Optimal Control, Trajectory Optimization, and Planning

CS 294-112: Deep Reinforcement Learning
Week 2, Lecture 2
Sergey Levine
Announcements

1. Assignment 1 will be out next week

2. Friday section
   • Review of automatic differentiation, SGD, training neural nets
   • Try the MNIST TensorFlow tutorial – if you’re having trouble, come to the section!
   • Fri 1/27 at 10 am
   • Sutardja Dai Hall 240
   • Chelsea Finn will teach the section
Overview

1. Last lecture: imitation learning from a human teacher

2. Today: can the machine make its own decisions?
   a. How can we choose actions under perfect knowledge of the system dynamics?
   b. Optimal control, trajectory optimization, planning

3. Next week: how can we learn unknown dynamics?

4. How can we then also learn policies? (e.g. by imitating optimal control)
Today’s Lecture

1. Making decisions under known dynamics
   • Definitions & problem statement
2. Trajectory optimization: backpropagation through dynamical systems
3. Linear dynamics: linear-quadratic regulator (LQR)
4. Nonlinear dynamics: differential dynamic programming (DDP) & iterative LQR
5. Discrete systems: Monte-Carlo tree search (MCTS)
6. Case study: imitation learning from MCTS
   • Goals:
     • Understand the terminology and formalisms of optimal control
     • Understand some standard optimal control & planning algorithms
Terminology & notation

- $x_t$ - state
- $o_t$ - observation
- $u_t$ - action

Cost function: $c(x_t, u_t)$
Reward function: $r(x_t, u_t)$

Minimize

$$\min_{u_1, \ldots, u_T} \sum_{t=1}^{T} \log p(o_t | x_t, u_t) \cdot \text{tiger}_t \cdot u_1 \cdot f(x_t, u_t, u_{t-1})$$
Trajectory optimization

$$\min_{u_1,\ldots,u_T} \sum_{t=1}^{T} c(x_t, u_t) \quad \text{s.t.} \quad x_t = f(x_{t-1}, u_{t-1})$$

$$\min_{u_1,\ldots,u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + \cdots + c(f(f(\ldots)\ldots), u_T)$$

usual story: differentiate via backpropagation and optimize!

need \( \frac{df}{dx_t}, \frac{df}{du_t}, \frac{dc}{dx_t}, \frac{dc}{du_t} \)

in practice, it really helps to use a 2nd order method!
Shooting methods vs collocation

shooting method: optimize over actions only

$$\min_{u_1,\ldots,u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + \cdots + c(f(f(\cdots) \cdots), u_T)$$
Shooting methods vs collocation

collocation method: optimize over actions and states, with constraints

$$\min_{u_1, \ldots, u_T, x_1, \ldots, x_T} \sum_{t=1}^{T} c(x_t, u_t) \text{ s.t. } x_t = f(x_{t-1}, u_{t-1})$$
Linear case: LQR

\[ \min_{u_1, \ldots, u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + \cdots + c(f(f(\ldots)), u_T) \]

\[ f(x_t, u_t) = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t \]

linear

\[ c(x_t, u_t) = \frac{1}{2} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T C_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T c_t \]

quadratic
Linear case: LQR

\[
\min_{u_1, \ldots, u_T} \ c(x_1, u_1) + c(f(x_1, u_1), u_2) + \cdots + c(f(f(\cdots), \cdots), u_T)
\]

\[
c(x_t, u_t) = \frac{1}{2} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T C_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T c_t
\]

\[
f(x_t, u_t) = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t
\]

Base case: solve for \( u_T \) only

\[
Q(x_T, u_T) = \text{const} + \frac{1}{2} \begin{bmatrix} x_T \\ u_T \end{bmatrix}^T C_T \begin{bmatrix} x_T \\ u_T \end{bmatrix} + \begin{bmatrix} x_T \\ u_T \end{bmatrix}^T c_T
\]

\[
\nabla_{u_T} Q(x_T, u_T) = C_{u_T,x_T} x_T + C_{u_T,u_T} u_T + c_{u_T}^T = 0
\]

\[
u_T = -C_{u_T,u_T}^{-1} (C_{u_T,x_T} x_T + c_{u_T}) \quad u_T = K_T x_T + k_T \quad k_T = -C_{u_T,u_T}^{-1} c_{u_T}
\]
Linear case: LQR

\[ u_T = K_T x_T + k_T \quad K_T = -C_{u_T,u_T}^{-1} C_{u_T,x_T} \quad k_T = -C_{u_T,u_T}^{-1} c_{u_T} \]

\[ Q(x_T, u_T) = \text{const} + \frac{1}{2} \begin{bmatrix} x_T \\ u_T \end{bmatrix}^T C_T \begin{bmatrix} x_T \\ u_T \end{bmatrix} + \begin{bmatrix} x_T \\ u_T \end{bmatrix}^T c_T \]

Since \( u_T \) is fully determined by \( x_T \), we can eliminate it via substitution!

\[ V(x_T) = \text{const} + \frac{1}{2} \begin{bmatrix} x_T \\ K_T x_T + k_T \end{bmatrix}^T C_T \begin{bmatrix} x_T \\ K_T x_T + k_T \end{bmatrix} + \begin{bmatrix} x_T \\ K_T x_T + k_T \end{bmatrix}^T c_T \]

\[ V(x_T) = \frac{1}{2} x_T^T C_{x_T,x_T} x_T + \frac{1}{2} x_T^T C_{x_T,u_T} K_T x_T + \frac{1}{2} x_T^T K_T^T C_{u_T,x_T} x_T + \frac{1}{2} x_T^T K_T^T C_{u_T,u_T} K_T x_T + x_T^T K_T^T C_{u_T,u_T} k_T + x_T^T c_{x_T} + x_T^T K_T^T c_{u_T} + \text{const} \]

\[ V(x_T) = \text{const} + \frac{1}{2} x_T^T V_T x_T + x_T^T v_T \quad V_T = C_{x_T,x_T} + C_{x_T,u_T} K_T + K_T^T C_{u_T,x_T} + K_T^T C_{u_T,u_T} K_T \quad v_T = c_{x_T} + C_{x_T,u_T} k_T + K_T^T C_{u_T} + K_T^T C_{u_T,u_T} K_T \]
Linear case: LQR

Solve for \( u_{T-1} \) in terms of \( x_{T-1} \)

\( u_{T-1} \) affects \( x_T \! \)

\[
f(x_{T-1}, u_{T-1}) = x_T = F_{T-1} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + f_{T-1}
\]

\[
Q(x_{T-1}, u_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T C_{T-1} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T c_{T-1} + V(f(x_{T-1}, u_{T-1}))
\]

\[
V(x_T) = \text{const} + \frac{1}{2} x_T^T V_T x_T + x_T^T v_T
\]

\[
V(x_T) = \text{const} + \frac{1}{2} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T F_{T-1}^T V_T F_{T-1} \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix} + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T F_{T-1}^T V_T f_{T-1} + \begin{bmatrix} x_{T-1} \\ u_{T-1} \end{bmatrix}^T F_{T-1}^T v_T
\]
Linear case: LQR

\[ Q(x_{T-1}, u_{T-1}) = \text{const} + \frac{1}{2} \left[ \begin{array}{c} x_{T-1} \\ u_{T-1} \end{array} \right]^T C_{T-1} \left[ \begin{array}{c} x_{T-1} \\ u_{T-1} \end{array} \right] + \left[ \begin{array}{c} x_{T-1} \\ u_{T-1} \end{array} \right]^T c_{T-1} + V(f(x_{T-1}, u_{T-1})) \]

\[ V(x_T) = \text{const} + \frac{1}{2} \left[ \begin{array}{c} x_{T-1} \\ u_{T-1} \end{array} \right]^T F_{T-1}^T V_T F_{T-1} \left[ \begin{array}{c} x_{T-1} \\ u_{T-1} \end{array} \right] + \left[ \begin{array}{c} x_{T-1} \\ u_{T-1} \end{array} \right]^T F_{T-1}^T V_T f_{T-1} + \left[ \begin{array}{c} x_{T-1} \\ u_{T-1} \end{array} \right]^T F_{T-1}^T v_t \]

\[ Q(x_{T-1}, u_{T-1}) = \text{const} + \frac{1}{2} \left[ \begin{array}{c} x_{T-1} \\ u_{T-1} \end{array} \right]^T Q_{T-1} \left[ \begin{array}{c} x_{T-1} \\ u_{T-1} \end{array} \right] + \left[ \begin{array}{c} x_{T-1} \\ u_{T-1} \end{array} \right]^T q_{T-1} \]

\[ Q_{T-1} = C_{T-1} + F_{T-1}^T V_T F_{T-1} \]

\[ q_{T-1} = c_{T-1} + F_{T-1}^T V_T f_{T-1} + F_{T-1}^T v_T \]

\[ \nabla_{u_{T-1}} Q(x_{T-1}, u_{T-1}) = Q_{u_{T-1}, x_{T-1}} x_{T-1} + Q_{u_{T-1}, u_{T-1}} u_{T-1} + q_{u_{T-1}}^T = 0 \]

\[ u_{T-1} = K_{T-1} x_{T-1} + k_{T-1} \]

\[ K_{T-1} = -Q_{u_{T-1}, u_{T-1}}^{-1} Q_{u_{T-1}, x_{T-1}} \]

\[ k_{T-1} = -Q_{u_{T-1}, u_{T-1}}^{-1} q_{u_{T-1}} \]
Linear case: LQR

Backward recursion

for $t = T$ to 1:

$$Q_t = C_t + F_t^T V_{t+1} F_t$$

$$q_t = c_t + F_t^T V_{t+1} f_t + F_t^T v_{t+1}$$

$$Q(x_t, u_t) = \text{const} + \frac{1}{2} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T Q_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T q_t$$

$$u_t \leftarrow \arg\min_{u_t} Q(x_t, u_t) = K_t x_t + k_t$$

$$K_t = -Q_{u_t,u_t}^{-1} Q_{u_t,x_t}$$

$$k_t = -Q_{u_t,u_t}^{-1} q_{u_t}$$

$$V_t = Q x_t, x_t + Q x_t, u_t K_t + K_t^T Q u_t, x_t + K_t^T Q u_t, u_t K_t$$

$$v_t = q x_t + Q x_t, u_t k_t + K_t^T Q u_t + K_t^T Q u_t, k_t$$

$$V(x_t) = \text{const} + \frac{1}{2} x_t^T V_t x_t + x_t^T v_t$$

Forward recursion

for $t = 1$ to $T$:

$$u_t = K_t x_t + k_t$$

$$x_{t+1} = f(x_t, u_t)$$

we know $x_1$!
Some useful definitions

Backward recursion

\[
\begin{align*}
Q_t &= C_t + F_t^T V_{t+1} F_t \\
q_t &= c_t + F_t^T V_{t+1} f_t + F_t^T v_{t+1} \\
Q(x_t, u_t) &= \text{const} + \frac{1}{2} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T Q_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T q_t \\

u_t &\leftarrow \arg\min_{u_t} Q(x_t, u_t) = K_t x_t + k_t \\
K_t &= -Q_{u_t,u_t}^{-1} Q_{u_t,x_t} \\
k_t &= -Q_{u_t,u_t}^{-1} q_{u_t} \\
V_t &= Q_{x_t,x_t} + Q_{x_t,u_t} K_t + K_t^T Q_{u_t,x_t} + K_t^T Q_{u_t,u_t} K_t \\
v_t &= q_{x_t} + Q_{x_t,u_t} k_t + K_t^T Q_{u_t} + K_t^T Q_{u_t,u_t} k_t \\
\frac{1}{2} x_t^T V_t x_t + x_t^T v_t &\text{ total cost from now until end if we take } u_t \text{ from state } x_t \\
V(x_t) &= \text{const} + \frac{1}{2} x_t^T V_t x_t + x_t^T v_t \\
V(x_t) &= \min_{u_t} Q(x_t, u_t)
\end{align*}
\]
Stochastic dynamics

\[ f(x_t, u_t) = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t \]

\[ x_{t+1} \sim p(x_{t+1}|x_t, u_t) \]

\[ p(x_{t+1}|x_t, u_t) = \mathcal{N} \left( F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t, \Sigma_t \right) \]

Solution: choose actions according to \( u_t = K_t x_t + k_t \)

\( x_t \sim p(x_t) \), no longer deterministic, but \( p(x_t) \) is Gaussian

no change to algorithm! can ignore \( \Sigma_t \) due to symmetry of Gaussians
(checking this is left as an exercise; hint: the expectation of a quadratic under a Gaussian has an analytic solution)
Nonlinear case: DDP/iterative LQR

Linear-quadratic assumptions:

\[
f(x_t, u_t) = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t
\]

\[
c(x_t, u_t) = \frac{1}{2} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T C_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T c_t
\]

Can we approximate a nonlinear system as a linear-quadratic system?

\[
f(x_t, u_t) \approx f(\hat{x}_t, \hat{u}_t) + \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix}
\]

\[
c(x_t, u_t) \approx c(\hat{x}_t, \hat{u}_t) + \nabla_{x_t, u_t} c(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix}^T \nabla_{x_t, u_t}^2 c(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix}
\]
Nonlinear case: DDP/iterative LQR

\[ f(x_t, u_t) \approx f(\hat{x}_t, \hat{u}_t) + \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix} \]

\[ c(x_t, u_t) \approx c(\hat{x}_t, \hat{u}_t) + \nabla_{x_t, u_t} c(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix}^T \nabla_{x_t, u_t}^2 c(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix} \]

\[ \bar{f}(\delta x_t, \delta u_t) = F_t \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix} \]
\[ \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t) \]

\[ \bar{c}(\delta x_t, \delta u_t) = \frac{1}{2} \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix}^T C_t \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix} + \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix}^T \begin{bmatrix} \delta x_t \\ \delta u_t \end{bmatrix} \]
\[ \nabla_{x_t, u_t}^2 c(\hat{x}_t, \hat{u}_t) \]
\[ \nabla_{x_t, u_t} c(\hat{x}_t, \hat{u}_t) \]

\[ \delta x_t = x_t - \hat{x}_t \]
\[ \delta u_t = u_t - \hat{u}_t \]

Now we can run LQR with dynamics \( \bar{f} \), cost \( \bar{c} \), state \( \delta x_t \), and action \( \delta u_t \)
Nonlinear case: DDP/iterative LQR

Iterative LQR (simplified pseudocode)

until convergence:

\[ F_t = \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t) \]
\[ c_t = \nabla_{x_t, u_t} c(\hat{x}_t, \hat{u}_t) \]
\[ C_t = \nabla^2_{x_t, u_t} c(\hat{x}_t, \hat{u}_t) \]

Run LQR backward pass on state \( \delta x_t = x_t - \hat{x}_t \) and action \( \delta u_t = u_t - \hat{u}_t \)

Run forward pass with real nonlinear dynamics and \( u_t = K_t (x_t - \hat{x}_t) + k_t \)

Update \( \hat{x}_t \) and \( \hat{u}_t \) based on states and actions in forward pass
Nonlinear case: DDP/iterative LQR

Why does this work?

Compare to Newton’s method for computing \( \min_x g(x) \):

until convergence:

\[
\begin{align*}
g &= \nabla_x g(\hat{x}) \\
H &= \nabla^2_x g(\hat{x}) \\
\hat{x} &\leftarrow \arg\min_x \frac{1}{2} (x - \hat{x})^T H (x - \hat{x}) + g^T (x - \hat{x})
\end{align*}
\]

Iterative LQR (iLQR) is the same idea: locally approximate a complex nonlinear function via Taylor expansion

In fact, iLQR is an approximation of Newton’s method for solving

\[
\min_{u_1, \ldots, u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + \cdots + c(f(f(\cdots), \cdots), u_T)
\]
Nonlinear case: DDP/iterative LQR

In fact, iLQR is an approximation of Newton’s method for solving

$$\min_{u_1, \ldots, u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + \cdots + c(f(\cdots), u_T)$$

To get Newton’s method, need to use second order dynamics approximation:

$$f(x_t, u_t) \approx f(\hat{x}_t, \hat{u}_t) + \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t) \left[ \begin{array}{c} \delta x_t \\ \delta u_t \end{array} \right] + \frac{1}{2} \left( \nabla_{x_t, u_t}^2 f(\hat{x}_t, \hat{u}_t) \cdot \left[ \begin{array}{c} \delta x_t \\ \delta u_t \end{array} \right] \right) \left[ \begin{array}{c} \delta x_t \\ \delta u_t \end{array} \right]$$

differential dynamic programming (DDP)
Nonlinear case: DDP/iterative LQR

\[
\hat{x} \leftarrow \arg \min_x \frac{1}{2}(x - \hat{x})^T H(x - \hat{x}) + g^T(x - \hat{x})
\]

why is this a bad idea?

until convergence:

\[
F_t = \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t)
\]

\[
c_t = \nabla_{x_t, u_t} c(\hat{x}_t, \hat{u}_t)
\]

\[
C_t = \nabla^2_{x_t, u_t} c(\hat{x}_t, \hat{u}_t)
\]

Run LQR backward pass on state \(\delta x_t = x_t - \hat{x}_t\) and action \(\delta u_t = u_t - \hat{u}_t\)

Run forward pass with real nonlinear dynamics and \(u_{tt} = K_t((x_t - \hat{x}_t) + dK_{tt})\)

Update \(\hat{x}_t\) and \(\hat{u}_t\) based on states and actions in forward pass
Additional reading

   • Original differential dynamic programming algorithm.

   • Practical guide for implementing non-linear iterative LQR.

   • Probabilistic formulation and trust region alternative to deterministic line search.
Case study: nonlinear model-predictive control

Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization

Yuval Tassa, Tom Erez and Emanuel Todorov
University of Washington

every time step:

observe the state $\mathbf{x}_t$

use iLQR to plan $\mathbf{u}_t, \ldots, \mathbf{u}_T$ to minimize $\sum_{t'=t}^{t+T} c(\mathbf{x}_{t'}, \mathbf{u}_{t'})$

execute action $\mathbf{u}_t$, discard $\mathbf{u}_{t+1}, \ldots, \mathbf{u}_{t+T}$
Synthesis of Complex Behaviors with Online Trajectory Optimization

Yuval Tassa, Tom Erez & Emo Todorov

IEEE International Conference on Intelligent Robots and Systems 2012
Discrete case: Monte Carlo tree search (MCTS)

Discrete planning as a search problem
Discrete case: Monte Carlo tree search (MCTS)

how to approximate value without full tree?

e.g., random policy
Discrete case: Monte Carlo tree search (MCTS)

can’t search all paths – where to search first?

intuition: choose nodes with best reward, but also prefer rarely visited nodes
Discrete case: Monte Carlo tree search (MCTS)

generic MCTS sketch

1. find a leaf $s_l$ using $\text{TreePolicy}(s_1)$
2. evaluate the leaf using $\text{DefaultPolicy}(s_l)$
3. update all values in tree between $s_1$ and $s_l$
take best action from $s_1$

UCT TreePolicy($s_t$)

if $s_t$ not fully expanded, choose new $a_t$
else choose child with best $\text{Score}(s_{t+1})$

$$\text{Score}(s_t) = \frac{Q(s_t)}{N(s_t)} + 2C\sqrt{\frac{2 \ln N(s_{t-1})}{N(s_t)}}$$
Additional reading

   • Survey of MCTS methods and basic summary.
Case study: imitation learning from MCTS
Case study: imitation learning from MCTS

DAgger

1. train $\pi_\theta(u_t|o_t)$ from human data $D = \{o_1, u_1, \ldots, o_N, u_N\}$
2. run $\pi_\theta(u_t|o_t)$ to get dataset $D_\pi = \{o_1, \ldots, o_M\}$
3. Choose actions label $D_\pi$ with $\pi$ acting MCTS
4. Aggregate: $D \leftarrow D \cup D_\pi$

Why train a policy?

• In this case, MCTS is too slow for real-time play
• Other reasons – perception, generalization, etc.: more on this later
What’s wrong with known dynamics?

Next time: learning the dynamics model