

Chapter 04

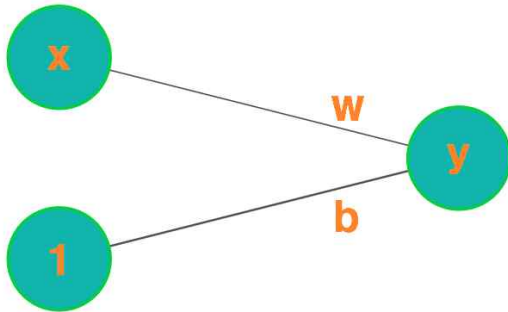
인공지능의 딥러닝 알고리즘

01 딥러닝 동작 원리 이해하기

여기서는 인공 신경의 동작을 상식적인 수준에서 살펴보면서 딥러닝의 동작 원리를 이해해 봅니다. 또 딥러닝과 관련된 중요한 용어들, 예를 들어, 순전파, 목표값, 역전파 오차, 오차 역전파와 같은 용어들을 이해해 보도록 합니다.

01 인공 신경 동작 살펴보기

다음은 앞에서 소개한 단일 인공 신경의 그림입니다. 이 인공 신경은 입력 노드 1개, 출력 노드 1개, 편향으로 구성된 단일 인공 신경입니다.



수식으로는 다음과 같이 표현합니다.

$$y = xw + 1b$$

이 수식에 대해서 구체적으로 생각해 봅니다. 다음과 같이 각 변수에 값을 줍니다.

$$\begin{aligned} x &= 2 \\ w &= 3 \\ b &= 1 \end{aligned}$$

그러면 식은 다음과 같이 됩니다.

$$y = 2 \times 3 + 1 \times 1$$

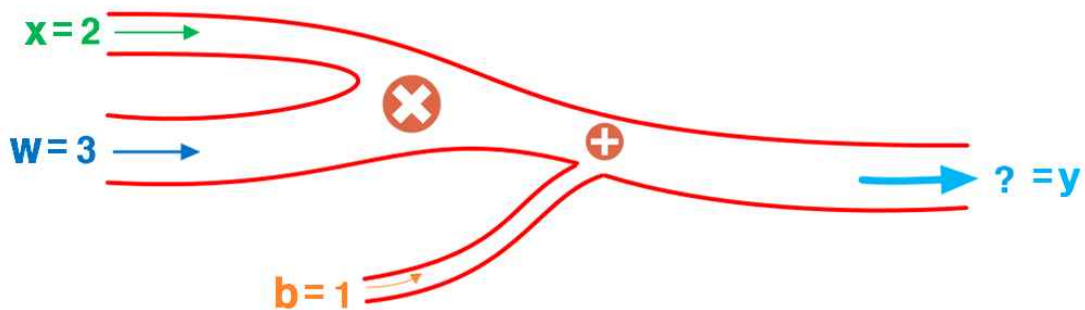
$$y = ?$$

y는 얼마가 될까요? 다음과 같이 계산해서 y는 7이 됩니다.

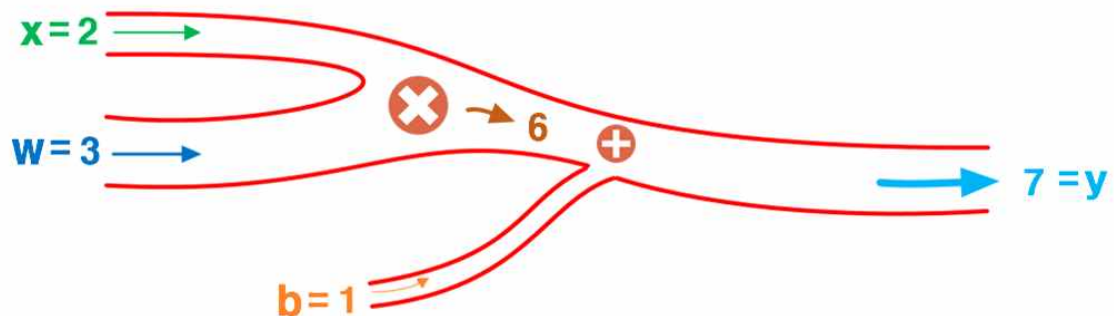
$$2 \times 3 + 1 \times 1 = 7$$

순전파

이 상황을 그림으로 생각해 봅시다. 다음과 같이 x, w, b 값이 y로 흘러가는 인공 신경 파이프가 있습니다. 이 과정을 인공 신경의 순전파라고 합니다.

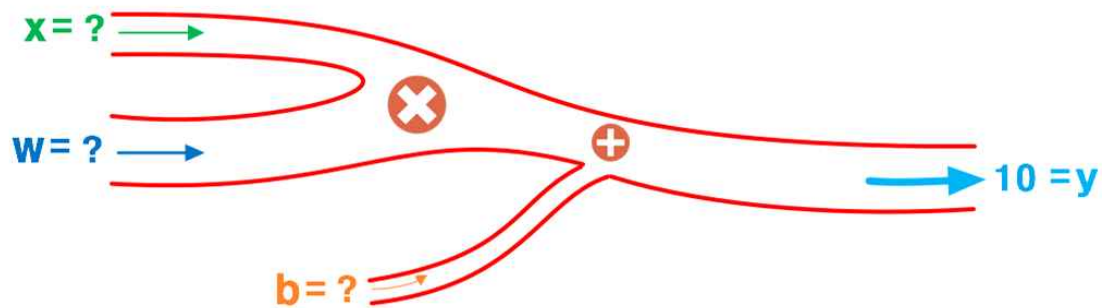


이 경우 y로 얼마가 나올까요? 앞에서 살펴본대로 다음과 같은 과정을 거쳐 7이 흘러나오게 됩니다.



목표값과 역전파 오차

그런데 y 로 10이 나오게 하려면 들어오는 값들을 어떻게 바꿔야 할까요?



y 값이 10이 되려면 3이 모자랍니다. x , w , b 값을 적당히 증가시키면 y 로 10에 가까운 값이 나오게 할 수 있겠죠? 그러면 x , w , b 값을 어떤 기준으로 얼마나 증가시켜야 할까요? 이 과정을 자세히 살펴봅시다.

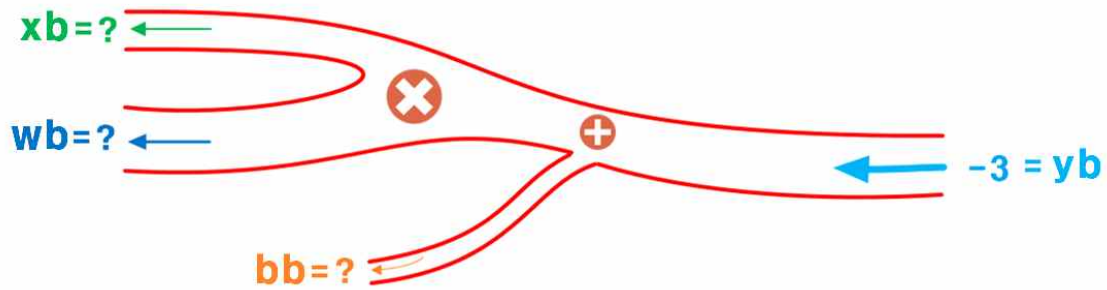
앞에서 y 로 7값이 흘러나갔는데 우리는 이 값이 10이 되기를 원합니다. 여기서 10값은 목표값이 됩니다. 다음 수식에서 t 는 목표값 10을 갖습니다.

$$\begin{aligned} t &= 10 \\ y &= 7 \\ y_b &= y - t \\ y_b &= -3 \end{aligned}$$

y 값은 현재값 7인 상태이며, y_b 는 현재값에서 목표값을 뺀 값 -3 이 됩니다. 이 때, y_b 값을 역전파 오차라고 하며, 역전파에 사용할 오차값입니다.

오차 역전파

이 상황을 그림으로 생각해 봅시다. 이번엔 y_b 의 값이 x_b , w_b , b_b 로 거꾸로 흘러가는 상황이 됩니다.



x , w , b 를 어떤 기준으로 얼마나 값을 할당해야 할까요? 다음과 같은 방법은 어떨까요?

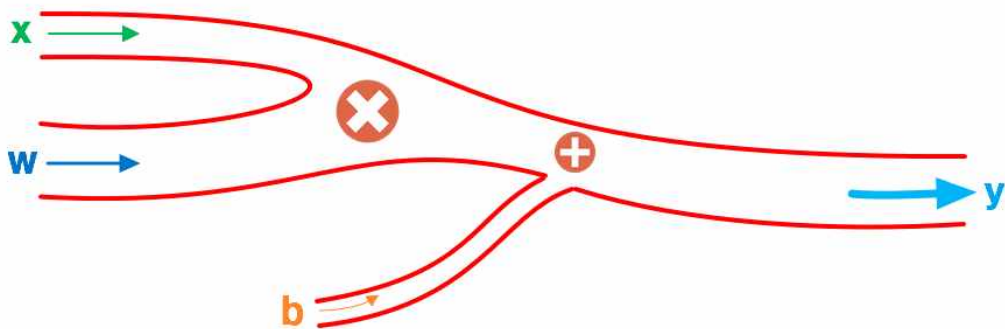
- 2는 3만큼 7로 갔어! 그러니까 -3도 3만큼 2로 돌아가게 하자!
- 3은 2만큼 7로 갔어! 그러니까 -3을 2만큼 3으로 돌아가게 하자!
- 1은 1만큼 7로 갔어! 그러니까 -3을 1만큼 1로 돌아가게 하자!

어떤가요? 설득력 있는 방법인가요? 이 방법이 바로 인공 신경에서 사용하는 오차 역전파입니다.

지금까지의 과정을 그림과 수식을 통해서 다시한 번 정리해봅시다.

순전파 정리하기

다음은 x , w , b 가 y 로 흘러가는 순전파를 나타냅니다.



다음은 이 그림에 대한 수식입니다.

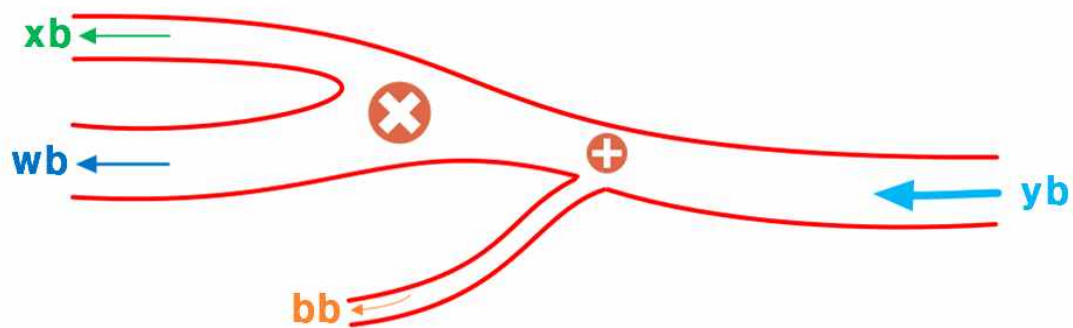
$$xw + 1b = y \quad ①$$

이 수식은 다음과 같은 의미를 갖습니다.

- x 는 w 만큼 y 로 갔어.
- w 는 x 만큼 y 로 갔어.
- b 는 1만큼 y 로 갔어.

역전파 정리하기

다음은 y 가 x , w , b 로 흘러가는 역전파를 나타냅니다.



우리는 다음 사항이 궁금합니다.

- y 는 얼마만큼 x 로 가야해?
- y 는 얼마만큼 w 로 가야해?
- y 는 얼마만큼 b 로 가야해?

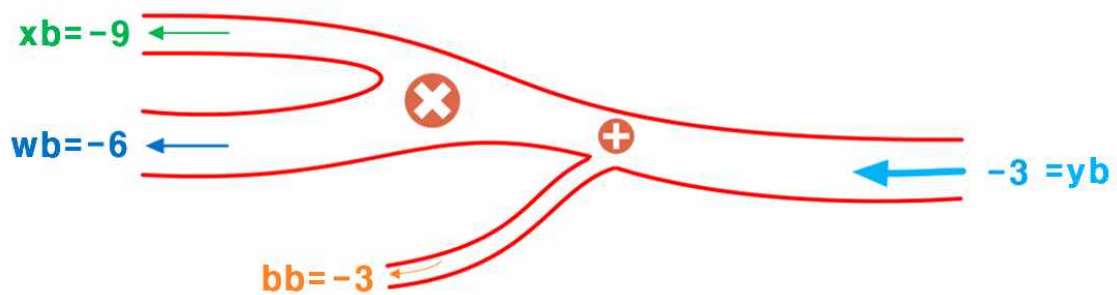
이에 대한 답은 다음과 같습니다.

- x 가 w 만큼 y 로 왔으니 y 도 w 만큼 x 로 가야하는거 아냐?
- w 가 x 만큼 y 로 왔으니 y 도 x 만큼 w 로 가야하는거 아냐?
- b 가 1만큼 y 로 왔으니 y 도 1만큼 b 로 가야하는거 아냐?

이를 수식으로 정리하면 다음과 같습니다.

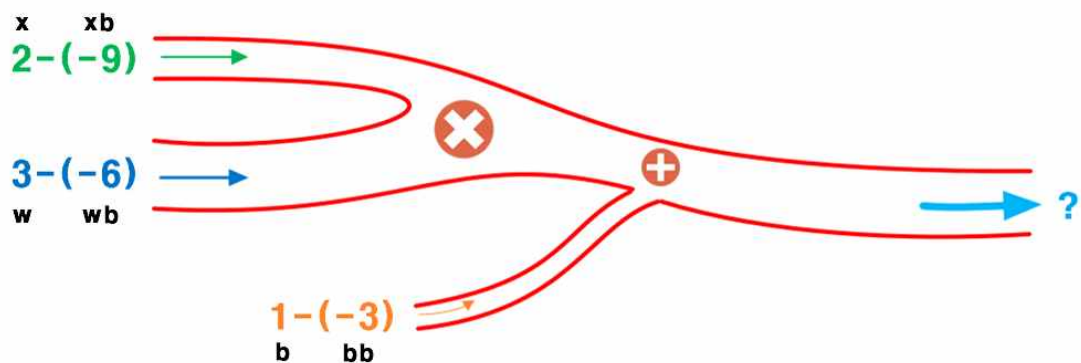
$x_b = y_b w$	2
$w_b = y_b x$	3
$b_b = y_b 1$	4

이 수식에 의해 x_b , w_b , y_b 는 다음 그림과 같이 계산됩니다.



최적화하기

이렇게 구한 값을 다시 다음과 같이 밀어 넣으면 될까요? 앞에서 구한 x_b , w_b , b_b 의 값이 음수가 되기 때문에 일단 빼주어야 합니다. 그래야 원래 값이 증가하기 때문입니다.

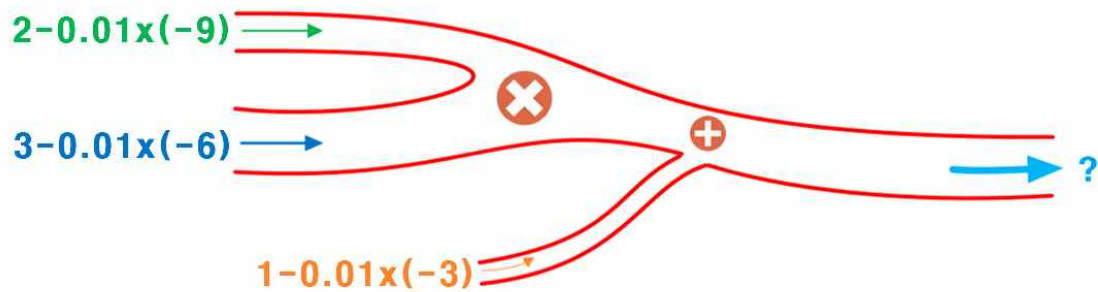


그런데 x_b , w_b , b_b 값이 너무 큼니다. 이 상태로 계산을 하면 새로운 y 값은 $(11 \times 9 + 4)$ 와 같이 계산되어 103이 되게 되며, 우리가 원하는 10보다 훨씬 큰 값이 나오게 됩니다.

학습률

그러면 이런 방법은 어떨까요? x_b , w_b , b_b 에 적당한 값을 곱해주어 값을 줄이는 겁니다. 여

기서는 0.01을 곱해줍니다. 그러면 다음과 같이 계산할 수 있습니다.



이렇게 하면 7.4254가 나옵니다. 오! 이렇게 조금씩 올려나 가면 10을 만들 수 있겠네요!

여기서 곱해준 0.01은 학습률이라고 하는 값입니다. 일반적으로 학습률 값은 0.01로 시작하여 학습이 진행되는 상황에 따라 조금씩 늘이거나 줄여서 사용합니다.

경사 하강법과 인공 신경망 학습

위 그림에 따라 새로운 x , w , b 값을 구하는 수식은 다음과 같습니다.

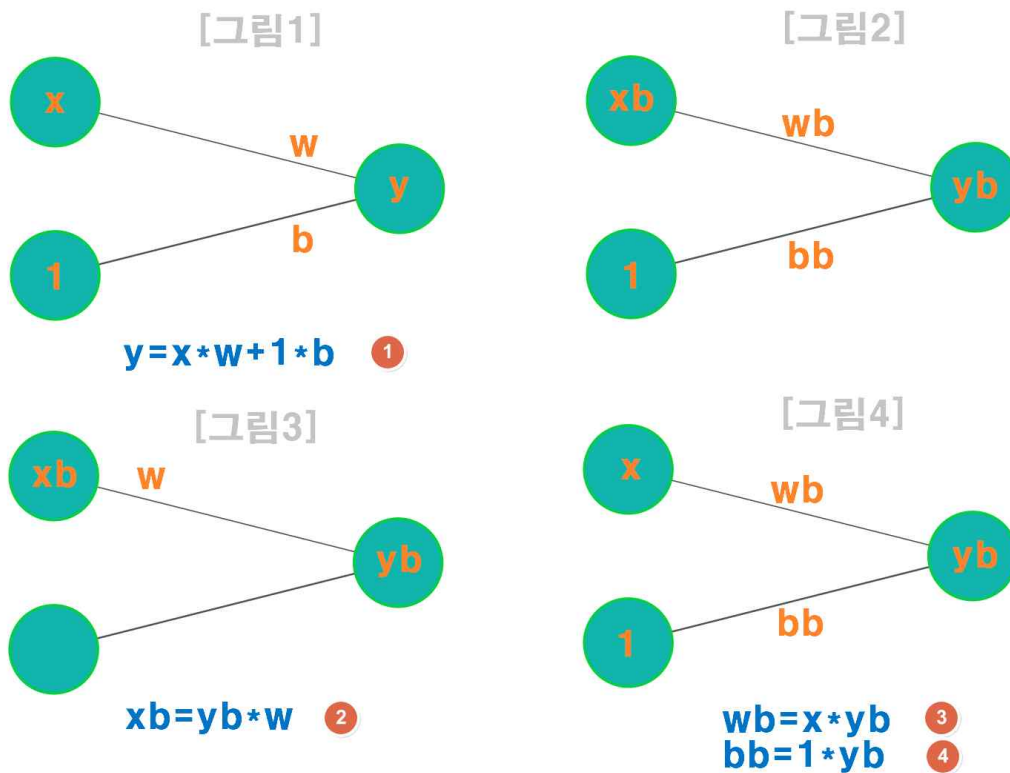
$x = x - \alpha x_b$	5
$w = w - \alpha w_b$	6
$b = b - \alpha b_b$	7

이 수식을 경사하강법이라고 합니다. 그리고 이 수식을 적용하여 x , w , b 값을 갱신하는 과정을 인공 신경망의 학습이라고 합니다. 여러분은 방금 전에 1회의 학습을 수행하는 과정을 보신겁니다. 이 과정을 컴퓨터를 이용하여 반복하여 수행하면 우리가 원하는 값을 얻게 해 주는 인공 신경을 만들 수 있습니다.

02 인공 신경 동작 구현해 보기

지금까지의 과정을 그림과 수식을 통해 정리한 후, 구현을 통해 확인해 봅니다.

다음 그림은 지금까지 살펴본 입력1 출력1로 구성된 인공 신경을 나타냅니다.



[그림1]은 순전파 과정에 필요한 변수와 수식을 나타냅니다.

[그림2]는 역전파에 필요한 변수입니다. 순전파에 대응되는 변수가 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 변수와 수식을 나타냅니다.

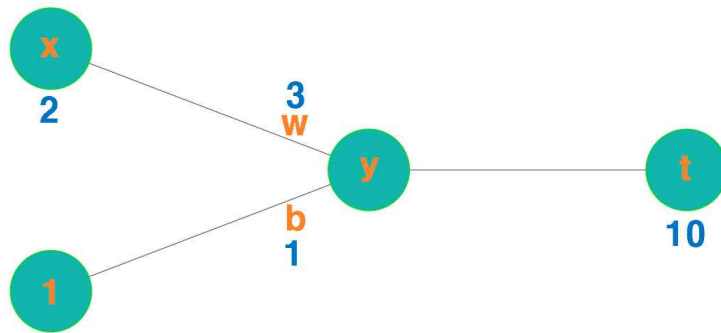
[그림4]는 가중치와 편향의 역전파에 필요한 변수와 수식을 나타냅니다.

*** ② x_b 노드가 입력층일 경우엔 이 수식은 필요하지 않습니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

순전파
$xw + 1b = y$ ①
입력 역전파
$y_b w = x_b$ ②
가중치, 편향 역전파
$x y_b = w_b$ ③
$1 y_b = b_b$ ④

지금까지의 과정을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 x , 가중치 w , 편향 b 는 각각 2, 3, 1이고 목표값 t 는 10입니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 다음과 같이 예제를 작성합니다.

```

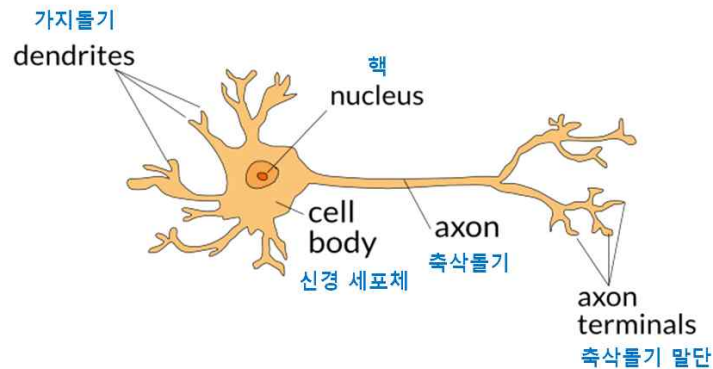
01 : x = 2
02 : t = 10
03 : w = 3
04 : b = 1
05 :
06 : y = x*w + 1*b ❶
07 : print('y = %6.3f' %y)
08 :
09 : yb = y - t
10 : xb = yb*w ❷
11 : wb = yb*x ❸
12 : bb = yb*1 ❹
13 : print('xb = %6.3f, wb = %6.3f, bb = %6.3f'%(xb, wb, bb))
14 :
15 : lr = 0.01
16 : x = x - lr*xb ❺
17 : w = w - lr*wb ❻
18 : b = b - lr*bb ❼
19 : print('x = %6.3f, w = %6.3f, b = %6.3f'%(x, w, b))
  
```

01 : 변수 x 를 선언한 후, 2로 초기화합니다.

02 : 변수 t 를 선언한 후, 10으로 초기화합니다.

03 : 가중치 변수 w 를 선언한 후, 3으로 초기화합니다. 가중치 w 는 입력값의 강도, 세기라고도 하며 입력값을 증폭 시키거나 감소시키는 역할을 합니다. 인공 신경도 가지돌기의 두께에 따라 입력 신호가 증폭되거나 감소될 수 있는데, 이런 관점에서 가중치는 가지돌기의 두께에

해당되는 변수로 생각할 수 있습니다.



04 : 편향 변수 b 를 선언한 후, 1로 초기화합니다. 편향은 가중치를 거친 입력값의 합(=전체 입력 신호)에 더해지는 값으로 입력신호를 좀 더 세게 해주거나 약하게 하는 역할을 합니다.

06 : 순전파 수식을 구현합니다.

07 : `print` 함수를 호출하여 순전파 결과값 y 를 출력합니다. 소수점 이하 3자리까지 출력합니다.

09 : y_b 변수를 선언한 후, 순전파 결과값에서 목표값을 빼 오차값을 넣어줍니다.

10 : x_b 변수를 선언한 후, 입력값에 대한 역전파 값을 받습니다.

11 : w_b 변수를 선언한 후, 가중치 값에 대한 역전파 값을 받습니다.

12 : b_b 변수를 선언한 후, 편향 값에 대한 역전파 값을 받습니다.


13 : `print` 함수를 호출하여 역전파 결과값 x_b , w_b , b_b 를 출력합니다. 소수점 이하 3자리까지 출력합니다.

15 : 학습률 변수 lr 을 선언한 후, 0.01로 초기화합니다.

16 : x_b 역전파값에 학습률을 곱한 후, x 값에서 빼줍니다. 이 과정에서 x 변수에 대한 학습이 이루어집니다.

17 : w_b 역전파값에 학습률을 곱한 후, w 값에서 빼줍니다. 이 과정에서 w 변수에 대한 학습이 이루어집니다.

18 : b_b 역전파값에 학습률을 곱한 후, b 값에서 빼줍니다. 이 과정에서 b 변수에 대한 학습이 이루어집니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
y = 7.000
xb = -9.000, wb = -6.000, bb = -3.000
x = 2.090, w = 3.060, b = 1.030
```


반복 학습 2회 수행하기

여기서는 반복 학습 2회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

```
01 : x = 2
02 : t = 10
03 : w = 3
04 : b = 1
05 :
06 : for epoch in range(2):
07 :
08 :     print('epoch = %d' %epoch)
09 :
10 :     y = x*w + 1*b
11 :     print(' y = %6.3f' %y)
12 :
13 :     yb = y - t
14 :     xb = yb*w
15 :     wb = yb*x
16 :     bb = yb*1
17 :     print(' xb = %6.3f, wb = %6.3f, bb = %6.3f'%(xb, wb, bb))
18 :
19 :     lr = 0.01
20 :     x = x - lr*xb
21 :     w = w - lr*wb
22 :     b = b - lr*bb
23 :     print(' x = %6.3f, w = %6.3f, b = %6.3f'%(x, w, b))
```

06 : epoch값을 0에서 2 미만까지 바꾸어가며 8~23줄을 2회 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 0
y = 7.000
xb = -9.000, wb = -6.000, bb = -3.000
x = 2.090, w = 3.060, b = 1.030
epoch = 1
y = 7.425
xb = -7.878, wb = -5.381, bb = -2.575
x = 2.169, w = 3.114, b = 1.056
```

y 값이 7에서 7.425로 바뀌는 것을 확인합니다.

반복 학습 20회 수행하기

여기서는 반복 학습 20회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

```
06 : for epoch in range(20):
```

06 : epoch값을 0에서 20 미만까지 수행합니다.

```
epoch = 17
y = 9.887
xb = -0.384, wb = -0.290, bb = -0.113
x = 2.566, w = 3.402, b = 1.179
epoch = 18
y = 9.909
xb = -0.311, wb = -0.235, bb = -0.091
x = 2.570, w = 3.404, b = 1.179
epoch = 19
y = 9.926
xb = -0.252, wb = -0.190, bb = -0.074
x = 2.572, w = 3.406, b = 1.180
```

y 값이 9.926까지 접근하는 것을 확인합니다.


반복 학습 200회 수행하기

여기서는 반복 학습 200회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

```
06 : for epoch in range(200):
```

06 : epoch값을 0에서 200 미만까지 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

epoch = 197
y = 10.000
xb = -0.000, wb = -0.000, bb = -0.000
x = 2.583, w = 3.414, b = 1.183
epoch = 198
y = 10.000
xb = -0.000, wb = -0.000, bb = -0.000
x = 2.583, w = 3.414, b = 1.183
epoch = 199
y = 10.000
xb = -0.000, wb = -0.000, bb = -0.000
x = 2.583, w = 3.414, b = 1.183

```

y 값이 10.000에 수렴하는 것을 확인합니다.

오차값 계산하기

여기서는 인공 신경망을 통해 얻어진 예측값과 목표값의 오차를 계산하는 부분을 추가해 봅니다. 오차(error)는 손실(loss) 또는 비용(cost)이라고도 합니다. 오차값이 작을수록 예측을 잘 하는 인공 신경망입니다.

1. 다음과 같이 예제를 수정합니다.

```

01 : x = 2
02 : t = 10
03 : w = 3
04 : b = 1
05 :
06 : for epoch in range(200):
07 :
08 :     print('epoch = %d' %epoch)
09 :
10 :     y = x*w + 1*b
11 :     print(' y = %6.3f' %y)
12 :
13 :     E = (y-t)**2/2
14 :     print(' E = %.7f' %E)
15 :     if E < 0.0000001:
16 :         break

```

```

17 :
18 :     yb = y - t
19 :     xb = yb*w
20 :     wb = yb*x
21 :     bb = yb*1
22 :     print(' xb = %6.3f, wb = %6.3f, bb = %6.3f'%(xb, wb, bb))
23 :
24 :     lr = 0.01
25 :     x = x - lr*xb
26 :     w = w - lr*wb
27 :     b = b - lr*bb
28 :     print(' x  = %6.3f, w  = %6.3f, b  = %6.3f'%(x, w, b))

```


13 : 변수 E를 선언한 후, 다음과 같은 형태의 수식을 구현합니다.

$$E = \frac{1}{2}(y - t)^2$$

y의 값이 t에 가까울수록 E의 값은 0에 가까워집니다. 즉, 오차값이 0에 가까워집니다. 이 수식을 오차함수 또는 손실함수 또는 비용함수라고 합니다.

14 : print 함수를 호출하여 오차값 E를 출력합니다. 소수점 이하 7자리까지 출력합니다.

15, 16 : 오차값 E가 0.0000001(1천만분의1)보다 작으면 break문을 수행하여 6줄의 for문을 빠져 나갑니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 41
y = 9.999
E = 0.0000002
xb = -0.002, wb = -0.002, bb = -0.001
x = 2.583, w = 3.414, b = 1.183
epoch = 42
y = 9.999
E = 0.0000001
xb = -0.002, wb = -0.001, bb = -0.001
x = 2.583, w = 3.414, b = 1.183
epoch = 43
y = 10.000
E = 0.0000001
```

epoch 값이 43(44회 째)일 때 for 문을 빠져 나갑니다. y값은 10에 수렴합니다.


학습률 변경하기

여기서는 학습률 값을 변경시켜 보면서 학습의 상태를 살펴봅니다.

1. 다음과 같이 예제를 수정합니다.

24 : lr = 0.05

24 : 학습률 값을 0.05로 변경합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.


```

epoch = 2
y = 9.931
E = 0.0023656
xb = -0.233, wb = -0.177, bb = -0.069
x = 2.588, w = 3.403, b = 1.192
epoch = 3
y = 9.997
E = 0.0000039
xb = -0.010, wb = -0.007, bb = -0.003
x = 2.588, w = 3.403, b = 1.192
epoch = 4
y = 10.000
E = 0.0000000


```

4회 째 학습이 완료되는 것을 볼 수 있습니다.

3. 다음과 같이 예제를 수정합니다.

```
24 : lr = 0.005
```

24 : 학습률 값을 0.005로 변경합니다.

4.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

epoch = 89
y = 9.999
E = 0.0000001
xb = -0.002, wb = -0.001, bb = -0.001
x = 2.582, w = 3.415, b = 1.183
epoch = 90
y = 10.000
E = 0.0000001
xb = -0.002, wb = -0.001, bb = -0.000
x = 2.582, w = 3.415, b = 1.183
epoch = 91
y = 10.000
E = 0.0000001

```

91회 째 학습이 완료되는 것을 볼 수 있습니다.


상수 입력값

여기서는 인공 신경의 입력값 x 를 상수로 준 상태로 인공 신경망을 학습시켜 봅니다. 입력값 x 가 이전 단계의 인공 신경의 출력에 연결된 경우(은닉층인 경우)에는 이전 단계의 인공 신경의 상태에 따라 값이 변할 수 있습니다. 그러나 현재 인공 신경이 최초 입력값을 받는 경우(입력층인 경우)엔 상수가 됩니다.

1. 다음과 같이 예제를 수정합니다.

```
24 : lr = 0.01
25 : # x = x - lr*xb
```

25 : x 값을 상수로 가정했기 때문에 학습시키지 않습니다. 이 경우는 x 가 입력층일 경우입니다.

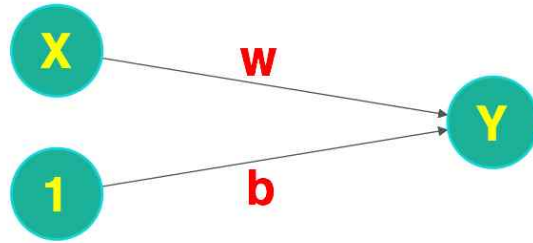
2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 170
y = 10.000
E = 0.0000001
xb = -0.002, wb = -0.001, bb = -0.000
x = 2.000, w = 4.200, b = 1.600
epoch = 171
y = 10.000
E = 0.0000001
xb = -0.002, wb = -0.001, bb = -0.000
x = 2.000, w = 4.200, b = 1.600
epoch = 172
y = 10.000
E = 0.0000001
```

172회 째 학습이 완료되는 것을 볼 수 있습니다. 가중치 w 는 4.2, 편향 b 는 1.6에 수렴합니다.

03 $y=3*x+1$ 학습시켜 보기

다음은 지금까지 살펴본 단일 인공 신경을 나타냅니다. 이 인공 신경은 입력 노드 1개, 출력 노드 1개, 편향으로 구성된 단일 인공 신경입니다.



우리는 다음과 같은 숫자들의 집합 X, Y를 이용하여, 이 인공 신경을 학습시켜 봅니다.

X:	-1	0	1	2	3	4
Y:	-2	1	4	7	10	13

그래서 다음 함수를 근사하는 인공 신경 함수를 만들어 보도록 합니다.

$$y = f(x) = 3 \cdot x + 1 \quad (x \text{는 실수})$$

인공 신경을 학습시키는 과정은 w, b 값을 X, Y 값에 맞추어 가는 과정입니다. 그래서 학습이 진행됨에 따라 w 값은 3에 가까운 값으로, b 값은 1에 가까운 값으로 이동하게 됩니다.

1. 다음과 같이 예제를 작성합니다.

221_1.py

```

01 : xs = [-1., 0., 1., 2., 3., 4.]
02 : ys = [-2., 1., 4., 7., 10., 13.]
03 :
04 : w = 10.
05 : b = 10.
06 :
07 : y = xs[0]*w + 1*b
08 : print("x = %6.3f, y = %6.3f" %(xs[0], y))
09 :
10 : t = ys[0]
11 : E = (y-t)**2/2
12 : print('E = %.7f' %E)
13 :
14 : yb = y - t
15 :
16 : wb = yb*xs[0]
17 : bb = yb*1
18 : print('wb = %6.3f, bb = %6.3f'%(wb, bb))
  
```

```

19 :
20 : lr = 0.01
21 :
22 : w = w - lr*wb
23 : b = b - lr*bb
24 : print('w = %6.3f, b = %6.3f'%(w, b))

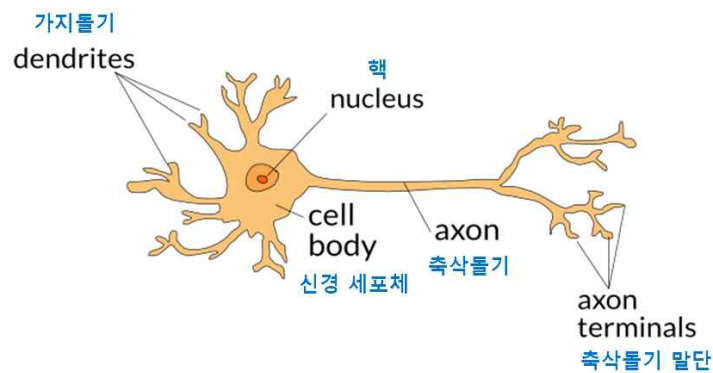
```

01, 02 : 실수형 리스트 변수 xs, ys를 선언한 후, 다음 X, Y 값으로 초기화합니다.

X:	-1	0	1	2	3	4
Y:	-2	1	4	7	10	13

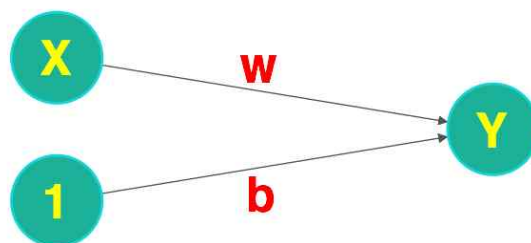
숫자 뒤에 점(.)은 실수를 나타냅니다.

04 : 입력값의 가중치값을 저장할 변수 w를 선언한 후, 10.으로 초기화합니다. 10.은 임의로 선택한 값입니다. 입력값의 가중치는 입력값의 강도, 세기라고도 하며 입력값을 증폭 시키거나 감소시키는 역할을 합니다. 인공 신경도 가지돌기의 두께에 따라 입력 신호가 증폭되거나 감소될 수 있는데, 이런 관점에서 가중치는 가지돌기의 두께에 해당되는 변수로 생각할 수 있습니다.



05 : 인공 신경의 편향값을 저장할 변수 b를 선언한 후, 10.으로 초기화합니다. 10.은 임의로 선택한 값입니다. 편향값은 가중치를 거친 입력값의 합(=전체 입력신호)에 더해지는 값으로 입력신호를 좀 더 세게 해주거나 약하게 하는 역할을 합니다.

07 : 다음과 같이 단일 인공 신경을 수식으로 표현합니다.



$$y = xw + 1b$$

$$= xw + b$$

일단 xs[0] 항목을 w에 곱한 후, b를 더해준 후, 변수 y에 대입해 줍니다. 이 과정에서 순전파가 이루어집니다. 즉, xs[0] 항목이 w에 곱해지고 b와 더해져 y에 도달하는 과정을 순전파라고 합니다. 순전파 결과 얻어진 y값을 인공 신경망에 의한 예측값이라고 합니다.

08 : print 함수를 호출하여 xs[0], y 값을 출력합니다.

10 : 변수 t를 선언한 후, ys[0]값을 받습니다. ys[0]은 인공 신경망에 대한 xs[0]값의 목표값입니다.

11 : 변수 E를 선언한 후, 다음과 같은 형태의 수식을 구현합니다.

$$E = \frac{1}{2}(y - t)^2$$

y의 값이 t에 가까울수록 E의 값은 0에 가까워집니다. 즉, 오차값이 0에 가까워집니다. 이 수식을 오차함수 또는 손실함수 또는 비용함수라고 합니다.

12 : print 함수를 호출하여 E 값을 출력합니다. 소수점 이하 7자리까지 출력합니다.

14 : yb 변수를 선언한 후, 순전파 결과값에서 목표값을 빼 오차값을 넣어줍니다.

16 : wb 변수를 선언한 후, 가중치 값에 대한 역전파 값을 받습니다.

17 : bb 변수를 선언한 후, 편향 값에 대한 역전파 값을 받습니다.


13 : print 함수를 호출하여 역전파 결과값 wb, bb를 출력합니다. 소수점 이하 3자리까지 출력합니다.

20 : 학습률 변수 lr을 선언한 후, 0.01로 초기화합니다.

22 : wb 역전파값에 학습률을 곱한 후, w값에서 빼줍니다. 이 과정에서 w 변수에 대한 학습이 이루어집니다.

23 : bb 역전파값에 학습률을 곱한 후, b값에서 빼줍니다. 이 과정에서 b 변수에 대한 학습이 이루어집니다.

24 : print 함수를 호출하여 학습이 1회 수행된 w, b 값을 출력합니다. 소수점 이하 3자리까지 출력합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
x = -1.000, y = 0.000
E = 2.0000000
wb = -2.000, bb = 2.000
w = 10.020, b = 9.980
```

w, b 값이 각각 10.1, 9.9으로 표시되는 것을 확인합니다.

전체 입력 데이터 학습 수행하기

이제 다음 좌표값 전체에 대해 1회 학습을 수행해 봅니다.

X:	-1	0	1	2	3	4
Y:	-2	1	4	7	10	13

1. 다음과 같이 예제를 수정합니다.

225_4.py


```
01 : xs = [-1., 0., 1., 2., 3., 4.]
02 : ys = [-2., 1., 4., 7., 10., 13.]
03 :
04 : w = 10.
05 : b = 10.
06 :
07 : for n in range(6):
08 :
09 :     y = xs[n]*w + 1*b
10 :     print("x  = %6.3f, y  = %6.3f" %(xs[n], y))
11 :
12 :     t = ys[n]
13 :     E = (y-t)**2/2
14 :     print('E  = %.7f' %E)
15 :
16 :     yb = y - t
17 :
18 :     wb = yb*xs[n]
19 :     bb = yb*1
20 :     print('wb = %6.3f, bb = %6.3f'%(wb, bb))
21 :
22 :     lr = 0.01
23 :
24 :     w = w - lr*wb
25 :     b = b - lr*bb
26 :     print('w  = %6.3f, b  = %6.3f'%(w, b))
27 :
28 :     print(" "*25)
```

7 : n값을 0에서 6 미만까지 바꾸어가며 9~28줄을 6회 수행합니다.

9, 10, 18 : xs[0]을 xs[n]으로 변경합니다.

12 : $ys[0]$ 을 $ys[n]$ 으로 변경합니다.

28 : 실행 경계를 표시하기 위해 "="을 25개 출력합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

x = -1.000, y = 0.000	x = 2.000, y = 29.453
E = 2.0000000	E = 252.0662245
wb = -2.000, bb = 2.000	wb = 44.906, bb = 22.453
w = 10.020, b = 9.980	w = 9.412, b = 9.507
=====	=====
x = 0.000, y = 9.980	x = 3.000, y = 37.742
E = 40.3202000	E = 384.8117627
wb = 0.000, bb = 8.980	wb = 83.226, bb = 27.742
w = 10.020, b = 9.890	w = 8.580, b = 9.229
=====	=====
x = 1.000, y = 19.910	x = 4.000, y = 43.547
E = 126.5672320	E = 466.5735925
wb = 15.910, bb = 15.910	wb = 122.190, bb = 30.547
w = 9.861, b = 9.731	w = 7.358, b = 8.924
=====	=====

가중치, 편향값 학습과정 살펴보기

가중치와 편향값만 확인해 봅니다.

1. 다음과 같이 예제를 수정합니다.

225_5.py


```
01 : xs = [-1., 0., 1., 2., 3., 4.]
02 : ys = [-2., 1., 4., 7., 10., 13.]
03 :
04 : w = 10.
05 : b = 10.
06 :
07 : for n in range(6):
08 :
09 :     y = xs[n]*w + 1*b
10 :
11 :     t = ys[n]
12 :     E = (y-t)**2/2
13 :
14 :     yb = y - t
15 :
16 :     wb = yb*xs[n]
17 :     bb = yb*1
18 :
```

```

19 : lr = 0.01
20 :
21 : w = w - lr*wb
22 : b = b - lr*bb
23 : print('w = %6.3f, b = %6.3f'%(w, b))

```

23 : w, b에 대한 출력만 합니다. 나머지 출력 루틴은 주석처리하거나 지워줍니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

W = 10.020, b = 9.980
W = 10.020, b = 9.890
W = 9.861, b = 9.731
W = 9.412, b = 9.507
W = 8.580, b = 9.229
W = 7.358, b = 8.924

```

학습 회수에 따라 w, b값이 바뀌는 것을 확인합니다.

반복 학습 2회 수행하기

여기서는 반복 학습 2회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

225_6.py

```

01 : xs = [-1., 0., 1., 2., 3., 4.]
02 : ys = [-2., 1., 4., 7., 10., 13.]
03 :
04 : w = 10.
05 : b = 10.
06 :
07 : for epoch in range(2):
08 :
09 :     for n in range(6):
10 :
11 :         y = xs[n]*w + 1*b
12 :
13 :         t = ys[n]
14 :         E = (y-t)**2/2
15 :

```




```

16 :         yb = y - t
17 :
18 :         wb = yb*xs[n]
19 :         bb = yb*1
20 :
21 :         lr = 0.01
22 :
23 :         w = w - lr*wb
24 :         b = b - lr*bb
25 :         print('w = %6.3f, b = %6.3f'%(w, b))

```

7 : epoch값을 0에서 2 미만까지 바꾸어가며 18~27줄을 2회 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

w = 10.020, b = 9.980
w = 10.020, b = 9.890
w = 9.861, b = 9.731
w = 9.412, b = 9.507
w = 8.580, b = 9.229
w = 7.358, b = 8.924
w = 7.393, b = 8.888
w = 7.393, b = 8.809
w = 7.271, b = 8.687
w = 6.947, b = 8.525
w = 6.366, b = 8.331
w = 5.534, b = 8.123

```

학습 회수에 따라 w, b값이 바뀌는 것을 확인합니다.

반복 학습 20회 수행하기

여기서는 반복 학습 20회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

225_7.py

```

01 : xs = [-1., 0., 1., 2., 3., 4.]
02 : ys = [-2., 1., 4., 7., 10., 13.]
03 :
04 : w = 10.
05 : b = 10.
06 :
07 : for epoch in range(20):

```


```

08 :
09 :     for n in range(6):
10 :
11 :         y = xs[n]*w + 1*b
12 :
13 :         t = ys[n]
14 :         E = (y-t)**2/2
15 :
16 :         yb = y - t
17 :
18 :         wb = yb*xs[n]
19 :         bb = yb*1
20 :
21 :         lr = 0.01
22 :
23 :         w = w - lr*wb
24 :         b = b - lr*bb
25 :         if epoch%2==1 and n==0 :
26 :             print('w = %6.3f, b = %6.3f'%(w, b))

```

07 : epoch값을 0에서 20 미만까지 바꾸어가며 9~26줄을 20회 수행합니다.

25 : epoch값을 2로 나눈 나머지가 1이고 n값이 0일 때 26줄을 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

w = 7.393, b = 8.888
w = 4.332, b = 7.464
w = 2.897, b = 6.614
w = 2.247, b = 6.051
w = 1.974, b = 5.636
w = 1.882, b = 5.303
w = 1.875, b = 5.016
w = 1.907, b = 4.760
w = 1.956, b = 4.526
w = 2.011, b = 4.309

```

학습 회수에 따라 w, b값이 바뀌는 것을 확인합니다.

반복 학습 200회 수행하기

여기서는 반복 학습 200회를 수행해 봅니다.

1. 다음과 같이 예제를 수정합니다.

225_8.py


```

01 : xs = [-1., 0., 1., 2., 3., 4.]
02 : ys = [-2., 1., 4., 7., 10., 13.]
03 :
04 : w = 10.
05 : b = 10.
06 :
07 : for epoch in range(200):
08 :
09 :     for n in range(6):
10 :
11 :         y = xs[n]*w + 1*b
12 :
13 :         t = ys[n]
14 :         E = (y-t)**2/2
15 :
16 :         yb = y - t
17 :
18 :         wb = yb*xs[n]
19 :         bb = yb*1
20 :
21 :         lr = 0.01
22 :
23 :         w = w - lr*wb
24 :         b = b - lr*bb
25 :         if epoch%20==1 and n==0 :
26 :             print('w = %6.3f, b = %6.3f'%(w, b))

```

07 : epoch값을 0에서 200 미만까지 바꾸어가며 9~26줄을 200회 수행합니다.

25 : epoch값을 20으로 나눈 나머지가 1이고 n값이 0일 때 26줄을 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

w = 7.393, b = 8.888
w = 2.067, b = 4.107
w = 2.499, b = 2.660
w = 2.732, b = 1.887
w = 2.857, b = 1.474
w = 2.923, b = 1.253
w = 2.959, b = 1.136
w = 2.978, b = 1.072
w = 2.988, b = 1.039
w = 2.994, b = 1.021

```

학습 회수에 따라 w, b값이 바뀌는 것을 확인합니다. w값은 3에, b값은 1에 가까워지는 것을

확인합니다.

반복 학습 2000회 수행하기

여기서는 반복 학습 2000회를 수행해 봅니다.


1. 다음과 같이 예제를 수정합니다.

225_8.py

```
01 : xs = [-1., 0., 1., 2., 3., 4.]
02 : ys = [-2., 1., 4., 7., 10., 13.]
03 :
04 : w = 10.
05 : b = 10.
06 :
07 : for epoch in range(2000):
08 :
09 :     for n in range(6):
10 :
11 :         y = xs[n]*w + 1*b
12 :
13 :         t = ys[n]
14 :         E = (y-t)**2/2
15 :
16 :         yb = y - t
17 :
18 :         wb = yb*xs[n]
19 :         bb = yb*1
20 :
21 :         lr = 0.01
22 :
23 :         w = w - lr*wb
24 :         b = b - lr*bb
25 :         if epoch%200==1 and n==0 :
26 :             print('w = %6.3f, b = %6.3f'%(w, b))
```

07 : epoch값을 0에서 200 미만까지 바꾸어가며 9~26줄을 2000회 수행합니다.

25 : epoch값을 200으로 나눈 나머지가 1이고 n값이 0일 때 26줄을 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
w = 7.393, b = 8.888
w = 2.997, b = 1.011
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
```

학습 회수에 따라 w, b값이 바뀌는 것을 확인합니다. w값은 3에 b값은 1에 수렴하는 것을 확인합니다.

가중치, 편향 바꿔보기 1

여기서는 가중치와 편향 값을 바꾸어 실습을 진행해 봅니다.


1. 다음과 같이 예제를 수정합니다.

225_9.py

```
04 : w = -10.
05 : b = 10.
```

04 : 가중치 w값을 -10.으로 바꿉니다.

05 : 편향 b값은 10.으로 둡니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
w = -6.877, b = 10.358
w = 2.993, b = 1.022
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
```

w값은 3에 b값은 1에 수렴하는 것을 확인합니다.

가중치, 편향 바꿔보기 2

1. 다음과 같이 예제를 수정합니다.


225_10.py

04 : $w = -100.$

05 : $b = 200.$

04 : 가중치 w 값을 $-100.$ 으로 바꿉니다.

05 : 편향 b 값은 $200.$ 으로 바꿉니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
w = -83.789, b = 194.356
w = 2.884, b = 1.385
w = 3.000, b = 1.001
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
w = 3.000, b = 1.000
```

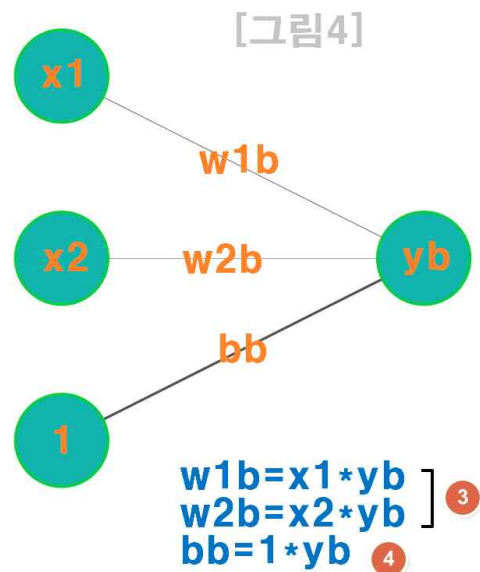
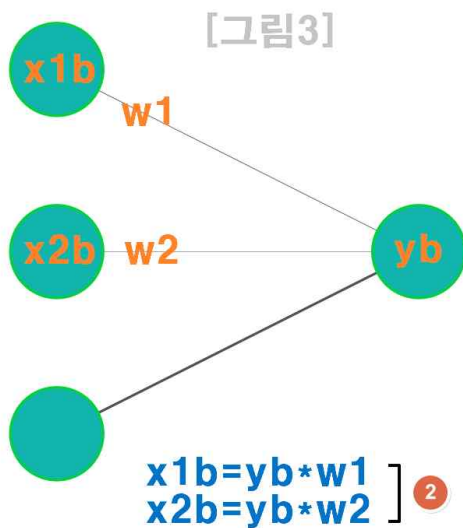
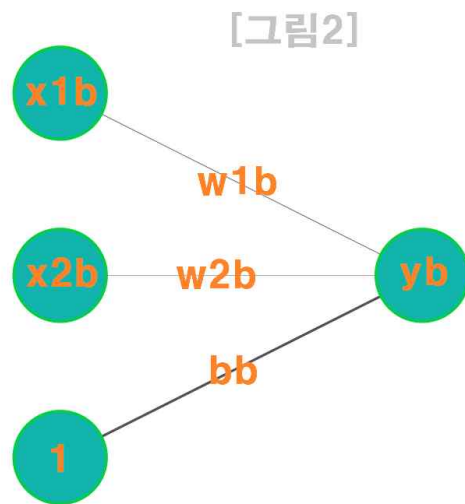
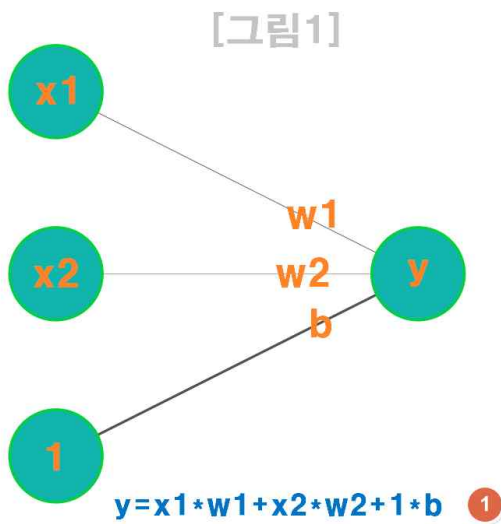
w 값은 3에 b 값은 1에 수렴하는 것을 확인합니다.

02 다양한 인공 신경 구현해 보기

우리는 앞에서 입력1 출력1로 구성된 인공 신경의 동작을 살펴보고 구현해 보았습니다. 여기서는 입력2 출력1, 입력2 출력2, 입력3 출력3으로 구성된 인공 신경의 구조를 살펴보고 수식을 세운 후, 해당 수식에 맞는 인공 신경을 구현해 봅니다.

01 입력2 출력1 인공 신경 구현하기

다음 그림은 입력2 출력1로 구성된 인공 신경을 나타냅니다.



[그림1]은 순전파 과정에 필요한 변수와 수식을 나타냅니다.

[그림2]는 역전파에 필요한 변수입니다. 순전파에 대응되는 변수가 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 변수와 수식을 나타냅니다.

[그림4]는 가중치와 편향의 역전파에 필요한 변수와 수식을 나타냅니다.

*** ② x_{1b} , x_{2b} 노드가 입력층일 경우엔 이 수식은 필요하지 않습니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

순전파

$$x_1 w_1 + x_2 w_2 + 1 b = y$$

입력 역전파

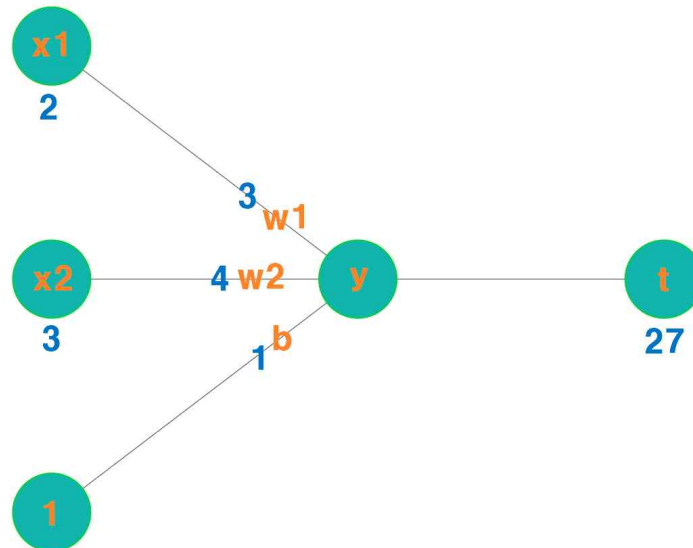
$$\left. \begin{array}{l} y_b w_1 = x_{1b} \\ y_b w_2 = x_{2b} \end{array} \right\} \textcircled{2}$$

가중치, 편향 역전파

$$\left. \begin{array}{l} x_1 y_b = w_{1b} \\ x_2 y_b = w_{2b} \end{array} \right\} \textcircled{3}$$

$$1 y_b = b_b \textcircled{4}$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 x_1 , x_2 는 각각 2, 3, 가중치 w_1 , w_2 는 각각 3, 4, 편향 b 는 1이고 목표값 t 는 27입니다. x_1 , x_2 를 상수로 고정한 채 w_1 , w_2 , b 에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

01 : x_1 , x_2 = 2, 3


02 : t = 27

03 : w_1 = 3


```

04 : w2 = 4
05 : b = 1
06 :
07 : for epoch in range(200):
08 :
09 :     print('epoch = %d' %epoch)
10 :
11 :     y = x1*w1 + x2*w2 + 1*b ❶
12 :     print(' y = %6.3f' %y)
13 :
14 :     E = (y-t)**2/2
15 :     print(' E = %.7f' %E)
16 :     if E < 0.0000001:
17 :         break
18 :
19 :     yb = y - t
20 :     x1b, x2b = yb*w1, yb*w2 ❷
21 :     w1b = yb*x1 ❸
22 :     w2b = yb*x2 ❸
23 :     bb = yb*1 ❹
24 :     print(' x1b, x2b = %6.3f, %6.3f'%(x1b, x2b))
25 :     print(' w1b, w2b, bb = %6.3f, %6.3f, %6.3f'%(w1b, w2b, bb))
26 :
27 :     lr = 0.01
28 :
29 :     w1 = w1 - lr*w1b
30 :     w2 = w2 - lr*w2b
31 :     b = b - lr*bb
32 :     print(' w1, w2, b = %6.3f, %6.3f, %6.3f'%(w1, w2, b))

```

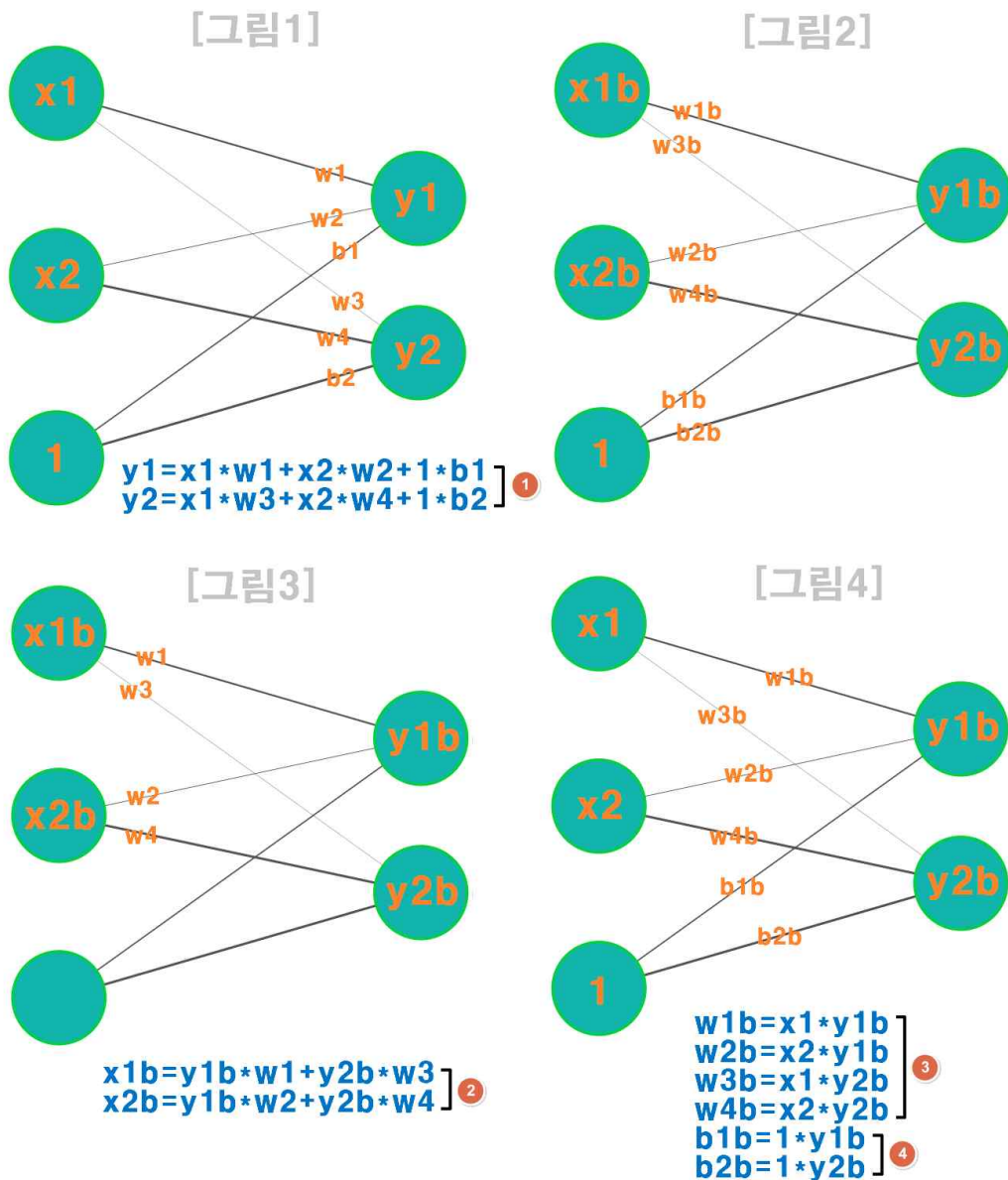
2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 64
y = 26.999
E = 0.0000001
x1b, x2b = -0.002, -0.003
w1b, w2b, bb = -0.001, -0.002, -0.001
w1, w2, b = 4.143, 5.714, 1.571
epoch = 65
y = 27.000
E = 0.0000001
```

64회 째 학습이 완료되는 것을 볼 수 있습니다. 가중치 w_1 , w_2 는 각각 4.143, 5.714, 편향 b 는 1.571에 수렴합니다.

02 입력2 출력2 인공 신경 구현하기

다음 그림은 입력2 출력2로 구성된 인공 신경을 나타냅니다.



[그림1]은 순전파 과정에 필요한 변수와 수식을 나타냅니다.

[그림2]는 역전파에 필요한 변수입니다. 순전파에 대응되는 변수가 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 변수와 수식을 나타냅니다.

[그림4]는 가중치와 편향의 역전파에 필요한 변수와 수식을 나타냅니다.

*** ② $x1b, x2b$ 노드가 입력층일 경우엔 이 수식은 필요하지 않습니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

순전파

$$\left. \begin{array}{l} x_1w_1 + x_2w_2 + 1b_1 = y_1 \\ x_1w_3 + x_2w_4 + 1b_2 = y_2 \end{array} \right] \textcircled{1}$$

입력 역전파

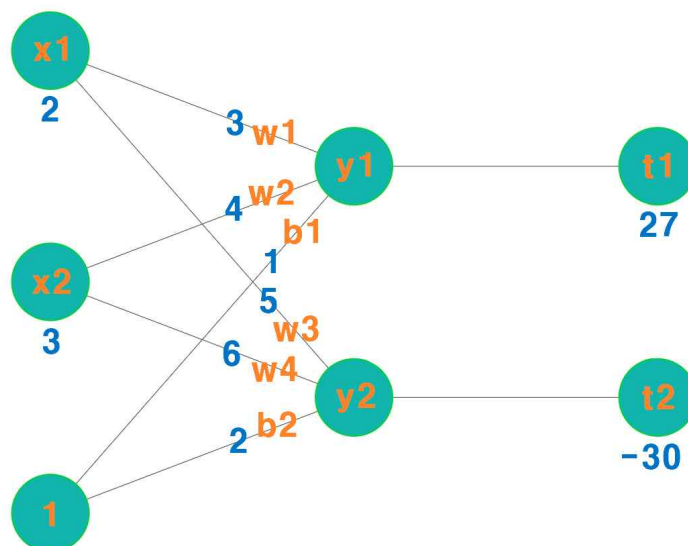
$$\left. \begin{array}{l} y_{1b}w_1 + y_{2b}w_3 = x_{1b} \\ y_{1b}w_2 + y_{2b}w_4 = x_{2b} \end{array} \right] \textcircled{2}$$

가중치, 편향 역전파

$$\left. \begin{array}{l} x_1y_{1b} = w_{1b} \\ x_2y_{1b} = w_{2b} \\ x_1y_{2b} = w_{3b} \\ x_2y_{2b} = w_{4b} \end{array} \right] \textcircled{3}$$

$$\left. \begin{array}{l} 1y_{1b} = b_{1b} \\ 1y_{2b} = b_{2b} \end{array} \right] \textcircled{4}$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 x_1 , x_2 는 각각 2, 3, 가중치 w_1 , w_2 , 편향 b_1 은 각각 3, 4, 1, 가중치 w_3 , w_4 , 편향 b_2 는 각각 5, 6, 2이고 목표값 t_1 , t_2 는 각각 27, -30입니다. x_1 , x_2 를 상수로 고정한 채 w_1 , w_2 , w_3 , w_4 , b_1 , b_2 에 대해 학습을 수행해 봅니다.


*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 이전 예제를 복사합니다.

2. 다음과 같이 예제를 수정합니다.

```
01 : x1, x2 = 2, 3
02 : t1, t2 = 27, -30
03 : w1, w3 = 3, 5
04 : w2, w4 = 4, 6
05 : b1, b2 = 1, 2
06 :
07 : for epoch in range(200):
08 :
09 :     print('epoch = %d' %epoch)
10 :
11 :     y1 = x1*w1 + x2*w2 + 1*b1 ❶
12 :     y2 = x1*w3 + x2*w4 + 1*b2 ❶
13 :     print(' y1, y2 = %6.3f, %6.3f' %(y1, y2))
14 :
15 :     E = (y1-t1)**2/2 + (y2-t2)**2/2
16 :     print(' E = %.7f' %E)
17 :     if E < 0.0000001:
18 :         break
19 :
20 :     y1b, y2b = y1 - t1, y2 - t2
21 :     x1b, x2b = y1b*w1+y2b*w3, y1b*w2+y2b*w4 ❷
22 :     w1b, w3b = x1*y1b, x1*y2b ❸
23 :     w2b, w4b = x2*y1b, x2*y2b ❸
24 :     b1b, b2b = 1*y1b, 1*y2b ❹
25 :     print(' x1b, x2b = %6.3f, %6.3f'%(x1b, x2b))
26 :     print(' w1b, w3b = %6.3f, %6.3f'%(w1b, w3b))
27 :     print(' w2b, w4b = %6.3f, %6.3f'%(w2b, w4b))
28 :     print(' b1b, b2b = %6.3f, %6.3f'%(b1b, b2b))
29 :
30 :     lr = 0.01
31 :
32 :     w1, w3 = w1 - lr*w1b, w3 - lr*w3b
33 :     w2, w4 = w2 - lr*w2b, w4 - lr*w4b
34 :     b1, b2 = b1 - lr*b1b, b2 - lr*b2b
35 :     print(' w1, w3 = %6.3f, %6.3f'%(w1, w3))
36 :     print(' w2, w4 = %6.3f, %6.3f'%(w2, w4))
```

```
37 : print(' b1, b2 = %6.3f, %6.3f'%(b1, b2))
```

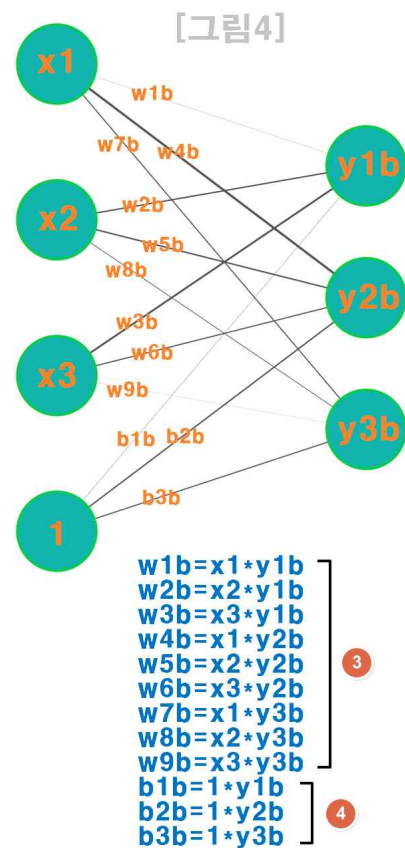
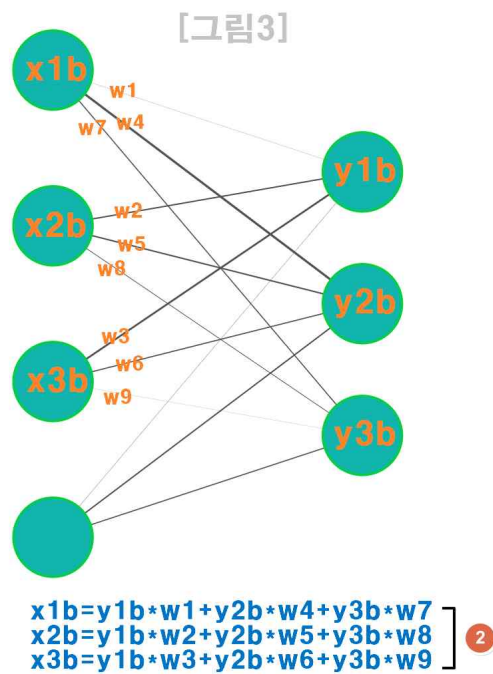
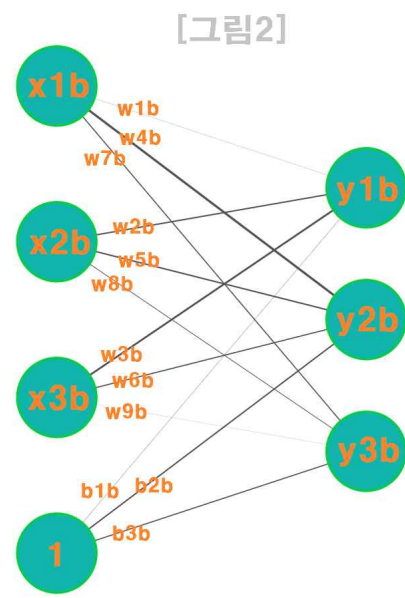
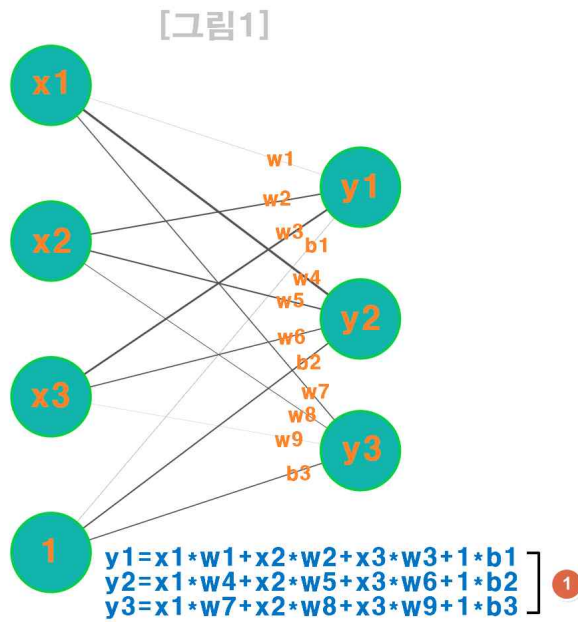
2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 78
y1, y2 = 27.000, -30.000
E = 0.0000001
x1b, x2b = -0.002, -0.004
w1b, w3b = -0.000, 0.001
w2b, w4b = -0.000, 0.001
b1b, b2b = -0.000, 0.000
w1, w3 = 4.143, -3.571
w2, w4 = 5.714, -6.857
b1, b2 = 1.571, -2.286
epoch = 79
y1, y2 = 27.000, -30.000
E = 0.0000001
```

79회 째 학습이 완료되는 것을 볼 수 있습니다. 가중치 w1, w2는 각각 4.143, 5.714, 편향 b1은 1.571, 가중치 w3, w4는 각각 -3.571, -6.857 편향 b2는 -2.286에 수렴합니다.

03 입력3 출력3 인공 신경 구현하기

다음 그림은 입력3 출력3으로 구성된 인공 신경을 나타냅니다.



[그림1]은 순전파 과정에 필요한 변수와 수식을 나타냅니다.

[그림2]는 역전파에 필요한 변수입니다. 순전파에 대응되는 변수가 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 변수와 수식을 나타냅니다.

[그림4]는 가중치와 편향의 역전파에 필요한 변수와 수식을 나타냅니다.

*** ② x_{1b} , x_{2b} , x_{3b} 노드가 입력층일 경우엔 이 수식은 필요하지 않습니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

순전파

$$\left. \begin{aligned} x_1 w_1 + x_2 w_2 + x_3 w_3 + 1b_1 &= y_1 \\ x_1 w_4 + x_2 w_5 + x_3 w_6 + 1b_2 &= y_2 \\ x_1 w_7 + x_2 w_8 + x_3 w_9 + 1b_3 &= y_3 \end{aligned} \right\} \textcircled{1}$$

입력 역전파

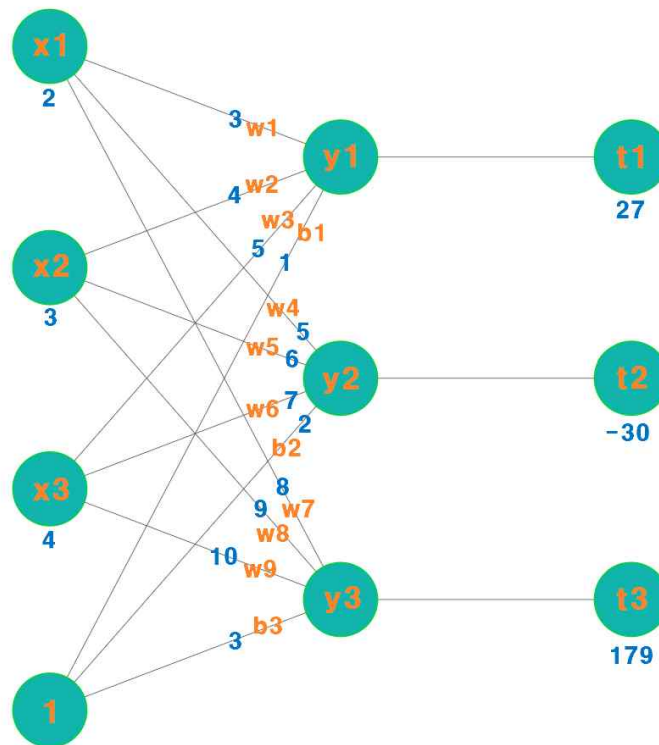
$$\left. \begin{aligned} y_{1b} w_1 + y_{2b} w_4 + y_{3b} w_7 &= x_{1b} \\ y_{1b} w_2 + y_{2b} w_5 + y_{3b} w_8 &= x_{2b} \\ y_{1b} w_3 + y_{2b} w_6 + y_{3b} w_9 &= x_{3b} \end{aligned} \right\} \textcircled{2}$$

가중치, 편향 역전파

$$\left. \begin{aligned} x_1 y_{1b} &= w_{1b} \\ x_2 y_{1b} &= w_{2b} \\ x_3 y_{1b} &= w_{3b} \\ x_1 y_{2b} &= w_{4b} \\ x_2 y_{2b} &= w_{5b} \\ x_3 y_{2b} &= w_{6b} \\ x_1 y_{3b} &= w_{7b} \\ x_2 y_{3b} &= w_{8b} \\ x_3 y_{3b} &= w_{9b} \end{aligned} \right\} \textcircled{3}$$

$$\left. \begin{aligned} 1y_{1b} &= b_{1b} \\ 1y_{2b} &= b_{2b} \\ 1y_{3b} &= b_{3b} \end{aligned} \right\} \textcircled{4}$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 x_1, x_2, x_3 은 각각 2, 3, 4, 가중치 w_1, w_2, w_3 , 편향 b_1 은 각각 3, 4, 5, 1, 가중치 w_4, w_5, w_6 , 편향 b_2 는 각각 5, 6, 7, 2, 가중치 w_7, w_8, w_9 , 편향 b_3 은 각각 8, 9, 10, 3이고 목표값 t_1, t_2, t_3 은 각각 27, -30, 179입니다. x_1, x_2, x_3 를 상수로 고정한 채 $w_1 \sim w_9, b_1 \sim b_3$ 에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 이전 예제를 복사합니다.


2. 다음과 같이 예제를 수정합니다.

```
01 : x1, x2, x3 = 2, 3, 4
02 : t1, t2, t3 = 27, -30, 179
03 : w1, w4, w7 = 3, 5, 8
04 : w2, w5, w8 = 4, 6, 9
05 : w3, w6, w9 = 5, 7, 10
06 : b1, b2, b3 = 1, 2, 3
07 :
08 : for epoch in range(200):
09 :
10 :     print('epoch = %d' %epoch)
11 :
```

```

12 : y1 = x1*w1 + x2*w2 + x3*w3 + 1*b1 ❶
13 : y2 = x1*w4 + x2*w5 + x3*w6 + 1*b2 ❶
14 : y3 = x1*w7 + x2*w8 + x3*w9 + 1*b3 ❶
15 : print(' y1, y2, y3 = %6.3f, %6.3f, %6.3f' %(y1, y2, y3))
16 :
17 : E = (y1-t1)**2/2 + (y2-t2)**2/2+ (y3-t3)**2/2
18 : print(' E = %.7f' %E)
19 : if E < 0.0000001:
20 :     break
21 :
22 : y1b, y2b, y3b = y1 - t1, y2 - t2, y3 - t3
23 : x1b = y1b*w1+y2b*w4+y3b*w7 ❷
24 : x2b = y1b*w2+y2b*w5+y3b*w8 ❷
25 : x3b = y1b*w3+y2b*w6+y3b*w9 ❷
26 : w1b, w4b, w7b = x1*y1b, x1*y2b, x1*y3b ❸
27 : w2b, w5b, w8b = x2*y1b, x2*y2b, x2*y3b ❸
28 : w3b, w6b, w9b = x3*y1b, x3*y2b, x3*y3b ❸
29 : b1b, b2b, b3b = 1*y1b, 1*y2b, 1*y3b ❹
30 : print(' x1b, x2b, x3b = %6.3f, %6.3f, %6.3f'%(x1b, x2b, x3b))
31 : print(' w1b, w4b, w7b = %6.3f, %6.3f, %6.3f'%(w1b, w4b, w7b))
32 : print(' w2b, w5b, w8b = %6.3f, %6.3f, %6.3f'%(w2b, w5b, w8b))
33 : print(' w3b, w6b, w9b = %6.3f, %6.3f, %6.3f'%(w3b, w6b, w9b))
34 : print(' b1b, b2b, b3b = %6.3f, %6.3f, %6.3f'%(b1b, b2b, b3b))
35 :
36 : lr = 0.01
37 :
38 : w1, w4, w7 = w1 - lr*w1b, w4 - lr*w4b, w7 - lr*w7b
39 : w2, w5, w8 = w2 - lr*w2b, w5 - lr*w5b, w8 - lr*w8b
40 : w3, w6, w9 = w3 - lr*w3b, w6 - lr*w6b, w9 - lr*w9b
41 : b1, b2, b3 = b1 - lr*b1b, b2 - lr*b2b, b3 - lr*b3b
42 : print(' w1, w4, w7 = %6.3f, %6.3f, %6.3f'%(w1, w4, w7))
43 : print(' w2, w5, w8 = %6.3f, %6.3f, %6.3f'%(w2, w5, w8))
44 : print(' w3, w6, w9 = %6.3f, %6.3f, %6.3f'%(w3, w6, w9))
45 : print(' b1, b2, b3 = %6.3f, %6.3f, %6.3f'%(b1, b2, b3))

```

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

epoch = 35
y1, y2, y3 = 27.000, -30.000, 179.000
E = 0.0000001
x1b, x2b, x2b = -0.005, -0.007, -0.009
w1b, w4b, w7b = 0.000, 0.001, -0.001
w2b, w5b, w8b = 0.000, 0.001, -0.001
w3b, w6b, w9b = 0.000, 0.001, -0.001
b1b, b2b, b3b = 0.000, 0.000, -0.000
w1, w4, w7 = 2.200, -0.867, 14.200
w2, w5, w8 = 2.800, -2.800, 18.300
w3, w6, w9 = 3.400, -4.733, 22.400
b1, b2, b3 = 0.600, -0.933, 6.100
epoch = 36
y1, y2, y3 = 27.000, -30.000, 179.000
E = 0.0000001

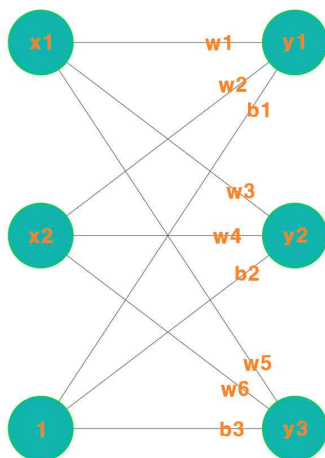
```

36회 째 학습이 완료되는 것을 볼 수 있습니다. 가중치 $w1 \sim w9$, 편향 $b1 \sim b3$ 이 수렴하는 값도 살펴봅니다.

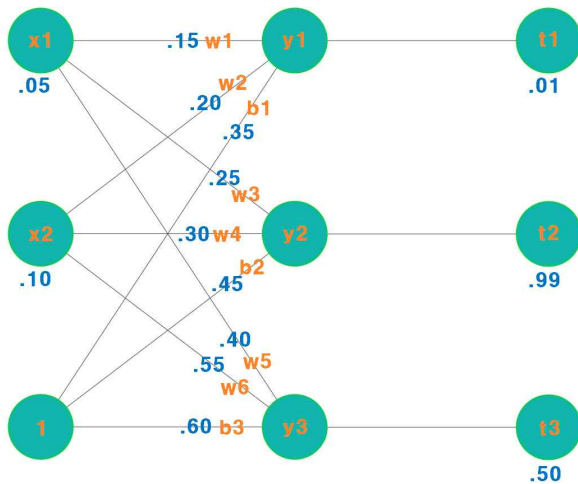
연습문제

❶ 입력2 출력3

1. 다음은 입력2 출력3의 단일 인공 신경입니다. 이 인공 신경의 순전파, 역전파 수식을 구합니다.

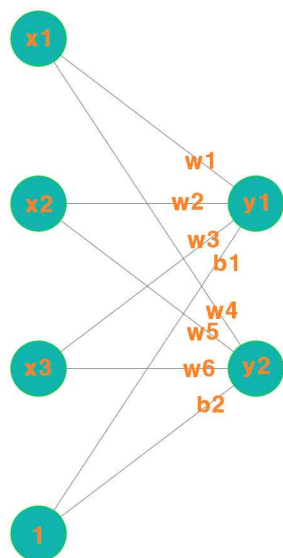


2. 앞에서 구한 수식을 이용하여 다음과 같이 초기화된 인공 신경을 구현하고 학습시켜 봅니다. 입력값 x_1 , x_2 는 상수로 처리합니다.

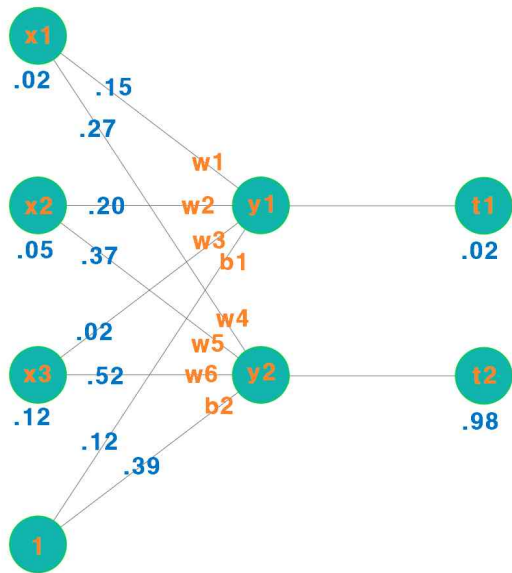


② 입력3 출력2

1. 다음은 입력2 출력3의 단일 인공 신경입니다. 이 인공 신경의 순전파, 역전파 수식을 구합니다.



2. 앞에서 구한 수식을 이용하여 다음과 같이 초기화된 인공 신경을 구현하고 학습시켜 봅니다. 입력값 x_1 , x_2 , x_3 은 상수로 처리합니다.

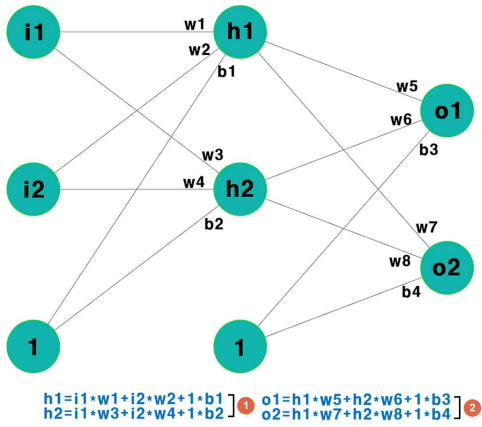


04 입력2 은닉2 출력2 인공 신경망 구현하기

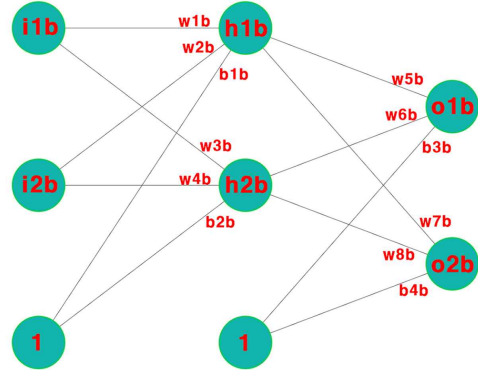
여기서는 은닉 신경을 추가한 인공 신경망을 구현해 봅니다. 은닉 신경이 추가되면 인공 신경망이 됩니다. 은닉 신경이 추가된 경우에도 순전파, 역전파 수식을 구하는 방식은 이전과 같습니다.

다음 그림은 입력2 은닉2 출력2로 구성된 인공 신경을 나타냅니다.

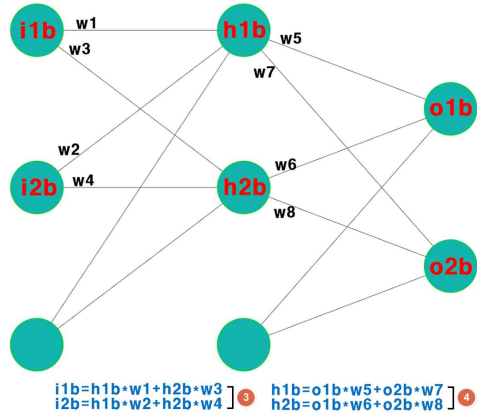
[그림1]



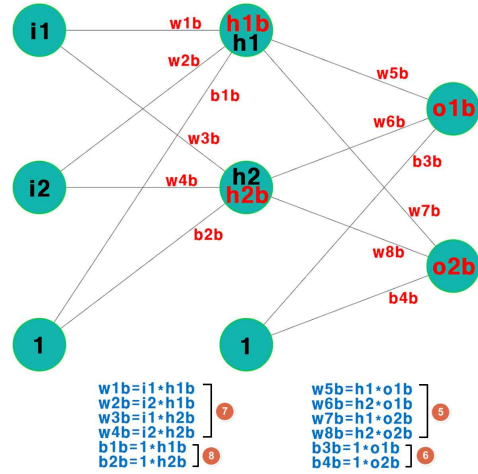
[그림2]



[그림3]



[그림4]



[그림1]은 순전파 과정에 필요한 변수와 수식을 나타냅니다.

[그림2]는 역전파에 필요한 변수입니다. 순전파에 대응되는 변수가 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 변수와 수식을 나타냅니다.

[그림4]는 가중치와 편향의 역전파에 필요한 변수와 수식을 나타냅니다.

*** ③ i1b, i2b 노드가 입력층일 경우엔 이 수식은 필요하지 않습니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

순전파	
$\begin{aligned} i_1 w_1 + i_2 w_2 + 1 b_1 &= h_1 \\ i_1 w_3 + i_2 w_4 + 1 b_2 &= h_2 \end{aligned} \quad \text{①}$	
$\begin{aligned} h_1 w_5 + h_2 w_6 + 1 b_3 &= o_1 \\ h_1 w_7 + h_2 w_8 + 1 b_4 &= o_2 \end{aligned} \quad \text{②}$	

입력 역전파

$$\begin{bmatrix} o_{1b}w_5 + o_{2b}w_7 = h_{1b} \\ o_{1b}w_6 + o_{2b}w_8 = h_{2b} \end{bmatrix} \quad 4$$

가중치, 편향 역전파

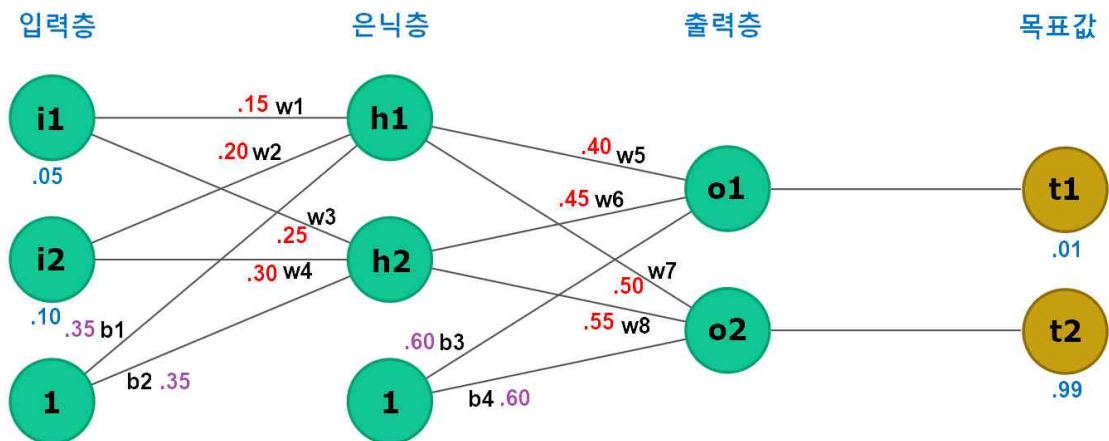
$$\begin{bmatrix} i_1h_{1b} = w_{1b} \\ i_2h_{1b} = w_{2b} \\ i_1h_{2b} = w_{3b} \\ i_2h_{2b} = w_{4b} \end{bmatrix} \quad 7$$

$$\begin{bmatrix} 1h_{1b} = b_{1b} \\ 1h_{2b} = b_{2b} \end{bmatrix} \quad 8$$

$$\begin{bmatrix} h_1o_{1b} = w_{5b} \\ h_2o_{1b} = w_{6b} \\ h_1o_{2b} = w_{7b} \\ h_2o_{2b} = w_{8b} \end{bmatrix} \quad 5$$

$$\begin{bmatrix} 1o_{1b} = b_{3b} \\ 1o_{2b} = b_{4b} \end{bmatrix} \quad 6$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



입력값, 가중치값, 편향값은 그림을 참조합니다. i1, i2를 상수로 고정한 채 w1~w8, b1~b4에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 다음과 같이 예제를 작성합니다.

```

01 : i1, i2 = .05, .10
11 : t1, t2 = .01, .99
02 :
03 : w1, w3 = .15, .25
04 : w2, w4 = .20, .30
05 : b1, b2 = .35, .35
06 :
07 : w5, w7 = .40, .50
08 : w6, w8 = .45, .55
09 : b3, b4 = .60, .60
07 :
08 : for epoch in range(1000):
09 :
10 :     print('epoch = %d' %epoch)
11 :
15 :     h1 = i1*w1 + i2*w2 + 1*b1 ❶
16 :     h2 = i1*w3 + i2*w4 + 1*b2 ❶
17 :     o1 = h1*w5 + h2*w6 + 1*b3 ❷
18 :     o2 = h1*w7 + h2*w8 + 1*b4 ❷
19 :     print(' h1, h2 = %6.3f, %6.3f' %(h1, h2))
20 :     print(' o1, o2 = %6.3f, %6.3f' %(o1, o2))
16 :
22 :     E = (o1-t1)**2/2 + (o2-t2)**2/2
18 :     print(' E = %.7f' %E)
19 :     if E < 0.0000001:
20 :         break
21 :
26 :     o1b, o2b = o1 - t1, o2 - t2
27 :     h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8 ❹
28 :     w1b, w3b = i1*h1b, i1*h2b ❷
29 :     w2b, w4b = i2*h1b, i2*h2b ❷
30 :     b1b, b2b = 1*h1b, 1*h2b ❸
31 :     w5b, w7b = h1*o1b, h1*o2b ❺
32 :     w6b, w8b = h2*o1b, h2*o2b ❺
33 :     b3b, b4b = 1*o1b, 1*o2b ❻
34 :     print(' w1b, w3b = %6.3f, %6.3f'%(w1b, w3b))
35 :     print(' w2b, w4b = %6.3f, %6.3f'%(w2b, w4b))
36 :     print(' b1b, b2b = %6.3f, %6.3f'%(b1b, b2b))
37 :     print(' w5b, w7b = %6.3f, %6.3f'%(w5b, w7b))
38 :     print(' w6b, w8b = %6.3f, %6.3f'%(w6b, w8b))

```




```

39 : print(' b3b, b4b = %6.3f, %6.3f'%(b3b, b4b))
35 :
36 : lr = 0.01
37 :
43 : w1, w3 = w1 - lr*w1b, w3 - lr*w3b
44 : w2, w4 = w2 - lr*w2b, w4 - lr*w4b
45 : b1, b2 = b1 - lr*b1b, b2 - lr*b2b
46 : w5, w7 = w5 - lr*w5b, w7 - lr*w7b
47 : w6, w8 = w6 - lr*w6b, w8 - lr*w8b
48 : b3, b4 = b3 - lr*b3b, b4 - lr*b4b
49 : print(' w1, w3 = %6.3f, %6.3f'%(w1, w3))
50 : print(' w2, w4 = %6.3f, %6.3f'%(w2, w4))
51 : print(' b1, b2 = %6.3f, %6.3f'%(b1, b2))
52 : print(' w5, w7 = %6.3f, %6.3f'%(w5, w7))
53 : print(' w6, w8 = %6.3f, %6.3f'%(w6, w8))
54 : print(' b3, b4 = %6.3f, %6.3f'%(b3, b4))

```

08 : epoch값을 0에서 1000 미만까지 바꾸어가며 10~54줄을 수행합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

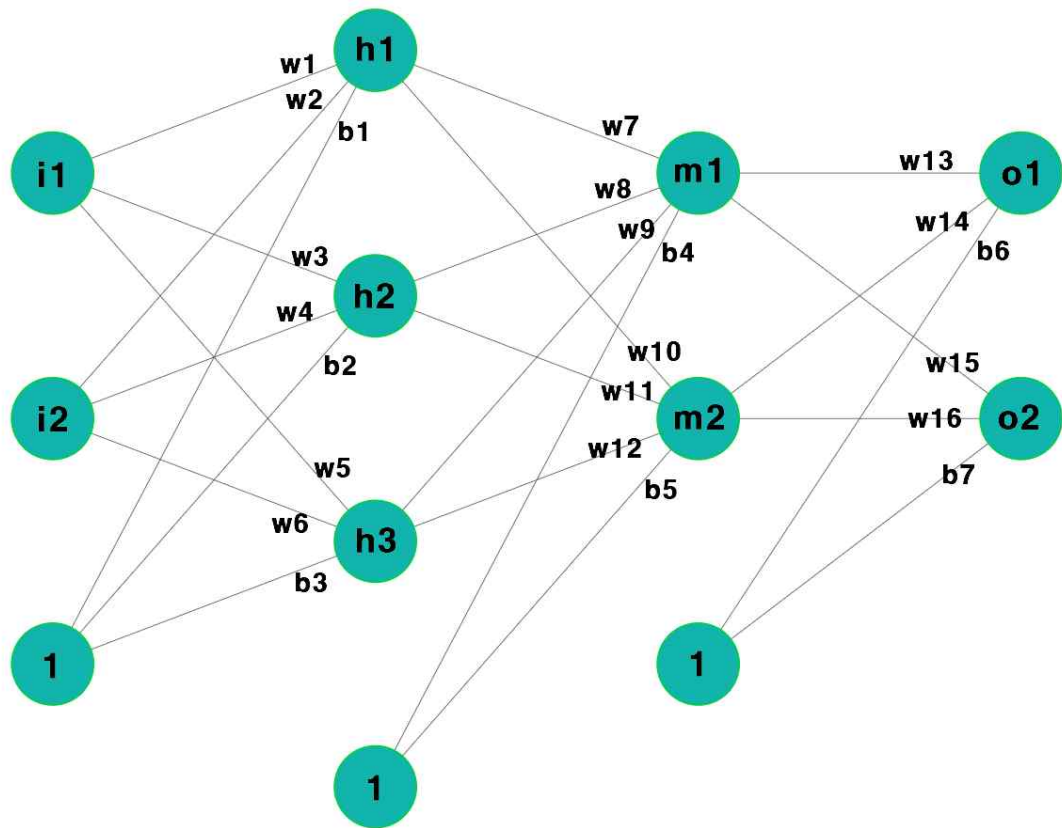
epoch = 664
h1, h2 = 0.239, 0.226
o1, o2 = 0.010, 0.990
E = 0.0000001
w1b, w3b = -0.000, 0.000
w2b, w4b = -0.000, 0.000
b1b, b2b = -0.000, 0.000
w5b, w7b = 0.000, -0.000
w6b, w8b = 0.000, -0.000
b3b, b4b = 0.000, -0.000
w1, w3 = 0.143, 0.242
w2, w4 = 0.186, 0.284
b1, b2 = 0.213, 0.186
w5, w7 = 0.203, 0.533
w6, w8 = 0.253, 0.583
b3, b4 = -0.095, 0.730
epoch = 665
h1, h2 = 0.239, 0.226
o1, o2 = 0.010, 0.990
E = 0.0000001

```

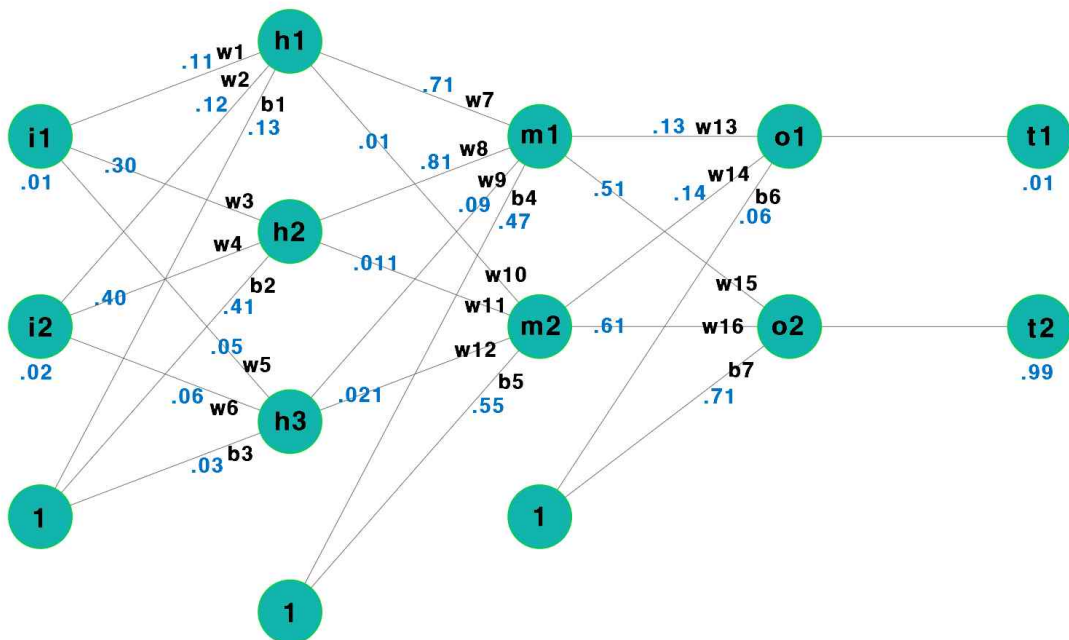
연습문제

입력2 은닉3 은닉2 출력2

1. 다음은 입력2 은닉3 은닉2 출력2의 심층 인공 신경망입니다. 이 신경망에는 2개의 은닉층이 포함되어 있습니다. 일반적으로 은닉층이 2층 이상일 경우 심층 인공 신경망이라고 합니다. 이 신경망의 순전파, 역전파 수식을 구합니다.



2. 앞에서 구한 수식을 이용하여 다음과 같이 초기화된 인공 신경을 구현하고 학습시켜 봅시다. 입력값 $i1, i2$ 는 상수로 처리합니다.



03 활성화 함수 추가하기

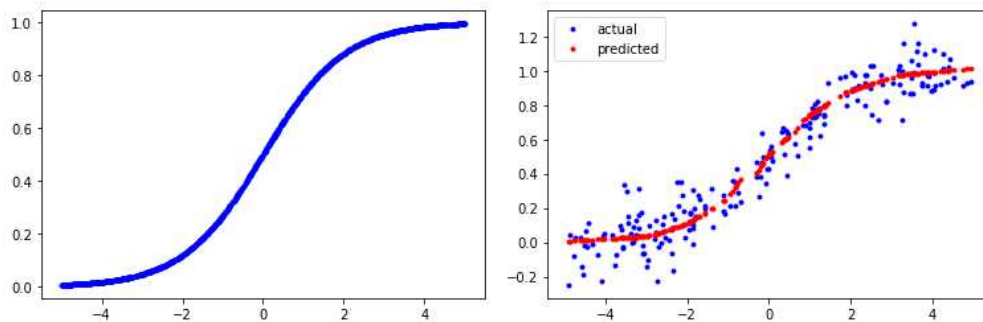
일반적으로 인공 신경의 출력단에는 활성화 함수가 추가됩니다. 활성화 함수를 추가하면 입력된 데이터에 대해 복잡한 패턴의 학습이 가능해집니다. 또 인공 신경의 출력값을 어떤 범위로 제한할 수도 있습니다. 여기서는 주로 사용되는 몇 가지 활성화 함수를 살펴보고, 활성화 함수가 필요한 이유에 대해서도 살펴봅니다. 그리고 활성화 함수를 추가한 인공 신경망을 구현해 봅니다.

01 활성화 함수 살펴보기

우리는 앞에서 다음과 같은 활성화 함수를 직접 그려보았습니다.

sigmoid 함수

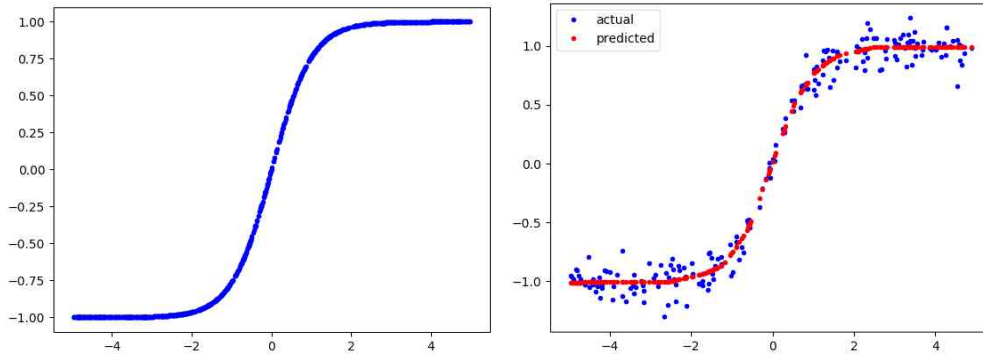
다음은 sigmoid 함수에 대한 그래프와 인공 신경망 학습 후, 예측 그래프입니다.



$$y = \frac{1}{1 + e^{-x}} \quad (-5 \leq x \leq 5)$$

tanh 함수

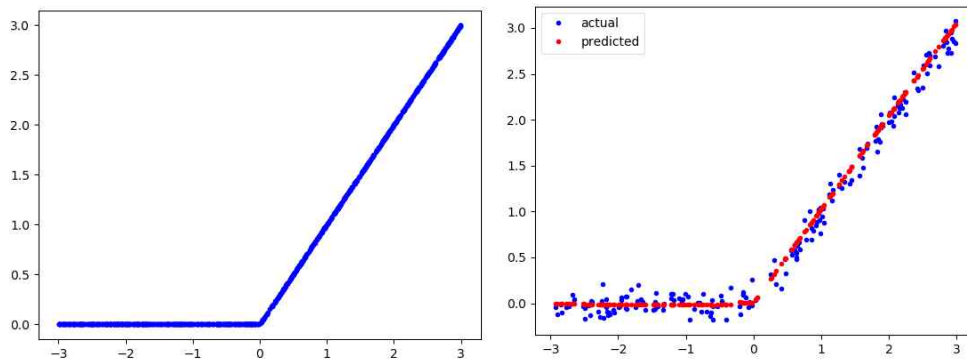
다음은 tanh 함수에 대한 그래프와 인공 신경망 학습 후, 예측 그래프입니다.



$$y = \tanh(x) \quad (-5 \leq x \leq 5)$$

relu 함수

다음은 relu 함수에 대한 그래프와 인공 신경망 학습 후, 예측 그래프입니다.



$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \quad (-3 \leq x \leq 3)$$

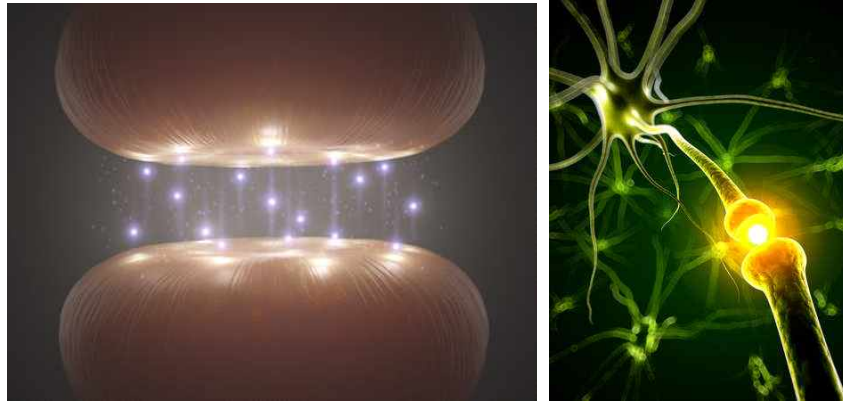
02 활성화 함수의 필요성

여기서는 활성화 함수가 무엇인지, 활성화 함수는 왜 필요한지, 어떤 활성화 함수가 있는지 살펴봅니다.

활성화 함수는 무엇인가요?

활성화 함수는 인공 신경망에 더해져 복잡한 패턴을 학습하게 해줍니다. 즉, 다양한 형태의 입력값에 대해 신경망을 거쳐 나온 출력값이 우리가 원하는 목표값에 가깝게 해 주기가 더 쉬워집니다. 우리 두뇌에 있는 생체 신경과 비교할 때, 활성화 함수는 신경 말단에서 다음 신경으로 전달될 신호를 결정하는 시냅스와 같은 역할을 합니다. 시냅스는 이전 신경 세포로부터

오는 출력 신호를 받아 다음 신경 세포가 받아들일 수 있는 입력 신호로 형태를 변경합니다. 마찬가지로 활성화 함수는 이전 인공 신경으로부터 오는 출력 신호를 받아 다음 인공 신경이 받아들일 수 있는 입력 신호로 형태를 변경해 주는 역할을 합니다.



[시냅스]

활성화 함수는 왜 필요한가요?

앞에서 언급했던 생물학적 유사성과는 별도로 인공 신경의 출력값을 우리가 원하는 어떤 범위로 제한해 줍니다. 이것은 활성화 함수로의 입력이 $w \cdot x + b$ 이기 때문입니다. 여기서 w 는 인공 신경의 가중치, x 는 입력, b 는 그것에 더해지는 편향입니다. 이 값은 어떤 범위로 제한되지 않으면 신경망을 거치며 순식간에 아주 커지게 됩니다. 특히 수백만 개의 매개변수(가중치와 편향)으로 구성된 아주 깊은 신경망의 경우에는 더욱 그렇습니다. 인공 신경을 거치며 반복적으로 계산되는 $w \cdot x + b$ 는 factorial 연산과 같은 효과를 내며 이것은 순식간에 컴퓨터의 계산 범위를 넘어서게 됩니다. 인공 신경망을 학습시키다보면 Nan이라고 표시되는 경우가 있는데 이 경우가 그런 경우에 해당합니다.

$n!$	$= n$
0	$= 1$
1	$= 1$
2	$= 2$
3	$= 6$
4	$= 24$
5	$= 120$
6	$= 720$
7	$= 5,040$
8	$= 40,320$
9	$= 362,880$
10	$= 3,628,800$
11	$= 39,916,800$
12	$= 479,001,600$

어떤 활성화 함수가 있나요?

❶ 시그모이드

시그모이드 활성화 함수는 단지 역사적인 이유로 여기에 소개되며 일반적으로 딥러닝에서 많이 사용되지 않습니다. 시그모이드 함수는 3층 정도로 구성된 인공 신경망에 적용될 때는 학습이 잘 되지만 깊은 신경망에 적용될 때는 학습이 잘 되지 않습니다. 시그모이드 함수는 계산에 시간이 걸리고, 입력값이 아무리 크더라도 출력값의 범위가 0에서 1사이로 매우 작아 신경망을 거칠수록 출력값은 순식간에 작아져 0에 수렴하게 됩니다. 이것은 신경을 거치면서 신호가 점점 작아져 출력에 도달하는 신호가 아주 작거나 없어지는 것과 같습니다. 출력에 미치는 신호가 아주 작거나 없다는 것은 역으로 전달될 신호도 아주 작거나 없다는 것을 의미합니다. 시그모이드 함수는 일반적으로 0이나 1로 분류하는 이진 분류 문제에 사용됩니다. 심층 신경망에서 시그모이드 함수를 사용해야 할 경우엔 출력층에서만 사용하도록 합니다.

❷ 소프트맥스

소프트맥스는 활성화 함수는 시그모이드 활성화 함수가 더욱 일반화된 형태입니다. 이것은 다중 클래스 분류 문제에 사용됩니다. 시그모이드 함수와 비슷하게 이것은 0에서 1사이의 값들을 생성합니다. 소프트맥스 함수는 은닉층에서는 사용하지 않으며, 다중 분류 모델에서 출력층에서만 사용됩니다.

❸ tanh

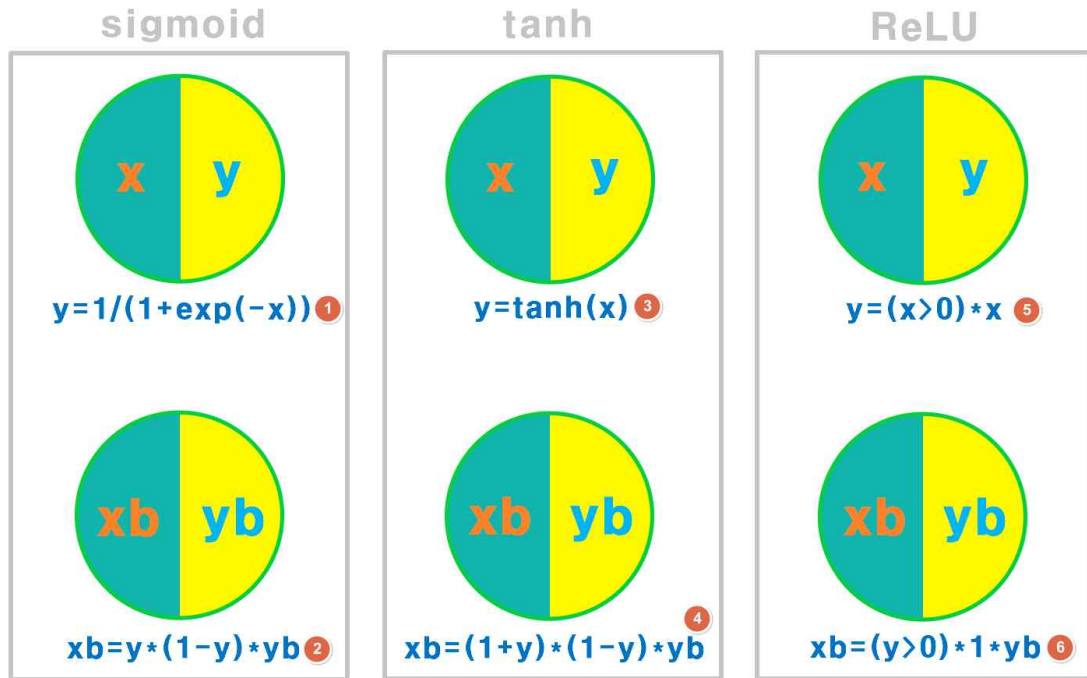
tanh 함수는 출력값의 범위가 -1에서 1사이라는 것을 빼고는 시그모이드 함수와 유사합니다. 시그모이드 함수와 같이 많이 사용되지 않습니다.

❹ ReLU

ReLU 함수는 딥러닝에서 가장 인기 있는 활성화 함수입니다. 특히 합성곱 신경망(CNN)에서 많이 사용됩니다. ReLU 함수는 계산이 빠르고 심층 신경망에서도 신호 전달이 잘 됩니다. ReLU 함수의 경우 입력값이 음수가 될 경우 출력이 0이 되기 때문에 이런 경우에는 어떤 노드를 완전히 죽게 하여 어떤 것도 학습하지 않게 합니다. 이러한 노드가 많으면 많을수록 신경망 전체적으로 학습이 되지 않는 단점이 있습니다. ReLU의 다른 문제는 활성화 값의 극대화입니다. 왜냐하면 ReLU의 상한값은 무한이기 때문입니다. 이것은 가끔 사용할 수 없는 노드를 만들어 학습을 방해하게 됩니다. 이러한 문제들은 초기 가중치 값을 고르게 할당하여 해결할 수 있습니다. 일반적으로 은닉층에는 ReLU 함수를 적용하고, 출력층은 시그모이드 함수나 소프트맥스 함수를 적용합니다.

03 활성화 함수의 순전파와 역전파

여기서는 sigmoid, tanh, ReLU 활성화 함수의 순전파와 역전파 수식을 살펴보고, 앞에서 구현한 인공 신경망에 활성화 함수를 적용하여 봅니다. 다음 그림은 활성화 함수의 순전파와 역전파 수식을 나타냅니다.



- sigmoid 함수의 경우 순전파 출력 y 값이 0이나 1에 가까울수록 역전파 x_b 값은 0에 가까워집니다. 순전파 출력 y 값이 0이나 1에 가깝다는 것은 순전파 입력값 x 의 크기가 양 또는 음의 방향으로 어느 정도 크다는 의미입니다.
- tanh 함수의 경우 순전파 출력 y 값이 -1이나 1에 가까울수록 역전파 x_b 값은 0에 가까워집니다. 순전파 출력 y 값이 -1이나 1에 가깝다는 것은 순전파 입력값 x 의 크기가 양 또는 음의 방향으로 어느 정도 크다는 의미입니다.
- ReLU 함수의 경우 순전파 입력값 x 값이 0보다 크면 x 값이 y 로 전달되며, 0보다 작거나 같으면 0값이 y 로 전달됩니다. 역전파의 경우 순전파 출력값 y 가 0보다 크면 y_b 값이 x_b 로 전달되며, 출력값 y 가 0보다 작거나 같으면 x_b 로 0이 전달됩니다. 이 경우 x_b 에서 전 단계의 모든 노드로 전달되는 역전파 값은 0이 됩니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

시그모이드 순전파와 역전파

$$y = \frac{1}{1 + e^{-x}} \quad (1) \quad x_b = y(1 - y)y_b \quad (2)$$

tanh 순전파와 역전파

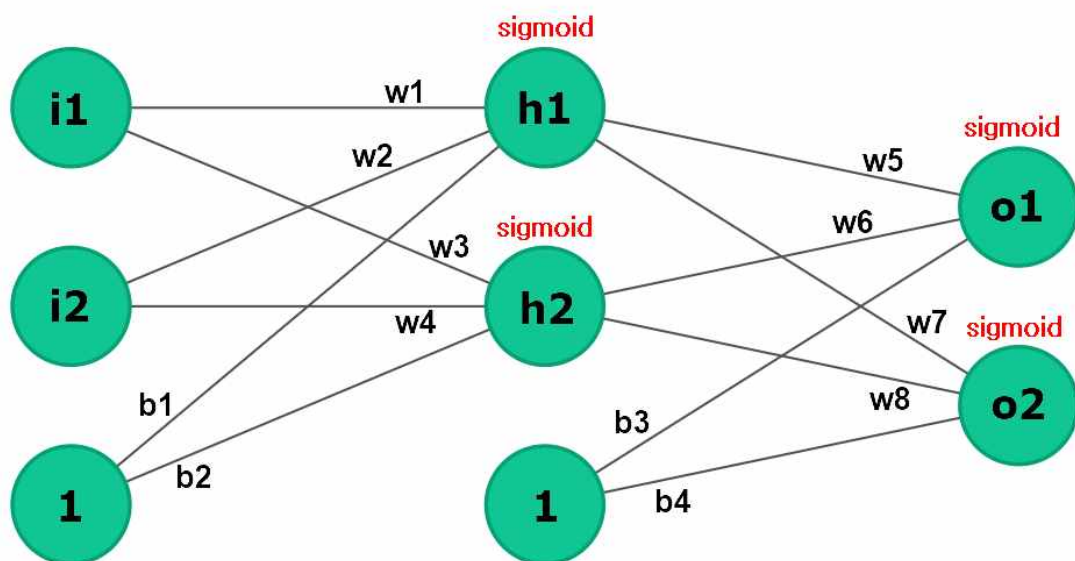
$$y = \tanh(x) \quad x_b = (1+y)(1-y)y_b$$

ReLU 순전파와 역전파

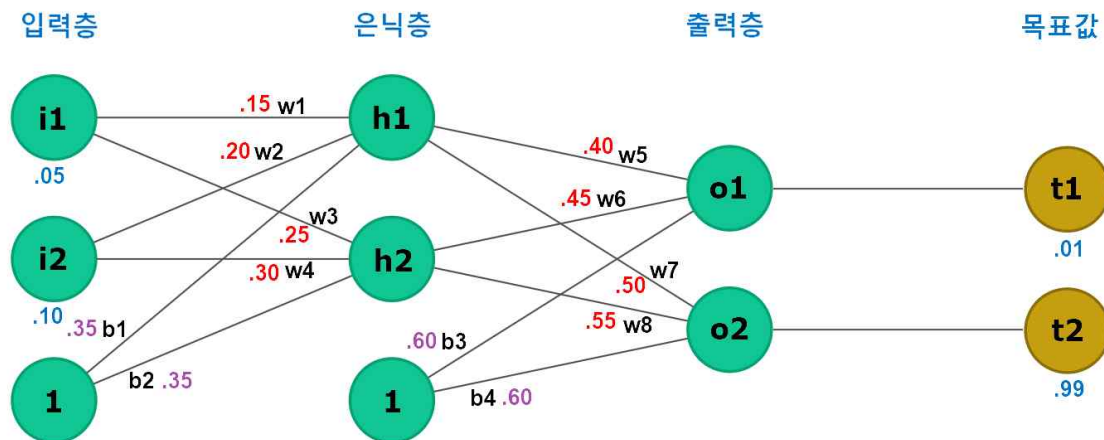
$$y = (x > 0)x \quad x_b = (y > 0)1y_b$$

시그모이드 함수 적용해 보기

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림은 앞에서 구현한 2개의 입력, 2개의 은닉 신경, 2개의 출력 신경으로 구성된 인공 신경망입니다. 여기서는 은닉 신경과 출력 신경에 시그모이드(sigmoid) 활성화 함수를 추가해 봅니다. 이전 예제와 같이 목표값은 각각 0.01, 0.99입니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

```

01 : from math import exp
02 :
03 : i1, i2 = .05, .10
04 : t1, t2 = .01, .99
05 :
06 : w1, w3 = .15, .25
07 : w2, w4 = .20, .30
08 : b1, b2 = .35, .35
09 :
10 : w5, w7 = .40, .50
11 : w6, w8 = .45, .55
12 : b3, b4 = .60, .60
13 :
14 : for epoch in range(1000):
15 :
16 :     print('epoch = %d' %epoch)
17 :
18 :     h1 = i1*w1 + i2*w2 + 1*b1
19 :     h2 = i1*w3 + i2*w4 + 1*b2
20 :     h1 = 1/(1+exp(-h1)) ❶
21 :     h2 = 1/(1+exp(-h2)) ❶
22 :
23 :     o1 = h1*w5 + h2*w6 + 1*b3
24 :     o2 = h1*w7 + h2*w8 + 1*b4
25 :     o1 = 1/(1+exp(-o1)) ❶

```

```

26 : o2 = 1/(1+exp(-o2)) ❶
27 :
28 : # print(' h1, h2 = %6.3f, %6.3f' %(h1, h2))
29 : print(' o1, o2 = %6.3f, %6.3f' %(o1, o2))
30 :
31 : E = (o1-t1)**2/2 + (o2-t2)**2/2
32 : # print(' E = %.7f' %E)
33 : if E < 0.0000001:
34 :     break
35 :
36 : o1b, o2b = o1 - t1, o2 - t2
37 : o1b, o2b = o1b*o1*(1-o1), o2b*o2*(1-o2) ❷
38 :
39 : h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
40 : h1b, h2b = h1b*h1*(1-h1), h2b*h2*(1-h2) ❷
41 :
42 : w1b, w3b = i1*h1b, i1*h2b
43 : w2b, w4b = i2*h1b, i2*h2b
44 : b1b, b2b = 1*h1b, 1*h2b
45 : w5b, w7b = h1*o1b, h1*o2b
46 : w6b, w8b = h2*o1b, h2*o2b
47 : b3b, b4b = 1*o1b, 1*o2b
48 : # print(' w1b, w3b = %6.3f, %6.3f'%(w1b, w3b))
49 : # print(' w2b, w4b = %6.3f, %6.3f'%(w2b, w4b))
50 : # print(' b1b, b2b = %6.3f, %6.3f'%(b1b, b2b))
51 : # print(' w5b, w7b = %6.3f, %6.3f'%(w5b, w7b))
52 : # print(' w6b, w8b = %6.3f, %6.3f'%(w6b, w8b))
53 : # print(' b3b, b4b = %6.3f, %6.3f'%(b3b, b4b))
54 :
55 : lr = 0.01
56 :
57 : w1, w3 = w1 - lr*w1b, w3 - lr*w3b
58 : w2, w4 = w2 - lr*w2b, w4 - lr*w4b
59 : b1, b2 = b1 - lr*b1b, b2 - lr*b2b
60 : w5, w7 = w5 - lr*w5b, w7 - lr*w7b
61 : w6, w8 = w6 - lr*w6b, w8 - lr*w8b
62 : b3, b4 = b3 - lr*b3b, b4 - lr*b4b
63 : # print(' w1, w3 = %6.3f, %6.3f'%(w1, w3))
64 : # print(' w2, w4 = %6.3f, %6.3f'%(w2, w4))
65 : # print(' b1, b2 = %6.3f, %6.3f'%(b1, b2))

```

```
66 : # print(' w5, w7 = %6.3f, %6.3f'%(w5, w7))
67 : # print(' w6, w8 = %6.3f, %6.3f'%(w6, w8))
68 : # print(' b3, b4 = %6.3f, %6.3f'%(b3, b4))
```

01 : math 모듈에서 exp 함수를 불러옵니다.


20, 21 : h1, h2 노드에 순전파 시그모이드 활성화 함수를 적용합니다.

25, 26 : o1, o2 노드에 순전파 시그모이드 활성화 함수를 적용합니다.

37 : o1b, o2b 노드에 역전파 시그모이드 활성화 함수를 적용합니다.

40 : h1b, h2b 노드에 역전파 시그모이드 활성화 함수를 적용합니다.

28, 32, 48~53, 63~68 : 지우거나 주석 처리합니다.


3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 997
o1, o2 = 0.306, 0.844
epoch = 998
o1, o2 = 0.306, 0.844
epoch = 999
o1, o2 = 0.306, 0.844
```

(999+1)번째에 o1, o2가 각각 0.306, 0.844가 됩니다.

4. 다음과 같이 예제를 수정합니다.

```
14 : for epoch in range(10000):
```


5.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 9997
o1, o2 = 0.063, 0.943
epoch = 9998
o1, o2 = 0.063, 0.943
epoch = 9999
o1, o2 = 0.063, 0.943
```

(9999+1)번째에 o1, o2가 각각 0.063, 0.943이 됩니다.

6. 다음과 같이 예제를 수정합니다.

```
14 : for epoch in range(100000):
```


7.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 99997
o1, o2 = 0.020, 0.981
epoch = 99998
o1, o2 = 0.020, 0.981
epoch = 99999
o1, o2 = 0.020, 0.981
```

(99999+1)번째에 o1, o2가 각각 0.020, 0.981이 됩니다.

8. 다음과 같이 예제를 수정합니다.

```
14 : for epoch in range(1000000):
```


9.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 999997
o1, o2 = 0.010, 0.990
epoch = 999998
o1, o2 = 0.010, 0.990
epoch = 999999
o1, o2 = 0.010, 0.990
```

(999999+1)번째에 o1, o2가 각각 0.010, 0.990이 됩니다. 아직 오차는 0.0000001(천만분의 1)보다 큼니다.

10. 다음과 같이 예제를 수정합니다.

```
14 : for epoch in range(1000000):
```

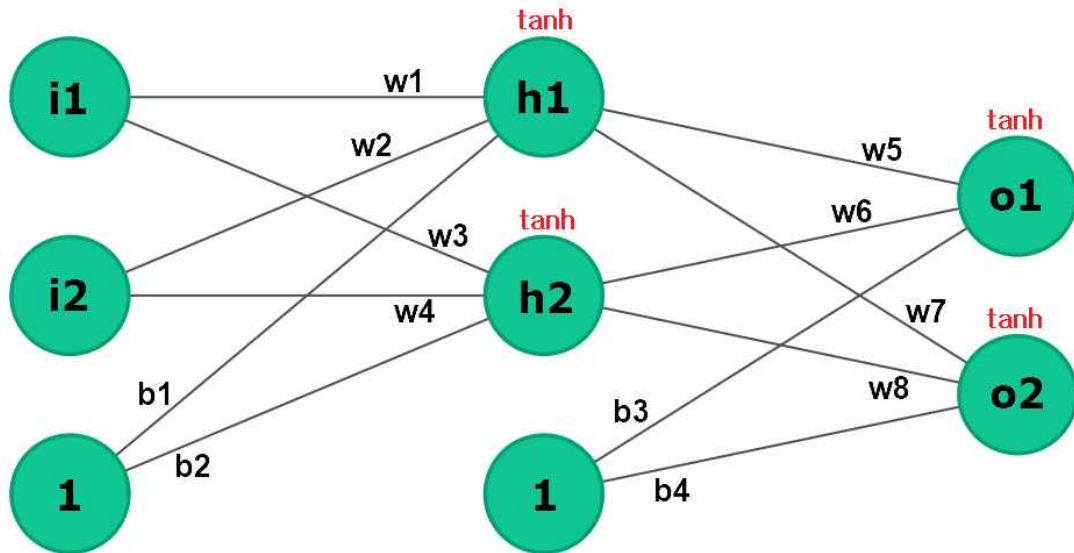
11.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 1078009
o1, o2 = 0.010, 0.990
epoch = 1078010
o1, o2 = 0.010, 0.990
epoch = 1078011
o1, o2 = 0.010, 0.990
```

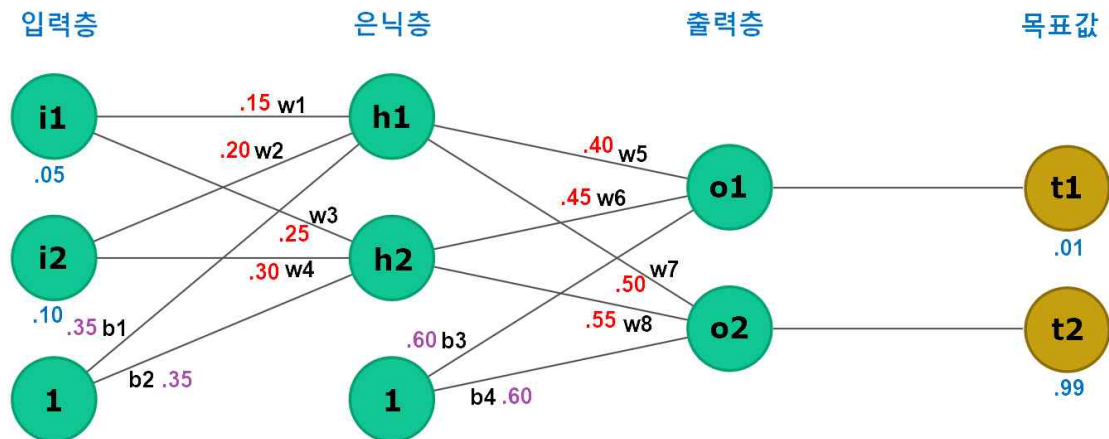
(1078011+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. o1, o2는 각각 0.010, 0.990이 된 상태입니다.

tanh 함수 적용해 보기

이번에는 이전 예제에 적용했던 sigmoid 함수를 tanh 함수로 변경해 봅니다. 다음 그림을 살펴봅니다.



여기서는 은닉 신경과 출력 신경에 tanh 활성화 함수를 적용해 봅니다. 이전 예제와 같이 목표값은 각각 0.01, 0.99입니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

```
01 : from math import exp, tanh
```

01 : math 모듈에서 tanh 함수를 추가로 불러옵니다.

```

18 : h1 = i1*w1 + i2*w2 + 1*b1
19 : h2 = i1*w3 + i2*w4 + 1*b2
20 : h1 = tanh(h1) ③
21 : h2 = tanh(h2) ③
22 :
23 : o1 = h1*w5 + h2*w6 + 1*b3
24 : o2 = h1*w7 + h2*w8 + 1*b4
25 : o1 = tanh(o1) ③
26 : o2 = tanh(o2) ③

```

20, 21 : h1, h2 노드에 순전파 tanh 활성화 함수를 적용합니다.

25, 26 : o1, o2 노드에 순전파 tanh 활성화 함수를 적용합니다.


```

36 : o1b, o2b = o1 - t1, o2 - t2
37 : o1b, o2b = o1b*(1+o1)*(1-o1), o2b*(1+o2)*(1-o2) ④
38 :
39 : h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
40 : h1b, h2b = h1b*(1+h1)*(1-h1), h2b*(1+h2)*(1-h2) ④

```

37 : o1b, o2b 노드에 역전파 tanh 활성화 함수를 적용합니다.

40 : h1b, h2b 노드에 역전파 tanh 활성화 함수를 적용합니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

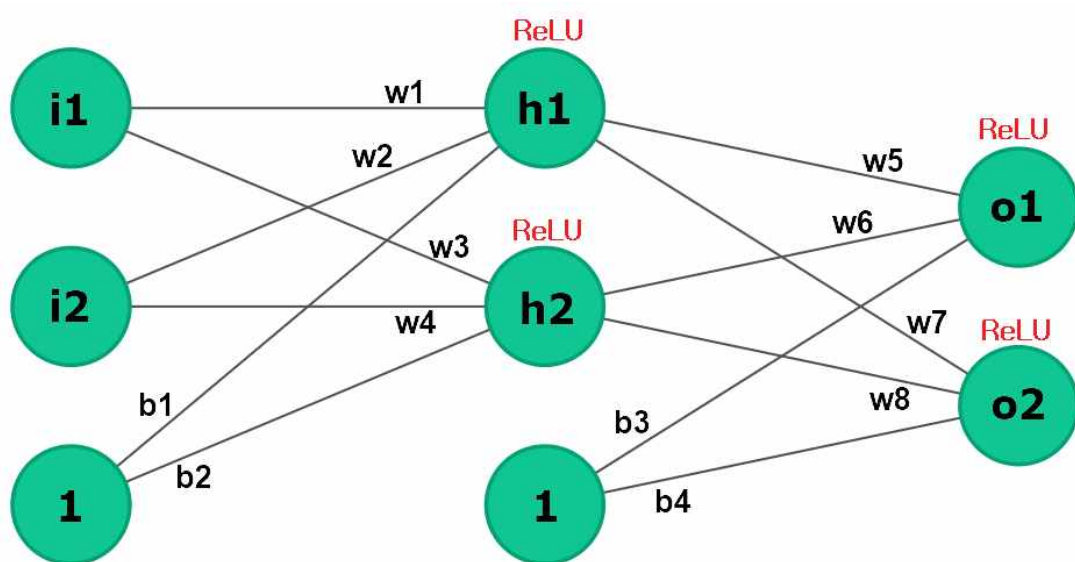
epoch = 236408
o1, o2 = 0.010, 0.990
epoch = 236409
o1, o2 = 0.010, 0.990
epoch = 236410
o1, o2 = 0.010, 0.990

```

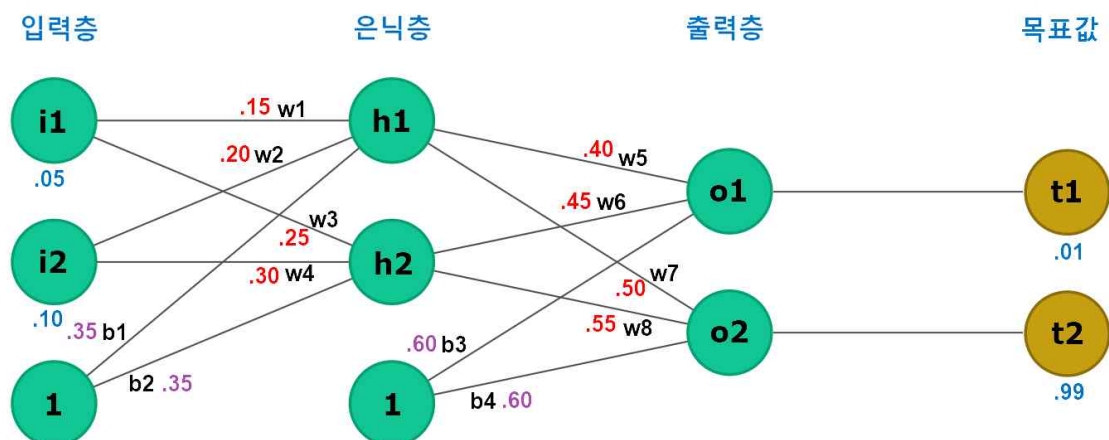
(236410+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. o1, o2는 각각 0.010, 0.990이 된 상태입니다. sigmoid 함수보다 결과가 더 빨리 나오는 것을 볼 수 있습니다.

ReLU 함수 적용해 보기

이번에는 이전 예제에 적용했던 tanh 함수를 ReLU 함수로 변경해 봅니다. 다음 그림을 살펴 봅니다.



여기서는 은닉 신경과 출력 신경에 ReLU 활성화 함수를 적용해 봅니다. 이전 예제와 같이 목표값은 각각 0.01, 0.99입니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

18 :	$h1 = i1*w1 + i2*w2 + 1*b1$
19 :	$h2 = i1*w3 + i2*w4 + 1*b2$
20 :	$h1 = (h1>0)*h1$ 5
21 :	$h2 = (h2>0)*h2$ 5
22 :	
23 :	$o1 = h1*w5 + h2*w6 + 1*b3$
24 :	$o2 = h1*w7 + h2*w8 + 1*b4$


```
25 : o1 = (o1>0)*o1 ⑤
26 : o2 = (o2>0)*o2 ⑤
```


20, 21 : h1, h2 노드에 순전파 ReLU 활성화 함수를 적용합니다.

25, 26 : o1, o2 노드에 순전파 ReLU 활성화 함수를 적용합니다.

```
36 : o1b, o2b = o1 - t1, o2 - t2
37 : o1b, o2b = o1b*(o1>0)*1, o2b*(o2>0)*1 ⑥
38 :
39 : h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
40 : h1b, h2b = h1b*(h1>0)*1, h2b*(h2>0)*1 ⑥
```

37 : o1b, o2b 노드에 역전파 ReLU 활성화 함수를 적용합니다.

40 : h1b, h2b 노드에 역전파 ReLU 활성화 함수를 적용합니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 663
o1, o2 = 0.010, 0.990
epoch = 664
o1, o2 = 0.010, 0.990
epoch = 665
o1, o2 = 0.010, 0.990
```

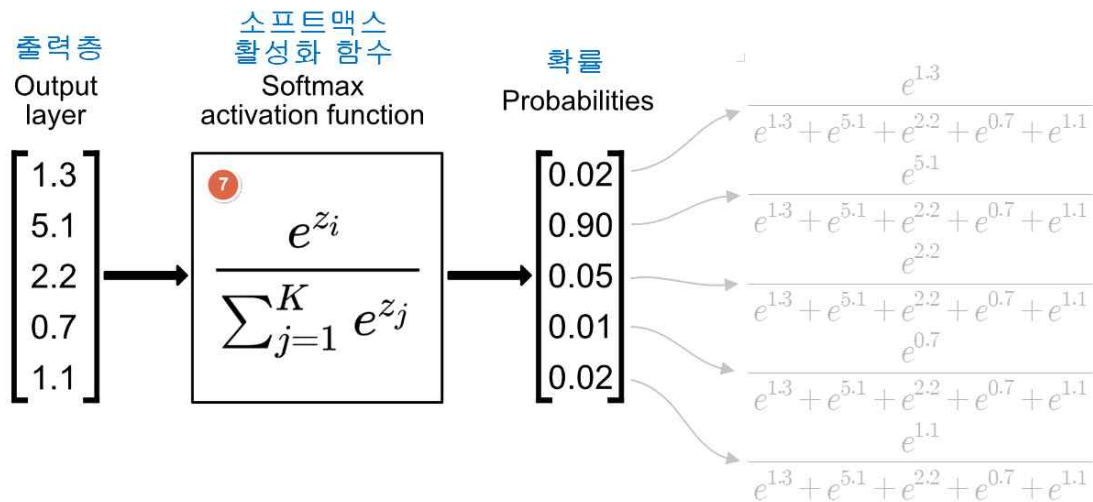
(665+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. o1, o2는 각각 0.010, 0.990이 된 상태입니다. sigmoid, tanh 함수보다 결과가 훨씬 더 빨리 나오는 것을 볼 수 있습니다.

04 출력층에 소프트맥스 함수 적용해 보기

이전 단원에서 우리는 은닉 신경과 출력 신경에 시그모이드(sigmoid), tanh, relu 활성화 함수를 차례대로 적용해 보았습니다. 이 단원에서는 출력 신경의 활성화 함수를 소프트맥스(softmax)로 변경해 봅니다. softmax 활성화 함수는 크로스 엔트로피 오차(cross entropy error) 함수와 같이 사용되며, 분류(classification)에 사용됩니다.

소프트맥스 함수 살펴보기

다음은 출력층에서 활성화 함수로 사용되는 소프트맥스(softmax) 함수를 나타냅니다.



소프트맥스 함수는 출력층에서 사용되는 활성화함수로 다중 분류를 위해 주로 사용됩니다. 소프트맥스 함수는 확률의 총합이 1이 되도록 만든 함수이며 아래에 나타낸 크로스 엔트로피 오차 함수와 같이 사용됩니다.

$$E = - \sum_k t_k \log o_k \quad 8$$

우리는 앞에서 다음과 같은 평균 제곱 오차 함수를 살펴보았습니다.

$$E = \sum_k \frac{1}{2} (o_k - t_k)^2$$

평균 제곱 오차 함수의 경우 역전파 시 전파되는 오차가 다음과 같이 예측값과 목표값의 차인 것을 우리는 이미 앞에서 살펴보았습니다.

$$o_{kb} = o_k - t_k$$

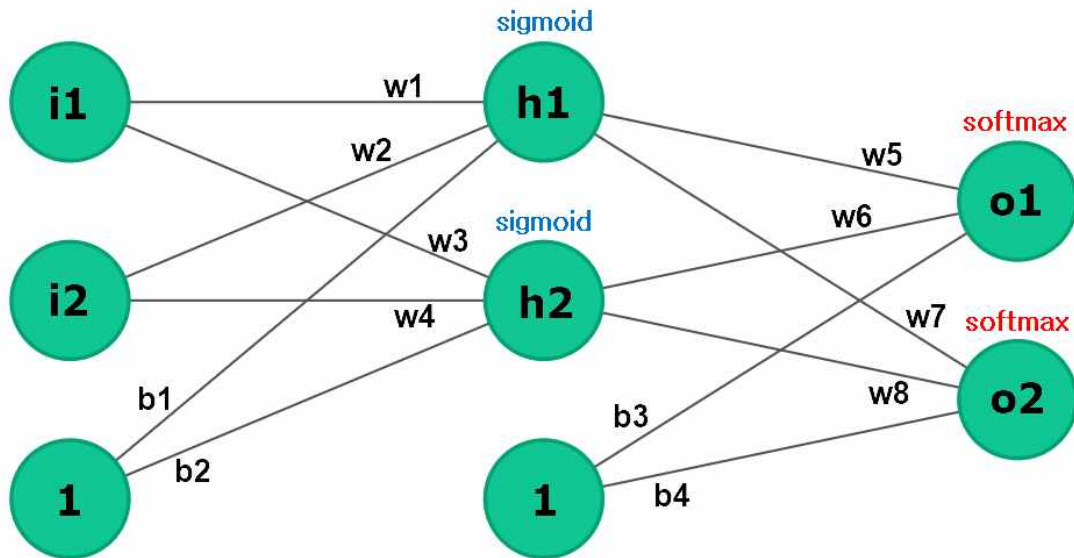
소프트맥스 함수는 크로스 엔트로피 함수와 같이 사용될 때 역전파 시 소프트맥스 함수를 역으로 거쳐 전파되는 오차가 다음과 같이 예측값과 목표값의 차가 됩니다.

$$o_{kb} = o_k - t_k \quad 9$$

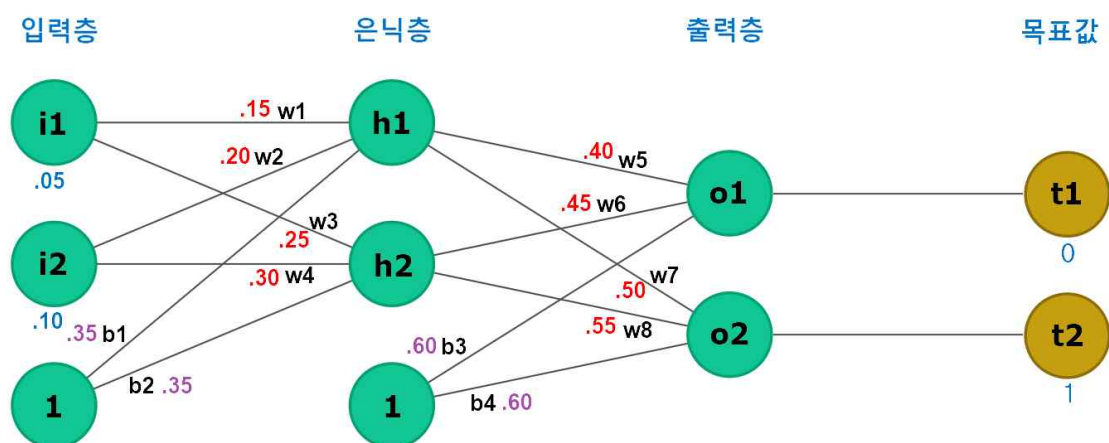
그래서 일반적으로 소프트맥스 함수를 활성화 함수로 사용할 경우 오차 함수는 크로스 엔트로피 오차 함수가 됩니다.

출력층에 소프트맥스 함수 적용해 보기

다음은 우리가 사용할 인공 신경망의 구조입니다.



여기서는 은닉 신경에는 sigmoid, 출력 신경에는 softmax 활성화 함수를 적용합니다. 목표값은 0, 1를 사용합니다. 소프트맥스 활성화 함수와 크로스 엔트로피 오차 함수를 같이 사용할 때 일반적으로 목표값은 0 또는 1의 값만 가지며, 총 합은 1이 됩니다. 다음 그림의 맨 오른쪽에 추가된 2개의 노드는 목표값을 나타내며, 출력층으로 나오는 예측값을 목표값에 가깝도록 가중치를 조정하게 됩니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

```

01 : from math import exp, tanh, log
02 :
03 : i1, i2 = .05, .10
04 : t1, t2 = 0, 1
05 :
06 : w1, w3 = .15, .25
07 : w2, w4 = .20, .30
08 : b1, b2 = .35, .35
09 :
10 : w5, w7 = .40, .50
11 : w6, w8 = .45, .55
12 : b3, b4 = .60, .60
13 :
14 : for epoch in range(10000000):
15 :
16 :     print('epoch = %d' %epoch)
17 :
18 :     h1 = i1*w1 + i2*w2 + 1*b1
19 :     h2 = i1*w3 + i2*w4 + 1*b2
20 :     h1 = 1/(1+exp(-h1)) ❶
21 :     h2 = 1/(1+exp(-h2)) ❶
22 :
23 :     o1 = h1*w5 + h2*w6 + 1*b3
24 :     o2 = h1*w7 + h2*w8 + 1*b4
25 :     o1m = o1 - max(o1, o2) ❷
26 :     o2m = o2 - max(o1, o2) ❷
27 :     o1 = exp(o1m)/(exp(o1m)+exp(o2m)) ❷
28 :     o2 = exp(o2m)/(exp(o1m)+exp(o2m)) ❷
29 :
30 :     # print(' h1, h2 = %6.3f, %6.3f' %(h1, h2))
31 :     print(' o1, o2 = %6.3f, %6.3f' %(o1, o2))
32 :
33 :     E = -t1*log(o1) + -t2*log(o2) ❸
34 :     # print(' E = %.7f' %E)
35 :     if E < 0.0001:
36 :         break
37 :
38 :     o1b, o2b = o1 - t1, o2 - t2 ❹
39 :
40 :

```

```

41 : h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
42 : h1b, h2b = h1b*h1*(1-h1), h2b*h2*(1-h2) ②
43 :
44 : w1b, w3b = i1*h1b, i1*h2b
45 : w2b, w4b = i2*h1b, i2*h2b
46 : b1b, b2b = 1*h1b, 1*h2b
47 : w5b, w7b = h1*o1b, h1*o2b
48 : w6b, w8b = h2*o1b, h2*o2b
49 : b3b, b4b = 1*o1b, 1*o2b
50 : # print(' w1b, w3b = %6.3f, %6.3f'%(w1b, w3b))
51 : # print(' w2b, w4b = %6.3f, %6.3f'%(w2b, w4b))
52 : # print(' b1b, b2b = %6.3f, %6.3f'%(b1b, b2b))
53 : # print(' w5b, w7b = %6.3f, %6.3f'%(w5b, w7b))
54 : # print(' w6b, w8b = %6.3f, %6.3f'%(w6b, w8b))
55 : # print(' b3b, b4b = %6.3f, %6.3f'%(b3b, b4b))
56 :
57 : lr = 0.01
58 :
59 : w1, w3 = w1 - lr*w1b, w3 - lr*w3b
60 : w2, w4 = w2 - lr*w2b, w4 - lr*w4b
61 : b1, b2 = b1 - lr*b1b, b2 - lr*b2b
62 : w5, w7 = w5 - lr*w5b, w7 - lr*w7b
63 : w6, w8 = w6 - lr*w6b, w8 - lr*w8b
64 : b3, b4 = b3 - lr*b3b, b4 - lr*b4b
65 : # print(' w1, w3 = %6.3f, %6.3f'%(w1, w3))
66 : # print(' w2, w4 = %6.3f, %6.3f'%(w2, w4))
67 : # print(' b1, b2 = %6.3f, %6.3f'%(b1, b2))
68 : # print(' w5, w7 = %6.3f, %6.3f'%(w5, w7))
69 : # print(' w6, w8 = %6.3f, %6.3f'%(w6, w8))
70 : # print(' b3, b4 = %6.3f, %6.3f'%(b3, b4))

```

01 : math 라이브러리에서 log 함수를 추가로 불러옵니다. log 함수는 자연로그 함수입니다.

04 : 목표값을 각각 0과 1로 변경합니다.

25~28 : 출력층의 활성화 함수를 소프트맥스로 변경합니다.

25, 26 : o1, o2에 대해 둘 중 더 큰 값을 빼줍니다. 이렇게 하면 26, 27 줄에서 오버플로우를 막을 수 있습니다. o1, o2에 대한 최종 결과는 같습니다. 자세한 내용은 [소프트맥스 오버플로우]를 검색해 봅니다.

33 : 오차 계산을 크로스 엔트로피 오차 형태의 수식으로 변경합니다. 소프트맥스 활성화 함수는 크로스 엔트로피 오차와 같이 사용합니다.


$$E = - \sum_k t_k \log o_k$$

35 : for 문을 빠져 나가는 오차값을 0.0001로 변경합니다. 여기서 사용하는 값의 크기에 따라 학습의 정확도와 학습 시간이 결정됩니다.

38 : 소프트맥스 함수의 역전파 오차 계산 부분은 다음과 같습니다. 소프트맥스 함수는 크로스 엔트로피 함수와 같이 사용될 때 역전파 시 소프트맥스 함수를 역으로 거쳐 전파되는 오차가 다음과 같이 예측값과 목표값의 차가 됩니다.

$$o_{kb} = o_k - t_k$$

그래서 일반적으로 소프트맥스 함수를 활성화 함수로 사용할 경우 오차 함수는 크로스 엔트로피 오차 함수가 됩니다.

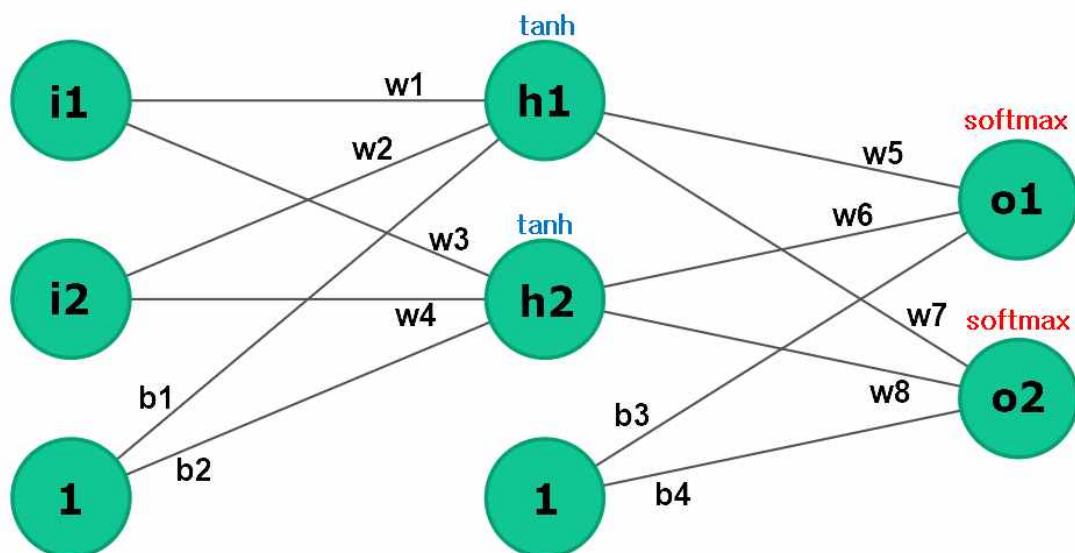
3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 211286
o1, o2 = 0.000, 1.000
epoch = 211287
o1, o2 = 0.000, 1.000
epoch = 211288
o1, o2 = 0.000, 1.000
```

(211288+1)번째에 오차가 0.0001(만분의 1)보다 작아집니다. o1, o2는 각각 0.000, 1.000이 된 상태입니다.

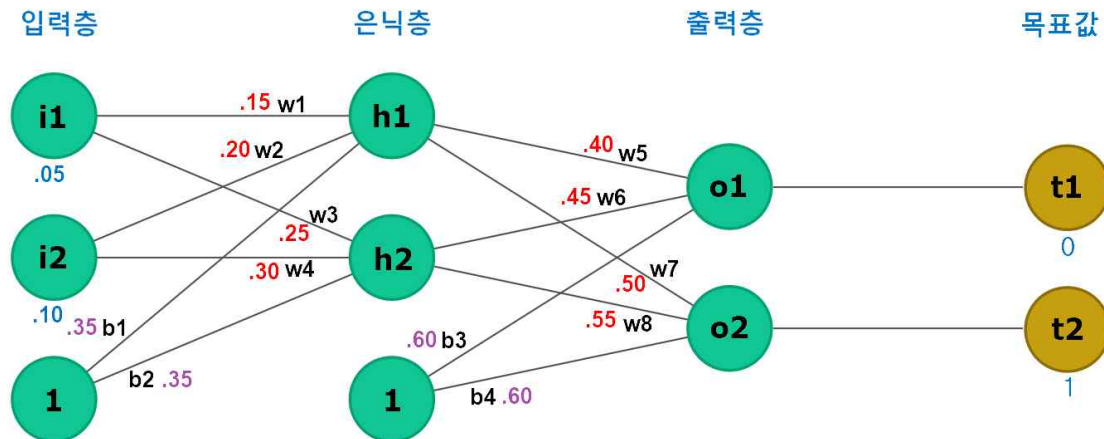
tanh와 소프트맥스

여기서는 은닉층 활성화 함수를 tanh로 변경해 봅니다. 다음 그림을 살펴봅시다.



여기서는 은닉 신경에는 tanh, 출력 신경에는 softmax 활성화 함수를 적용합니다. 목표값은

0, 1를 사용합니다. 소프트맥스 활성화 함수와 크로스 엔트로피 오차 함수를 같이 사용할 때 일반적으로 목표값은 0 또는 1의 값만 가지며, 총 합은 1이 됩니다. 다음 그림의 맨 오른쪽에 추가된 2개의 노드는 목표값을 나타내며, 출력층으로 나오는 예측값을 목표값에 가깝도록 가중치를 조정하게 됩니다.



1. 이전 예제를 복사합니다.

2. 다음과 같이 예제를 수정합니다.

```

18 : h1 = i1*w1 + i2*w2 + 1*b1
19 : h2 = i1*w3 + i2*w4 + 1*b2
20 : h1 = tanh(h1) ③
21 : h2 = tanh(h2) ③
22 :
23 : o1 = h1*w5 + h2*w6 + 1*b3
24 : o2 = h1*w7 + h2*w8 + 1*b4
25 : o1m = o1 - max(o1, o2) ⑦
26 : o2m = o2 - max(o1, o2) ⑦
27 : o1 = exp(o1m)/(exp(o1m)+exp(o2m)) ⑦
28 : o2 = exp(o2m)/(exp(o1m)+exp(o2m)) ⑦

```

20, 21 : h1, h2 노드에 순전파 tanh 활성화 함수를 적용합니다.

25~28 : 출력층의 활성화 함수는 softmax입니다.

```

38 : o1b, o2b = o1 - t1, o2 - t2 ⑨
39 :
40 :
41 : h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
42 : h1b, h2b = h1b*(1+h1)*(1-h1), h2b*(1+h2)*(1-h2) ④

```

38 : softmax 함수의 역전파 오차 계산 부분입니다.

42 : h1b, h2b 노드에 역전파 tanh 활성화 함수를 적용합니다.

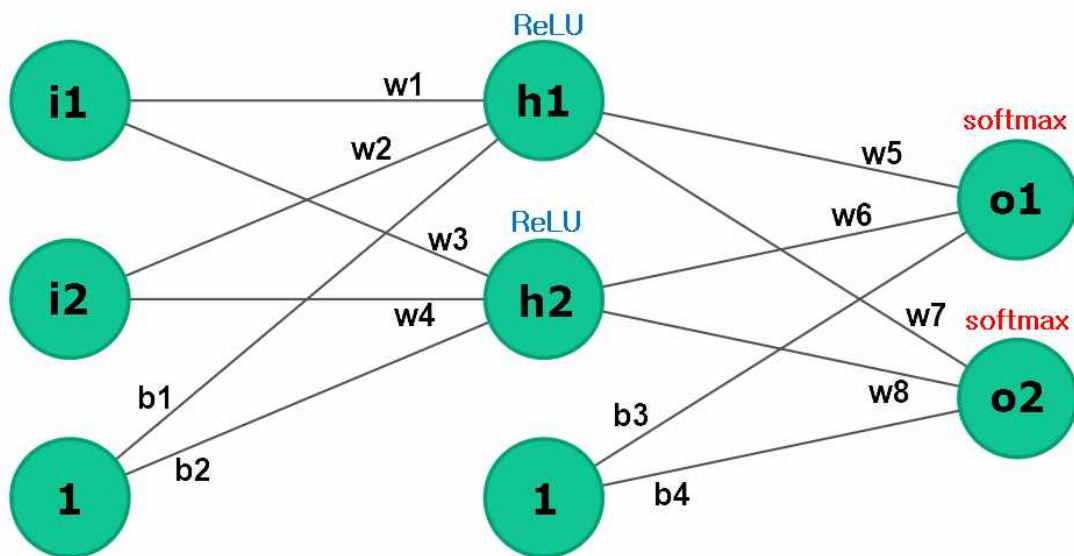
3. ▶ 버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 174991
o1, o2 = 0.000, 1.000
epoch = 174992
o1, o2 = 0.000, 1.000
epoch = 174993
o1, o2 = 0.000, 1.000
```

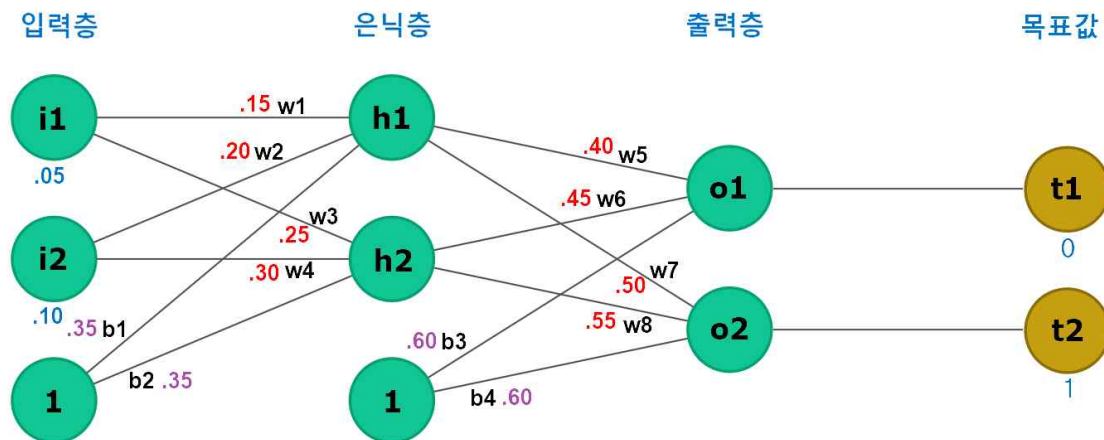
(174993+1)번째에 오차가 0.0001(만분의 1)보다 작아집니다. o1, o2는 각각 0.000, 1.000이 된 상태입니다.

ReLU와 소프트맥스

여기서는 은닉층 활성화 함수를 ReLU로 변경해 봅니다. 다음 그림을 살펴봅니다.



여기서는 은닉 신경에는 ReLU, 출력 신경에는 softmax 활성화 함수를 적용합니다. 목표값은 0, 1를 사용합니다. 소프트맥스 활성화 함수와 크로스 엔트로피 오차 함수를 같이 사용할 때 일반적으로 목표값은 0 또는 1의 값만 가지며, 총 합은 1이 됩니다. 다음 그림의 맨 오른쪽에 추가된 2개의 노드는 목표값을 나타내며, 출력층으로 나오는 예측값을 목표값에 가깝도록 가중치를 조정하게 됩니다.



1. 이전 예제를 복사합니다.

2. 다음과 같이 예제를 수정합니다.

```

18 : h1 = i1*w1 + i2*w2 + 1*b1
19 : h2 = i1*w3 + i2*w4 + 1*b2
20 : h1 = (h1>0)*h1 ⑤
21 : h2 = (h2>0)*h2 ⑤
22 :
23 : o1 = h1*w5 + h2*w6 + 1*b3
24 : o2 = h1*w7 + h2*w8 + 1*b4
25 : o1m = o1 - max(o1, o2) ⑦
26 : o2m = o2 - max(o1, o2) ⑦
27 : o1 = exp(o1m)/(exp(o1m)+exp(o2m)) ⑦
28 : o2 = exp(o2m)/(exp(o1m)+exp(o2m)) ⑦

```

20, 21 : h1, h2 노드에 순전파 ReLU 활성화 함수를 적용합니다.

25~28 : 출력층의 활성화 함수는 softmax입니다.


```

38 : o1b, o2b = o1 - t1, o2 - t2 ⑨
39 :
40 :
41 : h1b, h2b = o1b*w5+o2b*w7, o1b*w6+o2b*w8
42 : h1b, h2b = h1b*(h1>0)*1, h2b*(h2>0)*1 ⑥

```

38 : softmax 함수의 역전파 오차 계산 부분입니다.

42 : h1b, h2b 노드에 역전파 ReLU 활성화 함수를 적용합니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

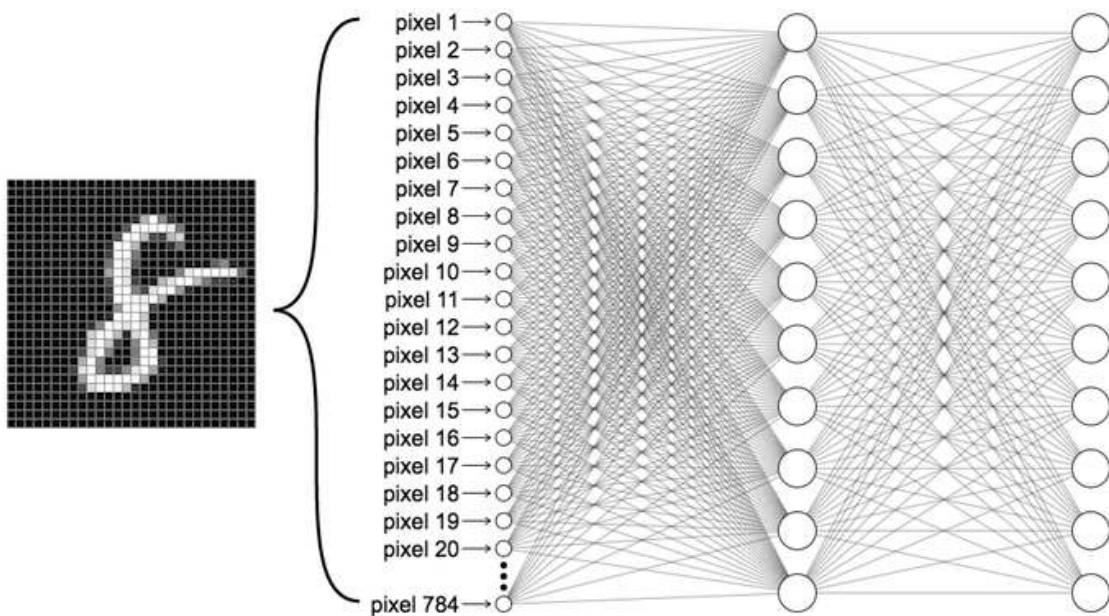
```
epoch = 56959
o1, o2 = 0.000, 1.000
epoch = 56960
o1, o2 = 0.000, 1.000
epoch = 56961
o1, o2 = 0.000, 1.000
```

(56961+1)번째에 오차가 0.0001(만분의 1)보다 작아집니다. o1, o2는 각각 0.000, 1.000이 된 상태입니다.

이상에서 출력층의 활성화 함수는 소프트맥스, 오차 계산 함수는 크로스 엔트로피 오차 함수인 인공 신경망을 구현해 보았습니다.

04 NumPy 인공 신경망 구현하기

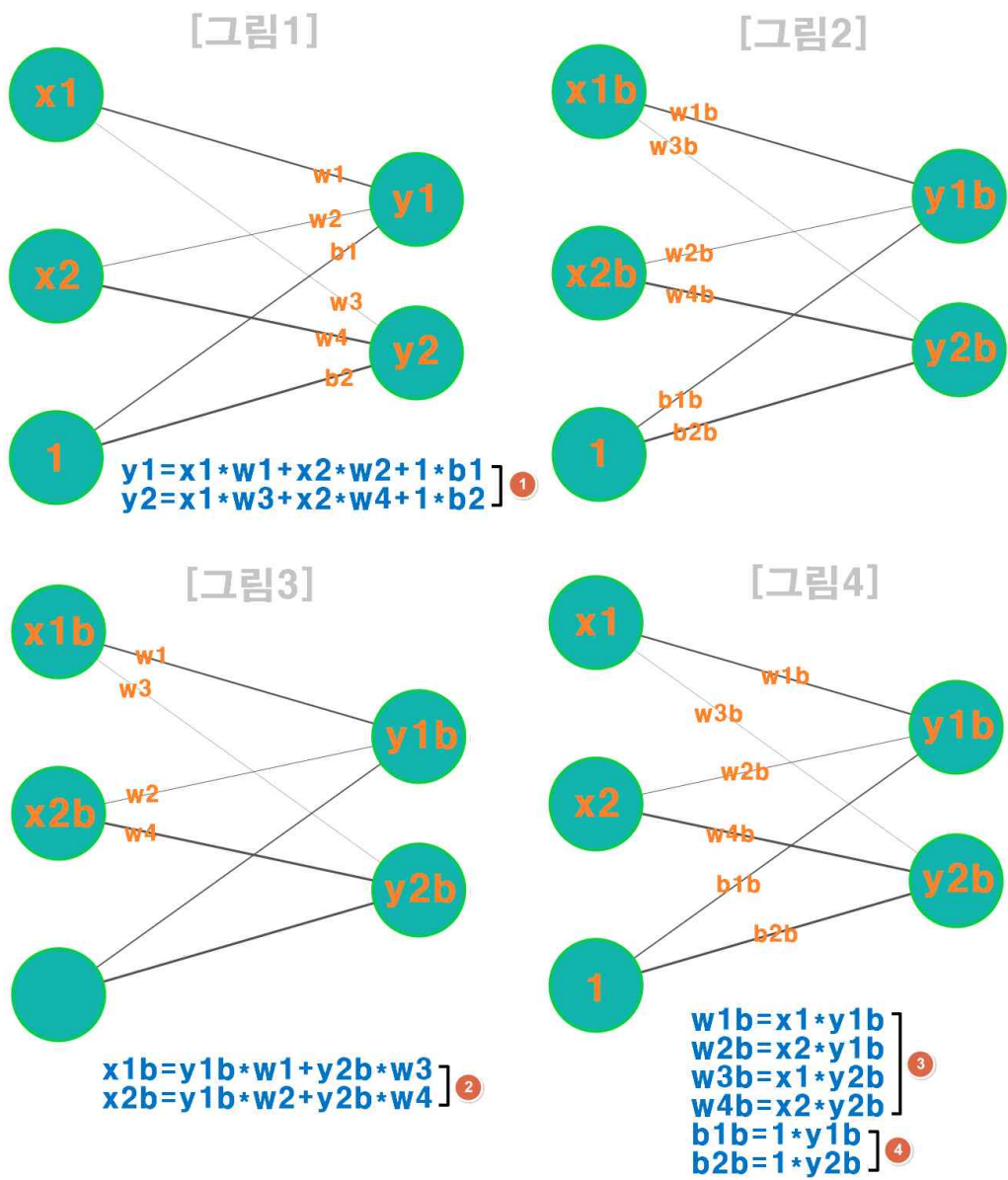
여기서는 인공 신경망을 확장할 수 있도록 NumPy 라이브러리를 활용하여 인공 신경망을 구현해 봅니다. NumPy 라이브러리를 이용하면, 커다란 인공 신경망을 자유롭게 구성하고 테스트해 볼 수 있습니다. 예를 들어, 1장에서 tensorflow 라이브러리를 이용하여 살펴 보았던 다음과 같은 형태의 인공 신경망을 구성해서 테스트해 볼 수 있습니다.



<784개의 입력, 64개의 은닉층, 10개의 출력층>

02 입력2 출력2 인공 신경 구현하기

다음 그림은 입력2 출력2로 구성된 인공 신경과 순전파 역전파 수식을 나타냅니다. 우리는 다음 수식을 행렬식으로 유도한 후, NumPy를 이용하여 인공 신경을 구현해 봅니다.



*** ❷ x1b, x2b 노드가 입력층일 경우엔 이 수식은 필요하지 않습니다.

이 그림을 통해 앞에서 우리는 다음과 같은 수식을 유도했습니다. 이런 형태의 수식을 다원일 차연립방정식이라고 합니다.

순전파

$$\begin{bmatrix} x_1 w_1 + x_2 w_2 + 1b_1 = y_1 \\ x_1 w_3 + x_2 w_4 + 1b_2 = y_2 \end{bmatrix} \quad 1$$

입력 역전파

$$\begin{bmatrix} y_{1b} w_1 + y_{2b} w_3 = x_{1b} \\ y_{1b} w_2 + y_{2b} w_4 = x_{2b} \end{bmatrix} \quad 2$$

가중치, 편향 역전파

$$\begin{bmatrix} x_1 y_{1b} = w_{1b} \\ x_2 y_{1b} = w_{2b} \\ x_1 y_{2b} = w_{3b} \\ x_2 y_{2b} = w_{4b} \end{bmatrix} \quad 3$$

$$\begin{bmatrix} 1 y_{1b} = b_{1b} \\ 1 y_{2b} = b_{2b} \end{bmatrix} \quad 4$$

다원일차연립방정식은 행렬을 이용하면 깔끔하게 정리할 수 있습니다. 위 수식을 행렬식으로 정리하면 다음과 같습니다.

순전파

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix} = \begin{bmatrix} y_1 & y_2 \end{bmatrix} \quad 1$$

입력 역전파

$$\begin{bmatrix} y_{1b} & y_{2b} \end{bmatrix} \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} =$$

$$\begin{bmatrix} y_{1b} & y_{2b} \end{bmatrix} \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix}^T = \begin{bmatrix} x_{1b} & x_{2b} \end{bmatrix} \quad 2$$

가중치, 편향 역전파

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} y_{1b} & y_{2b} \end{bmatrix} =$$

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix}^T \begin{bmatrix} y_{1b} & y_{2b} \end{bmatrix} = \begin{bmatrix} w_{1b} & w_{3b} \\ w_{2b} & w_{4b} \end{bmatrix} \quad 3$$

$$1 \begin{bmatrix} y_{1b} & y_{2b} \end{bmatrix} = \begin{bmatrix} b_{1b} & b_{2b} \end{bmatrix} \quad 4$$

❷에서 다음은 순전파 때 사용된 가중치의 전치 행렬입니다.

$$\begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix}^T$$

전치행렬은 가로줄과 세로줄이 바뀐 행렬입니다.

❸에서 다음은 순전파 때 사용된 입력의 전치 행렬입니다.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$$

위 수식에서 표현된 행렬들에 다음과 같이 이름을 붙여줍니다.

$$\begin{aligned} \begin{bmatrix} x_1 & x_2 \end{bmatrix} &= X \\ \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} &= W \\ \begin{bmatrix} b_1 & b_2 \end{bmatrix} &= B \\ \begin{bmatrix} y_1 & y_2 \end{bmatrix} &= Y \\ \begin{bmatrix} y_{1b} & y_{2b} \end{bmatrix} &= Y_b \\ \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} &= \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix}^T = W^T \\ \begin{bmatrix} x_{1b} & x_{2b} \end{bmatrix} &= X_b \\ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T = X^T \\ \begin{bmatrix} w_{1b} & w_{3b} \\ w_{2b} & w_{4b} \end{bmatrix} &= W_b \\ \begin{bmatrix} b_{1b} & b_{2b} \end{bmatrix} &= B_b \end{aligned}$$

그러면 앞의 행렬식은 다음과 같이 정리할 수 있습니다. 이 수식은 행렬의 크기와 상관없이 적용됩니다. 주의할 점은 행렬곱은 순서를 변경하면 안됩니다.

순전파

$$Y = XW + B \quad \text{❶}$$

입력 역전파

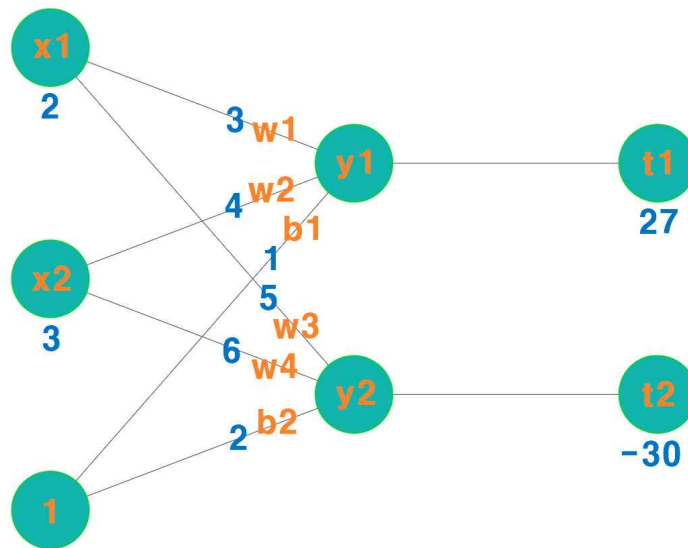
$$Y_b W^T = X_b \quad \textcircled{2}$$

가중치, 편향 역전파

$$X^T Y_b = W_b \quad \textcircled{3}$$

$$1 Y_b = B_b \quad \textcircled{4}$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 X, 가중치 W, 편향 B, 목표값 T는 다음과 같습니다.

$$\begin{aligned} [x_1 \ x_2] &= [2 \ 3] = X \\ \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} &= \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} = W \\ [b_1 \ b_2] &= [1 \ 2] = B \\ [t_1 \ t_2] &= [27 \ -30] = T \end{aligned}$$

X를 상수로 고정한 채 W, B에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 다음과 같이 예제를 작성합니다.

```


01 : import numpy as np
02 :
03 : np.set_printoptions(formatter={'float_kind':lambda x: "{0:6.3f}".format(x)})
04 :
05 : X = np.array([[2, 3]])
06 : T = np.array([[27, -30]])
07 : W = np.array([[3, 5],
08 :               [4, 6]])
09 : B = np.array([[1, 2]])
10 :
11 : for epoch in range(1000):
12 :
13 :     print('epoch = %d' %epoch)
14 :
15 :     Y = X @ W + B ❶
16 :     print(' Y =', Y)
17 :
18 :     E = np.sum((Y - T) ** 2 / 2)
19 :     print(' E = %.7f' %E)
20 :     if E < 0.0000001:
21 :         break
22 :
23 :     Yb = Y - T
24 :     Xb = Yb @ W.T ❷
25 :     Wb = X.T @ Yb ❸
26 :     Bb = 1 * Yb ❹
27 :     print(' Xb =\n', Xb)
28 :     print(' Wb =\n', Wb)
29 :     print(' Bb =\n', Bb)
30 :
31 :     lr = 0.01
32 :
33 :     W = W - lr * Wb
34 :     B = B - lr * Bb
35 :     print(' W =\n', W)
36 :     print(' B =\n', B)

```

01 : import문을 이용하여 numpy 모듈을 np라는 이름으로 불러옵니다. numpy 모듈은 행렬 계산을 편하게 해주는 라이브러리입니다. 인공 신경망은 일반적으로 행렬식으로 구성하게 됩니다.

03 : np.set_printoptions 함수를 호출하여 numpy의 실수 출력 방법을 변경합니다. 이 예제에서는 소수점 이하 3자리까지 출력합니다.

05 : np.array 함수를 호출하여 1x2 행렬을 생성하여 X 변수에 할당합니다.
 06 : np.array 함수를 호출하여 1x2 행렬을 생성하여 T 변수에 할당합니다.
 07, 08 : np.array 함수를 호출하여 2x2 행렬을 생성하여 W 변수에 할당합니다.
 09 : np.array 함수를 호출하여 1x2 행렬을 생성하여 B 변수에 할당합니다.
 11 : epoch값을 0에서 1000 미만까지 바꾸어가며 13~36줄을 1000회 수행합니다.
 13 : print 함수를 호출하여 Y값을 출력합니다.
 15 : 행렬곱 연산자 @을 이용하여 입력 X와 가중치 W에 대해 행렬곱을 수행한 후, 편향 B를 더해줍니다. 행렬곱의 순서를 변경하지 않도록 주의합니다.
 18 : 평균 제곱 오차를 구합니다.
 19 : print 함수를 호출하여 E값을 출력합니다. 소수점 이하 7자리까지 출력합니다.
 20, 21 : 평균 제곱 오차가 0.0000001(천만분의 1)보다 작으면 break문을 사용하여 11줄의 for 문을 빠져 나갑니다.
 23 : 예측값을 가진 Y 행렬에서 목표값을 가진 T 행렬을 뺀 후, 결과값을 Yb 변수에 할당합니다. Yb는 역전파 오차값을 갖는 행렬입니다.
 24 : W.T는 가중치 W의 전치 행렬을 내어줍니다. 행렬곱 연산자 @을 이용하여 Yb와 W.T에 대해 행렬곱을 수행한 후, 결과값을 Xb 변수에 할당합니다. 행렬곱의 순서를 변경하지 않도록 주의합니다.
 25 : X.T는 입력 X의 전치 행렬을 내어줍니다. 행렬곱 연산자 @을 이용하여 X.T와 Yb에 대해 행렬곱을 수행한 후, 결과값을 Wb 변수에 할당합니다. 행렬곱의 순서를 변경하지 않도록 주의합니다.
 26 : Yb 행렬에 1을 곱해주어 Bb에 할당합니다. 여기서 1은 수식을 강조하기 위해 생략하지 않았습니니다.
 27~29 : print 함수를 호출하여 Xb, Wb, Bb값을 출력합니다.
 31 : lr 변수를 선언한 후, 0.01을 할당합니다. lr 변수는 학습률 변수입니다.
 33 : 가중치를 갱신합니다.
 34 : 편향을 갱신합니다.
 35, 36 : print 함수를 호출하여 W, B값을 출력합니다.

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.


```

epoch = 78
Y = [[27.000 -30.000]]
E = 0.0000001
Xb =
[[-0.002 -0.004]]
Wb =
[[-0.000  0.001]
 [-0.000  0.001]]
Bb =
[[-0.000  0.000]]
W =
[[ 4.143 -3.571]
 [ 5.714 -6.857]]
B =
[[ 1.571 -2.286]]
epoch = 79
Y = [[27.000 -30.000]]
E = 0.0000001

```

79회 째 학습이 완료되는 것을 볼 수 있습니다.

앞의 예제의 결과와 비교해 봅니다.

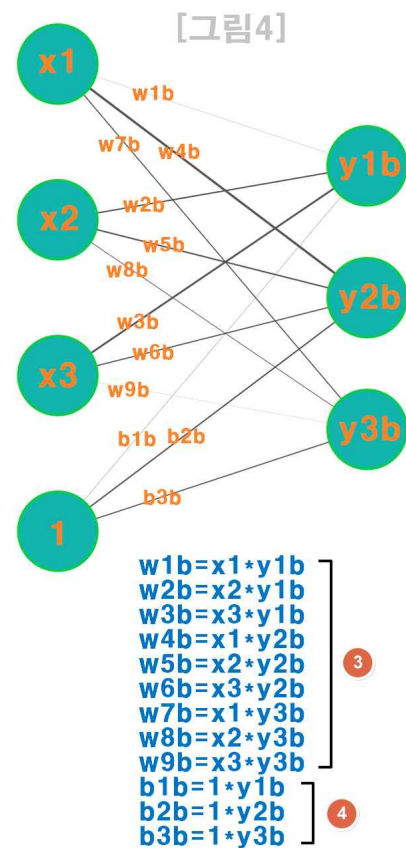
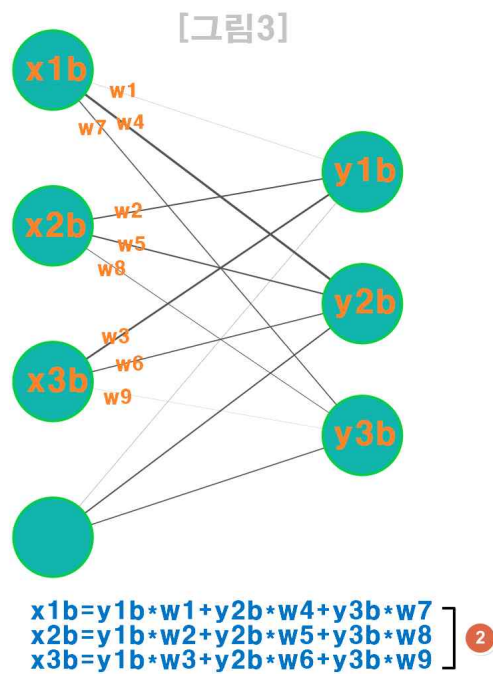
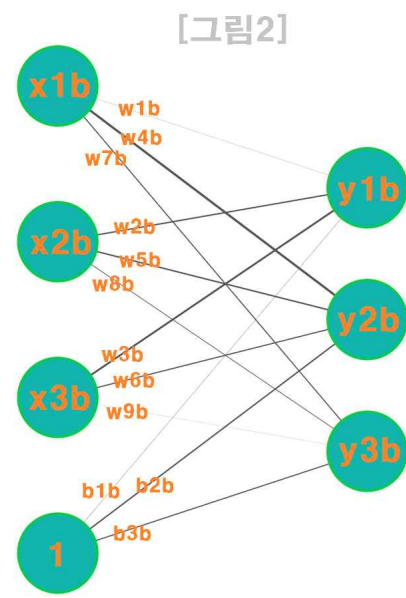
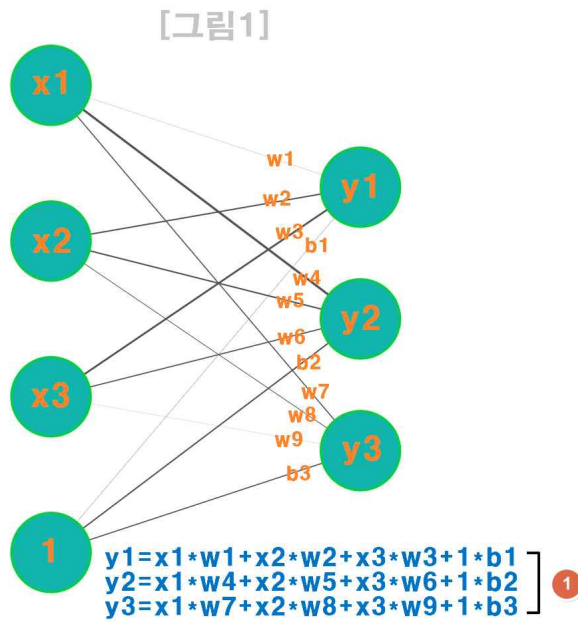
```

epoch = 78
y1, y2 = 27.000, -30.000
E = 0.0000001
x1b, x2b = -0.002, -0.004
w1b, w3b = -0.000,  0.001
w2b, w4b = -0.000,  0.001
b1b, b2b = -0.000,  0.000
w1, w3 =  4.143, -3.571
w2, w4 =  5.714, -6.857
b1, b2 =  1.571, -2.286
epoch = 79
y1, y2 = 27.000, -30.000
E = 0.0000001

```

03 입력3 출력3 인공 신경 구현하기

다음 그림은 입력3 출력3으로 구성된 인공 신경과 순전파 역전파 수식을 나타냅니다. 우리는 다음 수식을 행렬식으로 유도한 후, NumPy를 이용하여 인공 신경을 구현해 봅니다.



*** ② x1b, x2b, x3b 노드가 입력층일 경우엔 이 수식은 필요하지 않습니다.

이 그림을 통해 앞에서 우리는 다음과 같은 수식을 유도했습니다. 이런 형태의 수식을 다원일 차연립방정식이라고 합니다.

순전파

$$\left. \begin{aligned} x_1 w_1 + x_2 w_2 + x_3 w_3 + 1 b_1 &= y_1 \\ x_1 w_4 + x_2 w_5 + x_3 w_6 + 1 b_2 &= y_2 \\ x_1 w_7 + x_2 w_8 + x_3 w_9 + 1 b_3 &= y_3 \end{aligned} \right\} \textcircled{1}$$

입력 역전파

$$\left. \begin{aligned} y_{1b} w_1 + y_{2b} w_4 + y_{3b} w_7 &= x_{1b} \\ y_{1b} w_2 + y_{2b} w_5 + y_{3b} w_8 &= x_{2b} \\ y_{1b} w_3 + y_{2b} w_6 + y_{3b} w_9 &= x_{3b} \end{aligned} \right\} \textcircled{2}$$

가중치, 편향 역전파

$$\left. \begin{aligned} x_1 y_{1b} &= w_{1b} \\ x_2 y_{1b} &= w_{2b} \\ x_3 y_{1b} &= w_{3b} \\ x_1 y_{2b} &= w_{4b} \\ x_2 y_{2b} &= w_{5b} \\ x_3 y_{2b} &= w_{6b} \\ x_1 y_{3b} &= w_{7b} \\ x_2 y_{3b} &= w_{8b} \\ x_3 y_{3b} &= w_{9b} \end{aligned} \right\} \textcircled{3}$$
$$\left. \begin{aligned} 1 y_{1b} &= b_{1b} \\ 1 y_{2b} &= b_{2b} \\ 1 y_{3b} &= b_{3b} \end{aligned} \right\} \textcircled{4}$$

다원일차연립방적식은 행렬을 이용하면 깔끔하게 정리할 수 있습니다. 위 수식을 행렬식으로 정리하면 다음과 같습니다.

순전파

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_1 & w_4 & w_7 \\ w_2 & w_5 & w_8 \\ w_3 & w_6 & w_9 \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \textcircled{1}$$

입력 역전파

$$\begin{aligned}
 & \begin{bmatrix} y_{1b} & y_{2b} & y_{3b} \end{bmatrix} \begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} = \\
 & \begin{bmatrix} y_{1b} & y_{2b} & y_{3b} \end{bmatrix} \begin{bmatrix} w_1 & w_4 & w_7 \\ w_2 & w_5 & w_8 \\ w_3 & w_6 & w_9 \end{bmatrix}^T = \begin{bmatrix} x_{1b} & x_{2b} & x_{3b} \end{bmatrix} \quad \textcircled{2}
 \end{aligned}$$

가중치, 편향 역전파

$$\begin{aligned}
 & \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} y_{1b} & y_{2b} & y_{3b} \end{bmatrix} = \\
 & \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T \begin{bmatrix} y_{1b} & y_{2b} & y_{3b} \end{bmatrix} = \begin{bmatrix} w_{1b} & w_{4b} & w_{7b} \\ w_{2b} & w_{5b} & w_{8b} \\ w_{3b} & w_{6b} & w_{9b} \end{bmatrix} \quad \textcircled{3}
 \end{aligned}$$

$$1 \begin{bmatrix} y_{1b} & y_{2b} & y_{3b} \end{bmatrix} = \begin{bmatrix} b_{1b} & b_{2b} & b_{3b} \end{bmatrix} \quad \textcircled{4}$$

②에서 다음은 순전파 때 사용된 가중치의 전치 행렬입니다.

$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} = \begin{bmatrix} w_1 & w_4 & w_7 \\ w_2 & w_5 & w_8 \\ w_3 & w_6 & w_9 \end{bmatrix}^T$$

전치행렬은 가로줄과 세로줄이 바뀐 행렬입니다.

③에서 다음은 순전파 때 사용된 입력의 전치 행렬입니다.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$$

위 수식에서 표현된 행렬들에 다음과 같이 이름을 붙여줍니다.

$$\begin{aligned}
\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} &= X \\
\begin{bmatrix} w_1 & w_4 & w_7 \\ w_2 & w_5 & w_8 \\ w_3 & w_6 & w_9 \end{bmatrix} &= W \\
\begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} &= B \\
\begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} &= Y \\
\begin{bmatrix} y_{1b} & y_{2b} & y_{3b} \end{bmatrix} &= Y_b \\
\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} &= \begin{bmatrix} w_1 & w_4 & w_7 \\ w_2 & w_5 & w_8 \\ w_3 & w_6 & w_9 \end{bmatrix}^T = W^T \\
\begin{bmatrix} x_{1b} & x_{2b} & x_{3b} \end{bmatrix} &= X_b \\
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} &= \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T = X^T \\
\begin{bmatrix} w_{1b} & w_{4b} & w_{7b} \\ w_{2b} & w_{5b} & w_{8b} \\ w_{3b} & w_{6b} & w_{9b} \end{bmatrix} &= W_b \\
\begin{bmatrix} b_{1b} & b_{2b} & b_{3b} \end{bmatrix} &= B_b
\end{aligned}$$

그러면 앞의 행렬식은 다음과 같이 정리할 수 있습니다. 이 수식은 행렬의 크기와 상관없이 적용됩니다. 주의할 점은 행렬곱은 순서를 변경하면 안됩니다.

순전파

$$Y = XW + B \quad \textcircled{1}$$

입력 역전파

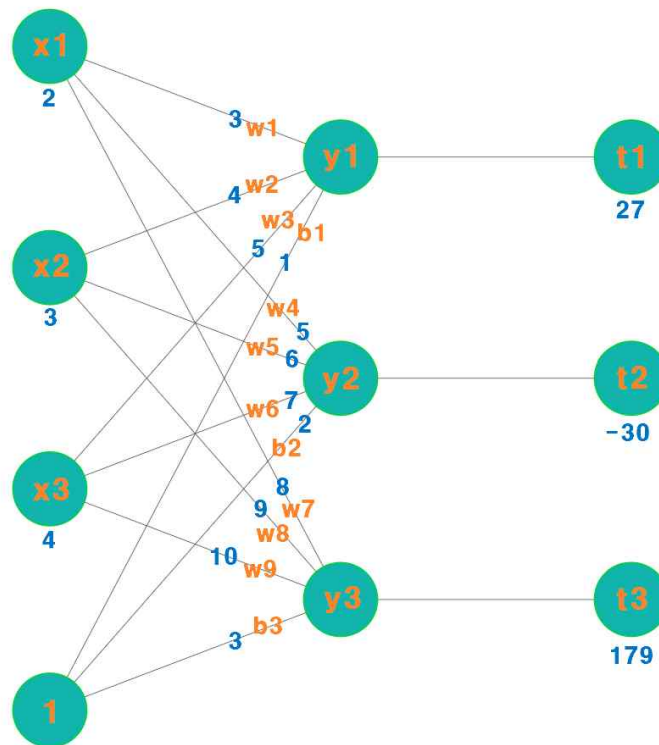
$$Y_b W^T = X_b \quad \textcircled{2}$$

가중치, 편향 역전파

$$X^T Y_b = W_b \quad \textcircled{3}$$

$$1 Y_b = B_b \quad \textcircled{4}$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 X , 가중치 W , 편향 B , 목표값 T 는 다음과 같습니다.

$$\begin{aligned} [x_1 \ x_2 \ x_3] &= [2 \ 3 \ 4] = X \\ \begin{bmatrix} w_1 & w_4 & w_7 \\ w_2 & w_5 & w_8 \\ w_3 & w_6 & w_9 \end{bmatrix} &= \begin{bmatrix} 3 & 5 & 8 \\ 4 & 6 & 9 \\ 5 & 7 & 10 \end{bmatrix} = W \\ [b_1 \ b_2 \ b_3] &= [1 \ 2 \ 3] = B \\ [t_1 \ t_2 \ t_3] &= [27 \ -30 \ 179] = T \end{aligned}$$

X 를 상수로 고정한 채 W , B 에 대해 학습을 수행해 봅니다.


*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

```
01~03 : # 이전 예제와 같습니다.
04 :
05 : X = np.array([[2, 3, 4]])
06 : T = np.array([[27, -30, 179]])
```

```
07 : W = np.array([[3, 5, 8],  
08 :               [4, 6, 9],  
09 :               [5, 7, 10]])  
10 : B = np.array([[1, 2, 3]])  
11 :  
12~끝 : # 이전 예제와 같습니다.
```

05~10 : X, T, W, B를 변경해줍니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 35  
Y = [[27.000 -30.000 179.000]]  
E = 0.0000001  
Xb =  
[[-0.005 -0.007 -0.009]]  
Wb =  
[[ 0.000  0.001 -0.001]  
 [ 0.000  0.001 -0.001]  
 [ 0.000  0.001 -0.001]]  
Bb =  
[[ 0.000  0.000 -0.000]]  
W =  
[[ 2.200 -0.867 14.200]  
 [ 2.800 -2.800 18.300]  
 [ 3.400 -4.733 22.400]]  
B =  
[[ 0.600 -0.933  6.100]]  
epoch = 36  
Y = [[27.000 -30.000 179.000]]  
E = 0.0000001
```

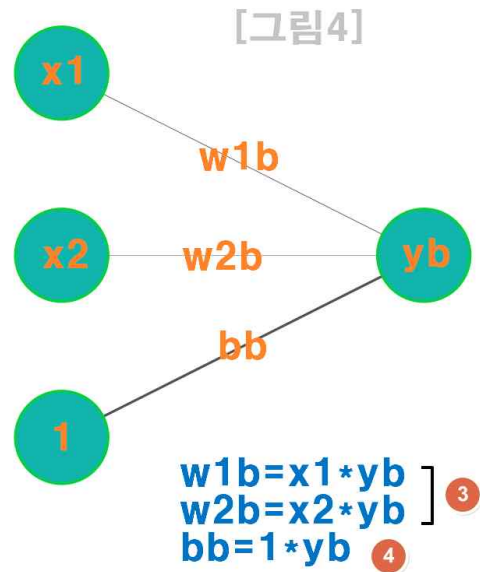
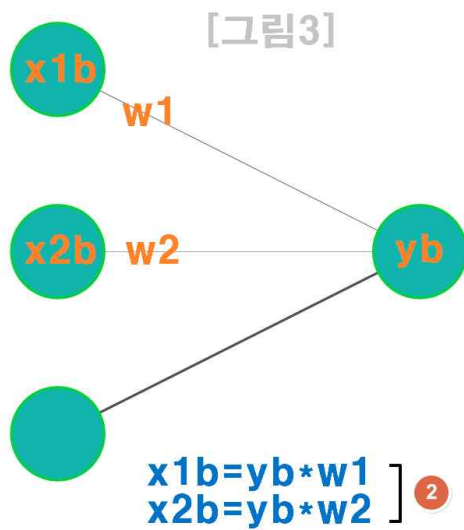
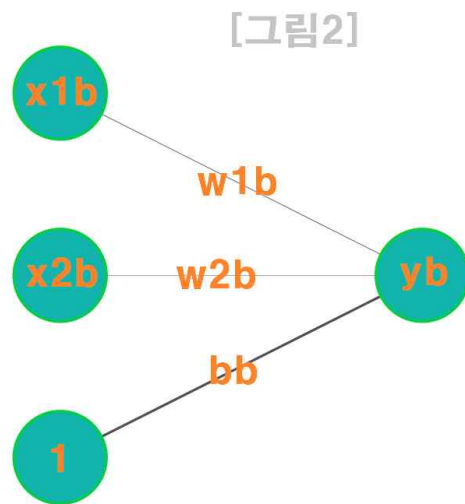
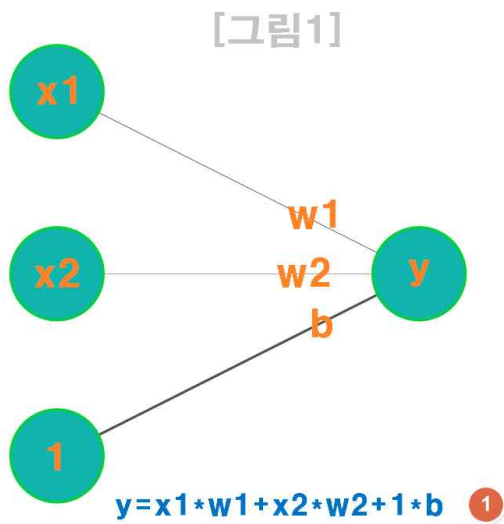
36회 째 학습이 완료되는 것을 볼 수 있습니다.

앞의 예제의 결과와 비교해 봅니다.

```
epoch = 35
y1, y2, y3 = 27.000, -30.000, 179.000
E = 0.0000001
x1b, x2b, x3b = -0.005, -0.007, -0.009
w1b, w4b, w7b = 0.000, 0.001, -0.001
w2b, w5b, w8b = 0.000, 0.001, -0.001
w3b, w6b, w9b = 0.000, 0.001, -0.001
b1b, b2b, b3b = 0.000, 0.000, -0.000
w1, w4, w7 = 2.200, -0.867, 14.200
w2, w5, w8 = 2.800, -2.800, 18.300
w3, w6, w9 = 3.400, -4.733, 22.400
b1, b2, b3 = 0.600, -0.933, 6.100
epoch = 36
y1, y2, y3 = 27.000, -30.000, 179.000
E = 0.0000001
```

04 입력2 출력1 인공 신경 구현하기

다음 그림은 입력2 출력1로 구성된 인공 신경과 순전파 역전파 수식을 나타냅니다. 우리는 다음 수식을 행렬식으로 유도한 후, NumPy를 이용하여 인공 신경을 구현해 봅니다.



*** ② x1b, x2b 노드가 입력층일 경우엔 이 수식은 필요하지 않습니다.

이 그림을 통해 앞에서 우리는 다음과 같은 수식을 유도했습니다. 이런 형태의 수식을 다원일차연립방정식이라고 합니다.

순전파

$$x_1 w_1 + x_2 w_2 + 1b = y$$
 ①

입력 역전파

$$\left. \begin{aligned} y_b w_1 &= x_{1b} \\ y_b w_2 &= x_{2b} \end{aligned} \right\}$$
 ②

가중치, 편향 역전파

$$\left. \begin{array}{l} x_1 y_b = w_{1b} \\ x_2 y_b = w_{2b} \end{array} \right] \textcircled{3}$$
$$1 y_b = b_b \textcircled{4}$$

다원일차연립방적식은 행렬을 이용하면 깔끔하게 정리할 수 있습니다. 위 수식을 행렬식으로 정리하면 다음과 같습니다.

순전파

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + [b] = [y] \textcircled{1}$$

입력 역전파

$$[y_b] [w_1 \ w_2] =$$
$$[y_b] \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}^T = [x_{1b} \ x_{2b}] \textcircled{2}$$

가중치, 편향 역전파

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} [y_b] =$$
$$[x_1 \ x_2]^T [y_b] = \begin{bmatrix} w_{1b} \\ w_{2b} \end{bmatrix} \textcircled{3}$$
$$1 [y_b] = [b_b] \textcircled{4}$$

②에서 다음은 순전파 때 사용된 가중치의 전치 행렬입니다.

$$[w_1 \ w_2] = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}^T$$

전치행렬은 가로줄과 세로줄이 바뀐 행렬입니다.

③에서 다음은 순전파 때 사용된 입력의 전치 행렬입니다.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$$

위 수식에서 표현된 행렬들에 다음과 같이 이름을 붙여줍니다.

$$\begin{aligned} \begin{bmatrix} x_1 & x_2 \end{bmatrix} &= X \\ \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} &= W \\ [b] &= B \\ [y] &= Y \\ [y_b] &= Y_b \\ \begin{bmatrix} w_1 & w_2 \end{bmatrix} &= \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}^T = W^T \\ \begin{bmatrix} x_{1b} & x_{2b} \end{bmatrix} &= X_b \\ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T = X^T \\ \begin{bmatrix} w_{1b} \\ w_{2b} \end{bmatrix} &= W_b \\ [b_b] &= B_b \end{aligned}$$

그러면 앞의 행렬식은 다음과 같이 정리할 수 있습니다. 이 수식은 행렬의 크기와 상관없이 적용됩니다. 주의할 점은 행렬곱은 순서를 변경하면 안됩니다.

순전파

$$Y = XW + B \quad \textcircled{1}$$

입력 역전파

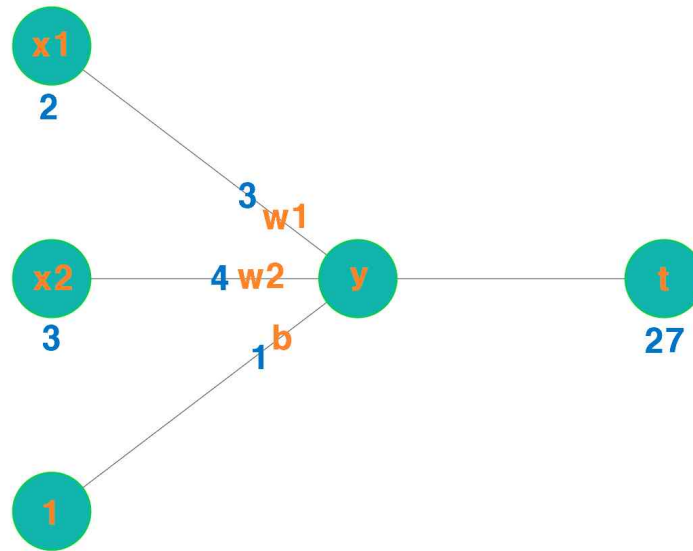
$$Y_b W^T = X_b \quad \textcircled{2}$$

가중치, 편향 역전파

$$X^T Y_b = W_b \quad \textcircled{3}$$

$$1 Y_b = B_b \quad \textcircled{4}$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 X , 가중치 W , 편향 B , 목표값 T 는 다음과 같습니다.

$$\begin{aligned} \begin{bmatrix} x_1 & x_2 \end{bmatrix} &= \begin{bmatrix} 2 & 3 \end{bmatrix} = X \\ \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} &= \begin{bmatrix} 3 \\ 4 \end{bmatrix} = W \\ [b] &= [1] = B \\ [t] &= [27] = T \end{aligned}$$

X 를 상수로 고정한 채 W , B 에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.


1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

```

01~03 : # 이전 예제와 같습니다.
04 :
05 : X = np.array([[2, 3]])
06 : T = np.array([[27]])
07 : W = np.array([[3],
08 :                [4]])
09 : B = np.array([[1]])
10 :
11~끝 : # 이전 예제와 같습니다.

```

05~09 : X , T , W , B 를 변경해줍니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 64
Y = [[26.999]]
E = 0.0000001
Xb =
[[-0.002 -0.003]]
Wb =
[[-0.001]
 [-0.002]]
Bb =
[[-0.001]]
W =
[[ 4.143]
 [ 5.714]]
B =
[[ 1.571]]
epoch = 65
Y = [[27.000]]
E = 0.0000001
```

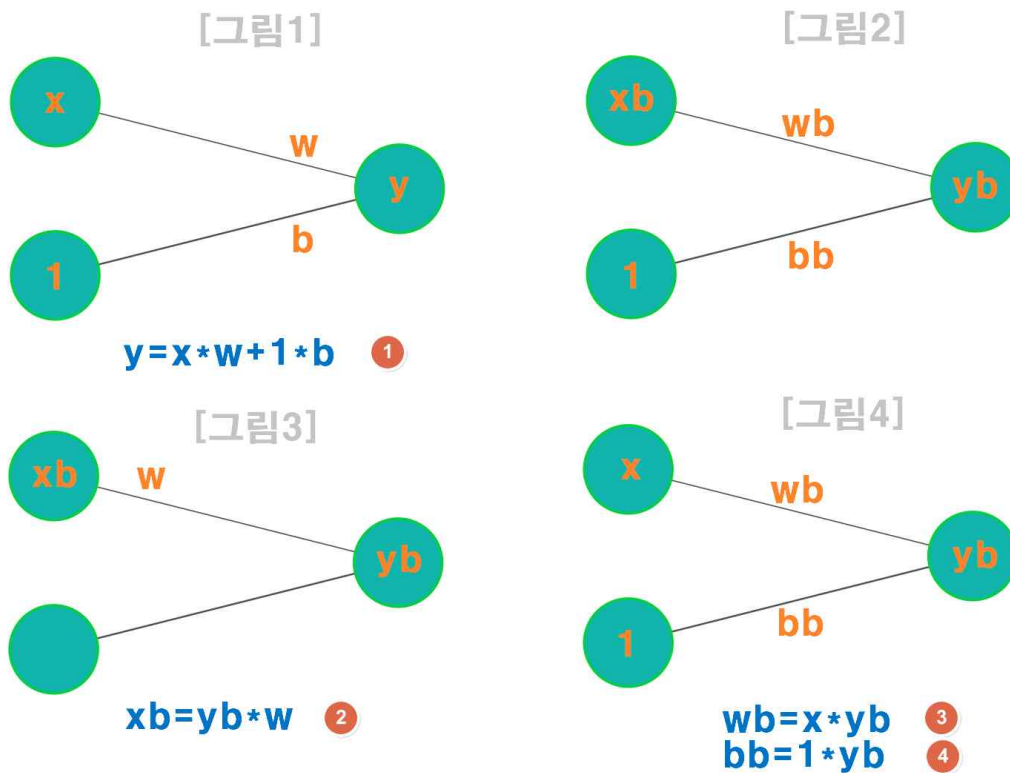
65회 째 학습이 완료되는 것을 볼 수 있습니다.

앞의 예제의 결과와 비교해 봅니다.

```
epoch = 64
y = 26.999
E = 0.0000001
x1b, x2b = -0.002, -0.003
w1b, w2b, bb = -0.001, -0.002, -0.001
w1, w2, b = 4.143, 5.714, 1.571
epoch = 65
y = 27.000
E = 0.0000001
```

05 입력1 출력1 인공 신경 구현하기

다음 그림은 입력1 출력1로 구성된 인공 신경과 순전파 역전파 수식을 나타냅니다. 우리는 다음 수식을 행렬식으로 유도한 후, NumPy를 이용하여 인공 신경을 구현해 봅니다.



*** ② xb 노드가 입력층일 경우엔 이 수식은 필요하지 않습니다.

이 그림을 통해 앞에서 우리는 다음과 같은 수식을 유도했습니다. 이런 형태의 수식을 다원일차연립방정식이라고 합니다.

순전파	
$xw + 1b = y$	①
입력 역전파	
$y_b w = x_b$	②
가중치, 편향 역전파	
$x y_b = w_b$	③
$1 y_b = b_b$	④

다원일차연립방정식은 행렬을 이용하면 깔끔하게 정리할 수 있습니다. 위 수식을 행렬식으로 정리하면 다음과 같습니다.

순전파	
$[x][w] + 1[b] = [y]$	①

입력 역전파

$$\begin{aligned} [y_b] [w] &= \\ [y_b] [w]^T &= [x_b] \quad \textcircled{2} \end{aligned}$$

가중치, 편향 역전파

$$\begin{aligned} [x] [y_b] &= \\ [x]^T [y_b] &= [w_b] \quad \textcircled{3} \\ 1 [y_b] &= [b_b] \quad \textcircled{4} \end{aligned}$$

②에서 다음은 순전파 때 사용된 가중치의 전치 행렬입니다.

$$[w] = [w]^T$$

전치행렬은 가로줄과 세로줄이 바뀐 행렬입니다. 여기서 가중치는 1x1 행렬이며 전치 행렬과 원 행렬의 모양은 같습니다. 여기서는 수식을 일반화하기 위해 전치 행렬 형태로 표현하고 있습니다.

③에서 다음은 순전파 때 사용된 입력의 전치 행렬입니다.

$$[x] = [x]^T$$

여기서 입력은 1x1 행렬이며 전치 행렬과 원 행렬의 모양은 같습니다. 여기서는 수식을 일반화하기 위해 전치 행렬 형태로 표현하고 있습니다.

위 수식에서 표현된 행렬들에 다음과 같이 이름을 붙여줍니다.

$$\begin{aligned} [x] &= X \\ [w] &= W \\ [b] &= B \\ [y] &= Y \\ [y_b] &= Y_b \\ [w] &= [w]^T = W^T \\ [x_b] &= X_b \\ [x] &= [x]^T = X^T \\ [w_b] &= W_b \\ [b_b] &= B_b \end{aligned}$$

그러면 앞의 행렬식은 다음과 같이 정리할 수 있습니다. 이 수식은 행렬의 크기와 상관없이 적용됩니다. 주의할 점은 행렬곱은 순서를 변경하면 안됩니다.

순전파

$Y = XW + B$ ①

입력 역전파

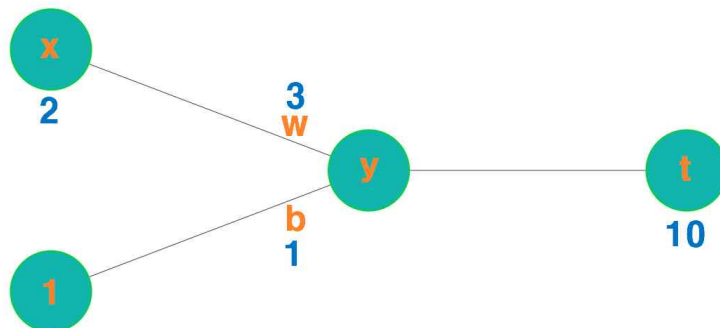
$Y_b W^T = X_b$ ②

가중치, 편향 역전파

$X^T Y_b = W_b$ ③

$1 Y_b = B_b$ ④

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 X , 가중치 W , 편향 B , 목표값 T 는 다음과 같습니다.

$$\begin{aligned} [x] &= [2] = X \\ [w] &= [3] = W \\ [b] &= [1] = B \\ [t] &= [10] = T \end{aligned}$$

X 를 상수로 고정된 채 W , B 에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

- 이전 예제를 복사합니다.
- 다음과 같이 예제를 수정합니다.


01~03 : # 이전 예제와 같습니다.


```

04 :
05 : X = np.array([[2]])
06 : T = np.array([[10]])
07 : W = np.array([3])
08 : B = np.array([1])
09 :
10~끝 : # 이전 예제와 같습니다.

```

05~08 : X, T, W, B를 변경해줍니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

epoch = 171
Y = [[10.000]]
E = 0.0000001
Xb =
[[-0.002]]
Wb =
[[-0.001]]
Bb =
[[-0.000]]
W =
[[ 4.200]]
B =
[[ 1.600]]
epoch = 172
Y = [[10.000]]
E = 0.0000001

```

172회 째 학습이 완료되는 것을 볼 수 있습니다.

앞의 예제의 결과와 비교해 봅니다.

```

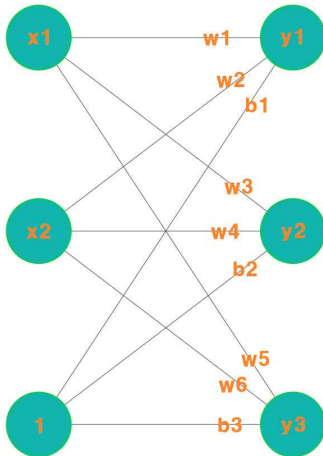
epoch = 170
y = 10.000
E = 0.0000001
xb = -0.002, wb = -0.001, bb = -0.000
x = 2.000, w = 4.200, b = 1.600
epoch = 171
y = 10.000
E = 0.0000001
xb = -0.002, wb = -0.001, bb = -0.000
x = 2.000, w = 4.200, b = 1.600
epoch = 172
y = 10.000
E = 0.0000001

```

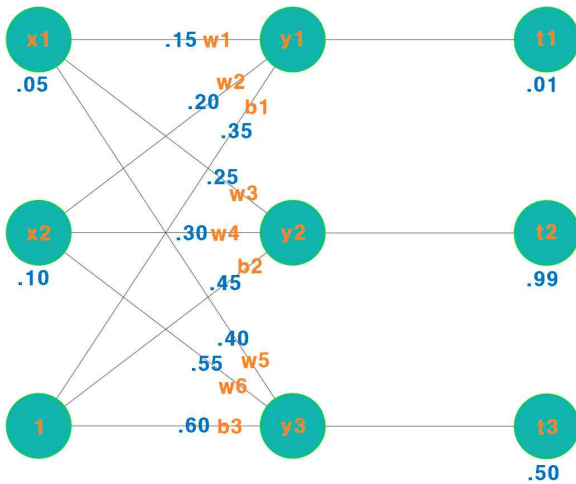
연습문제

❶ 입력2 출력3

1. 다음은 입력2 출력3의 단일 인공 신경입니다. 이 인공 신경의 순전파, 역전파 행렬식을 구합니다.

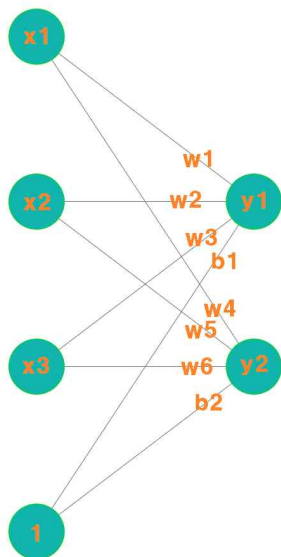


2. 앞에서 구한 행렬식을 이용하여 다음과 같이 초기화된 인공 신경을 NumPy를 이용하여 구현하고 학습시켜 봅니다. 입력값 X는 상수로 처리합니다.

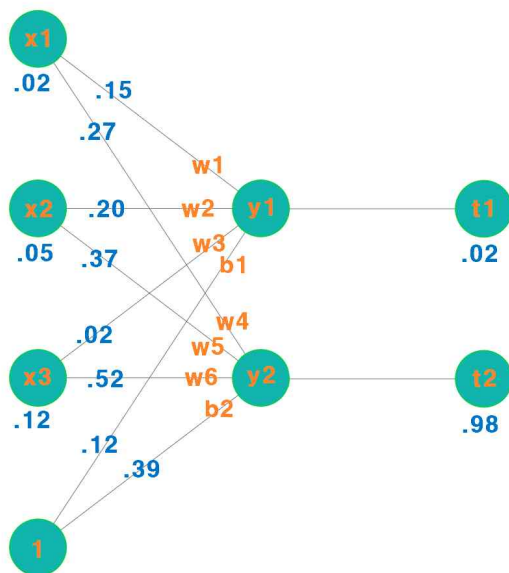


❷ 입력3 출력2

1. 다음은 입력2 출력3의 단일 인공 신경입니다. 이 인공 신경의 순전파, 역전파 행렬식을 구합니다.



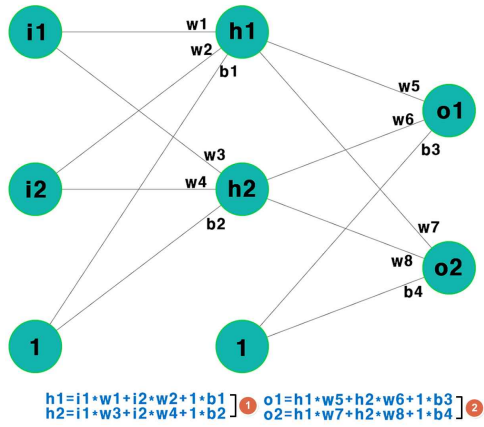
2. 앞에서 구한 행렬식을 이용하여 다음과 같이 초기화된 인공 신경을 NumPy를 이용하여 구현하고 학습시켜 봅니다. 입력값 X는 상수로 처리합니다.



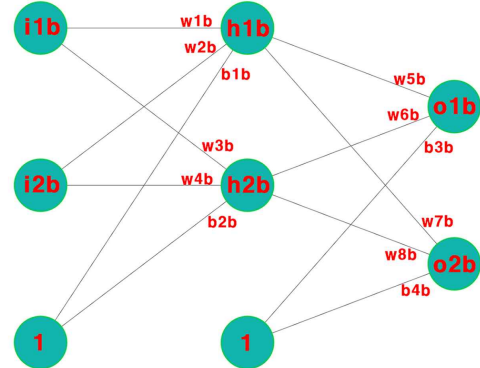
06 입력2 은닉2 출력2 인공 신경망 구현하기

다음 그림은 입력2 은닉2 출력2로 구성된 인공 신경과 순전파 역전파 수식을 나타냅니다. 우리는 다음 수식을 행렬식으로 유도한 후, NumPy를 이용하여 인공 신경을 구현해 봅니다.

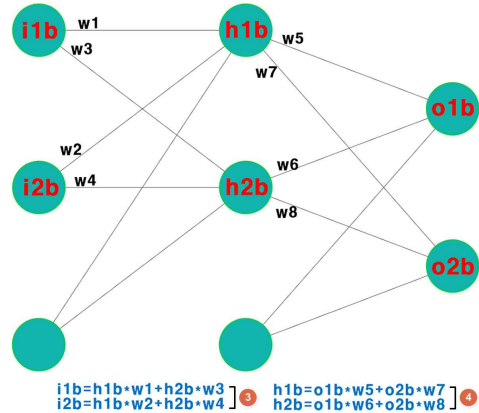
[그림1]



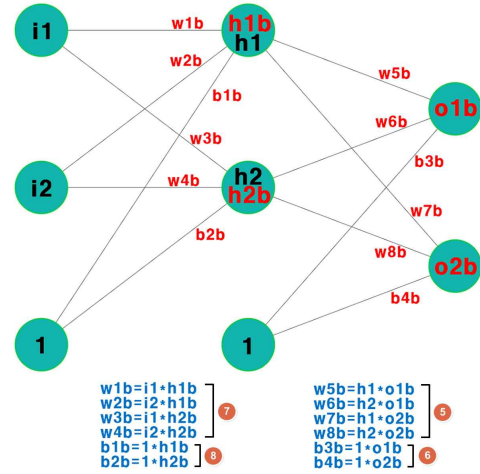
[그림2]



[그림3]



[그림4]



*** ㉓ i1b, i2b 노드가 입력층일 경우엔 이 수식은 필요하지 않습니다.

이 그림을 통해 앞에서 우리는 다음과 같은 수식을 유도했습니다. 이런 형태의 수식을 다원일 차연립방정식이라고 합니다.

순전파

$$\begin{aligned} i_1 w_1 + i_2 w_2 + 1 b_1 &= h_1 \\ i_1 w_3 + i_2 w_4 + 1 b_2 &= h_2 \end{aligned} \quad \textcircled{1}$$

$$\begin{aligned} h_1 w_5 + h_2 w_6 + 1 b_3 &= o_1 \\ h_1 w_7 + h_2 w_8 + 1 b_4 &= o_2 \end{aligned} \quad \textcircled{2}$$

역전파

$$\begin{aligned} o_{1b} w_5 + o_{2b} w_7 &= h_{1b} \\ o_{1b} w_6 + o_{2b} w_8 &= h_{2b} \end{aligned} \quad \textcircled{4}$$

가중치, 편향 역전파

$$\left[\begin{array}{l} i_1 h_{1b} = w_{1b} \\ i_2 h_{1b} = w_{2b} \\ i_1 h_{2b} = w_{3b} \\ i_2 h_{2b} = w_{4b} \end{array} \right] \quad \textcircled{7}$$

$$\left[\begin{array}{l} 1 h_{1b} = b_{1b} \\ 1 h_{2b} = b_{2b} \end{array} \right] \quad \textcircled{8}$$

$$\left[\begin{array}{l} h_1 o_{1b} = w_{5b} \\ h_2 o_{1b} = w_{6b} \\ h_1 o_{2b} = w_{7b} \\ h_2 o_{2b} = w_{8b} \end{array} \right] \quad \textcircled{5}$$

$$\left[\begin{array}{l} 1 o_{1b} = b_{3b} \\ 1 o_{2b} = b_{4b} \end{array} \right] \quad \textcircled{6}$$

다원일차연립방정식은 행렬을 이용하면 깔끔하게 정리할 수 있습니다. 위 수식을 행렬식으로 정리하면 다음과 같습니다.

순전파

$$\begin{bmatrix} i_1 & i_2 \end{bmatrix} \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 \end{bmatrix} \quad \textcircled{1}$$

$$\begin{bmatrix} h_1 & h_2 \end{bmatrix} \begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix} + \begin{bmatrix} b_3 & b_4 \end{bmatrix} = \begin{bmatrix} o_1 & o_2 \end{bmatrix} \quad \textcircled{2}$$

역전파

$$\left[\begin{array}{l} [o_{1b} \ o_{2b}] \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \end{bmatrix} = [h_{1b} \ h_{2b}] \\ [o_{1b} \ o_{2b}] \begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix}^T = [h_{1b} \ h_{2b}] \end{array} \right] \quad \textcircled{4}$$

가중치, 편향 역전파

$$\left[\begin{array}{l} [i_1] \\ [i_2] \end{array} \right] [h_{1b} \ h_{2b}] = \left[\begin{array}{l} [w_{1b} \ w_{3b}] \\ [w_{2b} \ w_{4b}] \end{array} \right] \quad \textcircled{7}$$

$$[i_1 \ i_2]^T [h_{1b} \ h_{2b}] = \left[\begin{array}{l} [w_{1b} \ w_{3b}] \\ [w_{2b} \ w_{4b}] \end{array} \right] \quad \textcircled{7}$$

$$1 [h_{1b} \ h_{2b}] = [b_{1b} \ b_{2b}] \quad \textcircled{8}$$

$$\left[\begin{array}{l} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \begin{bmatrix} o_{1b} & o_{2b} \end{bmatrix} = \begin{bmatrix} w_{5b} & w_{7b} \\ w_{6b} & w_{8b} \end{bmatrix} \\ \begin{bmatrix} h_1 & h_2 \end{bmatrix}^T \begin{bmatrix} o_{1b} & o_{2b} \end{bmatrix} = \begin{bmatrix} w_{5b} & w_{7b} \\ w_{6b} & w_{8b} \end{bmatrix} \\ 1 \begin{bmatrix} o_{1b} & o_{2b} \end{bmatrix} = \begin{bmatrix} b_{3b} & b_{4b} \end{bmatrix} \end{array} \right] \quad \begin{array}{l} \textcircled{5} \\ \textcircled{6} \end{array}$$

④에서 다음은 순전파 때 사용된 가중치의 전치 행렬입니다.

$$\begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \end{bmatrix} = \begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix}^T$$

전치행렬은 가로줄과 세로줄이 바뀐 행렬입니다.

⑦, ⑤에서 다음은 순전파 때 사용된 입력의 전치 행렬입니다.

$$\begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} i_1 & i_2 \end{bmatrix}^T$$

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 \end{bmatrix}^T$$

위 수식에서 표현된 행렬들에 다음과 같이 이름을 붙여줍니다.

$\begin{bmatrix} i_1 & i_2 \end{bmatrix} = I$	$\begin{bmatrix} h_{1b} & h_{2b} \end{bmatrix} = H_b$
$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = W_h$	$\begin{bmatrix} i_1 & i_2 \end{bmatrix}^T = I^T$
$\begin{bmatrix} b_1 & b_2 \end{bmatrix} = B_h$	$\begin{bmatrix} w_{1b} & w_{3b} \\ w_{2b} & w_{4b} \end{bmatrix} = W_{hb}$
$\begin{bmatrix} h_1 & h_2 \end{bmatrix} = H$	$\begin{bmatrix} b_{1b} & b_{2b} \end{bmatrix} = B_{hb}$
$\begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix} = W_o$	$\begin{bmatrix} h_1 & h_2 \end{bmatrix}^T = H^T$
$\begin{bmatrix} b_1 & b_2 \end{bmatrix} = B_o$	$\begin{bmatrix} w_{5b} & w_{7b} \\ w_{6b} & w_{8b} \end{bmatrix} = W_{ob}$
$\begin{bmatrix} o_1 & o_2 \end{bmatrix} = O$	$\begin{bmatrix} b_{3b} & b_{4b} \end{bmatrix} = B_{ob}$
$\begin{bmatrix} o_{1b} & o_{2b} \end{bmatrix} = O_b$	
$\begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix}^T = W_o^T$	

그러면 앞의 행렬식은 다음과 같이 정리할 수 있습니다. 이 수식은 행렬의 크기와 상관없이 적용됩니다. 주의할 점은 행렬곱은 순서를 변경하면 안됩니다.

순전파

$$H = IW_h + B_h \quad \textcircled{1}$$

$$O = HW_o + B_o \quad \textcircled{2}$$

역전파

$$O_b W_o^T = H_b \quad \textcircled{4}$$

가중치, 편향 역전파

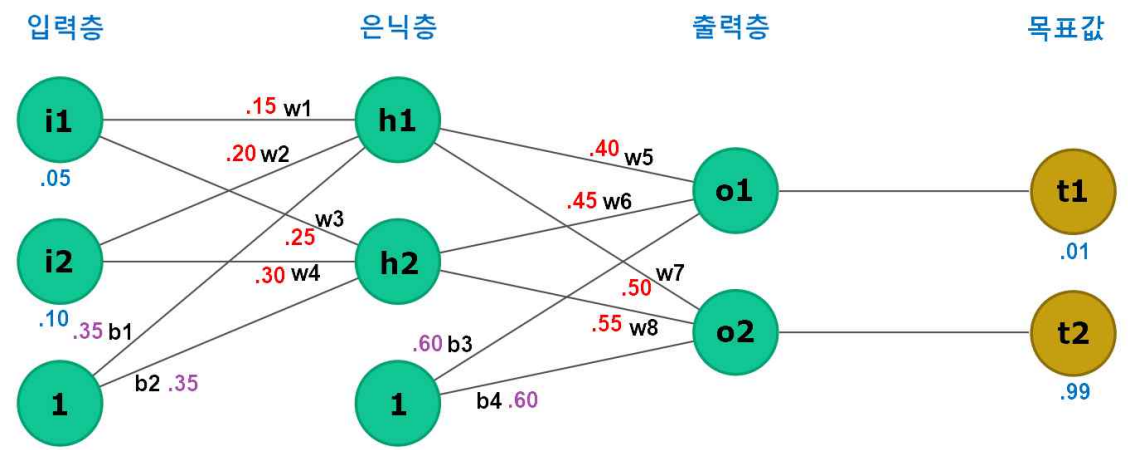
$$I^T H_b = W_{hb} \quad \textcircled{7}$$

$$1 H_b = B_{hb} \quad \textcircled{8}$$

$$H^T O_b = W_{ob} \quad \textcircled{5}$$

$$1 O_b = B_{ob} \quad \textcircled{6}$$

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 I , 가중치 W_h , W_o , 편향 B_h , B_o , 목표값 T 는 다음과 같습니다.

$$\begin{aligned} \begin{bmatrix} i_1 & i_2 \end{bmatrix} &= \begin{bmatrix} .05 & .10 \end{bmatrix} = I \\ \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} &= \begin{bmatrix} .15 & .25 \\ .20 & .30 \end{bmatrix} = W_h \\ \begin{bmatrix} b_1 & b_2 \end{bmatrix} &= \begin{bmatrix} .35 & .35 \end{bmatrix} = B_h \\ \begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix} &= \begin{bmatrix} .40 & .50 \\ .45 & .55 \end{bmatrix} = W_o \\ \begin{bmatrix} b_3 & b_4 \end{bmatrix} &= \begin{bmatrix} .60 & .60 \end{bmatrix} = B_o \\ \begin{bmatrix} t_1 & t_2 \end{bmatrix} &= \begin{bmatrix} .01 & .99 \end{bmatrix} = T \end{aligned}$$

I를 상수로 고정한 채 Wh, Wo, Bh, Bo에 대해 학습을 수행해 봅니다.

*** 이 값들은 임의의 값들입니다. 다른 값들을 사용하여 학습을 수행할 수도 있습니다.

1. 다음과 같이 예제를 작성합니다.


```
01 : import numpy as np
02 :
03 : np.set_printoptions(formatter={'float_kind':lambda x: "{0:6.3f}".format(x)})
04 :
05 : I = np.array([[.05, .10]])
06 : T = np.array([[.01, .99]])
07 : WH = np.array([[.15, .25],
08 :                 [.20, .30]])
09 : BH = np.array([[.35, .35]])
10 : WO = np.array([[.40, .50],
11 :                 [.45, .55]])
12 : BO = np.array([[.60, .60]])
13 :
14 : for epoch in range(1000):
15 :
16 :     print('epoch = %d' %epoch)
17 :
18 :     H = I @ WH + BH ❶
19 :     O = H @ WO + BO ❷
20 :     print(' O  =\n', O)
21 :
22 :     E = np.sum((O - T) ** 2 / 2)
23 :     print(' E  = %.7f' %E)
24 :     if E < 0.0000001:
25 :         break
```



```

26 :
27 :   Ob = O - T
28 :   Hb = Ob @ WO.T ④
29 :   WHb = I.T @ Hb ⑦
30 :   BHb = 1 * Hb ⑧
31 :   WOb = H.T @ Ob ⑤
32 :   BOb = 1 * Ob ⑥
33 :   print(' WHb =\n', WHb)
34 :   print(' BHb =\n', BHb)
35 :   print(' WOb =\n', WOb)
36 :   print(' BOb =\n', BOb)
37 :
38 :   lr = 0.01
39 :
40 :   WH = WH - lr * WHb
41 :   BH = BH - lr * BHb
42 :   WO = WO - lr * WOb
43 :   BO = BO - lr * BOb
44 :   print(' WH  =\n', WH)
45 :   print(' BH  =\n', BH)
46 :   print(' WO  =\n', WO)
47 :   print(' BO  =\n', BO)

```

2.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

epoch = 664
O  =
[[ 0.010  0.990]]
E  = 0.0000001
WHb =
[[-0.000  0.000]
 [-0.000  0.000]]
BHb =
[[-0.000  0.000]]
WOb =
[[ 0.000 -0.000]
 [ 0.000 -0.000]]
BOb =
[[ 0.000 -0.000]]
WH  =
[[ 0.143  0.242]
 [ 0.186  0.284]]
BH  =
[[ 0.213  0.186]]
WO  =
[[ 0.203  0.533]
 [ 0.253  0.583]]
BO  =
[[-0.095  0.730]]
epoch = 665
O  =
[[ 0.010  0.990]]
E  = 0.0000001

```

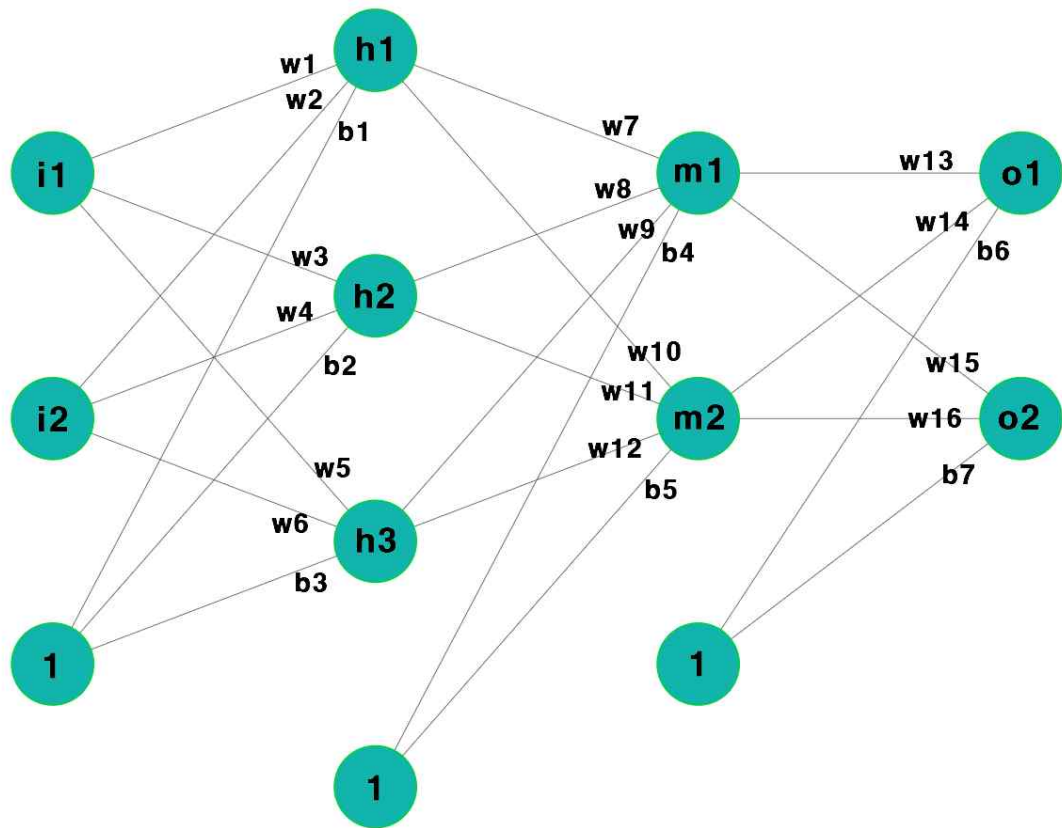
665회 째 학습이 완료되는 것을 볼 수 있습니다.

앞의 예제의 결과와 비교해 봅니다.

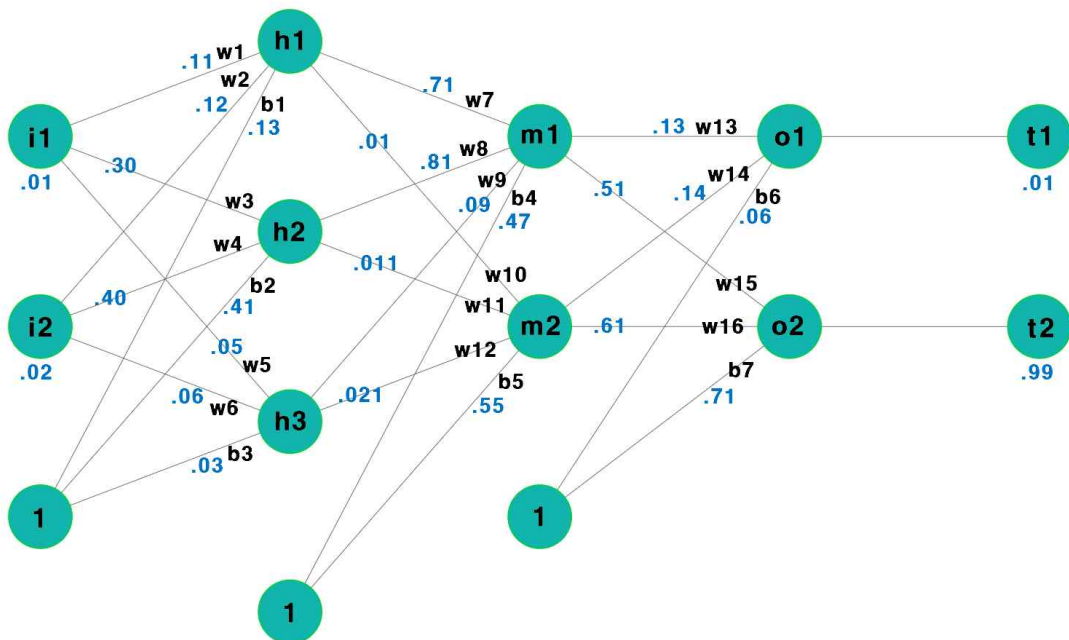
```
epoch = 664
h1, h2 = 0.239, 0.226
o1, o2 = 0.010, 0.990
E = 0.0000001
w1b, w3b = -0.000, 0.000
w2b, w4b = -0.000, 0.000
b1b, b2b = -0.000, 0.000
w5b, w7b = 0.000, -0.000
w6b, w8b = 0.000, -0.000
b3b, b4b = 0.000, -0.000
w1, w3 = 0.143, 0.242
w2, w4 = 0.186, 0.284
b1, b2 = 0.213, 0.186
w5, w7 = 0.203, 0.533
w6, w8 = 0.253, 0.583
b3, b4 = -0.095, 0.730
epoch = 665
h1, h2 = 0.239, 0.226
o1, o2 = 0.010, 0.990
E = 0.0000001
```

연습문제

1. 다음은 입력2 은닉3 은닉2 출력3의 심층 인공 신경망입니다. 이 신경망에는 2개의 은닉층이 포함되어 있습니다. 일반적으로 은닉층이 2층 이상일 경우 심층 인공 신경망이라고 합니다. 이 신경망의 순전파, 역전파 행렬식을 구합니다.

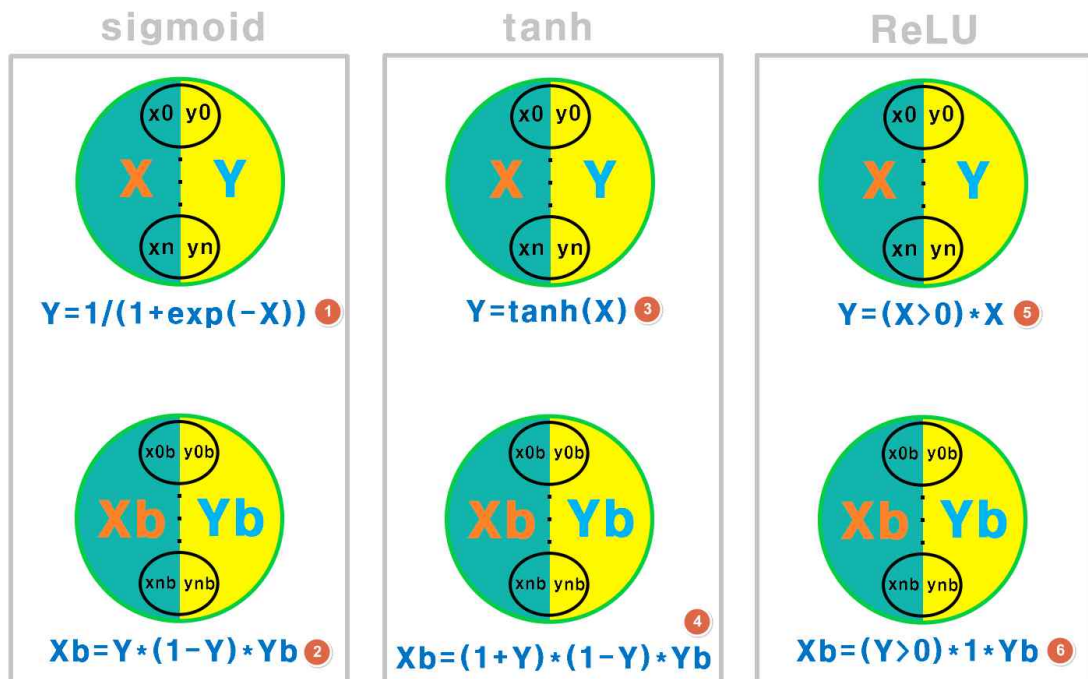


2. 앞에서 구한 행렬식을 이용하여 다음과 같이 초기화된 인공 신경을 구현하고 학습시켜 봅니다. 입력값 $i1, i2$ 는 상수로 처리합니다.



07 활성화 함수 적용하기

여기서는 sigmoid, tanh, ReLU 활성화 함수의 순전파와 역전파 수식을 살펴보고, 앞에서 NumPy를 이용해 구현한 인공 신경망에 활성화 함수를 적용하여 봅니다. 다음 그림은 활성화 함수의 순전파와 역전파 NumPy 수식을 나타냅니다.



이 그림에서 X, Y는 각각 $x_0 \sim x_n$, $y_0 \sim y_n$ (n 은 0보다 큰 정수)의 집합을 나타냅니다. 예를 들어, x_0 , y_0 는 하나의 노드내에서 활성화 함수의 입력과 출력을 의미합니다. X, Y는 하나의 층 내에서 활성화 함수의 입력과 출력 행렬을 의미합니다.

이상에서 필요한 행렬식을 정리하면 다음과 같습니다.

시그모이드 순전파와 역전파

$$Y = \frac{1}{1 + e^{-X}} \quad (1) \quad X_b = Y(1 - Y) Y_b \quad (2)$$

tanh 순전파와 역전파

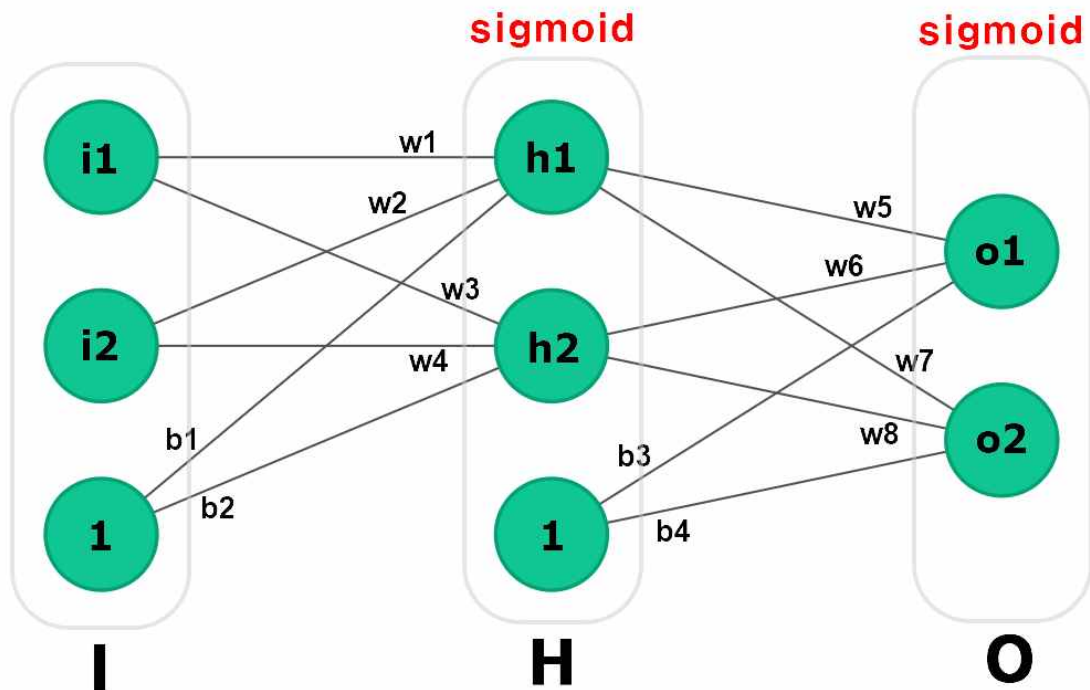
$$Y = \tanh(X) \quad (3) \quad X_b = (1 + Y)(1 - Y) Y_b \quad (4)$$

ReLU 순전파와 역전파

$$Y = (X > 0) X \quad (5) \quad X_b = (Y > 0) 1 Y_b \quad (6)$$

시그모이드 함수 적용해 보기

지금까지 정리한 행렬식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

```
01 : import numpy as np
02 :
03 : np.set_printoptions(formatter={'float_kind':lambda x: "{0:6.3f}".format(x)})
04 :
05 : I = np.array([[.05, .10]])
06 : T = np.array([[.01, .99]])
07 : WH = np.array([[.15, .25],
08 :                [.20, .30]])
09 : BH = np.array([[.35, .35]])
10 : WO = np.array([[.40, .50],
11 :                [.45, .55]])
12 : BO = np.array([[.60, .60]])
13 :
14 : for epoch in range(1000):
15 :
16 :     print('epoch = %d' %epoch)
17 :
```

```

18 : H = I @ WH + BH
19 : H = 1/(1+np.exp(-H)) ❶
20 :
21 : O = H @ WO + BO
22 : O = 1/(1+np.exp(-O)) ❶
23 :
24 : print(' O =\n', O)
25 :
26 : E = np.sum((O - T) ** 2 / 2)
27 : # print(' E = %.7f' %E)
28 : if E < 0.0000001:
29 :     break
30 :
31 : Ob = O - T
32 : Ob = Ob*O*(1-O) ❷
33 :
34 : Hb = Ob @ WO.T
35 : Hb = Hb*H*(1-H) ❷
36 :
37 : WHb = I.T @ Hb
38 : BHb = 1 * Hb
39 : WOb = H.T @ Ob
40 : BOb = 1 * Ob
41 : # print(' WHb =\n', WHb)
42 : # print(' BHb =\n', BHb)
43 : # print(' WOb =\n', WOb)
44 : # print(' BOb =\n', BOb)
45 :
46 : lr = 0.01
47 :
48 : WH = WH - lr * WHb
49 : BH = BH - lr * BHb
50 : WO = WO - lr * WOb
51 : BO = BO - lr * BOb
52 : # print(' WH =\n', WH)
53 : # print(' BH =\n', BH)
54 : # print(' WO =\n', WO)
55 : # print(' BO =\n', BO)

```


19 : 은닉층 H에 순전파 시그모이드 활성화 함수를 적용합니다.

22 : 출력층 O에 순전파 시그모이드 활성화 함수를 적용합니다.

32 : 역출력층 Ob에 역전파 시그모이드 활성화 함수를 적용합니다.

35 : 역은닉층 Hb에 역전파 시그모이드 활성화 함수를 적용합니다.

27, 41~44, 52~55 : 지우거나 주석 처리합니다.


3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 997
O =
[[ 0.306  0.844]]
epoch = 998
O =
[[ 0.306  0.844]]
epoch = 999
O =
[[ 0.306  0.844]]
```

(999+1)번째에 o1, o2가 각각 0.306, 0.844가 됩니다.

4. 다음과 같이 예제를 수정합니다.

```
14 : for epoch in range(10000):
```


5.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 9997
O =
[[ 0.063  0.943]]
epoch = 9998
O =
[[ 0.063  0.943]]
epoch = 9999
O =
[[ 0.063  0.943]]
```

(9999+1)번째에 o1, o2가 각각 0.063, 0.943이 됩니다.

6. 다음과 같이 예제를 수정합니다.

```
14 : for epoch in range(100000):
```



7.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 99997
O =
[[ 0.020  0.981]]
epoch = 99998
O =
[[ 0.020  0.981]]
epoch = 99999
O =
[[ 0.020  0.981]]
```

(99999+1)번째에 o1, o2가 각각 0.020, 0.981이 됩니다.

8. 다음과 같이 예제를 수정합니다.

```
14 : for epoch in range(1000000):
```


9.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 999997
O =
[[ 0.010  0.990]]
epoch = 999998
O =
[[ 0.010  0.990]]
epoch = 999999
O =
[[ 0.010  0.990]]
```

(999999+1)번째에 o1, o2가 각각 0.010, 0.990이 됩니다. 아직 오차는 0.0000001(천만분의 1)보다 큼니다.

10. 다음과 같이 예제를 수정합니다.

```
14 : for epoch in range(1000000):
```

11.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

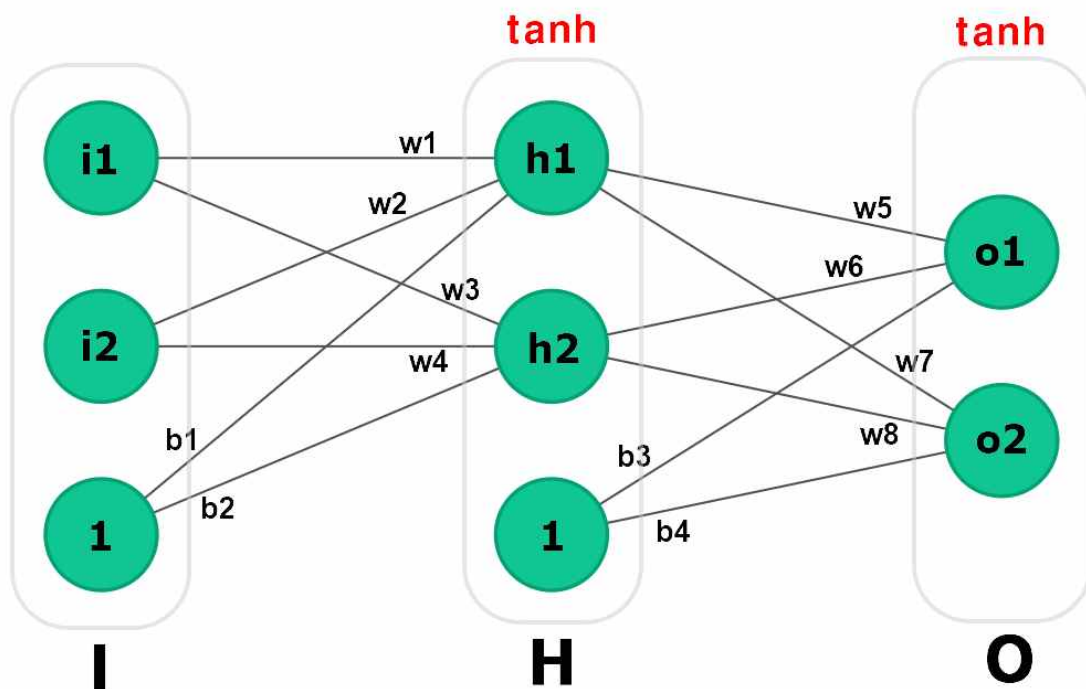
epoch = 1078009
O =
[[ 0.010  0.990]]
epoch = 1078010
O =
[[ 0.010  0.990]]
epoch = 1078011
O =
[[ 0.010  0.990]]

```

(1078011+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. o1, o2는 각각 0.010, 0.990이 된 상태입니다.

tanh 함수 적용해 보기

이번에는 이전 예제에 적용했던 sigmoid 함수를 tanh 함수로 변경해 봅니다. 다음 그림을 살펴봅니다.



1. 이전 예제를 복사합니다.

2. 다음과 같이 예제를 수정합니다.

```
18 : H = I @ WH + BH
19 : H = np.tanh(H) ③
20 :
21 : O = H @ WO + BO
22 : O = np.tanh(O) ③
```


19 : 은닉층 H에 순전파 tanh 활성화 함수를 적용합니다.

22 : 출력층 O에 순전파 tanh 활성화 함수를 적용합니다.

```
31 : Ob = O - T
32 : Ob = Ob*(1+O)*(1-O) ④
33 :
34 : Hb = Ob @ WO.T
35 : Hb = Hb*(1+H)*(1-H) ④
```

32 : 역출력층 Ob에 역전파 tanh 활성화 함수를 적용합니다.

35 : 역은닉층 Hb에 역전파 tanh 활성화 함수를 적용합니다.

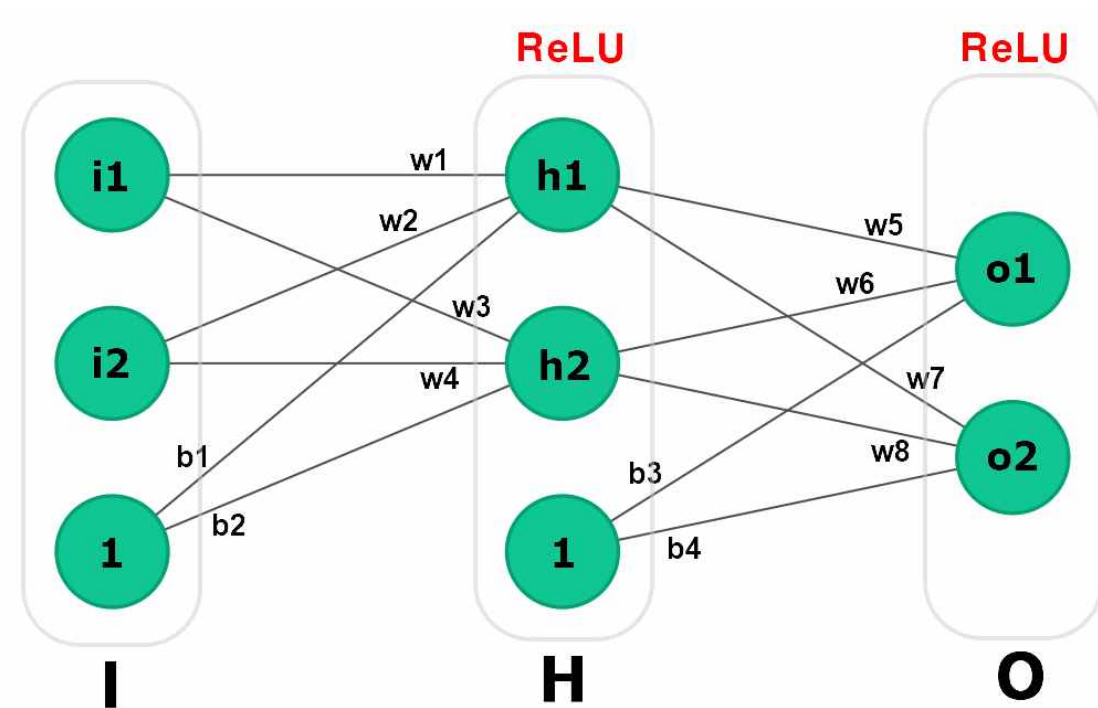
3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```
epoch = 236408
O =
[[ 0.010  0.990]]
epoch = 236409
O =
[[ 0.010  0.990]]
epoch = 236410
O =
[[ 0.010  0.990]]
```

(665+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. o1, o2는 각각 0.010, 0.990이 된 상태입니다. sigmoid, tanh 함수보다 결과가 훨씬 더 빨리 나오는 것을 볼 수 있습니다.

ReLU 함수 적용해 보기

이번에는 이전 예제에 적용했던 tanh 함수를 ReLU 함수로 변경해 봅니다. 다음 그림을 살펴 봅니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

```

18 :   H = I @ WH + BH
19 :   H = (H>0)*H ③
20 :
21 :   O = H @ WO + BO
22 :   O = (O>0)*O ③

```

- 19 : 은닉층 H에 순전파 ReLU 활성화 함수를 적용합니다.
 22 : 출력층 O에 순전파 ReLU 활성화 함수를 적용합니다.

```

31 :   Ob = O - T
32 :   Ob = Ob*(O>0)*1 ④
33 :
34 :   Hb = Ob @ WO.T
35 :   Hb = Hb*(H>0)*1 ④

```

- 32 : 역출력층 Ob에 역전파 ReLU 활성화 함수를 적용합니다.
 35 : 역은닉층 Hb에 역전파 ReLU 활성화 함수를 적용합니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

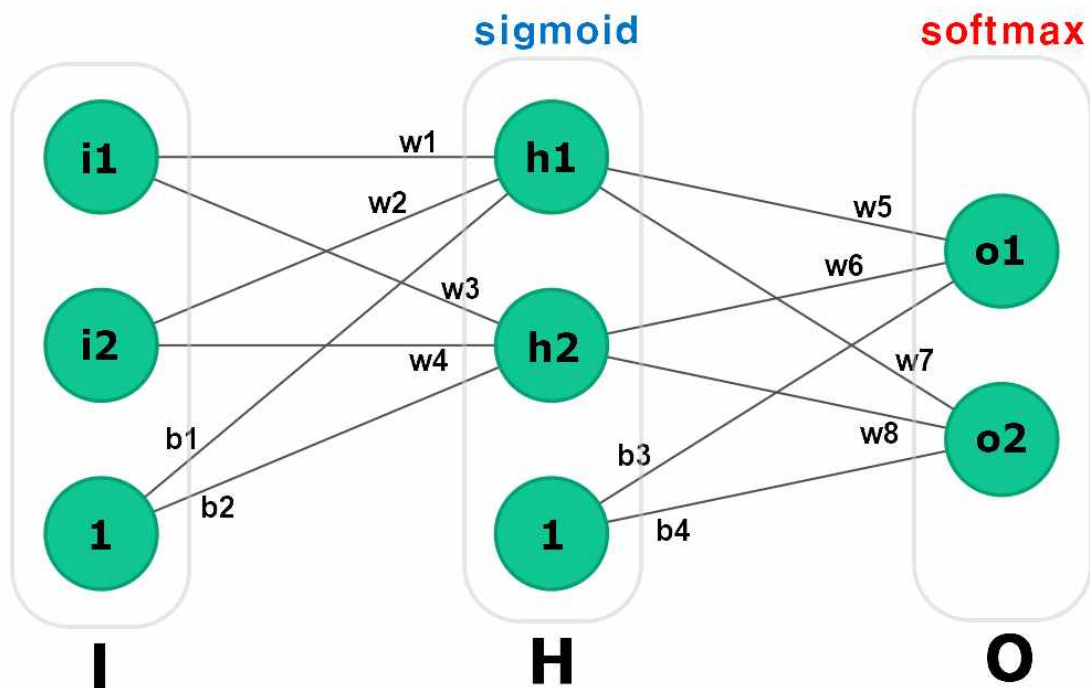
epoch = 663
O =
[[ 0.010  0.990]]
epoch = 664
O =
[[ 0.010  0.990]]
epoch = 665
O =
[[ 0.010  0.990]]

```

(665+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. o1, o2는 각각 0.010, 0.990이 된 상태입니다. sigmoid, tanh 함수보다 결과가 훨씬 더 빨리 나오는 것을 볼 수 있습니다.

출력층에 소프트맥스 함수 적용해 보기

이번에는 출력층에 소프트맥스 함수를 적용해 봅니다. 다음 그림을 살펴봅니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

```

01 : import numpy as np
02 :
03 : np.set_printoptions(formatter={'float_kind':lambda x: "{0:6.3f}".format(x)})
04 :
05 : I = np.array([[.05, .10]])
06 : T = np.array([[ 0, 1]])
07 : WH = np.array([[.15, .25],
08 :                 [.20, .30]])
09 : BH = np.array([[.35, .35]])
10 : WO = np.array([[.40, .50],
11 :                 [.45, .55]])
12 : BO = np.array([[.60, .60]])
13 :
14 : for epoch in range(10000000):
15 :
16 :     print('epoch = %d' %epoch)
17 :
18 :     H = I @ WH + BH
19 :     H = 1/(1+np.exp(-H))
20 :
21 :     O = H @ WO + BO
22 :     OM = O - np.max(O)
23 :     O = np.exp(OM)/np.sum(np.exp(OM))
24 :
25 :     print(' O =\n', O)
26 :
27 :     E = np.sum(-T*np.log(O))
28 :     # print(' E = %.7f' %E)
29 :     if E < 0.0001:
30 :         break
31 :
32 :     Ob = O - T
33 :
34 :
35 :     Hb = Ob @ WO.T
36 :     Hb = Hb*H*(1-H)
37 :
38 :     WHb = I.T @ Hb
39 :     BHb = 1 * Hb
40 :     WOb = H.T @ Ob

```

```

41 : BOb = 1 * Ob
42 : # print(' WHb =\n', WHb)
43 : # print(' BHb =\n', BHb)
44 : # print(' WOb =\n', WOb)
45 : # print(' BOb =\n', BOb)
46 :
47 : lr = 0.01
48 :
49 : WH = WH - lr * WHb
50 : BH = BH - lr * BHb
51 : WO = WO - lr * WOb
52 : BO = BO - lr * BOb
53 : # print(' WH =\n', WH)
54 : # print(' BH =\n', BH)
55 : # print(' WO =\n', WO)
56 : # print(' BO =\n', BO)

```

06 : 목표값을 각각 0과 1로 변경합니다.

22, 23 : 출력층의 활성화 함수를 소프트맥스로 변경합니다.

22 : O의 각 항목에서 O의 가장 큰 항목 값을 빼줍니다. 이렇게 하면 23 줄에서 오버플로우를 막을 수 있습니다. O에 대한 최종 결과는 같습니다. 자세한 내용은 [소프트맥스 오버플로우]를 검색해 봅니다.

27 : 오차 계산을 크로스 엔트로피 오차 형태의 수식으로 변경합니다. 소프트맥스 활성화 함수는 크로스 엔트로피 오차와 같이 사용합니다.


$$E = - \sum_k t_k \log o_k$$

29 : for 문을 빠져 나가는 오차값을 0.0001로 변경합니다. 여기서 사용하는 값의 크기에 따라 학습의 정확도와 학습 시간이 결정됩니다.

32 : 소프트맥스 함수의 역전파 오차 계산 부분은 다음과 같습니다. 소프트맥스 함수는 크로스 엔트로피 함수와 같이 사용될 때 역전파 시 소프트맥스 함수를 역으로 거쳐 전파되는 오차가 다음과 같이 예측값과 목표값의 차가 됩니다.

$$o_{kb} = o_k - t_k$$

그래서 일반적으로 소프트맥스 함수를 활성화 함수로 사용할 경우 오차 함수는 크로스 엔트로피 오차 함수가 됩니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

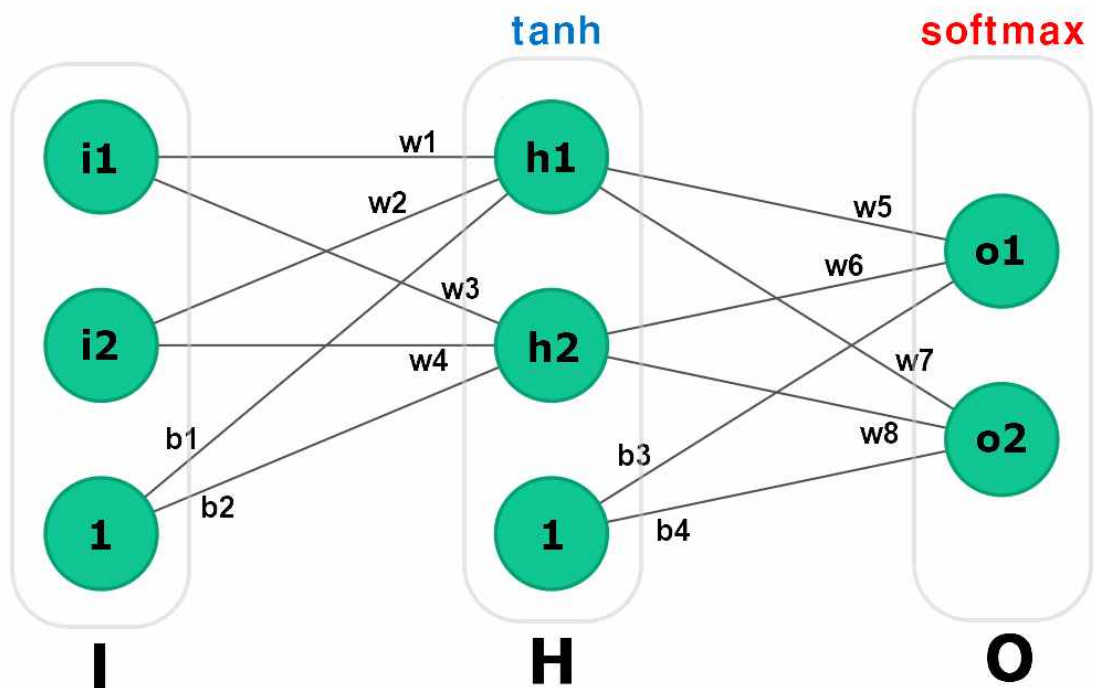
epoch = 211286
O =
[[ 0.000  1.000]]
epoch = 211287
O =
[[ 0.000  1.000]]
epoch = 211288
O =
[[ 0.000  1.000]]

```

(211288+1)번째에 오차가 0.0001(만분의 1)보다 작아집니다. o1, o2는 각각 0.000, 1.000이 된 상태입니다.

tanh와 소프트맥스

여기서는 은닉층 활성화 함수를 tanh로 변경해 봅니다. 다음 그림을 살펴봅니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

18 : $H = I @ WH + BH$


```

19 : H = np.tanh(H)
20 :
21 : O = H @ WO + BO
22 : OM = O - np.max(O)
23 : O = np.exp(OM)/np.sum(np.exp(OM))

```

19 : 은닉층 H에 순전파 tanh 활성화 함수를 적용합니다.

22, 23 : 출력층의 활성화 함수는 softmax입니다.


```

32 : Ob = O - T
33 :
34 :
35 : Hb = Ob @ WO.T
36 : Hb = Hb*(1+H)*(1-H)

```

32 : softmax 함수의 역전파 오차 계산 부분입니다.

36 : 역은닉층 Hb에 역전파 tanh 활성화 함수를 적용합니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

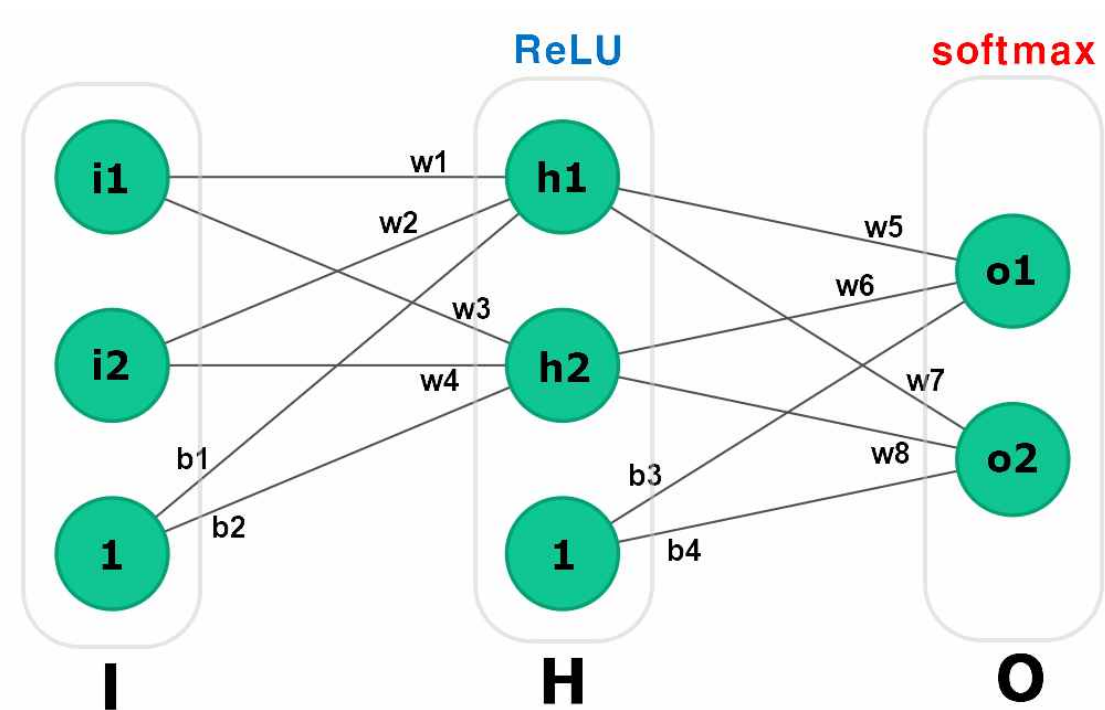
epoch = 174991
O =
[[ 0.000  1.000]]
epoch = 174992
O =
[[ 0.000  1.000]]
epoch = 174993
O =
[[ 0.000  1.000]]

```

(174993+1)번째에 오차가 0.0001(만분의 1)보다 작아집니다. o1, o2는 각각 0.000, 1.000이 된 상태입니다.

ReLU와 소프트맥스

여기서는 은닉층 활성화 함수를 ReLU로 변경해 봅니다. 다음 그림을 살펴봅니다.



1. 이전 예제를 복사합니다.
2. 다음과 같이 예제를 수정합니다.

```

18 :   H = I @ WH + BH
19 :   H = (H>0)*H
20 :
21 :   O = H @ WO + BO
22 :   OM = O - np.max(O)
23 :   O = np.exp(OM)/np.sum(np.exp(OM))

```


- 19 : 은닉층 H에 순전파 ReLU 활성화 함수를 적용합니다.
 22, 23 : 출력층의 활성화 함수는 softmax입니다.

```

32 :   Ob = O - T
33 :
34 :
35 :   Hb = Ob @ WO.T
36 :   Hb = Hb*(H>0)*1

```

- 32 : softmax 함수의 역전파 오차 계산 부분입니다.
 36 : 역은닉층 Hb에 역전파 ReLU 활성화 함수를 적용합니다.

3.  버튼을 눌러 프로그램을 실행시킵니다. 다음은 실행 결과 화면입니다.

```

epoch = 56959
O =
[[ 0.000  1.000]]
epoch = 56960
O =
[[ 0.000  1.000]]
epoch = 56961
O =
[[ 0.000  1.000]]

```

(56961+1)번째에 오차가 0.0001(만분의 1)보다 작아집니다. o1, o2는 각각 0.000, 1.000이 된 상태입니다.

이상에서 NumPy의 행렬식을 이용하여 출력층의 활성화 함수는 소프트맥스, 오차 계산 함수는 크로스 엔트로피 오차 함수인 인공 신경망을 구현해 보았습니다.

ReLU와 소프트맥스 예제 정리

다음은 마지막에 수행한 예제입니다. 뒤에서 이 예제를 계속해서 사용합니다.

```

01 : import numpy as np
02 :
03 : np.set_printoptions(formatter={'float_kind':lambda x: "{0:6.3f}".format(x)})
04 :
05 : I = np.array([[.05, .10]])
06 : T = np.array([[ 0,  1]])
07 : WH = np.array([[.15, .25],
08 :                 [.20, .30]])
09 : BH = np.array([[.35, .35]])
10 : WO = np.array([[.40, .50],
11 :                 [.45, .55]])
12 : BO = np.array([[.60, .60]])
13 :
14 : for epoch in range(10000000):
15 :
16 :     print('epoch = %d' %epoch)
17 :
18 :     H = I @ WH + BH
19 :     H = (H>0)*H

```

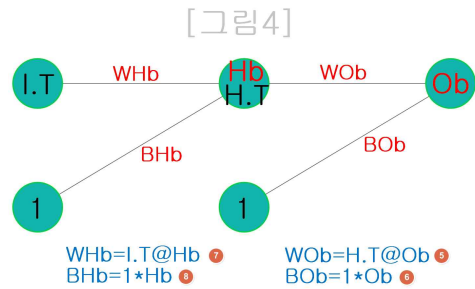
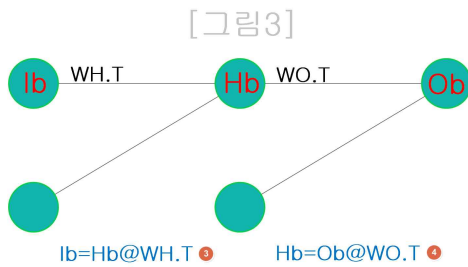
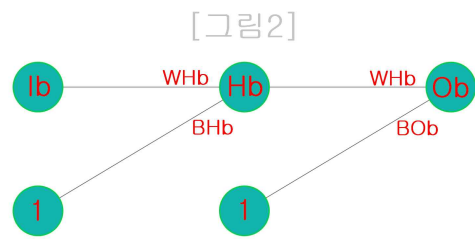
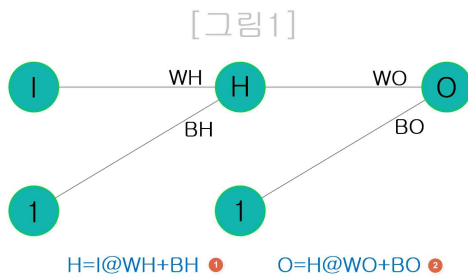
```

20 :
21 :     O = H @ WO + BO
22 :     OM = O - np.max(O)
23 :     O = np.exp(OM)/np.sum(np.exp(OM))
24 :
25 :     print(' O  =\n', O)
26 :
27 :     E = np.sum(-T*np.log(O))
28 :     if E < 0.0001:
29 :         break
30 :
31 :     Ob = O - T
32 :
33 :     Hb = Ob @ WO.T
34 :     Hb = Hb*(H>0)*1
35 :
36 :     WHb = I.T @ Hb
37 :     BHb = 1 * Hb
38 :     WOb = H.T @ Ob
39 :     BOb = 1 * Ob
40 :
41 :     lr = 0.01
42 :
43 :     WH = WH - lr * WHb
44 :     BH = BH - lr * BHb
45 :     WO = WO - lr * WOb
46 :     BO = BO - lr * BOb

```

08 인공 신경망 행렬식

여기서는 인공 신경망의 순전파 역전파를 행렬식으로 정리해 봅니다. 인공 신경망을 행렬식으로 정리하면 인공 신경망의 크기, 깊이와 상관없이 간결하게 정리할 수 있습니다. 다음 그림은 입력층 은닉층 출력층으로 구성된 인공 신경을 나타냅니다.



[그림1]은 순전파 과정에 필요한 행렬과 행렬식을 나타냅니다.

[그림2]는 역전파에 필요한 행렬입니다. 순전파에 대응되는 행렬이 모두 필요합니다.

[그림3]은 입력의 역전파에 필요한 행렬과 행렬식을 나타냅니다.

[그림4]는 가중치와 편향의 역전파에 필요한 행렬과 행렬식을 나타냅니다.

*** ③ I 층이 입력층일 경우엔 이 수식은 필요하지 않습니다.

*** @ 문자는 행렬곱을 의미합니다.

이상에서 필요한 행렬식을 정리하면 다음과 같습니다.

순전파
$H = I@WH + BH$ ①
$O = H@WO + BO$ ②
역전파 오차
$Ob = O - T$
입력 역전파
$Hb = Ob@WO.T$ ④
가중치, 편향 역전파

$$WHb = I.T@Hb \quad 7$$

$$BHb = 1 * Hb \quad 8$$

$$WO b = H.T@Ob \quad 5$$

$$BO b = 1 * Ob \quad 6$$

가중치, 편향 학습

$$WH = WH - lr * WHb$$

$$BH = BH - lr * BHb$$

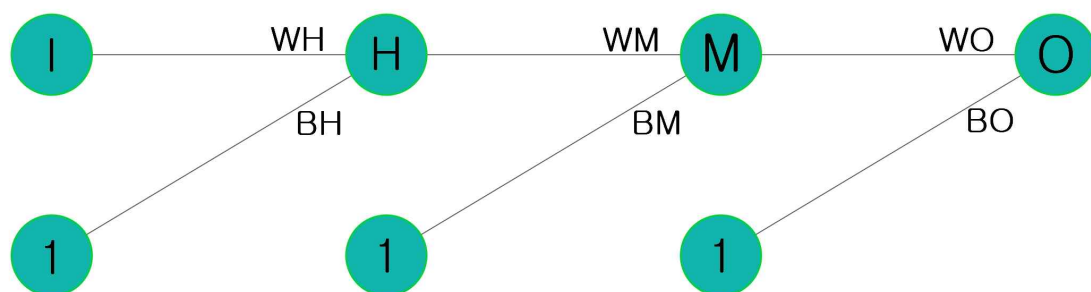
$$WO = WO - lr * WO b$$

$$BO = BO - lr * BO b$$

*** lr은 학습률을 나타냅니다.

연습문제

1. 다음은 입력I 은닉H 은닉M 출력O의 심층 인공 신경망입니다. 이 신경망에는 2개의 은닉층이 포함되어 있습니다. 일반적으로 은닉층이 2층 이상일 경우 심층 인공 신경망이라고 합니다. 이 신경망의 입력 역전파 그래프와 가중치, 편향 역전파 그래프를 그리고 순전파, 역전파 행렬식을 구합니다.



2. 다음은 입력I 은닉H 은닉M 은닉N 출력O의 심층 인공 신경망입니다. 이 신경망에는 3개의 은닉층이 포함되어 있습니다. 일반적으로 은닉층이 2층 이상일 경우 심층 인공 신경망이라고 합니다. 이 신경망의 입력 역전파 그래프와 가중치, 편향 역전파 그래프를 그리고 순전파, 역전파 행렬식을 구합니다.

