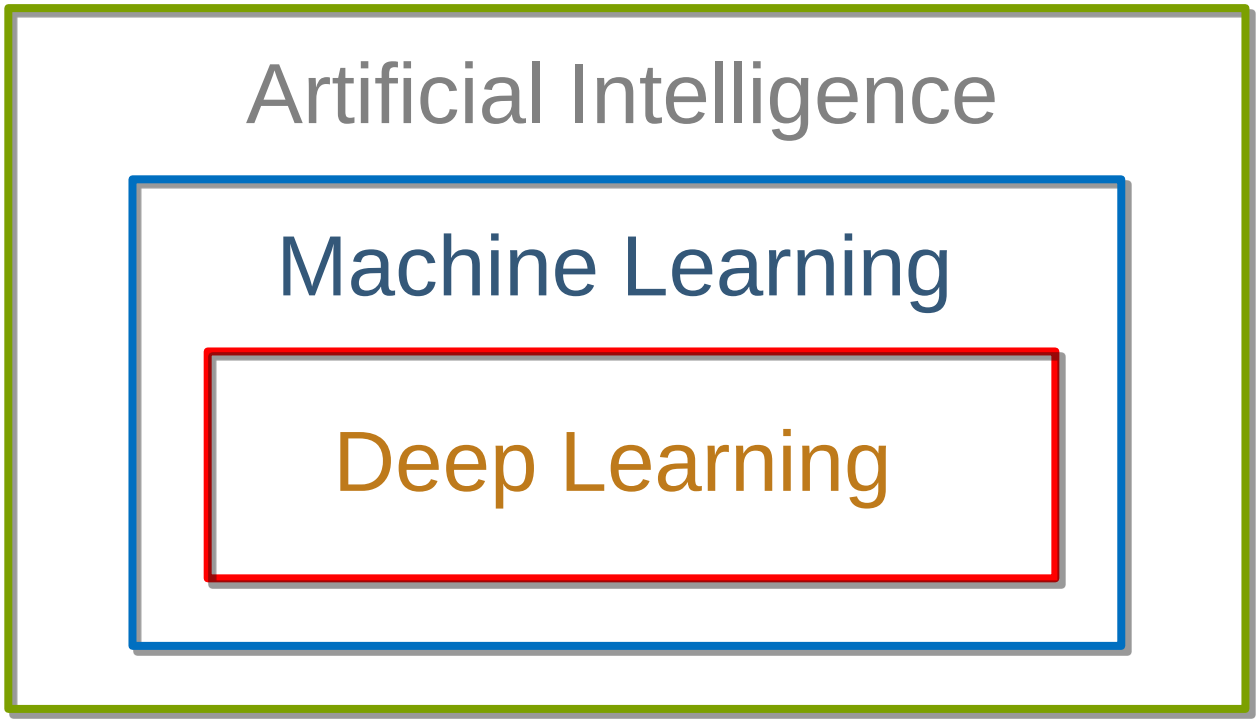
1. 딥러닝의 개요

머신러닝의 특성

- 인공지능, 머신러닝, 딥러닝
  - 기계가 학습을 할 수 있도록 하는 연구 분야
  - 인공지능 연구의 한 분야로서 최근들어 딥러닝을 통해서 빠르게 발전



머신러닝의 특성

- 학습 시스템과 머신러닝

- **학습 시스템**: 환경과의 상호작용으로부터 획득한 경험적인 데이터를 바탕으로 지식을 자동으로 습득하여 스스로 성능을 향상하는 시스템
- **머신러닝**: 인공적인 학습 시스템을 연구하는 과학과 기술. 즉, 경험적인 데이터를 바탕으로 지식을 자동으로 습득하여 스스로 성능을 향상하는 기술
  - 데이터를 기반으로 모델을 자동으로 생성하는 기술
  - 실세계의 복잡한 데이터로부터 규칙과 패턴을 발견하여 미래를 예측하는 기술
  - 입출력 데이터로부터 프로그램(알고리듬)을 자동으로 생성하는 기술

머신러닝의 특성

- 머신러닝의 정의
  - 머신러닝을 보다 형식화하여 정의하면 "환경(Environment, E)과의 상호작용을 통해서 축적되는 경험적인 데이터(Data, D)를 바탕으로 지식 즉 모델(Model, M) 을 자동으로 구축하고 스스로 성능(Performance, P)을 향상하는 시스템"이다 (Mitchell, 1997).

$$ML : D \xrightarrow{P} M$$

머신러닝의 특성

- 딥러닝
  - 많은 수의 신경층을 쌓아 입력된 데이터가 여러 단계의 특징 추출 과정을 거쳐 자동으로 고수준의 추상적인 지식을 추출하는 방식
  - 특징 추출과 특징 분류를 특징 학습의 문제로 통합(보다 자동화된 학습 기술)
  - 딥러닝을 통해서 신경망에 대한 관심이 다시 늘어남에 따라 머신러닝 연구에 관심을 다시 갖기 시작함
- 다양한 분야에의 응용
  - Google의 GoogLeNet, 백만여 장의 이미지로부터 천 가지 종류의 물체를 분류
  - Facebook의 DeepFace, 사람의 얼굴을 인식하는 문제에서 인간 수준의 성능
  - Microsoft의 딥러닝을 적용한 음성인식 기술
  - Google의 DeepMind, 실제 사람처럼 비디오 게임을 학습하는 기술

프로그래밍 방식과의 차이점

- 일반적인 컴퓨터 프로그램
  - 사람이 알고리듬 설계 및 코딩
  - 주어진 문제(데이터)에 대한 답 출력

- 머신러닝 프로그램
  - 사람이 코딩
  - 기계가 알고리듬을 자동 프로그래밍 (Automatic Programming)
  - 데이터에 대한 프로그램을 출력

프로그래밍 방식과의 차이점

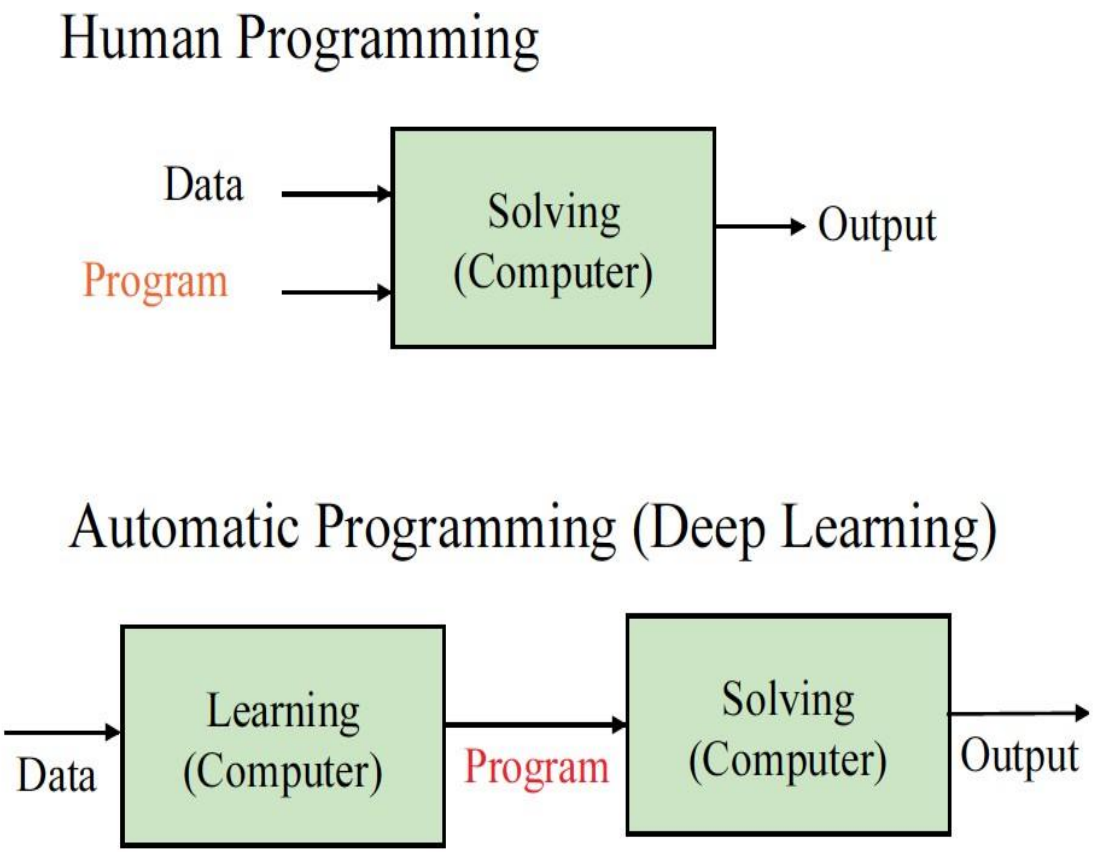
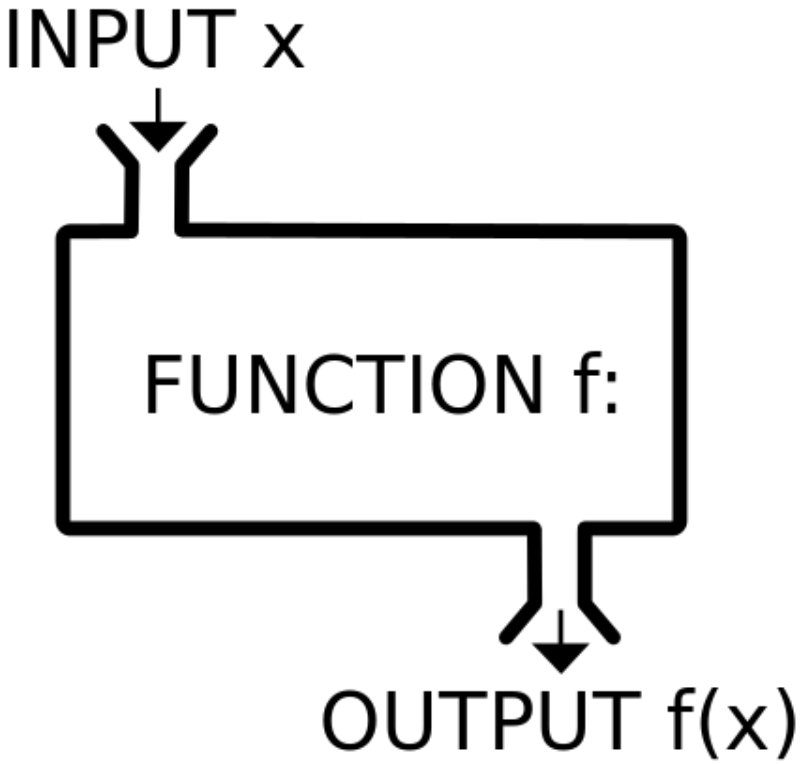


그림 1.2 Human Programming과 Automatic Programming의 차이점

프로그래밍 방식과의 차이점

- Actually….
  - 입력값으로부터 결과 값을 내는 함수를 만들어 내는 것



프로그래밍 방식과의 차이점

- 이런 함수는 어떻게 만들 수 있을까?

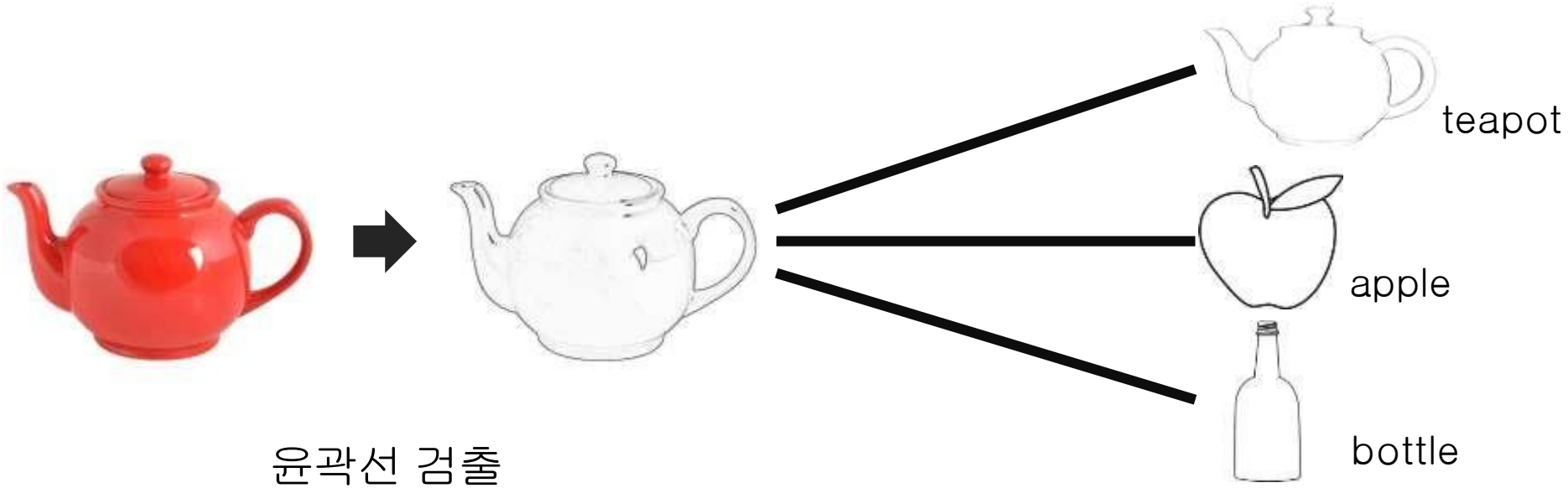
```
public static string WhatIsIt ( Bitmap input ) {

  // ? ? ? ?

  return nameOfIt;
}
```

input = 

비트맵 이미지를 입력 받아 무엇인지 출력하는 함수

프로그래밍 방식과의 차이점

- 예를 들면 이렇게 해볼 수 있다



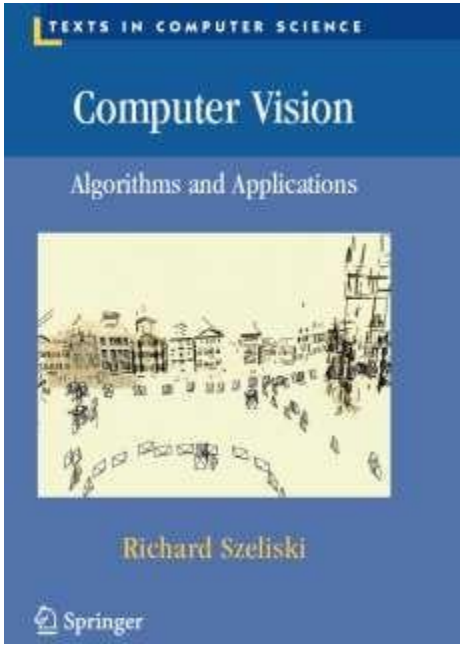
윤곽선 검출
Edge Detection

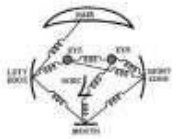
윤곽선 템플릿들

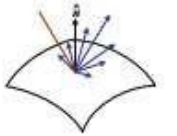
윤곽선 템플릿을 겹쳐보았을 때 얼마나 잘 겹치는지를 비교하여 제일 잘 겹친 것을 선택!

프로그래밍 방식과의 차이점

- 컴퓨터 비전에서 배우는 내용







앞 페이지에서 말한 내용을 포함한 각종 기하학적 전-후처리 동원

프로그래밍 방식과의 차이점

- 성능 올리기가 쉽지 않다

반대로 돌아간 teapot은?

무늬가 있는 teapot은?

특이하게 생긴 teapot은?

배경과 섞여 있는 특이한 teapot은?

사람은?

고양이는?

선배들이 매우 다양한 방법을 사용해봤지만, 은탄환은 없었다

프로그래밍 방식과의 차이점

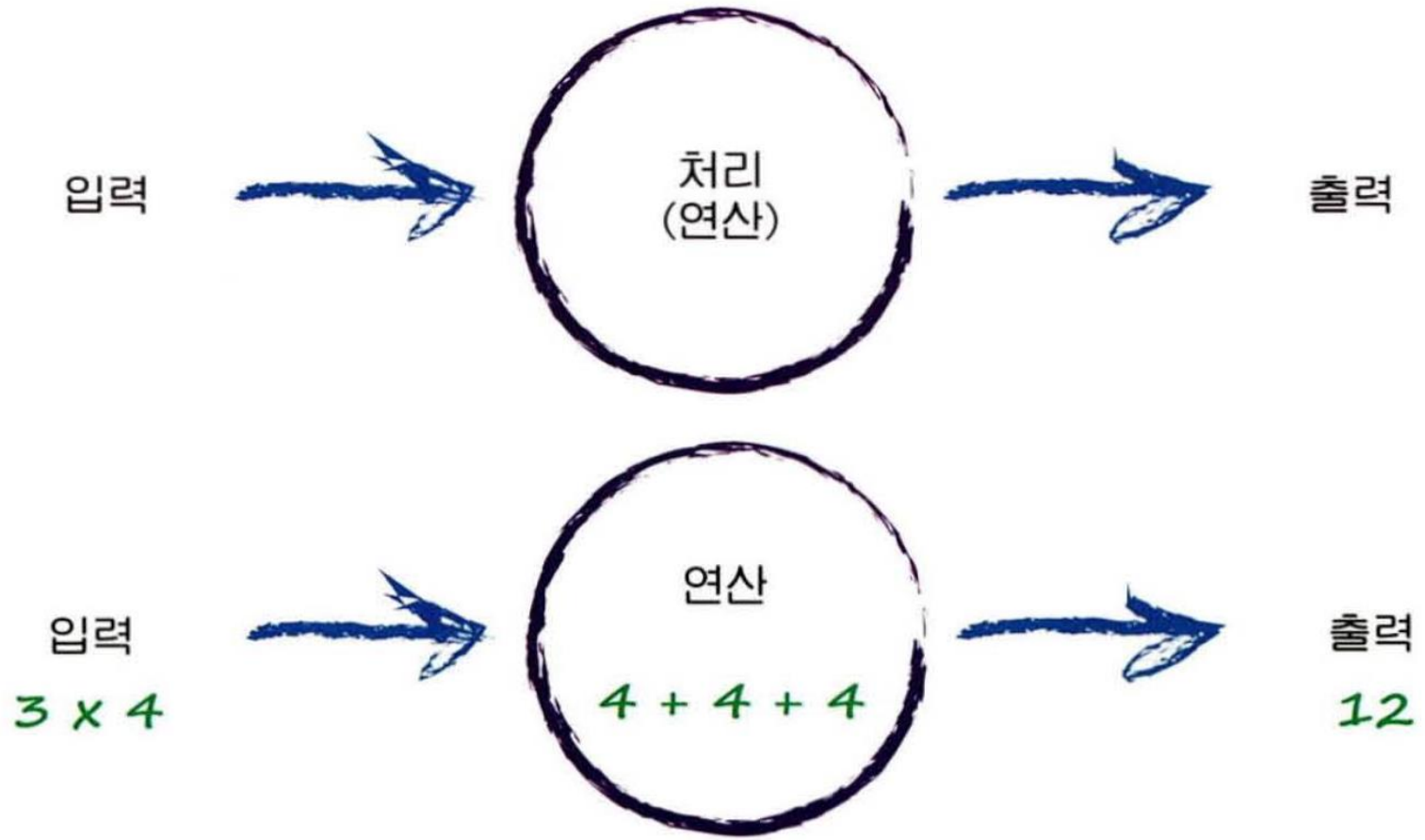
- 기계 학습 : 데이터로부터 임의의 함수를 도출해보자!
  - 지도 학습 (supervised learning)
    - 입력에 대한 정답값이 있음. 대조하며 학습
    - Classfication
  - 비지도 학습 (unsupervised learning)
    - 정답 없이 입력값들만 주어짐
    - clustering, autoencoder
  - 강화 학습 (reinforcement learning)
    - 입력값(액션)에 대한 평가만 주어짐
    - Q-learning

프로그래밍 방식과의 차이점

- 머신 러닝의 학습과정
  - 주어진 데어터를 기반으로 해서 미지의 데이터를 예측하는 식(모델)을 생각한다.
  - 모델에 포함된 파라미터의 좋고 나쁨을 판단하는 오차 함수를 준비한다.
  - 오차 함수를 최소화할 수 있도록 파라미터 값을 결정한다.

간단한 예측자

- 일반적인 프로그래밍 절차



간단한 예측자

- 킬로미터를 마일로 변환하는 장치
  - 킬로미터를 마일로 변환하는 공식을 모른다고 가정.
  - 다만, 킬로미터와 마일의 관계가 선형(linear) 라는 것은 알고 있음.
  - 따라서, 다음과 같은 모델을 가정할 수 있다.

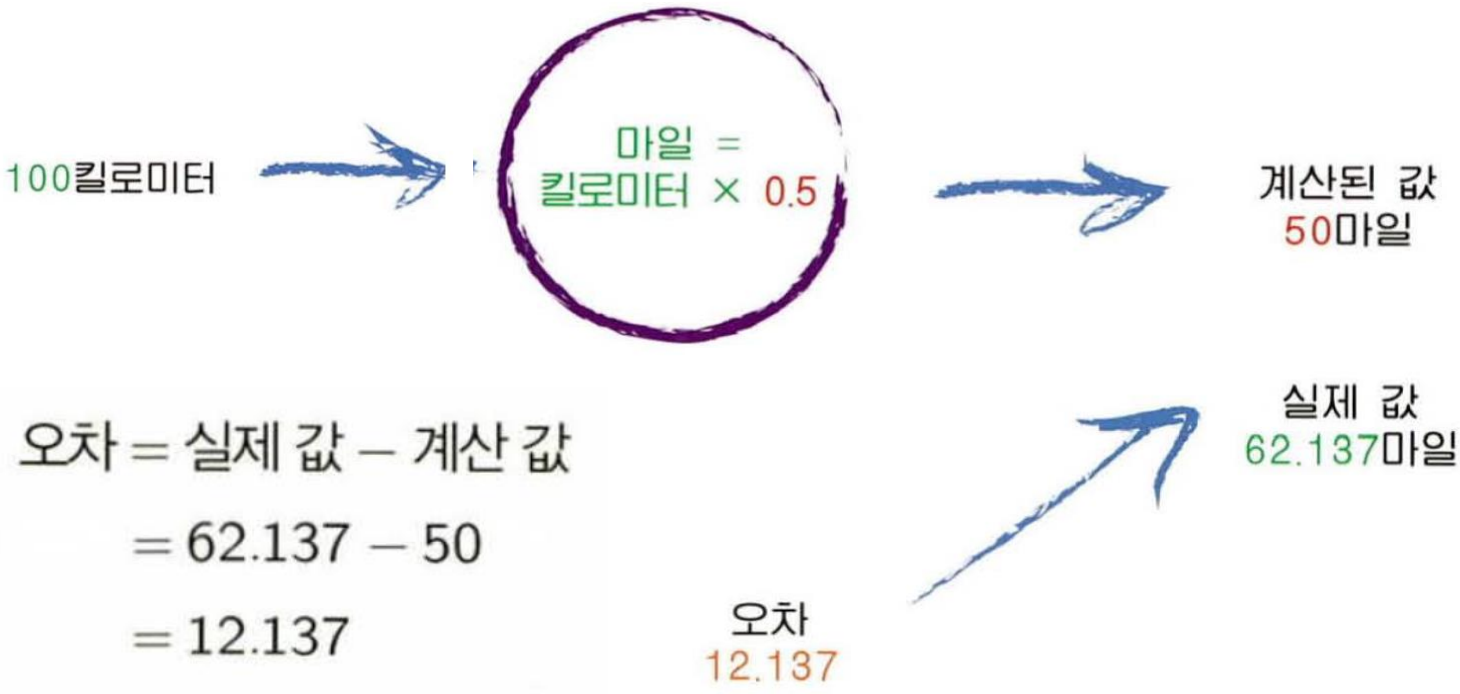
$$Mile = kilometer \times c \quad (c : \text{constant})$$

  - C 값을 알기 위해서는 현실 세계에서의 측정 값이 필요

실제 사례	킬로미터	마일
1	0	0
2	100	62.137

간단한 예측자

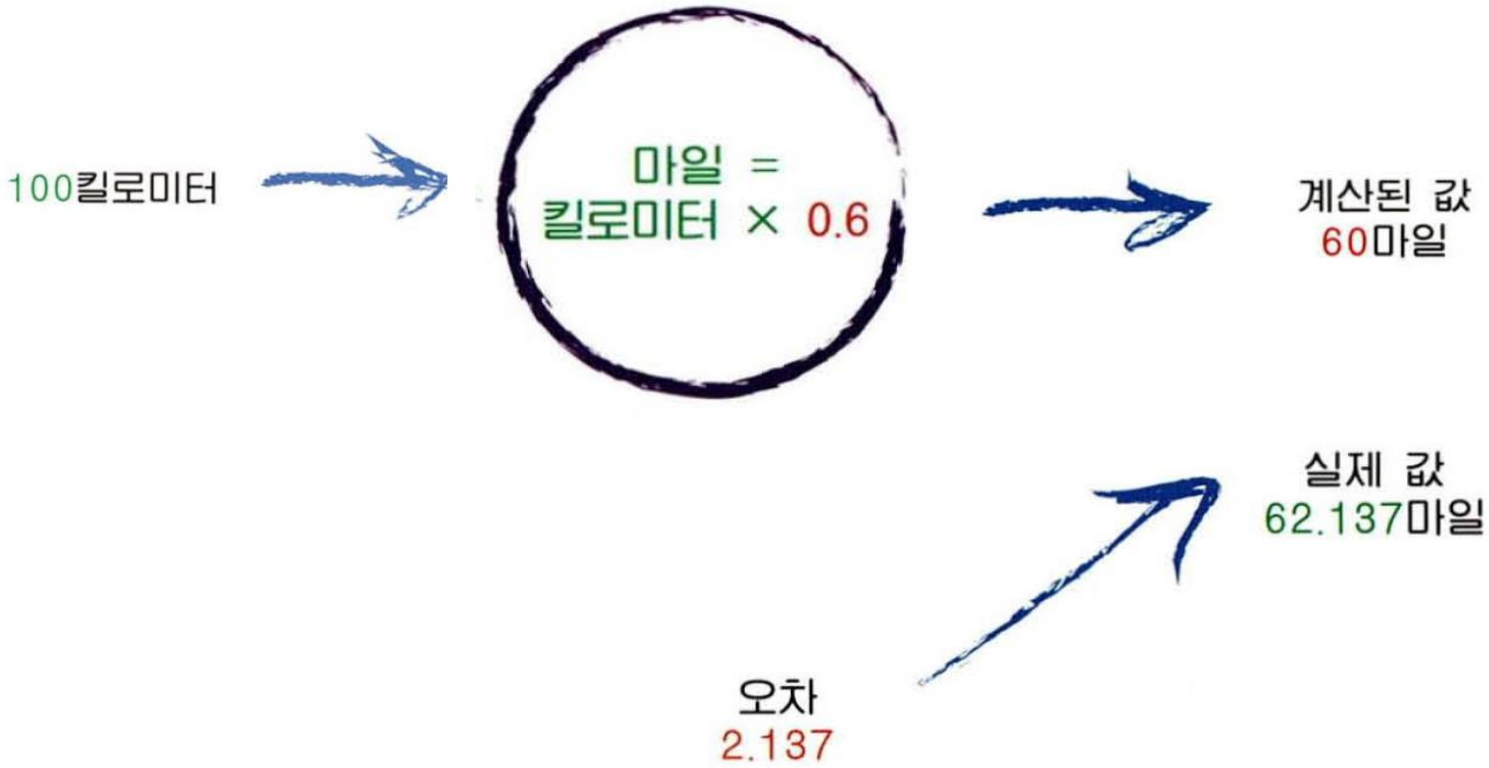
- 킬로미터를 마일로 변환하는 장치
  - C에 임의의 값을 대입 ⇒ c = 0.5



$$\begin{aligned} \text{오차} &= \text{실제 값} - \text{계산 값} \\ &= 62.137 - 50 \\ &= 12.137 \end{aligned}$$

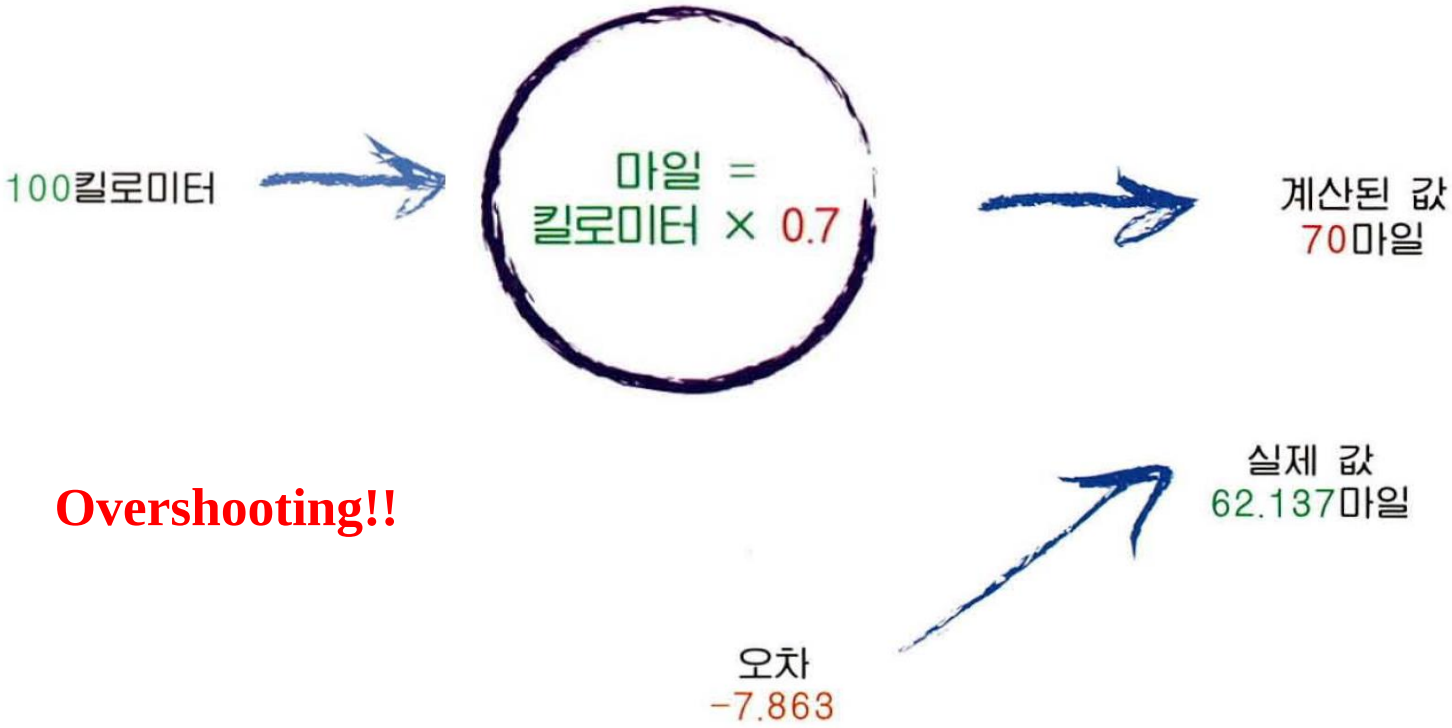
간단한 예측자

- 킬로미터를 마일로 변환하는 장치
  - C에 조금 더 증가된 값을 대입 ⇒ c = 0.6



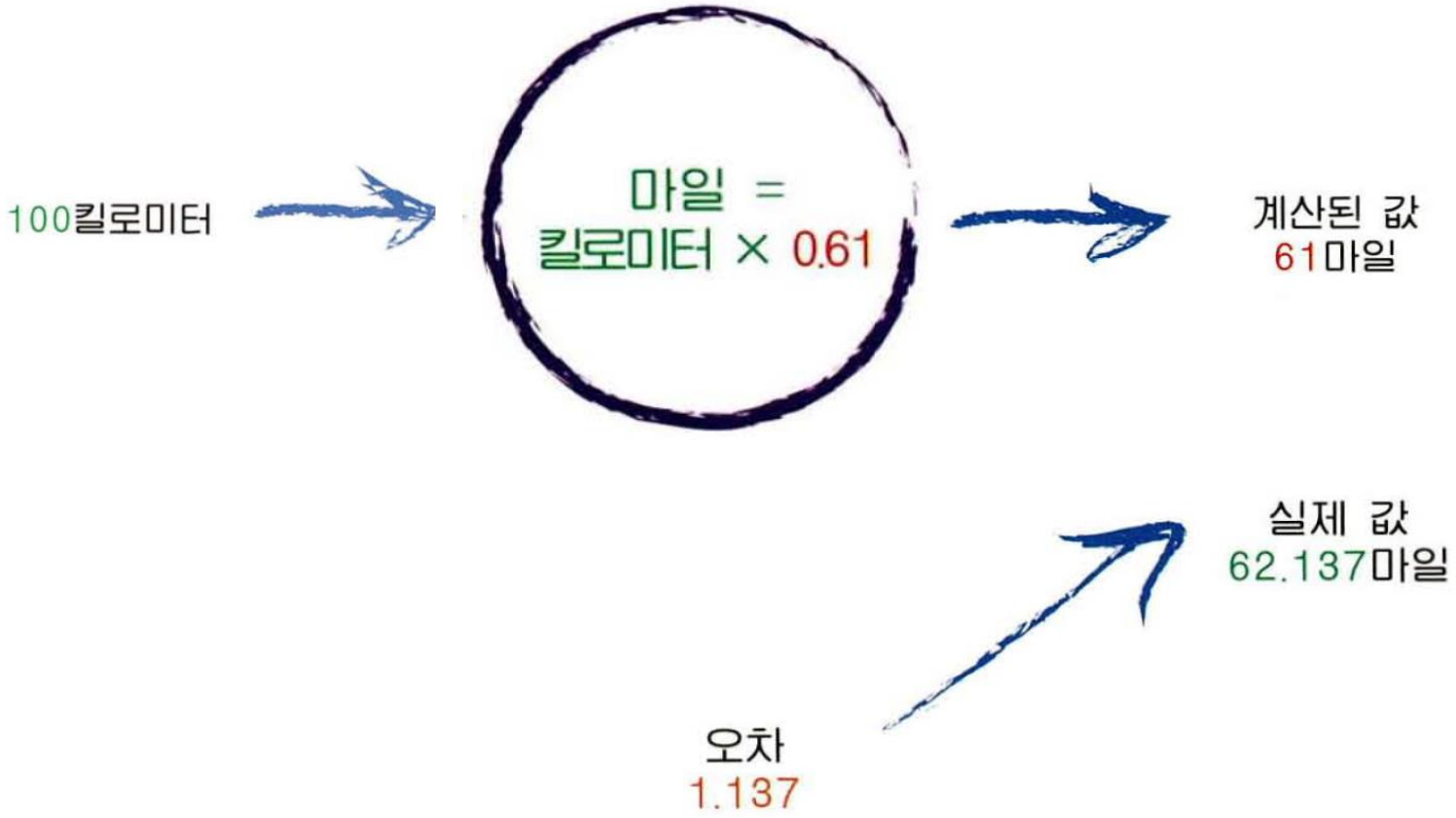
간단한 예측자

- 킬로미터를 마일로 변환하는 장치
  - C에 조금 더 증가된 값을 대입 ⇒ c = 0.7



간단한 예측자

- 킬로미터를 마일로 변환하는 장치
  - C에 0.6과 0.7 사이의 값을 대입 ⇒ c = 0.61

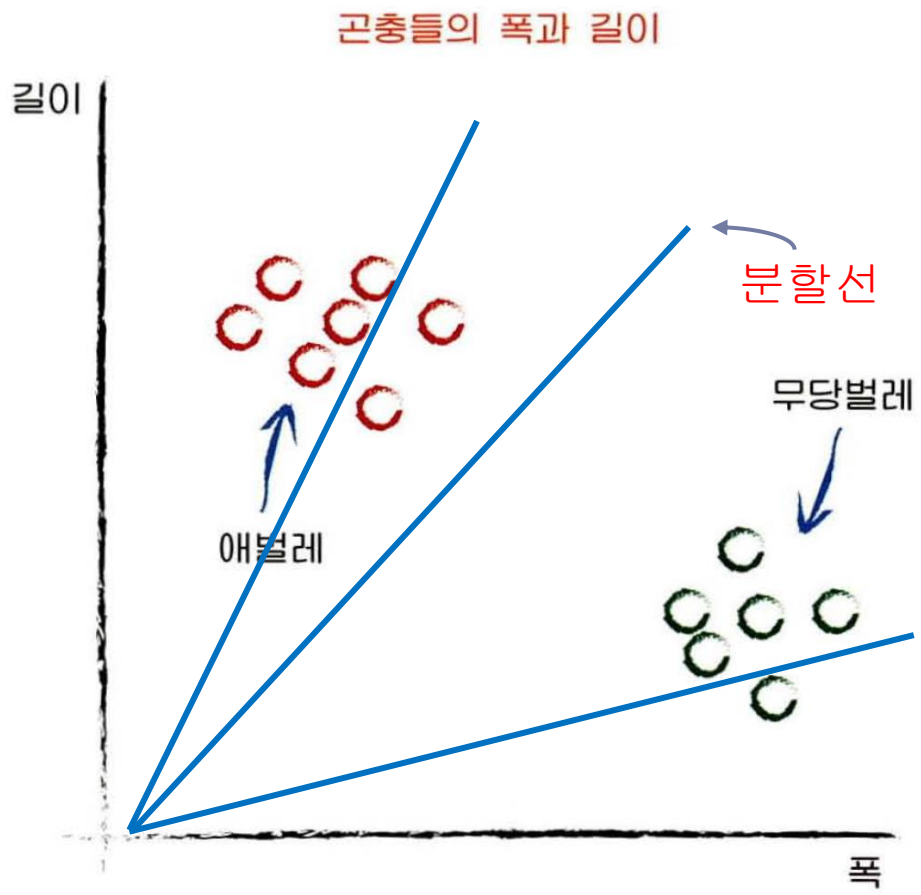


간단한 예측자

- 수학에서의 반복법(iteration method)
  - 미지수에 임의의 값을 대입하여 오차를 구함으로써 이를 개선해 나가는 방법
  - 머신러닝도 동일한 방식으로 모델을 학습함.

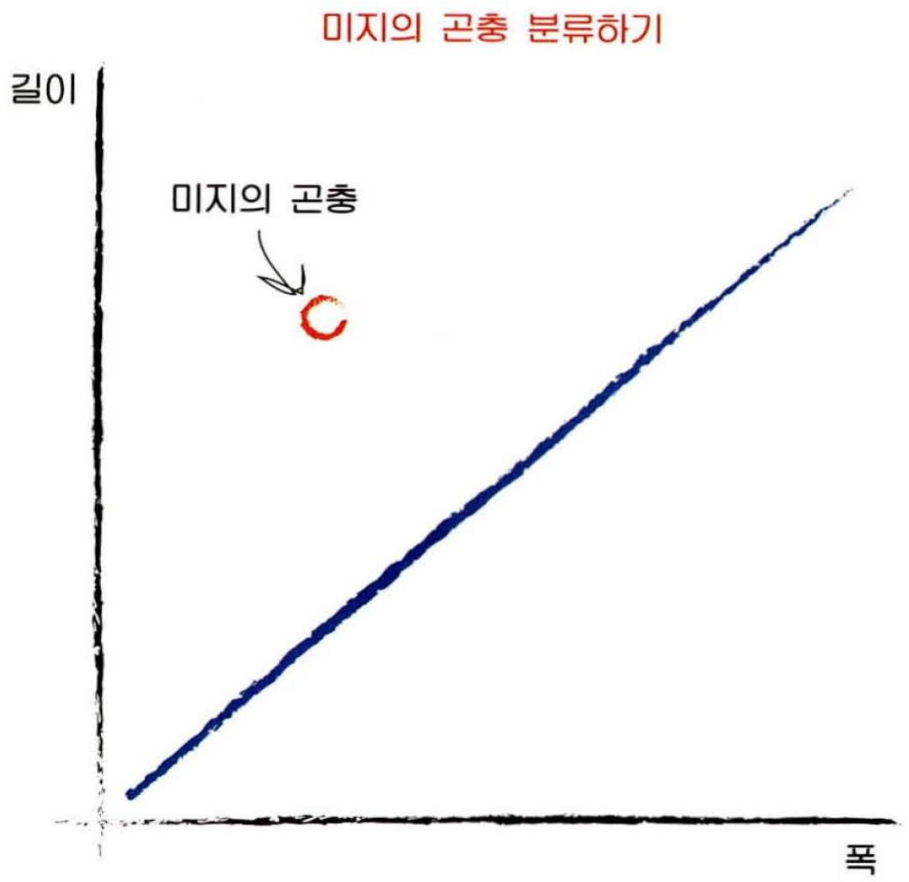
분류는 예측과 그다지 다르지 않다

- 예측자의 핵심은 선형함수
- 그렇다면 분류자는??



분류는 예측과 그다지 다르지 않다

- 미지의 곤충을 분류
  - 이 곤충의 폭과 길이는 이미 측정



분류는 예측과 그다지 다르지 않다

- 직선을 이용하면 두 종류의 데이터를 분류가 가능
- 즉, 선형함수를 이용하면 미지의 데이터를 분류가 가능
- 단, 선형함수의 기울기를 어떻게 구해야 하는가?
- 선형함수의 기울기를 구하는 것이 바로 머신러닝의 핵심.

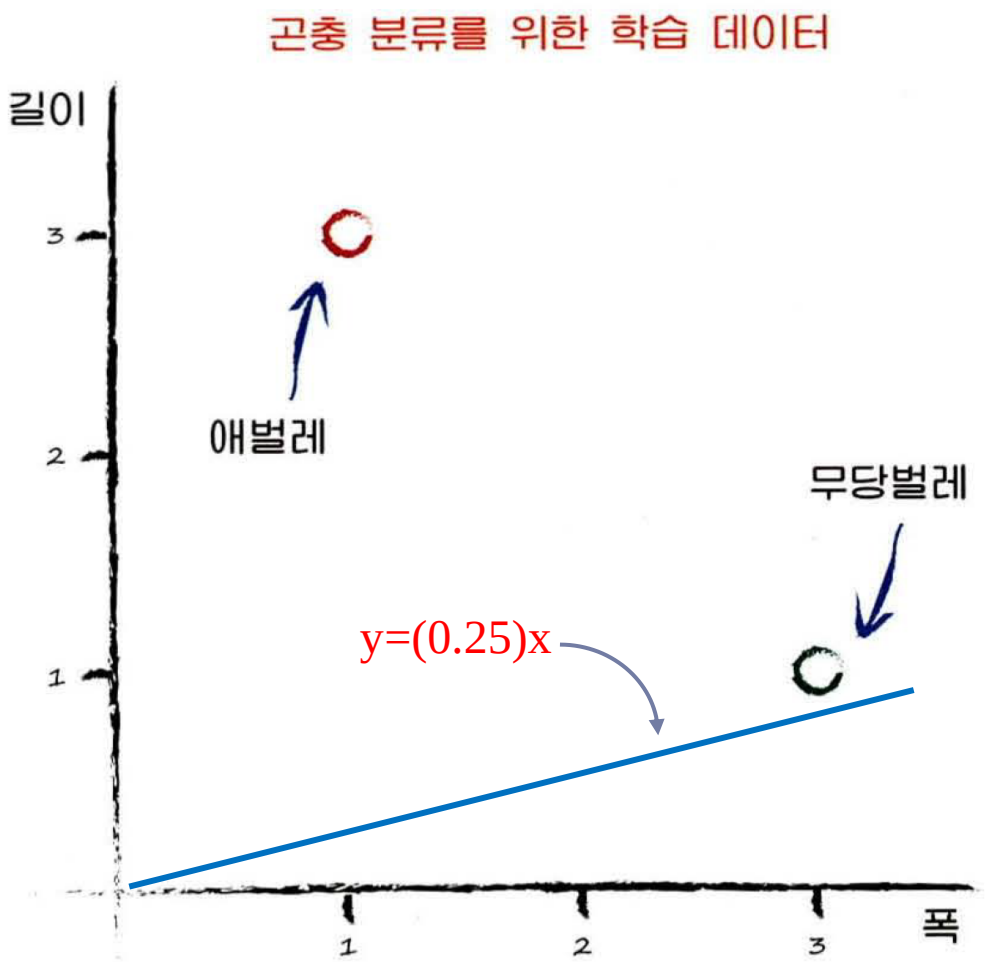
분류자 학습시키기

- 애벌레와 무당벌레를 잘 분류할 수 있도록 선형분류자를 학습
  - 2개의 그룹을 분리하는 직선의 기울기를 어떻게 결정하는가?
  - y=Ax
- 학습데이터
  - 예측자 또는 분류자에게 학습을 하기 위한 실제 값을 알려주는 역할

예제	폭	길이	곤충
1	3.0	1.0	무당벌레
2	1.0	3.0	애벌레

분류자 학습시키기

- A = 0.25라고 가정

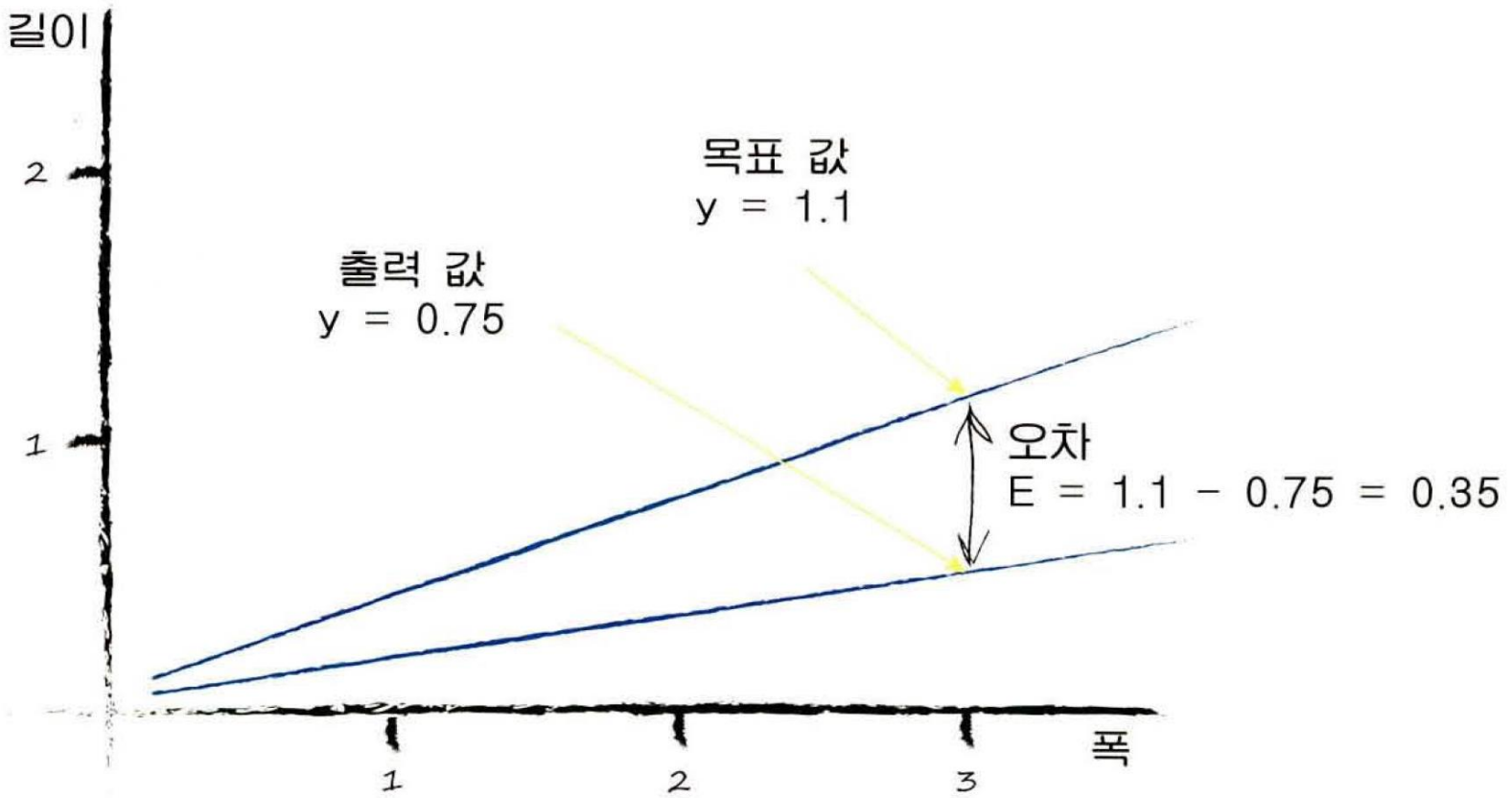


분류자 학습시키기

- 첫번째 학습데이터
  - x=3.0, y=1.0
- A = 0.25라고 가정
- y=0.25*3.0=0.75
- 무당벌레와 애벌레를 분리하기 위해서는 직선이 무당벌레의 위치 (3.0, 1.0)보다 위를 통과해야 함.
- 또한, 직선이 애벌레의 위치인 (1.0, 3.0)보다 아래에 위치해야 함.
- 따라서, 본 예제에서는 x=3.0일 때 y=1.1이 되는 것을 목표로 함.
  - y의 값은 1.1~8.9 사이의 값을 취하면 어떤 값이든 무방.

분류자 학습시키기

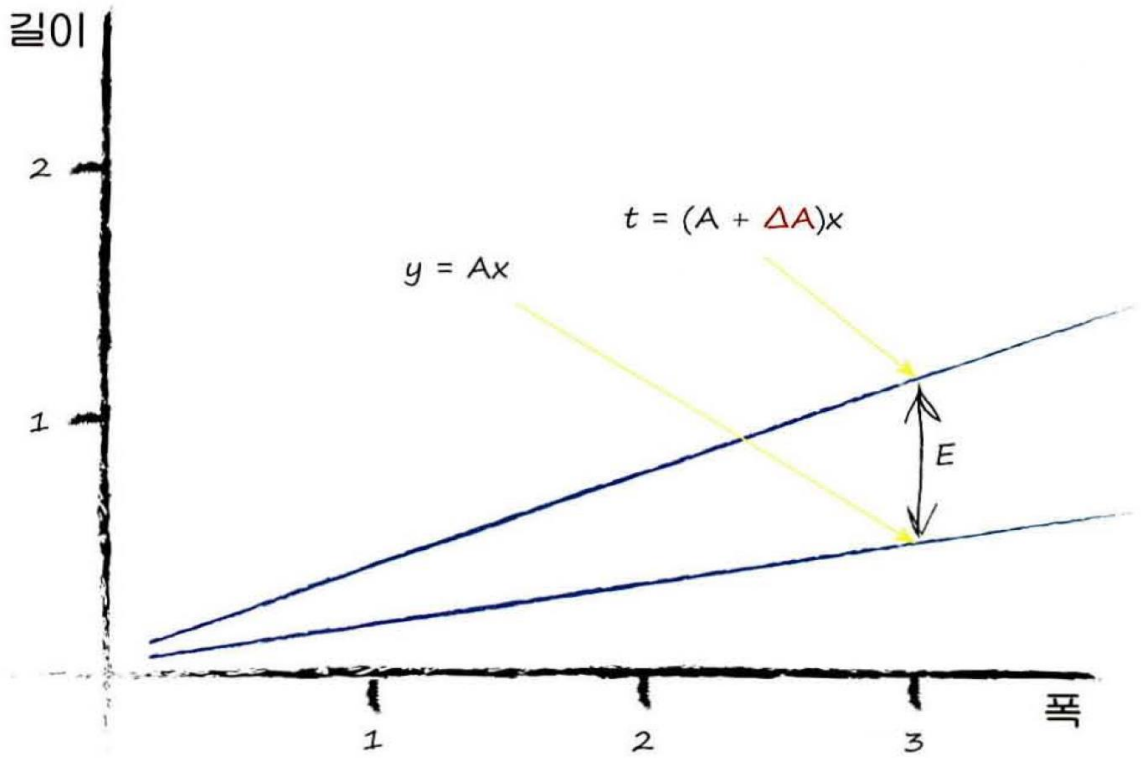
- 오차 = 목표 값 – 실제 값
  - E = 1.1 – 0.75 = 0.35



분류자 학습시키기

- 오차, E 값은 매개변수 A와 어떠한 관계를 갖는가?
- 우리가 얻고자 하는 목표값을 t라고 한다면 다음과 같은 식을 정의할 수 있다.

$$t = \left(A + \Delta A\right) x$$



분류자 학습시키기

- 따라서, 오차 E는 다음과 같이 정의할 수 있다.

$$E = t - y$$
$$= (A + \Delta A)x - Ax$$
$$= Ax + \Delta Ax - Ax$$
$$= \Delta Ax$$

- 머신러닝의 목표는 분류를 더 잘할 수 있도록 직선의 기울기 A를 개선하는 것.
- 따라서, 기울기 A의 변화량을 오차 E에 대해 표현하면 다음과 같이 정의할 수 있다.

$$\Delta A = E / x$$

- 머신러닝의 학습은 오차 E 값에 기초해 기울기 A를 ΔA씩 업데이트하면서 최적의 기울기 A를 찾는 것이다.

분류자 학습시키기

- 예제에서의 기울기 업데이트
- 오차는 0.35, x는 3.0이므로 ΔA는 다음과 같이 구할 수 있다
  .
$$\Delta A = E / x = 0.35 / 3.0 = 0.1167$$
- 이에 따라 업데이트를 하면 새로운 A값은 다음과 같다.
$$A_{new} = A + \Delta A = 0.25 + 0.1167 = 0.3667$$
- 새로운 A값을 y에 적용하면 다음과 같다.
$$y = Ax = 0.3667 * 3.0 = 1.1$$

분류자 학습시키기

- 예제의 두번째 학습데이터
  - x=1.0, y=3.0
- A=3.667로 업데이트한 선형함수에 x=1.0을 대입하면 출력값은 다음과 같다.

$$y = 0.3667 * 1.0 = 0.3667$$

- 학습데이터의 실제 y값과 차이가 큼
- 이번에는 목표값을 2.9로 설정하면 직선이 애벌레를 지나지 않고 바로 밑에 위치하게 됨.
- 이 경우 오차는 다음과 같다.

$$E = 2.9 - 0.3667 = 2.5333$$

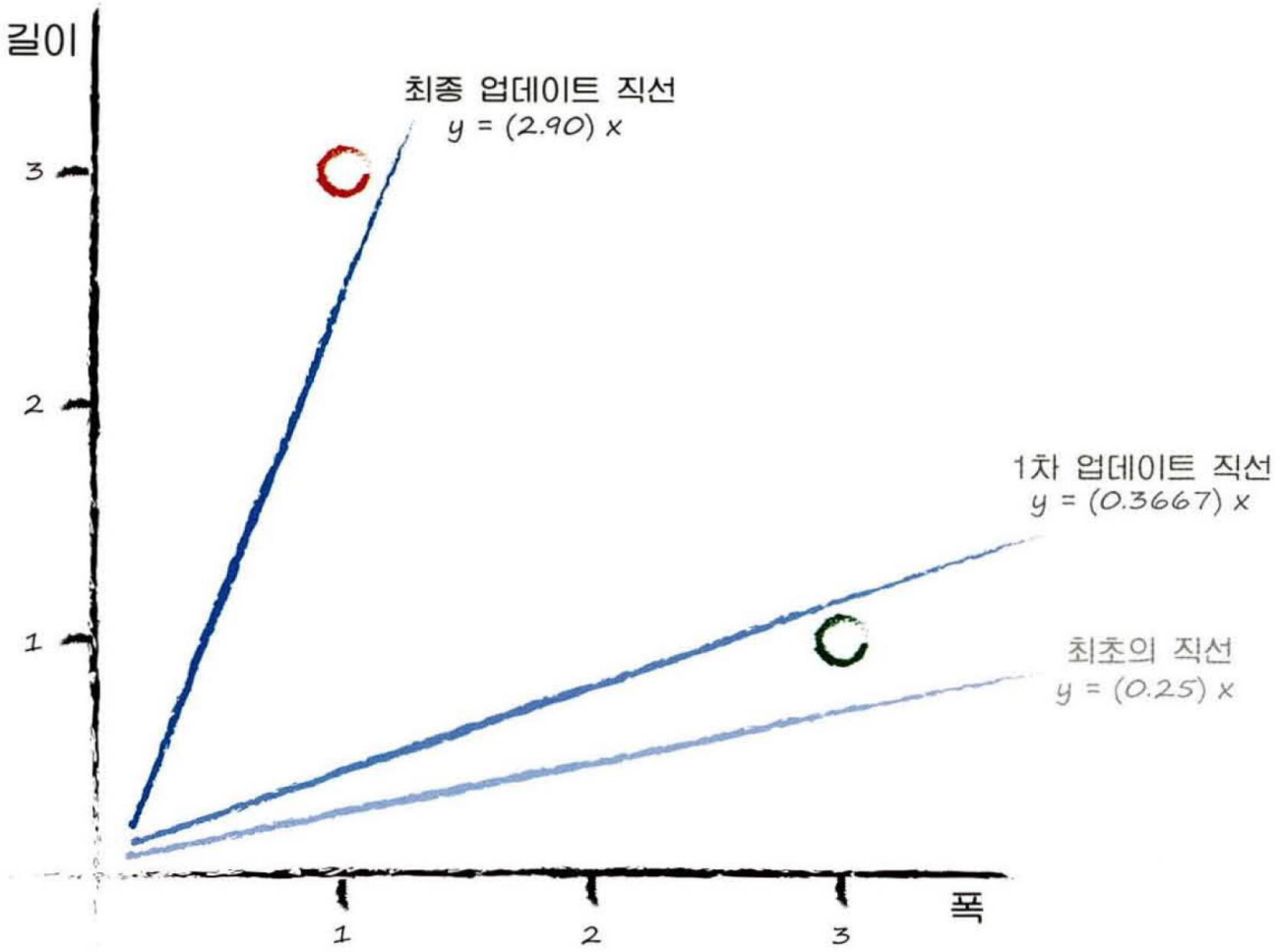
- A를 다시 업데이트하면 다음과 같다.

$$\Delta A = E / x = 2.5333 / 1.0 = 2.5333$$

$$A_{new} = A + \Delta A = 0.3667 + 2.5333 = 2.9$$

분류자 학습시키기

- 예제의 두번째 학습 데이터



분류자 학습시키기

- 무엇이 문제인가?
  - 각각의 학습데이터에 대해 기울기를 계속 업데이트를 하면 최종적으로는 그저 마지막 학습데이터에 맞춰진 업데이트 결과만을 얻게 됨.
  - 즉, 마지막 학습데이터에 기반한 업데이트 이전에 수행했던 학습에 기반한 업데이트들은 의미가 없어짐.
- 해결책은 업데이트의 정도를 조금씩 조정하는 것.
  - 각각의 새로운 A로 바로 점프하는 것이 아니라 ΔA의 일부씩만 업데이트하는 것.
  - 즉, 기존 여러 번의 학습에 기초해 업데이트된 값을 유지하면서, 학습데이터가 제시하는 방향으로 조금씩만 직선을 이동.
  - 이러한 방식은 학습데이터의 오차나 잡음의 영향을 약화시켜줌.

분류자 학습시키기

- 기울기 업데이트에 대한 정의를 다음과 같이 재정의

$$\Delta A = L\left(E / x\right)$$

- 머신러닝에서는 조정인자 L을 학습률(learning rate)라고 함
- L=0.5로 설정한 후 다시 첫번째 학습데이터에 대해 업데이트를 수행하면 다음과 같다.

$$y = Ax = 0.25 * 3.0 = 0.75$$

$$E = t - y = 1.1 - 0.75 = 0.35$$

$$\Delta A = L(E / x) = 0.5 * (0.35 / 3.0) = 0.0583$$

$$\therefore A_{new} = A + \Delta A = 0.25 + 0.0583 = 0.3083$$

- 업데이트된 A를 첫번째 학습데이터에 적용하면 다음과 같다.

$$y = A_{new} x = 0.3083 * 3.0 = 0.9250$$

분류자 학습시키기

- 업데이트된 A를 두번째 학습데이터에 적용하면 다음과 같다.

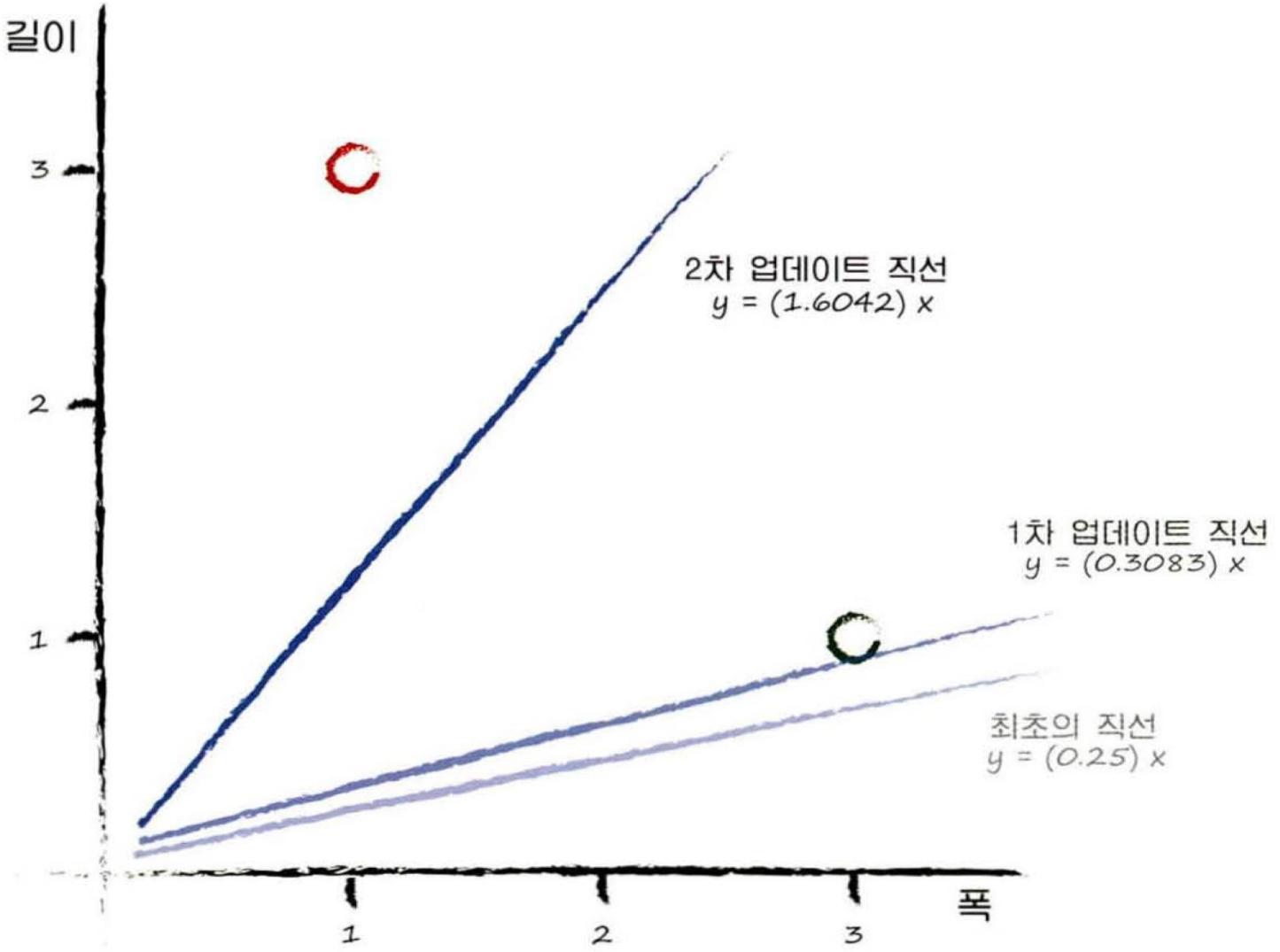
$$y = Ax = 0.3083 * 1.0 = 0.3083$$

$$E = t - y = 2.9 - 0.3083 = 2.5917$$

$$\Delta A = L(E / x) = 0.5 * (2.5917 / 1.0) = 1.2958$$

$$\therefore A_{new} = A + \Delta A = 0.3083 + 1.2958 = 1.6042$$

분류자 학습시키기



더 많은 분류자가 필요

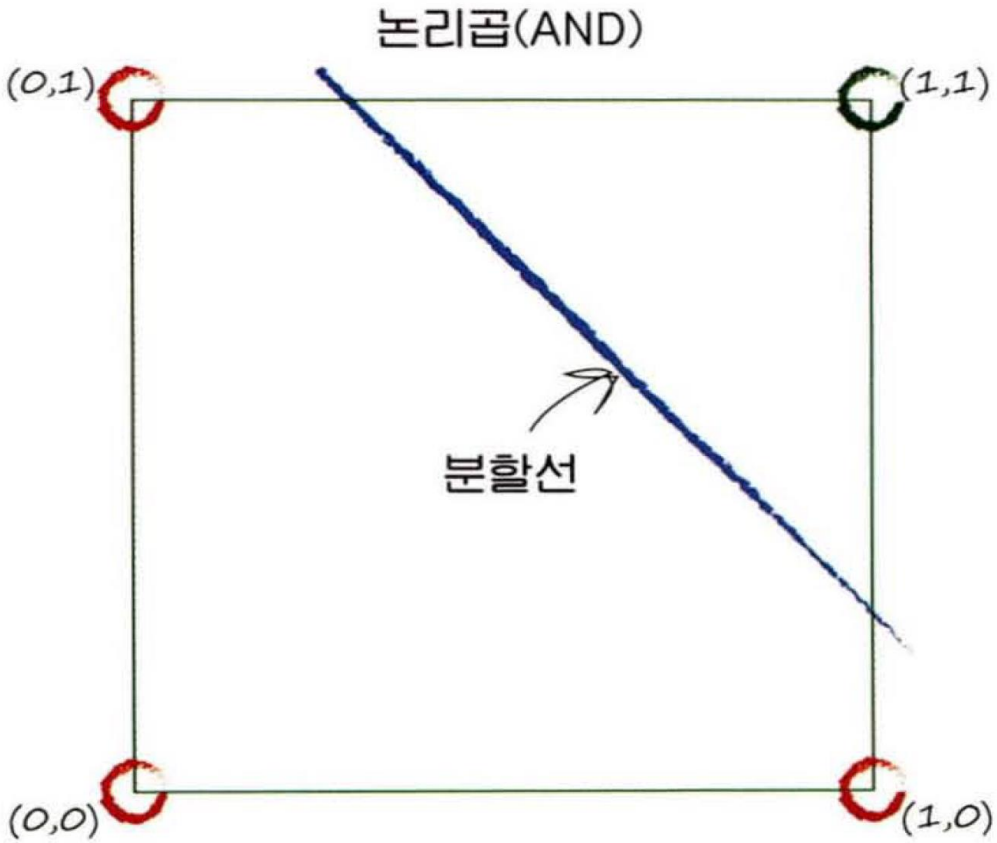
- 불 논리함수의 표현

입력 A	입력 B	논리곱	논리합
0	0	0	0
0	I	0	I
I	0	0	I
I	I	I	I

더 많은 분류자가 필요

- 논리곱(AND)의 결과를 선형 분류자로 분류
  - 그림의 직선은 선형 분류자가 학습할 수 있는 선형함수

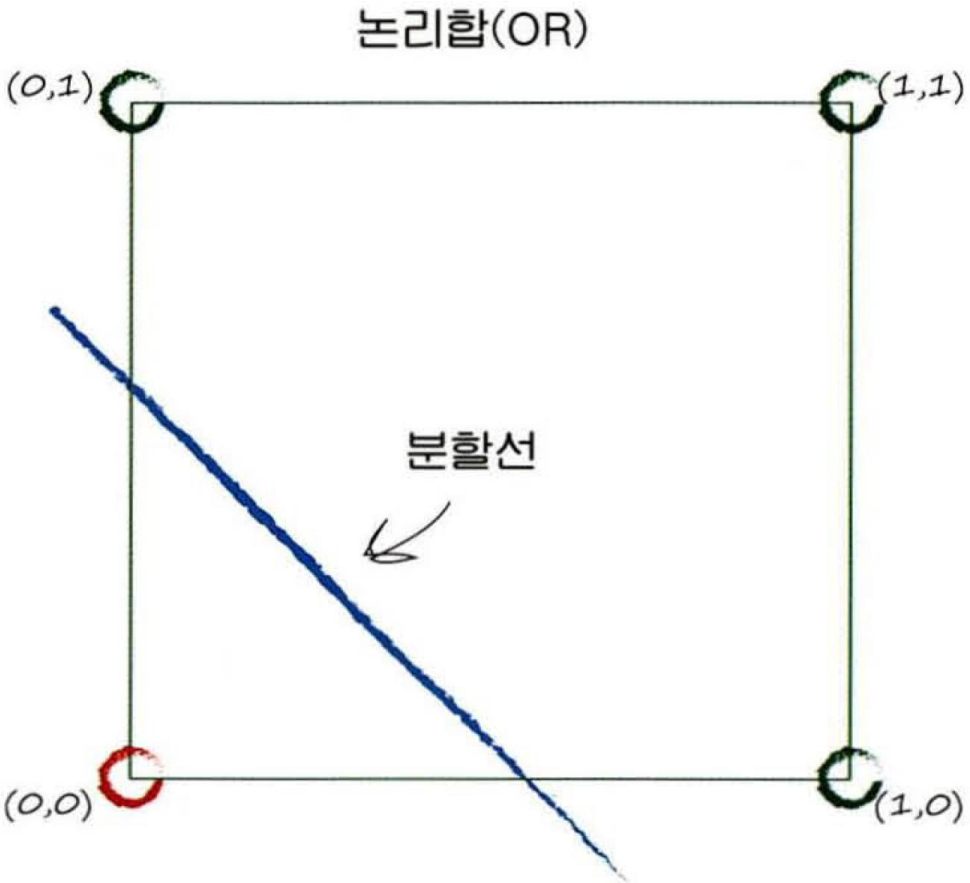
$$y = Ax + B$$



더 많은 분류자가 필요

- 논리합 (OR)의 결과를 선형 분류자로 분류
  - 그림의 직선은 선형 분류자가 학습할 수 있는 선형함수

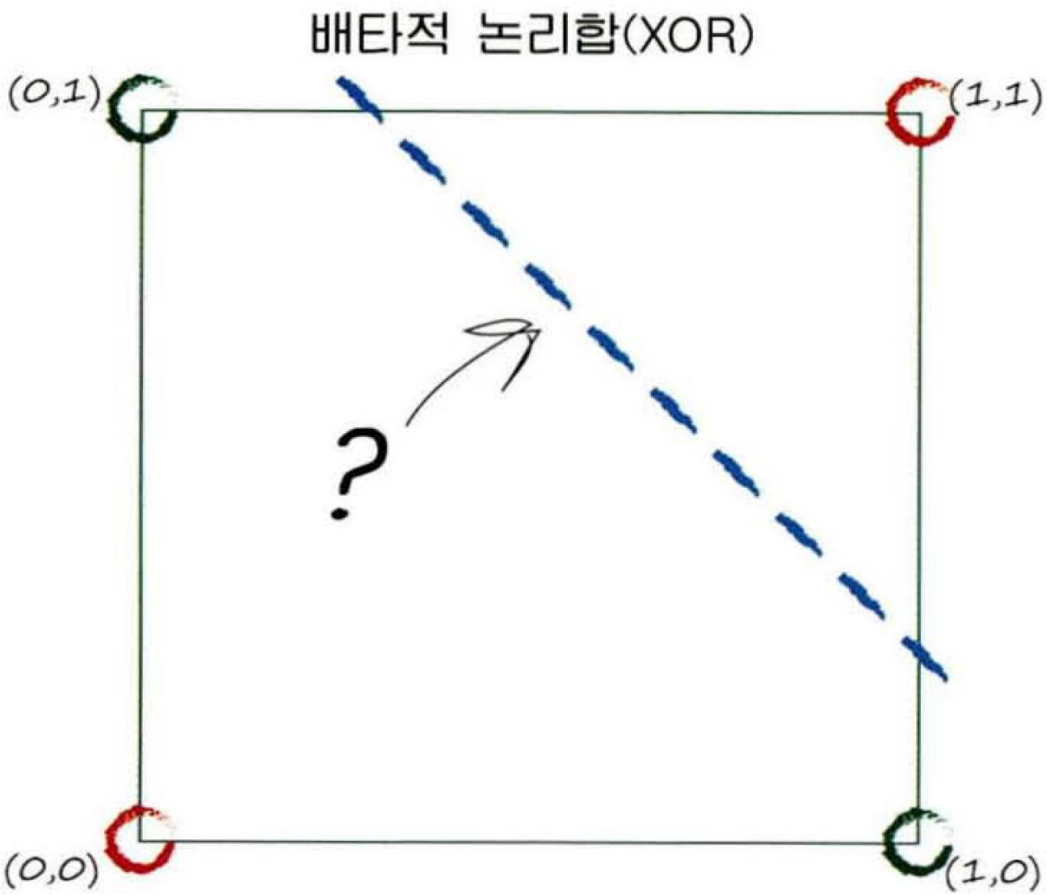
$$y = Ax + B$$



더 많은 분류자가 필요

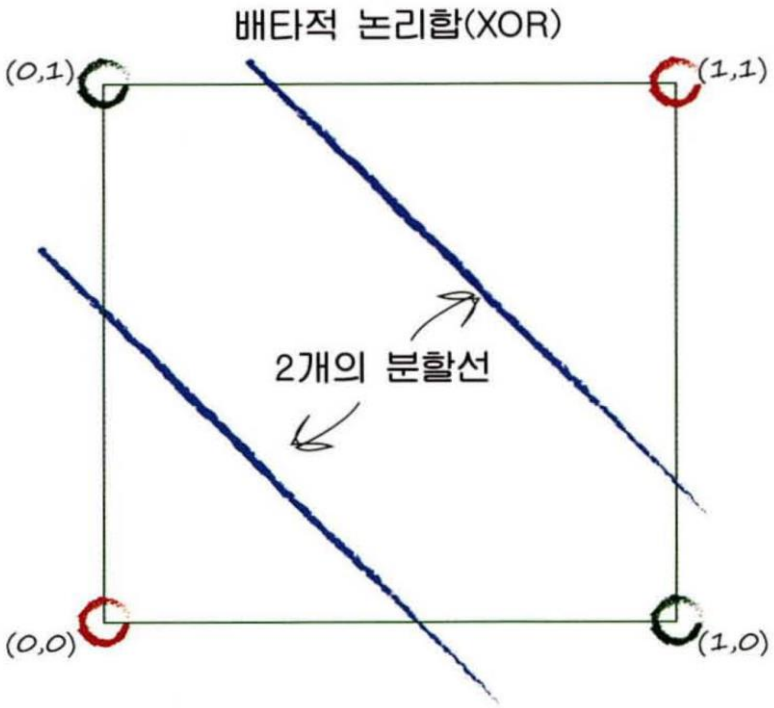
- 배타적 논리합 (XOR)의 결과를 선형 분류자로 분류

입력 A	입력 B	XOR
0	0	0
0	1	1
1	0	1
1	1	0



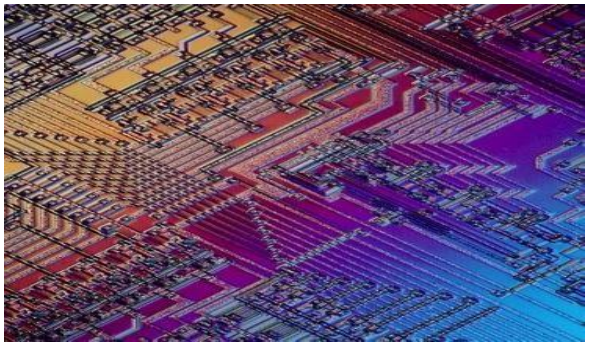
더 많은 분류자가 필요

- 선형 분류자의 한계
  - 단순한 선형분류자는 당면한 문제가 단일 직선에 의해 구분되지 않는다면 아무 소용이 없다.
- 해결 방법은?
  - 여러 개의 분류자를 이용하는 것!!!



대자연의 컴퓨터, 뉴런

- 컴퓨터와 뇌의 비교

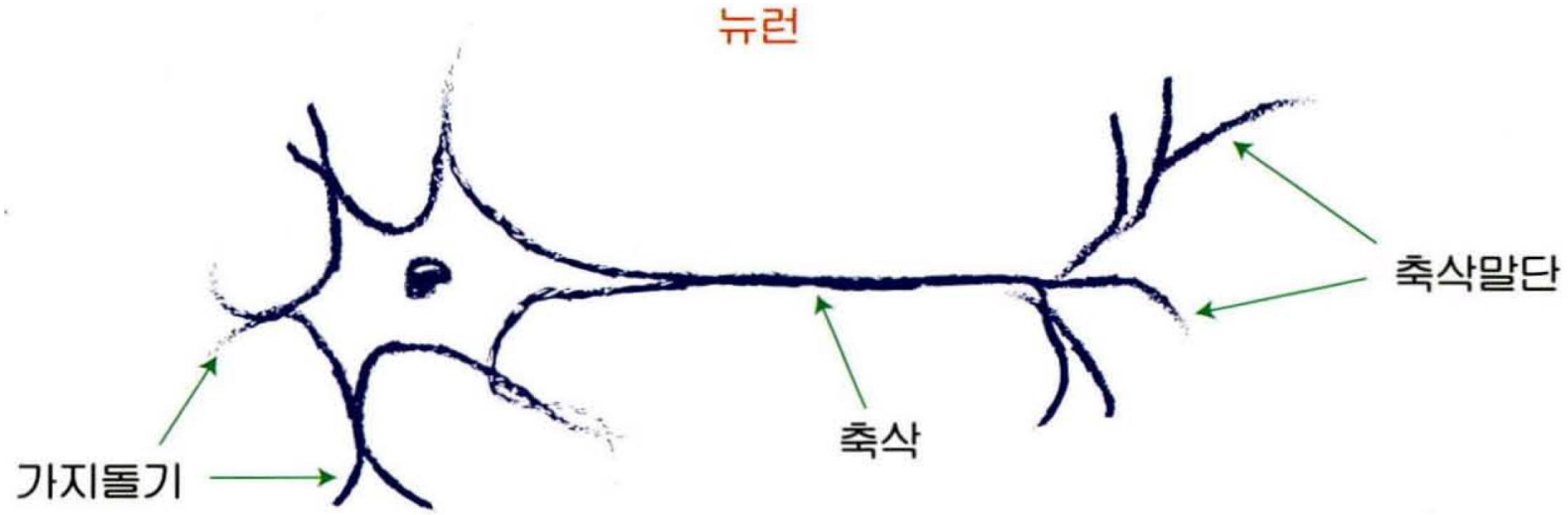


- 정확한 소자 (논리 소자)
- 소자 속도 매우 빠름 (10^{-9} 초)
- 디지털 회로망 (전자 회로망)
- 주소기반 메모리 (국지적/독립적)
- 논리/산술적 연산 (조작적)
- 중앙집중식 순차 처리
- 프로그래밍기반 명시적 지식

- 부정확한 소자 (10^{11} 개 뉴런)
- 소자 속도 느림 (10^{-3} 초)
- 아날로그 회로망 (10^{14} 개 연결선)
- 내용기반 메모리 (전역적/관계적)
- 패턴/연상기반 연산 (연관적)
- 분산적 병렬 처리
- 학습(경험/데이터)기반 암묵적 지식

대자연의 컴퓨터, 뉴런

- 뉴런의 동작 원리
  - 뉴런은 전기 입력을 받아 또 다른 전기 신호를 발생
  - 이는 앞에서 살펴본 분류 또는 예측자에서 입력을 받아 어떤 처리를 해 결과를 출력하는 것과 매우 유사

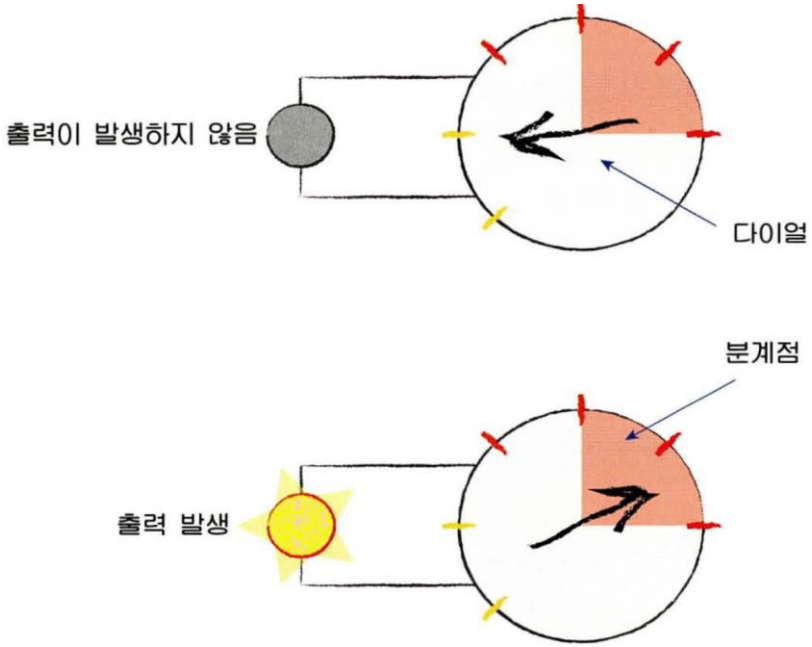


대자연의 컴퓨터, 뉴런

- 뉴런을 선형함수로 표현할 수 있을까?
  - 뉴런은 입력을 받았을 때 즉시 반응하지 않는다.
  - 즉, 뉴런의 출력은 다음과 같은 형태가 될 수 없다.

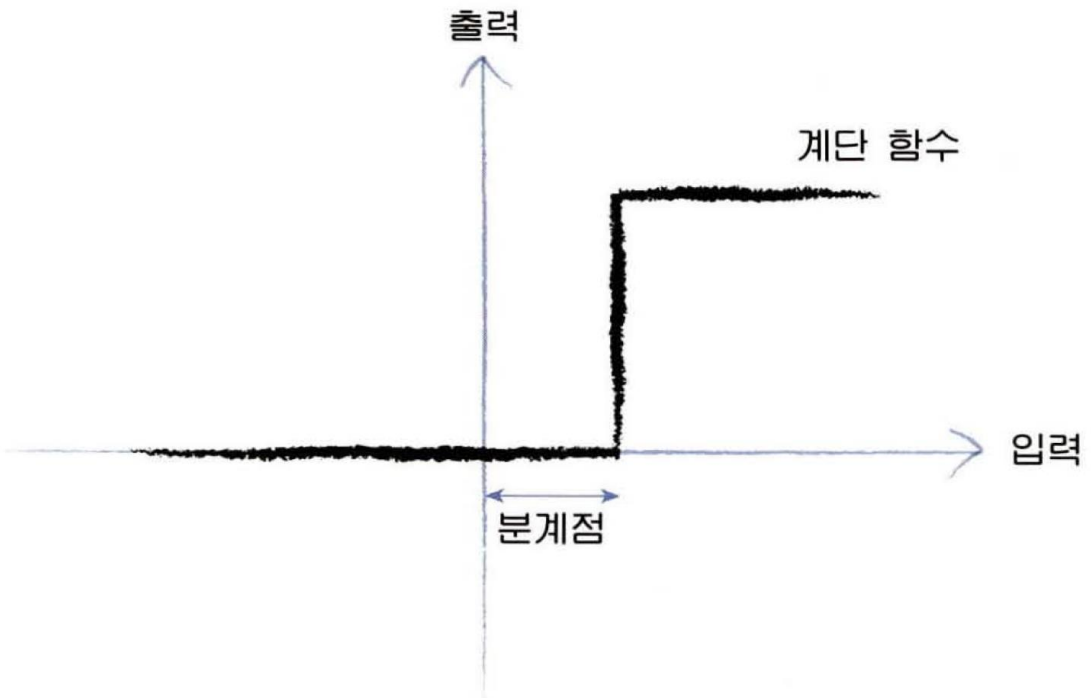
$$y = Ax + B$$

  - 뉴런은 입력이 누적되어 특정 수준으로 커진 경우에만 신호를 출력
  - 즉, 입력값이 임계점에 도달해야 출력이 발생



대자연의 컴퓨터, 뉴런

- 이처럼 입력 신호를 받아 특정 임계점(threshold)를 넘어서는 경우에 출력 신호를 생성해주는 함수를 활성화 함수(activation function)이라 한다.
- 수학적으로 가장 단순한 형태인 활성화 함수는 계단 함수(step function)이다.



대자연의 컴퓨터, 뉴런

- 계단 함수의 한계
  - 0 또는 1의 이진값만을 출력
  - 오차는 0 아니면 1만 가능: 미세한 오차 조정 불가능
- 계단 함수의 개선 – 시그모이드 함수
  - 선형퍼셉트론은 선형 분리가 가능한 패턴분류 문제만 해결 가능
  - 복잡한 문제를 풀기 위해서는 비선형 결정 경계를 생성해야 함
  - 미분이 가능하려면 연속함수일 필요가 있음
  - 시그모이드(sigmoid) 함수: 비선형, 미분 가능한 연속함수

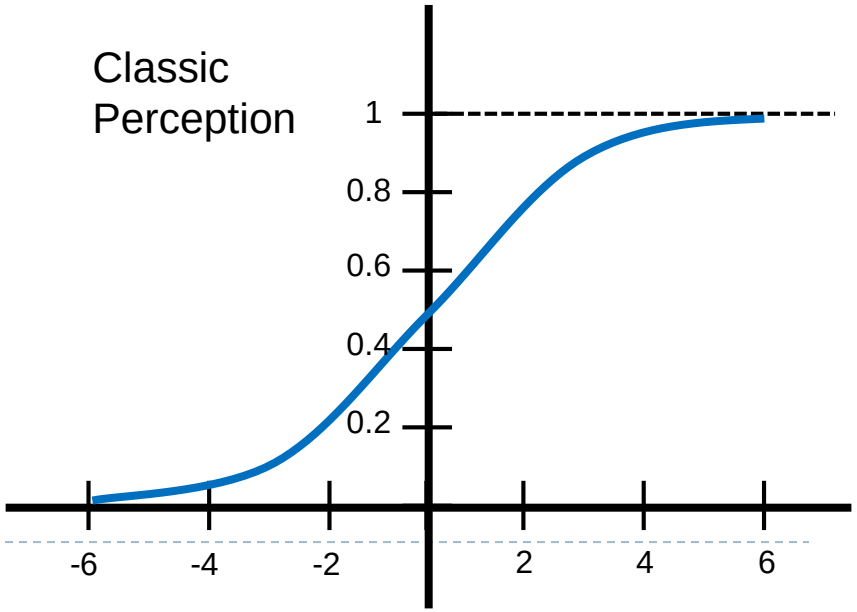
대자연의 컴퓨터, 뉴런

- 시그모이드 뉴런 신경망
  - 가중치 입력 계산시 비선형의 시그모이드 전이 함수를 거쳐서 출력함 (Aizerman, Braverman, & Rozonoer, 1964)

$$s = \sum_{i=0}^{n} w_i x_i \qquad f = \sigma(x) = \frac{1}{1+\exp(-x)}$$

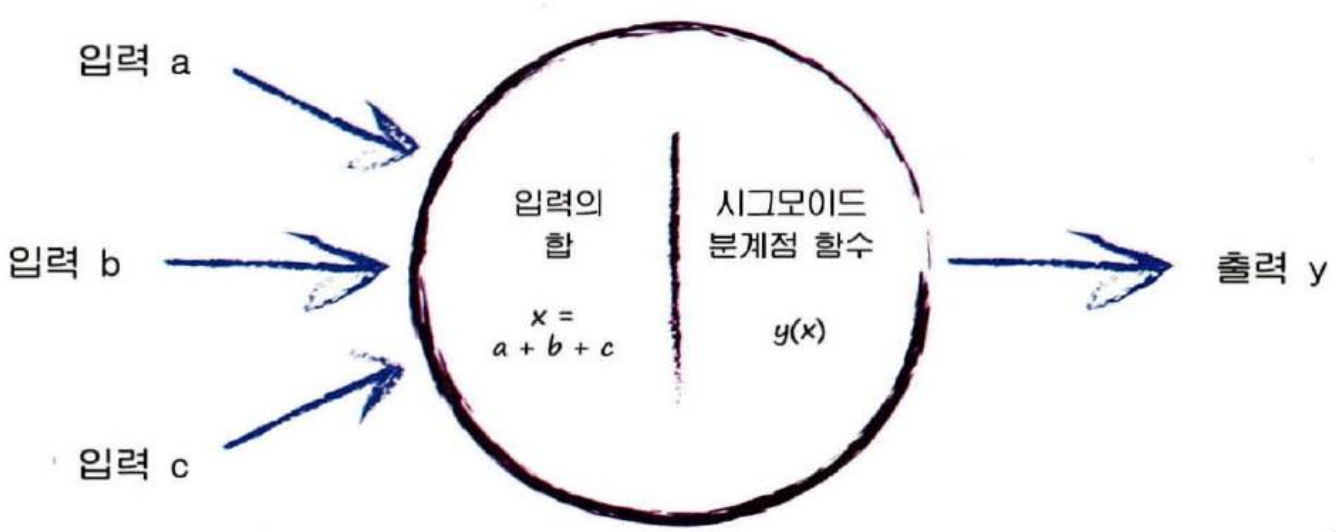
  - $[-\infty, +\infty]$의 입력 구간 s에 대한 출력을 $[0, 1]$ 의 구간으로 변환해 줌
  - 미분 가능

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$$



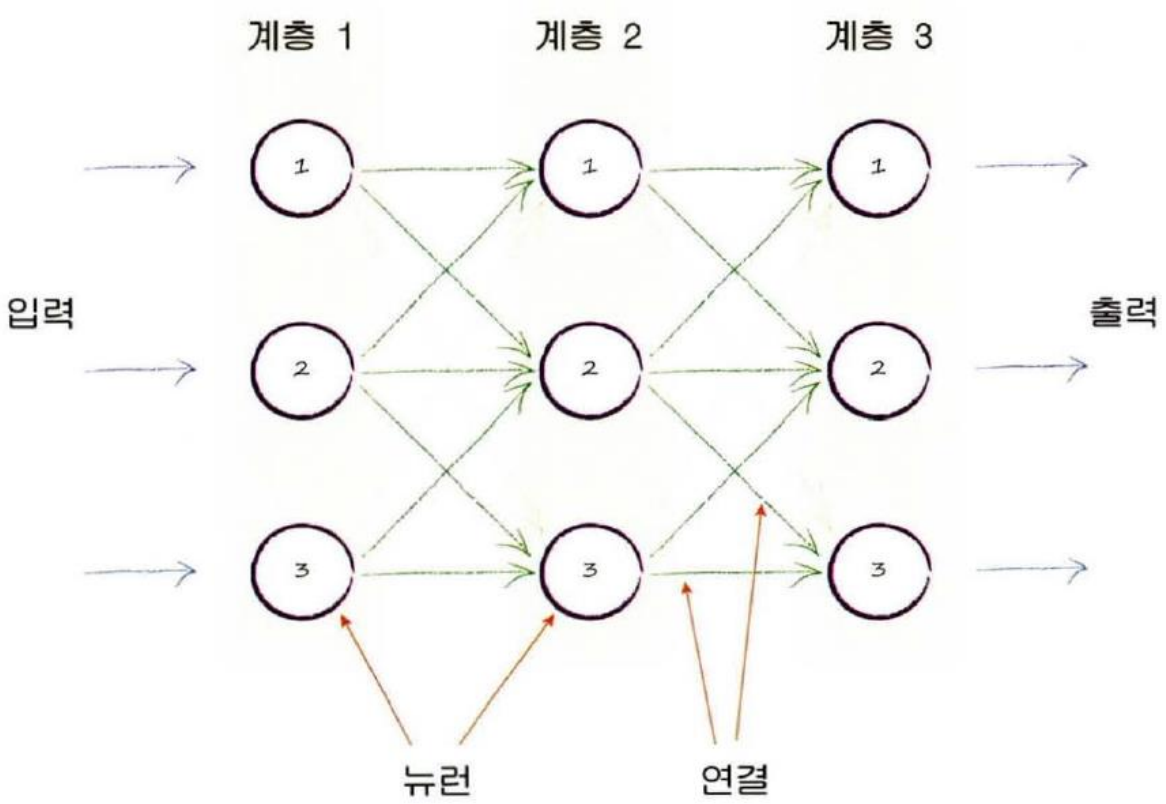
대자연의 컴퓨터, 뉴런

- 시그모이드 뉴런 신경망
  - 만약 입력 a, 입력 b, 입력 c의 합인 x가 임계점을 넘어설 정도로 충분히 크지 않다면 시그모이드 함수는 아무것도 출력하지 않는다.
  - 반대로 x가 임계점을 넘으면 시그모이드 함수는 이 뉴런을 작동
  - 나머지 입력 값들이 매우 작다고 하더라도 단지 1 개의 입력 값만 충분히 크다면 뉴런은 작동할 수 있다.



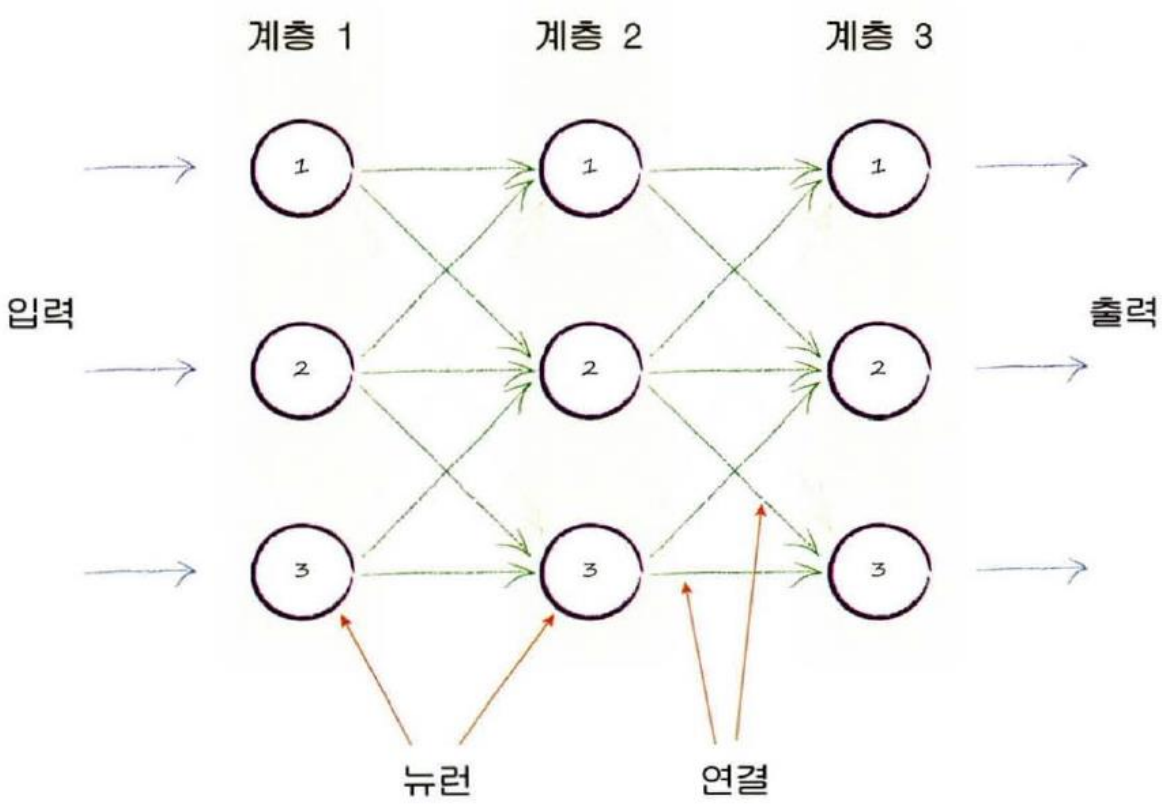
대자연의 컴퓨터, 뉴런

- 생물학적 뉴런을 인공적으로 모델화하려면?
  - 뉴런을 여러 계층(layer)에 걸쳐 위치시키고 각각의 뉴런은 직전 계층과 직후 계층에 있는 모든 뉴런들과 상호 연결되어 있는 식으로 표현.



대자연의 컴퓨터, 뉴런

- 생물학적 뉴런을 인공적으로 모델화하려면?
  - 뉴런을 여러 계층(layer)에 걸쳐 위치시키고 각각의 뉴런은 직전 계층과 직후 계층에 있는 모든 뉴런들과 상호 연결되어 있는 식으로 표현.

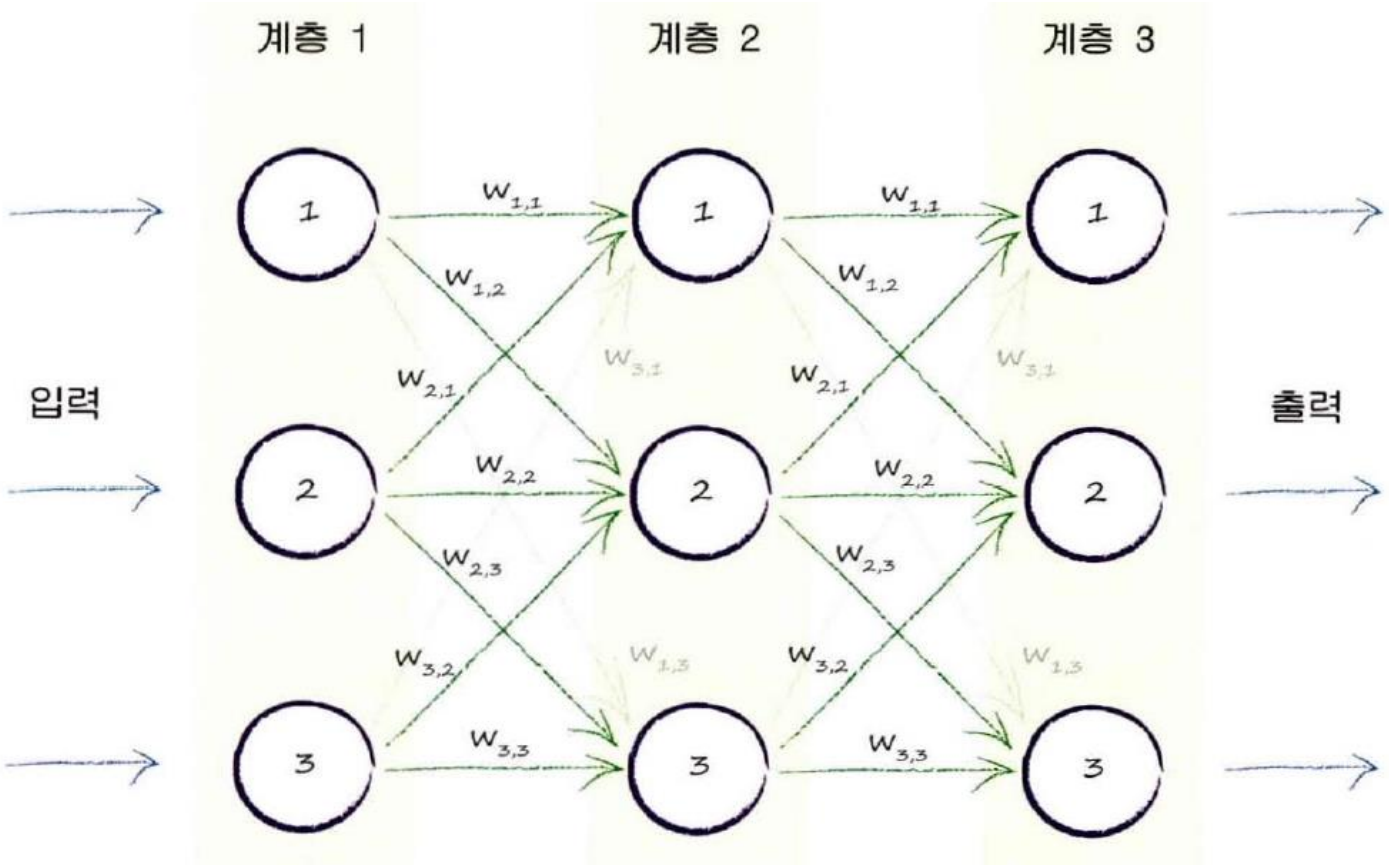


대자연의 컴퓨터, 뉴런

- 인공 신경망에서 과연 어떤 부분이 학습을 하는 것일까?
- 학습 데이터를 통해 학습을 진행하게 되면 도대체 무엇을 조정해야 하는 것일까?
- 앞에서 살펴본 선형 분류자의 기울기처럼 우리가 업데이트 해나가야 할 어떤 매개변수가 존재하는 것일까?
  - 하나의 노드 내에서 입력 값들의 합을 조정
  - 시그모이드 함수의 형태를 조정
  - 노드 간 연결의 강도를 조정

대자연의 컴퓨터, 뉴런

- 인공 신경망에서 과연 어떤 부분이 학습을 하는 것일까?
  - 가장 간단한 방법은 노드간 연결의 강도, 가중치 (weight)를 조정하는 것!!!

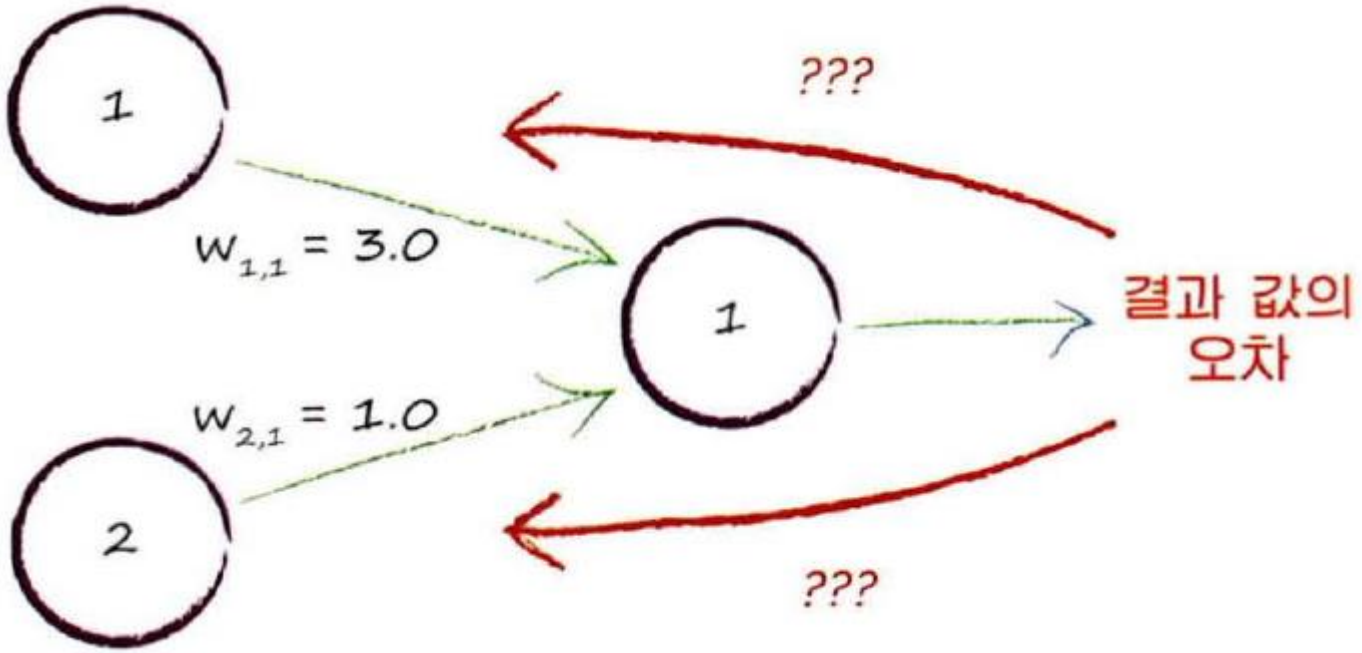


대자연의 컴퓨터, 뉴런

- 꼭 하나의 노드가 직전 계층과 직후 계층의 모든 노드와 연결되어야 하는 것일까?
  - 사실 꼭 이렇게 모든 노드가 연결되어야 한다는 법은 없다.
  - 일반적으로 모든 노드 간에 연결을 하는 이유는 이렇게 연결해야 프로그램으로 구현하기가 편리
  - 문제를 해결하기 위해 필요로 하는 절대적인 최솟값보다 연결이 몇 개 더 있다고 해서 별문제가 되는 것도 아니기 때문
  - 꼭 필요한 연결이 아니라고 하면 그 연결은 실제로 학습 과정에서 자동적으로 가중치가 매우 낮아지게 됨.

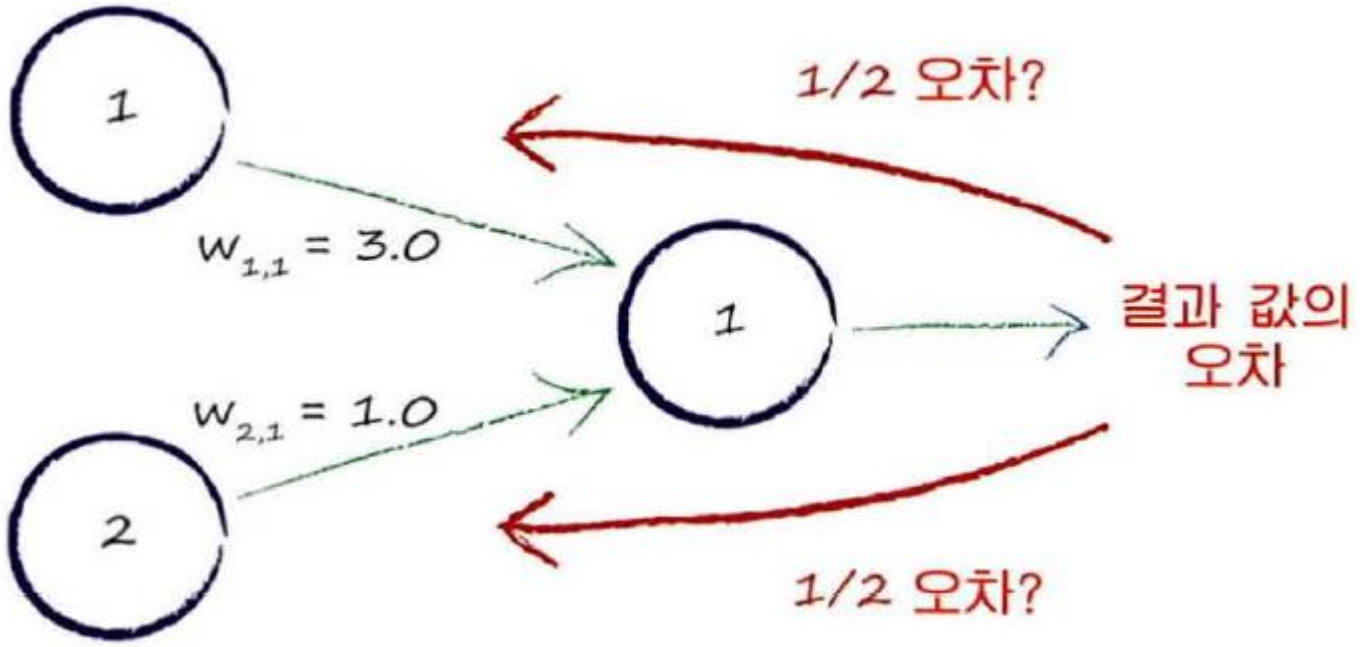
여러 노드에서 가중치 학습하기

- 앞에서 오차에 기반을 두고 선형 분류자를 정교화시켰을 때 오차와 기울기의 관계가 매우 단순했기 때문에 이를 조정해가는 것은 별로 어려운 과정은 아니었다.
- 그렇다면 여러 개의 노드가 결과 값과 오차에 영향을 주는 경우에는 가중치를 어떻게 업데이트해야 할까??



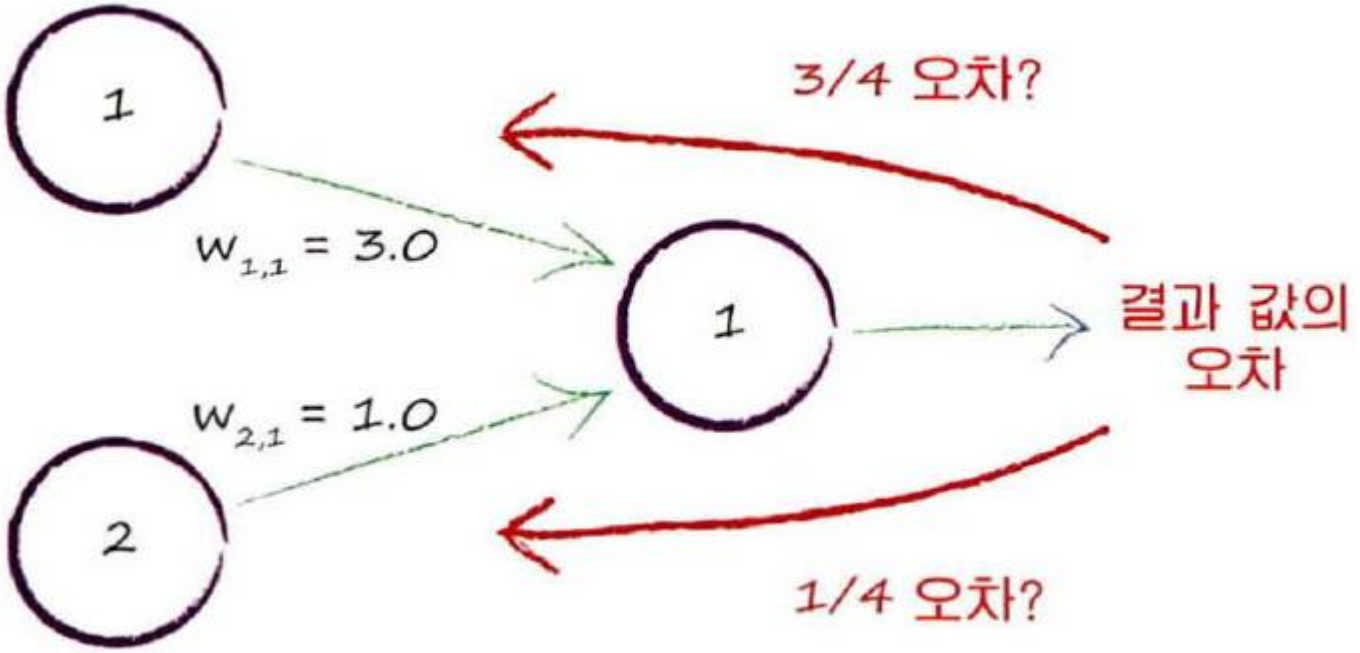
여러 노드에서 가중치 학습하기

- 결과 노드에 영향을 주는 노드가 한 개인 경우에는 훨씬 간단했지만, 이와 같이 2개의 노드를 가지는 경우에는 결과 값의 오차를 어떻게 활용해야 할까??
- 한 가지 방법은 그림과 같이 모든 연결된 노드에 대해 오차를 균일하게 나누어 분배하는 것



여러 노드에서 가중치 학습하기

- 또 다른 방법은 오차를 나누어 분배하지만 차별을 두는 것인데, 더 큰 가중치를 가지는 연결에 더 큰 오자를 분배하는 것!!!

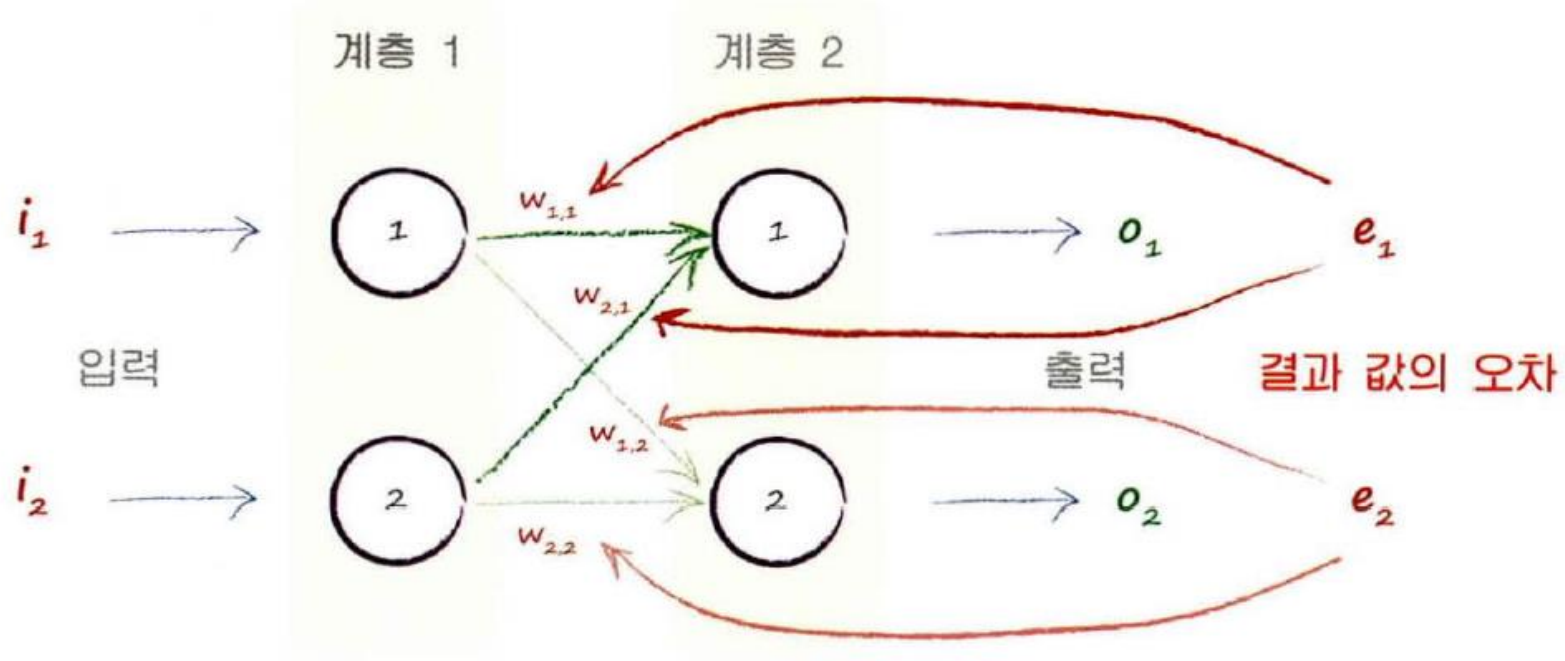


여러 노드에서 가중치 학습하기

- 또 다른 방법은 오차를 나누어 분배하지만 차별을 두는 것인데, 더 큰 가중치를 가지는 연결에 더 큰 오자를 분배하는 것!!!
  - 이 방법은 노드가 더 많아지더라도 동일하게 적용 가능
  - 만약 100 개의 노드가 다음 계층의 노드에 연결되어 있다고 한다면 오차를 100개의 연결에 나누어주되 그 값은 각각의 가중치에 띠라 즉 각각의 연결이 오차에 영향을 주는 정도에 비례해서 전달
- 따라서, 인공신경망은 가중치를 두 가지 방법으로 활용
  - 우선 앞에서 여러 사례를 살펴본 바와 같이 신호를 하나의 계층에서 다음 계층으로 전파하는 데에 가중치를 이용
  - 두 번째로 오차를 하나의 계층에서 직전 계층으로, 즉 역으로 전파하는 데에도 가중치를 이용
  - 첫 번째 방식은 앞에서 이미 본 순전파법이고 두 번째 방법을 역전파법 (back propagation) 이라고 한다.

여러 노드에서의 오차의 역전파

- 2개의 출력 노드를 가지는 인공신경망의 예
  - 출력 노드가 여러 개라고 해서 변하는 것은 아무 것도 없다.
  - 그저 첫번째 출력 노드에서 했던 작업을 두 번째 출력 노드에서 동일하게 반복하면 된다.
  - 이유는 하나의 출력 노드로의 연결은 다른 출력 노드로의 연결과 완전 별개이기 때문



가중치의 진짜 업데이트

- 네트워크의 각 계층에 걸쳐 역전파되는 오차를 구하는 이유는 인공 신경망이 보다 나은 답을 출력하게 하기 위해 가중치를 조정 해가는 데 지침 역할을 하는 것이 오차이기 때문.
- 하지만 신경망에서 노드는 단순한 선형 분류자가 아니다!!!
- 노드는 입력되는 신호에 가중치를 적용한 후 이의 합을 구하고 다시 여기에 시그모이드 활성화 함수를 적용하는 식으로 좀 더 복잡한 구조를 가지는 노드 사이를 연결하는 연결 노드의 가중치를 어떻게 업데이트해야 할까??

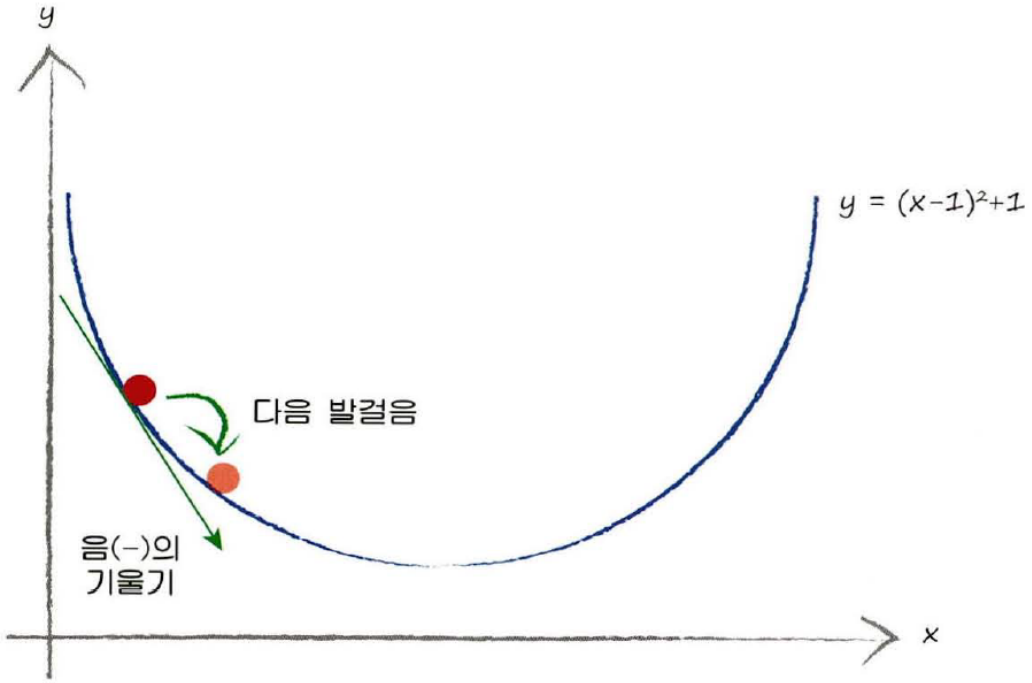
가중치의 진짜 업데이트

- 어떤 대수학 공식을 이용해 가중치를 단번에 구해낼 수 없는 것일까??
  - 신호가 여러 개의 계층을 타고 전파되어 나갈 때 한 계층을 거칠 때마다 직전 계층의 출력 값이 다음 계층의 입력 값이 되므로 함수의 함수 함수의 함수의 함수 ... 같은 식으로 수많은 함수의 조합이 필요.
  - 따라서 수학 연산의 과정이 너무 복잡하게 되므로 가중치를 한 방에 풀어주는 대수학을 활용할 수 없다.
  - 다음은 3개 노드로 구성되는 3 개의 계층으로 이루어진 신경망에서 출력 노드의 결과값을 입력 노드와 가중치의 함수로 표현한 것이다.

$$o_k = \frac{1}{1+e^{-\sum_{j=1}^{3}\left(w_{j,k}\cdot\frac{1}{1+e^{-\sum_{i=1}^{3}(w_{i,j}\cdot x_i)}}\right)}}$$

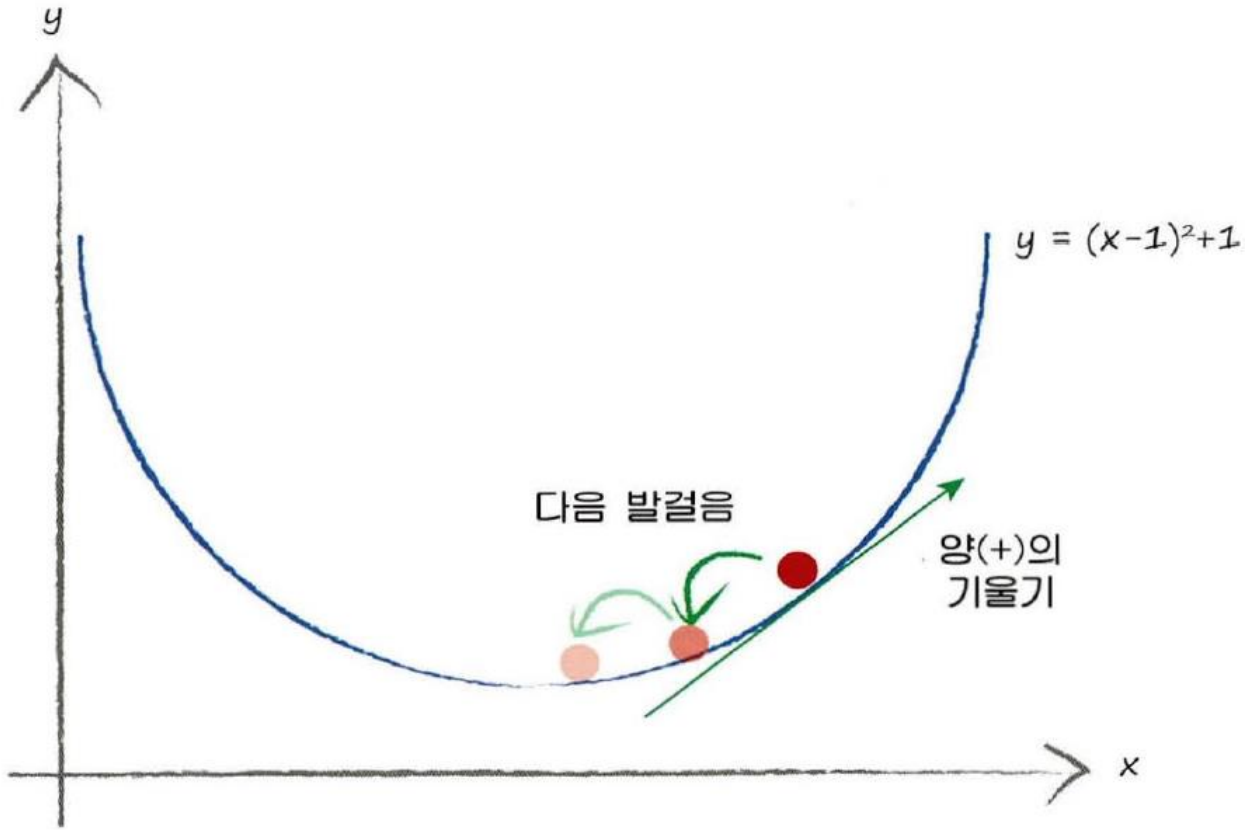
가중치의 진짜 업데이트

- 경사 하강법 (Gradient descent)
  - 복잡한 함수에 대한 이해가 없더라도 경사 하강법을 이용해 최저점을 찾을 수는 있다.
  - 함수가 너무 복잡해 대수학을 통해서는 최저점을 찾아내기 어렵다면 경사 하강법을 대신 이용할 수 있다.
  - 경사하강법의 아이디어



가중치의 진짜 업데이트

- 경사 하강법 (Gradient descent)
  - 다른 지점에서 시작하는 경우

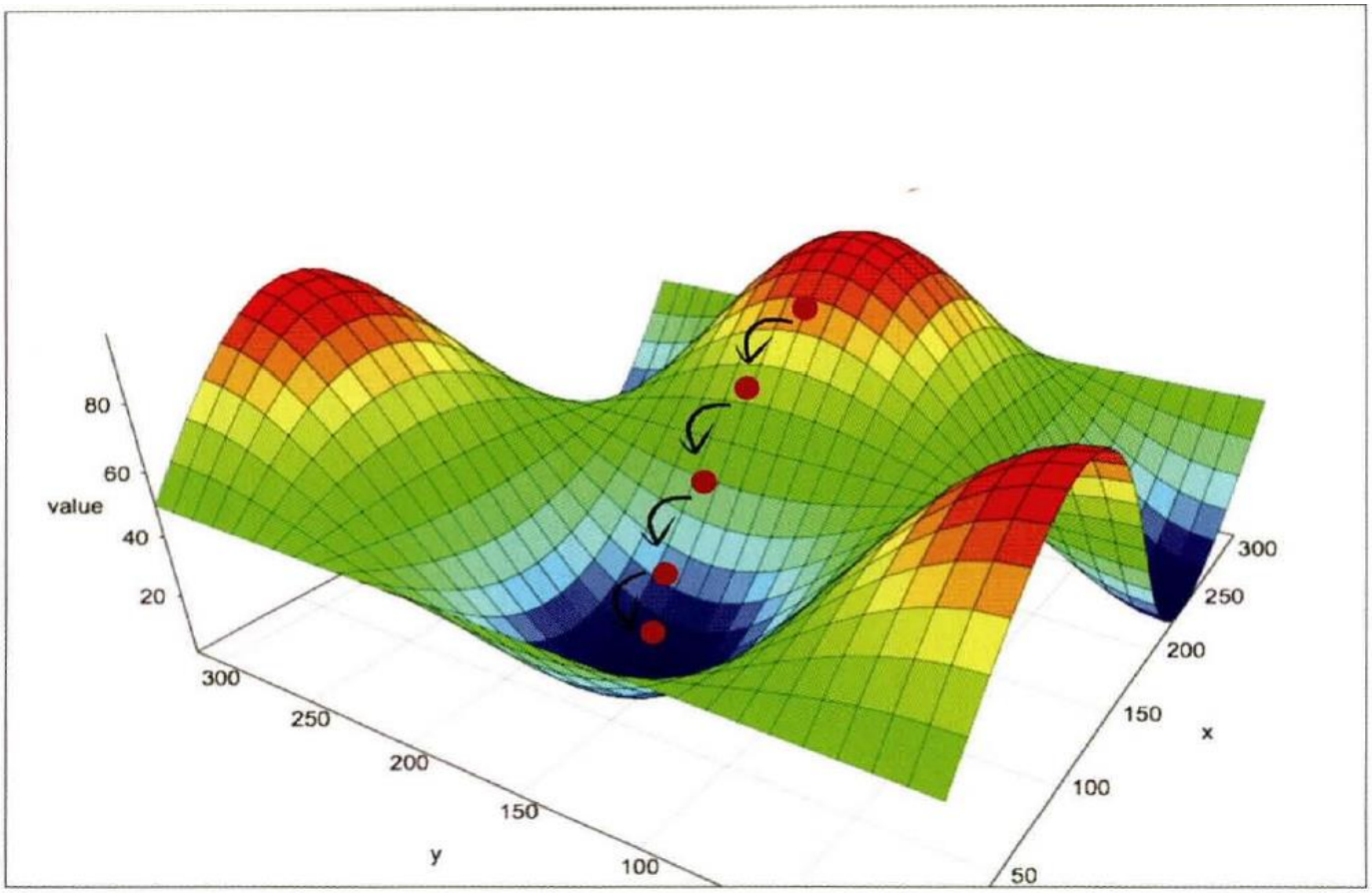


가중치의 진짜 업데이트

- 경사 하강법 (Gradient descent)
  - 한 가지 주의해야 할 점은 내딛는 걸음의 크기를 잘 조정해야 한다는 것이다.
  - 걸음 폭이 너무 크면 최저점을 단숨에 지나치게 되며(오버슈팅) 영원히 최저점에 도달하지 못하고 양쪽을 왔다 갔다 하는 결과가 되기 때문.
  - 경사 하강법에서의 가정은 최저점에 가까워질수록 기울기는 완만해진다는 사실이다.
  - 이는 대부분의 매끄럽고 연속적인 함수에서 옳은 가정이지만 보통 수학자들이 불연속 (discontinuity)이라고 부르는 지그재그 형태의 복잡한 함수에 대해서는 올바른 가정이 아니다.

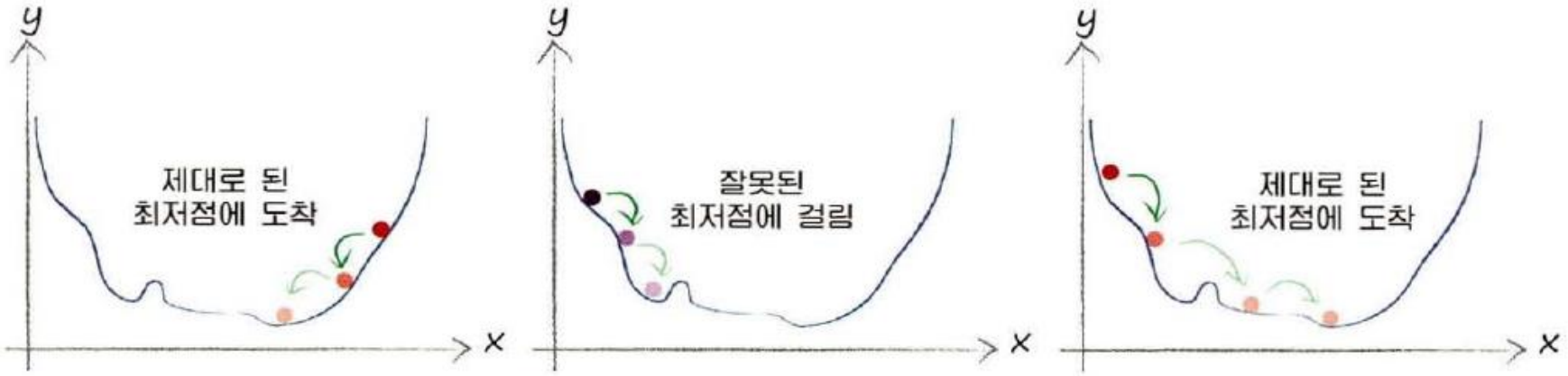
가중치의 진짜 업데이트

- 경사 하강법 (Gradient descent)
  - 2개의 매개변수에 의해 결정 되는 조금 더 복잡한 함수의 예



가중치의 진짜 업데이트

- 경사 하강법 (Gradient descent)
  - 경사 하강법에 의해 최저점으로 접근하는 세 가지 예



  - 잘못된 계곡에 빠지는 것을 피하려면 각각 다른 출발점에서 시작해 여러 번 학습하는 방법을 취할 수 있다.
  - 출발점이 다르다는 것은 매개변수의 초기 값을 다르게 준다는 것을 의미.
  - 즉, 인공 신경망에서는 가중치의 초기 값을 다르게 준다는 것을 의미.

가중치의 진짜 업데이트

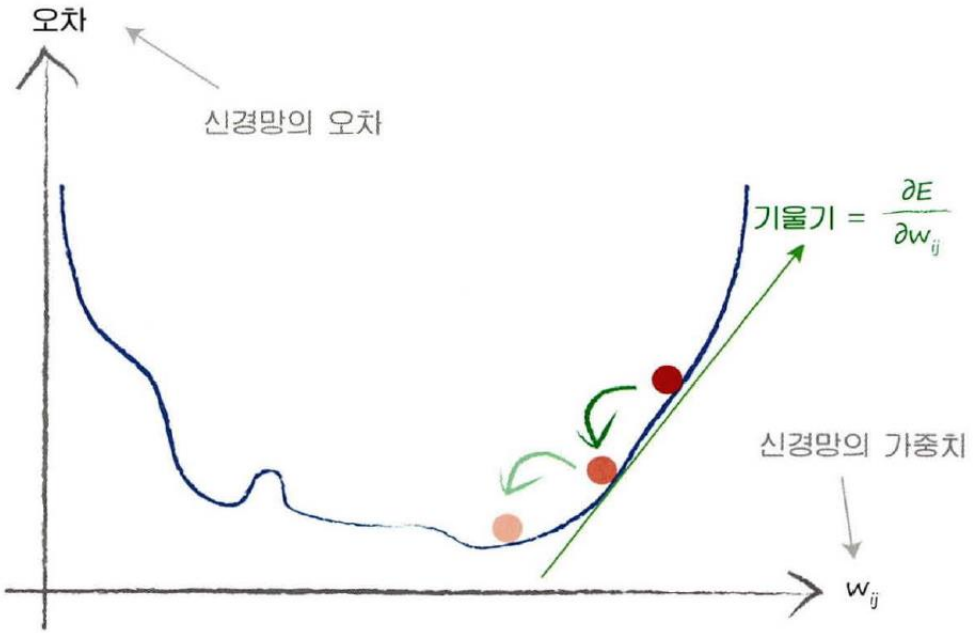
- 경사 하강법 (Gradient descent)
  - 경사 하강법을 사용하기 위해서는 올바른 오차 함수 (error/cost function)를 선택하는 것이 필수

실제 결과 값	목표 값	오차 (목표 값 – 실제 값)	오차 |목표 값 – 실제 값|	오차 (목표 값 – 실제 값)^2
0.4	0.5	0.1	0.1	0.01
0.8	0.7	–0.1	0.1	0.01
1.0	1.0	0	0	0
합		0	0.2	0.02

  - 경사 하강법을 이용하기 위해서는 제곱오차 방식을 이용
    - 오차함수로 제곱오차 방식을 사용하면 경사 하강법의 기울기를 구하는 대수학이 간결해진다.
    - 오차함수가 부드럽고 연속적이므로 경사 하강법이 잘 동작하게 되며, 값이 갑자기 상승하거나 빈 틈이 존재하지 않게 된다.
    - 최저점에 접근함에 따라 경사가 점점 작아지므로 목표물을 오버슈팅 할 가능성이 작아진다.

가중치의 진짜 업데이트

- 경사 하강법 (Gradient descent)
  - 미분으로 오차함수의 기울기 구하기
    - 미분은 어떤 하나의 변화가 다른 것의 변화에 어떤 영향을 주는지를 구하는 수학적 접근 방법.
    - 우리의 관심은 신경망 내에서 오차함수가 가중치에 의해 얼마나 영향을 받는가이다.
    - 다르게 표현하면 '오차는 가중치의 변화에 얼마나 민감한가'이다.



Q&A