Optimal Control and Planning

CS 285

Instructor: Sergey Levine
UC Berkeley

Today's Lecture

model-based algorithms look very different

1. Introduction to model-based reinforcement learning

2. What if we know the dynamics? How can we make decisions?

3. Stochastic optimization methods   simple and commonly used

4. Monte Carlo tree search (MCTS)

5. Trajectory optimization   LQR and its non-linear extensions

- Goals:
    - Understand how we can perform planning with known dynamics models in discrete and continuous spaces
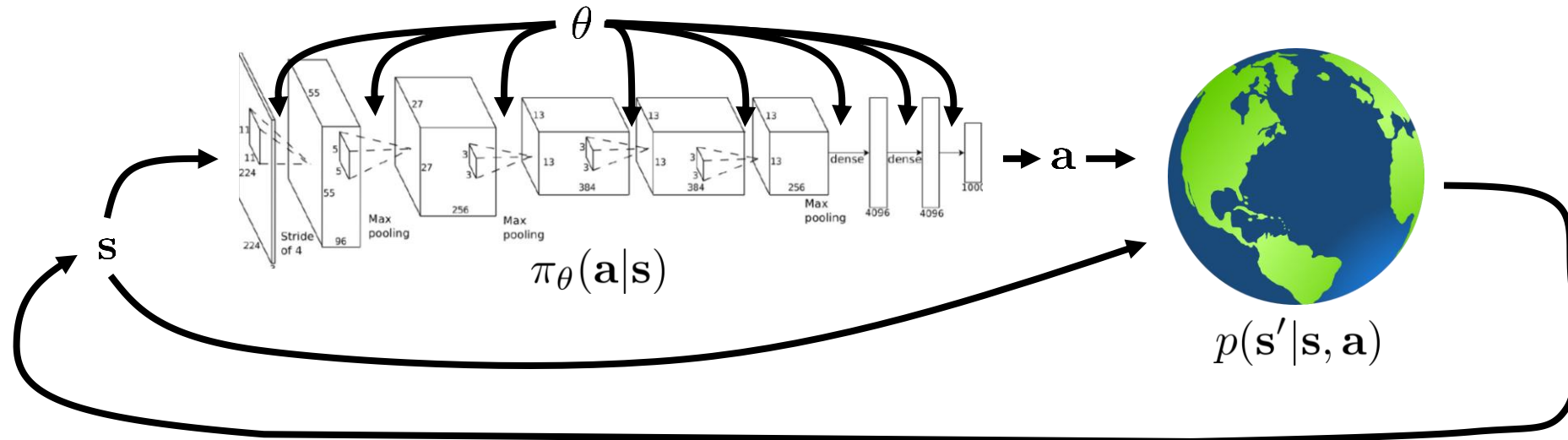
Recap: the reinforcement learning objective
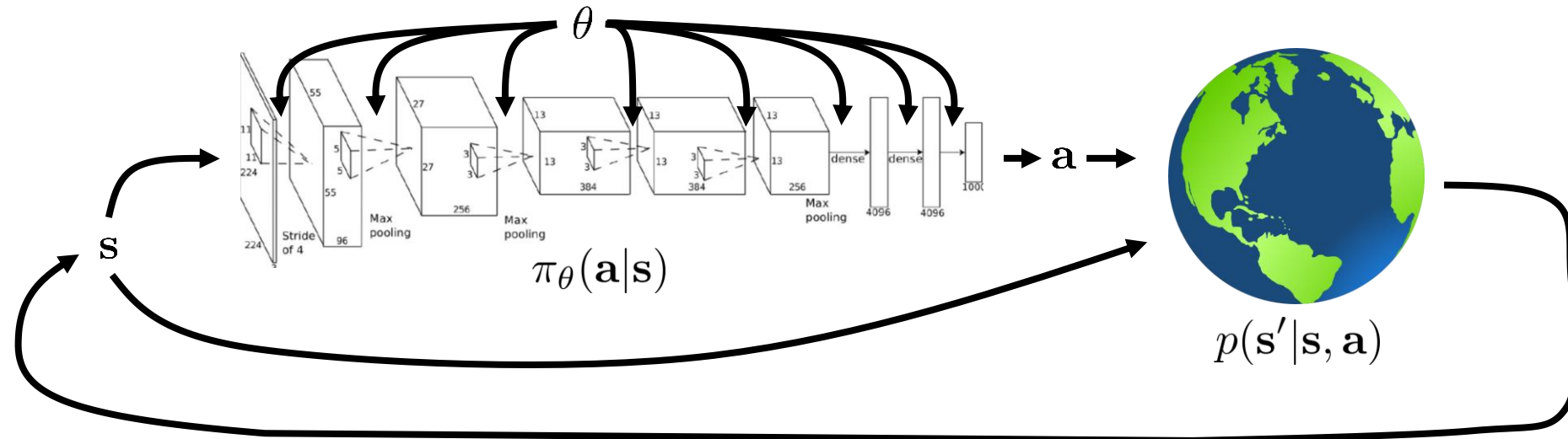
$$p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\underbrace{\phantom{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}}_{\pi_\theta(\tau)}$$

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

Recap: model-free reinforcement learning

$$p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \cancel{p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)}$$

$$\underbrace{\phantom{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}}_{\pi_\theta(\tau)}$$

assume this is unknown
don't even attempt to learn it

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

What if we knew the transition dynamics?

- Often we do know the dynamics
  1. Games (e.g., Atari games, chess, Go)
  2. Easily modeled systems (e.g., navigating a car)
  3. Simulated environments (e.g., simulated robots, video games)

- Often we can learn the dynamics
  1. System identification – fit unknown parameters of a known model
  2. Learning – fit a general-purpose model to observed transition data

  this is the focus of many model-based RL models were learn about in this course

Does knowing the dynamics make things easier?

Often, yes!

Model-based reinforcement learning

1. Model-based reinforcement learning: learn the transition dynamics, then figure out how to choose actions

2. Today: how can we make decisions if we *know* the dynamics?
    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*)
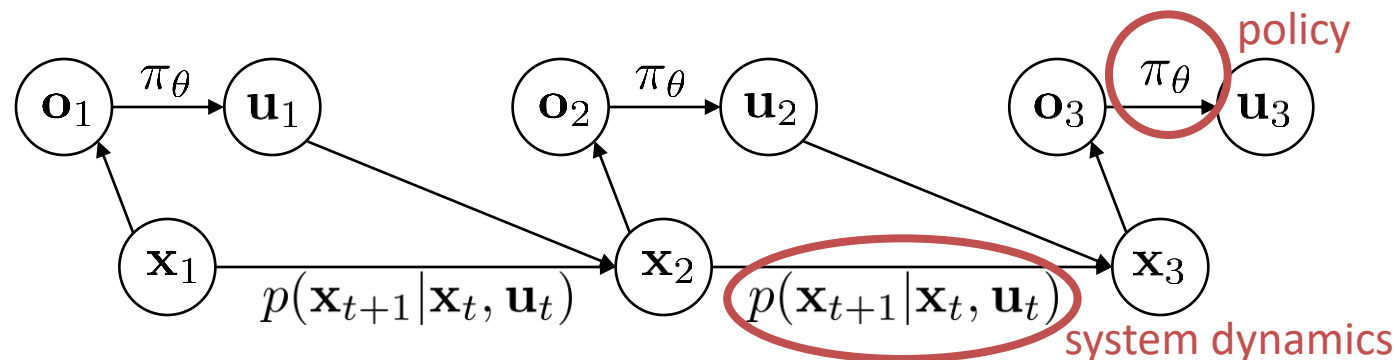
The objective
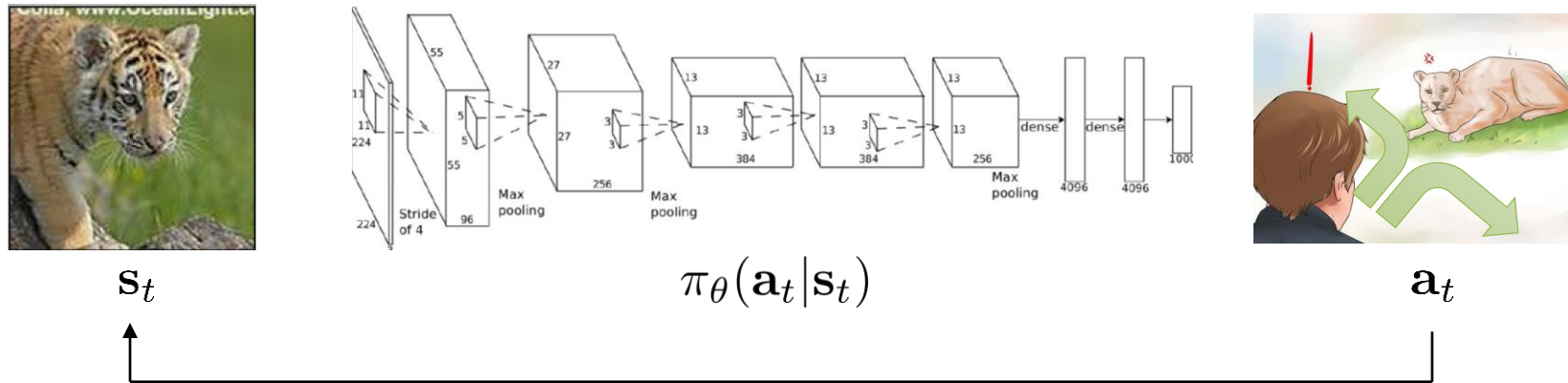
$$\mathbf{s}_t \qquad \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \qquad \mathbf{a}_t$$

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \log p(\text{eaten by tiger}|\mathbf{a}_1,\ldots,\mathbf{a}_T)$$

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \sum_{t=1}^{T} c(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{s}_t = f(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$$

so we can write this as a constrained optimization problem, where we want to minimize cost resulting from our actions for all timesteps. This is subject to the constraint that state s_t is a function of the previous state and the action we took

The deterministic case

after the environment tells the agent which state its in, the agent performs an optimization given their current state. they imagine a sequence of actions that minimize total cost. then they send these actions back to the world and they get executed

in the deterministic case, this works and you can get optimal behavior. but in the stochastic case, it doesn't and can be very suboptimal
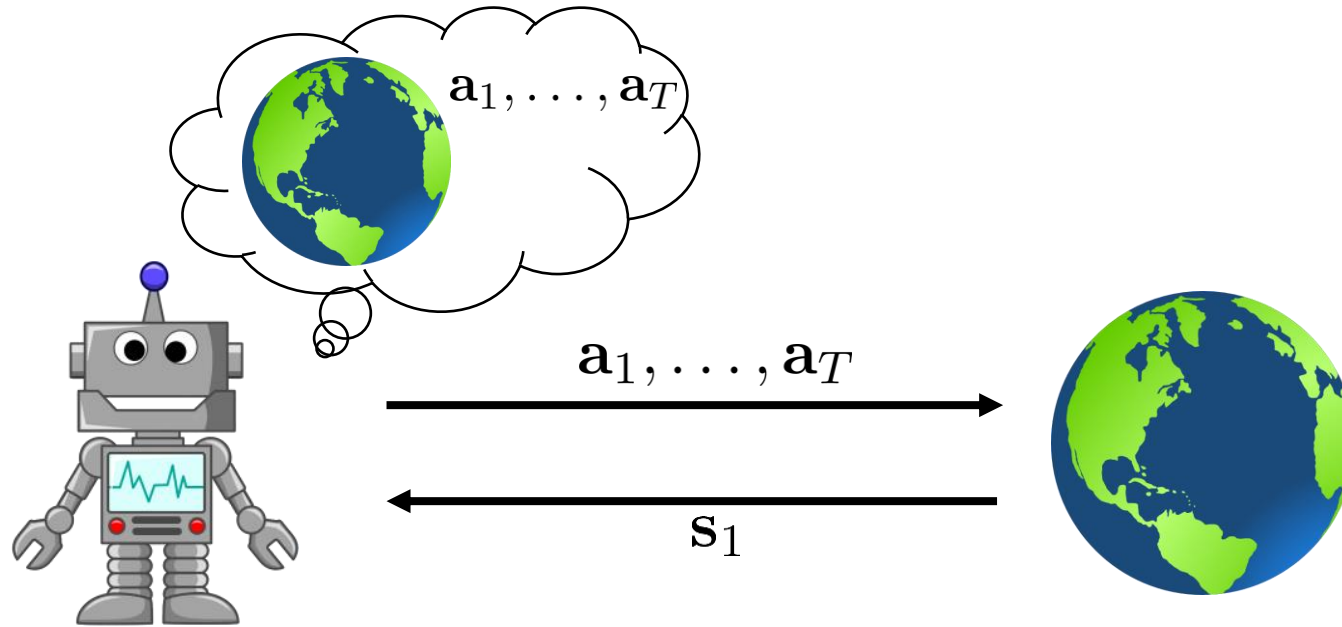
$$\mathbf{a}_1, \ldots, \mathbf{a}_T$$

$$\mathbf{a}_1, \ldots, \mathbf{a}_T$$

$$\mathbf{s}_1$$
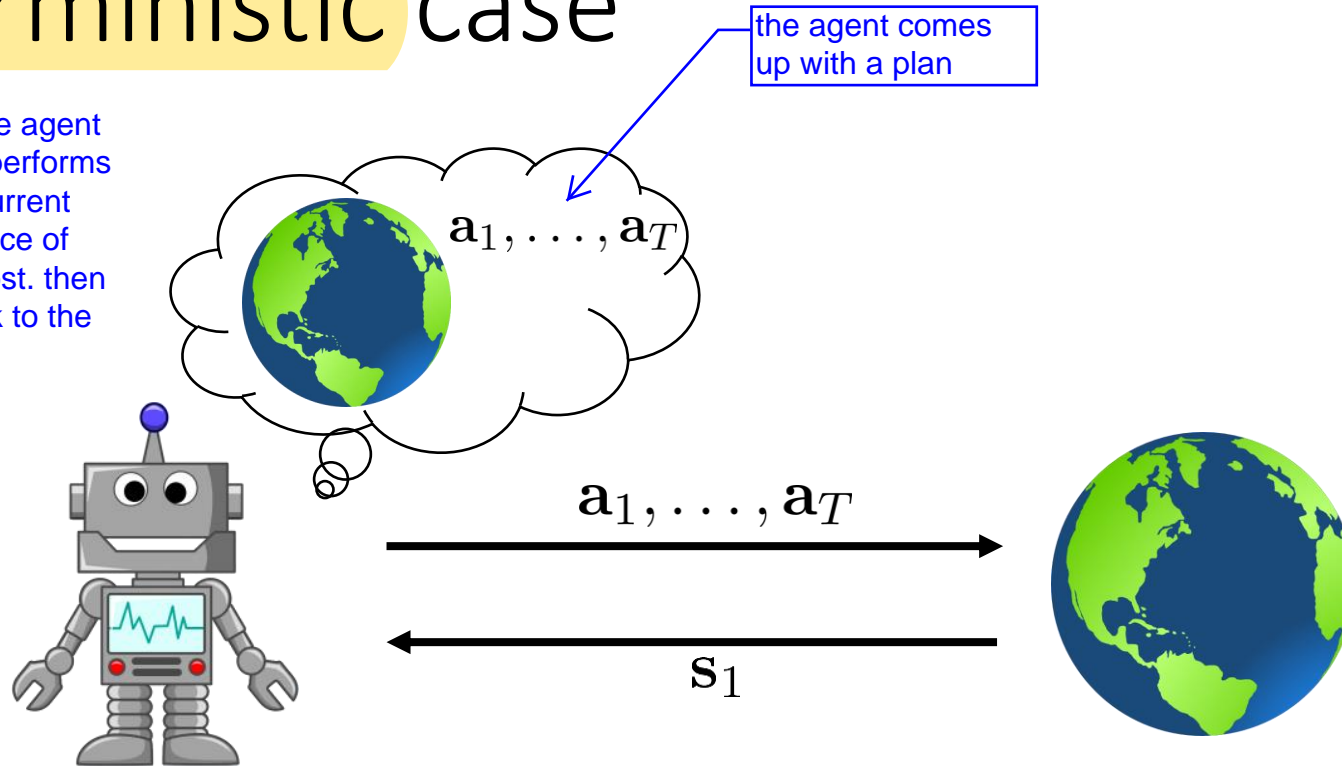
$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

this is a mistake. it should be s_t+1

The stochastic open-loop case

$$p_\theta(\mathbf{s}_1, \ldots, \mathbf{s}_T | \mathbf{a}_1, \ldots, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1,\ldots,\mathbf{a}_T} E\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{a}_1, \ldots, \mathbf{a}_T\right]$$
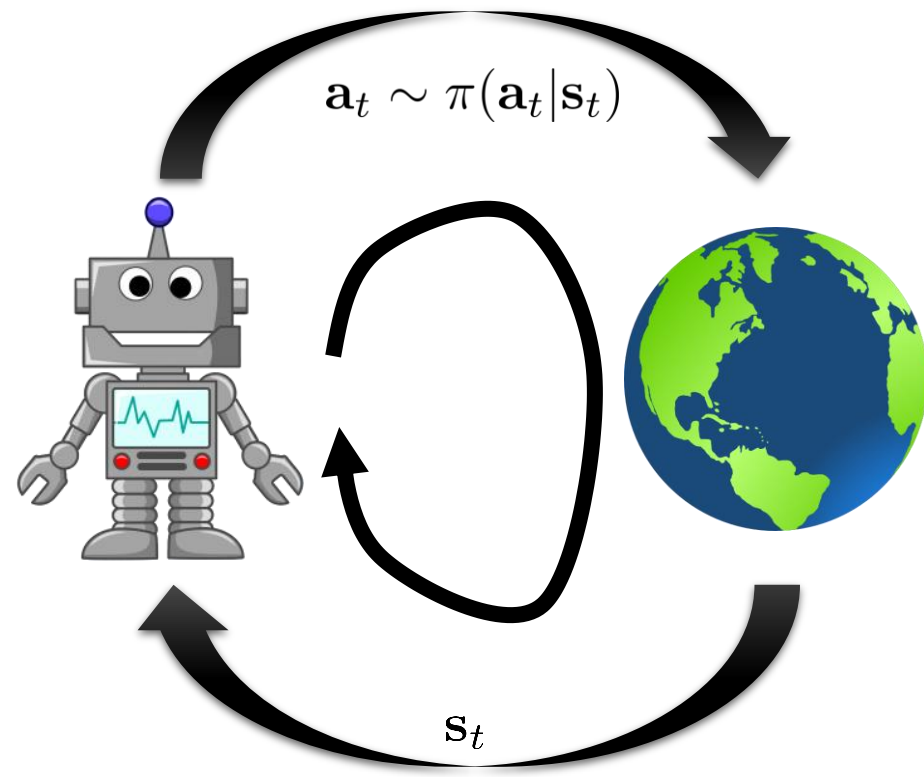
because it's stochastic, you don't know which states you'll transition to. thus, you don't necessarily want to use the plan you started with, because you could be in a state you didn't expect. Thus, you will want to update your plan along the way.

why is this suboptimal?
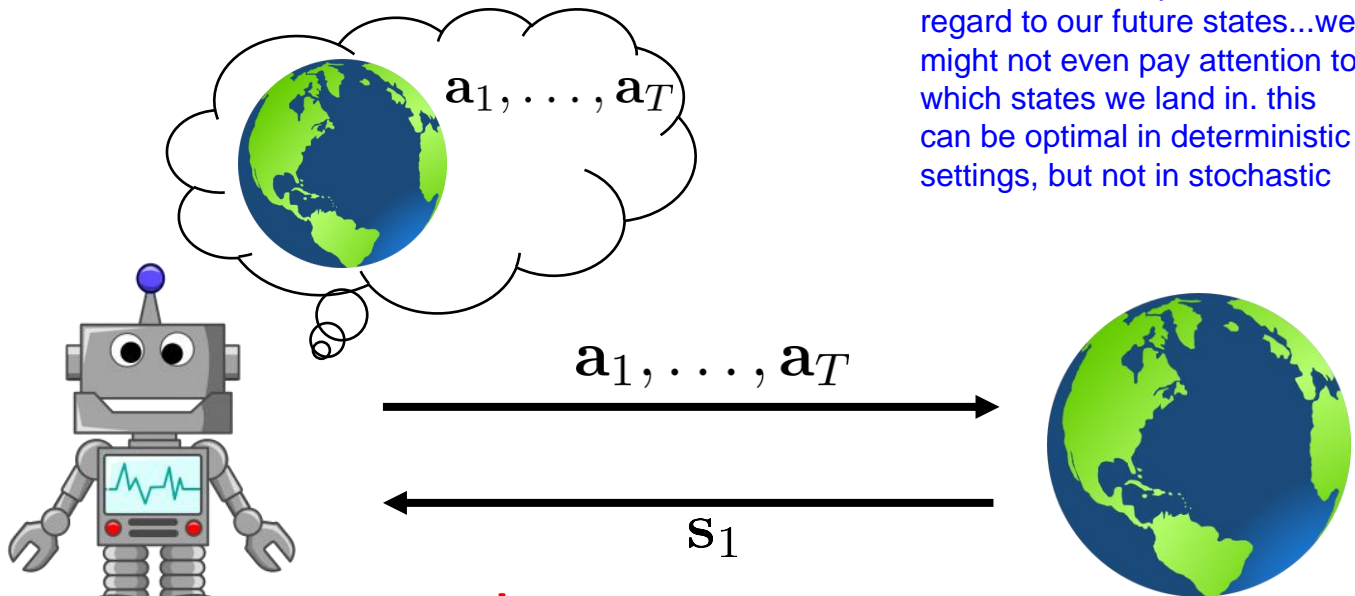
Aside: terminology

what is this "loop"?

closed-loop

open-loop

$\mathbf{a}_t \sim \pi(\mathbf{a}_t | \mathbf{s}_t)$

$\mathbf{a}_1, \ldots, \mathbf{a}_T$

$\mathbf{a}_1, \ldots, \mathbf{a}_T$

$\mathbf{s}_1$

$\mathbf{s}_t$

only sent at t = 1, then it's one-way!

The stochastic **closed**-loop case

$$\pi(\mathbf{a}_t|\mathbf{s}_t)$$

instead of sending back a
sequence of actions, you send
back a mapping from states to
actions (i.e. a policy)

$$\pi(\mathbf{a}_t|\mathbf{s}_t)$$

$$\mathbf{s}_1$$

form of π?

neural net ~ global

its "global" in that it tells the
agent what to do in every state
it could possible encounter

time-varying linear
$$\mathbf{K}_t\mathbf{s}_t + \mathbf{k}_t$$
local

find a policy that depends on
our current state. we assume
that we won't deviate too
much from our current state,
so the policy is specific to our
state or some small-region
around our state.

$$p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\pi = \arg\max_{\pi} E_{\tau \sim p(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

(more on this later)

Open-Loop Planning

But for now, open-loop planning
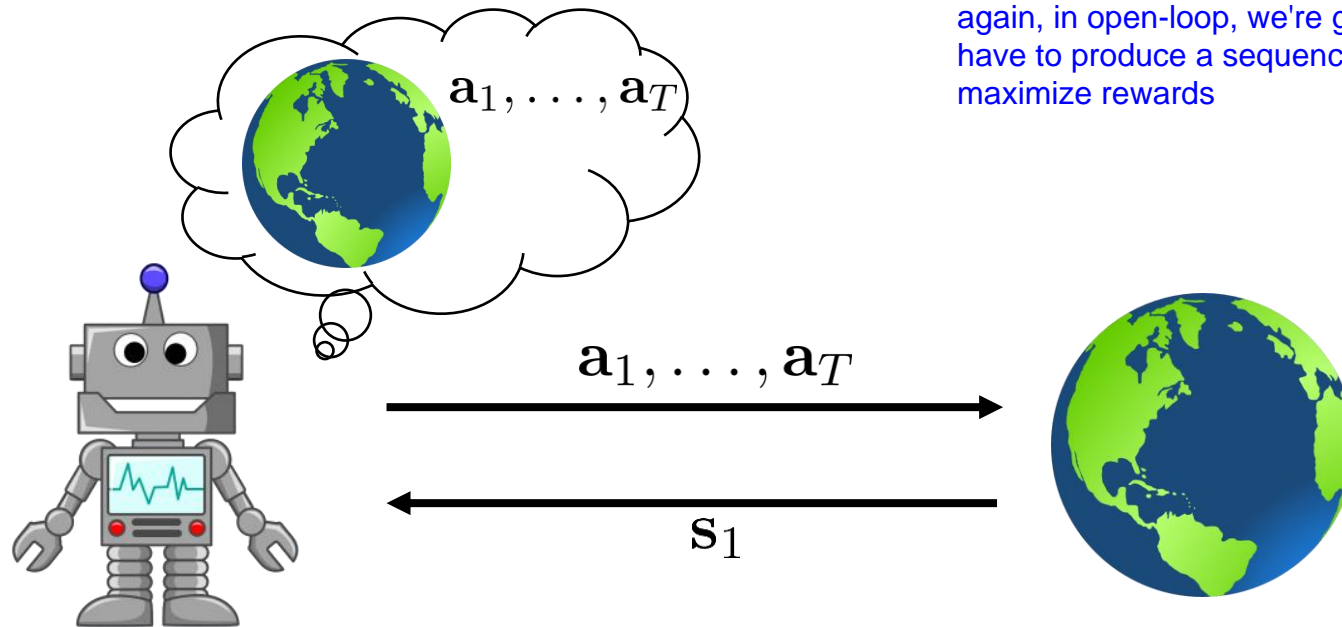
again, in open-loop, we're given a state and we have to produce a sequence of actions that maximize rewards

$$\mathbf{a}_1, \ldots, \mathbf{a}_T$$

$$\mathbf{s}_1$$

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

Stochastic optimization

we will represent the entire sequence as A. Because in stochastic opt, they don't even care about the sequences

abstract away optimal control/planning:

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} J(\mathbf{a}_1, \ldots, \mathbf{a}_T)$$

$$\mathbf{A} = \arg \max_{\mathbf{A}} J(\mathbf{A})$$

don't care what this is

simplest method: guess & check   "random shooting method"

1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)

2. choose $\mathbf{A}_i$ based on $\arg \max_i J(\mathbf{A}_i)$

choose the best action sequence of all the sequences that we attempted

for low-dimension and small horizons, this can actually work well in practice. one major advantage of this approach is that it's extremely simple to implement. it's also very efficient on modern hardware. you could run A_1 through A_N in parallel. A disadvantage is that you might not pick very good actions, because you're relying on one of the sequences being very good
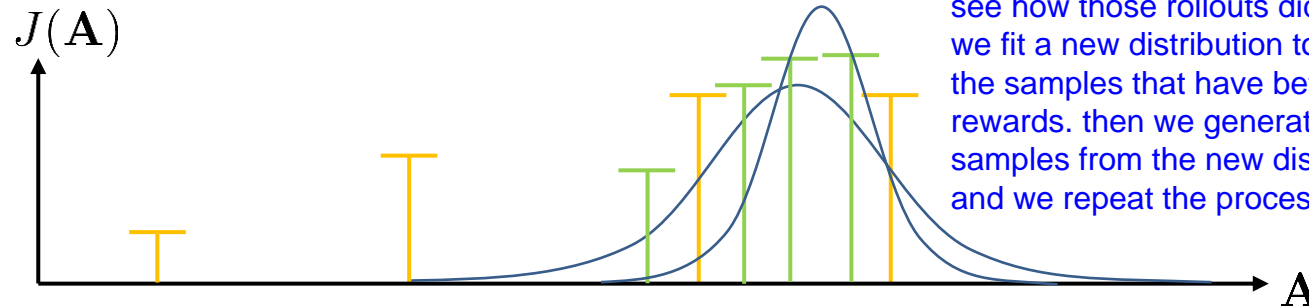
Cross-entropy method (CEM)

1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)

2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

can we do better?

we start with a uniform distribution and do some rollouts. then, we see how those rollouts did. then, we fit a new distribution towards the samples that have better rewards. then we generate more samples from the new distribution, and we repeat the process

typically use Gaussian distribution

see also: CMA-ES (sort of like CEM with momentum)

this is an extension to CEM

cross-entropy method with continuous-valued inputs:

1. sample $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from $p(\mathbf{A})$

2. evaluate $J(\mathbf{A}_1), \ldots, J(\mathbf{A}_N)$

3. pick the *elites* $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$ with the highest value, where $M < N$

pick a subset of your samples with the highest value. A rule of thumb is to to pick M where M is 10% of N

4. refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$

if you choose your distribution to be a Gaussian, you'd fit a gaussian by finding the max likelihood fit to the best M samples

What's the upside?

1. Very fast if parallelized

2. Extremely simple

3. Doesn't require model to be differentiable w.r.t. actions
4. Can work for discrete actions too by using other distribution classes

What's the problem?

we are relying on the random sampling procedure to get us good coverage.

1. Very harsh dimensionality limit

2. Only open-loop planning

if you have more than 30-60 dimensions, these methods don't work very well. I think dimensions refers to action-dimensions times the number of timesteps.

if you're doing planning, rule of thumb, 10 dimensions and 15 timesteps is about the max you can do

Discrete case: Monte Carlo tree search (MCTS)

discrete planning as a search problem

s_t

a_t

closed-loop feedback
can work for continuous and discrete space, but more
commonly used in discrete space.

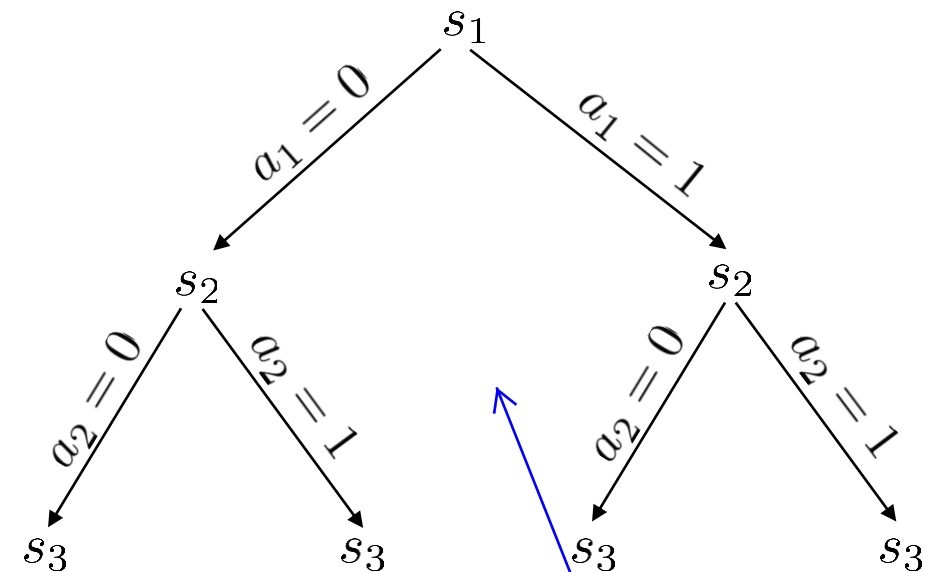this is often used in board games (i.e. AlphaGo)

MCTS is good for games of chance (e.g. poker)

here are your
possible actions

just plan out what would happen if you
took different actions at different times.
this would give you the optimal action
to take but it's exponential so is
intractable

s_1

$a_1 = 0$

$a_1 = 1$

s_2

s_2

$a_2 = 0$

$a_2 = 1$

$a_2 = 0$

$a_2 = 1$

s_3

s_3

s_3

s_3

Discrete case: Monte Carlo tree search (MCTS)

this isn't an optimal strategy, but can get pretty good. in practice this is a very good algorithm for discrete stochastic settings in the closed-loop case

how to approximate value without full tree?

so you could pick a depth, then expand the tree to that depth. then you could just run some baseline or random policy. and the value you get when you run that random policy won't be correct, and especially if you have a discount factor, the random policy will give us some idea of how good the state is
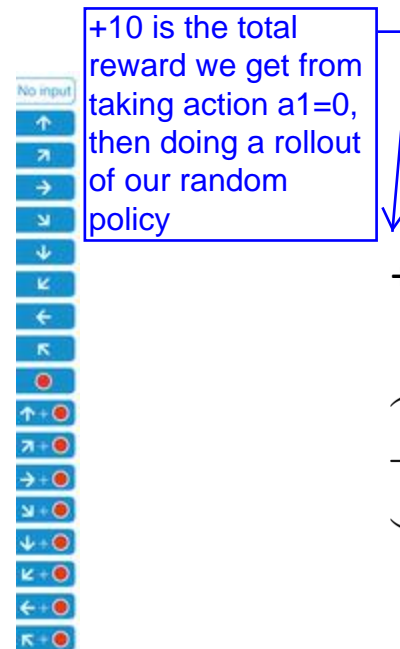
s_t

a_t

e.g., random policy

s_1

$a_1 = 0$ $a_1 = 1$

s_2 s_2

$a_2 = 0$ $a_2 = 1$ $a_2 = 0$ $a_2 = 1$

s_3 s_3 s_3 s_3
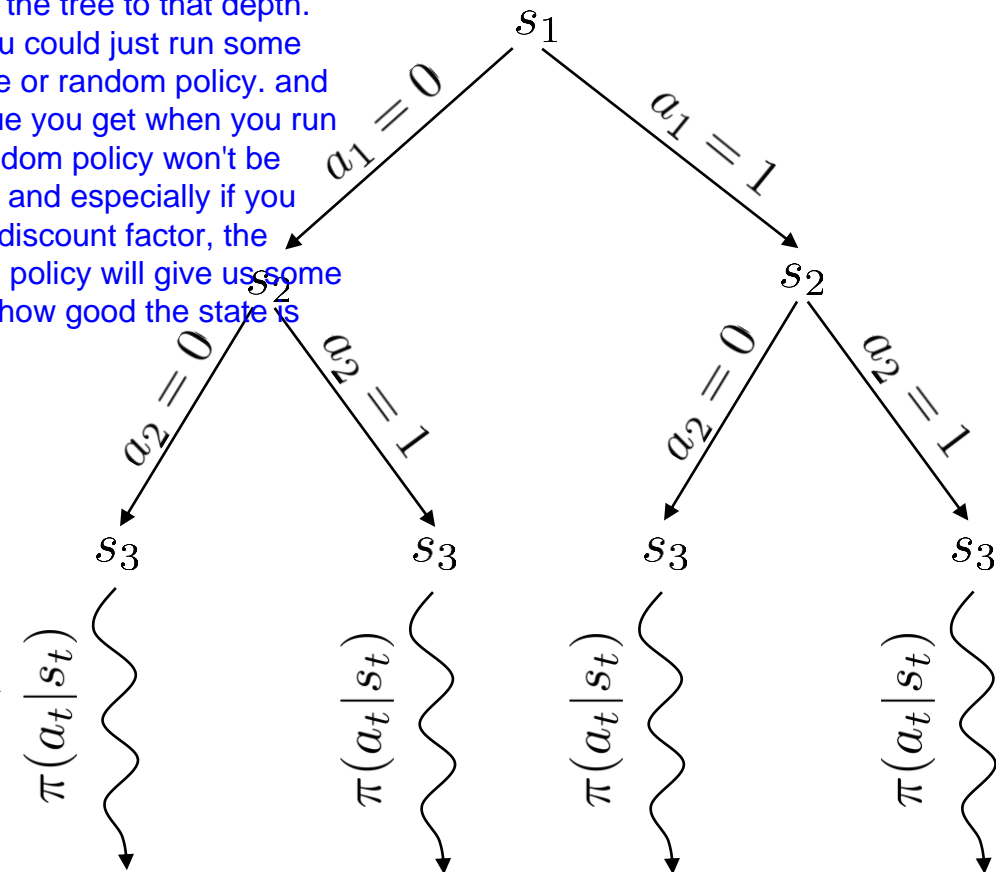
$\pi(a_t|s_t)$ $\pi(a_t|s_t)$ $\pi(a_t|s_t)$ $\pi(a_t|s_t)$

Discrete case: Monte Carlo tree search (MCTS)

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

+10 is the total reward we get from taking action a1=0, then doing a rollout of our random policy

s_1

$a_1 = 0$

$a_1 = 1$

+10

+15

s_2

s_2

$\pi(a_t|s_t)$

$\pi(a_t|s_t)$

s_t

a_t
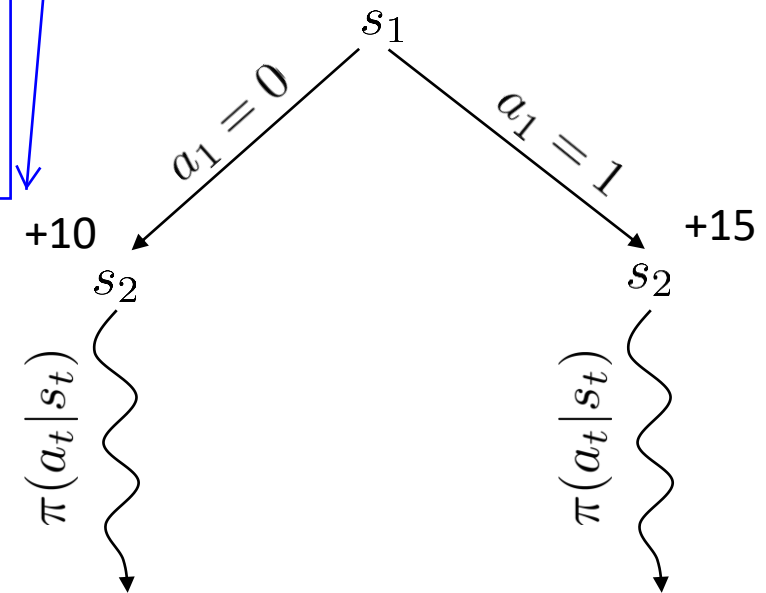
since we are in a stochastic system, if we took action a1=0 and then ran our policy again, we might not get +10 again. this is because the outcome of a1=0 is random and our policy is random. so +10 is just a sample-based estimate of the real value of taking that action

intuition: choose nodes with best reward, but also prefer rarely visited nodes

Discrete case: Monte Carlo tree search (MCTS)

generic MCTS sketch

find which leaf node to expand

N = the number of times we've visited a node

1. find a leaf s_l using TreePolicy(s_1)

Q/N = the average reward

2. evaluate the leaf using 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 Score(s_{t+1})

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

s_1

$a_1 = 0$ $a_1 = 1$

Q = 30   Q = 22
N = 3   N = 3

s_2 s_2

$a_2 = 0$ $\pi(a_t|s_t)$ $a_2 = 1$ $a_2 = 0$ $\pi(a_t|s_t)$ $a_2 = 1$

Q = 12   Q = 8   Q = 16
N = 1 s_3 s_3 N = 1 s_3 s_3 N = 1

            Q = 10
            N = 1

$\pi(a_t|s_t)$ $\pi(a_t|s_t)$ $\pi(a_t|s_t)$ $\pi(a_t|s_t)$
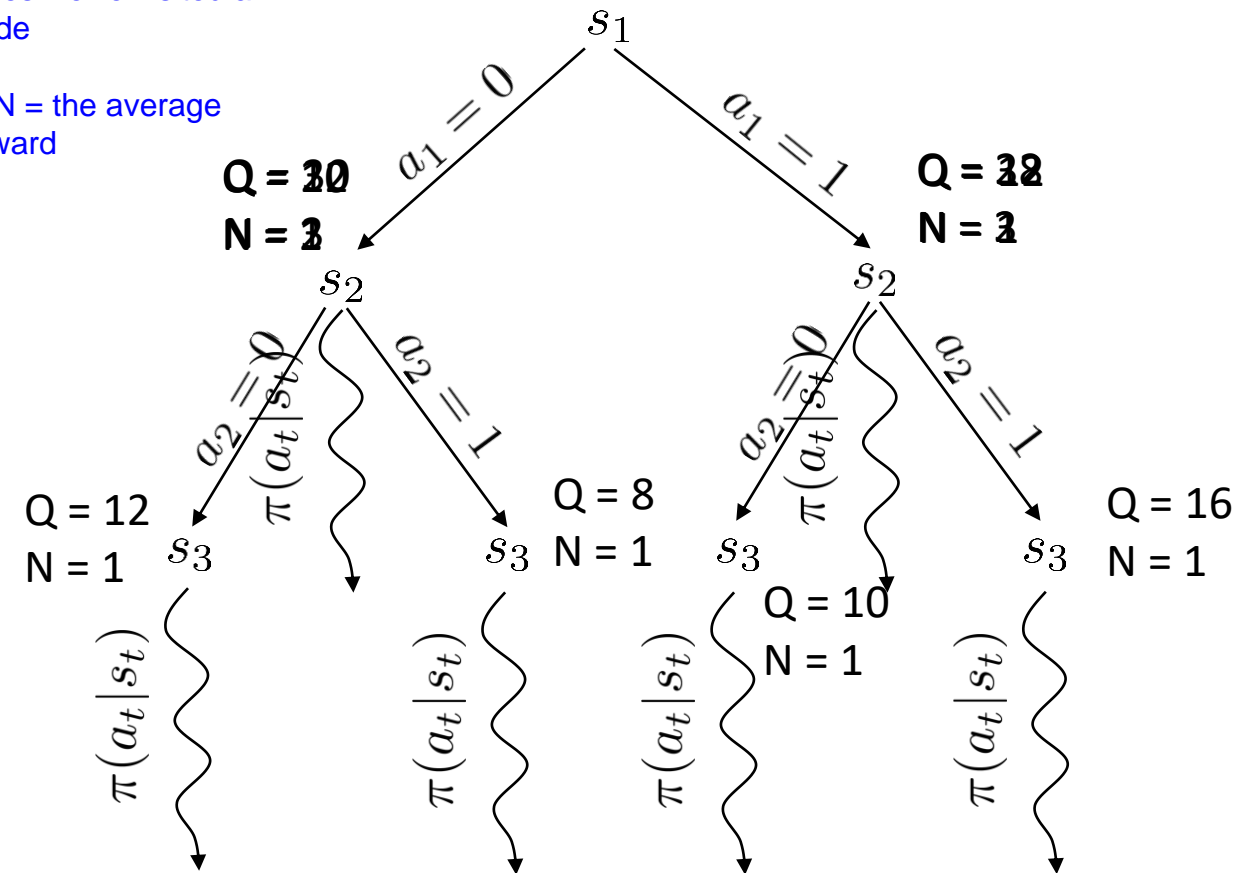
Additional reading

1. Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, Colton. (2012). A Survey of Monte Carlo Tree Search Methods.
   - Survey of MCTS methods and basic summary.

Trajectory Optimization with Derivatives

Can we use derivatives?

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

when you have constrained optimization with equality constraints, you can substitute it into the cost function to get unconstrained optimization

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$
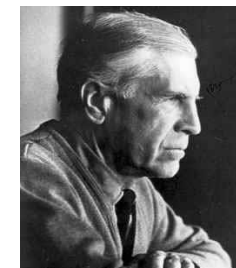
usual story: differentiate via backpropagation and optimize!

need $\dfrac{df}{d\mathbf{x}_t}, \dfrac{df}{d\mathbf{u}_t}, \dfrac{dc}{d\mathbf{x}_t}, \dfrac{dc}{d\mathbf{u}_t}$

$\mathbf{s}_t$ – state
$\mathbf{a}_t$ – action

$\mathbf{x}_t$ – state
$\mathbf{u}_t$ – action

in practice, it really helps to use a 2nd order method!

Shooting methods vs collocation

shooting method: optimize over actions only

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$

the first action has a huge impact,
while the last action doesn't have a
big impact
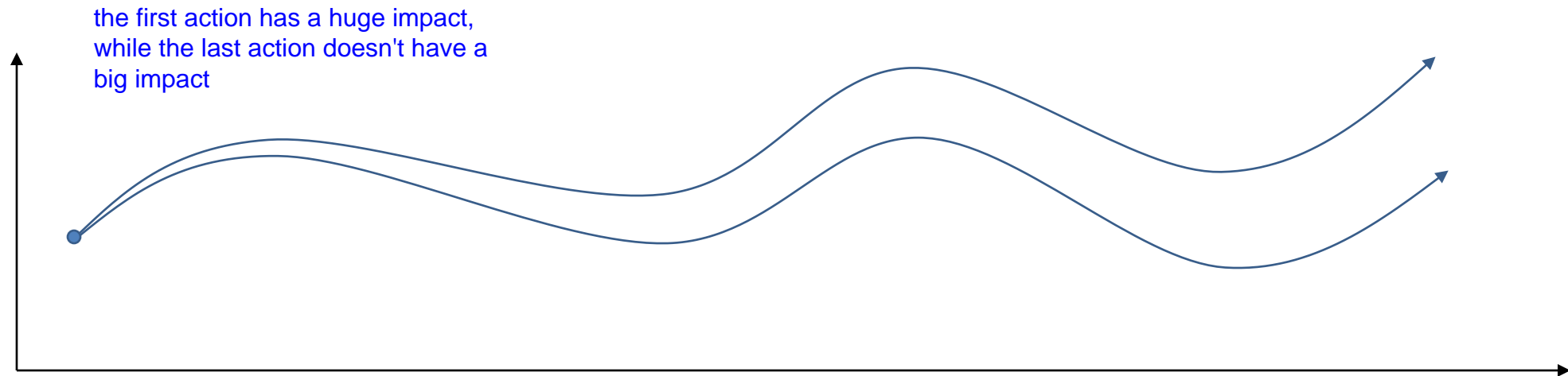
Shooting methods vs collocation

collocation method: optimize over actions and states, with constraints

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T,\mathbf{x}_1,\ldots,\mathbf{x}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

Linear case: LQR

we regulate our trajectory. we
have linear dynamics and
quadratic costs. we're allowed
to have different F, f, C, and c
for every time step!

in this case, it's deterministic

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$
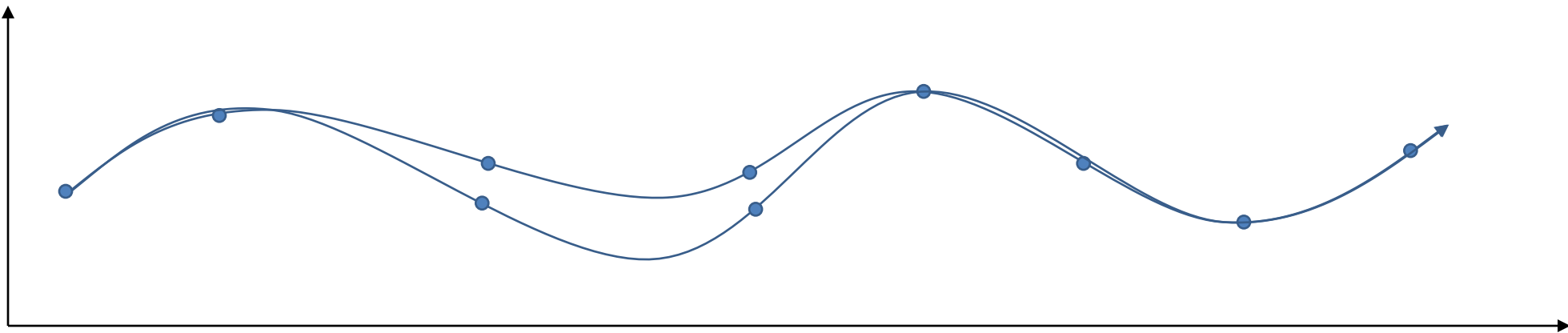
$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

$$\underline{\text{linear}}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

$$\underline{\text{quadratic}}$$

Linear case: LQR

$$\mathbf{x}_T \text{ (unknown)}$$

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots),\mathbf{u}_T)$$

only term that depends on $\mathbf{u}_T$

$$c(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{2}\begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

$$f(\mathbf{x}_t,\mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

Base case: solve for $\mathbf{u}_T$ *only*

solve for last action only

$$\mathbf{C}_T = \begin{bmatrix} \mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \\ \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T} \end{bmatrix}$$

$$Q(\mathbf{x}_T,\mathbf{u}_T) = \text{const} + \frac{1}{2}\begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{c}_T$$

$$\mathbf{c}_T = \begin{bmatrix} \mathbf{c}_{\mathbf{x}_T} \\ \mathbf{c}_{\mathbf{u}_T} \end{bmatrix}$$

$$\nabla_{\mathbf{u}_T} Q(\mathbf{x}_T,\mathbf{u}_T) = \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{u}_T + \mathbf{c}_{\mathbf{u}_T}^T = 0$$

$$\mathbf{K}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}$$

$$\mathbf{u}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}(\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{c}_{\mathbf{u}_T}) \qquad \mathbf{u}_T = \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T \qquad \mathbf{k}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\mathbf{c}_{\mathbf{u}_T}$$

this is an optimal choice for uT

Linear case: LQR

$$\mathbf{u}_T = \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \qquad\qquad \mathbf{K}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1} \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} \qquad\qquad \mathbf{k}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1} \mathbf{c}_{\mathbf{u}_T}$$

$$Q(\mathbf{x}_T, \mathbf{u}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{c}_T$$

Since $\mathbf{u}_T$ is fully determined by $\mathbf{x}_T$, we can eliminate it via substitution!

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{c}_T$$

$$V(\mathbf{x}_T) = \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \mathbf{K}_T \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T} \mathbf{K}_T \mathbf{x}_T +$$

$$\mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T} \mathbf{k}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \mathbf{k}_T + \mathbf{x}_T^T \mathbf{c}_{\mathbf{x}_T} + \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{c}_{\mathbf{u}_T} + \text{const}$$

expanding out the matrix algebra

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \mathbf{x}_T^T \mathbf{V}_T \mathbf{x}_T + \mathbf{x}_T^T \mathbf{v}_T$$

$$\mathbf{V}_T = \mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \mathbf{K}_T + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T} \mathbf{K}_T$$

$$\mathbf{v}_T = \mathbf{c}_{\mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \mathbf{k}_T + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T} + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T} \mathbf{k}_T$$

Linear case: LQR

Solve for $\mathbf{u}_{T-1}$ in terms of $\mathbf{x}_{T-1}$ $\qquad$ $\mathbf{u}_{T-1}$ affects $\mathbf{x}_T$!

$$f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \mathbf{x}_T = \mathbf{F}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \mathbf{f}_{T-1}$$

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{C}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{c}_{T-1} + V(f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}))$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \mathbf{x}_T^T \mathbf{V}_T \mathbf{x}_T + \mathbf{x}_T^T \mathbf{v}_T$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underline{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underline{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1}} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underline{\mathbf{F}_{T-1}^T \mathbf{v}_?}$$

$\qquad\qquad\qquad\qquad\qquad$ quadratic $\qquad\qquad\qquad\qquad\qquad\qquad$ linear $\qquad\qquad\qquad$ linear

Linear case: LQR

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{C}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{c}_{T-1} + V(f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}))$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underline{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underline{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1}} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underline{\mathbf{F}_{T-1}^T \mathbf{v}_7}$$

<div align="center">quadratic   linear   linear</div>

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{Q}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{q}_{T-1}$$

$$\mathbf{Q}_{T-1} = \mathbf{C}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}$$

$$\mathbf{q}_{T-1} = \mathbf{c}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{v}_T$$

$$\nabla_{\mathbf{u}_{T-1}} Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{x}_{T-1}} \mathbf{x}_{T-1} + \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}} \mathbf{u}_{T-1} + \mathbf{q}_{\mathbf{u}_{T-1}}^T = 0$$

$$\mathbf{u}_{T-1} = \mathbf{K}_{T-1} \mathbf{x}_{T-1} + \mathbf{k}_{T-1} \qquad\qquad \mathbf{K}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}}^{-1} \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{x}_{T-1}}$$

$$\mathbf{k}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}}^{-1} \mathbf{q}_{\mathbf{u}_{T-1}}$$

Linear case: LQR

Backward recursion

for $t = T$ to 1:

express Q function in quadratic form

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$$\mathbf{u}_t \leftarrow \arg\min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t \quad \text{choose action}$$
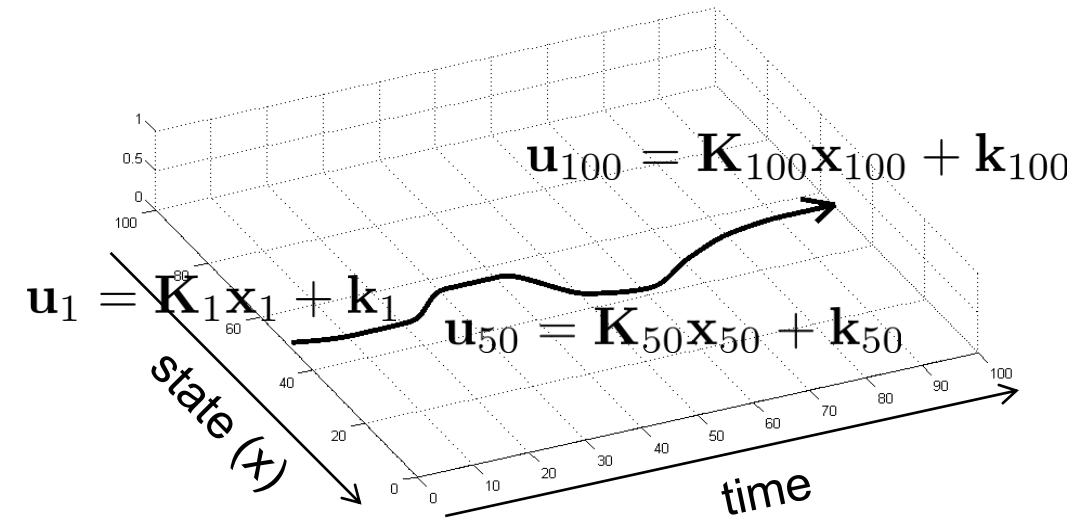
$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t, \mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{k}_t$$

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t \quad \text{express value function which is quadratic}$$

$$\mathbf{u}_{100} = \mathbf{K}_{100} \mathbf{x}_{100} + \mathbf{k}_{100}$$

$$\mathbf{u}_1 = \mathbf{K}_1 \mathbf{x}_1 + \mathbf{k}_1$$

$$\mathbf{u}_{50} = \mathbf{K}_{50} \mathbf{x}_{50} + \mathbf{k}_{50}$$

state (x)

time

we know $\mathbf{x}_1$!

Forward recursion

for $t = 1$ to T:

$$\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$$

since we know X1, we can then go forward and numerically calculation values for u

when we've completed the recursion, we've produced an expression for K and k for every single timestep

Linear case: LQR

Backward recursion

for $t = T$ to 1:   total cost from now until end if we take $\mathbf{u}_t$ from state $\mathbf{x}_t$

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$$\mathbf{u}_t \leftarrow \arg\min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

total cost from now until end from state $\mathbf{x}_t$

$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t}$$

$$V(\mathbf{x}_t) = \min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t)$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t, \mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{k}_t$$

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$$

LQR for Stochastic and Nonlinear Systems

now lets extend LQR to stochastic
dynamics and non-linear systems

Stochastic dynamics

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

$$\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t)$$

$$p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N} \left(\mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t, \Sigma_t \right)$$

Solution: choose actions according to $\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$

$\mathbf{x}_t \sim p(\mathbf{x}_t)$, no longer deterministic, but $p(\mathbf{x}_t)$ is Gaussian

no change to algorithm! can ignore Σ_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)

The stochastic **closed**-loop case

$\pi(\mathbf{a}_t | \mathbf{s}_t)$

$\pi(\mathbf{a}_t | \mathbf{s}_t)$

$\mathbf{s}_1$
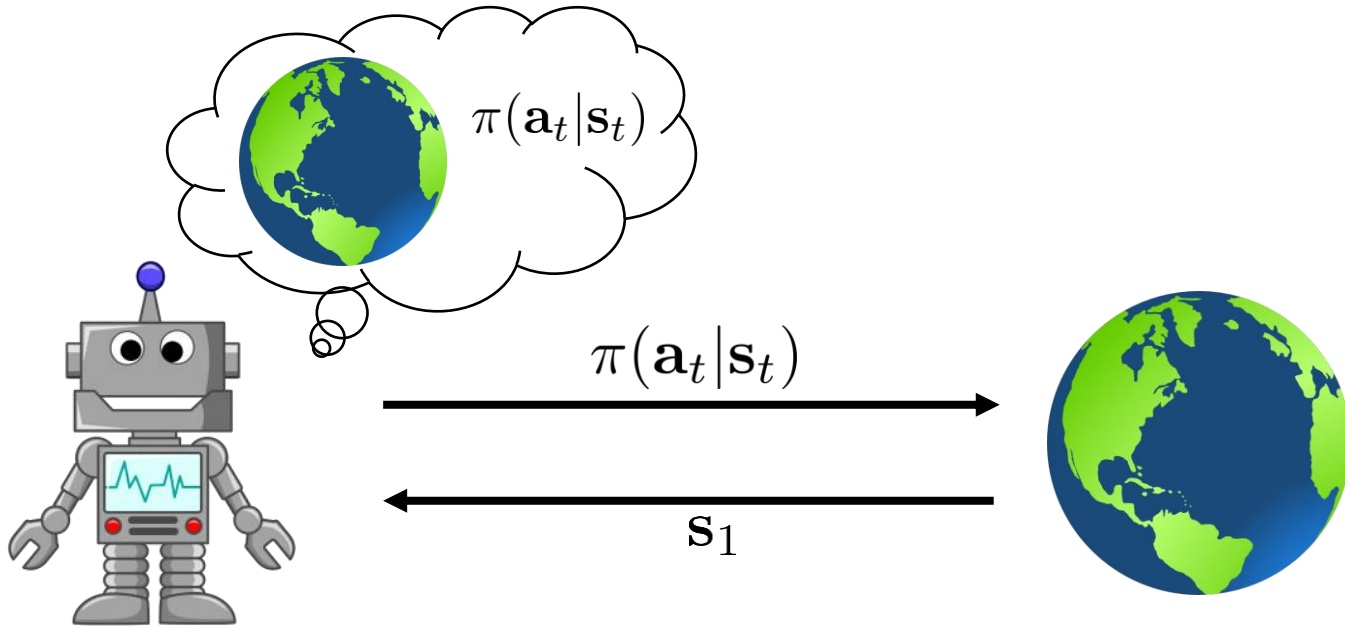
form of π?

$$p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

time-varying linear

$$\mathbf{K}_t \mathbf{s}_t + \mathbf{k}_t$$

$$\pi = \arg \max_{\pi} E_{\tau \sim p(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Nonlinear case: DDP/iterative LQR

Linear-quadratic assumptions:

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t \qquad\qquad c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

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

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla^2_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

Nonlinear case: DDP/iterative LQR

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla^2_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

$$\bar{f}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \underbrace{\mathbf{F}_t}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} \qquad \bar{c}(\delta\mathbf{x}_t, \delta\mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^T \underbrace{\mathbf{C}_t}_{\nabla^2_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^T \underbrace{\mathbf{c}_t}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)}$$

$$\delta\mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$$

$$\delta\mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$$

Now we can run LQR with dynamics $\bar{f}$, cost $\bar{c}$, state $\delta\mathbf{x}_t$, and action $\delta\mathbf{u}_t$

Nonlinear case: DDP/iterative LQR

Iterative LQR (simplified pseudocode)

until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{c}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{C}_t = \nabla^2_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

Run LQR backward pass on state $\delta\mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ and action $\delta\mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$

Run forward pass with real nonlinear dynamics and $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t$

Update $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ based on states and actions in forward pass

Nonlinear case: DDP/iterative LQR

Why does this work?

Compare to Newton's method for computing $\min_{\mathbf{x}} g(\mathbf{x})$:

until convergence:

$$\mathbf{g} = \nabla_{\mathbf{x}} g(\hat{\mathbf{x}})$$

$$\mathbf{H} = \nabla_{\mathbf{x}}^2 g(\hat{\mathbf{x}})$$

$$\hat{\mathbf{x}} \leftarrow \arg\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{H}(\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{g}^T(\mathbf{x} - \hat{\mathbf{x}})$$

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

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\dots)\dots), \mathbf{u}_T)$$

Nonlinear case: DDP/iterative LQR

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

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$

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

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} + \frac{1}{2}\left(\nabla^2_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \cdot \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} \right) \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}$$
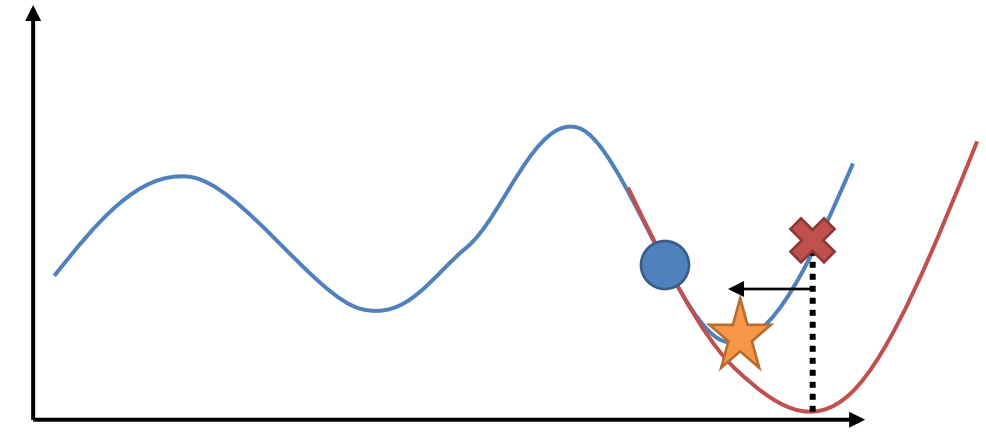
differential dynamic programming (DDP)

Nonlinear case: DDP/iterative LQR

$$\hat{\mathbf{x}} \leftarrow \arg\min_{\mathbf{x}} \frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{H}(\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{g}^T(\mathbf{x} - \hat{\mathbf{x}})$$

why is this a bad idea?

until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{c}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{C}_t = \nabla^2_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

Run LQR backward pass on state $\delta\mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ and action $\delta\mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$

Run forward pass with $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}\mathbf{k}_t + \hat{\mathbf{u}}\hat{\mathbf{u}}_t$

Update $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ based on states and actions in forward pass

search over α
until improvement achieved

Case Study and Additional Readings

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

Model predictive control: every timestep, you observe your current state xt, then you use your favorite planning or control method to figure out a plan (i.e. a sequence of actions all the way to time T). Then you do the first step of that plan. Then you discard the remaining part of your plan, and create a new plan based on your new state.

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

Additional reading

1. Mayne, Jacobson. (1970). Differential dynamic programming.
   - Original differential dynamic programming algorithm.

2. Tassa, Erez, Todorov. (2012). Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization.
   - Practical guide for implementing non-linear iterative LQR.

3. Levine, Abbeel. (2014). Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics.
   - Probabilistic formulation and trust region alternative to deterministic line search.

What's wrong with known dynamics?
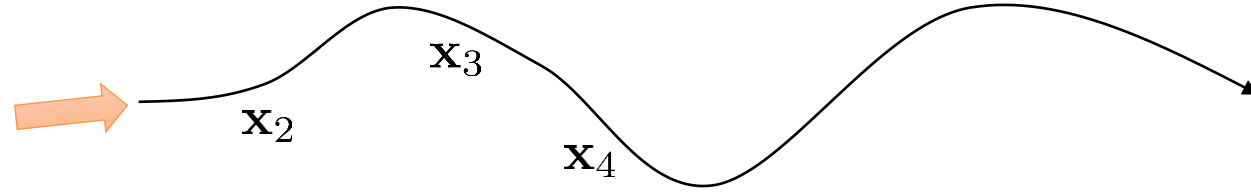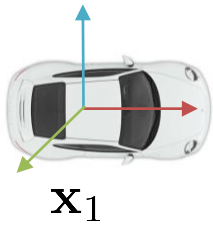
Next time: learning the dynamics model