딥 러닝(Deep Learning) 기초 교육 자료

Part 1

Natural Language Processing

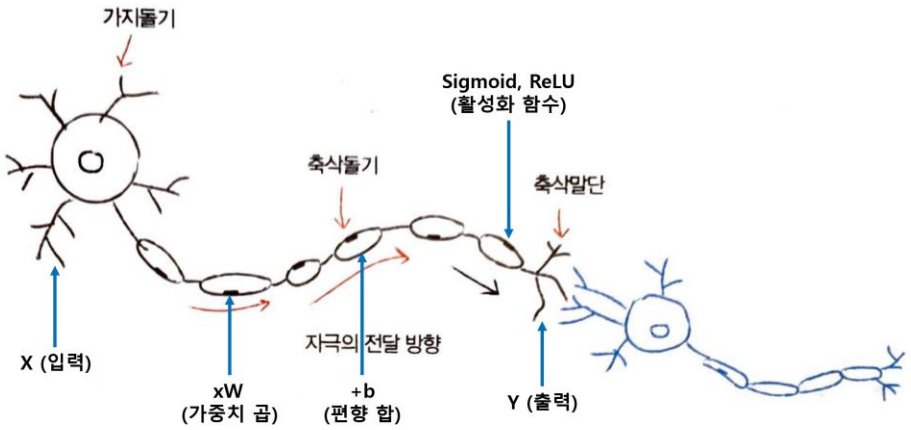
이성희

2022-04-18

목차

1-1. 딥 러닝 개요

- 딥 러닝(Deep Learning)이란?
  - 딥 러닝 또는 심층 학습은 기계 학습(ML: Machine Leaning)의 한 분야로서, 여러 비선형 변환 기법의 조합을 통해 높은 수준의 추상화를 시도하는 알고리즘의 집합으로 정의된다.
    - 추상화 : 대량의 데이터나 복잡한 자료들 속에서 핵심적인 내용 또는 기능을 요약하는 작업
  - 인간의 신경망인 뉴런(Neuron)의 작동 원리를 모방한 인공 신경망(Artificial Neural Network) 이용
    - 뉴런은 입력으로 들어온 신호(정보)를 다음 뉴런으로 전달할 때, 신호가 약해지거나, 전달되지 않거나 또는 강하게 전달되기도 하며, 이러한 원리를 지닌 수억, 수조 개의 뉴런 조합을 통해 손가락을 움직이거나 물체를 판별하는 등 다양한 조작과 판단을 수행한다.

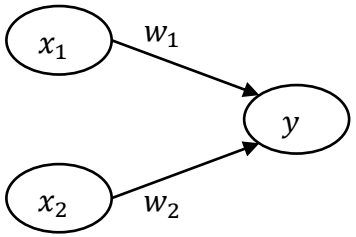


실제 뉴런과 인공 뉴런

1-2. 인공 신경망과 퍼셉트론 (1/7)

- 인공 신경망(ANN: Artificial Neural Network)이란?
  - 수학적 논리학이 아닌, 인간의 두뇌(뉴런)을 모방한 수많은 간단한 처리기들의 네트워크를 통한 문제 해결 모델
    - (수많은) 간단한 처리기 → 퍼셉트론 (신경망의 기원이 되는 알고리즘)

- 퍼셉트론(Perceptron)
  - 초기의 인공 신경망으로 여러 개의 입력 데이터(x_n)에 대한 하나의 결과(y) 값을 출력
    - x_1과 x_2는 입력(신호), w_1과 w_2는 가중치, y는 출력(결과)를 뜻한다.
    - 아래 그림의 원은 뉴런 또는 노드라고 부른다.
    - 입력 신호에 대하여, 각각 고유한 가중치를 곱하고 신호의 총합이 정해진 한계(임계값: θ)를 넘어설 때만 1을 출력한다.
      (이를 '뉴런이 활성화된다.'고 표현한다.)

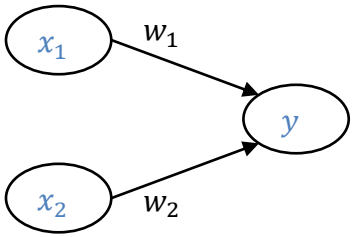


$$y = \begin{cases} 0 : (x_1 w_1 + x_2 w_2 \leq \theta) \\ 1 : (x_1 w_1 + x_2 w_2 > \theta) \end{cases}$$

1-2. 인공 신경망과 퍼셉트론 (2/7)

- 퍼셉트론(Perceptron)을 이용한 논리 회로 구현
  - AND 게이트는 입력이 둘이고 출력은 하나인 논리 회로이며, 두 입력이 모두 1일 때만 1을 출력하고, 그 외에는 0을 출력
    - 퍼셉트론으로 구현하기 위해서는, 아래 진리표와 같이 x_1, x_2에 대응되는 y를 얻기 위해 w_1, w_2, θ의 값을 설정해야 한다.
      - (w_1, w_2, θ) = (0.5, 0.5, 0.7) 또는 (0.5, 0.5, 0.8), (1.0, 1.0, 1.0) 등과 같이 무한히 많다.

    - x_1, x_2에 대응되는 y를 얻기 위해, w_1, w_2, θ를 결정해야 되는데, 이를 자동으로 결정하는 것이 신경망 학습이다.



x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

AND 게이트의 진리표

$$y = \begin{cases} 0 : (x_1 w_1 + x_2 w_2 \leq \theta) \\ 1 : (x_1 w_1 + x_2 w_2 > \theta) \end{cases}$$

w_1	w_2	θ
0.5	0.5	0.7
1.0	1.0	1.0
...	...	...

AND 게이트의 매개 변수(파라미터)

진리표와 동일한 결과를 얻기 위해, 파라미터 값을 자동으로 결정하는 것이 신경망 학습

1-2. 인공 신경망과 퍼셉트론 (3/7)

- 퍼셉트론(Perceptron)을 이용한 논리 회로 구현
  - 동일한 방법으로 퍼셉트론을 이용하여 NAND, OR 논리 회로 구현 가능
    - w_1, w_2, θ의 값을 어떻게 설정하느냐에 따라, 구현하려는 논리 회로가 결정된다.
    - 단, XOR 게이트는 단층 퍼셉트론으로 구현 불가능

AND 게이트의 진리표

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

NAND 게이트의 진리표

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

OR 게이트의 진리표

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

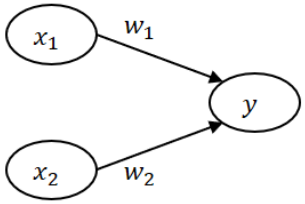
XOR 게이트의 진리표

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

1-2. 인공 신경망과 퍼셉트론 (4/7)

- 퍼셉트론(Perceptron)을 이용한 논리 회로 구현
  - 퍼셉트론은 가중치(Weight)를 곱하여 합하고, 편향(bias)을 더한 값을 이용
    - 아래 왼쪽의 식에서 θ를 $-b$로 치환하면, 오른쪽 식이 되고, 이 식에서 b를 편향(bias)라고 한다.
      - 하지만 인공 신경망에서 사용되는 편향(bias)의 개념이 임계치(θ)의 개념과 같은 것은 아니다.
        초기의 퍼셉트론을 이용한 논리 회로 구현에서 θ를 $-b$로 치환하면 임계치가 편향의 개념으로 바뀌는 것일 뿐 두 개념이 동일한 것은 아니다.

    - '편향'이라는 용어는 '한 쪽으로 치우쳐 균형을 깬다.'라는 의미를 가지고 있으며,
      실제로 아래 식에서 두 입력(x_1, x_2)이 모두 0이어도 결과로 (0이 아닌) 편향 값을 출력한다.



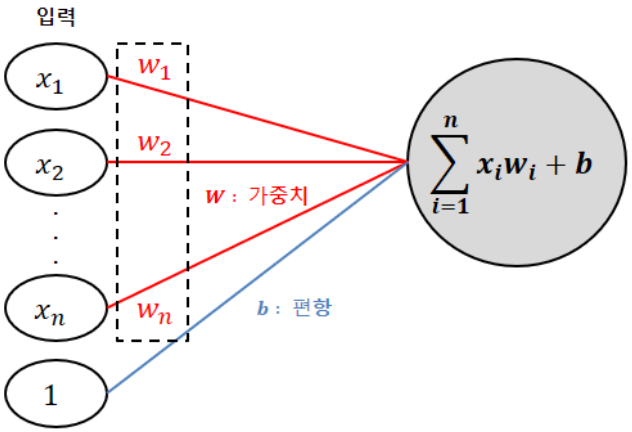
$$y = \begin{cases} 0 : (x_1 w_1 + x_2 w_2 \leq \theta) \\ 1 : (x_1 w_1 + x_2 w_2 > \theta) \end{cases}$$

$$\rightarrow \quad y = \begin{cases} 0 : (x_1 w_1 + x_2 w_2 + b \leq 0) \\ 1 : (x_1 w_1 + x_2 w_2 + b > 0) \end{cases}$$

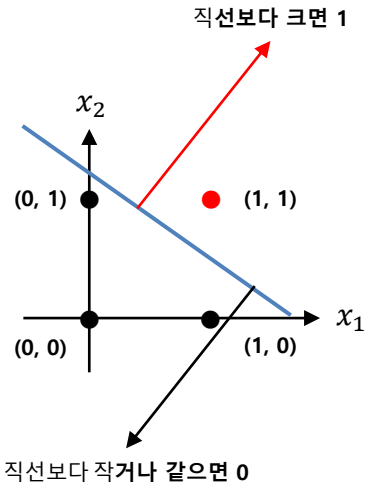
  - 즉, 퍼셉트론의 동작 원리를 수식으로 표현하자면 직선의 방정식($Wx + b$)과 유사하다.
    - W : 가중치 (Weight)
    - b : 편향 (bias)

1-2. 인공 신경망과 퍼셉트론 (5/7)

- 퍼셉트론(Perceptron)을 이용한 논리 회로 구현
  - 기계 학습 측면에서 정리하자면, 데이터를 가장 잘 표현하는(나눌 수 있는) 하나의 직선을 찾는 문제
    - (0과 1 두 영역으로 나눌 수 있는) 하나의 직선을 찾는 문제를 우리는 선형 회귀(Linear Regression)라고 부른다.
    - 찾은 직선보다 작거나 같으면 0, 크면 1을 출력하기 때문에 (이진) 분류 문제로도 볼 수 있다.
    - 즉, 하나의 직선을 찾는 선형(회귀)의 개념과 0과 1을 출력하는 (이진) 분류의 개념이 합쳐져, 퍼셉트론을 선형 분류기라고도 부른다.



$$y = \begin{cases} 0 : (x_1 w_1 + x_2 w_2 + b \leq 0) \\ 1 : (x_1 w_1 + x_2 w_2 + b > 0) \end{cases}$$

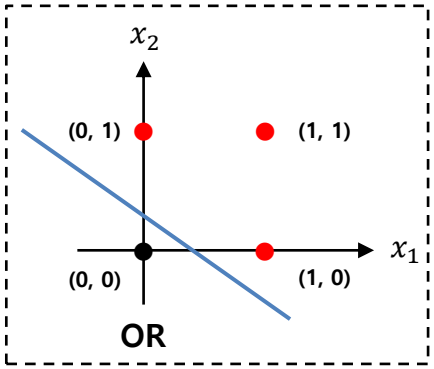


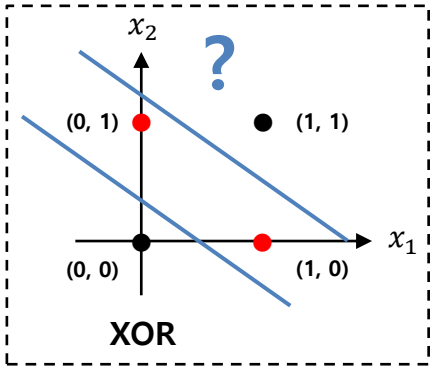
x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

AND 게이트의 진리표

1-2. 인공 신경망과 퍼셉트론 (6/7)

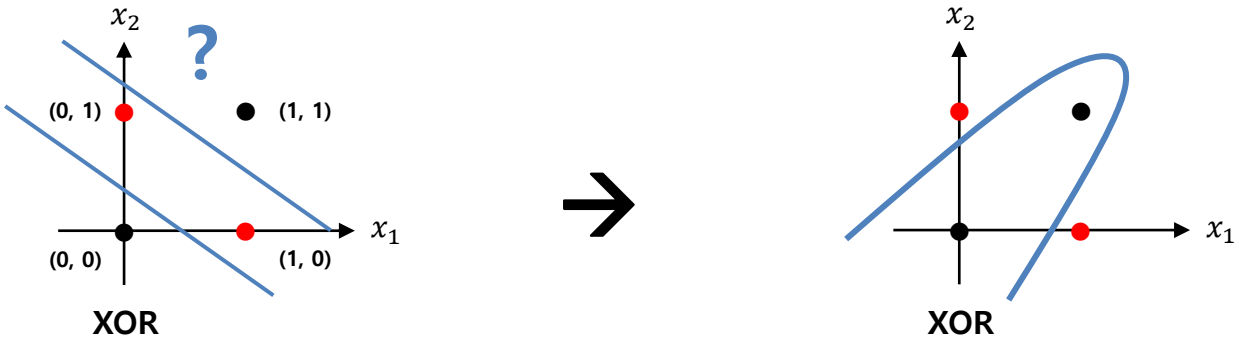
- 퍼셉트론(Perceptron)을 이용한 논리 회로 구현의 한계
  - XOR(배타적 논리합) 게이트는 앞에서 다룬 (단층) 퍼셉트론으로는 구현할 수 없다.
    - 퍼셉트론은 선형 분류기라고도 부르며, 아래와 같이 직선으로 나뉜 두 영역을 만든다.
    - 하지만, XOR의 경우 하나의 직선으로 두 영역을 나눌 수 없기 때문에, (단층) 퍼셉트론으로 구현할 수 없다.





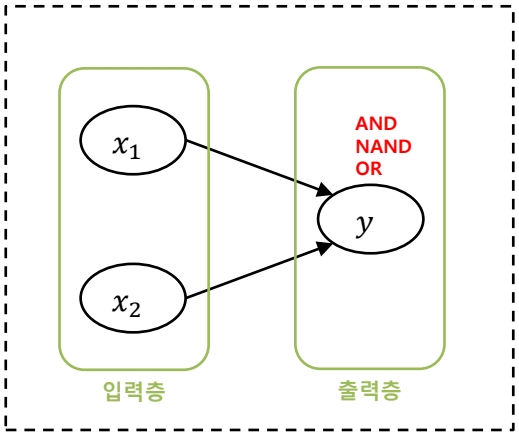
1-2. 인공 신경망과 퍼셉트론 (7/7)

- 퍼셉트론(Perceptron)을 이용한 논리 회로 구현의 한계
  - XOR은 하나의 직선으로 두 영역을 나눌 수 없다. → (단층) 퍼셉트론으로 구현할 수 없다.
  - 하지만 직선 즉, 선형이라는 제약을 없앤 비선형이라면 아래와 같이 두 영역으로 나눌 수 있다.

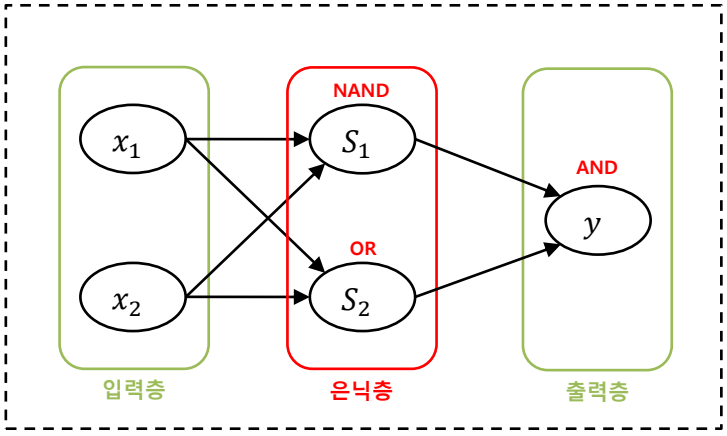


1-3. 다층 퍼셉트론과 심층 신경망 (1/13)

- 다층 퍼셉트론 (MLP: Multi Layer Perceptron)
  - 앞에서 다룬 단층 퍼셉트론은 입력과 출력 두 단계로만 이루어지고, 각 단계를 층(layer)이라고 부른다.
    - 입력층(input layer)과 출력층(output layer)
  - XOR은 기존의 AND, NAND, OR을 조합하여 구현할 수 있다.
    - 퍼셉트론 관점에서는 층(layer)을 하나 더 쌓는 개념이고, 단층 퍼셉트론에서 입력층과 출력층 사이에 새로운 층을 추가한다.
    - 이렇게 입력층과 출력층 사이에 존재하는 층을 은닉층(hidden layer)이라고 부르며, 은닉층 여부에 따라 '단층/다층'이 결정된다.



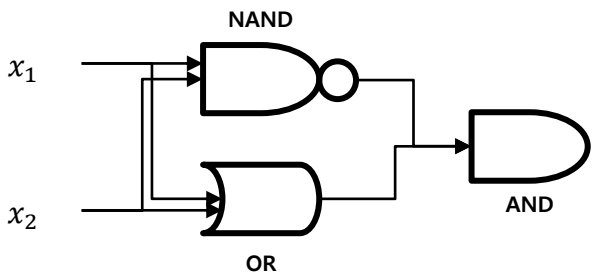
단층 퍼셉트론

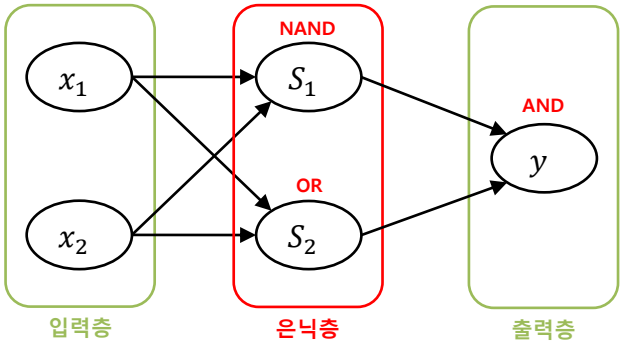


다층 퍼셉트론

1-3. 다층 퍼셉트론과 심층 신경망 (2/13)

- 다층 퍼셉트론 (MLP: Multi Layer Perceptron)
  - XOR의 경우, 하나의 직선으로 두 영역을 나눌 수 없기 때문에, 단층 퍼셉트론으로 구현이 불가능하다.
  - 논리 회로를 [NAND, OR] – [AND]의 형태로 설계하면, XOR 데이터를 하나의 선으로 나눌 수 있게 된다.



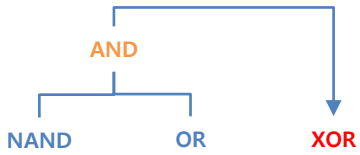


1-3. 다층 퍼셉트론과 심층 신경망 (3/13)

- 다층 퍼셉트론 (MLP: Multi Layer Perceptron)
  - XOR의 경우, 하나의 직선으로 두 영역을 나눌 수 없기 때문에, 단층 퍼셉트론으로 구현이 불가능하다.
  - 논리 회로를 [NAND, OR] – [AND]의 형태로 설계하면, XOR 데이터를 하나의 선으로 나눌 수 있게 된다.

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

XOR 게이트의 진리표



x_1	x_2	S_1	S_2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

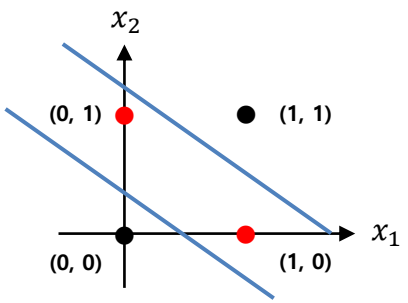
1-3. 다층 퍼셉트론과 심층 신경망 (4/13)

- 다층 퍼셉트론 (MLP: Multi Layer Perceptron)
  - XOR의 경우, 하나의 직선으로 두 영역을 나눌 수 없기 때문에, 단층 퍼셉트론으로 구현이 불가능하다.
  - 논리 회로를 [NAND, OR] – [AND]의 형태로 설계하면, XOR 데이터를 하나의 선으로 나눌 수 있게 된다.

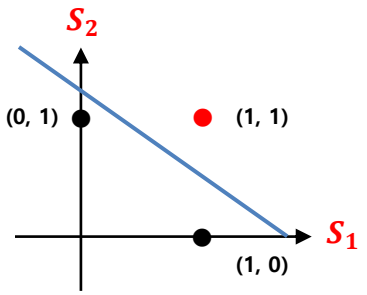
x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

XOR 게이트의 진리표

x_1	x_2	S_1	S_2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0



XOR

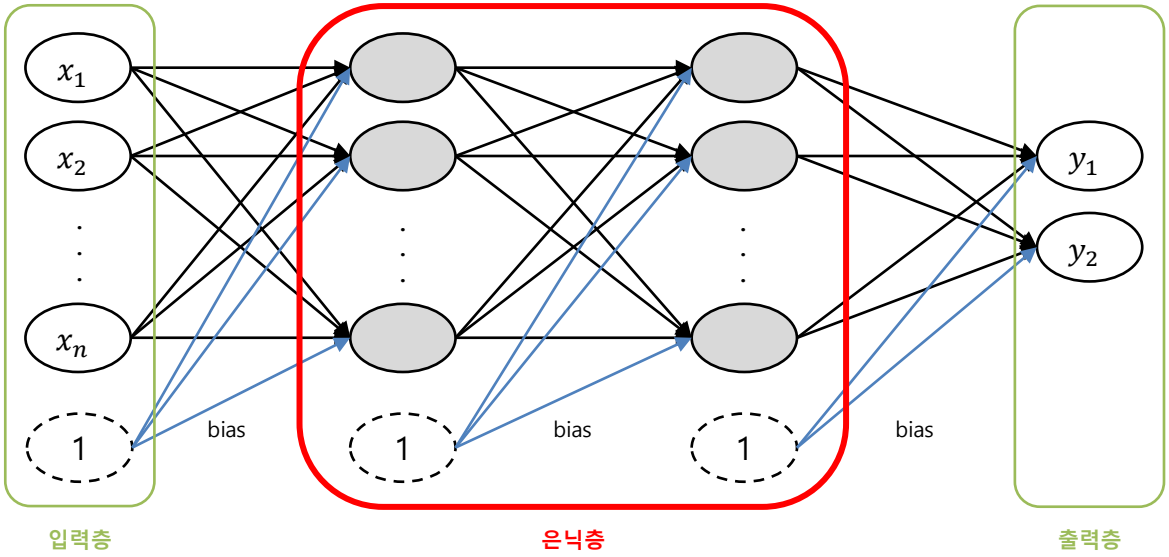


AND

1-3. 다층 퍼셉트론과 심층 신경망 (5/13)

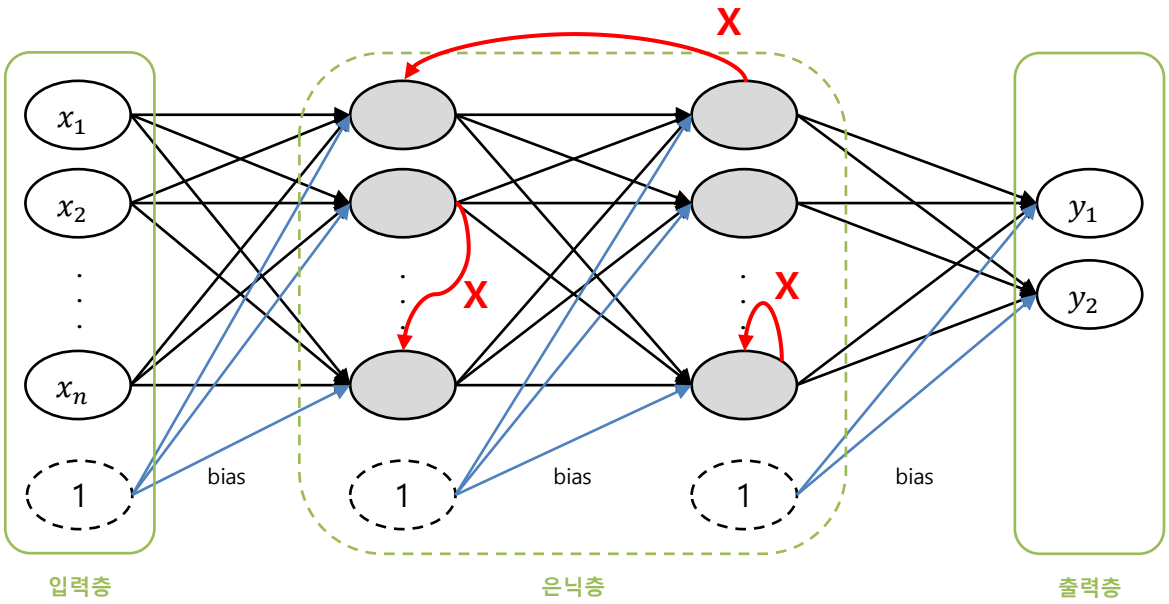
- 다층 퍼셉트론 (MLP: Multi Layer Perceptron)
  - XOR 예제에서는 하나의 은닉층으로 문제를 해결할 수 있었지만, 다층 퍼셉트론은 본래 은닉층이 1개 이상인 퍼셉트론을 뜻하며, 복잡한 문제를 해결하기 위하여 중간에 수많은 은닉층을 추가할 수 있다.

    - 은닉층의 수는 2개일 수도 있고 수십 개일 수도 있으며, 사용자에 의해 결정된다.
    - 아래는 복잡한 문제를 풀기 위해, 은닉층을 하나 더 추가하고, 뉴런의 개수를 늘린 다층 퍼셉트론의 모습이다.



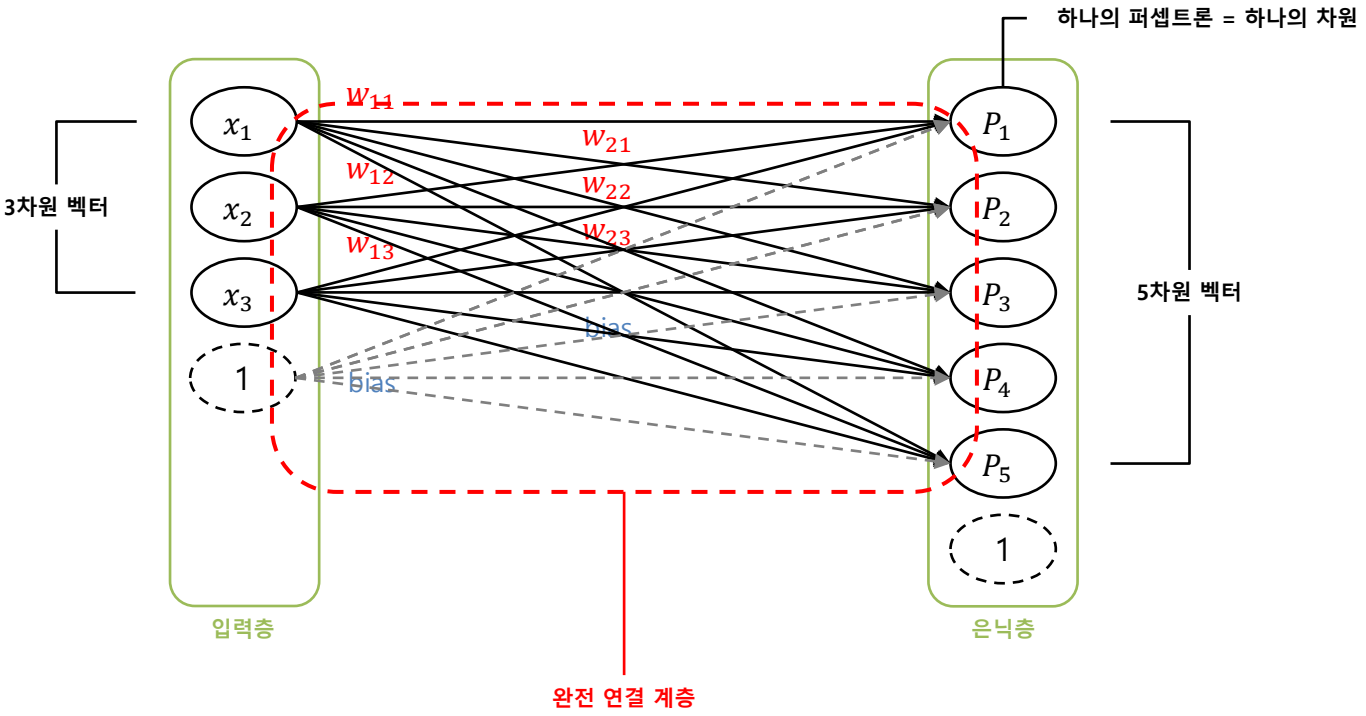
1-3. 다층 퍼셉트론과 심층 신경망 (6/13)

- 다층 퍼셉트론 (MLP: Multi Layer Perceptron)
  - MLP는 퍼셉트론으로 이루어진 층(layer) 여러 개를 순차적으로 이어 붙인 형태
  - 초기의 MLP는 정방향 인공 신경망(FFNN: Feed-Forward Neural Network)라고도 부른다.
    - 입력에 가까운 층을 왼쪽, 출력에 가까운 층을 오른쪽이라고 했을 때, 신호는 무조건 왼쪽에서 오른쪽으로만 전달된다.
    - 인접한 층 간의 연결은 있어도, 같은 층의 퍼셉트론 간의 연결은 없다. (자기 자신과의 연결도 없다.)
    - 또, 한 번 지나간 층으로 다시 연결되는 피드백(feedback)도 없다.



1-3. 다층 퍼셉트론과 심층 신경망 (7/13)

- 다층 퍼셉트론 (MLP: Multi Layer Perceptron) 연산 원리
  - Ex). 3차원 벡터를 입력으로 받아 5차원 벡터를 출력
    - 아래 그림과 같이 인접한 두 층이 모두 연결된 형태를 완전 연결 계층(FC: Fully-Connected layer)라고 한다.
      (또는 조밀층(Dense layer))

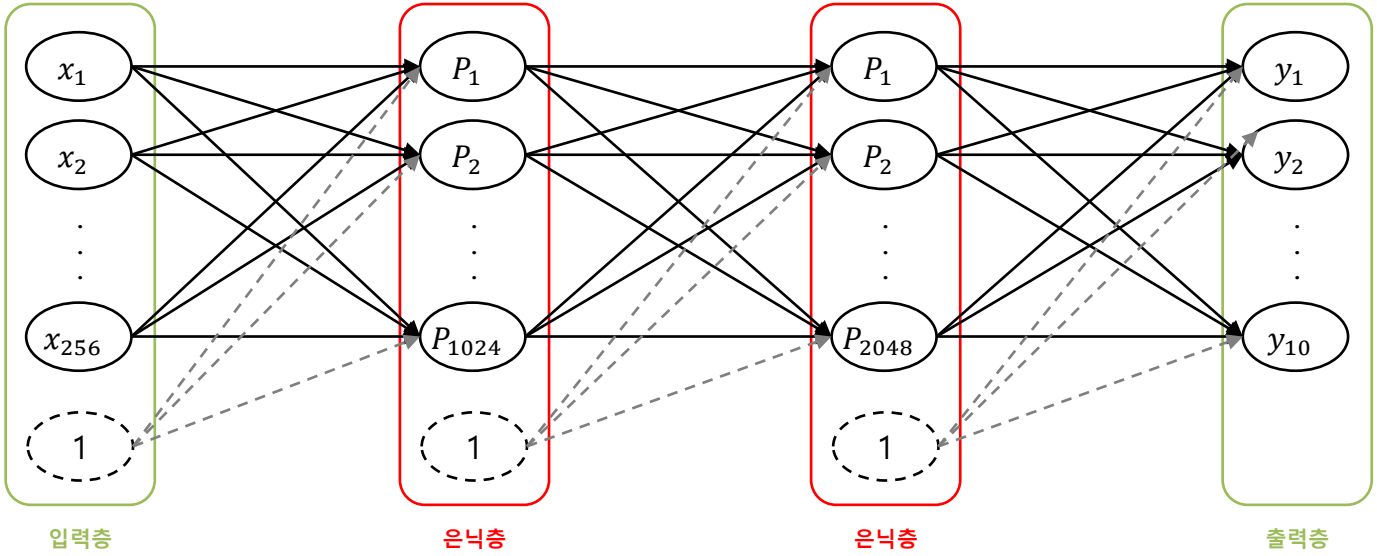


1-3. 다층 퍼셉트론과 심층 신경망 (8/13)

- 다층 퍼셉트론 (MLP: Multi Layer Perceptron) 연산 원리
  - Ex). 차원의 수가 다음과 같을 때, MLP의 매개 변수

깊이는 3층

층(layer)	입력층	첫 번째 은닉층	두 번째 은닉층	출력층
차원 수	256	1,024	2,048	10
매개 변수 수	Weight			
	bias			

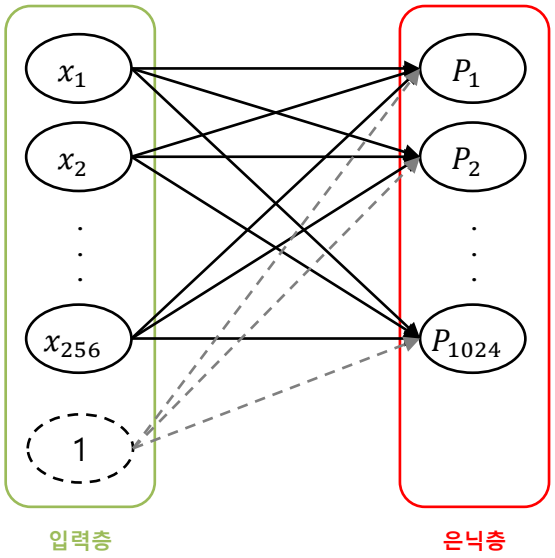


1-3. 다층 퍼셉트론과 심층 신경망 (9/13)

- 다층 퍼셉트론 (MLP: Multi Layer Perceptron) 연산 원리
  - Ex). 차원의 수가 다음과 같을 때, MLP의 매개 변수

깊이는 3층

층(layer)	입력층	첫 번째 은닉층	두 번째 은닉층	출력층
차원 수	256	1,024	2,048	10
매개 변수 수	Weight	256 x 1,024		
	bias	1,024		



[첫 번째 은닉층]
입력은 256 차원, 출력은 1,024 차원

하나의 출력 차원(퍼셉트론)은
입력 차원 수 만큼의 Weight와 1개의 bias를 가지고 있다.

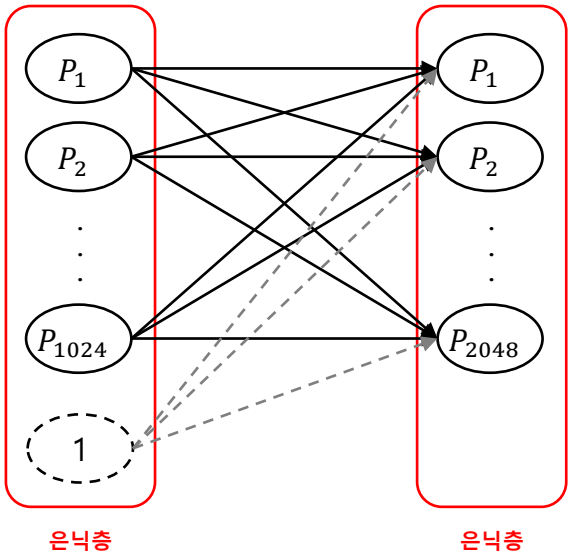
Weight : 256 x 1,024 = 262,144
bias : 1,024

1-3. 다층 퍼셉트론과 심층 신경망 (10/13)

- 다층 퍼셉트론 (MLP: Multi Layer Perceptron) 연산 원리
  - Ex). 차원의 수가 다음과 같을 때, MLP의 매개 변수

깊이는 3층

층(layer)	입력층	첫 번째 은닉층	두 번째 은닉층	출력층
차원 수	256	1,024	2,048	10
매개 변수 수	Weight	256 x 1,024	1,024 x 2,048	
	bias	1,024	2,048	



[두 번째 은닉층]
입력은 1,024 차원, 출력은 2,048 차원

Weight : 1,024 x 2,048 = 2,097,152
bias : 2,048

1-3. 다층 퍼셉트론과 심층 신경망 (11/13)

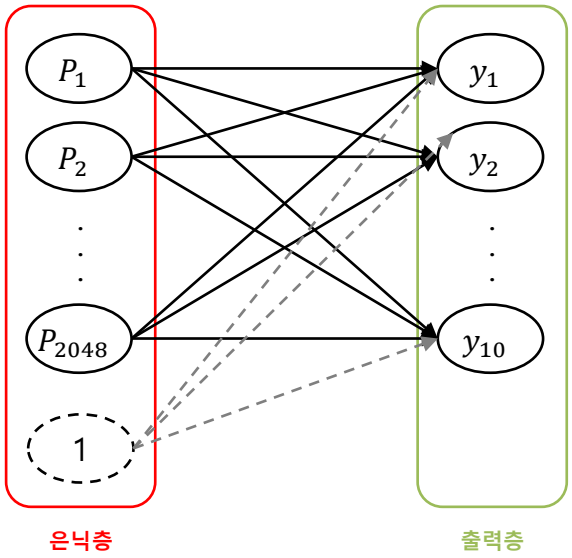
- 다층 퍼셉트론 (MLP: Multi Layer Perceptron) 연산 원리
  - Ex). 차원의 수가 다음과 같을 때, MLP의 매개 변수

깊이는 3층

층(layer)	입력층	첫 번째 은닉층	두 번째 은닉층	출력층
차원 수	256	1,024	2,048	10
매개 변수 수	Weight	256 x 1,024	1,024 x 2,048	2,048 x 10
	bias	1,024	2,048	10

[출력층]
입력은 2,048 차원, 출력은 10 차원

Weight : 2,048 x 10 =20,480
bias : 10

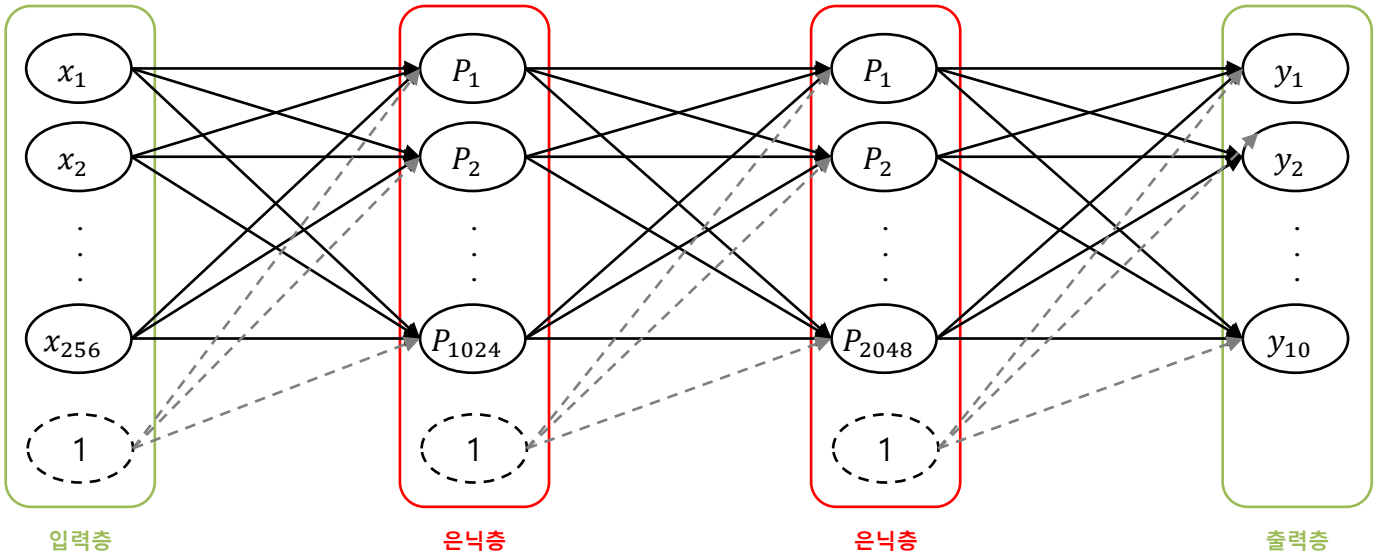


1-3. 다층 퍼셉트론과 심층 신경망 (12/13)

- 다층 퍼셉트론 (MLP: Multi Layer Perceptron) 연산 원리
  - Ex). 차원의 수가 다음과 같을 때, MLP의 매개 변수
    - 전체 매개 변수 수
      → 2,382,828 (약 2.4M)

깊이는 3층

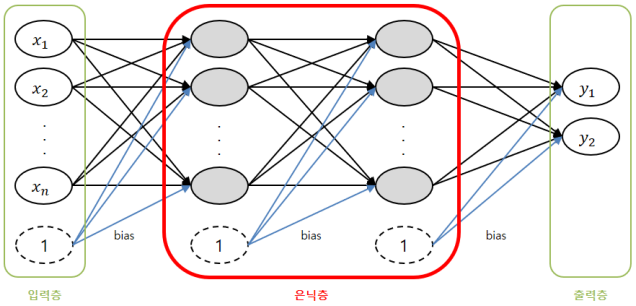
층(layer)	입력층	첫 번째 은닉층	두 번째 은닉층	출력층
차원 수	256	1,024	2,048	10
매개 변수 수	Weight	256 x 1,024	1,024 x 2,048	2,048 x 10
	bias	1,024	2,048	10

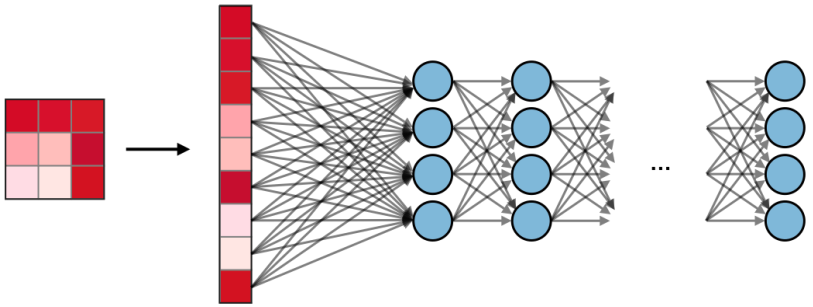


1-3. 다층 퍼셉트론과 심층 신경망 (13/13)

- 심층 신경망 (DNN: Deep Neural Network)
  - 은닉층이 2개 이상인 신경망을 심층 신경망이라고 부르며, 심층 신경망은 다층 퍼셉트론 뿐만 아니라, 여러 변형된 다양한 신경망들도 은닉층이 2개 이상이라면, 심층 신경망이라고 한다.

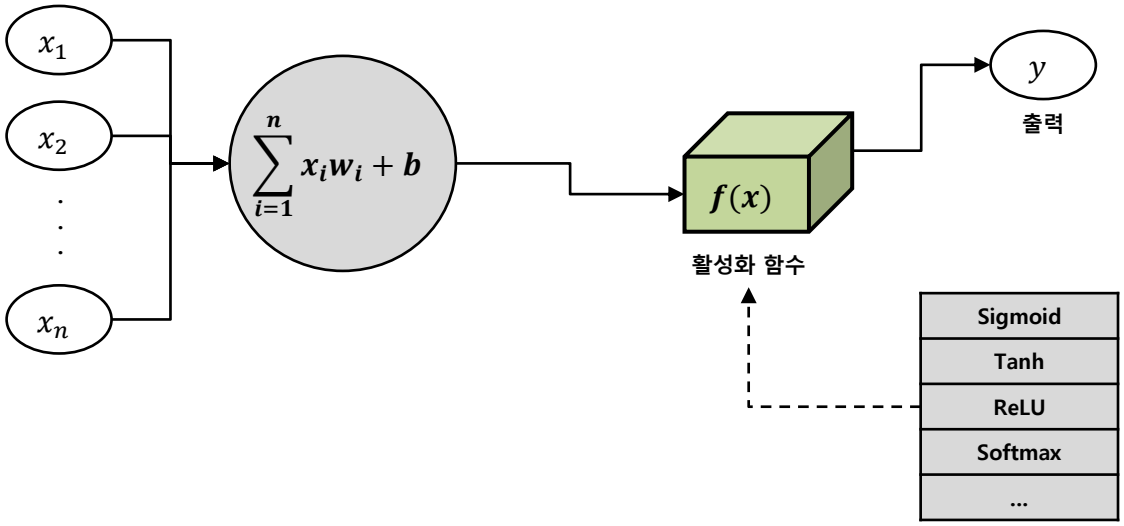
  - 머신 러닝에서 학습 시키는 인공 신경망이 심층 신경망일 경우, 심층 신경망을 학습시킨다고 하여 딥 러닝(Deep Learning)이라고 한다.





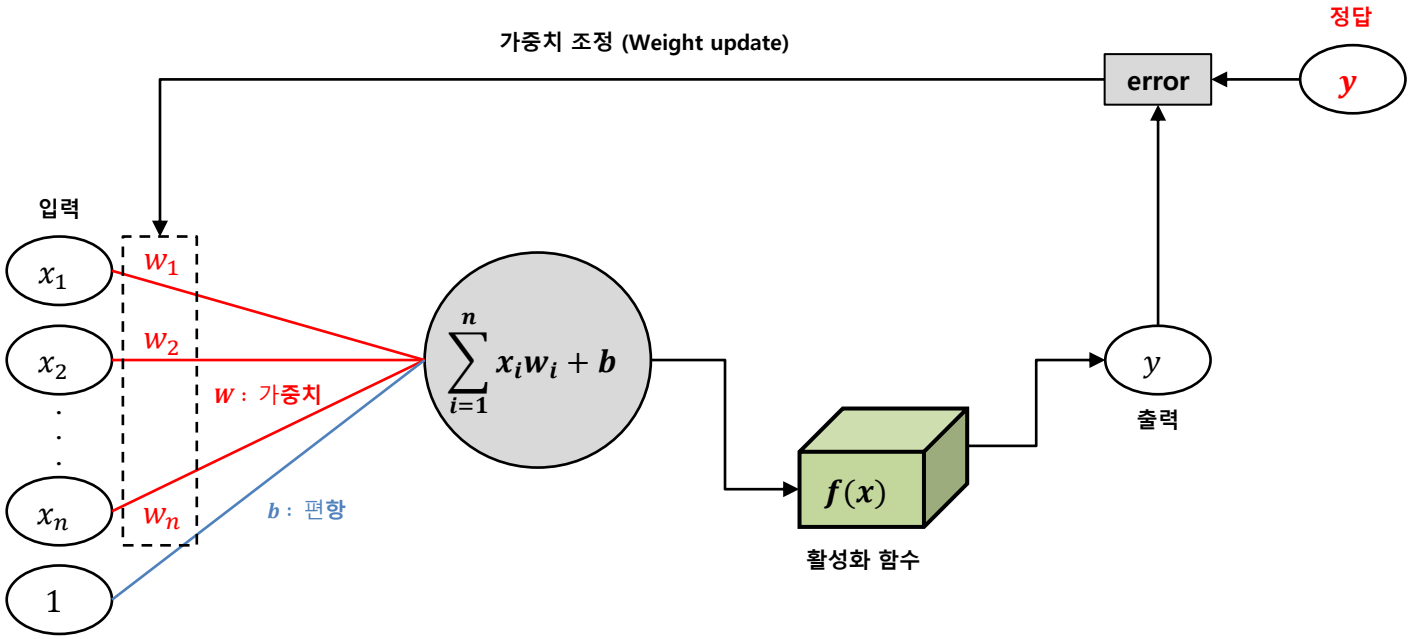
2-1. 퍼셉트론과 활성화 함수 (1/2)

- 활성화 함수(Activation function)이란?
  - 퍼셉트론에서 입력(x_n)에 대하여, 가중 합을 계산하고 편향 값을 더한 뒤, 이 값을 출력 값으로 변환하는 함수
  - 인공 신경망에서는 이전 계층(layer)에서 전달된 데이터(x_n)에 대하여 값을 계산하고,
    활성화 함수를 통해 변환된 출력 값을 다음 계층으로 전달한다.
  - 신경망의 목적에 따라, 혹은 레이어의 역할에 따라 선택적으로 적용



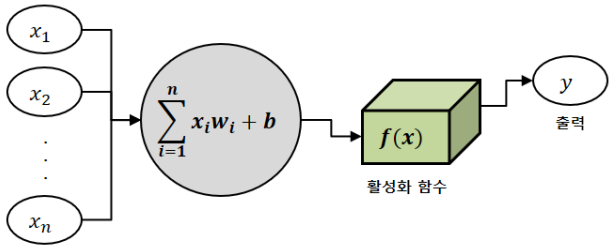
2-1. 퍼셉트론과 활성화 함수 (2/2)

- 활성화 함수(Activation function)을 포함한 퍼셉트론의 구조



2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (1/16)

- 활성화 함수의 기본 정의
  - 입력에 대한 가중 합을 계산하고 편향 값을 더한 뒤, 활성화 함수를 통해 출력 값으로 변환하는 개념은 모두 동일하다.
    - 목적에 맞는 값을 얻기 위해 사용되는 것이 활성화 함수이며, 목적에 따라 활성화 함수의 종류가 결정된다.



- ① 초기의 인공 신경망에서 퍼셉트론과 활성화 함수
  - 초기 인공 신경망에서는 퍼셉트론의 활성화 함수로 계단 함수를 사용
    - 계단 함수의 특성으로 출력 값은 오직 0 또는 1
    - 출력 값이 0 또는 1이므로, 초기 인공 신경망으로 이진 분류 문제를 해결할 수 있었다.
  - 하지만, 계단 함수를 사용한 초기 인공 신경망은 기울기가 무한대($x = 0$)인 구간에서 미분이 불가하여, 가중치의 업데이트 과정에서 문제가 발생한다.

    따라서 계단 함수는 단일 퍼셉트론의 활성화 함수로만 사용되며, 다층 퍼셉트론(딥러닝)의 활성화 함수로는 부적절하다.

2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (2/16)

- ② 현재의 인공 신경망에서 퍼셉트론과 활성화 함수
  - 계단 함수를 사용한 초기 인공 신경망의 한계를 극복하기 위하여, Sigmoid, ReLU 등의 다양한 활성화 함수 등장
  - 이 다양한 활성화 함수들을 신경망의 목적 또는 레이어의 역할에 따라 선택적으로 적용
    - 예를 들어, Sigmoid 함수는 입력 값이 커지면 1, 작아지면 0에 수렴하는 성질을 갖고 있다.
      이 성질 때문에, 출력 값으론 0부터 1까지의 값만을 가지며, 출력 값이 0.5 보다 크면 1(true), 작거나 같으면 0(false)으로 변환하여
      이진 분류 문제를 해결할 수 있다.

      ※ Sigmoid 함수를 사용하면 자동으로 0 또는 1의 값을 출력하는 것이 아니라,
      0부터 1까지의 값을 특정 임계치 기준으로 나누어서 0 또는 1의 값으로 변환하는 작업이 필요하다.
    - 즉, 계단 함수를 사용한 초기 인공 신경망의 퍼셉트론은 출력 자체가 0 또는 1이기 때문에, 바로 이진 분류 문제를 해결할 수 있는 개념
    - Sigmoid 함수를 사용한 퍼셉트론은 출력으로 0부터 1까지 값만을 가지므로, 이진 분류 분포를 설명하기에 적합하며,
      Sigmoid 함수를 가설로 사용하여 이진 분류 문제를 해결하는 대표적인 알고리즘이 로지스틱 회귀이다.

      (Sigmoid 함수를 이용하여 실수 전체의 영역을 0과 1사이의 값으로 변환(압축)하면서 선형을 비선형으로 바꾸는 개념이 적용된다.)

2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (3/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 다층 퍼셉트론(MLP)는 복잡한 문제를 해결하기 위하여, 은닉층의 수 또는 각 층의 차원을 조절한다.
    - 심층 신경망은 MLP에서 은닉층의 수가 2개 이상인 경우를 뜻한다.
  - 심층 신경망에서 활성화 함수의 역할은 2가지로 정의할 수 있다.
    - (1) 네트워크의 목적에 맞는 출력 값으로 변환
    - (2) 심층으로 쌓은 은닉층의 선형성을 비선형성으로 변환
  - 위 (1)의 개념은 기존의 활성화 함수 역할과 동일하다.
    - 출력층에서 목적에 맞는 최종 출력 값으로 변환
  - 그렇다면, (2)의 은닉층의 선형을 비선형으로 변환한다는 뜻은 무엇일까?
  - 선형을 비선형으로 변환한다는 개념을 배우기 전에, 먼저 심층 신경망에서 선형이 가지는 의미를 알 필요가 있다.

2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (4/16)

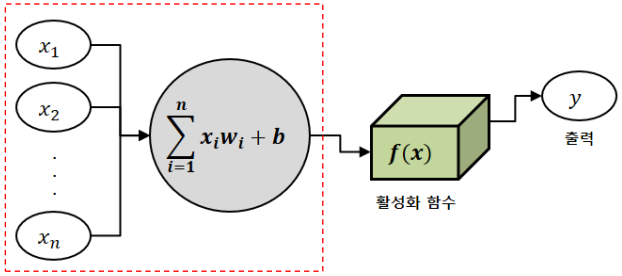
- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 심층 신경망에서 선형이라는 개념은 무슨 뜻일까?

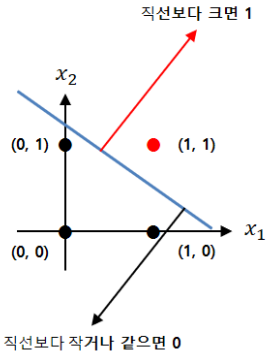
    - 계단 함수를 사용한 초기의 퍼셉트론은 두 영역으로 나눌 수 있는 하나의 직선을 찾고, 직선보다 작거나 같으면 0, 크면 1을 출력함으로써 선형 분류기로 불린다고 했다.

    - 여기서, 0 또는 1을 출력하는 것이 활성화 함수 부분이고, 그 이전에 직선을 찾는 부분이 선형이라고 표현된다.

      임의의 값 x에 대하여, 출력(예측) 값 y를 얻으려면 x와 y 사이의 관계식을 찾아내야 하는데
      (관계식은 기계 학습 관점에서의 가설)

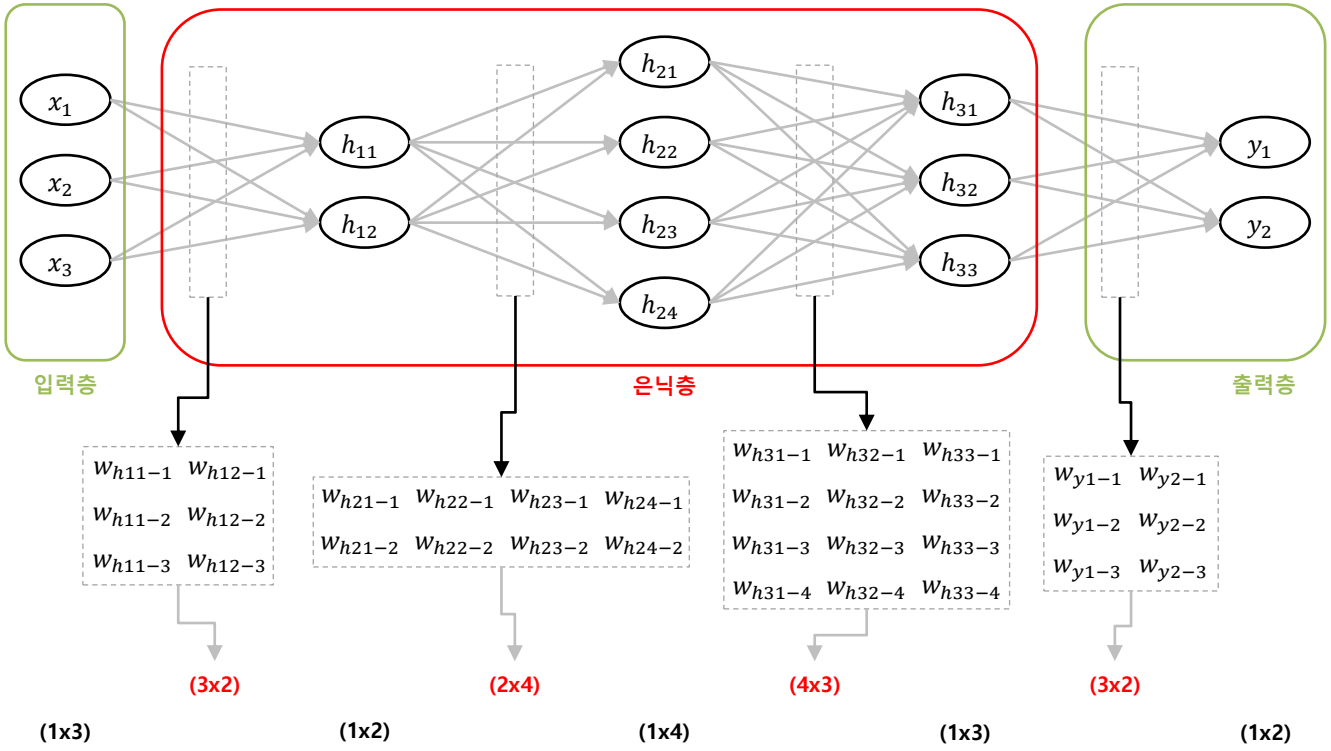
      이 관계식이 직선의 형태라면, 얻어지는 출력 값은 모두 직선 위의 한 점이 되고, 모든 출력 값을 연결하면 하나의 직선이 된다.
      즉, x와 y 사이의 관계식이 직선의 방정식이라면, 가장 적합한 하나의 직선을 찾는 과정이 학습이고 출력 값으로 직선 위의 값(점)을 얻으므로 선형이라고 부른다.





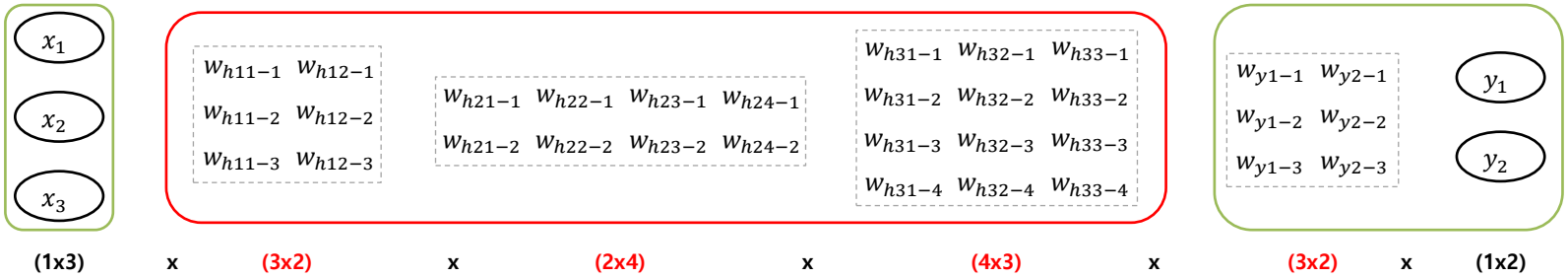
2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (5/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 은닉층의 퍼셉트론에서 활성화 함수를 제외하고 직선을 찾는 부분(선형)만 학습한다면 어떻게 될까?
    - 아래는 3개의 은닉층을 가지는 간단한 심층 신경망의 모습이며, 빠른 이해를 위해 bias는 생략한다.

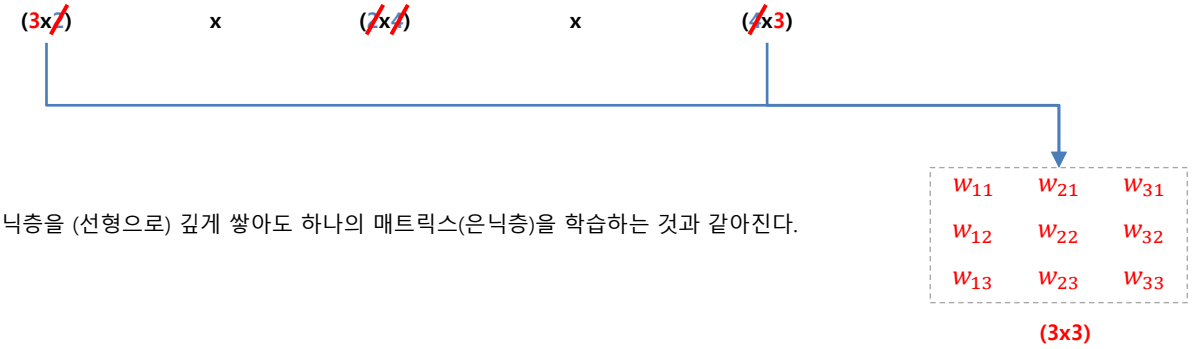


2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (6/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 심층 신경망을 학습한다는 것은 아래 가중치(Weight) 집합에 대하여, 적절한 값을 찾는 과정이다. (bias 생략)
    - 은닉층에 대한 연산 과정은 아래 붉은 박스 부분이다.
    - 하나의 매트릭스는 하나의 은닉층이 가지는 매개 변수(학습을 통해 찾아야 하는 변수)와 동일하다.



- 하지만, 실제 행렬의 곱셈 연산을 수행하면, 하나의 행렬로 축소되는데



- 결국 아무리 은닉층을 (선형으로) 깊게 쌓아도 하나의 매트릭스(은닉층)을 학습하는 것과 같아진다.

2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (7/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 심층 신경망의 퍼셉트론에서 활성화 함수 부분을 제외하고, 선형으로만 학습을 진행할 경우,
    심층으로 쌓은 수많은 은닉층들을 학습하는 것이 결국엔 하나의 은닉층을 학습하는 것과 동일해진다고 했다.
    이를 수학적 수식으로 간략하게 풀어보면 다음과 같다.

    - 하나의 은닉층 연산 과정을 단순하게 $f(x) = Wx + b$로 정의한다면, 3개의 은닉층에 대한 수식은 다음과 같아진다.

      → $f_3(f_2(f_1))$
      → $f_3(f_2(W_1x + b_1))$
      → $f_3(W_2W_1x + W_2b_1 + b_2)$
      → $W_3W_2W_1x + W_3W_2b_1 + W_3b_2 + b_3$

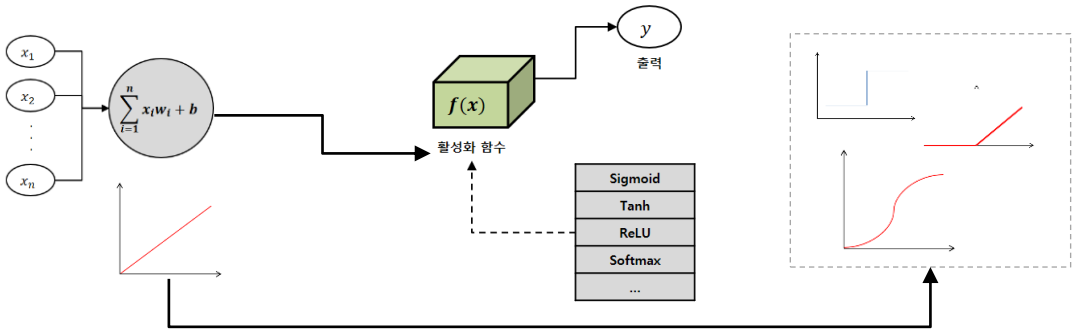
      * $\boldsymbol{W}_{final} = W_3W_2W_1$
      * $\boldsymbol{b}_{final} = W_3W_2b_1 + W_3b_2 + b_3$

    - 역시 마찬가지로 $W_{final}x + b_{final} = Wx + b$, 하나의 직선의 방정식을 학습하는 것과 동일하다.

    - 즉, 선형 레이어는 아무리 심층으로 쌓아도 결국 하나의 선형 레이어를 학습하는 것과 동일해지기 때문에
      심층으로 학습하는 의미가 없어진다.

2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (8/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 그렇다면, 심층 신경망에선 선형 레이어를 사용하면 안 되는가?
    - 그건 아니다. 선형 레이어를 연속해서 쌓는 것이 문제이지, 사용하면 안 되는 것은 아니다.
      하지만, 연속해서 쌓을 경우 심층으로 쌓는 의미가 사라져버린다.
  - 직선에 직선을 넣고 다시 그 결과인 직선을 또 직선에 넣어봤자 결국, 다시 하나의 직선이 된다.
  - 이를 방지하기 위해, 선형을 비선형으로 바꾸는 작업이 필요하며, 그 역할을 수행하는 것이 활성화 함수이다.
    - 활성화 함수의 그래프를 보면 알 수 있듯이, 모든 활성화 함수의 모양은 직선이 아닌 다른 모양을 가진다.
    - 이 말 뜻은 활성화 함수를 통해 나오는 출력 값은 직선의 형태가 아닌 다른 형태를 가진다는 뜻이며,
      결국 선형을 비선형으로 바꾸는 것과 같은 개념이다.

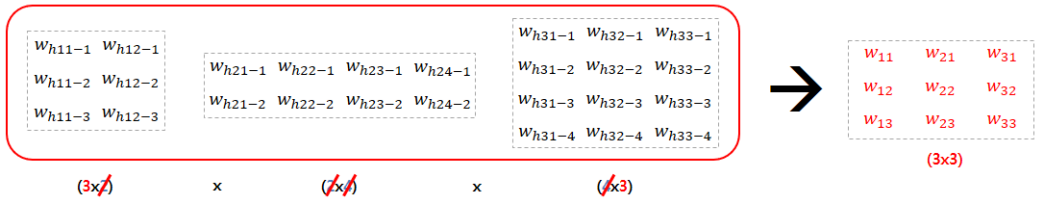


2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (9/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 심층 신경망에서 비선형 레이어의 역할

    - 선형 레이어를 연속으로 쌓았을 때, 결국 하나의 선형 레이어가 된다는 것은 결국 은닉층이 1개인 다층 퍼셉트론을 학습하는 것과 같다.

      인공 신경망에서의 학습은 각 레이어의 매개 변수(Weight, bias)들의 적절한 값을 찾는 과정이며, 이 매개 변수의 수가 많을 수록 신경망의 복잡도는 증가하고, 복잡도가 증가할 수록 어려운 문제를 풀 가능성이 커진다.



    - 하지만, 위 그림처럼 3개의 은닉층(매트릭스)를 선형으로 쌓았다면, 결국 하나의 은닉층(매트릭스)로 축소되면서 신경망의 복잡도는 감소한다.

    - 이는 문제 해결을 위해 높은 복잡도를 요구하는 심층 신경망으로 보기 어려우며,
      (심층 신경망의 최소 은닉층 수는 '2'이다.)

      은닉층의 수가 감소한다는 것은 학습하려는 매개 변수의 수가 감소한다는 뜻이며, 이는 신경망의 복잡도를 감소시킨다는 뜻이다.
      이를 방지하기 위해 사용되는 것이 활성화 함수이며, 선형 레이어에 활성화 함수를 결합한 비선형 레이어를 사용한다면 복잡도를 유지할 수 있다.

2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (10/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 심층 신경망에서 비선형 레이어를 이용한 복잡도 유지(감소 방지)
    - 비선형 레이어의 활성화 함수를 이용하여, 복잡도를 유지한다는 개념은 아래의 수식으로 확인할 수 있다.
      - (왼쪽) 선형 – 선형 – 선형 / (오른쪽) 선형 – 비선형 – 선형

$f_1, f_2, f_3 = f(x) = Wx + b$	$f_1, f_3 = f(x) = Wx + b \;\; / \;\; f_2 = \frac{1}{1 + e^{-(Wx+b)}}$
$f_3(f_2(f_1)) = f_3(f_2(W_1x + b_1))$	$f_3(f_2(f_1)) = f_3(f_2(W_1x + b_1))$
$f_3(f_2(W_1x + b_1)) = f_3(W_2(W_1x + b_1) + b_2)$ $= f_3(W_2W_1x + W_2b_1 + b_2) = f_3(W'x + b')$	$f_3(f_2(W_1x + b_1)) = f_3\left(\frac{1}{1 + e^{-(W_2(W_1x+b_1)+b_2)}}\right)$ $= f_3\left(\frac{1}{1 + e^{-(W_2W_1x + W_2b_1 + b_2)}}\right) = f_3\left(\frac{1}{1 + e^{-(W'x + b')}}\right)$
$= f_3(W_2W_1x + W_2b_1 + b_2)$ $= W_3(W_2W_1x + W_2b_1 + b_2) + b_3$ $= W_3W_2W_1x + W_3W_2b_1 + W_3b_2 + b_3$ $= W''x + b''$	$f_3\left(\frac{1}{1 + e^{-(W_2W_1x + W_2b_1 + b_2)}}\right)$ $= W_3\left(\frac{1}{1 + e^{-(W_2W_1x + W_2b_1 + b_2)}}\right) + b_3 \neq W''x + b''$

2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (11/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 심층 신경망에서 비선형 레이어를 이용한 복잡도 유지(감소 방지)

    - 아래의 최종 수식을 보면 은닉층을 선형으로 쌓았을 경우, 결국 하나의 은닉층과 동일한 수식이 되는 것을 확인할 수 있다.

$$W_3W_2W_1x + W_3W_2b_1 + W_3b_2 + b_3 = W''x + b''$$

    - 하지만, 중간에 활성화 함수가 사용되어, 선형 간의 연결을 막는다면 하나의 은닉층으로 축소되는 것을 방지할 수 있다.

$$W_3\left(\frac{1}{1 + e^{-(W_2W_1x + W_2b_1 + b_2)}}\right) + b_3 \neq W''x + b''$$

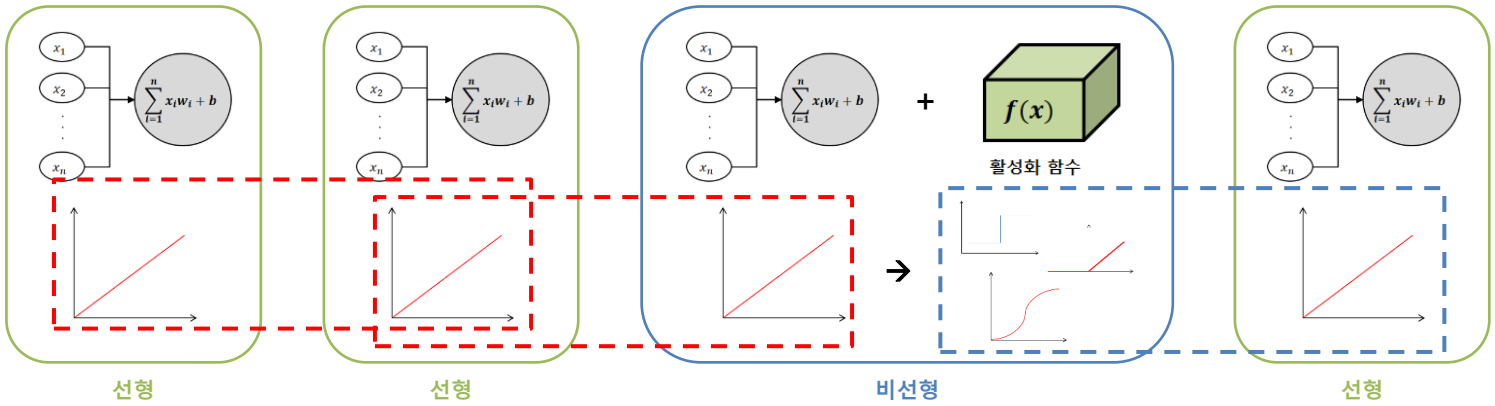
2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (12/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 심층 신경망에서 비선형 레이어를 이용한 복잡도 유지(감소 방지)

    - 자, 이제 비선형 레이어로 복잡도를 유지한다는 개념은 이해가 됐을 것이다.
      그렇다면, 은닉층에서 선형 간의 연결만 없다면, 선형 레이어를 사용하는 것은 문제가 없는가?

    - 그렇지 않다.
      이전에 계산한 수식에서 비선형 레이어로 복잡도가 유지되는 부분은 비선형 레이어와 그 다음 레이어 간의 연결이다.

    - 만약, 선형 다음에 비선형이 연결된다면, 연속적인 선형 연결과 동일하게 복잡도가 감소되는 것을 확인할 수 있다.
      왜냐하면, 비선형 레이어는 결국 선형 레이어와 활성화 함수의 결합이기 때문인데 아래와 같이 선형 간의 연결이 생겨버린다.



2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (13/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 심층 신경망에서 선형 레이어가 가지는 진짜 역할

    - 매개 변수의 수는 신경망의 복잡도를 결정하는 중요한 요인이다.
    - 비선형 레이어는 매개 변수의 수가 줄어들지 않도록 즉, 신경망의 복잡도를 유지하는 역할

    - 그렇다면, 선형 레이어의 역할은 무엇이고 언제(왜) 사용할까?

      선형 레이어는 "매개 변수의 수를 조절"하는 역할이라고 볼 수 있다.

    - 연속으로 쌓지 않은 하나의 선형 레이어는 연결된 비선형 레이어와의 매개 변수를 조절하는 역할로 정의할 수 있다.

    - 예를 들어, 자연어 입력을 벡터로 변환하는 임베딩 레이어는 활성화 함수를 사용하지 않는 선형 레이어이다.
      임베딩 레이어는 주어진 텍스트를 연결된 비선형 레이어의 입력 벡터(차원)으로 변환한다.

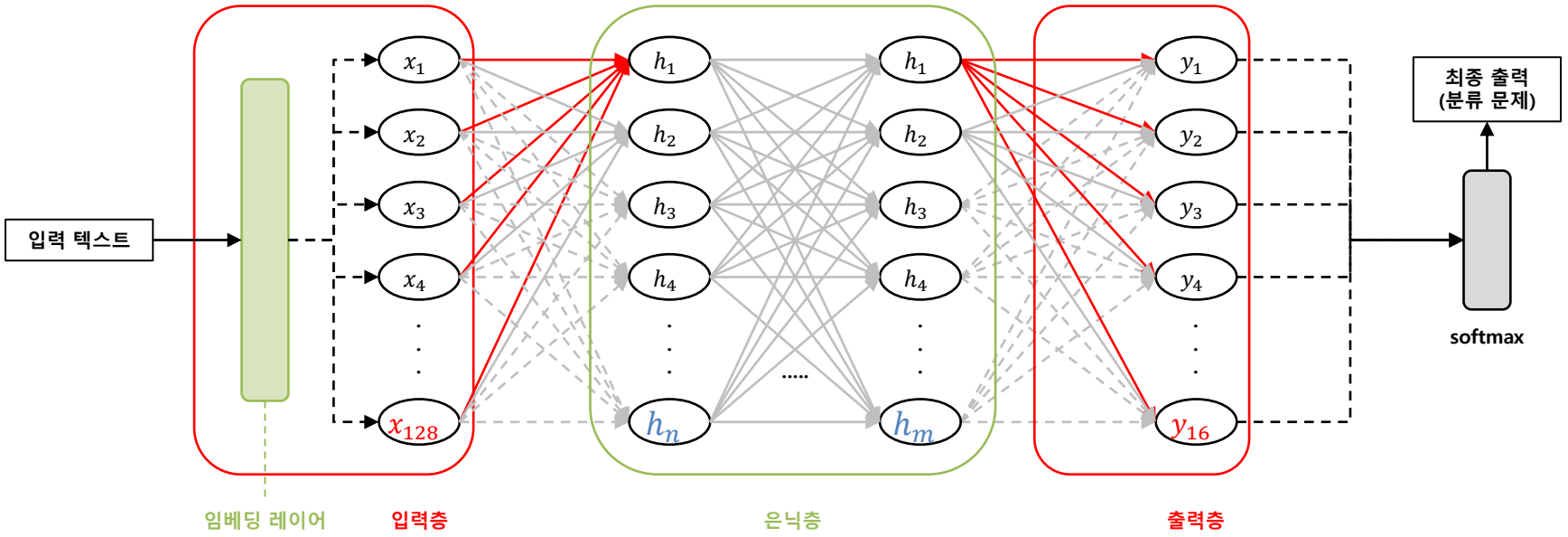
      (신경망의 복잡도를 높이기 위해, 은닉층의 입력 차원을 늘렸다면 임베딩 레이어는 은닉층의 입력 차원과 동일한 벡터를 생성하는 역할)
      (임베딩 레이어는 값을 정규화하지 않고, 단순히 n차원의 벡터로 표현한다는 뜻에서 "Representation"이라고 부르기도 한다.)

    - 선형 레이어는 레이어 간의 매개 변수의 수(차원)을 조절하는 역할로 정의할 수 있으며 입력층과 출력층의 차원을 조절하는 역할이다.

      즉, 심층 신경망에서 입력층과 출력층을 제외한 은닉층에선 비선형 레이어만을 사용한다.
      (은닉층에서 선형 레이어를 사용함으로써 발생하는 모든 복잡도 감소 현상은 물리적인 오류는 아니지만, 논리적인 오류에 해당하기 때문이다.)

2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (14/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 그렇다면, 심층 신경망에서 선형 레이어를 언제(왜) 사용하는가?
    - 선형 레이어는 입력층과 출력층의 차원(매개 변수)를 조절할 때 사용된다.
      신경망에서 이전 레이어의 출력 차원과 현재 레이어의 입력 차원은 동일해야 하기 때문이다.
    - 예를 들어, 첫 번째 은닉층의 입력 차원이 128 차원이고, 출력층의 출력 차원이 16 차원인 신경망의 구조는 다음과 같다.



2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (15/16)

- ③ 심층 신경망(DNN: Deep Neural Network)에서의 퍼셉트론과 활성화 함수
  - 은닉층에서 사용 가능한 레이어의 연결 구조

은닉층에서 레이어의 연결 구조	사용 가능 여부	설명
선형 – 선형 – 선형 선형 – 선형 – 비선형 비선형 – 선형 – 선형	X	선형과 선형이 연결되면, 하나의 선형을 학습하는 것과 동일하기 때문에, 심층 신경망에서 선형 간의 연결 구조는 사용하지 않는다.
선형 – 비선형 – 선형 선형 – 비선형 – 비선형 비선형 – 선형 – 비선형	X	선형과 비선형이 연결되면, 직접적인 선형 간의 연결 구조는 아니지만, 비선형은 선형과 활성화 함수의 결합이므로, 선형과 비선형의 연결은 결국 [선형 – 선형 – 활성화 함수]의 연결과 동일하며, 이는 성형이 연결된 형태로 볼 수 있다.
비선형 – 비선형 – 비선형	O	심층 신경망의 은닉층에선 비선형 간의 연결 구조만 사용할 수 있다.
비선형 – 비선형 – 선형	X	예외로 마지막 레이어가 선형 즉, 출력층이 선형인 경우에는 은닉층이 아니므로 사용 가능하지만, 은닉층의 마지막이 선형인 경우에는 사용하지 않는다.

2-2. 보는 시각에 따른 퍼셉트론과 활성화 함수의 정의 (16/16)

- 최종 정리
  - ① 초기의 인공 신경망, ② 현재의 인공 신경망, ③ 심층 신경망
    - 초기 인공 신경망과 현재의 인공 신경망의 차이는 오직 활성화 함수의 종류 뿐이다.
    - 인공 신경망과 심층 신경망에서 활성화 함수의 개념적 차이
      - 모든 신경망에서 활성화 함수의 역할은 목적에 맞는 출력 값으로 변환하는 것이다.
      - 인공 신경망의 경우, 네트워크의 목적 또는 레이어의 역할에 따라 활성화 함수를 선택 적용한다. (Sigmoid, ReLU, TanH 등)
      - 하지만, 심층 신경망에서 활성화 함수의 역할은 목적에 맞는 출력 값으로 변환시키는 개념보다, 선형을 비선형으로 바꾸면서 네트워크의 복잡도를 증가시켜 어려운 문제를 해결하는 개념이 더 크다.

① 초기의 인공 신경망	입력에 대한 가중 합을 계산하고 편향 값을 더한 뒤, 활성화 함수를 통해 목적에 맞는 출력 값으로 변환	**계단 함수를 사용하여, 0 또는 1의 출력 값을 반환하므로 이진 분류 문제를 해결하기에 적합**
② 현재의 인공 신경망		**다양한 활성화 함수의 등장으로 목적에 맞는 출력 값을 얻기 위해, 활성화 함수를 선택적으로 적용 (정규화 개념)**
③ 심층 신경망		**네트워크의 복잡도를 증가시키기 위해 선형을 비선형으로 변환**

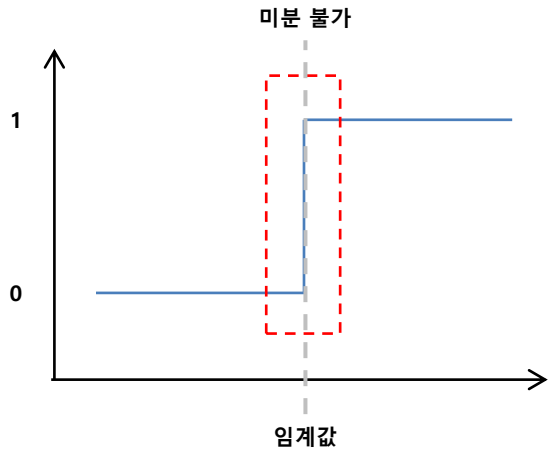
2-3. 활성화 함수의 종류와 특성 (1/9)

- 극성에 따른 분류
  - 단극성 활성화 함수
    - 출력 값이 양수만 가능한 함수
  - 양극성 활성화 함수
    - 출력 값이 음수와 양수 모두 가능한 함수

- 함수 형태에 따른 분류
  - 계단(step) 함수
  - Sigmoid, TanH
  - ReLU
  - ELU
  - Softmax
  - ...

2-3. 활성화 함수의 종류와 특성 (2/9)

- 함수 형태에 따른 분류
  - 계단 함수
    - 활성화 함수의 가장 간단한 형태, 임계값을 기준으로 활성 여부를 결정하며 "0" 또는 "1"의 값을 출력
      - 임계값보다 작거나 같으면 "0", 크면 "1"
      - "1"을 출력하는 경우, 뉴런이 활성화(activation) 되었다고 한다.
    - 신경망의 활성화 함수로 계단 함수를 사용한다면, 기울기가 무한대인 구간에서 미분이 불가능하여 가중치의 업데이트 과정에서 문제 발생
    - 따라서, 계단 함수는 신경망이 아닌 단일 퍼셉트론의 활성화 함수로만 사용된다.

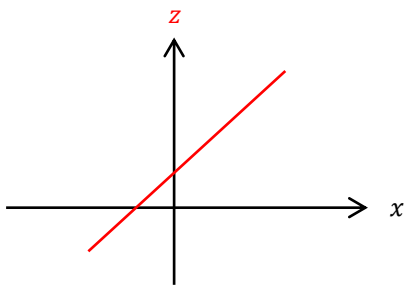


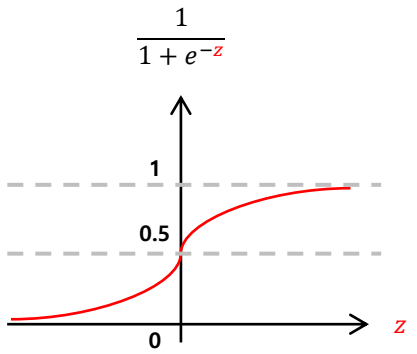
2-3. 활성화 함수의 종류와 특성 (3/9)

- 함수 형태에 따른 분류
  - Sigmoid
    - Sigmoid는 S자 형태라는 의미로, Sigmoid 함수는 S자형 곡선의 함수 방정식이다.
    - 입력은 실수 전체 구간이지만, 출력은 유한한 구간(0, 1) 사이의 한정된 값으로 반환된다.
    - 계단 함수와 마찬가지로, 임계값(0.5)를 기준으로 활성 여부를 결정한다면, 이진 분류 문제를 해결할 수도 있다.

$$\sum_{i=1}^{n} x_i w_i + b = z$$

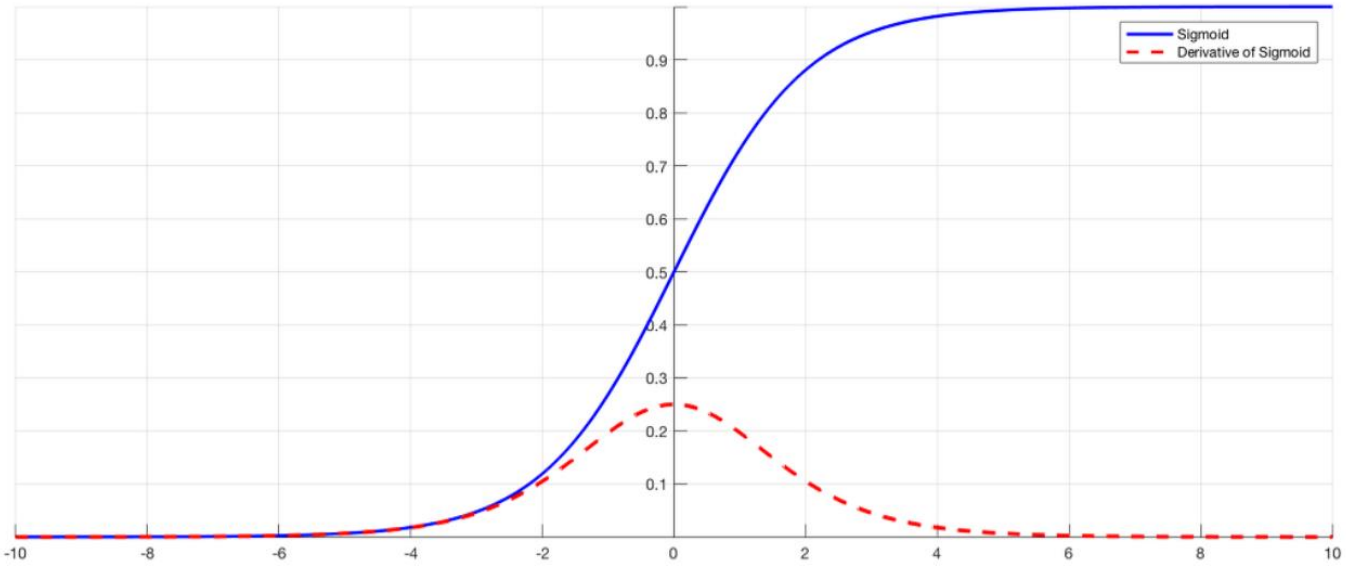
$$sigmoid(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(\sum x_i w_i + b)}}$$





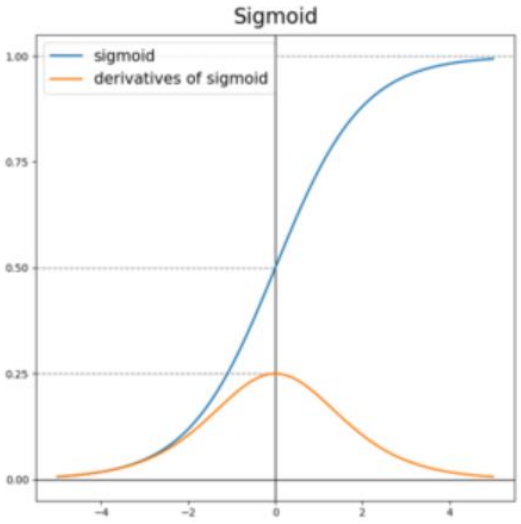
2-3. 활성화 함수의 종류와 특성 (4/9)

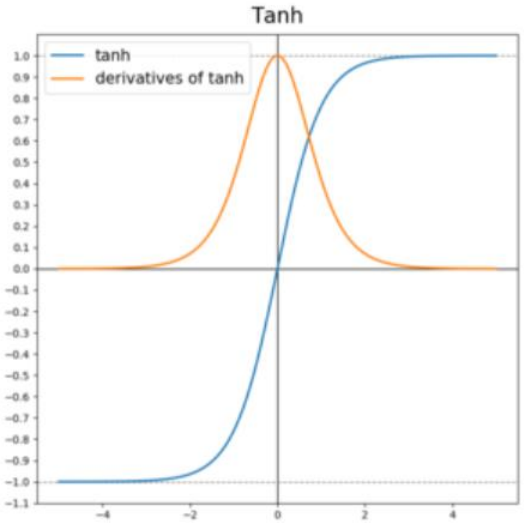
- 함수 형태에 따른 분류
  - Sigmoid
    - 단, 미분한 함수의 그래프를 보면 알 수 있듯이 정의역(입력)의 절대값이 커질 수록 미분 값은 "0"으로 수렴한다.
      때문에, 가중치가 업데이트되지 않고 소실되는 Vanishing gradient 문제가 발생할 수 있다.



2-3. 활성화 함수의 종류와 특성 (5/9)

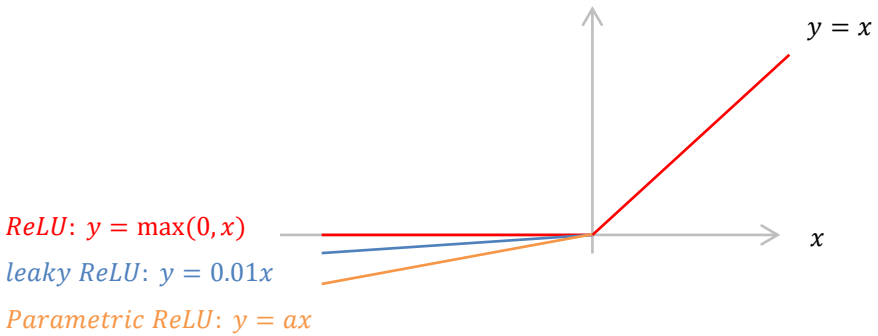
- 함수 형태에 따른 분류
  - TanH
    - Sigmoid 함수와 거의 유사하며, 차이는 (-1 ~ 1)의 범위를 가지며 데이터의 평균이 "0.5"가 아닌 "0"이라는 것 뿐이다.
    - 대부분의 경우에서 Sigmoid 보다 높은 성능을 보이지만, 마찬가지로 Vanishing gradient 문제가 발생할 수 있다.





2-3. 활성화 함수의 종류와 특성 (6/9)

- 함수 형태에 따른 분류
  - ReLU (Rectified Linear Unit)
    - 입력 값이 "0" 보다 작거나 같으면 "0", 양수이면 입력 값 그대로 출력
    - Sigmoid의 단점인 Vanishing gradient 문제를 해결할 수 있으며, 단순하지만 좋은 성능을 보여 자주 사용된다.
    - 그러나 가중합이 음수인 노드들은 다시 활성화되지 않는 dying ReLU(dead neuron) 현상이 발생할 수 있으며, 이를 보완하기 위해 "0" 이하의 값에서도 기울기를 가지는 함수들이 사용된다.

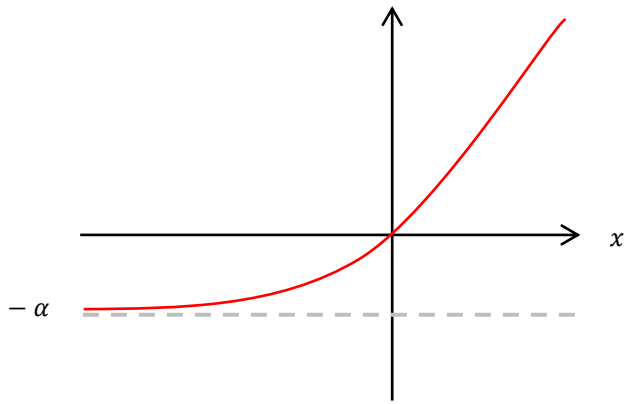


2-3. 활성화 함수의 종류와 특성 (7/9)

- 함수 형태에 따른 분류
  - ELU (Exponential Linear Unit)
    - ReLU와 유사한 형태이나 입력 값이 "0"인 지점이 sharp point가 아니기 때문에, 미분이 가능하며 "0" 이하의 모든 출력이 "0"이 아니라, 미분한 기울기가 "0"으로 수렴하는 형태의 함수이다.

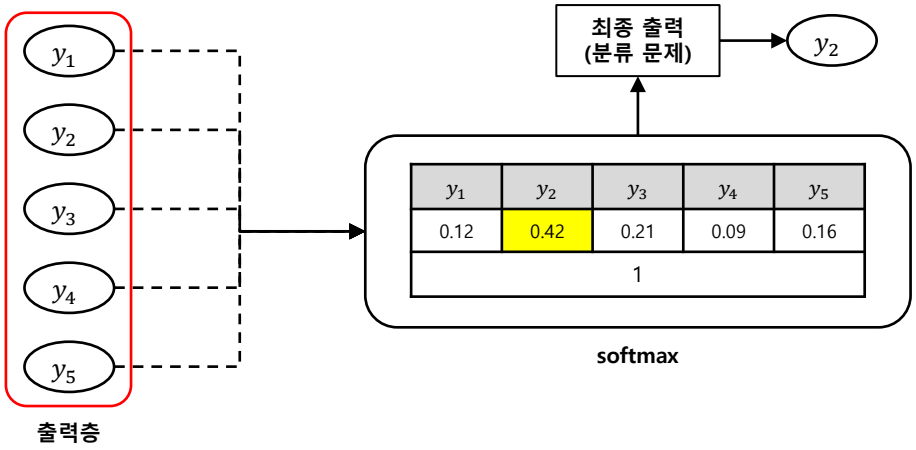
    - dying ReLU 현상을 해결하고 ReLU의 장점은 모두 가졌기 때문에, 딥러닝에서 자주 사용된다.
      (하지만, 선형 함수로만 구성된 ReLU에 비해서 지수 함수에 대한 추가 계산 비용이 발생한다.)

$$ELU: \quad y = \begin{cases} x & x \geq 0 \\ \alpha\,(\,e^{x} - 1\,) & x < 0 \end{cases}$$



2-3. 활성화 함수의 종류와 특성 (8/9)

- 함수 형태에 따른 분류
  - Softmax
    - 출력 값이 "0"과 "1" 사이의 값으로 정규화되며, 모든 출력 값의 총합이 항상 "1"이 되는 특성을 가진 함수
    - 주로 인공 신경망의 출력층에서 다중 클래스를 분류하는 목적으로 사용된다.
      여러 개의 클래스에 대한 예측 값을 총합이 "1"이 되도록 "0"과 "1"사이의 값으로 정규화하고, 이를 확률처럼 사용한다.



2-3. 활성화 함수의 종류와 특성 (9/9)

- 함수 형태에 따른 분류

	특성	장점	단점
계단 함수	임계값을 기준으로 "0" 또는 "1" 출력	이진 분류 가능	- 미분이 불가능한 부분 존재 - 이진 분류 외에는 사용할 수 없음 - 값의 크기에 대한 정보 소실
Sigmoid	출력값을 "0"과 "1" 사이의 값으로 정규화	이진 분류 가능 (임계값을 기준으로 분리)	Vanishing gradient 문제 발생
TanH	출력값을 "-1"과 "1" 사이의 값으로 정규화	- 이진 분류 가능 (임계값을 기준으로 분리) - Vanishing gradient 문제 개선 (Sigmoid와 비교해서)	Vanishing gradient 문제 발생
ReLU	$x \leq 0 \rightarrow y = 0$ $x > 0 \rightarrow y = x$ "은닉층에서 어떤 활성화 함수를 사용할지 모르겠으면 ReLU를 사용하면 된다."라는 말이 있을 정도로 일반적으로 높은 성능을 보인다.	대부분의 입력에 대해 기울기가 '0'이 아니기 때문에 빠른 학습이 가능해진다. (입력이 '0' 보다 작은 경우 기울기가 '0'이 되지만, 실제로 은닉층에서 대부분 노드의 z 값은 '0' 보다 크기 때문에 기울기가 '0'이 되는 경우는 많지 않다.)	- 입력이 '0' 보다 작은 경우에는 기울기가 '0'이 되고, 입력이 '0'인 경우에는 미분이 불가능하다. - Dying ReLU 현상 발생
ELU (leaky ReLU, PRELU)	ReLU와 유사한 형태이며, 입력이 음수인 경우에 기울기가 '0'이 되지 않게 한다.	입력이 '0' 보다 작은 경우에도 기울기가 '0'이 되지 않으므로, ReLU 보다 학습이 더 잘 되는 경우가 많다.	ReLU 보다 높은 계산 비용
Softmax	출력 값이 '0'과 '1' 사이의 값으로 정규화 되며, 모든 출력 값의 합은 '1'이다.	다중 클래스 분류 문제에 적합하다.	-