Optimal Control for Modern Robotics:
Lecture 3

경희대학교
로봇 제어 및 지능 연구실
석박통합과정 2기 강민형

Optimal Control

Linear Quadratic Regulator (LQR)

Linear Dynamics

$$\dot{x} = Ax + Bu$$

$$x \in \mathbb{R}^n, \qquad u \in \mathbb{R}^m$$

Quadratic Cost

$$J = \int_0^\infty \left(x^\top Q x + u^\top R u\right) dt$$

$$Q = Q^\top \succeq 0, \qquad R = R^\top \succ 0$$

Objective: 최적의 cost–to–go function J^* 찾기

HJB

$$\min_u \left[x^\top Q x + u^\top R u + \frac{\partial J^*}{\partial x}(Ax + Bu) \right] = 0$$

이 식을 만족하는 u^*와 J^* 를 찾으면 그것이 바로 최적해가 된다

이 두 조건이 깨지면 LQR의 $u^*(x) = -Kx, \qquad K = R^{-1}B^\top S.$ 가 존재하지 않게 됨

Optimal Control

Linear Quadratic Regulator (LQR)

하지만 저번 시간에 LQR도 비선형 시스템을 선형화하는 방법을 배웠다

$$\dot{x} = f(x, u)$$

$$\text{operating point } (x_0, u_0), \qquad f(x_0, u_0) = 0$$

Equilibrium point 또는 steady state

Dynamics가 비선형이어서 사용이 어려운 문제를 steady state를 기준으로 선형화해서 제어 가능

$$\bar{x} = x - x_0, \qquad \bar{u} = u - u_0$$

$$\therefore \quad \dot{\bar{x}} = \dot{x} = f(x, u).$$

Taylor Expansion

$$\dot{\bar{x}} \approx f(x_0, u_0) + \left.\frac{\partial f}{\partial x}\right|_{(x_0, u_0)} (x - x_0) + \left.\frac{\partial f}{\partial u}\right|_{(x_0, u_0)} (u - u_0)$$

Optimal Control

Discrete Time

Continuous Time

$$\dot{x}(t) = Ax(t) + Bu(t)$$

Discrete Time

$$x_{k+1} = A_d x_k + B_d u_k$$

$$x((k+1)\Delta t) = A_d \, x(k\Delta t) + B_d \, u(k\Delta t)$$

$$\dot{x}(t) = Ax(t) \qquad x_k := x(k\Delta t), \qquad x_{k+1} := x((k+1)\Delta t)$$

$$x(t) = e^{At}x(0) \qquad x_{k+1} = x((k+1)\Delta t) = e^{A(k+1)\Delta t}x(0) = e^{A\Delta t}\, e^{Ak\Delta t}x(0) = e^{A\Delta t}x(k\Delta t) = e^{A\Delta t}x_k$$

$$x_{k+1} = A_d x_k, \qquad A_d = e^{A\Delta t}$$

Optimal Control

Discrete Time

Continuous Time

$$\dot{x}(t) = Ax(t) + Bu(t)$$

Discrete Time

$$x_{k+1} = A_d x_k + B_d u_k$$

$$x((k+1)\Delta t) = A_d\, x(k\Delta t) + B_d\, u(k\Delta t)$$

1D로 표현

$$\dot{x}(t) = a\, x(t) + b\, u_k$$

$$\dot{x} - ax = bu_k$$

$$\Rightarrow$$

$$e^{-at}(\dot{x} - ax) = bu_k e^{-at}$$

$$\frac{d}{dt}\left(e^{-at}x(t)\right) = bu_k e^{-at}$$

$$\Rightarrow$$

$$\int_{t_k}^{t_{k+1}} \frac{d}{dt}\left(e^{-at}x(t)\right) dt = \int_{t_k}^{t_{k+1}} bu_k e^{-at}\, dt$$

$$e^{-at_{k+1}}x_{k+1} - e^{-at_k}x_k = \frac{b}{a}u_k(e^{-at_k} - e^{-at_{k+1}})$$

$$\boxed{x_{k+1} = e^{a\Delta t}x_k + \frac{b}{a}(e^{a\Delta t} - 1)u_k} \qquad A_d = e^{a\Delta t}, \qquad B_d = \frac{b}{a}(e^{a\Delta t} - 1)$$

Optimal Control

● iterative LQR

이산화 -> 컴퓨터로 Trajectory Optimization 풀 준비 완료

iterative LQR

반복적으로 LQR을 사용

- Nonlinear dynamics

$$x_{k+1} = f(x_k, u_k)$$

- NonConvex Cost

$$J = \sum_{k=0}^{N-1} \ell(x_k, u_k) + \ell_f(x_N)$$

핵심 아이디어: 비선형 문제를 매 단계에서 선형 + 이차 (LQR 문제)로 근사해서 반복적으로 개선하는 알고리즘

- 지금 trajectory 근처에서 Linearize
- Linearized System을 LQR
- LQR 결과로 trajectory update
- 이 과정을 반복

Optimal Control

iterative LQR

문제 정의

$$x_{k+1} = f(x_k,\, u_k)$$

$$J = \sum_{k=0}^{N-1} \ell(x_k, u_k) + \ell_f(x_N)$$

다음을 최소로 만드는 x_k, u_k를 찾는 것 $X = \{x_0, x_1, \ldots, x_N\},\quad U = \{u_0, u_1, \ldots, u_{N-1}\}$ → Trajectory

iLQR의 목적 –〉 최적의 U를 찾아서 X를 만들고 cost를 최소화

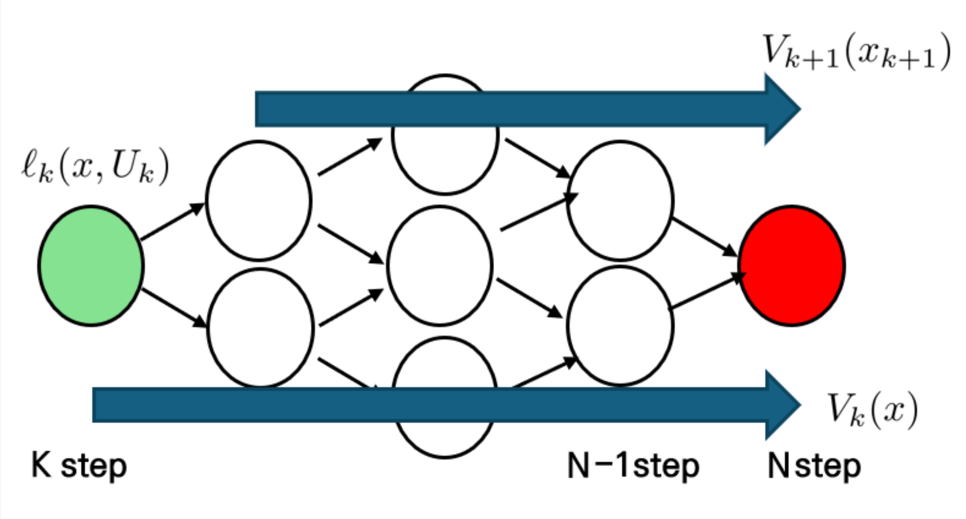
$$U^* = \arg\min_U \; J(x_0, U) \quad \Longrightarrow \quad \text{Bellman's principle of optimality}$$

Optimal Control

iterative LQR

Cost–to–go function

$$J_k(x_k, U_k) = \sum_{j=k}^{N-1} \ell(x_j, u_j) + \ell_f(x_N)$$: 시점 k부터 끝까지의 cost



지금 상태가 x일 때 앞으로 최적 행동만 했을 때 얻을 수 있는 최소 비용 Value Function $V_k(x) = \min_{U_k} J_k(x, U_k)$

Optimal Control

iterative LQR

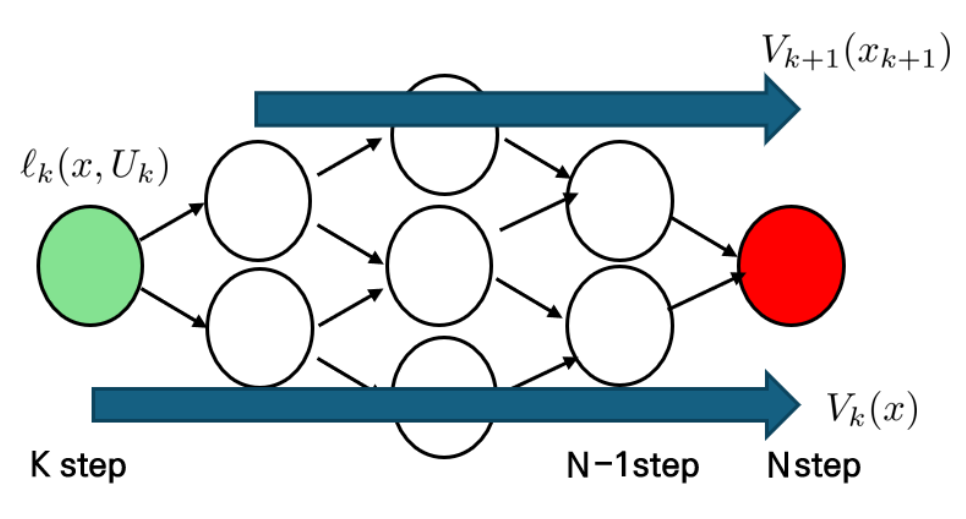
Bellman's Principle of Optimality

$$V_k(x) = \min_{u_k}\big(\ell_k(x,u_k) + V_{k+1}(x_{k+1})\big)$$

전체가 최적이라면 바로 다음의 작은 step도 최적이어야 한다

$$= \min_{u_k}\Big(\boxed{\ell_k(x,u_k)} + V_{k+1}\big(\boxed{f(x,u_k)}\big)\Big)$$

전부 비선형 -> 직접 최소화 불가능



Optimal Control

iterative LQR

우선 현재 초기 궤적 존재

$(\bar{x}_0,\ \bar{x}_1,\ \bar{x}_2,\ \ldots,\ \bar{x}_N), \qquad (\bar{u}_0,\ \bar{u}_1,\ \ldots,\ \bar{u}_{N-1})$ 이 궤적을 수정하면서 cost가 작아지는 새로운 궤적을 찾는다

궤적 근처를 생각

$x_k = \bar{x}_k + \delta x_k, \qquad u_k = \bar{u}_k + \delta u_k$ $\delta x_k, \delta u_k$ 를 어떻게 바꿨을 때 cost-go-function 값은 어떻게 되는가?

함수로 표현 가능 $Q_k(\delta x_k, \delta u_k)$

$$Q_k(\delta x_k, \delta u_k) = \ell_k(x_k + \delta x_k,\ u_k + \delta u_k) + V_{k+1}\big(f(x_k + \delta x_k,\ u_k + \delta u_k)\big).$$

Taylor Expansion

$$Q_k(\delta x, \delta u) \approx Q_k(0,0) + Q_{x,k}^\top\,\delta x + Q_{u,k}^\top\,\delta u + \frac{1}{2}\,\delta x^\top Q_{xx,k}\,\delta x + \frac{1}{2}\,\delta u^\top Q_{uu,k}\,\delta u + \delta u^\top Q_{ux,k}\,\delta x$$

Optimal Control

iterative LQR

Taylor Expansion

$$Q_k(\delta x, \delta u) \approx \boxed{Q_k(0,0)} + Q_{x,k}^\top \delta x + Q_{u,k}^\top \delta u + \frac{1}{2}\delta x^\top Q_{xx,k}\, \delta x + \frac{1}{2}\delta u^\top Q_{uu,k}\, \delta u + \delta u^\top Q_{ux,k}\, \delta x$$

$$\ell_k(x_k, u_k) + V_{k+1}(f(x_k, u_k))$$

iLQR에서는 dynamics 식 $x_{k+1} = f(x_k,\, u_k)$

1차로 근사

$\therefore$ f의 이계도함수가 존재하지 않음

이 부분이 Differntial Dynamic Programming

(DDP) 와의 차이점

$\therefore$ f 2차 근사까지 이루어지면 DDP

$$Q_x = \ell_x + f_x^\top V_{x,k+1} = \left.\frac{\partial Q}{\partial x}\right|_{x_k,u_k},$$

$$Q_u = \ell_u + f_u^\top V_{x,k+1} = \left.\frac{\partial Q}{\partial u}\right|_{x_k,u_k},$$

$$Q_{xx} = \ell_{xx} + f_x^\top V_{xx,k+1} f_x = \left.\frac{\partial^2 Q}{\partial x^2}\right|_{x_k,u_k},$$

$$Q_{xu} = \ell_{xu} + f_x^\top V_{xx,k+1} f_u = \left.\frac{\partial^2 Q}{\partial x\, \partial u}\right|_{x_k,u_k},$$

$$Q_{uu} = \ell_{uu} + f_u^\top V_{xx,k+1} f_u = \left.\frac{\partial^2 Q}{\partial u^2}\right|_{x_k,u_k}.$$

Minimize

$$\delta u^* = \arg\min_{\delta u} Q(\delta x, \delta u)$$

$$\nabla_{\delta u} Q = Q_u + Q_{uu}\, \delta u + Q_{ux}\, \delta x$$

Optimal Control

iterative LQR

iLQR $x_{k+1} = f(x_k, u_k)$ 의 1차 근사

$$Q_x = \ell_x + f_x^\top V_{x,k+1} = \left.\frac{\partial Q}{\partial x}\right|_{x_k,u_k},$$

$$Q_u = \ell_u + f_u^\top V_{x,k+1} = \left.\frac{\partial Q}{\partial u}\right|_{x_k,u_k},$$

$$Q_{xx} = \ell_{xx} + f_x^\top V_{xx,k+1} f_x = \left.\frac{\partial^2 Q}{\partial x^2}\right|_{x_k,u_k},$$

$$Q_{xu} = \ell_{xu} + f_x^\top V_{xx,k+1} f_u = \left.\frac{\partial^2 Q}{\partial x\,\partial u}\right|_{x_k,u_k},$$

$$Q_{uu} = \ell_{uu} + f_u^\top V_{xx,k+1} f_u = \left.\frac{\partial^2 Q}{\partial u^2}\right|_{x_k,u_k}.$$

DDP $x_{k+1} = f(x_k, u_k)$ 의 2차 근사

$$Q_x = \ell_x + f_x^\top V_{x,k+1} = \left.\frac{\partial Q}{\partial x}\right|_{x_k,u_k},$$

$$Q_u = \ell_u + f_u^\top V_{x,k+1} = \left.\frac{\partial Q}{\partial u}\right|_{x_k,u_k},$$

$$Q_{xx} = \ell_{xx} + f_x^\top V_{xx,k+1} f_x + V_{x,k+1} f_{xx} = \left.\frac{\partial^2 Q}{\partial x^2}\right|_{x_k,u_k},$$

$$Q_{ux} = \ell_{ux} + f_u^\top V_{xx,k+1} f_x + V_{x,k+1} f_{ux} = \left.\frac{\partial^2 Q}{\partial u\,\partial x}\right|_{x_k,u_k},$$

$$Q_{uu} = \ell_{uu} + f_u^\top V_{xx,k+1} f_u + V_{x,k+1} f_{uu} = \left.\frac{\partial^2 Q}{\partial u^2}\right|_{x_k,u_k}.$$

Optimal Control

iterative LQR

Minimize

$$\delta u^* = \arg\min_{\delta u} Q(\delta x, \delta u)$$

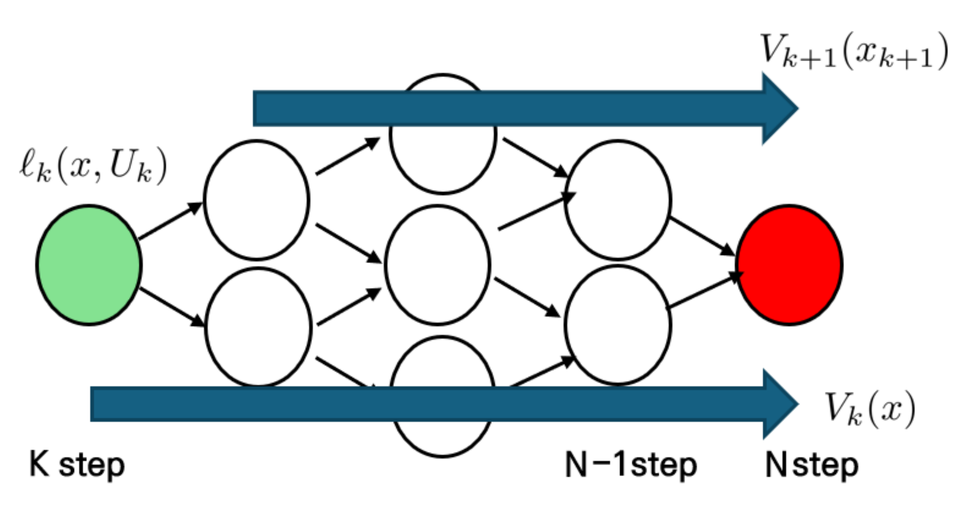
$$\nabla_{\delta u} Q = Q_u + Q_{uu}\,\delta u + Q_{ux}\,\delta x = 0$$

$$\boxed{\begin{gathered}\delta u^* = -Q_{uu}^{-1}Q_u - Q_{uu}^{-1}Q_{ux}\,\delta x \\ k + K\,\delta x\end{gathered}}$$

시점이 k일 때의 최적 제어 입력

우리는 final 값을 안다

Backward Pass



$$V_N(x) = \ell_f(x_N)$$

$$V_{x,N} = \left.\frac{\partial V}{\partial x}\right|_{x_N,\,u_N} = \ell_x(x_N).$$

$$\boxed{V_{xx,N} = \ell_{xx}, \qquad V_{x,N} = \ell_x.}$$

Optimal Control

iterative LQR

Backward Pass

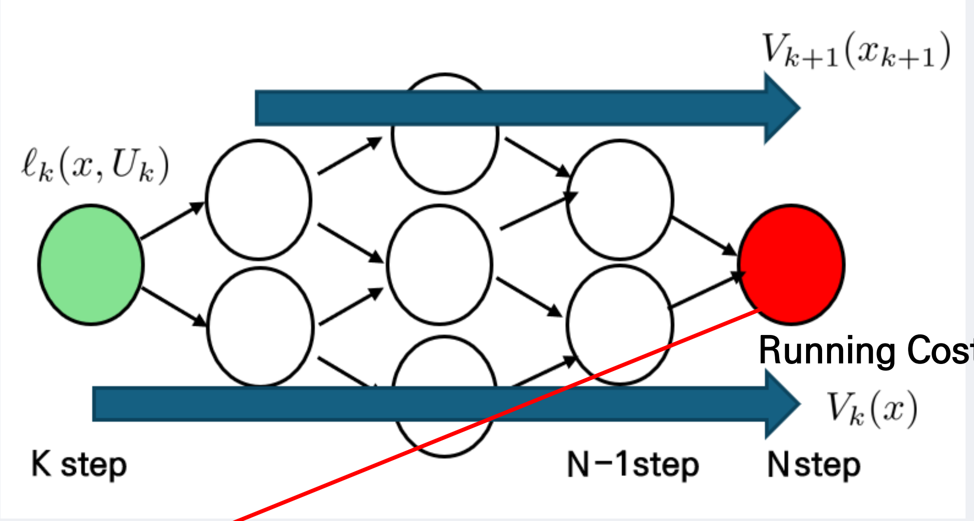
$$Q_x = \ell_x + f_x^\top V_{x,k+1} = \left.\frac{\partial Q}{\partial x}\right|_{x_k,u_k},$$

$$Q_u = \ell_u + f_u^\top V_{x,k+1} = \left.\frac{\partial Q}{\partial u}\right|_{x_k,u_k},$$

$$Q_{xx} = \ell_{xx} + f_x^\top V_{xx,k+1} f_x = \left.\frac{\partial^2 Q}{\partial x^2}\right|_{x_k,u_k},$$

$$Q_{xu} = \ell_{xu} + f_x^\top V_{xx,k+1} f_u = \left.\frac{\partial^2 Q}{\partial x\,\partial u}\right|_{x_k,u_k},$$

$$Q_{uu} = \ell_{uu} + f_u^\top V_{xx,k+1} f_u = \left.\frac{\partial^2 Q}{\partial u^2}\right|_{x_k,u_k}.$$



Running Cost 없음

$V_N(x) = \ell_f(x_N)$ u에 대한 영향 x −〉 상수 (시작점이 된다)

$$V_{x,N} = \left.\frac{\partial V}{\partial x}\right|_{x_N,u_N} = \ell_x(x_N).$$

$$\boxed{V_{xx,N} = \ell_{xx}, \qquad V_{x,N} = \ell_x.}$$

Also, 확정된 상수 값

Optimal Control

iterative LQR

Backward Pass

$$Q_x = \ell_x + f_x^\top V_{x,k+1} = \left.\frac{\partial Q}{\partial x}\right|_{x_k,u_k},$$

$$Q_u = \ell_u + f_u^\top V_{x,k+1} = \left.\frac{\partial Q}{\partial u}\right|_{x_k,u_k},$$

$$Q_{xx} = \ell_{xx} + f_x^\top V_{xx,k+1} f_x = \left.\frac{\partial^2 Q}{\partial x^2}\right|_{x_k,u_k},$$

$$Q_{xu} = \ell_{xu} + f_x^\top V_{xx,k+1} f_u = \left.\frac{\partial^2 Q}{\partial x\,\partial u}\right|_{x_k,u_k},$$

$$Q_{uu} = \ell_{uu} + f_u^\top V_{xx,k+1} f_u = \left.\frac{\partial^2 Q}{\partial u^2}\right|_{x_k,u_k}.$$

Step N 일 때의 값

$$V_{x,N} = \left.\frac{\partial V}{\partial x}\right|_{x_N,u_N} = \ell_x(x_N).$$

$$V_{xx,N} = \ell_{xx}, \qquad V_{x,N} = \ell_x.$$

이전 step N−1에서의 Q에 대한 정보를 얻을 수 있음

$$\begin{pmatrix} Q_{x,\,N-1} \\ Q_{u,\,N-1} \\ Q_{xx,\,N-1} \\ Q_{ux,\,N-1} \\ Q_{uu,\,N-1} \end{pmatrix}$$

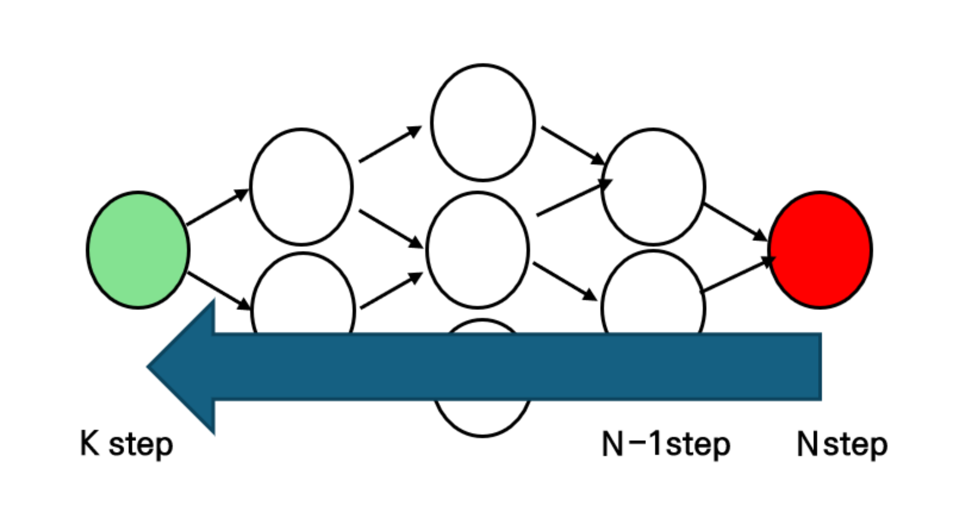
이전 step N−1에서의 최적 제어 입력 u

$$\delta u^*_{N-1} = -Q^{-1}_{uu,N-1} Q_{u,N-1} - Q^{-1}_{uu,N-1} Q_{ux,N-1}\,\delta x_{N-1}$$

Optimal Control

iterative LQR

이를 일반화하면 k+1 시점으로부터 k 시점에서의 Q에 대한 정보를 얻을 수 있다



$$\delta u^* = -Q_{uu}^{-1}Q_u \; - \; Q_{uu}^{-1}Q_{ux}\,\delta x.$$

각 step에서의 최적 제어 입력 도출

최적의 Value Function도 도출 가능

$$\delta u^* = k + K\,\delta x$$

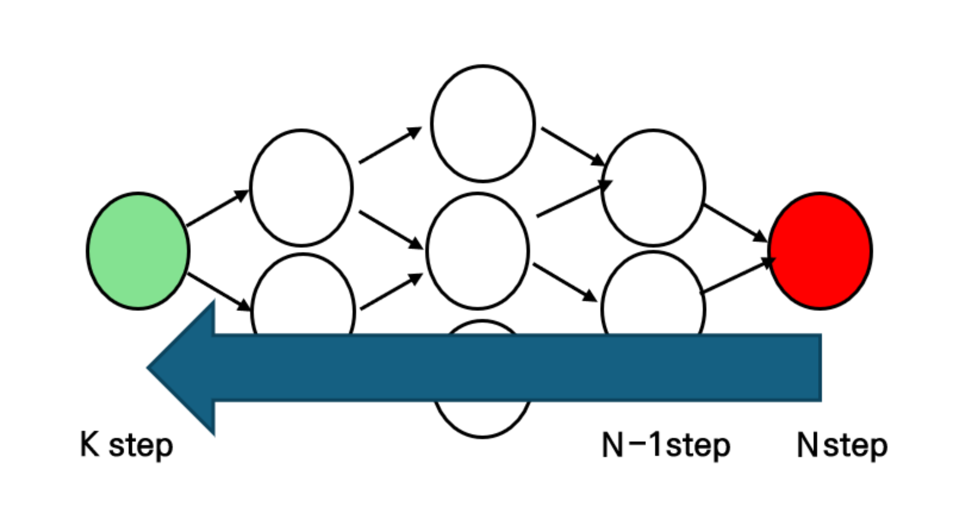
를 Value Function에 대입

$$\begin{aligned} &Q_{N-1}(x,u) + Q^{\mathsf{T}}_{x,\,N-1}\,\delta x + Q^{\mathsf{T}}_{u,\,N-1}(K_{N-1}\delta x + k_{N-1}) \\ &\quad + \tfrac{1}{2}\,\delta x^{\mathsf{T}} Q_{xx,\,N-1}\,\delta x + \tfrac{1}{2}(K_{N-1}\delta x + k_{N-1})^{\mathsf{T}} Q_{uu,\,N-1}(K_{N-1}\delta x + k_{N-1}) \\ &\quad + \delta x^{\mathsf{T}} Q_{xu,\,N-1}(K_{N-1}\delta x + k_{N-1}). \end{aligned}$$

Optimal Control

- iterative LQR

N step 값을 통해 N−1의 정보를 얻어내기



$$\delta u^* = -Q_{uu}^{-1}Q_u - Q_{uu}^{-1}Q_{ux}\,\delta x.$$

각 step에서의 최적 제어 입력 도출

최적의 Value Function도 도출 가능

δu^* 를 Q function에 넣었으니

앞으로 최적 행동만 했을 때 얻을 수 있는 최소 비용 Value Function $V_k(x) = \min_{U_k} J_k(x, U_k)$

$$V = \tfrac{1}{2}\,\delta x^{\mathsf{T}}\boxed{V_{xx,\,N-1}}\,\delta x + \boxed{V_{x,\,N-1}^{\mathsf{T}}}\delta x + V_{0,\,N-1}.$$

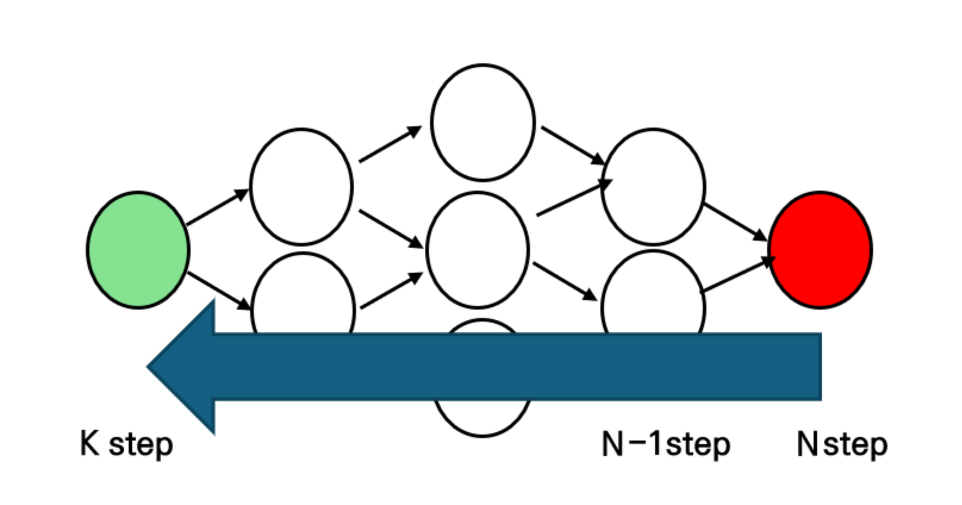
$$V_{xx,\,N-1} = Q_{xx,\,N-1} + K_{N-1}^{\mathsf{T}}Q_{uu,\,N-1}K_{N-1} + Q_{xu,\,N-1}^{\mathsf{T}}K_{N-1} + K_{N-1}^{\mathsf{T}}Q_{ux,\,N-1}.$$

$$V_{x,\,N-1} = Q_{x,\,N-1} + K_{N-1}^{\mathsf{T}}Q_{u,\,N-1} + Q_{xu,\,N-1}^{\mathsf{T}}k_{N-1} + K_{N-1}^{\mathsf{T}}Q_{uu,\,N-1}k_{N-1}.$$

Optimal Control

iterative LQR

반복 -> N-1, N-2, N-3 … k 시점까지 도달



$$\delta u^* = -Q_{uu}^{-1}Q_u - Q_{uu}^{-1}Q_{ux}\,\delta x.$$

각 step에서의 최적 제어 입력 도출

최적의 Value Function도 도출 가능

δu^* 를 Q function에 넣었으니

앞으로 최적 행동만 했을 때 얻을 수 있는 최소 비용 Value Function $V_k(x) = \min_{U_k} J_k(x, U_k)$

$$V = \frac{1}{2}\,\delta x^{\mathsf T} \boxed{V_{xx,\,N-1}}\,\delta x + \boxed{V_{x,\,N-1}^{\mathsf T}}\,\delta x + V_{0,\,N-1}.$$

$$V_{xx,\,N-1} = Q_{xx,\,N-1} + K_{N-1}^{\mathsf T} Q_{uu,\,N-1} K_{N-1} + Q_{xu,\,N-1}^{\mathsf T} K_{N-1} + K_{N-1}^{\mathsf T} Q_{ux,\,N-1}.$$

$$V_{x,\,N-1} = Q_{x,\,N-1} + K_{N-1}^{\mathsf T} Q_{u,\,N-1} + Q_{xu,\,N-1}^{\mathsf T} k_{N-1} + K_{N-1}^{\mathsf T} Q_{uu,\,N-1} k_{N-1}.$$

Optimal Control

iterative LQR

Forward Pass

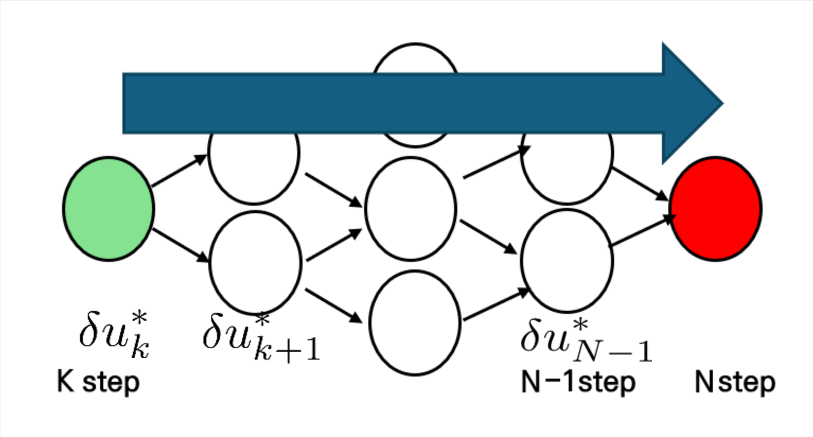
$$u_k^{\text{new}} = u_k + \delta u_k^*, \qquad x_0^{\text{new}} = x_0.$$

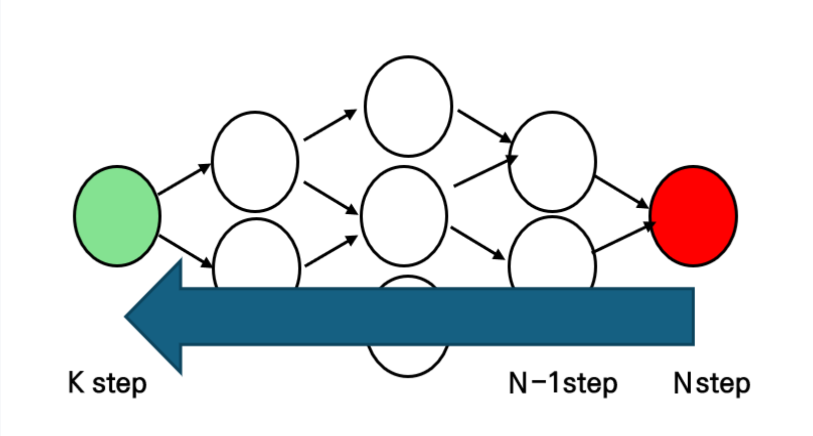
$$u_k^{\text{new}} = u_k + K_k(x_k^{\text{new}} - x_k) + k_k.$$

$$x_{k+1}^{\text{new}} = f(x_k^{\text{new}},\ u_k^{\text{new}})\,.$$

Backward Pass

$$\begin{pmatrix} Q_{x,\,N-1} \\ Q_{u,\,N-1} \\ Q_{xx,\,N-1} \\ Q_{ux,\,N-1} \\ Q_{uu,\,N-1} \end{pmatrix} \Rightarrow \begin{pmatrix} Q_{x,\,N-2} \\ Q_{u,\,N-2} \\ Q_{xx,\,N-2} \\ Q_{ux,\,N-2} \\ Q_{uu,\,N-2} \end{pmatrix} \Rightarrow \begin{pmatrix} Q_{x,\,k} \\ Q_{u,\,k} \\ Q_{xx,\,k} \\ Q_{ux,\,k} \\ Q_{uu,\,k} \end{pmatrix}$$





Optimal Control

https://github.com/minhyeong10/optimal_control_tutorial.git

● iterative LQR

```python
class InvertedPendulumEnv:
    """
    Inverted pendulum environment with:

      x = [sin(theta), cos(theta), omega]
      u = torque

    Dynamics are discretized with dt, and cost is quadratic around
    the upright equilibrium x = [0, 1, 0], u = 0.
    """

    def __init__(self):
        # ------------------------------
        # Environment parameters
        # ------------------------------
        self.m = 1.0
        self.l = 1.0
        self.g = 10.0
        self.dt = 0.05
        self.T = 200  # horizon length

        # ------------------------------
        # Cost function parameters
        # ------------------------------
        self.Q1 = 10.0    # weight on angle error via sin/cos
        self.Q2 = 0.1     # weight on angular velocity
        self.R = 0.001    # weight on control

        # ------------------------------
        # State / action dimensions
        # ------------------------------
        self.x = None
        self.u = None
        self.x_dim = 3
        self.u_dim = 1

        # control limits [umin, umax] for each dimension
        self.u_lims = np.array([[-2.0, 2.0]])
```

```python
    # Q-function 미분들
    Qu = cu_t + fu_t.T @ Vx_next               # (m,
    Qx = cx_t + fx_t.T @ Vx_next               # (n,

    Qxx = cxx_t + fx_t.T @ Vxx_next @ fx_t     # (n,
    Quu = cuu_t + fu_t.T @ Vxx_next @ fu_t     # (m,
    Qux = cux_t + fu_t.T @ Vxx_next @ fx_t     # (m,
```

```python
# ========== unconstrained (no control limit) ==
if lims is None or lims[0, 0] > lims[0, 1]:
    try:
        R = np.linalg.cholesky(QuuF)
    except np.linalg.LinAlgError:
        diverge = t + 1
        return diverge, Vx, Vxx, k, K, dV

    # Solve for k,K:  [Qu | Qux_reg]
    rhs = np.concatenate((Qu[:, None], Qux_reg), axis=1)  # (m, 1+n)
    kK = np.linalg.solve(-R, np.linalg.solve(R.T, rhs))   # (m, 1+n)

    k_t = kK[:, 0]
    K_t = kK[:, 1 : 1 + n]
```

```python
for t in range(N):
    # 기본: nominal control 복제
    unew[t, :, :] = np.tile(u[:, t][:, None], (1, K))

    # feedforward term (du * alpha)
    if du is not None:
        du_t = du[:, t][:, None]                # (m,1)
        unew[t, :, :] += du_t @ Alpha[None, :]  # (m,K)

    # feedback term: L * (x - x_nominal)
    if L is not None:
        dx = xnew[t, :, :] - np.tile(x[:, t][:, None], (1, K))  # (n,K)
        unew[t, :, :] += L[t, :, :] @ dx                         # (m,K)
```

감사합니다.