Optimal Control and Planning

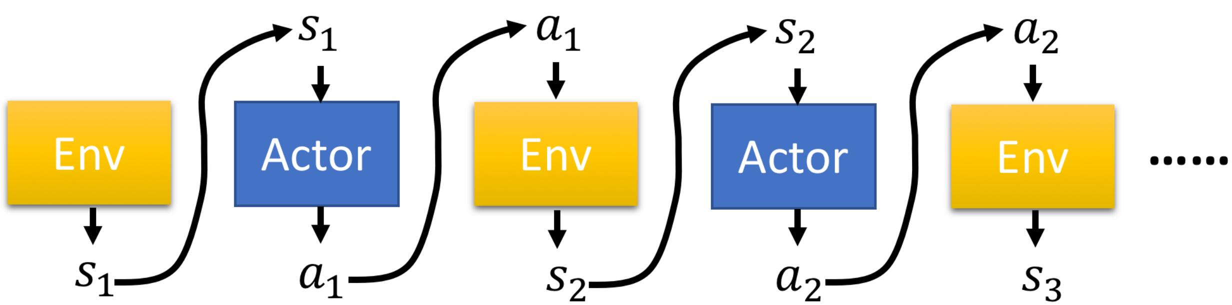
Spring 2024

Introduction

- **Open-Loop and Closed Loop Planning**
- **Cross-Entropy Method (CEM)**
- **Monte Carlo Tree Search (MCTS)**
- **Linear Quadratic Regulator (LQR)**

Reinforcement Learning Objective

Actor, Environment, Reward



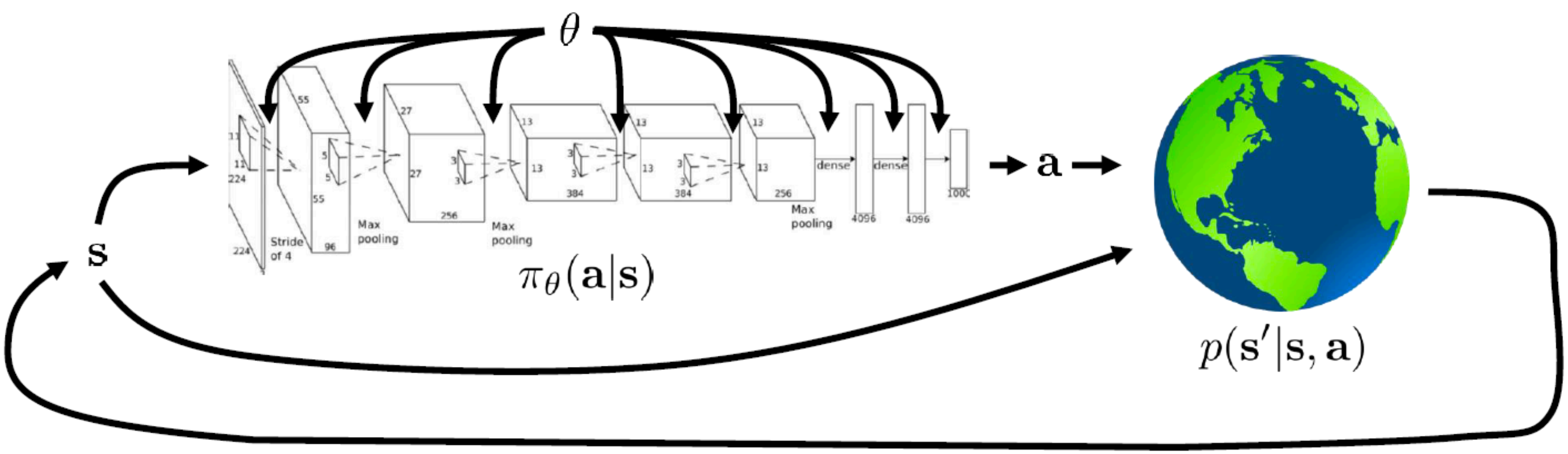
Trajectory $\tau = \{s_1, a_1, s_2, a_2, \cdots, s_T, a_T\}$

$$p_\theta(\tau)$$

$$= p(s_1)p_\theta(a_1|s_1)p(s_2|s_1,a_1)p_\theta(a_2|s_2)p(s_3|s_2,a_2)\cdots$$

$$= p(s_1)\prod_{t=1}^{T} p_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)$$

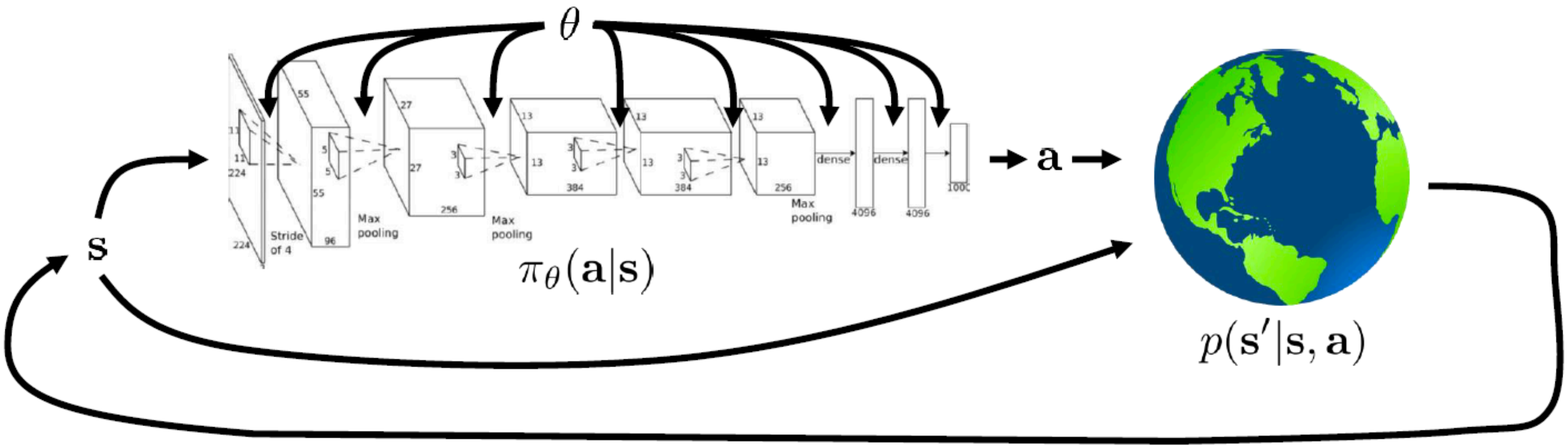
Reinforcement Learning Objective



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

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

Reinforcement Learning Objective



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

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

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

If transition dynamics is known

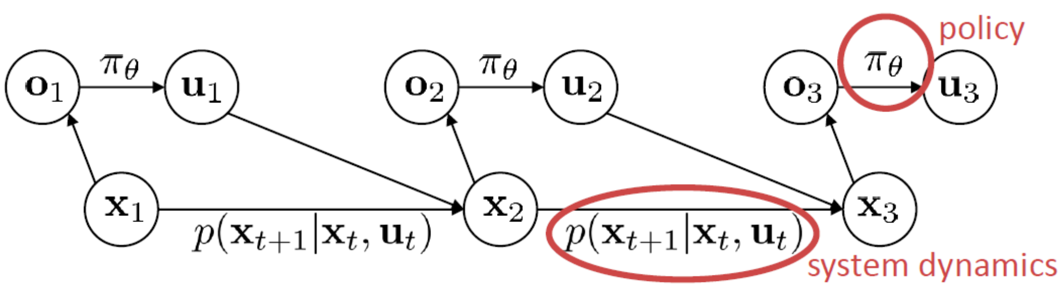
- Often we do know the dynamics
  1. Games (e.g., Atari games, chess, Go)
  2. Easily modeled systems (e.g., navigating a car)
  3. Simulated environments (e.g., simulated robots, video games)
- Often we can learn the dynamics
  1. System identification – fit unknown parameters of a known model
  2. Learning – fit a general-purpose model to observed transition data

Does knowing the dynamics make things easier?

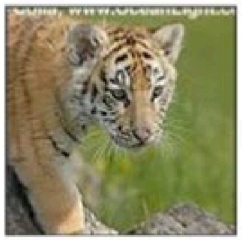
Often, yes!

Model-based Reinforcement Learning

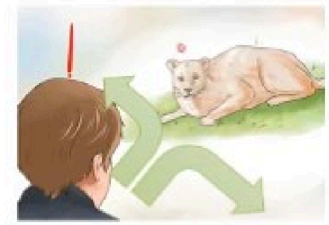
1. Model-based reinforcement learning: learn the transition dynamics, then figure out how to choose actions
2. Today: how can we make decisions if we *know* the dynamics?
   a. How can we choose actions under perfect knowledge of the system dynamics?
   b. Optimal control, trajectory optimization, planning
3. Next week: how can we learn *unknown* dynamics?
4. How can we then also learn policies? (*e.g. by imitating optimal control*)



Objective



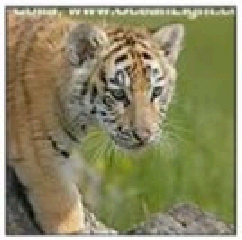
$\mathbf{s}_t$



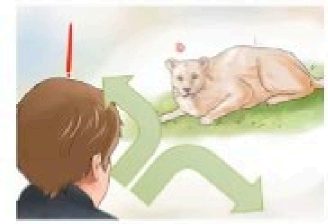
$\mathbf{a}_t$

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

Objective



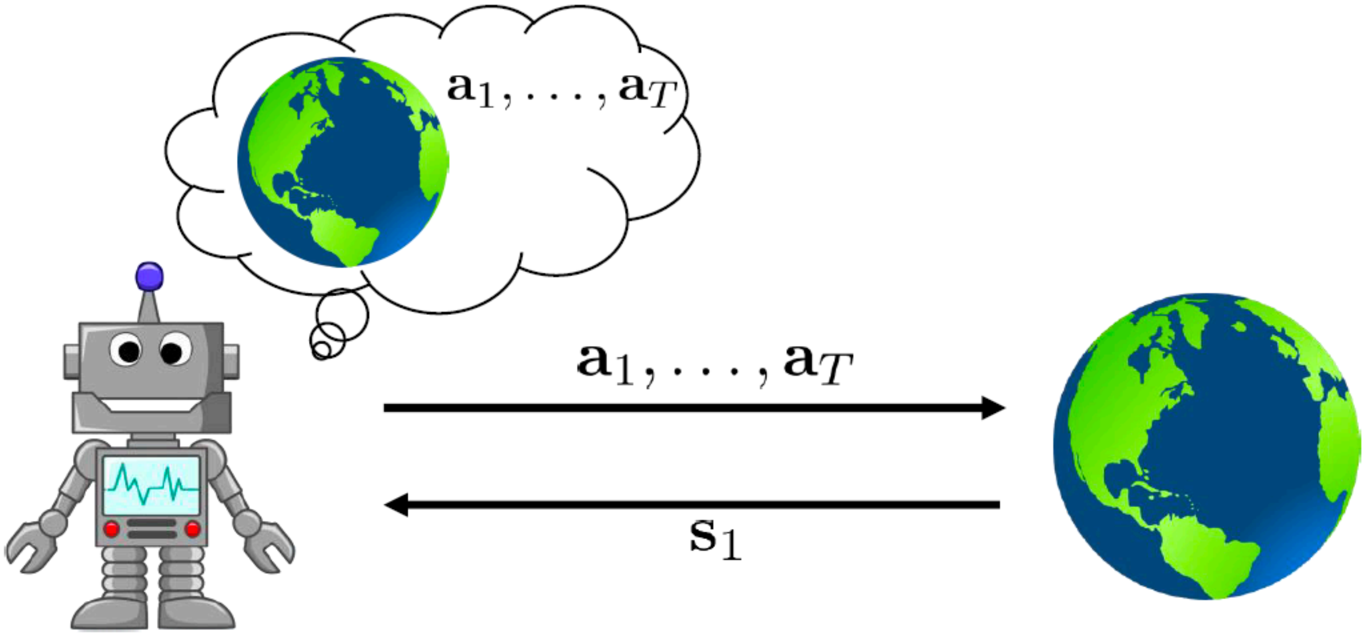
$\mathbf{s}_t$



$\mathbf{a}_t$

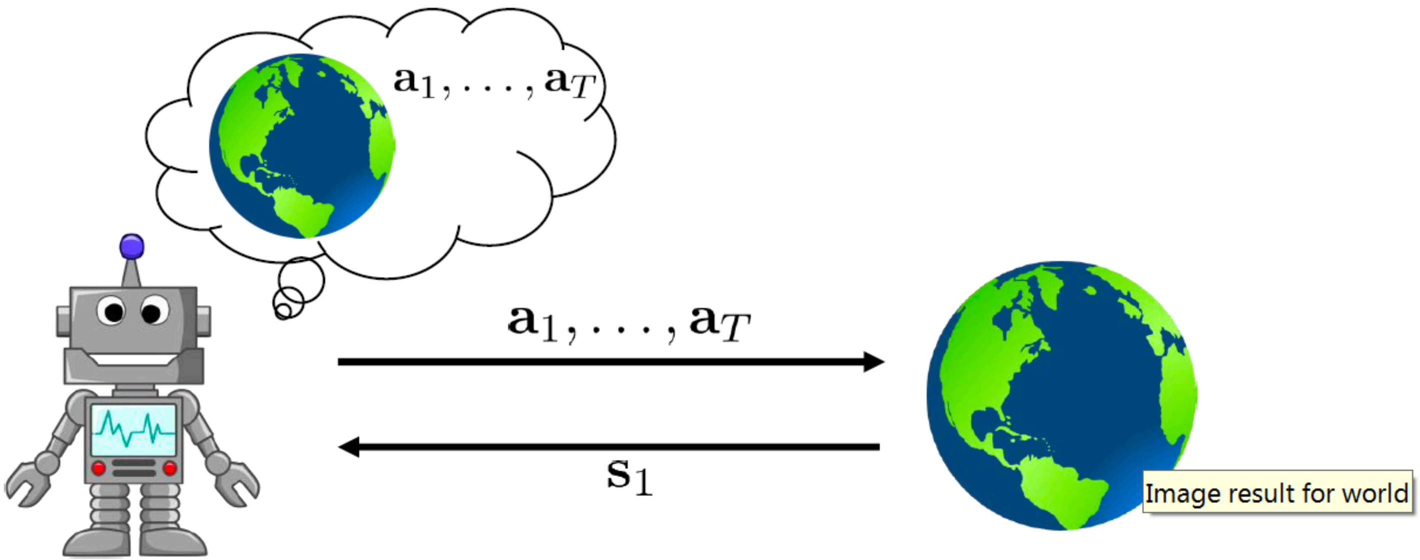
$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \sum_{t=1}^{T} c(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{s}_t = f(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$$

Deterministic Case



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

Stochastic Open-Loop Case



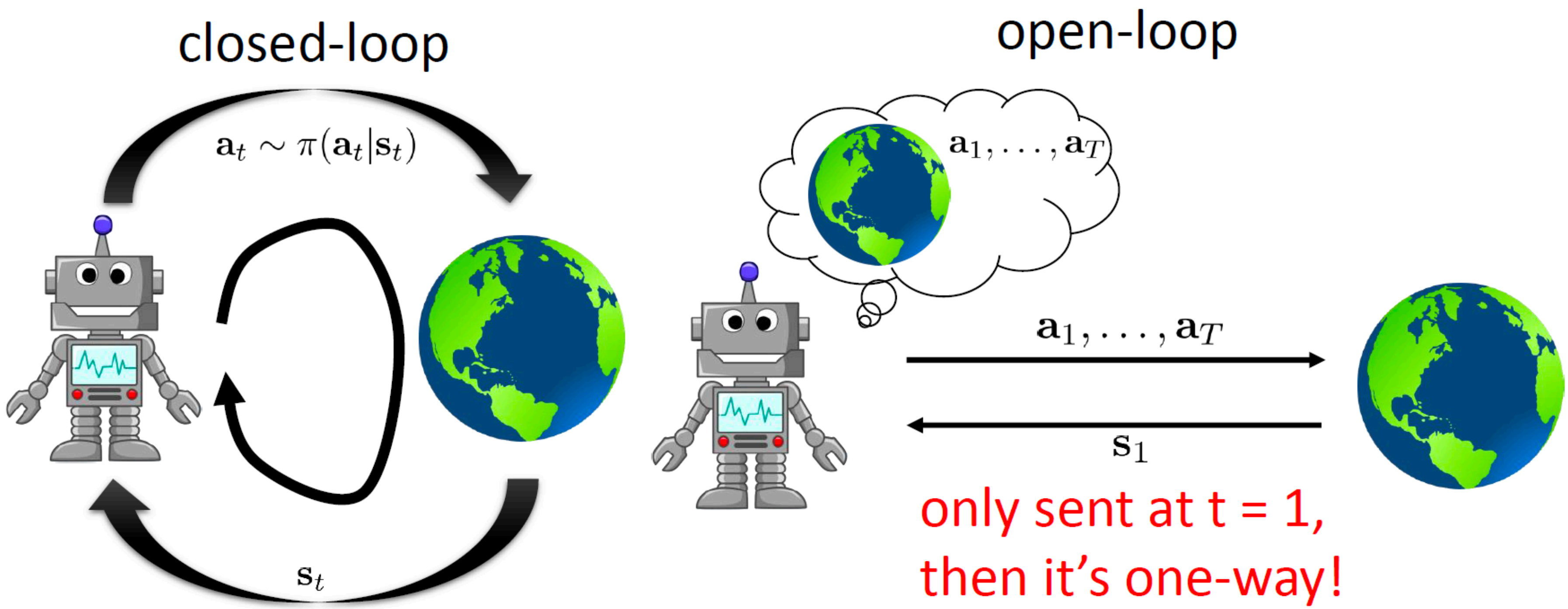
$$p_\theta(\mathbf{s}_1, \ldots, \mathbf{s}_T | \mathbf{a}_1, \ldots, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} E\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{a}_1, \ldots, \mathbf{a}_T\right]$$

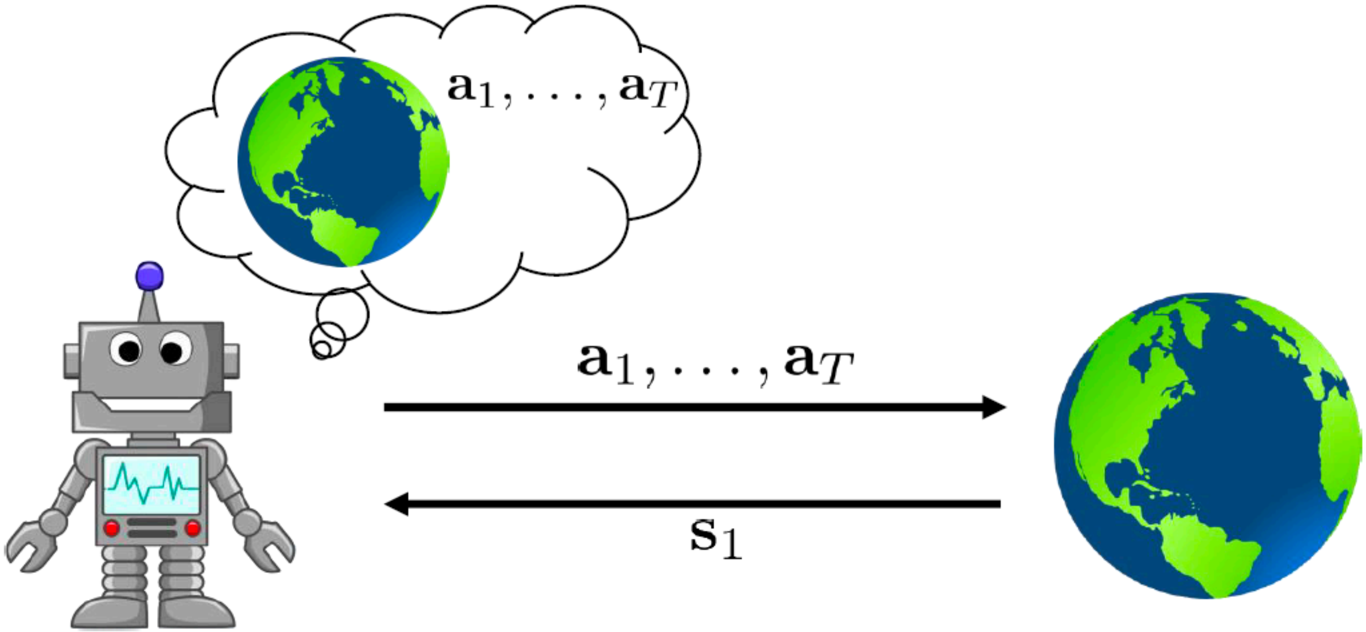
why is this suboptimal?

Open-Loop vs. Closed Loop

what is this “loop”?



Open-Loop Planning



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

Stochastic Optimization

abstract away optimal control/planning:

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \underbrace{J(\mathbf{a}_1, \ldots, \mathbf{a}_T)}_{\text{don't care what this is}}$$

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

simplest method: guess & check "random shooting method"

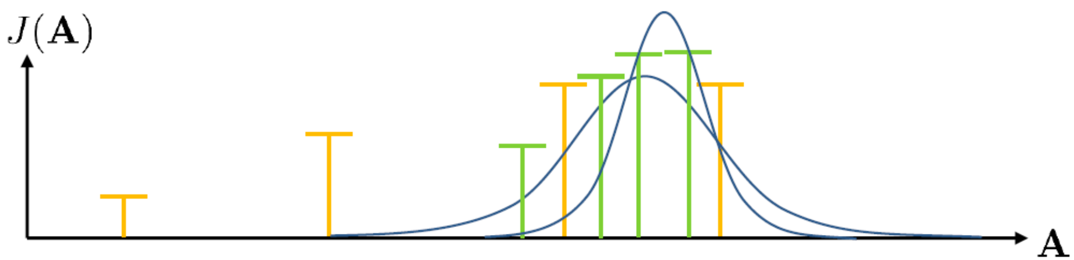
1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

Cross-Entropy Method (CEM)

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

can we do better?

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



typically use Gaussian distribution

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

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

What’s the upside?

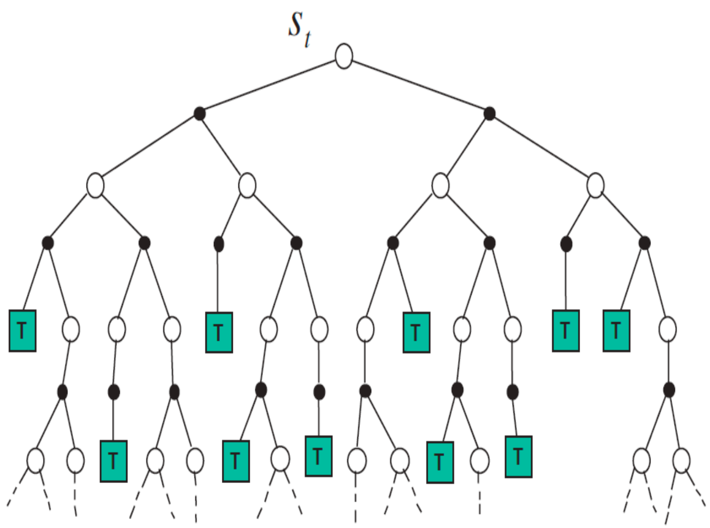
1. Very fast if parallelized
2. Extremely simple

What’s the problem?

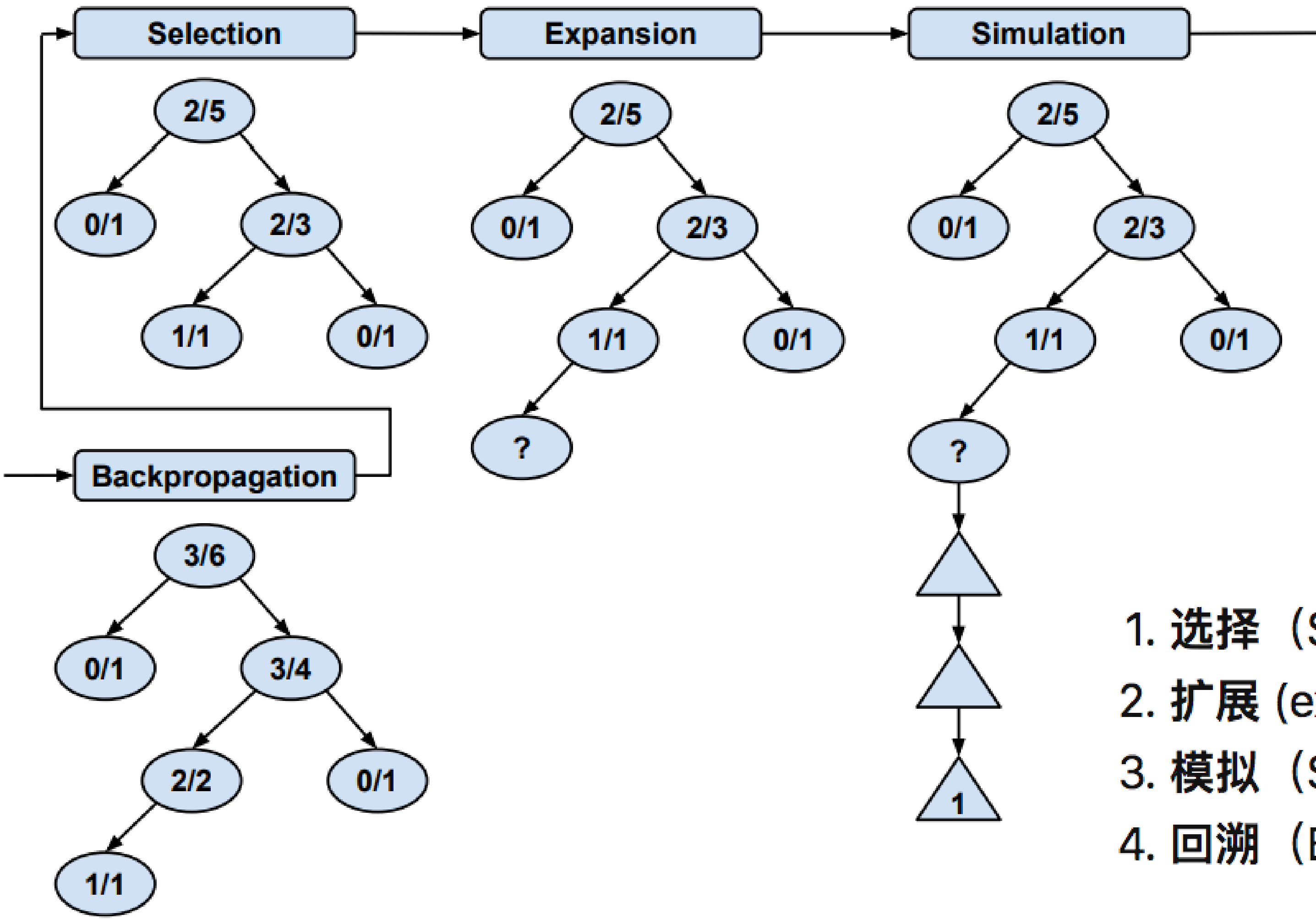
1. Very harsh dimensionality limit
2. Only open-loop planning

Discrete Case: Monte Carlo Tree Search

- A search algorithm that uses random simulations to grow a tree for selecting best actions
- MCTS has experienced a lot of success in domains with vast search spaces such as Go
- The tree is built with the current state S_t at the root.
- Using a model of the MDP to look ahead
- No need to solve whole MDP, just sub-MDP starting from now



Monte Carlo Tree Search



1. 选择（Selection）
2. 扩展 (expansion)
3. 模拟（Simulation）
4. 回溯（Backpropagation）

Monte Carlo Tree Search

Selection - In the selection process, the MCTS algorithm traverses the current tree using a tree policy. A tree policy uses an evaluation function that prioritize nodes with the greatest estimated value. Once a node is reached in the traversal that has children (or moves) left to be added, then MCTS transitions into the expansion step. In figure 2, starting from the root node, the tree policy must make a decision between the 0/1 node and the 2/3 node. Since 2/3 is greater than 0/1, the tree policy will choose the 2/3 node in its traversal. Once at the 2/3 node, the tree policy will then choose the 1/1 node because it is greater than 0/1. This is the first node with children yet to be added, so now MCTS will transition into the expansion step.

Max Magnuson, Monte Carlo Tree Search and Its Applications, 2015

Monte Carlo Tree Search

Expansion - In the expansion step, a new node is added to the tree as a child of the node reached in the selection step. The algorithm is currently at the 1/1 node, so there is a child node added onto that node indicated by the node with the ?. There is only one node added to the tree in each iteration, and it is at this step.

Simulation - In this step, a simulation (also referred to as a playout or rollout) is performed by choosing moves until either an end state or a predefined threshold is reached. In the case of Go or TicTacToe, an end state is reached when the game ends. Then based on the result of the simulation, the value of the newly added node is established. For example, a simulation for a node in Go reaches the end of a game (the end state), and then determines a value based on whether the player won or lost. In figure 2 the simulation ended in a 1. Therefore, the value of the new node is 1/1. One simulation resulted in a win, and one simulation has been performed.

Monte Carlo Tree Search

Backpropagation - Now that the value of the newly added node has been determined, the rest of the tree must be updated. Starting at the new node, the algorithm traverses back to the root node. During the traversal the number of simulations stored in each node is incremented, and if the new node's simulation resulted in a win then the number of wins is also incremented. In figure 2 only the nodes with values 0/1 are not updated since they are not an ancestor of the newly added node. This step ensures that the values of each node accurately reflect simulations performed in the subtrees that they represent.

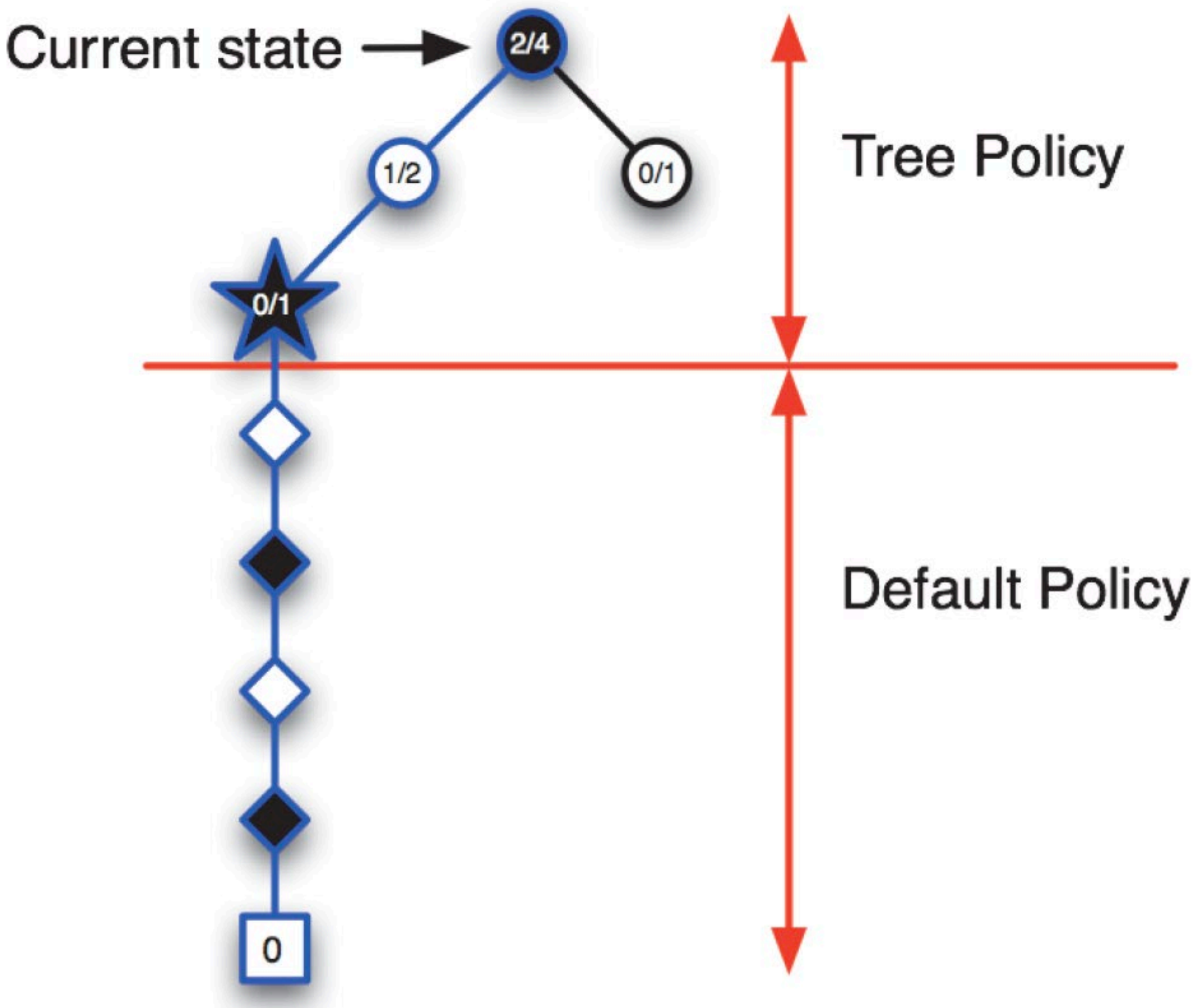
Monte Carlo Tree Search

UCT balances exploration and exploitation by giving relatively unexplored nodes an exploration bonus.

$$UCT(node) = \frac{W(node)}{N(node)} + c\sqrt{\frac{ln(N(parentNode))}{N(node)}}$$

When traversing the tree, the child node that returns the greatest value from this equation will be selected [1]. N represents the total number of simulations performed at that node and its descendants. W represents how many of those simulations resulted in a winning state. C represents an exploration constant that is found experimentally. The first part of the UCT takes into consideration the estimated value of the node from the ratio of simulations won to total simulations. This is the exploitation part of the equation. The second part of the UCT is the exploration bonus. This compares the total number of simulations performed at the parent node and its descendants to the total number of simulations performed at the examined node and its descendants. This means that the lower the number of simulations that have been performed at this node, the greater this part of the equation will be.

Monte Carlo Tree Search



Trajectory Optimization with Derivatives

Can we use derivatives?

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

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

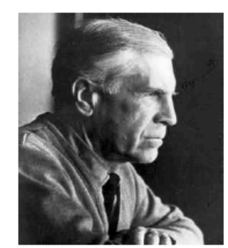
usual story: differentiate via backpropagation and optimize!

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

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

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

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



Linear Quadratic Regulator (LQR)

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

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

Linear Quadratic Regulator (LQR)

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \cdots + \underbrace{c(\overbrace{f(f(\ldots)\ldots)}^{\mathbf{x}_T \text{ (unknown)}},\mathbf{u}_T)}_{\text{only term that depends on } \mathbf{u}_T}$$

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

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

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

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

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

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

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

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

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

Linear Quadratic Regulator (LQR)

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

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

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

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

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

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

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

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

Linear Quadratic Regulator (LQR)

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

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

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

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

Linear Quadratic Regulator (LQR)

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

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

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

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

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

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

$$\mathbf{u}_{T-1} = \mathbf{K}_{T-1} \mathbf{x}_{T-1} + \mathbf{k}_{T-1}$$

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

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

Linear Quadratic Regulator (LQR)

Backward recursion

for $t = T$ to 1:

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

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

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

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

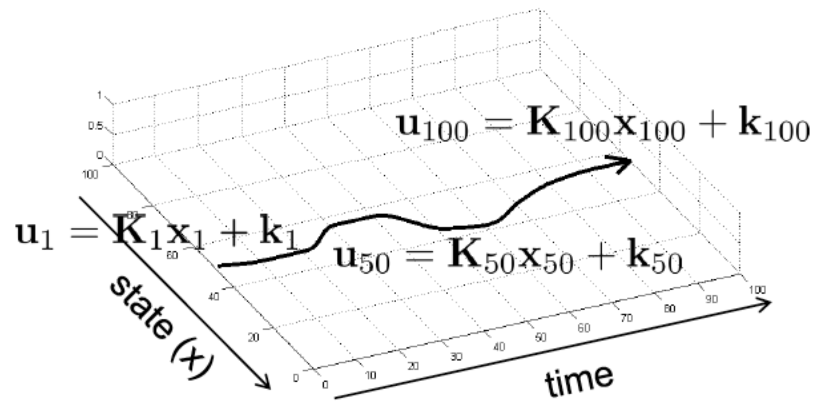
$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t}$$

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

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

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

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



we know $\mathbf{x}_1$!

Forward recursion

for $t = 1$ to T:

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

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