

Consider the following scenario:

There are two envelopes, each of which has an unknown amount of money in it. You get to choose one of the envelopes. Given this is all you get to know, how should you choose?

Consider the changed scenario:

Same as above, but before you get to choose, you can ask me to disclose the amount in one of the envelopes. Without any distributional assumptions on the amounts of money, is there a strategy that could improve your expected pay-off over simply picking an envelope at random?

Parking lot navigation



- Reward function trades off:
 - Staying "on-road,"
 - Forward vs. reverse driving,
 - Amount of switching between forward and reverse,
 - Lane keeping,
 - On-road vs. off-road,
 - Curvature of paths.

[Abbeel et al., IROS 08]

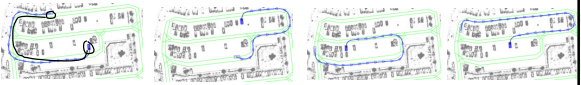
CS 287: Advanced Robotics
Fall 2009

Lecture 17: Policy search

Pieter Abbeel
 UC Berkeley EECS

Experimental setup

- Demonstrate parking lot navigation on "train parking lots."



- Run our apprenticeship learning algorithm to find the reward function.
- Receive "test parking lot" map + starting point and destination.
- Find the trajectory that maximizes the *learned reward function* for navigating the test parking lot.

Problem setup

- Input:
 - State space, action space
 - Transition model $P_{s_t}(s_{t+1} | s_t, a_t)$
 - No reward function
 - Teacher's demonstration: $s_0, a_0, s_1, a_1, s_2, a_2, \dots$
 (= trace of the teacher's policy π^*)
- Inverse RL:
 - Can we recover R ?
- Apprenticeship learning via inverse RL
 - Can we then use this R to find a good policy ?
- Vs. Behavioral cloning (which directly learns the teacher's policy using supervised learning)
 - Inverse RL: leverages compactness of the reward function
 - Behavioral cloning: leverages compactness of the policy class considered, does not require a dynamics model

Nice driving style



Sloppy driving-style



Without learning



Quadruped



- Reward function trades off 25 features.

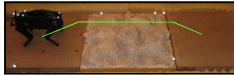
Hierarchical max margin [Kolter, Abbeel & Ng, 2008]

With learned reward function

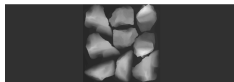


Experimental setup

- Demonstrate path across the "training terrain"



- Run our apprenticeship learning algorithm to find the reward function
- Receive "testing terrain"---height map.



- Find the optimal policy with respect to the *learned reward function* for crossing the testing terrain.

Hierarchical max margin [Kolter, Abbeel & Ng, 2008]

Announcements

- Assignment 1 was due yesterday at 23:59pm
 - For late day policy details, see class webpage.
- Reminder: Project milestone (1 page progress report) due Friday November 6.
- Grading:
 - 3 assignments: 25%, 25%, 5%
 - Final project: 45%
- Final project presentations:
 - Dec 1 & Dec 3
 - November 24: Zico Kolter guest lecture

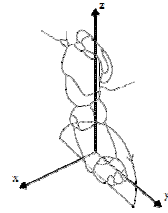
As part of the course requirements, I expect you to attend these 3 lectures

Lecture outline

- Inverse RL case studies wrap-up
- *Policy search*
 - Assume some policy class parameterized by a vector θ , search for the optimal setting of θ
 - Perhaps the largest number of success stories of RL

Policy class for quadruped locomotion on flat terrain

- 12 parameters define the Aibo's gait:
 - The front locus (3 parameters: height, x-pos., y-pos.)
 - The rear locus (3 parameters)
 - Locus length
 - Locus skew multiplier in the x-y plane (for turning)
 - The height of the front of the body
 - The height of the rear of the body
 - The time each foot takes to move through its locus
 - The fraction of time each foot spends on the ground



Kohl and Stone, ICRA2004

Policy class for helicopter hover

x, y, z : x points forward along the helicopter, y sideways to the right, z downward.

n_x, n_y, n_z : rotation vector that brings helicopter back to "level" position (expressed in the helicopter frame).

$$u_{collective} = \theta_1 \cdot f_1(z^* - z) + \theta_2 \cdot \dot{z}$$

$$u_{elevator} = \theta_3 \cdot f_2(x^* - x) + \theta_4 f_4(\dot{x}) + \theta_5 \cdot q + \theta_6 \cdot n_y$$

$$u_{aileron} = \theta_7 \cdot f_3(y^* - y) + \theta_8 f_5(\dot{y}) + \theta_9 \cdot p + \theta_{10} \cdot n_x$$

$$u_{rudder} = \theta_{11} \cdot r + \theta_{12} \cdot n_z$$

Kohl and Stone, ICRA 2004



Before learning (hand-tuned)



After learning

[Policy search was done through trials on the actual robot.]



[Policy search was done in simulation]

Ng + al, ISER 2004

Policy class for tetris

Let $S = s_1, s_2, \dots, s_n$ be the set of board situations available depending on the choice of placement of the current block. Then the probability of choosing the action that leads to board situation s_i is taken is given by:

$$\frac{\exp(\theta^T \phi(s_i))}{\sum_{j=1}^n \exp(\theta^T \phi(s_j))}$$

Deterministic, known dynamics

$$\max_{\theta} U(\theta) = \max_{\theta} \mathbb{E} \left[\sum_{t=0}^H R(s_t, a_t, s_{t+1}) | \pi_{\theta} \right]$$

Numerical optimization: find the gradient w.r.t. θ and take a step in the gradient direction.

$$\frac{\partial U}{\partial \theta_i} = \sum_{t=0}^H \frac{\partial R}{\partial s}(s_t, u_t, s_{t+1}) \frac{\partial s_t}{\partial \theta_i} + \frac{\partial R}{\partial u}(s_t, u_t, s_{t+1}) \frac{\partial u_t}{\partial \theta_i} + \frac{\partial R}{\partial s'}(s_t, u_t, s_{t+1}) \frac{\partial s_{t+1}}{\partial \theta_i}$$

$$\frac{\partial s_t}{\partial \theta_i} = \frac{\partial f}{\partial s}(s_{t-1}, u_{t-1}) \frac{\partial s_{t-1}}{\partial \theta_i} + \frac{\partial f}{\partial s'}(s_{t-1}, u_{t-1}) \frac{\partial u_{t-1}}{\partial \theta_i}$$

$$\frac{\partial u_t}{\partial \theta_i} = \frac{\partial \pi_{\theta}}{\partial \theta_i}(s_t, \theta) + \frac{\partial \pi_{\theta}}{\partial s}(s_t, \theta) \frac{\partial s_t}{\partial \theta_i}$$

Computing these recursively, starting at time 0 is a discrete time instantiation of Real Time Recurrent Learning (RTRL)

Finite differences

We can compute the gradient g using standard finite difference methods, as follows:

$$\frac{\partial U}{\partial \theta_j}(\theta) = \frac{U(\theta + \epsilon e_j) - U(\theta - \epsilon e_j)}{2\epsilon}$$

Where:

$$e_j = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow j\text{'th entry}$$

Stochastic, known dynamics

$$\max_{\theta} U(\theta) = \max_{\theta} \mathbb{E} \left[\sum_{t=0}^H R(s_t, a_t, s_{t+1}) | \pi_{\theta} \right] = \max_{\theta} \sum_{s, u} \sum_{s', u'} P_t(s_t = s, u_t = u; \theta) R(s, u)$$

Numerical optimization: find the gradient w.r.t. θ and take a step in the gradient direction.

$$\frac{\partial U}{\partial \theta_i} = \sum_{t=0}^H \sum_{s, u} \frac{\partial P_t}{\partial \theta_i}(s_t = s, u_t = u; \theta) R(s, u)$$

Using the chain rule we obtain the following recursive procedure:

$$P_t(s_t = s, u_t = u; \theta) = \sum_{s_{-}, u_{-}} P_{t-1}(s_{t-1} = s_{-}, u_{t-1} = u_{-}; \theta) T(s_{-}, u_{-}, s) \pi(u; \theta)$$

$$\frac{\partial P_t}{\partial \theta_i}(s_t = s, u_t = u; \theta) = \sum_{s_{-}, u_{-}} \frac{\partial P_{t-1}}{\partial \theta_i}(s_{t-1} = s_{-}, u_{t-1} = u_{-}; \theta) T(s_{-}, u_{-}, s) \pi(u; \theta) + P_{t-1}(s_{t-1} = s_{-}, u_{t-1} = u_{-}; \theta) T(s_{-}, u_{-}, s) \frac{\partial \pi}{\partial \theta_i}(u; \theta)$$

Computationally impractical for most large state/action spaces!

Finite differences --- generic points

Locally around our current estimate θ , we can approximate the utility $U(\theta^{(i)})$ at an alternative point $\theta^{(i)}$ with a linear approximation:

$$U(\theta) \approx U(\theta) + g^T (\theta^{(i)} - \theta)$$

Let's say we have a set of small perturbations $\theta^{(0)}, \theta^{(1)}, \dots, \theta^{(m)}$ for which we are willing to evaluate the utility $U(\theta^{(i)})$. We can get an estimate of the gradient through solving the following set of equations for the gradient g through least squares:

$$\begin{aligned} U(\theta^{(0)}) &= U(\theta) + (\theta^{(0)} - \theta)^T g \\ U(\theta^{(1)}) &= U(\theta) + (\theta^{(1)} - \theta)^T g \\ &\dots \\ U(\theta^{(m)}) &= U(\theta) + (\theta^{(m)} - \theta)^T g \end{aligned}$$

Policy gradient

	Deterministic		Stochastic	
	Known Dynamics	Unknown Dynamics	Known Dynamics	Unknown Dynamics
Analytical	OK Taking derivatives---potentially time consuming and error-prone	N/A	OK Often computationally impractical	N/A

Issue in stochastic setting

- Noise could dominate the gradient evaluation



"Fixing" the randomness

- Intuition by example: *wind influence on a helicopter is stochastic, but if we assume the same wind pattern across trials, this will make the different choices of θ more readily comparable.*
- General instantiation:
 - Fix the random seed
 - Result: deterministic system
- ****If**** the stochasticity can be captured in an appropriate way, this will result in a significantly more efficient gradient computation
 - Details of "appropriate": Ng & Jordan, 2000

Likelihood ratio method

Policy gradient

	Deterministic		Stochastic	
	Known Dynamics	Unknown Dynamics	Known Dynamics	Unknown Dynamics
Analytical	OK Taking derivatives--- potentially time consuming and error-prone	N/A	OK Often computationally impractical	N/A
Finite differences	OK Sometimes computationally more expensive than analytical	OK	OK N = #roll-outs: Naive: $O(N^{1/4})$, or $O(N^{2/5})$ Fix random seed: $O(N^{1/2})$ [1]	Same as known dynamics, but no fixing of random seed.

[1] P. Glynn, "Likelihood ratio gradient estimation: an overview," in *Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, 1987*, pp. 366–375.

Likelihood ratio method

Likelihood ratio method

- Assumption:
 - Stochastic policy $\pi_{\theta}(u_t | s_t)$
 - Stochasticity:
 - Required for the methodology
 - + Helpful to ensure exploration
 - - Optimal policy is often not stochastic (though it can be!!)

Likelihood ratio method

Likelihood ratio method

Likelihood ratio method

$$\begin{aligned}\nabla_{\theta} \log P(\tau^{(i)}; \theta) &= \nabla_{\theta} \log \left[\prod_{t=0}^H \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_{\theta}(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right] \\ &= \nabla_{\theta} \left[\sum_{t=0}^H \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^H \log \pi_{\theta}(u_t^{(i)} | s_t^{(i)}) \right] \\ &= \nabla_{\theta} \sum_{t=0}^H \log \pi_{\theta}(u_t^{(i)} | s_t^{(i)}) \\ &= \sum_{t=0}^H \underbrace{\nabla_{\theta} \log \pi_{\theta}(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}\end{aligned}$$

Likelihood ratio method

We let τ denote a state-action sequence $s_0, u_0, \dots, s_H, u_H$. We overload notation: $R(\tau) = \sum_{t=0}^H R(s_t, u_t)$.

$$U(\theta) = \mathbb{E} \left[\sum_{t=0}^H R(s_t, u_t); \pi_{\theta} \right] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

In our new notation, our goal is to find θ :

$$\max_{\theta} U(\theta) = \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

Likelihood ratio method

The following expression provides us with an unbiased estimate of the gradient, and we can compute it without access to a dynamics model:

$$\hat{g} = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

Here:

$$\nabla_{\theta} \log P(\tau^{(i)}; \theta) = \sum_{t=0}^H \underbrace{\nabla_{\theta} \log \pi_{\theta}(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}$$

Unbiased means:

$$\mathbb{E}[\hat{g}] = \nabla_{\theta} U(\theta)$$

We can obtain a gradient estimate from a single trial run! While the math we did is sound (and constitutes the commonly used derivation), this could seem a bit surprising at first. Let's perform another derivation which might give us some more insight.

Likelihood ratio method

$$U(\theta) = \sum_{\tau} P(\tau; \theta) R(\tau)$$

Taking the gradient w.r.t. θ gives

$$\begin{aligned}\nabla_{\theta} U(\theta) &= \nabla_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau) \\ &= \sum_{\tau} \nabla_{\theta} P(\tau; \theta) R(\tau) \\ &= \sum_{\tau} \frac{P(\tau; \theta)}{P(\tau; \theta)} \nabla_{\theta} P(\tau; \theta) R(\tau) \\ &= \sum_{\tau} P(\tau; \theta) \frac{\nabla_{\theta} P(\tau; \theta)}{P(\tau; \theta)} R(\tau) \\ &= \sum_{\tau} P(\tau; \theta) \nabla_{\theta} \log P(\tau; \theta) R(\tau)\end{aligned}$$

Approximate with the empirical estimate for m sample paths under policy π_{θ} :

$$\nabla_{\theta} U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$