



System Intelligence Laboratory

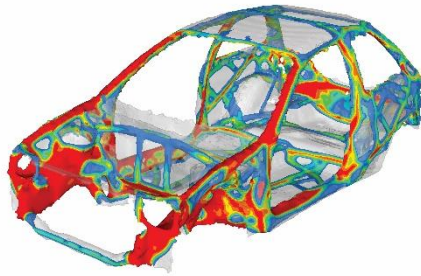
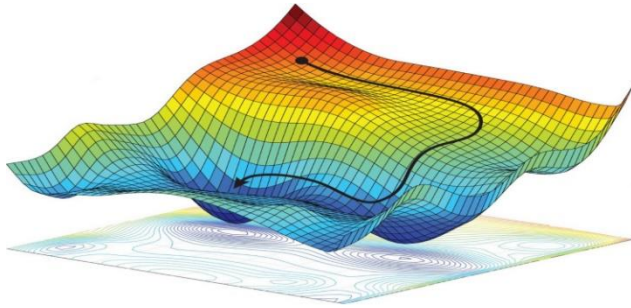


WONIK-KAIST FTC 기술 전수 세미나: Decision-Making with Data-Driven Model

*Dept. of Industrial & Systems engineering, KAIST
Chihyeon Song, Haewon Jung, Jinkyoo Park*

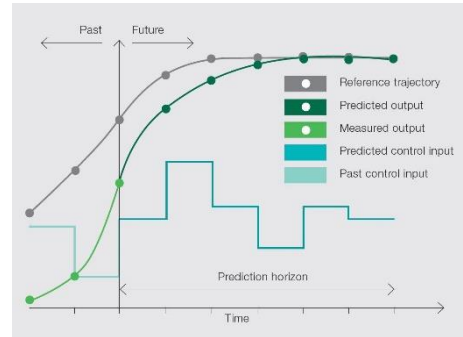
Decision-Making

- Decision-making: What we really want to do

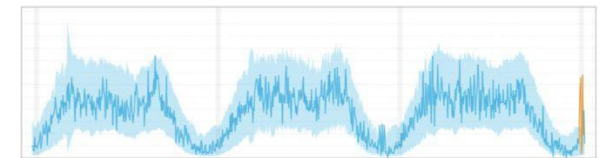
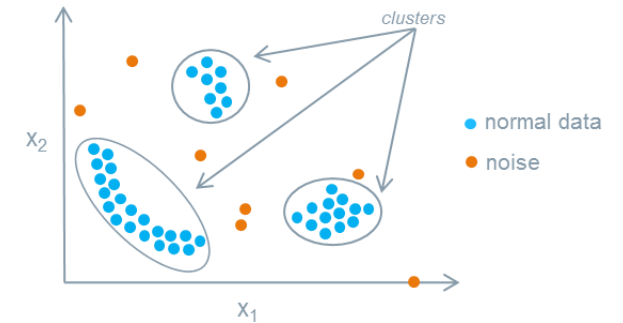


Model-based optimizations

$$\begin{aligned} \min_{u_1, \dots, u_T} & \sum_{t=1}^T c(x_t, u_t) \\ \text{s.t.} & \quad x_{t+1} = f(x_t, u_t) \end{aligned}$$

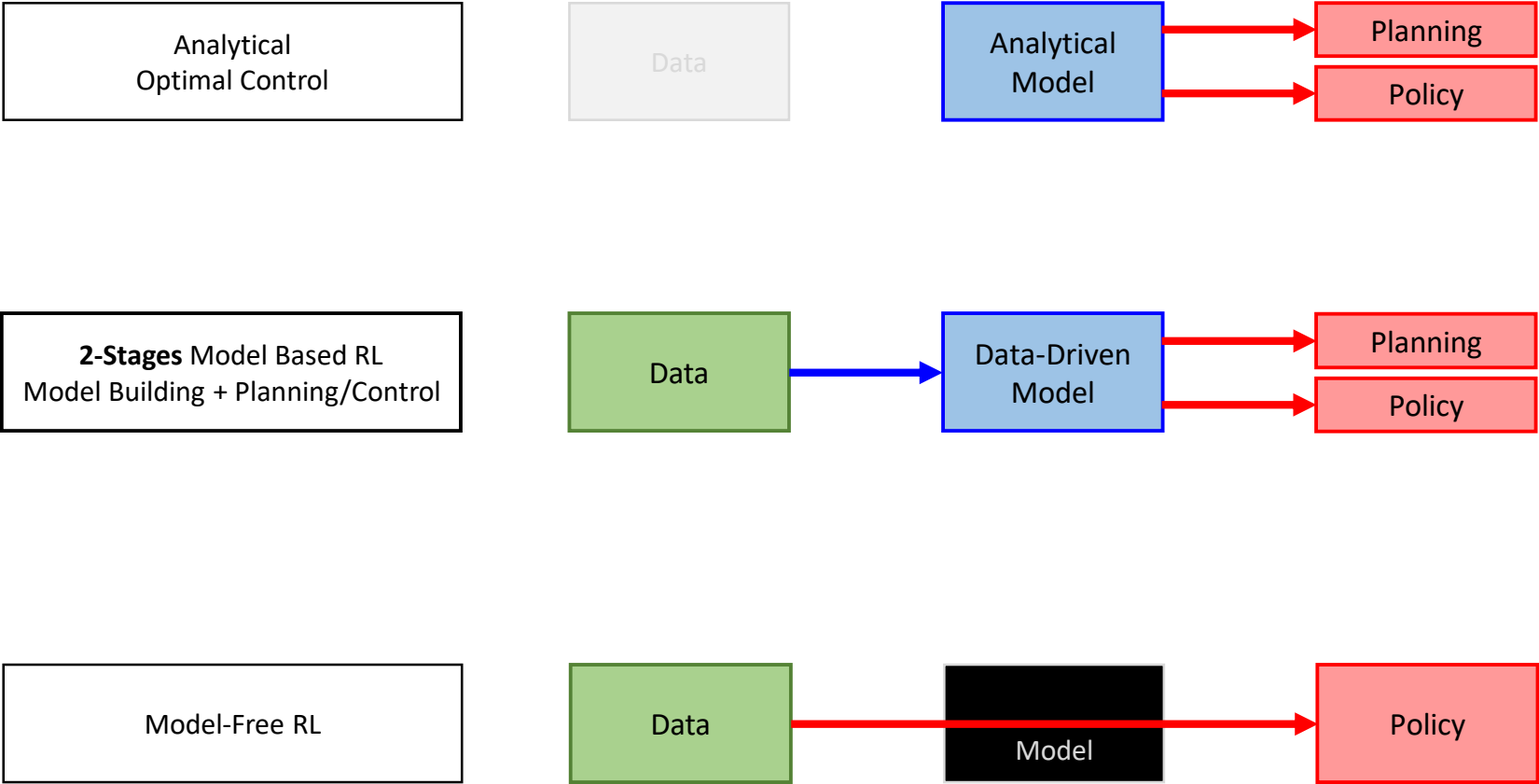


Optimal controls



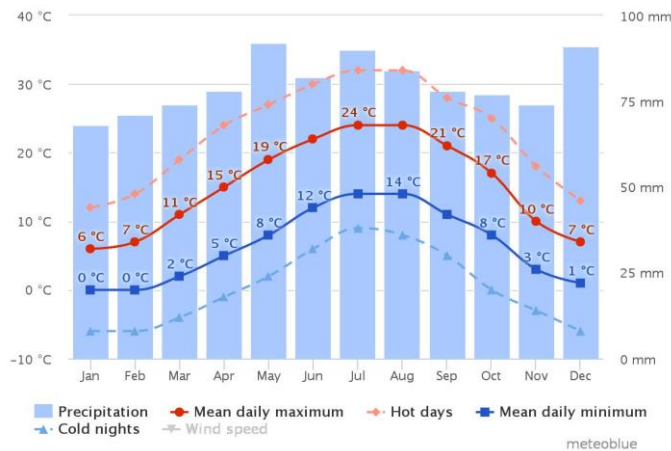
Anomaly detections

Ways to Make a Decision

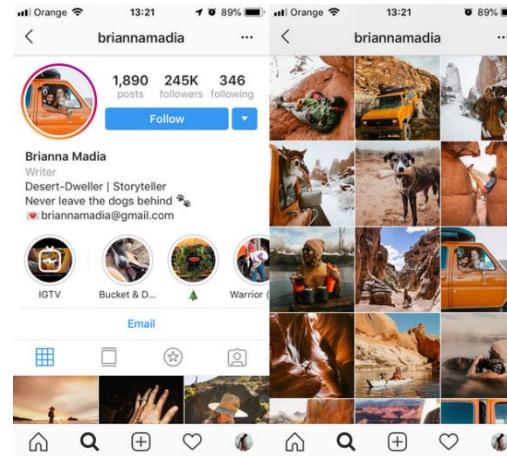


Decision-Making with Data-Driven Models

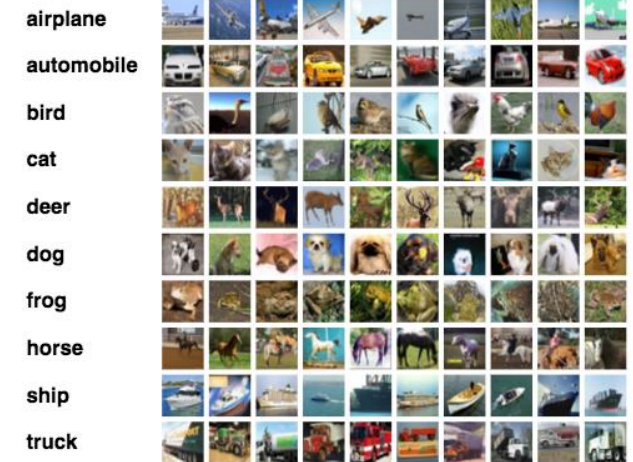
- Data: discrete value that contains information



Weather



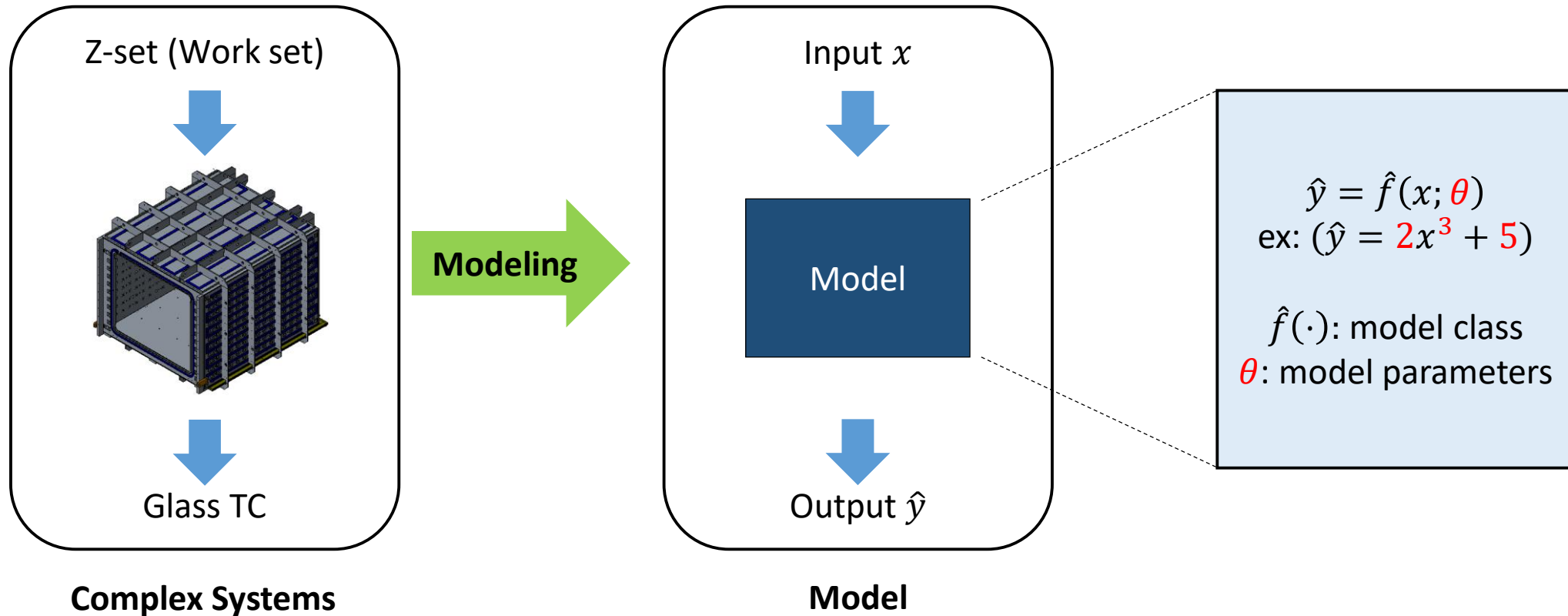
SNS



Image

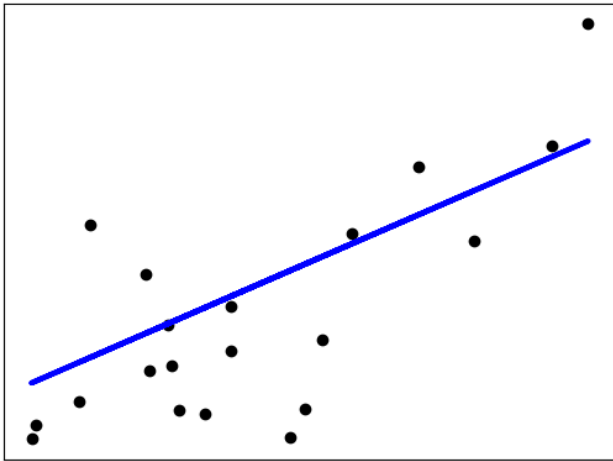
Decision-Making with Data-Driven Models

- Model: Theoretical representation of a system
 - Will be used in decision-making instead of the actual system

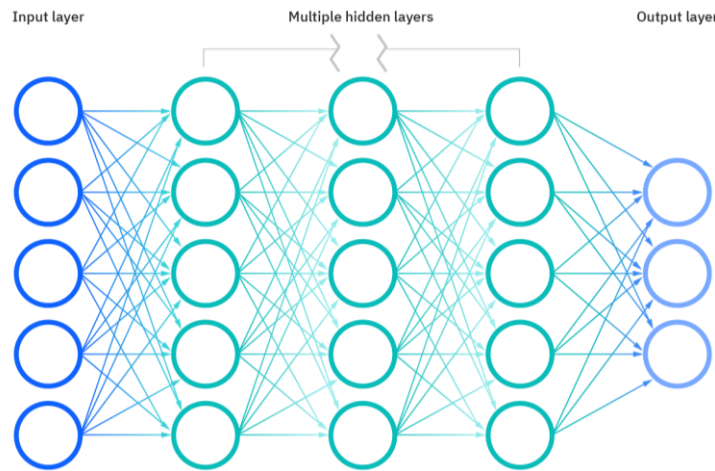


Decision-Making with Data-Driven Models

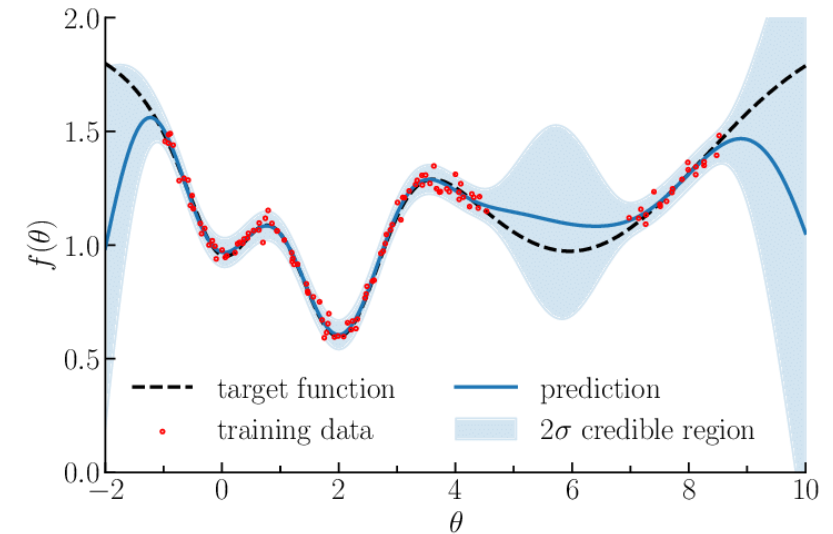
- Data-driven model: Model parameters are *learned* from the data
 - Collect the information from the data in terms of modeling



Linear model



Deep neural networks



Gaussian process

Why Data-Driven Decision-Making?

- Complexity of modern systems
 - Data-driven approach doesn't require any background knowledge of the system

$$\frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(k \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left(k \frac{\partial T}{\partial z} \right) + q_v = \rho c_p \frac{\partial T}{\partial t}$$

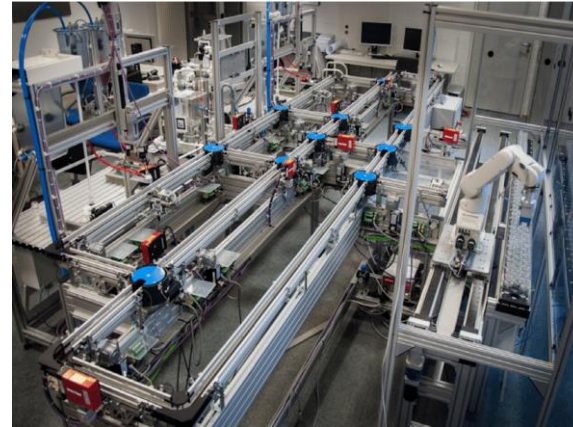
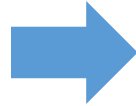
where

k is the materials conductivity [$\text{W.m}^{-1}.\text{K}^{-1}$]

q_v is the rate at which energy is generated per unit volume of the medium [W.m^{-3}]

ρ is the density [kg.m^{-3}]

c_p is the specific heat capacity [$\text{J.kg}^{-1}.\text{K}^{-1}$]

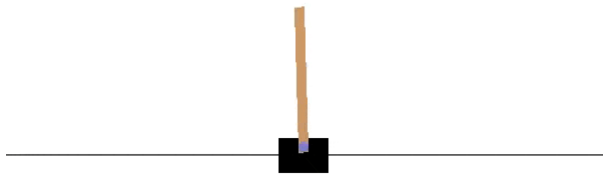


Scientific Knowledge

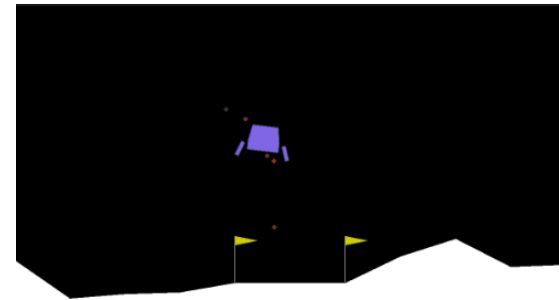
Modern Complex Systems

Why Data-Driven Model?

- In the real-world, we often have limited amount of data
- With an appropriate model, more data-efficient than model-free approach
 - More practical approach to the real-world systems

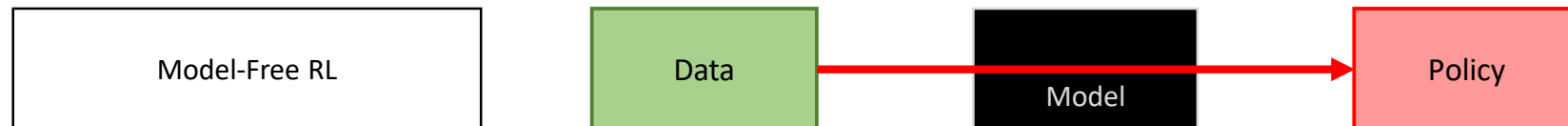
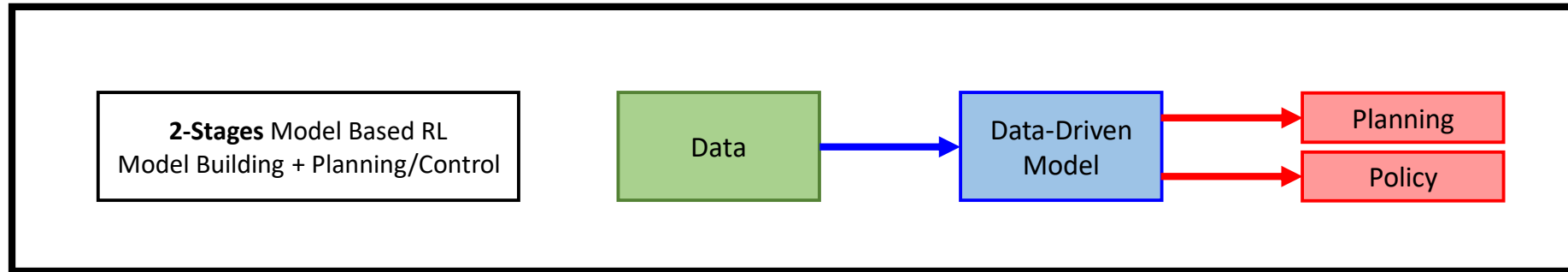
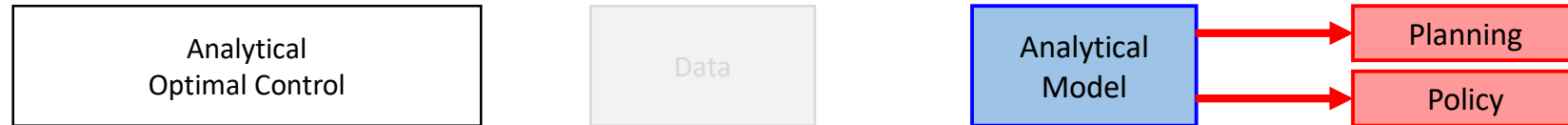


CartPole



LunarLander

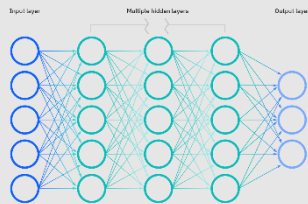
How to Make a Decision



Two Big Questions

Q1. How we build data-driven model?

- Formulate System
- Model Selection
- Data Preprocessing
- Learn Model



	A	B	C	D	E	F
1	12/9/2018 11:00	\$1,400	\$1,400	\$1,200	\$1,400	0
2	12/9/2018 10:30	\$1,460	\$1,400	\$1,300	\$1,440	-1
3	12/9/2018 10:00	\$4,130	\$4,440	\$3,210	\$3,450	-2
4	12/9/2018 9:30	\$1,630	\$4,230	\$3,600	\$4,120	-3
5	12/9/2018 9:00	\$1,490	\$3,880	\$1,400	\$1,630	-4
6	12/9/2018 8:30	\$1,440	\$3,770	\$1,380	\$1,490	-5
7	12/9/2018 8:00	\$1,270	\$1,520	\$1,170	\$1,430	-6
8	12/9/2018 7:30	\$1,230	\$1,380	\$1,380	\$1,270	-7
9	12/9/2018 7:00	\$1,200	\$1,440	\$1,010	\$1,200	-8
10	12/9/2018 6:30	\$1,050	\$1,350	\$2,980	\$1,280	-9
11	12/9/2018 6:00	\$1,080	\$1,250	\$2,940	\$1,040	-10
12	12/9/2018 5:30	\$1,180	\$1,210	\$2,960	\$1,080	-11
13	12/9/2018 5:00	\$1,300	\$1,380	\$1,140	\$1,180	-12
14	12/9/2018 4:30	\$1,090	\$1,420	\$1,050	\$1,290	-13

Q2. How we make a decision with model?

- Define Decision-Making
- Formulate Optimization Problem
- Solve Control Optimization
- Validate Control Performance

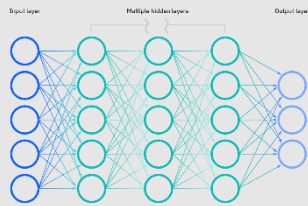
$$\begin{aligned} \min_x \quad & f(x) \\ \text{s. t.} \quad & g_i(x) \leq 0 \quad \forall i = 1, \dots, I \\ & h_j(x) = 0 \quad \forall j = 1, \dots, J \end{aligned}$$

```
%% Optimization problem
% Objective function
f = @(x) 0.5*x(1)^2 + 0.5*x(2)^2;
% Constraints
g = @(x) [x(1)-1; x(2)-1; x(1)+x(2)-1];
h = @(x) [x(1)^2 + x(2)^2 - 1];
% Initial guess
x0 = [0; 0];
% Optimization options
options = optimoptions('fmincon','Display','none','Algorithm','quasi-newton');
% Solve the optimization problem
[x_opt, f_opt] = fmincon(f,x0,[],[],[],[],[],[],[],options);
% Display the results
disp('Optimal solution:');
disp([x_opt, f_opt]);
```

Two Big Questions

Q1. How we build data-driven model?

- Formulate System
- Model Selection
- Data Preprocessing
- Learn Model



	A	B	C	D	E	F
1	12/9/2018 11:00	\$1,400	\$1,400	\$1,200	\$1,400	0
2	12/9/2018 10:30	\$1,460	\$1,600	\$1,200	\$1,460	-1
3	12/9/2018 10:00	\$4,130	\$4,440	\$3,210	\$3,450	-2
4	12/9/2018 9:30	\$1,630	\$4,230	\$1,600	\$4,120	-3
5	12/9/2018 9:00	\$1,490	\$1,880	\$1,400	\$1,630	-4
6	12/9/2018 8:30	\$1,440	\$1,770	\$1,380	\$1,490	-5
7	12/9/2018 8:00	\$1,270	\$1,520	\$1,170	\$1,430	-6
8	12/9/2018 7:30	\$1,230	\$1,380	\$1,380	\$1,270	-7
9	12/9/2018 7:00	\$1,200	\$1,440	\$1,010	\$1,200	-8
10	12/9/2018 6:30	\$1,050	\$1,350	\$2,980	\$1,280	-9
11	12/9/2018 6:00	\$1,080	\$1,250	\$2,940	\$1,040	-10
12	12/9/2018 5:30	\$1,180	\$1,210	\$2,960	\$1,080	-11
13	12/9/2018 5:00	\$1,300	\$1,380	\$1,140	\$1,180	-12
14	12/9/2018 4:30	\$1,050	\$1,420	\$1,050	\$1,290	-13

Day 1

Q2. How we make a decision with model?

- Define Decision-Making
- Formulate Optimization Problem
- Solve Control Optimization
- Validate Control Performance

$$\begin{aligned} \min_x & f(x) \\ \text{s. t. } & g_i(x) \leq 0 \quad \forall i = 1, \dots, I \\ & h_j(x) = 0 \quad \forall j = 1, \dots, J \end{aligned}$$

```
def objective(x):  
    return x[0]  
  
def constraint1(x):  
    return x[1] - 10  
  
def constraint2(x):  
    return x[2] - 5  
  
def constraint3(x):  
    return x[3] - 2  
  
def constraint4(x):  
    return x[4] - 1  
  
def constraint5(x):  
    return x[5] - 0.5  
  
def constraint6(x):  
    return x[6] - 0.2  
  
def constraint7(x):  
    return x[7] - 0.1  
  
def constraint8(x):  
    return x[8] - 0.05  
  
def constraint9(x):  
    return x[9] - 0.02  
  
def constraint10(x):  
    return x[10] - 0.01  
  
def constraint11(x):  
    return x[11] - 0.005  
  
def constraint12(x):  
    return x[12] - 0.002  
  
def constraint13(x):  
    return x[13] - 0.001  
  
def constraint14(x):  
    return x[14] - 0.0005  
  
def constraint15(x):  
    return x[15] - 0.0002  
  
def constraint16(x):  
    return x[16] - 0.0001  
  
def constraint17(x):  
    return x[17] - 0.00005  
  
def constraint18(x):  
    return x[18] - 0.00002  
  
def constraint19(x):  
    return x[19] - 0.00001  
  
def constraint20(x):  
    return x[20] - 0.000005  
  
def constraint21(x):  
    return x[21] - 0.000002  
  
def constraint22(x):  
    return x[22] - 0.000001  
  
def constraint23(x):  
    return x[23] - 0.0000005  
  
def constraint24(x):  
    return x[24] - 0.0000002  
  
def constraint25(x):  
    return x[25] - 0.0000001  
  
def constraint26(x):  
    return x[26] - 0.00000005  
  
def constraint27(x):  
    return x[27] - 0.00000002  
  
def constraint28(x):  
    return x[28] - 0.00000001  
  
def constraint29(x):  
    return x[29] - 0.000000005  
  
def constraint30(x):  
    return x[30] - 0.000000002  
  
def constraint31(x):  
    return x[31] - 0.000000001  
  
def constraint32(x):  
    return x[32] - 0.0000000005  
  
def constraint33(x):  
    return x[33] - 0.0000000002  
  
def constraint34(x):  
    return x[34] - 0.0000000001  
  
def constraint35(x):  
    return x[35] - 0.00000000005  
  
def constraint36(x):  
    return x[36] - 0.00000000002  
  
def constraint37(x):  
    return x[37] - 0.00000000001  
  
def constraint38(x):  
    return x[38] - 0.000000000005  
  
def constraint39(x):  
    return x[39] - 0.000000000002  
  
def constraint40(x):  
    return x[40] - 0.000000000001  
  
def constraint41(x):  
    return x[41] - 0.0000000000005  
  
def constraint42(x):  
    return x[42] - 0.0000000000002  
  
def constraint43(x):  
    return x[43] - 0.0000000000001  
  
def constraint44(x):  
    return x[44] - 0.00000000000005  
  
def constraint45(x):  
    return x[45] - 0.00000000000002  
  
def constraint46(x):  
    return x[46] - 0.00000000000001  
  
def constraint47(x):  
    return x[47] - 0.000000000000005  
  
def constraint48(x):  
    return x[48] - 0.000000000000002  
  
def constraint49(x):  
    return x[49] - 0.000000000000001  
  
def constraint50(x):  
    return x[50] - 0.0000000000000005  
  
def constraint51(x):  
    return x[51] - 0.0000000000000002  
  
def constraint52(x):  
    return x[52] - 0.0000000000000001  
  
def constraint53(x):  
    return x[53] - 0.00000000000000005  
  
def constraint54(x):  
    return x[54] - 0.00000000000000002  
  
def constraint55(x):  
    return x[55] - 0.00000000000000001  
  
def constraint56(x):  
    return x[56] - 0.000000000000000005  
  
def constraint57(x):  
    return x[57] - 0.000000000000000002  
  
def constraint58(x):  
    return x[58] - 0.000000000000000001  
  
def constraint59(x):  
    return x[59] - 0.0000000000000000005  
  
def constraint60(x):  
    return x[60] - 0.0000000000000000002  
  
def constraint61(x):  
    return x[61] - 0.0000000000000000001  
  
def constraint62(x):  
    return x[62] - 0.00000000000000000005  
  
def constraint63(x):  
    return x[63] - 0.00000000000000000002  
  
def constraint64(x):  
    return x[64] - 0.00000000000000000001  
  
def constraint65(x):  
    return x[65] - 0.000000000000000000005  
  
def constraint66(x):  
    return x[66] - 0.000000000000000000002  
  
def constraint67(x):  
    return x[67] - 0.000000000000000000001  
  
def constraint68(x):  
    return x[68] - 0.0000000000000000000005  
  
def constraint69(x):  
    return x[69] - 0.0000000000000000000002  
  
def constraint70(x):  
    return x[70] - 0.0000000000000000000001  
  
def constraint71(x):  
    return x[71] - 0.00000000000000000000005  
  
def constraint72(x):  
    return x[72] - 0.00000000000000000000002  
  
def constraint73(x):  
    return x[73] - 0.00000000000000000000001  
  
def constraint74(x):  
    return x[74] - 0.000000000000000000000005  
  
def constraint75(x):  
    return x[75] - 0.000000000000000000000002  
  
def constraint76(x):  
    return x[76] - 0.000000000000000000000001  
  
def constraint77(x):  
    return x[77] - 0.0000000000000000000000005  
  
def constraint78(x):  
    return x[78] - 0.0000000000000000000000002  
  
def constraint79(x):  
    return x[79] - 0.0000000000000000000000001  
  
def constraint80(x):  
    return x[80] - 0.00000000000000000000000005  
  
def constraint81(x):  
    return x[81] - 0.00000000000000000000000002  
  
def constraint82(x):  
    return x[82] - 0.00000000000000000000000001  
  
def constraint83(x):  
    return x[83] - 0.000000000000000000000000005  
  
def constraint84(x):  
    return x[84] - 0.000000000000000000000000002  
  
def constraint85(x):  
    return x[85] - 0.000000000000000000000000001  
  
def constraint86(x):  
    return x[86] - 0.0000000000000000000000000005  
  
def constraint87(x):  
    return x[87] - 0.0000000000000000000000000002  
  
def constraint88(x):  
    return x[88] - 0.0000000000000000000000000001  
  
def constraint89(x):  
    return x[89] - 0.00000000000000000000000000005  
  
def constraint90(x):  
    return x[90] - 0.00000000000000000000000000002  
  
def constraint91(x):  
    return x[91] - 0.00000000000000000000000000001  
  
def constraint92(x):  
    return x[92] - 0.000000000000000000000000000005  
  
def constraint93(x):  
    return x[93] - 0.000000000000000000000000000002  
  
def constraint94(x):  
    return x[94] - 0.000000000000000000000000000001  
  
def constraint95(x):  
    return x[95] - 0.0000000000000000000000000000005  
  
def constraint96(x):  
    return x[96] - 0.0000000000000000000000000000002  
  
def constraint97(x):  
    return x[97] - 0.0000000000000000000000000000001  
  
def constraint98(x):  
    return x[98] - 0.00000000000000000000000000000005  
  
def constraint99(x):  
    return x[99] - 0.00000000000000000000000000000002  
  
def constraint100(x):  
    return x[100] - 0.00000000000000000000000000000001
```

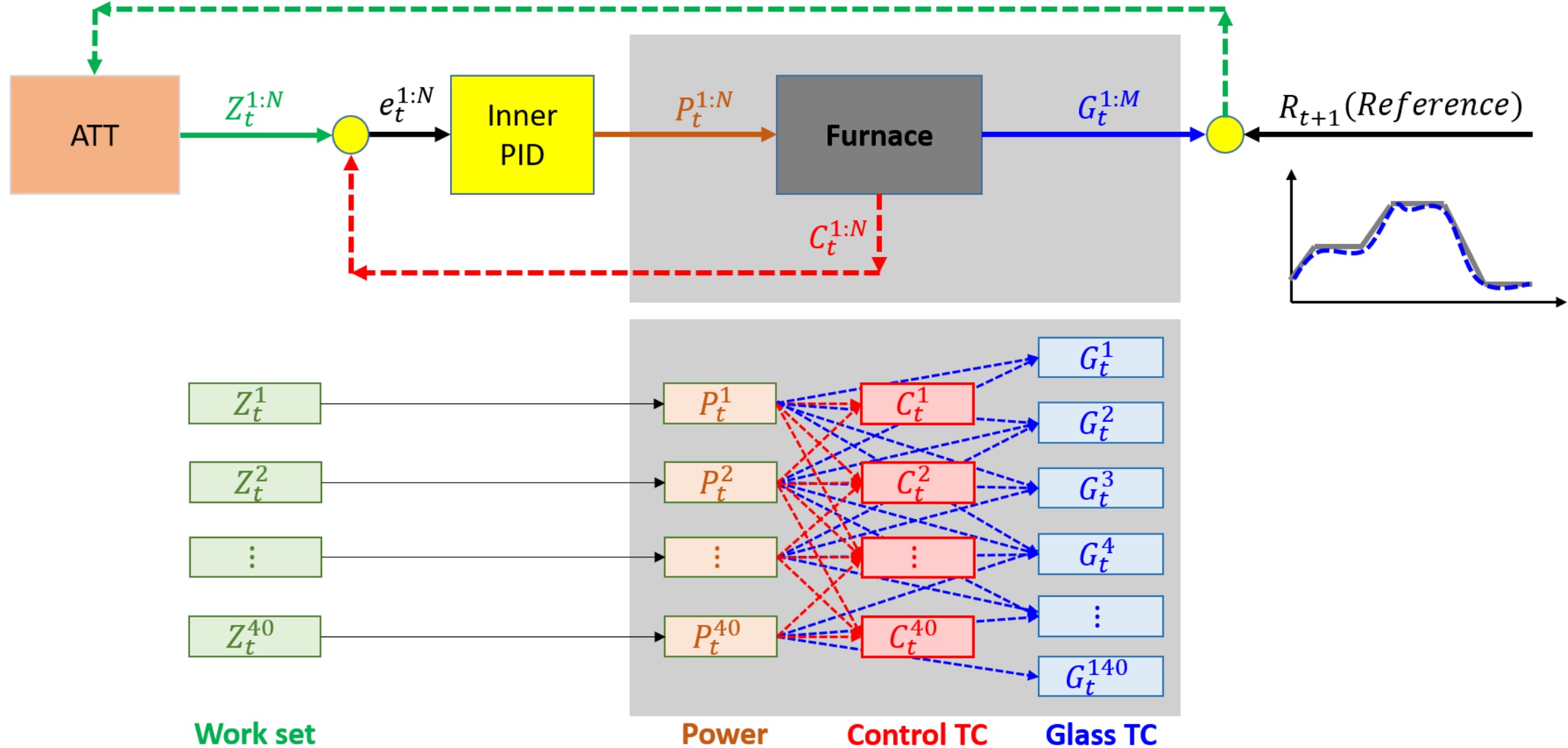
Day 2

Q1. How We Build Data-Driven Model?

Q1. How We Build Data-Driven Model?

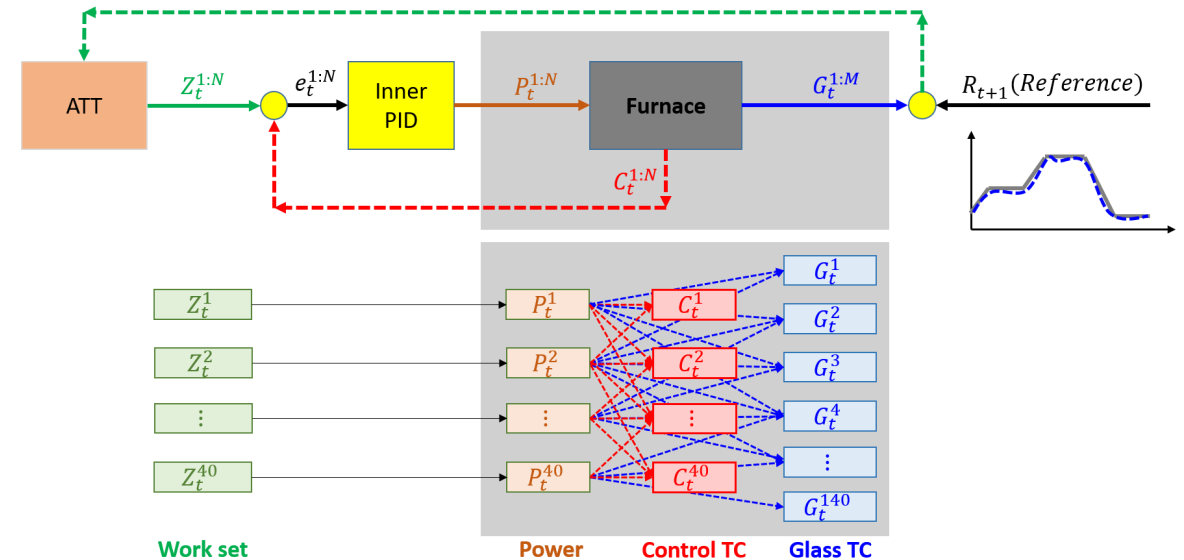
- Step 1. Formulate into a mathematical system
- Step 2. Select the model class
- Step 3. Prepare the data
- Step 4. Learn (Train) the model

Step 1. Formulate Into Mathematical System



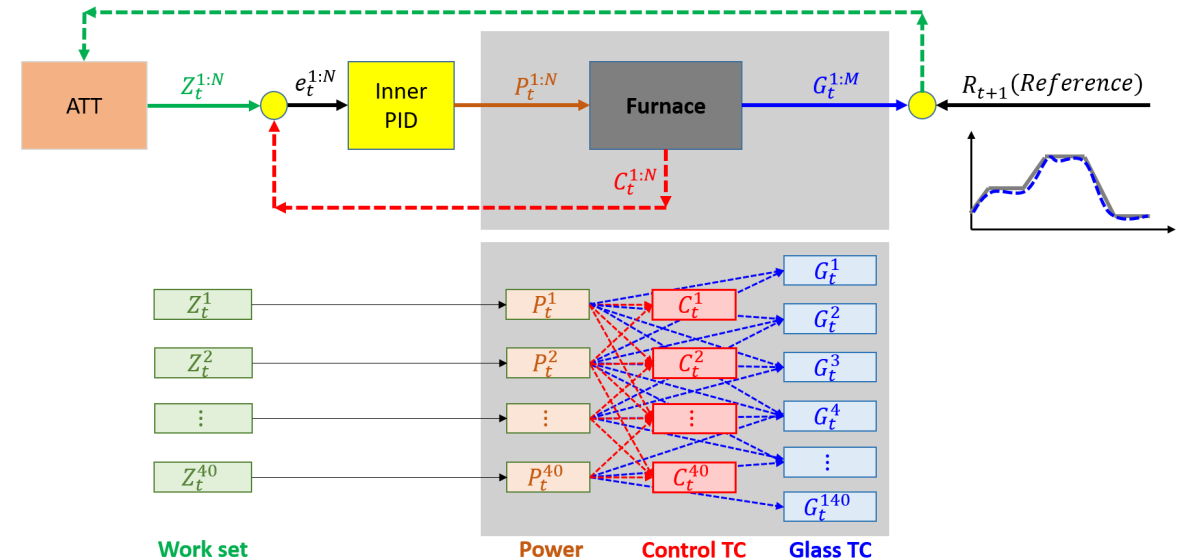
Step 1. Formulate Into Mathematical System

- Control Input u_t : What we can control to the system
 - Workset
- Observation o_t : What we can observe from the system
 - Power, Control TC, (Glass TC)
- State x_t : What we care in the system
 - Glass TC
 - NOT always observable



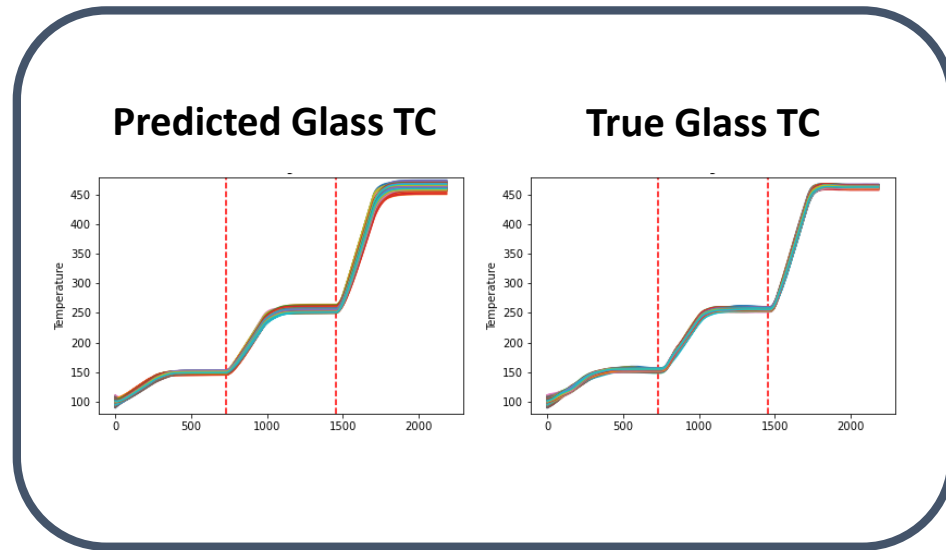
Step 1. Formulate Into Mathematical System

- Assume Glass TC is observable
- Model: What we want to know in the system
 - Interaction between future Glass TC and other variables
 - $\hat{x}_{t+1} = \hat{f}(x_t, u_t; \theta)$, predicted Glass TC
 - Let true system is $x_{t+1} = f(x_t, u_t)$

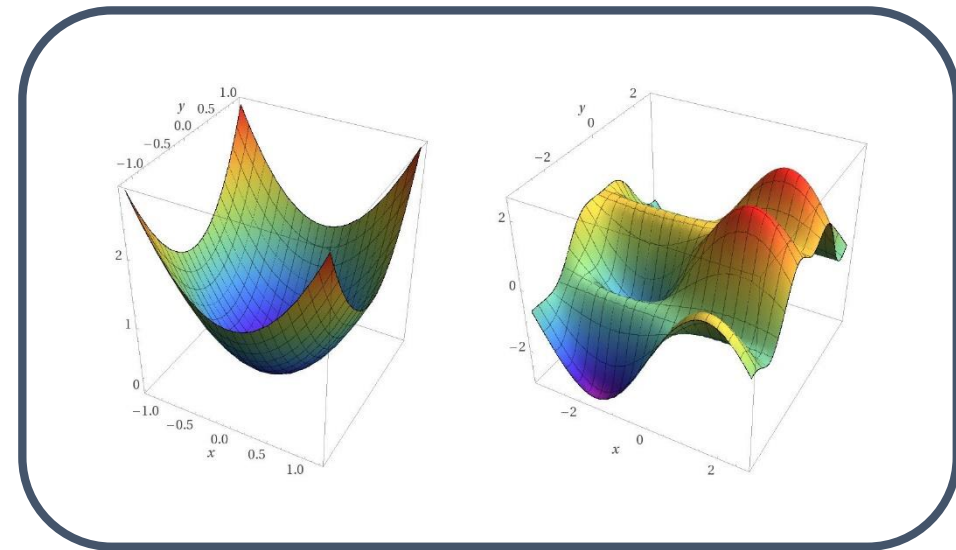


Step 2. Select Model Class

- Many different model classes
 - Linear, Deep neural network, Graph neural network, Gaussian process, etc.
- There is no perfect answer to choose the model class



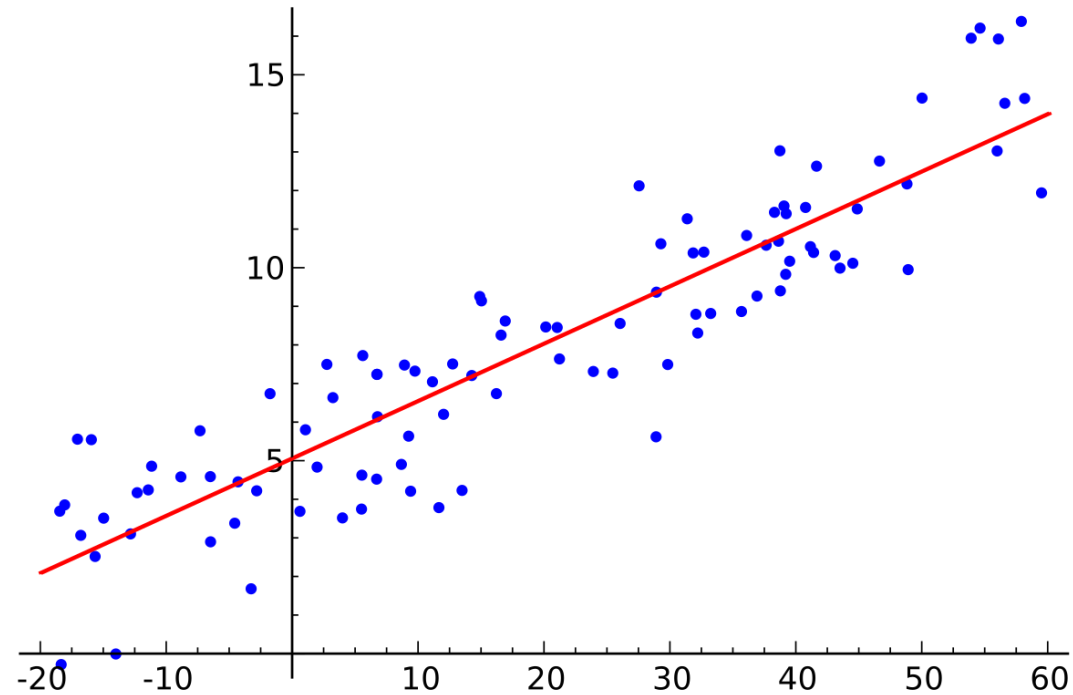
Model expressivity



Optimization solvability

Linear Model

- Simplest data-driven model
- Parameters: a weight matrix W and bias vector b



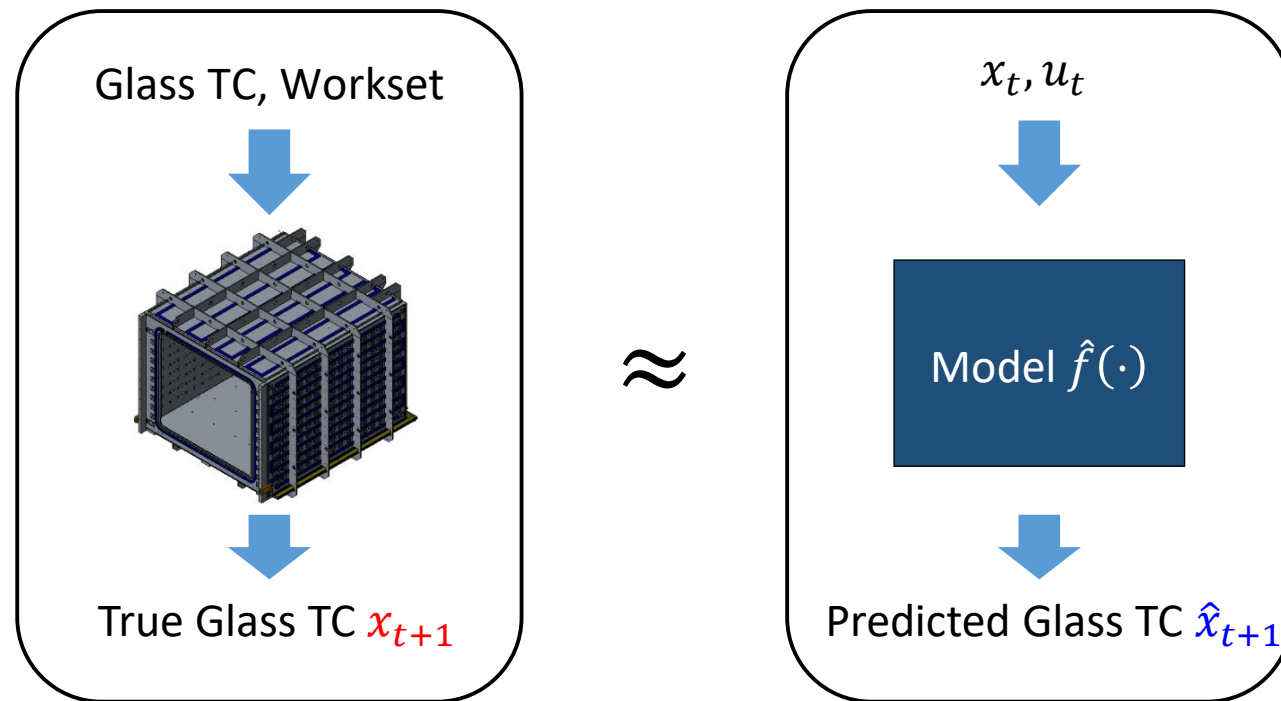
$$y = Wx + b$$

Step 3. Prepare the Data

- Let's skip for now

Step 4. Learn the Model

- Role of model: Substitute the true system in decision-making
 - Good model = Good approximation to the true system
- Make the **predicted Glass TC** as close as the **true Glass TC**



Step 4. Learn the Model

- Meaning of 'learning model'
 - Find the optimal model parameters θ from the data
 - $\hat{f}(x_t, u_t; \theta) \approx f(\cdot)$ as possible as we can

- Mathematically,

$$\begin{aligned} \min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(x_t, u_t, \mathbf{x}_{t+1}) \in \mathcal{D}} (\mathbf{x}_{t+1} - \hat{\mathbf{x}}_{t+1})^2 \\ \text{s.t. } \hat{\mathbf{x}}_{t+1} = \hat{f}(x_t, u_t; \theta) \end{aligned}$$

- \mathcal{D} : prepared dataset
- How can we solve this? Gradient Descent Algorithm!

General Description of Optimization Problem

Objective function $\min_x f(x)$

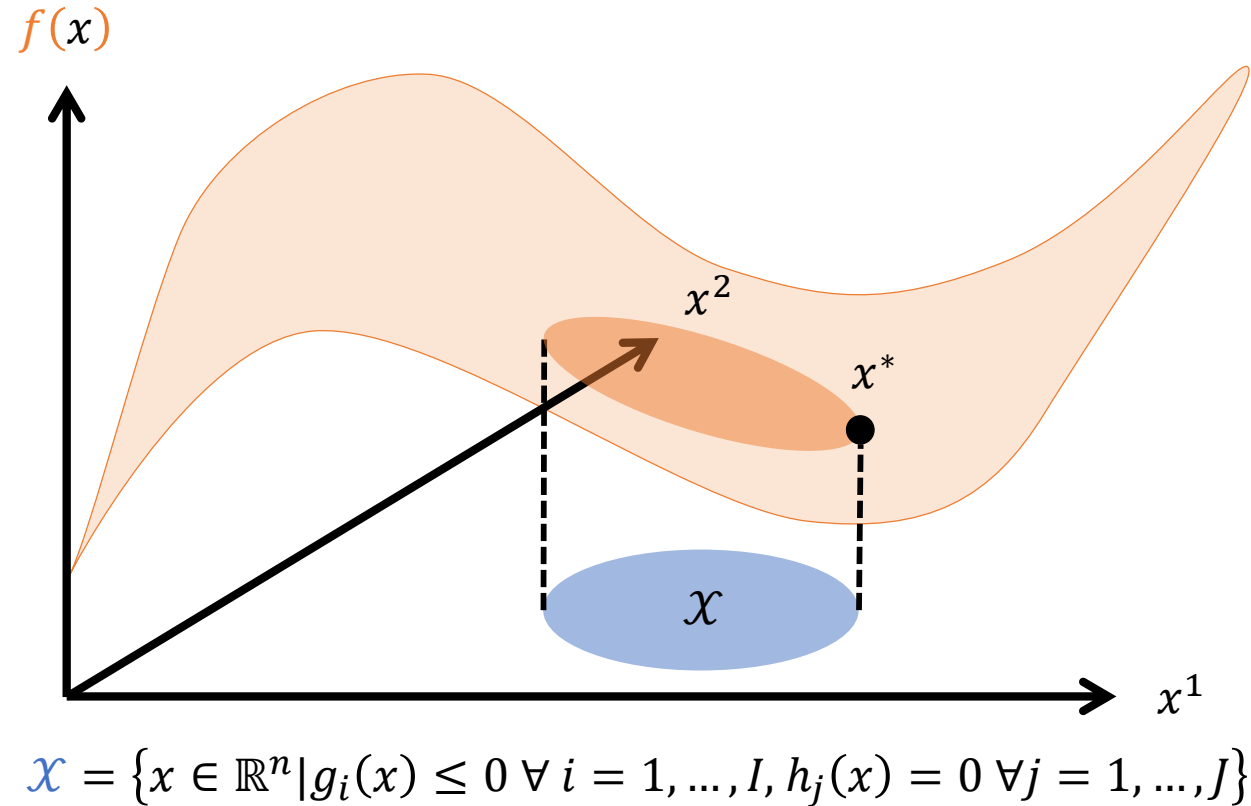
Constraints $s. t. \begin{cases} g_i(x) \leq 0 \quad \forall i = 1, \dots, I & \text{Inequality constraints} \\ h_j(x) = 0 \quad \forall j = 1, \dots, J & \text{Equality constraints} \end{cases}$

(s. t. : Acronym of 'subject to')

where $x = \begin{bmatrix} x^1 \\ \vdots \\ x^n \end{bmatrix} \in \mathbb{R}^n, f, g_i, h_j: \mathbb{R}^n \rightarrow \mathbb{R} \quad \forall i = 1, \dots, I, j = 1, \dots, J$

- **Optimal solution** x^* is the minimizers of the optimization problem. Possibly exist **multiple** optimal solutions
- **Optimal value** $f(x^*)$ is the minimal (maximal) function value
- **Feasible solutions (search space)** $\mathcal{X} = \{x \in \mathbb{R}^n \mid g_i(x) \leq 0 \quad \forall i = 1, \dots, I, h_j(x) = 0 \quad \forall j = 1, \dots, J\}$
 - If objective or constraints are not **scalar valued**, then we can't solve the problem with optimization techniques.
 - such setting requires different solution concepts \rightarrow Game Theory

Visual Understanding of Optimization Problem



- Naïve approach: “Trial-and-error”
 - Plug in every solution in \mathcal{X} and Find the solution that is corresponds to the minimal function value.

Can we Do Better?

- If the objective or constraint not differentiable,
 - We can come up with heuristics:
 - Simulated Annealing, Genetic algorithms.
 - If the optimization problem is combinatorics, use Branch and Bound (B&B) or MILP etc.
- What if the objective and constraint is differentiable?
 - Use Gradient / Hessians to solve the optimization problem efficiently.

Sidewalk: Extremum and Optimum

Recall our memory of high school mathematic class

문제) 함수 $f(x) = (x - 3)^2 + 2$ 일 때, 함수의 최솟값을 구하면?

- ① 5 ② 4 ③ 3 ④ 2 ⑤ 1

해설) $\frac{df(x)}{dx} = 2(x - 3)$ 이며, 극댓값 정리, $\frac{df(x)}{dx} = 0$ 인 극값에서, 최대/최소값이 존재, 를 활용해서 $2(x - 3) = 0$ 를 만족하는 극값을 찾으면, $x = 3$.

극점 (Extreme point = local optima) 는 gradient $\frac{df(x)}{dx}$ 가 0 이 되는 지점!

Example 1) $f(x) = ax^2 + bx + c$ for $a > 0$, then $x^* = -\frac{b}{2a}$

Example 2) $f(x) = \sin x$, then $x^* = \left(2k + \frac{3}{2}\right)\pi \quad \forall k \in \mathbb{Z}$

How to Solve Unconstrained Optimization Problem?

$$\min_x f(x)$$

- What if $f(x)$ is complicate so we can't find analytical form of optimal solution?
- Use “Gradient descent algorithm” to iteratively find the optimal solution x^*

Taylor Approximation

- Taylor approximation is a local approximation of function.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^n(a)}{n!} (x-a)^n$$

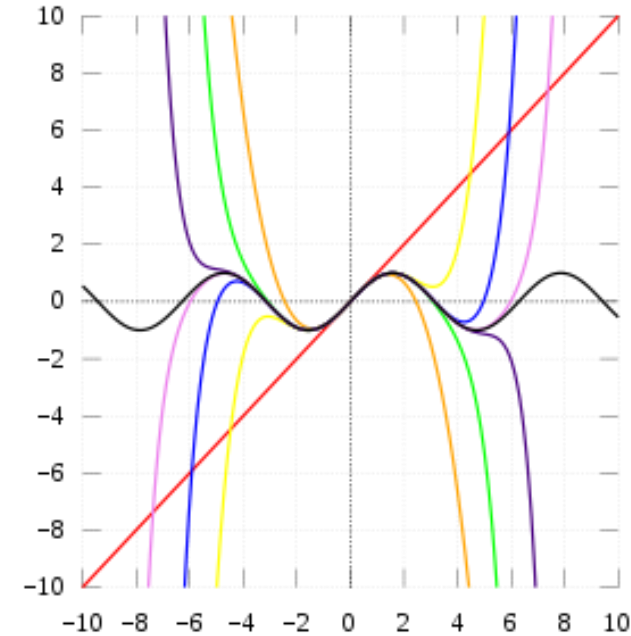
Taylor series

$$= \sum_{n=0}^T \frac{f^n(a)}{n!} (x-a)^n$$

T order
Taylor approximation

$f^n(a)$: n^{th} derivative of f at a

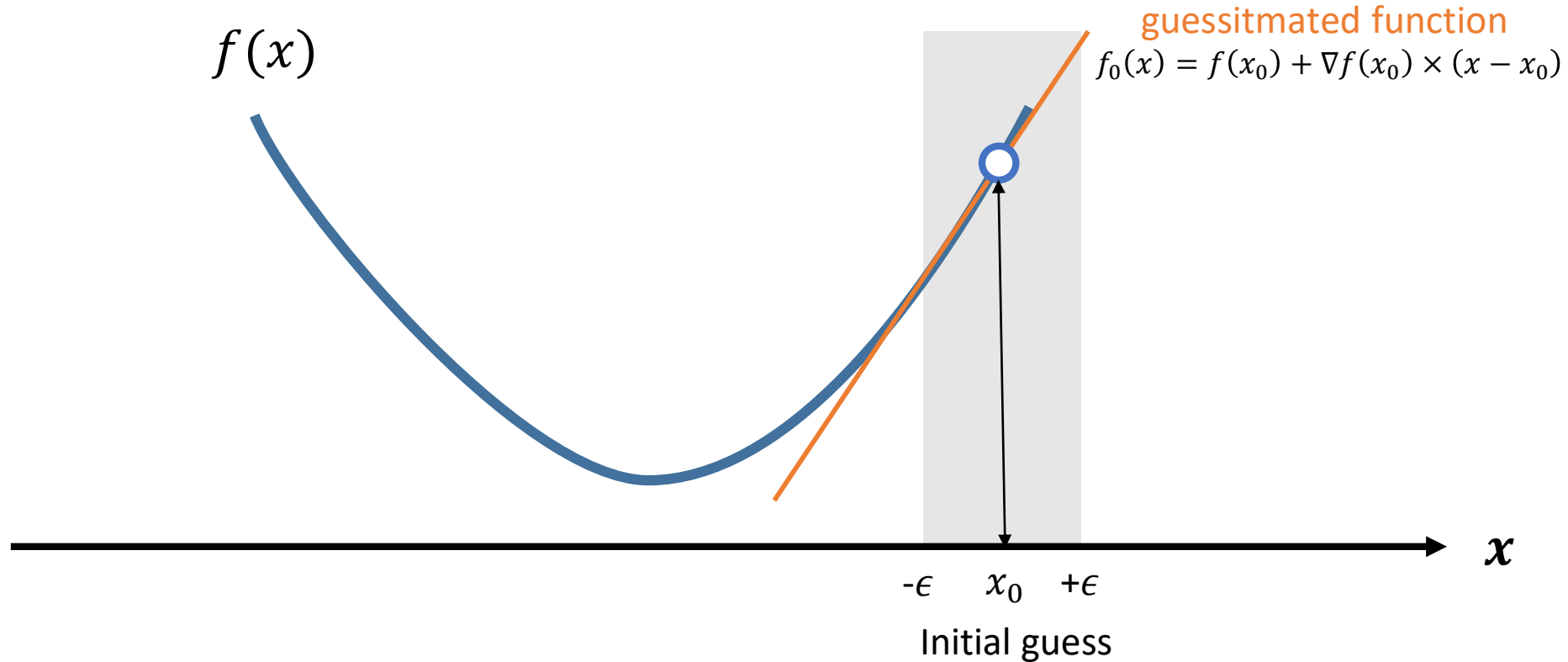
This approximation is valid when $|x - a| < \epsilon$.i.e., local approximation



Taylor approximations of function

Gradient Descent Algorithm

*Utilize 1st order local approximation of function to find the next good point!



- Since Taylor approximation becomes more inaccurate as we move far from x_0
- Trust the approximated function in locale only! i.e., Move our guess slightly
- Set next good point as $x_1 \leftarrow x_0 + \eta \nabla f(x_0)$. step size (learning rate) η is sufficiently small number.

*Utilizing 2nd order local approximation of function can be also used → Newton methods

Gradient Descent Algorithm

(Vanilla) Gradient Descent

1) Initialize an arbitrary $x = x_0 \in \mathbb{R}^n$, step size $\eta > 0$, tolerance $\epsilon > 0$

2) For $t = 0, 1, \dots$,

2-1) Compute $\nabla f(x_t) = \begin{bmatrix} \frac{\partial f}{\partial x^1}(x_t) \\ \vdots \\ \frac{\partial f}{\partial x^n}(x_t) \end{bmatrix}$

2-2) Update $x_{t+1} \leftarrow x_t - \eta \nabla f(x_t)$

2-3) break if $|x_{t+1} - x_t| < \epsilon$

Example of Gradient Descent Algorithm

$$\min_x f(x) = (x - 3)^2 + 1$$

- Start with $x = x_0 = 5$, $\eta = 0.1$, $\epsilon = 0.01$

- For $t = 0$,

Compute $\nabla f(x_0) = 2(x_0 - 3) = 4$

Update $x = x_1 = x_0 - \eta \nabla f(x_0) = 5 - 0.1 * 4 = 4.6$

Since $|5 - 4.6| > \epsilon$, continue for loop

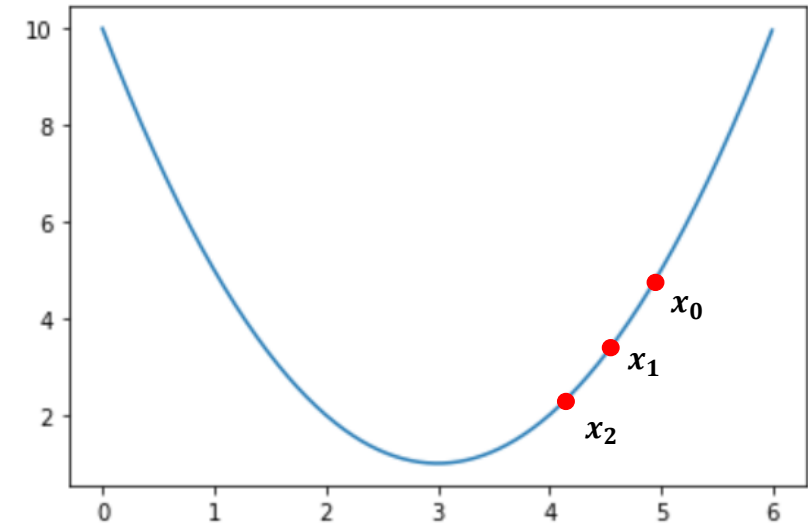
- For $t = 1$,

Compute $\nabla f(x_1) = 2(x_1 - 3) = 3.2$

Update $x = x_2 = x_1 - \eta \nabla f(x_1) = 4.6 - 0.1 * 3.2 = 4.28$

Since $|4.6 - 4.28| > \epsilon$, continue for loop

- And so forth...



Limitations of Gradient Descent Algorithm

- If $f(x)$ is a convex function, GD find the optimal solution x^*
- If $f(x)$ is not convex, GD does not guarantee the optimality of converged solution.
- As a remedy, several GD variants are developed and works well in practice.
 - Stochastic Gradient Descent (SGD)
 - Momentum
 - AdaGrad
 - AdaDelta
 - RMSprop
 - Adam

Code Exercise!

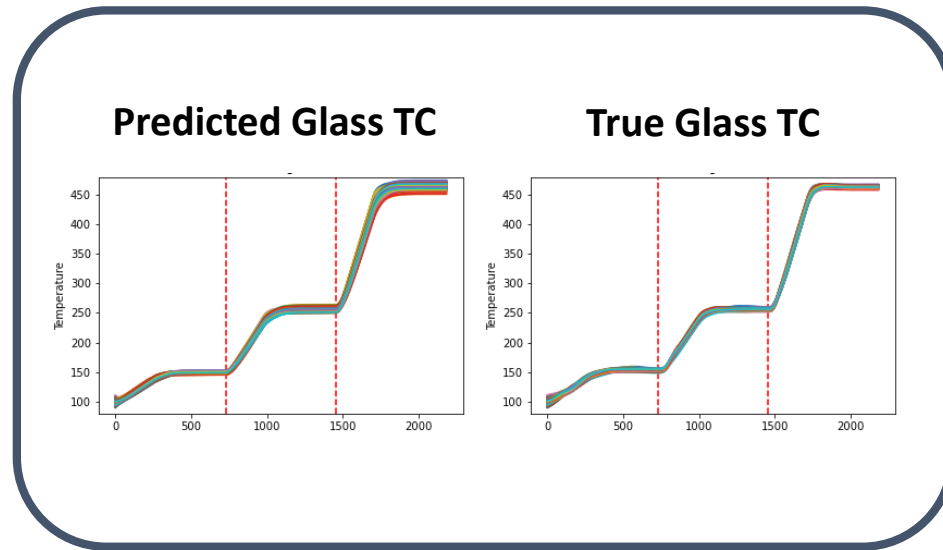
github.com/song970407/WONIK-KAIST-Day1/settings

Real-World is Different...

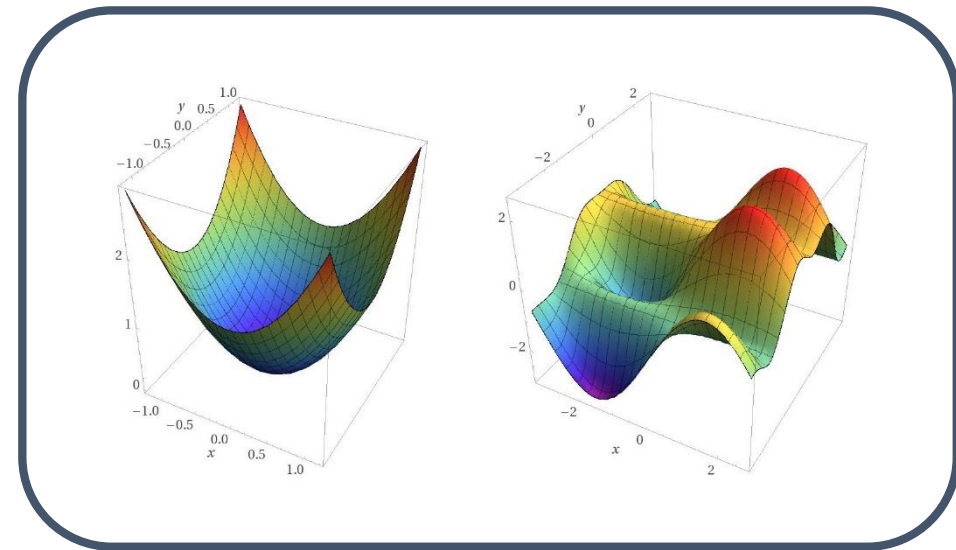
- Lots of variables
 - In case of WONIK, 40 control inputs and 140 state variables
- More complicate true system
 - Linear model may be insufficient to approximate the true system
- Different data scale
 - Bad for training
- Introduce more techniques to apply in the real-world problem

Step 2. Select Model Class

- Many different model classes
 - Linear, Deep neural network, Graph neural network, Gaussian process, etc
- Important: Balance between Model expressivity and Optimization solvability

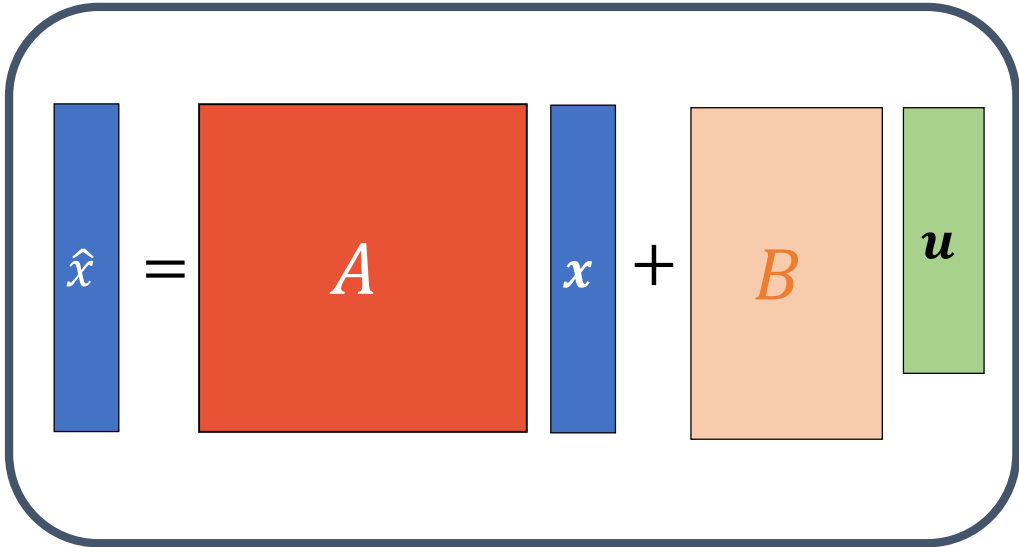


Model expressivity

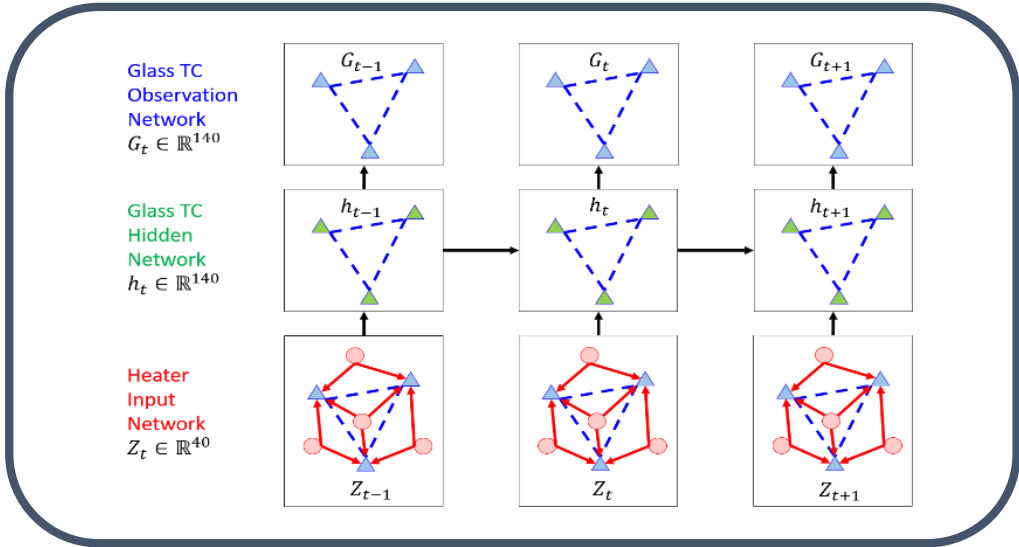


Optimization solvability

(Recap) Step 2. Select Model Class



Linear Model



Non-Linear Model

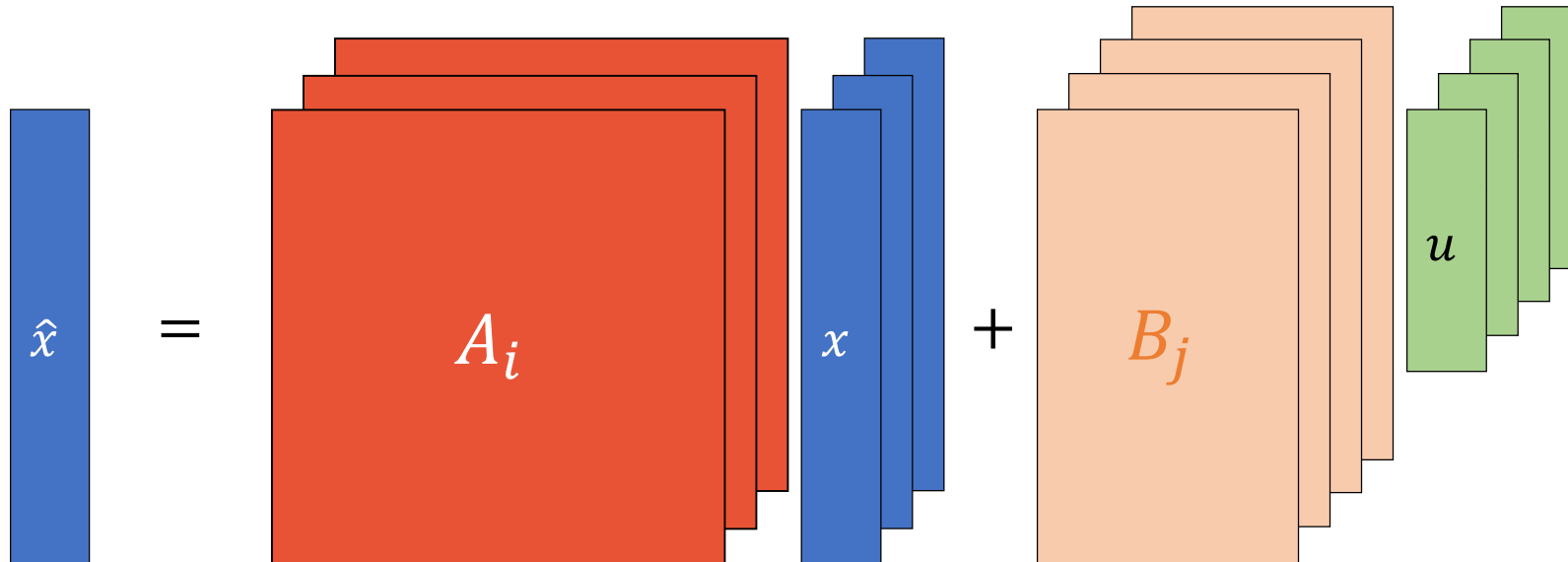
	Linear Model	Non-Linear Model
Model Expressivity	↓	↑
Optimization Solvability	↑	↓
Example	Linear / Multistep Linear	NN, GNN

Multistep Linear Model

Predicted Glass TC

Past Glass TC

Past Workset



$$\hat{x}_{t+1} = \sum_{i=0}^{I-1} A_i x_{t-i} + \sum_{j=0}^{J-1} B_j u_{t-j}$$

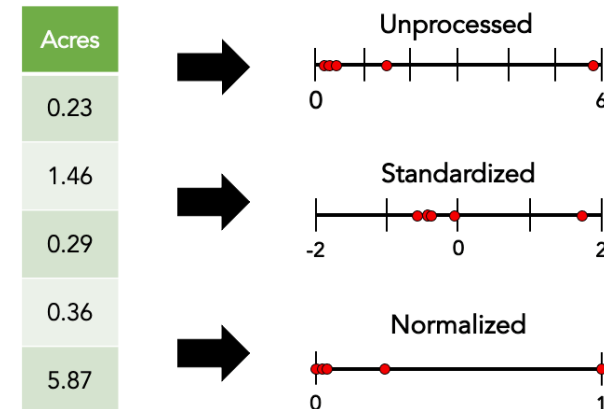
$A_i \in \mathbb{R}^{140 \times 140}$
 $B_j \in \mathbb{R}^{140 \times 40}$

- 현재의 Glass TC와 Work set 뿐만 아니라 과거 값도 사용 → Furnace의 관성 모델링 가능
- **Model expressivity** 와 **Optimization solvability** 를 적절히 밸런싱

Step 3. Prepare the Data (a.k.a. Data Preprocessing)

- Make sure that data is ready to be used
- How?
 - **Missing value**, outlier
 - Categorical variable
 - Variable reduction
 - **Feature scaling**

Table (1): A hypothetical example of numerical data			
	Column 1	Column 2	Column 3
Row 1	26	22	12
Row 2	Green	8	7
Row 3	84	60	-



Hyperparameter Tuning

- Model parameter vs Hyperparameter
 - Model parameter: can be estimated from the data
 - Hyperparameter: cannot be estimated from the data
- Select the best hyperparameter by comparing the model performance
 - The devil is here...
- Learning rate, epoch, state/action order, etc

Trick 1: Multistep Prediction

- Not only predict \hat{x}_{t+1} but also $\hat{x}_{t+2}, \hat{x}_{t+3}, \dots$
 - Important: Use x_t and a sequence of future control input (u_t, u_{t+1}, \dots)
 - $\hat{x}_{t+1} = \hat{f}(x_t, u_t; \theta), \hat{x}_{t+2} = \hat{f}(\hat{x}_{t+1}, u_{t+1}; \theta), \dots$
 - Much harder than one step prediction
- Furnace does not change dramatically within just 5 seconds, which means $x_t \approx x_{t+1}$
 - The learned model can be a trivial model, which is $\hat{f}(x_t, u_t) = x_t$
 - By minimizing MSE between $(x_{t+1}, x_{t+2}, \dots)$ and $(\hat{x}_{t+1}, \hat{x}_{t+2}, \dots)$, we can obtain more reasonable model

Trick 2: Good Model Parameter Initialization

- Initial model parameter is super important
 - Stable & fast training
- Naïve idea: Just return the current state $\hat{x}_{t+1} = \hat{f}(x_t, u_t; \theta) = x_t$
 - Initialize A, B as $A_0 = I$, otherwise $A_i = B_j = 0$.
 - $\hat{x}_{t+1} = \sum_{i=0}^{I-1} A_i x_{t-i} + \sum_{j=0}^{J-1} B_j u_{t-j} = A_0 x_t + 0 = x_t$

Dive into the Real Code
