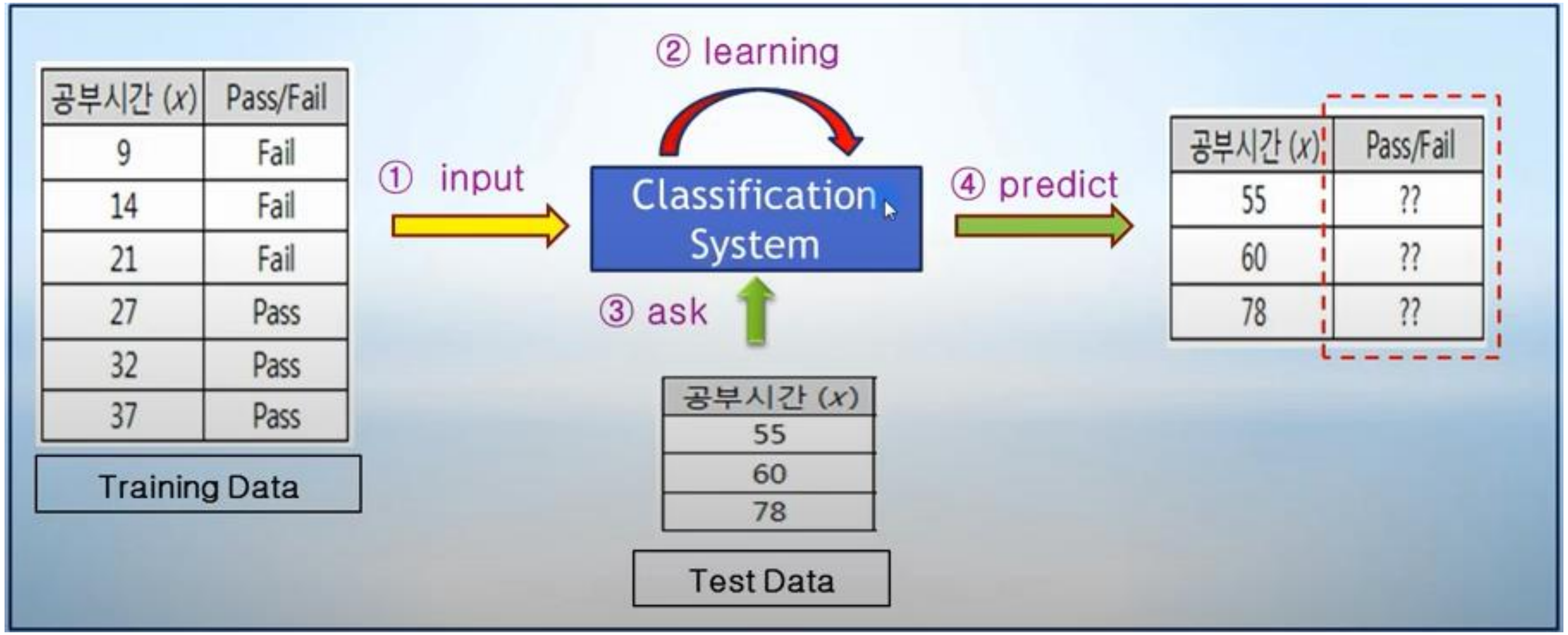
4. 지도학습-2

Classification

Classification – Logistic Regression

❖ 분류(Classification)

- Training Data 특성과 관계 등을 파악한 후에, 미지의 입력 데이터에 대해서 결과가 어떤 종류의 값으로 분류될 수 있는지를 예측하는 것
- 예 : 스팸문자 분류[Spam(1) or Ham(0)], 암 판별[악성종양(1) or 종양(0)]



Classification

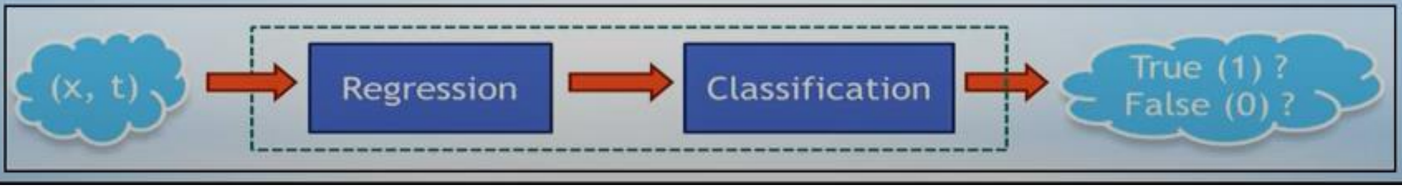
❖ Logistic Regression algorithm Flow

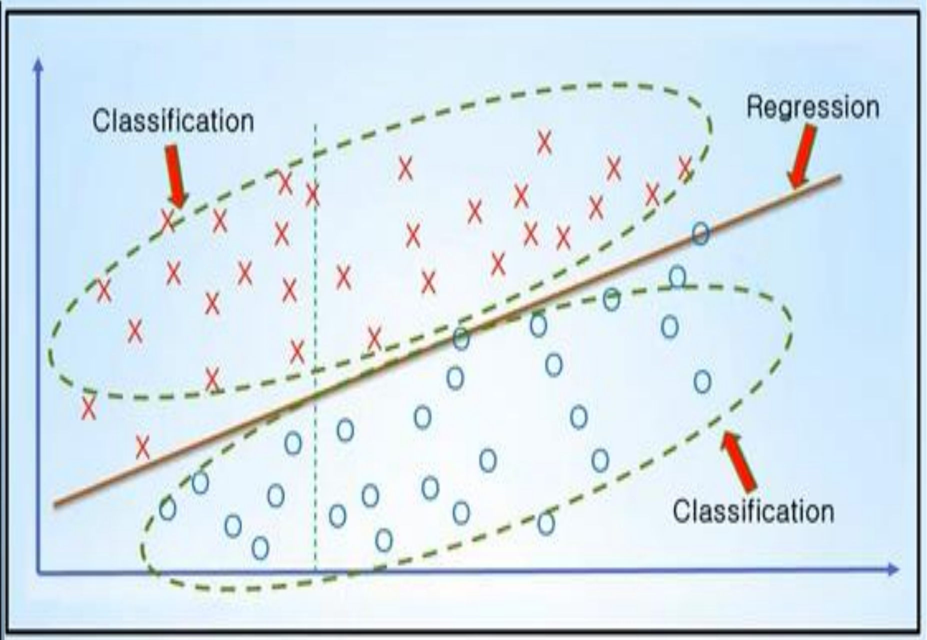
➤ 즉, Logistic Regression 알고리즘은,

① Training Data 특성과 분포를 나타내는 최적의 직선을 찾고(Linear Regression)

② 그 직선을 기준으로 데이터를 위(1) 또는 아래(0) 등으로 분류(Classification) 해주는 알고리즘

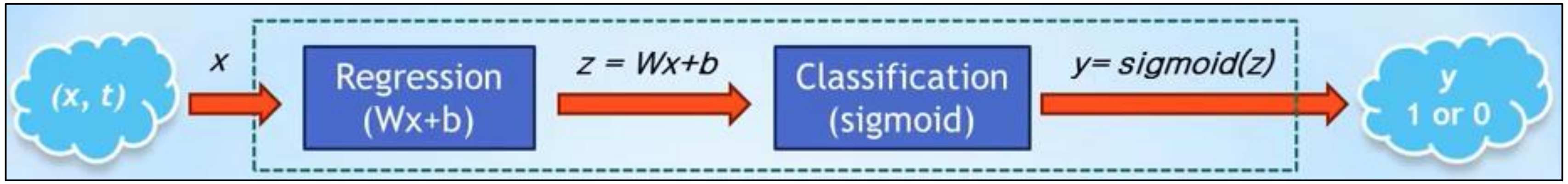
⇒ 이러한 Logistic Regression은 Classification 알고리즘 중에서도 정확도가 높은 알고리즘으로 알려져 있어서 Deep Learning에서 기본 Component로 사용되고 있음





Classification

❖ Logistic Regression algorithm Flow



➢ 출력 값 y 가 1 또는 0 만을 가져야만 하는 분류(classification) 시스템에서, 함수 값으로 0~1 사이의 값을 가지는 sigmoid 함수를 사용 할 수 있음

⇒ 즉, linear regression 출력 Wx+b 가 어떤 값을 갖더라도, 출력 함수로 sigmoid 를 사용해서 ① sigmoid 계산 값이 0.5보다 크면 결과로 1 이 나올 확률이 높다는 것이기 때문에 출력 값 y 는 1 을 정의하고 ② sigmoid 계산 값이 0.5 미만이면 결과로 0 이 나올 확률이 높다는 것이므로 출력 값 y는 0 정의하여 classification 시스템을 구현할 수 있음

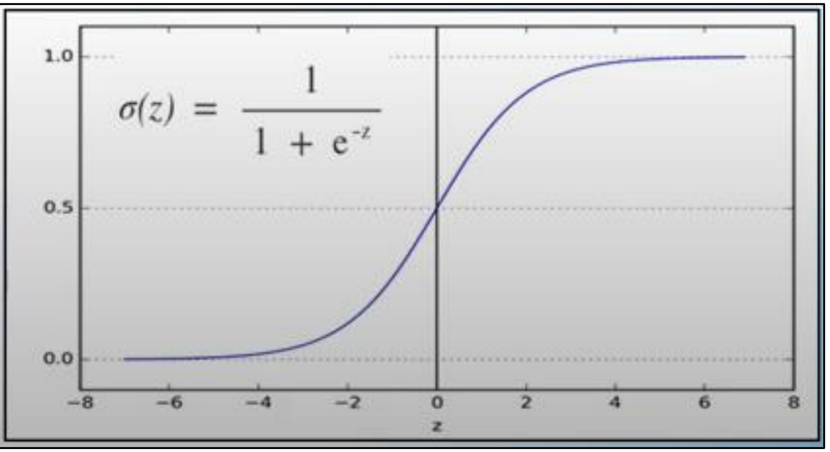
※ sigmoid 함수의 실제 계산 값 sigmoid(z)는 결과가 나타날 확률을 의미함

$$y = \frac{1}{1 + e^{-(Wx+b)}}$$

또는

$$z = Wx + b$$

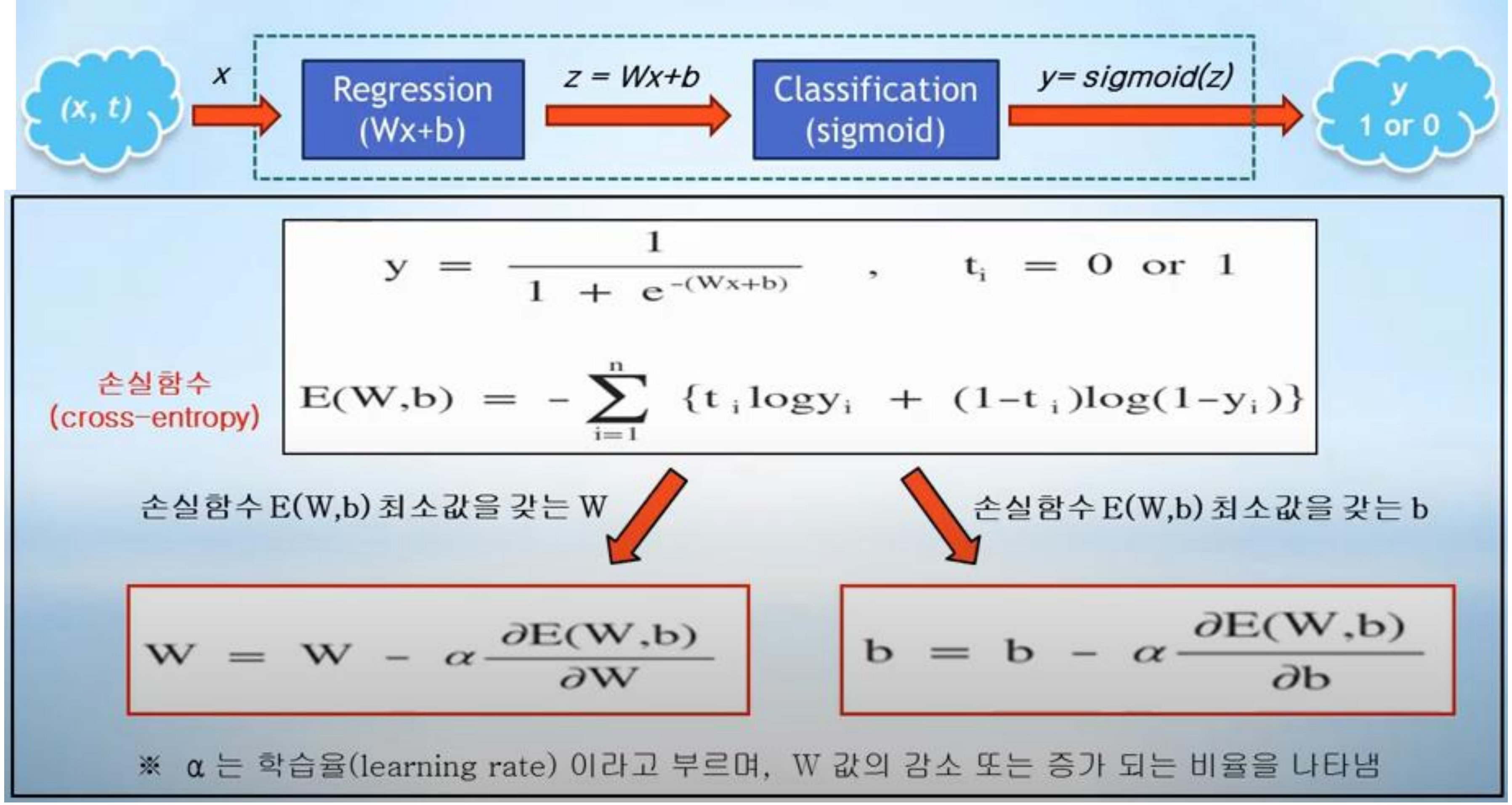
$$y = \text{sigmoid}(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Classification

❖ 손실함수(loss function), W, b

- 분류 시스템(classification) 최종 출력 값 y는 sigmoid 함수에 의해 논리적으로 1 또는 0 값을 가지기 때문에, 연속 값을 갖는 선형회귀 때와는 다른 손실함수 필요함



Classification

❖ 손실함수(loss function) cross-entropy 유도

- Classification 최종 출력 값 y는 sigmoid 함수에 의해 0~1 사이의 값을 갖는 확률적인 분류 모델이므로, 다음과 같이 확률변수 C를 이용해 출력 값을 나타냄

수식	수식 설명
$p(C=1\|x) = y = \text{sigmoid}(Wx+b)$	입력 x 에 대해 출력 값이 1 일 확률을 y 로 정의. y 는 1 또는 0 이므로 y=sigmoid(Wx+b) 나타낼 수 있음
$p(C=0\|x) = 1 - p(C=1\|x) = 1 - y$	입력 x 에 대해 출력 값이 0 일 확률이며, 확률은 모두 더한 것이 1 이므로, 출력 값이 0일 확률은 1 - y 임 (y 는 출력 값이 1일 확률로, sigmoid(Wx+b) 로 정의함)
$p(C=t\|x) = y^{t}(1-y)^{1-t}$	확률변수 C 는 0 이나 1 밖에는 값을 가질 수 없으므로, (즉, 정답 t = 0 or 1) 다음처럼 나타냄
우도함수란, 입력 x 에 대해 정답 t 가 발생될 확률을 나타낸 함수 확률은 독립적이므로, 각 입력 데이터의 발생 확률을 곱해서 우도함수 나타냄 $L(W,b) = \prod_{i=1}^{n} p(C=t_i\|x_i) = \prod_{i=1}^{n} y_i^{t_i}(1-y_i)^{1-t_i}$	가중치 W와 바이어스 b를 최우추정하기 위한 우도함수 (likelihood function)은 다음과 같이 나타낼 수 있으며, 이 우도함수값이 최대가 되도록(최우추정) W와 b 를 업데이트 해 나가면 머신러닝에서 학습이 잘 된것임.
$E(W,b) = -\log L(W,b)$ $= -\sum_{i=1}^{n}\{t_i \log y_i + (1-t_i)\log(1-y_i)\}$	함수가 최대값이 되는것을 알기 위해서는 W와 b 에 대해 편미분을 해야 하는데, 곱하기는 미분이 불편하므로 양변에 log 를 취해 덧셈 형태로 바꾸어 주고, 함수의 부호를 바꾸어 주면 함수의 최대화 문제는 최소화 문제로 바꿀수 있으므로 다음처럼 나타냄 (파라미터 최적화를 위해서는 함수 최소값을 구하는것이 일반적임)

❖ 로그 함수 특징

로그 함수는 다음과 같은 특수한 특징을 가지고 있다.

상수 법칙	$\log_a 1 = 0, \log_a a = 1$
덧셈 법칙	$\log_a xy = \log_a x + \log_a y$
뺄셈 법칙	$\log_a \frac{x}{y} = \log_a x - \log_a y$
지수 법칙	$\log_a x^b = b \log_a x$
밑 변환 법칙	$\log_b x = \frac{\log_k x}{\log_k b}$ (단, $k > 0,\ k \neq 1$)
역수 법칙	$\log_b x = \frac{1}{\log_x b}$ (단, $b \neq 1$)

Classification

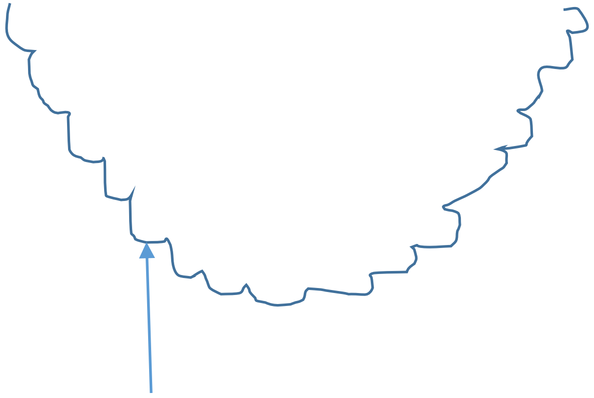
❖ Cf : loss_func를 MSE를 사용할 경우

$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

$$H(x) = Wx + b$$

$$H(X) = \frac{1}{1+e^{-W^TX}}$$

0< H(x)<
1

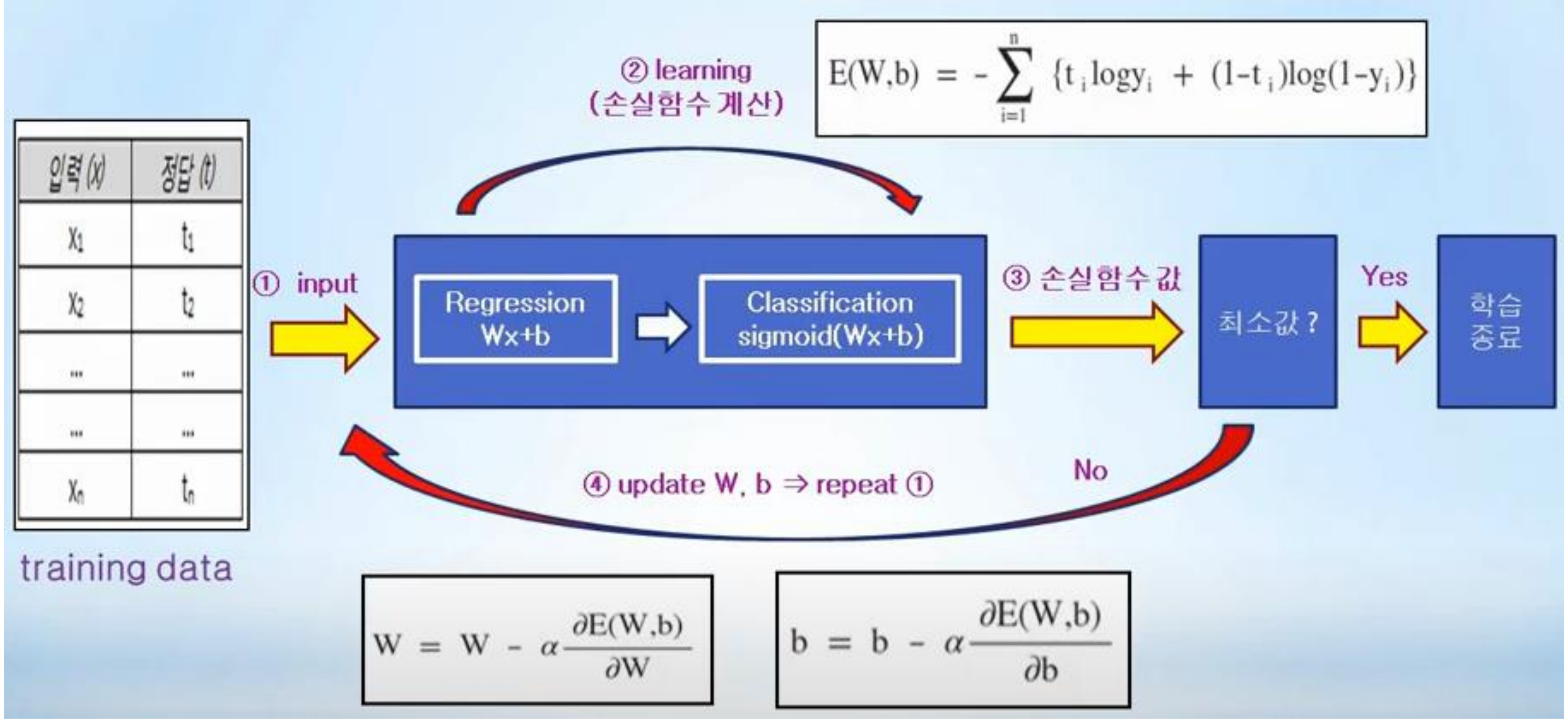


local
minimum

logsitic regression의 cost function은 non-convex function이
된다. logistic regression의 cost funciton은 조금 다르게 정
의한다.

Classification

❖ Classification에서의 [W,b] 계산 프로세스



Logistic regression implementation using python

순서	데이터 및 수식	Python 구현
1	입력 (x) | 정답 (t) x_1 | t_1 x_2 | t_2 … | … … | … x_n | t_n	slicing 또는 list comprehension 등을 이용하여 입력 x 와 정답 t 를 numpy 타입으로 분리 (t = 0 또는 1)
2	$z = Wx + b$	`W = numpy.random.rand(...)` `b = numpy.random.rand(...)`

3	classification 손실함수
	$z = Wx + b,\quad y = sigmoid(z)$
	$E(W,b) = -\sum_{i=1}^{n}\{t_i \log y_i + (1-t_i)\log(1-y_i)\}$
		`def sigmoid(x):`
		`    return 1 / ( 1+numpy.exp(-x) )`
		
		`def loss_func(...):`
		`  delta = 1e-7`
		`  z = numpy.dot(X, W) + b`
		`  y = sigmoid(z)`
		`  return -numpy.sum( t*numpy.log(y+delta) +`
		`            (1-t)*numpy.log(1-y+delta) )`
		log 무한대를 방지하기 위해 delta 정의하고 더해줌

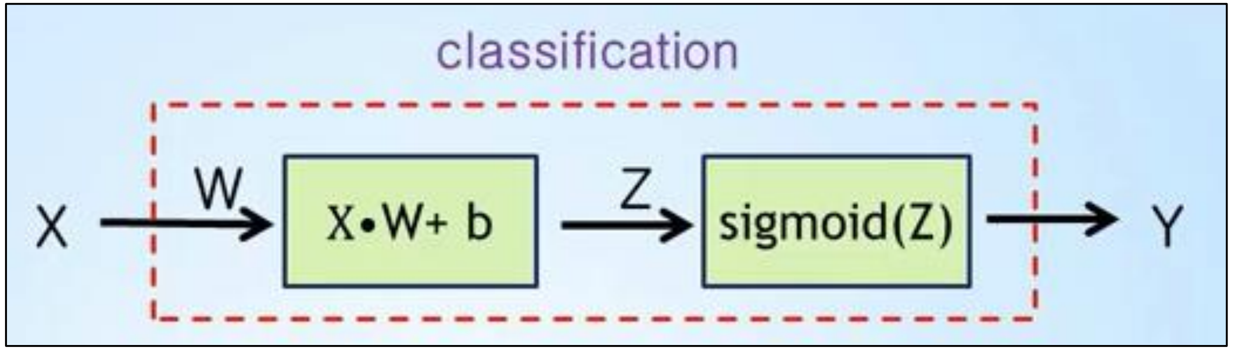
4	학습률 α	`learning_rate = 1e-3 or 1e-4 or 1e-5 ...`

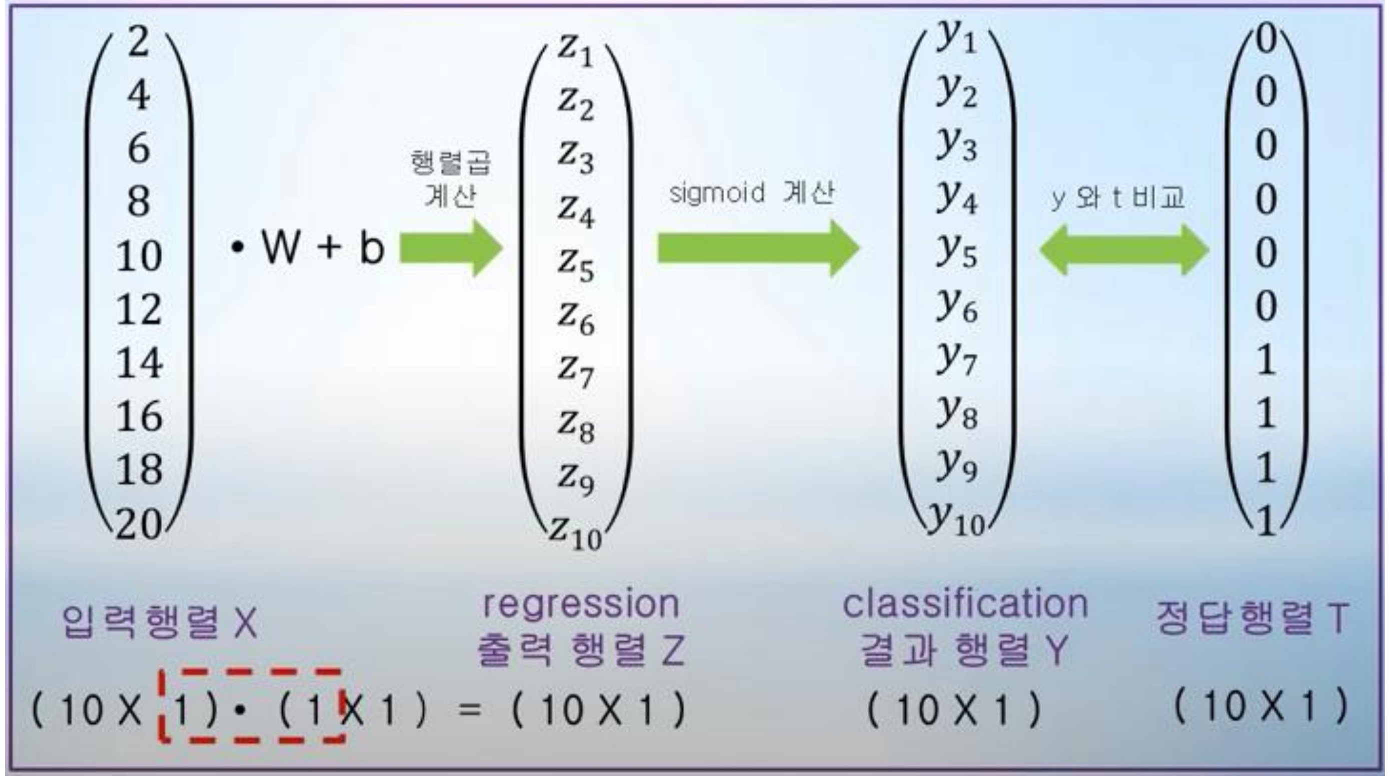
5	가중치 W, 바이어스 b
	$W = W - \alpha \frac{\partial E(W,b)}{\partial W}$
	$b = b - \alpha \frac{\partial E(W,b)}{\partial b}$
		`f = lambda x : loss_func(...)`
		[머신러닝 강의 10] 참조
		
		`for step in range(6000):  # 6000 은 임의값`
		
		`    W -= learning_rate * numerical_derivative(f, W)`
		`    b -= learning_rate * numerical_derivative(f, b)`

Simple logistic regression(classification)

training data

공부시간 (x)	Fail/Pass (t)
2	0
4	0
6	0
8	0
10	0
12	0
14	1
16	1
18	1
20	1





Simple logistic regression(classification)-example

[1] 학습 데이터(Training Data) 준비

공부시간 (x)	Fail/Pass (t)
2	0
4	0
6	0
8	0
10	0
12	0
14	1
16	1
18	1
20	1

```python
import numpy as np

x_data = np.array([2, 4, 6, 8, 10, 12, 14, 16, 18, 20]).reshape(10,1)
t_data = np.array([0, 0, 0, 0,  0,  0,  1,  1,  1,  1]).reshape(10,1)
```

[2] 임의의 직선 z=Wx+b 정의

$$z = Wx + b$$

```python
W = np.random.rand(1,1)
b = np.random.rand(1)
print("W = ", W, ", W.shape = ", W.shape, ", b = ", b, ", b.shape = ", b.shape)
```

```
W =  [[0.89991601]] , W.shape =  (1, 1) , b =  [0.88775392] , b.shape =  (1,)
```

[3] 손실함수 E(W,b) 정의

$$z = Wx + b\ ,\quad y = sigmoid(z)$$

$$E(W,b) = -\sum_{i=1}^{n} \{t_i \log y_i + (1-t_i)\log(1-y_i)\}$$

```python
def sigmoid(x):
    return 1 / (1+np.exp(-x))

def loss_func(x, t):

    delta = 1e-7    # log 무한대 발산 방지

    z = np.dot(x,W) + b
    y = sigmoid(z)

    # cross-entropy
    return  -np.sum( t*np.log(y + delta) + (1-t)*np.log((1 - y)+delta ) )
```

Simple logistic regression(classification)-example

[4] 수치 미분 함수 정의

```python
def numerical_derivative(f, x):
    delta_x = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x) # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x) # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

        x[idx] = tmp_val
        it.iternext()

    return grad
```

```python
def error_val(x, t):
    delta = 1e-7    # log 무한대 발산 방지

    z = np.dot(x,W) + b
    y = sigmoid(z)

    # cross-entropy
    return  -np.sum( t*np.log(y + delta) + (1-t)*np.log((1 - y)+delta ) )

def predict(x):

    z = np.dot(x,W) + b
    y = sigmoid(z)

    if y > 0.5:
        result = 1  # True
    else:
        result = 0  # False

    return y, result
```

[5] learning rate 초기화 및 손실함수가 최소가 될때까지 W,b 업데이트

$$W = W - \alpha\frac{\partial E(W,b)}{\partial W}$$

$$b = b - \alpha\frac{\partial E(W,b)}{\partial b}$$

```python
learning_rate = 1e-2  # 발산하는 경우, 1e-3 ~ 1e-6 등으로 바꾸어서 실행

f = lambda x : loss_func(x_data,t_data)  # f(x) = loss_func(x_data, t_data)

print("Initial error value = ", error_val(x_data, t_data), "Initial W = ", W, "\n", ", b = ", b )

for step in  range(10001):

    W -= learning_rate * numerical_derivative(f, W)

    b -= learning_rate * numerical_derivative(f, b)

    if (step % 400 == 0):
        print("step = ", step, "error value = ", error_val(x_data, t_data), "W = ", W, ", b = ",b )
```

Simple logistic regression(classification)-example

[6] 학습 결과(오차함수 값 감소 확인) 및 미래 값 예측

```
step =  0 error value =  25.536384032569956 W =  [[0.48341446]] , b =  [0.83002958]
step =  400 error value =  2.836559345023459 W =  [[0.27556358]] , b =  [-4.08265047]
step =  800 error value =  1.7868245496545063 W =  [[0.45257282]] , b =  [-5.63052857]
step =  1200 error value =  1.5197018269509306 W =  [[0.53010552]] , b =  [-6.66295462]
step =  1600 error value =  1.353620406715284 W =  [[0.59147126]] , b =  [-7.47786268]
step =  2000 error value =  1.2368337686586899 W =  [[0.64305797]] , b =  [-8.16140924]
step =  2400 error value =  1.1484801870526067 W =  [[0.68801019]] , b =  [-8.75598634]
step =  2800 error value =  1.0783212136203029 W =  [[0.72812535]] , b =  [-9.28580415]
step =  3200 error value =  1.0206609487469689 W =  [[0.76453508]] , b =  [-9.76608854]
step =  3600 error value =  0.972041212260172 W =  [[0.79800183]] , b =  [-10.20708674]
step =  4000 error value =  0.9302223339656082 W =  [[0.82906586]] , b =  [-10.61605159]
step =  4400 error value =  0.8936798164784363 W =  [[0.85812516]] , b =  [-10.99832041]
step =  4800 error value =  0.861334125458589 W =  [[0.88548219]] , b =  [-11.35794466]
step =  5200 error value =  0.832395816354841 W =  [[0.91137263]] , b =  [-11.69807862]
step =  5600 error value =  0.8062719524510953 W =  [[0.93598405]] , b =  [-12.02123026]
step =  6000 error value =  0.7825070402369488 W =  [[0.95946842]] , b =  [-12.32942952]
step =  6400 error value =  0.7607443619652774 W =  [[0.98195071]] , b =  [-12.62434456]
step =  6800 error value =  0.7406998604919448 W =  [[1.00353507]] , b =  [-12.90736448]
step =  7200 error value =  0.7221440214084223 W =  [[1.0243093]] , b =  [-13.1796594]
step =  7600 error value =  0.704889005702817 W =  [[1.04434816]] , b =  [-13.44222508]
step =  8000 error value =  0.6887793212694522 W =  [[1.06371591]] , b =  [-13.69591654]
step =  8400 error value =  0.6736849354506804 W =  [[1.08246818]] , b =  [-13.94147397]
step =  8800 error value =  0.6594961063489078 W =  [[1.10065352]] , b =  [-14.1795428]
step =  9200 error value =  0.6461194468295698 W =  [[1.11831459]] , b =  [-14.41068954]
step =  9600 error value =  0.6334748873658479 W =  [[1.13548904]] , b =  [-14.63541445]
step =  10000 error value =  0.621493304189263 W =  [[1.15221034]] , b =  [-14.85416169]
```

손실함수 값, W, b 확인

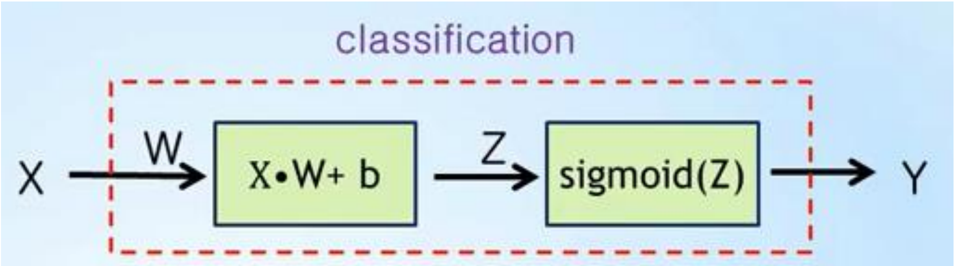
```
(real_val, logical_val) = predict(3)

print(real_val, logical_val)

[[1.12150565e-05]] 0
```

3시간 입력 미래값 예측

```
(real_val, logical_val) = predict(17)

print(real_val, logical_val)

[[0.9912827]] 1
```

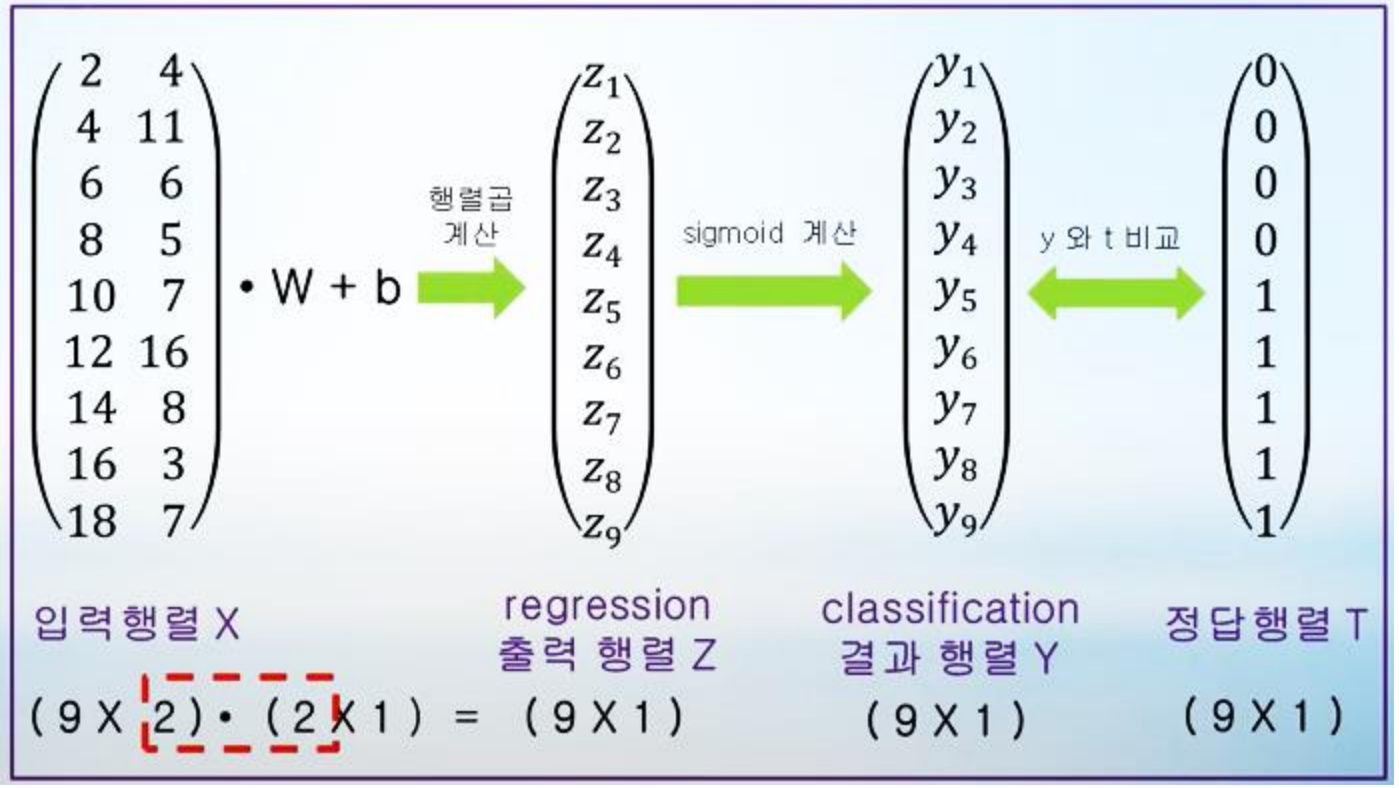
17시간 입력 미래값 예측

Multi-variable logistic regression(classification)-overview



training data

예습시간 (x1)	복습시간 (x2)	Fail/Pass (t)
2	4	0
4	11	0
6	6	0
8	5	0
10	7	1
12	16	1
14	8	1
16	3	1
18	7	1



Multi-variable logistic regression(classification)-example

[1] 학습데이터(Training Data) 준비

예습시간 ($x1$)	복습시간 ($x2$)	Fail/Pass (t)
2	4	0
4	11	0
6	6	0
8	5	0
10	7	1
12	16	1
14	8	1
16	3	1
18	7	1

```python
import numpy as np

x_data = np.array([ [2, 4], [4, 11], [6, 6], [8, 5], [10, 7], [12, 16], [14, 8], [16, 3], [18, 7] ])
t_data = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1]).reshape(9, 1)
```

[2] 임의의 직선 $z = W_1x_1 + W_2x_2 + b$ 정의 (가중치 W, 바이어스 b 초기화)

$$z = W_1x_1 + W_2x_2 + b$$

```python
W = np.random.rand(2, 1)  # 2X1 행렬
b = np.random.rand(1)
print("W = ", W, ", W.shape = ", W.shape, ", b = ", b, ", b.shape = ", b.shape)
```

```
W =  [[0.7579448 ]
 [0.76848043]] , W.shape =  (2, 1) , b =  [0.82990875] , b.shape =  (1,)
```

[3] 손실함수 E(W,b) 정의

$$z = W_1x_1 + W_2x_2 + b\ ,\quad y = \text{sigmoid}(z)$$

$$E(W,b) = -\sum_{i=1}^{n}\{t_i\log y_i + (1-t_i)\log(1-y_i)\}$$

```python
def sigmoid(x):
    return 1 / (1+np.exp(-x))

def loss_func(x, t):

    delta = 1e-7    # log 무한대 발산 방지

    z = np.dot(x,W) + b
    y = sigmoid(z)

    # cross-entropy
    return  -np.sum( t*np.log(y + delta) + (1-t)*np.log((1 - y)+delta ) )
```

Multi-variable logistic regression(classification)-example

[4] 수치미분 numerical_derivative 및 utility 함수 정의

```python
def numerical_derivative(f, x):
    delta_x = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x) # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x) # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

        x[idx] = tmp_val
        it.iternext()

    return grad
```

```python
def error_val(x, t):
    delta = 1e-7    # log 무한대 발산 방지

    z = np.dot(x,W) + b
    y = sigmoid(z)

    # cross-entropy
    return  -np.sum( t*np.log(y + delta) + (1-t)*np.log((1 - y)+delta ) )

def predict(x):

    z = np.dot(x,W) + b
    y = sigmoid(z)

    if y > 0.5:
        result = 1  # True
    else:
        result = 0  # False

    return y, result
```

[5] 학습율 (learning rate) 초기화 및 손실함수가 최소가 될 때까지 W, b 업데이트

$$W = W - \alpha \frac{\partial E(W,b)}{\partial W}$$

$$b = b - \alpha \frac{\partial E(W,b)}{\partial b}$$

```python
learning_rate = 1e-2  # 1e-2, 1e-3 은 손실함수 값 발산

f = lambda x : loss_func(x_data,t_data)

print("Initial error value = ", error_val(x_data, t_data), "Initial W = ", W, "\n", ", b = ", b )

for step in  range(80001):

    W -= learning_rate * numerical_derivative(f, W)

    b -= learning_rate * numerical_derivative(f, b)

    if (step % 400 == 0):
        print("step = ", step, "error value = ", error_val(x_data, t_data), "W = ", W, ", b = ",b )
```

Multi-variable logistic regression(classification)-example

[6] 학습 결과 (오차 함수 값 감소 확인)

```
step =  0 error value =  27.66493464754572 W =  [[0.55940731]
 [0.51142089]] , b =  [0.79011283]
step =  400 error value =  2.328341300532911 4 W =  [[ 0.41352506]
 [-0.09117871]] , b =  [-2.51884116]
step =  800 error value =  1.6124129302 90204 W =  [[ 0.53139281]
 [-0.02857654]] , b =  [-4.19776047]
step =  1200 error value =  1.2912531567080934 W =  [[0.61957496]
 [0.00756889]] , b =  [-5.32709382]
step =  1600 error value =  1.1053115599716365 W =  [[0.69056282]
 [0.0331307 ]] , b =  [-6.18738479]
step =  2000 error value =  0.9816631487006757 W =  [[0.75029335]
 [0.05339485]] , b =  [-6.88915481]
step =  2400 error value =  0.8920039908822527 W =  [[0.80202179]
 [0.0706829 ]] , b =  [-7.48678277]
step =  2800 error value =  0.8230538507 1534 W =  [[0.8477278]
 [0.0861879]] , b =  [-8.01086251]
step =  3200 error value =  0.767743435002518 W =  [[0.88870922]
 [0.10058654]] , b =  [-8.48023558]
```

....

```
step =  76800 error value =  0.07605165836605418 W =  [[2.25903629]
 [1.04689388]] , b =  [-26.5269181]
step =  77200 error value =  0.0756791394582931 2 W =  [[2.26194816]
 [1.0489385 ]] , b =  [-26.56536897]
step =  77600 error value =  0.07531023094468238 W =  [[2.26484608]
 [1.05097284]] , b =  [-26.603633]
step =  78000 error value =  0.0749448808167 9372 W =  [[2.2677302 ]
 [1.05299702]] , b =  [-26.64171198]
step =  78400 error value =  0.0745830380570064 3 W =  [[2.27060065]
 [1.05501113]] , b =  [-26.67960768]
step =  78800 error value =  0.07422465261511503 W =  [[2.27345755]
 [1.05701528]] , b =  [-26.71732185]
step =  79200 error value =  0.07386967538554952 W =  [[2.27630103]
 [1.05900955]] , b =  [-26.7548562]
step =  79600 error value =  0.07351805818525665 W =  [[2.27913122]
 [1.06099405]] , b =  [-26.79221243]
step =  80000 error value =  0.073 975373220189 W =  [[2.28194823]
 [1.06296887]] , b =  [-26.8293922
```

손실함수 확인

Multi-variable logistic regression(classification)-example

[7] 미래 값 예측

```
test_data = np.array([3, 17]) # (예습, 복습) = (3, 17) => Fail (0)
predict(test_data)
```

```
(array([0.12867978]), 0)
```

```
test_data = np.array([5, 8]) # (예습, 복습) = (5, 8) => Fail (0)

predict(test_data)
```

```
(array([0.00099032]), 0)
```

```
test_data = np.array([7, 21]) # (예습, 복습) = (7, 21) => Pass (1)

predict(test_data)
```

```
(array([0.99998955]), 1)
```

```
test_data = np.array([12, 0])  # (예습, 복습) = (12, 0) => Pass (1)

predict(test_data)
```

```
(array([0.63499634]), 1)
```

미래 값을 예측해보면, 복습보다는 예습시간이 합격(Pass)에 미치는 영향이 크다는 것을 알 수 있음 (즉, 예습시간에 대한 가중치 W_1 = 2.28, 복습시간에 대한 가중치 W_2 = 1.06 에서 보듯이 예습시간이 복습시간에 비해 최종결과에 미치는 영향이 2배 이상임)

머신러닝 XOR 문제

❖ **AND, OR, NAND, XOR**

AND

x1	x2	t
0	0	0
0	1	0
1	0	0
1	1	1

OR

x1	x2	t
0	0	0
0	1	1
1	0	1
1	1	1

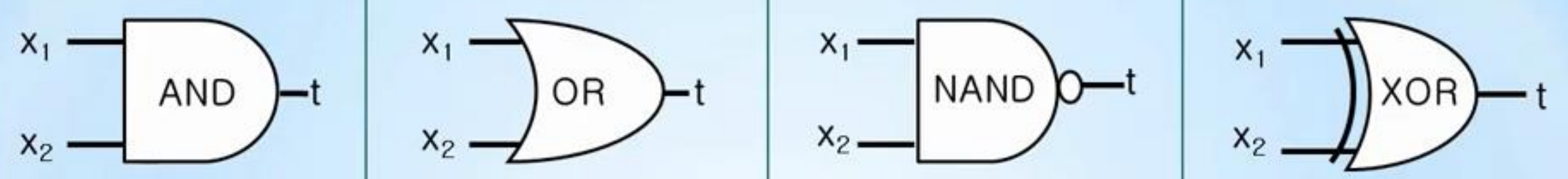
NAND

x1	x2	t
0	0	1
0	1	1
1	0	1
1	1	0

XOR

x1	x2	t
0	0	0
0	1	1
1	0	1
1	1	0

논리게이트 –AND . OR . NAND . XOR



x1	x2	t
0	0	0
1	0	0
0	1	0
1	1	1

x1	x2	t
0	0	0
1	0	1
0	1	1
1	1	1

x1	x2	t
0	0	1
1	0	1
0	1	1
1	1	0

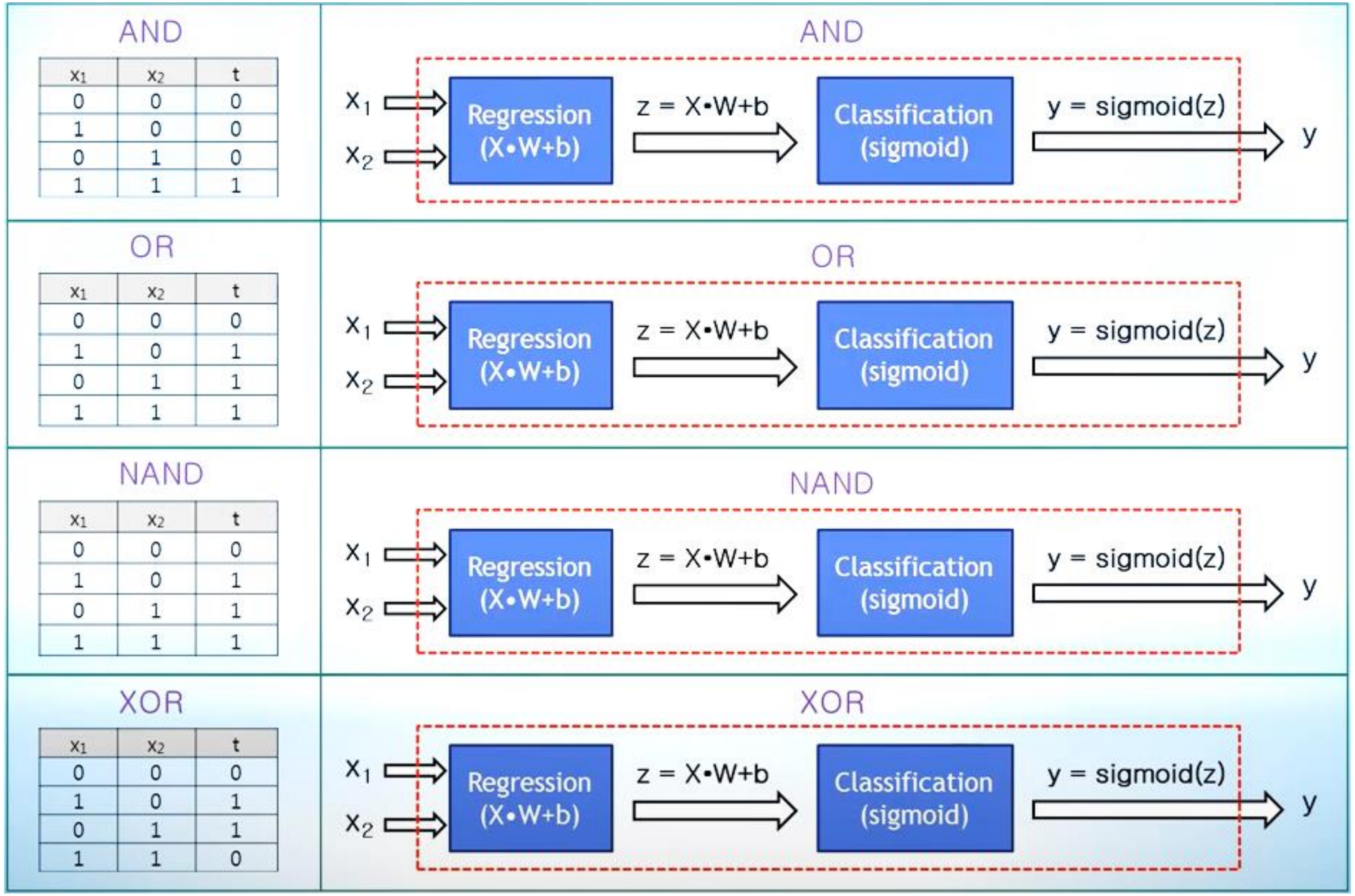
x1	x2	t
0	0	0
1	0	1
0	1	1
1	1	0

- AND, OR, NAND, XOR 논리테이블(Logic Table)은 입력데이터 (x_1, x_2), 정답데이터 t (0 또는 1) 인 머신러닝 Training Data와 개념적으로 동일함

  ⇒ 즉, 논리게이트는 손실함수로 cross-entropy를 이용해서 Logistic Regression (Classification)* 알고리즘으로 데이터를 분류 하고 결과를 예측 할 수 있음

* Logistic Regression (Classification) 자세한 내용은 이전 강의 [머신러닝 강의 16], [머신러닝 강의 17] 참조

AND . OR . NAND . XOR Internal



LogicGate class-AND . OR . NAND . XOR 검증

External function

```
def  sigmoid(x):                       # 0 또는 1 을 출력하기 위한 sigmoid 함수

def  numerical_derivative(f, x):      #  수치미분함수 (소스코드는 [머신러닝 강의 10] 참조)
```

LogicGate class

```
class  LogicGate:
    def    __init__(self, gate_name, xdata, tdata)    # __xdata, __tdata, __W, __b 초기화
    def    __loss_func(self)          # 손실함수 cross-entropy
    def    error_val(self)            # 손실함수 값 계산
    def    train(self)                # 수치미분을 이용하여 손실함수 최소값 찾는  method
    def    predict(self, xdata)       # 미래 값 예측  method
```

usage

```
xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])      # 입력 데이터 생성
tdata  = np.array([0, 0, 0, 1])                             # 정답 데이터 생성 (AND 예시)

AND_obj = LogicGate("AND_GATE", xdata, tdata)             # LogicGate 객체생성
AND_obj.train()                                             # 손실함수 최소값 갖도록 학습

AND_obj.predict(…)                                          # 임의 데이터에 대해 결과 예측
```

LogicGate class-AND . OR . NAND . XOR : 구현 코드

```python
import numpy as np

# sigmoid 함수
def sigmoid(x):
    return 1 / (1+np.exp(-x))

# 수치미분 함수
def numerical_derivative(f, x):
    delta_x = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x) # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x) # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

        x[idx] = tmp_val
        it.iternext()

    return grad
```

```python
# LogicGate Class

class LogicGate:

    def __init__(self, gate_name, xdata, tdata):  # xdata, tdata => numpy.array(...)

        self.name = gate_name

        # 입력 데이터, 정답 데이터 초기화
        self.__xdata = xdata.reshape(4,2)
        self.__tdata = tdata.reshape(4,1)

        # 가중치 W, 바이어스 b 초기화
        self.__W = np.random.rand(2,1)  # weight, 2 X 1 matrix
        self.__b = np.random.rand(1)

        # 학습률 learning rate 초기화
        self.__learning_rate = 1e-2

    # 손실함수
    def __loss_func(self):

        delta = 1e-7    # log 무한대 발산 방지

        z = np.dot(self.__xdata, self.__W) + self.__b
        y = sigmoid(z)

        # cross-entropy
        return  -np.sum( self.__tdata*np.log(y + delta) + (1-self.__tdata)*np.log((1 - y)+delta ) )
```

LogicGate class-AND . OR . NAND . XOR : 구현 코드

```python
# LogicGate Class

class LogicGate:
```

... 생략

```python
    # 손실 값 계산
    def error_val(self):

        delta = 1e-7    # log 무한대 발산 방지

        z = np.dot(self.__xdata, self.__W) + self.__b
        y = sigmoid(z)

        # cross-entropy
        return  -np.sum( self.__tdata*np.log(y + delta) + (1-self.__tdata)*np.log((1 - y)+delta ) )

    # 수치미분을 이용하여 손실함수가 최소가 될때 까지 학습하는 함수
    def train(self):

        f = lambda x : self.__loss_func()

        print("Initial error value = ", self.error_val())

        for step in  range(8001):

            self.__W -= self.__learning_rate * numerical_derivative(f, self.__W)

            self.__b -= self.__learning_rate * numerical_derivative(f, self.__b)

            if (step % 400 == 0):
                print("step = ", step, "error value = ", self.error_val())
```

```python
# LogicGate Class

class LogicGate:
```

... 생략

```python
    # 미래 값 예측 함수
    def predict(self, input_data):

        z = np.dot(input_data, self.__W) + self.__b
        y = sigmoid(z)

        if y > 0.5:
            result = 1  # True
        else:
            result = 0  # False

        return y, result
```

LogicGate class-AND . OR . NAND . XOR : 구현 코드

❖ AND 게이트 훈련

```
xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
tdata = np.array([0, 0, 0, 1])

AND_obj = LogicGate("AND_GATE", xdata, tdata)

AND_obj.train()
```

❖ AND 게이트 테스트

```
# AND Gate prediction
print(AND_obj.name, "\n")

test_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])

for input_data in test_data:
    (sigmoid_val, logical_val) = AND_obj.predict(input_data)
    print(input_data, " = ", logical_val, "\n")
```

```
AND_GATE

[0 0]  =  0

[0 1]  =  0

[1 0]  =  0

[1 1]  =  1
```

❖ OR 게이트 훈련

```
xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
tdata = np.array([0, 1, 1, 1])

OR_obj = LogicGate("OR_GATE", xdata, tdata)

OR_obj.train()
```

❖ OR 게이트 테스트

```
# OR Gate prediction
print(OR_obj.name, "\n")

test_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])

for input_data in test_data:
    (sigmoid_val, logical_val) = OR_obj.predict(input_data)
    print(input_data, " = ", logical_val, "\n")
```

```
OR_GATE

[0 0]  =  0

[0 1]  =  1

[1 0]  =  1

[1 1]  =  1
```

LogicGate class-AND . OR . NAND . XOR : 구현 코드

❖ NAND 게이트 훈련

```
xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
tdata = np.array([1, 1, 1, 0])

NAND_obj = LogicGate("NAND_GATE", xdata, tdata)

NAND_obj.train()
```

❖ NAND 게이트 테스트

```
# NAND Gate prediction
print(NAND_obj.name, "\n")

test_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])

for input_data in test_data:
    (sigmoid_val, logical_val) = NAND_obj.predict(input_data)
    print(input_data, " = ", logical_val, "\n")
```

```
NAND_GATE

[0 0]  =  1

[0 1]  =  1

[1 0]  =  1

[1 1]  =  0
```

❖ XOR 게이트 훈련

```
xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
tdata = np.array([0, 1, 1, 0])


XOR_obj = LogicGate("XOR_GATE", xdata, tdata)

# XOR Gate 를 보면, 손실함수 값이 2.7 근처에서 더 이상 감소하지 않는것을 볼수 있음
XOR_obj.train()
```

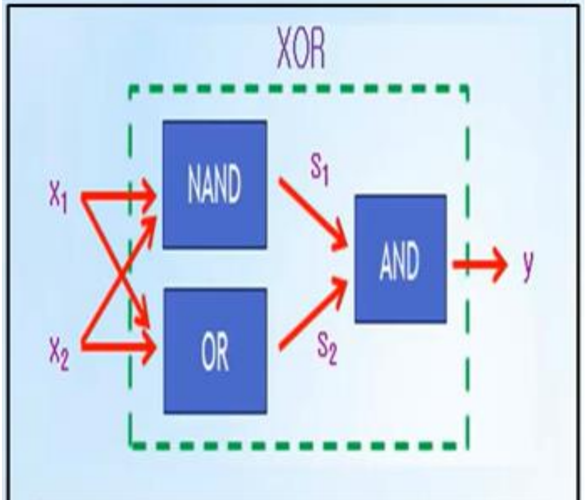
❖ XOR 게이트 테스트

```
# XOR Gate prediction => 예측이 되지 않음
print(XOR_obj.name, "\n")

test_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])

for input_data in test_data:
    (sigmoid_val, logical_val) = XOR_obj.predict(input_data)
    print(input_data, " = ", logical_val, "\n")
```

```
XOR_GATE

[0 0]  =  0

[0 1]  =  0

[1 0]  =  0

[1 1]  =  1
```

결과 값 오류

❖ XOR=NAND + OR 조합으로 계산



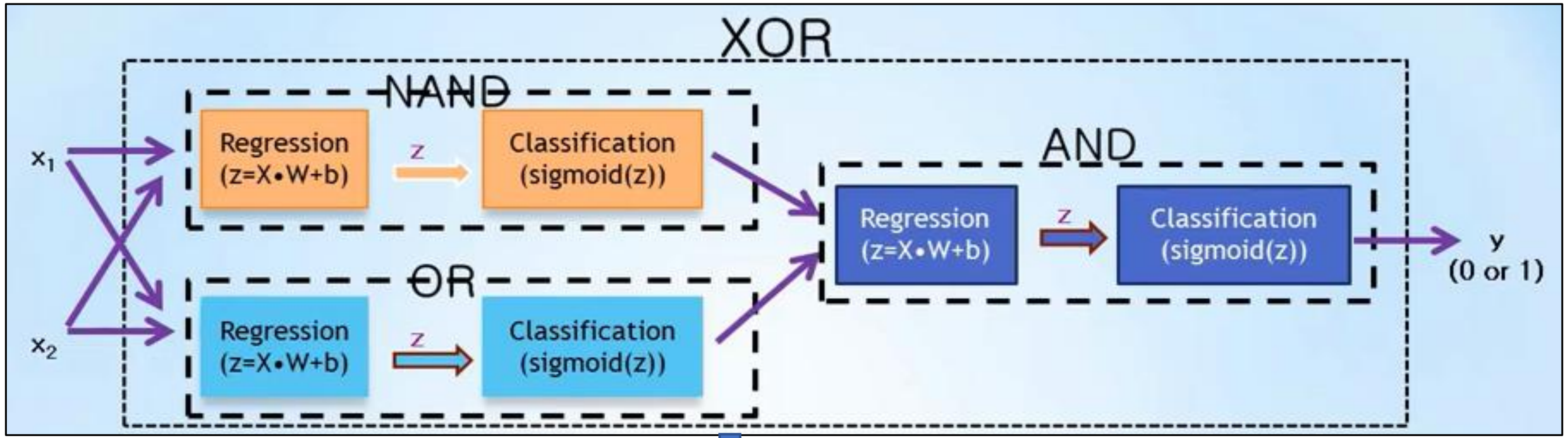
x_1	x_2	s_1 (NAND)	s_2 (OR)	y (XOR)
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

```python
# XOR 을 NAND + OR => AND 조합으로 계산함
input_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])

s1 = []     # NAND 출력
s2 = []     # OR 출력

new_input_data = []  # AND 입력
final_output = []    # AND 출력

for index in range(len(input_data)):

    s1 = NAND_obj.predict(input_data[index])  # NAND 출력
    s2 = OR_obj.predict(input_data[index])    # OR 출력

    new_input_data.append(s1[-1])    # AND 입력
    new_input_data.append(s2[-1])    # AND 입력

    (sigmoid_val, logical_val) = AND_obj.predict(np.array(new_input_data))

    final_output.append(logical_val)    # AND 출력, 즉 XOR 출력
    new_input_data = []    # AND 입력 초기화


for index in range(len(input_data)):
    print(input_data[index], " = ", final_output[index], end='')
    print("\n")
```

실행 결과 값

```
[0 0]  =  0

[0 1]  =  1

[1 0]  =  1

[1 1]  =  0
```

XOR 구현



- 머신러닝 XOR 문제는 다양한 Gate 조합인 Multi-Layer*로 해결 할 수 있음

  * Layer : 데이터를 처리하거나 계산이 이루어 지는 단위

- 각각의 Gate(NAND, OR, AND)는 Logistic Regression(Classification) 시스템으로 구성됨
- 이전 Gate 모든 출력은 (previous output) 다음 Gate 입력 (next input) 으로 들어감

insight

신경망(Neural Network 기반 딥러닝(Deep Learning)핵심 아이디어