

# 인공지능시스템

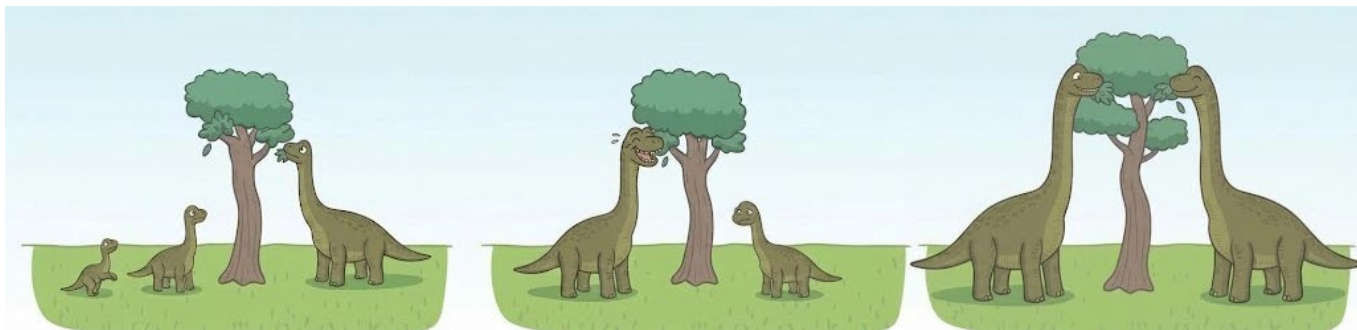
## AI System

int**Ro**-Lab.

청주대학교 무인항공기학과  
김상현 교수 ([www.intro-lab.com](http://www.intro-lab.com))

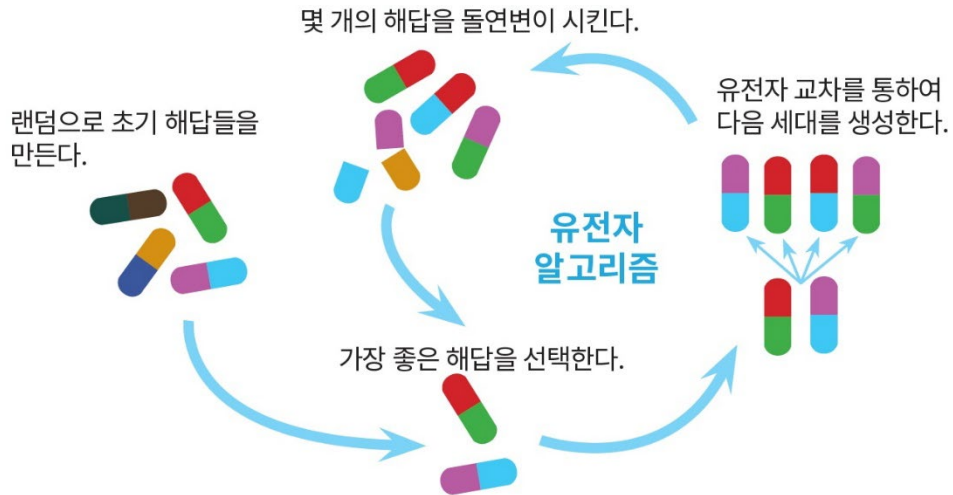
## 유전자 알고리즘

- ◆ 유전자 알고리즘은 생물체의 염색체가 유전되는 현상에서 영감을 얻은 최적화 알고리즘.
- ◆ 적자생존 원칙에 의해 교차, 돌연변이, 도태 등의 과정을 반영.
- ◆ 즉, 주어진 환경에 적합한 유전자만이 살아남는 자연계의 현상을 알고리즘으로 만든 것.



# 유전자 알고리즘

## ◆ 유전자 알고리즘 수행 과정



# 유전자 알고리즘

## ◆ 적합도(Fitness)

- 현재의 염색체가 얼마나 문제를 잘 해결하고 있는지에 대한 점수

## ◆ 평가함수(Evaluation Function)

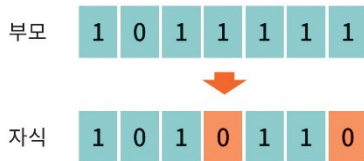
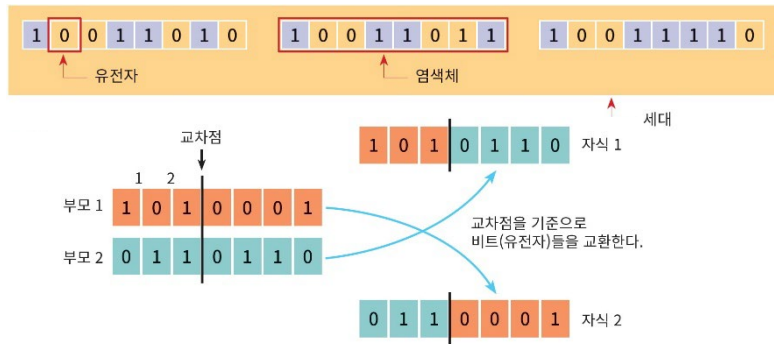
- 적합도를 계산하는 함수

## ◆ 교차(Crossover)

- 부모 염색체의 일부가 교환되는 것

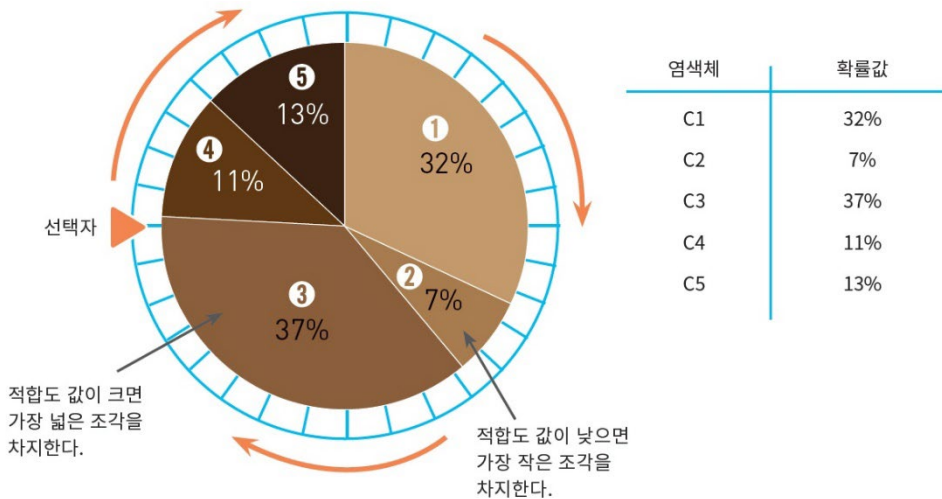
## ◆ 돌연변이(Mutation)

- 염색체에서 랜덤한 위치의 유전자가 값이 변동되는 것

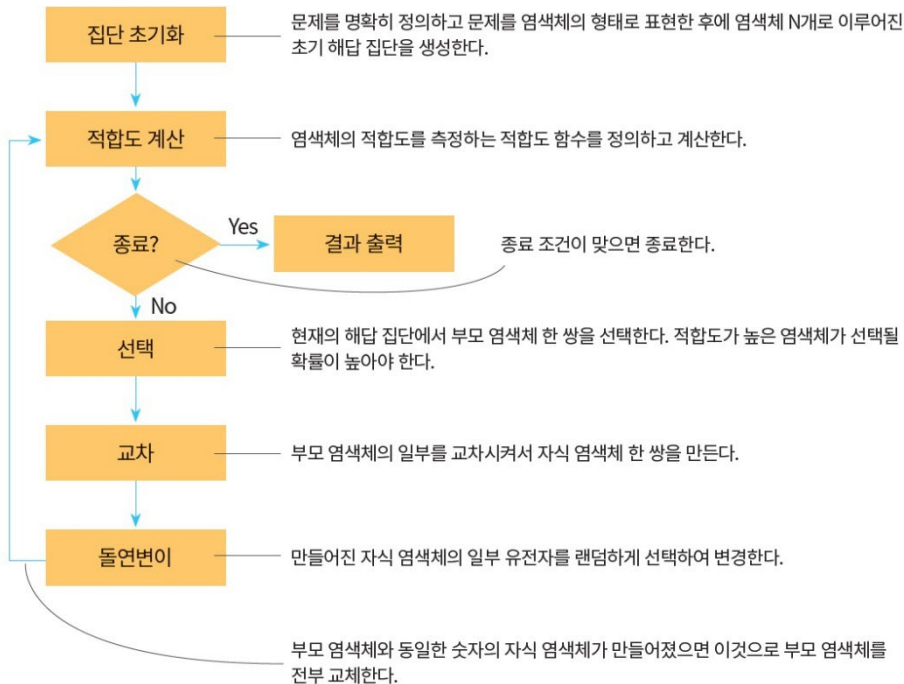


## ◆ 선택 연산자

- 유전자 알고리즘은 재생산을 위하여 우수한 성능(높은 적합도)을 가지는 부모 염색체 2개를 선택
- 이때 사용되는 알고리즘이 룰렛 휠 선택 → 즉, 높은 적합도이면 높은 선택 확률

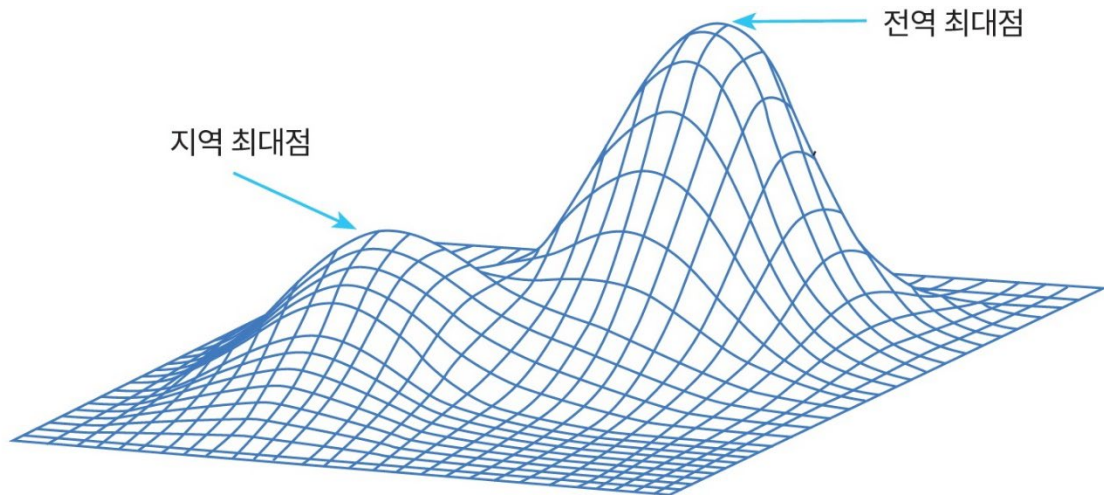


## ◆ 전체 과정



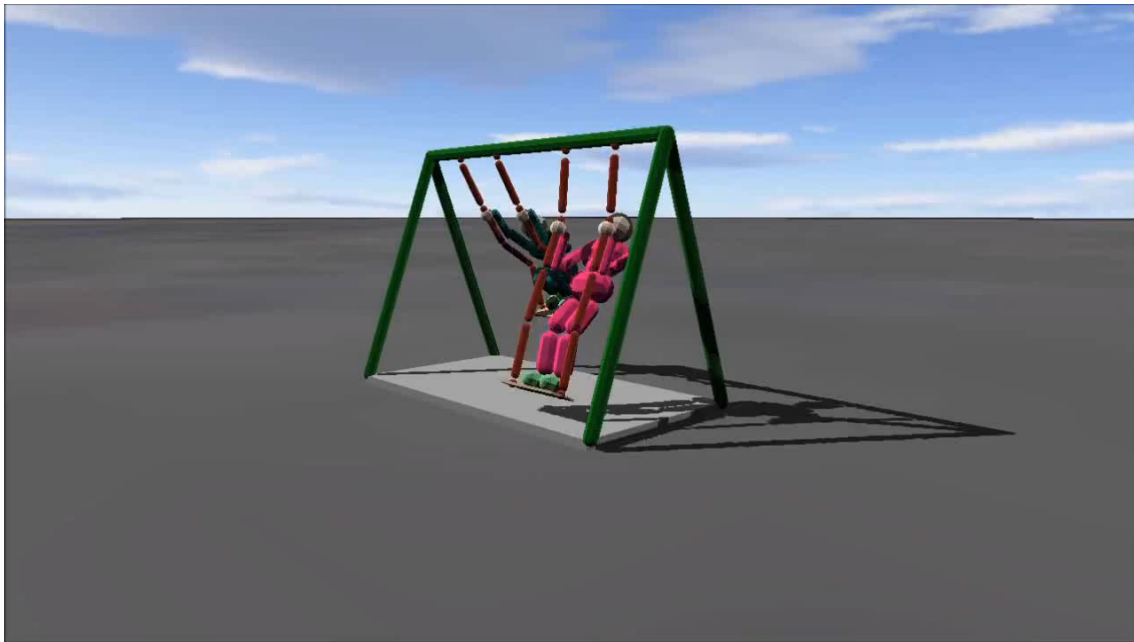
## ◆ 돌연변이 연산이 필요한 이유

- 지역 최대 문제에 빠지는 것을 방지



## ◆ 유전자 알고리즘 시뮬레이션

[https://youtu.be/Yr\\_nRnqeDp0](https://youtu.be/Yr_nRnqeDp0)



## ◆ 장단점

- 결정론적 알고리즘이 존재하는 문제에 대해서는 유전자 알고리즘 접근법이 의미가 없다.
  - 즉, 명확한 답이 정해져 있는 문제에 적용하는 것은 무의미
- 유전자 알고리즘은 본질적으로 확률적 특성으로 인해 수행 시간을 예측하는 것이 어렵다.
- 만족 시켜야 하는 제약 조건이 많은 문제처럼 기존의 방법으로는 해결하기 어렵고 완벽한 최적 솔루션을 찾을 필요가 없는 문제에 주로 사용된다.

# 인공지능 코딩

## ◆ 유전자 알고리즘 코드

염색체(개체) 속 유전자는 1개

염색체(개체)는 10개

적합도:  $f(x) = (x - 1)^2$

최소값 찾는 문제

선택: 개체 1순위, 2순위 선택

교차: 부모 유전자 평균

돌연변이: 유전자 값 임의로 변경

```
1 import random
2
3 # [목적] f(x) = (x-1)^2 의 최솟값 찾기 (정답 x = 1)
4
5 # 1. 설정값 (매우 단순화)
6 POP_SIZE = 10 # 개체수 10개
7 GENERATIONS = 30 # 30번 세대 교체
8 MUTATION_RATE = 0.3 # 30% 확률로 돌연변이 발생
9
10 # 임의의 초기 개체 생성 (-10.0 ~ 10.0 사이의 실수 10개)
11 population = [random.uniform(-10.0, 10.0) for _ in range(POP_SIZE)]
12
13 print("--- 초간단 유전자 알고리즘 시작 ---")
14
15 for generation in range(GENERATIONS):
16     # [평가 & 정렬] f(x) = x^2 값이 '작은 순서'대로 정렬합니다.
17     # 즉, 0번 인덱스(population[0])가 이번 세대에서 가장 우수한 정답(벨리트)이 됩니다.
18     population = sorted(population, key=lambda x: (x-1)**2)
19
20     # 현재 세대의 가장 우수한 해 출력
21     best_x = population[0]
22     print(f"[{generation + 1} 세대] 가장 우수한 x: {best_x:8.4f}, f(x): {(best_x-1)**2:8.4f}")
23
24     # 다음 세대를 저장할 리스트 (가장 똑똑한 상위 2개는 무조건 그대로 살려둠)
25     next_population = [population[0], population[1]]
26
27     # 나머지 8개의 자식을 채울 때까지 반복
28     while len(next_population) < POP_SIZE:
29         # [선택] 상위 4개(성적이 좋은 우등생들) 중에서 부모 2명을 무작위로 선택
30         parent1 = random.choice(population[:4])
31         parent2 = random.choice(population[:4])
32
33         # [교차] 두 부모의 중간값으로 자식을 만들 (부모의 성질을 닮음)
34         child = (parent1 + parent2) / 2
35
36         # [돌연변이] 일정 확률로 자식의 값을 살짝 비틀어줌 (+- 0.5 범위 내에서)
37         if random.random() < MUTATION_RATE:
38             child += random.uniform(-0.5, 0.5)
39
40         next_population.append(child)
41
42     # 세대 교체
43     population = next_population
44
45 # 최종 결과
46 final_best = population[0]
47 print("\n--- 최종 결과 ---")
48 print(f"찾아낸 최적의 x: {final_best:.4f}")
49
```

## ◆ 유전자 알고리즘 코드

```
import random
# [목적]  $f(x) = (x-1)^2$ 의 최솟값 찾기 (정답  $x = 1$ )
# 1. 설정값 (매우 단순화)
POP_SIZE = 10      # 개체수 10개
GENERATIONS = 30   # 30번 세대 교체
MUTATION_RATE = 0.3 # 30% 확률로 돌연변이 발생
# 임의의 초기 개체 생성 (-10.0 ~ 10.0 사이의 실수 10개)
population = [random.uniform(-10.0, 10.0) for _ in range(POP_SIZE)]
print("--- 초간단 유전자 알고리즘 시작 ---")
for generation in range(GENERATIONS):
    # [평가 & 정렬]  $f(x) = x^2$  값이 '작은 순서'대로 정렬합니다.
    # 즉, 0번 인덱스(population[0])가 이번 세대에서 가장 우수한 정답(엘리트)이 됩니다.
    population = sorted(population, key=lambda x: (x-1)**2)

    # 현재 세대의 가장 우수한 해 출력
    best_x = population[0]
    print(f"[{generation + 1} 세대] 가장 우수한 x: {best_x:8.4f}, f(x): {(best_x-1)**2:8.4f}")

    # 다음 세대를 저장할 리스트 (가장 똑똑한 상위 2개는 무조건 그대로 살려둬)
    next_population = [population[0], population[1]]

    # 나머지 8개의 자식을 채울 때까지 반복
    while len(next_population) < POP_SIZE:
        # [선택] 상위 4개(성적이 좋은 우등생들) 중에서 부모 2명을 무작위로 선택
        parent1 = random.choice(population[:4])
        parent2 = random.choice(population[:4])

        # [교차] 두 부모의 중간값으로 자식을 만들 (부모의 성질을 닮음)
        child = (parent1 + parent2) / 2

        # [돌연변이] 일정 확률로 자식의 값을 살짝 비틀어줌 (+/- 0.5 범위 내에서)
        if random.random() < MUTATION_RATE:
            child += random.uniform(-0.5, 0.5)

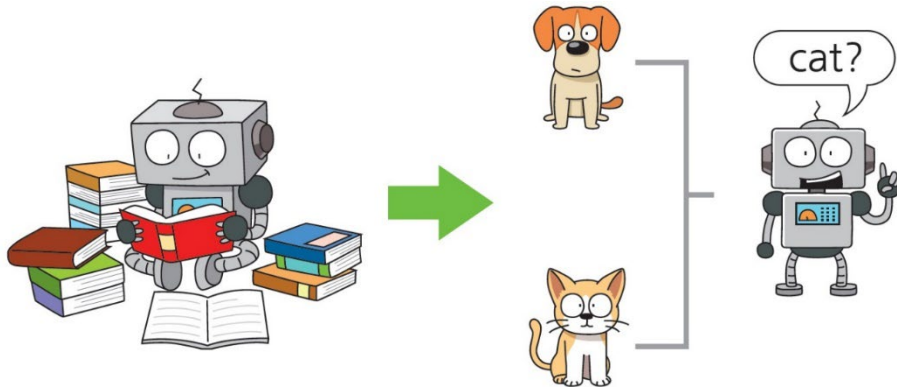
        next_population.append(child)

    # 세대 교체
    population = next_population
# 최종 결과
final_best = population[0]
print("\n--- 최종 결과 ---")
print(f"[찾아낸 최적의 x: {final_best:4f}]")
```

# 기계 학습

## ◆ 기계 학습(Machine Learning)

- 컴퓨터와 같은 기계 장치가 스스로 학습하는 것.



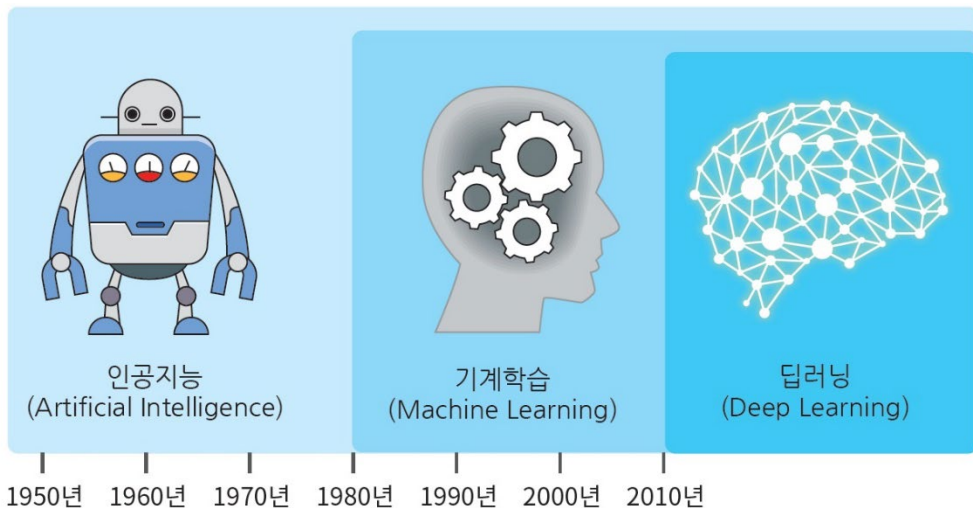
# 기계 학습

## ◆ 기계 학습 응용 분야

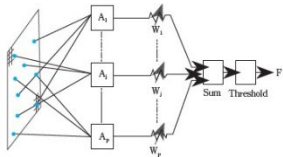


## ◆ 인공지능, 기계 학습, 딥러닝

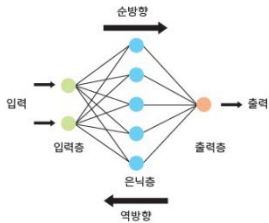
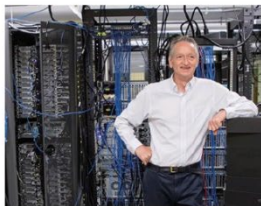
- 인공지능 분야 중에 기계 학습이 있음.
- 기계 학습 분야 중에 딥러닝이 있음.



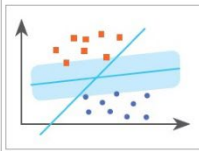
## ◆ 기계 학습의 역사



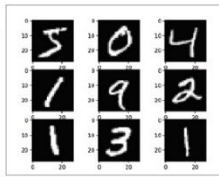
1957 퍼셉트론



1985 역전파 학습 알고리즘



1995 SVM

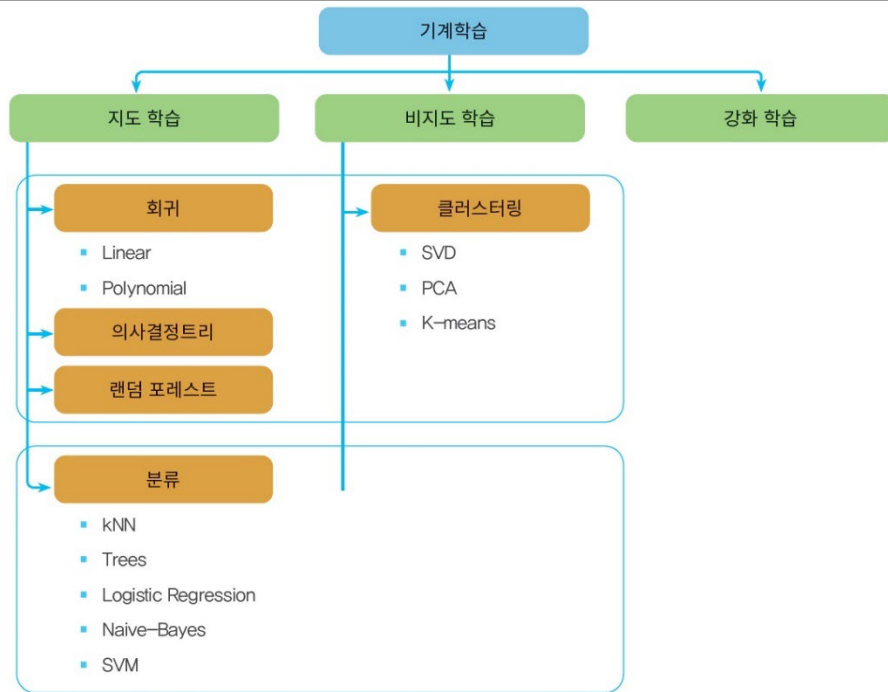


1998 필기체 인식



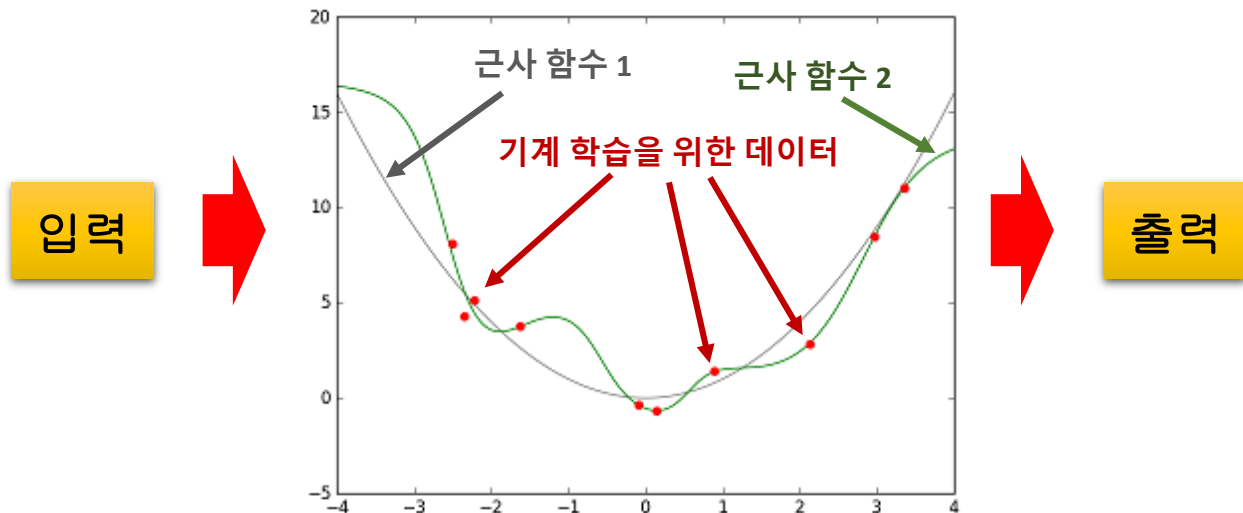
2012  
ImageNet에서  
우승

## ◆ 기계 학습의 종류



# 기계 학습

- ◆ 기계 학습은 입력과 출력에 대한 함수  $y = f(x)$ 를 학습한다고 볼 수 있다.
  - 즉, 기계 학습은 함수 근사(Function Approximation)의 일종이다.
  - 여기서, 근사된 함수가 바로 인공지능의 **학습 모델**이다.



## ◆ 샘플(Sample)

- 근사 함수  $y = f(x)$ 에서  $x$ 변수에 해당
- 학습 모델에 입력되는 값
- 예, 이미지 인식 AI에서 입력되는 사진

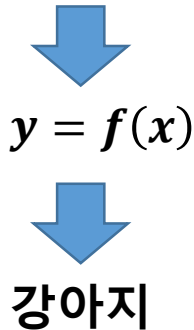
$x$ :



## ◆ 레이블(Label)

- 근사 함수  $y = f(x)$ 에서  $y$ 변수에 해당
- 학습 모델에서 출력되는 값
- 예, 이미지 인식 AI에서 출력되는 결과

$y$ :



## ◆ 학습 데이터와 실험 데이터

- 학습 데이터: 함수 모델을 찾을 때 사용되는 데이터
- 실험 데이터: 함수 모델의 성능을 분석할 때 사용되는 데이터

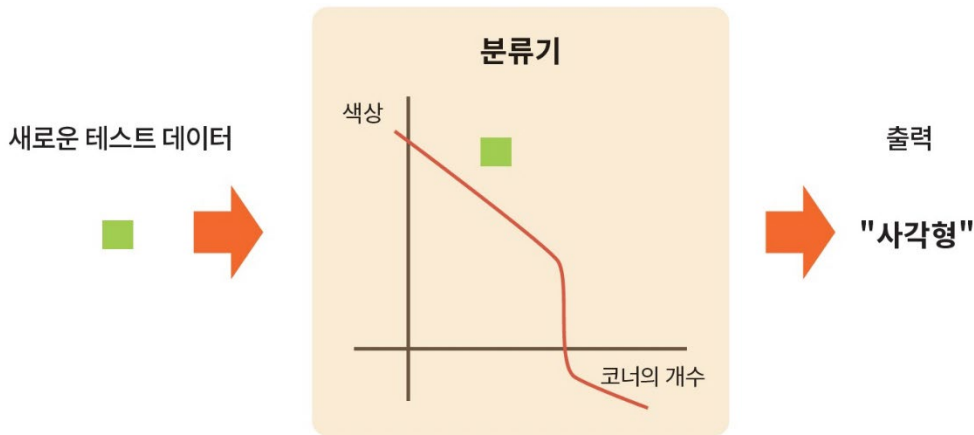
## ◆ 학습(Learning) 과정

- 학습 데이터를 이용하여 **함수 모델**을 찾음.
- 수집된 데이터를 통해 모델(패턴)을 찾는 과정.



## ◆ 실험 과정 혹은 예측(Prediction) 과정

- 학습 모델을 이용하여 실험 데이터에 대한 예측 결과를 출력.
- 사람이 인식한 결과와 학습 모델의 예측 결과를 비교하여 성능 분석 가능.



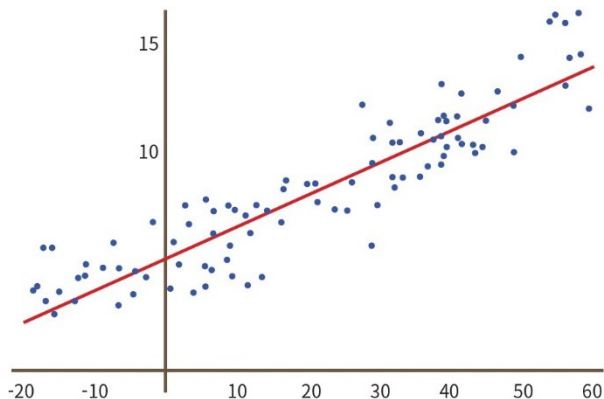
# 기계 학습

## ◆ 지도 학습(Supervised Learning)

- 지도 학습에는 회귀와 분류 기법이 있다.

## ◆ 회귀(Regression)

- 실수(Real Number) 입력과 실수(Real Number) 출력이 주어질 때, 입력과 출력의 관계를 가장 잘 표현하는 수학 함수를 찾는 방법.



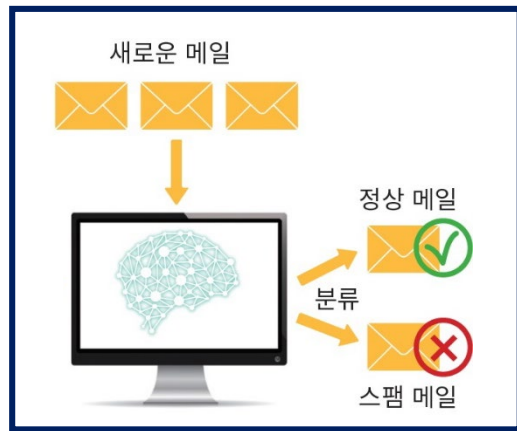
$$y = f(x)$$

## ◆ 분류(Classification)

- 입력되는 데이터가 A인지 B인지, 아니면 C인지를 인식하는 것.
- 분류에 대한 레이블(A, B, C)을 제공해야 한다.



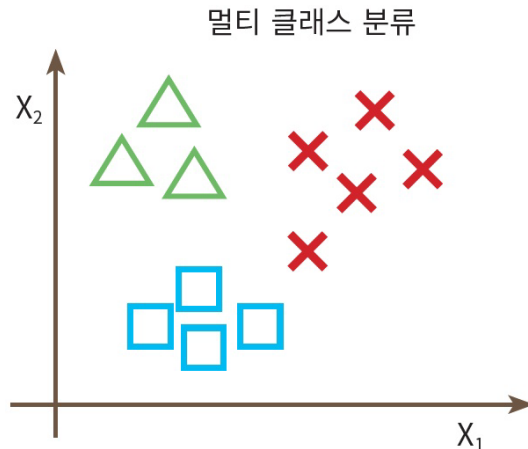
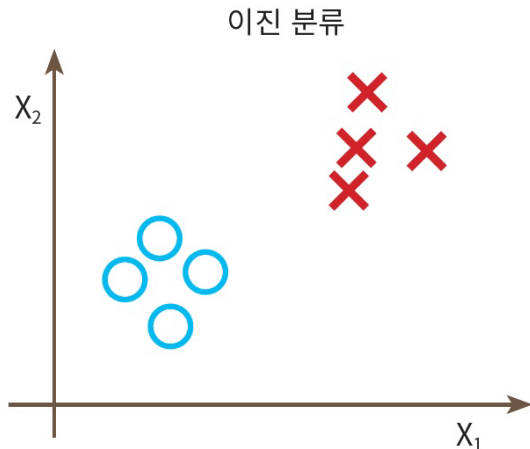
학습 과정



실험 과정

## ◆ 분류(Classification)

- 일반적인 알고리즘: kNN(k-Nearest Neighbor), SVM(Support Vector Machine) 등
- 두 개를 분류하는 이진 분류와 세 개 이상을 분류하는 멀티 클래스 분류가 있다.



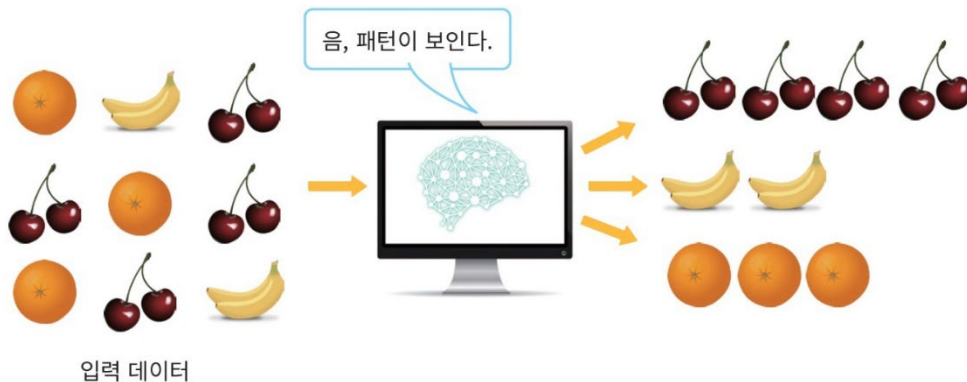
## ◆ 분류 알고리즘 체험

<https://transcranial.github.io/keras-js/#/mnist-cnn>

The screenshot shows the Keras.js Basic Convnet for MNIST demo interface. On the left is a sidebar with the Keras.js logo and a list of demos: Basic Convnet (MNIST), Convolutional VAE (MNIST), AC-GAN (MNIST), ResNet-50 (ImageNet), Inception v3 (ImageNet), DenseNet-121 (ImageNet), SqueezeNet v1.1 (ImageNet), Bidirectional LSTM (IMDB), and Image Super-Resolution. Below the demos are links to the GitHub repo and MD.ai, and a contact section. The main area has a green header 'Basic Convnet for MNIST' with an info icon. Below the header is a drawing area with the text 'Draw any digit (0-9) here' and a green box containing a handwritten digit '7'. To the right of the drawing area is a 'use GPU' toggle switch (checked) and a 'CLEAR' button. Below the drawing area is a bar chart showing the probability distribution for digits 0-9, with the bar for '7' being the highest. At the bottom, there is a 'Conv2D' section with the text '32 3x3 filters, padding valid, 1x1 strides' and a grid of 20 small images showing the digit '7' with various filters applied.

## ◆ 비지도 학습(Unsupervised Learning)

- 선생님없이 컴퓨터(기계)가 스스로 입력들을 분류하는 것을 의미.
- 레이블( $y$ )이 주어지지 않는 기계 학습.
- 방법: 기계가 입력 값들의 상관도를 분석하여 유사한 데이터들을 스스로 모으고 학습.

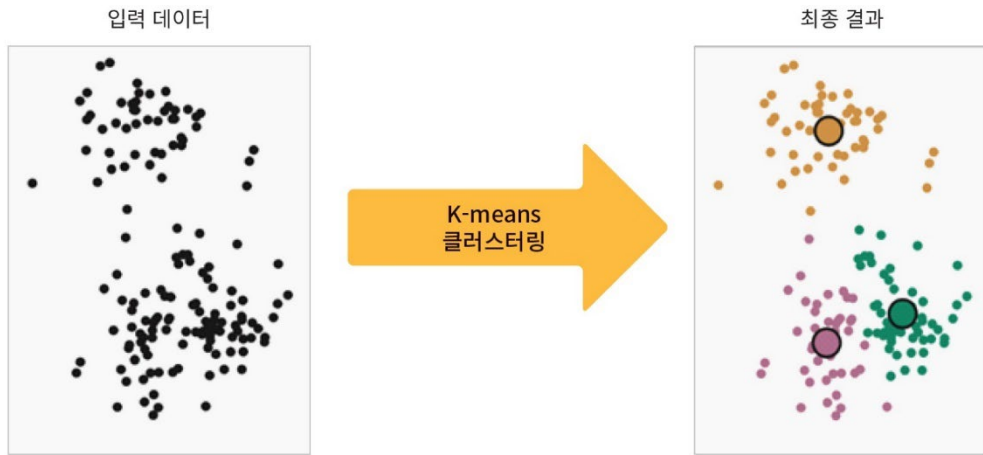


지도 학습: 체리, 바나나, 오렌지를 미리 알려줌. → 과일 이름을 알게 됨.

비지도 학습: 과일 이름을 알려주지 않음. → 과일 이름은 모르나 비슷한 것을 그냥 모으는 것.

## ◆ 비지도 학습(Unsupervised Learning)

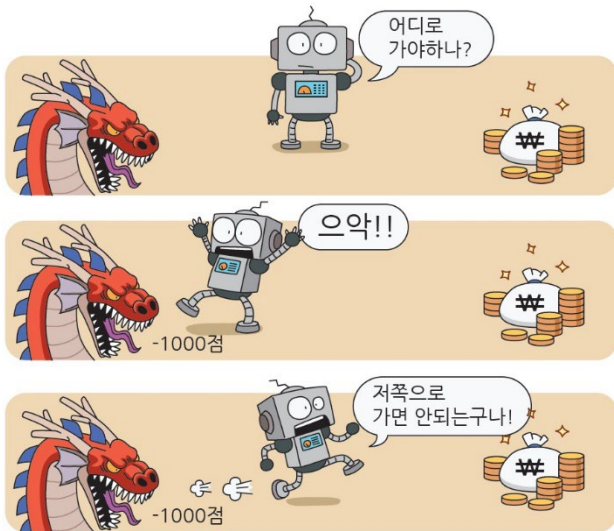
- 클러스터링(군집화, Clustering) 기법을 이용.
- 클러스터링이란 학습 데이터 간 거리를 계산하여 몇 개의 그룹으로 나누는 방식
- 대표적인 클러스터링 알고리즘: k-Means



입력 데이터를 3개의 카테고리로 분류

## ◆ 강화 학습(Reinforcement Learning)

- 컴퓨터(기계)가 어떤 행동을 할 때마다 외부에서 처벌이나 보상을 주는 것.
- 컴퓨터(기계)는 처벌을 피하고 보상을 많이 얻을 수 있는 전략을 스스로 터득.



속담(격언)

매가 약이다.

칭찬은 고래도 춤추게 한다.

즉, 당근과 채찍

