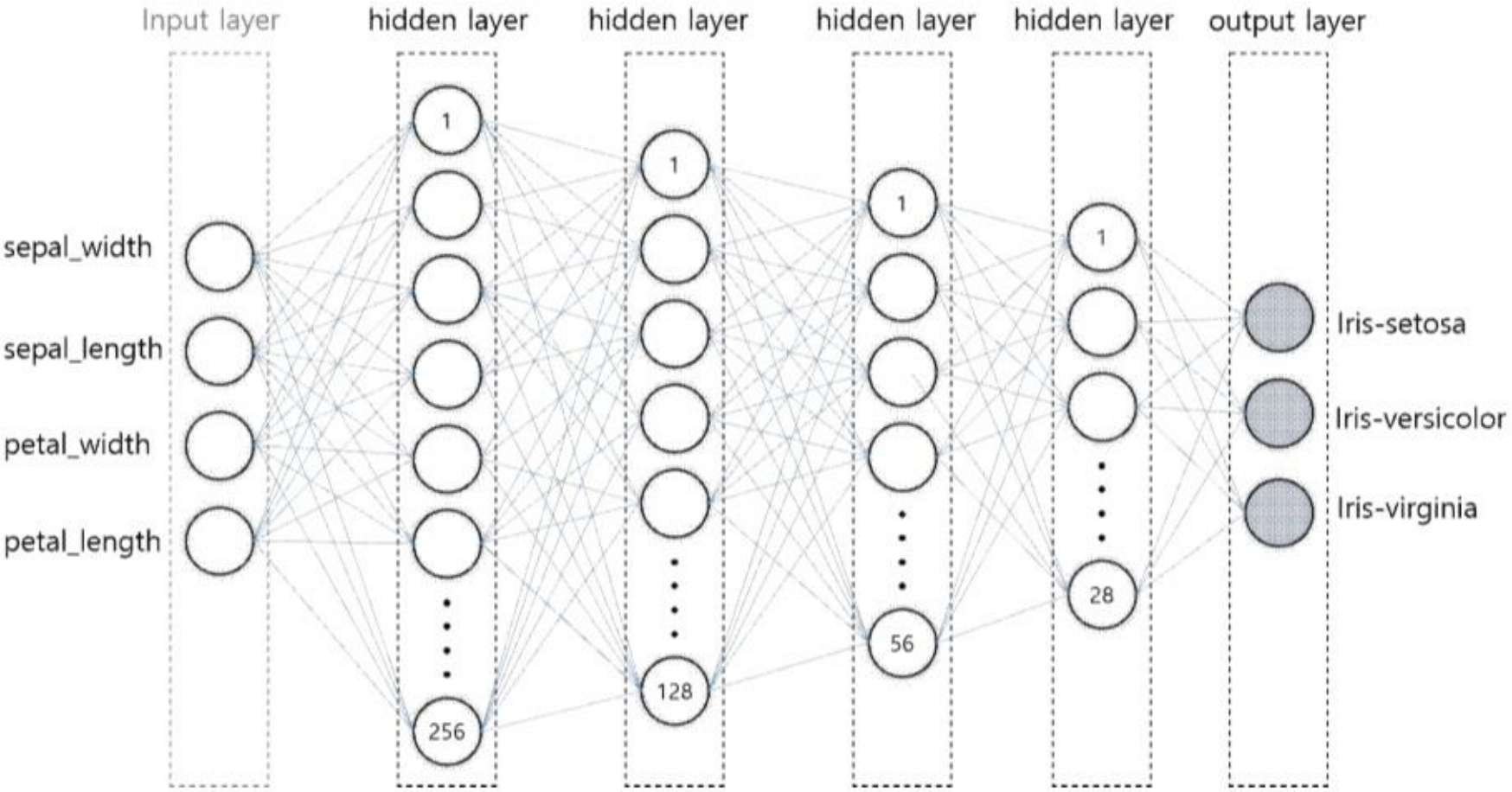
DNN(Deep Neural Network)

딥러닝이란?

개념소개

- 딥러닝의 기본이 되는 인공신경망(ANN, Artificial Neural Network)의 개념은 1943년에 신경생리학자 워렌 맥 컬럭(Warren McCulloch)과 수학자 월터 피트(Walter Pitss)가 처음 제안하였다.
- 여러 겹의 신경망 레이어를 쌓아 올린 형태이고, 레이어는 노드(node)라는 유닛의 집합이다.
- 노드는 바로 앞의 레이어들로부터 값을 전달받아 각 값들에 가중치를 곱한 후 모두 더해 새로운 값을 만든다. 그리고 새 로운 값은 미리 지정해둔 노드의 활성 함수를 통해 변환된 후 다음 레이어로 전달된다. 그리고 다음 레이어에서 는 같은 과정을 반복한다.
- 활성 함수는 딥러닝을 비선형 모델로 만들어 성능을 향상시킨다.
- 아이리스 데이터셋을 예로 들어 딥러닝 모델을 그림으로 표현하면 아래와 같다.
  - input layer, hidden layer, output layer로 구성된다.
  - input layer : 값을 입력받는 역할만 하는 레이어이다. 노드의 개수는 피쳐의 개수 와 동일하다.
  - hidden layer : 모델의 성능을 좌우하는 중요한 레이어다. 딥러닝에서는 히든 레이어의 개수와 각 레이어의 노 드 개수가 하이퍼 파라미터다.
  - output layer : 최종 결과를 산출하는 레이어다. 분류 분석에서는 클래스의 개수 만큼 output layer의 노드 수를 지정한다. 회귀 문제에서는 output layer의 노드는 1이다.

DNN(Deep Neural Network)



DNN(Deep Neural Network)

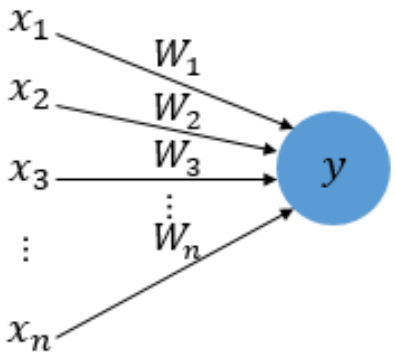
딥러닝의 개념

퍼셉트론(Perceptron)

- 프랑크 로젠블라트(Frank Rosenblatt)가 1957년에 제안한 초기 형태의 인공 신경망으로 다수의 입력으로부터 하나의 결과를 내보내는 알고리즘이다.
- 퍼셉트론은 실제 뇌를 구성하는 신경 세포 뉴런의 동작과 유사한데, 신경 세포 뉴런의 그림을 보면 뉴런은 가지돌기에서 신호를 받아들이고, 이 신호가 일정치 이상의 크기를 가지면 축삭돌기를 통해서 신호를 전달한다.

DNN(Deep Neural Network)

- 퍼셉트론의 그림을 보면 신경 세포 뉴런의 입력 신호와 출력 신호가 퍼셉트론에서 각각 입력값과 출력값에 해당된다.



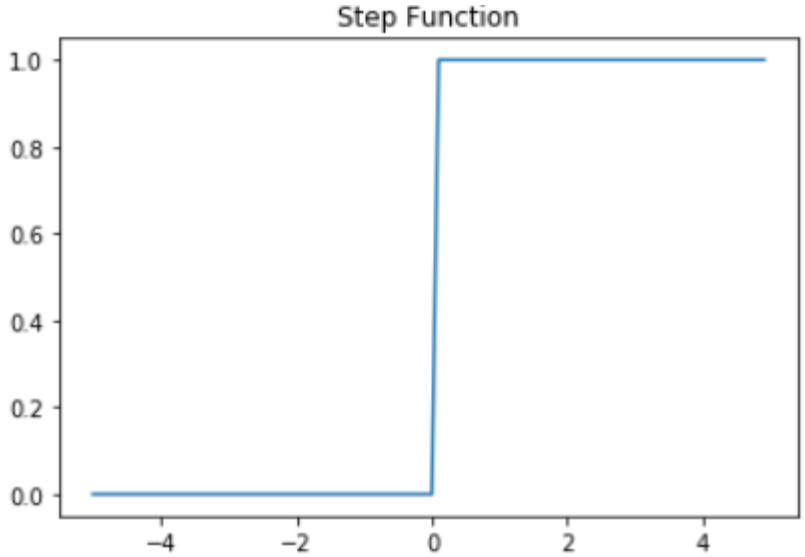
- 각각의 입력값에는 각각의 가중치가 존재하는데, 이때 가중치의 값이 크면 클수록 해당 입력값이 중요하다는 것을 의미한다.
- 이 임계치값을 수식으로 표현할 때는 보통 세타(Θ)로 표현합니다. 이를 식으로 표현하면 다음과 같다.

$$if \sum_{i}^{n} W_i x_i \ \geq \theta \rightarrow y = 1$$

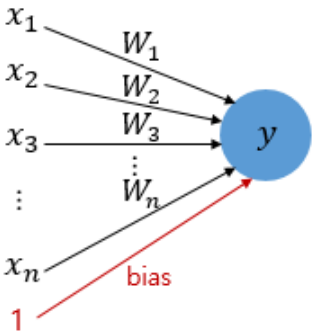
$$if \sum_{i}^{n} W_i x_i \ < \theta \rightarrow y = 0$$

DNN(Deep Neural Network)

- 각 입력값이 가중치와 곱해져서 인공 뉴런에 보내지고, 각 입력값과 그에 해당되는 가중치의 곱의 전체 합이 임계치(threshold)를 넘으면 종착지에 있는 인공 뉴런은 출력 신호로서 1을 출력하고, 그렇지 않을 경우에는 0을 출력한다. 이러한 함수를 계단 함수(Step function)라고 하며, 아래는 그래프는 계단 함수의 하나의 예를 보여준다.



- 이때 계단 함수에 사용된 편향 b를 추가하면 아래 그림처럼 표현할 수 있다.



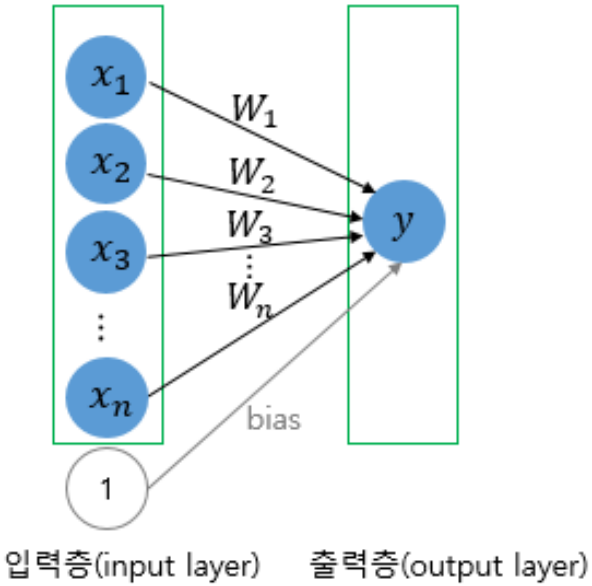
$$if \sum_{i}^{n} W_i x_i + b \geq 0 \rightarrow y = 1$$

$$if \sum_{i}^{n} W_i x_i + b < 0 \rightarrow y = 0$$

DNN(Deep Neural Network)

단층 퍼셉트론(Single-Layer Perceptron)

- 구조
  - 값을 보내는 단계와 값을 받아서 출력하는 두 단계로만 이루어진다.
  - 이 각 단계를 보통 층(layer)라고 부르며, 이 두 개의 층을 입력층(input layer)과 출력층(output layer)이라고 한다.



  - 단층 퍼셉트론을 이용하면 AND, NAND, OR 게이트를 쉽게 구현할 수 있다. 게이트 연산에 쓰이는 것은 두 개의 입력 값과 하나의 출력 값이다.

DNN(Deep Neural Network)

- AND 게이트
  - 단층 퍼셉트론의 식을 통해 AND 게이트를 만족하는 두 개의 가중치와 편향 값에는 뭐가 있을까? 각각 w1, w2, b라고 한다면 [0.5, 0.5, -0.7], [0.5, 0.5, -0.8] 또는 [1.0, 1.0, -1.0] 등 이 외에도 다양한 가중치와 편향의 조합이 나올 수 있다.

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

```
def AND_gate(x1, x2):
    w1=0.5
    w2=0.5
    b=-0.7
    result = x1*w1 + x2*w2 + b
    if result <= 0:
        return 0
    else:
        return 1
```

  - 위의 함수에 AND 게이트의 입력값을 모두 넣어보면 오직 두 개의 입력값이 1인 경우에만 1을 출력한다.

DNN(Deep Neural Network)

- NAND 게이트
  - 두 개의 입력값이 1인 경우에만 출력값이 0, 나머지 입력값의 쌍(pair)에 대해서는 모두 출력값이 1이 나오는 NAND 게이트는 다음과 같이 구할 수 있다.

```
def NAND_gate(x1, x2):
    w1=-0.5
    w2=-0.5
    b=0.7
    result = x1*w1 + x2*w2 + b
    if result <= 0:
        return 0
    else:
        return 1
```

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

DNN(Deep Neural Network)

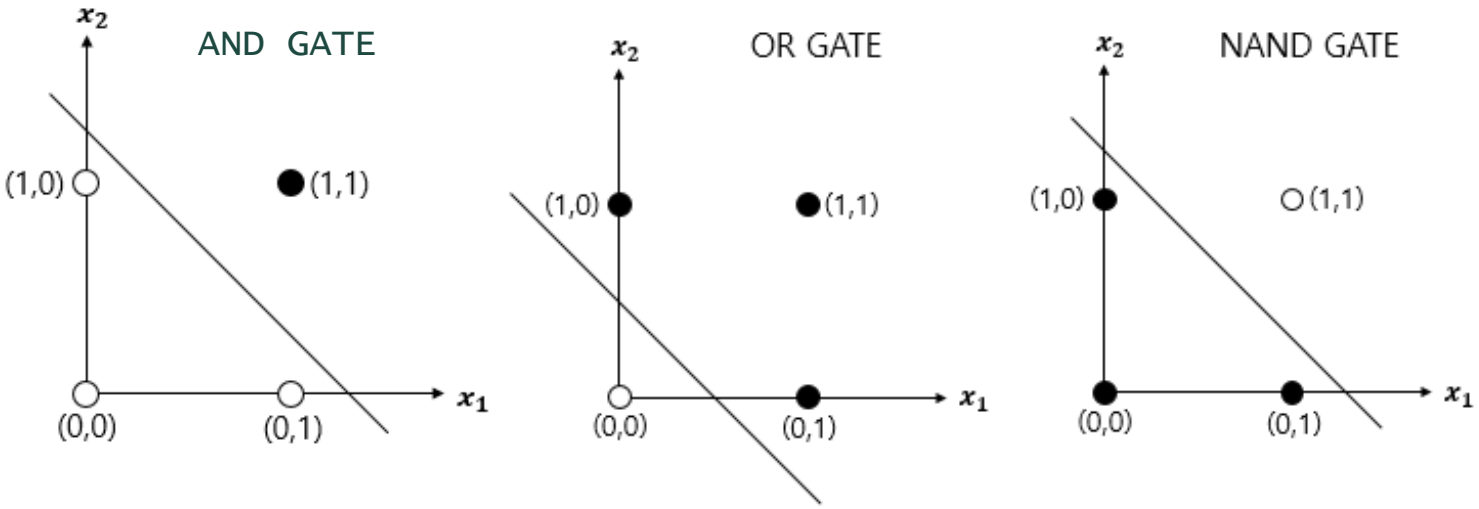
- OR 게이트
  - 두 개의 입력이 모두 0인 경우에 출력값이 0이고 나머지 경우에는 모두 출력값이 1인 OR 게이트 또한 적절한 가중치 값과 편향 값만 찾으면 단층 퍼셉트론의 식으로 구현할 수 있다.
  - 예를 들어 각각 가중치와 편향에 대해서 [0.6, 0.6, -0.5]를 선택하면 OR 게이트를 충족한다.

```
def OR_gate(x1, x2):
    w1=0.6
    w2=0.6
    b=-0.5
    result = x1*w1 + x2*w2 + b
    if result <= 0:
        return 0
    else:
        return 1
```

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

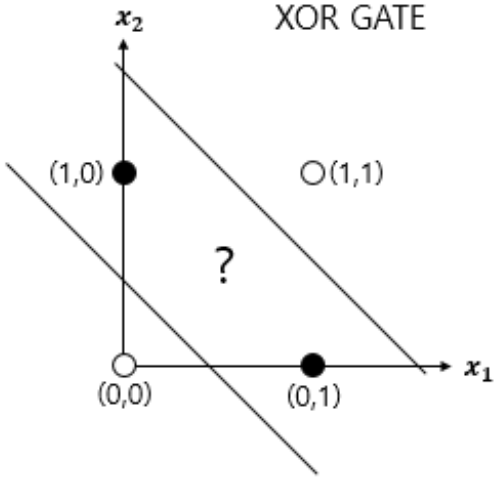
DNN(Deep Neural Network)

- **XOR 게이트**
  - 단층 퍼셉트론은 AND 게이트, NAND 게이트, OR 게이트 또한 구현할 수 있다. 하지만 단층 퍼셉트론으로 구현이 불가능한 게이트가 있는데 바로 XOR 게이트이다.
  - XOR 게이트는 입력값 두 개가 서로 다른 값을 갖고 있을 때에만 출력값이 1이 되고, 입력값 두 개가 서로 같은 값을 가지면 출력값이 0이 되는 게이트이다.
  - 위의 파이썬 코드에 아무리 수많은 가중치와 편향을 넣어봐도 XOR 게이트를 구현하는 것은 불가능하다. 그 이유는 단층 퍼셉트론은 직선 하나로 두 영역을 나눌 수 있는 문제에 대해서만 구현이 가능하기 때문이다.



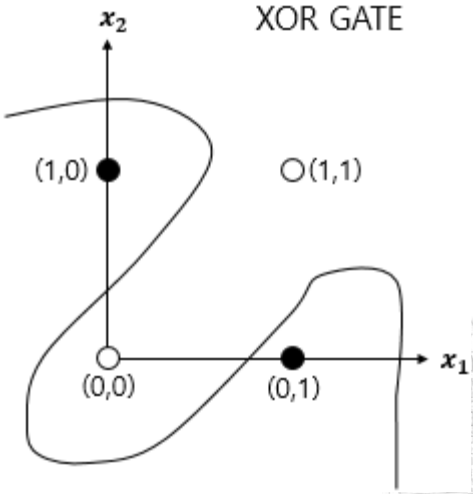
DNN(Deep Neural Network)

- XOR 게이트는 어떨까요? XOR 게이트는 입력값 두 개가 서로 다른 값을 갖고 있을때에만 출력값이 1이 되고, 입력값 두 개가 서로 같은 값을 가지면 출력값이 0이 되는 게이트입니다. XOR 게이트를 시각화해보면 다음과 같다.



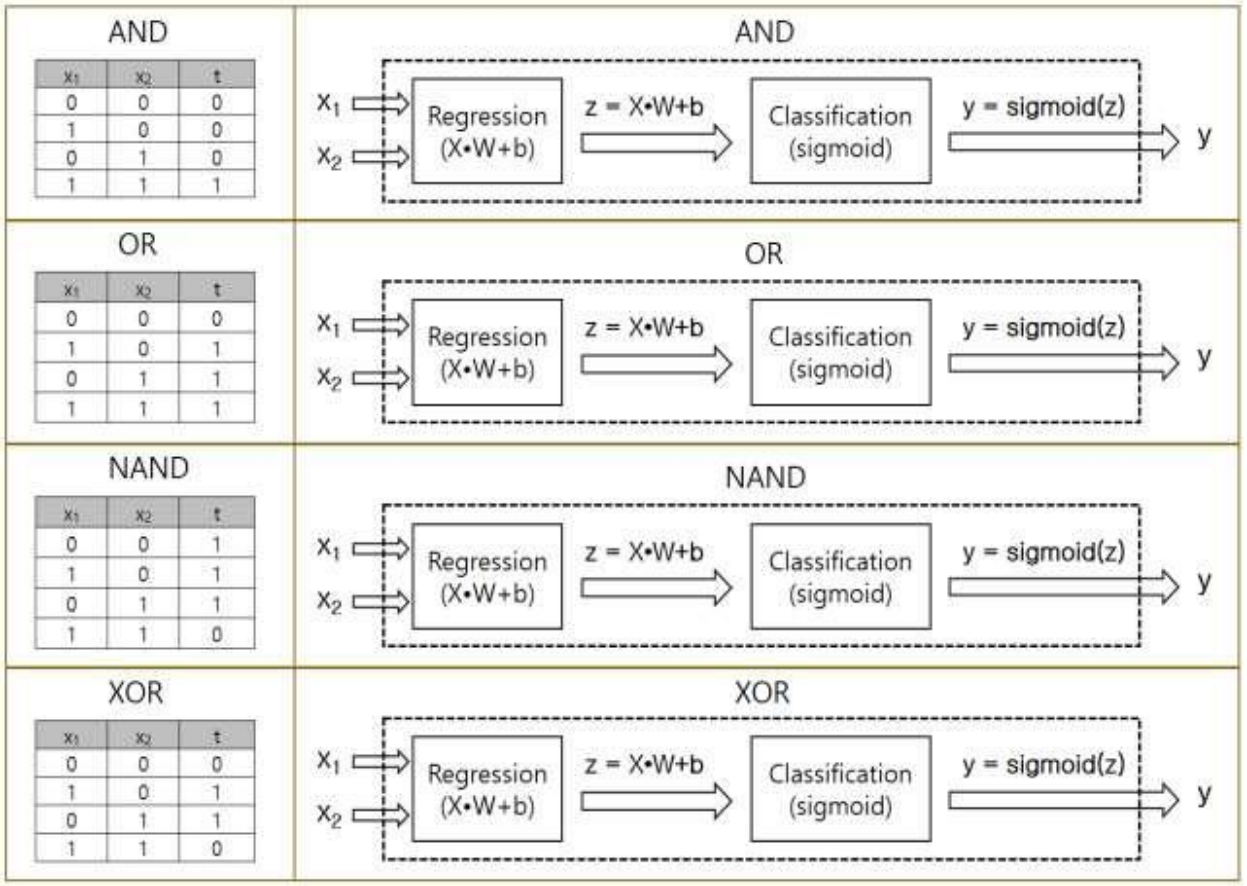
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

- XOR 게이트는 직선이 아닌 곡선. 비선형 영역으로 분리하면 구현이 가능하다.



DNN(Deep Neural Network)

- **XOR 문제**
  - **논리게이트 AND / OR / NAND / XOR**
    - AND / OR / NAND / XOR 게이트(Gate) 내부 구조는 아래 그림과 같이 선형회귀 값을 계산하는 Regression과 시그모이드 함수 값을 출력하는 Classification 으로 구성된 시스템으로 나타낼 수 있다.



DNN(Deep Neural Network)

- 논리게이트 (Logic gate) 클래스 구현

```
import numpy as np
def sigmoid(x):
    return 1. / (1. + np.exp(-x))

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] = float(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
```

DNN(Deep Neural Network)

```
class LogicGate:
    def __init__(self, gate_name, xdata, tdata):
        self.name = gate_name
        self.xdata = xdata.reshape(4,2) # 입력 데이터 초기화
        self.tdata = tdata.reshape(4,1) # 정답 데이터 초기화
        self.W = np.random.rand(self.xdata.shape[1], 1) # 가중치 W 초기화
        self.b = np.random.rand(1) # 바이어스 b 초기화
        self.learning_rate = 1e-2 # 학습률 learning rate 초기화

    def loss_func(self):
        delta = 1e-7 # log 무한대 발산 방지
        z = np.dot(self.xdata, self.W) + self.b
        y = sigmoid(z)
        return -np.sum(self.tdata*np.log(y+delta)+(1-self.tdata)*np.log((1 - y)+delta))

    def train(self): # 경사하강법 이용하여 W, b 업데이트
        f = lambda x : self.loss_func()
        print("Initial loss value = ", self.loss_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 % 1000 == 0):
                print("step = ", step, "loss value = ", self.loss_val())
```

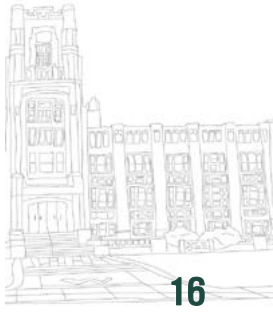
DNN(Deep Neural Network)

```
def predict(self, input_data): # 미래 값 예측
    z = np.dot(input_data, self.W) + self.b
    y = sigmoid(z)
    if y > 0.5:
        result = 1
    else:
        result = 0
    return y, result

# 정확도 예측 함수(추가 내용)
def accuracy(self, test_xdata, test_tdata):
    matched_list = []
    not_matched_list = []
    for index in range(len(xdata)):
        (real_val, logical_val) = self.predict(test_xdata[index])
        if logical_val == test_tdata[index]:
            matched_list.append(index)
        else:
            not_matched_list.append(index)
    accuracy_val = len(matched_list) / len(test_xdata)

    return accuracy_val
```

DNN(Deep Neural Network)

- AND 논리 게이트 검증

```
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()

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)

test_tdata = np.array([ 0, 0, 0, 1])
accuracy_ret = AND_obj.accuracy(test_xdata, test_tdata)
print("Accuracy => ", accuracy_ret)
```



DNN(Deep Neural Network)

- OR 논리 게이트 검증

```
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()

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)

print('--------------------')
test_tdata = np.array([ 0, 1, 1, 1])
accuracy_ret = OR_obj.accuracy(test_xdata, test_tdata)
print("Accuracy => ", accuracy_ret)
```

DNN(Deep Neural Network)

- NAND 논리 게이트 검증

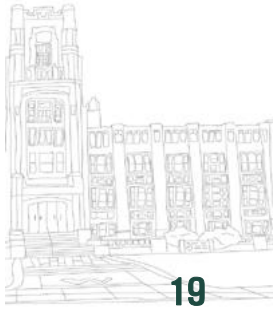
```
xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
tdata = np.array([1, 1, 1, 0])
NAND_obj = LogicGate("NAND_GATE", xdata, tdata)
NAND_obj.train()

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)

print('--------------------')
test_tdata = np.array([ 1, 1, 1, 0])
accuracy_ret = NAND_obj.accuracy(test_xdata, test_tdata)
print("Accuracy => ", accuracy_ret)
```

DNN(Deep Neural Network)

- XOR 논리 게이트 검증

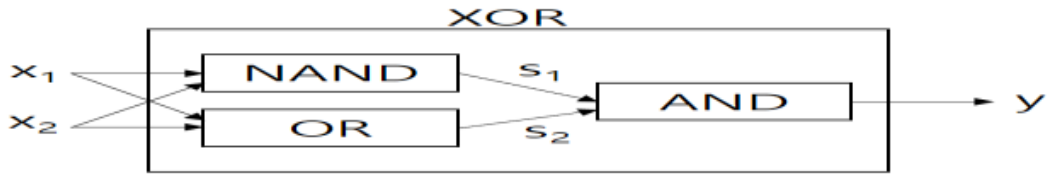
```
xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
tdata = np.array([0, 1, 1, 0])
XOR_obj = LogicGate("XOR_GATE", xdata, tdata)
XOR_obj.train()

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)

print('--------------------')
test_tdata = np.array([ 0, 1, 1, 0])
accuracy_ret = XOR_obj.accuracy(test_xdata, test_tdata)
print("Accuracy => ", accuracy_ret)
```

– 학습을 마친 후에 총 4 개의 데이터(test_data)에 대한 predict() 메서드의 예측 값을 출력해보면, 검증결과에 나타난 것처럼 XOR 논리 게이트 각 입력 값에 대해 결과값이 일치하지 않는다. 즉 XOR 논리 게이트는 우리가 알아보았던 분류(Classification) 알고리즘만으로는 분류하기 어렵다는 것을 알 수 있다.



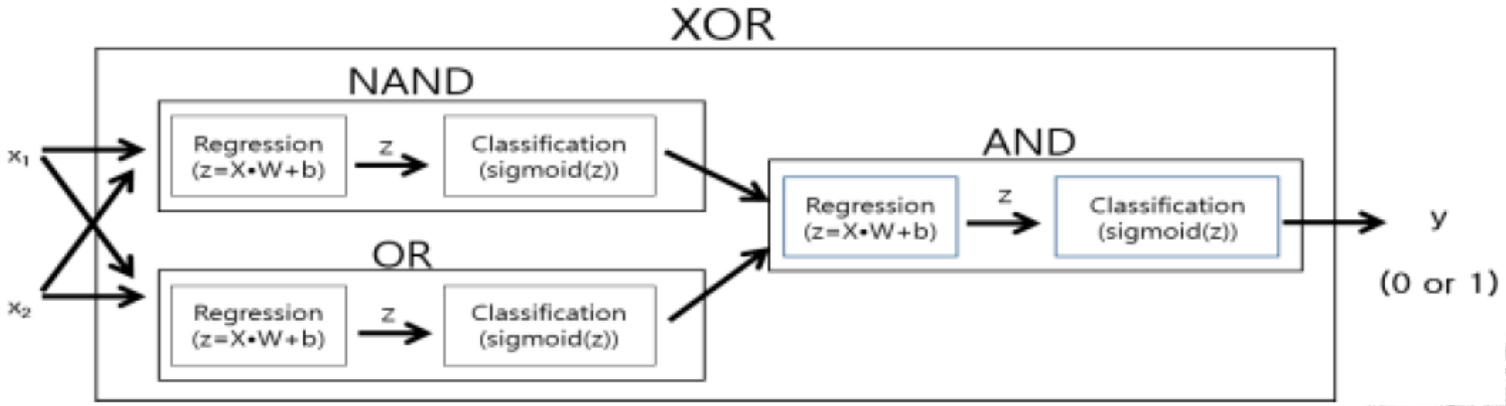
DNN(Deep Neural Network)

- **XOR 해결 방법**



x1	x2	s1 (NAND)	s2 (OR)	y (XOR)
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

- **즉 2 개 입력 [x1 , x2]에 대한 NAND 출력을 s1, OR 출력을 s2 라고 하고 이러한 s1 과 s2를 AND 입력으로 주어 최종 출력을 계산하면 XOR 논리 게이트를 구현할 수 있다는 원리이다.**



- **(문제) NAND / OR / AND 조합을 이용한 XOR 논리 게이트 학습 및 검증 코드 작성하기**