Adaptive Control and Reinforcement Learning (16-899C, S09) Lecture #7 (Feb. 5, 2009)

Trajectory Optimization / DDP / Learning Models

Lecturer: Chris Atkeson *Scribe: Mehmet R Dogar*

1 Trajectory Optimization

Scalar Example:

- Initial state x_0
- Control mega-vector $U = (u_0, u_1, \ldots)$ where u_0 is all the controls on the first step, u_1 is all the controls on the second step, and so on.
- Our score is a function of the mega-vector. It is the sum of one-step costs: $S(U) = \sum L(x_i, u_i)$.
- Gradient is given by $\frac{\partial S(U)}{\partial u_i}$, and the Hessian is given by $\frac{\partial^2 S(U)}{\partial u_i \partial u_j}$.
- Let's say we want to do gradient descent. Can we eficiently compute the gradient or the Hessian? When we say "efficient", is there a special structure to the gradient or the Hessian that we can exploit? Yes there is.
- For $i < j$, since we can't change the past:
  $\frac{\partial L_i}{\partial u_j} = 0$
- For $i = j$
  $\frac{\partial L_i}{\partial u_i} = L_u$
- For $i > j$, using the chain rule to find $\frac{\partial L_i}{\partial u_j}$
  $\frac{\partial L_i}{\partial u_j} = \frac{\partial L_i}{\partial x_j}\frac{\partial x_i}{\partial u_j}$
  $\frac{\partial x_i}{\partial u_j} = \frac{\partial x_{j+1}}{\partial u_j}\frac{\partial x_{j+2}}{\partial x_{j+1}}\cdots\frac{\partial x_i}{\partial x_{i-1}}$
- Is there more structure? Yes.
  If we define $V_i = \sum_i^n L(x_k, u_k)$ then
  $\frac{\partial s(U)}{\partial u_k} = \frac{\partial L_k}{\partial u_k} + \frac{\partial V_{k+1}}{\partial x_{k+1}}\frac{\partial x_{k+1}}{\partial u_k}$

2 Regulator Tasks

- Examples: balance a pole, move at a constant velocity.
- A reasonable starting point is a Linear Quadratic Regulator (LQR controller).

- Might have nonlinear dynamics $x_{k+1} = f(x_k, u_k)$, but since stay around x_d, can locally linearize $x_{k+1} = Ax_k + Bu_k$.
- Might have complex scoring function $c(x, u)$, but can locally approximate with a quadratic model $c \approx x^T Qx + u^T Ru$. (Why do we use a quadratic to approximate instead of a linear approximation? Because we want it to have a bottom.)
- dlqr() does these for you in Matlab.

Linearization Example: Pendulum

- $I\ddot{\theta} = -mgl\sin(\theta) - \mu\dot{\theta} + \tau$
- We need to (i) linearize, (ii) discretize time, (iii) vectorize.
- Linearization:

  Only nonlinear term is $\sin(\theta)$ and we know that $\sin(\theta) = \theta$ for sufficiently small θ. Hence $I\ddot{\theta} = -mgl\theta - \mu\dot{\theta} + \tau$
- Discretize time and vectorize:
  $$\begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix}_{k+1} = \begin{pmatrix} 1 & T \\ \frac{-mglT}{I} & \frac{1-\mu T}{I} \end{pmatrix} \begin{pmatrix} \theta \\ \dot{\theta} \end{pmatrix}_k + \begin{pmatrix} 0 \\ T/I \end{pmatrix}_k (\tau)$$

3 LQR Derivation

- Assume $V()$ is quadratic: $V_{k+1}(x) = x^T V_{xx:k+1} x$
- Cost of executing action u in state x is the immediate cost plus the value of the next state: $C(x, u) = x^T Qx + u^T Ru + (Ax + Bu)^T V_{xx:k+1}(Ax + Bu)$.
- We want $\frac{\partial C}{\partial u} = 0$.
- Then we have: $B^T V_{xx:k+1} Ax = (B^T V_{xx:k+1} B + R)u$.
- Hence, we have a linear controller: $u = Kx$
- where $K = -(B^T V_{xx:k+1} B + R)^{-1} B^T V_{xx:k+1} A$
- and $V_{xx:k} = A^T V_{xx:k+1} A + Q + A^T V_{xx:k+1} BK$.

If we relate this to the gradient calculation that we did earlier:

$\frac{\partial x_i}{\partial u_j} = BA \ldots A$

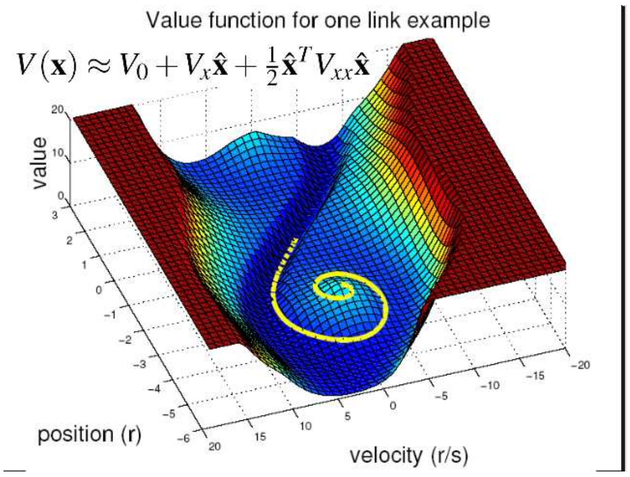

Figure 1: Value function for one-link example.

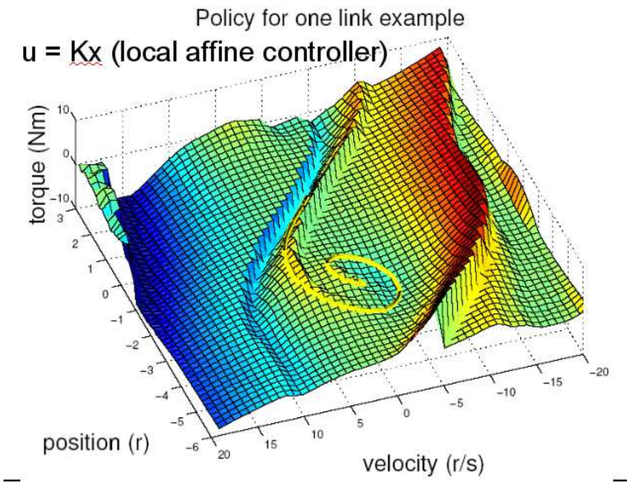

Figure 2: Policy for one-link example.

4 Trajectory Optimization with Differential Dynamic Programming (DDP)

Local models about (x_p, u_p):

- $V(x) \approx V_0 + V_x\hat{x} + \frac{1}{2}\hat{x}^T V_{xx}\hat{x}$
- $f(x,u) \approx f_0 + f_x\hat{x} + f_u\hat{u} + \frac{1}{2}\hat{x}^T f_{xx}\hat{x} + \hat{x}^T f_{xu}\hat{u} + \frac{1}{2}\hat{u}^T f_{uu}\hat{u}$
- $L(x,u) \approx L_0 + L_x\hat{x} + L_u\hat{u} + \frac{1}{2}\hat{x}^T L_{xx}\hat{x} + \hat{x}^T L_{xu}\hat{u} + \frac{1}{2}\hat{u}^T L_{uu}\hat{u}$

Value function for one-link example: see Fig. 1.

Policy for one-link example: see Fig. 2.

5 Q function

- x: state, u: control or action
- Dynamics: $x_{k+1} = f(x_k, u_k)$
- Cost function: $L(x, u)$
- Value function $V(x) = \sum L(x, u)$
- Q function $Q(x, u) = L(x, u) + V(f(x, u))$
- Bellman's Equation $V(x) = \min_u Q(x, u)$
- Policy/control law: $u(x) = \arg\min_u Q(x, u)$

6 Propagating local models along a trajectory

DDP gradient version:

- $V_{x:k-1} = Q_x = L_x + V_x f_x$
- $\Delta u = Q_u = L_u + V_x f_u$

DDP version:

- $Q_x = L_x + V_x f_x$
- $Q_u = L_u + V_x f_u$
- $Q_{xx} = L_{xx} + V_x f_{xx} + (f_x)^T V_{xx} f_x$
- $Q_{ux} = L_{ux} + V_x f_{ux} + (f_u)^T V_{xx} f_x$
- $Q_{uu} = L_{uu} + V_x f_{uu} + (f_u)^T V_{xx} f_u$
- $\Delta u = (Q_{uu})^{-1} Q_u$
- $K = (Q_{uu})^{-1} Q_{ux}$
- $V_{x_{k-1}} = Q_x - Q_u K$
- $V_{xx_{k-1}} = Q_{xx} - Q_{xu} K$

7 Levenberg Marquardt

Suppose we have $y = f(x)$ and we want to minimize

$\min_x(s = y^T y/2)$

The gradient is given by

$\frac{\partial s}{\partial x} = (\frac{\partial f}{\partial x})^T y = J^T y$

and the Hessian is

$\frac{\partial^2 s}{\partial^2 x} = H = (\frac{\partial^2 f}{\partial^2 x})y + J^T J$

Then we can do second order gradient descent using $\Delta x = H^{-1} J^T y$. But a problem is that H may not be positive definite.

Solution: $\Delta x = (H + \lambda I)^{-1} J^T y$

- As λ gets smaller this is the second order approach and the search is faster but not guaranteed to minimize.
- As λ gets larger this is the first order approach ($\Delta x = J^T y/\lambda$), so guaranteed to minimize, but slower.
- You can adjust λ on-the-fly, according to how good you are doing.

Trick 2: $H \approx J^T J$.

$J^T J$ is always at least semi positive-definite.

Levenberg Marquardt-like DDP:

- $\Delta u = (Q_{uu} + \lambda I)^{-1} Q_u$
- $K = (Q_{uu} + \lambda I)^{-1} Q_{ux}$
- Drop f_{xx}, f_{xu}, f_{ux}, and f_{uu} terms.

Other tricks:

- If Δu fails, try $\epsilon \Delta u$.
- Just optimize last part of trajectory.

8 Learning to balance a pole

It's a regulator.

What could be learned?

- J
- model: A,B. (In Machine Learning literature $f(x,u)$, i.e. learning dynamics). This is "Indirect AC Learning".
- K (In ML literature this is policy learning). This is "Direct AC Learning".

Indirect AC Learning is good because you can use the model even if the goal changes. But planners are generally not that great, so sometimes directly having the policy might be better.

In the study that Chris showed the video of, the robot mainly learned the model.

Why not do trajectory learning? Maybe the robot is not capable of doing exactly what the teacher did.

Optimization based learning:

- Regulator at top.
- Learn model (parametric, LWR).
- Learn optimization criterion.
- Use demonstration as initial trajectory.
- Reoptimize using new model, ignore past performance otherwise.

Trying to learn only the model may not be sufficient either. It was not sufficient in the case Chris showed. The problem was the residual steady-state error, where the robot can not improve its model anymore. Needed direct goal-based learning as well.

Bad models:

- Trajectory must always match model.
- Integrating bad models generates garbage and stability problems.
- Value functions are inaccurate as well.
- Optimization-based planners greedily take advantage of modeling errors.
- Plan with model violations:

  $c(x,u) += \lambda\,|x_{k+1} - f(x_k,u_k)|^2$

  If you are confident of your model you can use a large λ; if you are not confident you can use a smaller λ.

  Drew's comment: "Does not that let the planner be more greedy in taking advantage of modeling errors?"