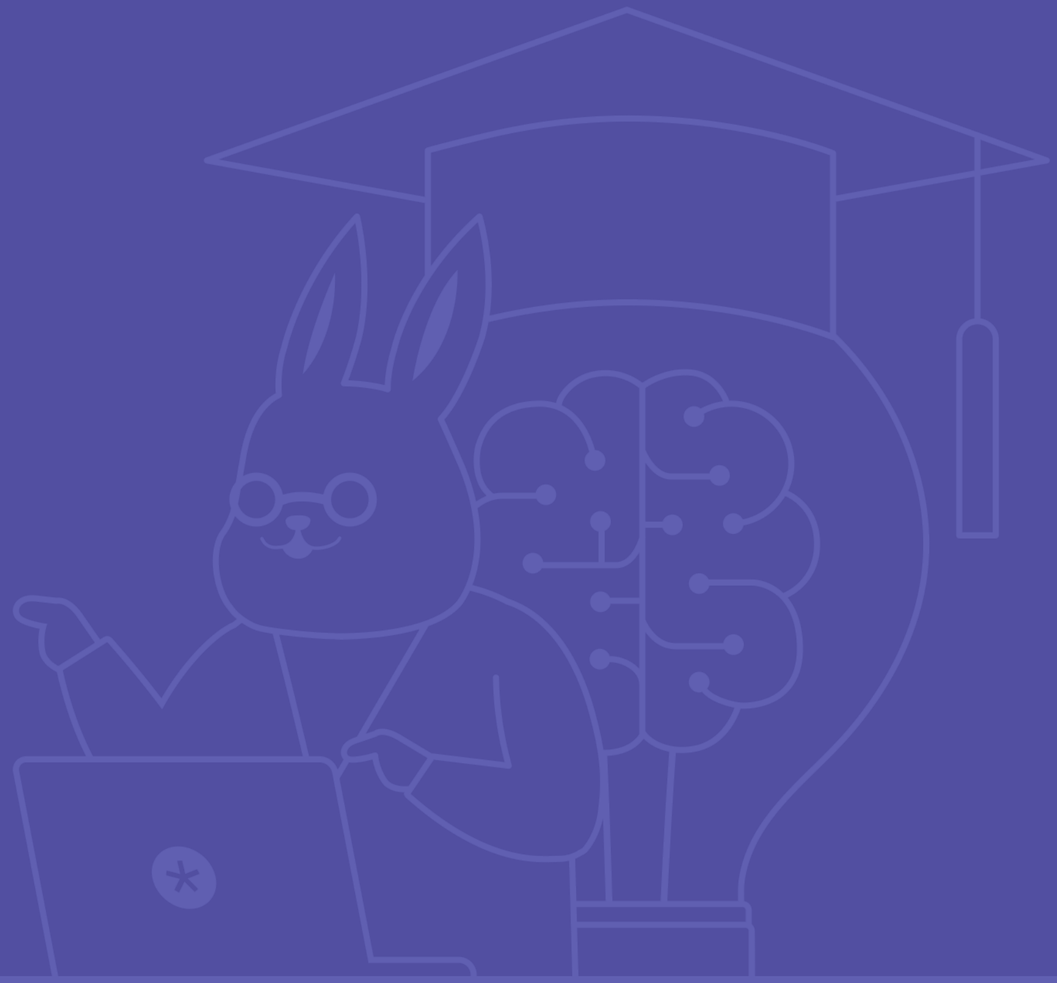
2020 AI College

10장 딥러닝의 개념 (1)

정민수 강사



/* elice */

Contents

Target

텐서 플로우에 대해 살펴본다.

딥러닝 프레임워크인 텐서 플로우에 대하여 살펴본다.

퍼셉트론과 다층 퍼셉트론을 이해한다.

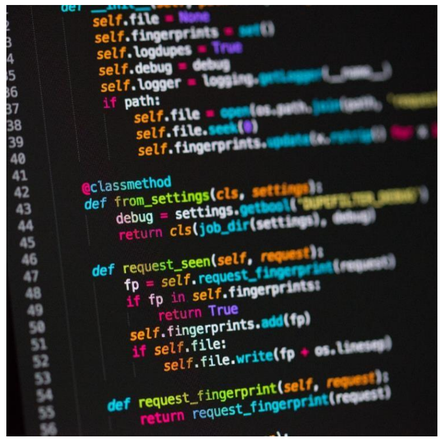
퍼셉트론과 다층 퍼셉트론에 대해 이해한다.

01

텐서플로우 (Tensorflow)

딥러닝 프레임워크

- 딥러닝 모델을 사용하기 위해서는 다음이 필요
  - 딥러닝 모델
  - 좋은 연산 장치
  - 연산장치 제어 기술



파이썬

하드웨어

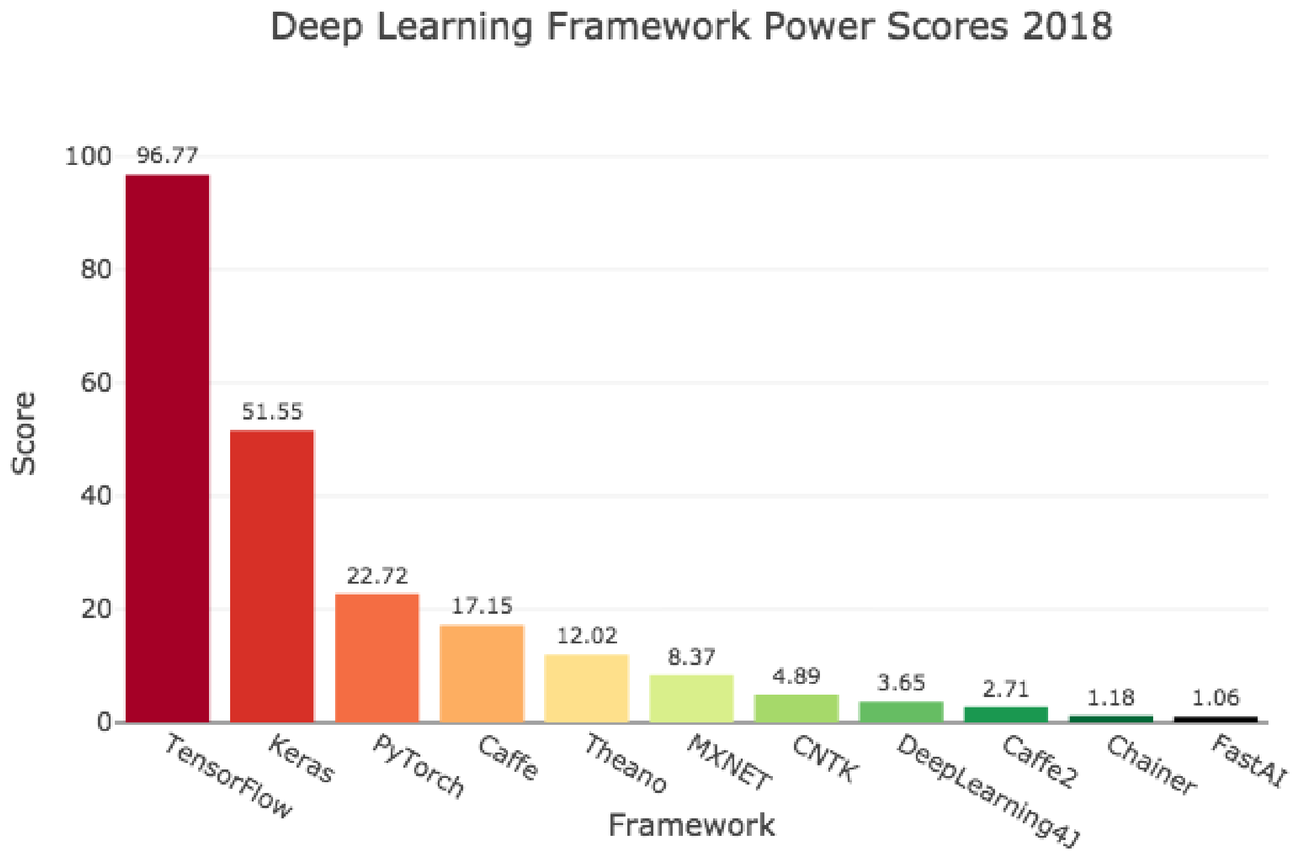
C/C++

딥러닝 프레임워크

- 딥러닝 프레임워크를 통해 해결 가능!

- 프레임워크(Framework)란?
  - 딥러닝 모델을 쉽게 구현하게 해주는 프로그램
  - 쉽고 빠른 코딩 지원
  - 자동 자원(GPU) 분배 및 활용

딥러닝 프레임워크 순위



Tensorflow (Keras + Theano) vs PyTorch (Caffe + Caffe2)

텐서플로우

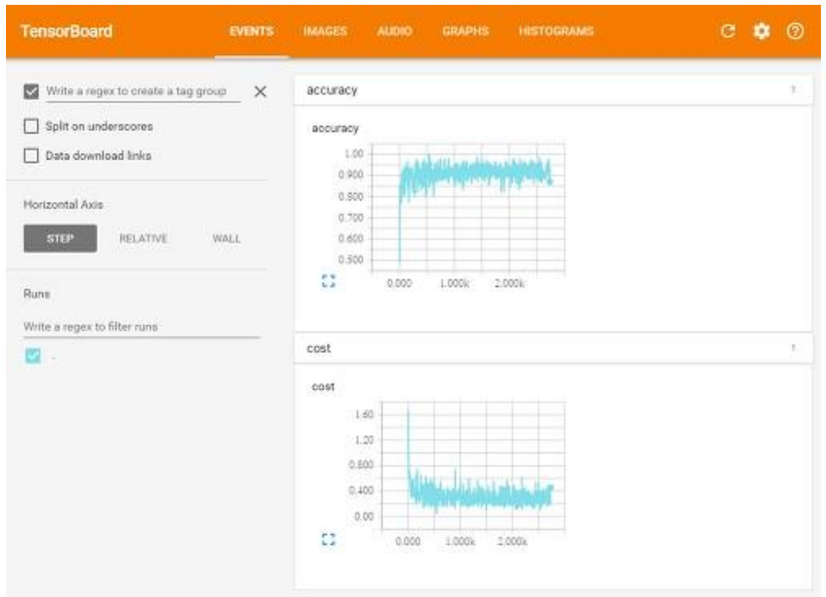
유연하고, **효율적**이며, **확장성**이 있는 딥러닝 프레임워크
대형 **클러스터 컴퓨터**부터 **스마트폰**까지 다양한 디바이스에서 동작
Python 2/3, C/C++ 지원

텐서플로우

"TensorFlow (Tensor + Flow) is an open source software library for numerical computation using data flow graphs"

- Tensor: 수학적 의미로 여러 방향을 가진 벡터 (다차원 배열 데이터)
- Flow: 데이터의 흐름, 즉 텐서라는 데이터의 흐름

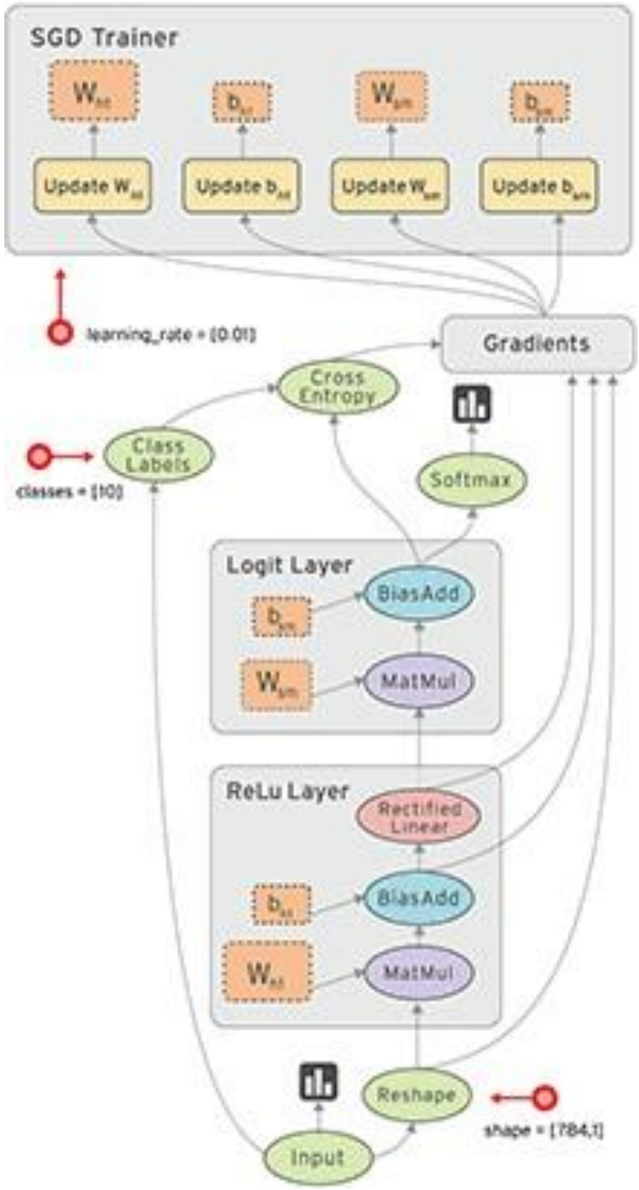
텐서보드



학습 과정에서 시각적인 분석 방법을 쉽게 사용 가능

텐서플로우 설계 방법: Data Flow

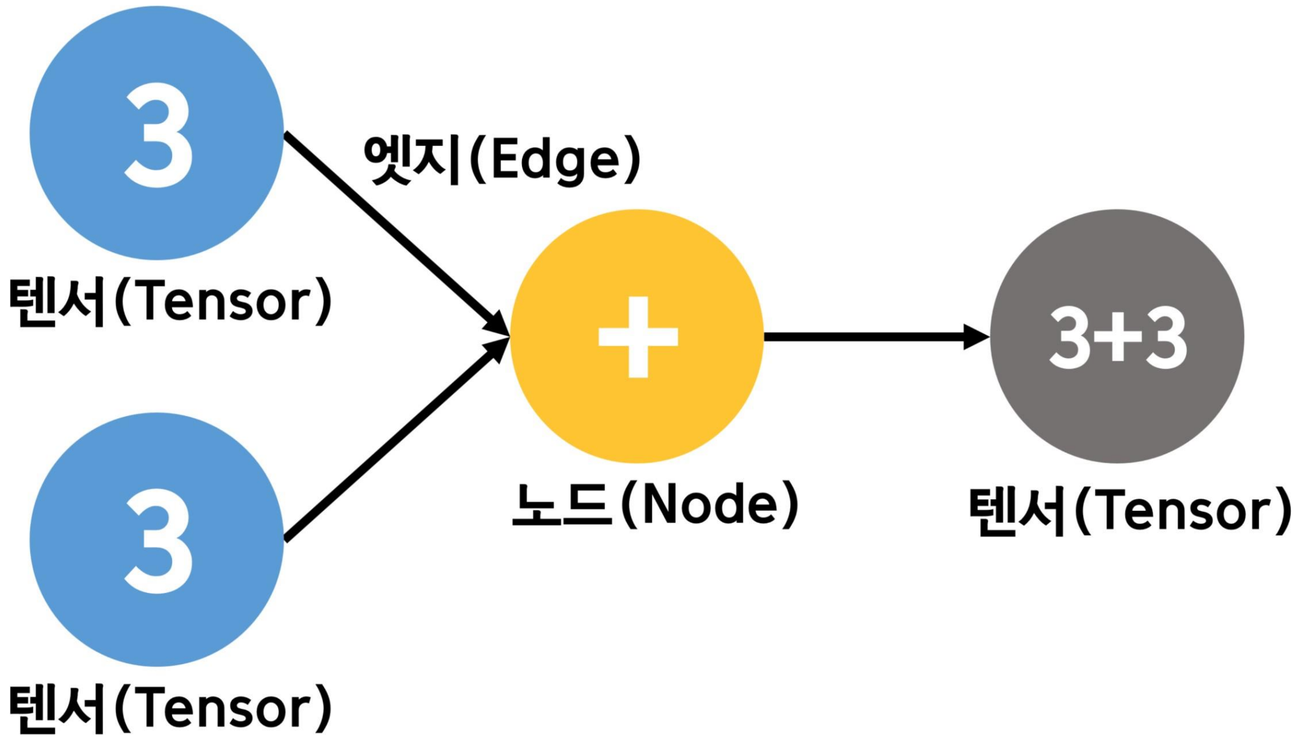
- 데이터 연산 과정을 flow로 표현
- 각 변수와 값들을 노드로 표현
- 화살표를 통해 노드 간 선행 관계를 표현



/* elice */

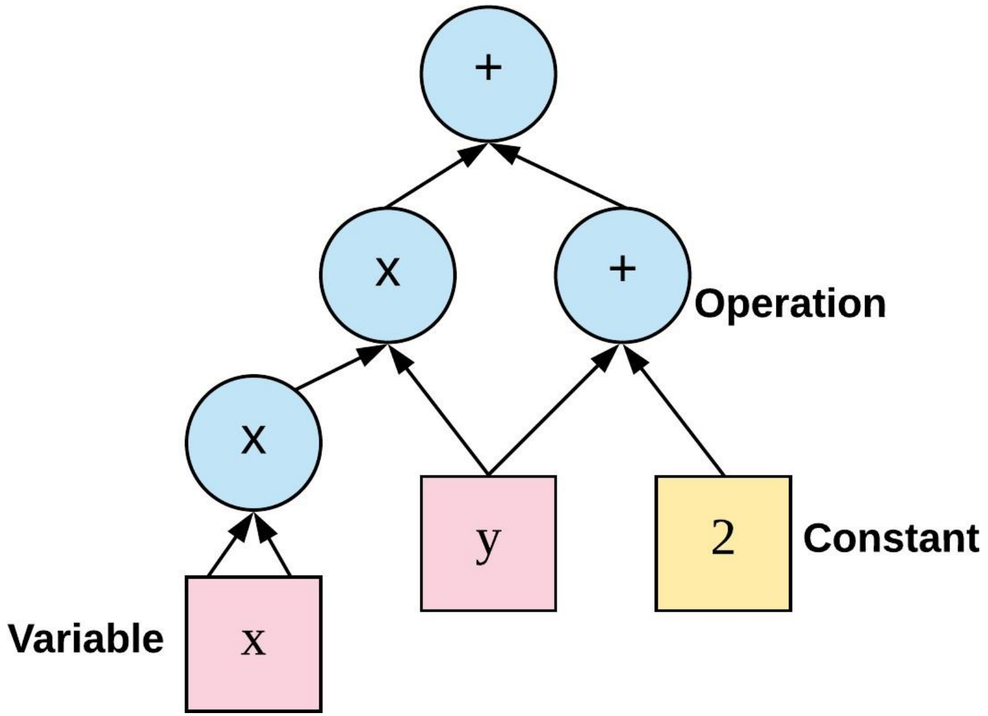
Graph model

- 그래프는 노드(Node)와 엣지(Edge)로 구성 노드는 연산(Operation), 엣지는 데이터가 오가는 통로



Graph model 사용 이유

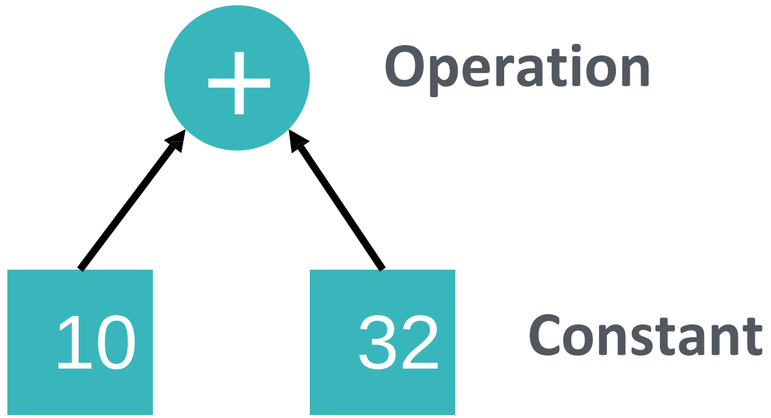
- ‘y+2’로 표시해도 되는데, 그래프로 표현하는 이유는?
- 복잡한 수식을 잘게 쪼개어 단순화
  - 단순화한 값은 미분에 유리함



Graph model 예시

Example

```
import tensorflow as tf
a = tf.constant (10)
b = tf.constant (32)
sess1 = tf.Session ()
print (sess1.run (a+b))
```



세션 (Session): 그래프를 실행하기 위해 필요
오퍼레이션의 실행 환경을 캡슐화

텐서플로우 실행 예시

Example

```
import tensorflow as tf
x = tf.constant(35, name='x')
y = tf.Variable(x+5,name='y')
model = tf.global_variables_initializer()
with tf.Session() as session:
session.run(model)
print(session.run(y))
```

1. Tensorflow 모듈 호출
2. x라는 변수 생성, 35 값으로 초기화
3. y라는 변수 생성, 방정식 x+5 정의
4. 모델 내부의 변수 초기화
5. 값을 계산하기 위해 세션 생성
6. 4에서 만든 모델 실행
7. 변수 y만 실행, 현재 값 출력

텐서플로우 2.0

- 편리성 향상
  - Keras와 즉시 실행(Eager execution)을 이용한 쉬운 모델 작성
  - 어떤 플랫폼에서든 튼튼한(Robust) 모델 배포
  - Deprecated된 API를 정리하고 중복을 줄여서 API 단순화

즉시 실행(Eager execution)

- Eager execution은 그래프 생성 없이 연산을 즉시 실행
- 실행할 계산 그래프 없이 실제 값을 얻는 것 가능
  - 직관적인 인터페이스
  - 쉬운 디버깅

예를 들면, 텐서플로우에서 간단한 계산을 수행한다고 생각해봅시다:

```
x = tf.placeholder(tf.float32, shape=[1, 1])
m = tf.matmul(x, x)

with tf.Session() as sess:
  print(sess.run(m, feed_dict={x: [[2.]]}))

# Will print [[4.]]
```

즉시 실행 (Eager execution)은 이 작업을 단순하게 할 수 있습니다:

```
x = [[2.]]
m = tf.matmul(x, x)

print(m)
```

Keras API

- 텐서플로우에 비해 고차원 (High-Level) API
- 고차원 API의 사용을 통해 사용자가 쉽게 개발 가능

```keras
import numpy as np
from keras.models import Sequential
from keras.layers.core import Activation, Dense

training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
target_data = np.array([[0],[1],[1],[0]], "float32")

model = Sequential()
model.add(Dense(32, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])

model.fit(training_data, target_data, nb_epoch=1000, verbose=2)

print model.predict(training_data)
```

Keras

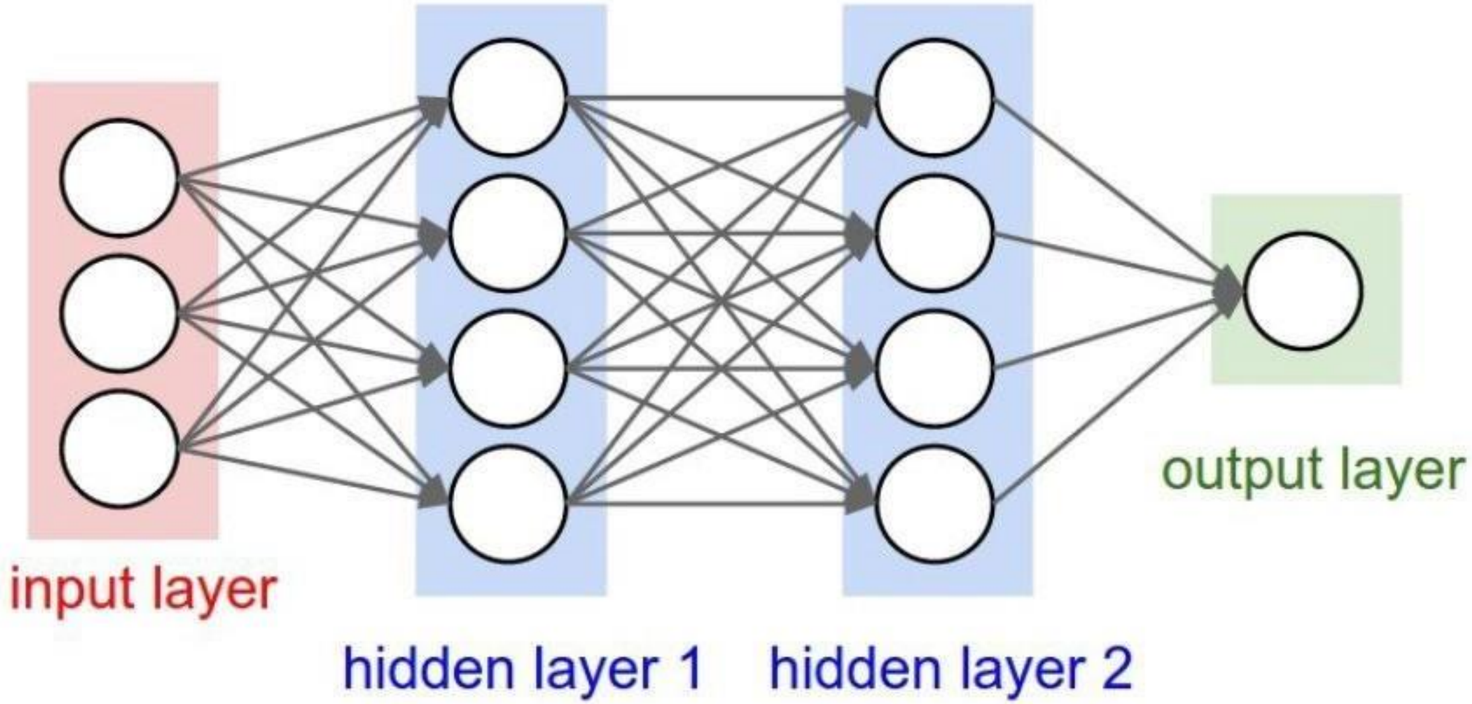
```tensorflow
import tensorflow as tf

input_data = [[0., 0.], [0., 1.], [1., 0.], [1., 1.]]  # XOR input
output_data = [[0.], [1.], [1.], [0.]]  # XOR output

n_input = tf.placeholder(tf.float32, shape=[None, 2], name="n_input")
n_output = tf.placeholder(tf.float32, shape=[None, 1], name="n_output")

hidden_nodes = 5

b_hidden = tf.Variable(tf.random_normal([hidden_nodes]), name="hidden_bias")
W_hidden = tf.Variable(tf.random_normal([2, hidden_nodes]), name="hidden_weights")
hidden = tf.sigmoid(tf.matmul(n_input, W_hidden) + b_hidden)

W_output = tf.Variable(tf.random_normal([hidden_nodes, 1]), name="output_weights")  # output layer's weight matrix
output = tf.sigmoid(tf.matmul(hidden, W_output))  # calc output layer's activation

cross_entropy = tf.square(n_output - output)  # simpler, but also works

loss = tf.reduce_mean(cross_entropy)  # mean the cross_entropy
optimizer = tf.train.AdamOptimizer(0.01)  # take a gradient descent for optimizing with a "stepsize" of 0.1
train = optimizer.minimize(loss)  # let the optimizer train

init = tf.initialize_all_variables()

sess = tf.Session()  # create the session and therefore the graph
sess.run(init)  # initialize all variables

for epoch in xrange(0, 2001):
    # run the training operation
    cvalues = sess.run([train, loss, W_hidden, b_hidden, W_output],
                       feed_dict={n_input: input_data, n_output: output_data})

    if epoch % 200 == 0:
        print("")
        print("step: {:>3}".format(epoch))
        print("loss: {}".format(cvalues[1]))

print("")
print("input: {} | output: {}".format(input_data[0], sess.run(output, feed_dict={n_input: [input_data[0]]})))
print("input: {} | output: {}".format(input_data[1], sess.run(output, feed_dict={n_input: [input_data[1]]})))
print("input: {} | output: {}".format(input_data[2], sess.run(output, feed_dict={n_input: [input_data[2]]})))
print("input: {} | output: {}".format(input_data[3], sess.run(output, feed_dict={n_input: [input_data[3]]})))
```

Tensorflow

/* elice */

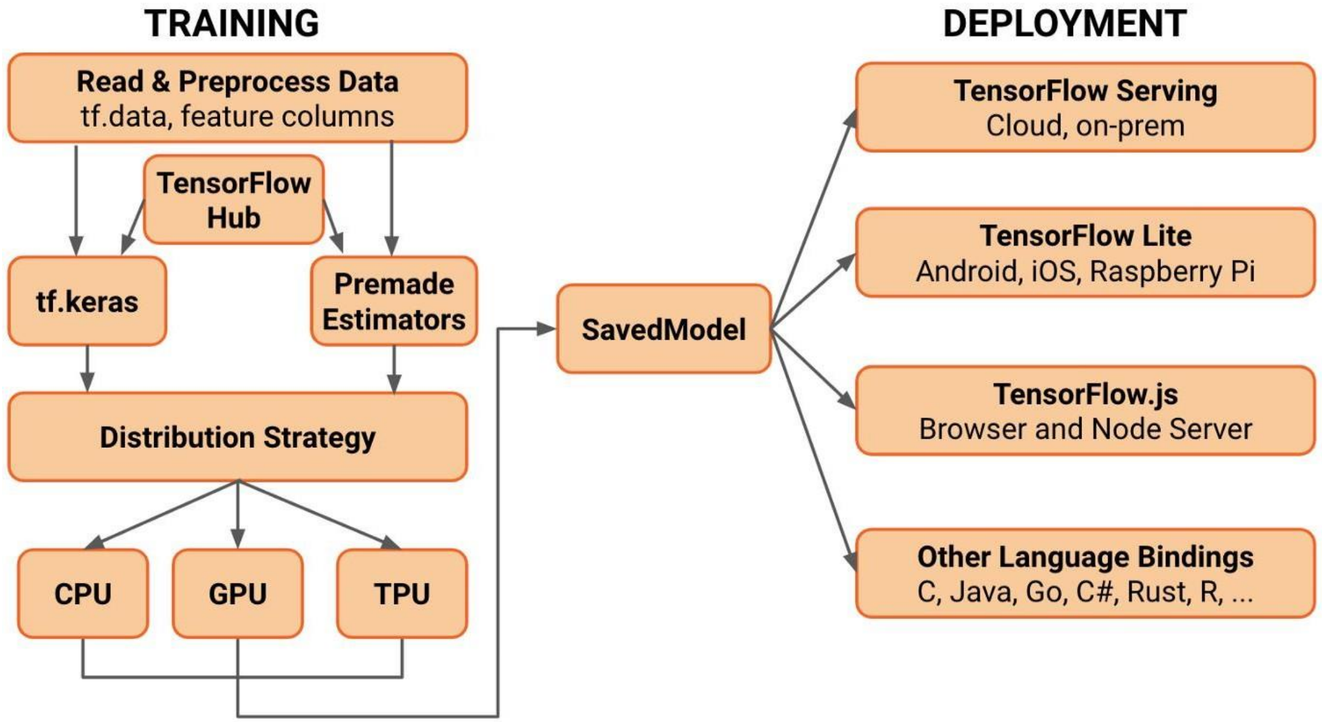
Keras API

- 텐서플로우는 그래프 기반으로 모델을 설계하지만 Keras는 레이어들을 선형으로 쌓는 Sequential 모델



튼튼한(Robust) 모델 배포

- 학습된 모델을 다양한 Deployment에서 동작 가능

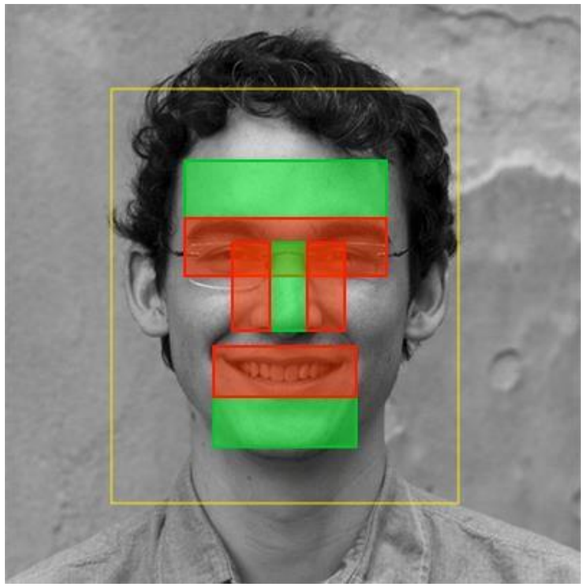


02

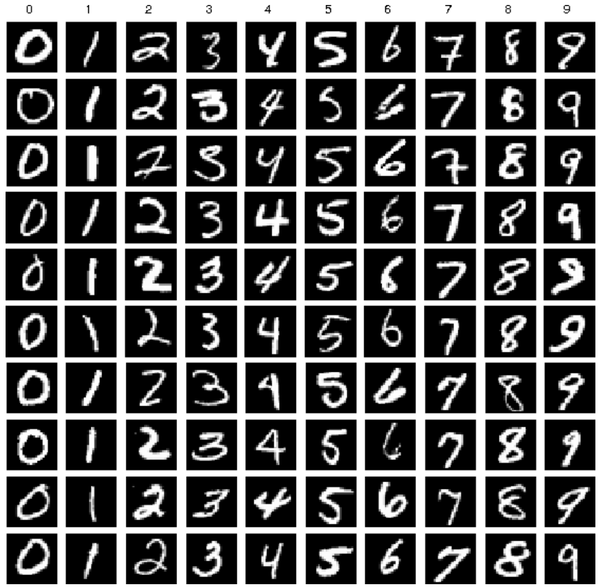
신경망 기초

신경망 이전의 연구

- 프로그래밍으로 풀 수 없는 문제들이 많았음
- 프로그래밍으로 풀기에 모든 경우에 대한 상황 처리가 불가능



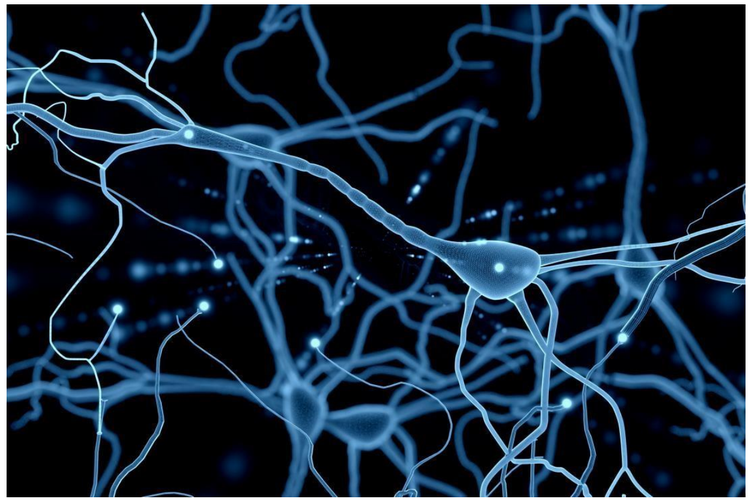
얼굴인식

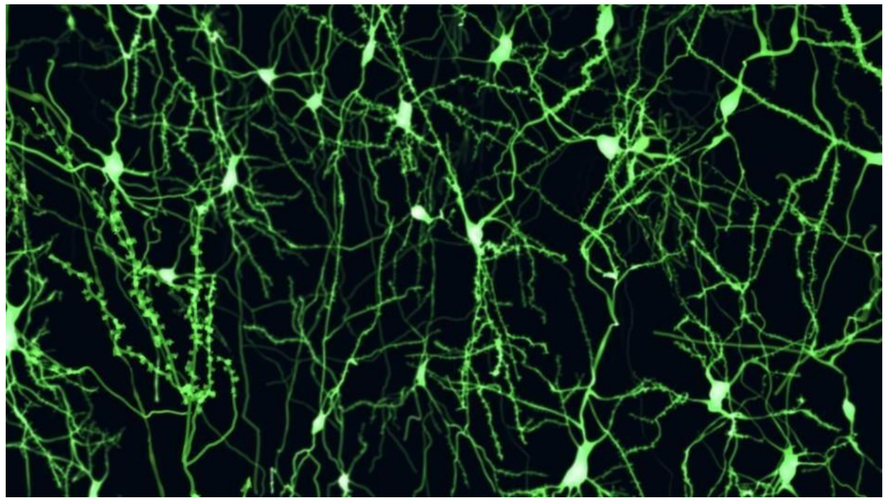


숫자 및 문자 인식

신경망이란?

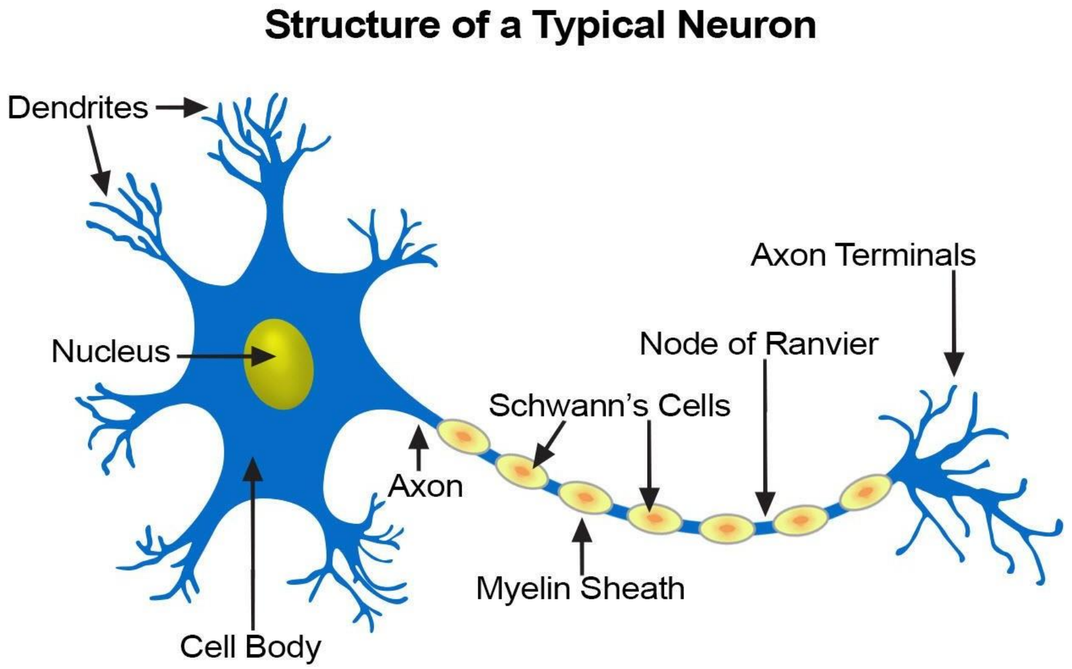
- 사람의 신경 시스템 (Neuron System)





Neuron

- 두뇌의 가장 작은 정보처리 단위
- 세포체 (cell body) : 간단한 연산, 수상돌기 (dendrite): 신호 수신,
- 축삭 (axon) : 처리 결과를 전송
- 사람은 1011개의 뉴런을 가지며, 뉴런은 103개 가량 다른 뉴런과 연결



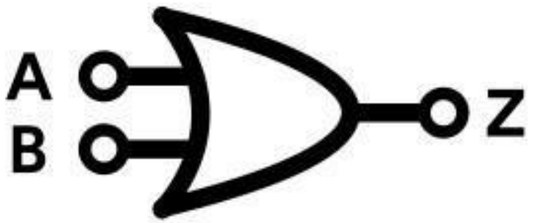
최초의 신경망

- 1943년 맥클럭-피츠 모델 (뉴런 세포를 논리적인 값으로 모델링)
- 뉴런은 활성화되거나 활성화되지 않는 2가지 상태
- 뉴런 흥분, 2개 이상의 고정된 수의 시냅스가 활성화

AND gate

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

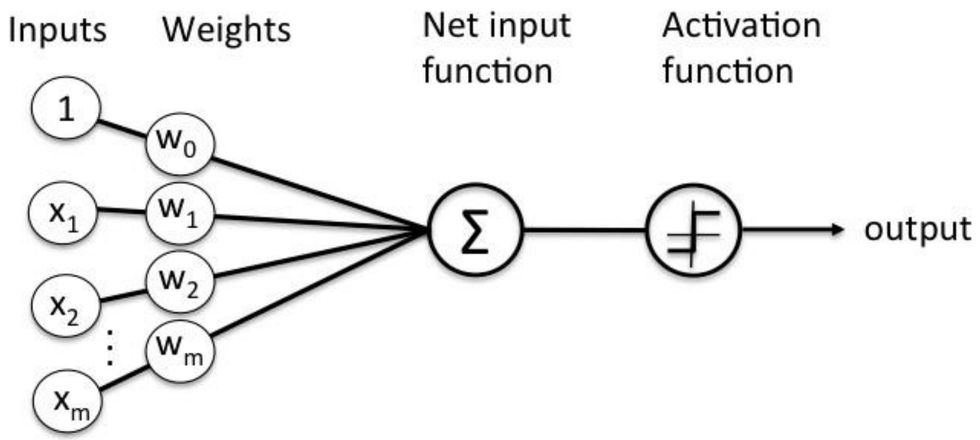
OR gate



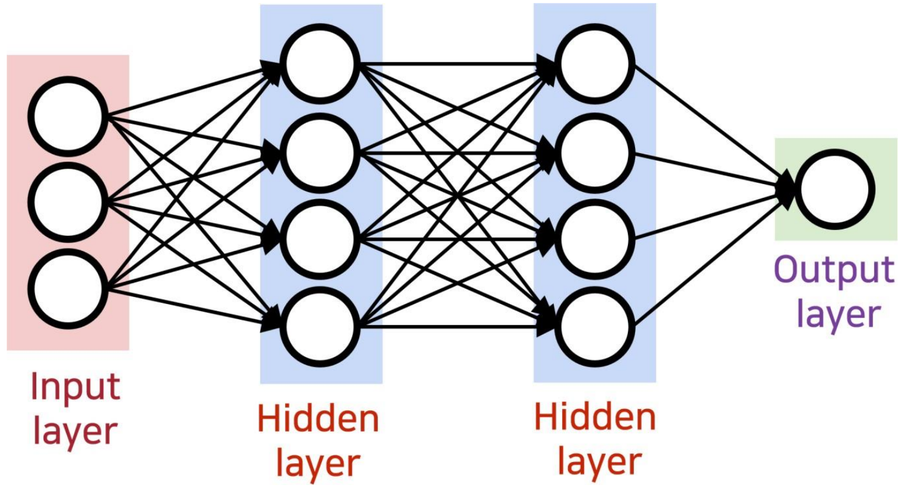
A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

퍼셉트론과 다층 퍼셉트론

- 1958년 퍼셉트론 (선형분류기)
- Input과 Weight의 곱을 모두 더한 뒤, 활성 함수를 통해 최종 결과



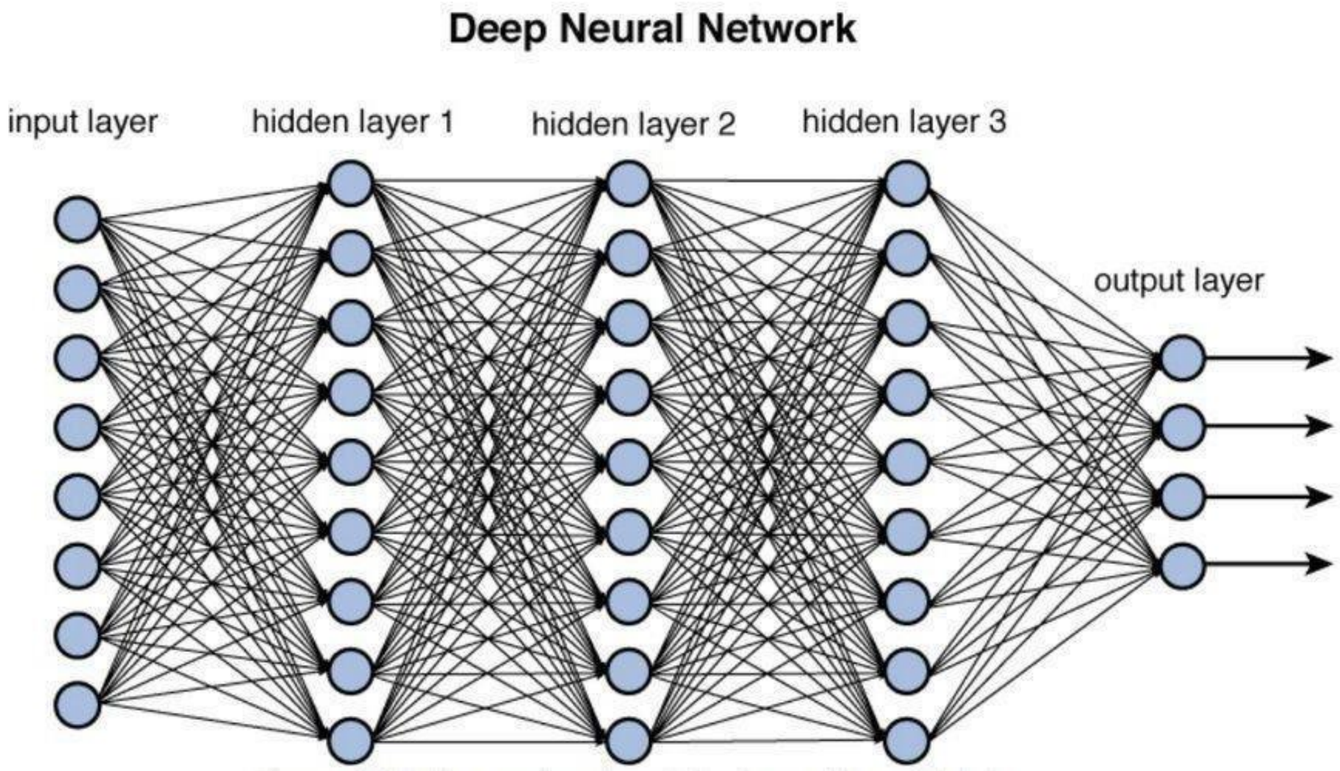
얼굴인식



다층 퍼셉트론

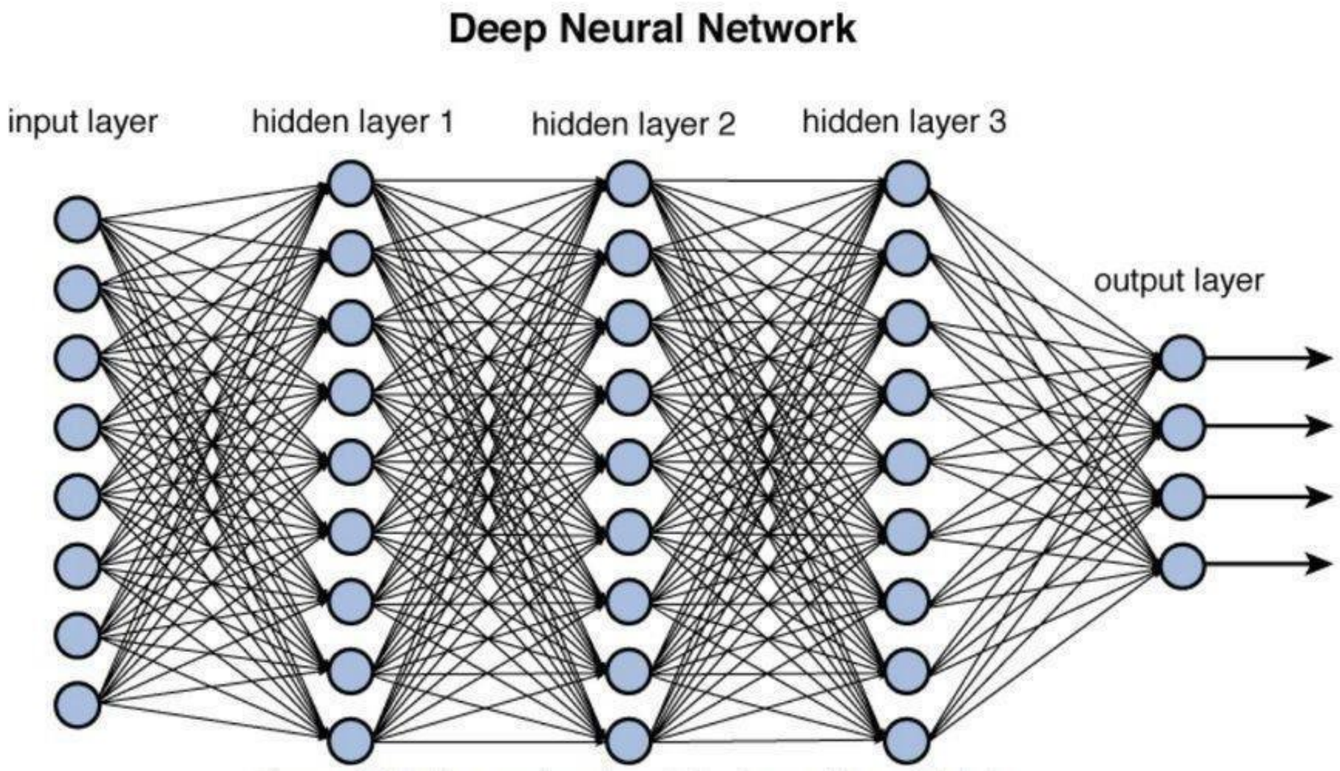
딥러닝

- 심층 신경망 (딥러닝): 망의 깊이가 깊은 신경망
- 학습 알고리즘의 개선, 컴퓨팅 파워의 향상으로 구현 가능



딥러닝

- 심층 신경망 (딥러닝): 망의 깊이가 깊은 신경망
- 학습 알고리즘의 개선, 컴퓨팅 파워의 향상으로 구현 가능

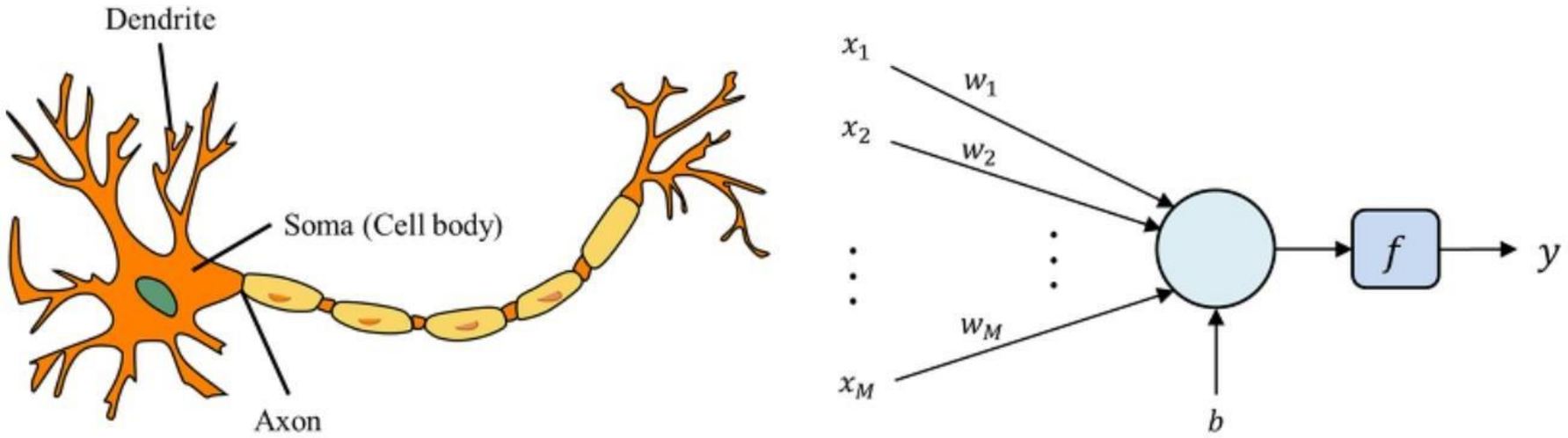


03

퍼셉트론 (Perceptron)

퍼셉트론

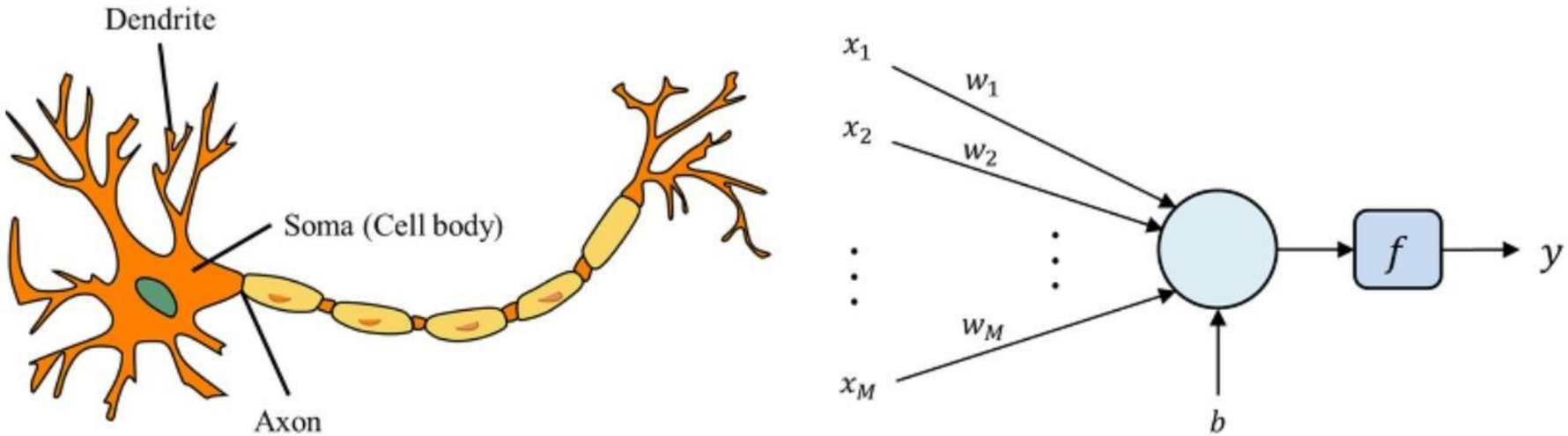
- 인공신경망 시스템은 동물의 신경계 시스템을 모사하여 설계



생물체의 neuron (좌)과 artificial neuron (우)

퍼셉트론 구조

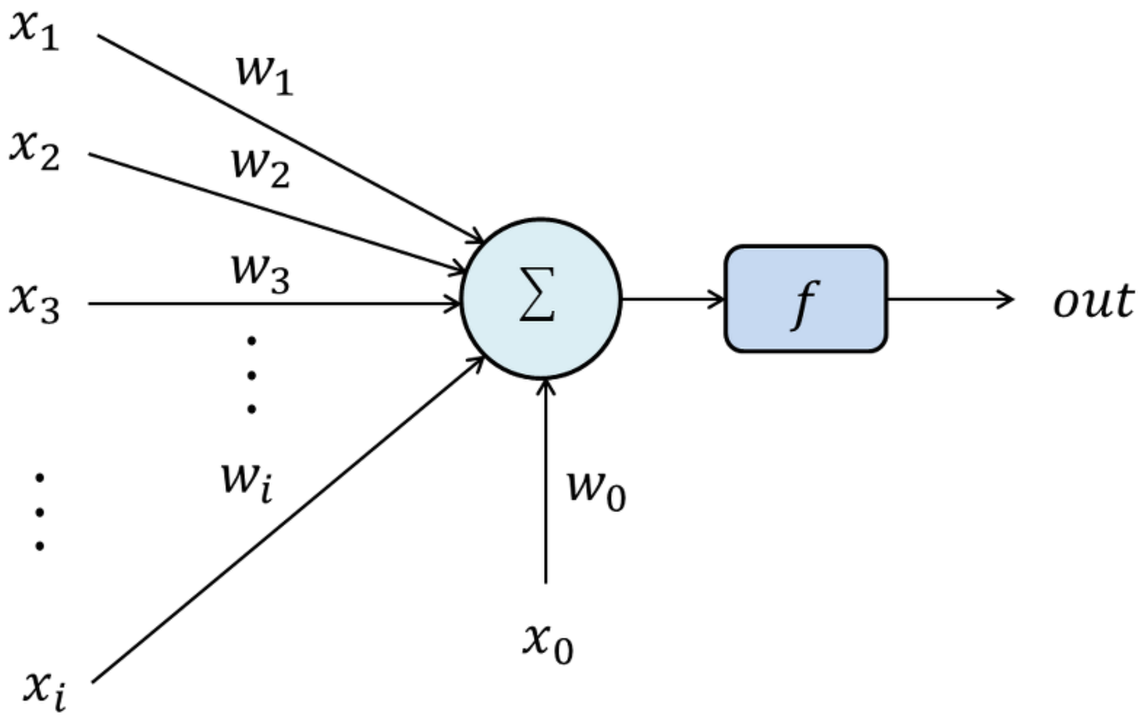
- 다수의 신호($x1$~xi)를 입력 받아 하나의 신호(out)을 출력
- 신호를 전달하는 역할($w1$~wi) : 가중치(weight)
- 퍼셉트론 동작 경향성($x0$) : 바이어스(bias)
- 신호의 전달 여부 결정(f) : 액티베이션 함수(activation)



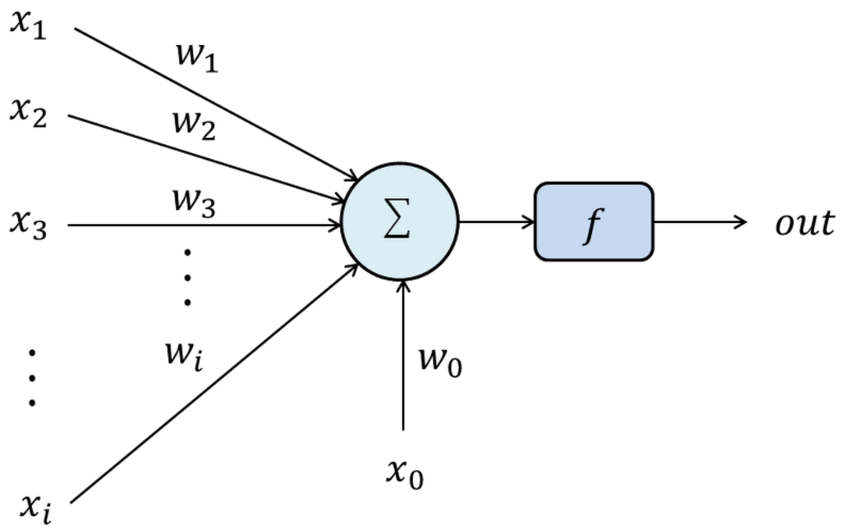
생물체의 neuron (좌)과 artificial neuron (우)

퍼셉트론 동작 방법

- 다수의 신호($x1$~xi)를 입력
- 각 신호에 대한 가중치(weight)를 곱한 후 모두 더함
- 계산된 값에 액티베이션 함수를 씌운 후 출력



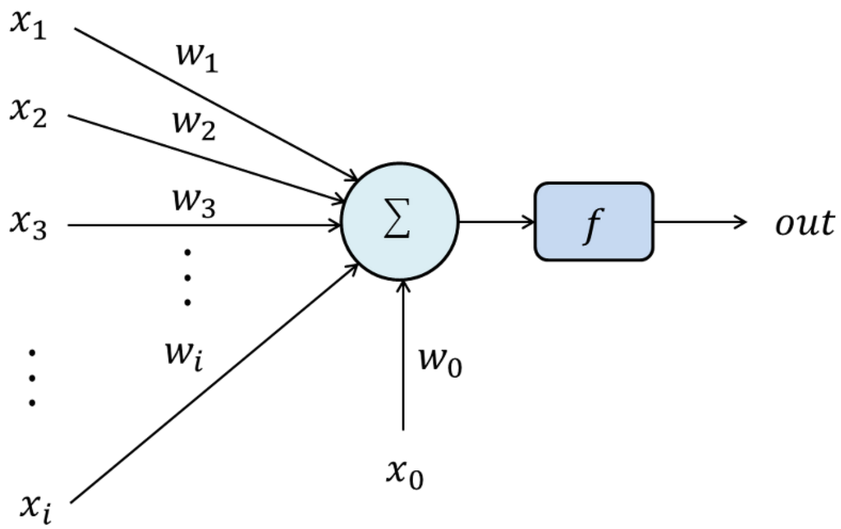
퍼셉트론 예시



x, w	의미
x_1	비가온다
x_2	친구가 만나자고한다
w_1	비를 좋아하는정도
w_2	친구를 좋아하는정도
y	외출한다/안한다

- 액티베이션 함수: y값이 얼마일때 친구를 보기 위해 외출은 하는가 ?
  - 1이면, 10이면, 0 보다 크면, -1.0이면, 0.5이면 등

퍼셉트론 예시



x, w, y	값	의미
x_1	1	비가온다
x_2	1	친구가 만나자고한다
w_1	-5	비를 좋아하는정도
w_2	6	친구를 좋아하는정도
y		외출한다/안한다

$$y = 1 * -5 + 1 * 6 = 1$$

난 태생적으로 비가 싫어 (bias): -2

$$y = 1 - 2 = -1$$

- y의 값이 0보다 작기 때문에, 외출 안함

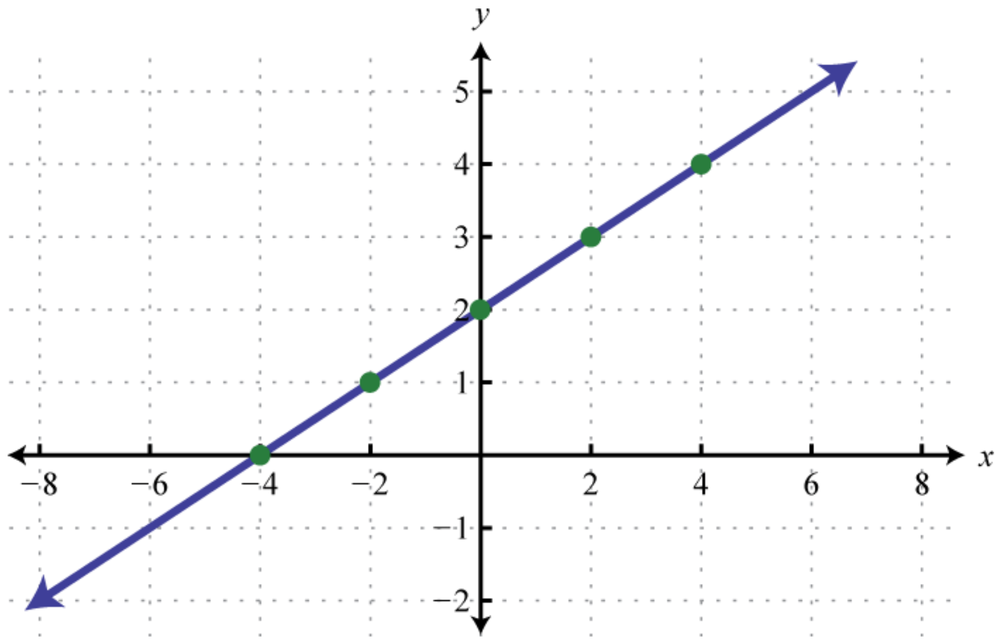
퍼셉트론 예시

$$y = 1 * -5 + 1 * 6 = 1$$

난 태생적으로 비가 싫어 (bias): -2

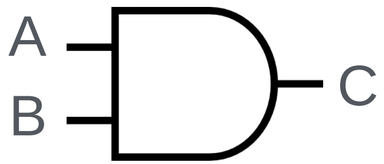
$$y = 1 - 2 = -1$$

$$y = x_1 * w_1 + x_2 * w_2 + bias$$



- 퍼셉트론은 선형 방정식으로 표현 가능함 (=선형 분류기)
- 퍼셉트론을 사용하여 논리 연산 (AND, OR, NAND) 가능

퍼셉트론 예시 - 게이트



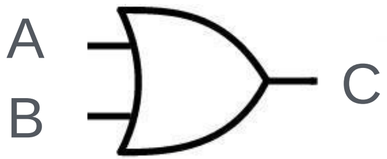
$$C > threshold, 1, else\ 0$$
$$(w1, w2, threshold) = (0.5, 0.5, 0.7)$$

AND

A/B	C
$0/0$	0
$1/0$	0
$0/1$	0
$1/1$	1

$0 * 0.5 + 0.5 * 1 < 0.7$

$1 * 0.5 + 0.5 * 1 > 0.7$



$$C > threshold, 1, else\ 0$$
$$(w1, w2, threshold) = (0.5, 0.5, 0.2)$$

OR

A/B	C
$0/0$	0
$1/0$	1
$0/1$	1
$1/1$	1

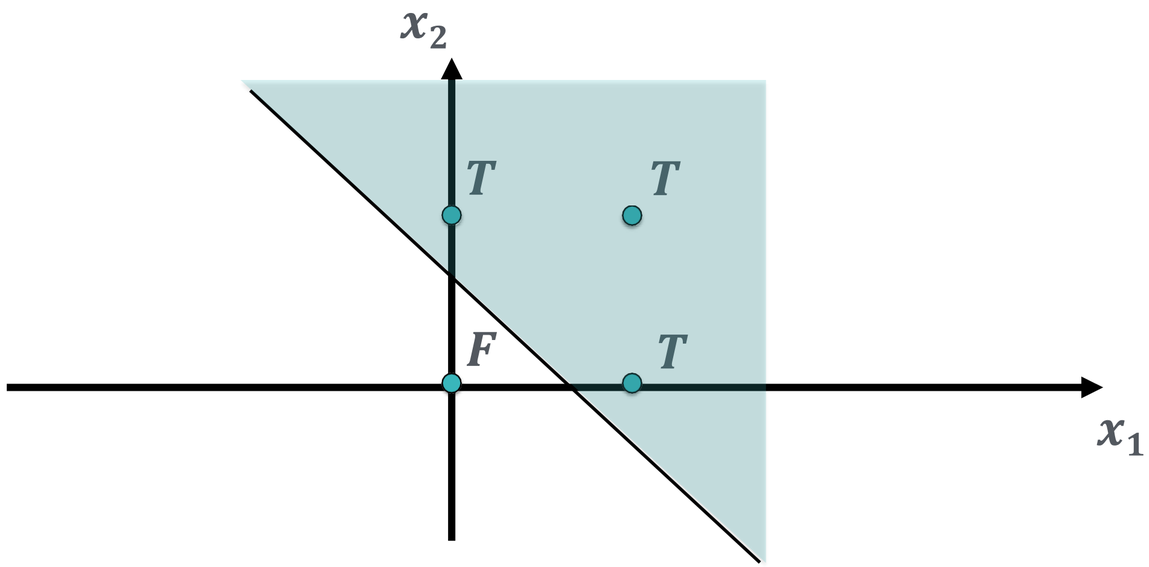
$0 * 0.5 + 0.5 * 1 > 0.2$

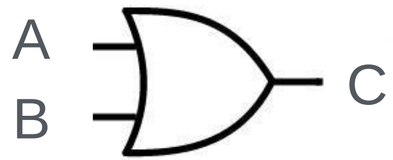
$1 * 0.5 + 0.5 * 0 > 0.2$

/* elice */

퍼셉트론 예시 – OR 게이트

- 퍼셉트론의 선형 방정식을 통해 OR 게이트 표현 가능





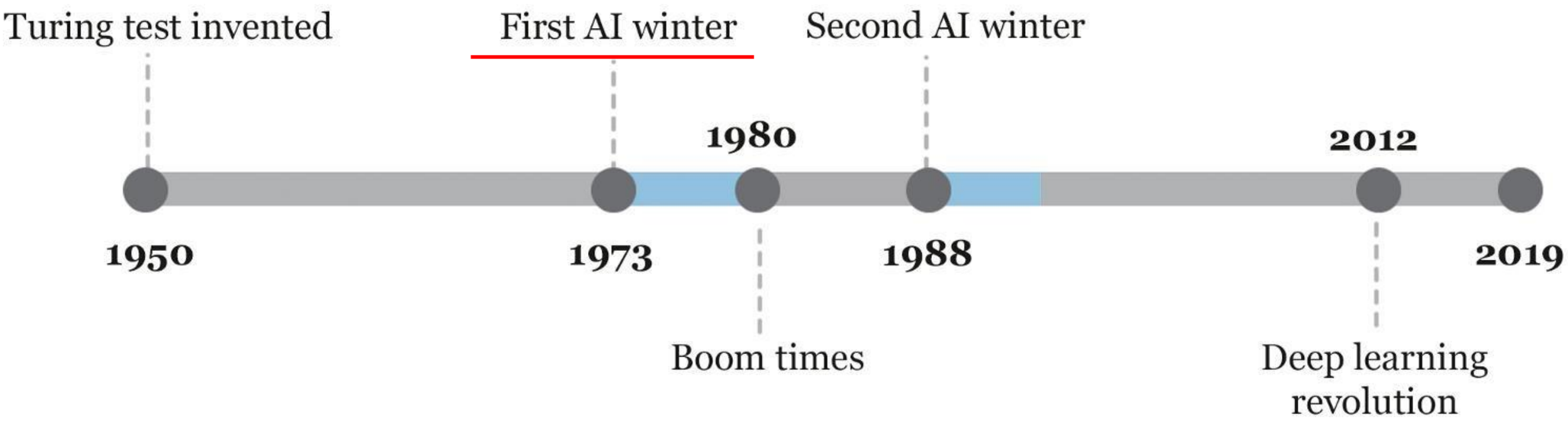
OR

A/B	C
$0/0$	0
$1/0$	1
$0/1$	1
$1/1$	1

/* elice */

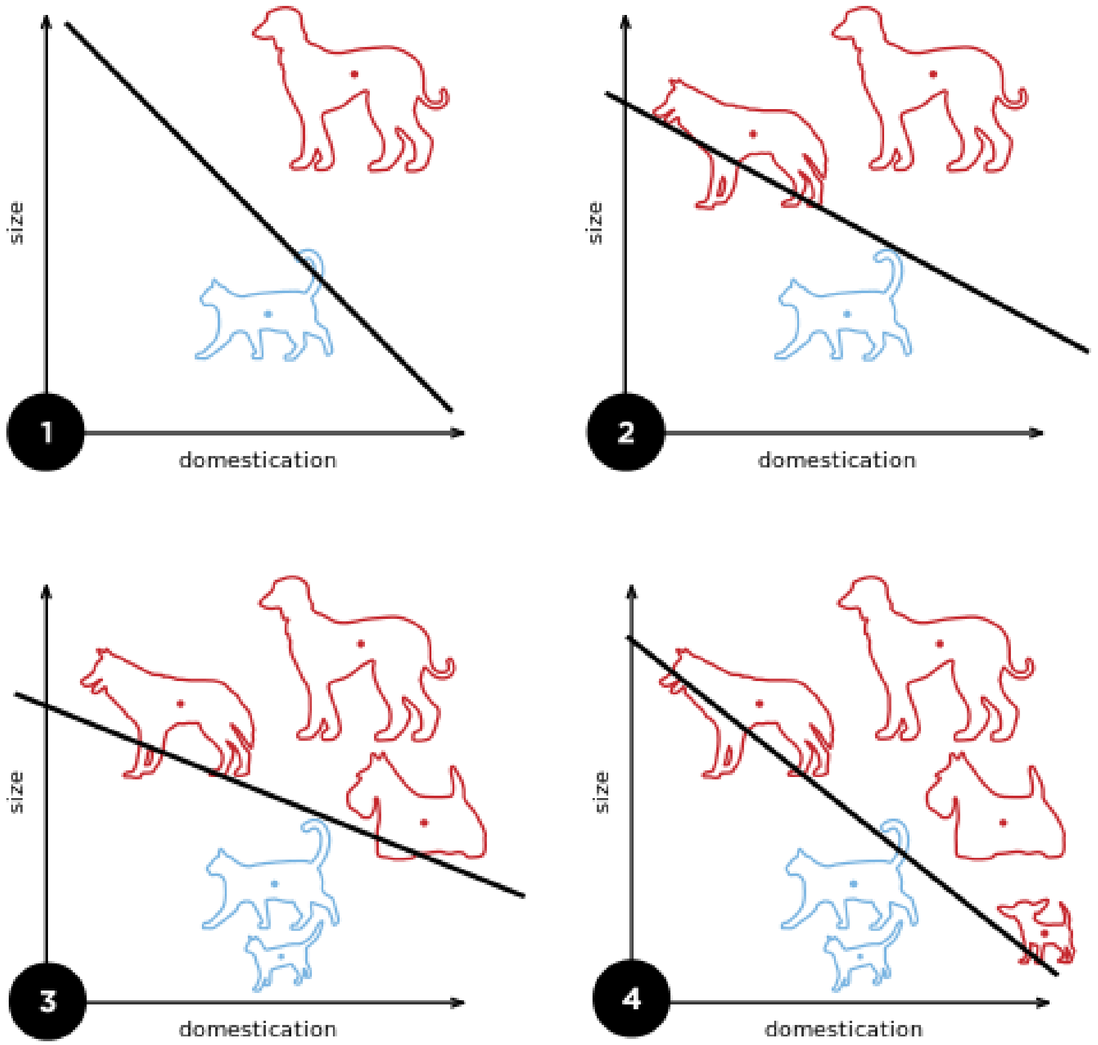
1차 AI 겨울

- 그렇다면 비 선형적인 문제는 어떻게 해결할까?



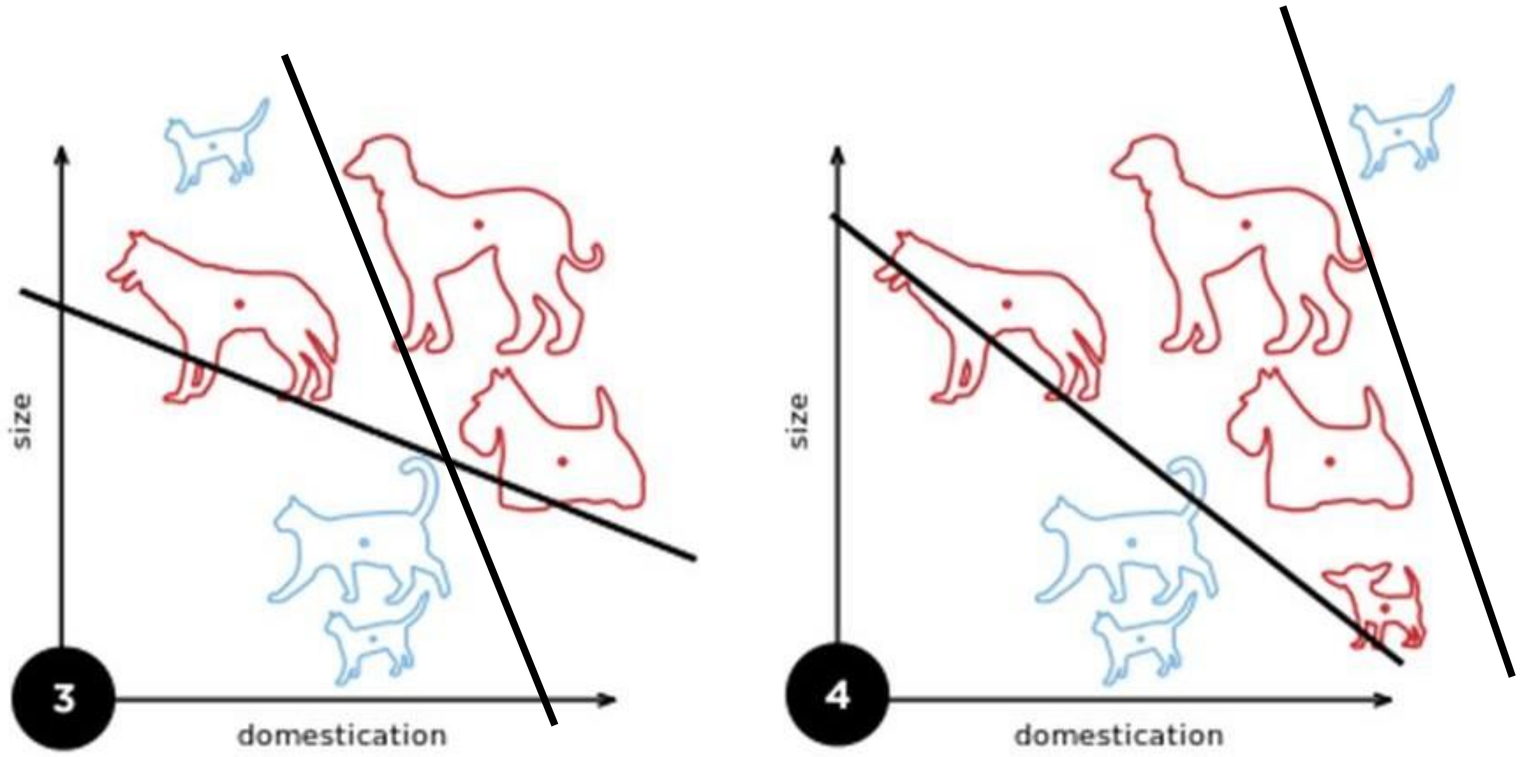
선형적 문제

- 직선으로 분류가 되는 문제



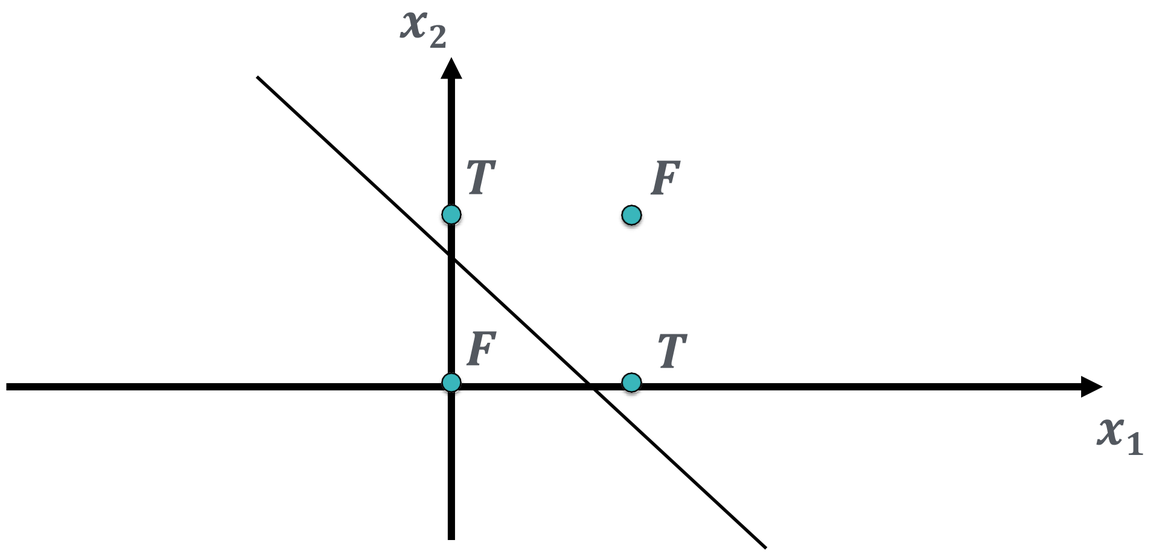
비선형적 문제

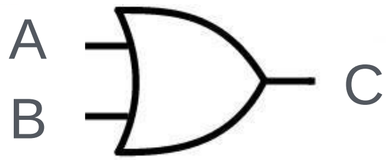
- 직선으로 분류가 되지 않는 문제



비선형적 문제 – XOR 게이트

- XOR 게이트는 선형 방정식으로 표현 불가능





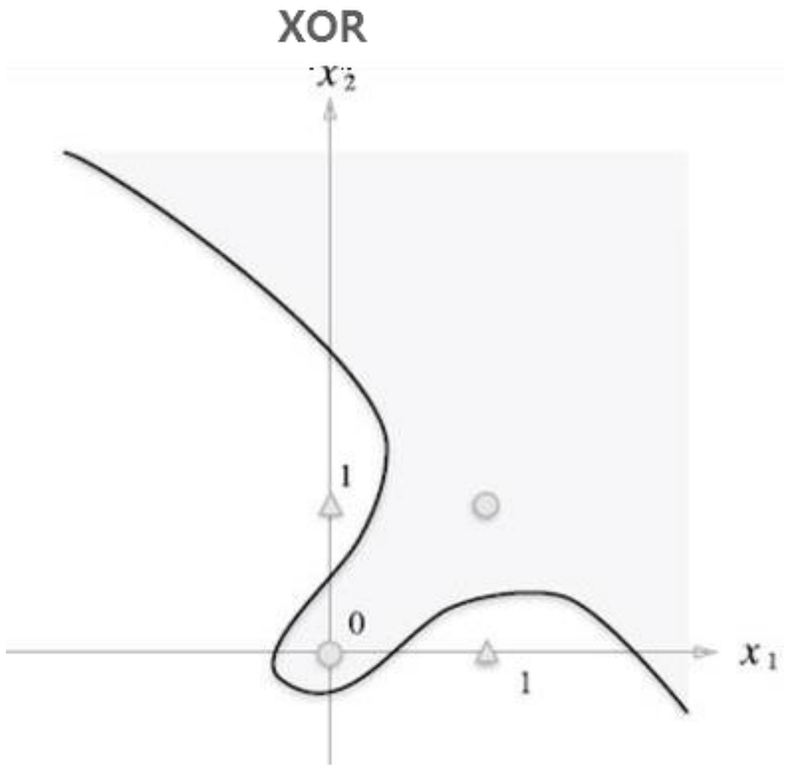
XOR

A/B	C
$0/0$	0
$1/0$	1
$0/1$	1
$1/1$	0

/* elice */

퍼셉트론의 문제점

- 단층 퍼셉트로은 XOR 문제 뿐만 아니라 다양한 문제의 해결 불가능
- 선형적인 특성을 벗어난 비선형 적인 접근 방법 필요

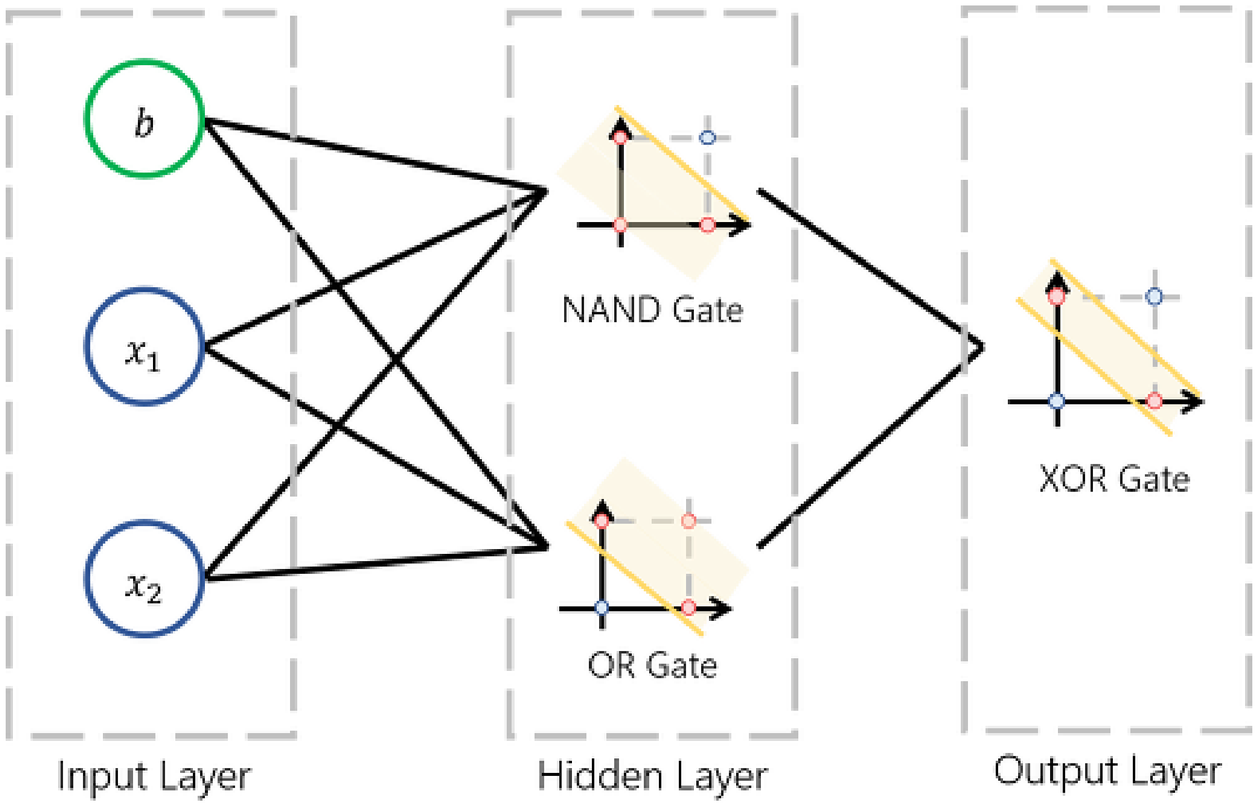


04

다층 퍼셉트론 (Multi-layer Perceptron)

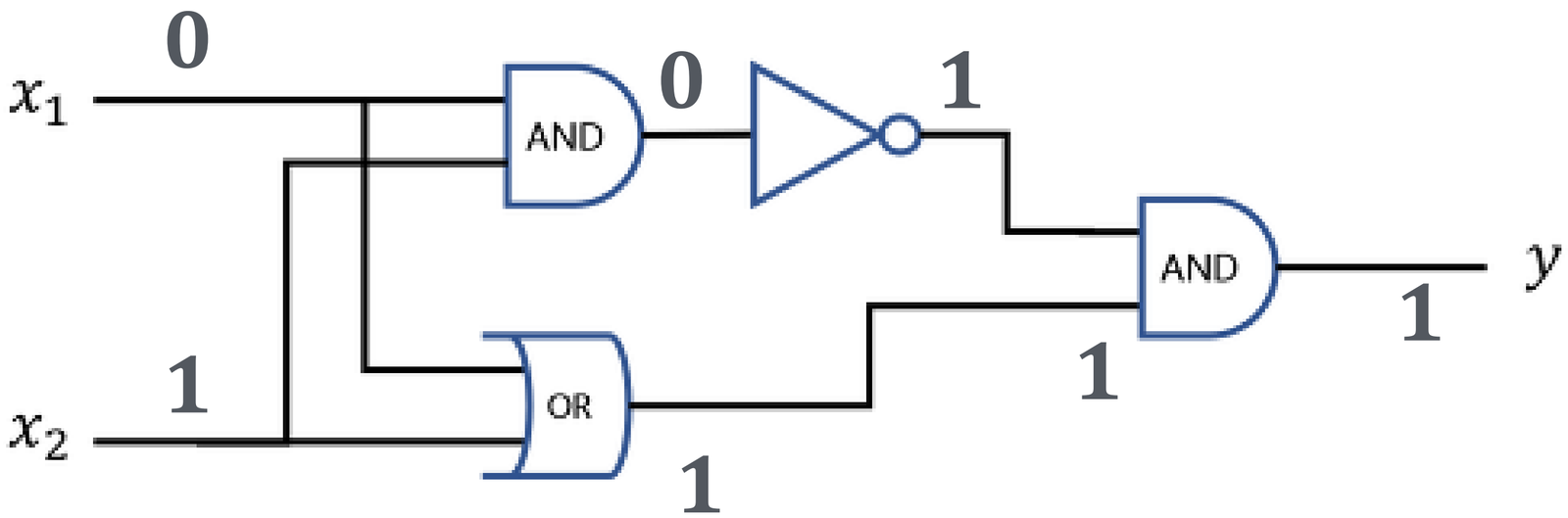
다층 퍼셉트론

- XOR 연산은 하나의 레이어를 사용하여 표현하는 것은 불가능
- 하지만 NAND와 OR 연산을 사용하여 표현 가능 (여러 퍼셉트론을 쌓아서 문제 해결)



XOR = NAND + OR + AND

- 여러 개의 퍼셉트론을 조합하면 XOR 게이트를 만들 수 있음

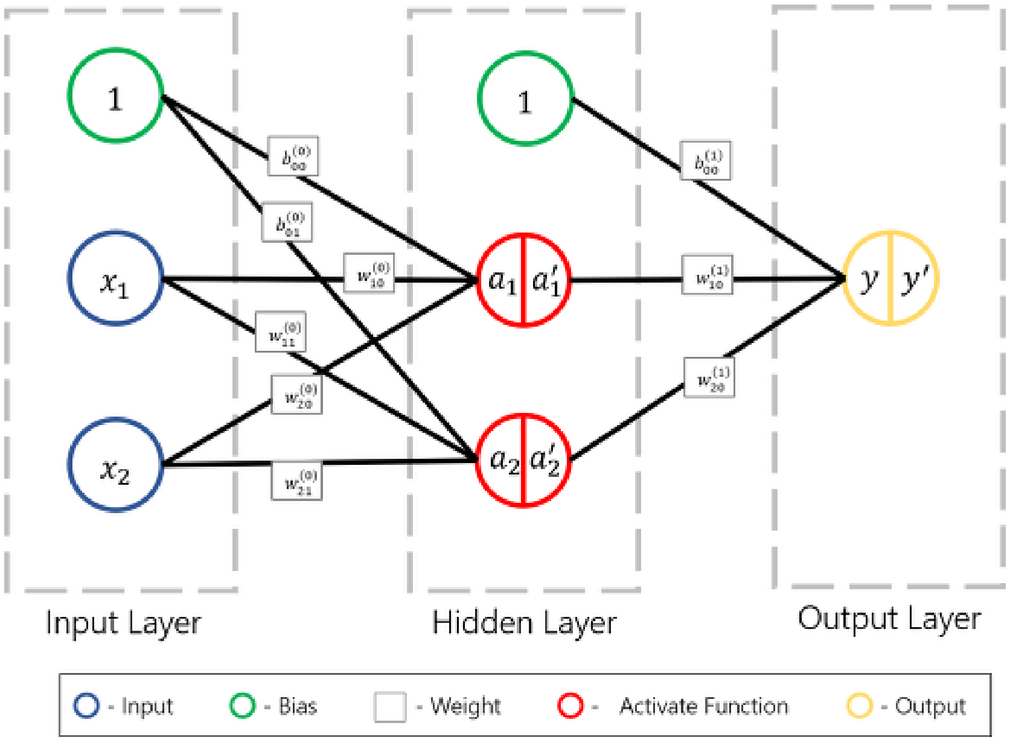


A/B	C
0/0	0
1/0	1
0/1	1
1/1	0

/* elice */

다층 퍼셉트론

- Multi Layer Perceptron으로 MLP라고 불림
- 처음 값 (Input), 결과 출력 (Output), 중간 부분 (Hidden)으로 구성됨
- Hidden Layer가 3층 이상 되면 Deep NN (DNN)

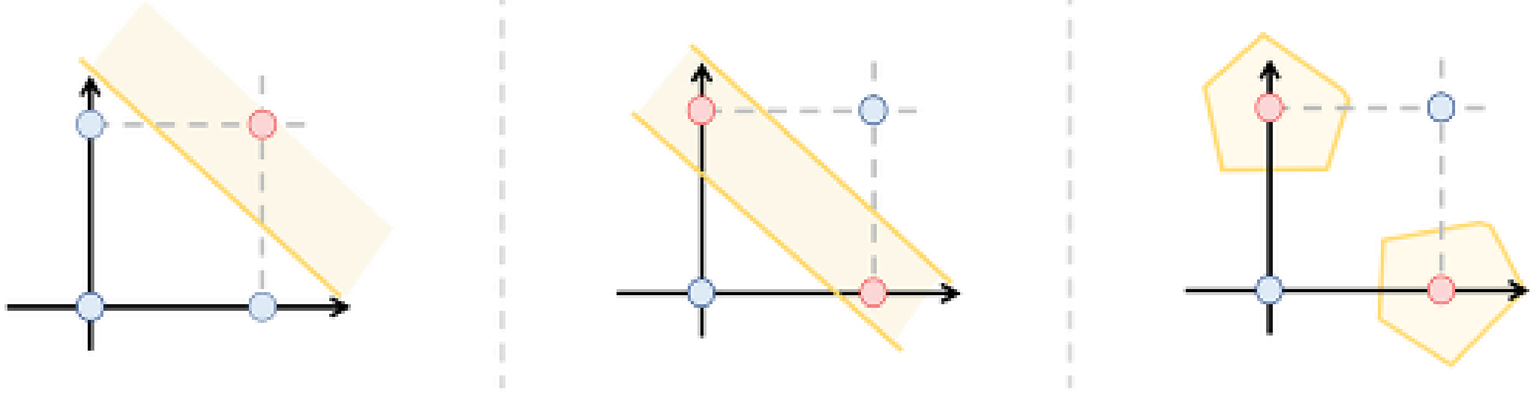


/* elice */

MLP가 결정할 수 있는 영역

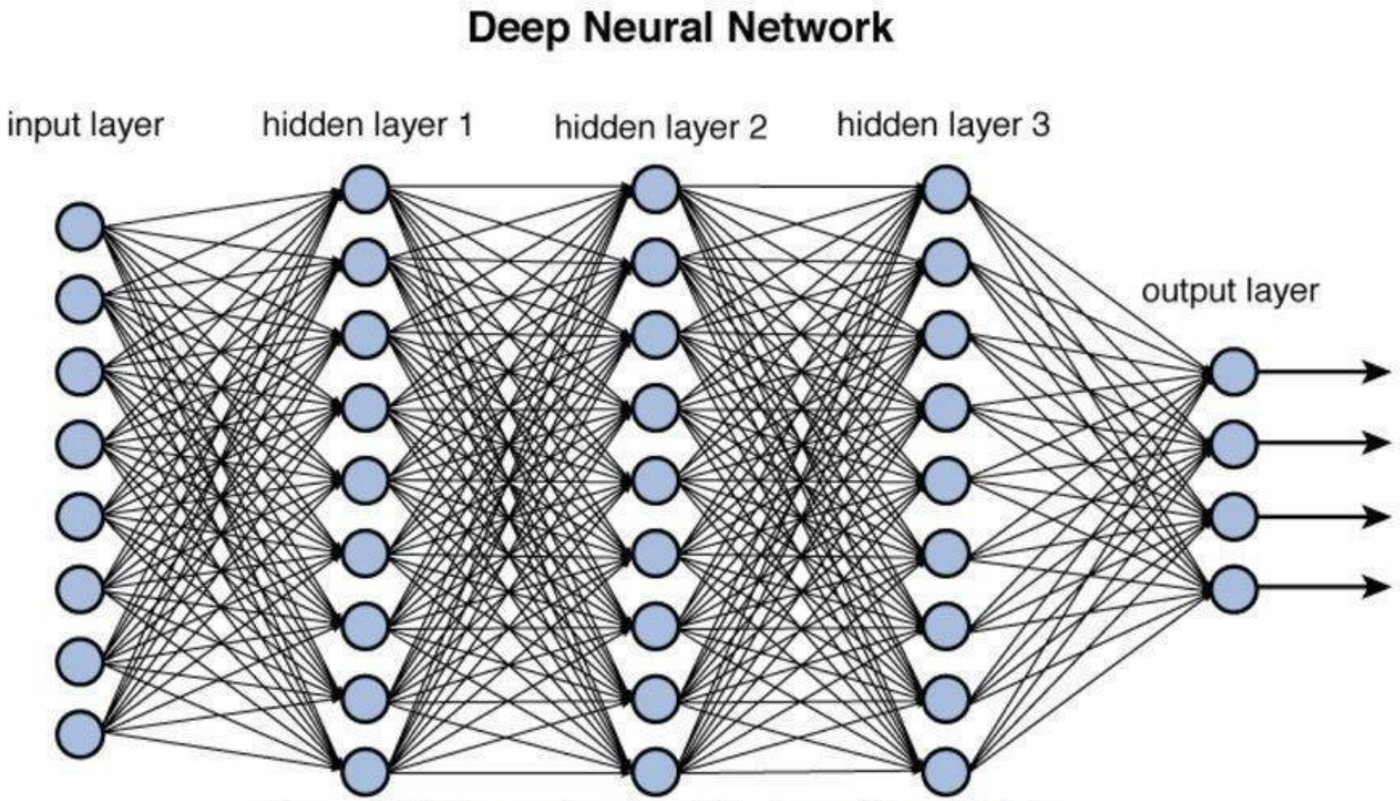
- 1층일 경우는 선형분리만 가능
- 2층은 구역분리 가능
- 3층은 더 세분화된 분리 가능

Layer 의 개수에 따라 결정할 수 있는 영역



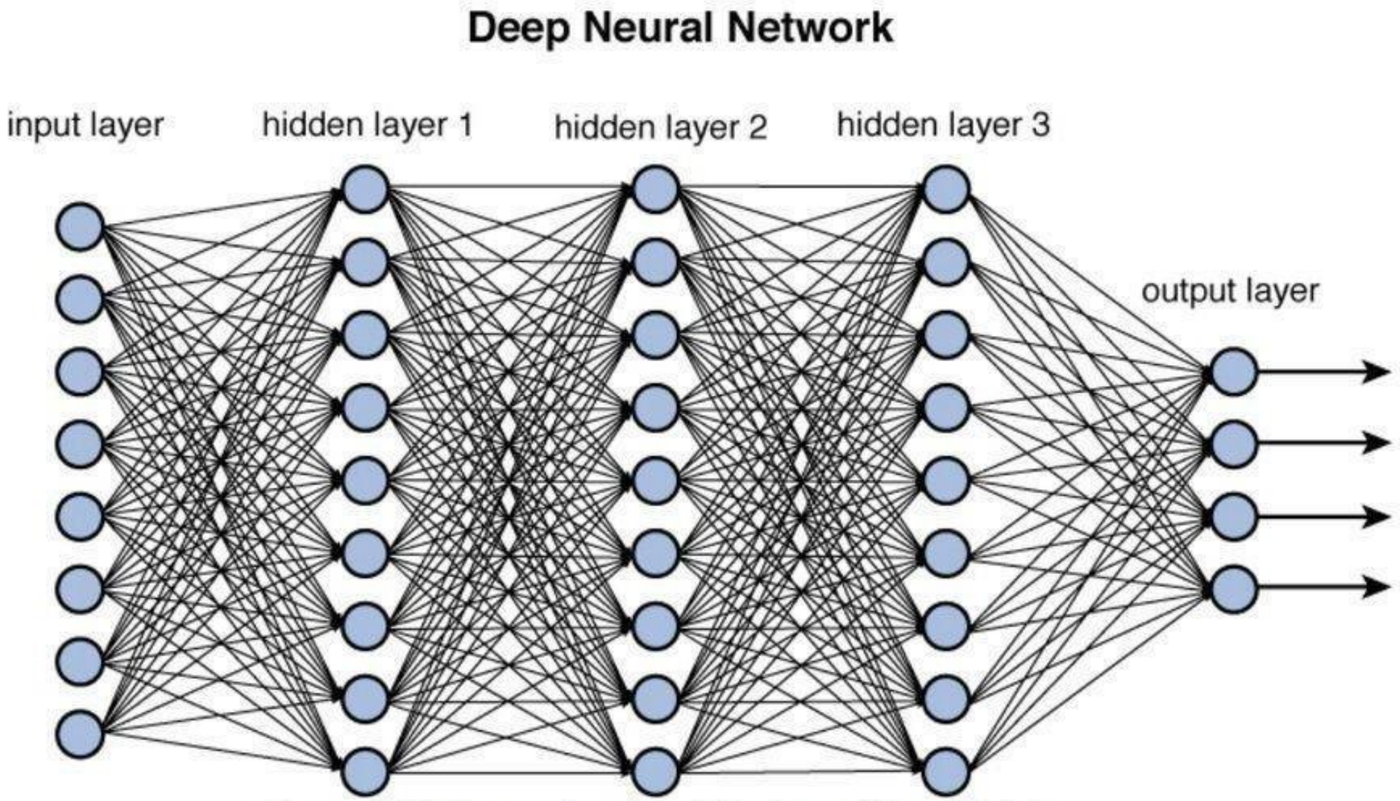
망의 깊이가 깊어진다면?

- 층수가 깊어지고 한 층의 노드 수가 많아진다면 Parameter 수가 증가함
- Parameter: 가중치 (weight), 바이어스 (bias)



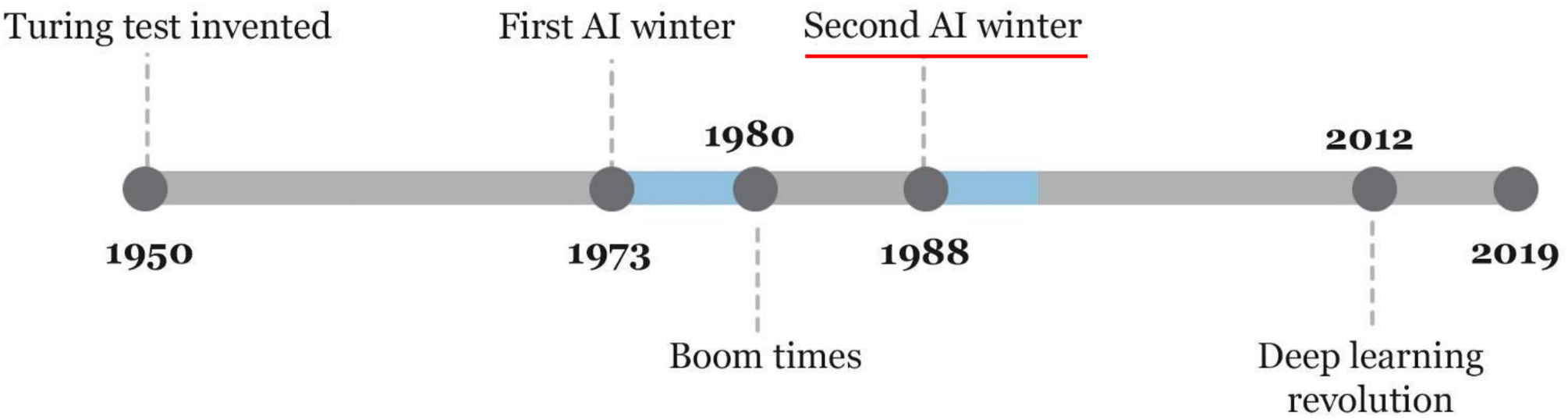
Vanishing Gradient

- 더 깊고 더 넓은 망을 학습시키는 과정에서 Output 값과 멀어질수록 학습이 잘 안되는 현상



2차 AI 겨울

- 층의 깊이가 깊어지는 만큼 계산량이 증가함
- 깊이가 깊어지는 만큼 증가하는 변수는 학습의 수렴 효과 감소
  - Vanishing gradient 문제

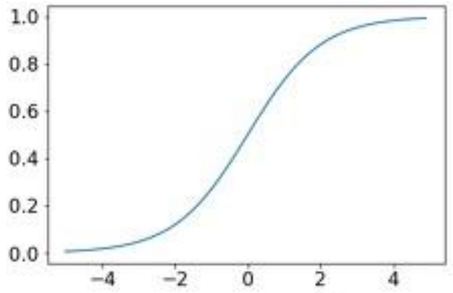


Vanishing Gradient 해결

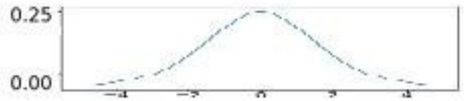
- 2006년, 힌튼이 Sigmoid 함수 대신 ReLU(Rectified Linear Unit)을 사용할 것을 제안
- 내부 hidden layer에는 ReLU를 적용하고, output layer에서만 Tanh 함수를 적용하면 Vanishing Gradient 문제가 해결됨

Vanishing Gradient 해결

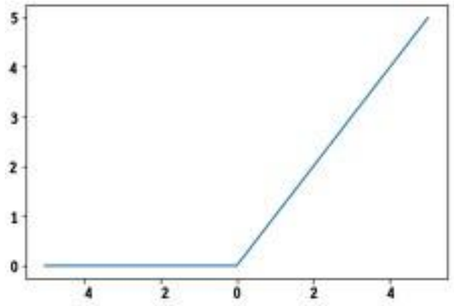
- Sigmoid
  - 도함수의 최대값이 0.25
  - 망이 깊어질 수록 Gradient가 1/4 씩 줄어듦
- ReLU
  - 도함수 값이 0이나 1이기 때문에 컴퓨팅 측면에서 경제적
  - 도함수가 1이기 때문에 망이 깊어져도 Gradient가 줄어들지 않음



$$Sigmoid(x) = \frac{1}{1+e^{-x}}$$

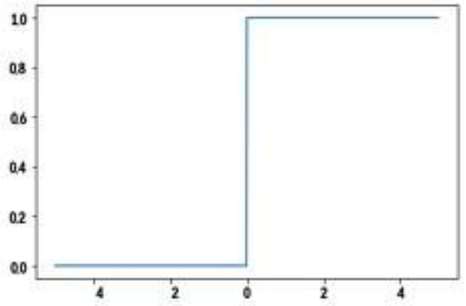


$$S'(x) = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right)$$



$$ReLU(x) = \max(0,\ x)$$

Rectified Linear Unit



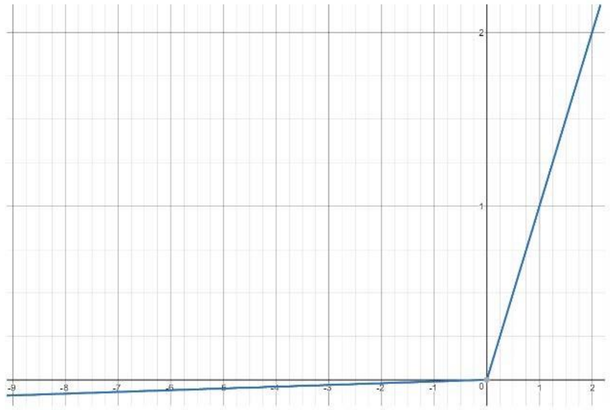
$$R'(y) = \begin{cases} 1 & (x \geq 0) \\ 0 & (x < 0) \end{cases}$$

:e */

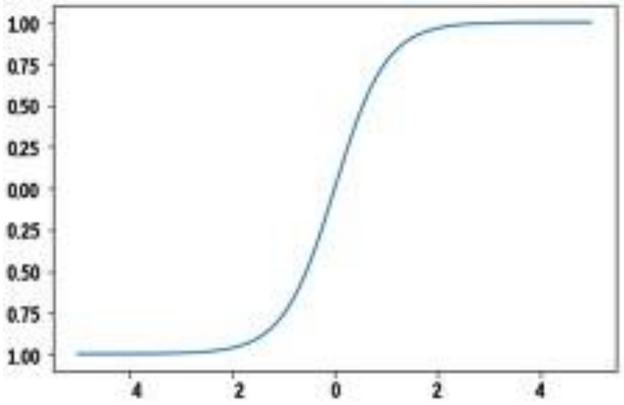
Vanishing Gradient 해결

- Leaky ReLU
  - 0보다 작은 경우 ReLU에서 신경이 죽어버리는 현상을 극복

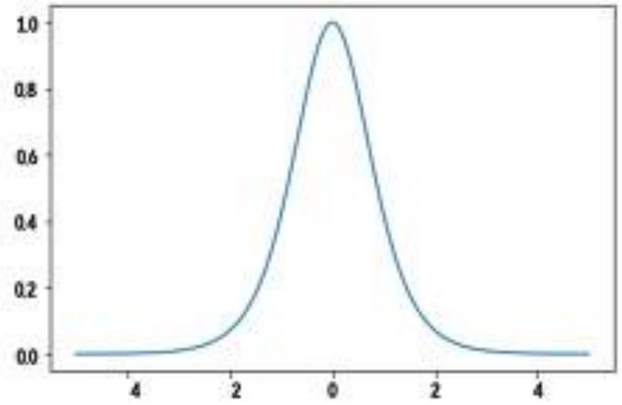
- Tanh
  - 함수값의 범위가 (-1, 1), 도함수의 최댓값이 1



Leaky ReLU



$$Tanh(x) = \frac{e^{2x}+1}{e^{2x}-1}$$

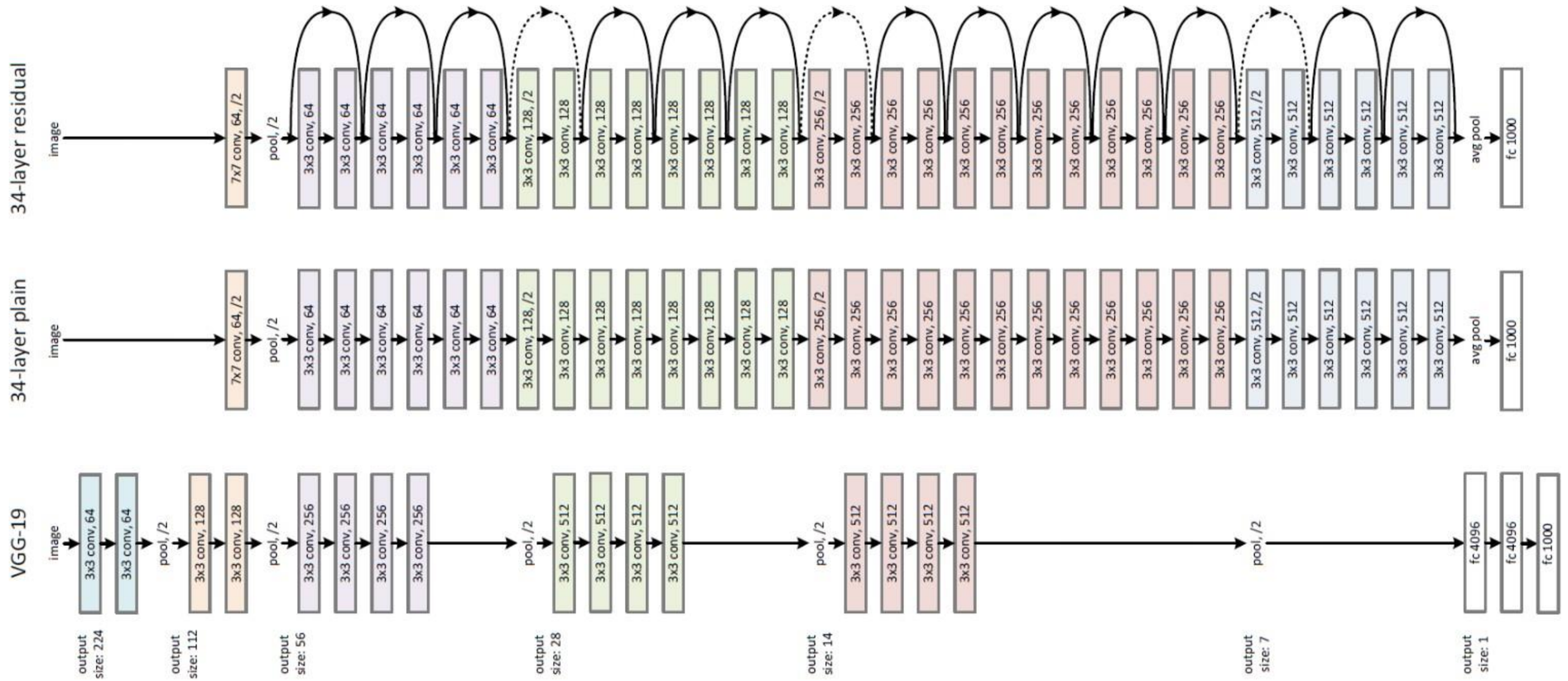


$$T'(x) = 1 - \tanh^2(x)$$

/* elice */

해결 이후

- 망의 깊이가 152 이상의 신경망의 학습 가능



/* elice */

Credit

/* elice */

코스 매니저

콘텐츠 제작자
정민수

강사
정민수

감수자

디자인

Contact

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io