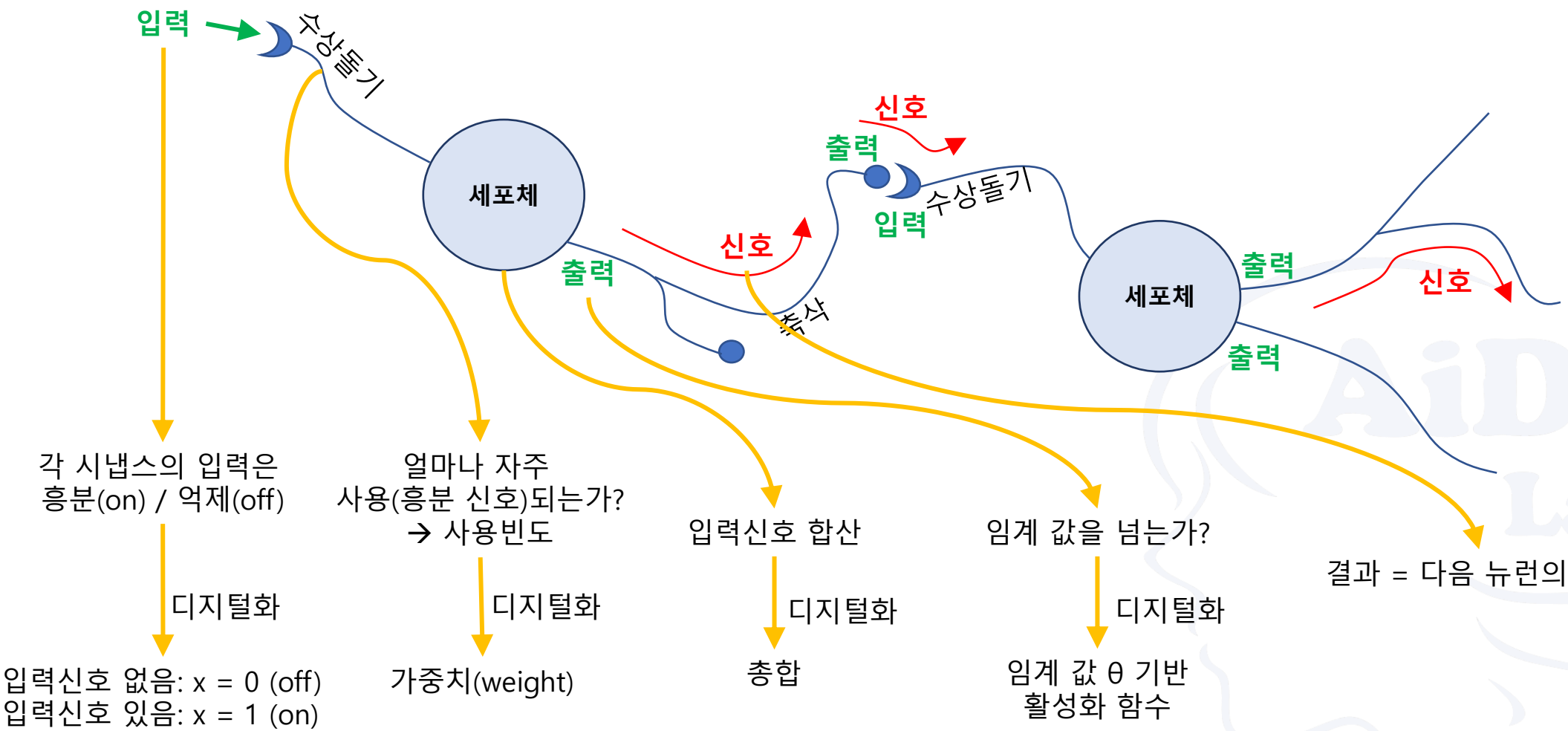
AiDA Lab.
Artificial Intelligence & Data Analysis Laboratory

수치미분

Seokhwan Yang

뉴런 동작 분석 → 퍼셉트론


입력
수상돌기
세포체
신호
출력
입력
수상돌기
출력
신호
시냅스
세포체
출력
신호
출력
각 시냅스의 입력은
흥분(on) / 억제(off)
얼마나 자주
사용(흥분 신호)되는가?
→ 사용빈도
입력신호 합산
임계 값을 넘는가?
결과 = 다음 뉴런의 입력
디지털화
디지털화
디지털화
디지털화
입력신호 없음: x = 0 (off)
입력신호 있음: x = 1 (on)
가중치(weight)
총합
임계 값 θ 기반
활성화 함수

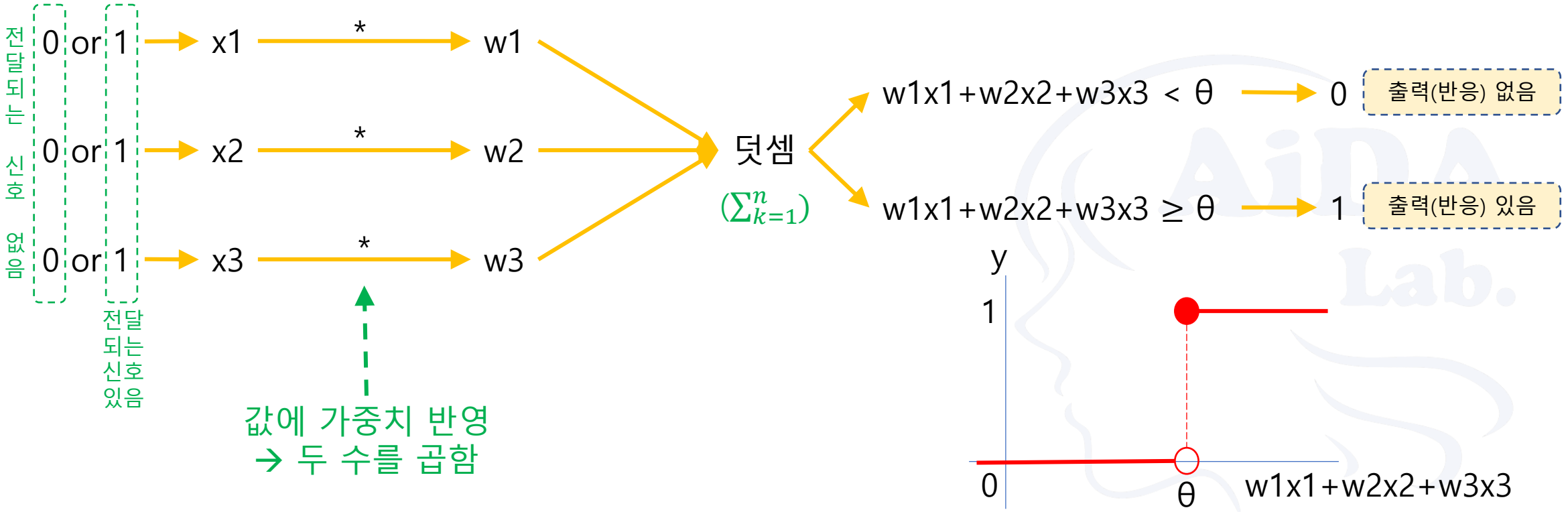
뉴런 동작 분석 → 퍼셉트론

입력신호 없음: x = 0 (off)
입력신호 있음: x = 1 (on)

가중치(weight)

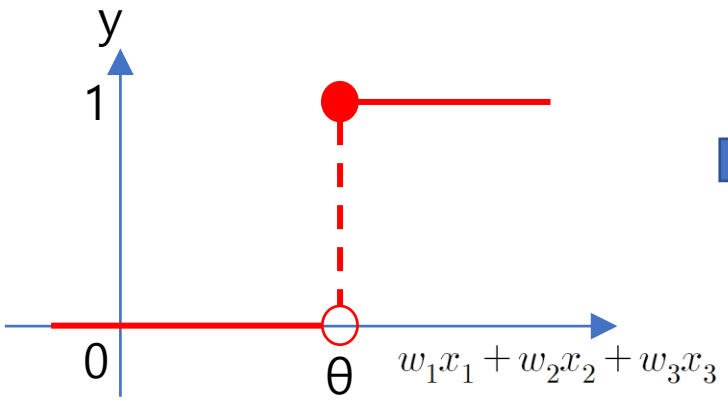
총합

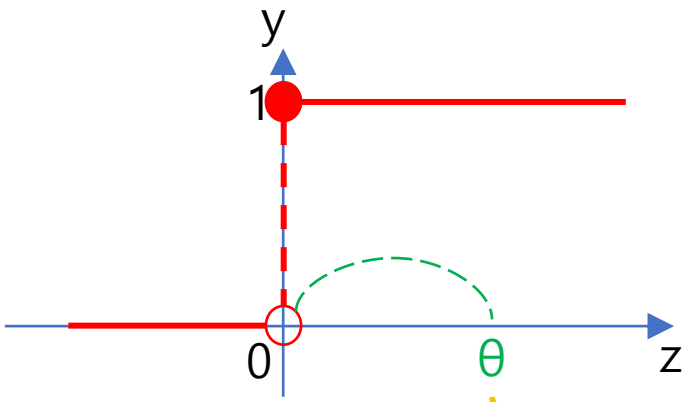
임계 값 θ 기반
활성화 함수



뉴런 뉴런 동작 분석 → 퍼셉트론 분석

- 계단 함수의 적용





$$y = u(z) \begin{cases} z < 0 : 0 \\ z \geq 0 : 1 \end{cases}$$

$$y = u(z) = u(w_1x_1 + w_2x_2 + w_3x_3 - \theta)$$

y	w1x2+w2x2+w3x3	z=w1x2+w2x2+w3x3-θ	u(z)
0 (반응하지 않음)	θ보다 작음	z < 0	0
1 (반응함)	θ보다 큼	z ≥ 0	1

(z : 뉴런의 가중 입력)

뉴런 동작 분석 → 퍼셉트론

- **활성화 함수**

$$y = u(z) = u(w_1x_1 + w_2x_2 + w_3x_3 - \theta)$$

활성화 함수

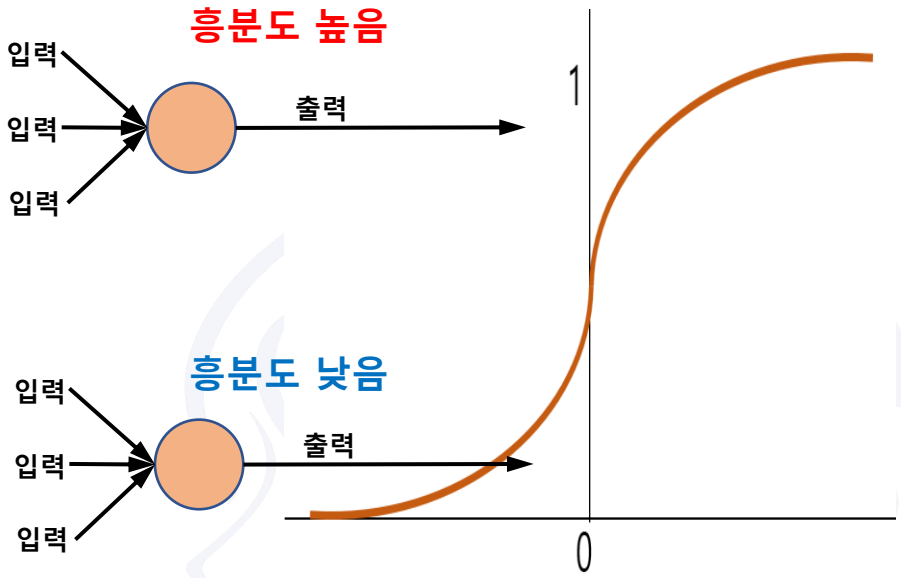
생물의 신경세포에 대한 모델링에서는 입/출력은 0, 1뿐이지만
신경망 모델에서는 **입/출력이 임의의 수가 될 수 있음**

	뉴런	신경망 모델의 유닛(퍼셉트론)
활성화 함수	단위 계단 함수	사용자 정의 가능 → 시그모이드 함수가 많이 사용됨
출력 값 y	0 또는 1	활성화 함수를 사용할 수 있는 임의의 수
출력 신호 해석	반응 여부	유닛(퍼셉트론)의 흥분도, 반응도, 활성도

뉴런 동작 분석 → 퍼셉트론

- 시그모이드 함수

$$\sigma(z) = \frac{1}{1+e^{-z}}, (e = 2.718281\cdots)$$



  - 0보다 크고 1보다 작은 값 출력
  - 유닛(퍼셉트론)의 흥분도, 반응도 출력
    - 출력 값이 1에 가까우면 흥분도가 높음
    - 출력 값이 0에 가까우면 흥분도가 낮음
  - 시그모이드 외의 미분가능한 단조증가 함수도 원리는 동일함

뉴런 동작 분석 → 퍼셉트론

- **편향 값(bias)**

$$y = u(z) = u(w_1x_1 + w_2x_2 + w_3x_3 - \theta)$$

편향 값

  - **자연 현상에서는 임계 값(편향 값으로 적용됨)이 음수인 경우가 거의 없지만**
  - **퍼셉트론 모델에서는 음수도 허용됨**
  - **예시**
    - 입력 x1의 가중치는 2, x2의 가중치는 3, 편향 값은 -1인 경우

$$y = u(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+2.7^{0.3}} \fallingdotseq 0.43$$

입력 x1	입력 x2	가중입력 z	출력 y
0.2	0.1	2 x 0.2 + 3 x 0.1 - 1 = -0.3	1/(1+2.7^-(-0.3))=1/2.35=0.43
0.6	0.5	2 x 0.6 + 3 x 0.5 – 1 = 1.7	1/(1+2.7^-1.7)=1/1.18=0.84

(e=2.7로 계산함)

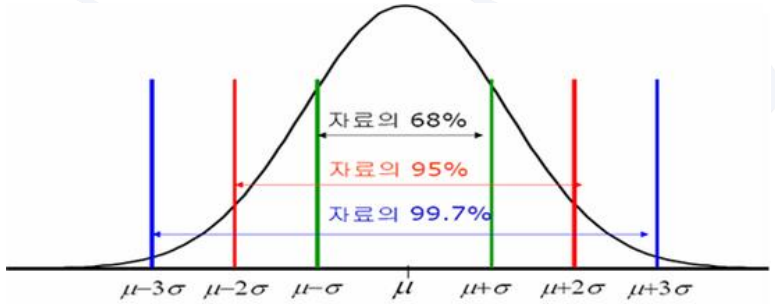
$$y = u(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+2.7^{-1.7}} \fallingdotseq 0.84$$

신경망 초기화

- **신경망 설정**
  - **가중치 및 편향 값의 초기화**
    - 임의의 값으로 초기화(난수 사용)하는 것이 기본 원칙
    - 초기값을 결정할 때는 가능하면 정규분포 난수를 사용함

      → 원인은 아직 정확하지 않으나, 정규분포 난수를 사용할 경우 신경망 학습 결과가 대체로 좋게 나옴
    - 정규분포: 아래의 확률밀도함수 f(x)를 따르는 확률분포

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$



신경망을 위한 미분

- **수열과 신경망**
  - **수열: 숫자의 열. n번째 수는 일반적으로 a_n 으로 표시**
  - **수열의 일반항 표시**
    - 일반항: 주어진 수열의 n번째 변수를 n을 이용한 식으로 나타낸 것
    - 예: $a = 2, 4, 6, 8, 10, \cdots \Rightarrow a_n = 2n$
      $a = 1, 3, 5, 7, 9, \cdots \Rightarrow a_n = 2n - 1$

신경망을 위한 미분

- **신경망에서는 유닛(퍼셉트론)의 가중 입력과 출력을 수열로 간주함**
  - "몇 번째 층의 몇 번째 수는 몇 개"와 같이 순서로 값이 결정되므로
  - 신경망에서는 아래와 같이 표현

a_j^l : l층 j번째 유닛(퍼셉트론)의 출력 값

신경망을 위한 미분

- 점화식
  - 수열을 표시하는 일반항 이외의 표현법
  - 1항 a_1 과 인접한 2개의 항 a_n , a_{n+1} 의 관계식으로 수열 $\{a_n\}$ 표시
  - 예

$$a_1 = 1$$

$$a_2 = a_{1+1} = a_1 + 2 = 1 + 2$$

$$a_3 = a_{2+1} = a_2 + 2 = 3$$

신경망을 위한 미분

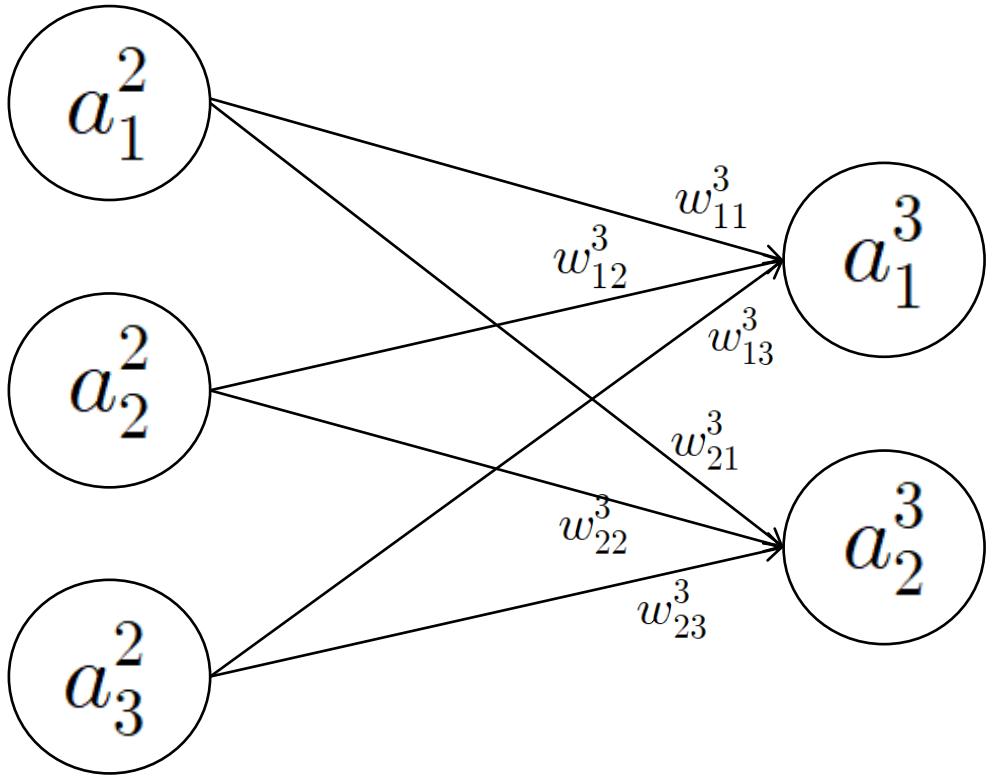
- 연립 점화식
  - 여러 수열이 몇 가지 관계식으로 연결된 것
  - 예: $a_1 = 1, b_1 = 1$ 이고 $\begin{cases} a_{n+1} = a_n + 2b_n + 2 \\ b_{n+1} = 2a_n + 3b_n + 1 \end{cases}$ 이면,

$$\begin{cases} a_2 = a_1 + 2b_1 + 2 = 1 + 2 \times 1 + 2 = 5 \\ b_2 = 2a_1 + 3b_1 + 1 = 2 \times 1 + 3 \times 1 + 1 = 6 \end{cases}$$

$$\begin{cases} a_3 = a_2 + 2b_2 + 2 = 5 + 2 \times 6 + 2 = 19 \\ b_3 = 2a_2 + 3b_2 + 1 = 2 \times 5 + 3 \times 6 + 1 = 29 \end{cases}$$

신경망을 위한 미분

- 신경망에서는 모든 유닛(퍼셉트론)의 입력과 출력이 연립 점화식으로 연결되어 있다고 생각함



$$a_1^3 = a(w_{11}^3 a_1^2 + w_{12}^3 a_2^2 + w_{13}^3 a_3^2 + b_1^3)$$

$$a_2^3 = a(w_{21}^3 a_1^2 + w_{22}^3 a_2^2 + w_{23}^3 a_3^2 + b_2^3)$$

신경망을 위한 미분

- n!을 계산할 경우
  - 사람은 $1 \times 2 \times 3 \times \cdots \times n = n!$ 와 같이 계산하지만
  - 컴퓨터 알고리즘에서는

$$a_1 = 1,\ a_{n+1} = (n+1)a_n$$

와 같이 계산하도록 구현되어 있으므로 점화식 계산이 유리함

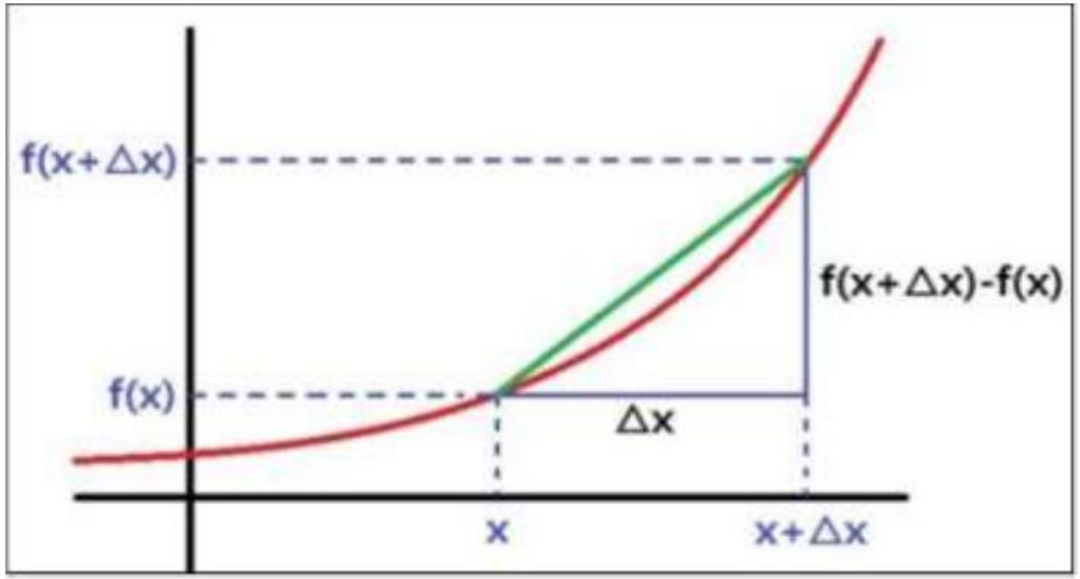
  - 신경망의 오차역전파 알고리즘에서도 이러한 방식이 유용함

신경망을 위한 미분

- 미분(Derivative)
  - 미분은 순간 변화율이다
  - 미분은 한 점에서의 접선의 기울기이다
  - 그런데 왜 미분을 사용해야 하는가?

신경망을 위한 미분

- 미분
  - 가로 축의 Δx 는 입력 값 x 의 변화량
  - Δx는 미분 공식의 분모에 해당
  - f(x)는 x에 대한 함수 값
  - f(x + Δx)– f(x) 는 함수 값의 변화량
  - f(x + Δx)– f(x) 는 미분 공식에서 분자(numerator)에 해당



$$f'(x) = \frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

  - 그래프는 입력 변수 x 가 변할 때 출력 변수 f(x)는 얼마나 변하는가를 나타냄

신경망을 위한 미분

- 미분 공식

$$f'(x) = \frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

  - 극한(limit)을 통해서 Δx 는 0에 근접함 → 입력 값 x 의 변화량은 거의 없음 (변화가 있더라도 무시할 수 있을 만큼 작음)

  - 미분 공식에서 극한 부분과 분모, 분자부분을 재정의해서 미분 개념을 해석해 보면?

신경망을 위한 미분

- 함수 f(x) 를 미분 한다는 것은
  - 입력 변수 x 가 (아주 미세하게) 변할 때, 함수 f(x) 는 얼마나 변하는지 알 수 있는 식을 구하라!!
  - 함수 f(x)가 입력 값 x 의 미세한 변화에 얼마나 민감하게 반응하는지 확인 하라!!

→ 미분을 통해서 얻을 수 있는 것 (Insight)

신경망을 위한 미분

- 머신러닝의 최종 목표
  - 미분을 이용하여 가중치(weight)와 바이어스(bias)를 점진적으로 계산하면서 최적의 값을 찾는 것
  - 머신러닝에서 자주 사용되는 미분공식들

기본함수 f(x)	미분함수 f′(x)	미분 예시
$f(x) = 상수$	$f'(x) = 0$	$f(x) = 3x^2 + e^x + 7 \Rightarrow f'(x) = 6x + e^x$ $f(x) = \ln x + \frac{1}{x} \Rightarrow f'(x) = \frac{1}{x} - \frac{1}{x^2}$
$f(x) = ax^n$	$f'(x) = anx^{(n-1)}$	
$f(x) = e^x$	$f'(x) = e^x$	
$f(x) = \ln x$	$f'(x) = \frac{1}{x}$	

신경망을 위한 미분

- 편미분
  - 입력 변수가 1 개 이상인 다변수 함수에서
  - 미분하고자 하는 변수 1 개를 제외한 나머지 변수들은 상수로 취급하고
  - 특정한 1 개의 변수에 대해서만 미분 하는 것

신경망을 위한 미분

- **편미분 예시**
  - $$\frac{\partial f(x,y)}{\partial x} = \frac{\partial(2x + 3xy + y^3)}{\partial x} = 2 + 3y$$

    f(x, y) = 2x + 3xy + y^3, 변수 x 에 대한 f(x, y) 편미분

  - $$\frac{\partial f(x,y)}{\partial y} = \frac{\partial(2x + 3xy + y^3)}{\partial y} = 3x + 3y^2$$

    f(x, y) = 2x + 3xy + y^3, 변수 y 에 대한 f(x, y) 편미분

신경망을 위한 미분

- **편미분은 실제 우리 생활에서 어떻게 쓰이고 있나?**
  - **체중에 대한 운동과 야식의 편미분**

체중 함수가 '체중(야식, 운동)' 처럼 야식/운동에 영향을 받는 2 변수 함수로 가정 할 경우, 편미분을 이용하면 각 변수 변화에 따른 체중 변화량을 구할 수 있음

현재 먹는 야식의 양에서 조금 변화를 줄 경우 체중은 얼마나 변하는가 ? $\Rightarrow \frac{\partial 체중}{\partial 야식}$	현재 하고있는 운동량에 조금 변화를 줄 경우 체중은 얼마나 변하는가? $\Rightarrow \frac{\partial 체중}{\partial 운동}$

- 현재 먹는 야식의 양에서 조금 더 먹거나(+) 또는 덜 먹는(-) 등의 야식 양에 변화를 줄 경우 체중이 얼마나 변하는지 알고 싶다면 편미분을 이용!!

신경망을 위한 미분

- 편미분은 머신러닝 외에도 기상관측, 지진, 일기예보 등 다양한 분야에서 활용되고 있다.

신경망을 위한 미분

- 체인 룰(Chain Rule)
  - 합성 함수를 미분하기 위해 사용되는 방식
  - 합성 함수: 여러 개의 함수로 구성되어 있는 함수
  - 합성 함수의 예

[합성함수 예 1]	$f(x) = e^{3x^2}$	⇒	함수 e^t ,	함수	$t = 3x^2$ 조합
[합성함수 예 2]	$f(x) = e^{-x}$	⇒	함수 e^t ,	함수	$t = -x$ 조합

  - 여러 개의 함수가 조합된 합성 함수를 미분하려면 각 함수를 연쇄적으로 미분해 나가야 하므로 그를 위한 체인 룰이 필요함

신경망을 위한 미분

- 체인 룰의 적용 방식

$f(x) = e^{3x^2}$ 을 chain rule 로 미분하는 경우, $t = 3x^2$ 으로 놓으면 $f(x) = e^t$

chain rule 적용(약분 개념)

$t = 3x^2$ 대입

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial t}\frac{\partial t}{\partial x} = \frac{\partial (e^t)}{\partial x}\frac{\partial (3x^2)}{\partial x} = (e^t)(6x) = \left(e^{3x^2}\right)(6x) = 6xe^{3x^2}$$

- 합성된 두 함수에서
- 상대적으로 미분이 쉬운 기본함수로 나타내기 위해 $t = 3x^2$으로 치환하면 f(x)는 e^t 함수와 $3x^2$이라는 두 개의 함수가 합쳐져 있는 형태가 됨

신경망을 위한 미분

- 최종 목표는 $\frac{\partial f(x)}{\partial x}$를 구하는 것이지만
- 앞의 예제와 같이 약분을 통해 식을 변형해 줄 수 있다.

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\cancel{\partial t}} \frac{\cancel{\partial t}}{\partial x}$$ (보이지 않는 요소인 t를 추가해 줌)

- 이와 같이 분모와 분자에 동일한 값(∂t)을 곱해 주어 각각의 개별적인 변수에 대한 미분의 곱으로 나타내는 방식을 "체인 룰을 적용한다"라고 함
- 체인 룰을 적용하면 첫 번째 부분인 $\frac{\partial f(t)}{\partial t}$은 $\frac{\partial e^t}{\partial t}$ 으로 나타낼 수 있고
  두 번째 항인 $\frac{\partial t}{\partial x}$ 부분은 $\frac{\partial (3x^2)}{\partial x}$을 적용하여 각각의 변수에 대해 계산 가능

신경망을 위한 미분

- 즉, $\frac{\partial f(x)}{\partial x} = \frac{\partial f(t)}{\partial t} \cdot \frac{\partial t}{\partial x} = e^t \cdot 6x$

- 그런데 최종 목표는 x에 대하여 f를 미분하는 것이므로, t를 다시 x의 형태로 바꿔주면 ($t = 3x^2$ 로 치환)

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f(t)}{\partial t} \cdot \frac{\partial t}{\partial x} = e^t \cdot 6x \Rightarrow \frac{\partial f(x)}{\partial x} = \frac{\partial f(t)}{\partial t} \cdot \frac{\partial t}{\partial x} = e^t \cdot 6x = e^{3x^2} \cdot 6x$$

신경망을 위한 미분

- **체인 룰은**
  - **합성 함수를 미분 할 때**
    ① 분모와 분자에 동일한 변수 값 (예제에서는 ∂t) 을 곱해 주어 개별적인 항의 곱으로 분리 한 후에
    ② 각각의 항에 대해 미분을 수행하는 것을 의미
  - **체인 룰(chain rule)을 이용하는 방식은 차후에 알아볼 딥러닝의 꽃으로 불리는 오차역전파(Back Propagation)를 구현 할 때 반드시 필요한 기법**

신경망을 위한 미분

- 수치 미분 (Numerical Derivative)
  - C 언어나 파이썬 등의 프로그래밍 언어를 이용하여 미분 값, 즉 입력 값이 아주 미세하게 변할 때 함수 f 는 얼마나 변하는지를 계산해 주는 것

신경망을 위한 미분

- 입력 변수가 1 개인 간단한 함수의 미분을 파이썬으로 구현해보자

① 미분 하려는 함수 f(x) 정의

② 극한 개념을 구현하기 위해
Δx 는 작은 값으로 설정

③ 분자 / 분모 구현

$$f'(x) = \frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \fallingdotseq \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x}$$

수치해석 오차를 줄이기 위해 일반적으로
분모 / 분자를 위와 같이 변형함

1변수 함수에 대한 미분 공식

신경망을 위한 미분

- **3단계의 과정을 거쳐 미분을 정의함**
  - ① 미분하고자 하는 함수를 정의
  - ② 극한 개념을 구하기 위해 Δx를 작은 값으로 설정
  - ③ 정의된 함수를 이용하여 분자와 분모 구현

```
import numpy as np

# 미분 공식 구현
def simple_derivative(f, var): # ① f 는 외부에서 lambda 로 정의
    delta = 1e-5 # ② 극한 구현
    diff_val = ( f(var+delta) - f(var-delta) ) / (2*delta) # ③ 미분 값 계산
    return diff_val

# 미분대상 함수
def func1(x):
    return x**2

# ① lambda function 을 이용하여 함수 f 로 정의
f = lambda x : func1(x)
ret_val = simple_derivative(f, 3.0)
print(ret_val)
```

미분 공식을 그대로 적용한 예제

신경망을 위한 미분

- 다변수 함수에 대한 수치 미분을 파이썬으로 구현해 보자
  - 실무에서는 입력 변수가 하나 이상인 다변수 함수가 일반적이다
  - 예시

$f(x, y) = 2x + 3xy + y^3$ 일때,

입력 변수 x, y 는 두 개 이므로 $\frac{\partial y}{\partial x}$, $\frac{\partial y}{\partial x}$ 각각 수치미분 수행

$f'(1.0,\ 2.0)$ 값을 계산하기 위해서는,

$\Rightarrow$ x = 1.0 에서의 미분계수는 변수 y = 2.0 을 상수로 대입하여 $\frac{\partial f(x,\ 2.0)}{\partial x}$ 수행

$\Rightarrow$ y = 2.0 에서의 미분계수는 변수 x = 1.0 을 상수로 대입하여 $\frac{\partial f(1.0,\ y)}{\partial y}$ 수행

신경망을 위한 미분

```python
import numpy as np

def derivative(f, var):
    if var.ndim == 1: # ① vector
        temp_var = var # ② 원본 값 저장
        delta = 1e-5
        diff_val = np.zeros(var.shape) # ③ 미분 계수 값 보관 변수
        for index in range(len(var)): # ④ 벡터의 모든 열(column) 반복
            target_var = float(temp_var[index])
            temp_var[index] = target_var + delta
            func_val_plust_delta = f(temp_var) # x+delta 에 대한 함수 값 계산
            temp_var[index] = target_var - delta
            func_val_minus_delta = f(temp_var) # x-delta 에 대한 함수 값 계산
            # ⑤ 미분 계수 ( f(x+Δx) - f(x-Δx) ) / (2Δx) 계산
            diff_val[index] = (func_val_plust_delta - func_val_minus_delta) / (2*delta)
            temp_var[index] = target_var
        return diff_val
```

신경망을 위한 미분

```
    elif var.ndim == 2: # ① matrix
        temp_var = var # ② 원본 값 저장
        delta = 1e-5

        diff_val = np.zeros(var.shape) # ③ 미분 계수 값 보관 변수
        rows = var.shape[0]
        columns = var.shape[1]

        for row in range(rows): # ④ 행렬의 모든 행(row)과 열(column) 반복
            for column in range(columns): # ④ 행렬의 모든 행(row)과 열(column) 반복
                target_var = float(temp_var[row,column])
                temp_var[row,column] = target_var + delta
                func_val_plus_delta = f(temp_var) # x+delta 에 대한 함수 값 계산
                temp_var[row,column] = target_var - delta
                func_val_minus_delta = f(temp_var) # x-delta 에 대한 함수 값 계산
                # ⑤ 미분 계수 ( f(x+Δx) - f(x-Δx) ) / (2Δx) 계산
                diff_val[row,column] = (func_val_plus_delta - func_val_minus_delta) / (2*delta)
                temp_var[row,column] = target_var
        return diff_val
```

신경망을 위한 미분

```
import numpy as np

def func2(W): # ①
    x = W[0]
    y = W[1]
    return ( 2*x + 3*x*y + np.power(y,3) )

f = lambda W : func2(W) # ②
ret_val = derivative( f, np.array([1.0, 2.0]) ) # ③
print(ret_val)
```