AI(Artificial Intelligence)
Machine Learning & Deep Learning

Deep Learning

미지의 일을 예측하는 힘

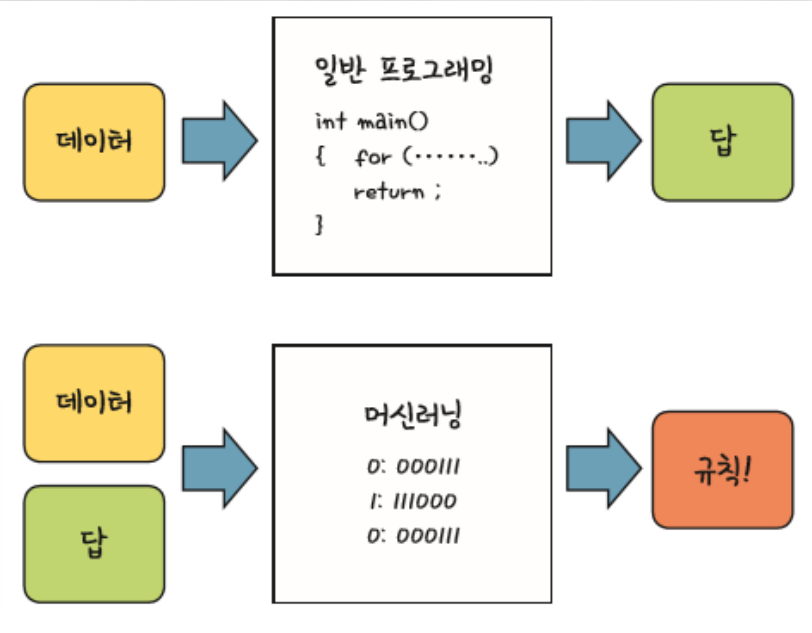
미지의 일을 예측하는 힘

"기존 환자의 데이터를 이용해 새로운 환자의 생사를 예측하는 프로그램을 짜봐!"

미지의 일을 예측하는 힘

미지의 일을 예측하는 힘

"기존 환자의 데이터를 이용해 새로운 환자의 생사를 예측하는 프로그램을 짜봐!" **→ 머신 러닝을 이용해야 함!**



머신러닝과 일반 프로그래밍 비교

- 기존의 프로그래밍은 데이터를 입력해서 답을 구한다.
- 머신러닝은 데이터와 답을 입력해서 규칙을 찾는다. 이 규칙을 다른 데이터에도 적용! (예측)

→ 미지의 일을 예측하기 위해 서는 기존 프로그래밍 방식이 아니라 머신 러닝 기법을 써야함

미지의 일을 예측하는 힘

미지의 일을 예측하는 힘

- 중환자를 전문으로 수술하는 어느 병원 외과 의사의 질문

"혹시 수술하기 전에 수술 후의 생존율을 수치로 예측할 방법이 없을까?"

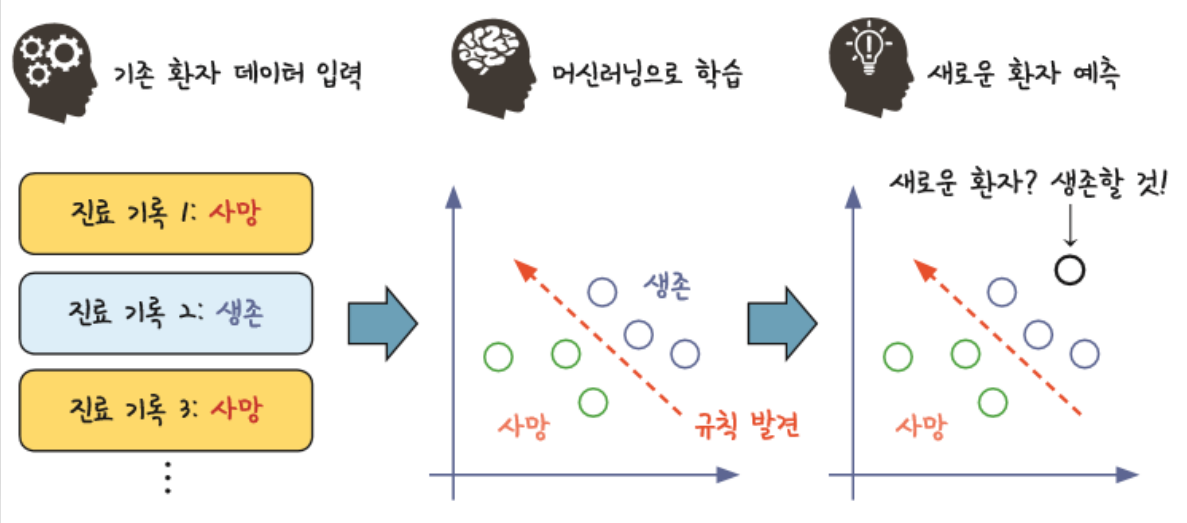
→ 방법은 자신이 그동안 집도한 수술 환자의 수술 전 상태와 수술 후의 생존율을 정리해 놓은 데이터를 머신러닝 알고리즘에 넣는 것

- 기존 환자의 데이터는 머신러닝에 입력되는 순간, 그 패턴과 규칙이 분석됨
  → 이를 학습(training) 이라고 함
- 분석 결과를 새로운 환자의 데이터와 비교해 생존 가능성이 몇 퍼센트인지 알려 줌

  이것이 바로 머신러닝이 하는 일

미지의 일을 예측하는 힘

미지의 일을 예측하는 힘



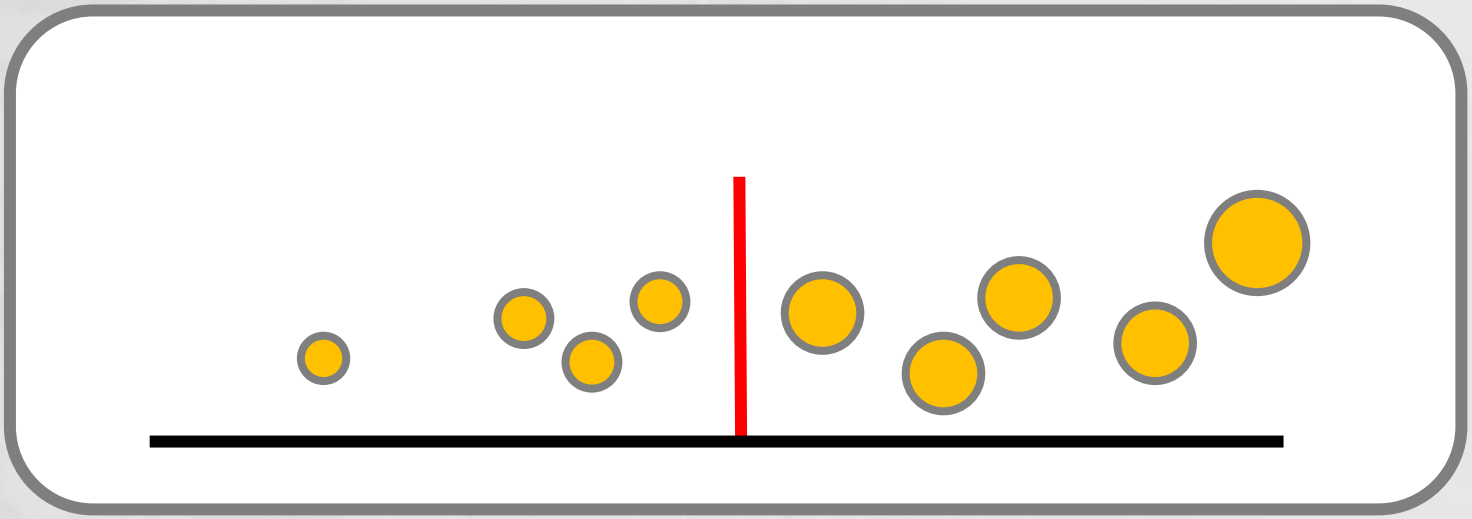
머신러닝의 학습 및 예측 과정

- **학습** : 깨끗한 좌표 평면에 기존 환자들을 하나씩 배치하는 과정
  - → 환자들의 분포를 그래프 위에 펼쳐 놓고
  - → 이 분포도 위에 생존과 사망 여부를 구분짓는 경계를 그려넣음.
  - → 이를 잘 저장해 놓았다가 새로운 환자가 오면 분포도의 어디쯤 위치하는지를 파악
  - → **머신러닝의 예측 성공률은 결국 얼마나 정확한 경계선을 긋느냐에 달려 있다!**

Deep Learning

Classification

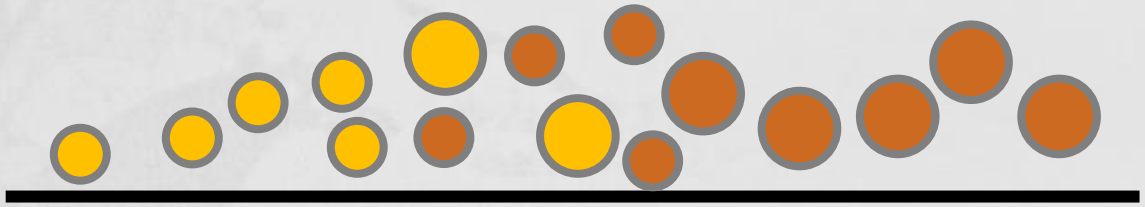
- 분류의 문제 = 선긋기(기준 찾기)
- 과일 분류하기(수확 가능여부 확인)
- 고정 기준 = if (크기 > 9) then 수확 else 미수확
- 기준 찾기 = Learning(or Training Data Set iteration)



Deep Learning

Classification

- 크기를 기준으로 생성한 모델을 Test Data Set에 적용
- Training Data Set에서 높은 정확도를 보여도 Test Data Set의 결과는 부적합
- 색의 진하기를 분류 기준으로 추가

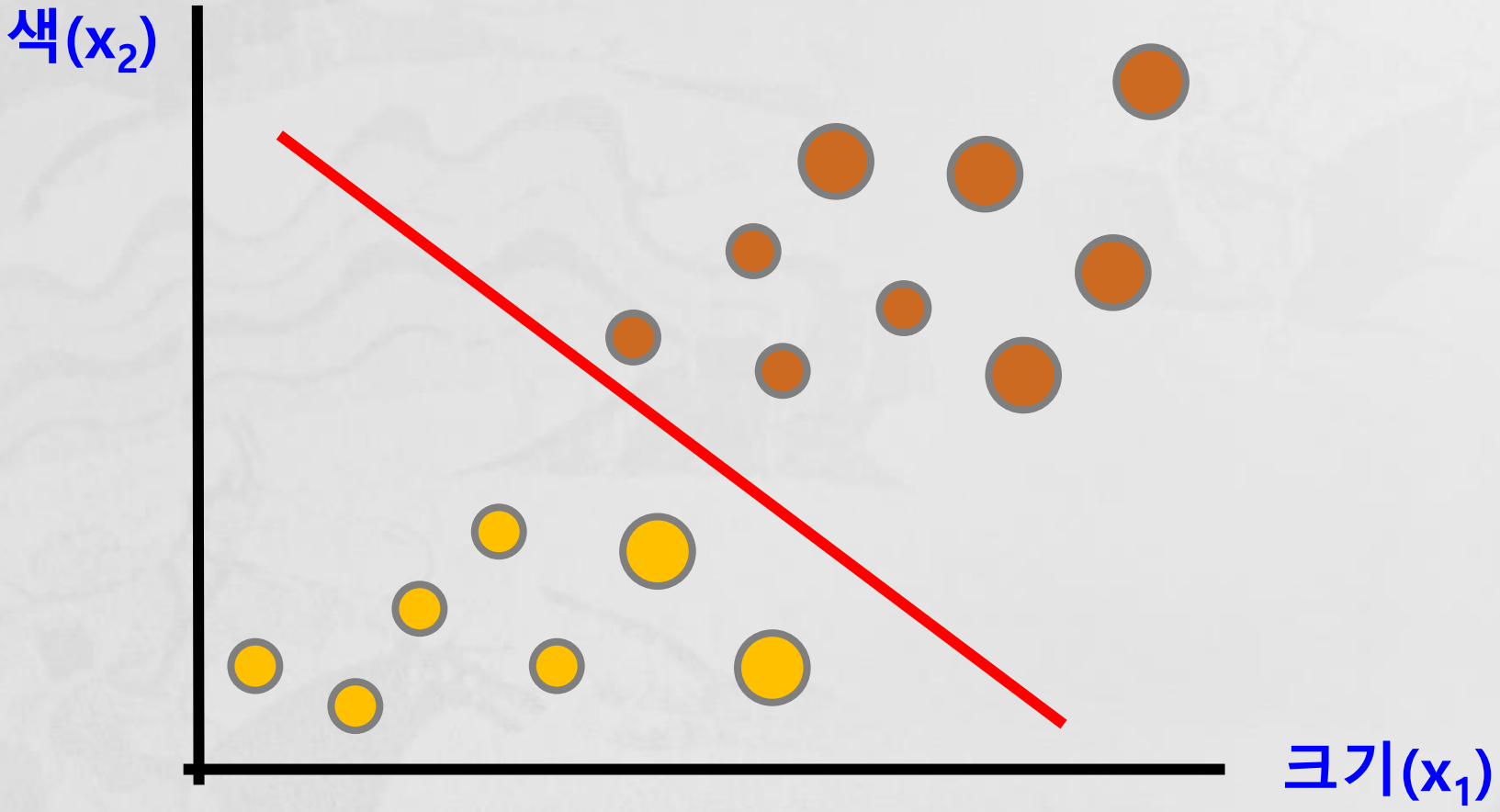


* 과적합 문제 발생

Deep Learning

Classification

- 2차원으로 표현

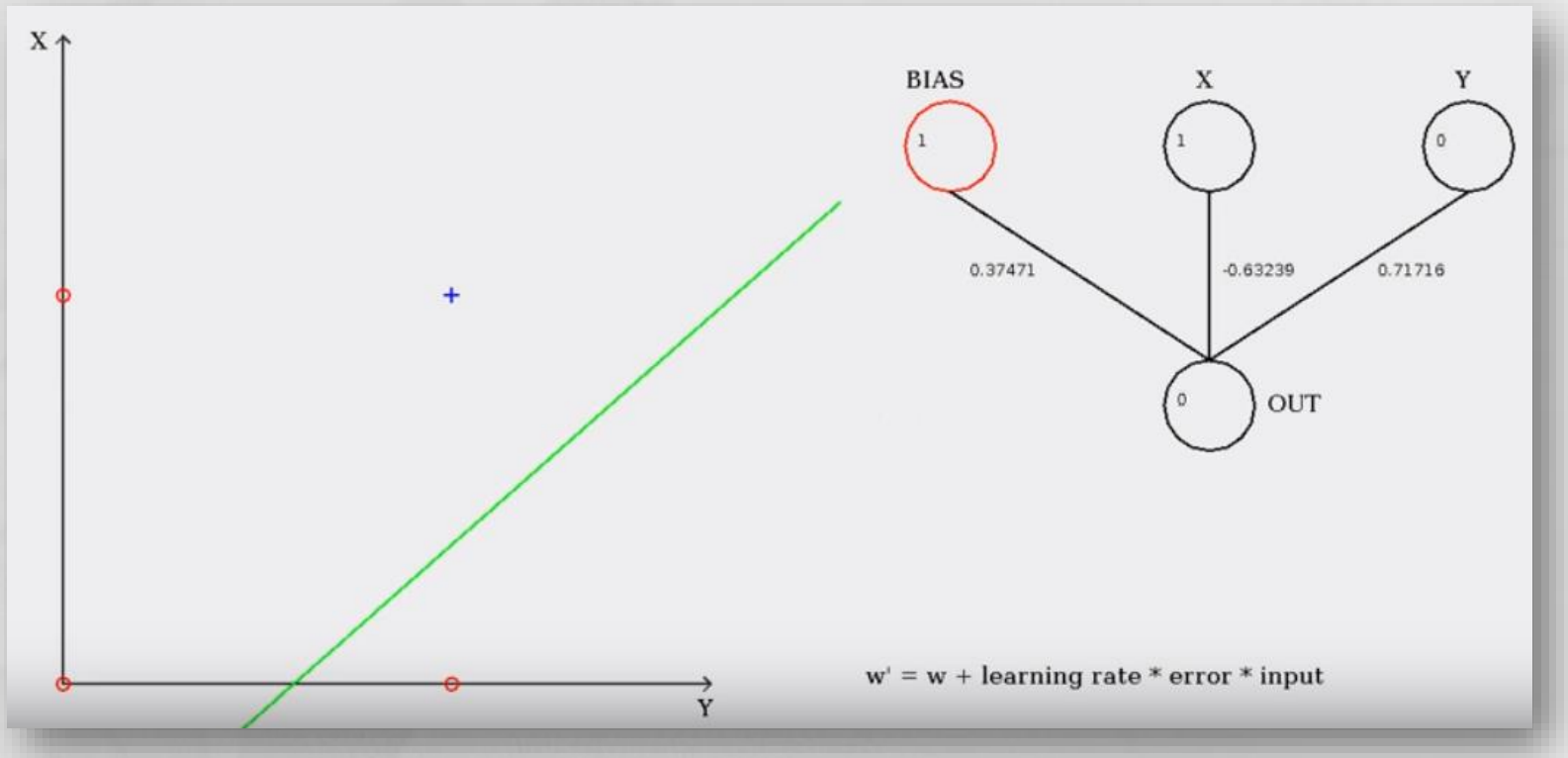


* $w_1x_1 + w_2x_2 = b$(기준 찾기 = w_1, w_2, b 찾기)

Deep Learning

Classification

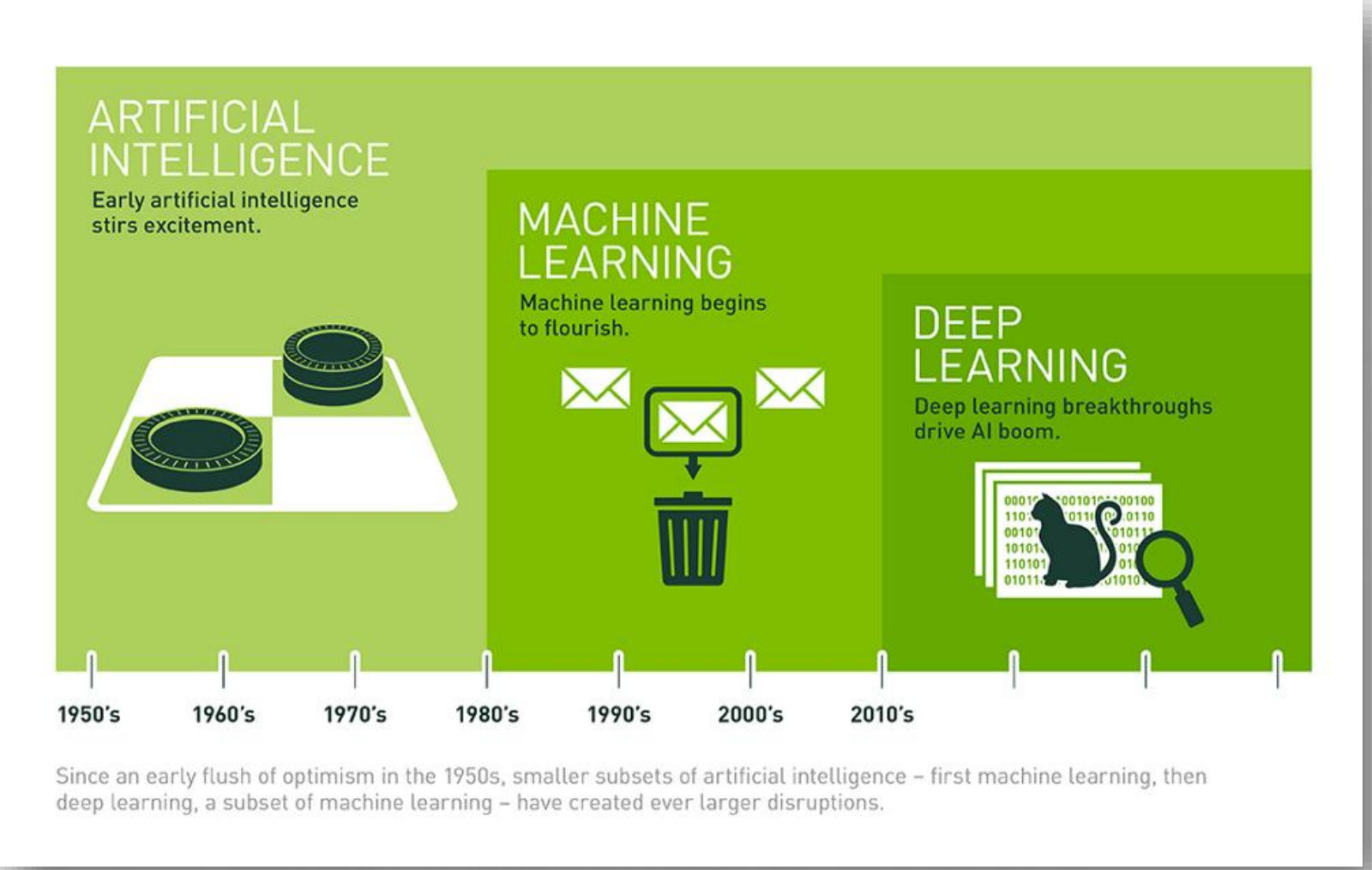
- Learning Algorithm
- Ex) https://www.youtube.com/watch?v=tYxkIOTdeu8



* y = wx + b

Deep Learning

Definition



* https://towardsdatascience.com/deep-learning-weekly-piece-the-differences-between-ai-ml-and-dl-b6a203b70698

Deep Learning

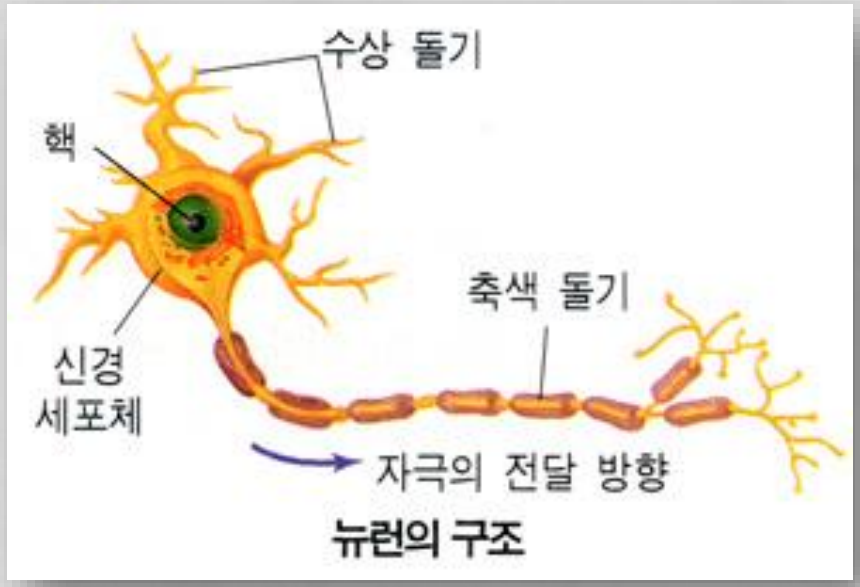
Neural Network

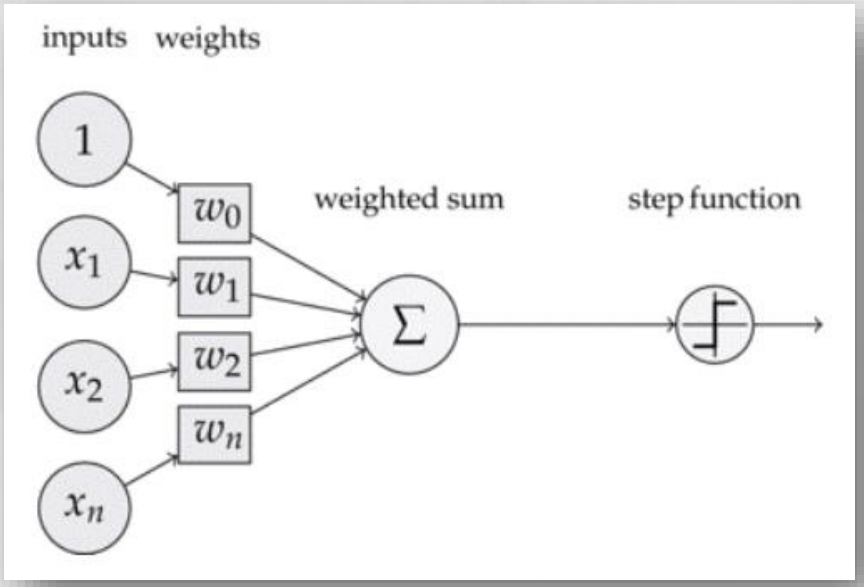
- 인공 신경망
- 기계 학습 분야에서 연구되는 학습 알고리즘
- 시계열 자료의 예측, 분류, 패턴 인식, 제어 분야에 응용
- 인간의 뇌 구조를 모방하여 만들어짐(신경세포:Neuron)
- 수상돌기(Dendrite), 신경세포체(Soma), 축색(Axon), 시냅스(Synapse)
- 수상돌기 = 입력, 축색 = 출력
- 신경세포체 = 노드, 시냅스 = 가중값을 갖는 연결 네트워크

Deep Learning

Neural Network

- 생물학적 신경망 vs. 인공 신경망



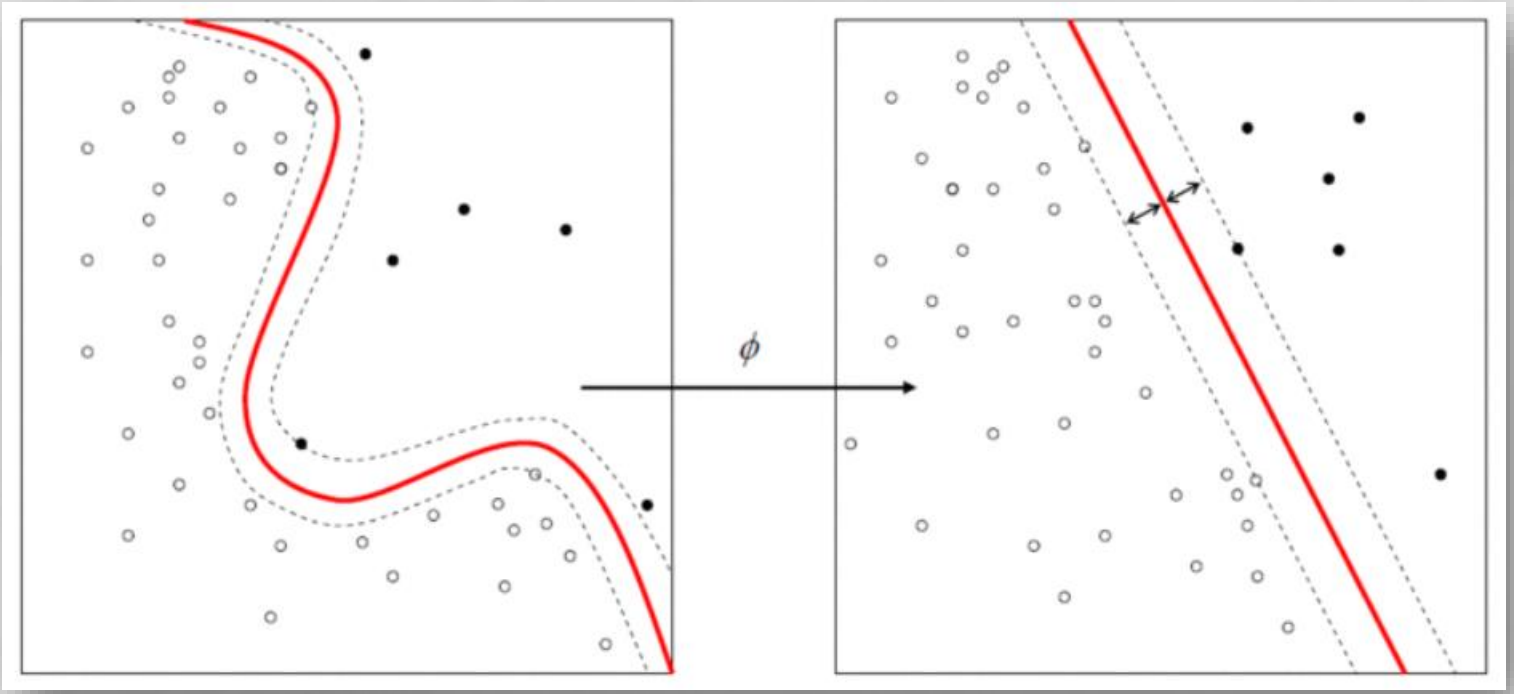


* 퍼셉트론(Perceptron) : 인공 신경망의 한 종류(선형 분리기)

Deep Learning

Multi Layer Perceptron(MLP)

- 다층 퍼셉트론(or Deep Neural Network)
- 퍼셉트론으로 해결할 수 없는 비선형 분리 문제에 필요

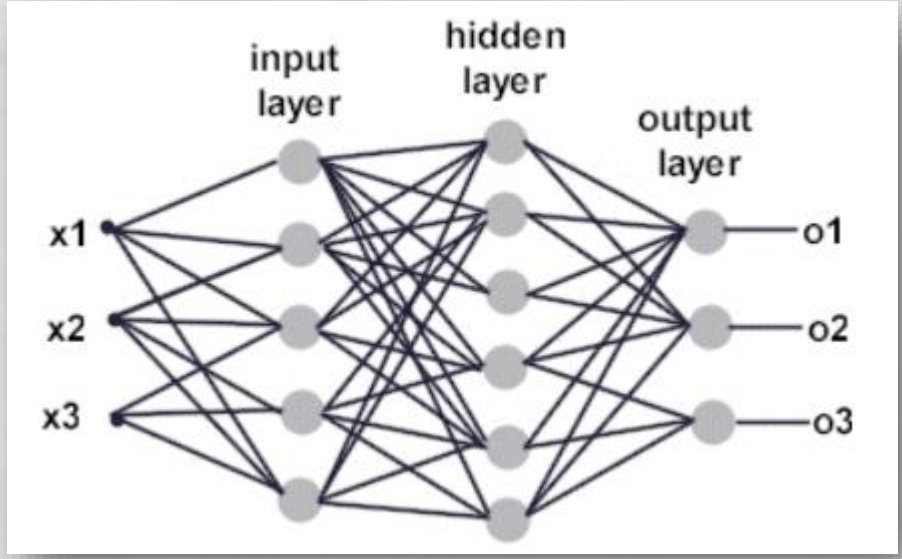


* https://en.wikipedia.org/wiki/Multilayer_perceptron

Deep Learning

Multi Layer Perceptron(MLP)

- 다층 퍼셉트론(or Deep Neural Network)
- 여러 층의 퍼셉트론을 쌓아서 동작
- 직선을 여러 번 그어서 비선형 분리 문제를 해결

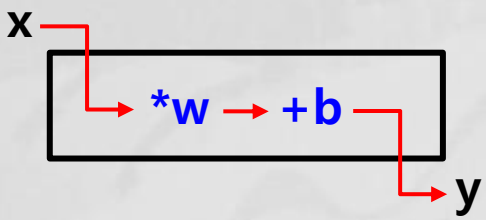


* https://en.wikipedia.org/wiki/Multilayer_perceptron

Deep Learning

Multi Layer Perceptron(MLP)

- Linear Model

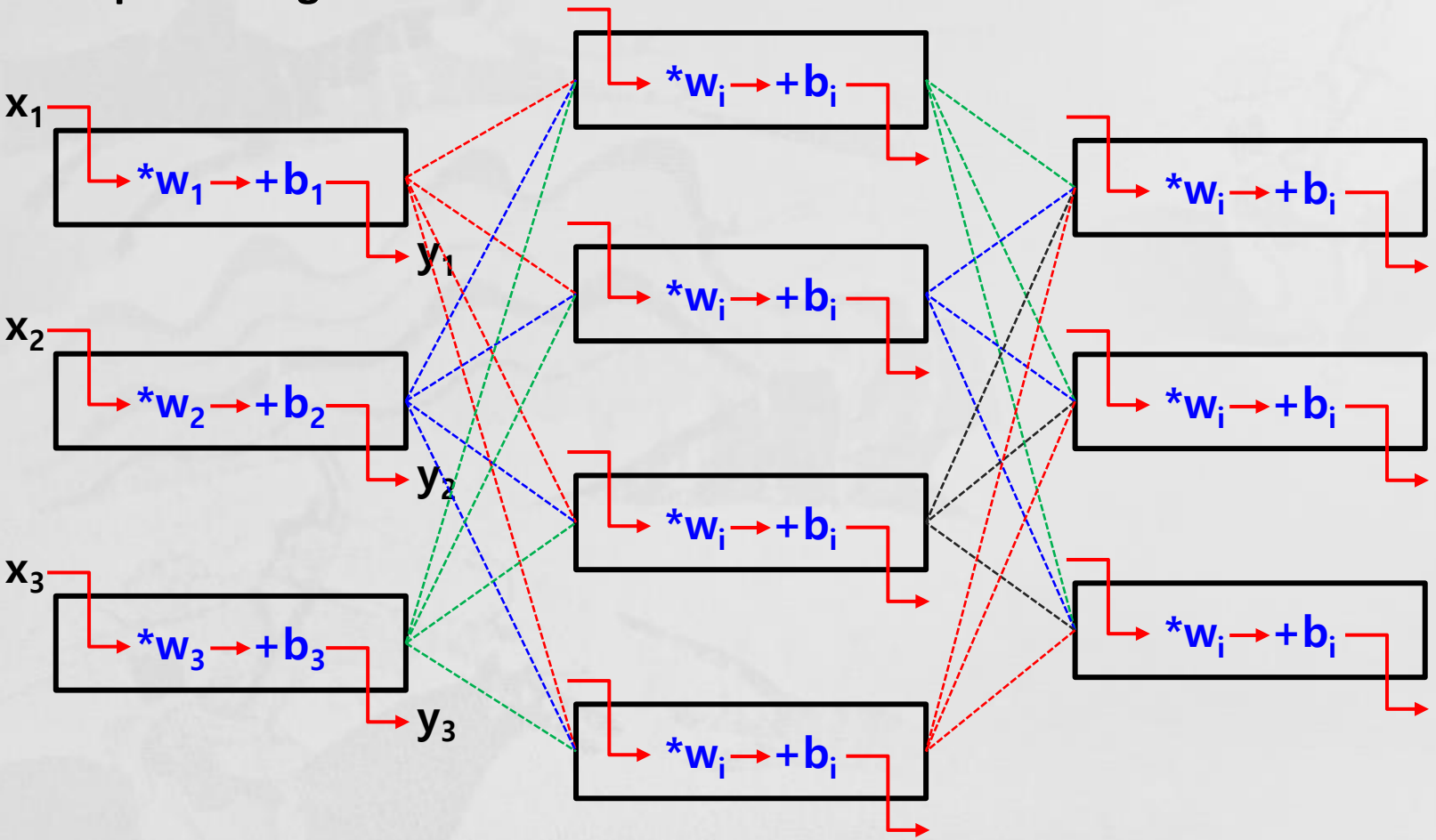


* y = wx + b

Deep Learning

Multi Layer Perceptron(MLP)

- Deep Learning

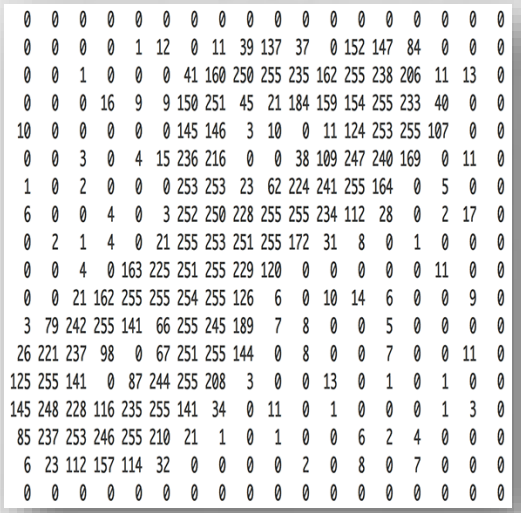


* y = wx + b

Deep Learning

Artificial Neural Network(ANN)

- Handwritten digits in the M(Mixed)-NIST database



0	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	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	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	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	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	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	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

* https://algobeans.com/2016/03/13/how-do-computers-recognise-handwriting-using-artificial-neural-networks/

Deep Learning

Artificial Neural Network(ANN)

- 1000개의 이미지 테스크 결과(92.2% 정확도)

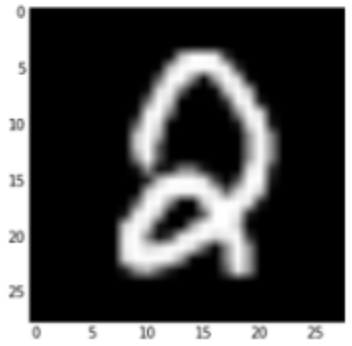
		Predicted Digit											
		0	1	2	3	4	5	6	7	8	9	Total	%
Actual Digit	0	84	0	0	0	0	0	1	0	0	0	85	99
	1	0	125	0	0	0	0	1	0	0	0	126	99
	2	1	0	105	0	0	0	0	4	5	1	116	91
	3	0	0	3	96	0	6	0	1	0	1	107	90
	4	0	0	2	0	99	0	2	0	2	5	110	90
	5	2	0	0	5	0	77	1	0	1	1	87	89
	6	3	0	1	0	1	2	80	0	0	0	87	92
	7	0	3	3	0	1	0	0	90	0	2	99	91
	8	1	0	1	3	1	0	0	2	81	0	89	91
	9	0	0	0	0	1	0	0	6	2	85	94	90
	Total	91	128	115	104	103	85	85	103	91	95	1000	-

* https://algobeans.com/2016/03/13/how-do-computers-recognise-handwriting-using-artificial-neural-networks/

Deep Learning

Artificial Neural Network(ANN)

- 1000개의 이미지 테스크 결과(92.2% 정확도)

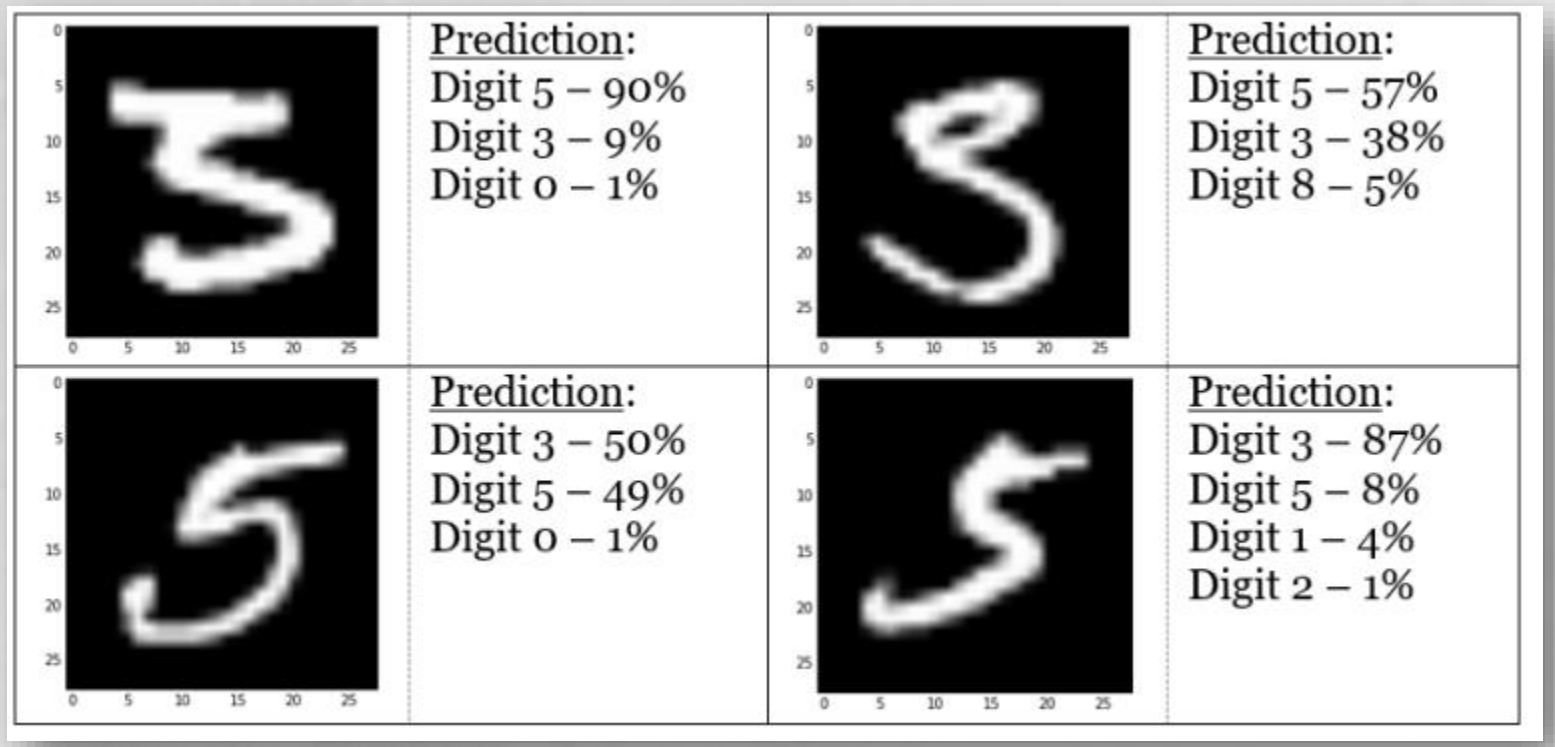
	Prediction: Digit 7 – 99% Digit 3 – 1%
	Prediction: Digit 7 – 94% Digit 2 – 5% Digit 3 – 1%
	Prediction: Digit 8 – 48% Digit 2 – 47% Digit 3 – 4% Digit 1 – 1%
	Prediction: Digit 8 – 58% Digit 2 – 27% Digit 6 – 12% Digit 0 – 2% Digit 5 – 1%

* https://algobeans.com/2016/03/13/how-do-computers-recognise-handwriting-using-artificial-neural-networks/

Deep Learning

Artificial Neural Network(ANN)

- 1000개의 이미지 테스크 결과(92.2% 정확도)



* https://algobeans.com/2016/03/13/how-do-computers-recognise-handwriting-using-artificial-neural-networks/

Deep Learning

Wrap Up

- 데이터 저장 및 공유 기술의 발달로 신경망 학습에 필요한 데이터 제공 증가
- GPU(Graphics Processing Unit)를 사용하여 컴퓨팅 파워 증가
- 개선된 알고리즘
- 인공 신경망은 샘플 크기가 커야 함(많은 학습 데이터 필요)
- 서로 다른 활성화 규칙에 따라 여러 개의 레이어로 구성되어 해석이 어려움
- 역전파(Backpropagation) : 예측 정확도에 따라 활성화 규칙을 정제
- 인공 신경망은 데이터가 크고 고성능 컴퓨팅 파워를 사용 가능할 때 적합

* 활성화 규칙 : 뉴런의 활성화 여부는 입력 활성화의 강도와 출력으로 결정

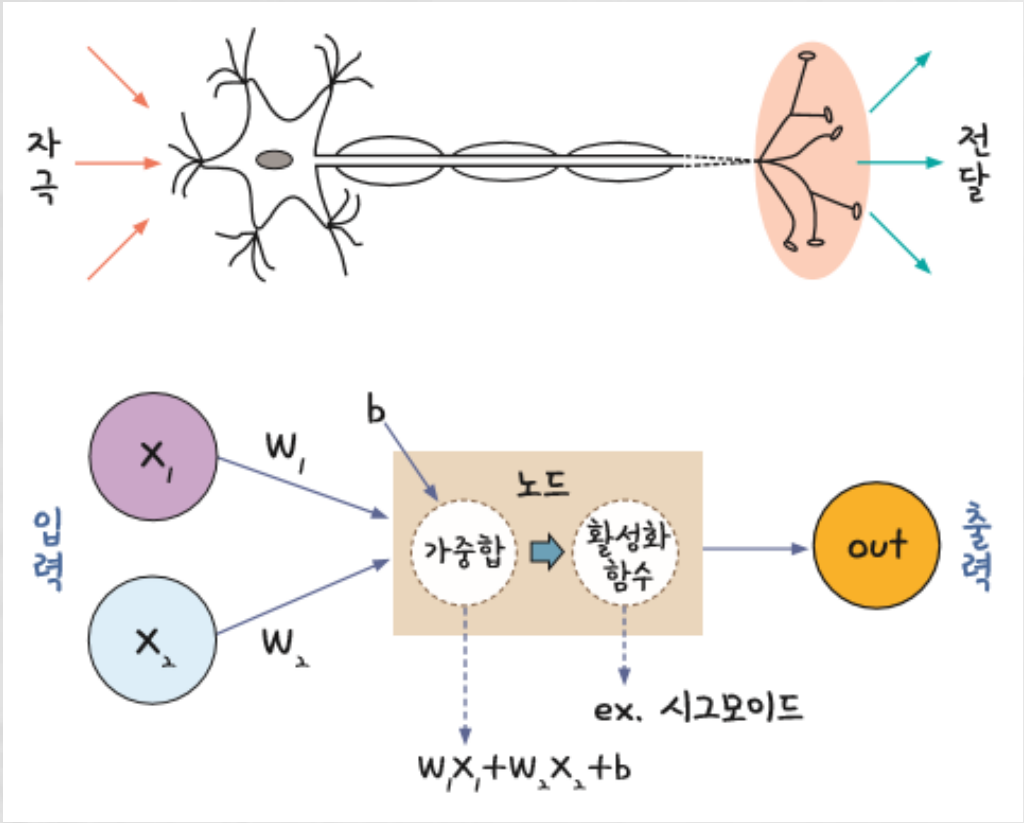
신경망으로 딥러닝

퍼셉트론

- 인간의 뇌는 치밀하게 연결된 약 1000억 개의 뉴런으로 이루어져 있음
- 뉴런과 뉴런 사이에는 시냅스라는 연결 부위가 있는데, 신경 말단에서 자극을 받으면 시냅스에서 화학 물질이 나와 전위 변화를 일으킴
- 전위가 임계 값을 넘으면 다음 뉴런으로 신호를 전달하고, 임계 값에 미치지 못하면 아무것도 하지 않음 → **퍼셉트론의 개념과 유사!**

신경망으로 딥러닝

퍼셉트론



뉴런과 퍼셉트론의 비교

- 신경망을 이루는 가장 중요한 기본 단위는 **퍼셉트론**(perceptron)
- 퍼셉트론은 입력 값과 활성화 함수를 사용해 출력 값을 다음으로 넘기는 가장 작은 신경망 단위

신경망으로 딥러닝

가중치, 가중합, 바이어스, 활성화 함수

- 기울기 a나 y 절편 b와 같은 용어를 퍼셉트론의 개념에 맞춰 좀 더 '딥러닝 답게' 표현해 보면 다음과 같음

$$y = ax + b \ (a\text{는 기울기}, \ b\text{는 } y \text{ 절편})$$

$$\rightarrow \ y = wx + b \ (w\text{는 가중치}, \ b\text{는 바이어스})$$

- 먼저 기울기 a는 퍼셉트론에서는 **가중치**를 의미하는 w(weight)로 표기됨
- y 절편 b는 똑같이 b라고 씀, 하지만 y = ax + b의 b가 아니라 편향, 선입견이라는 뜻인 **바이어스**(bias)에서 따온 b

신경망으로 딥러닝

가중치, 가중합, 바이어스, 활성화 함수

- **가중합**(weighted sum): 입력 값(x)과 가중치(w)의 곱을 모두 더한 다음 거기에 바이어스(b)를 더한 값

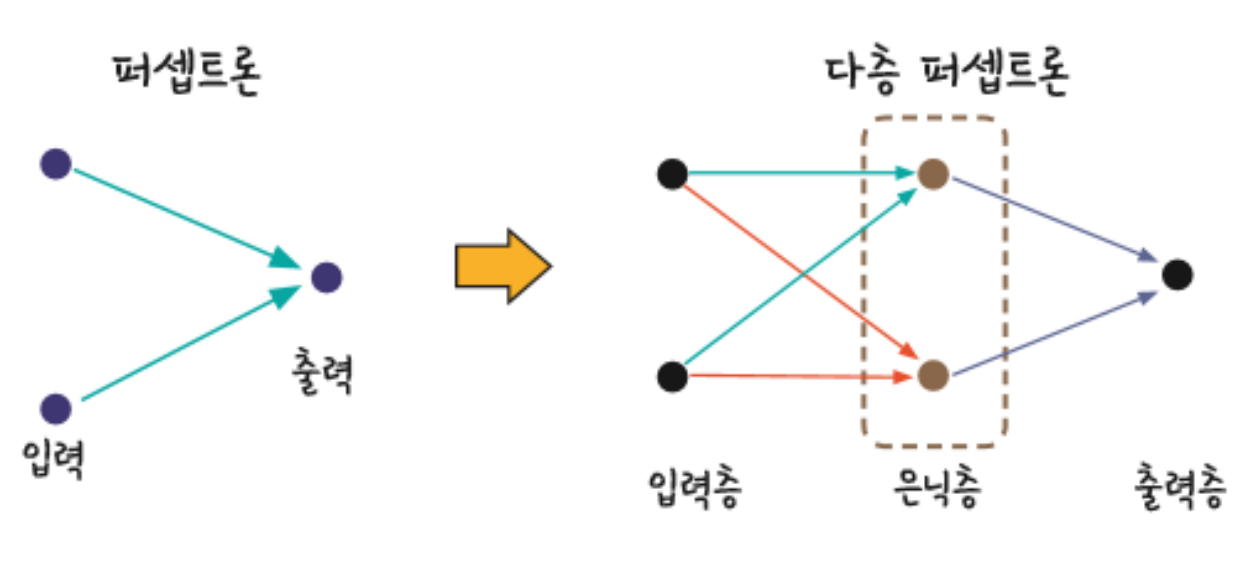
- 가중합의 결과를 놓고 1 또는 0을 출력해서 다음으로 보냄

- 여기서 0과 1을 판단하는 함수가 있는데, 이를 **활성화 함수**(activation function) 라고 함. 앞서 배웠던 시그모이드 함수가 바로 대표적인 활성화 함수

신경망으로 딥러닝

다층 퍼셉트론

- XOR 문제를 해결하기 위해서 두 개의 퍼셉트론을 한 번에 계산할 수 있어야 함
- 이를 가능하게 하려면 숨어있는 층, 즉 은닉층(hidden layer)을 만들면 됨

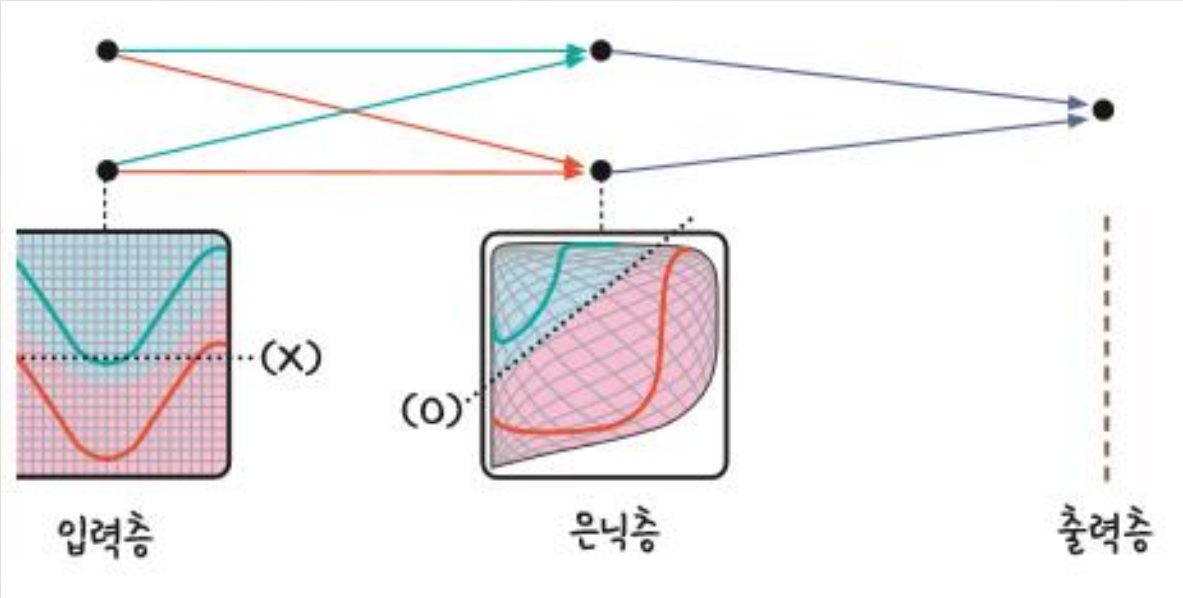


퍼셉트론에서 다층 퍼셉트론으로

신경망으로 딥러닝

다층 퍼셉트론

- 입력층과 은닉층의 그래프를 집어넣어 보면
- 은닉층이 좌표 평면을 왜곡시키는 결과를 가져옴 → 두 영역을 가로지르는 선이 직선으로 바뀜

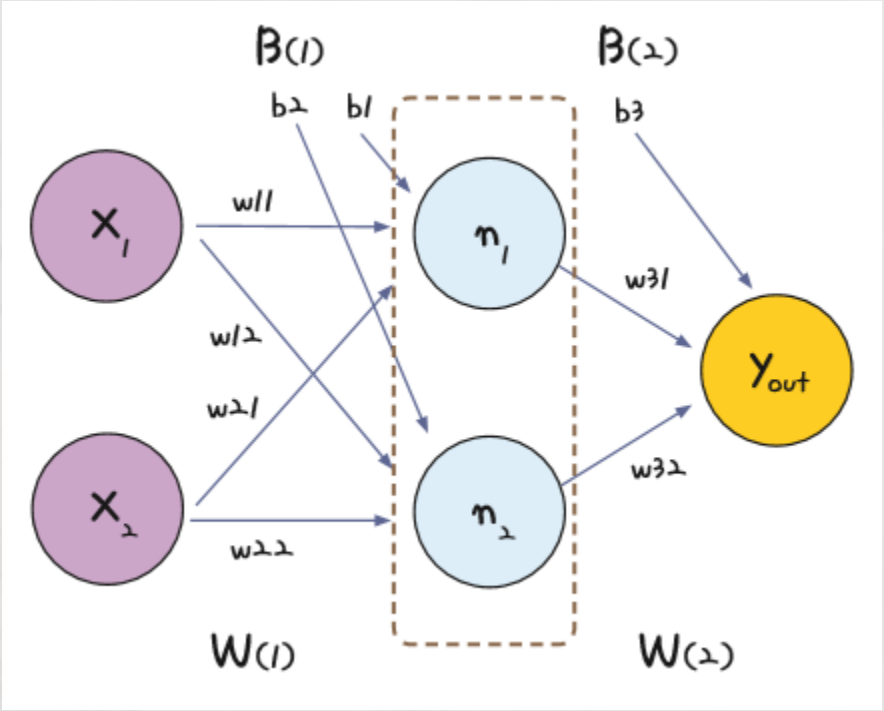


은닉층의 공간 왜곡

신경망으로 딥러닝

다층 퍼셉트론

- 다층 퍼셉트론이 입력층과 출력층 사이에 숨어있는 은닉층을 만드는 것을 도식

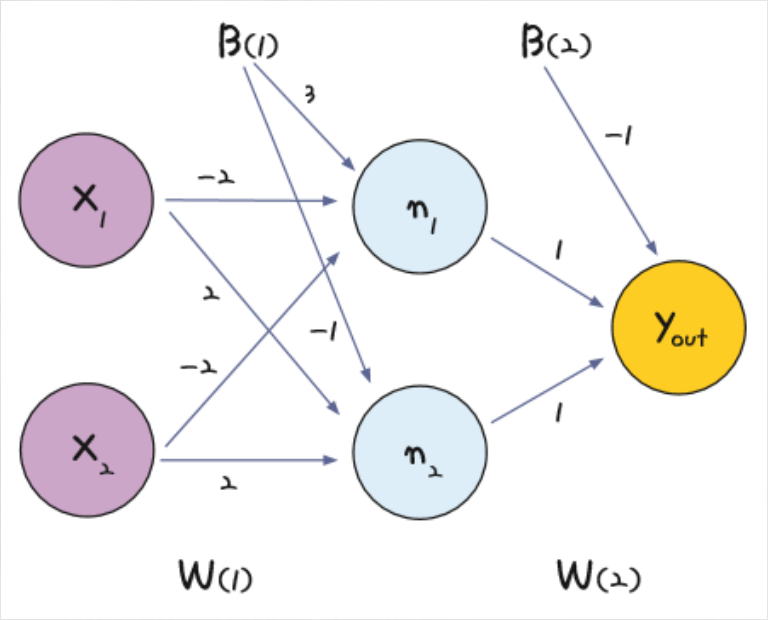


다중 퍼셉트론의 내부

신경망으로 딥러닝

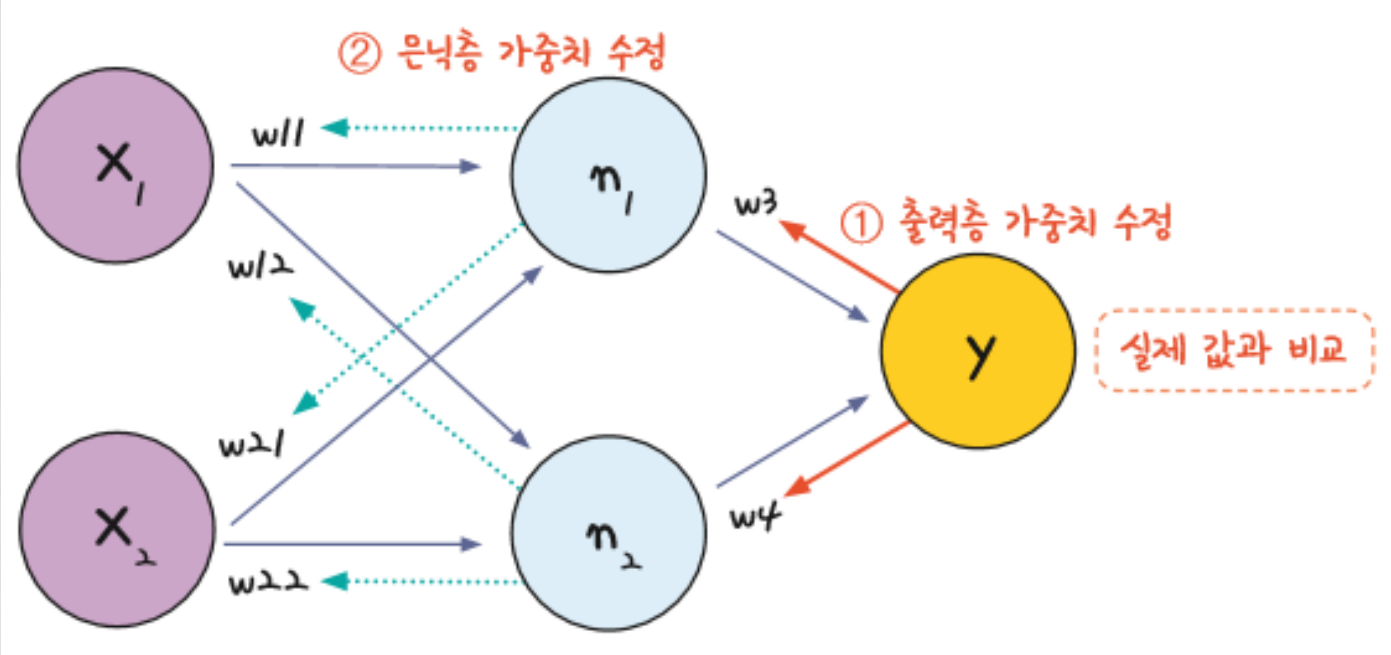
다층 퍼셉트론

- 가운데 숨어있는 은닉층으로 퍼셉트론이 각각 자신의 가중치(w)와 바이어스(b) 값을 보내고, 이 은닉층에서 모인 값이 한 번 더 시그모이드 함수(기호로 σ 라고 표시)를 이용해 최종 값으로 결과를 보냄
- 은닉층에 모이는 중간 정거장을 노드(node)라고 하며, 여기서는 n_1과 n_2로 표현



신경망으로 딥러닝

오차 역전파



다층 퍼셉트론에서의 오차 수정

신경망으로 딥러닝

오차 역전파

- 그러다 보니 최적화의 계산 방향이 출력층에서 시작해 앞(뒤어서 앞으로)으로 진행됨 : **오차 역전파**(back propagation)라고 부름

- 오차 역전파 구동 방식의 정리

1) 임의의 초기 가중치($w_{(1)}$)를 준 뒤 결과(y_{out})를 계산한다.
2) 계산 결과와 우리가 원하는 값 사이의 오차를 구한다.
3) 경사 하강법을 이용해 바로 앞 가중치를 오차가 작아지는 방향으로 업데이트한다.
4) 1~3 과정을 더이상 오차가 줄어들지 않을 때까지 반복한다.

신경망으로 딥러닝

오차 역전파

- 여기서 '오차가 작아지는 방향으로 업데이트한다'는 의미는 미분 값이 0에 가까워지는 방향으로 나아간다는 말

- 즉, '기울기가 0이 되는 방향'으로 나아가야 하는데, 이 말은 가중치에서 기울기를 뺐을 때 가중치의 변화가 전혀 없는 상태를 말함

- 따라서 오차 역전파를 다른 방식으로 표현하면 가중치에서 기울기를 빼도 값의 변화가 없을 때까지 계속해서 가중치 수정 작업을 반복하는 것

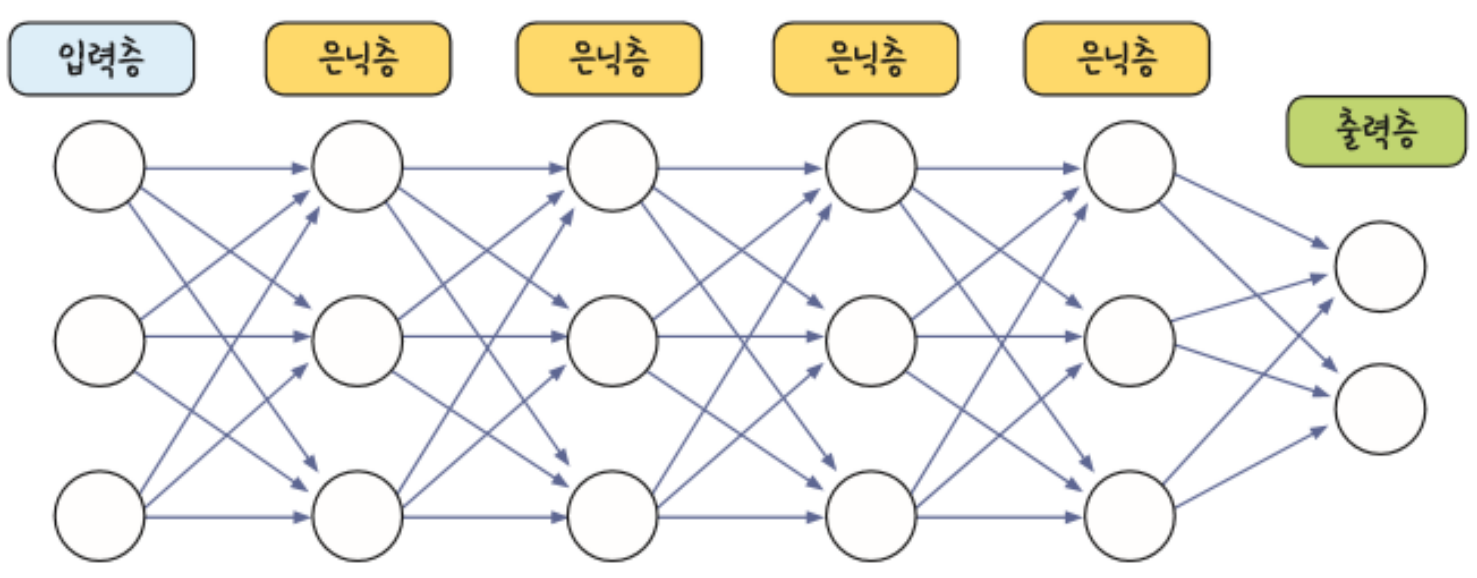
새 가중치는 현 가중치에서 '가중치에 대한 기울기'를 뺀 값!

$$W(t+1) = Wt - \frac{\partial 오차}{\partial W}$$

신경망으로 딥러닝

신경망으로 딥러닝

- 다층 퍼셉트론이 오차 역전파를 만나 신경망이 되었고, 신경망은 XOR 문제를 가볍게 해결
- 하지만 기대만큼 결과가 좋아지지 않았음
- 이유가 무엇일까?



다층 확장

신경망으로 딥러닝

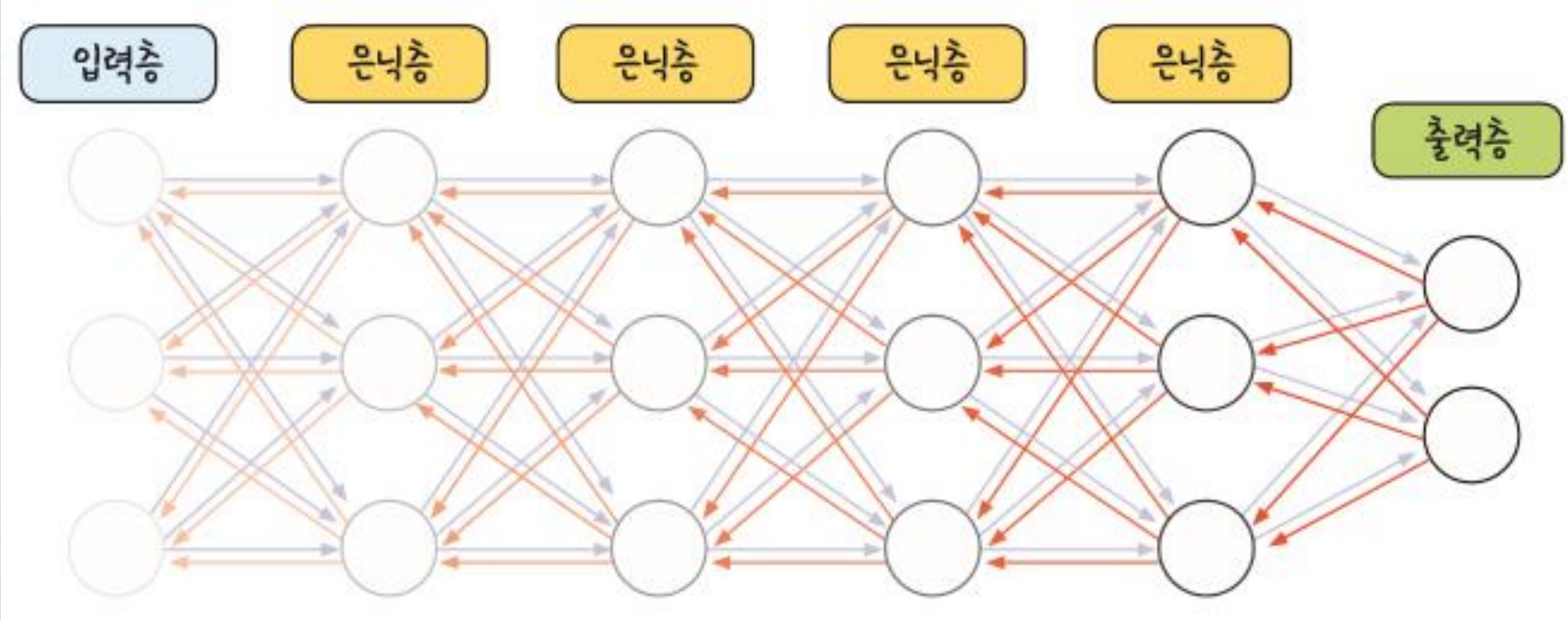
기울기 소실 문제와 활성화 함수

- 기울기 소실 문제!

- 오차 역전파는 출력층으로부터 하나씩 앞으로 되돌아가며 각 층의 가중치를 수정하는 방법
- 가중치를 수정하려면 미분 값, 즉 기울기가 필요하다고 배움
- 그런데 층이 늘어나면서 기울기가 중간에 0이 되어버리는 **기울기 소실** (vanishing gradient) 문제가 발생하기 시작

신경망으로 딥러닝

기울기 소실 문제와 활성화 함수

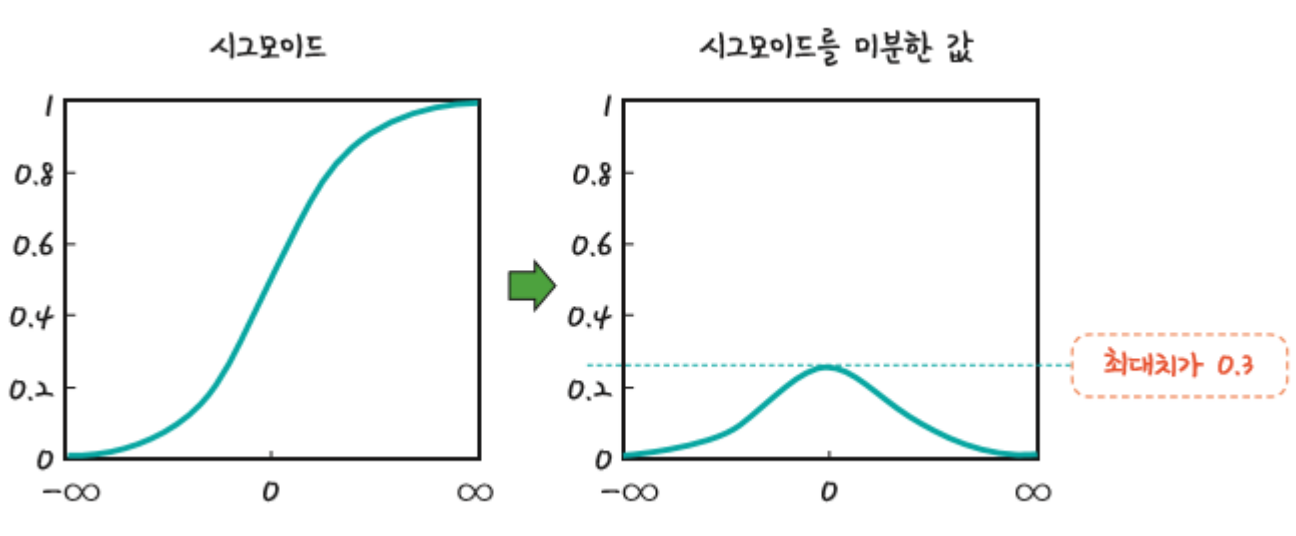


기울기 소실 문제 발생

신경망으로 딥러닝

기울기 소실 문제와 활성화 함수

- 이는 활성화 함수로 사용된 시그모이드 함수의 특성 때문임
- 시그모이드를 미분하면 최대치가 0.3
- 1보다 작으므로 계속 곱하다 보면 0에 가까워짐
- 따라서 층을 거쳐 갈수록 기울기가 사라져 가중치를 수정하기가 어려워지는 것

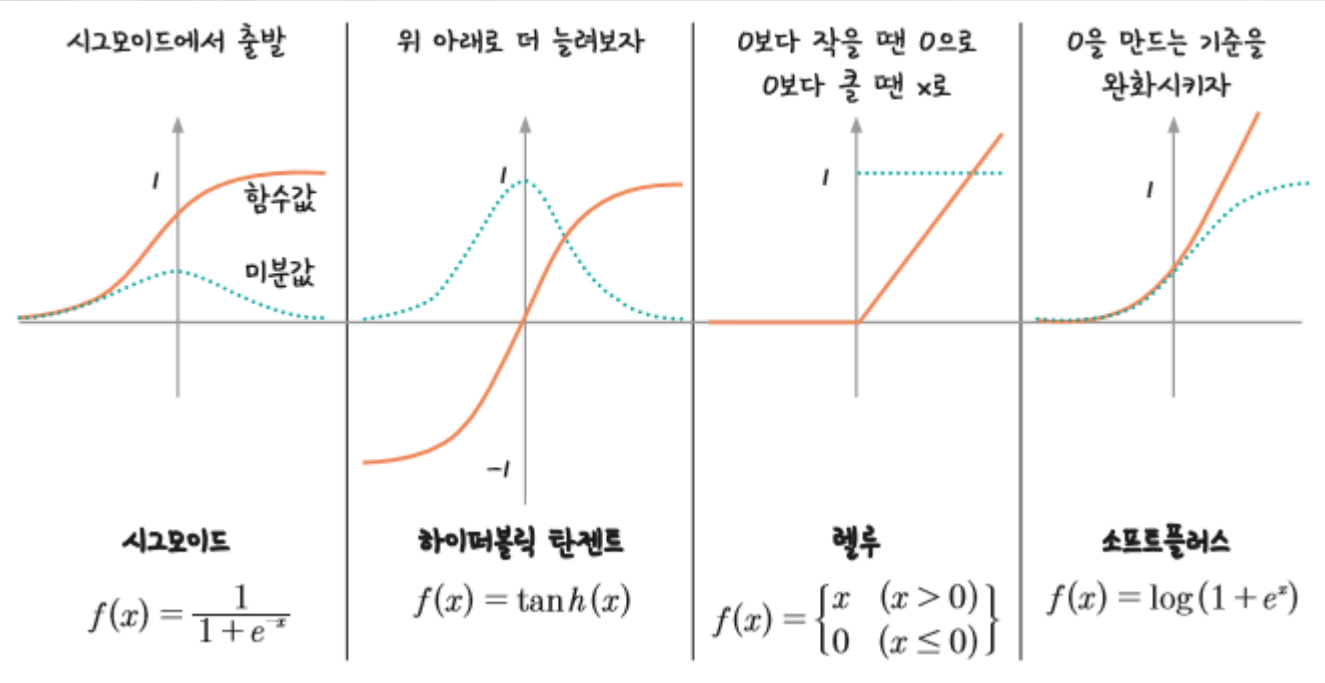


시그모이드의 미분

신경망으로 딥러닝

기울기 소실 문제와 활성화 함수

- 이를 해결하고자 활성화 함수를 시그모이드가 아닌 여러 함수로 대체하기 시작



여러 활성화 함수의 도입

신경망으로 딥러닝

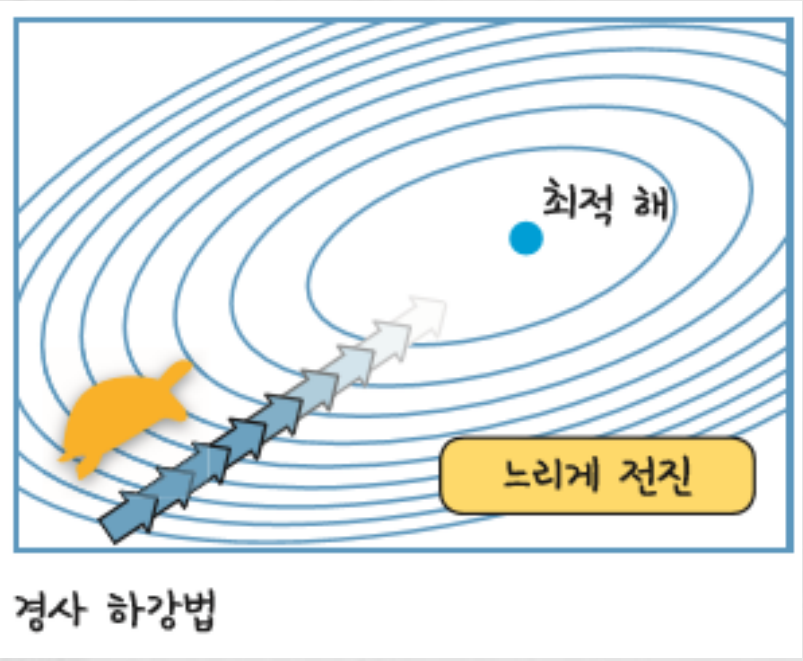
기울기 소실 문제와 활성화 함수

- 토론토대학교의 제프리 힌튼 교수가 제안한 렐루(ReLU)는 시그모이드의 대안으로 떠오르며 현재 가장 많이 사용되는 활성화 함수
- 렐루는 x가 0보다 작을 때는 모든 값을 0으로 처리하고, 0보다 큰 값은 x를 그대로 사용하는 방법. 이 방법을 쓰면 x가 0보다 크기만 하면 미분 값이 1이 됨
- 따라서 여러 은닉층을 거치며 곱해지더라도 맨 처음 층까지 사라지지 않고 남아 있을 수 있음: 딥러닝의 발전에 속도가 붙게 됨

신경망으로 딥러닝

속도와 정확도 문제를 해결하는 고급 경사 하강법

- 속도와 정확도 문제!
- 경사 하강법은 정확하게 가중치를 찾아가지만, 한 번 업데이트할 때마다 전체 데이터를 미분해야 하므로 계산량이 매우 많다는 단점이 있음

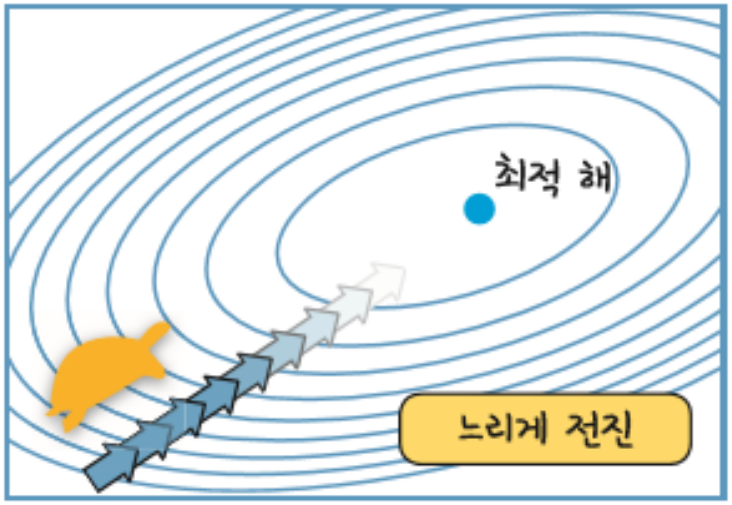


신경망으로 딥러닝

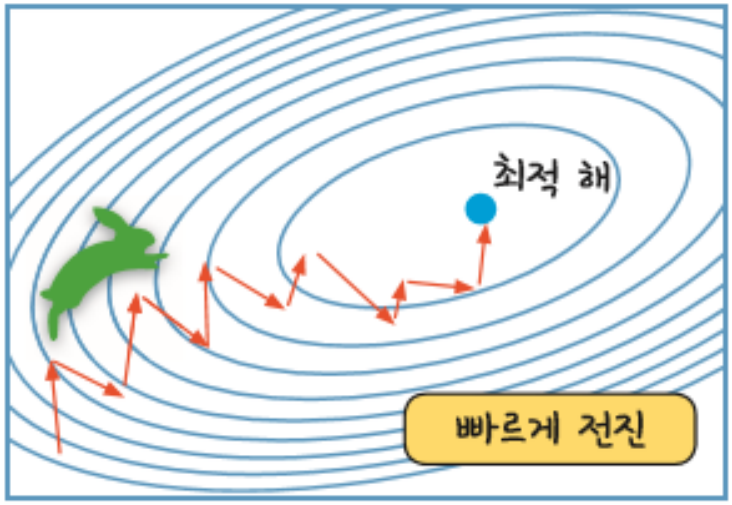
속도와 정확도 문제를 해결하는 고급 경사 하강법

확률적 경사 하강법(SGD)

- 전체 데이터를 사용하는 것이 아니라, 랜덤하게 추출한 일부 데이터를 사용
- 일부 데이터를 사용하므로 더 빨리 그리고 자주 업데이트를 하는 것이 가능해짐



경사 하강법



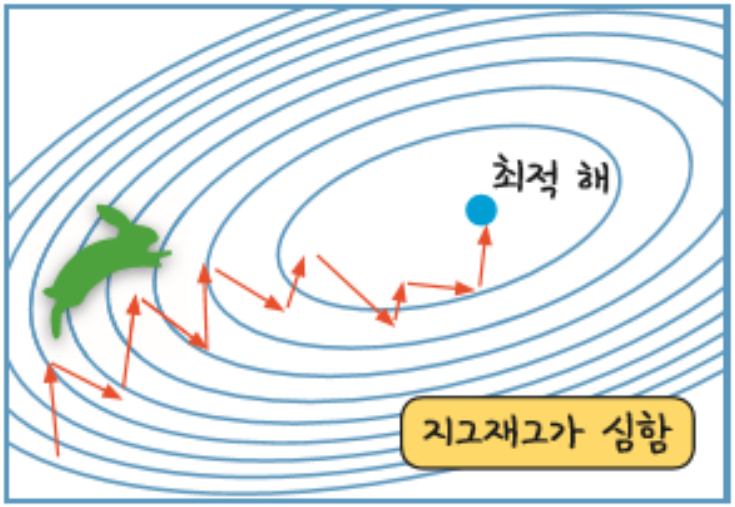
확률적 경사 하강법

신경망으로 딥러닝

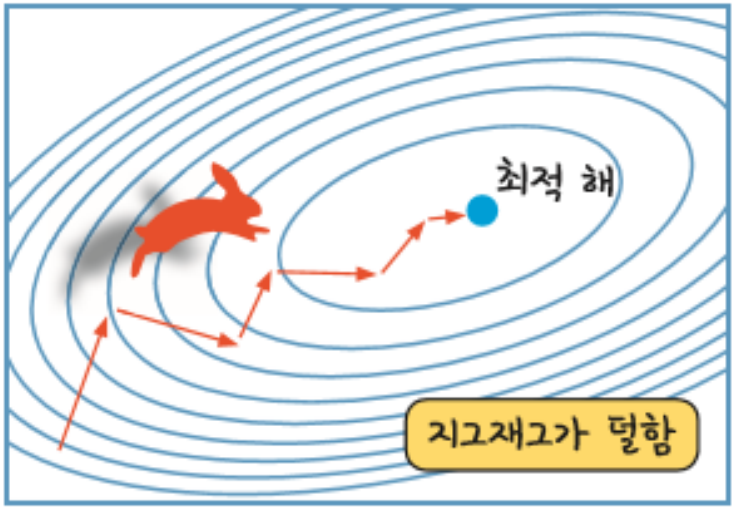
속도와 정확도 문제를 해결하는 고급 경사 하강법

모멘텀

- 경사 하강법과 마찬가지로 매번 기울기를 구하지만, 이를 통해 오차를 수정하기 전 바로 앞 수정 값과 방향(+, -)을 참고하여 같은 방향으로 일정한 비율만 수정되게 하는 방법 (이도에 탄력을 더한다)



확률적 경사 하강법



모멘텀을 적용한 확률적 경사 하강법

신경망으로 딥러닝

속도와 정확도 문제를 해결하는 고급 경사 하강법

- 딥러닝 구동에 필요한 고급 경사 하강법과 케라스 내부에서의 활용법 정리

고급 경사 하강법	개요	효과	케라스 사용법
1. 확률적 경사 하강법 (SGD)	랜덤하게 추출한 일부 데이터를 사용해 더 빨리, 자주 업데이트를 하게 하는 것	속도 개선	keras.optimizers.SGD(lr = 0.1) 케라스 최적화 함수를 이용합니다.
2. 모멘텀 (Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9) 모멘텀 계수를 추가합니다.
3. 네스테로프 모멘텀 (NAG)	모멘텀이 이동시킬 방향으로 미리 이동해서 그레이디언트를 계산. 불필요한 이동을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9, nesterov = True) 네스테로프 옵션을 추가합니다.

신경망으로 딥러닝

속도와 정확도 문제를 해결하는 고급 경사 하강법

- 딥러닝 구동에 필요한 고급 경사 하강법과 케라스 내부에서의 활용법 정리

4. 아다그라드 (Adagrad)	변수의 업데이트가 잦으면 학습률을 적게 하여 이동 보폭을 조절하는 방법	보폭 크기 개선	keras.optimizers.Adagrad(lr = 0.01, epsilon = 1e − 6) 아다그라드 함수를 사용합니다. ※ 참고: 여기서 epsilon, rho, decay 같은 파라미터는 바꾸지 않고 그대로 사용하기를 권장하고 있습니다. 따라서 lr, 즉 learning rate(학습률) 값만 적절히 조절하면 됩니다.
5. 알엠에스프롭 (RMSProp)	아다그라드의 보폭 민감도를 보완한 방법	보폭 크기 개선	keras.optimizers.RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e − 08, decay = 0.0) 알엠에스프롭 함수를 사용합니다.
6. 아담(Adam)	모멘텀과 알엠에스프롭 방법을 합친 방법	정확도와 보폭 크기 개선	keras.optimizers.Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e − 08, decay = 0.0) 아담 함수를 사용합니다.

폐암 수술 환자의 생존율 예측하기

폐암 수술 환자의 생존율 예측하기 실습

```
#1. 환경 준비
# 텐서플로 라이브러리 안에 있는 케라스 API에서 필요한 함수들을 불러옵니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# 데이터를 다루는 데 필요한 라이브러리를 불러옵니다.
import numpy as np

#2. 데이터 준비
# 준비된 수술 환자 데이터를 불러옵니다.
Data_set = np.loadtxt("./data/ThoracicSurgery3.csv", delimiter=",")
X = Data_set[:,0:16]      # 환자의 진찰 기록을 x로 지정합니다.
y = Data_set[:,16]        # 수술 1년 후 사망/생존 여부를 y로 지정합니다
```

폐암 수술 환자의 생존율 예측하기

폐암 수술 환자의 생존율 예측하기 실습

```
3. 구조 결정
# 딥러닝 모델의 구조를 결정합니다.
model = Sequential()


model.add(Dense(30, input_dim=16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


4.모델 실행
# 딥러닝 모델을 실행합니다.
model.compile(loss='binary_crossentropy', optimizer='adam', me
trics=['accuracy'])
history=model.fit(X, y, epochs=5, batch_size=16)
```

폐암 수술 환자의 생존율 예측하기

폐암 수술 환자의 생존율 예측하기 실습

- **실행 결과**

```
(중략)
Epoch 26/30 30/30 [==============================] - 0s 2ms/step - loss:
0.3944 - accuracy: 0.8489
Epoch 27/30 30/30 [==============================] - 0s 2ms/step - loss:
0.3916 - accuracy: 0.8489
Epoch 28/30 30/30 [==============================] - 0s 2ms/step - loss:
0.3927 - accuracy: 0.8511
Epoch 29/30 30/30 [==============================] - 0s 2ms/step - loss:
0.4021 - accuracy: 0.8489
Epoch 30/30 30/30 [==============================] - 0s 2ms/step - loss:
0.3946 - accuracy: 0.8511
```

- 여기서 눈여겨 봐야 할 값은 맨 아래 줄의 정확도(Accuracy)
- 정확도가 1.0 이면 예측 정확도가 100%라는 뜻
- 정확도가 0.8511이라면 이는 이 스크립트를 통해 구현된 딥러닝의 예측 정확도가 85.11%라는 뜻

폐암 수술 환자의 생존율 예측하기

폐암 수술 환자의 생존율 예측하기 실습

속성 / 클래스

항목 \ 줄	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	293	1	3.8	2.8	0	0	0	0	0	0	12	0	0	0	1	0	62	0
2	1	2	2.88	2.16	1	0	0	0	1	1	14	0	0	0	1	0	60	0
3	8	2	3.19	2.5	1	0	0	0	1	0	11	0	0	1	1	0	66	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
470	447	8	5.2	4.1	0	0	0	0	0	0	12	0	0	0	0	0	49	0

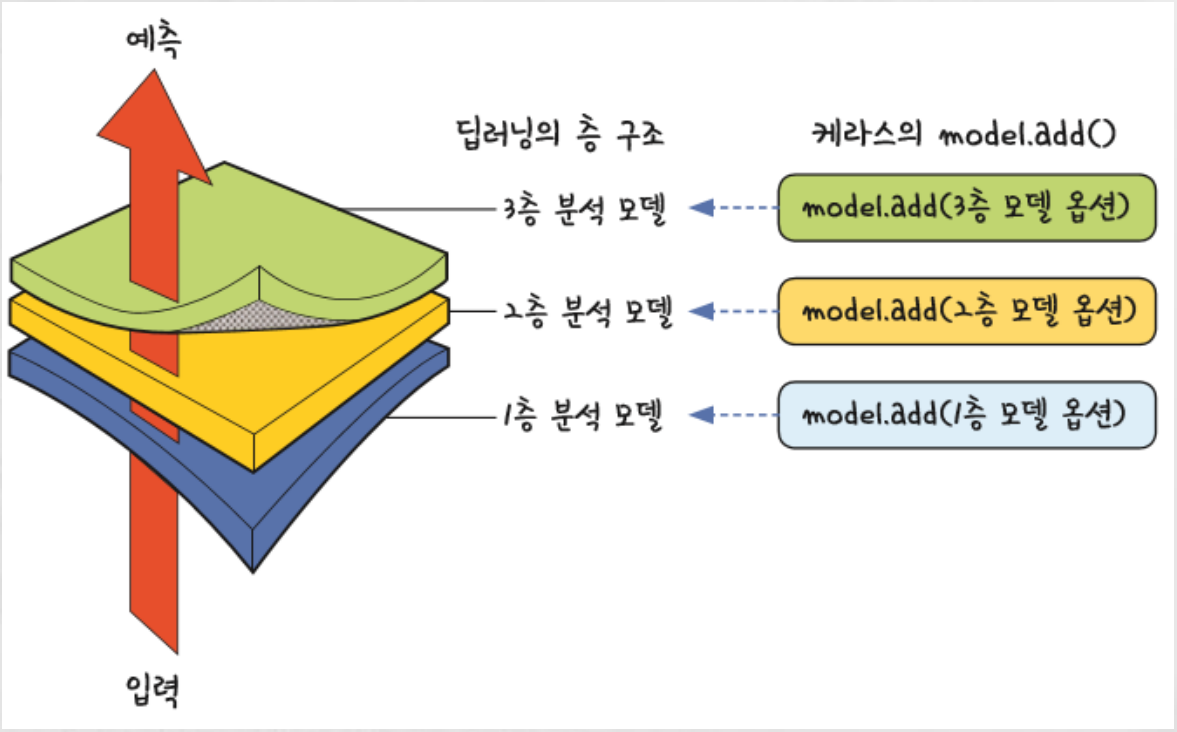
폐암 수술 환자의 의료 기록 데이터

1번째 항목부터 17번째 항목까지를 '**속성(attribute)**'이라고 하고, 정답에 해당하는 18번째 항목을 '**클래스(class)**'라고 함

폐암 수술 환자의 생존율 예측하기

폐암 수술 환자의 생존율 예측하기 실습

- 딥러닝은 여러 층이 쌓여 결과를 만들어 냄
- Sequential 함수는 딥러닝의 구조를 한 층 한 층 쉽게 쌓아올릴 수 있게 해 줌
- Sequential 함수를 선언하고 나서 model.add() 함수를 사용해 필요한 층을 차례로 추가하면 됨



딥러닝의 층 구조와 케라스

폐암 수술 환자의 생존율 예측하기

폐암 수술 환자의 생존율 예측하기 실습

- model.add() 함수를 이용해 두 개의 층을 쌓아 올렸음

```
model = Sequential()
model.add(Dense(30, input_dim=16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

- 층을 몇 개 쌓을지는 데이터에 따라 그때 그때 결정
- 케라스의 가장 큰 장점 중 하나는 model.add() 함수를 이용해 필요한 만큼의 층을 빠르고 쉽게 쌓아 올릴 수 있다는 것

폐암 수술 환자의 생존율 예측하기

폐암 수술 환자의 생존율 예측하기 실습

- model.add() 안에는 Dense() 함수가 포함되어 있음

  → 영어 단어 dense는 '조밀하게 모여있는 집합'이란 뜻, 여기서는 각 층이 제각각 어떤 특성을 가질지 옵션을 설정하는 역할을 함

- 딥러닝의 구조와 층별 옵션을 정하고 나면 compile() 함수를 이용해 이를 실행시킴

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history=model.fit(X, Y, epochs=30, batch_size=16)
```

- 여기에 쓰인 loss, optimizer, activation 등의 키워드를 이해하는 것이 바로 딥러닝 학습의 핵심

폐암 수술 환자의 생존율 예측하기

폐암 수술 환자의 생존율 예측하기 실습

```
# 결과를 출력합니다.
print("₩n Accuracy: %.4f" % (model.evaluate(X, y)[1]))
```

- 출력 부분에서는 model.evaluate() 함수를 이용해 앞서 만든 딥러닝의 모델이 어느 정도 정확하게 예측하는지를 점검할 수 있음
- 이 코드를 통해 출력되는 정확도(Accuracy)는 학습 대상이 되는 기존 환자들의 데이터 중에 일부를 랜덤하게 추출해, 새 환자인 것으로 가정하고 테스트한 결과 좀 더 신뢰할 수 있는 정확도를 측정하려면, 학습 단계에서 미리 일부를 떼어내어 테스트셋으로 저장하고 테스트할 때는 오직 이 테스트셋만을 사용

모델의 설계

모델의 정의

- 딥러닝의 모델을 설정하고 구동하는 부분은 모두 **model**이라는 함수를 선언하며 시작이 됨

- **model = Sequential()**로 시작되는 부분은 딥러닝의 구조를 짜고 층을 설정하는 부분

- **model.compile()** 부분은 위에서 정해진 모델을 컴퓨터가 알아들을 수 있게 끔 컴파일 하는 부분

- **model.fit()**으로 시작하는 부분은 모델을 실제로 수행하는 부분

모델의 설계

입력층, 은닉층, 출력층

- Sequential() 함수를 model로 선언해 놓고 model.add()라는 라인을 추가하면 새로운 층이 만들어짐

  → 코드에는 model.add()로 시작되는 라인이 두 개가 있으므로 두 개의 층을 가진 모델을 만든 것

- 맨 마지막 층은 결과를 출력하는 '출력층'이 됨

- 나머지는 모두 '은닉층'의 역할을 함

- 각각의 층은 Dense라는 함수를 통해 구체적으로 그 구조가 결정됨

모델의 설계

입력층, 은닉층, 출력층

- Dense(30, input_dim=16, activation='relu')

  → 이 층에 몇 개의 노드를 만들 것인지를 결정
- 30이라고 되어 있는 것은 이 층에 30개의 노드를 만들겠다는 것

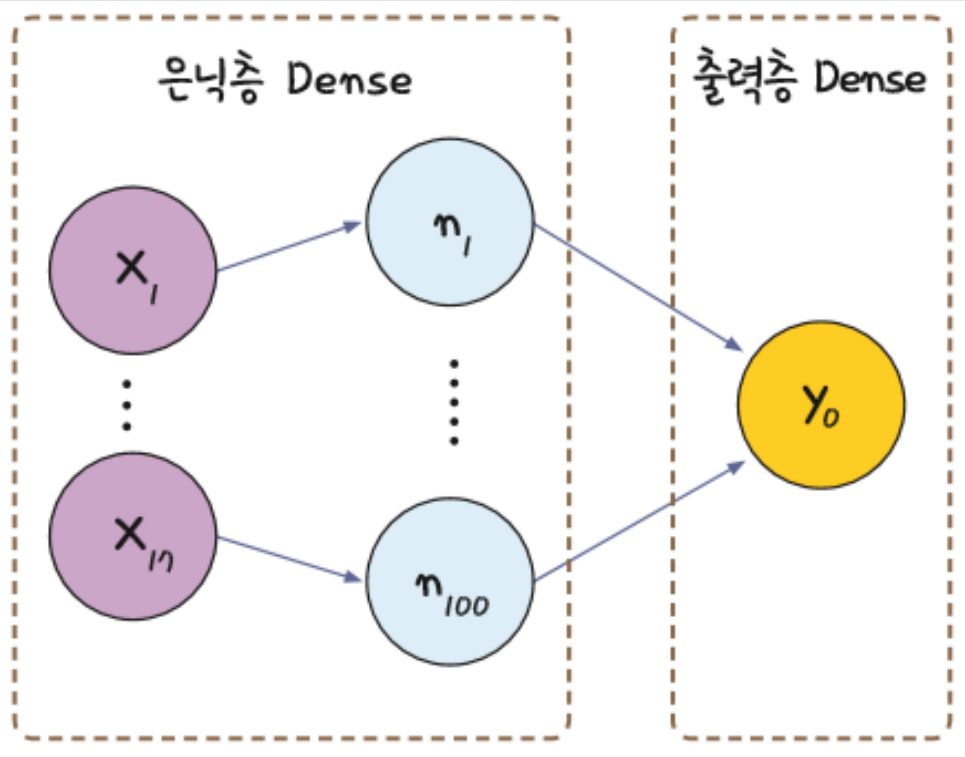
- input_dim 변수는 입력 데이터로부터 몇 개의 값이 들어올지를 정하는 것
- keras는 입력층을 따로 만드는 것이 아니라, 첫 번째 은닉층에 input_dim 을 적어 줌으로써 첫 번째 Dense가 은닉층 + 입력층의 역할을 겸함

- 폐암 수술 환자의 생존 여부 데이터에는 16개의 입력 값들이 있음

  → 데이터에서 16개의 값을 받아 은닉층의 30개 노드로 보낸다는 뜻

모델의 설계

입력층, 은닉층, 출력층



첫 번째 Dense는 입력층과 첫 번째 은닉층을, 두 번째 Dense는 출력층을 의미

모델의 설계

입력층, 은닉층, 출력층

- 이제 은닉층의 각 노드는 16개의 입력 값으로부터 임의의 가중치를 가지고 각 노드로 전송되어 활성화 함수를 만남
- 그리고 활성화 함수를 거친 결과 값이 출력층으로 전달됨
- 이때 활성화 함수로 무엇을 사용할지 지정해야 함 (activation 부분에)

모델의 설계

입력층, 은닉층, 출력층

- 두 번째 나오는 model.add(Dense(1, activation='sigmoid'))는 마지막 층이므로 이 층이 곧 출력층이 됨

- 출력 값을 하나로 정해서 보여 줘야 하므로 출력층의 노드 수는 1개

- 이 노드에서 입력받은 값 역시 활성화 함수를 거쳐 최종 출력 값으로 나와야 함

- 여기서는 시그모이드(sigmoid)를 활성화 함수로 사용

모델의 설계

모델 컴파일

- 다음으로 model.compile 부분

```
model.compile(loss=' binary_crossentropy ', optimizer='adam', metrics=['accuracy'])
```

- model.compile 부분은 앞서 지정한 모델이 효과적으로 구현될 수 있게 여러 가지 환경을 설정해 주면서 컴파일하는 부분

- 먼저 어떤 오차 함수를 사용할지를 정해야 함

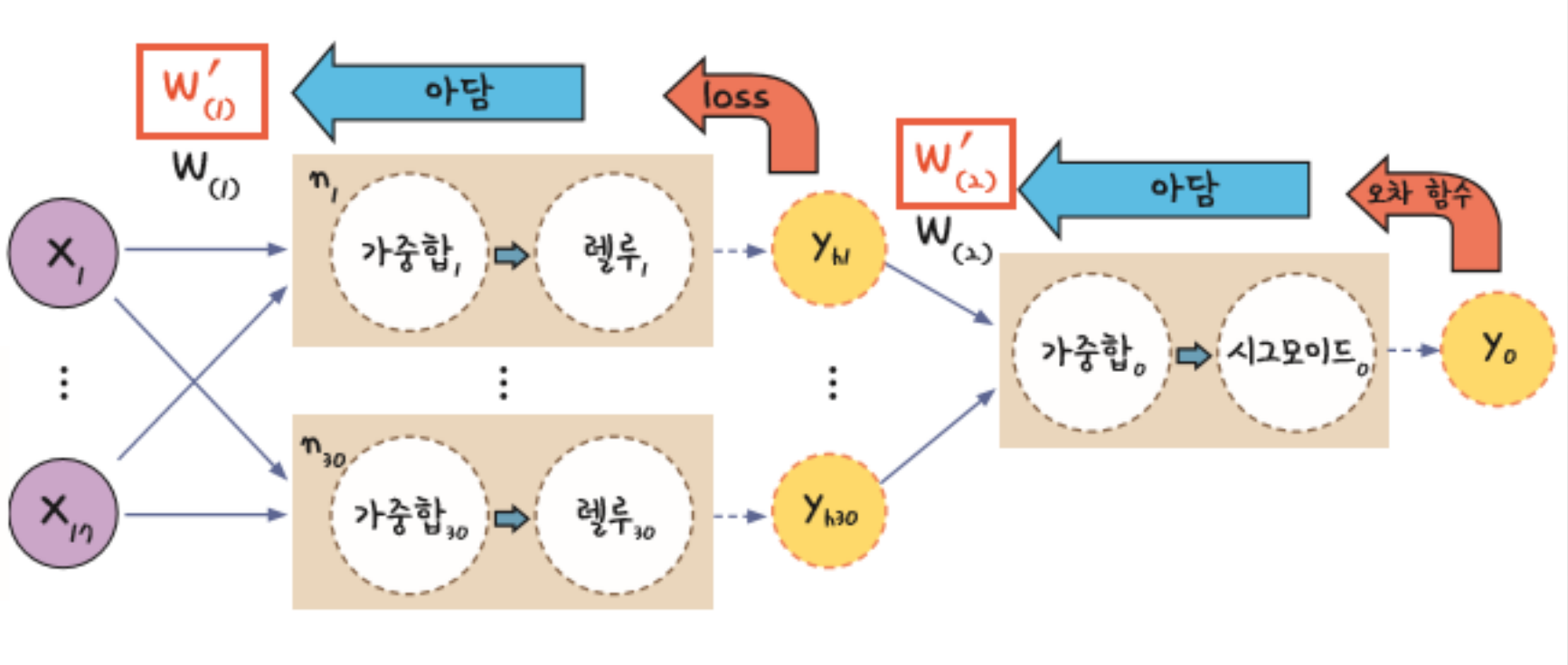
  → binary_crossentropy(이항 교차 엔트로피)를 사용

참고) 평균 제곱 오차 함수(mean_ squared_error)

모델의 설계

모델 컴파일

- 최적화를 위해 아담(adam)을 사용.



폐암 환자 생존율 예측 신경망 모델의 도식화. 여기서 W는 각 층별 가중치(w)들의 집합을 말함.

모델의 설계

교차 엔트로피

- 오차 함수에는 평균 제곱 오차 계열의 함수 외에도 교차 엔트로피 계열의 함수가 있음
  - → 평균 제곱 오차는 수렴하기까지 속도가 많이 걸린다는 단점이 있다
  - → 교차 엔트로피는 출력 값에 로그를 취해서, 오차가 커지면 수렴 속도가 빨라지고 오차가 작아지면 속도가 감소하게끔 만든 것
- 교차 엔트로피는 주로 분류 문제에서 많이 사용
  - → 예측 값이 참과 거짓 둘 중 하나인 형식일 때는 binary_crossentropy(이항 교차 엔트로피)를 씀
  - → binary_crossentropy를 적용하려면 앞 식의 mean_squared_error 자리에 binary_crossentropy를 입력하면 됨

모델의 설계

교차 엔트로피

- 케라스에서 사용되는 대표적인 오차 함수

* 실제 값을 yt, 예측 값을 yo라고 가정할 때

평균 제곱 계열	mean_squared_error	평균 제곱 오차 계산: mean(square(yt − yo))
	mean_absolute_error	평균 절대 오차(실제 값과 예측 값 차이의 절댓값 평균) 계산: mean(abs(yt − yo))
	mean_absolute_percentage_error	평균 절대 백분율 오차(절댓값 오차를 절댓값으로 나눈 후 평균) 계산: mean(abs(yt − yo)/abs(yt) (단, 분모 ≠ 0)
	mean_squared_logarithmic_error	평균 제곱 로그 오차(실제 값과 예측 값에 로그를 적용한 값의 차이를 제곱한 값의 평균) 계산: mean(square((log(yo) + 1) − (log(yt) + 1)))
교차 엔트로피 계열	categorical_crossentropy	범주형 교차 엔트로피(일반적인 분류)
	binary_crossentropy	이항 교차 엔트로피(두 개의 클래스 중에서 예측할 때)

대표적인 오차 함수

모델의 설계

모델 실행하기

- 모델의 실행: model.fit()

```
model.fit(X, y, epochs=30, batch_size=16)
```

- 학습 프로세스가 모든 샘플에 대해 한 번 실행되는 것을 1 epoch(에포크)
- Epochs=30으로 지정한 것은 각 샘플이 처음부터 끝까지 30번 재사용될 때까지 실행을 반복하라는 뜻
- batch_size는 샘플을 한 번에 몇 개씩 처리할지를 정하는 부분
- batch_size=16은 전체 470개의 샘플을 16개씩 끊어서 집어넣으라는
  - → batch_size가 너무 크면 학습 속도가 느려지고, 너무 작으면 각 실행 값의 편차가 생겨서 전체 결괏값이 불안정해질 수 있음
  - → 자신의 컴퓨터 메모리가 감당할 만큼의 batch_size를 찾아 설정

Any Questions?