

# 딥러닝 입문

임도형, dh-rim@hanmail.net

2020/04/01

# 임도형

- 대학원 AI, 신경망 전공
- 15년간 백엔드 SW 개발
- 최근 3년간 딥러닝 실무
- 현재 딥러닝 구축 컨설팅



분류하는 값

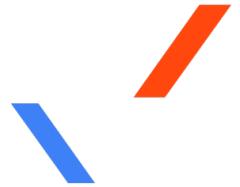
# 굴 익은 것 분류하기 - 사람

2개의 상자가 있다.

- 익은 것
- 안익은 것

사람이 분류해야 한다.

2개 상자의 것들을 보고 감 잡고 분류한다.

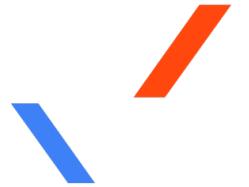


# 굴 익은 것 분류하기 - 로봇

로봇을 위한 프로그램을 작성해야 한다.

- ‘xxx하면 우측 상자에, 그렇지 않은 경우 좌측 상자에’

감잡고 분류할 때 사용한 기준을 하드코딩해야 한다.



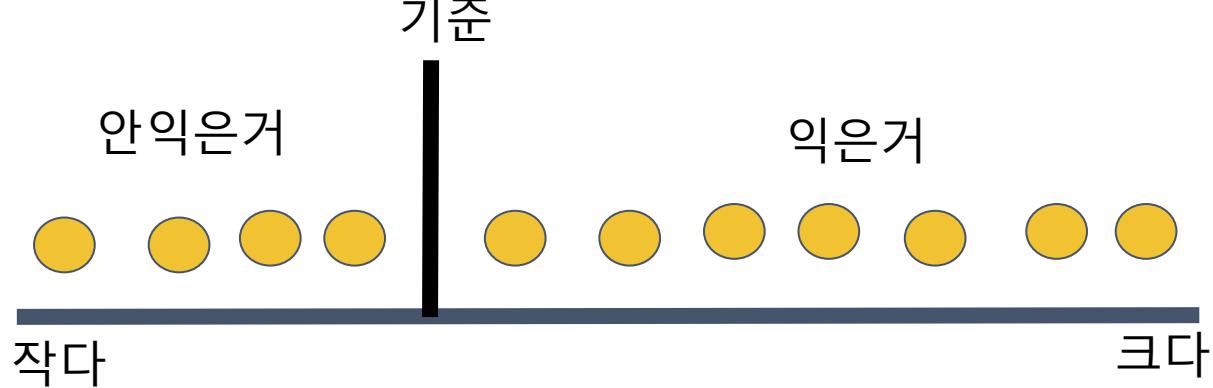
# 굴 익은 것 분류하기 - 로봇

2개 상자의 것을 보고 삼았던 기준은 크기

큰 것은 익은 것, 작은 것은 안 익은 것.

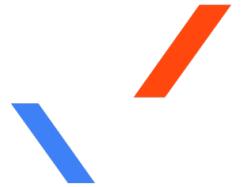
대충 있었던 크기를 측정하고 그것을 하드 코딩.

- ‘if(size > 7.5) then ...’

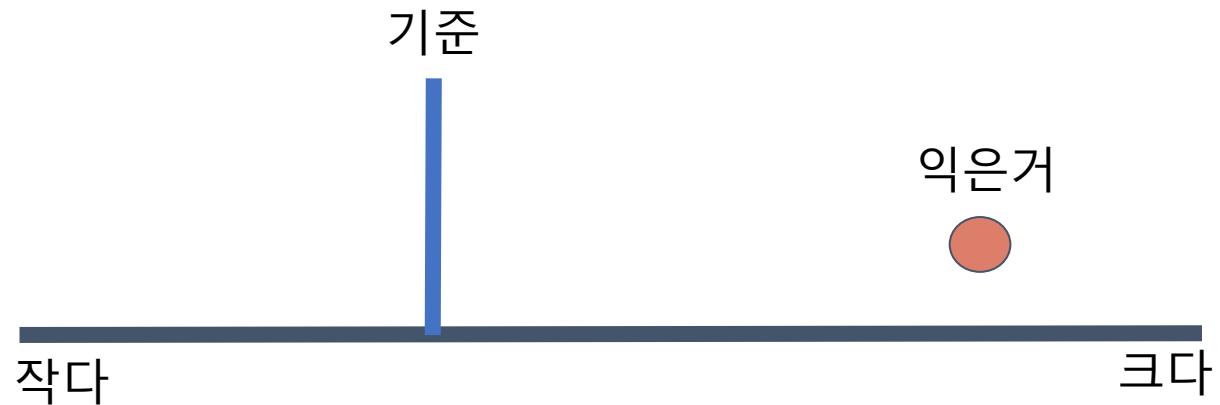


# 사과 익은 거 분류하기 - 로봇

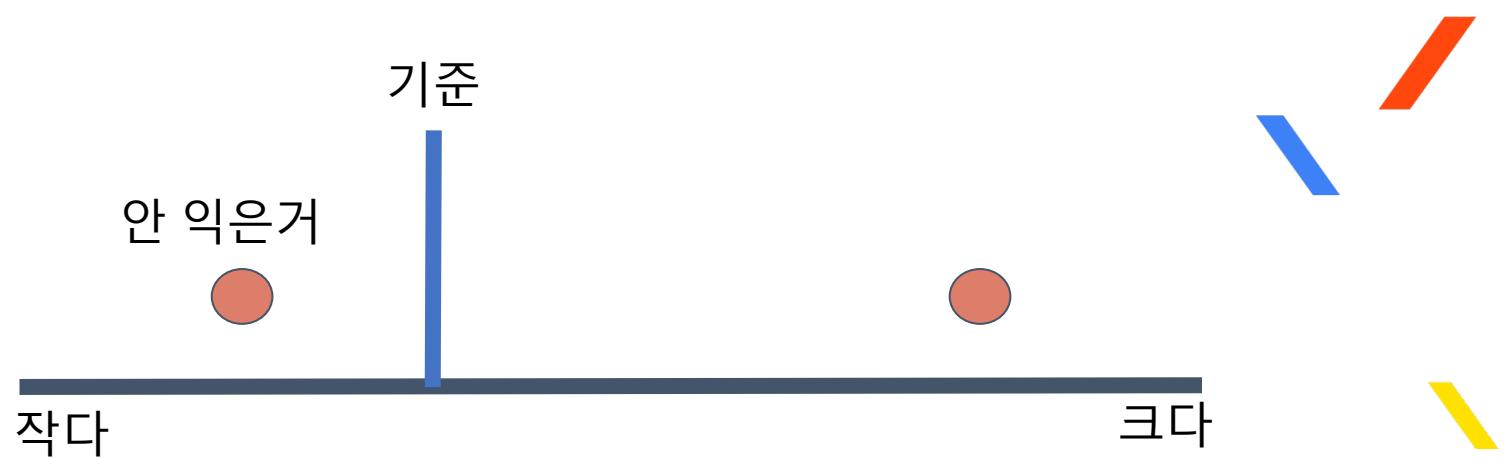
하드 코딩한 기준 마저 스스로 찾아내게 하고 싶다.



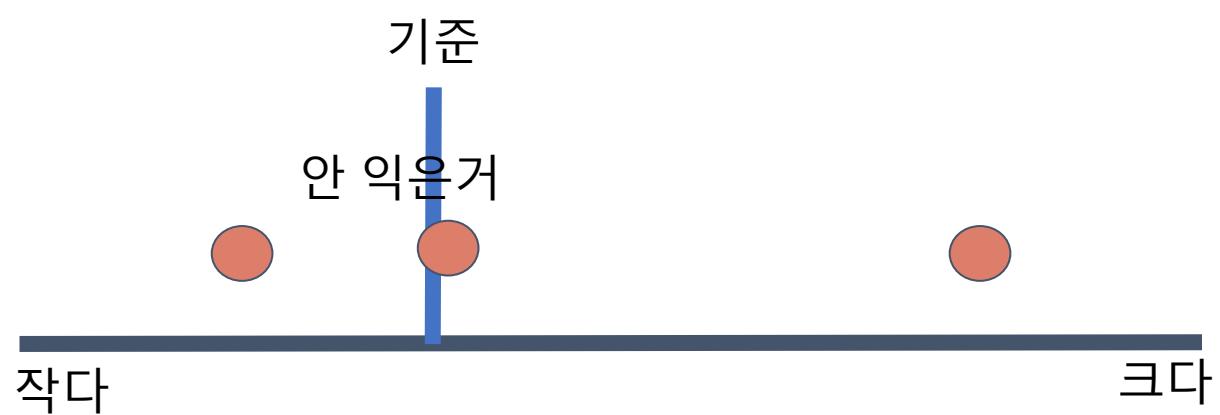
# 사과 기준 찾기



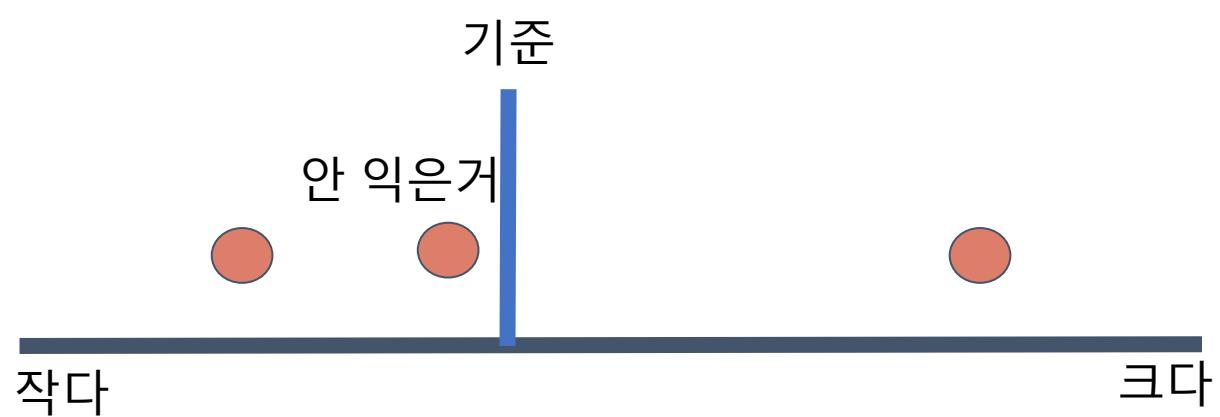
# 사과 기준 찾기



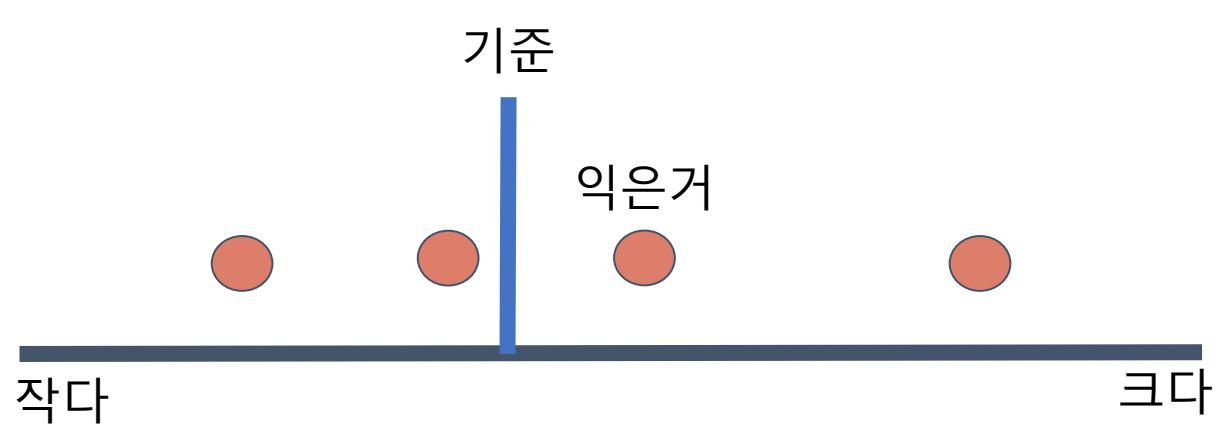
# 사과 기준 찾기



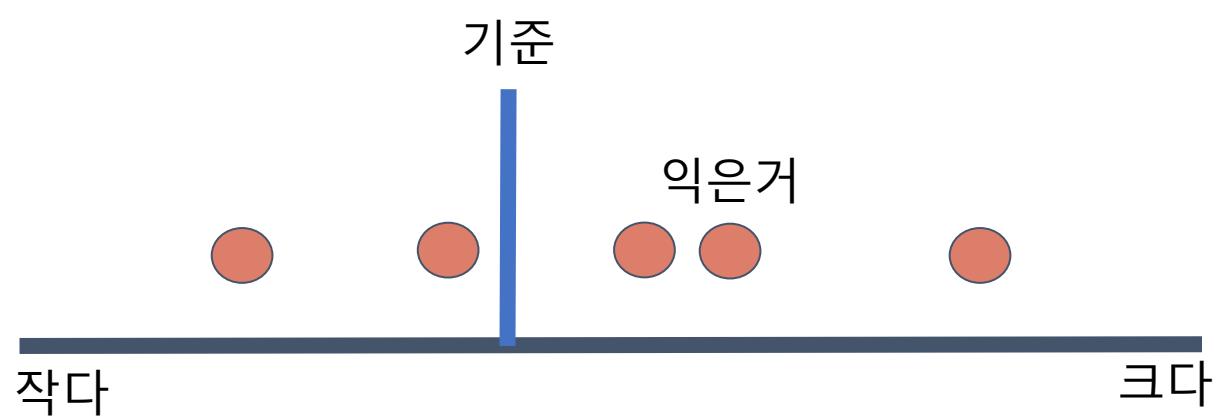
# 사과 기준 찾기



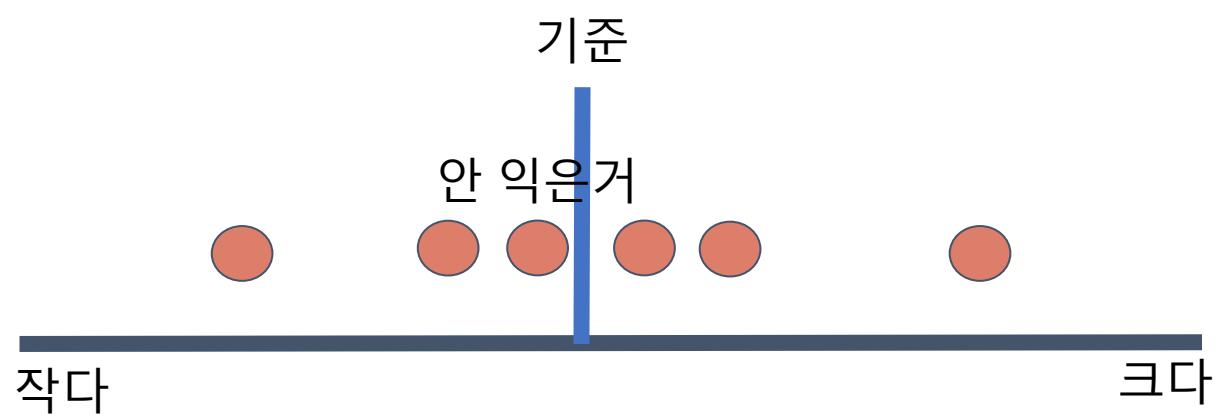
# 사과 기준 찾기



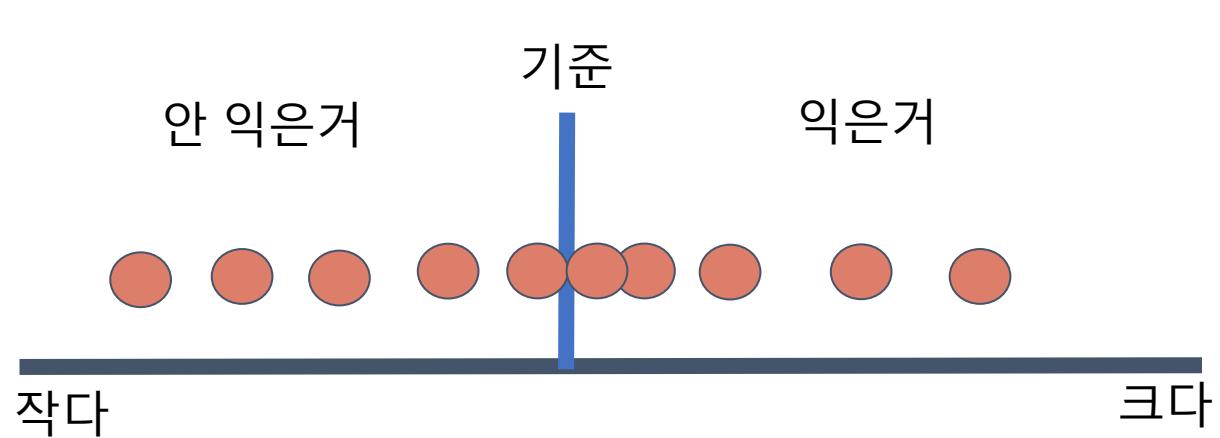
# 사과 기준 찾기



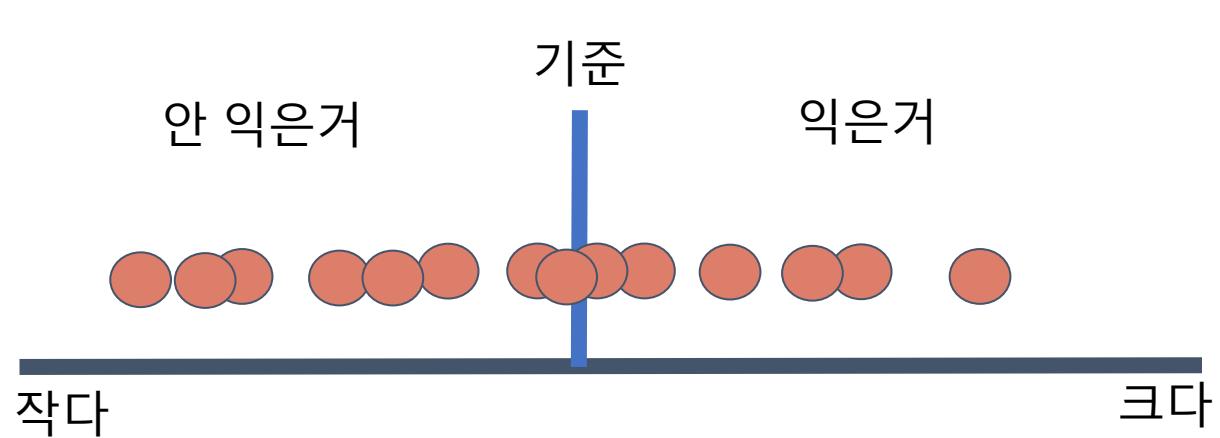
# 사과 기준 찾기



# 사과 기준 찾기



# 사과 기준 찾기 – 찾았다!



# 분류기

입력을 받아 몇개의 그룹으로 분류한다.

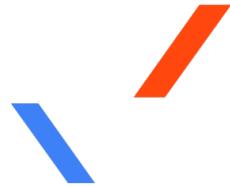


귤은 내가 기준을 찾았다. (직접 코딩한 알고리즘)  
사과는 기계가 기준을 찾았다.(기계가 찾은 알고리즘)

하지만 둘다 목적과 쓰임새는 동일.

# 용어

- 분류를 위해 ‘크기를 재서 기준 크기를 가지고 구별’ ←model
- 학습 시키려면 사과 2상자가 필요. 익은거 상자, 안익은거 상자.  
←training set
- 여러번 반복해서, ← iteration, step, epoch
- 원하는 기준 크기를 찾아냄. ← learning, training
- 그리고 구분 안된 바구니에 적용. ← test set, evaluation set





분류하는 선

# 사과 분류 적용

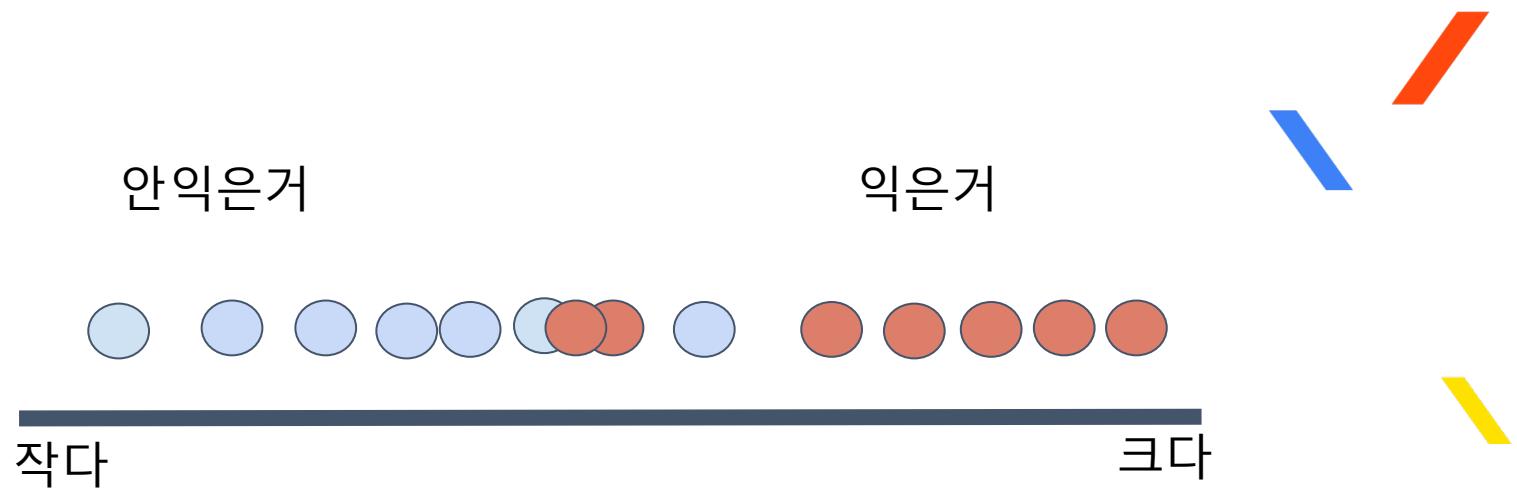
사과 분류를 학습한 기준으로 적용했더니, 결과가 않좋다.

Training set으로 100% 하더라도, test set은 결과가 안좋을 수 있다.

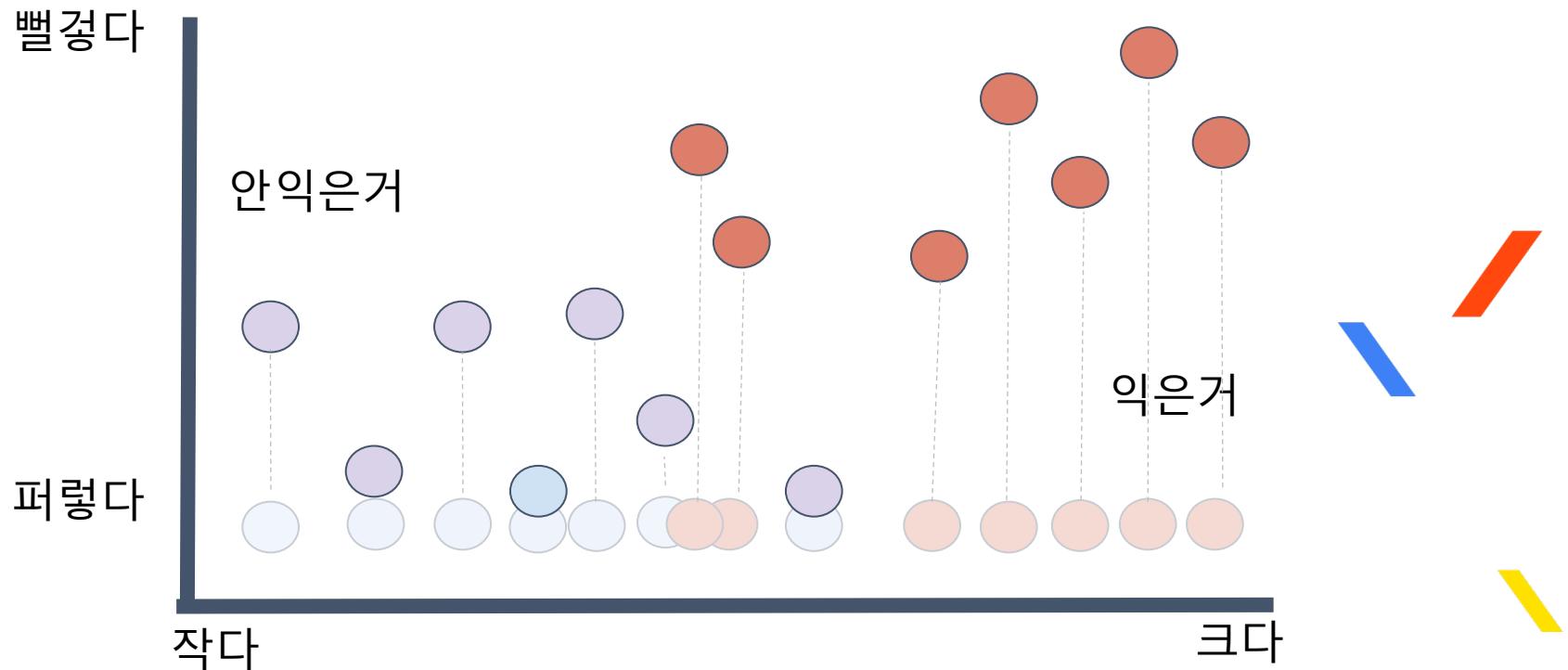


# 사과 분류 적용

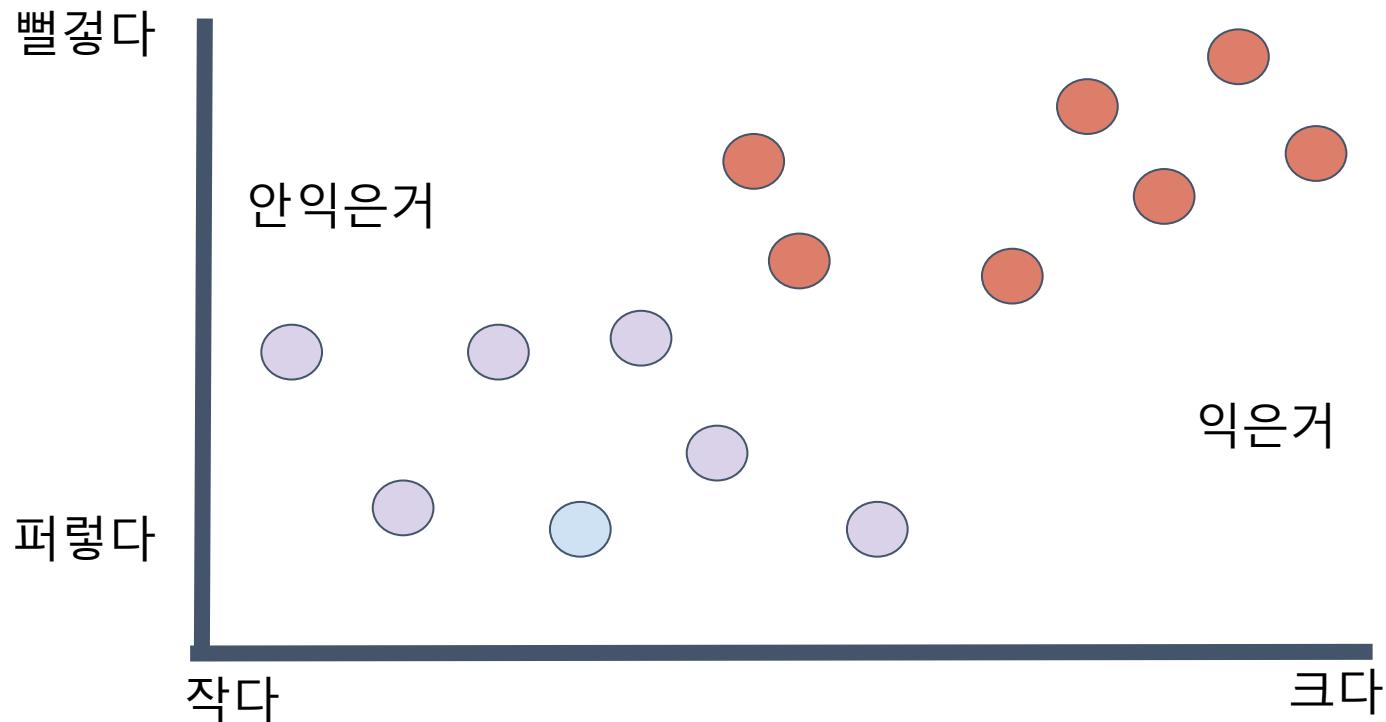
분류해 놓은 것을 확인해 보니 크기가 큰데도, 색깔이 덜 빨깐 사과  
가 있다.



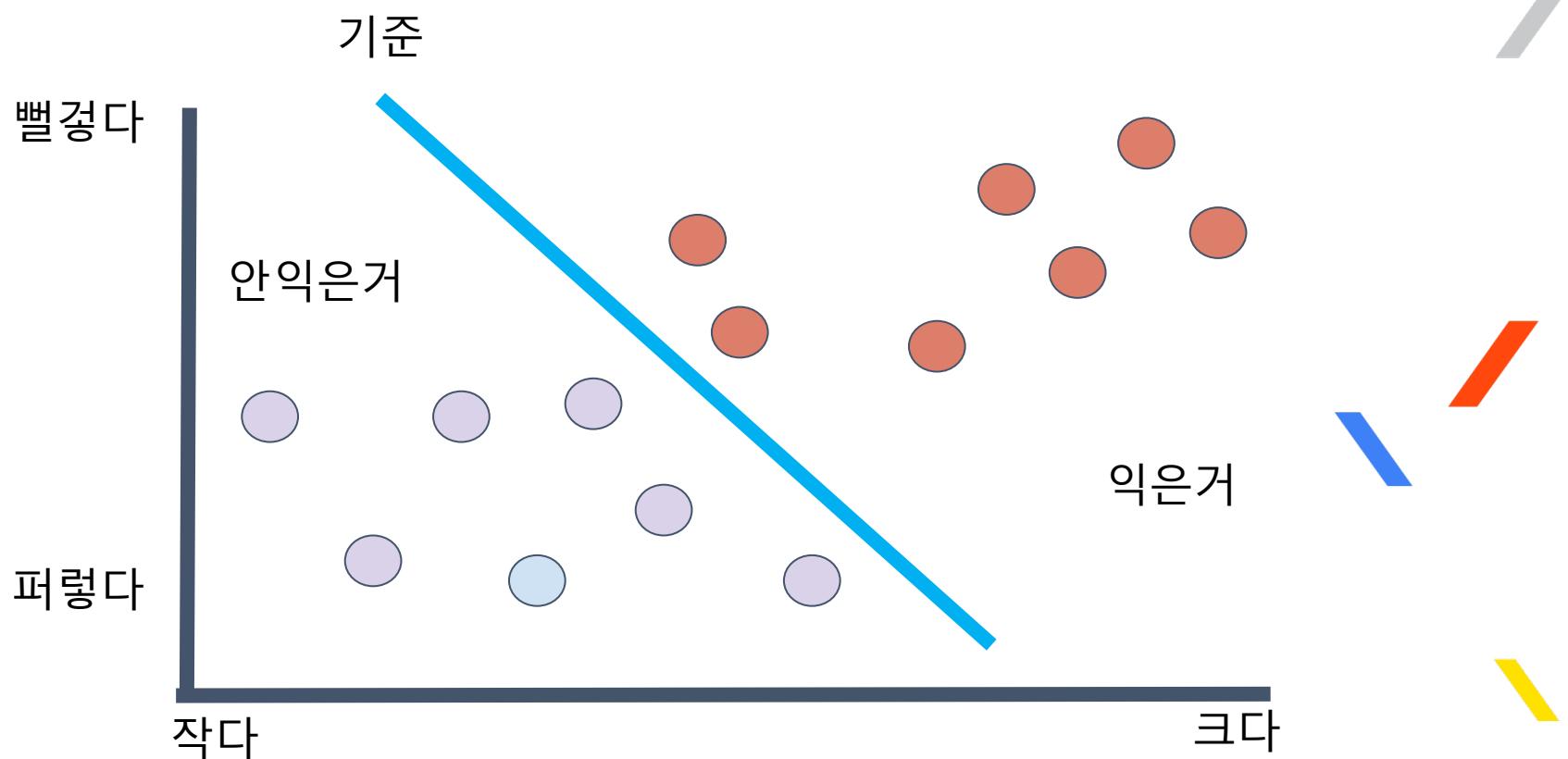
# 기준을 다시 찾자 – 색깔을 고려



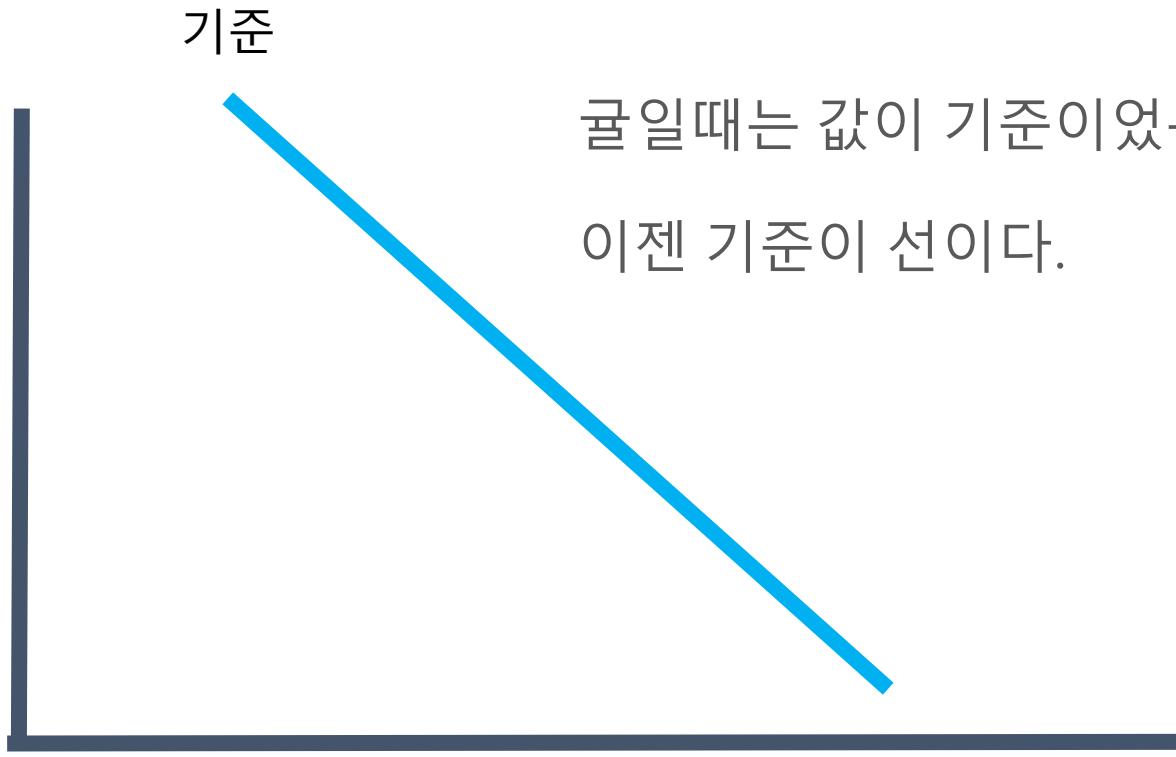
# 기준을 다시 찾자 – 색깔을 고려



# 기준을 다시 찾자 – 가르자

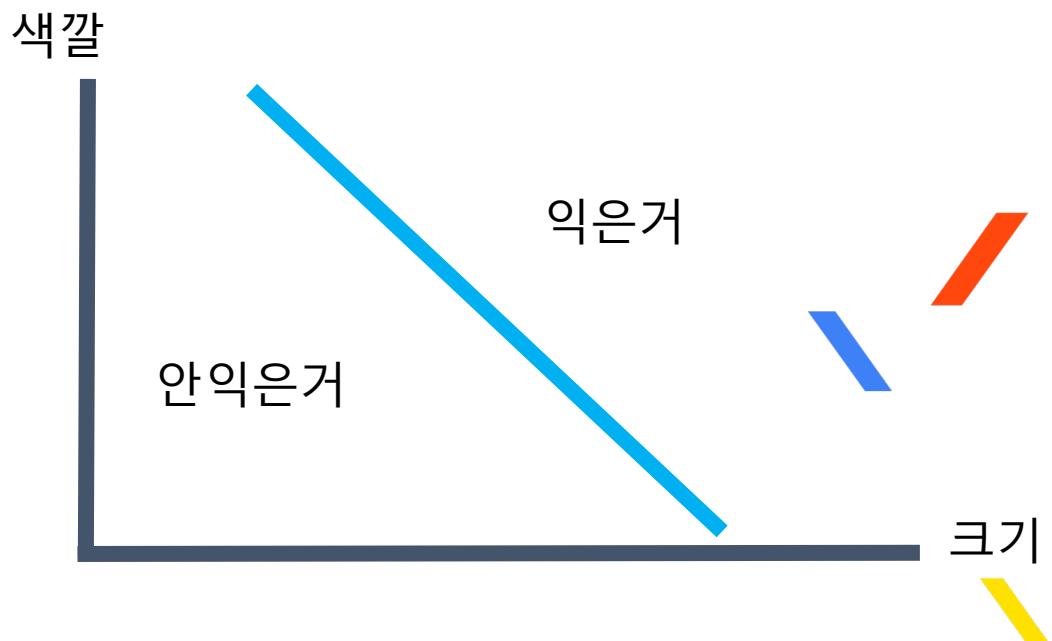


# 기준이 선이다



# 새 기준의 의미

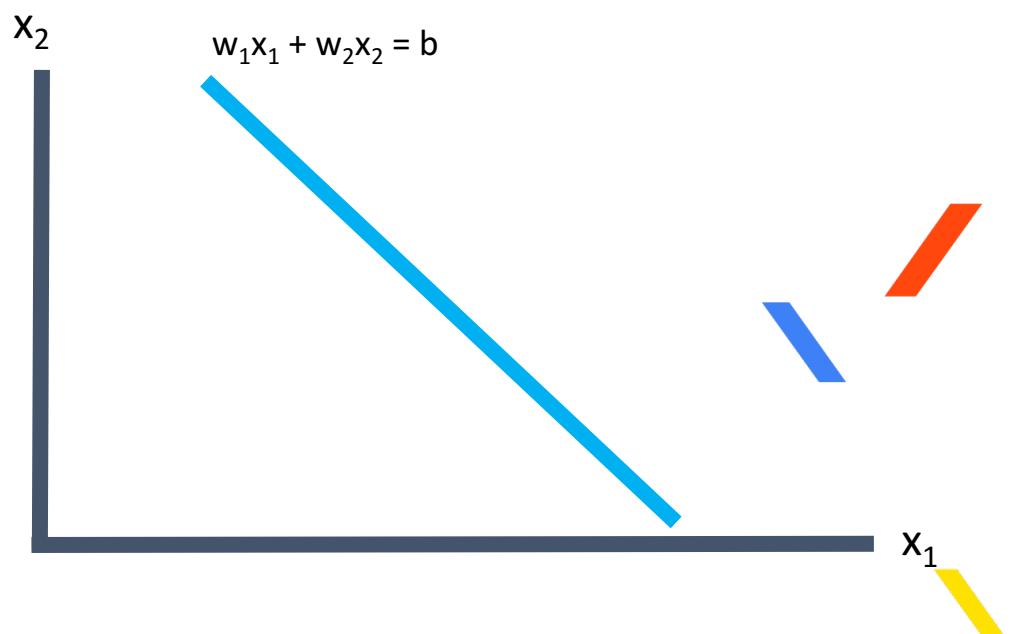
크기하고 색깔하고 같이 고려해서 어느정도 이상이면 익은거



# 처음이자 마지막 수식

선은 다음 식으로 표현

$$w_1x_1 + w_2x_2 = b$$



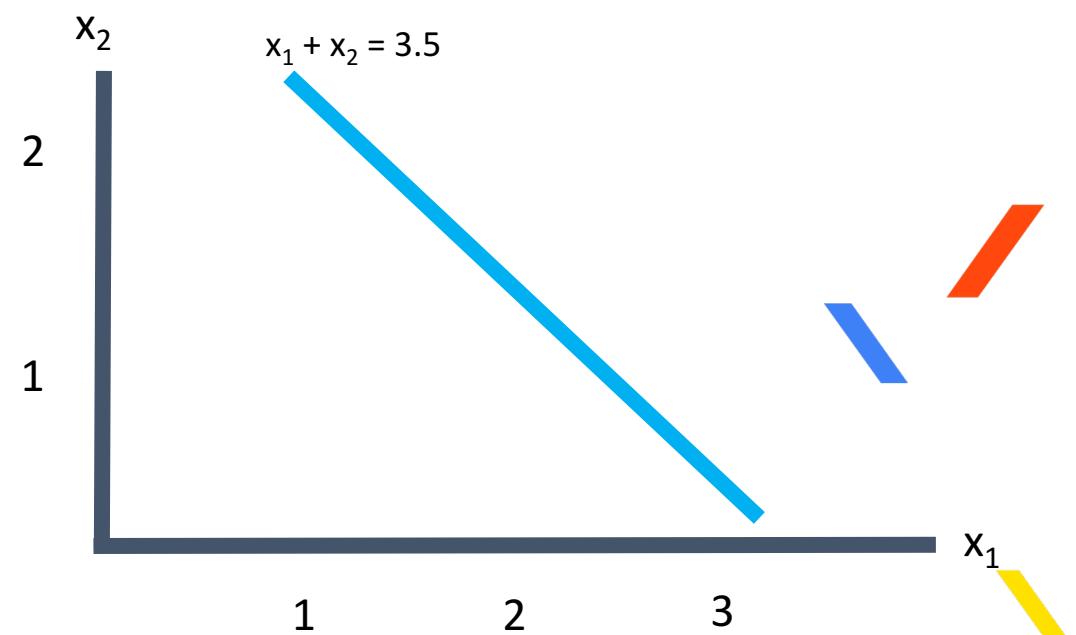
# 선

$$x_1 + x_2 = 3.5$$

크기 정도( $x_1$ )하고

색깔 정도( $x_2$ )를 합쳐서

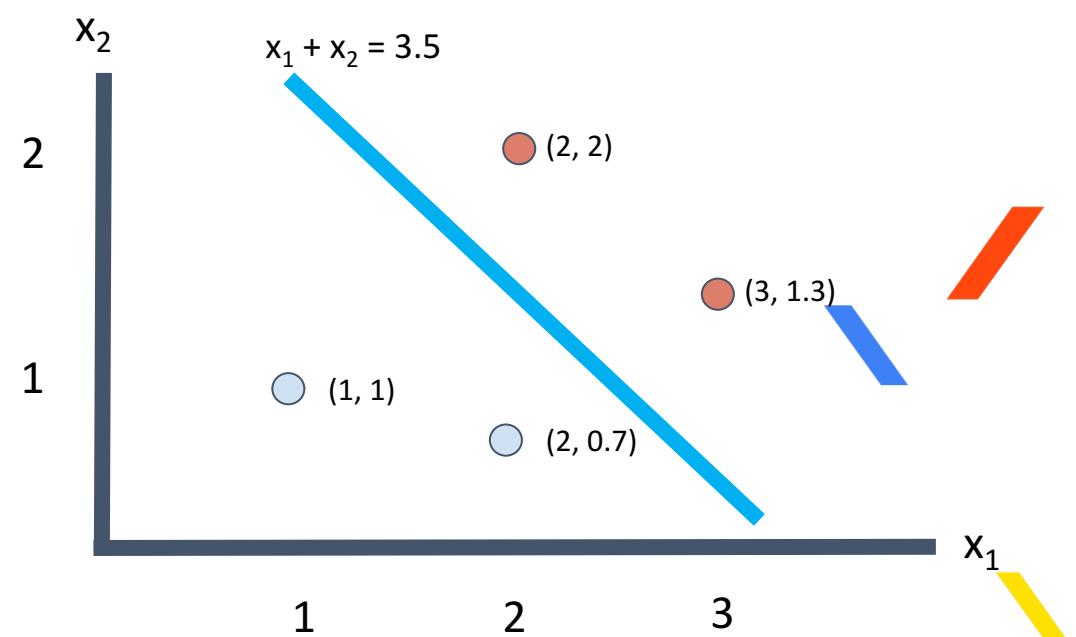
3.5정도인 거



# 선과 점

$$x_1 + x_2 = 3.5$$

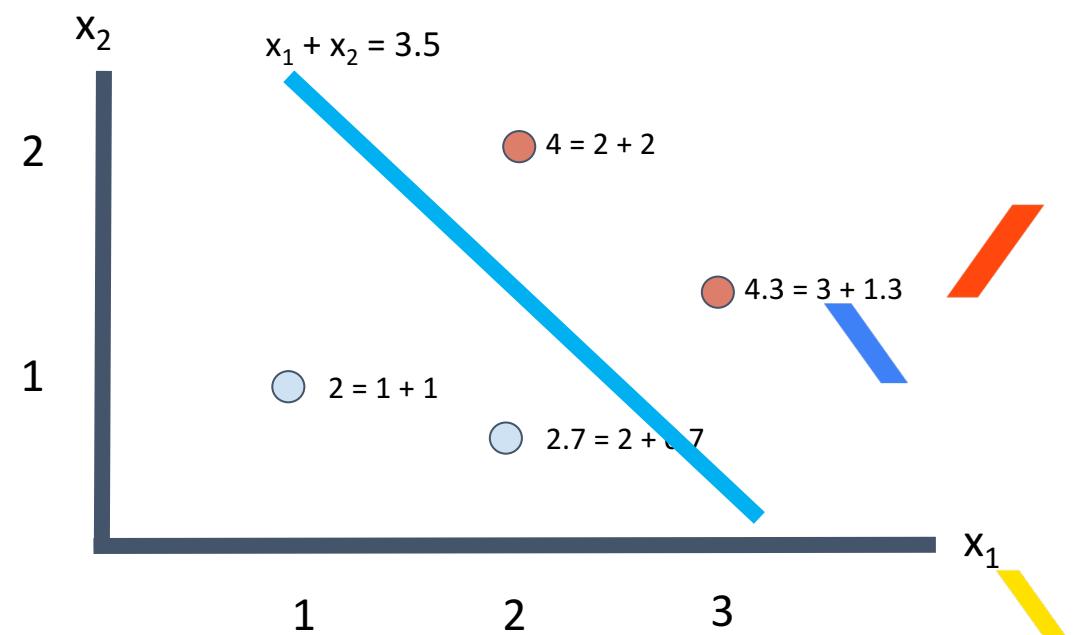
크기 정도( $x_1$ )하고  
색깔 정도( $x_2$ )를 합쳐서  
3.5정도인 거



# 점의 값

$$x_1 + x_2 = 3.5$$

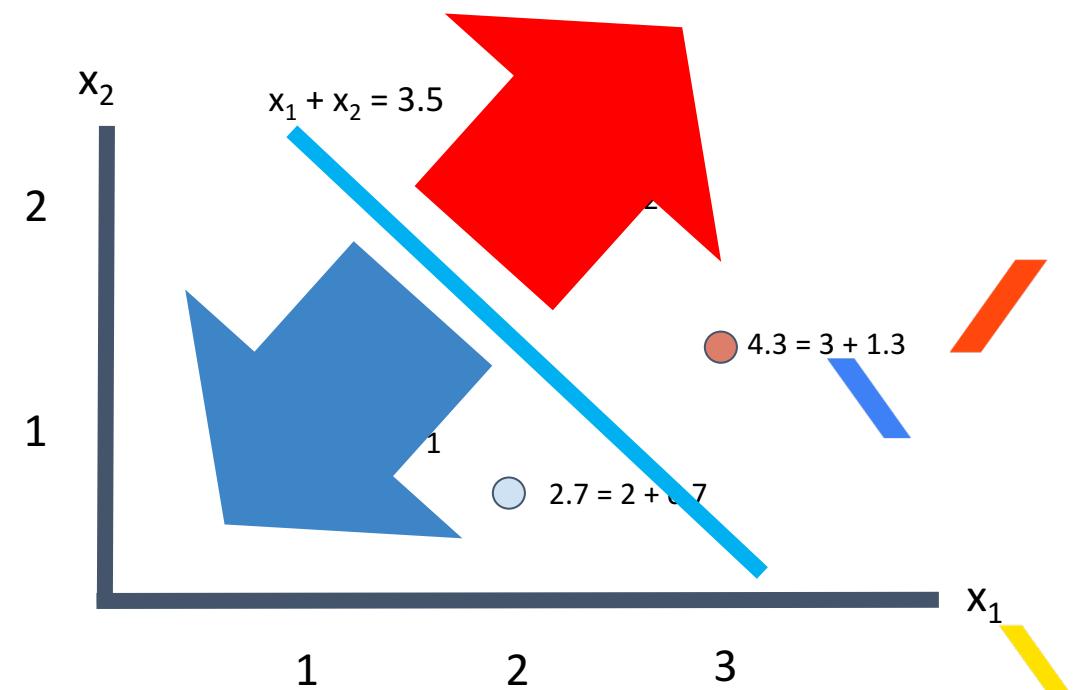
크기 정도( $x_1$ )하고  
색깔 정도( $x_2$ )를 합쳐서  
3.5정도인 거



# 선으로 구분된 영역

$$x_1 + x_2 = 3.5$$

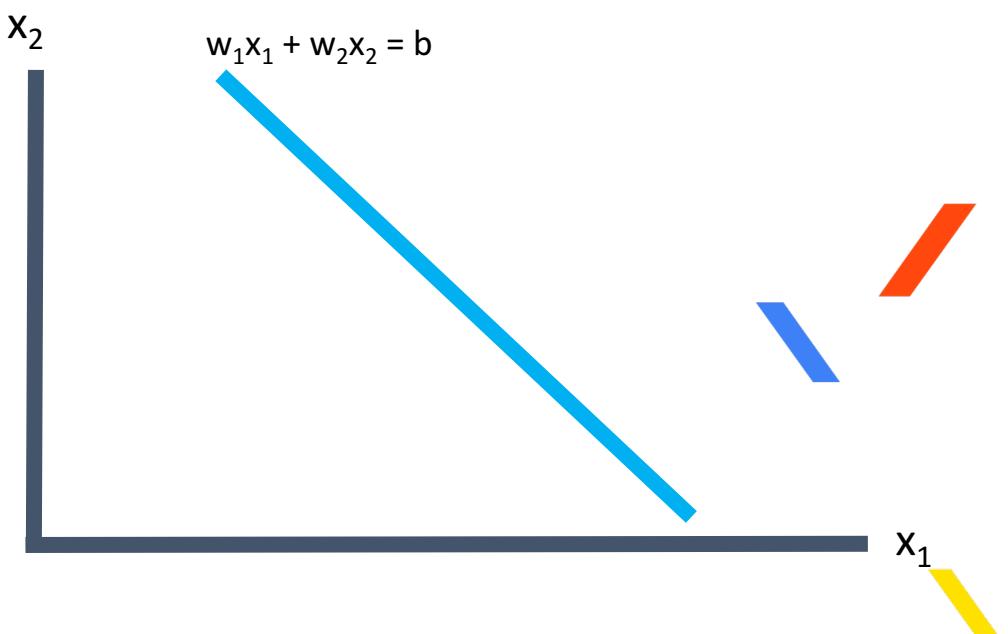
크기 정도( $x_1$ )하고  
색깔 정도( $x_2$ )를 합쳐서  
3.5정도인 거



# 기준을 찾는다는 건

선을 표현하는  $w_1, w_2, b$ 을 찾는 것.

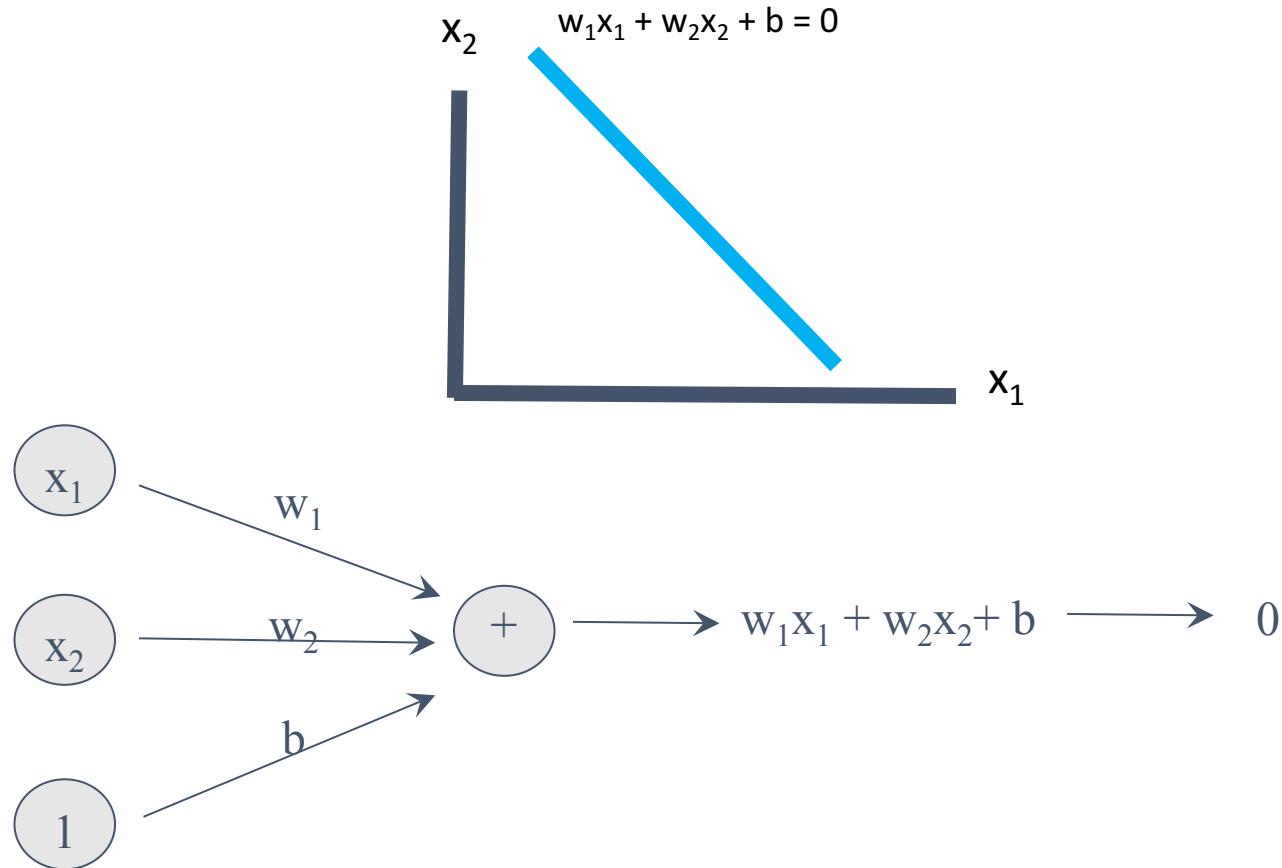
$$w_1x_1 + w_2x_2 = b$$





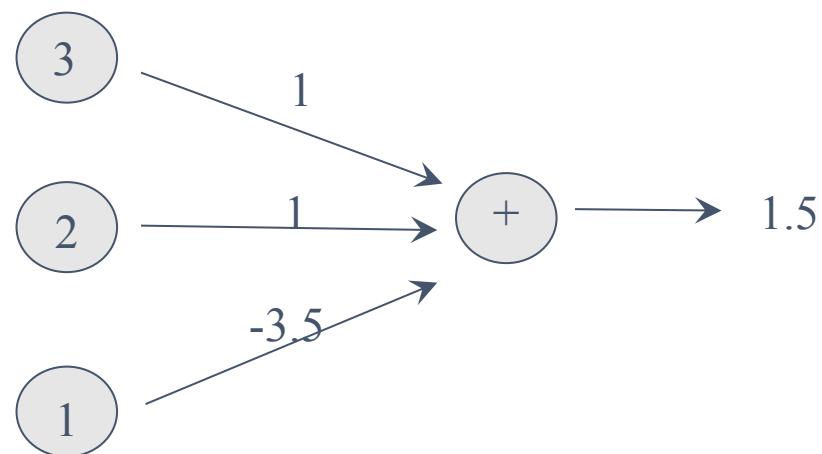
분류하는 선을 찾자

# 선식의 다른 표현



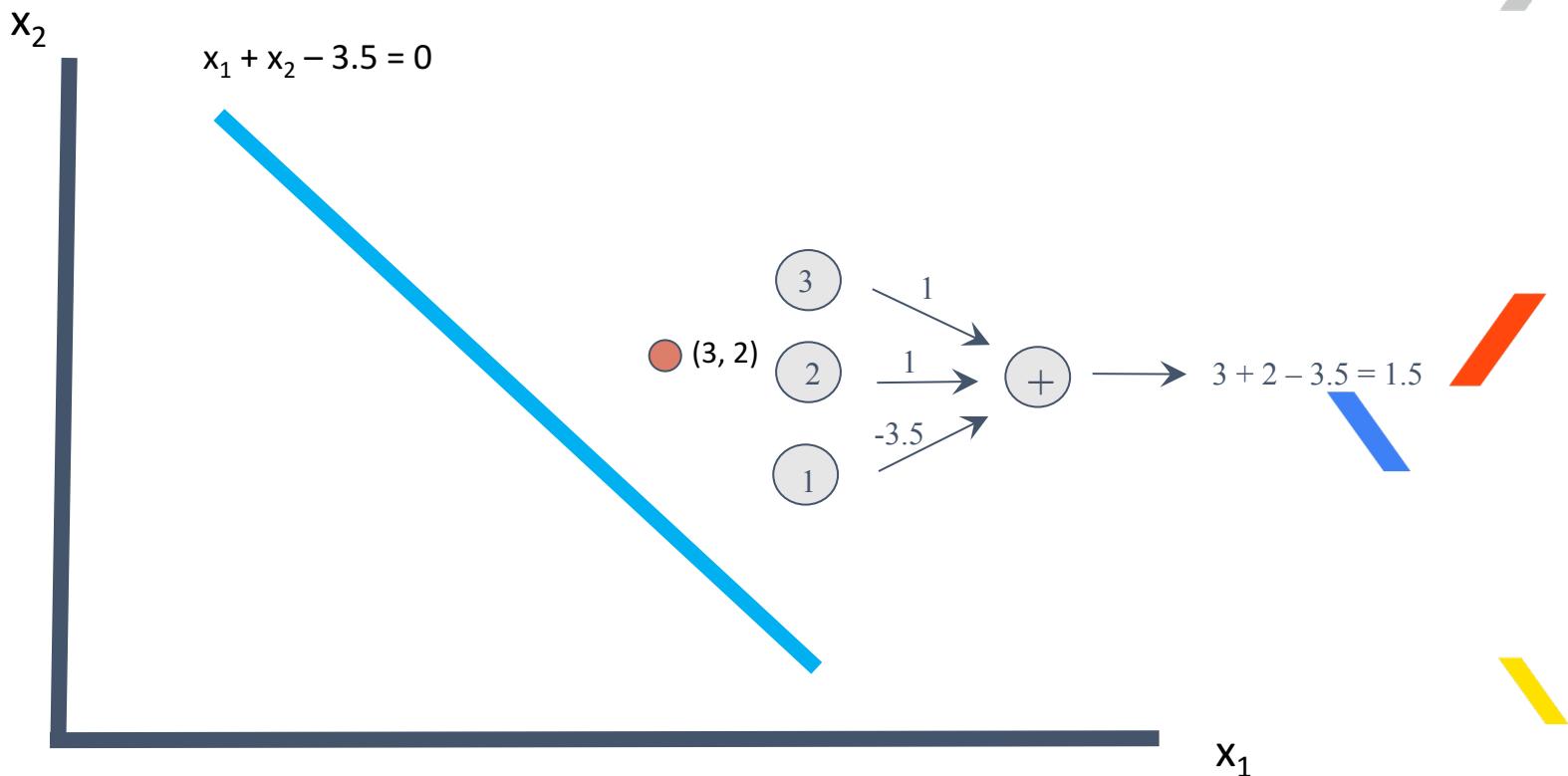
# 그림 표현의 예

- 크기가 3이고, 색깔이 2인 경우
- 계산 값은 1.5



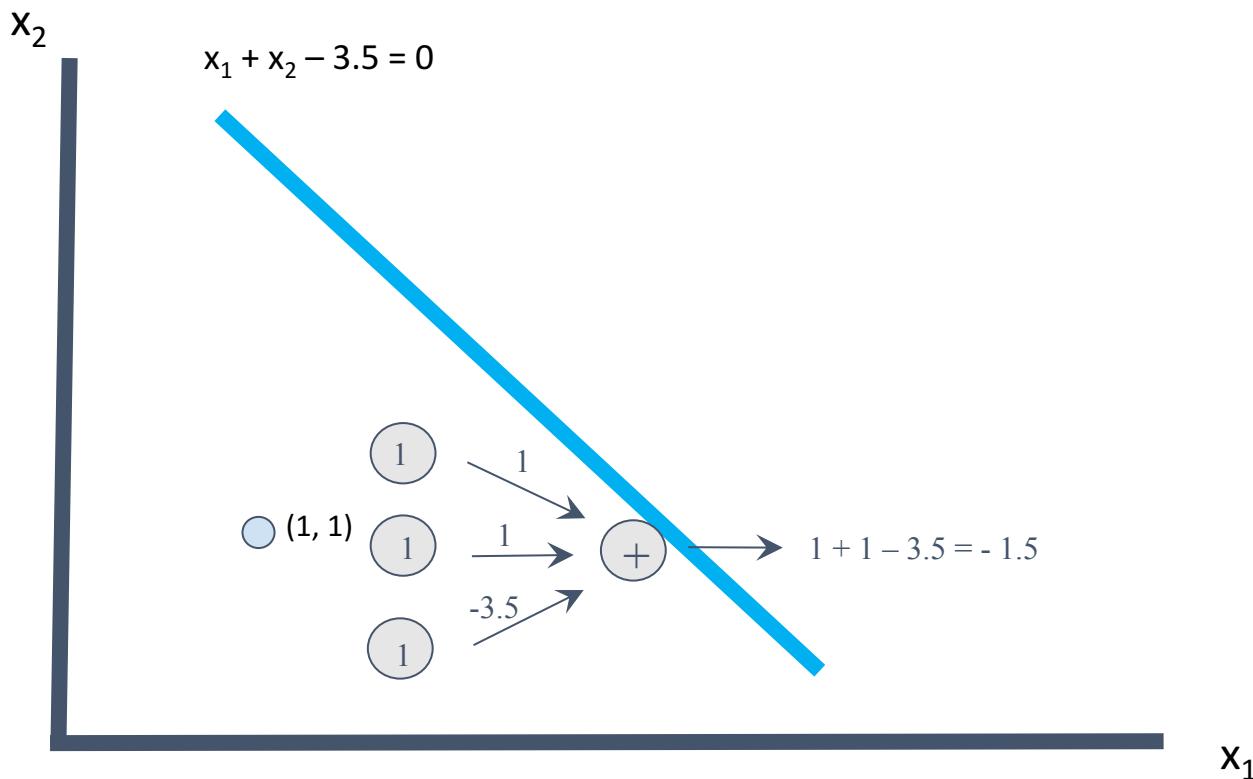
# 익은 사과

크기가 3, 색깔이 2. 계산 값은 1.5  
0 보다 크다. 선 우측에 있다.



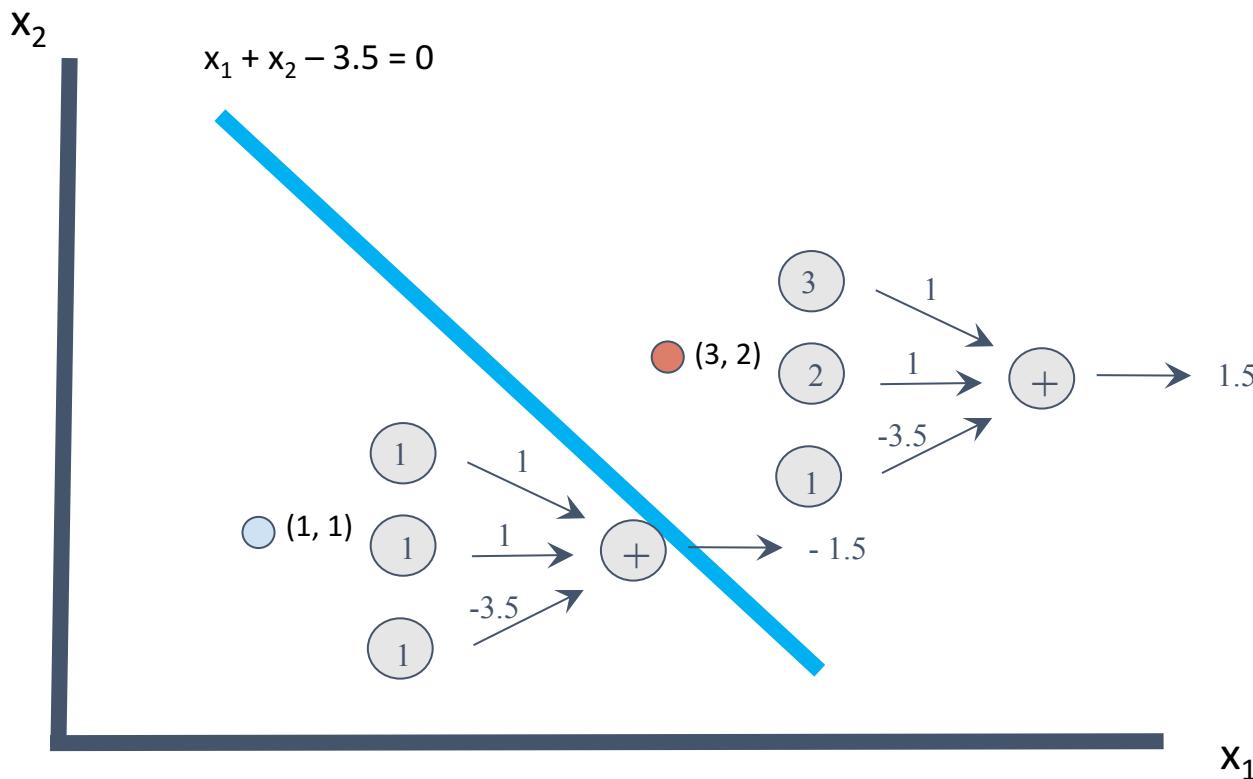
# 안 익은 사과

크기가 1, 색깔이 1. 계산 값은 -1.5  
0 보다 작다. 선 좌측에 있다.



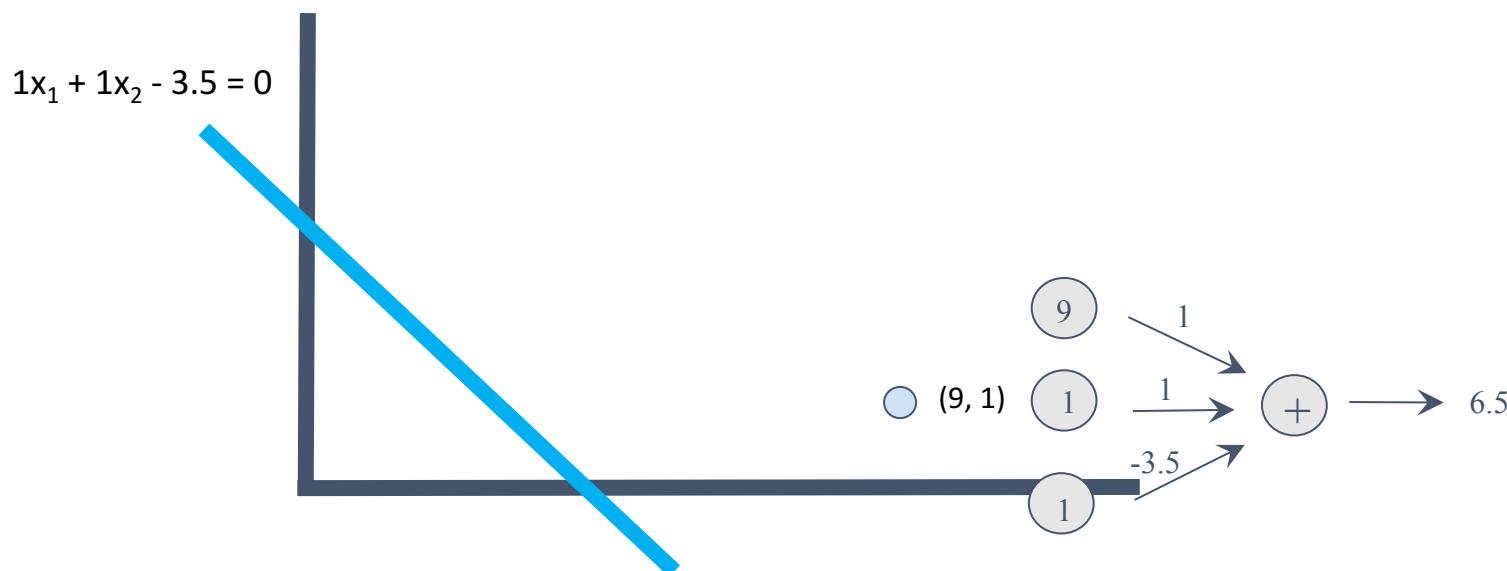
# 그림 표현

0보다 큰 것은 우측  
0보다 작은 것은 좌측



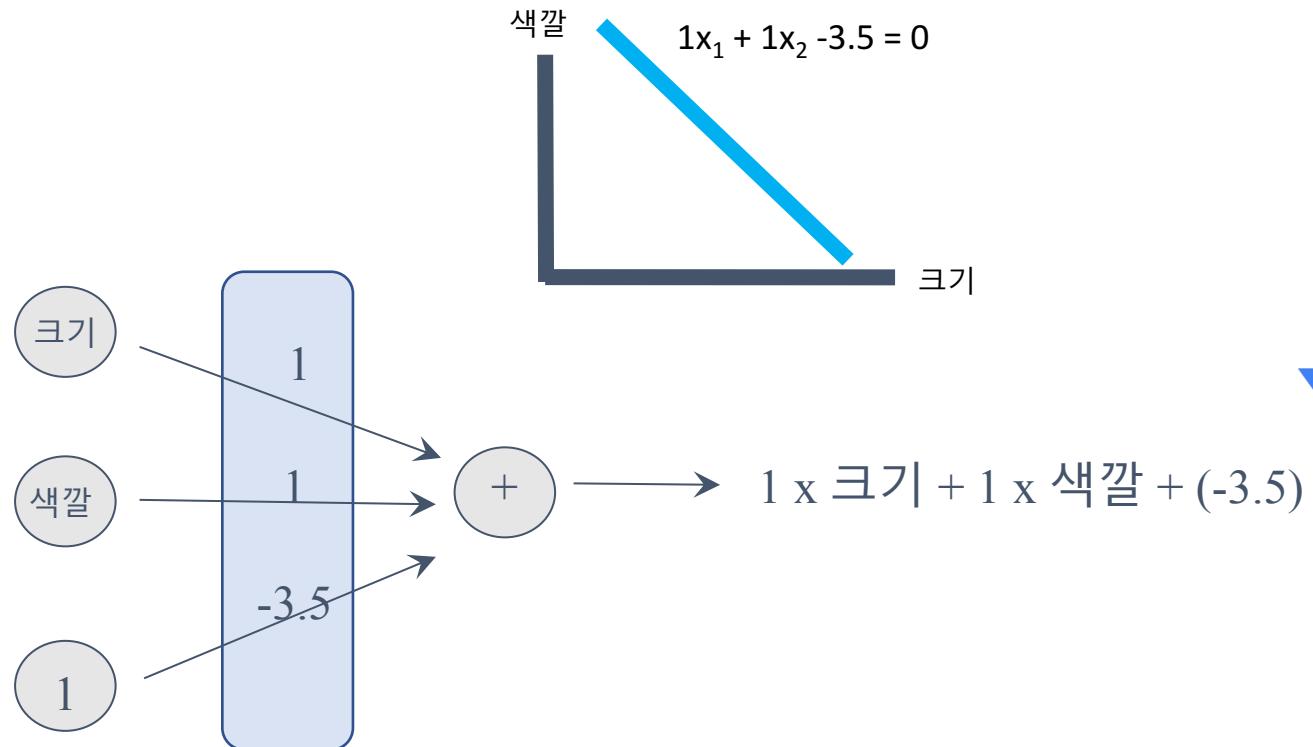
# 잘못된 분류

크기가 9이고 색깔이 1.  
안익었으니 계산값이 0보다 작아야 하는데 크다.



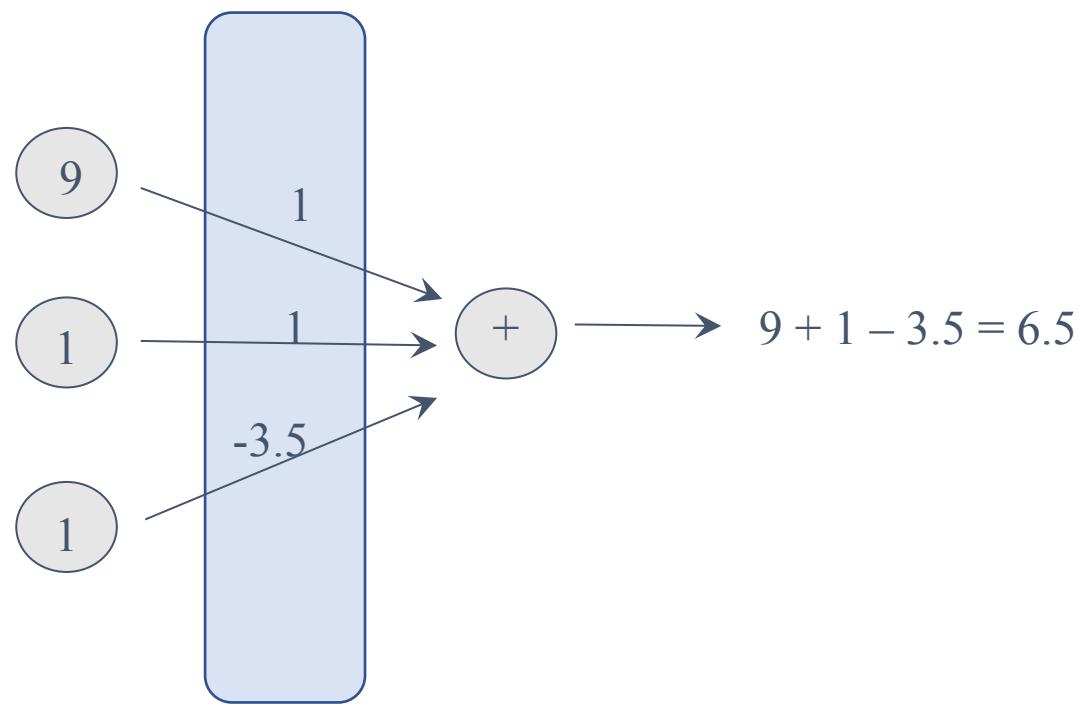
# 그림 표현

그림표현에서의 화살표 위의 숫자가 기준 선을 의미.



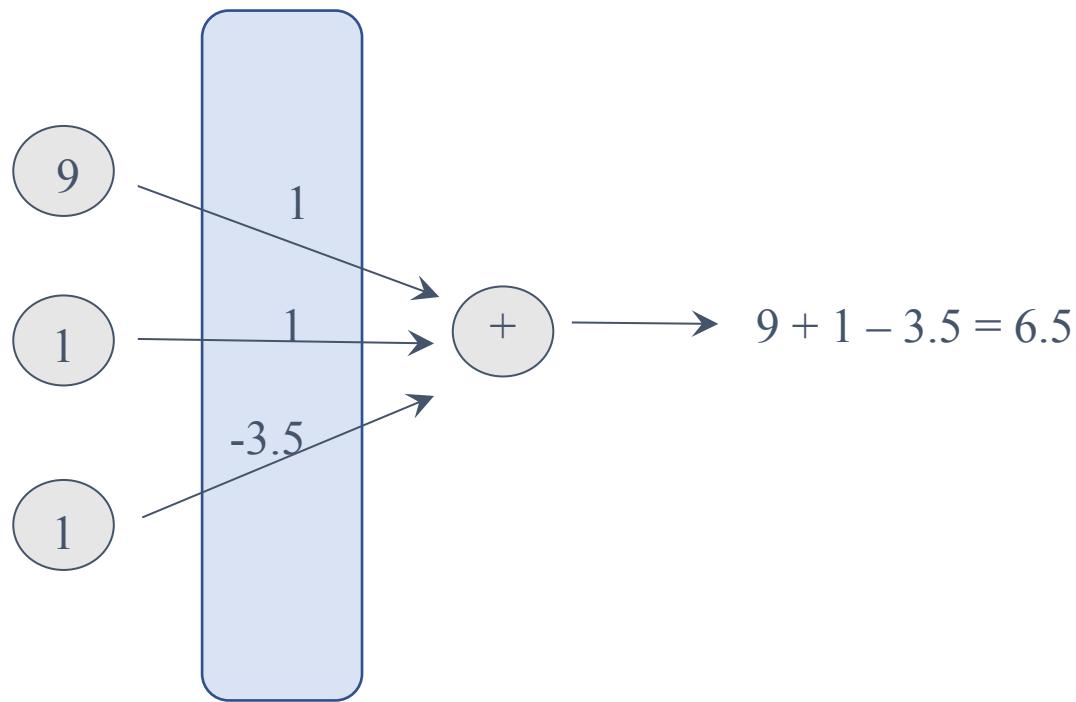
# 선 조정 필요

선을 나타내는 1, 1, -3.5의 값을 조정해서  
계산 값이 0보다 작게 해야 한다.



# 선 조정 아이디어

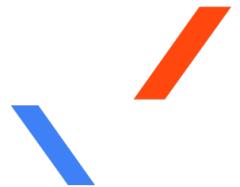
계산 값에 왕창 영향 끼치는 크기(9)의 것을 색깔(1) 보다 더 크게 조정하자.



# 선 조정 아이디어

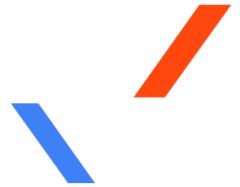
계산 값에 왕창 영향 끼치는 크기(9)의 것을 색깔(1) 보다  
더 크게 조정하자.

물론 잘 분류했을 때에는 조정할 필요 없고.



# 선 조정 아이디어

조정 값 = 오차 \* 입력값 \* 학습율



# 선 조정 예

조정 값 = 오차 \* 입력값 \* 학습율

조정된 크기 비중 = 이전 크기 비중 - ( 1 x 9 x 0.1 )

조정된 색깔 비중 = 이전 색깔 비중 - ( 1 x 1 x 0.1 )

조정된 기준값 = 이전 기준값 - ( 1 x 1 x 0.1 )

# 선 조정 예

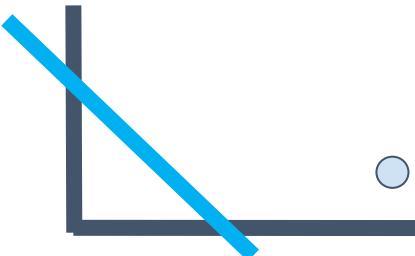
조정 값 = 오차 \* 입력값 \* 학습율

$$\text{크기 비중} = 1 - (1 \times 9 \times 0.1) = 0.1$$

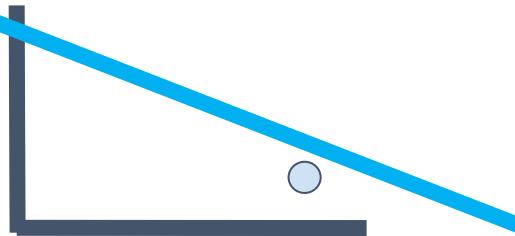
$$\text{색깔 비중} = 1 - (1 \times 1 \times 0.1) = 0.9$$

$$\text{기준값} = -3.5 - (1 \times 1 \times 0.1) = -3.6$$

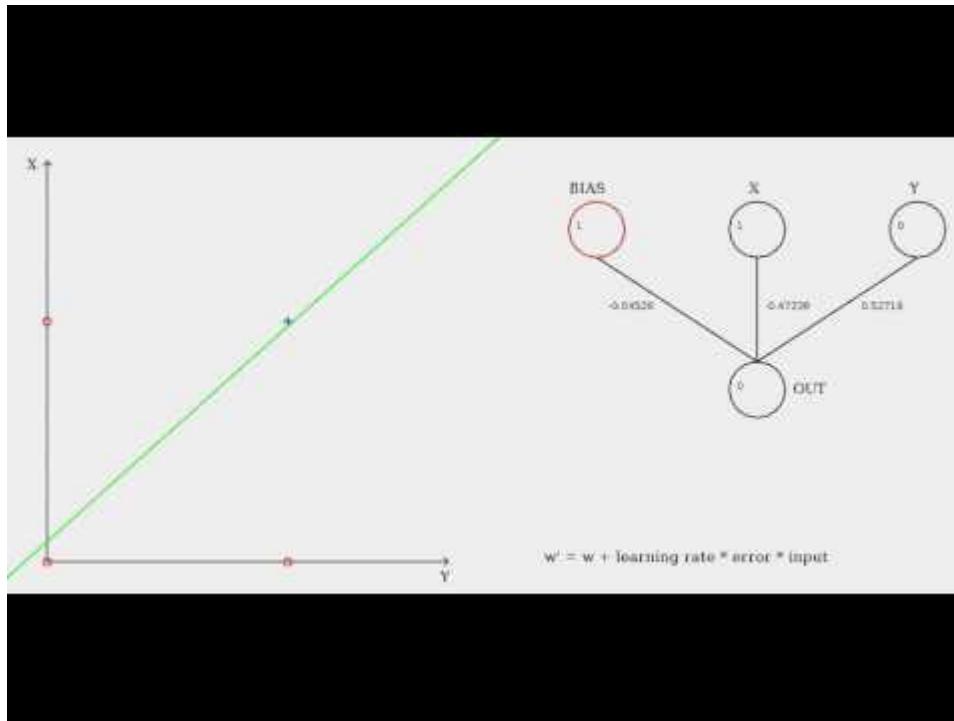
$$1x_1 + 1x_2 - 3.5 = 0$$



$$0.1x_1 + 0.9x_2 - 3.6 = 0$$

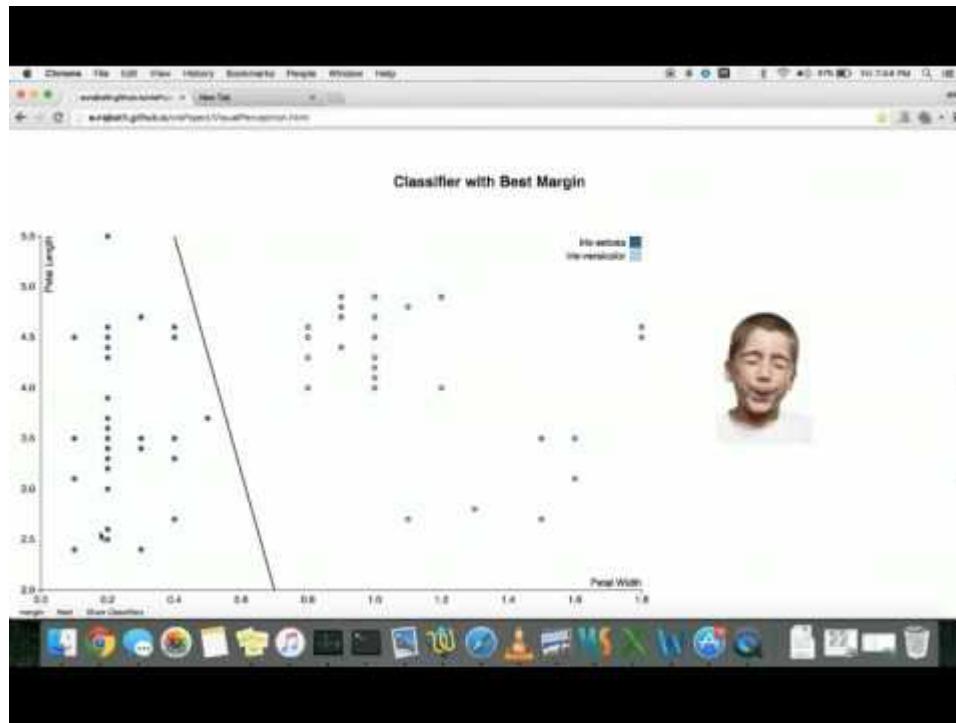


# 선의 학습 실 예



<https://www.youtube.com/watch?v=tYxkIOTdeu8>

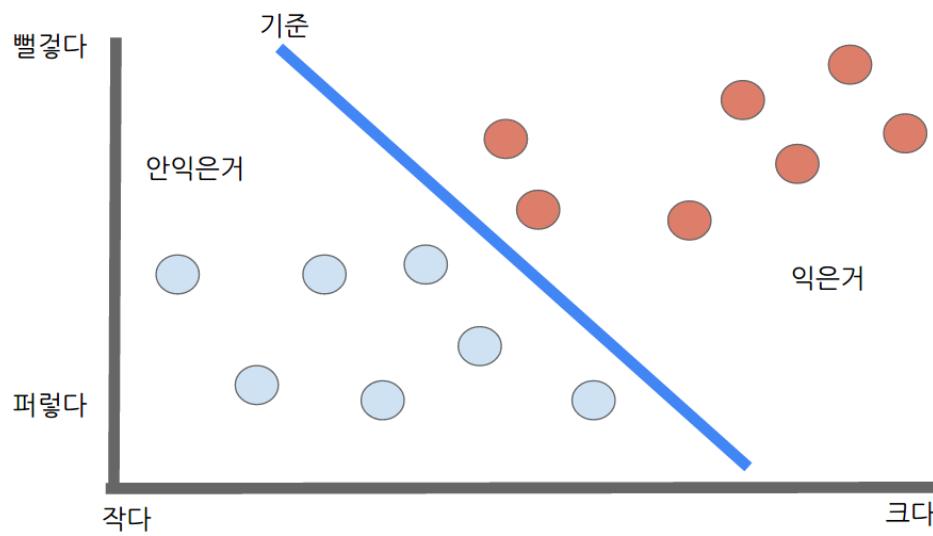
# 선의 학습 실 예



<https://www.youtube.com/embed/V2MMyk7bdWY>

# 선을 찾을 수 있다

실제로 저 간단한 로직으로 선을 찾을 수 있다.  
학습할 수 있다.



# 용어

- 크기마다 색깔을 추가하니 더 잘된다. ← 크기, 색깔을 입력, 특  
질(feature)라 한다. 
- 각 사과마다 2개 값의 특질이 있다. 크기 3, 색깔 2 ← 입력 벡터  
혹은 특질 벡터라 한다.
- 사과 예의 경우 2개 특질(크기, 색깔)을 사용했다. ← 입력 벡터가  
2차원. 
- 학습을 한다는 건 선을 구성하는 3개의 값을 찾는 것이다.
- 각 입력에 곱해지는  $w_1, w_2$ 들을 가중치(weight)라 칭한다. 

# 비용 함수와 학습

---

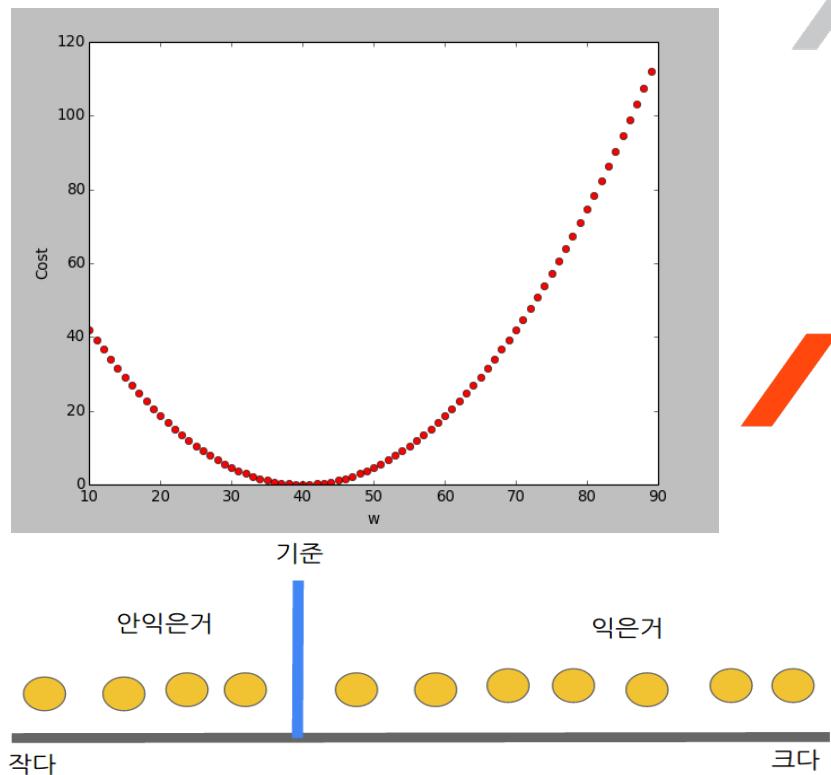
# 비용 함수(Cost Function)

율의 예에서, 그 기준의 값에 따라 오차의 크기가 결정된다.

결국 학습의 목표는 오차를 최소로, 혹은 오차함수의 값이 최소가되는  $w$ 를 찾는것.

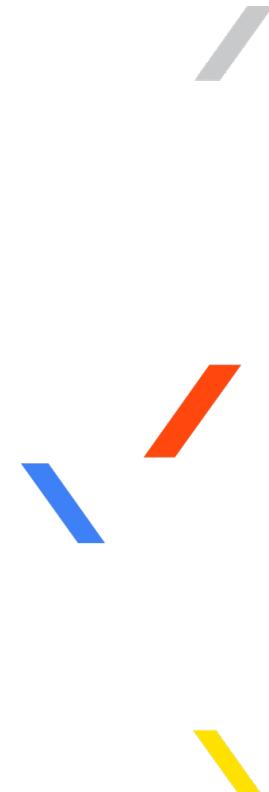
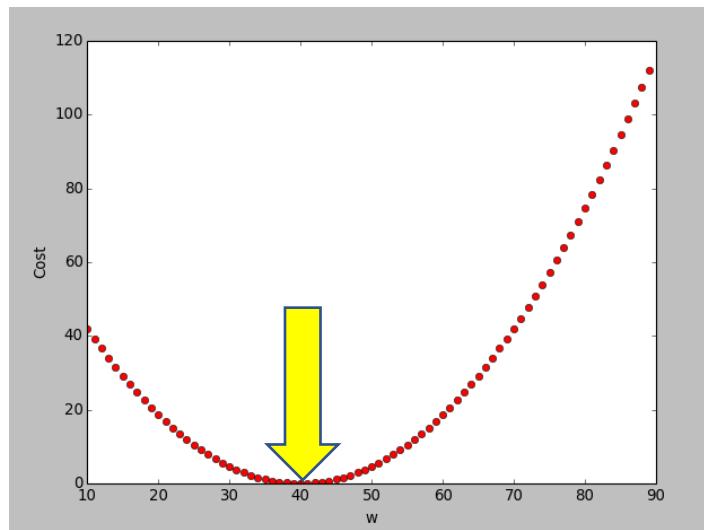
비용 함수는 모델을 구성하는 가중치( $w$ )의 함수이다.

예에서의 기준값에 의해 발생하는 오차의 정도.



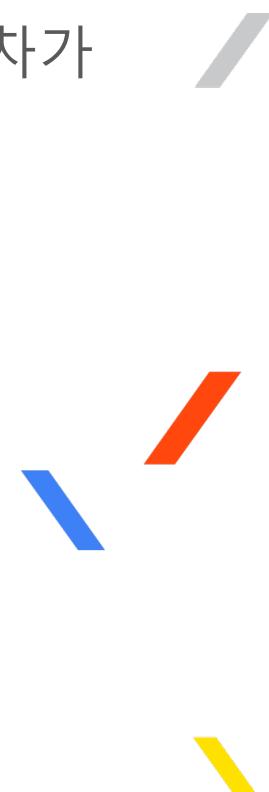
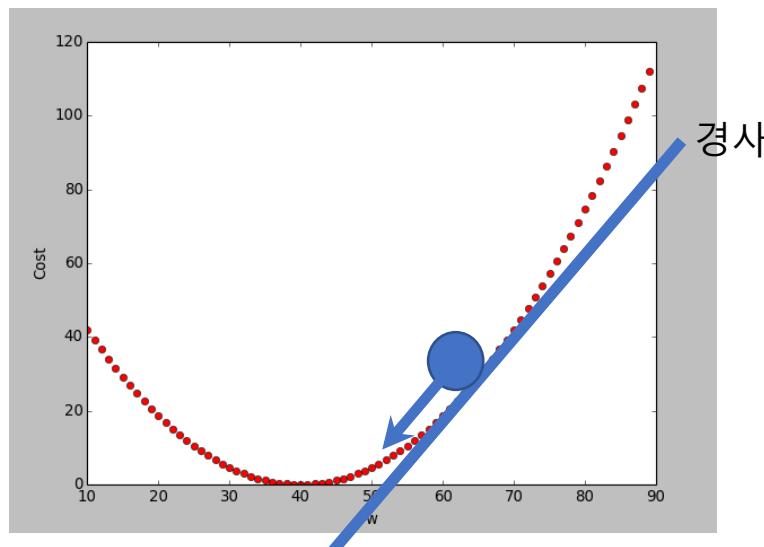
# 학습과 비용함수

비용함수를 최소로 하는  $w$ 를 찾는 것이 학습이다



# 경사 하강법(Gradient Descent)

오차평면에서 공을 올려 놓았을 때 공이 굴러가는 방향(오차가 적어지는 방향)으로  $w$ 를 조정한다.



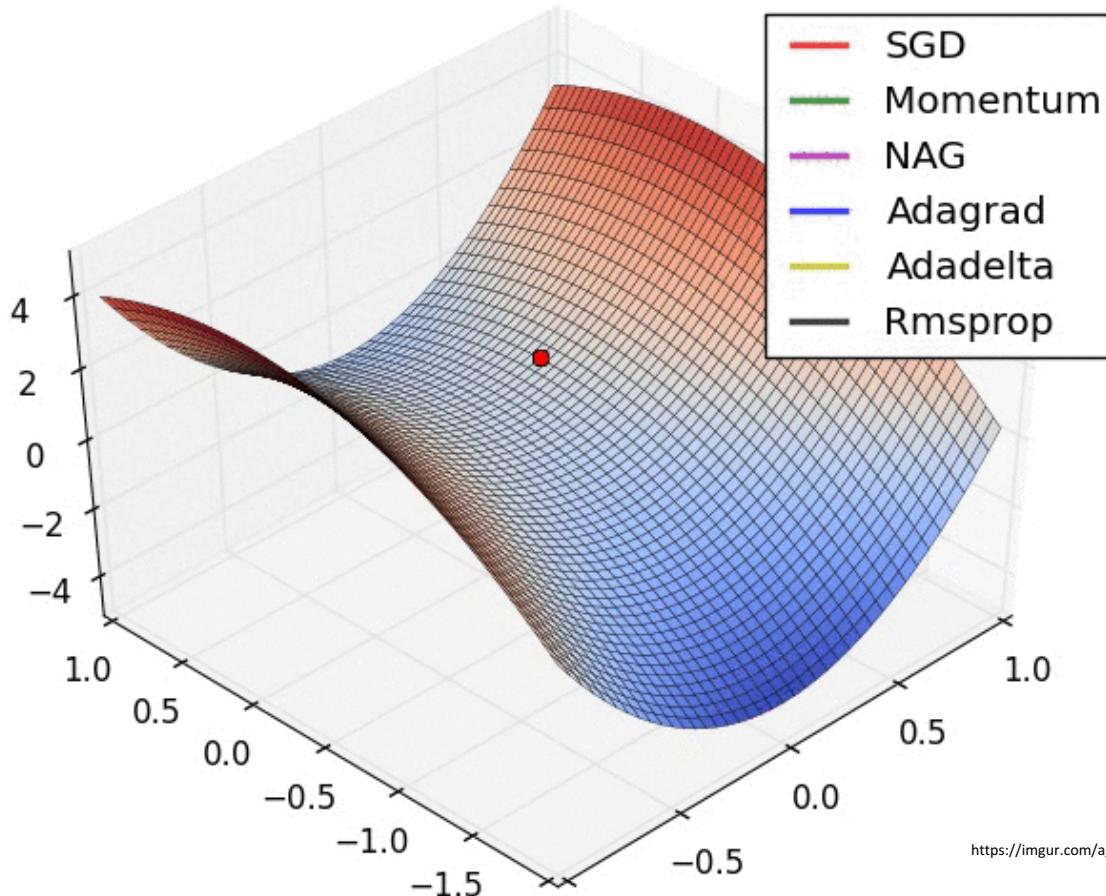
# 사과 예에서 학습방법

경사하강법은 비용함수를 구하고, 이를 각 weight 별로 편미분하면 각 weight 별로 수정할 다음 값을 구할 수 있다. 그렇다고 한다.

선형 모델( $w_1x_1 + w_2x_2 = b$ )에서는 경사하강법의 결과가 직관적으로 설명했던 방법과 동일하다.

결국 사과예에서 학습한 방법은 경사하강법을 사용한 것이다.

# 경사하강법 알고리즘들





신경망

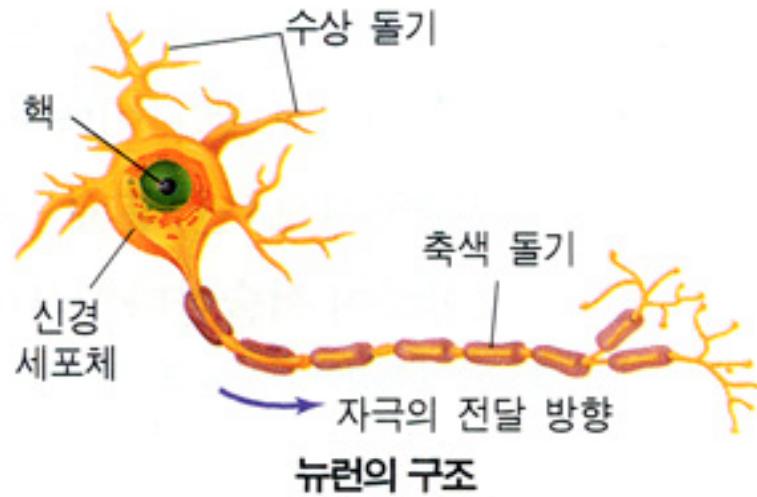


# 신경세포

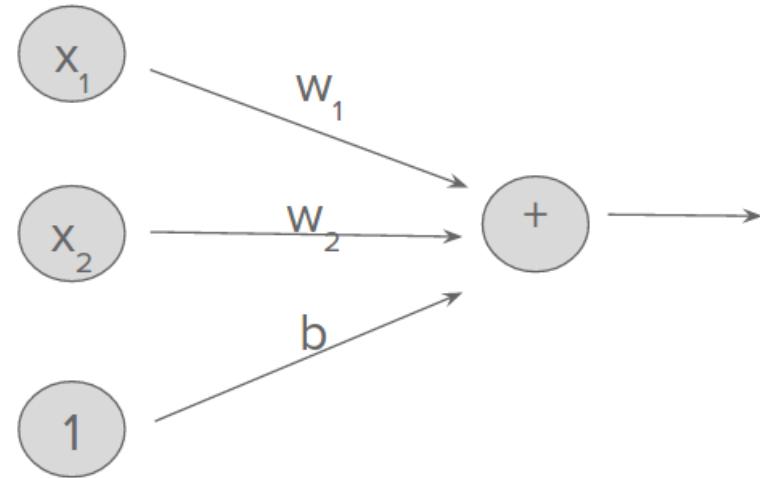
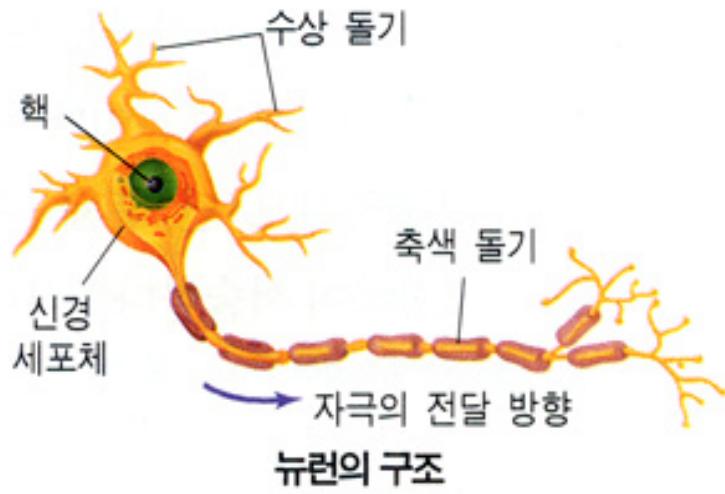
여러개의 수상돌기에서 자극이 합해져서

그 값이 어느 값 이상일 경우

축색돌기로 자극을 발생시킨다.

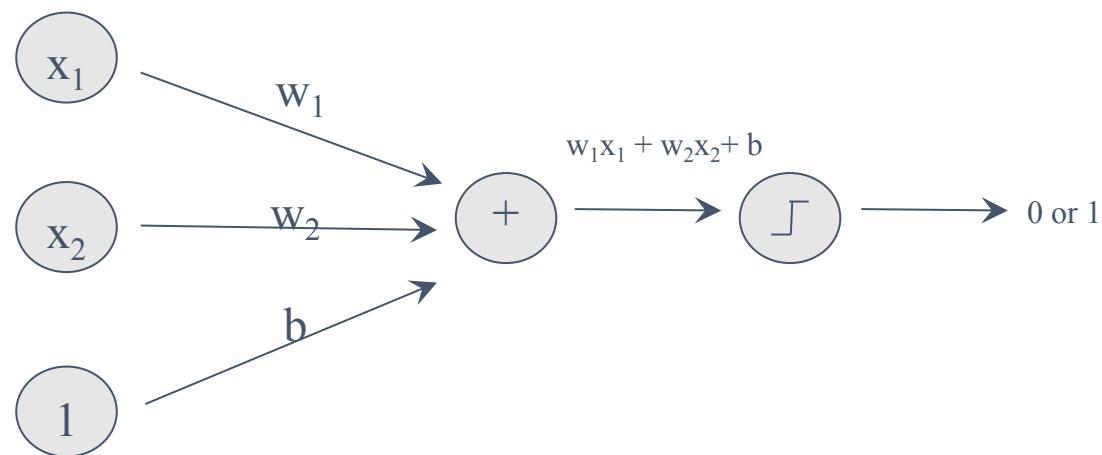


# 신경세포와 유사



# 퍼셉트론(Perceptron)

선형분리의 문제를 학습할 수 있다.

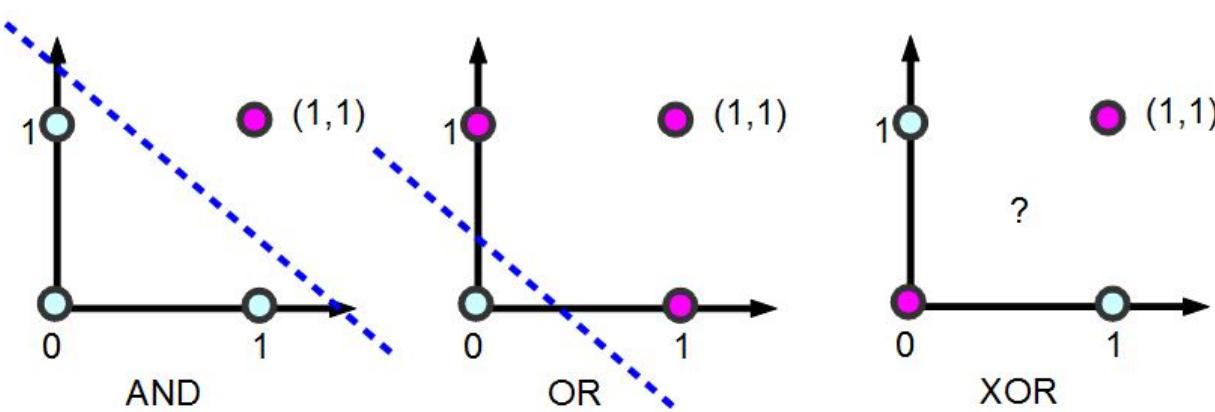


# 퍼셉트론의 한계

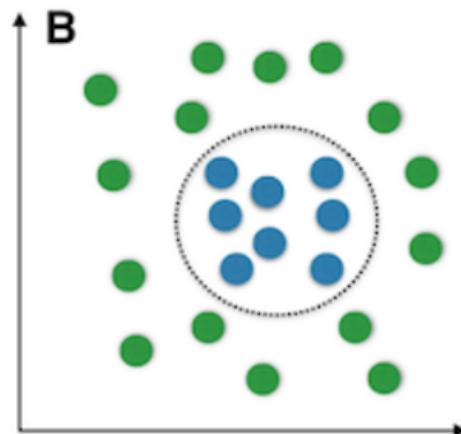
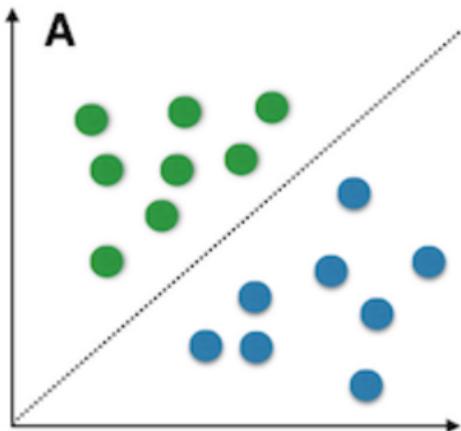
하지만 간단한 XOR도 못한다.

선형분리가 불가능한 것은 풀지 못한다.

XOR는 선형분리로 풀지 못하는 문제이다.

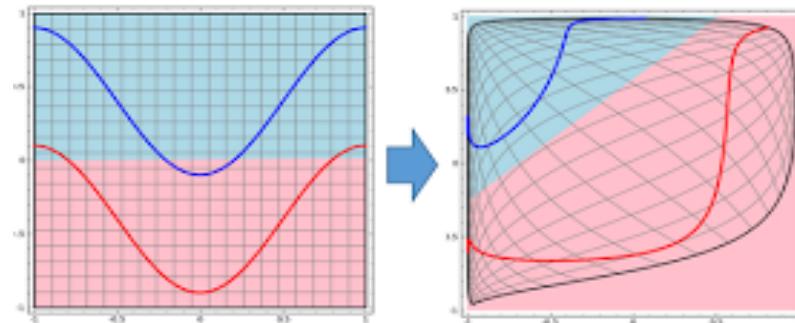


# 선형 분리 여부



# 선형 분리 불가 문제의 해결법

- 입력 차원을 늘린다.(사과 예)
- 입력을 비선형 변환하여 선형분리 가능하도록 한다.

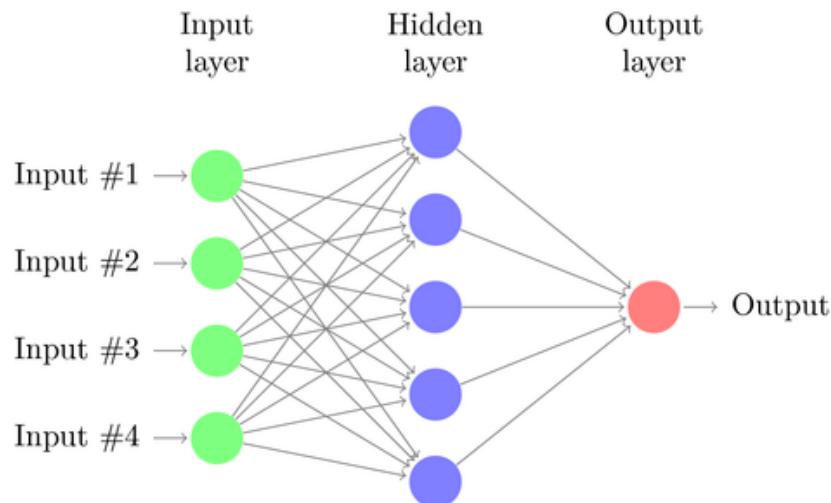


- 혹은 MLP

# MLP(Multi Layer Perceptron)

입력과 출력 사이에 층이 더 있다.

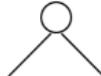
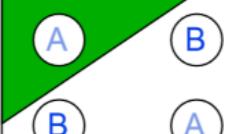
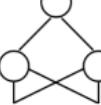
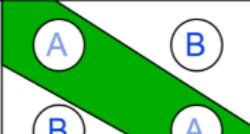
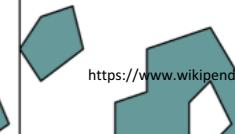
개별 perceptron의 결과를 다음 층의 입력으로 사용하고 결과적으로 선형 분리의 제약을 극복한다.



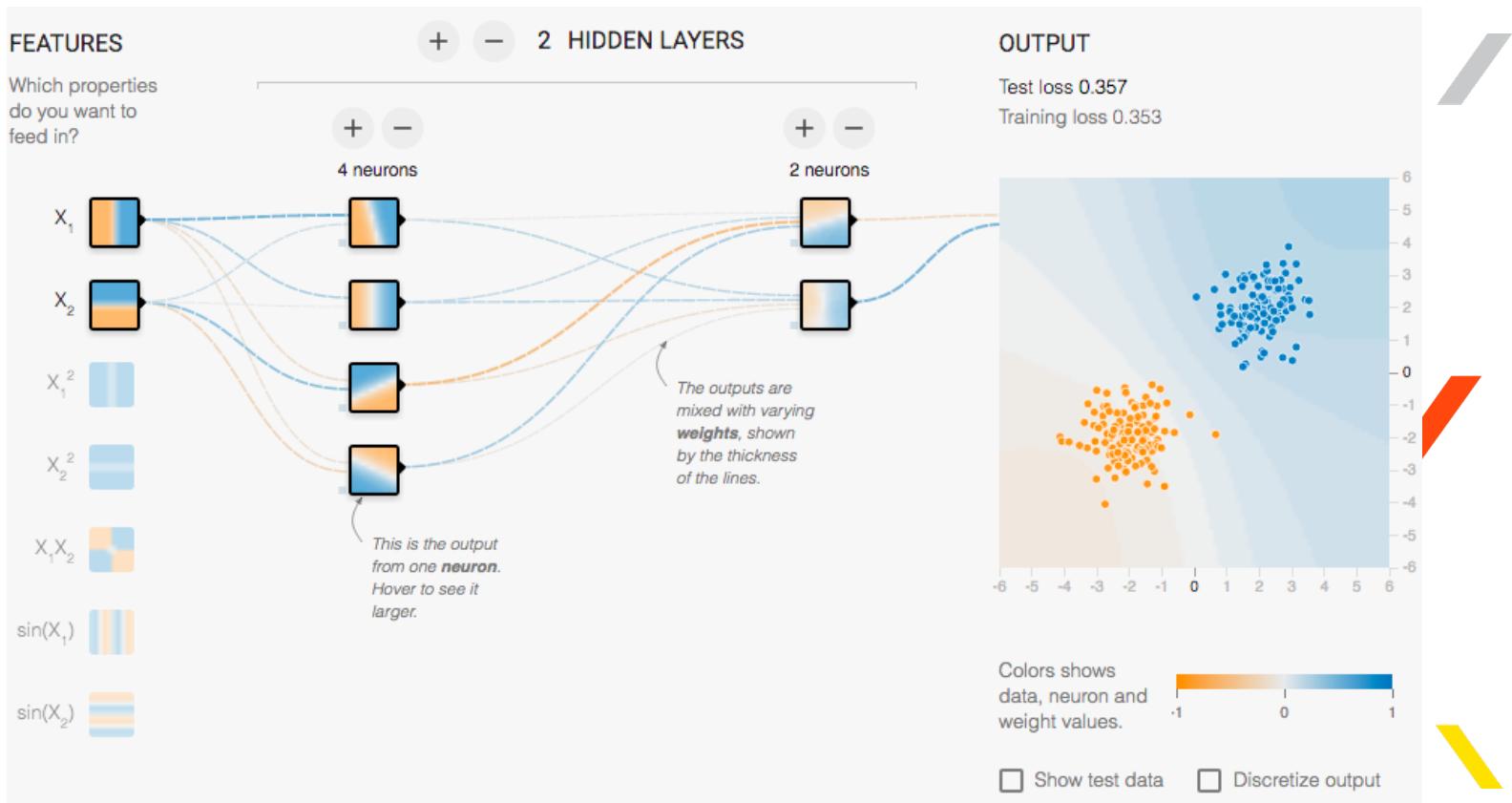
# 퍼셉트론의 능력

1개의 퍼셉트론은 1개의 선형 분리를 할 수 있다.

퍼셉트론의 결과를 다른 퍼셉트론의 입력으로 하면 여러개의 선으로 분리를 할 수 있다.

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyperplane 			
Two-Layer 	Convex Open Or Closed Regions 			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes) 			 <a href="https://www.wikipendium.no/TDT4137_Cognitive_Architectures">https://www.wikipendium.no/TDT4137_Cognitive_Architectures</a>

# 온라인 시뮬레이션

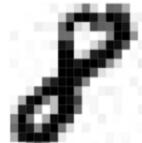


<http://playground.tensorflow.org>

# 숫자 인식의 예

이미지를 구성하는 pixel의 각 값으로 구성된 입력벡터를 NN의 입력으로 한다.

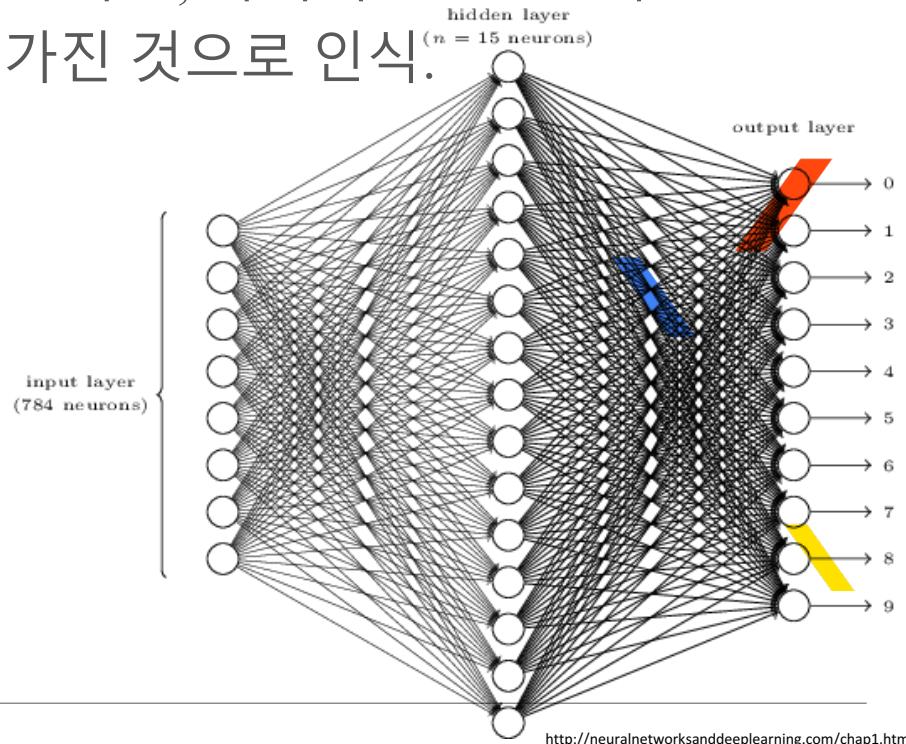
그리고 학습시 해당 출력 노드만 1로 하고, 나머지는 0으로 학습.  
test 시에는 출력 노드중 최대 값을 가진 것으로 인식.



=

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2, 55, 233, 40, 0, 0, 10, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 187, 0, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0, 38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 0, 4, 0, 163, 225, 251, 255, 229, 128, 0, 0, 0, 0, 11, 0, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 18, 14, 6, 0, 0, 9, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 0, 145, 248, 228, 116, 235, 255, 141, 34, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

<https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721#.ylp7hd52a>



<http://neuralnetworksanddeeplearning.com/chap1.html>

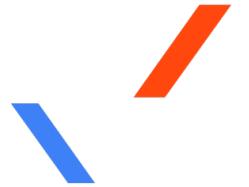


함수 근사화 능력

# DNN의 능력

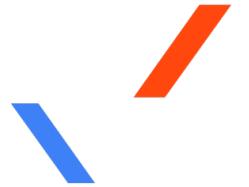
## Universal Approximator

어떠한 함수도 근사화 할 수 있다.

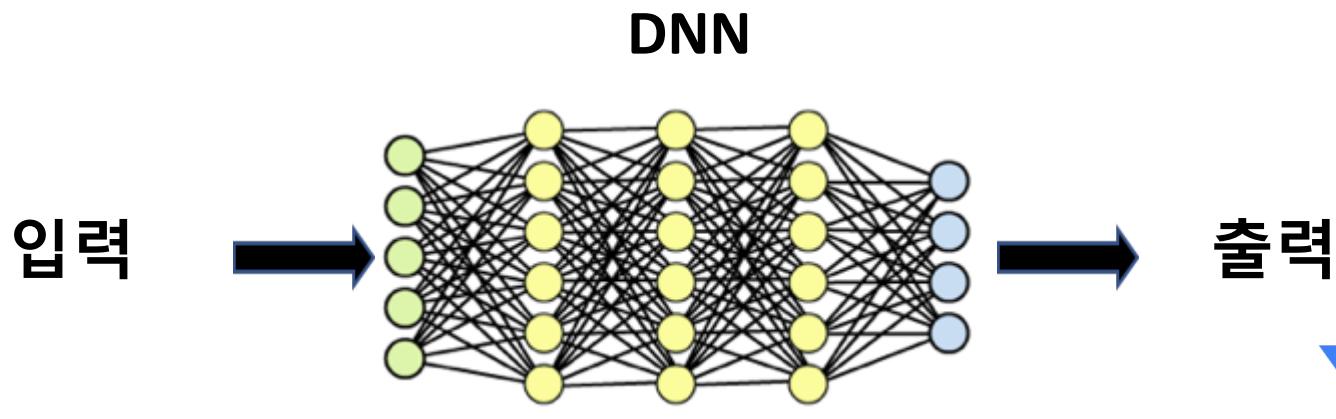


# DNN의 함수 근사화 능력

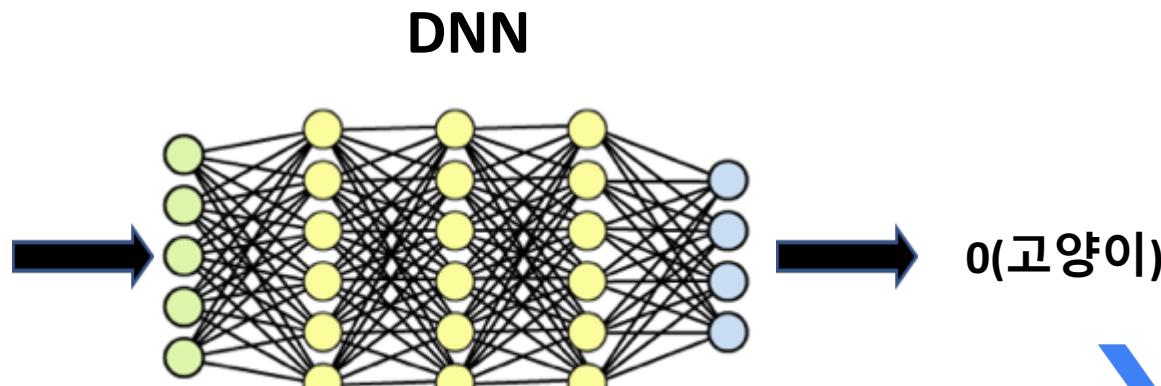
- DNN은 임의의 함수를 근사화 할 수 있다.
- 함수의 내부를 모르더라도.
- 입력과 출력 데이터로.



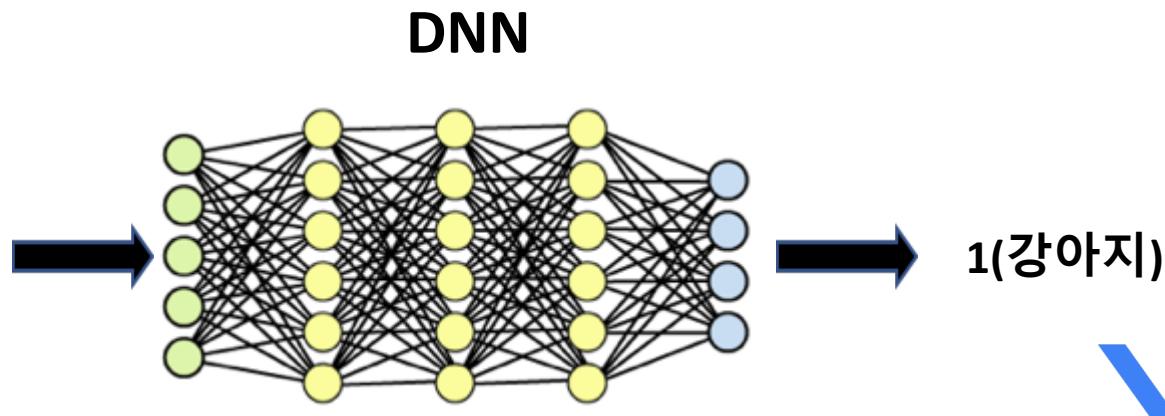
# 함수



# 학습

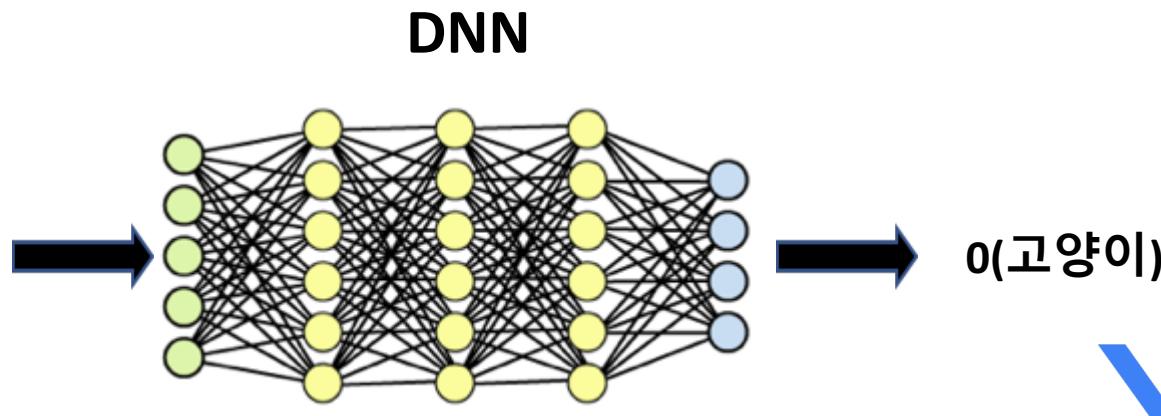


# 학습



1(강아지)

# 학습

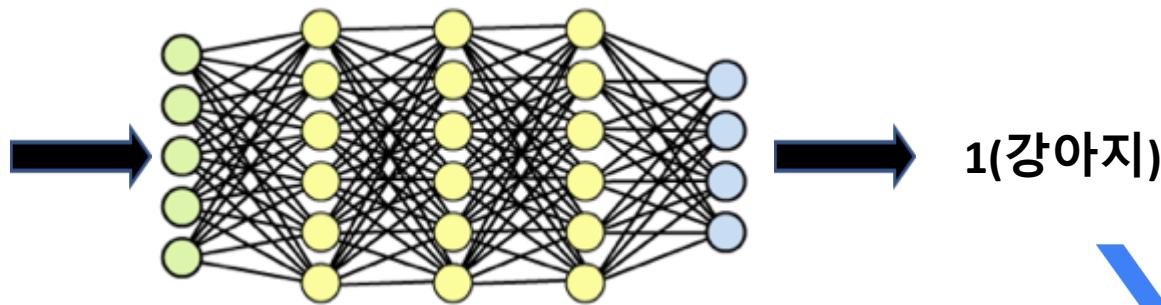


0(고양이)

# 학습



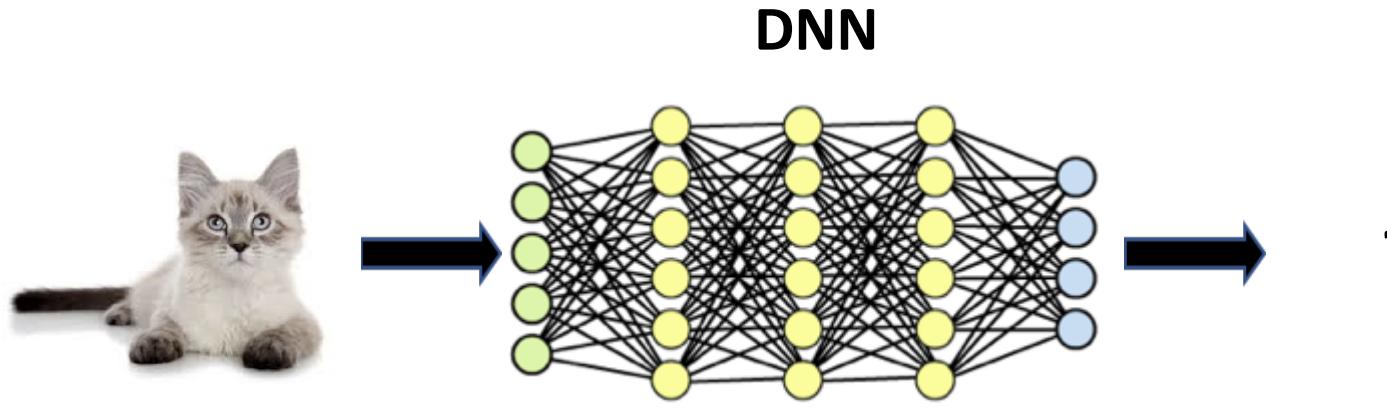
DNN



1(강아지)



# 학습



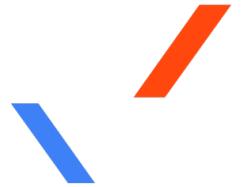
처음 보는 입력이라도 출력을 낸다. 의미 있는.

# 함수를 근사화 한 것이다

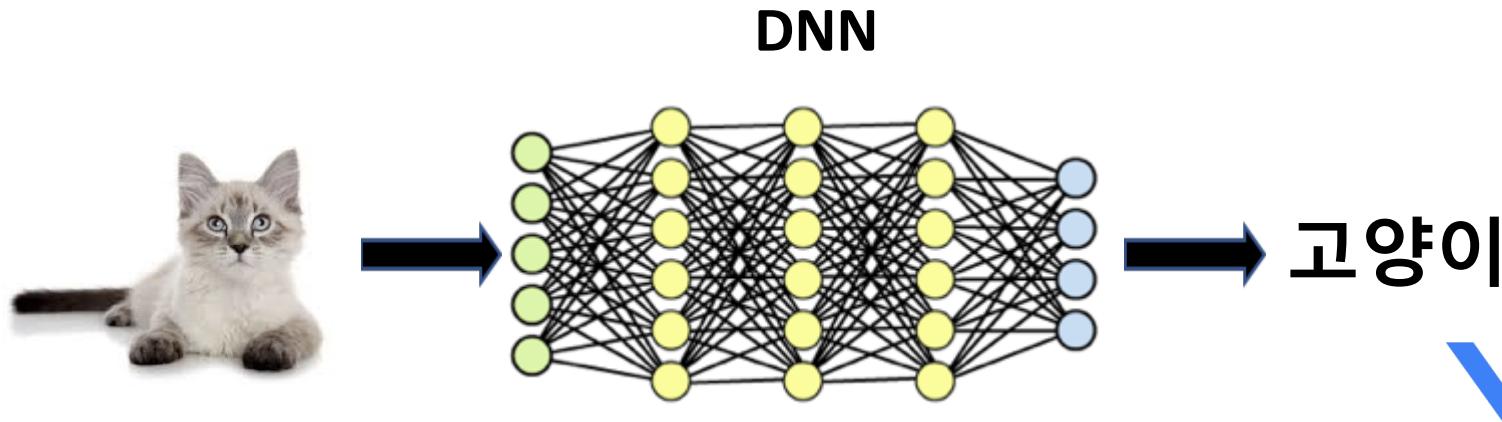
- 고양이, 강아지 구분 함수
- 어떻게 구분하는 지 정의하지 않았다.
- 정의하기는 힘들어도, 그 함수는 존재한다.
- 단순한 입출력 쌍 데이터로 그 함수를 근사화 하였다.

# 단순, 하지만 강력한 방법

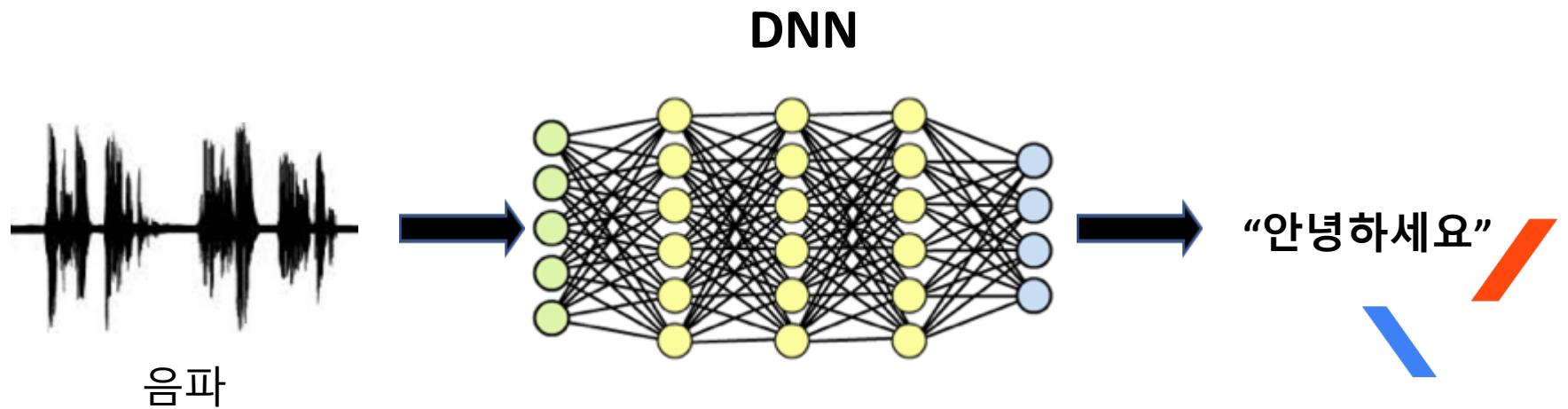
- 단순히 입출력 쌍을 반복하여 학습시킨다.
- 하지만 로직을 찾아내기 어려운 문제에는 아주 효과적이다.
- 얼굴 인식, 물체 인식 같은.



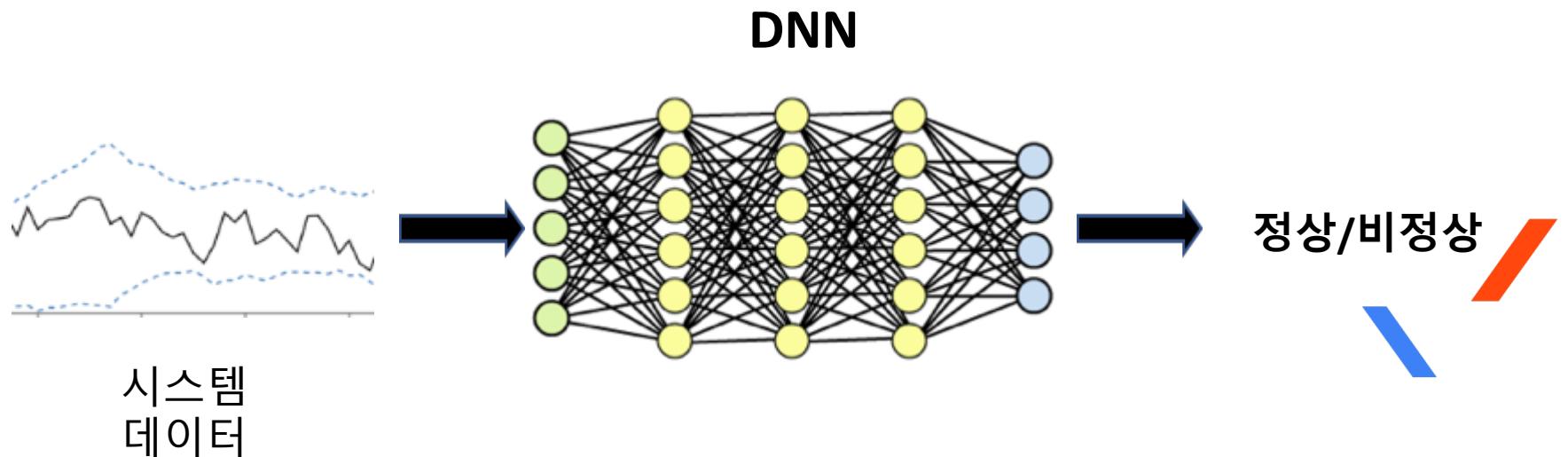
# 함수 예



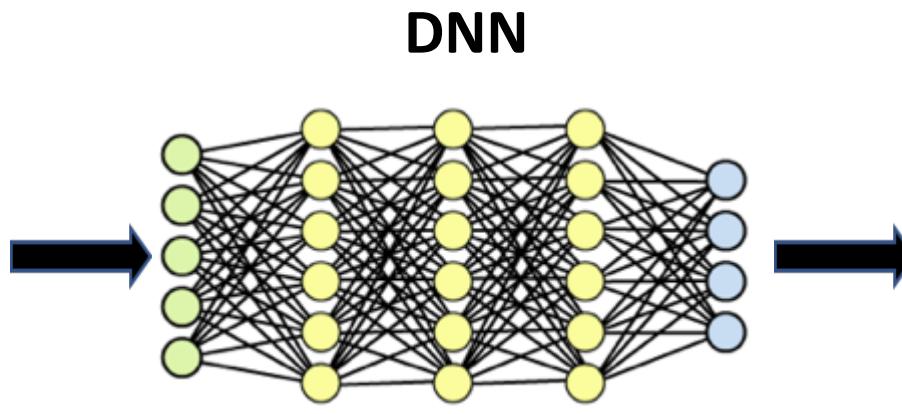
# 함수 예



# 함수 예

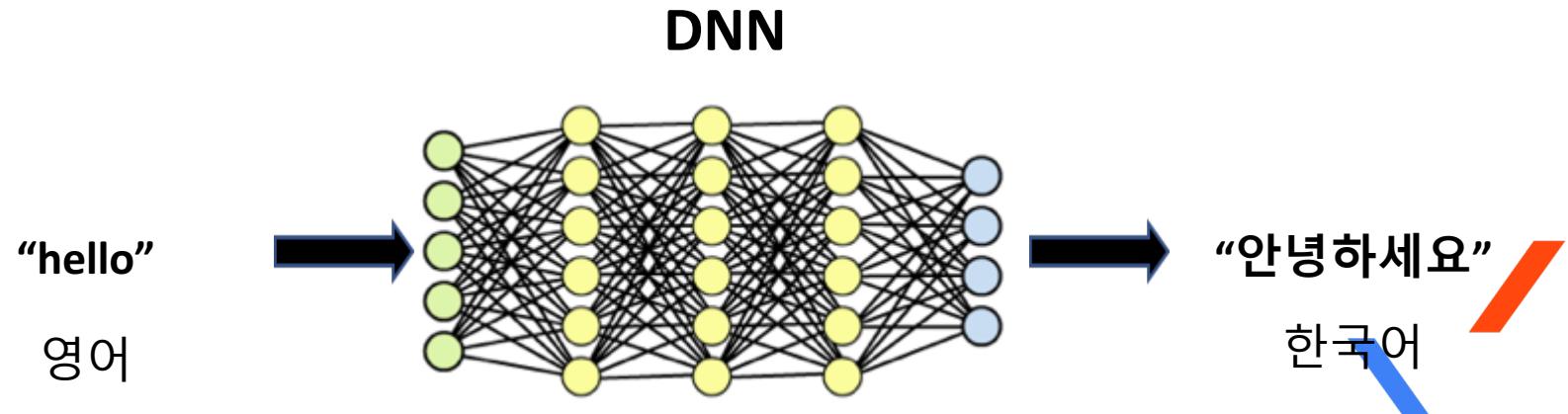


# 함수 예

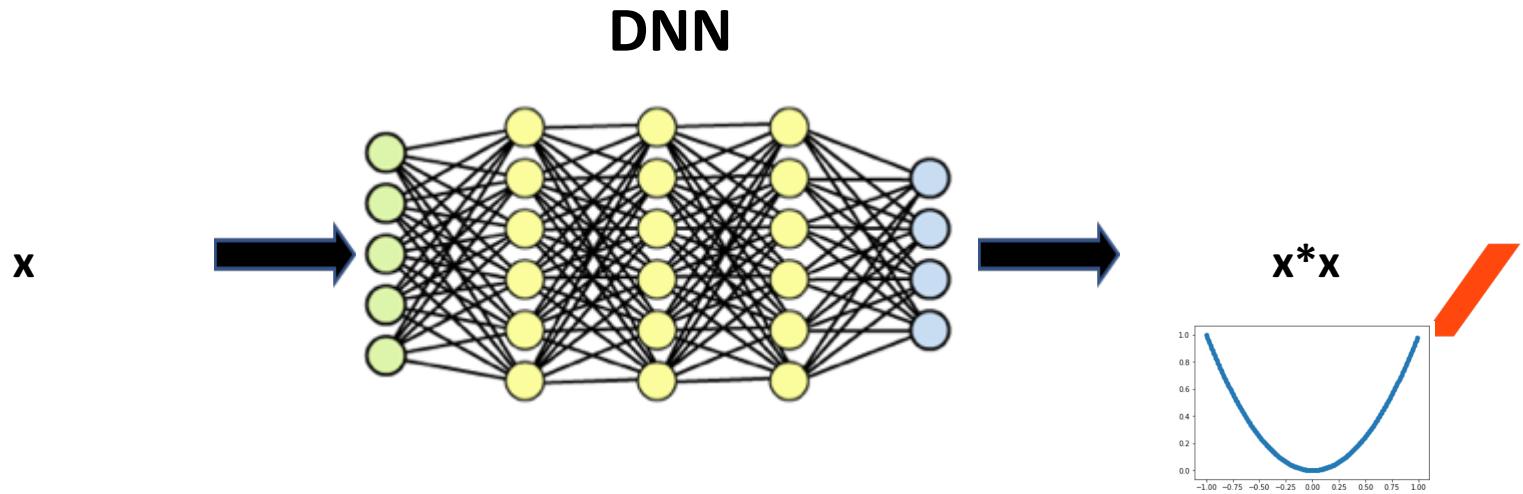


다음 수

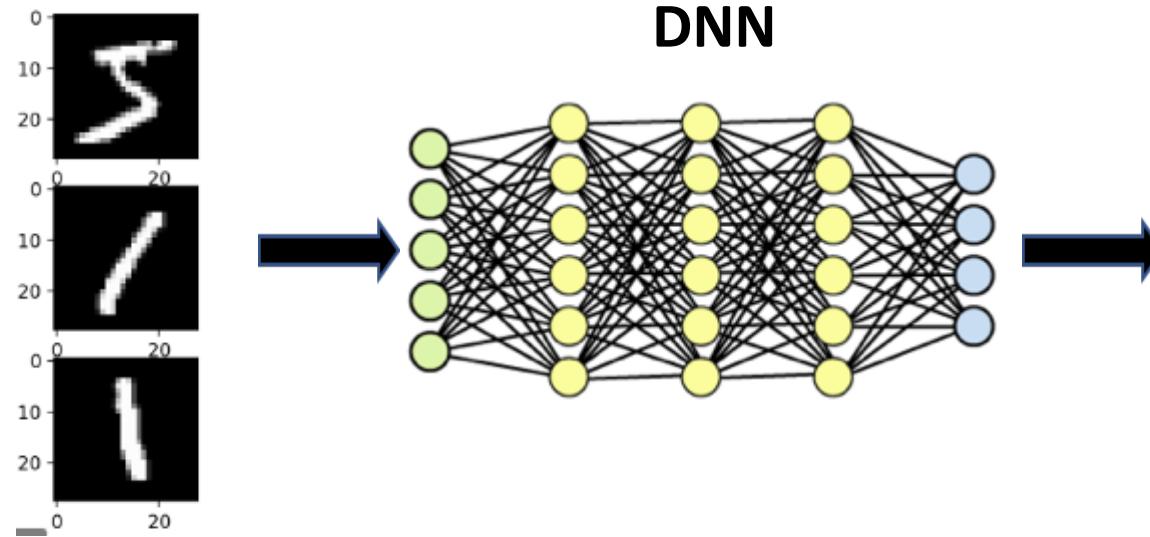
# 함수 예



# 함수 예



# 함수 예



0 ~ 9

# 딥러닝

- DNN은 함수 근사화 능력이 있다.
  - 입출력 쌍을 반복적으로 제공하여 내부를 업데이트 한다.
  - 충분한 입출력 데이터와 컴퓨팅 파워를 필요.
- 
- DNN으로 특정 함수 근사화하는 것을 딥러닝이라 한다.

인공지능, 머신러닝,  
딥러닝

# 인공지능(AI; Artificial Intelligence)

- 사람이 아닌 기계가 알아서 하면 인공지능이다.
- 전문가의 지식을 기반으로하거나(전문가 시스템)
- 혹은 데이터를 기반으로하거나(머신러닝)

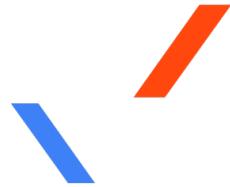
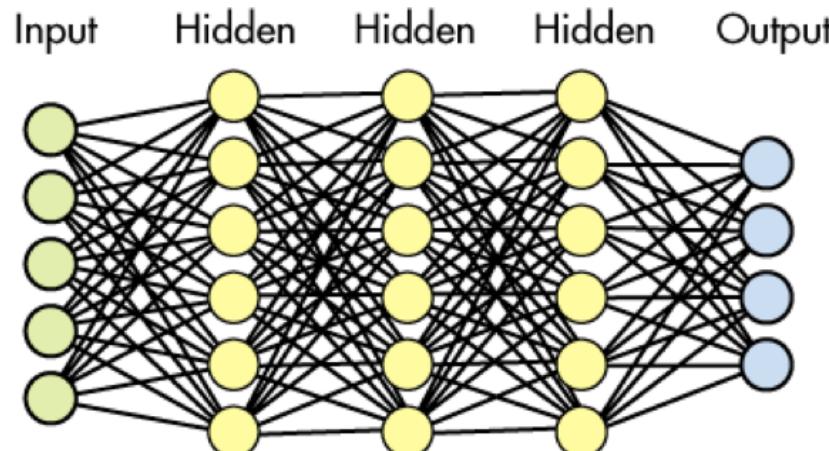


# 머신러닝(ML; Machine Learning)

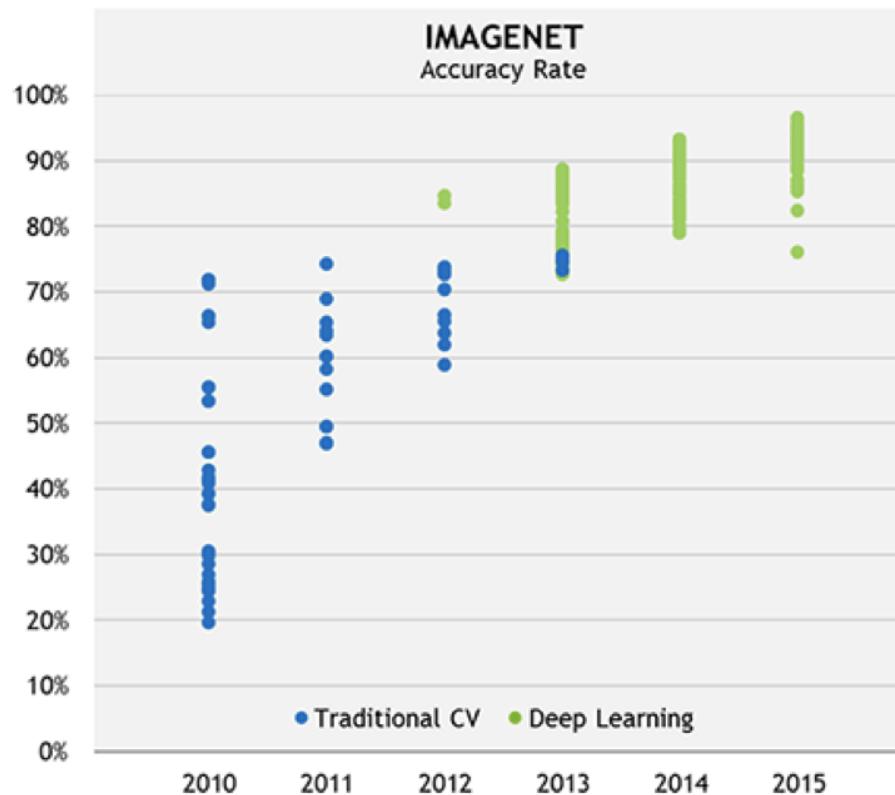
- 인공지능의 한 분야
- 데이터에서 가치를 찾아내는 것
- 아주 다양한 방법이 있다.
  - SVM, 의사결정트리, Random Forest, Bayesian, K-Means Clustering, K-NN, Neural Network

# 딥러닝(DL; Deep Learning)

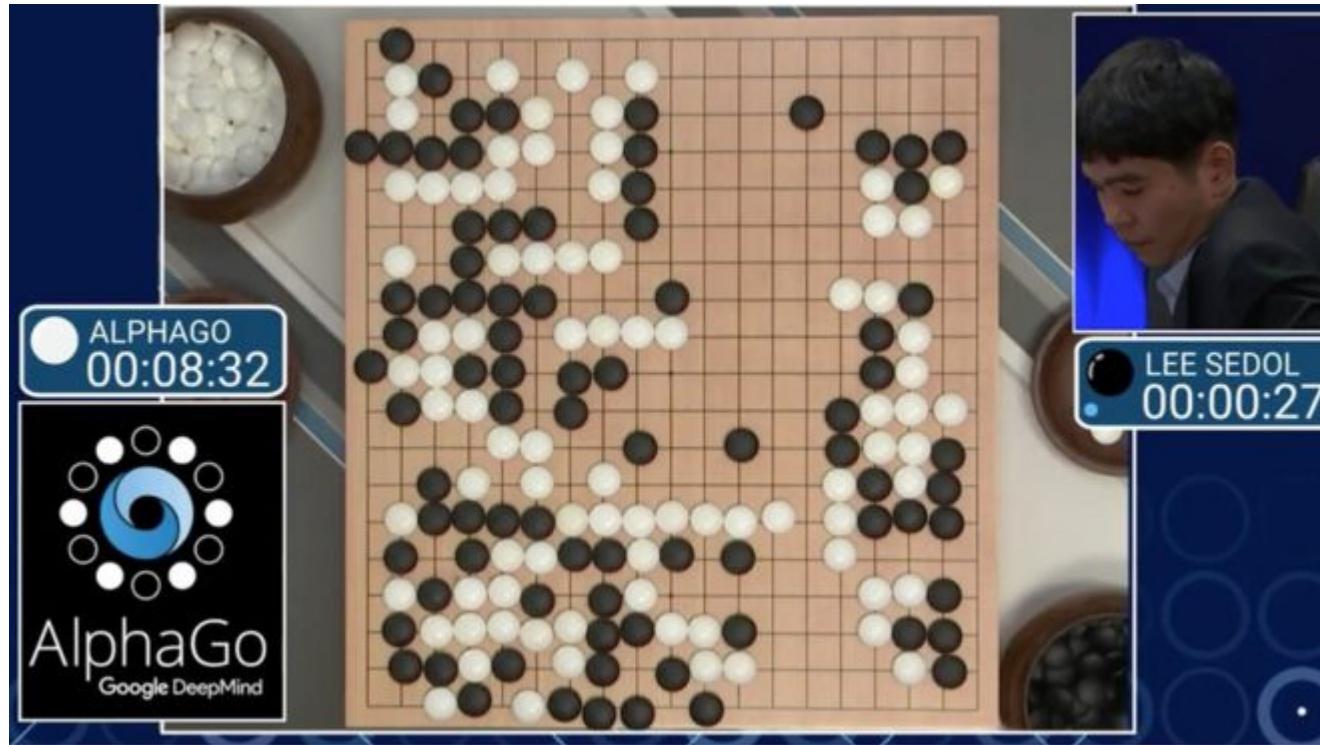
- 신경망(NN; Neural Network)을 사용한 머신러닝 방법
- 신경망의 은닉층이 많아서(deep) DNN(Deep NN)이라 부른다



# 2012년 AI 부활



# 2016년 화려한 복귀

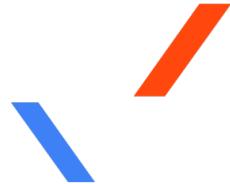


# 신경망(NN)과 심층신경망(DNN)

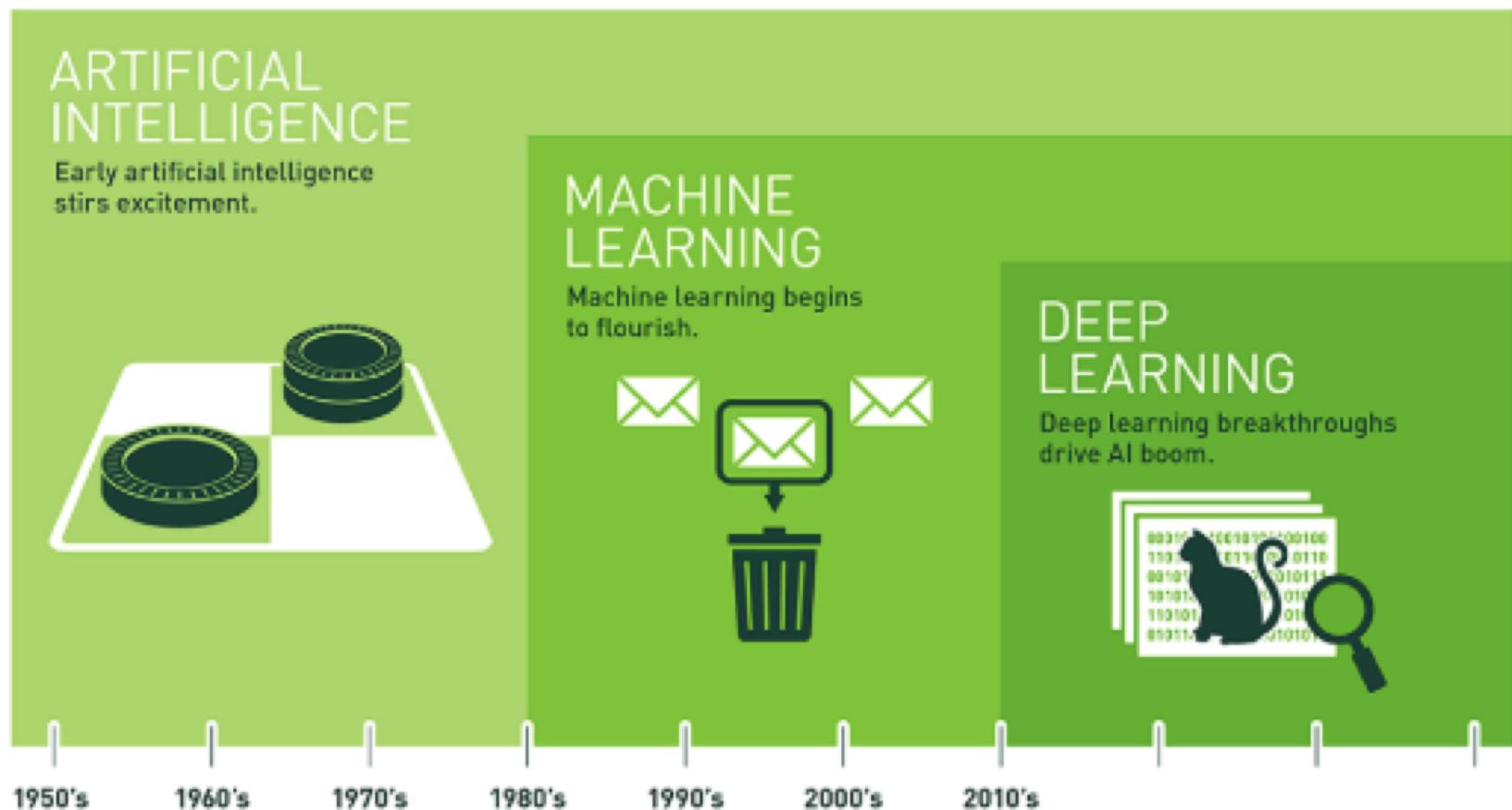
- 별 차이 없다.
- 신경망에 대한 큰 기대만큼 큰 실망
- 근래 뚜렷한 성과를 보이면서 다시 큰 관심.
- DNN: 실망했던 용어를 대신하여 봄을 일으키기 위한 용어.

# AI, ML, DL

- 인간이 고안한 알고리즘이건 기계가 학습한 알고리즘 이건, 기계가 스스로 처리하면 AI
- 데이터를 기반으로 하면 ML
- 그중 신경망을 사용하는 것이 DL
- 서로 다르지만, 그냥  $AI = ML = DL$  이라 부른다.



# AI, ML, DL



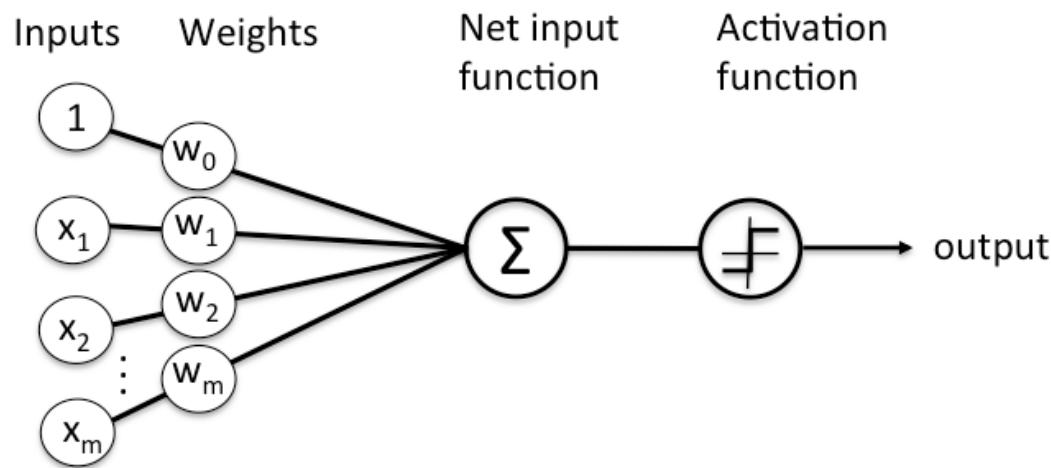


딥러닝 조금 더

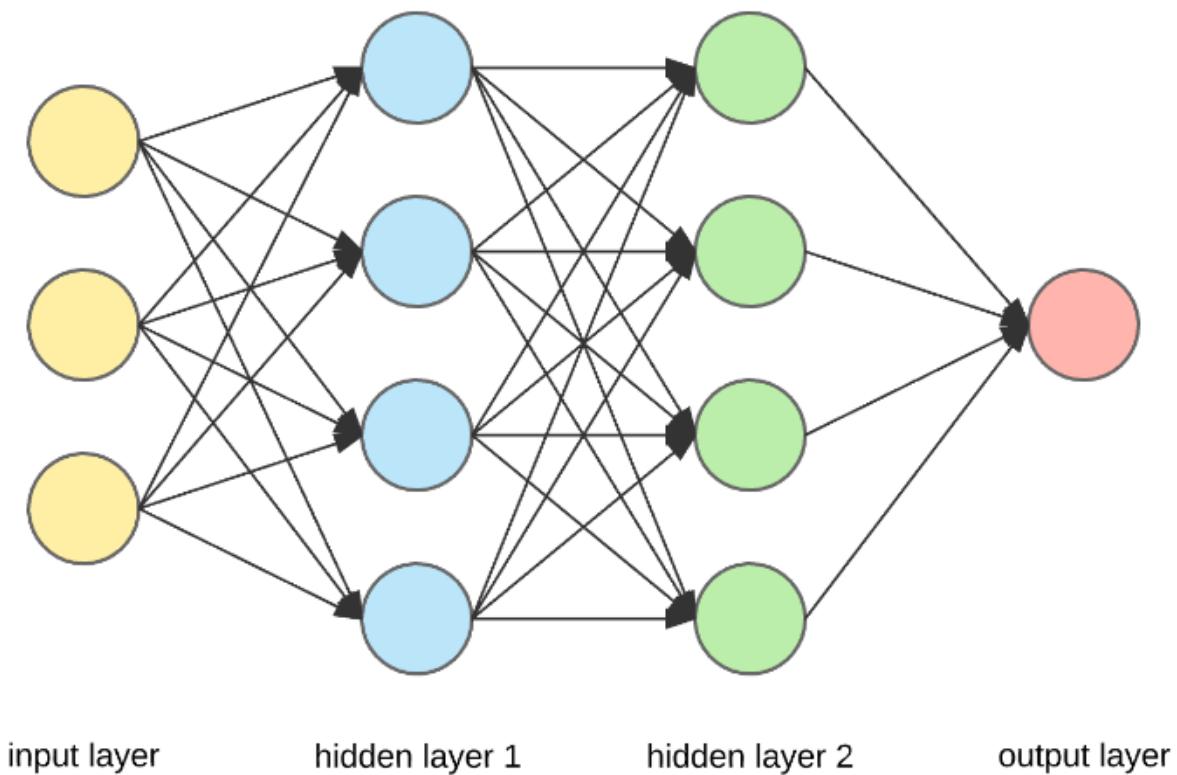


# 신경망의 노드

- 입력이 가중치만큼 곱해져서 전부 더해진다.
- 이후 특정함수를 거쳐 출력된다.

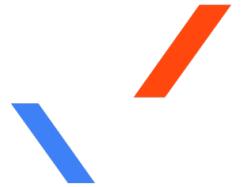


# 노드, 층, 네트워크, 모델



# 학습

- 입력 값들은 가중치에 의해 출력이 결정된다.
- 원하는 출력이 되도록 가중치를 조절하는 것이 학습이다.
- 학습된 결과는 학습된 가중치 들이다.



# 역전파, 경사하강법

- 역전파(BP, Back Propagation) 알고리즘
- 경사하강법(GD, Gradient Descent) 알고리즘
- 가중치를 업데이트 하기 위해 사용되는 알고리즘

# TensorFlow, Keras

- 이러한 알고리즘과 기타 편의성을 제공하는 프레임워크.
- 현재의 딥러닝 개발은 이러한 프레임워크 사용을 기반으로 한다.
- BP, GD와 같은 알고리즘이 지원되며, 실제 구현 시에 BP, GD를 몰라도 된다.



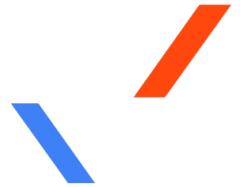
# 딥러닝 장단점

- 장점 : 대상 함수의 내부를 몰라도 된다.
- 단점 : 비싸다
  - 많은 데이터, 많은 연산량



# ML과 DL의 선택

- 기존의 방법으로 이미 풀린 문제는 ML
- 기존의 방법으로 못풀었는데 데이터가 있으면 DL
  - 바둑, 얼굴인식, 물체인식, 음성인식, 번역



# 딥러닝 실제 코드

```
model = build_model() // 모델 구성
```

```
for i in range(1000): // 모든 데이터에 대하여 1000번 반복  
    model.fit(datas)
```

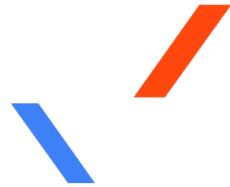
```
model.save('my_model.h5') // 저장. 파일 크기는 보통 100메가 단위
```

```
model = load('my_model.h5') // 저장된 모델 로딩  
predicted = model.predict(input) // 예측 실행
```

# 딥러닝 트렌드

- 2대 적용 분야 : 자연어, 영상
- 일부 작업은 이미 안정화 단계에 있다.
  - 영상분류, 영상인식
  - Keras의 배포본에 포함

```
model = keras.vgg16.VGG16()  
predicted = model.predict(image_data)
```



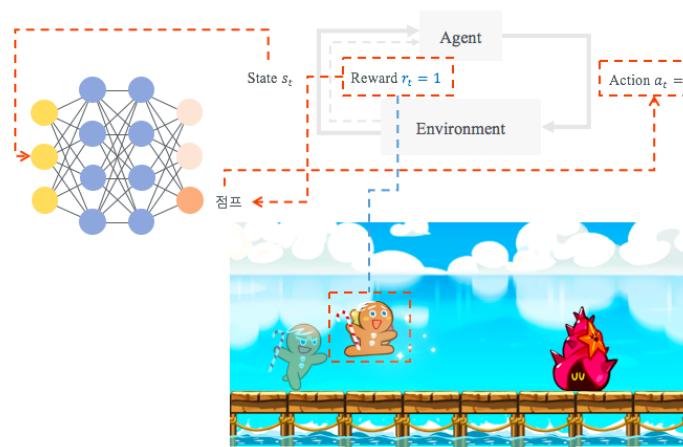
# 딥러닝의 큰 단점

- 입출력 데이터 쌍을 구하기 어렵다.
- 특히 출력 데이터. 레이블링 데이터(labeling data)



# 비지도 학습, 강화 학습

- 레이블링 데이터 문제를 해결하기 위한.
- 비지도 학습 : 모델 구조를 통해 레이블링 데이터 없이. GAN
- 강화 학습 : 환경과 동적으로 연동하여 레이블링 데이터를 취득.

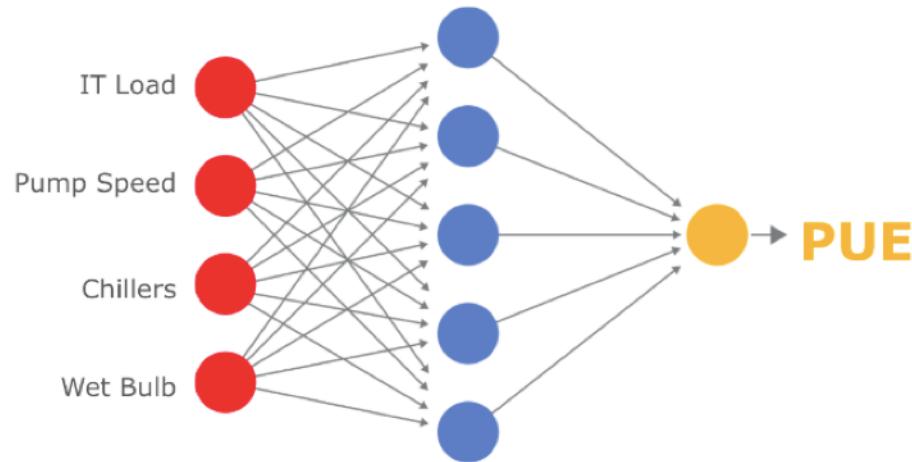




예 - 구글 데이터 센터

# DNN 적용

- 제어값을 입력으로 하고, PUE\*를 출력으로 하여 학습
- 실제 데이터 사용

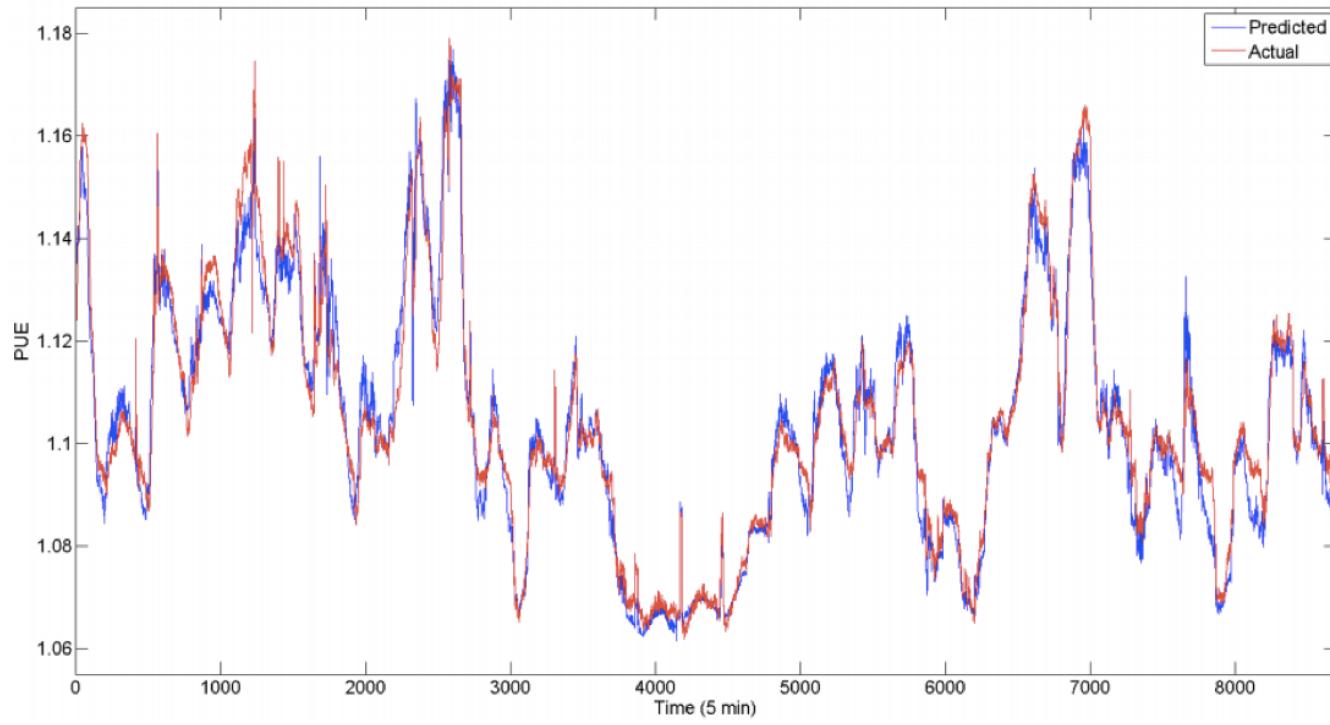


\*PUE(Power Usage Effectiveness) : 에너지 사용 효율도

# 입력

1. Total server IT load [kW]
2. Total Campus Core Network Room (CCNR) IT load [kW]
3. Total number of process water pumps (PWP) running
4. Mean PWP variable frequency drive (VFD) speed [%]
5. Total number of condenser water pumps (CWP) running
6. Mean CWP variable frequency drive (VFD) speed [%]
7. Total number of cooling towers running
8. Mean cooling tower leaving water temperature (LWT) setpoint [F]
9. Total number of chillers running
10. Total number of drycoolers running
11. Total number of chilled water injection pumps running
12. Mean chilled water injection pump setpoint temperature [F]
13. Mean heat exchanger approach temperature [F]
14. Outside air wet bulb (WB) temperature [F]
15. Outside air dry bulb (DB) temperature [F]
16. Outside air enthalpy [kJ/kg]
17. Outside air relative humidity (RH) [%]
18. Outdoor wind speed [mph]
19. Outdoor wind direction [deg]

# 예측 결과

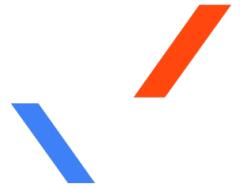


# 학습된 DNN

- 제어값과 PUE의 관계를 학습
- 임의의 제어값에 대한 PUE를 알 수 있다.
- 학습된 DNN은 시뮬레이터로 사용할 수 있다.

# 시뮬레이션이 가능하면

- 임의의 제어값에 대한 PUE를 미리 알 수 있다.
- 다양한 입력에 대한 물레이션으로, 최선의 PUE를 찾을 수 있다.



# 딥러닝 개발 환경



# 많은 연산량

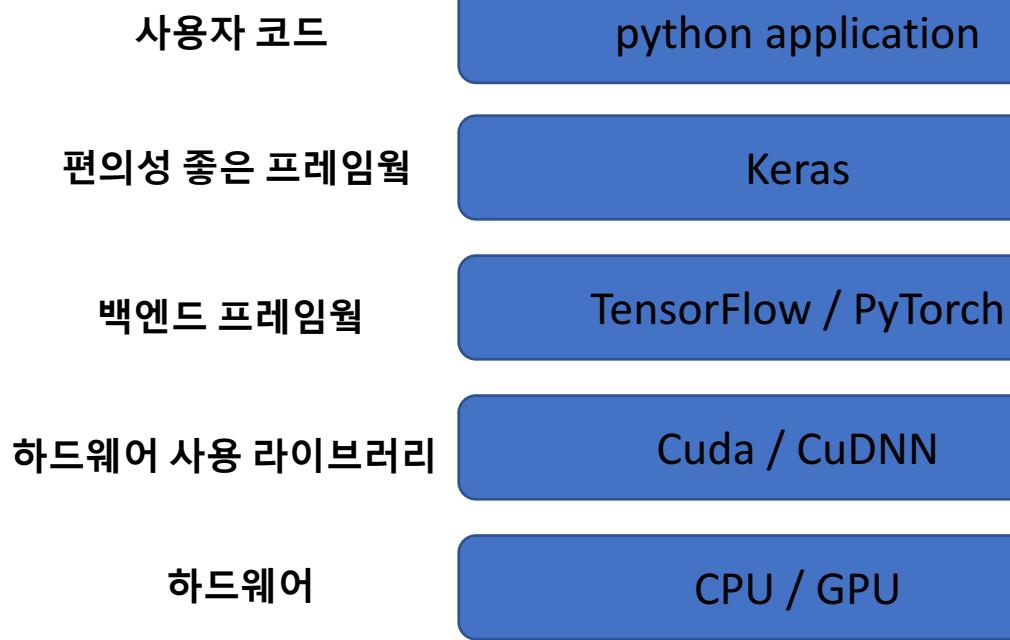
- 일반적인 그래픽 카드
- 딥러닝 학습 계산의 대부분은 단순 곱셈과 덧셈.
- 많은 수의 노드가 개별적으로 계산된다.

# GPU

- 많은 수의 코어.
- 병렬 연산 가능



# 딥러닝 개발 스택



# TensorFlow와 Keras

- TensorFlow는 어느정도 난해하다.
- Keras로 편하게 코드 작성
- Keras는 TensorFlow를 호출하여 실행된다.
- 2019년 9월에 공식적으로 TensorFlow에 Keras가 포함되었다.

# Jupyter Notebook

- 웹을 기반으로 함.
  - 코드를 서버로 전달하여 실행하고
  - 그 결과를 웹 화면으로 보여줌
- 
- 작업 페이지 내에 그림도 보여주고
  - 작업 페이지 내에 문서도 작성하고



# Jupyter Notebook

- 사용자 컴퓨터에 개발 환경이 없어도 됨.
- 작업 결과를 파일로 저장 가능. ipynb 파일

The screenshot shows a Google Colab interface. At the top, there's a navigation bar with back, forward, and refresh icons, followed by the URL `colab.research.google.com/drive/1E0CG30zeIDFOT6ukDiJ9OW54v2dwufe4#scrollTo=`. Below the URL are two buttons: '+ 코드' (Code) and '+ 텍스트' (Text). The main area is titled '주피터 노트북' (Jupyter Notebook). It contains explanatory text in Korean: '코드를 작성하고 실행시키면', '코드가 Jupyter 서버로 전송되고', and '실행된 결과가 이 페이지에 보여집니다.' (Write and run code, code is sent to the Jupyter server, and the results are displayed on this page). Below this text is a code cell with a play button icon and the Python code `print("hello Keras")`. The output of the code is shown as a text cell with a right-pointing arrow icon and the text 'hello Keras'. The bottom of the screen shows a toolbar with various icons.

# Colab

- 구글의 Jupyter 서비스
- 구글 계정(gmail 계정) 생성 만으로 준비 끝.
- 무료로 GPU 사용 가능
- <https://colab.research.google.com>



# Kaggle

- 머신러닝 공개 문제 서비스
- 문제와 데이터, 그리고 답이 있다.
- 무료 Jupyter를 사용할 수 있다.
- <https://www.kaggle.com>



# 딥러닝 기술 용어들

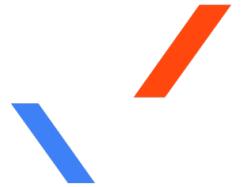
---

# Cost Function 종류

- MSE(Mean Squared Error)
- CE(Cross Entropy)
- KL-Divergence
- MLE(Maximum Likelihood Estimation)

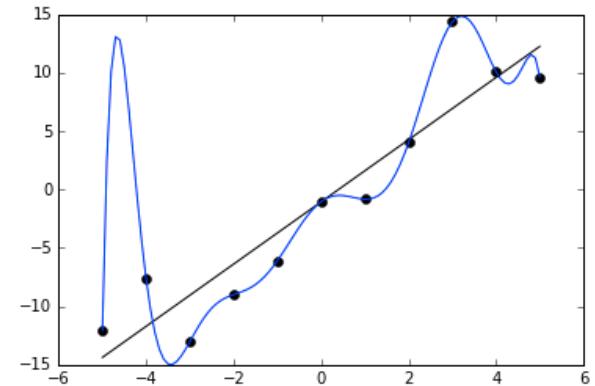
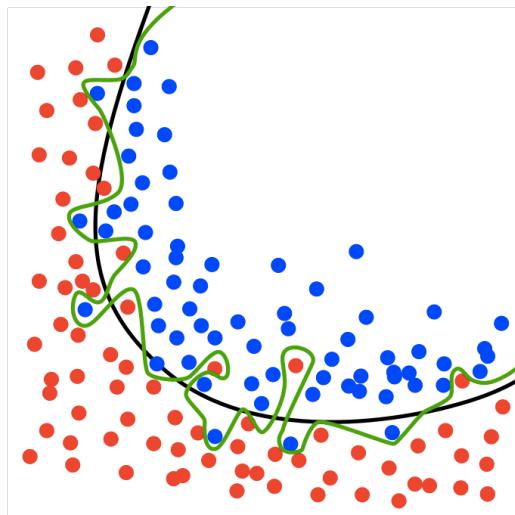
# Optimizer 종류

- 오차에 대하여  $w$ 를 업데이트 시키는 알고리즘들.
- GD(Gradient Descent)
- Batch GD
- **Mini-Batch GD**
- **SGD**(Stochastic GD)
- Momentum
- AdaGrad
- AdaDelta
- Adam
- RMSprop



# Overfitting 방지법 종류

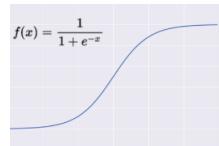
- DropOut
- BN(Batch Normalization)
- Regularization
- Data Augmentation



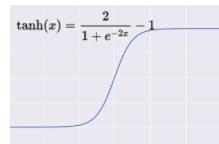
<https://en.wikipedia.org/wiki/Overfitting>

# 활성화 함수 종류

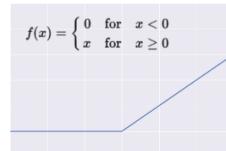
- sigmoid (= logistics)



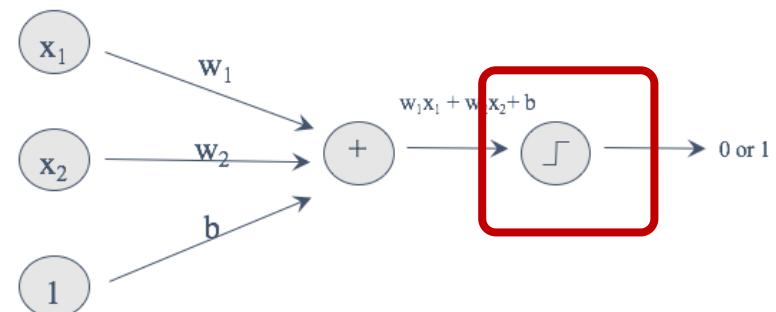
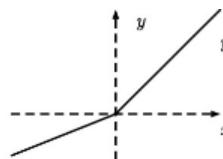
- Tanh



- ReLU

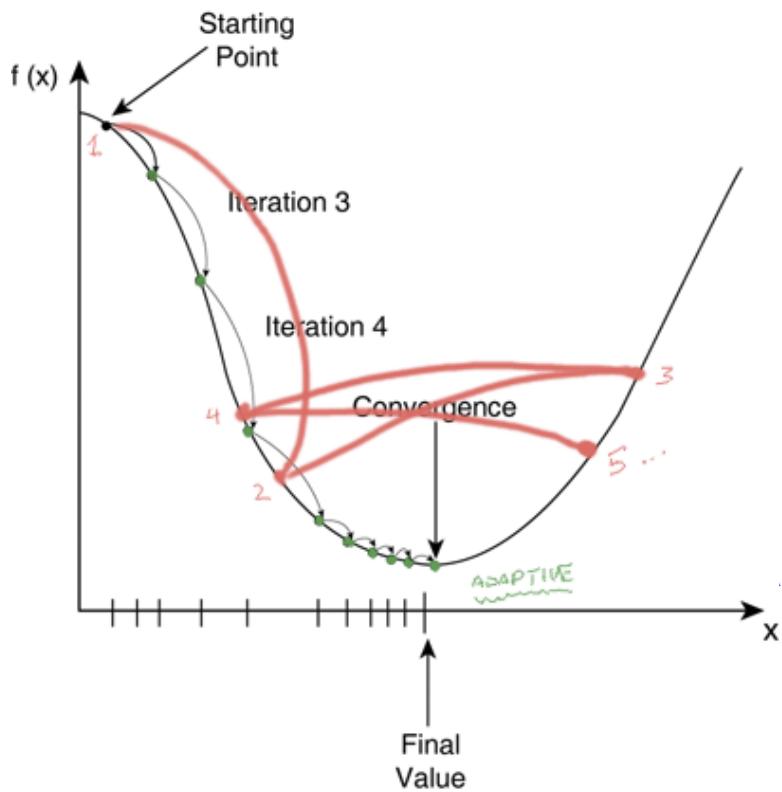


- Leaky ReLU



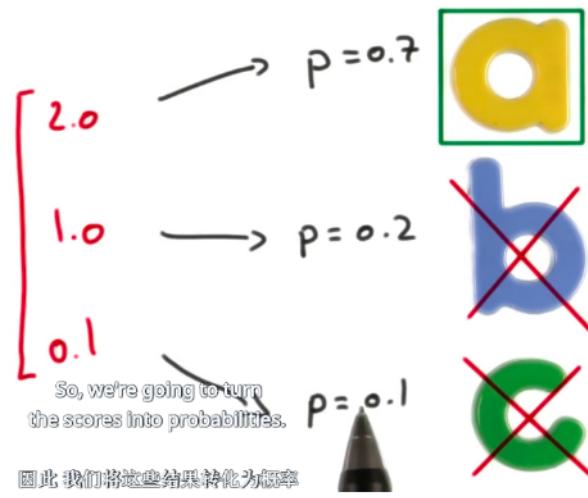
# 학습율

- 가중치가 변경되는 정도



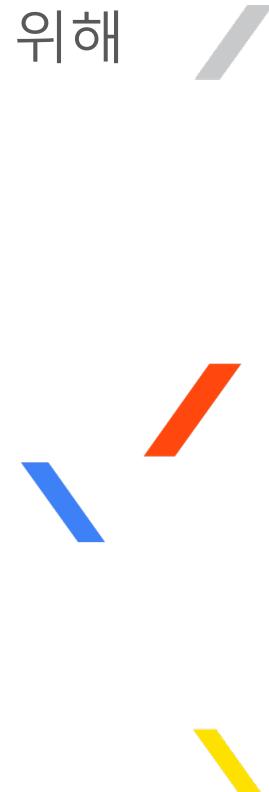
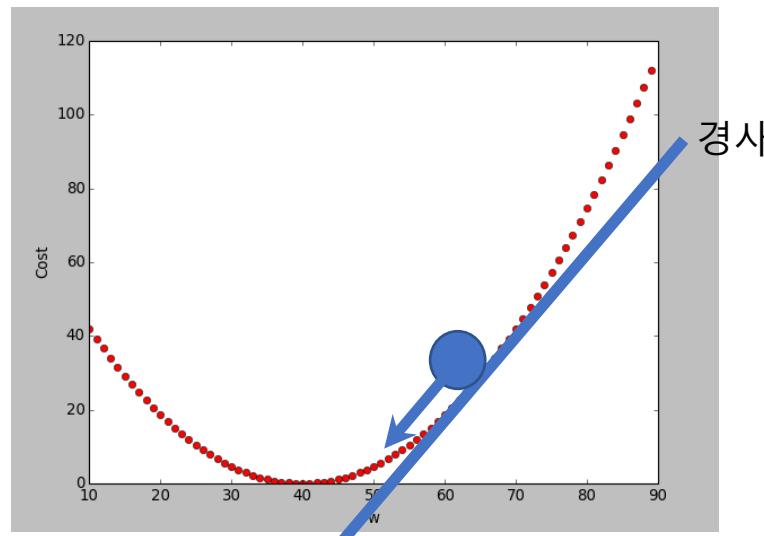
# SoftMax

- activation function 중의 하나.
- 최종 출력층에 사용되며, 여러개의 출력 노드의 합 중에 비중의 값으로 나타낸다.
- 확률처럼 표현된다.



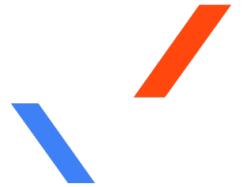
# Gradient Descent

- 함수가 학습될 바를 정의한 비용함수의 값이 최소로 하기 위해  
가중치를 업데이트하기 위한 알고리즘



# BackPropagation

- 출력된 값과 원하는 값과의 차이를 가지고 그 전의  $w$  값을 변경하는 알고리즘.
- 뒤에서부터 그 오차의 값이 전파된다는 이름.
- 실제 변경되는 값의 크기는 GD로 결정됨.



# IRIS 분류

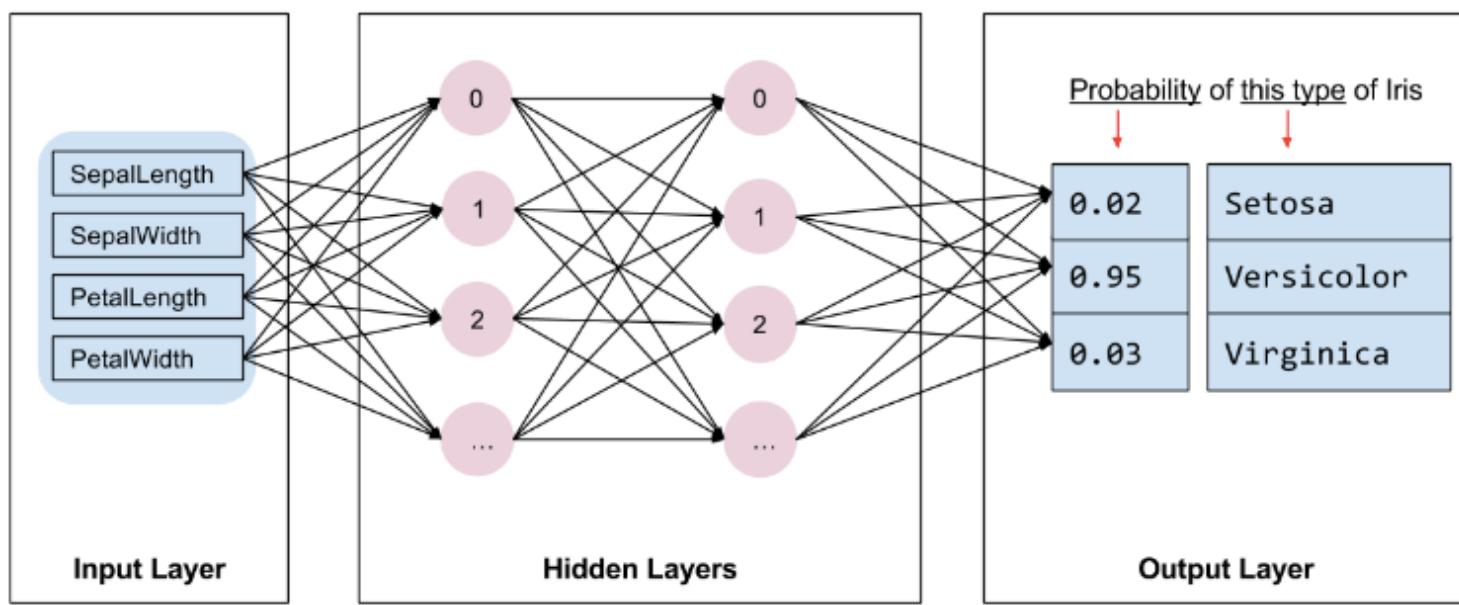
---

# IRIS 분류 문제

- 4개의 입력
- 3 종류로 분류



# 모델 구조

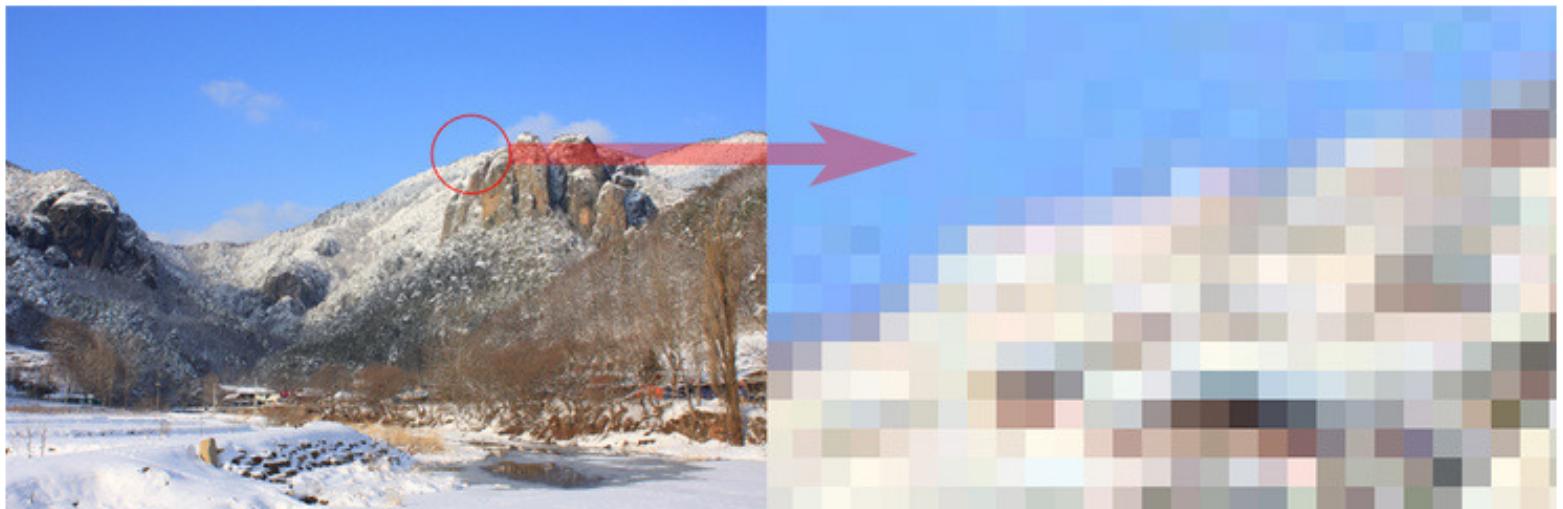


# 영상 데이터

---

# 영상과 픽셀

- 영상은 픽셀로 구성됨



# 픽셀의 값

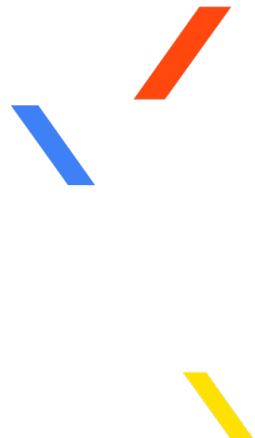
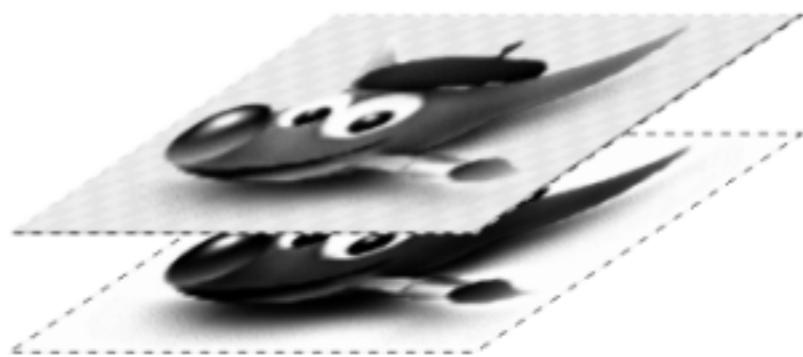
- 회색 영상의 경우 1개의 픽셀은 밝기의 값 1개로 나타냄.



187	187	187	194	197	173	77	25	19	19
190	187	190	191	158	37	15	14	20	20
187	182	180	127	32	16	13	16	14	12
184	186	172	100	20	13	15	18	13	18
186	190	187	127	18	14	15	14	12	10
189	192	192	148	16	15	11	10	10	9
192	195	181	37	13	10	10	10	10	10
189	194	54	14	11	10	10	10	9	8
189	194	19	16	11	11	10	10	9	9
192	88	12	11	11	10	10	10	9	9

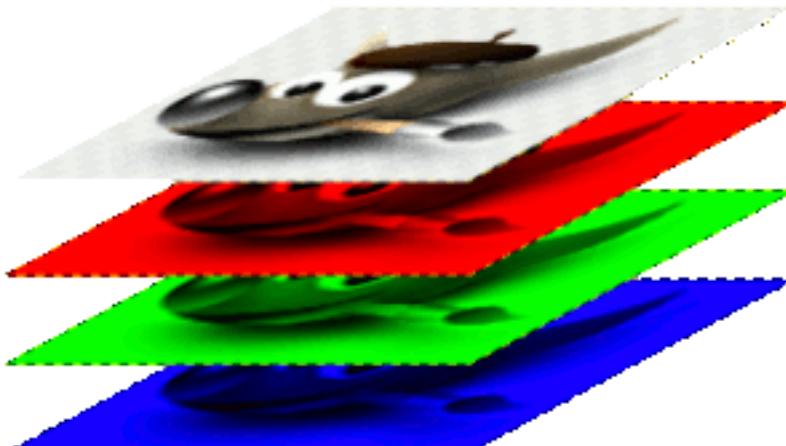
# 픽셀의 값

- 회색 영상의 경우 1개의 픽셀은 밝기의 값 1개로 나타냄.



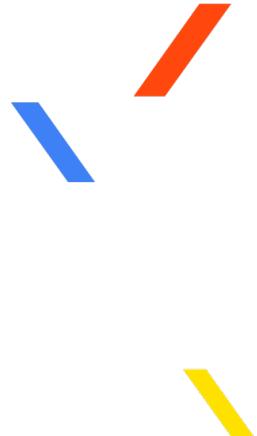
# 픽셀의 값

- 칼라 영상의 경우 1개의 픽셀은 3개의 색깔 Red, Green, Blue을 표시한 3개의 값으로 나타냄.



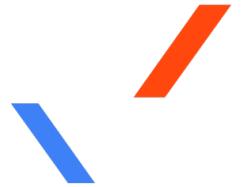
# 영상 데이터

- 가로 세로로 배열된 픽셀의 값을 숫자로 표시한 데이터
- 단지 숫자들이다.
- 모니터에서는 픽셀의 값에 따라 밝기 혹은 색깔로 표시하여 보여 준다.



# 영상 파일 포맷

- bitmap은 pixel 값 그대로 저장
- gif, jpeg, png 등은 압축하여 저장한 것.

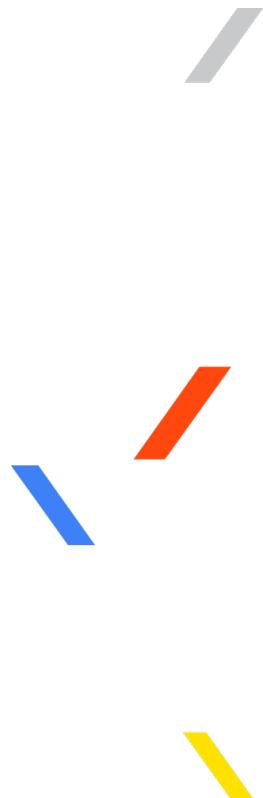
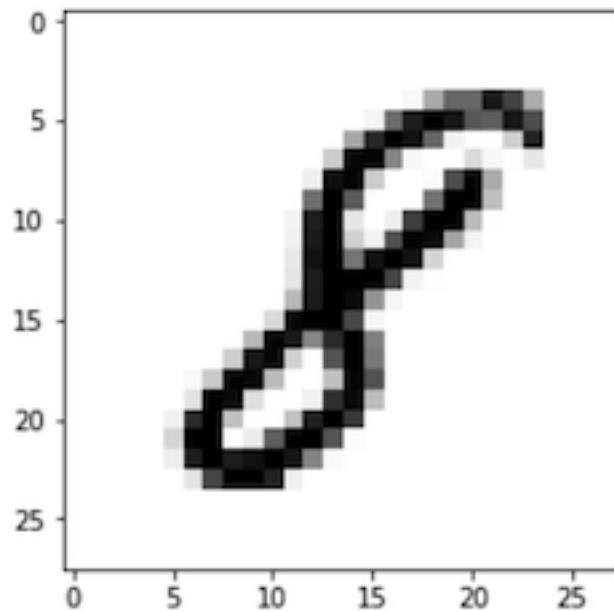


# MNIST 분류

---

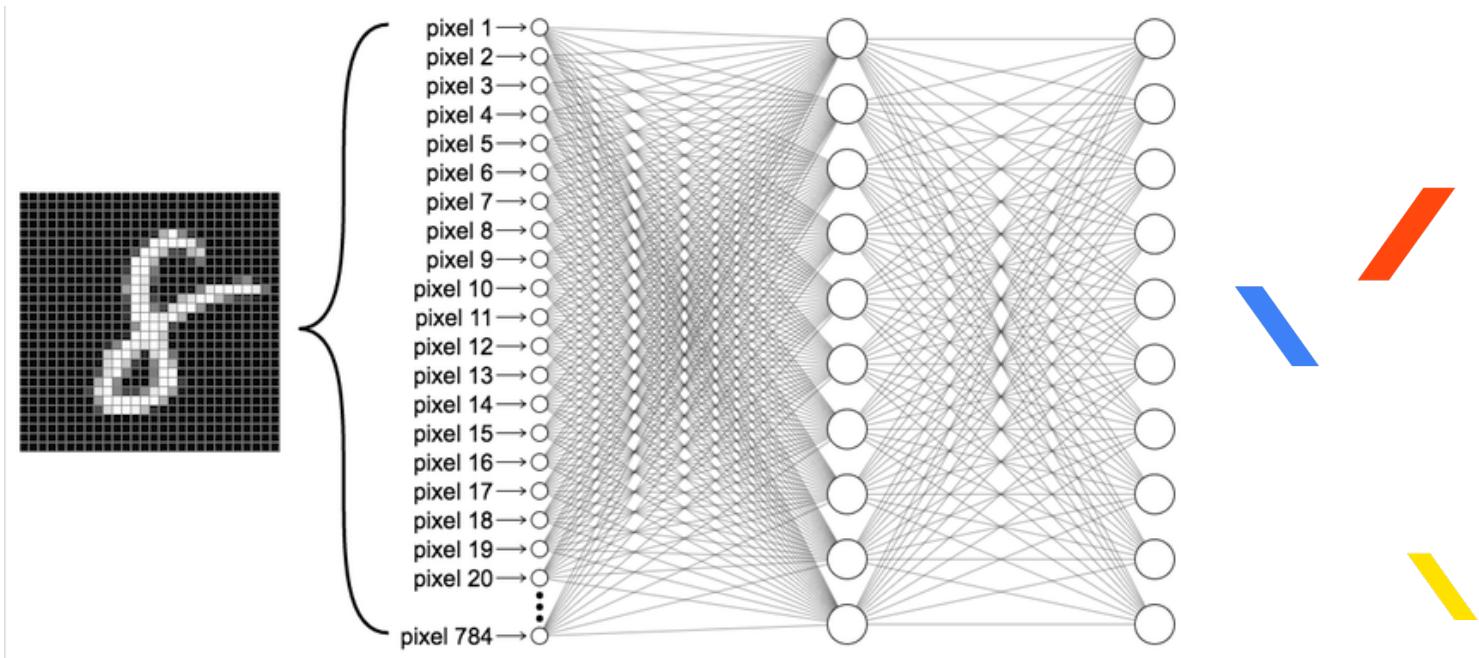
# MNIST 데이터

- 필기체 숫자 영상
- 가로 28개, 세로 28개

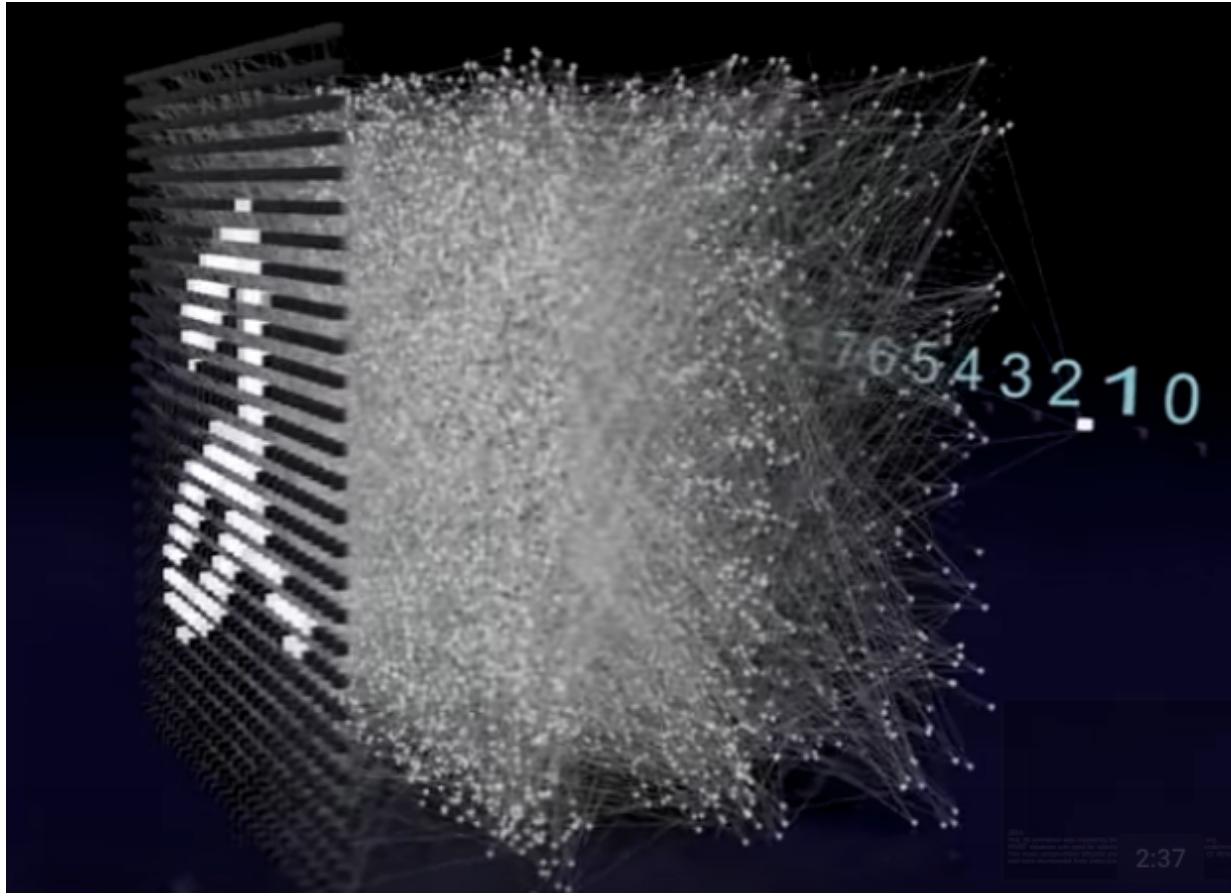


# MNIST 분류 문제

- 784개 입력
- 10개 카테고리, 10개 출력



# DNN MNIST 시뮬레이션



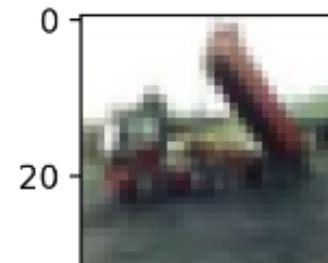
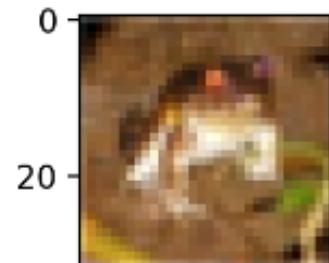
<https://youtu.be/3JQ3hYko51Y?t=48>

# CIFAR10 분류



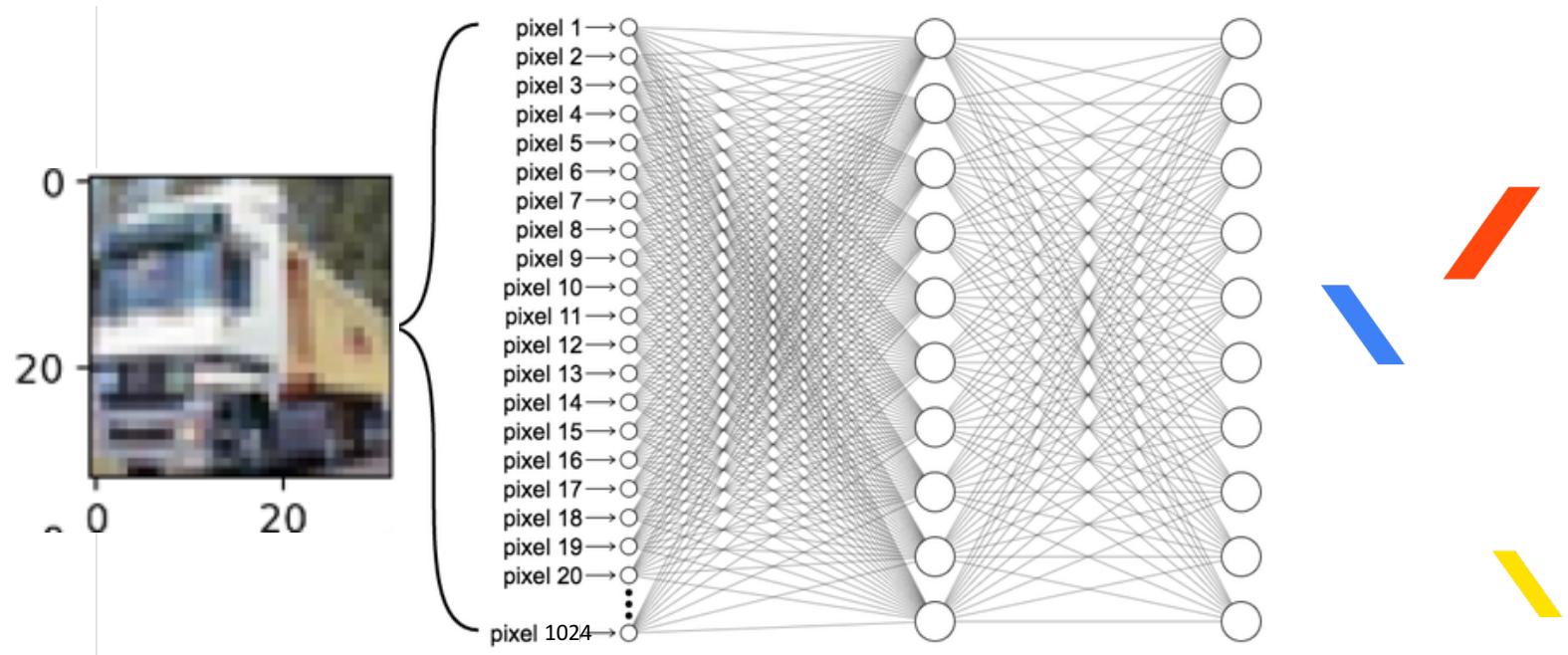
# MNIST 데이터

- 10개 대상 컬러 영상
- 가로 32개, 세로 32개



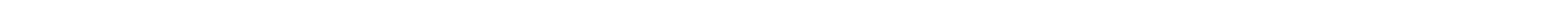
# MNIST 분류 문제

- 1024개(=32\*32) 입력
- 10개 카테고리, 10개 출력



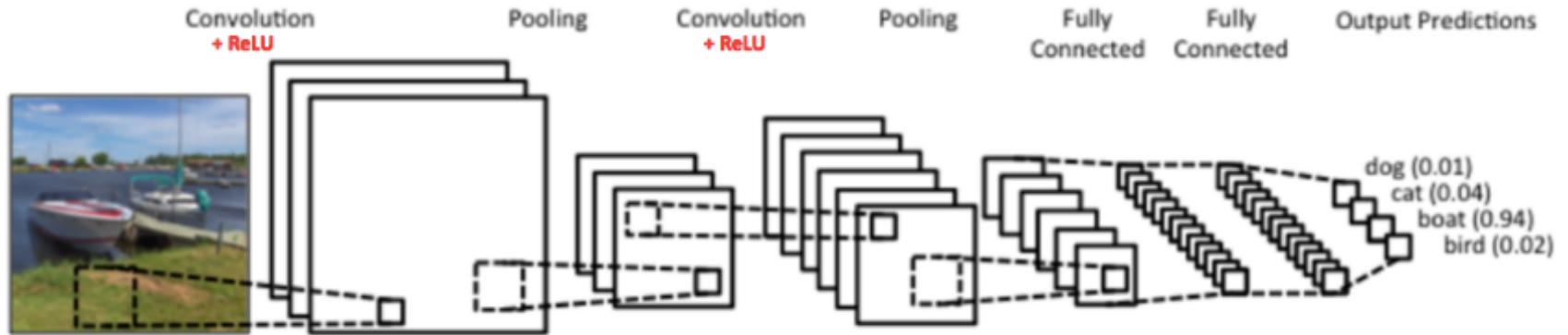


CNN



# CNN(Convolutional NN) 구조

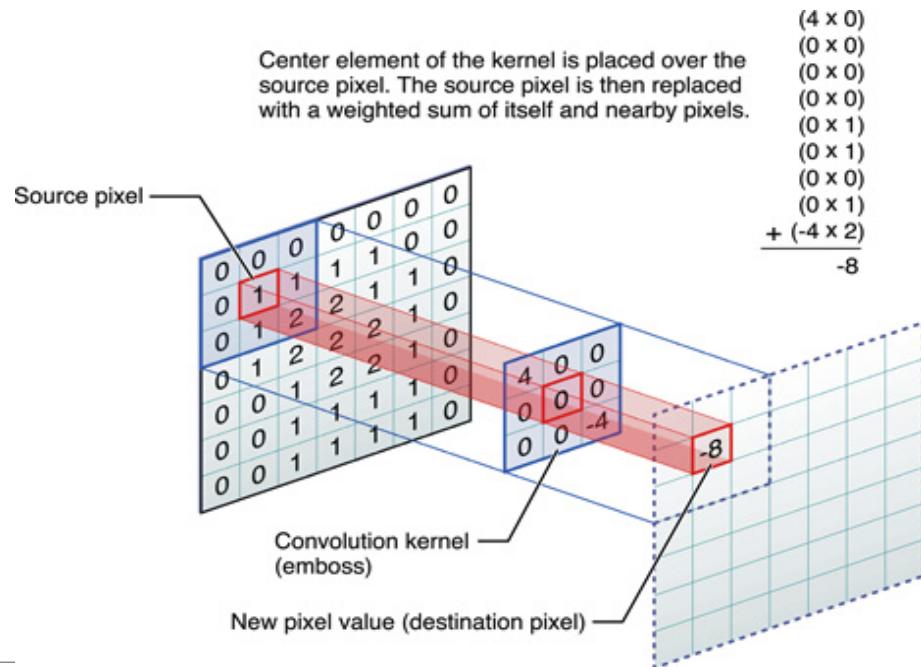
- Convolution filter와 pooling을 반복.
- 이후 일반 DNN에 적용.



# Convolution Filter

사람눈에는 특정 모양에 반응하는 신경세포들이 있다.

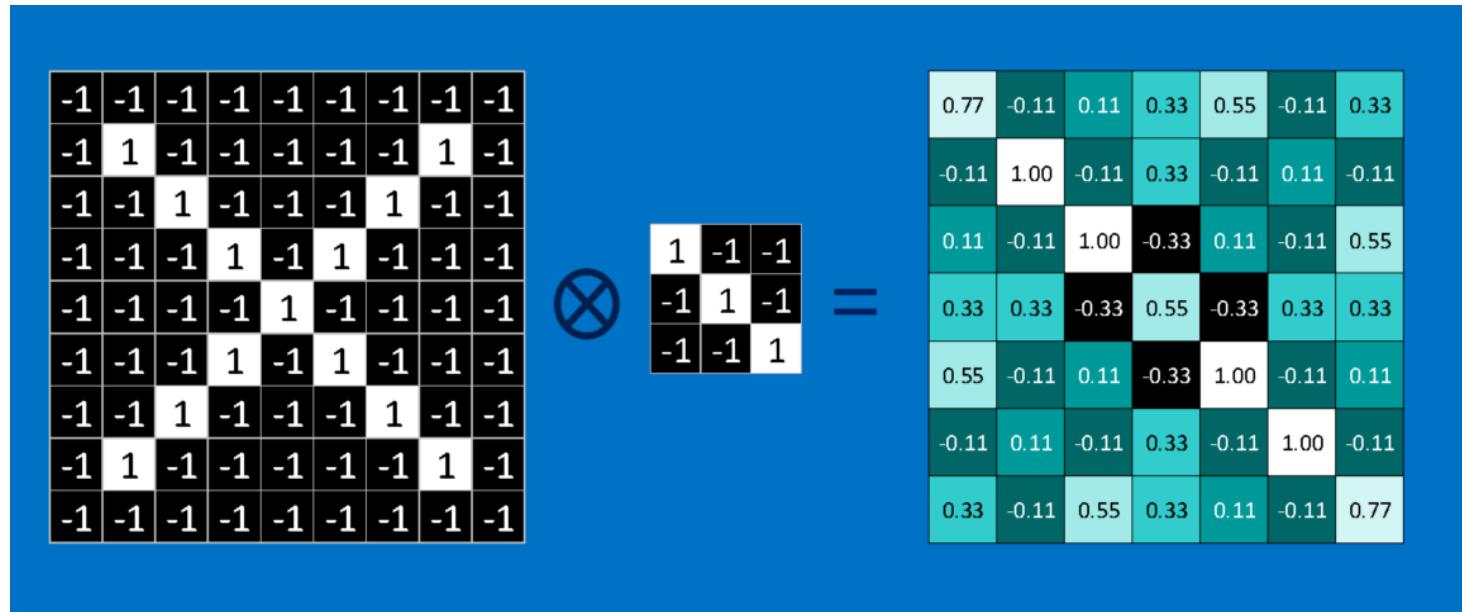
이를 구현한 가장 간단한 방법이 convolution filter이다.



# Convolution Filter - 의미

필터 모양의 자극이 있으면 그 결과가 최대가 된다.

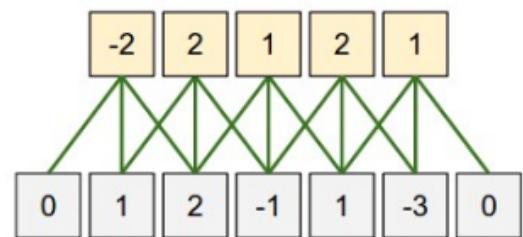
필터 모양이 있는지 찾아낸다.



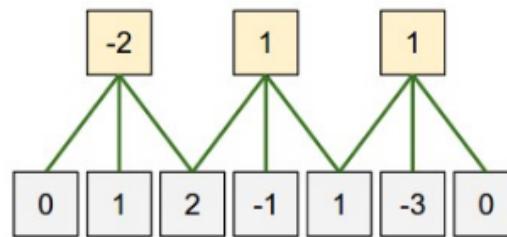
# Convolution Filter - Stride

필터 적용 시의 이동 칸 수.

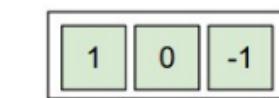
보통은 1.



stride 1

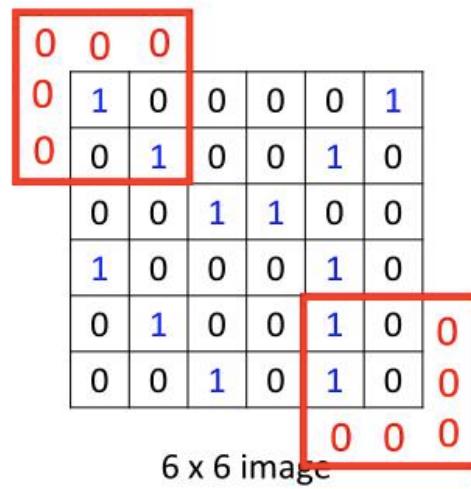


stride 2

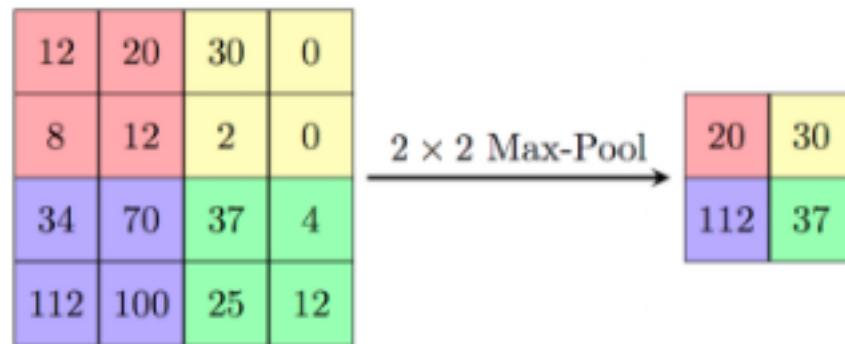


# Convolution Filter - Padding

필터 특성상 이미지가 작아지는 것을 방지하기 위해 원 이미지를 키우는 것.

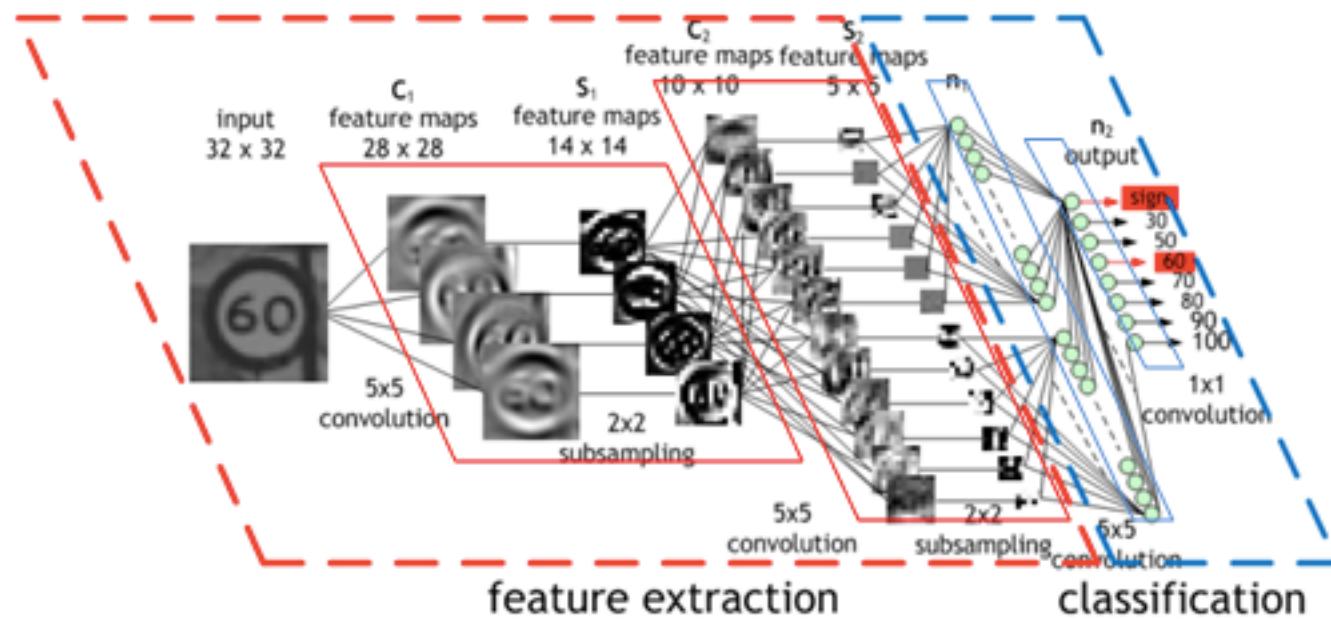


# Max Pooling



# 특징 추출과 분류

- convolution 필터와 pooling이 반복하여 feature 추출.
- 이후 일반 DNN로 분류.

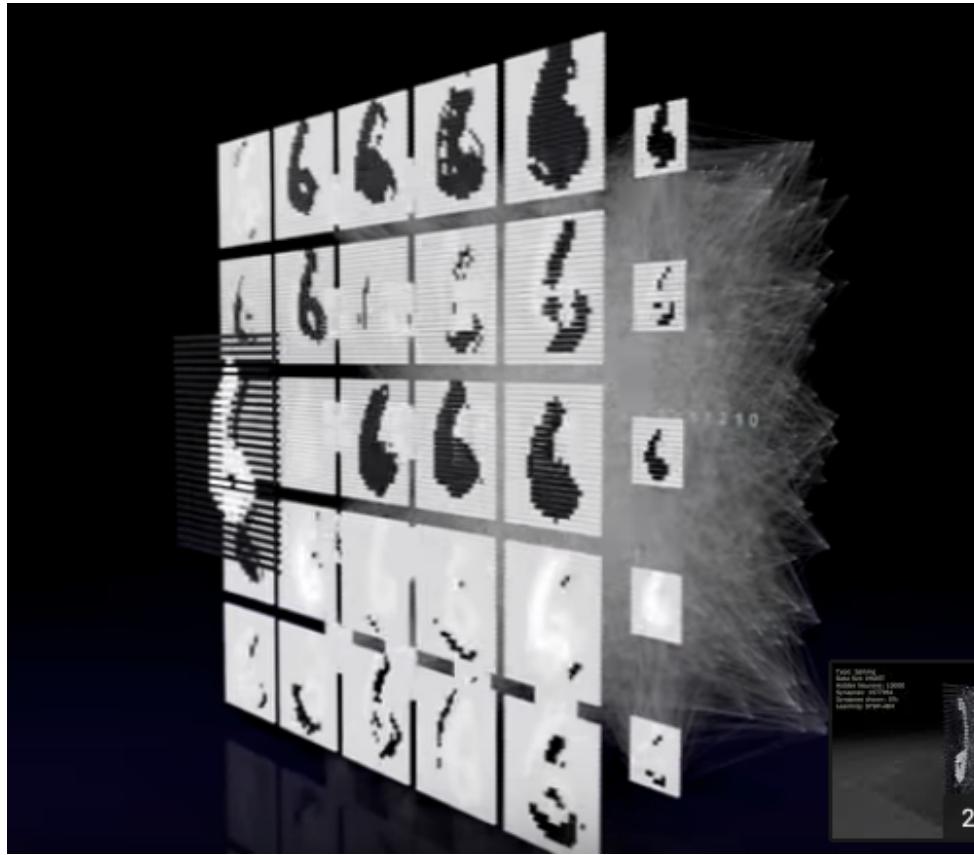


# CNN 성능

- 이미지 처리에 뛰어남.
- 이미지 외의 것에도 좋은 성능을 보임.



# CNN MNIST 시뮬레이션



<https://youtu.be/3JQ3hYko51Y?t=84>