Aalto University
School of Electrical Engineering

ELEC-E8125 Reinforcement Learning
Model-based RL

Joni Pajarinen

24.10.2023

Learning goals

- Understand how optimal control relates to model-based reinforcement learning
- Understand basic sampling based policy optimization

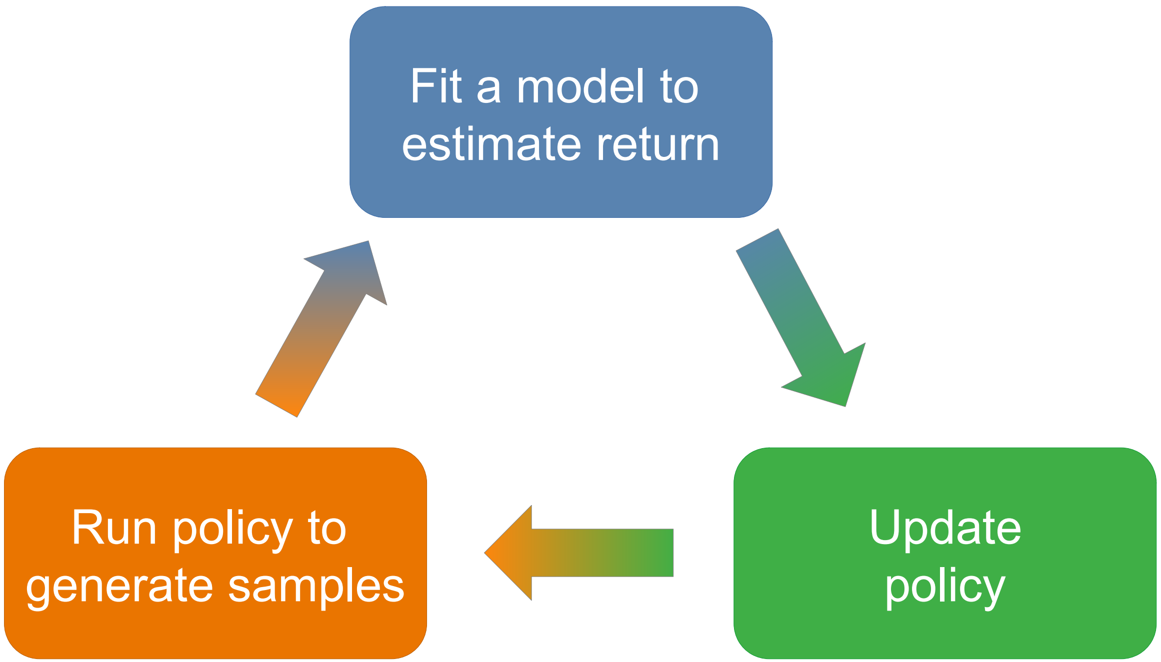
Motivation from two perspectives

- Reinforcement learning has limited sample efficiency
  - Locally optimal control can control complex systems
    - For example, whole body control of a humanoid robot https://www.youtube.com/watch?v=vI-8xgJ6ct0
  - Caveat: optimal control requires knowing the system dynamics

- Learned policies are task, that is, reward-function-specific, learned knowledge cannot be reused

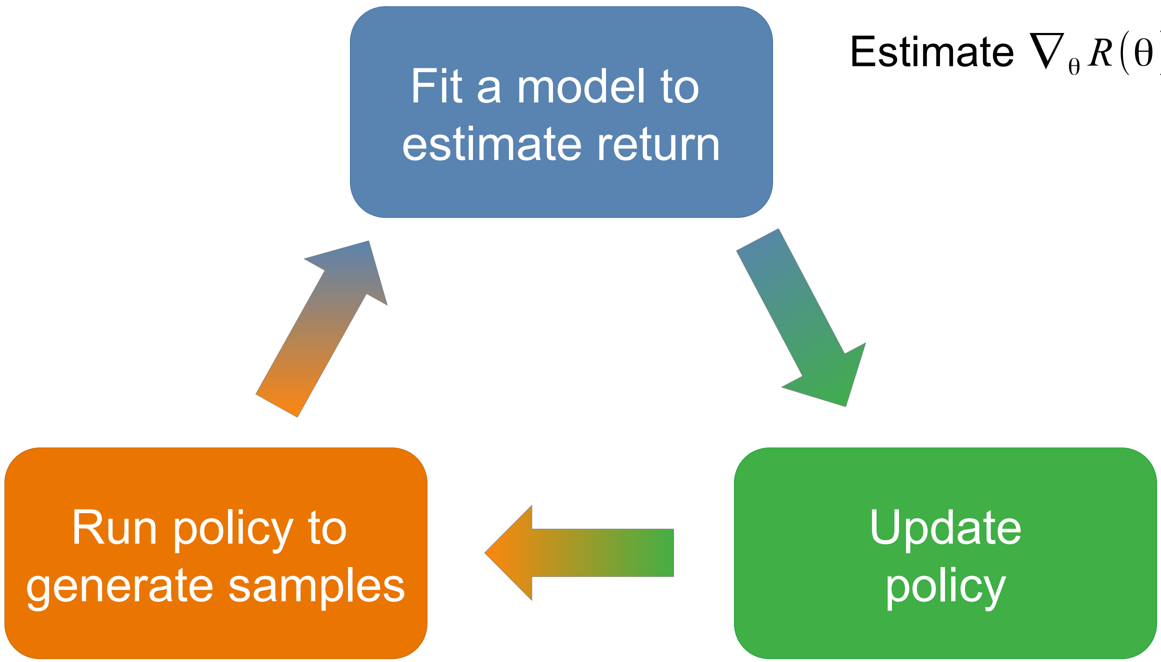
Can we somehow combine RL and optimal control?

Anatomy of reinforcement learning



Adopted from Sergey Levin.

Anatomy of reinforcement learning: *Policy gradient*

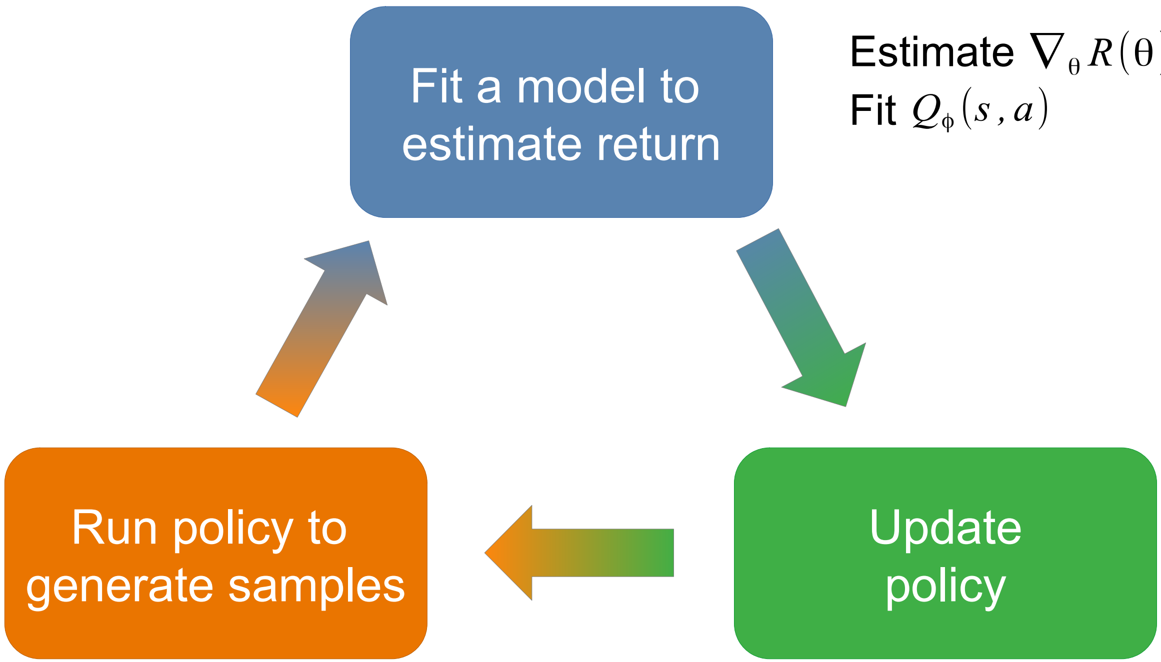


Estimate $\nabla_\theta R(\theta)$ (policy gradient)

$\theta \leftarrow \theta + \alpha \nabla_\theta R(\theta)$ (policy gradient)

Adopted from Sergey Levin.

Anatomy of reinforcement learning: *Value-function based*



Estimate $\nabla_\theta R(\theta)$ (policy gradient)

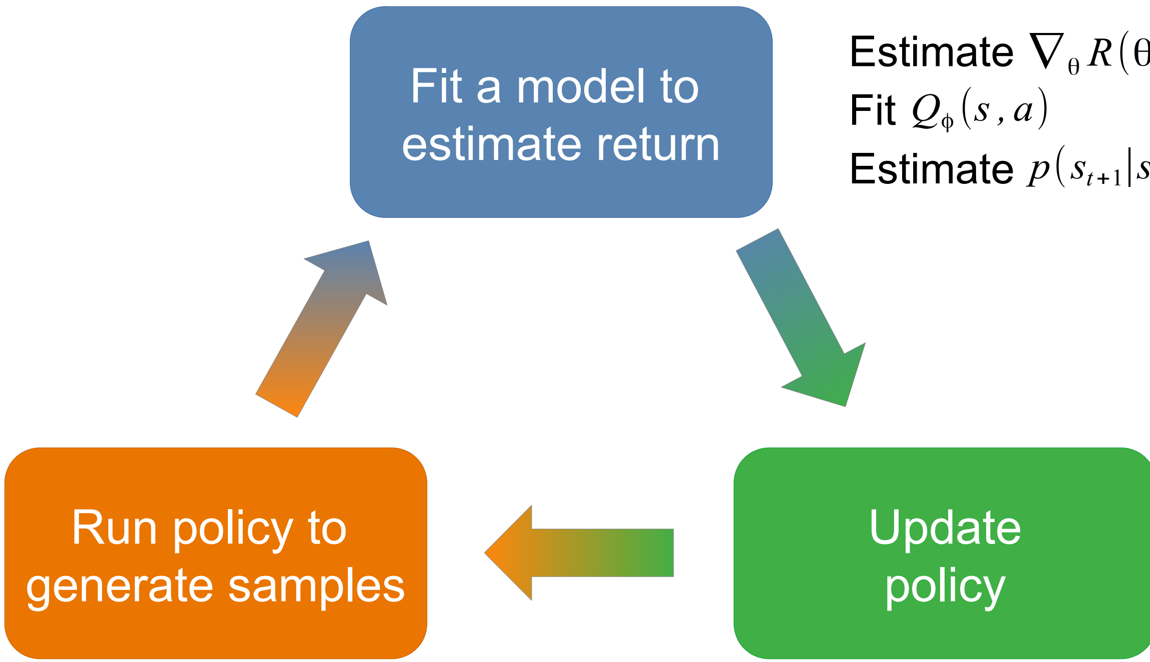
Fit $Q_\phi(s,a)$ (Q-learning, actor-critic)

$\theta \leftarrow \theta + \alpha \nabla_\theta R(\theta)$ (policy gradient)

$arg\ max_a Q_\phi(s,a)$ (Q-learning)

Adopted from Sergey Levin.

Anatomy of reinforcement learning: *Model-based*

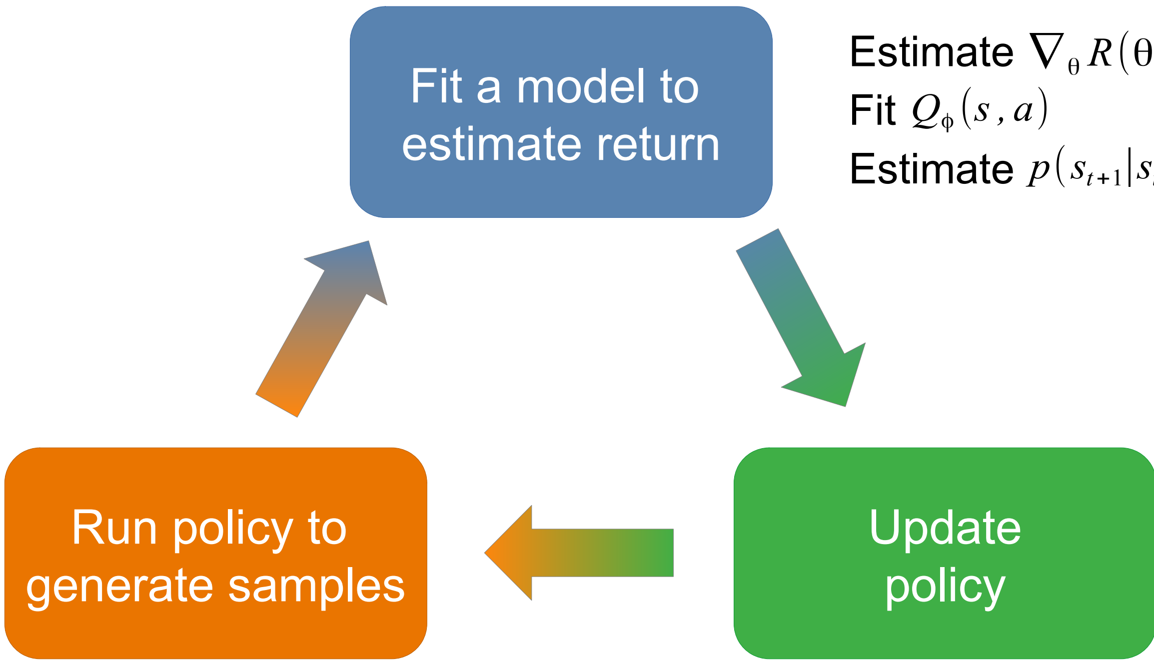


Estimate $\nabla_\theta R(\theta)$ (policy gradient)
Fit $Q_\phi(s, a)$ (Q-learning, actor-critic)
Estimate $p(s_{t+1}|s_t, a_t)$ (model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta R(\theta)$ (policy gradient)
$arg\ max_a Q_\phi(s, a)$ (Q-learning)
Optimize $\pi_\theta(a|s)$ (model-based)

Adopted from Sergey Levin.

Anatomy of reinforcement learning
Model-based



Estimate $\nabla_\theta R(\theta)$ (policy gradient)
Fit $Q_\phi(s,a)$ (Q-learning, actor-critic)
Estimate $p(s_{t+1}|s_t,a_t)$ (model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta R(\theta)$ (policy gradient)
$arg\ max_a Q_\phi(s,a)$ (Q-learning)
Optimize $\pi_\theta(a|s)$ (model-based)

Today this for known dynamics.

Adopted from Sergey Levin.

Solving optimal control problems

Optimal control
optimization objective

$$min \sum_t c(\boldsymbol{s}_t, \boldsymbol{a}_t)$$

cost
function

Reinforcement learning
optimization objective

$$max \sum_t r(\boldsymbol{s}_t, \boldsymbol{a}_t)$$

reward
function

$$c(\boldsymbol{s}_t, \boldsymbol{a}_t) = -r(\boldsymbol{s}_t, \boldsymbol{a}_t)$$

Solving (deterministic, finite-horizon) optimal control problems

$$min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_T}\sum_t c(\boldsymbol{s}_t,\boldsymbol{a}_t)\quad s.t.\quad \boldsymbol{s}_{t+1}=f(\boldsymbol{s}_t,\boldsymbol{a}_t)$$

cost function

system dynamics

Can also be written as:

$$min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_T} c(\boldsymbol{s}_1,\boldsymbol{a}_1)+c(f(\boldsymbol{s}_1,\boldsymbol{a}_1),\boldsymbol{a}_2)+\ldots+c(f(f(\ldots)),\boldsymbol{a}_T)$$

How to solve these?

Shooting vs collocation

Shooting methods: Optimize actions

$$min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_T} c(\boldsymbol{s}_1,\boldsymbol{a}_1)+c(f(\boldsymbol{s}_1,\boldsymbol{a}_1),\boldsymbol{a}_2)+\ldots+c(f(f(\ldots)),\boldsymbol{a}_T)$$

Collocation methods: Optimize actions and states (constrained optimization)

$$\min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_T,\boldsymbol{s}_1,\ldots,\boldsymbol{s}_T} \sum_t c(\boldsymbol{s}_t,\boldsymbol{a}_t) \quad s.t. \quad \boldsymbol{s}_{t+1}=f(\boldsymbol{s}_t,\boldsymbol{a}_t)$$

LQR (linear-quadratic regulator)
Problem definition (finite horizon)

$$min_{a_1,\ldots,a_T} c(s_1, a_1) + c(f(s_1, a_1), a_2) + \ldots + c(f(f(\ldots)), a_T)$$

$$f(\mathbf{s}_t, \mathbf{a}_t) = \begin{pmatrix} \mathbf{A}_t & \mathbf{B}_t \end{pmatrix} \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix} + \mathbf{f}_t = \mathbf{F}_t \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix} + \mathbf{f}_t$$

$$c_t(\mathbf{s}_t, \mathbf{a}_t) = \frac{1}{2} \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}^T \mathbf{C}_t \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix} + \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}^T \mathbf{c}_t$$

Note: costs for different time steps may vary. For example, different costs for final time step.

Note: We will follow notation that clumps together state and action, opposite to traditional control literature, because most recent RL papers use that. We also include the bias term from the beginning.

$$\mathbf{C}_t = \begin{pmatrix} \mathbf{C}_{s_t,s_t} & \mathbf{C}_{s_t,a_t} \\ \mathbf{C}_{a_t,s_t} & \mathbf{C}_{a_t,a_t} \end{pmatrix} \qquad \mathbf{c}_t = \begin{pmatrix} \mathbf{c}_{s_t} \\ \mathbf{c}_{a_t} \end{pmatrix}$$

Example system: 1-D particle motion

$$f(s_t, a_t) = F_t \begin{pmatrix} s_t \\ a_t \end{pmatrix} + f_t$$

$$c_t(s_t, a_t) = \frac{1}{2} \begin{pmatrix} s_t \\ a_t \end{pmatrix}^T C_t \begin{pmatrix} s_t \\ a_t \end{pmatrix} + \begin{pmatrix} s_t \\ a_t \end{pmatrix}^T c_t$$

How to solve?

LQR partial derivation, final step

$$\min_{a_1,\ldots,a_T} c(s_1, a_1) + c(f(s_1, a_1), a_2) + \ldots + \underbrace{c(f(f(\ldots)), a_T)}_{\text{Only cost depending on } a_T}$$

$$f(s_t, a_t) = F_t \begin{pmatrix} s_t \\ a_t \end{pmatrix} + f_t$$

$$c_t(s_t, a_t) = \frac{1}{2} \begin{pmatrix} s_t \\ a_t \end{pmatrix}^T C_t \begin{pmatrix} s_t \\ a_t \end{pmatrix} + \begin{pmatrix} s_t \\ a_t \end{pmatrix}^T c_t$$

Action-value function:

$$Q(s_T, a_T) = const + \frac{1}{2} \begin{pmatrix} s_T \\ a_T \end{pmatrix}^T C_T \begin{pmatrix} s_T \\ a_T \end{pmatrix} + \begin{pmatrix} s_T \\ a_T \end{pmatrix}^T c_T$$

$$\nabla_{a_t} Q(s_T, a_T) = C_{a_T, s_T} s_T + C_{a_T, a_T} a_t + c_{a_t} = 0$$

$$a_T = -C_{a_T, a_T}^{-1} \left(C_{a_t, s_t} s_t + c_{a_t} \right)$$

$$\boxed{\begin{aligned} a_T &= K_T s_T + k_T \\ K_T &= -C_{a_T, a_T}^{-1} C_{a_t, s_t} \\ k_T &= -C_{a_T, a_T}^{-1} c_{a_t} \end{aligned}}$$

$$C_t = \begin{pmatrix} C_{s_t, s_t} & C_{s_t, a_t} \\ C_{a_t, s_t} & C_{a_t, a_t} \end{pmatrix} \qquad c_t = \begin{pmatrix} c_{s_t} \\ c_{a_t} \end{pmatrix}$$

LQR partial derivation, final step

$$min_{a_1,\ldots,a_T} c(\boldsymbol{s}_1, \boldsymbol{a}_1) + c(f(\boldsymbol{s}_1, \boldsymbol{a}_1), \boldsymbol{a}_2) + \ldots + c(f(f(\ldots)), \boldsymbol{a}_T)$$

$$\boldsymbol{a}_T = \boldsymbol{K}_T \boldsymbol{s}_T + \boldsymbol{k}_T \qquad \boldsymbol{K}_T = -\boldsymbol{C}_{a_T, a_T}^{-1} \boldsymbol{C}_{a_t, s_t} \qquad \boldsymbol{k}_T = -\boldsymbol{C}_{a_T, a_T}^{-1} \boldsymbol{c}_{a_t}$$

State-value function (by substitution):

$$V(\boldsymbol{s}_T) = const + \frac{1}{2} \begin{pmatrix} \boldsymbol{s}_T \\ \boldsymbol{K}_T \boldsymbol{s}_T + \boldsymbol{k}_T \end{pmatrix}^T \boldsymbol{C}_T \begin{pmatrix} \boldsymbol{s}_T \\ \boldsymbol{K}_T \boldsymbol{s}_T + \boldsymbol{k}_T \end{pmatrix} + \begin{pmatrix} \boldsymbol{s}_T \\ \boldsymbol{K}_T \boldsymbol{s}_T + \boldsymbol{k}_T \end{pmatrix}^T \boldsymbol{c}_T$$

State value function is quadratic in $\boldsymbol{s}_T$!

$$V(\boldsymbol{s}_T) = const + \frac{1}{2} \boldsymbol{s}_T^T \boldsymbol{V}_T \boldsymbol{s}_T + \boldsymbol{s}_T^T \boldsymbol{v}_T$$

What about other time steps?

$$V(s_T) = const + \frac{1}{2} s_T^T V_T s_T + s_T^T v_T$$

LQR partial derivation, other steps

$$Q(s_t, a_t) = const + \overbrace{\frac{1}{2}\begin{pmatrix} s_t \\ a_t \end{pmatrix}^T C_t \begin{pmatrix} s_t \\ a_t \end{pmatrix} + \begin{pmatrix} s_t \\ a_t \end{pmatrix}^T}^{\text{quadratic}} c_t + \overbrace{V(f(s_t, a_t))}^{\text{quadratic}}$$

$$= const + \frac{1}{2}\begin{pmatrix} s_t \\ a_t \end{pmatrix}^T Q_t \begin{pmatrix} s_t \\ a_t \end{pmatrix} + \begin{pmatrix} s_t \\ a_t \end{pmatrix}^T q_t$$

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

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

Note: We skip here the derivation of V_t, v_t

Let's optimize the action! (how?)

$$V(s_T) = const + \frac{1}{2} s_T^T V_T s_T + s_T^T v_T$$

LQR partial derivation, other steps

$$Q(s_t, a_t) = const + \overbrace{\frac{1}{2} \begin{pmatrix} s_t \\ a_t \end{pmatrix}^T C_t \begin{pmatrix} s_t \\ a_t \end{pmatrix} + \begin{pmatrix} s_t \\ a_t \end{pmatrix}^T}^{\text{quadratic}} c_t + \overbrace{V(f(s_t, a_t))}^{\text{quadratic}}$$

$$= const + \frac{1}{2} \begin{pmatrix} s_t \\ a_t \end{pmatrix}^T Q_t \begin{pmatrix} s_t \\ a_t \end{pmatrix} + \begin{pmatrix} s_t \\ a_t \end{pmatrix}^T q_t$$

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

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

$$\nabla_{a_t} Q(s_t, a_t) = Q_{a_t, s_t} s_t + Q_{a_t, a_t} a_t + q_t^T = 0$$

$$\boxed{a_t = K_t s_t + k_t \qquad K_t = -Q_{a_t, a_t}^{-1} Q_{a_t, s_t} \qquad k_t = -Q_{a_t, a_t}^{-1} q_{a_t}}$$

Aalto University
School of Electrical
Engineering

Again: Optimal controller is linear in s_t !

LQR algorithm

Backward recursion:

For t = T down to 1

$$Q_t = C_t + F_t^T V_{t+1} F_t$$
$$q_t = c_t + F_t^T V_{t+1} f_t + F_t^T v_{t+1}$$
$$K_t = -Q_{a_t,a_t}^{-1} Q_{a_t,s_t}$$
$$k_t = -Q_{a_t,a_t}^{-1} q_{a_t}$$
$$V_t = Q_{s_t,s_t} + Q_{s_t,a_t} K_t + K_t^T Q_{a_t,s_t} + K_t^T Q_{a_t,a_t} K_t$$
$$v_t = q_{s_t} + Q_{s_t,a_t} k_t + K_t^T q_{a_t} + K_t^T Q_{a_t,a_t} k_t$$

First: compute the gains.

Forward recursion:

For t = 1 to T

$$a_t = K_t s_t + k_t$$
$$s_{t+1} = f(s_t, a_t)$$

Then: apply the law to compute controls.

System uncertainty / stochastic dynamics

Gaussian noise

$$f(\boldsymbol{s}_t, \boldsymbol{a}_t) = \boldsymbol{F}_t \begin{pmatrix} \boldsymbol{s}_t \\ \boldsymbol{a}_t \end{pmatrix} + \boldsymbol{f}_t + \boldsymbol{w}_t \qquad \boldsymbol{w}_t \sim N(\boldsymbol{0}, \boldsymbol{\Sigma}_t)$$

$$p(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t) \sim N\left(\boldsymbol{F}_t \begin{pmatrix} \boldsymbol{s}_t \\ \boldsymbol{a}_t \end{pmatrix} + \boldsymbol{f}_t, \boldsymbol{\Sigma}_t \right)$$

- A linear system with Gaussian noise can be controlled optimally using *separation principle*:
  - Use optimal observer (Kalman filter) to observe state
  - Control system using LQR with mean predicted state
- No change in algorithm!

But many systems are not linear?

Non-linear systems - Iterative LQR

- Approximate a non-linear system as a linear-quadratic

$$f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{F}_t \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}$$

$$c_t(\mathbf{s}_t, \mathbf{a}_t) = \frac{1}{2} \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}^T \mathbf{C}_t \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix} + \begin{pmatrix} \mathbf{s}_t \\ \mathbf{a}_t \end{pmatrix}^T \mathbf{c}_t$$

$$f(\mathbf{s}_t, \mathbf{a}_t) \approx f(\hat{\mathbf{s}}_t, \hat{\mathbf{a}}_t) + \nabla_{s_t, a_t} f(\hat{\mathbf{s}}_t, \hat{\mathbf{a}}_t) \begin{pmatrix} \mathbf{s}_t - \hat{\mathbf{s}}_t \\ \mathbf{a}_t - \hat{\mathbf{a}}_t \end{pmatrix}$$

$$c_t(\mathbf{s}_t, \mathbf{a}_t) \approx c(\hat{\mathbf{s}}_t, \hat{\mathbf{a}}_t) + \frac{1}{2} \begin{pmatrix} \mathbf{s}_t - \hat{\mathbf{s}}_t \\ \mathbf{a}_t - \hat{\mathbf{a}}_t \end{pmatrix}^T \nabla^2_{s_t, a_t} c(\hat{\mathbf{s}}_t, \hat{\mathbf{a}}_t) \begin{pmatrix} \mathbf{s}_t - \hat{\mathbf{s}}_t \\ \mathbf{a}_t - \hat{\mathbf{a}}_t \end{pmatrix} + \nabla_{s_t, a_t} c(\hat{\mathbf{s}}_t, \hat{\mathbf{a}}_t) \begin{pmatrix} \mathbf{s}_t - \hat{\mathbf{s}}_t \\ \mathbf{a}_t - \hat{\mathbf{a}}_t \end{pmatrix}$$

Note: System dynamics known and differentiable!

Non-linear systems - Iterative LQR

$$f(\boldsymbol{s}_t, \boldsymbol{a}_t) \approx f(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t) + \nabla_{s_t, a_t} f(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t) \begin{pmatrix} \boldsymbol{s}_t - \hat{\boldsymbol{s}}_t \\ \boldsymbol{a}_t - \hat{\boldsymbol{a}}_t \end{pmatrix}$$

$$c_t(\boldsymbol{s}_t, \boldsymbol{a}_t) = c(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t) + \frac{1}{2} \begin{pmatrix} \boldsymbol{s}_t - \hat{\boldsymbol{s}}_t \\ \boldsymbol{a}_t - \hat{\boldsymbol{a}}_t \end{pmatrix}^T \nabla^2_{s_t, a_t} c(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t) \begin{pmatrix} \boldsymbol{s}_t - \hat{\boldsymbol{s}}_t \\ \boldsymbol{a}_t - \hat{\boldsymbol{a}}_t \end{pmatrix} + \nabla_{s_t, a_t} c(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t) \begin{pmatrix} \boldsymbol{s}_t - \hat{\boldsymbol{s}}_t \\ \boldsymbol{a}_t - \hat{\boldsymbol{a}}_t \end{pmatrix}$$

$$\bar{f}(\delta \boldsymbol{s}_t, \delta \boldsymbol{a}_t) = \boldsymbol{F}_t \begin{pmatrix} \delta \boldsymbol{s}_t \\ \delta \boldsymbol{a}_t \end{pmatrix} \qquad \bar{c}_t(\delta \boldsymbol{s}_t, \delta \boldsymbol{u}_t) = \frac{1}{2} \begin{pmatrix} \delta \boldsymbol{s}_t \\ \delta \boldsymbol{a}_t \end{pmatrix}^T \boldsymbol{C}_t \begin{pmatrix} \delta \boldsymbol{s}_t \\ \delta \boldsymbol{a}_t \end{pmatrix} + \begin{pmatrix} \delta \boldsymbol{s}_t \\ \delta \boldsymbol{a}_t \end{pmatrix}^T \boldsymbol{c}_t$$

$$\nabla_{s_t, a_t} f(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t)$$

$$\nabla^2_{s_t, a_t} c(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t) \qquad \nabla_{s_t, a_t} c(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t)$$

Thus we have a LQR in $\delta \boldsymbol{s}_t, \delta \boldsymbol{a}_t$

Iterative LQR (iLQR) – Algorithm outline

Repeat

$$F_t = \nabla_{s_t, a_t} f(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t)$$

$$C_t = \nabla^2_{s_t, a_t} c(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t)$$

$$c_t = \nabla_{s_t, a_t} c(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t)$$

Run LQR backward pass with $\delta \boldsymbol{s}_t, \delta \boldsymbol{a}_t$

Run LQR forward pass with real dynamics and $\boldsymbol{a}_t = \boldsymbol{K}_t \delta \boldsymbol{s}_t + \boldsymbol{k}_t + \hat{\boldsymbol{a}}_t$

Update $\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{a}}_t$ to results of forward pass

until convergence

Practical considerations:

- Usually receding horizon is used: At every time-step, state is observed, iLQR is applied, and (only) first action is executed.
- On first iteration, gradients can be evaluated at starting point.

Good source for details: Tassa, Erez, Todorov (2012). Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization.

Planning by sampling – Shooting methods

Shooting methods: optimize actions

$$V(s_0) = min_{\boldsymbol{a}_0,\ldots,\boldsymbol{a}_{T-1}} c(\boldsymbol{s_0}, \boldsymbol{a_0}) + \ldots + c(f(f(\ldots)), \boldsymbol{a_{T-1}})$$

$$V(s_0) = max_{\boldsymbol{a}_0,\ldots,\boldsymbol{a}_{T-1}} R(\boldsymbol{s_0}, \boldsymbol{a_0}) + \ldots + R(f(f(\ldots)), \boldsymbol{a_{T-1}})$$

How to solve? Random shooting:

- Simulate multiple trajectories using random policy (remember Monte Carlo policy evaluation from lecture 3?)

$$Q_\pi(s_0, a_0) \approx \frac{1}{N} \sum_{i=0}^{N-1} \sum_{t=0}^{T-1} \gamma^t r_t$$

- Execute action with lowest cost / highest return
- Repeat

How to select policy? Uniformly random policy?

The cross-entropy method (CEM) general background

- Can be used to estimate $P_{\pi}\left(V\left(s\right)\geq d\right)$
- Monte Carlo estimation does not work well when $P_{\pi}\left(V\left(s\right)\geq d\right)$ is tiny
- CEM provides efficient estimation based on importance sampling (details in [De Boer 2005])
- The approach can be used also in optimization:
  - Select π to yield high probability for $P_{\pi}\left(V\left(s\right)\geq d\right)$
  - Increase d to reach higher V(s) values
  - Repeat

CEM for optimization

- Goal: maximize Q(a)
- Choose sampling distribution. We choose a Gaussian
  $\pi_\theta(a) = \mathcal{N}(a|\mu, \sigma^2)$
- While not converged:
  - Sample N samples a^i from current sampling distribution $\pi_\theta(a)$
  - Evaluate objective function $Q(a^i)$ at each a^i
  - Select M (M < N) samples a^i with the highest $Q(a^i)$
  - Fit parameters $\theta = (\mu, \sigma^2)$ to the selected samples
  - Repeat

How to use this in model-based RL?

Exercise 7 includes CEM

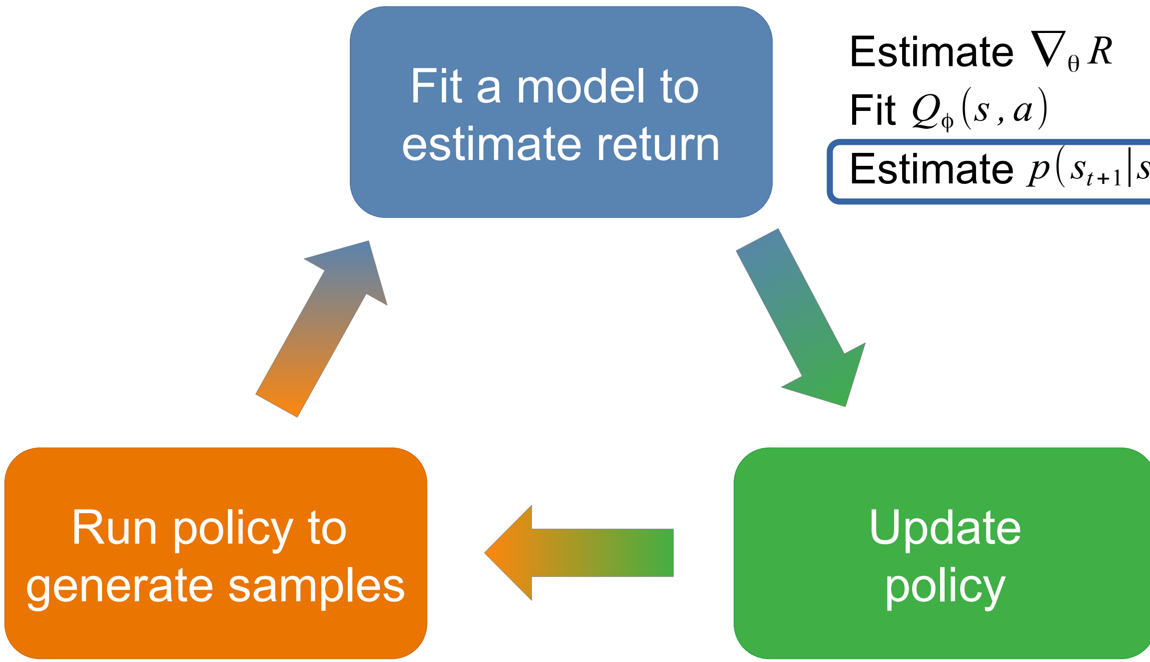
CEM in model-based RL

$$V\left(s_t^i\right)=\sum_{k=0}^{H} \gamma^k r_{t+k}^i$$

- Goal: maximize $V\left(s_0\right)$
- Sampling distribution at each time step t:
  $\pi_{\theta(t)}\left(a_t\right)=\mathcal{N}\left(a_t \mid \mu(t), \sigma^2(t)\right)$
- While not converged:
  - Perform Monte Carlo evaluation (Lecture 3) over N trajectories using sampling distribution and dynamics model
    → We get for trajectory i at time step t sample s_t^i with value $V\left(s_t^i\right)$
  - For each time step t fit parameters $\theta(t)=\left(\mu(t), \sigma^2(t)\right)$ of the sampling distribution to M (M < N) samples with the highest $V\left(s_t^i\right)$
  - Repeat

Are there other ways of using CEM in model-based RL?

Anatomy of reinforcement learning
Model-based



Estimate $\nabla_\theta R$	(policy gradient)
Fit $Q_\phi(s, a)$	(Q-learning, actor-critic)
$\boxed{\text{Estimate } p(s_{t+1} \mid s_t, a_t)}$	(model-based)

$\theta \leftarrow \theta + \alpha \nabla_\theta R(\theta)$	(policy gradient)
$arg\,max_a Q_\phi(s, a)$	(Q-learning)
$\boxed{\text{Optimize } \pi_\theta(a \mid s)}$	(model-based)

Aalto University
School of Electrical
Engineering

Next week: put these together.

Adopted from Sergey Levin.

Teaser: Basic iterative model-based RL

Input: base policy π_0

Run base policy to collect data $D \leftarrow \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')_i\}$

Repeat

  Fit dynamics model $f(\boldsymbol{s}, \boldsymbol{a})$ to minimize $\sum_i \|f(\boldsymbol{s}_i, \boldsymbol{a}_i) - \boldsymbol{s}_i'\|^2$

  Use model to plan (e.g. iLQR, CEM) actions

  Execute first planned action, observe resulting state $\boldsymbol{s}'$

  Update dataset $D \leftarrow D \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}')\}$

Viewpoint: Use learned model as “simulator” that allows exploring various options to choose one that is (locally) optimal.

Summary

- Optimal control for linear systems with quadratic costs can be determined with LQR
- Locally optimal control for nonlinear systems can be performed using linearization of dynamics in iterative LQR
- CEM allows for sample based planning with arbitrary costs/reward and dynamics
- Model-based reinforcement learning aims especially to increase data efficiency

Next: Model-based RL – again – but with learned models

- What kind of dynamics model to use?

- How to optimize a general policy function using a dynamics model?

- Reading: Sutton & Barto, ch. 8-8.2. No quiz for next week's lecture.