zero-base /

Life Changing Education

Chapter 42. Deep Learning from scratch

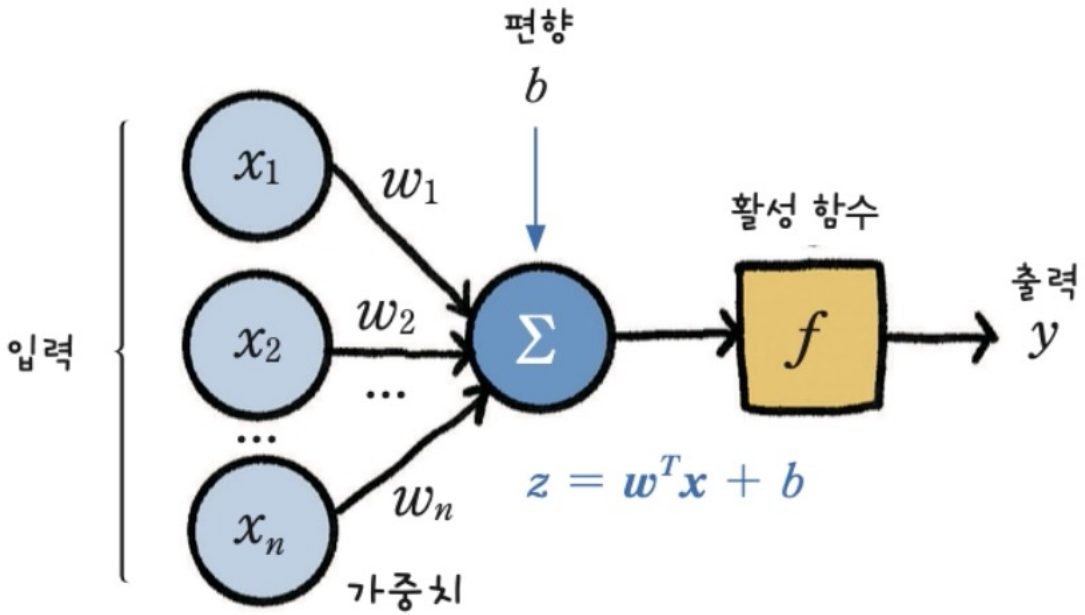
딥러닝 쌩~으로 이해하기

Chapter

zero-base /

순방향 연산 – 그것이 추론~

입력을 주고 출력을 관찰하는 것이 추론, 순방향 연산

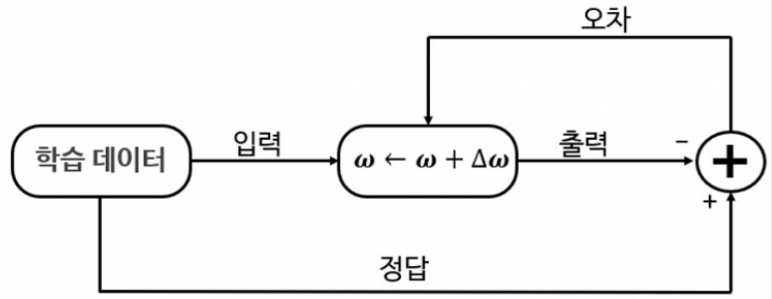


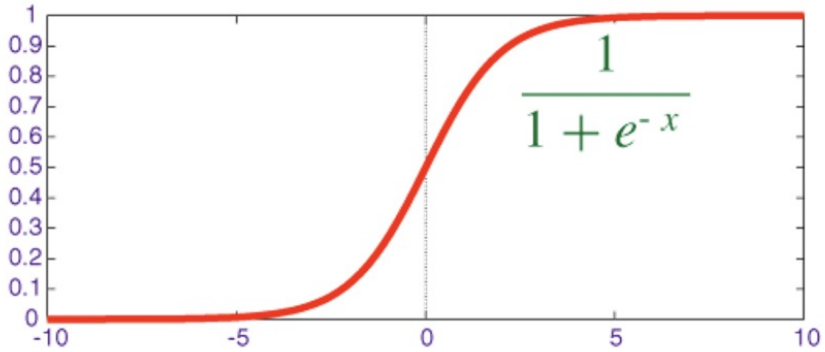
데이터를 준비해 주고

```
import numpy as np

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

활성화 함수를 준비하고 - sigmoid 함수

```
def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))
```

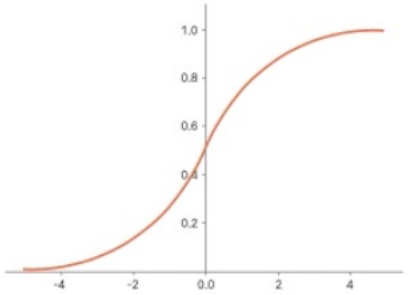




활성화 함수의 종류

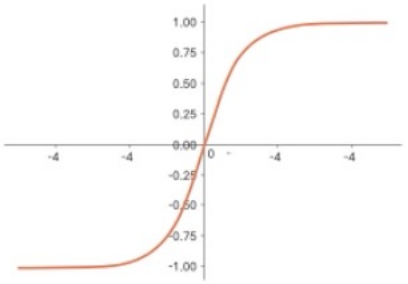
시그모이드

$$\sigma(x)=\frac{1}{1+e^{-x}}$$



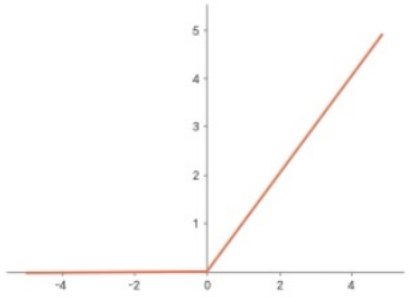
하이퍼볼릭 탄젠트

$$\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$



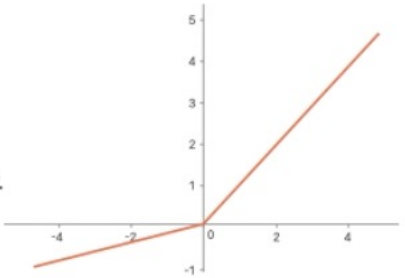
ReLU

$$f(x)=\begin{cases} x & if\ x\geq 0 \\ 0 & \text{그 외의 경우} \end{cases}$$



리키 ReLU

$$f(x)=\begin{cases} x & if\ x\geq 0 \\ 0.01x & \text{그 외의 경우} \end{cases}$$

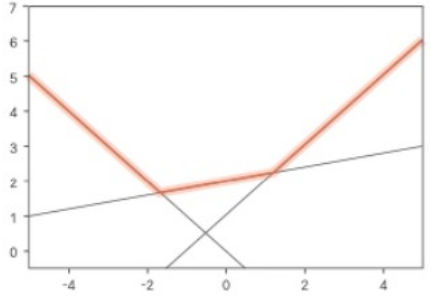


맥스아웃

$$\max\left(\boldsymbol{w}_i^{T}x+b_i\right)$$

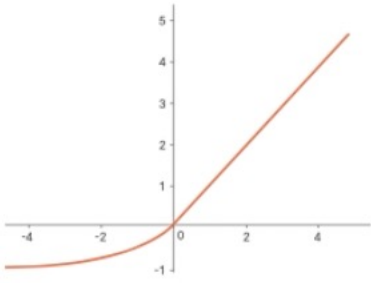
$$i=1,\ 2,\ \cdots,\ k$$

k: 선형구간 개수



ELU

$$\begin{cases} x & if\ x\geq 0 \\ \alpha(e^{x}-1) & \text{그 외의 경우} \end{cases}$$



가중치를 랜덤하게 선택하자.
원래는 학습이 완료된 가중치를 사용해야 한다

```
W = 2*np.random.random((1, 3)) - 1
W
```

```
array([[-0.00367437, -0.0481017 ,  0.94802035]])
```

추론 결과

```
N = 4
for k in range(N):
    x = X[k, :].T
    v = np.matmul(W, x)
    y = sigmoid(v)

    print(v)
```

```
[0.94802035]
[0.89991864]
[0.94434598]
[0.89624427]
```

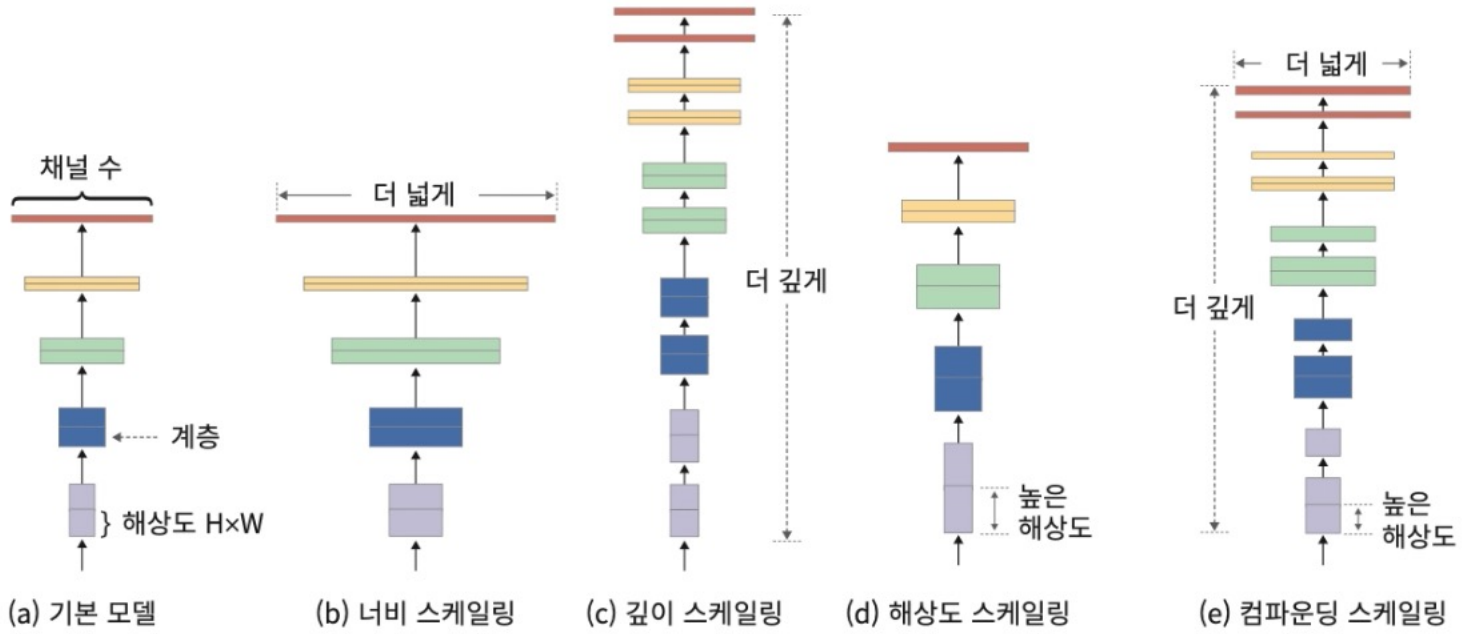
가중치가 이제 정답을 맞추도록 학습을 시켜야 한다

일단 정답을 주자 - AND

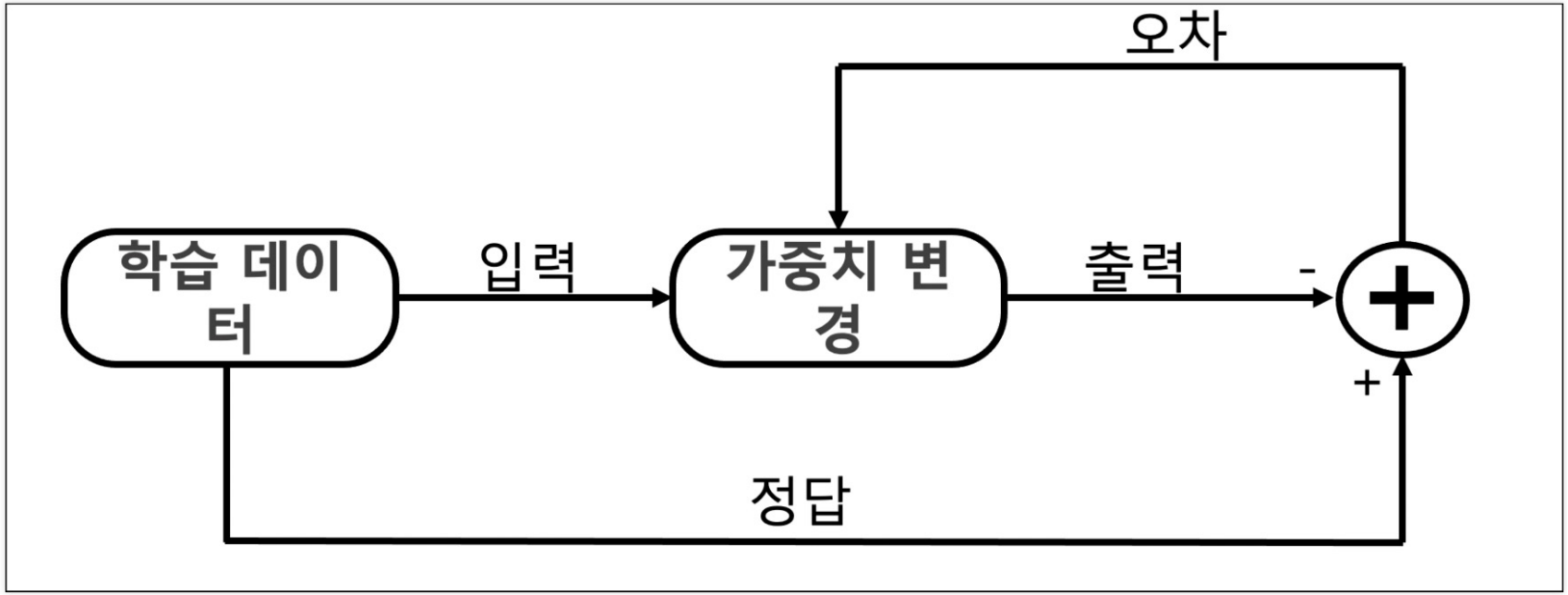
```
import numpy as np

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

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

zero-base /

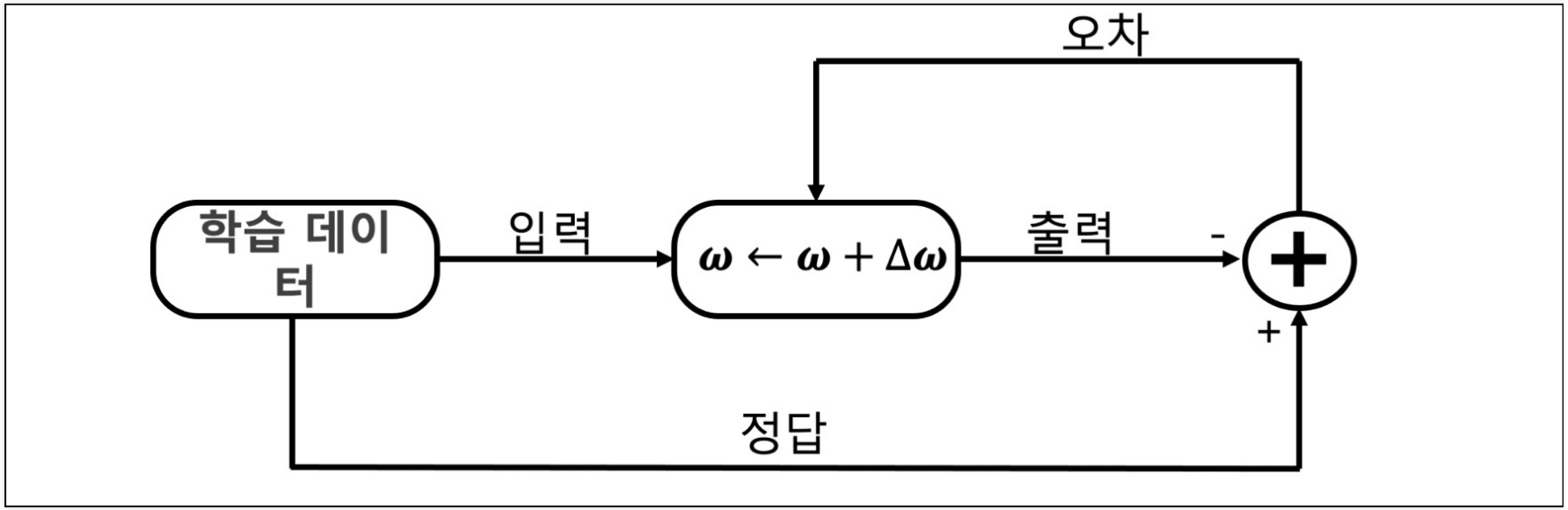
딥러닝 모델의 큰 규모



딥러닝 학습의 가장 간단한 절차

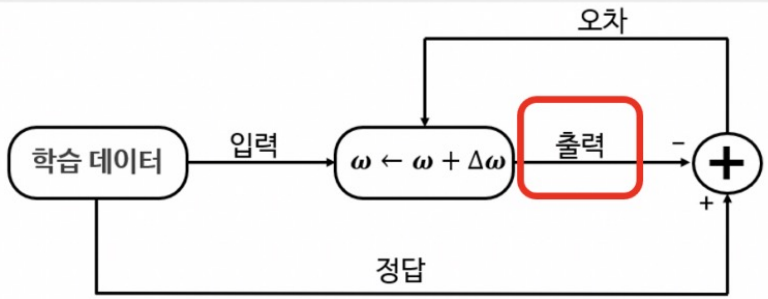


가중치를 어떻게 갱신하지?



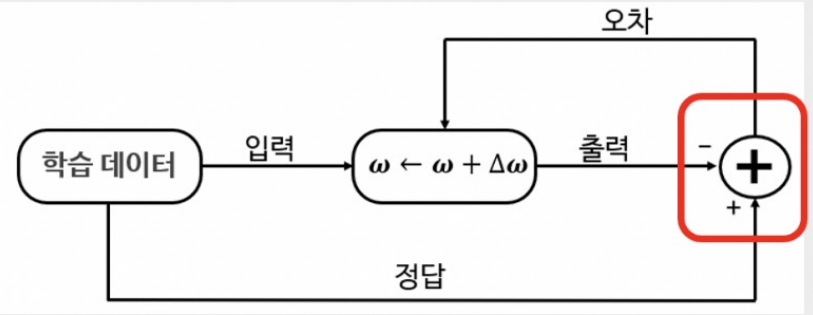
일단 모델의 출력을 계산하는 함수

```
def calc_output(W,x):
    v = np.matmul(W, x)
    y = sigmoid(v)

    return y
```



오차를 계산.. 그런데

```
def calc_error(d, y):
    e = d - y
    delta = y*(1-y) * e

    return delta
```



zero-base /

시그모이드의 미분은 이렇게 생김

```
import sympy as sym

z = sym.Symbol('z')
s = 1 / (1 + sym.exp(-z))

sym.diff(s)
```

$$\frac{e^{-z}}{(1+e^{-z})^2}$$

sigmoid의 미분

$$
\begin{aligned}
\frac{d}{dx} sigmoid(x) &= \frac{d}{dx}(1+e^{-x})^{-1} \\
&= (-1)\frac{1}{(1+e^{-x})^2}\frac{d}{dx}(1+e^{-x}) \\
&= (-1)\frac{1}{(1+e^{-x})^2}(0+e^{-x})\frac{d}{dx}(-x) \\
&= (-1)\frac{1}{(1+e^{-x})^2}e^{-x}(-1) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1+e^{-x}-1}{(1+e^{-x})^2} \\
&= \frac{(1+e^{-x})}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}}(1-\frac{1}{1+e^{-x}}) \\
&= sigmoid(x)(1-sigmoid(x))
\end{aligned}
$$

한 epoch에 수행되는 W의 계산

```
def delta_GD(W, X, D, alpha):
    for k in range(4):
        x = X[k, :].T
        d = D[k]

        y = calc_output(W,x)
        delta = calc_error(d, y)

        dW = alpha*delta*x
        W = W + dW

    return W
```

zero-base /

가중치를 랜덤하게 초기화하고 학습 시작~

```
W = 2*np.random.random((1, 3)) - 1
```

```
alpha = 0.9
for epoch in range(10000):
    W = delta_GD(W, X, D, alpha)
    print(W)
```

```
[[ 9.56373383 -0.20888086 -4.57421841]]
[[ 9.56383654 -0.20888053 -4.57427003]]
[[ 9.56393924 -0.2088802  -4.57432164]]
```

가중치는 업데이트가 된다

```
[[ 0.71587794  0.43684654  0.19604547]]
[[ 0.82698445  0.3470833   0.05300676]]
[[ 0.94486098  0.26879669 -0.07035605]]
[[ 1.06560051  0.20256763 -0.17534932]]
[[ 1.18600488  0.14729025 -0.26497327]]
[[ 1.30382016  0.10120609 -0.34253299]]
[[ 1.41764723  0.06255945 -0.4108846 ]]
```

가중치가 변해가는 과정

```
[[ 9.564934   -0.2084175  -4.57509576]]
[[ 9.56503659 -0.20841721 -4.57514729]]
[[ 9.56513916 -0.20841693 -4.57519881]]
[[ 9.56524173 -0.20841665 -4.57525033]]
[[ 9.56534428 -0.20841637 -4.57530185]]
[[ 9.56544683 -0.20841609 -4.57535336]]
```

결과 확인

```
N = 4
for k in range(N):
    x = X[k,:].T
    v = np.matmul(W, x)
    y = sigmoid(v)
    print(y)
```

```
[0.01020176]
[0.00829464]
[0.99324188]
[0.99168508]
```

XOR

zero-base /

XOR 데이터 다시 던져주고~

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

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

W = 2*np.random.random((1, 3)) - 1
```

zero-base /

학습하고

```
alpha = 0.9
for epoch in range(10000):      #train
    W = delta_GD(W, X, D, alpha)
```

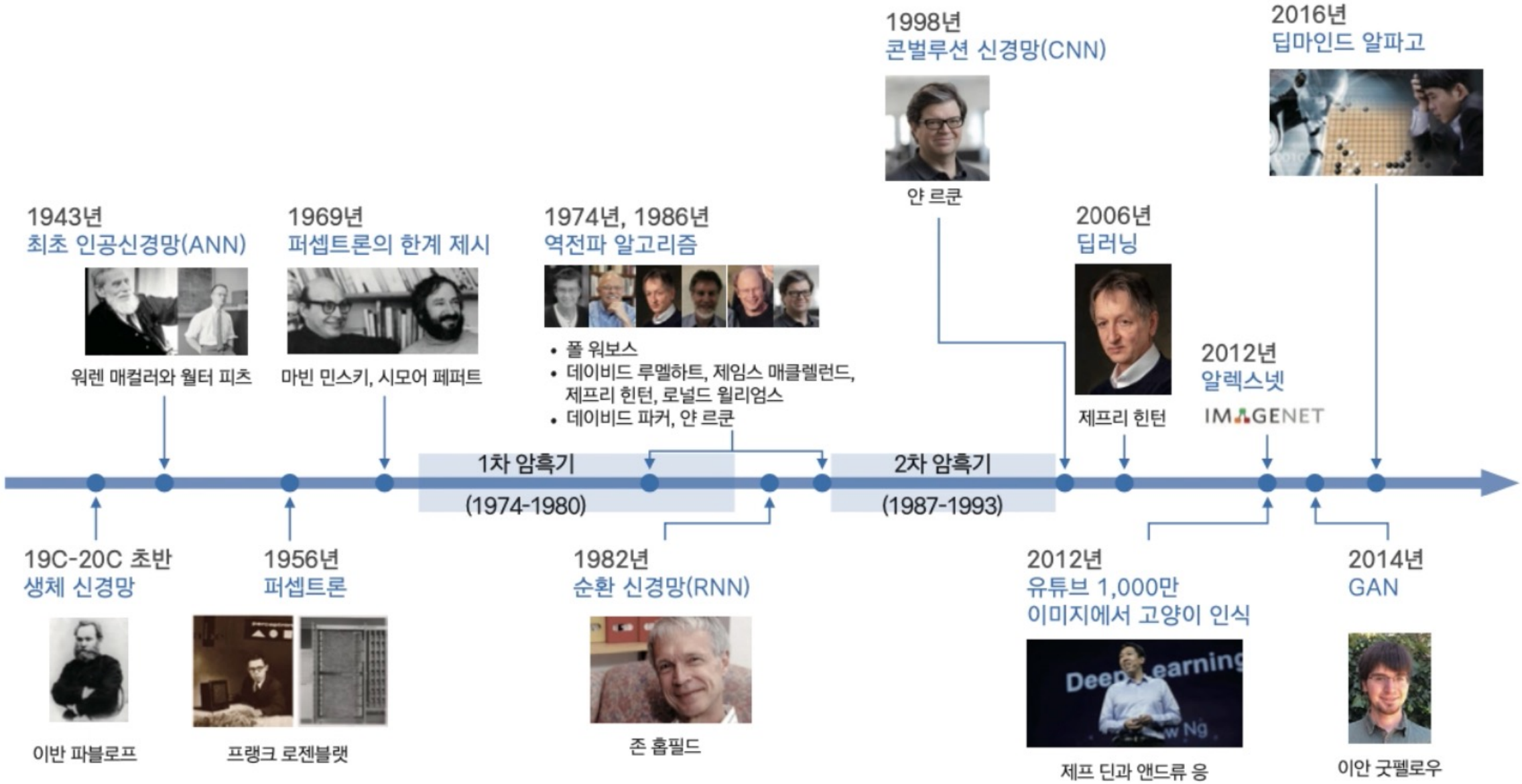
zero-base /

결과 확인 - 엉망

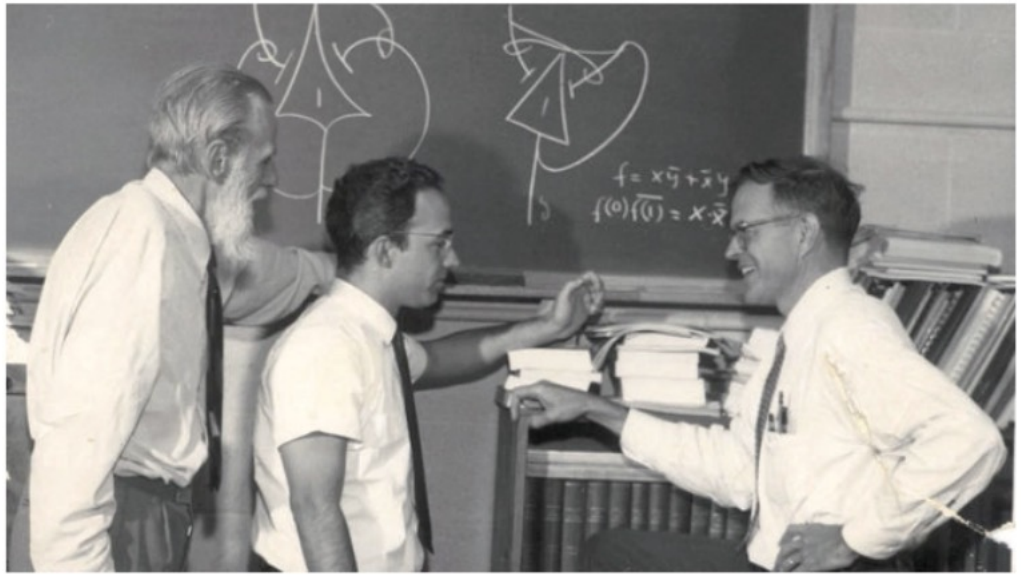
```
N = 4                                   #inference
for k in range(N):
    x = X[k,:].T
    v = np.matmul(W, x)
    y = sigmoid(v)
    print(y)
```

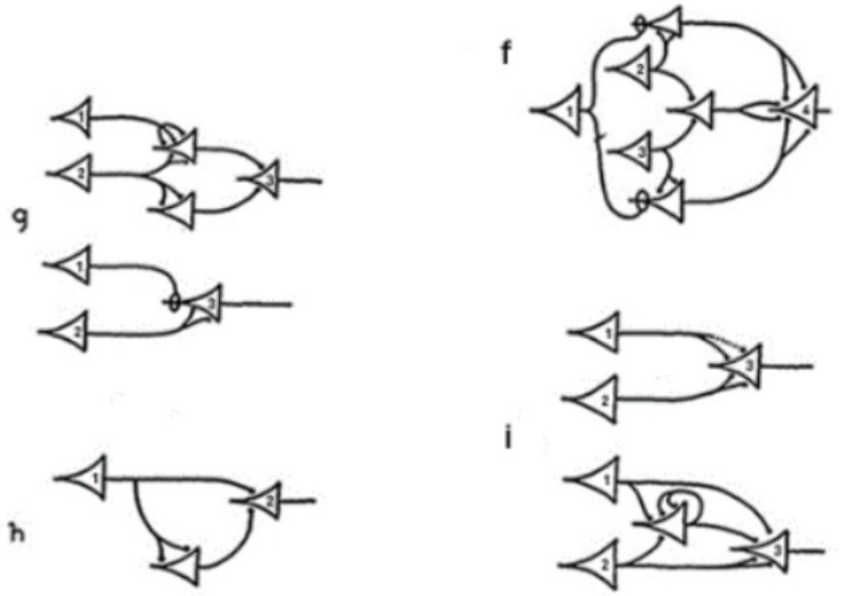
```
[0.52965337]
[0.5]
[0.47034663]
[0.44090112]
```

다시보는 역사



최초의 신경망 – 매컬러 피츠 모델

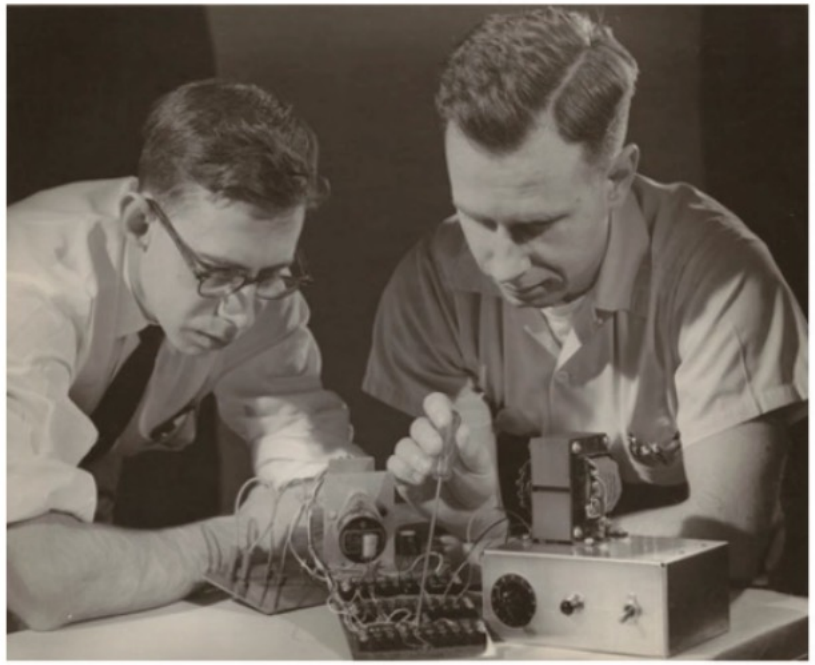


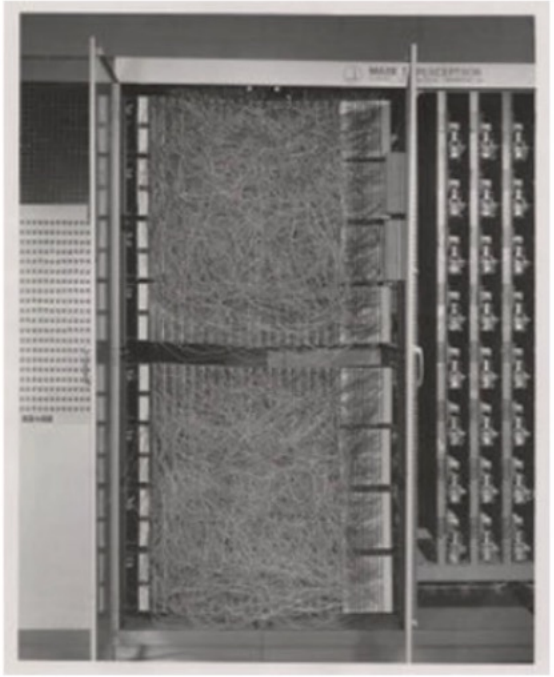


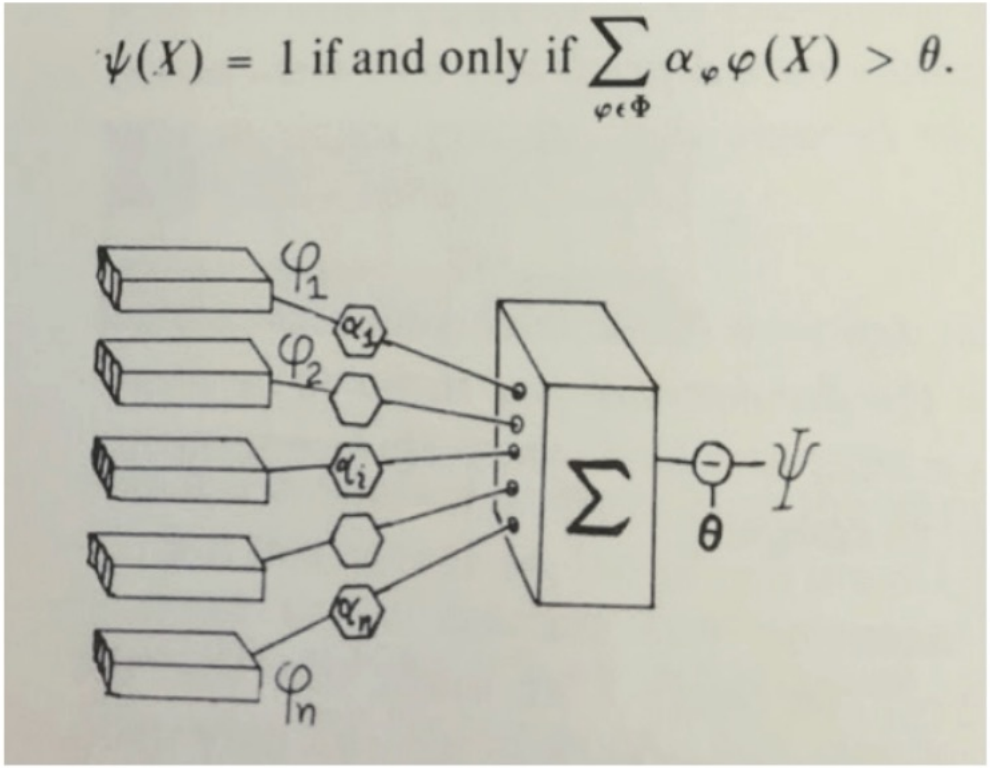
Chapter

zero-base /

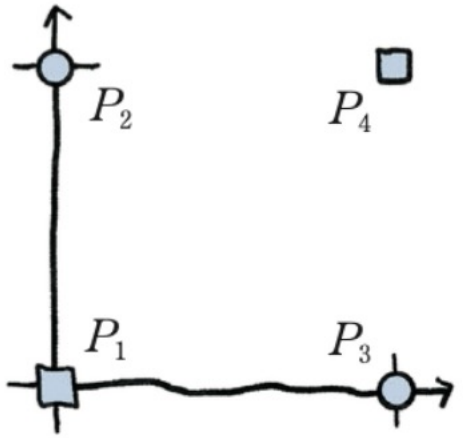
로젠블랏의 퍼셉트론







XOR 문제로 한계를 지적한 민스키 – 책이름이 퍼셉트론.ㅠㅠ



$$\left\{P_1=\begin{bmatrix}0\\0\end{bmatrix},\ t_1=0\right\}\quad \left\{P_2=\begin{bmatrix}0\\1\end{bmatrix},\ t_2=1\right\}$$

$$\left\{P_3=\begin{bmatrix}1\\0\end{bmatrix},\ t_3=1\right\}\quad \left\{P_4=\begin{bmatrix}1\\1\end{bmatrix},\ t_4=0\right\}$$

역전파 알고리즘 발견

BEYOND REGRESSION:
NEW TOOLS FOR PREDICTION AND ANALYSIS
IN THE BEHAVIORAL SCIENCES

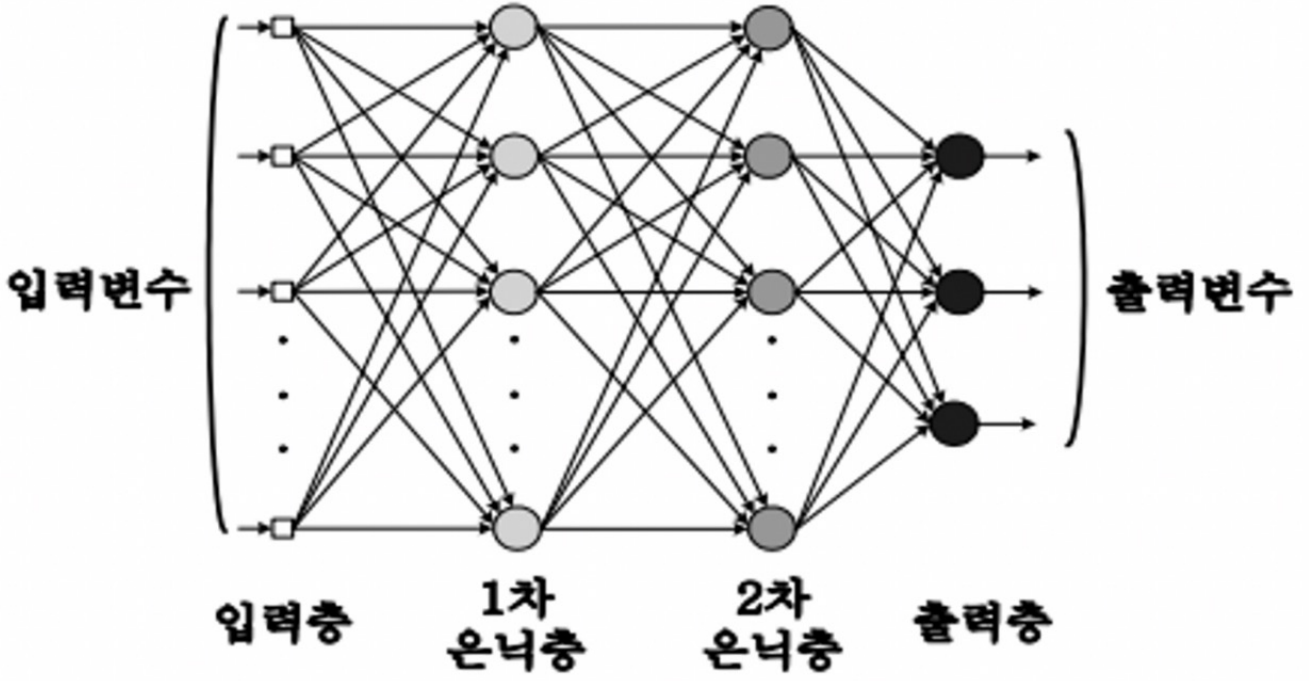
A thesis presented
by
Paul John Werbos
to
The Committee on Applied Mathematics
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
statistics

Harvard University
Cambridge, Massachusetts
August, 1974

역전파

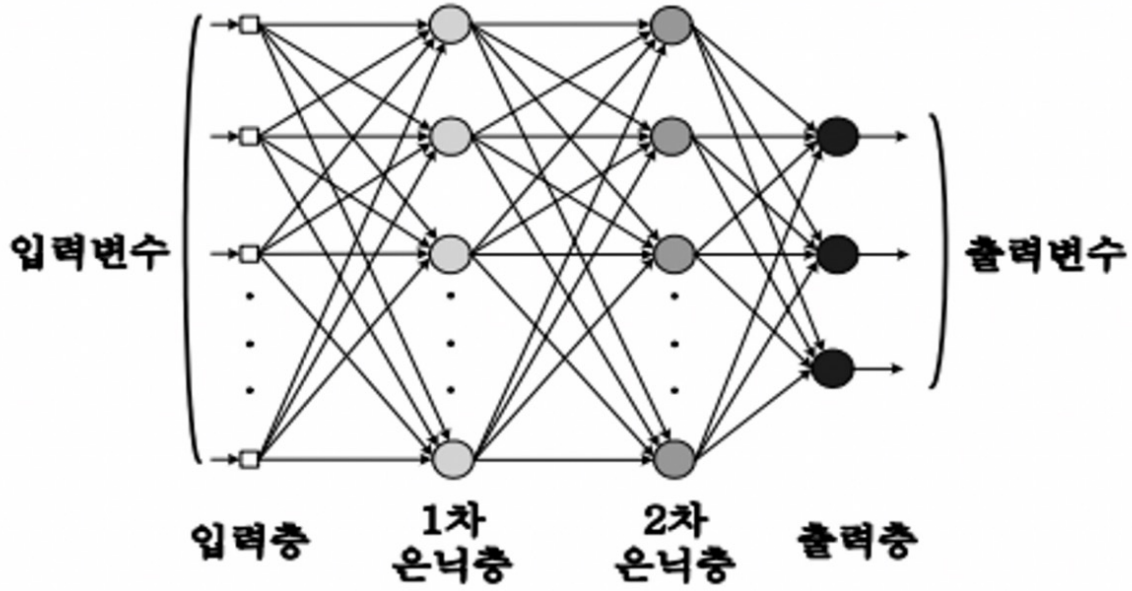
zero-base /

XOR 극복하기



다층, 심층 신경망을 이용하면 한계를 극복할 수 있다...

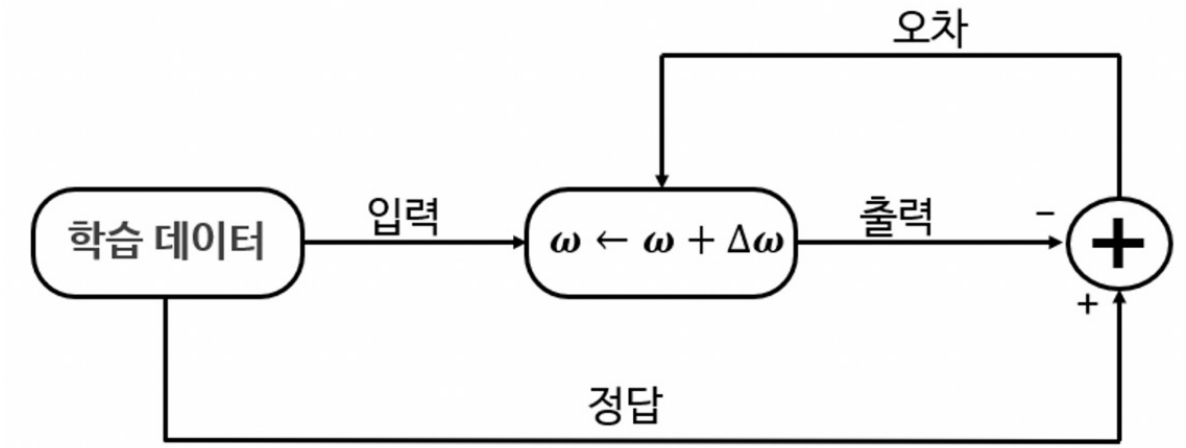
는 잘 안됨



다층, 심층 신경망을 이용하면 한계를 극복할 수 있는 줄 알았다...

zero-base /

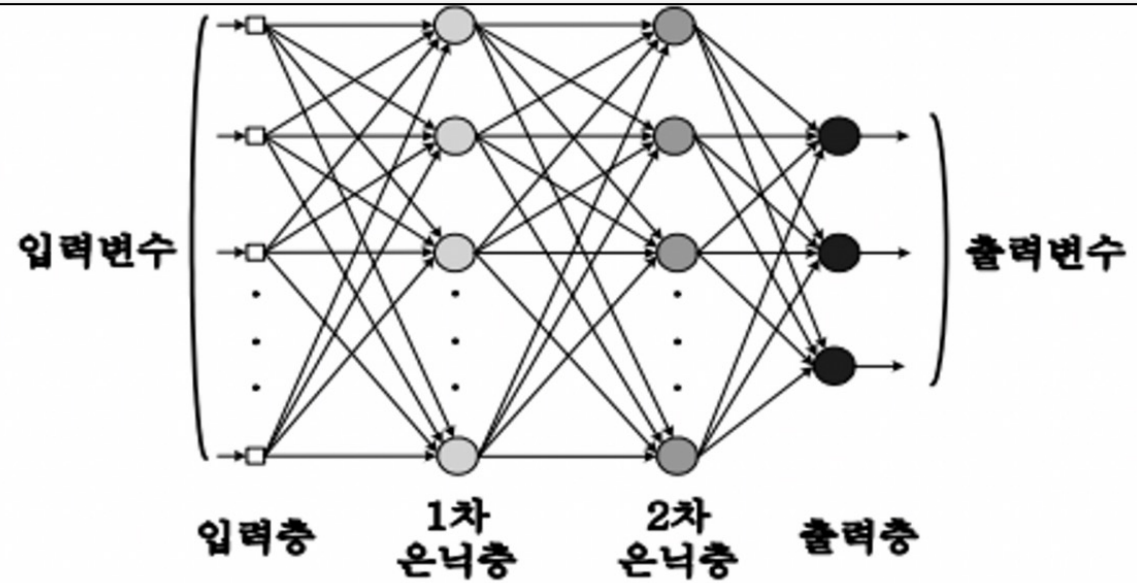
단층은 정답과 현재값의 오차를 계산할 수 있다



단층의 경우 정답과 출력의 오차를 계산해서 가중치를 갱신할 수 있다.

zero-base /

다층 신경망은 그럴 수 없다



다층의 경우 1차, 2차 은닉층의 경우 오차를 계산할 수 없다... 왜??

→ 정답을 모르니까... → 그러면 은닉층의 가중치를 갱신할 수 없다.

그래서 AI winter

그래서...

1960년대에 만들어진

인공 신경망은 침체기를 가진다...

Backpropagation

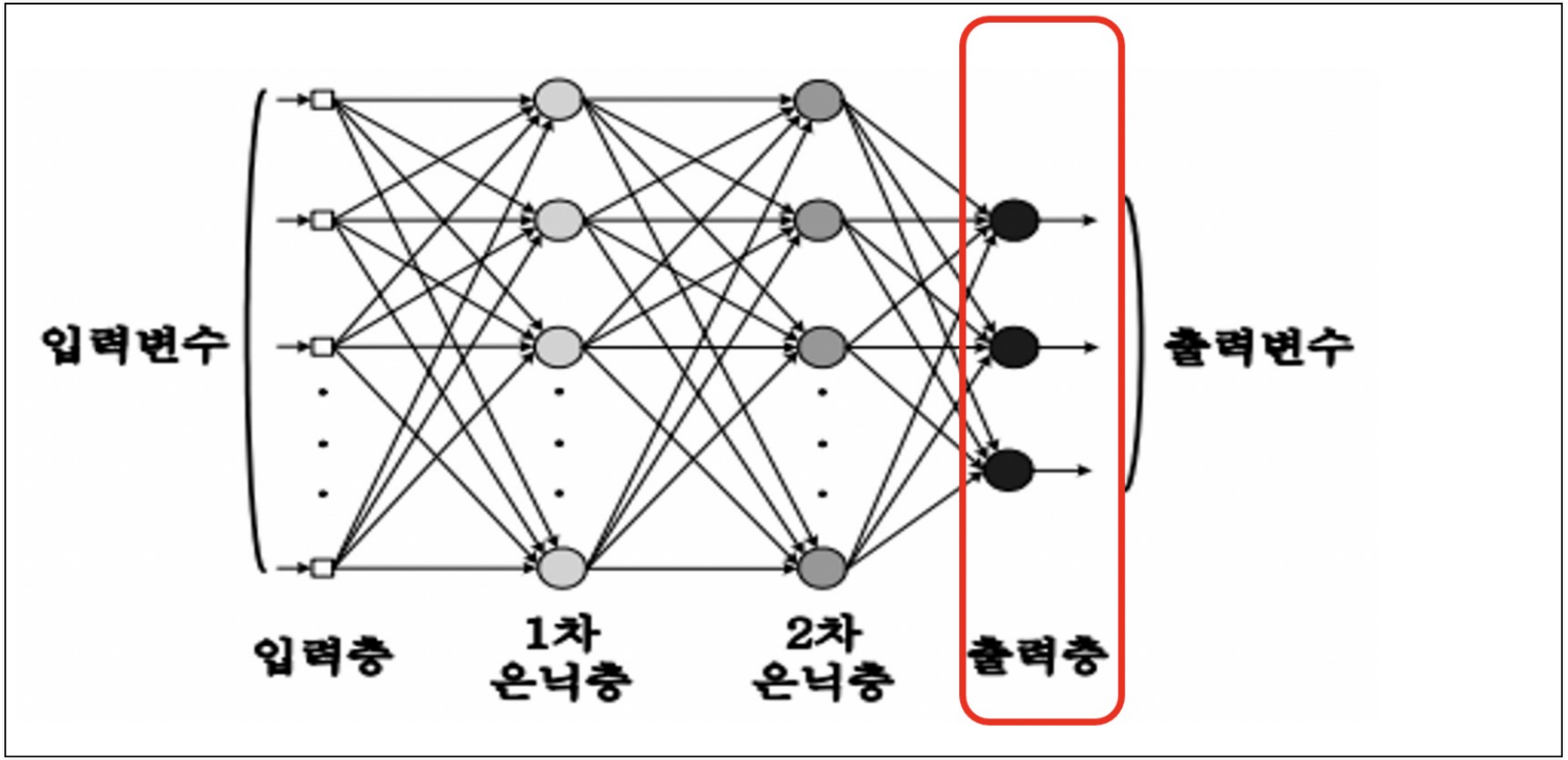
엄청난 일이 생김... 1986년에...

역전파

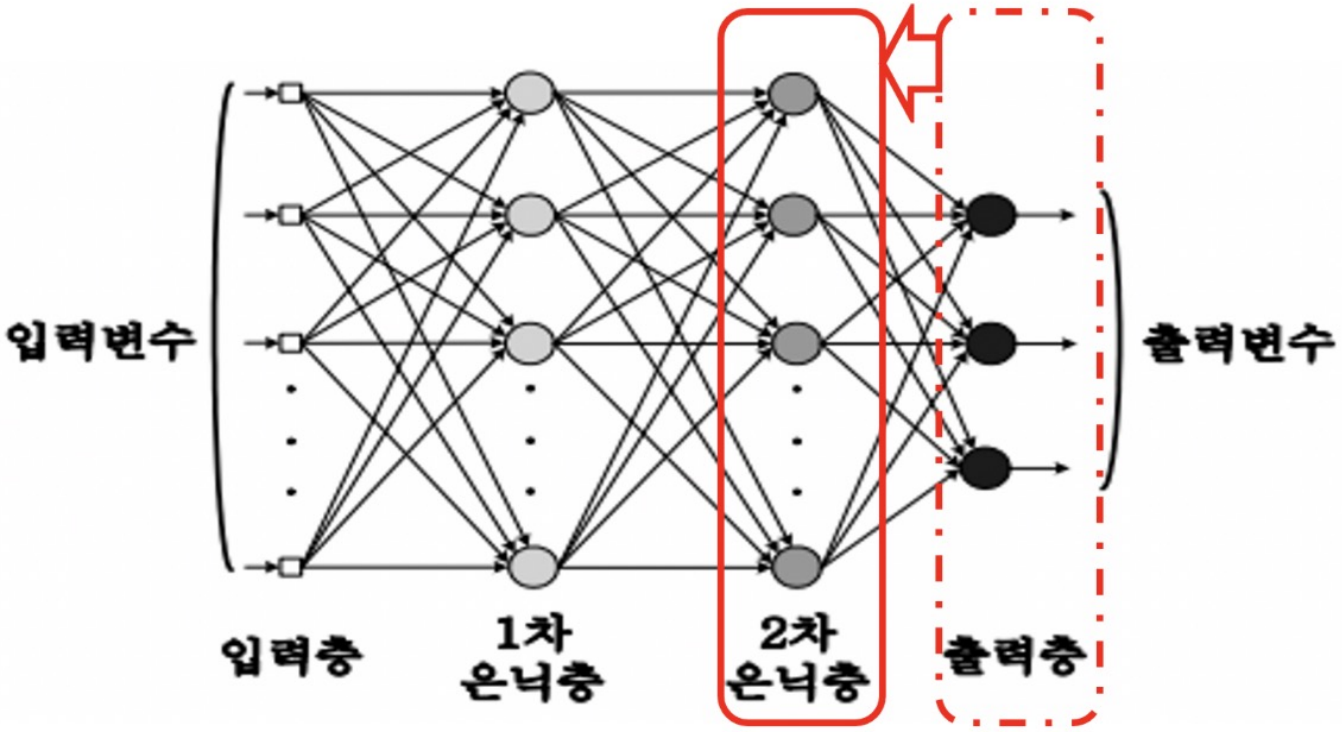
Backpropagation 개념이 개발...

zero-base /

먼저 출력층의 오차 계산



출력층 가까이 있는 은닉층의 가중치와 델타 계산

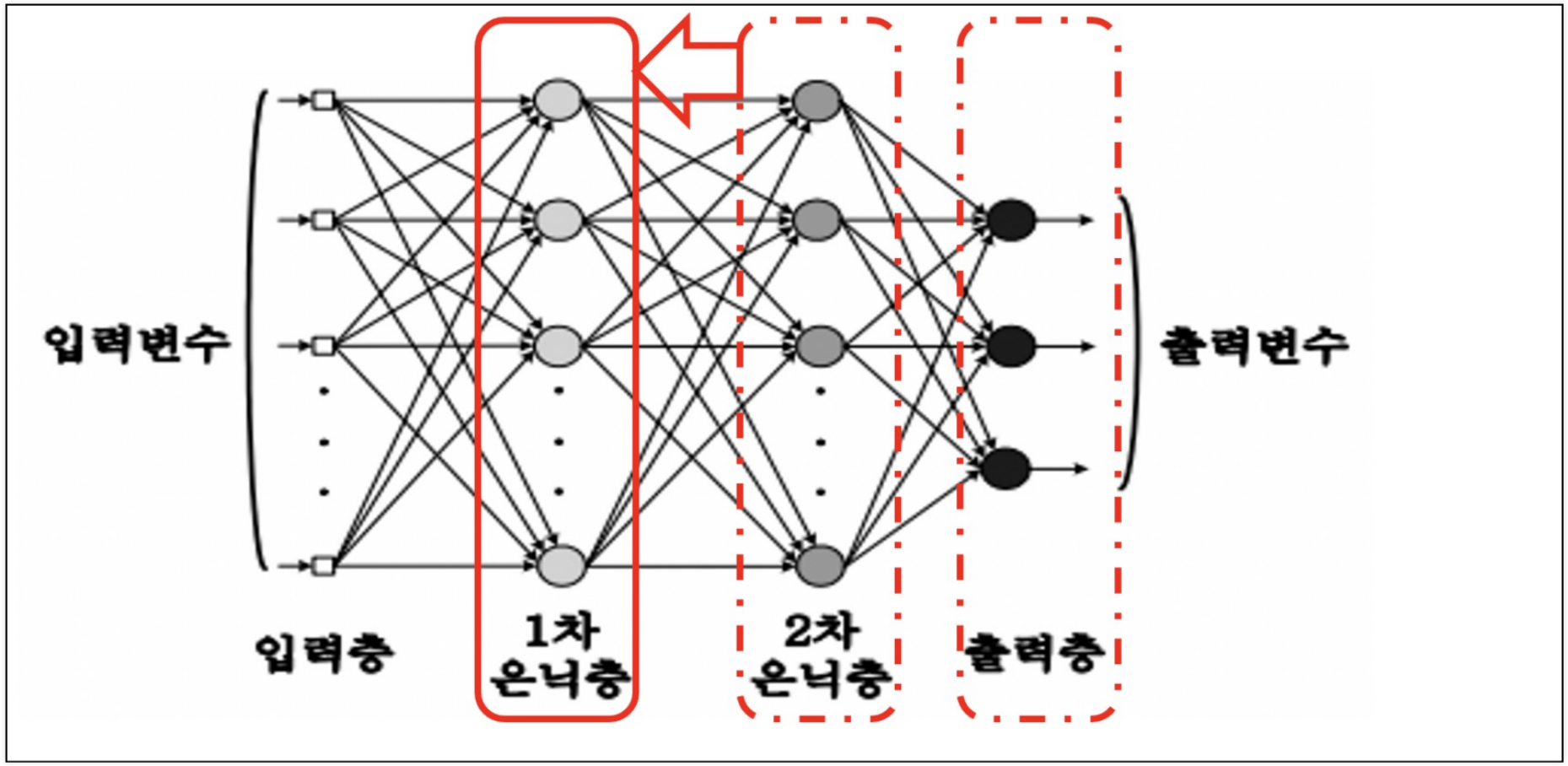


2차 은닉층의 가중치를 역산하고...

2차 은닉층 출력과 델타를 계산한다.

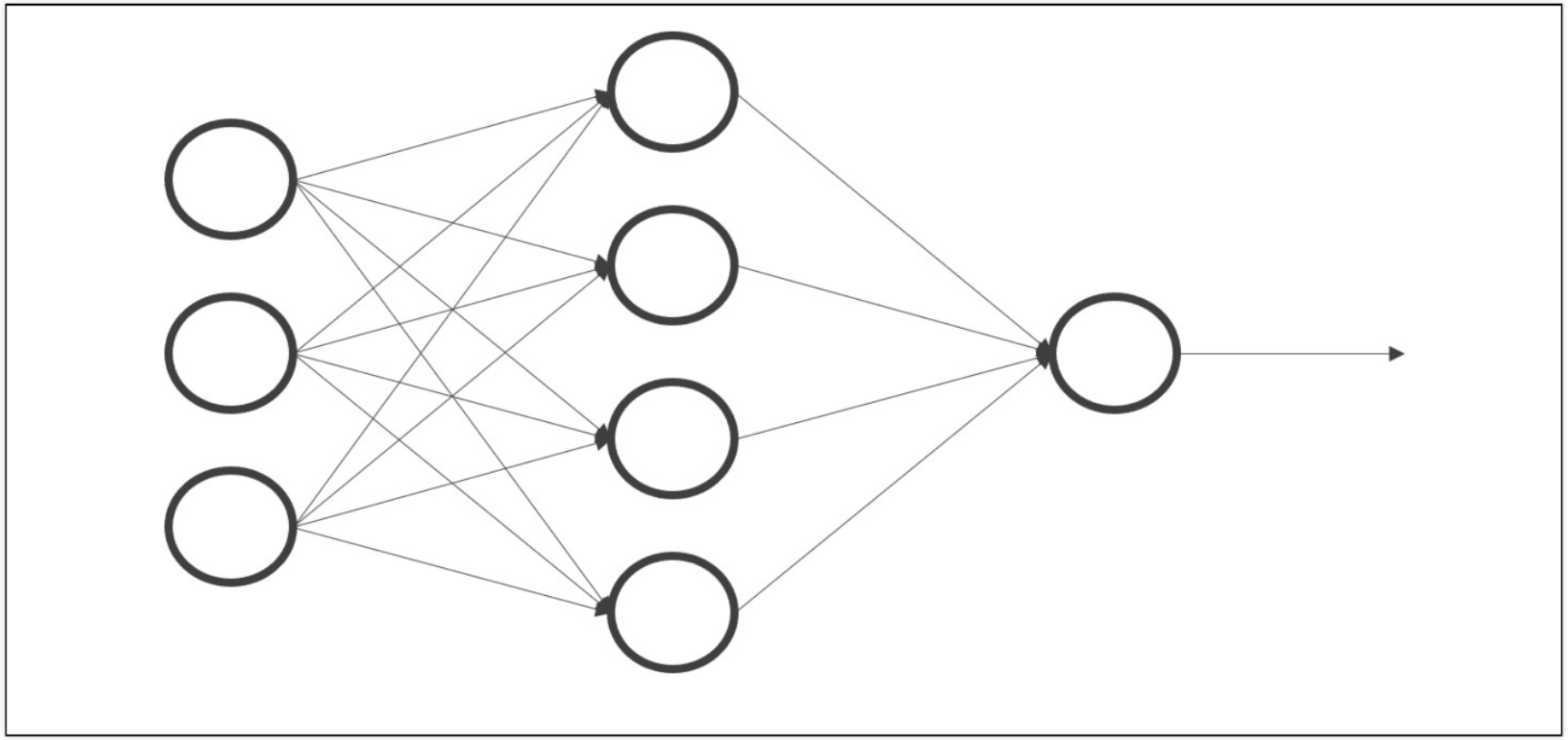
zero-base /

다시 그 전 은닉층으로 계산



zero-base /

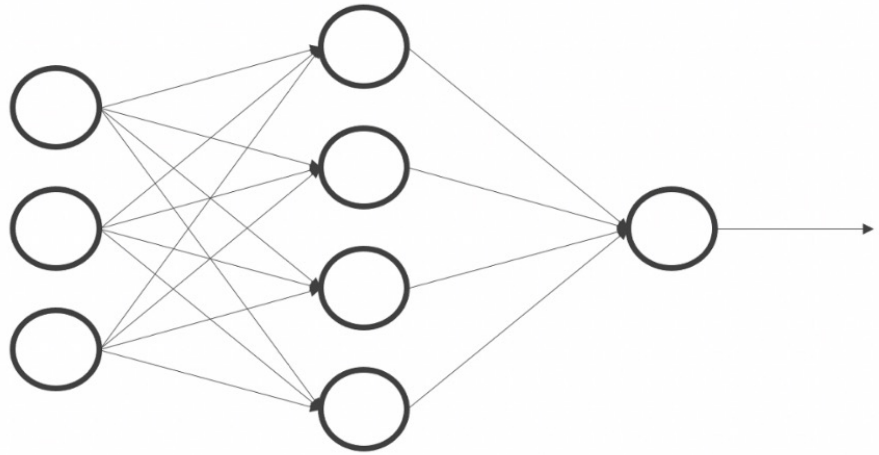
다시 새로운 구조



zero-base /

output 계산 함수

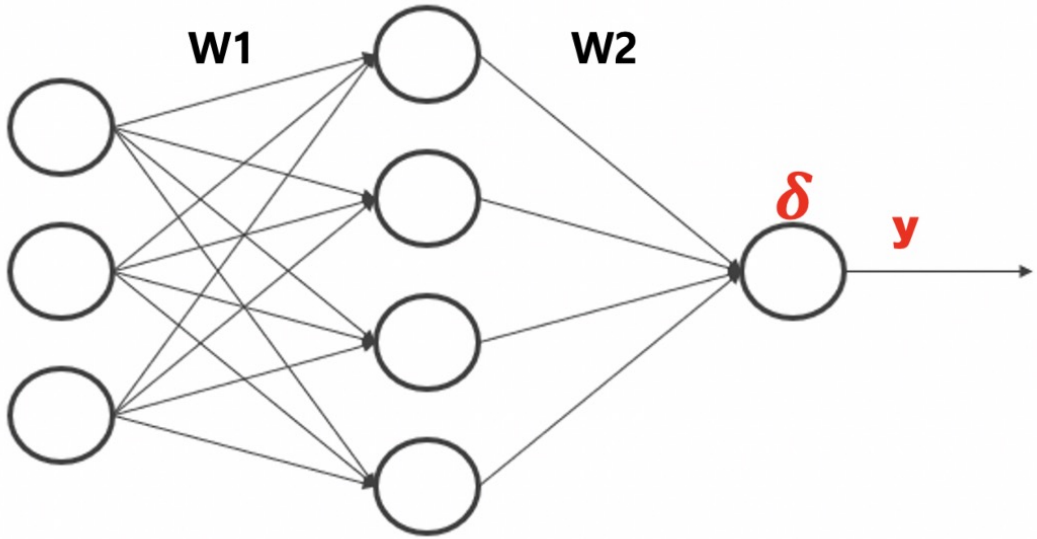
```
def calc_output(W1, W2, x):
    v1 = np.matmul(W1, x)
    y1 = sigmoid(v1)
    v  = np.matmul(W2, y1)
    y  = sigmoid(v)

    return y, y1
```



zero-base /

출력층의 델타 계산

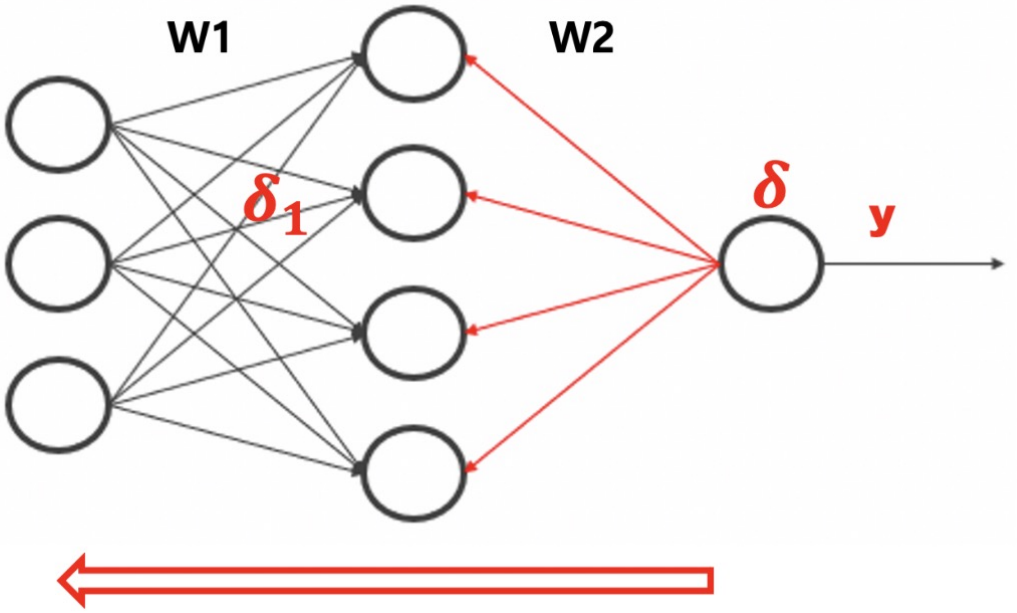
```
def calc_delta(d, y):
    e = d - y
    delta = y*(1-y) * e

    return delta
```



zero-base /

은닉층의 델타 계산

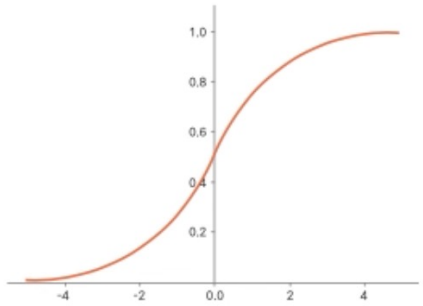
```
def calc_delta1(W2, delta, y1):
    e1 = np.matmul(W2.T, delta)
    delta1 = y1*(1-y1) * e1

    return delta1
```



몇몇 활성화 함수들의 미분값

시그모이드

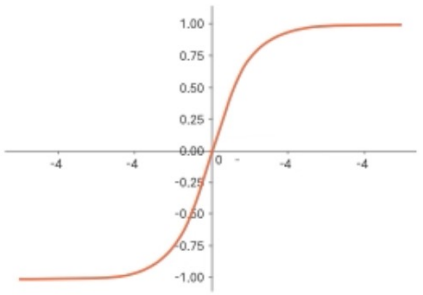
$$f(x)=\frac{1}{1+e^{-x}}$$



$$f'(x)=f(x)(1-f(x))$$

하이퍼볼릭 탄젠트

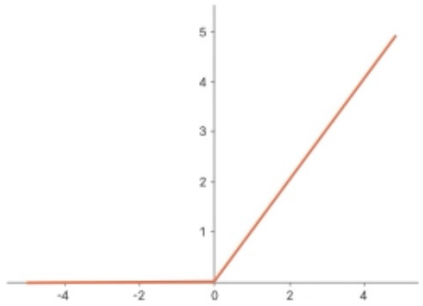
$$f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$



$$f'(x)=1-f(x)^{2}$$

ReLU

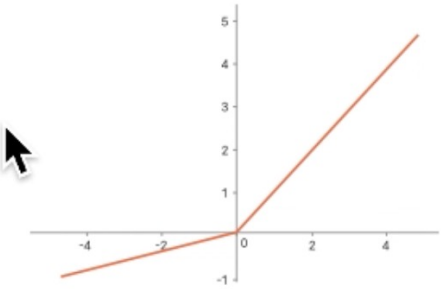
$$f(x)=\begin{cases}x & if\ x>0\\ 0 & otherwise\end{cases}$$



$$f'=\begin{cases}1 & if\ x>0\\ 0 & otherwise\end{cases}$$

리키 ReLU

$$f(x)=\begin{cases}x & if\ x>0\\ 0.01x & otherwise\end{cases}$$



$$f'(x)=\begin{cases}1 & if\ x>0\\ 0.01 & otherwise\end{cases}$$

zero-base /

역전파 코드

```
def backprop_XOR(W1, W2, X, D, alpha):
    for k in range(4):
        x = X[k, :].T
        d = D[k]

        y, y1 = calc_output(W1, W2, x)
        delta = calc_delta(d, y)
        delta1 = calc_delta1(W2, delta, y1)

        dW1 = (alpha*delta1).reshape(4, 1) * x.reshape(1, 3)
        W1  = W1 + dW1

        dW2 = alpha * delta * y1
        W2  = W2 + dW2

    return W1, W2
```

zero-base /

데이터를 준비하고 가중치를 랜덤하게 초기화

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

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

W1 = 2*np.random.random((4, 3)) - 1
W2 = 2*np.random.random((1, 4)) - 1
```

zero-base /

학습

```
alpha = 0.9
for epoch in range(10000):
    W1, W2 = backprop_XOR(W1, W2, X, D, alpha)
```

zero-base /

결과 ~ 짠

```
N = 4
for k in range(4):
    x  = X[k ,:].T
    v1 = np.matmul(W1, x)
    y1 = sigmoid(v1)
    v  = np.matmul(W2, y1)
    y  = sigmoid(v)
    print(y)
```

```
[0.0087978]
[0.98603219]
[0.98981411]
[0.01291759]
```

Chapter

zero-base /

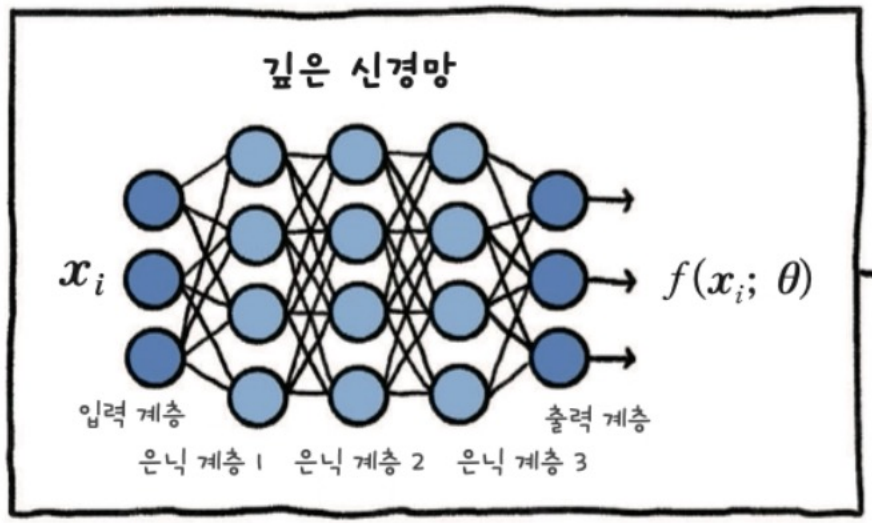
Loss 함수 교체~

zero-base /

또 하나 더~ 비용함수가 하나만 있는 것은 아니다

$$J = \sum_{i=1}^{M} (d_i - y_i)^2$$

$$J = \sum_{i=1}^{M} \{-d_i \ln(y_i) - (1 - d_i) \ln(1 - y_i)\}$$



회귀

가우시안 분포

평균제곱오차

$$J(\theta)=\frac{1}{N}\sum_{i=1}^{N}\|t_i-y(x_i;\ \theta)\|_2^2$$

N: 샘플 개수

분류

카테고리 분포

크로스 엔트로피

$$J(\theta)=-\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}t_{ik}\cdot\log\mu(x_i;\ \theta)_k$$

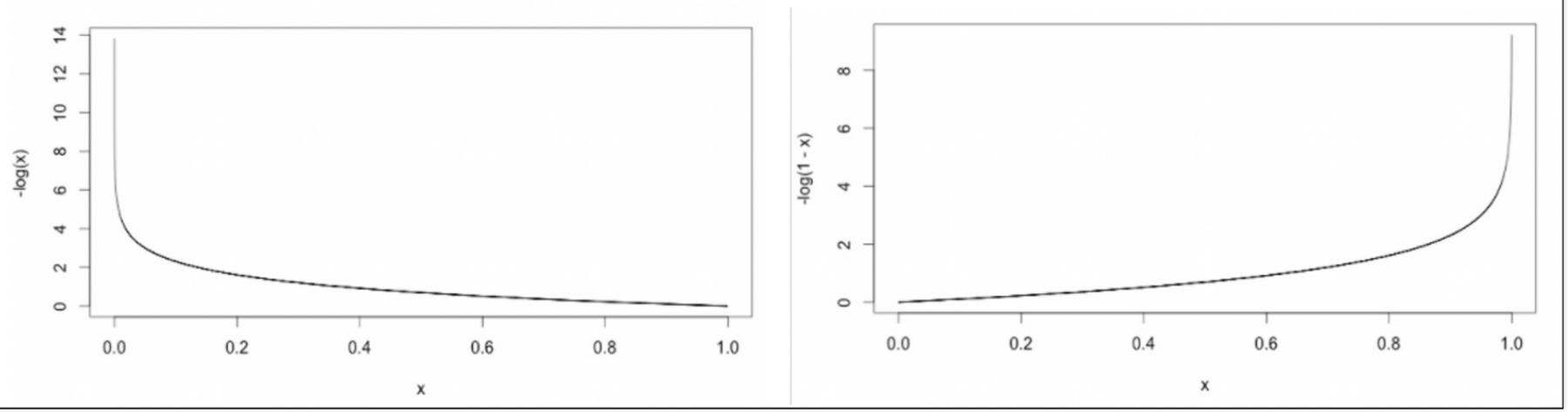
N: 샘플 개수, K: 클래스 개수

zero-base /

cross entropy

$$J = \sum_{i=1}^{M} \{-d_i \ln(y_i) - (1 - d_i) \ln(1 - y_i)\}$$

위 괄호 안의 수식을 Cross Entropy 함수라고 함



cross_entropy의 델타

In [10]:
```
def calcDelta_ce(d, y):
    e = d - y
    delta = e

    return delta
```

수학적 문제이지만, 델타를 구할 때 Cross Entropy 함수를 사용하면, 델타는 오차와 같습니다.

zero-base /

은닉층에서

```
In [11]: def calcDelta1_ce(W2, delta, y1):
             e1 = np.matmul(W2.T, delta)
             delta1 = y1*(1-y1) * e1

             return delta1
```

이건 그냥 동일하지만, 이름의 일관성을 위해...

zero-base /

다시 역전파

```
def BackpropCE(W1, W2, X, D, alpha):
    for k in range(4):
        x = X[k, :].T
        d = D[k]

        y, y1 = calc_output(W1, W2, x)
        delta = calcDelta_ce(d, y)
        delta1 = calcDelta1_ce(W2, delta, y1)

        dW1 = (alpha*delta1).reshape(4, 1) * x.reshape(1, 3)
        W1  = W1 + dW1

        dW2 = alpha * delta * y1
        W2  = W2 + dW2

    return W1, W2
```

zero-base /

학습

```
W1 = 2*np.random.random((4, 3)) - 1
W2 = 2*np.random.random((1, 4)) - 1

alpha = 0.9
for epoch in range(10000):
    W1, W2 = BackpropCE(W1, W2, X, D, alpha = 0.9)
```

zero-base /

결과

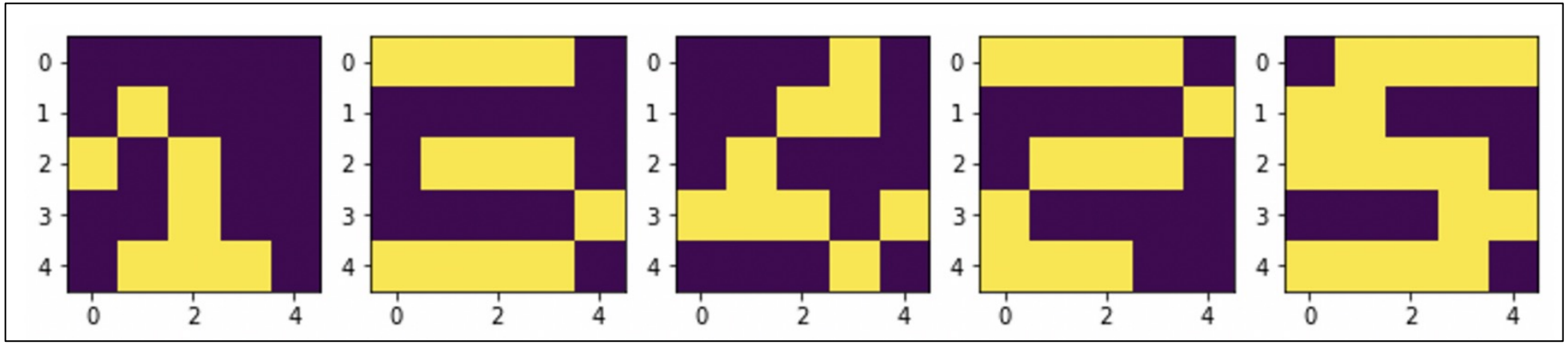
```
N = 4
for k in range(N):
    x  = X[k, :].T
    v1 = np.matmul(W1, x)
    y1 = sigmoid(v1)
    v  = np.matmul(W2, y1)
    y  = sigmoid(v)
    print(y)
```

```
[0.00032238]
[0.99975358]
[0.99986849]
[0.00016117]
```

예제

zero-base /

숫자 맞추기 (그노므 숫자)



zero-base /

import

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook
%matplotlib inline
```

Softmax

In [2]:

```
def Softmax(x):
    x  = np.subtract(x, np.max(x))       # prevent overflow
    ex = np.exp(x)

    return ex / np.sum(ex)
```

zero-base /

훈련용 데이터

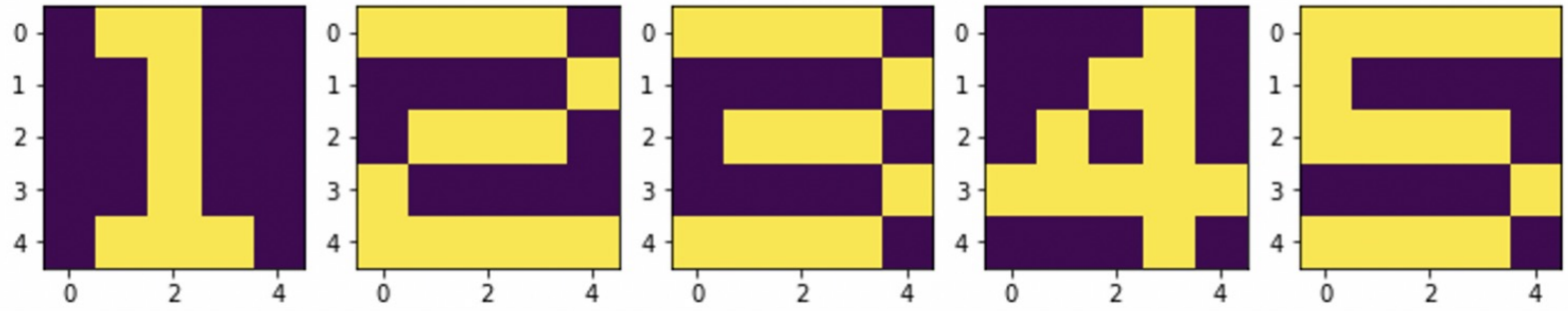
In [3]:

```
X = np.zeros((5, 5, 5))

X[:, :, 0] = [[0,1,1,0,0],  [0,0,1,0,0],  [0,0,1,0,0], [0,0,1,0,0], [0,1,1,1,0]]
X[:, :, 1] = [[1,1,1,1,0], [0,0,0,0,1], [0,1,1,1,0], [1,0,0,0,0], [1,1,1,1,1]]
X[:, :, 2] = [[1,1,1,1,0], [0,0,0,0,1], [0,1,1,1,0], [0,0,0,0,1], [1,1,1,1,0]]
X[:, :, 3] = [[0,0,0,1,0], [0,0,1,1,0], [0,1,0,1,0], [1,1,1,1,1], [0,0,0,1,0]]
X[:, :, 4] = [[1,1,1,1,1], [1,0,0,0,0], [1,1,1,1,0], [0,0,0,0,1], [1,1,1,1,0]]

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

zero-base /

이렇게 생김

```
In [4]: plt.figure(figsize=(12,4))
        for n in range(5):
            plt.subplot(1,5,n+1)
            plt.imshow(X[:, :, n])
        plt.show()
```



zero-base /

ReLU

In [5]:
```
def ReLU(x):
    return np.maximum(0, x)
```

zero-base /

ReLU를 이용한 정방향 계산

In [6]:

```
def calcOutput_ReLU(W1, W2, W3, W4, x):
    v1 = np.matmul(W1, x)
    y1 = ReLU(v1)
    v2 = np.matmul(W2, y1)
    y2 = ReLU(v2)
    v3 = np.matmul(W3, y2)
    y3 = ReLU(v3)
    v = np.matmul(W4, y3)
    y = Softmax(v)

    return y, v1, v2, v3, y1,y2, y3
```

zero-base /

역전파

In [7]:

```
def backpropagation_ReLU(d, y, W2, W3, W4, v1,v2,v3):
    e = d - y
    delta = e

    e3 = np.matmul(W4.T, delta)
    delta3 = (v3 > 0) * e3

    e2 = np.matmul(W3.T, delta3)
    delta2 = (v2 > 0) * e2

    e1 = np.matmul(W2.T, delta2)
    delta1 = (v1 > 0) * e1

    return delta, delta1, delta2, delta3
```

zero-base /

가중치 계산

In [8]:

```
def calcWs(alpha, delta, delta1, delta2, delta3, y1, y2, y3, x, W1, W2, W3, W4):
    dW4 = alpha * delta * y3.T
    W4  = W4 + dW4

    dW3 = alpha * delta3 * y2.T
    W3  = W3 + dW3

    dW2 = alpha * delta2 * y1.T
    W2  = W2 + dW2

    dW1 = alpha * delta1 * x.T
    W1  = W1 + dW1

    return W1, W2, W3, W4
```

zero-base /

가중치 업데이트

In [9]:

```
def DeepReLU(W1, W2, W3, W4, X, D, alpha):
    for k in range(5):
        x  = np.reshape(X[:, :, k], (25, 1))
        d    = D[k, :].T

        y, v1, v2, v3, y1,y2, y3 = calcOutput_ReLU(W1, W2, W3, W4, x)
        delta, delta1, delta2, delta3 = backpropagation_ReLU(d, y, W2, W3, W4, v1,v2,v3)
        W1, W2, W3, W4 = calcWs(alpha, delta, delta1, delta2, delta3,
                                            y1, y2, y3, x, W1, W2, W3, W4)

    return W1, W2, W3, W4
```

zero-base /

학습 시작~

```
In [10]: W1 = 2*np.random.random((20, 25)) - 1
         W2 = 2*np.random.random((20, 20)) - 1
         W3 = 2*np.random.random((20, 20)) - 1
         W4 = 2*np.random.random(( 5, 20)) - 1

         alpha = 0.01
         for epoch in tqdm_notebook(range(10000)):
             W1, W2, W3, W4 = DeepReLU(W1, W2, W3, W4, X, D, alpha)
```

한 10000번 epoch 시킴...

zero-base /

훈련 데이터 검증

In [11]:

```
def verify_algorithm(x, W1, W2, W3, W4):
    v1 = np.matmul(W1, x)
    y1 = ReLU(v1)

    v2 = np.matmul(W2, y1)
    y2 = ReLU(v2)

    v3 = np.matmul(W3, y2)
    y3 = ReLU(v3)

    v = np.matmul(W4, y3)
    y = Softmax(v)

    return y
```

훈련 데이터 검증

zero-base /

결과

In [12]:

```
N = 5
for k in range(N):
    x  = np.reshape(X[:, :, k], (25, 1))

    y = verify_algorithm(x, W1, W2, W3, W4)

    print("Y = {}: ".format(k+1))
    print(np.argmax(y, axis=0)+1)
    print(y)
    print('------------------------------')
```

zero-base /

잘 한 듯?

```
Y = 1:
[1]
[[  9.99971563e-01]
 [  2.26083018e-12]
 [  1.56967170e-06]
 [  1.58062305e-05]
 [  1.10613584e-05]]
------------------------------
Y = 2:
[2]
[[  4.60220642e-08]
 [  9.99976570e-01]
 [  1.48646391e-05]
 [  5.43885329e-06]
 [  3.08003683e-06]]
------------------------------
Y = 3:
[3]
[[  1.61187071e-06]
 [  1.10845963e-05]
 [  9.99958233e-01]
 [  8.19251588e-06]
 [  2.08783359e-05]]
```

```
Y = 4:
[4]
[[  2.02699258e-06]
 [  1.06033403e-05]
 [  2.09081271e-06]
 [  9.99980320e-01]
 [  4.95932376e-06]]
--------------------
Y = 5:
[5]
[[  3.83503967e-06]
 [  6.36761412e-06]
 [  1.20387935e-05]
 [  1.17865410e-09]
 [  9.99977757e-01]]
--------------------
```

zero-base /

테스트 데이터 만들기

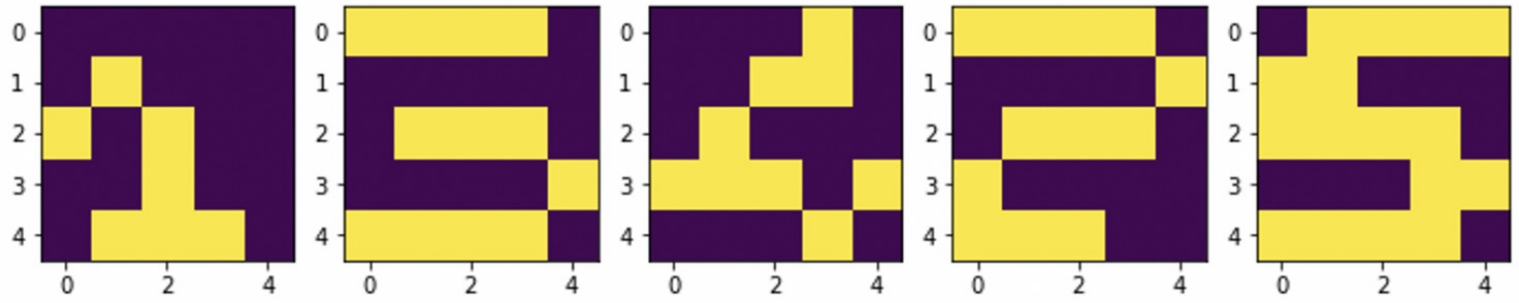
In [13]:

```
X_test = np.zeros((5, 5, 5))

X_test[:, :, 0] = [[0,0,0,0,0], [0,1,0,0,0], [1,0,1,0,0], [0,0,1,0,0], [0,1,1,1,0]]
X_test[:, :, 1] = [[1,1,1,1,0], [0,0,0,0,0], [0,1,1,1,0], [0,0,0,0,1], [1,1,1,1,0]]
X_test[:, :, 2] = [[0,0,0,1,0], [0,0,1,1,0], [0,1,0,0,0], [1,1,1,0,1], [0,0,0,1,0]]
X_test[:, :, 3] = [[1,1,1,1,0], [0,0,0,0,1], [0,1,1,1,0], [1,0,0,0,0], [1,1,1,0,0]]
X_test[:, :, 4] = [[0,1,1,1,1], [1,1,0,0,0], [1,1,1,1,0], [0,0,0,1,1], [1,1,1,1,0]]
```

zero-base /

이렇게~

```
In [14]: plt.figure(figsize=(12,4))
         for n in range(5):
             plt.subplot(1,5,n+1)
             plt.imshow(X_test[:, :, n])
         plt.show()
```



zero-base /

테스트

In [15]:

```
learning_result = [0,0,0,0,0]
for k in range(5):
    x  = np.reshape(X_test[:, :, k], (25, 1))

    y = verify_algorithm(x, W1, W2, W3, W4)

    learning_result[k] = np.argmax(y, axis=0)+1

    print("Y = {}: ".format(k+1))
    print(np.argmax(y, axis=0)+1)
    print(y)
    print('------------------------------')
```

zero-base /

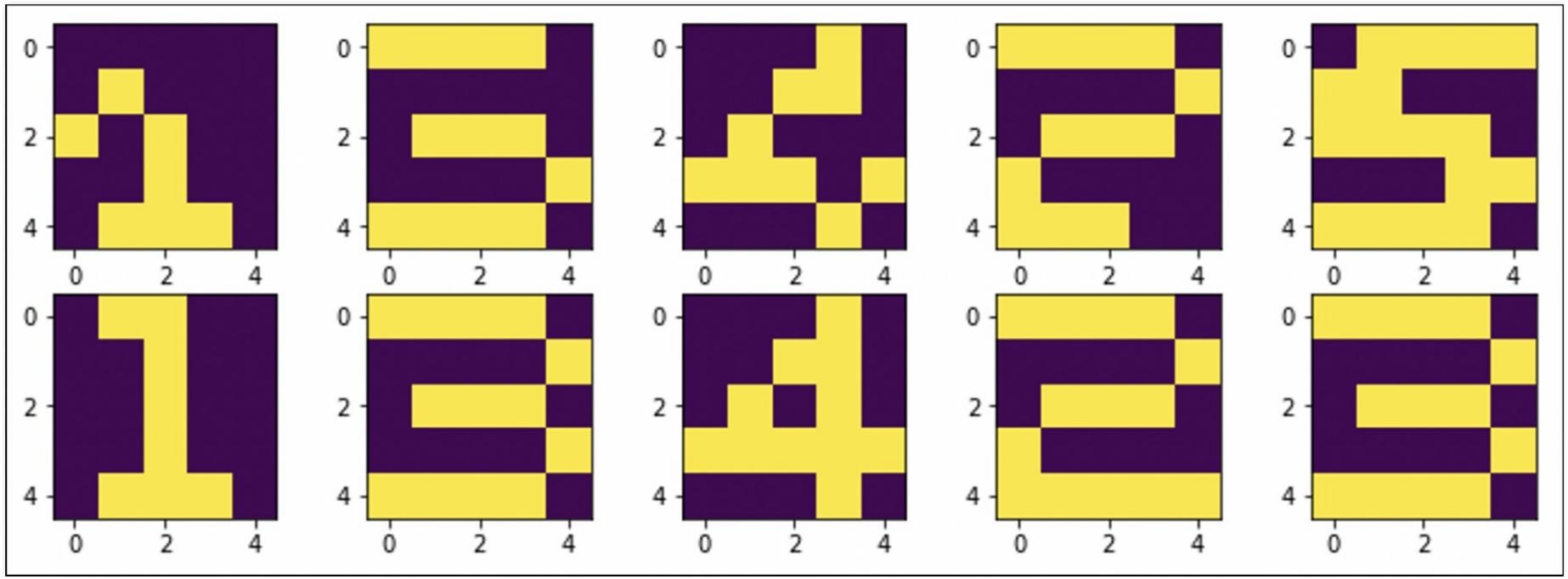
그려 보자

In [16]:

```
plt.figure(figsize=(12,4))

for k in range(N):
    plt.subplot(2,5,k+1)
    plt.imshow(X_test[:, :, k])
    plt.subplot(2,5,k+6)
    plt.imshow(X[:, :, learning_result[k][0]-1])

plt.show()
```

zero-base /

틀린 것도 있네



zero-base /

dropout 함수 생성

In [18]:

```
def Dropout(y, ratio):
    ym = np.zeros_like(y)

    num = round(y.size*(1-ratio))
    idx = np.random.choice(y.size, num, replace=False)
    ym[idx] = 1.0 / (1.0 - ratio)

    return ym
```

zero-base /

dropout 적용

In [19]:

```
def calcOutput_Dropout(W1, W2, W3, W4, x):
    v1 = np.matmul(W1, x)
    y1 = Sigmoid(v1)
    y1 = y1 * Dropout(y1, 0.2)

    v2 = np.matmul(W2, y1)
    y2 = Sigmoid(v2)
    y2 = y2 * Dropout(y2, 0.2)

    v3 = np.matmul(W3, y2)
    y3 = Sigmoid(v3)
    y3 = y3 * Dropout(y3, 0.2)

    v = np.matmul(W4, y3)
    y = Softmax(v)

    return y, y1, y2, y3, v1, v2, v3
```

정방향 출력을 계산하고...

zero-base /

역전파

In [20]:

```
def backpropagation_Dropout(d, y, y1, y2, y3, W2, W3, W4, v1,v2,v3):
    e = d - y
    delta = e

    e3 = np.matmul(W4.T, delta)
    delta3 = y3*(1-y3) * e3

    e2 = np.matmul(W3.T, delta3)
    delta2 = y2*(1-y2) * e2

    e1 = np.matmul(W2.T, delta2)
    delta1 = y1*(1-y1) * e1

    return delta, delta1, delta2, delta3
```

역전파로 델타를 계산하고

zero-base /

dropout 적용해서 다시 계산

```
def DeepDropout(W1, W2, W3, W4, X, D):
    for k in range(5):
        x = np.reshape(X[:, :, k], (25, 1))
        d = D[k,:].T

        y, y1, y2, y3, v1, v2, v3 = calcOutput_Dropout(W1, W2, W3, W4, x)
        delta, delta1, delta2, delta3 = backpropagation_Dropout(d, y, y1, y2, y3,
                                                    W2, W3, W4, v1,v2,v3)
        W1, W2, W3, W4 = calcWs(alpha, delta, delta1, delta2, delta3,
                                            y1, y2, y3, x, W1, W2, W3, W4)

    return W1, W2, W3, W4
```

다시 학습

In [22]:
```
W1 = 2*np.random.random((20, 25)) - 1
W2 = 2*np.random.random((20, 20)) - 1
W3 = 2*np.random.random((20, 20)) - 1
W4 = 2*np.random.random(( 5, 20)) - 1

for epoch in tqdm_notebook(range(10000)):
    W1, W2, W3, W4 = DeepDropout(W1, W2, W3, W4, X, D)
```

zero-base /

아까 만들었지만

In [23]:

```
def verify_algorithm(x, W1, W2, W3, W4):
    v1 = np.matmul(W1, x)
    y1 = Sigmoid(v1)

    v2 = np.matmul(W2, y1)
    y2 = Sigmoid(v2)

    v3 = np.matmul(W3, y2)
    y3 = Sigmoid(v3)

    v = np.matmul(W4, y3)
    y = Softmax(v)

    return y
```

zero-base /

결과 출력

In [25]:

```
N = 5
for k in range(N):
    x  = np.reshape(X_test[:, :, k], (25, 1))

    y = verify_algorithm(x, W1, W2, W3, W4)

    learning_result[k] = np.argmax(y, axis=0)+1

    print("Y = {}: ".format(k+1))
    print(np.argmax(y, axis=0)+1)
    print(y)
    print('------------------------------')
```

zero-base /

결과 확인

In [26]:

```
plt.figure(figsize=(12,4))

for k in range(N):
    plt.subplot(2,5,k+1)
    plt.imshow(X_test[:, :, k])
    plt.subplot(2,5,k+6)
    plt.imshow(X[:, :, learning_result[k][0]-1])

plt.show()
```

zero-base /

결과는? 랜덤성이 있어서 다를 수도 있어요~