where $u^s = \hat{\mu}(i^s)$.
- In particular, introduce a parametric family of policies $\tilde{\mu}(i,r)$. Then obtain r by

$$\min_r \sum_{s=1}^{q} \left\| u^s - \tilde{\mu}(i^s, r) \right\|^2.$$

Learning from a Software or Human Expert

- Suppose we have a software or human expert that can choose a “good" or “near-optimal" control u^s at any state i^s.
- We form a sample set of representative state-control pairs (i^s, u^s), $s = 1, \ldots, q$.
- We introduce a parametric family of policies $\tilde{\mu}(i, r)$. Then obtain r by

$$\min_r \sum_{s=1}^{q} \left\| u^s - \tilde{\mu}(i^s, r) \right\|^2.$$

- This approach is known as expert supervised training.
- It has been used (in various forms) in backgammon and in chess.
- It can be used, among others, for initialization of other methods.

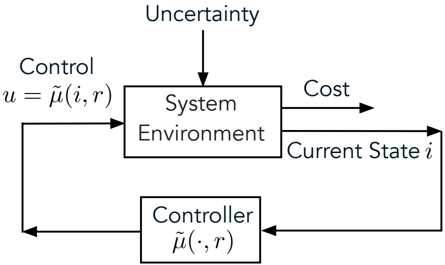
Unconventional Information Structures

- Approximation in value space is based on a DP formulation, so the controller has access to the exact state (or a belief state in case of partial state information).
- In some contexts this may not be true. There is a DP-like structure, but no full state or belief state is available.
- Example 1: The controller “forgets" information, e.g., “limited memory".
- Example 2: Some control components may be chosen on the basis of different information that others.

Example: Multiagent systems with local agent information

- Suppose decision making and information gathering is distributed among multiple autonomous agents.
- Each agent’s action depends only on his/her local information.
- Agents may be receiving delayed information from other agents.
- Then conventional DP and much of the approximation in value space methodology breaks down.
- Approximation in policy space is still applicable.

Optimization/Training Framework



Training by Cost Optimization

- Each r defines a stationary policy $\tilde{\mu}(r)$, with components $\tilde{\mu}(i, r)$, $i = 1, \ldots, n$.
- Determine r through the minimization

$$\min_r J_{\tilde{\mu}(r)}(i_0)$$

  where $J_{\tilde{\mu}(r)}(i_0)$ is the cost of the policy $\tilde{\mu}(r)$ starting from initial state i_0.
- More generally, determine r through the minimization

$$\min_r E\big\{J_{\tilde{\mu}(r)}(i_0)\big\}$$

  where the $E\{\cdot\}$ is with respect to a suitable probability distribution of i_0.

Training by Random Search

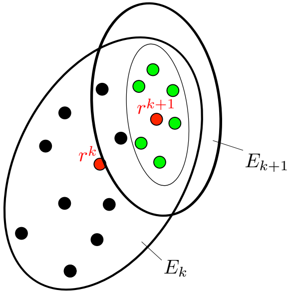
Random search methods apply to the general minimization $\min_{r \in R} F(r)$

- They generate a parameter sequence $\{r^k\}$ aiming for cost reduction.
- Given r^k, points are chosen in some random fashion in a neighborhood of r^k, and some new point r^{k+1} is chosen within this neighborhood.
- In theory they have good convergence properties. In practice they can be slow.
- They are not affected as much by local minima (as for example gradient-type methods).
- They don't require a differentiable cost function, and they apply to discrete as well as continuous minimization.
- There are many methods and variations thereoff.

Some examples

- Evolutionary programming.
- Tabu search.
- Simulated annealing.
- Cross entropy method.

Cross-Entropy Method - A Sketch



- At the current iterate r^k, construct an ellipsoid E_k centered at r^k.
- Generate a number of random samples within E_k. “Accept" a subset of the samples that have “low" cost.
- Let r^{k+1} be the sample “mean" of the accepted samples.
- Construct a sample “covariance" matrix of the accepted samples, form the new ellipsoid E_{k+1} using this matrix, and continue.
- Limited convergence rate guarantees. Success depends on domain-specific insight and the skilled use of implementation heuristics.
- Simple and well-suited for parallel computation. Resembles a “gradient method".

Policy Gradient Method for Deterministic Problems

Consider the minimization of $J_{\tilde{\mu}(r)}(i_0)$ over r by using the gradient method

$$r^{k+1} = r^k - \gamma^k \nabla J_{\tilde{\mu}(r^k)}(i_0)$$

assuming that $J_{\tilde{\mu}(r)}(i_0)$ is differentiable with respect to r.

- The difficulty is that the gradient $\nabla J_{\tilde{\mu}(r^k)}(i_0)$ may not be explicitly available.
- Then the gradient must be approximated by finite differences of cost function values $J_{\tilde{\mu}(r^k)}(i_0)$.
- When the problem is deterministic the gradient method may work well.
- When the problem is stochastic, the cost function values may be computable only through Monte Carlo simulation. Very hard to get accurate gradients by differencing function values.

Policy Gradient Method for Stochastic Problems

Consider the generic optimization problem $\min_{z \in Z} F(z)$

We take an unusual step: Convert this problem to the stochastic optimization problem

$$\min_{p \in \mathcal{P}_Z} E_p\{F(z)\}$$

where

- z is viewed as a random variable.
- $\mathcal{P}_Z$ is the set of probability distributions over Z.
- p denotes the generic distribution in $\mathcal{P}_Z$.
- $E_p\{\cdot\}$ denotes expected value with respect to p.

How does this relate to our infinite horizon DP problems?

- For this framework to apply to a stochastic DP context, we must enlarge the set of policies to include randomized policies, mapping a state i into a probability distribution over the set of controls $U(i)$.
- Note that in our DP problems, optimization over randomized policies gives the same results as optimization over ordinary/nonrandomized policies.
- In the DP context, z is the state-control trajectory: $z = \{i_0, u_0, i_1, u_1, \ldots\}$.

Gradient Method for Approximate Solution of $\min_{z \in Z} F(z)$

Parametrization of the probability distributions

- We restrict attention to a parametrized subset $\tilde{\mathcal{P}}_Z \subset \mathcal{P}_Z$ of probability distributions $p(z; r)$, where r is a continuous parameter.
- In other words, we approximate the problem $\min_{z \in Z} F(z)$ with the restricted problem

$$\min_r E_{p(z;r)}\{F(z)\}$$

- We use a gradient method for solving this problem:

$$r^{k+1} = r^k - \gamma^k \nabla \Big(E_{p(z;r^k)}\{F(z)\} \Big)$$

- Key fact: There is a useful formula for the gradient, which involves the gradient with respect to r of the natural logarithm $\log\big(p(z; r^k)\big)$.

The Gradient Formula (Reverses the Order of $E\{\cdot\}$ and ∇)

Assuming that $p(z;r^k)$ is a discrete distribution, we have

$$\begin{aligned}
\nabla\Big(E_{p(z;r^k)}\big\{F(z)\big\}\Big) &= \nabla\left(\sum_{z\in Z} p(z;r^k)F(z)\right) \\
&= \sum_{z\in Z} \nabla p(z;r^k)F(z) \\
&= \sum_{z\in Z} p(z;r^k)\frac{\nabla p(z;r^k)}{p(z;r^k)}F(z) \\
&= E_{p(z;r^k)}\Big\{\nabla\Big(\log\big(p(z;r^k)\big)\Big)F(z)\Big\}
\end{aligned}$$

Sample-Based Gradient Method for Parametric Approximation of $\min_{z\in Z} F(z)$

- At r^k obtain a sample z^k according to the distribution $p(z;r^k)$.
- Compute the sample gradient $\nabla\Big(\log\big(p(z^k;r^k)\big)\Big)F(z^k)$.
- Use it to iterate according to

$$r^{k+1} = r^k - \gamma^k\nabla\Big(\log\big(p(z^k;r^k)\big)\Big)F(z^k)$$

Policy Gradient Method - Discounted Problem

- Denote by z the infinite horizon state-control trajectory:

$$z = \{i_0, u_0, i_1, u_1, \ldots\}.$$

- We consider a parametrization of randomized policies $p(u \mid i; r)$ with parameter r, i.e., the control at state i is generated according to a distribution $p(u \mid i; r)$ over $U(i)$.
- Then for a given r, the state-control trajectory z is a random trajectory with probability distribution denoted $p(z; r)$.
- The cost corresponding to the trajectory z is

$$F(z) = \sum_{m=0}^{\infty} \alpha^m g(i_m, u_m, i_{m+1}),$$

 and the problem is to minimize $E_{p(z;r)}\{F(z)\}$, over r.

- The gradient needed in the gradient iteration

$$r^{k+1} = r^k - \gamma^k \nabla\Big(\log\big(p(z^k; r^k)\big)\Big) F(z^k)$$

 is given by

$$\nabla\Big(\log\big(p(z^k; r^k)\big)\Big) = \sum_{m=0}^{\infty} \nabla\Big(\log\big(p(u_m \mid i_m; r^k)\big)\Big)$$

Unusual Aspects of the Policy Gradient Method

- It involves the cost function of the discounted problem, but not its gradient ... In fact the cost per stage g may be nondifferentiable!
- The problem solved is a randomized version of the original ... so if $r^k \to \bar{r}$ and the distribution $p(z, \bar{r})$ is not atomic, a solution has to be extracted from this distribution.

Some of the implementation issues

- How to collect the trajectory samples z^k to strike a balance between convenient implementation and exploration of the search space.
- How to reduce the large noise in the cost calculation $F(z^k)$.
- Use of baseline b, i.e., iterate according to

$$r^{k+1} = r^k - \gamma^k \nabla\Big(\log\big(p(z^k; r^k)\big)\Big)\big(F(z^k) - b\big)$$

instead of

$$r^{k+1} = r^k - \gamma^k \nabla\Big(\log\big(p(z^k; r^k)\big)\Big)F(z^k)$$

There is theoretical basis for this (see the next slide).

Cost Shaping Technique - Can Serve for Noise Reduction

Introduce an equivalent “variational" problem (known since the 1960s)

- Subtract any known function $V(x)$ from $J^*(x)$:

$$\hat{J}(x) = J^*(x) - V(x), \qquad x = 1, \ldots, n$$

- Replace the cost per stage $g(x,u,y)$ with

$$\hat{g}(x,u,y) = g(x,u,y) + \alpha V(y) - V(x), \qquad x = 1, \ldots, n$$

- Then the original problem's Bellman's equation is written as another Bellman equation

$$\hat{J}(x) = \min_{u \in U(x)} \sum_{y=1}^{n} p_{xy}(u)\big(\hat{g}(x,u,y) + \alpha \hat{J}(y)\big), \qquad x = 1, \ldots, n$$

- $\hat{J}$ is the optimal cost of another problem: $g(x,u,y)$ is replaced by $\hat{g}(x,u,y)$
- The reformulated problem is equivalent as far as exact solution is concerned
- BUT $\hat{J}$ may have more favorable “shape" for approximation, i.e., policy gradient and other methods may work better for the reformulated problem
- Example: If $V \approx J^*$, approximation methods can capture more easily small scale variations in J^* ... compare with the discussion on advantage updating (Lecture 8)

Robustness of Policy Gradient Methods

There is a generic difficulty with using a fixed policy on-line:

- It is all-training no on-line play. (This could be good but could be very bad.)
- It does not adapt to changes in the problem's parameters.
- So approximation in policy space may not work well in adaptive control contexts.
- Also it does not yield the benefit of on-line lookahead minimization/rollout.
- Approximation in value space, and rollout may work much better (e.g., in AlphaZero).

An alternative use of approximation in policy space methods (including policy gradient)

It can provide a base policy for use in (truncated) rollout or can be used in Monte Carlo Tree Search. This is what is done in AlphaZero.

About the Next Lecture

We will cover approximation in value space by aggregation.

Check videolectures 11 and 12 from 2019 ASU class