Aalto University
School of Electrical
Engineering

ELEC-E8125 Reinforcement Learning
Model-based reinforcement learning

Ville Kyrki

12.11.2019

Learning goals

- Understand basic approaches of model-based reinforcement learning.

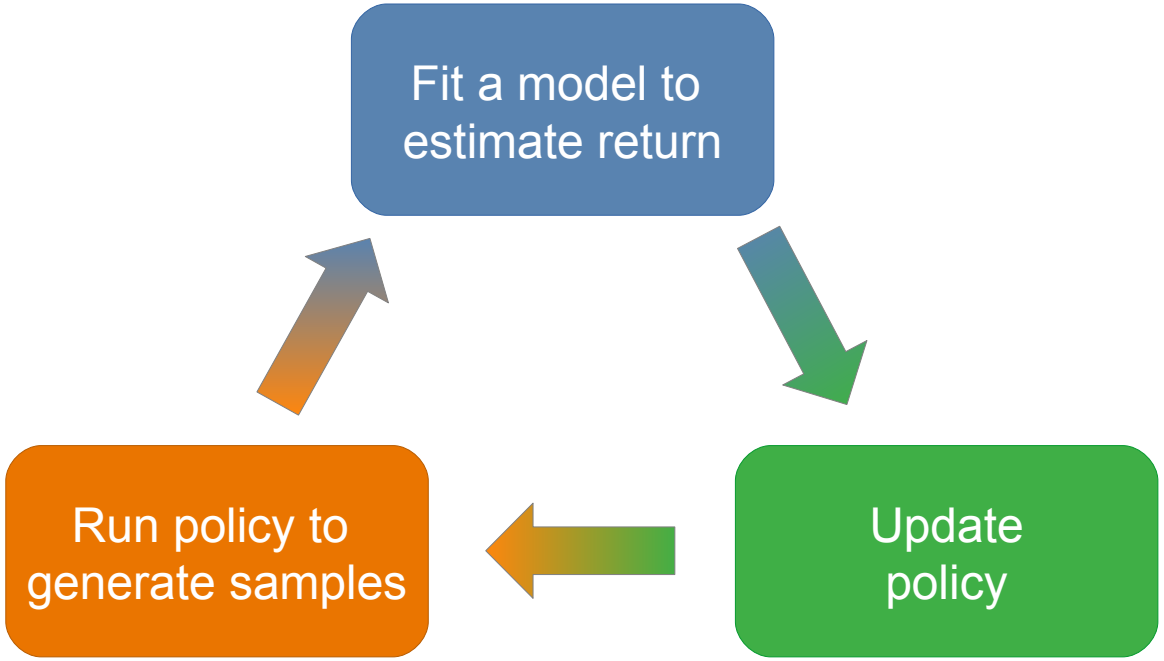
Motivation from two perspectives

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

- Learned policies are task(reward-function)-specific, learned knowledge cannot be reused.

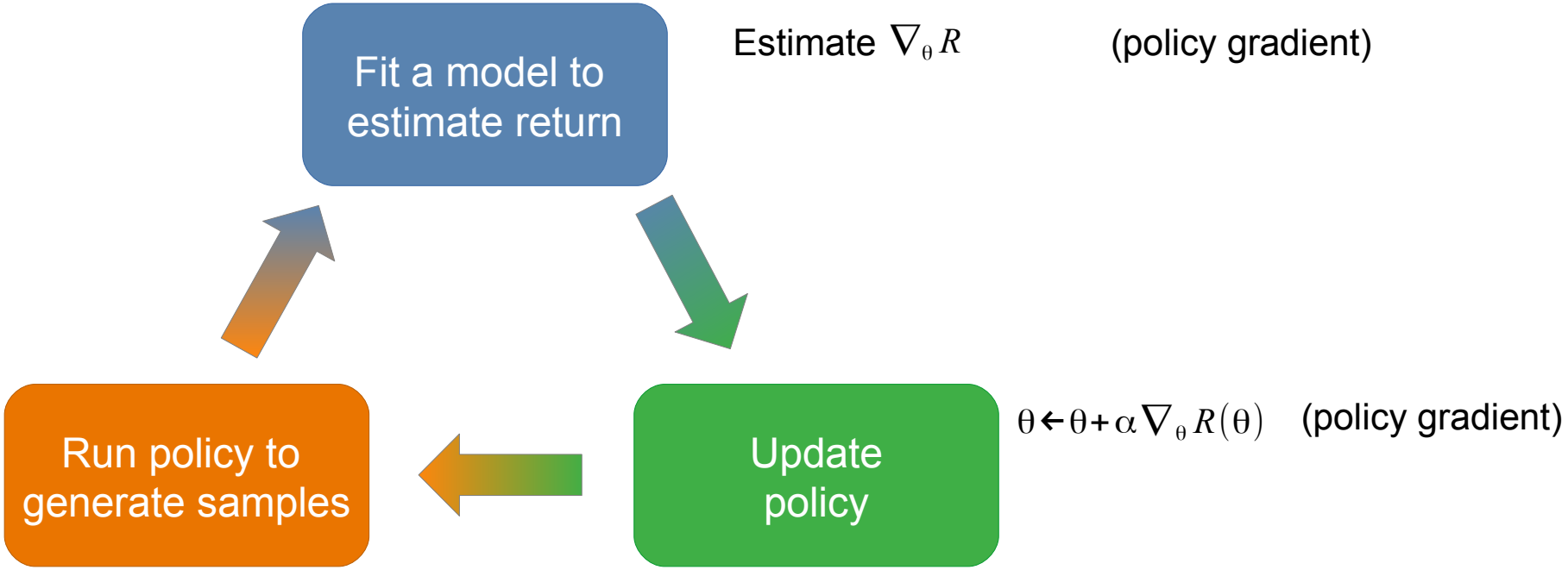
Can we somehow combine RL and optimal control?

Anatomy of reinforcement learning



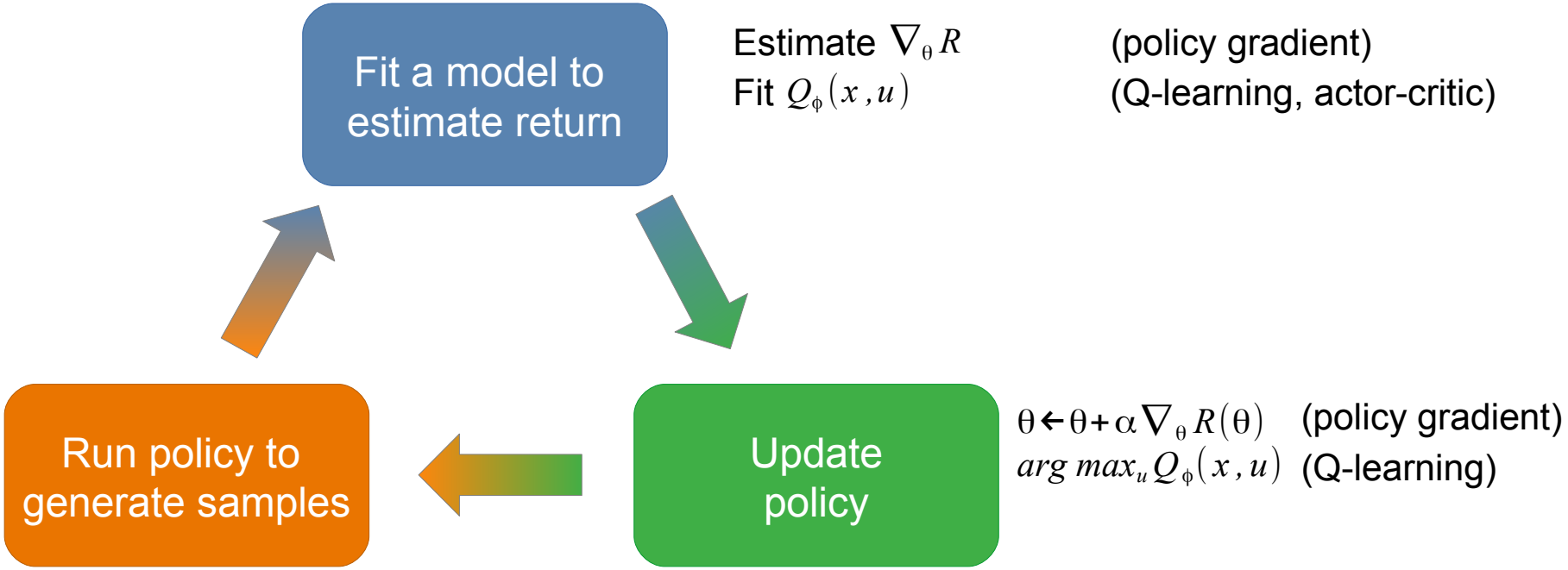
Adopted from Sergey Levin.

Anatomy of reinforcement learning
Policy gradient



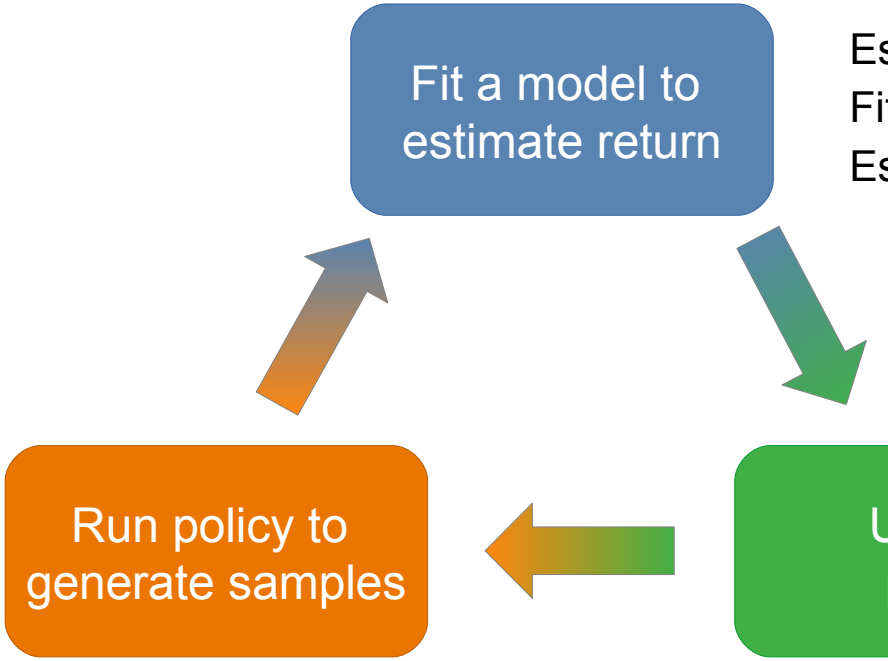
Adopted from Sergey Levin.

Anatomy of reinforcement learning Value-function based



Adopted from Sergey Levin.

Anatomy of reinforcement learning Model-based

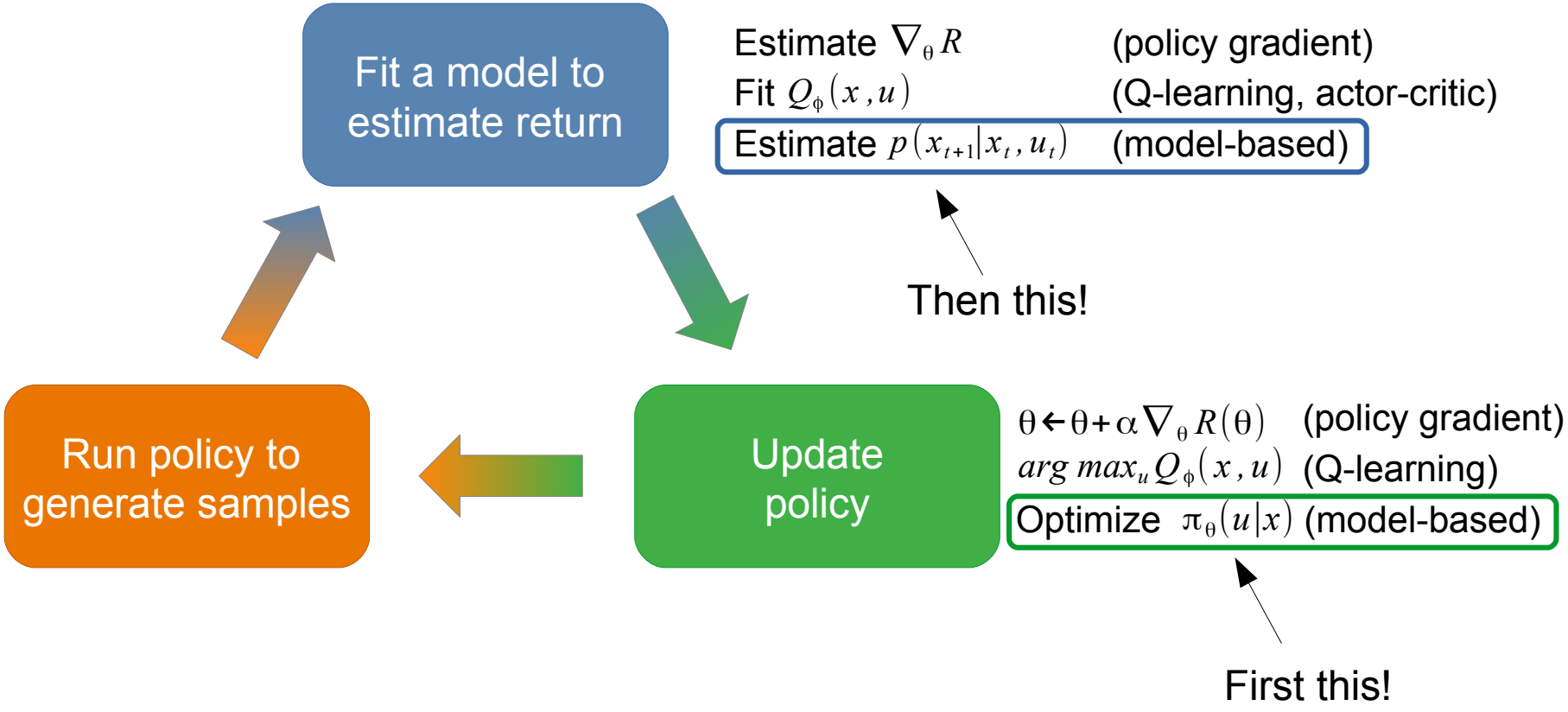


Estimate $\nabla_\theta R$ (policy gradient)
Fit $Q_\phi(x,u)$ (Q-learning, actor-critic)
Estimate $p(x_{t+1}|x_t,u_t)$ (model-based)

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

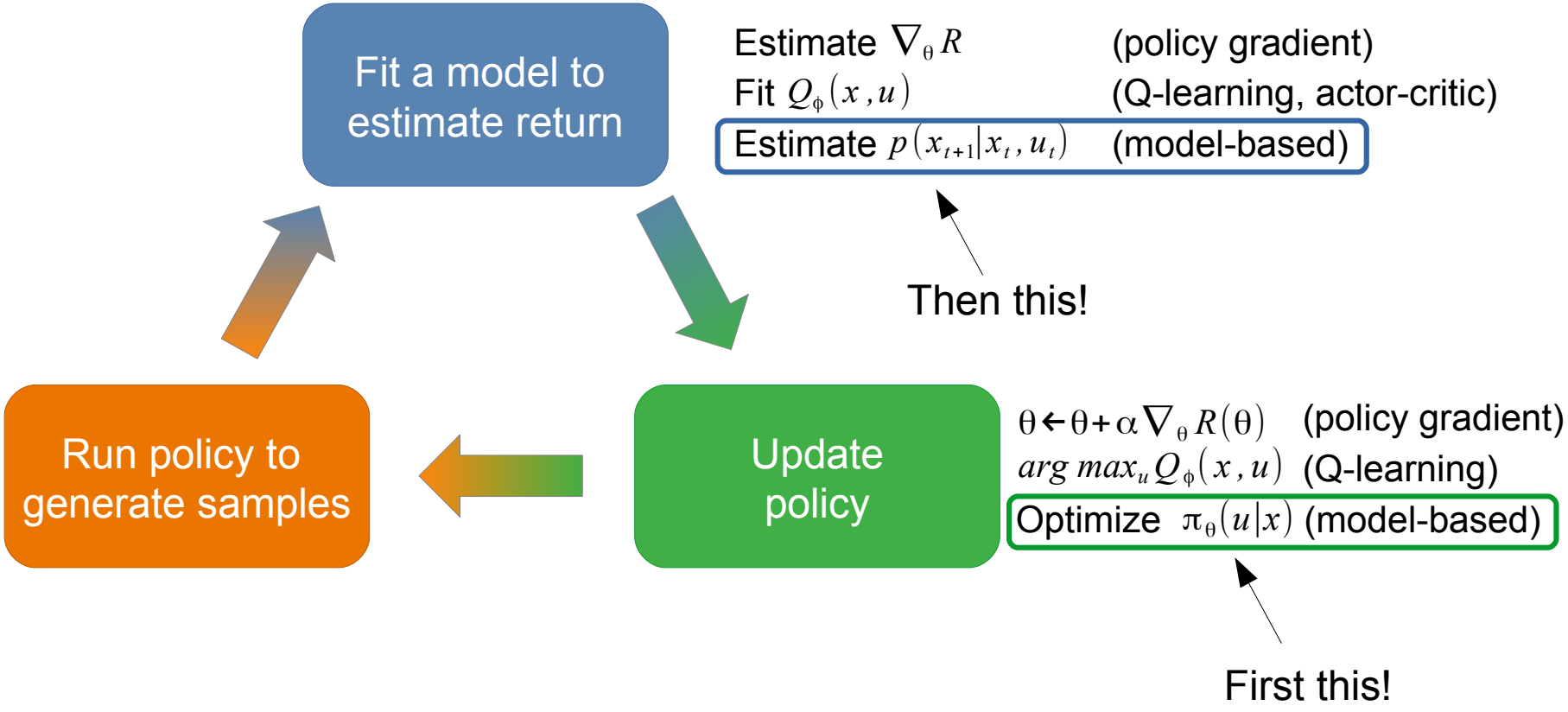
Adopted from Sergey Levin.

Anatomy of reinforcement learning Model-based



Adopted from Sergey Levin.

Anatomy of reinforcement learning Model-based



Let's start with a deterministic case.

Adopted from Sergey Levin.

Solving (deterministic, finite-horizon) optimal control problems

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

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

How to solve these?

Shooting vs collocation

Shooting methods: Optimize actions

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

Collocation methods: Optimize actions and states (constrained optimization)

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

How to solve optimal control with linear dynamics?

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

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

$$f(x_t,u_t)=\begin{pmatrix} A_t & 0 \\ 0 & B_t \end{pmatrix}\begin{pmatrix} x_t \\ u_t \end{pmatrix}+f_t=F_t\begin{pmatrix} x_t \\ u_t \end{pmatrix}+f_t$$

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

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.

$$C_t=\begin{pmatrix} C_{x_t,x_t} & C_{x_t,u_t} \\ C_{u_t,x_t} & 0 \quad C_{u_t,u_t} \end{pmatrix} \qquad c_t=\begin{pmatrix} c_{x_t} \\ c_{u_t} \end{pmatrix}$$

LQR partial derivation, final step

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

Only cost depending on u_T

$$f(x_t,u_t)=F_t\begin{pmatrix}x_t\\u_t\end{pmatrix}+f_t$$

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

Action-value function:

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

$$\nabla_{u_t} Q(x_T,u_T)=C_{u_T,x_T}x_T+C_{u_T,u_T}u_t+c_{u_t}=0$$

$$u_T=-C_{u_T,u_T}^{-1}\left(C_{u_t,x_t}x_t+c_{u_t}\right)$$

$$u_T=K_T x_T+k_T$$

$$K_T=-C_{u_T,u_T}^{-1}C_{u_t,x_t}$$

$$k_T=-C_{u_T,u_T}^{-1}c_{u_t}$$

$$C_t=\begin{pmatrix}C_{x_t,x_t} & C_{x_t,u_t}\\ C_{u_t,x_t} & 0\end{pmatrix}\qquad c_t=\begin{pmatrix}c_{x_t}\\c_{u_t}\end{pmatrix}$$

LQR partial derivation, final step

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

$$u_T = K_T x_T + k_T \qquad K_T = -C^{-1}_{u_T,u_T} C_{u_t,x_t} \qquad k_T = -C^{-1}_{u_T,u_T} c_{u_t}$$

State-value function (by substitution):

$$V(x_T) = const + \frac{1}{2}\begin{pmatrix} x_T \\ K_T x_T + k_T \end{pmatrix}^T C_T \begin{pmatrix} x_T \\ K_T x_T + k_T \end{pmatrix} + \begin{pmatrix} x_T \\ K_T x_T + k_T \end{pmatrix}^T c_T$$

State value function is quadratic in x_T !

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

What about other timesteps?

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

LQR partial derivation, other steps

quadratic

quadratic

$$Q(\boldsymbol{x}_t, \boldsymbol{u}_t) = const + \frac{1}{2}\begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix}^T \boldsymbol{C}_t \begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix} + \begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix}^T \boldsymbol{c}_t + V(f(\boldsymbol{x}_t, \boldsymbol{u}_t))$$

$$= const + \frac{1}{2}\begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix}^T \boldsymbol{Q}_t \begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix} + \begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix}^T \boldsymbol{q}_t$$

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

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

Note: We skip here the derivation of $\boldsymbol{V}_t, \boldsymbol{v}_t$

Let's optimize the action! (how?)

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

LQR partial derivation, other steps

quadratic

quadratic

$$Q(\boldsymbol{x}_t, \boldsymbol{u}_t) = const + \frac{1}{2}\begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix}^T \boldsymbol{C}_t \begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix} + \begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix}^T \boldsymbol{c}_t + V(f(\boldsymbol{x}_t, \boldsymbol{u}_t))$$

$$= const + \frac{1}{2}\begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix}^T \boldsymbol{Q}_t \begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix} + \begin{pmatrix}\boldsymbol{x}_t \\ \boldsymbol{u}_t\end{pmatrix}^T \boldsymbol{q}_t$$

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

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

$$\nabla_{\boldsymbol{u}_t} Q(\boldsymbol{x}_t, \boldsymbol{u}_t) = \boldsymbol{Q}_{\boldsymbol{u}_t, \boldsymbol{x}_t} \boldsymbol{x}_t + \boldsymbol{Q}_{\boldsymbol{u}_t, \boldsymbol{u}_t} \boldsymbol{u}_t + \boldsymbol{q}_t^T = 0$$

$$\boldsymbol{u}_t = \boldsymbol{K}_t \boldsymbol{x}_t + \boldsymbol{k}_t \qquad \boldsymbol{K}_t = -\boldsymbol{Q}_{\boldsymbol{u}_t, \boldsymbol{u}_t}^{-1} \boldsymbol{Q}_{\boldsymbol{u}_t, \boldsymbol{x}_t} \qquad \boldsymbol{k}_t = -\boldsymbol{Q}_{\boldsymbol{u}_t, \boldsymbol{u}_t}^{-1} \boldsymbol{q}_{\boldsymbol{u}_t}$$

Again: Optimal controller is linear in $\boldsymbol{x}_t$!

LQR algorithm

Backward recursion:

For t = T downto 1

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

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

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

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

$$V_t = Q_{x_t,x_t} + Q_{x_t,u_t} K_t + K_t^T Q_{u_t,x_t} + K_t^T Q_{u_t,u_t} K_t$$

$$v_t = q_{x_t} + Q_{x_t,u_t} k_t + K_t^T q_{u_t} + K_t^T Q_{u_t,u_t} k_t$$

First: compute the gains.

Forward recursion:

For t = 1 to T

$$u_t = K_t x_t + k_t$$

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

Then: apply the law to compute controls.

System uncertainty / stochastic dynamics

Gaussian noise

$$f(\boldsymbol{x}_t, \boldsymbol{u}_t) = \boldsymbol{F}_t \begin{pmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{pmatrix} + \boldsymbol{f}_t + \boldsymbol{w}_t \quad \boldsymbol{w}_t \sim N(\boldsymbol{0}, \boldsymbol{\Sigma}_t)$$

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

- 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

$$f(\boldsymbol{x}_t, \boldsymbol{u}_t) = \boldsymbol{F}_t \begin{pmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{pmatrix}$$

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

$$f(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \nabla_{x_t, u_t} f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \begin{pmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{pmatrix}$$

$$c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) = c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \frac{1}{2} \begin{pmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{pmatrix}^T \nabla^2_{x_t, u_t} c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \begin{pmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{pmatrix} + \nabla_{x_t, u_t} c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \begin{pmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{pmatrix}$$

Note: System dynamics known and differentiable!

Non-linear systems - Iterative LQR

$$f(\boldsymbol{x}_t, \boldsymbol{u}_t) \approx f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \nabla_{x_t, u_t} f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \begin{pmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{pmatrix}$$

$$c_t(\boldsymbol{x}_t, \boldsymbol{u}_t) = c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) + \frac{1}{2} \begin{pmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{pmatrix}^T \nabla^2_{x_t, u_t} c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \begin{pmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{pmatrix} + \nabla_{x_t, u_t} c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \begin{pmatrix} \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t \\ \boldsymbol{u}_t - \hat{\boldsymbol{u}}_t \end{pmatrix}$$

$$\bar{f}(\delta \boldsymbol{x}_t, \delta \boldsymbol{u}_t) = \boldsymbol{F}_t \begin{pmatrix} \delta \boldsymbol{x}_t \\ \delta \boldsymbol{u}_t \end{pmatrix}$$

$$\nabla_{x_t, u_t} f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$$

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

$$\nabla^2_{x_t, u_t} c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t) \qquad \nabla_{x_t, u_t} c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$$

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

Iterative LQR (iLQR) – Algorithm outline

Repeat

$$\boldsymbol{F}_t = \nabla_{x_t, u_t} f(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$$
$$\boldsymbol{C}_t = \nabla^2_{x_t, u_t} c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$$
$$\boldsymbol{c}_t = \nabla_{x_t, u_t} c(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t)$$

Run LQR backward pass with $\delta \boldsymbol{x}_t, \delta \boldsymbol{u}_t$

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

Update $\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t$ to results of forward pass

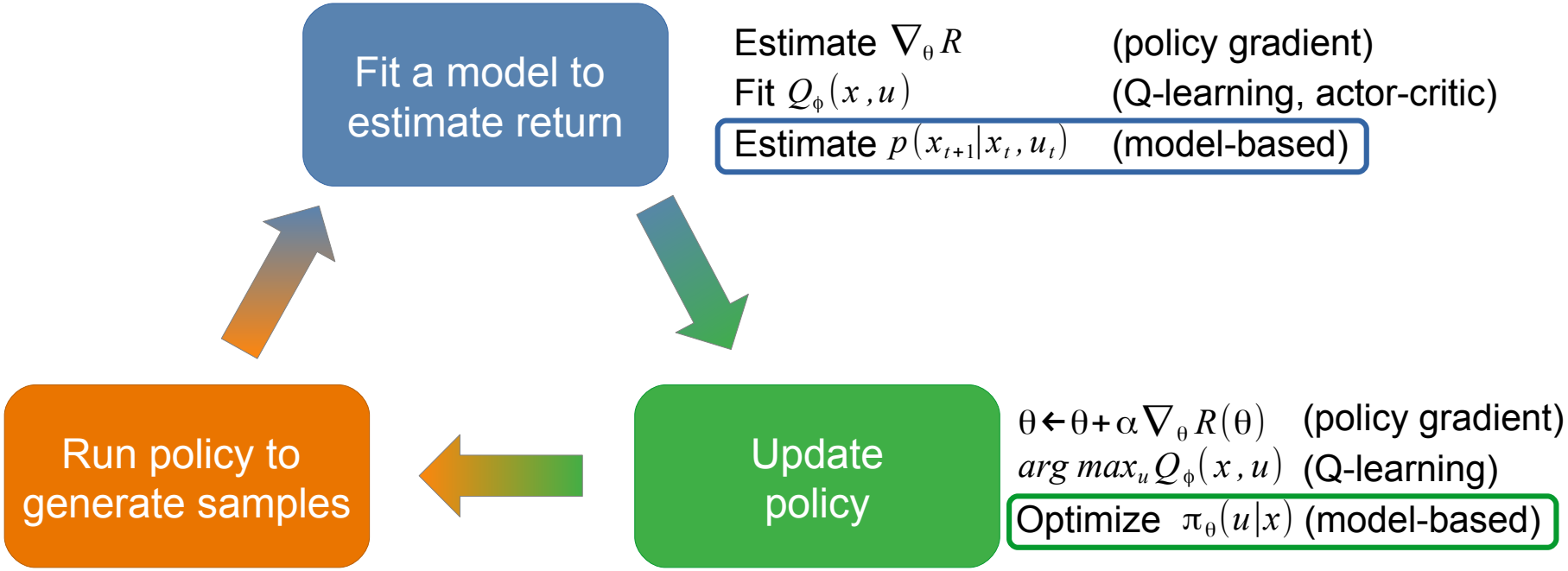
until convergence

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.

Anatomy of reinforcement learning
Model-based



Let's now put these together.

Adopted from Sergey Levin.

Basic iterative model-based RL

Input: base policy π_0

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

Repeat

- Fit dynamics model $f(\boldsymbol{x}, \boldsymbol{u})$ to minimize $\sum_i \| f(\boldsymbol{x}_i, \boldsymbol{u}_i) - \boldsymbol{x}_i' \|^2$
- Use model to plan (e.g. iLQR) actions
- Execute first planned action, observe resulting state $\boldsymbol{x}'$
- Update dataset $D \leftarrow D \cup \{(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')\}$

Basic iterative model-based RL

Input: base policy π_0

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

Repeat

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

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

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

Update dataset $D \leftarrow D \cup \{(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')\}$

Notes:

- Sample efficient.
- Computationally expensive for two reasons.
  - Dynamics fitting costly: model may be fitted only periodically (every n steps).
  - Planning costly for long horizons.
- Robust to moderate model errors.
- Choice of regression model is an important design parameter.

This is model-predictive control (MPC) with learned dynamics.
MPC horizon length is limited, can we do something?

Basic iterative model-based RL

Input: base policy π_0

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

Repeat

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

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

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

Update dataset $D \leftarrow D \cup \{(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')\}$

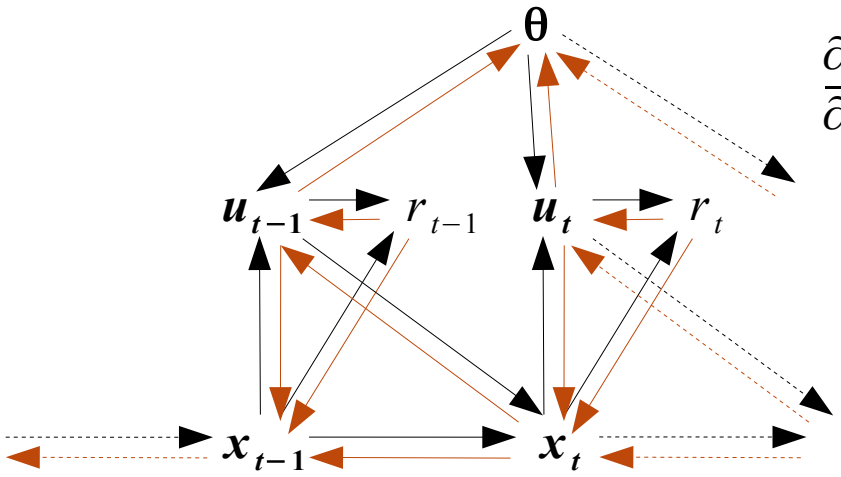
Notes:

- Sample efficient.
- Computationally expensive for two reasons.
  - Dynamics fitting costly: model may be fitted only periodically (every n steps).
  - **Planning costly for long horizons.**
- Robust to moderate model errors.
- Choice of regression model is an important design parameter.

This is model-predictive control (MPC) with learned dynamics.
MPC horizon length is limited, can we do something?

Backprop ~ chain rule of partial derivatives

Combining parametric policy with learned dynamics by backpropagation

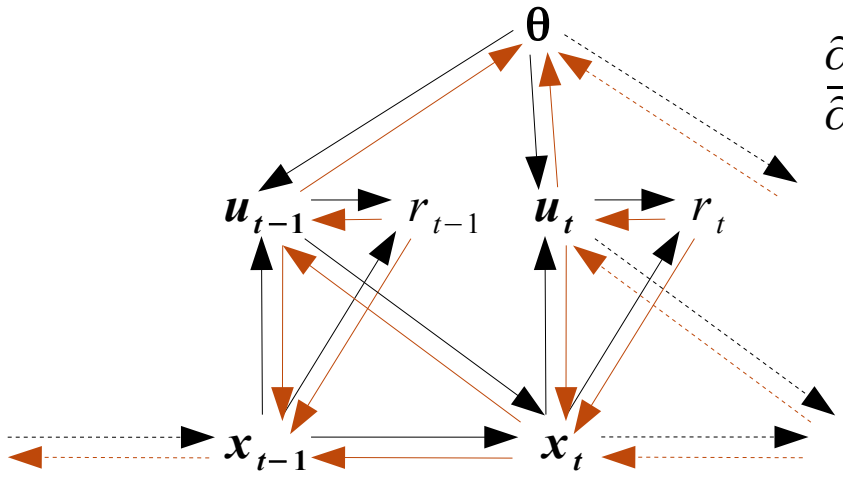


$$\frac{\partial r_t}{\partial \theta} = \frac{\partial r_t}{\partial u_t}\left(\frac{\partial u_t}{\partial \theta} + \frac{\partial u_t}{\partial x_t}\right) + \frac{\partial r_t}{\partial x_t}\left(\frac{\partial x_t}{\partial x_{t-1}} + \frac{\partial x_t}{\partial u_{t-1}}\right)$$

$$= \ldots$$

Backprop ~ chain rule of partial derivatives

Combining parametric policy with learned dynamics by backpropagation



$$\frac{\partial r_t}{\partial \theta} = \frac{\partial r_t}{\partial u_t}\left(\frac{\partial u_t}{\partial \theta} + \frac{\partial u_t}{\partial x_t}\right) + \frac{\partial r_t}{\partial x_t}\left(\frac{\partial x_t}{\partial x_{t-1}} + \frac{\partial x_t}{\partial u_{t-1}}\right)$$

$$= \ldots$$

Run base policy to collect data $D \leftarrow \{(x, u, x')_i\}$

Repeat

- Fit dynamics model $f_\phi(x, u)$ to minimize $\sum_i \| f_\phi(x_i, u_i) - x_i' \|^2$
- Calculate policy gradient by backpropagating through dynamics
- Execute updated policy (1 or more steps), collect data
- Update dataset $D \leftarrow D \cup \{(x, u, x')\}$

Basic iterative model-based RL

Input: base policy π_0

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

Repeat

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

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

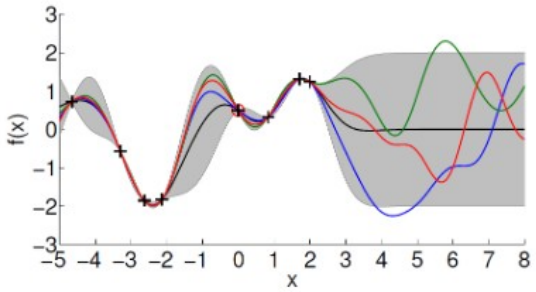
Execute first planned action, observe resulting state $\boldsymbol{x}'$

Update dataset $D \leftarrow D \cup \{(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')\}$

Notes:

- Sample efficient.
- Computationally expensive for two reasons.
  - Dynamics fitting costly: model may be fitted only periodically (every n steps).
  - Planning costly for long horizons.
- Robust to moderate model errors.
- **Choice of regression model is an important design parameter.**

Which model to use for dynamics?



Gaussian process (GP)

- Data-efficient
- Slow with big datasets
- May be too smooth for non-smooth dynamics

Neural networks

- Expressive
- Unpredictable with sparse data (overfit)

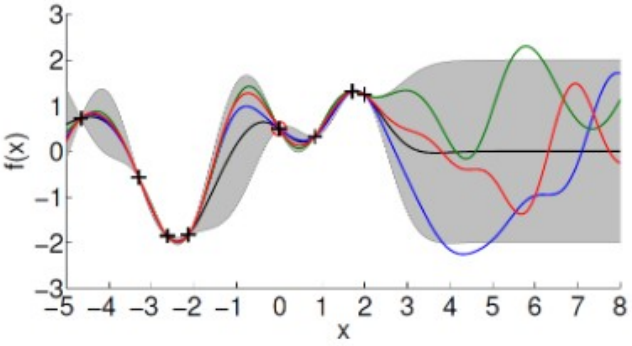
Linear models

- May be used locally
- Do not overfit

Domain specific parametric models (e.g. physics parameters) can also be used.
→ Traditional control engineering approach of model identification + control.

Example
PILCO (Deisenroth&Rasmussen, 2011)

- Dynamics learning: Use Gaussian process models to include model uncertainty.
- Simulation: Simulate trajectory with learned model, including uncertainty.
- Policy: Radial basis function.
- Policy update: Calculate analytically policy gradient using learned dynamics and optimize with quasi-Newton optimizer (BFGS).



What if reward function is unknown?

What about unknown rewards?

- Idea: Learn also regression function for rewards.
- BlackDROPS (2017) uses a Gaussian process to model reward function as well as dynamics.
- Uses CMA-ES (gradient free optimizer) for planning.

Summary

- Model-based RL requires typically less data than value-based or policy gradient approaches.
- Learned dynamics can be transferred across tasks.
- Potentially suboptimal: models do not optimize for task performance.
- Sometimes models are harder to learn than policy.
- Often require explicit choices (e.g. time horizon).

Next: Partial observability and POMDPs

- What changes if we cannot observe state directly?
- Reading: Tony Cassandra's on-line tutorial (see MyCourses for details)