데이터분석기법

11주차: Defintion of Machine Learning, Linear Models and Optimisation Algorithms

양승진
slowmoyang@gmail.com

세종대학교

2025년 5월 13일

Goals for Today

- Understand the basic concept of machine learning (ML)
- Learn how a linear regression model works
- Explore how gradient descent is used for model optimization

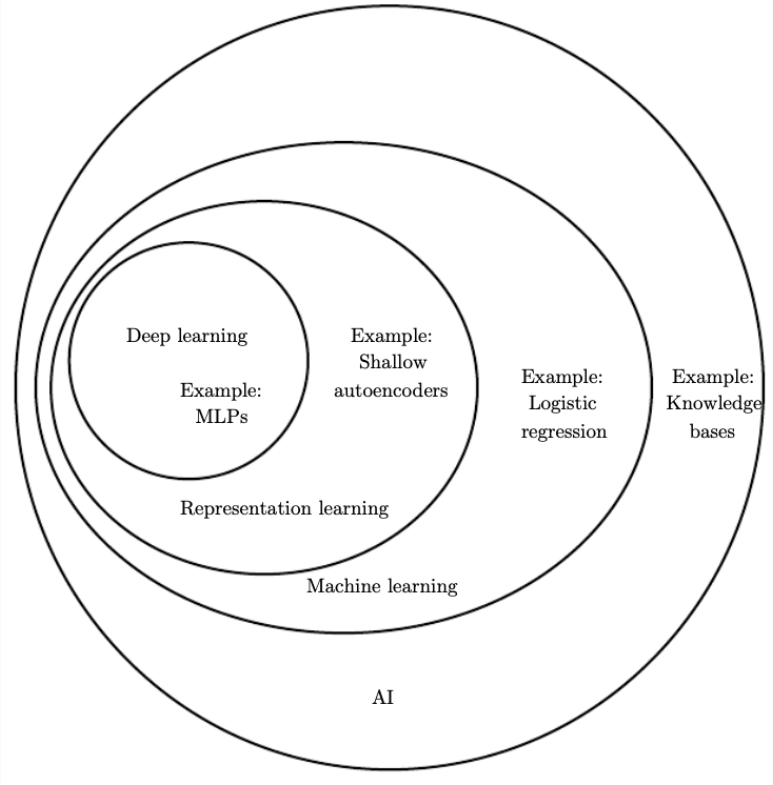
Machine Learning

What is Machine Learning?

A computer program is said to learn from experience E w.r.t. some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [3]

Source: [4]

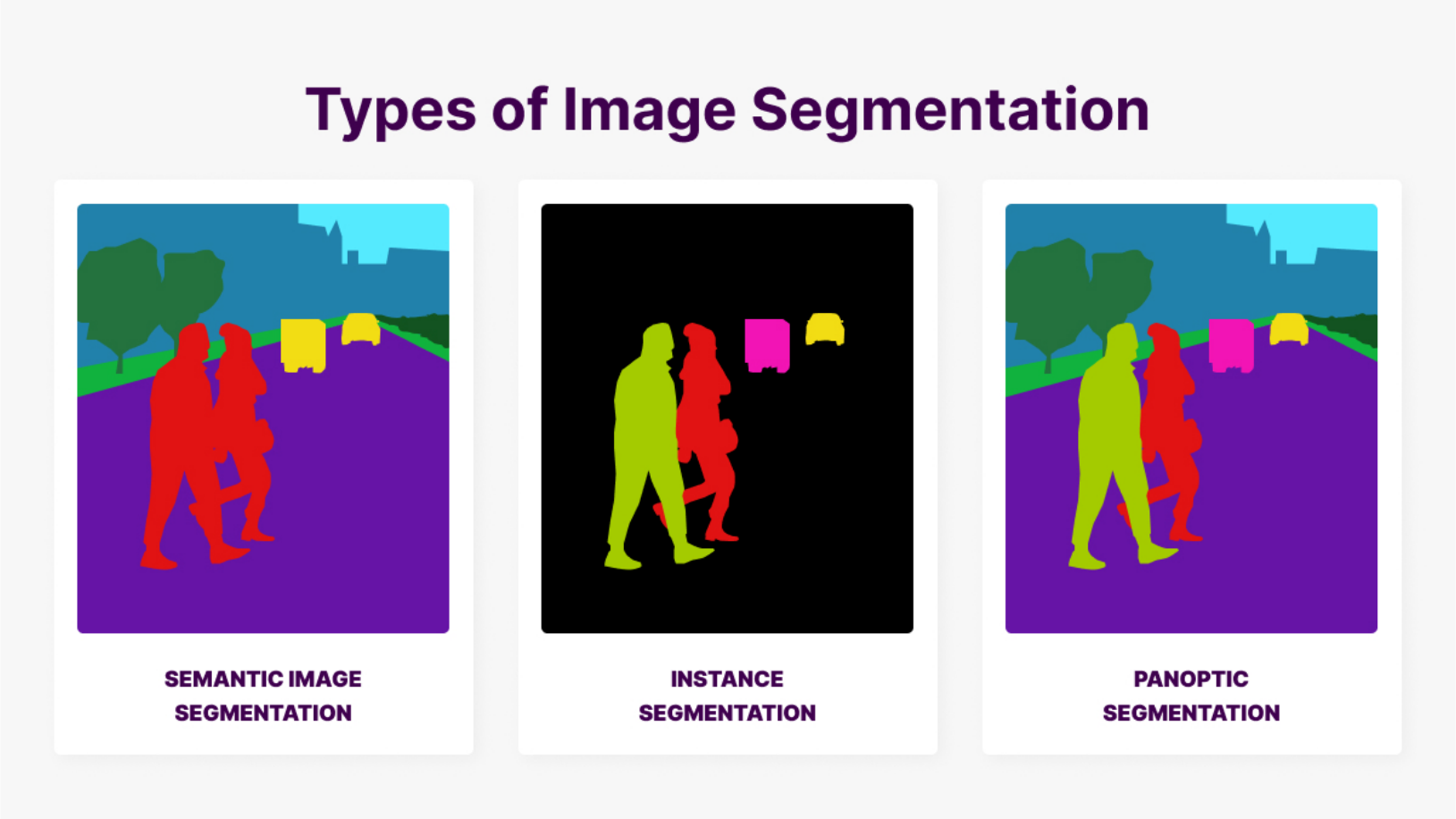
Machine Learning & Artificial Intelligence



Source: [1]

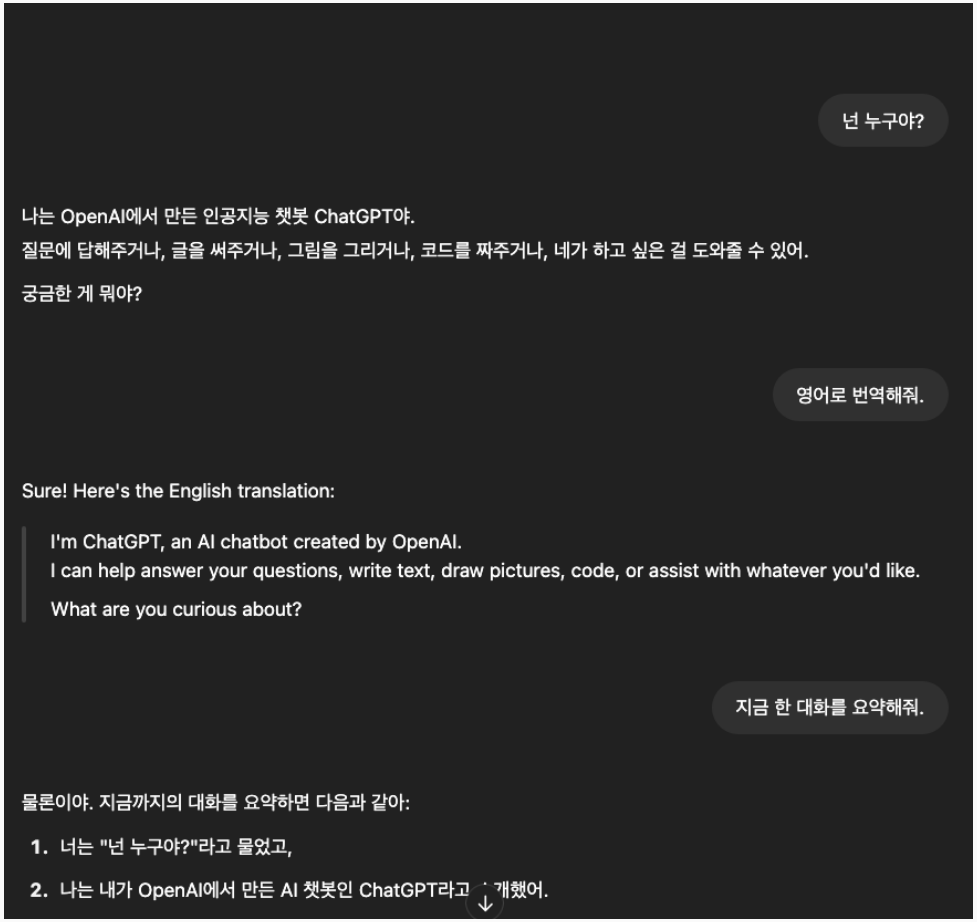
Machine Learning (ML) $\subset$ Artificial Intelligence (AI)

Examples of ML Applications



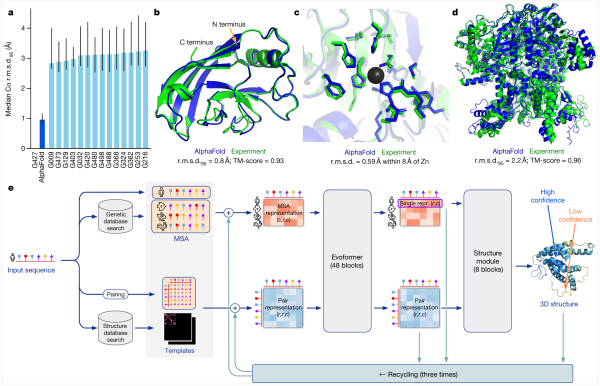
Computer vision: image classification, object detection, semantic segmentation, pose estimation, image generation

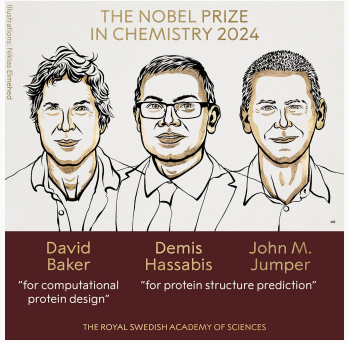
Examples of ML Applications (cont.)



Natural language processing: text generation, translation, text summarization

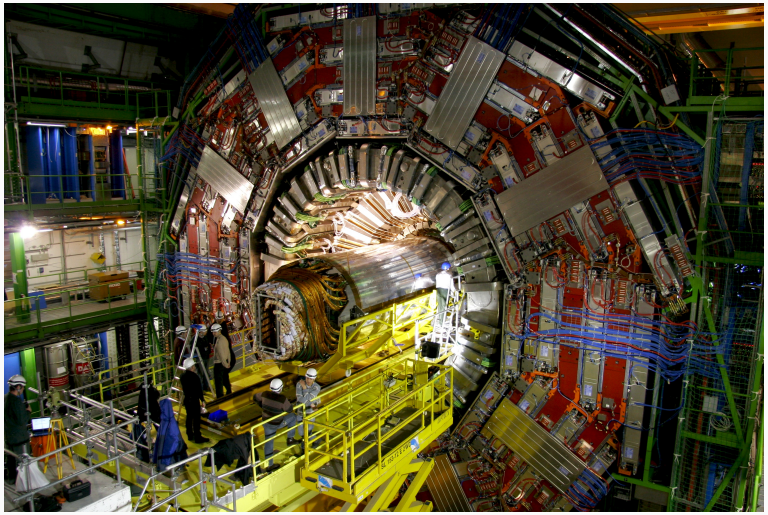
Examples of ML Applications (cont.)

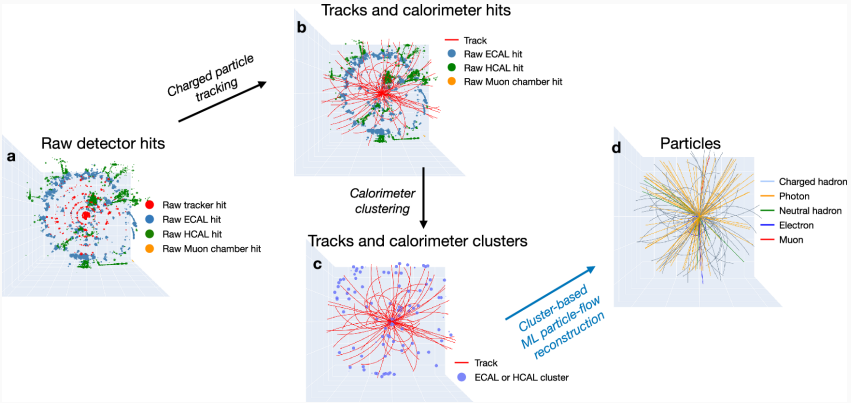




Go playing agent & protein folding prediction

Examples of ML Applications (cont.)





Various applications of machine learning in particle physics experiments, from particle reconstruction to statistical inference

Categories of ML

[X] **Supervised Learning:** The model learns from labelled data, where each input is paired with the desired output

$\longrightarrow$ classification, regression, etc.

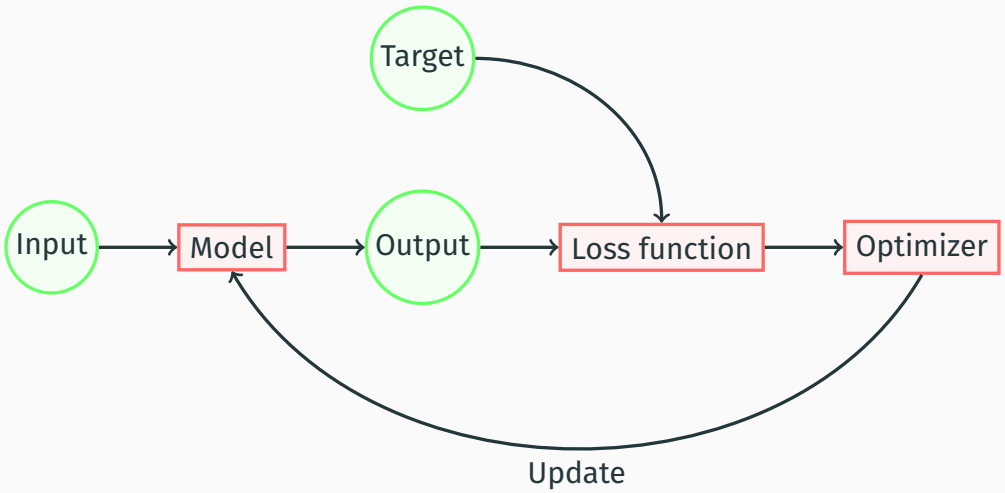
[ ] **Self-Supervised Learning** (or Unsupervised learning)**:** The model generates its own supervision signal from the data itself, learning useful representations without explicit labels

$\longrightarrow$ clustering, generative modelling, etc.

[ ] **Reinforcement Learning:** The model learns by interacting with an environment and receiving feedback

$\longrightarrow$ robot control, game playing agent, etc.

Supervised Learning Setup



- **Input**, x: the information given to a model for predictions
- **Target**, t: labels on x
- **Example**, (x, t): a pair of x and y
- **Dataset**, $\mathcal{D}$: $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
- **Model**, f: an adjustable function that maps x to output, y
- **Loss function**, J: a measure of how far y are from t
- **Optimizer**: an algorithm used to adjust a model to minimize J

ML: Input& Target

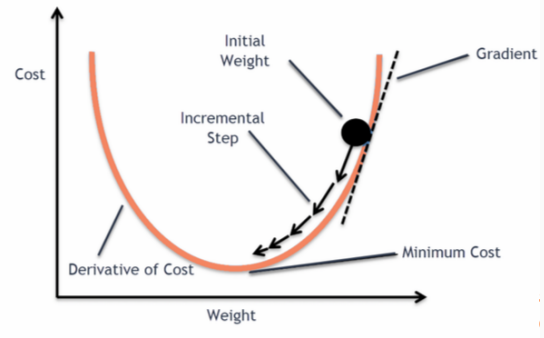
- Typically, input data is represented as an array of vectors
  - feature vector: numerical properties of an object
    - → cat = (length=70 cm,# of legs=4, cuteness=100)
  - text: a sequence of integer numbers, each representing an alphabet or a word.
    - → cat = "A cat is a cute carnivorous mammal" = [1, 32, 6, 1, 102, 4082, 1064]
  - image: an 2D array of pixels
    - → cat=🐈
  - 3D image, network, point cloud, etc.
- In supervised learning, the target is the output or label that the model aims to predict from the input data
  - classification: An integer index representing the category of the input
  - regression[1]: continuous values corresponding to the input

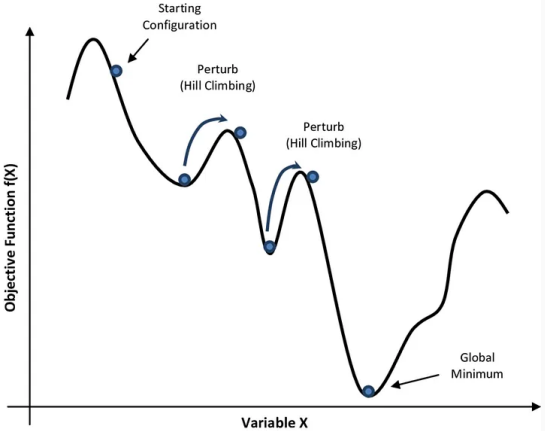
[1] "Regression" was coined by Francis Galton in the 19th century to describe how offspring of tall ancestors tend to have heights closer to the average (the so called regression toward the mean)

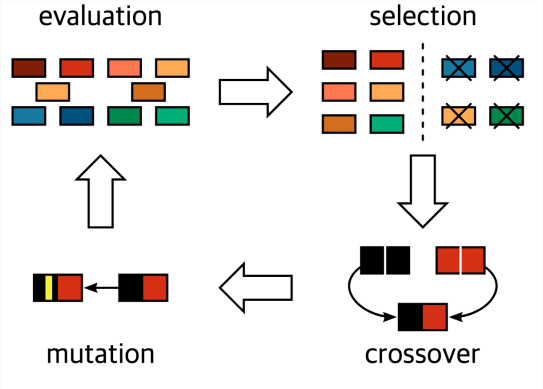
ML: Models

- [X] Linear Models
- [X] Decision Trees
- [ ] Support Vector Machines
- [ ] k-Nearest Neighbors
- [ ] ⋮
- [X] Deep Learning (DL)
  - [X] Multi-Layer Perceptrons
  - [X] Convolutional Neural Networks
  - [ ] Recurrent Neural Networks
  - [ ] Transformers
  - [ ] ⋮

ML: Optimizers







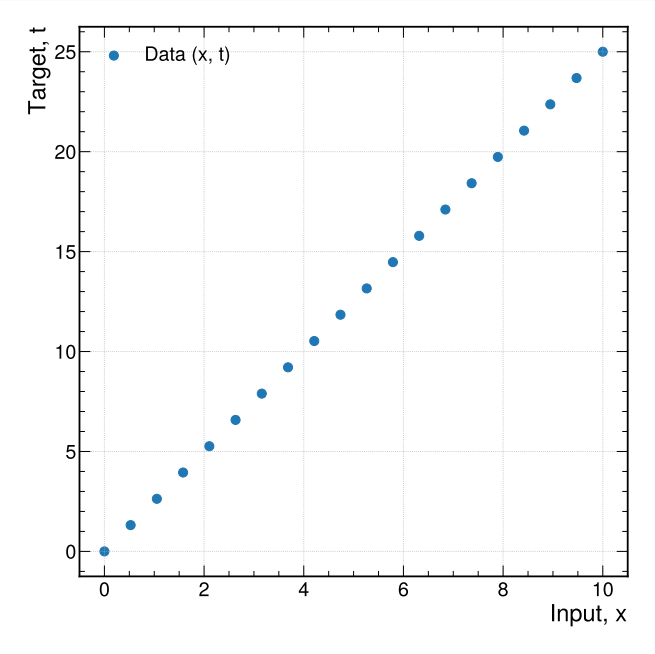
Gradient descent, simulated annealing, and genetic algorithm

Various optimization algorithms, based on statistical learning theory, are used to find optimal solutions in machine learning

- **Gradient Descent** follows the slope to descend to a local minimum
- **Simulated Annealing** explores the solution space with occasional uphill moves to escape local minima,
- **Genetic Algorithm** mimics natural evolution through selection, crossover, and mutation

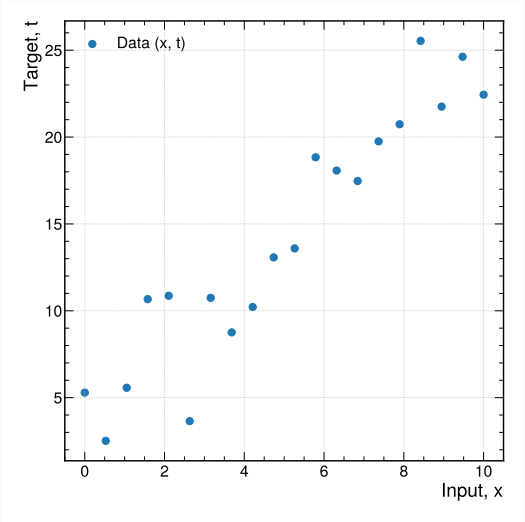
Linear Regression

Setup



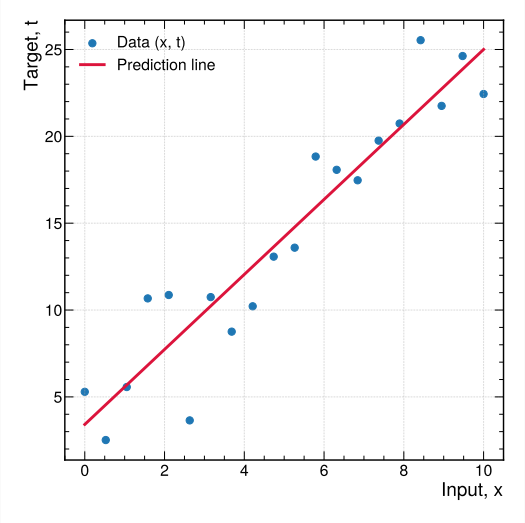
- Our goal is to train a function f that maps input x to a prediction y close to the target t
- Here is the simplest example where **target t has a linear relationship with input x**

Setup (cont.)



- Our goal is to train a function f that maps input x to a prediction y close to the target t
- Here is the simplest example where target t has a linear relationship with input x
- In real-world data, there are always **unpredictable variations** that cannot be fully described by a statistical model, known as **random noise**

Linear Regression Model

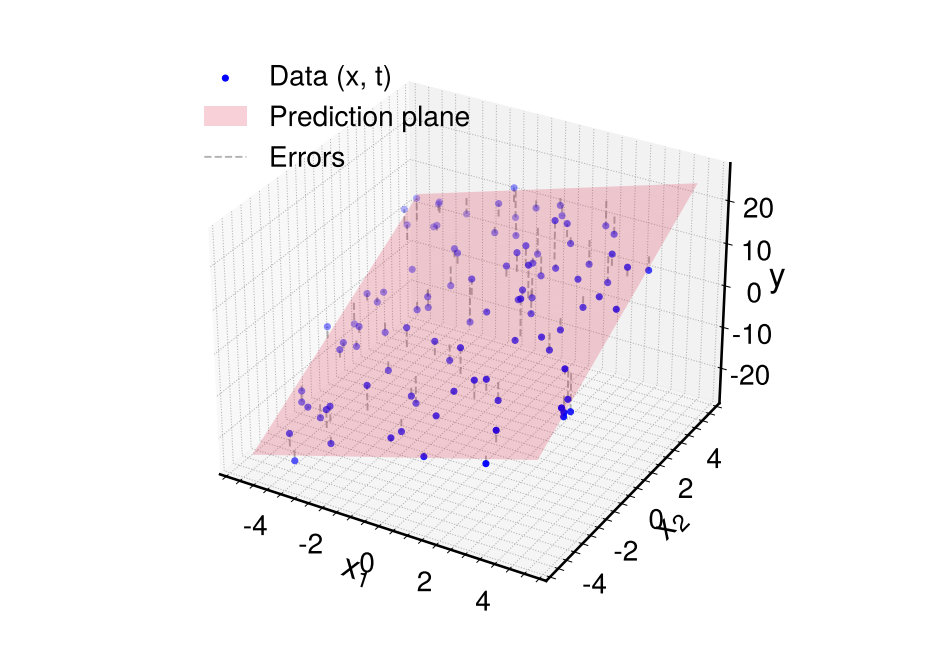


- In the 1D case, linear regression is a ML model of the form:

$$y = f(x) = wx + b, \tag{1}$$

  - where w is **weight** and b is the **bias** – both are trainable **parameters**
- The output y has a linear relationship with the input x
- Note that the linear model is also linear w.r.t. the parameters w

Multiple Linear Regression

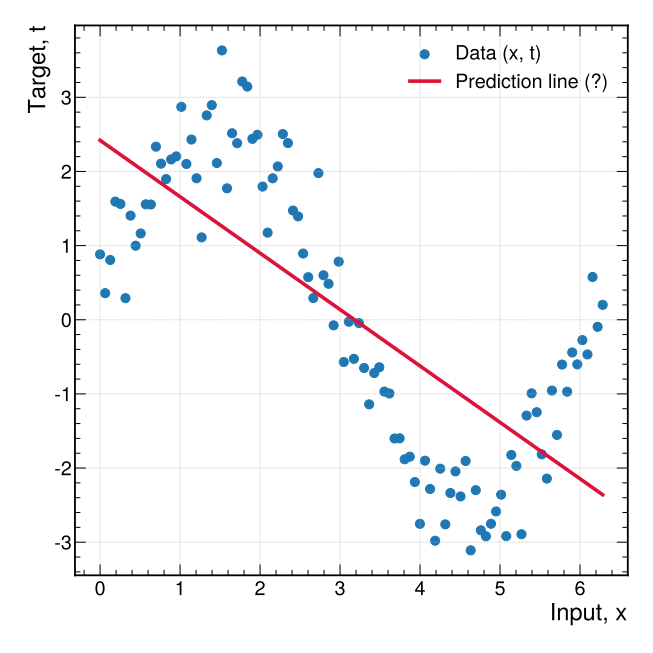


If the input has D features, the linear model can be naturally extended as

$$y = f\left(x_1, \cdots, x_D; w_1, \cdots, w_D, b\right) = \sum_{i=1}^{D} x_i w_i + b = \sum_{i=0}^{D} x_i w_i = \mathbf{x}^T \mathbf{w}, \tag{2}$$

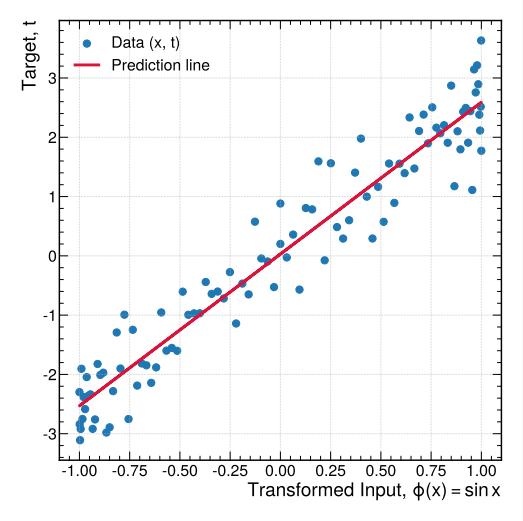
where $w_0 = b$, and $x_0 = 1$

Nonlinearity in Data



Can you model this data using a linear regression model? 🤔

Basis Function

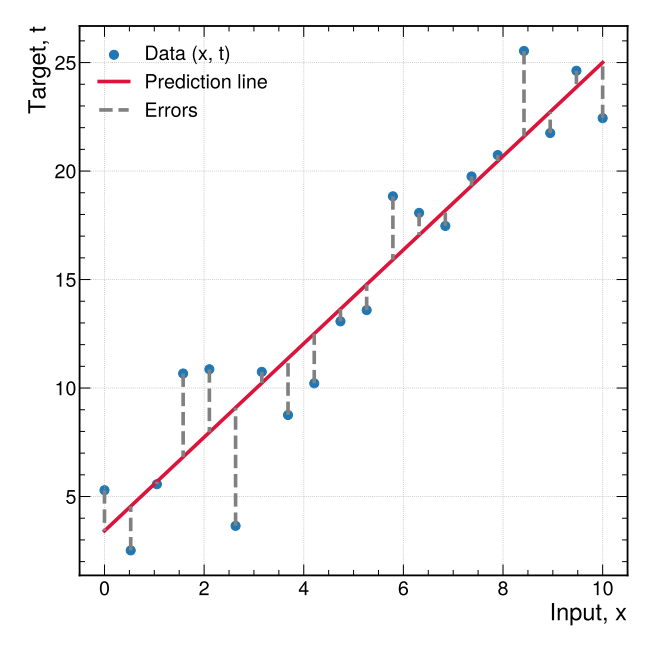


A **basis function** transforms $\mathbf{x}$ to enable a linear regression model to capture nonlinearity between t and $\mathbf{x}$

$$y = f(\mathbf{x}; \mathbf{w}) = \sum_{j=0}^{M} w_j \phi_j(\mathbf{x}), \tag{3}$$

where $\phi_0(\mathbf{x}) = 1$

Loss Function



- A **loss function** measures how well a model makes predictions
- Mean squared errors (MSE)

$$J(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (y_i - t_i)^2 \tag{4}$$

difference y - t is called the error or residual

Direct Solution

- Matrix notation: We can collect all the training input data into a single matrix X, called the **design matrix**

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1D} \\ & \vdots & & \\ 1 & x_{N1} & \cdots & x_{ND} \end{pmatrix}, \quad \mathbf{t} = \begin{pmatrix} t_1 \\ \vdots \\ t_N \end{pmatrix} \tag{5}$$

$$J(\mathbf{w}) = \frac{1}{2N} \|X\mathbf{w} - \mathbf{t}\|^2 \tag{6}$$

- Because the output of a linear regression model is linear in the weight, the MSE loss is a parabolic function of that weight
- In linear regression, we can analytically find the optimal parameters $\mathbf{w}^*$, by setting the gradient of the loss function to zero

$$0 = \nabla_{\mathbf{w}} J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}^*} = \nabla_{\mathbf{w}} \left(\frac{1}{2N} (X\mathbf{w} - \mathbf{t})^T (X\mathbf{w} - \mathbf{t}) \right) |_{\mathbf{w}=\mathbf{w}^*} = \frac{1}{N} X^T (X\mathbf{w}^* - \mathbf{t}) \tag{7}$$

- If $X^T X$ is invertible,

$$\mathbf{w}^* = \left(X^T X \right)^{-1} X^T \mathbf{t} \tag{8}$$

- Inversion of $X^T X$ has a complexity of $\mathcal{O}(D^3)$

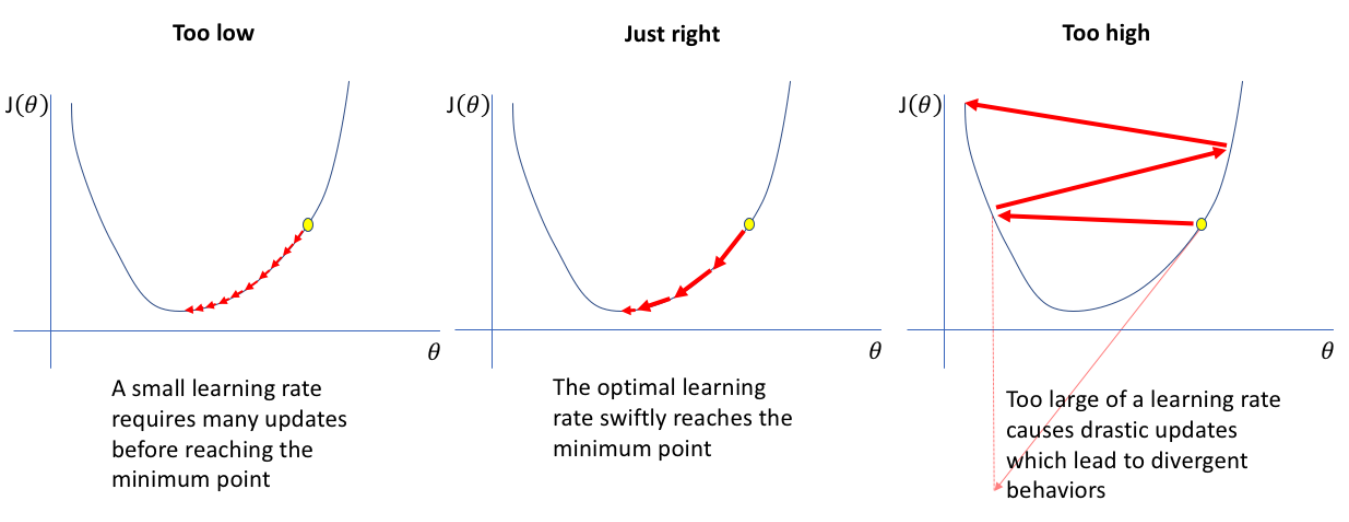
Gradient Descent

- Let's explore a more general approach to minimizing the cost function: gradient descent.
- Gradient descent (GD) is an iterative method—it updates the model step by step until a stopping condition is satisfied
- Starting from an initial guess (like setting all weights to zero), we repeatedly update the weights in the direction that most reduces the cost

$$w_j \leftarrow w_j - \eta \frac{\partial L}{\partial w_j}, \tag{9}$$

  where η is the learning rate
- Each GD update has a complexity of $\mathcal{O}(ND)$, making it much more efficient than the direct solution when $D \gg 1$
- In modern ML softwares such as SCIKIT-LEARN, PYTORCH and TENSORFLOW, GD is fully automated

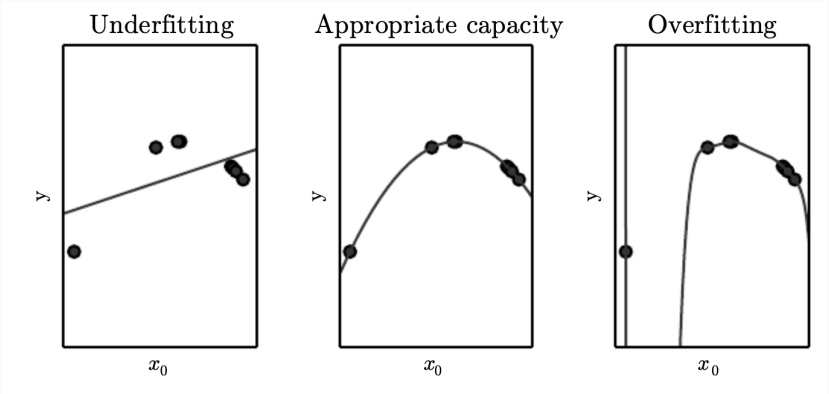
Gradient Descent (cont.)



Source: [2]

- In GD, the learning rate is a parameter that cannot be adjusted by the GD itself because it is not differentiable w.r.t. the model parameters, so it needs to be tuned manually
- The learning rate is a so-called hyperparameter
- We typically choose a small value for the learning rate, usually between 0.0001 and 0.01

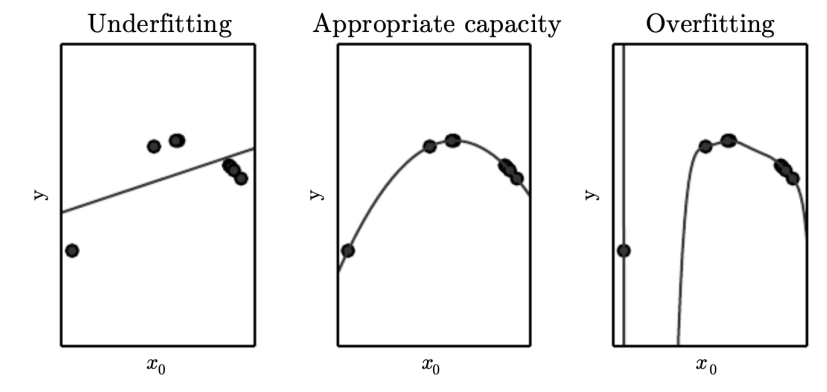
Underfitting & Overfitting



Source: [1]

- Consider the case of a polynomial basis for simplicity
  - $\phi_k(x) = x^k \rightarrow y = f(x; \mathbf{w}) = \sum_{k=0}^{K} w_k x^k$
- data complexity: how "wiggly" or detailed the true underlying function is
- model capacity: how flexible our model is (here, polynomial degree K)

Underfitting & Overfitting (cont.)



Source: [1]

- Consider the case of a polynomial basis for simplicity
  - $\phi_k(x) = x^k \rightarrow y = f(x; \mathbf{w}) = \sum_{k=0}^{K} w_k x^k$
- Left: **capacity < complexity** $\rightarrow$ underfitting
  - model too simple to capture the data's underlying structure
- Middle: capacity $\approx$ complexity$\rightarrow$ good fit
  - model just flexible enough to match the data patterns
- Right: **capacity > complexity** $\rightarrow$ overfitting
  - model too flexible and starts fitting noise or outliers

Training, Validation and Test Sets

- Training and evaluating a model on the same dataset can yield overly optimistic performance estimates, so we split the data into three non-overlapping sets: training, validation, and test
  - adjust the parameters of a model using the **training set**
  - select model's parameters and hyperparameters on the **validation set**
  - assess final performance on the **test set**
- Validation and test sets must never be used for training
- You must never change model's parameters and hyperparameters based on test set performance
- There is no universal ratio for splitting a dataset into training, validation, and test sets
- As a rule of thumb, split the data in a $64 : 16 : 20$ ratio for training, validation, and test sets, respectively

Summary

Do you have a task to solve? If so,

1. Collect input data (e.g. images of 🐈, 🐈‍⬛, 🐕, 🦁, 🐩, . . .)
2. Label your data (cat, cat, dog, cat, dog, . . .)
3. Split the dataset into a training, validation, and test sets (recommended ratio = 64 : 16 : 20)
4. Choose a model type, loss function, optimizer appropriate for your task
5. Choose the hyperparameters of a model
6. Train the model on the training set and tune its parameters using the validation set
7. Iterate 5–6 until you're satisfied
8. Select the best model based on validation performance
9. Evaluate the final model on the test set
10. Solve your problem with the trained model (or go back to the step 1)

Questions?

References i

I. Goodfellow, Y. Bengio, and A. Courville.
***Deep Learning*.**
MIT Press, 2016.
http://www.deeplearningbook.org.

J. Jordan.
Setting the learning rate of your neural network, Mar. 2018.
Accessed: 2025-05-12.

T. Mitchell.
***Machine Learning*.**
McGraw-Hill Education, 1997.

References ii

R. Munroe.
Machine learning.
https://xkcd.com/1838/, May 2017.
Webcomic.