Model Predictive Control and Reinforcement Learning
– Lecture 7.2: Imitation Learning from Nonlinear MPC –

Andrea Ghezzi

Fall School on Model Predictive Control and Reinforcement Learning
Freiburg, 6-10 October 2025

universität freiburg

NTNU | Norwegian University of Science and Technology

Disclaimer

It is the first time this talk is given, watch out for typos :/

Outline of the lecture

Implicit and explicit model predictive control

Loss functions for imitation learning

Improving performance of learned controllers

Data collection – how to sample?

Verification of learned controllers

Model predictive control

A control technique that let us specify performance objectives, system dynamics and properties in optimal control problem (OCP).

Compute the next control by solving the OCP at the given state. Tackled in two ways:

- **Implicit MPC.** Control policy computed online
- **Explicit MPC.** Control policy offline and only evaluated online
  - For linear MPC: exact policy representation via piece-wise affine functions
  - For nonlinear MPC: only possible to resort to approximations $\Rightarrow$ Online computation preferred, but: realtime requirements, computational power, ...

Explicit linear MPC

- convex quadratic cost
- linear dynamics
- affine constraints
- parametric in $\bar{x}_0$

$$\begin{aligned} \min_{\mathbf{x},\,\mathbf{u}} \quad & x_N^\top P x_N + \frac{1}{N}\sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k \\ \text{s.t.} \quad & x_0 - \bar{x}_0 = 0, \\ & x_{k+1} - A x_k - B u_k = 0,\ k \in [N-1], \\ & C x_k + D u_k + c_k \leq 0. \end{aligned}$$

Feedback law:

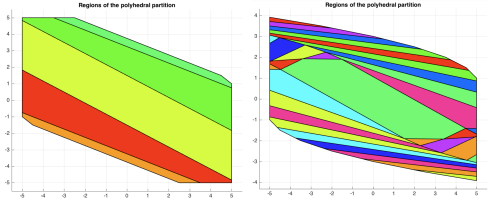
$$u_0^*(x_0) = \begin{cases} K_1 x_0 + d_1 & \text{if } H_1 x_0 \leq h_1, \\ \vdots & \\ K_M x_0 + d_M & \text{if } H_M x_0 \leq h_M. \end{cases}$$

Small example:

- $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, $B = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$
- $Q = \mathrm{diag}(1, 1)$, $R = 1$
- $(-5, -5) \leq x_k \leq (5, 5)$, $-1 \leq u_k \leq 1$

Resulting polyhedral partition for $N = 2$, $N = 10$

- x-axis: $x^{[1]}$, y-axis: $x^{[2]}$, color: feedback law



Function approximation via deep neural networks

Clear that we will soon have issues with storing and retrieving the right feedback law!

Not suited for nonlinear MPC – the feedback law is not PWA!

Observation: MPC is a **parametric** problem defining an implicit map $x_0 \to u^*$ (simplest case)

Can we learn this map?

Imitation Learning

"Definition." Learn a policy by imitating an expert policy in a supervised manner.

Expert policy. Any policy $\pi^\star$ that accomplishes the considered task in the way we desire (safe, time/energy-optimal, feasible, ...)

Aim. Approximate $\pi^\star$ as well as possible by a parameterized policy $\pi(\cdot\,;\theta) : \mathbb{R}^{n_x} \to \mathbb{R}^{n_u}$

- A parameterized policy could be a Neural Network where the parameters $\theta \in \mathbb{R}^{n_\theta}$ are the weights of the Neural Network.

Imitation Learning

The Imitation Learning objective can be defined generically as

$$\begin{aligned} \mathcal{L}(\theta) &:= \mathbb{E}_{x\sim\mathcal{D}}\left[\ell\left(x, \pi(\cdot \mid x; \theta)\right)\right], && \text{with a stochastic policy} \\ &:= \mathbb{E}_{x\sim\mathcal{D}}\left[\ell\left(x, \pi(x; \theta)\right)\right], && \text{with a deterministic policy} \end{aligned}$$

where

- ℓ the point-wise loss function of the policy $\pi(\cdot\,;\theta)$ for a given state x
- $\mathcal{D}$ is a given state distribution over $\mathbb{R}^{n_x}$

The optimal combination of parameters $\theta^\star$ that minimizes the expected loss $\mathcal{L}(\theta)$ is given by

$$\theta^\star = \arg\min_{\theta\in\mathbb{R}^\theta} \mathcal{L}(\theta).$$

Exact representation of linear MPC via ReLU NN

Old idea: [Parisini and Zoppoli, 1995]

- However, limited to small systems
- Adoption of NN with 1 wide layer due to *Universal Function Approximation*

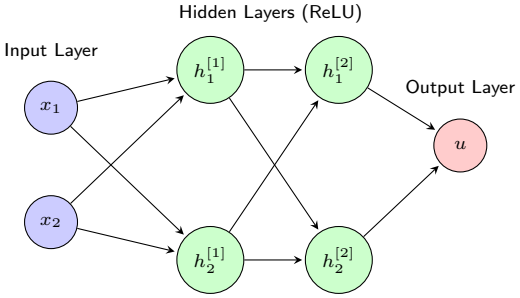
Exact representation of linear MPC via ReLU NN

Old idea: [Parisini and Zoppoli, 1995]

- However, limited to small systems
- Adoption of NN with 1 wide layer due to *Universal Function Approximation*

Why rediscovered? Thanks to deep learning, number of linear regions grows exponentially with number of layers [Montufar, 2014]

$$n_{\mathrm{r}} = \left(\prod_{l=1}^{L-1} \left\lfloor \frac{M}{n_x} \right\rfloor^{n_x} \right) \sum_{j=0}^{n_x} \binom{L}{j}, \text{ with } M \geq n_x$$



Exact representation of linear MPC via ReLU NN

Old idea: [Parisini and Zoppoli, 1995]

- However, limited to small systems
- Adoption of NN with 1 wide layer due to *Universal Function Approximation*

Why rediscovered? Thanks to deep learning, number of linear regions grows exponentially with number of layers [Montufar, 2014]

$$n_{\mathrm{r}} = \left(\prod_{l=1}^{L-1} \left\lfloor \frac{M}{n_x} \right\rfloor^{n_x} \right) \sum_{j=0}^{n_x} \binom{L}{j}, \text{ with } M \geq n_x$$

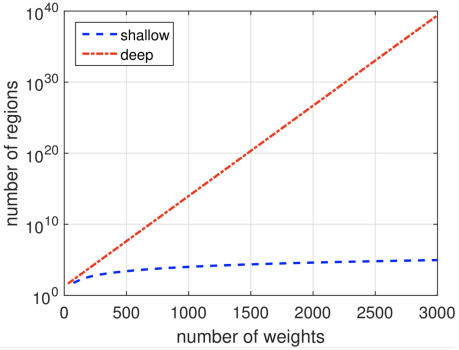


Figure: $n_x = 2$, $n_u = 4$, layers $L \in [1, 50]$, neurons $M = 10$. [Karg and Lucia, 2020]

Possible to compute the number of neurons and layers necessary to exactly represent the solution of a linear MPC problem.

Outline of the lecture

Implicit and explicit model predictive control

Loss functions for imitation learning

Improving performance of learned controllers

Data collection – how to sample?

Verification of learned controllers

Imitation Learning

The Imitation Learning objective can be defined generically as

$$\mathcal{L}(\theta) \coloneqq \mathbb{E}_{x \sim \mathcal{D}} \left[\ell \left(x, \pi(x; \theta) \right) \right],$$

where

- ℓ the point-wise loss function of the policy $\pi(\cdot\,; \theta)$ for a given state x
- $\mathcal{D}$ is a given state distribution over $\mathbb{R}^{n_x}$

The optimal combination of parameters $\theta^\star$ that minimizes the expected loss $\mathcal{L}(\theta)$ is given by

$$\theta^\star = \arg\min_{\theta \in \mathbb{R}^\theta} \mathcal{L}(\theta).$$

Behavioral Cloning

In general, Imitation Learning assumes no prior knowledge about the internal objective of the expert policy (e.g., human expert) $\Rightarrow$ **Behavioral Cloning (BC)**

BC assumes a surrogate loss function ℓ that measures the behavioral difference between the policy π and the expert policy $\pi^\star$.

Behavioral Cloning

In general, Imitation Learning assumes no prior knowledge about the internal objective of the expert policy (e.g., human expert) $\Rightarrow$ **Behavioral Cloning (BC)**

BC assumes a surrogate loss function ℓ that measures the behavioral difference between the policy π and the expert policy $\pi^\star$.

A popular choice is the quadratic loss function ℓ_2 defined as

$$\ell_2(x, \pi) \coloneqq \left(\pi(x) - \pi^\star(x)\right)^2,$$

which results in the following expected quadratic loss:

$$\mathcal{L}^2(\theta) \coloneqq \mathbb{E}_{x \sim \mathcal{D}} \left[\ell_2\left(x, \pi(\cdot\,; \theta)\right)\right].$$

Other popular loss: Huber loss, ℓ_1 loss, cross-entropy loss in the case of stochastic policies.

Optimal control problem formulation

Differently from BC we want to imitate a MPC controller, which solves the discrete-time OCP

Optimal control problem formulation

Differently from BC we want to imitate a MPC controller, which solves the discrete-time OCP

$$\min_{\substack{x_0,u_0,s_0,\dots,\\ u_{N-1},x_N,s_N}} \quad \sum_{k=0}^{N-1} \tilde{L}(x_k,u_k,s_k) + \tilde{E}(x_N,s_N)$$

$$\text{s.t.}$$

$$\begin{aligned}
&x_0 = \bar{x}_0, \\
&x_{k+1} = f(x_k,u_k), k=0,\dots,N-1, \\
&h(x_k,u_k) \le s_k, \quad k=0,\dots,N-1, \\
&r(x_N) \le s_N, \\
&s_k \ge 0, \qquad\qquad k=0,\dots,N,
\end{aligned}$$

- $x_k \in \mathbb{R}^{n_x}$ and $u_k \in \mathbb{R}^{n_u}$ represent the state and control, respectively
- Functions L, E, f, h, r are twice continuously differentiable in their respective variables.
- $s_k \in \mathbb{R}^{n_{s,k}}$ are **slack variables** and we penalize their use in the cost function
  - stage cost: $\tilde{L}(x_k,u_k,s_k) := L(x_k,u_k) + z^\top s_k + \|s_k\|_Z^2$
  - terminal cost: $\tilde{E}(x_N,s_N) := E(x_N) + z_e^\top s_N + \|s_N\|_{Z_e}^2$

The exact Q-loss

- **Our expert is the NMPC policy** $\implies \pi^\star(x) \coloneqq u_0^\star(x)$, with $x = \bar{x}_0$.
- In such case the internal objective of the expert is known.
- Given an initial state for the OCP, it is possible to assign a cost to every possible control.
  - Concept of Q-value (state-action value) in Optimal Control / Reinforcement Learning.
- Approximate Q-values can be computed by solving the OCP associated with a given state-action pair.

The exact Q-loss

Idea. Fix the first control u_0 of the OCP by the value returned from the policy, $\bar{u}_0 = \pi(x;\theta)$, then solve the resulting OCP to assign a cost to the policy value $\pi(x;\theta)$.

$\Rightarrow$ Given $\bar{x}_0$ we define the exact Q-loss by the following "Q-function OCP"

$$
\begin{aligned}
Q(\bar{x}_0, \bar{u}_0) \coloneqq \min_{\substack{x_0,u_0,s_0,\dots,\\ u_{N-1},x_N,s_N}} \quad & \sum_{k=0}^{N-1} \tilde{L}(x_k, u_k, s_k) + \tilde{E}(x_N, s_N) \\
\text{s.t.} \quad & x_0 - \bar{x}_0 = 0, \\
& u_0 - \bar{u}_0 = 0, \\
& x_{k+1} - f(x_k, u_k) = 0, \; k = 0, \dots, N-1, \\
& h(x_k, u_k) \leq s_k, \qquad k = 0, \dots, N-1, \\
& r(x_N) \leq s_N, \\
& s_k \geq 0, \qquad k = 0, \dots, N,
\end{aligned}
$$

The exact Q-loss

Lemma (Gradient computation)

The gradient of the Q-loss is given by the Lagrangian multiplier $\bar{\lambda}_u$ corresponding to the constraint $u_0^\star - \bar{u}_0 = 0$ for the optimal solution

$$\bar{\lambda}_u = \nabla_u Q(x, u)|_{u=\pi(x;\theta)} .$$

Lemma (Distance function)

If $\pi^\star(\bar{x}_0)$ is a unique minimizer of the original OCP then $Q(\bar{x}_0, \bar{u}_0) > Q(\bar{x}_0, \pi^\star(\bar{x}_0))$ for any $\bar{u}_0 \neq \pi^\star(\bar{x}_0)$. Thus, the exact Q-loss penalizes any deviation of $\bar{u}_0$ from $\pi^\star(\bar{x}_0)$.

The exact Q-loss

With the exact Q-loss, the Imitation Learning objective becomes

$$\mathcal{L}^{\mathrm{Q}}(\theta) \coloneqq \mathbb{E}_{x\sim\mathcal{D}}\left[Q(x, \pi(x;\theta))\right].$$

The gradient of $\mathcal{L}^{\mathrm{Q}}(\theta)$ is defined as

$$\nabla_\theta \mathcal{L}^{\mathrm{Q}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\Big[\nabla_\theta \pi(x;\theta)\ \nabla_u Q(x,u)|_{u=\pi(x;\theta)}\Big].$$

The exact Q-loss

With the exact Q-loss, the Imitation Learning objective becomes

$$\mathcal{L}^{\mathrm{Q}}(\theta) \coloneqq \mathbb{E}_{x\sim\mathcal{D}}\left[Q(x, \pi(x;\theta))\right].$$

The gradient of $\mathcal{L}^{\mathrm{Q}}(\theta)$ is defined as

$$\nabla_\theta \mathcal{L}^{\mathrm{Q}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\Big[\nabla_\theta \pi(x;\theta)\ \nabla_u Q(x,u)|_{u=\pi(x;\theta)}\Big].$$

Remark (Connection to actor-critic methods)

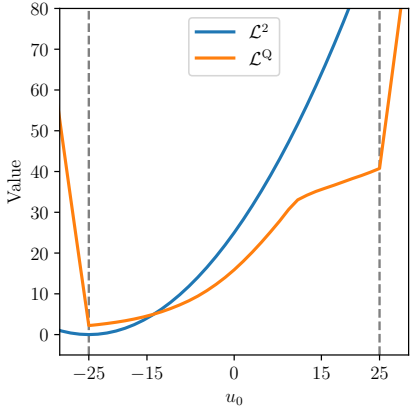
The Q-loss can been seen as a critic while the actor is the policy π. Following this perspective, the gradient of the Q-loss is directly related to deterministic policy gradients.

Exact Q-loss properties

- Computation cost compared to the $\mathcal{L}^2$ loss
  - $\nabla_\theta \mathcal{L}^2(\theta) = \mathbb{E}_{x\sim\mathcal{D}}[\nabla_\theta \pi(x;\theta)(\pi(x;\theta) - \pi^\star(x))]$
  - $\nabla_\theta \mathcal{L}^{\mathrm{Q}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\left[\nabla_\theta \pi(x;\theta)\ \nabla_u Q(x,u)|_{u=\pi(x;\theta)}\right]$
- OCP feasibility during training – need to evaluate the Q-loss for any $(x, \pi(x;\theta))$
- The Q-function for a nonlinear OCP might be nonconvex and nonlinear – tough to optimize

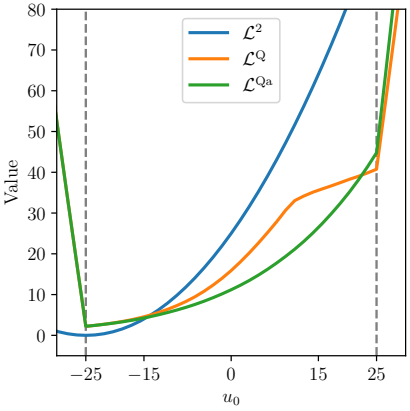
Exact Q-loss properties

- Computation cost compared to the $\mathcal{L}^2$ loss
  - $\nabla_\theta \mathcal{L}^2(\theta) = \mathbb{E}_{x\sim\mathcal{D}}[\nabla_\theta \pi(x;\theta)(\pi(x;\theta) - \pi^\star(x))]$
  - $\nabla_\theta \mathcal{L}^{\mathrm{Q}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\left[\nabla_\theta \pi(x;\theta)\ \nabla_u Q(x,u)|_{u=\pi(x;\theta)}\right]$
- OCP feasibility during training – need to evaluate the Q-loss for any $(x, \pi(x;\theta))$
- The Q-function for a nonlinear OCP might be nonconvex and nonlinear – tough to optimize

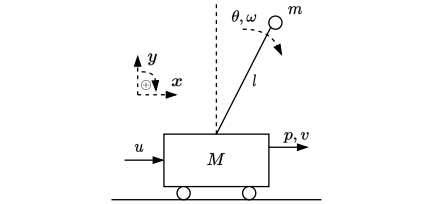


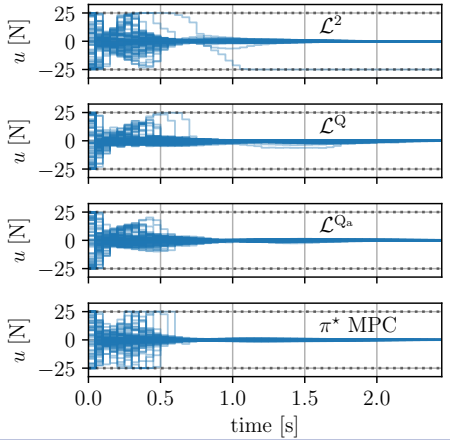
Exact Q-loss properties

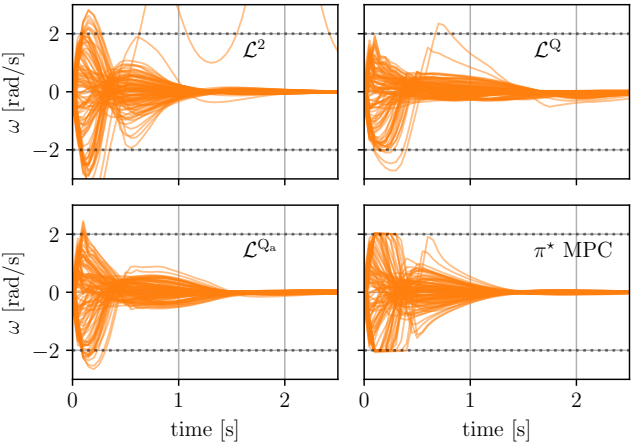
- Computation cost compared to the $\mathcal{L}^2$ loss
  - $\nabla_\theta \mathcal{L}^2(\theta) = \mathbb{E}_{x\sim\mathcal{D}}[\nabla_\theta \pi(x;\theta)(\pi(x;\theta) - \pi^\star(x))]$
  - $\nabla_\theta \mathcal{L}^{\mathrm{Q}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\left[\nabla_\theta \pi(x;\theta)\ \nabla_u Q(x,u)|_{u=\pi(x;\theta)}\right]$
- OCP feasibility during training – need to evaluate the Q-loss for any $(x, \pi(x;\theta))$
- The Q-function for a nonlinear OCP might be nonconvex and nonlinear – tough to optimize
  - **introduce a quadratic programming approximation!** (cf. [Ghezzi, Hoffmann, et al. 2023])



Comparing policies for cartpole swing-up







Other ideas to improve the loss function

Imitating a "generic" policy.

Sobolev Training [Lueken, Brandner, Lucia, 2023]

- includes in the training data the sensitivity $\frac{\partial u}{\partial x} \implies$ dataset of tuples t as $(x \in \mathbb{R}^{n_x}, u \in \mathbb{R}^{n_u}, \frac{\partial u}{\partial x} \in \mathbb{R}^{n_u \times n_x})$
- obtain predictions: $\hat{u} = \pi(\hat{x}; \theta)$, $\frac{\partial u}{\partial x}\big|_{\hat{x},\hat{u}} = \frac{\partial \pi(\cdot;\theta)}{\partial x}\big|_{\hat{x}}$
- Sobolev loss function: $\mathcal{L}^{\text{sob}}(x, u, \frac{\partial u}{\partial x}; \theta) = \mathbb{E}_{t\sim\mathcal{D}}\left[\|u - \pi(x;\theta)\|_2^2 + \alpha\|\frac{\partial u}{\partial x} - \frac{\partial \pi(\cdot;\theta)}{\partial x}\big|_x\|_2^2\right]$

Other ideas to improve the loss function

Imitating a MPC-based policy.

Augmenting the loss function using KKT information of the MPC problem (primal and dual feasibility) [Adhau, Naik, Skogestad, 2021]

- $\mathcal{L}^{\lambda}(x;\theta) = \mathbb{E}_{x\sim\mathcal{D}}[\|\pi_{\lambda}(x;\theta) - \pi^{*}_{\lambda}(x)\|_2^2 + \tau_1\|\log(-h(x,u))\|_2^2 + \tau_2\|-\log(-\mu)\|_2^2]$
- with $\pi_{\lambda}(x_i)$ predicting (u_i, x_i^{+}, μ_i), function h defines the stage-wise inequality constraints of the MPC problem, μ is the associated multiplier

PlanNetX [Hoffmann et al., 2024]

- Considering the full MPC trajectory as training data, $(\mathbf{x}^*, \mathbf{u}^*) = (x_0^*, \dots, x_N^*, u_0^*, \dots, u_{N-1}^*)$
- PlanNetX loss: $\mathcal{L}^{\mathsf{P}} = \mathbb{E}_{x_0\sim\mathcal{D}}\left[\frac{1}{N}\sum_{k=0}^{N}\gamma^k\|\hat{x}_k(\mathbf{u};x_0,\theta) - x_k^*\|_W^2\right]$, with $\hat{x}_{k+1} = f(\hat{x}_k, \pi(\hat{x}_k;\theta))$, $\hat{x}_0 = x_0, k = 0,\dots,N-1$, $W \succeq 0$.
- Possibly combined with a loss involving the controls

Outline of the lecture

Implicit and explicit model predictive control

Loss functions for imitation learning

Improving performance of learned controllers

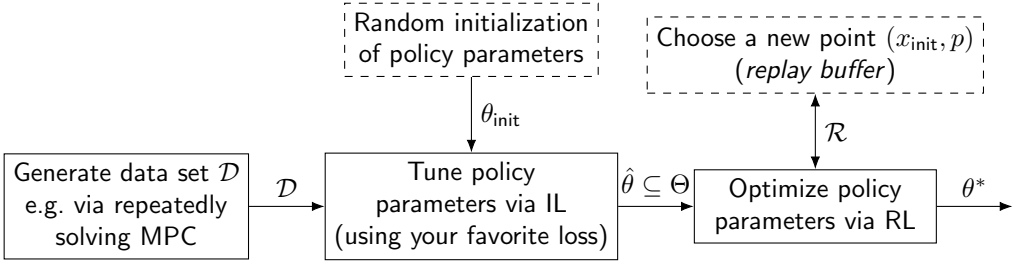
Data collection – how to sample?

Verification of learned controllers

Reinforcing a parametric control law

Assuming that an **accurate simulator** is available, it is possible to adapt the IL controller

- IL imitates MPC which might have a simplified model compared to reality
- In deployment we discover effects we did not considered: estimation errors, ...
- The environment is changed compared to when we trained the IL controller
- Blend a new control objective in the existing IL controller



Activation patterns in a neural network I

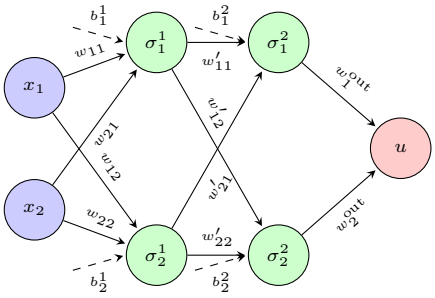
In a ReLU network each layer: $f_l(\xi_{l-1}) = W_l \xi_{l-1} + b_l$, with ReLU $\sigma_l = \max(0, f_l)$, and $\xi_0 = x$.
The network is the composition $f_{L+1} \circ \sigma_L \circ f_L \circ \cdots \circ \sigma_1 \circ f_1(x)$

Activation patterns. Assign a binary value to every neuron in the each hidden layer.

We assume NN with fixed width n_w, activation patter $\Gamma = \{\gamma_1, \ldots, \gamma_L\}$, $\gamma_i \in \{0,1\}^{n_w}$
$G(x) \coloneqq \{\beta \circ f_l(\xi_{l-1}) \in [0,1]^L \mid \xi_0 = x\}$

β is a Heaviside-step function:

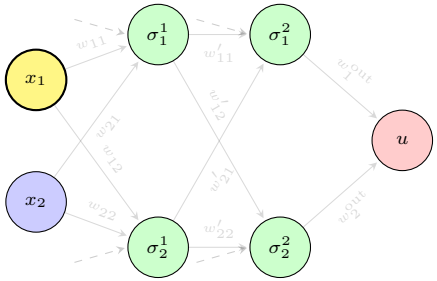
$$\beta \circ f_l(W_{l-1})^{(i)} = \begin{cases} 1 \text{ if } W_l^{(i)} \xi_{l-1} + b_l^{(i)} \geq 0, \\ 0 \text{ otherwise} \end{cases}$$



Activation patterns in a neural network II

Given an input x_i we can obtain Γ_i
Γ_i describes a polytopic region in the state space $\mathcal{S}_{\Gamma_i} = \{x \in \mathbb{R}^{n_x} \mid \Gamma_i = G(x)\}$
We can write in H-representation as $\mathcal{S}_i = \{x \in \mathbb{R}^{n_x} \mid F_i x \leq g_i\}$

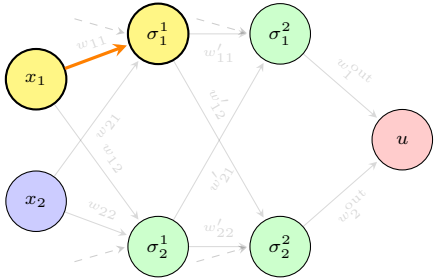
Using activation pattern we can describe the network as
$f_{L+1} \circ \gamma_L \odot f_L \circ \cdots \circ \gamma_1 \odot f_1(x) = W_{\Gamma_i} x + b_{\Gamma_i}$



Activation patterns in a neural network II

Given an input x_i we can obtain Γ_i
Γ_i describes a polytopic region in the state space $\mathcal{S}_{\Gamma_i} = \{x \in \mathbb{R}^{n_x} \mid \Gamma_i = G(x)\}$
We can write in H-representation as $\mathcal{S}_i = \{x \in \mathbb{R}^{n_x} \mid F_i x \leq g_i\}$

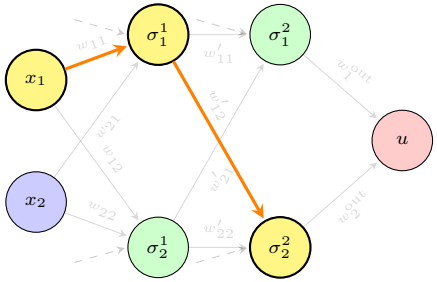
Using activation pattern we can describe the network as
$f_{L+1} \circ \gamma_L \odot f_L \circ \cdots \circ \gamma_1 \odot f_1(x) = W_{\Gamma_i} x + b_{\Gamma_i}$



Activation patterns in a neural network II

Given an input x_i we can obtain Γ_i
Γ_i describes a polytopic region in the state space $\mathcal{S}_{\Gamma_i} = \{x \in \mathbb{R}^{n_x} \mid \Gamma_i = G(x)\}$
We can write in H-representation as $\mathcal{S}_i = \{x \in \mathbb{R}^{n_x} \mid F_i x \leq g_i\}$

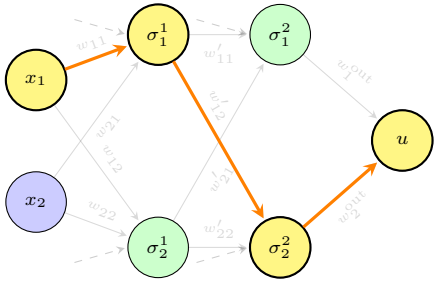
Using activation pattern we can describe the network as
$f_{L+1} \circ \gamma_L \odot f_L \circ \cdots \circ \gamma_1 \odot f_1(x) = W_{\Gamma_i} x + b_{\Gamma_i}$



Activation patterns in a neural network II

Given an input x_i we can obtain Γ_i
Γ_i describes a polytopic region in the state space $\mathcal{S}_{\Gamma_i} = \{x \in \mathbb{R}^{n_x} \mid \Gamma_i = G(x)\}$
We can write in H-representation as $\mathcal{S}_i = \{x \in \mathbb{R}^{n_x} \mid F_i x \leq g_i\}$

Using activation pattern we can describe the network as
$f_{L+1} \circ \gamma_L \odot f_L \circ \cdots \circ \gamma_1 \odot f_1(x) = W_{\Gamma_i} x + b_{\Gamma_i}$



Feedback in the neighborhood of an equilibrium [Karg and Lucia, 2020]

Find Γ_{eq} at the equilibrium x_{eq} as $G(x_{\mathrm{eq}}) \implies \mathcal{S}_{\mathrm{eq}} = \{x \in \mathbb{R}^{n_x} \mid F_{\mathrm{eq}} x \leq g_{\mathrm{eq}}\}$
In a neighborhood of x_{eq}, we want to have

- the LQR feedback law $\pi^{\mathrm{LQR}}(x) = -Kx$
- the closed-loop system $x^+ = f(x, \pi^{\mathrm{LQR}}(x))$ asymptotically stable $\implies$ $\|\mathrm{eig}(A - BK)\|_\infty < 1$

Feedback in the neighborhood of an equilibrium [Karg and Lucia, 2020]

Find Γ_{eq} at the equilibrium x_{eq} as $G(x_{\text{eq}}) \implies \mathcal{S}_{\text{eq}} = \{x \in \mathbb{R}^{n_x} \mid F_{\text{eq}} x \leq g_{\text{eq}}\}$
In a neighborhood of x_{eq}, we want to have

- the LQR feedback law $\pi^{\text{LQR}}(x) = -Kx$
- the closed-loop system $x^+ = f(x, \pi^{\text{LQR}}(x))$ asymptotically stable $\implies$ $\|\text{eig}(A - BK)\|_\infty < 1$

The network feedback for Γ_{eq} is $\pi(x;\theta) = W_{L+1}(W_{\Gamma_{\text{eq}}} x + b_{\Gamma_{eq}}) + b_{L+1}$

Set the affine term to zero: $W_{L+1} b_{\Gamma_{eq}} + b_{L+1} = 0$
Set the stability condition: $\|\text{eig}(A - B W_{L+1} W_{\Gamma_{\text{eq}}})\|_\infty < 1$

Feedback in the neighborhood of an equilibrium [Karg and Lucia, 2020]

Find Γ_{eq} at the equilibrium x_{eq} as $G(x_{\text{eq}}) \implies \mathcal{S}_{\text{eq}} = \{x \in \mathbb{R}^{n_x} \mid F_{\text{eq}} x \leq g_{\text{eq}}\}$
In a neighborhood of x_{eq}, we want to have

- the LQR feedback law $\pi^{\text{LQR}}(x) = -Kx$
- the closed-loop system $x^+ = f(x, \pi^{\text{LQR}}(x))$ asymptotically stable $\implies$ $\|\text{eig}(A - BK)\|_\infty < 1$

The network feedback for Γ_{eq} is $\pi(x;\theta) = W_{L+1}(W_{\Gamma_{\text{eq}}} x + b_{\Gamma_{eq}}) + b_{L+1}$

Set the affine term to zero: $W_{L+1} b_{\Gamma_{eq}} + b_{L+1} = 0$
Set the stability condition: $\|\text{eig}(A - B W_{L+1} W_{\Gamma_{\text{eq}}})\|_\infty < 1$

Tune the weights of the last layer $L+1$ via

$$\begin{aligned} \min_{\hat{W}_{L+1}, \hat{b}_{L+1}} \quad & \|\hat{W}_{L+1} - W_{L+1}\|_2^2 + \|\hat{b}_{L+1} - b_{L+1}\|_2^2 \\ \text{s.t.} \quad & \hat{W}_{L+1} W_{\Gamma_{\text{eq}}} = K^{\text{LQR}}, \\ & \hat{W}_{L+1} b_{\Gamma_{eq}} + \hat{b}_{L+1} = 0. \end{aligned}$$

Outline of the lecture

Implicit and explicit model predictive control

Loss functions for imitation learning

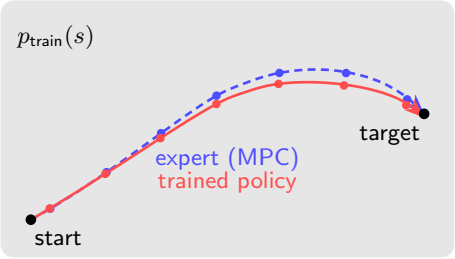
Improving performance of learned controllers

Data collection – how to sample?

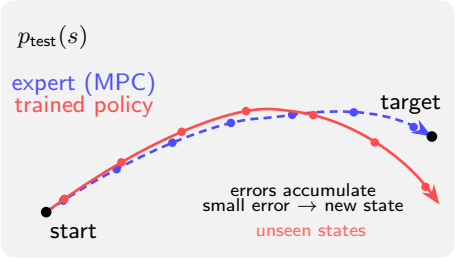
Verification of learned controllers

Covariate shift problem

Training (MPC + imitation)



Deployment (rollout)



Imitation Learning fails if the policy never learns how to act in its own state distribution.

DAgger (Dataset Aggregation) [Ross et al. 2011]

Require: Expert policy π^*, learner policy class Π

1: Initialize $\hat{\pi}_1$ to any policy in Π
2: Initialize dataset $\mathcal{D} \leftarrow \emptyset$
3: **for** $i = 1$ to N **do**
4: Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
5: Rollout T-step trajectories using π_i
6: Get dataset $\mathcal{D} = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by the expert
7: **Aggregate datasets** $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
8: Train policy $\hat{\pi}_{i+1}$ on $\mathcal{D}$
9: **end for**
10: **Return:** best $\hat{\pi}_i$ on validation

in general $\beta_1 = 1$ and $\beta_i = p^{i-1}$ (usage of the expert decays exponentially)

Idea:

- the learner policy interacts with the environment
- for each state visited by learner, we collect the correct action from the expert

Data augmentation

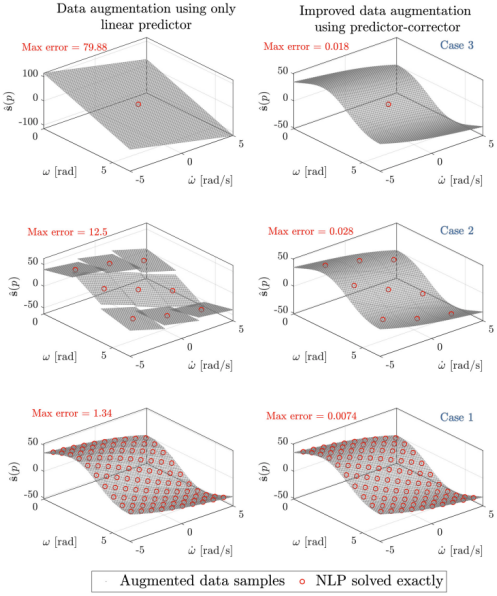
In case we imitate from nonlinear MPC collecting data might be expensive.

- Always have to solve NLPs!

How can we exploit at maximum the data at hand?
Using the NLP sensitivities! (*Rembember lecture 4!*)

$$z(p) \approx z(p^*) + \frac{\mathrm{d}z(p^*)}{\mathrm{d}p}(p - p^*)$$

[Krishnamoorthy, 2021]



Outline of the lecture

Implicit and explicit model predictive control

Loss functions for imitation learning

Improving performance of learned controllers

Data collection – how to sample?

Verification of learned controllers

Guarantees for learned controllers

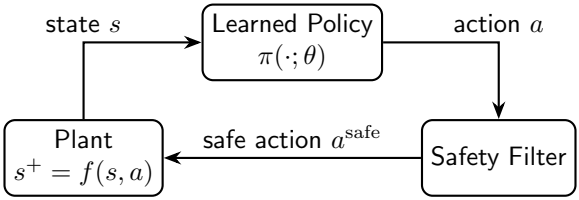
For systems with a linear dynamics and quadratic cost ("LQR case")

- projection-based methods [Chen et al., 2018]
- a-priori verification via output-range analysis [Karg and Lucia, 2020]
- worst-case approximation errors and Lipschitz constants [Fabiani and Goulart, 2023], [Schwan, Jones, Kuhn, 2023]

For nonlinear systems:

- *Safety filter* (projection-based) [Wabersich et al., 2021, ...]
- Probabilistic verification [Tempo et al., 1997, Alamo et al, 2015, Karg and Lucia, 2021]

Safety filter in a nutshell



$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{u}} \quad & \|\pi(x_0;\theta) - u_0\|_2^2 \\
\text{s.t.} \quad & x_0 - \bar{x_0} = 0, \\
& x_{k+1} - f(x_k, u_k) = 0,\ k \in \mathbb{Z}_{[0,N-1]}, \\
& g(x_k, u_k) \leq 0,\ k \in \mathbb{Z}_{[0,N-1]}
\end{aligned}
$$