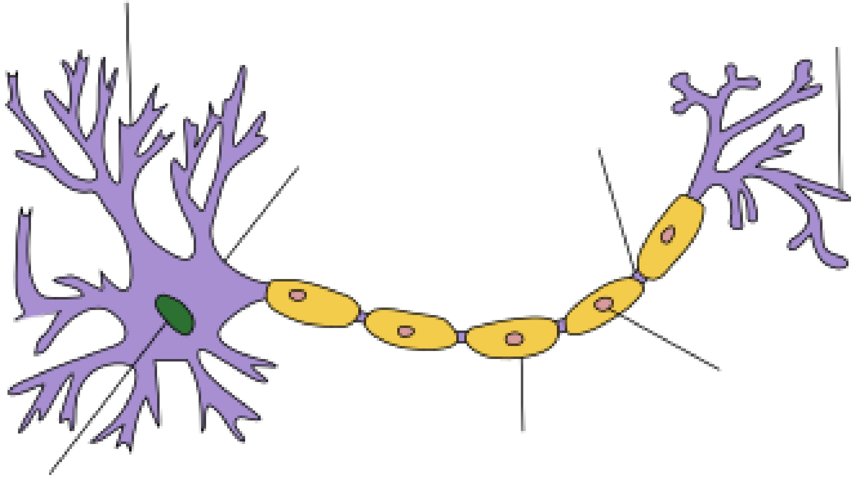
2020 SCP winter study

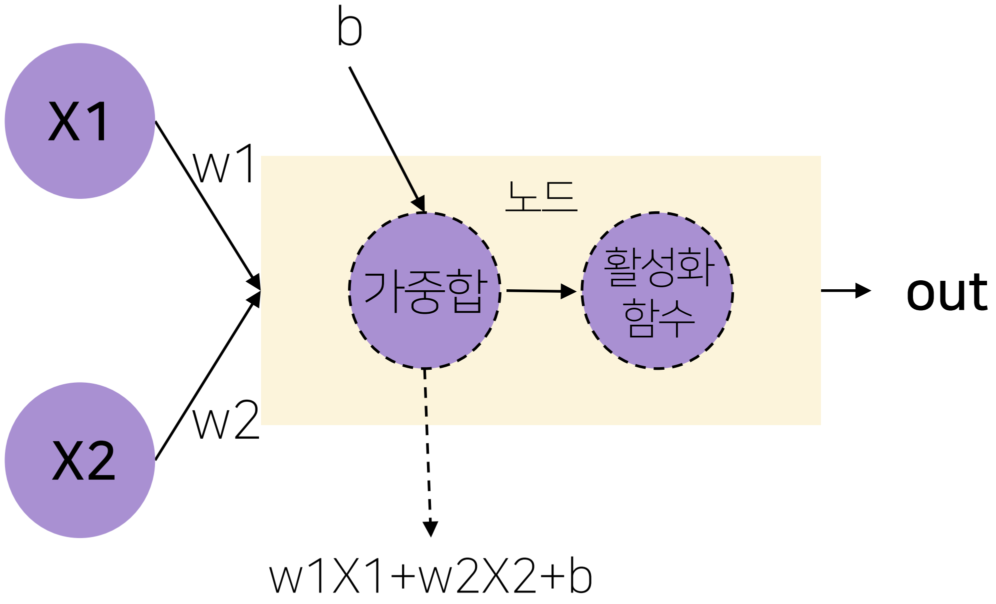
오차 역전파

Contents

1. 오차 역전파 배경

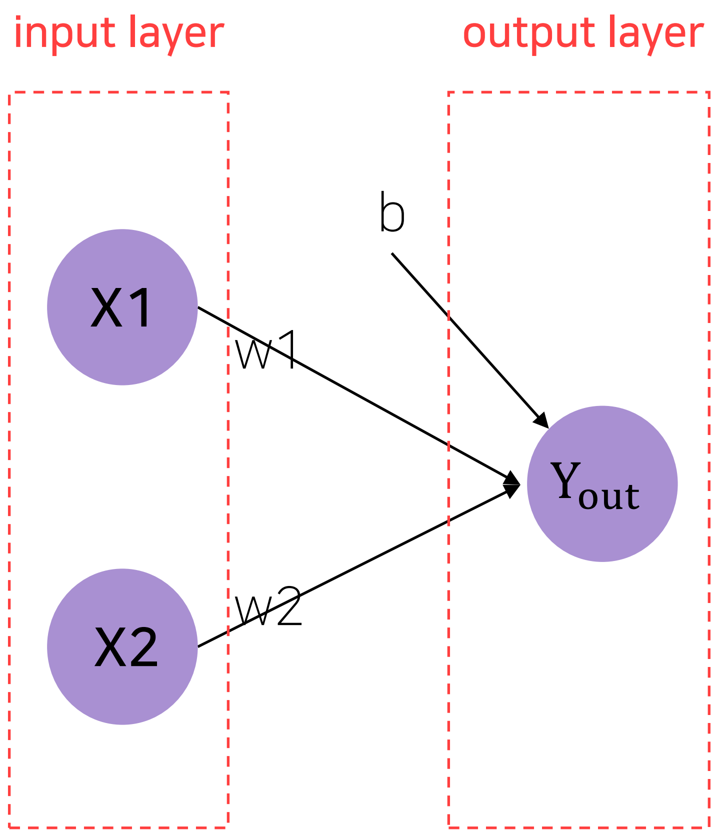


neuron

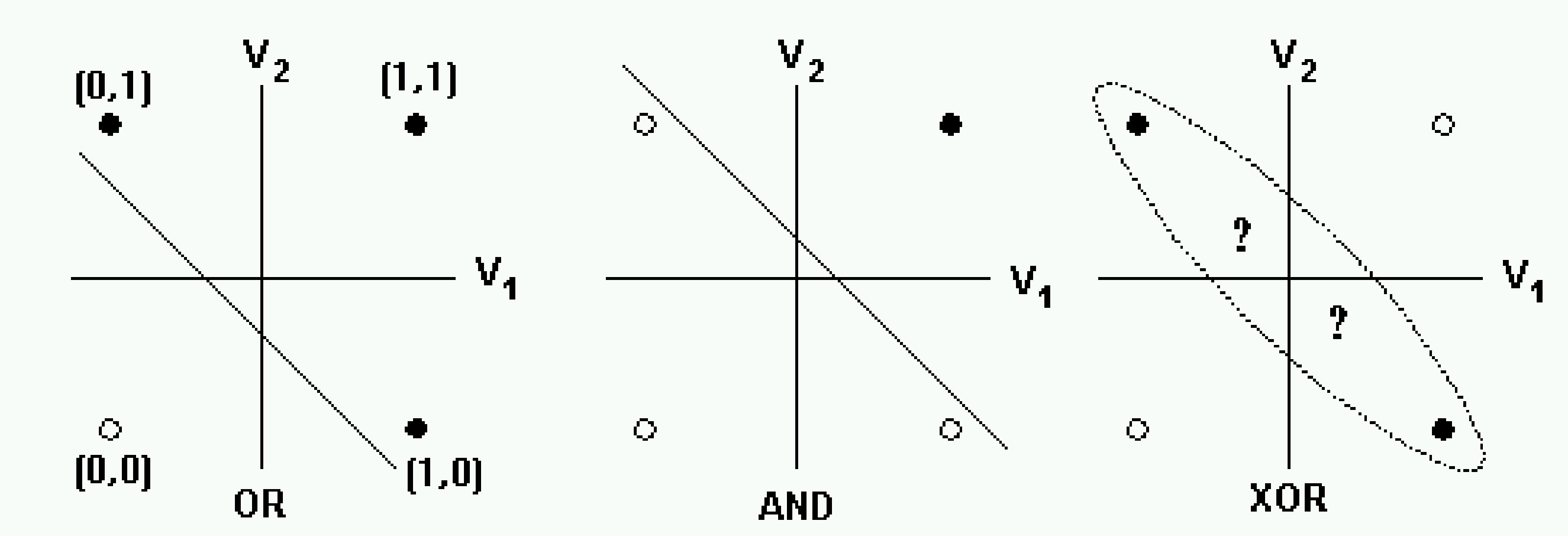


perceptron

1. 오차 역전파 배경



1. 오차 역전파 배경

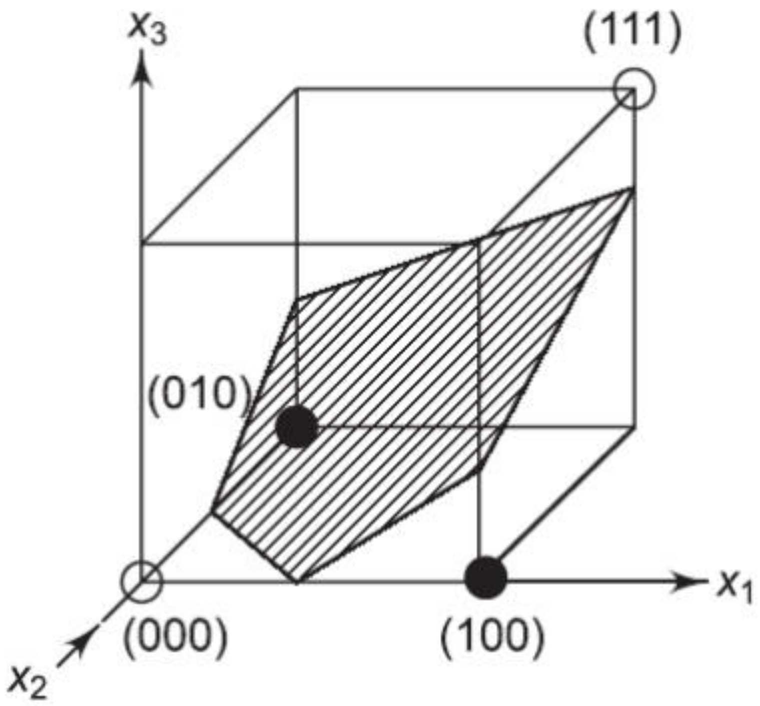


X1	X2	OUT (결과값)
0	0	0
0	1	1
1	0	1
1	1	1

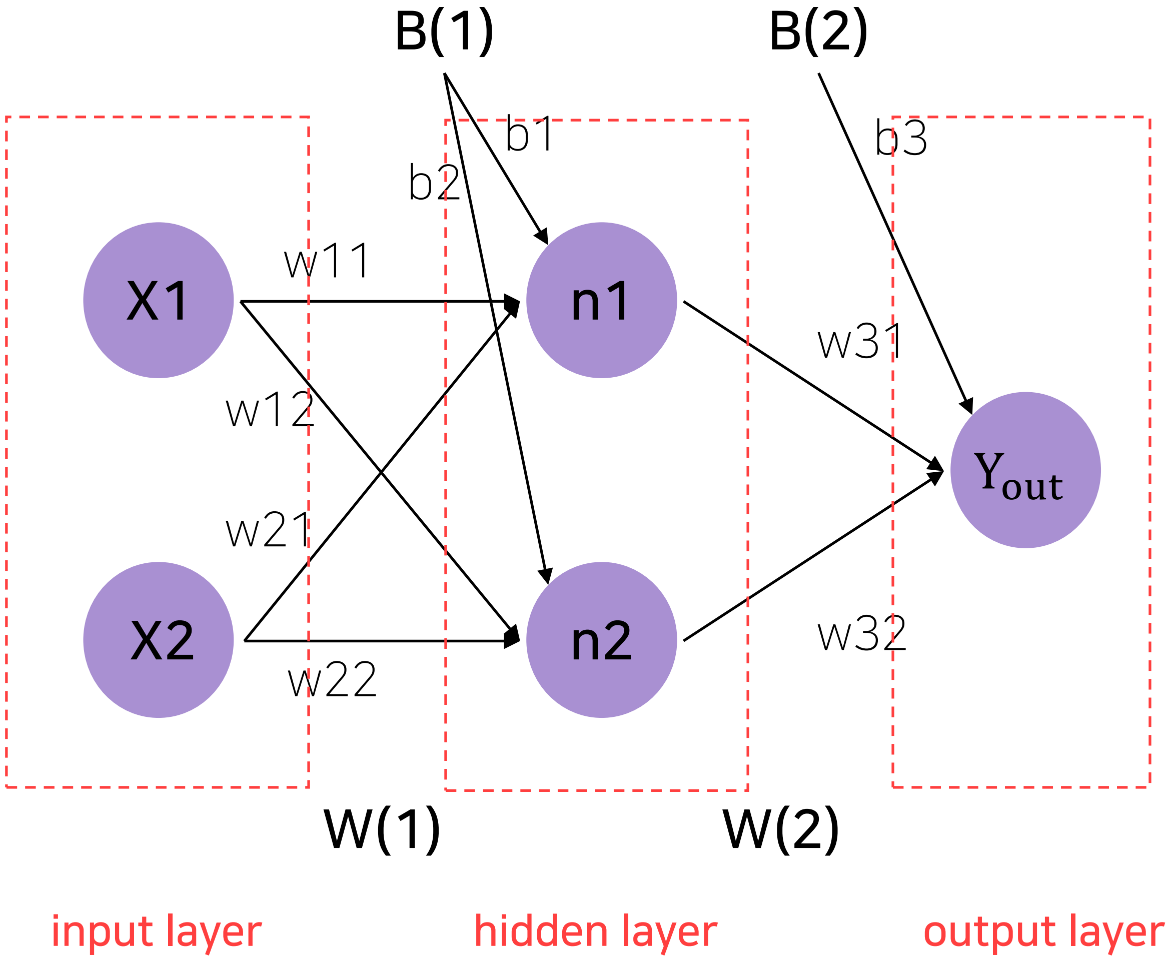
X1	X2	OUT (결과값)
0	0	0
0	1	0
1	0	0
1	1	1

X1	X2	OUT (결과값)
0	0	0
0	1	1
1	0	1
1	1	0

1. 오차 역전파 배경

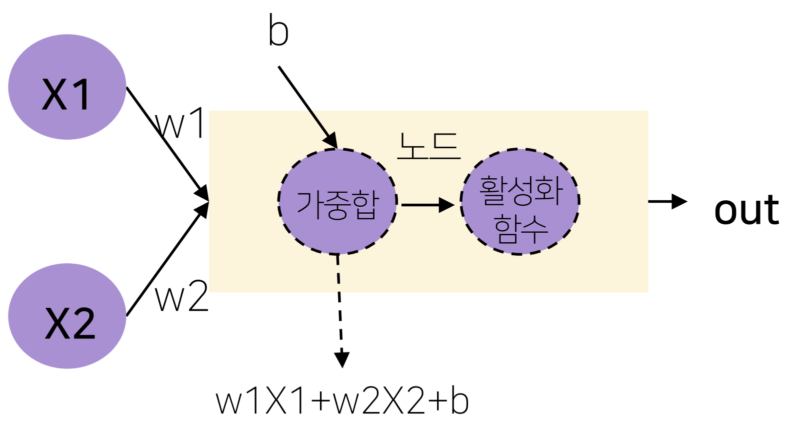


x_3
(111)
(010)
(000)
(100)
x_1
x_2

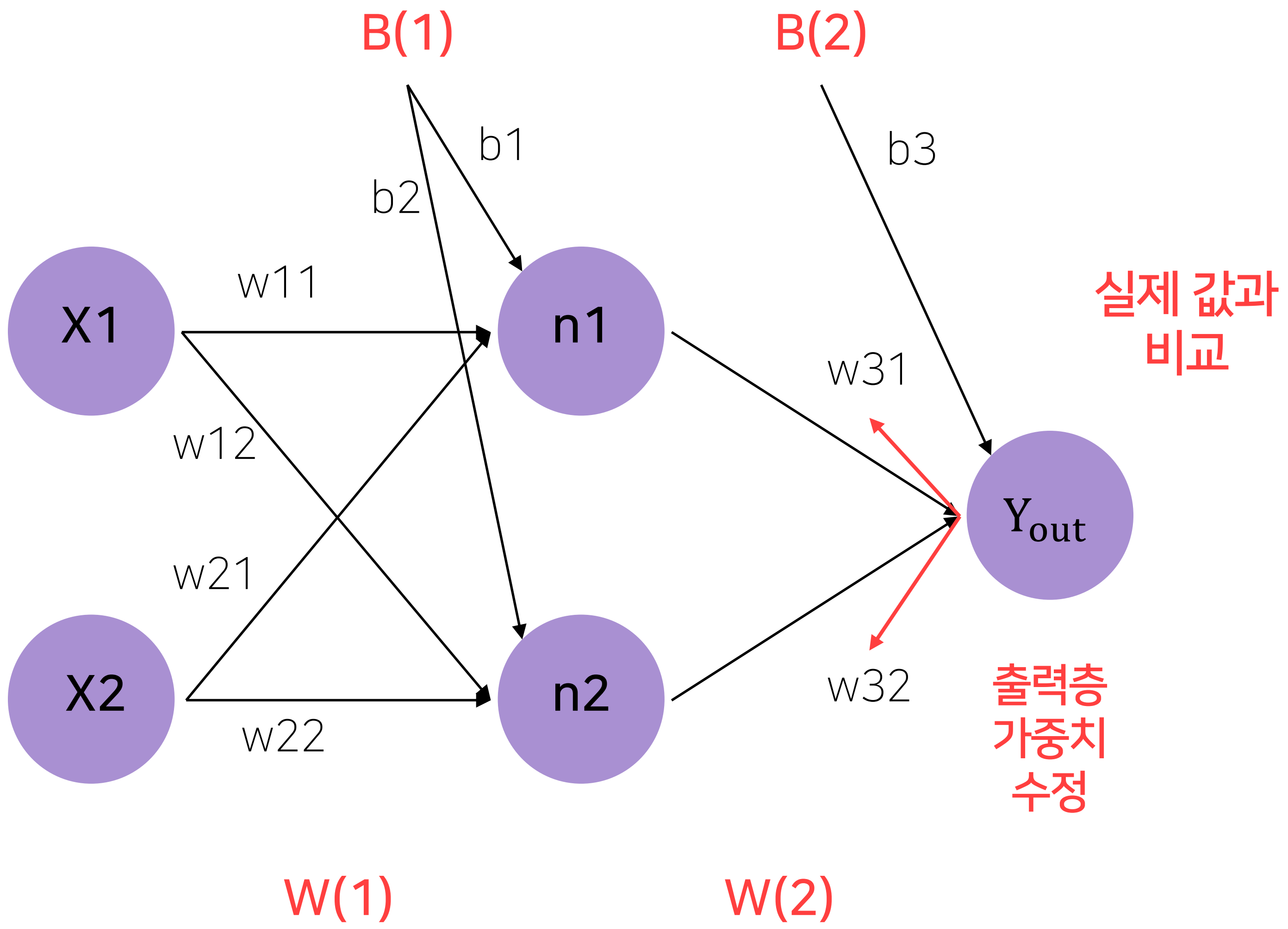


B(1)
B(2)
b1
b2
b3
X1
X2
n1
n2
Y_{out}
w11
w12
w21
w22
w31
w32
W(1)
W(2)
input layer
hidden layer
output layer

2. 오차 역전파 개념



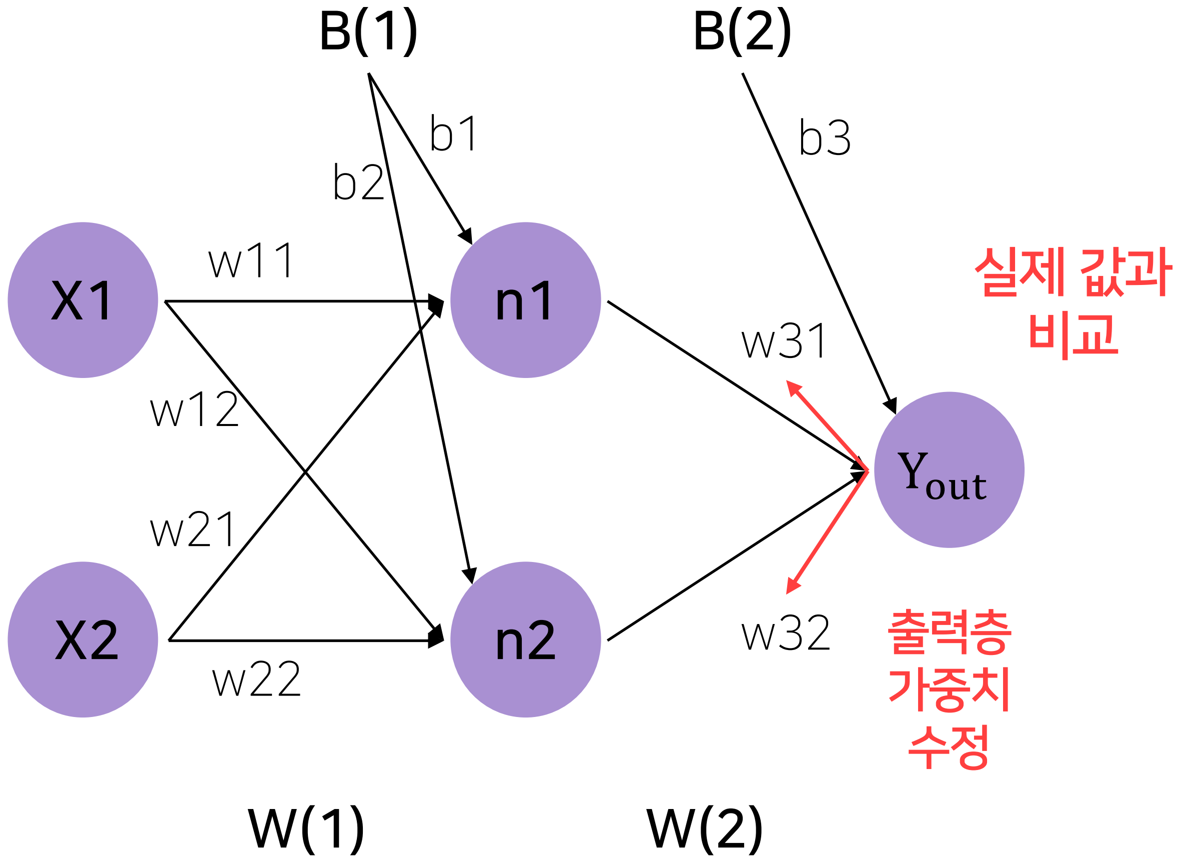
b
X1
w1
노드
가중합
활성화 함수
out
X2
w2
w1X1+w2X2+b



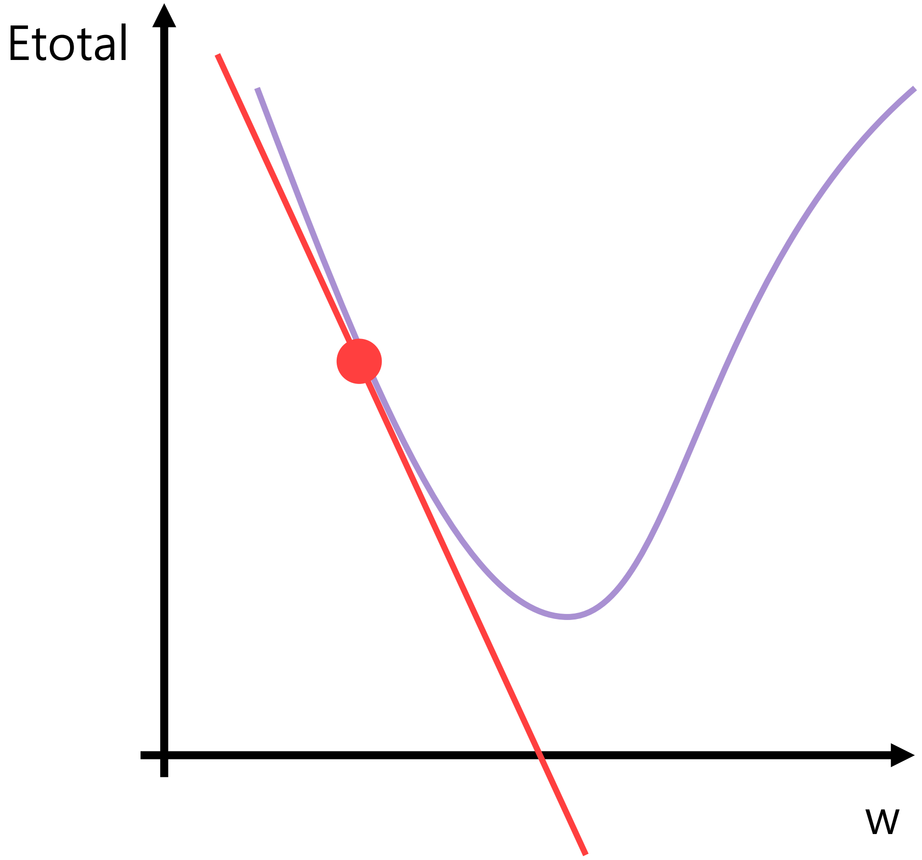
B(1)
B(2)
b1
b2
b3
w11
X1
n1
실제 값과 비교
w31
w12
Yout
w21
X2
n2
w32
출력층 가중치 수정
w22
W(1)
W(2)

2. 오차 역전파 개념

1) 임의의 초기 가중치(W)를 준 뒤 결과(Yout)를 계산
2) 계산 결과와 우리가 원하는 값 사이의 오차를 구함
3) 경사 하강법을 이용해 바로 앞 가중치를 오차가 작아지는 방향으로 업데이트
4) 위 과정을 더 이상 오차가 줄어들지 않을 때까지 반복



2. 오차 역전파 개념



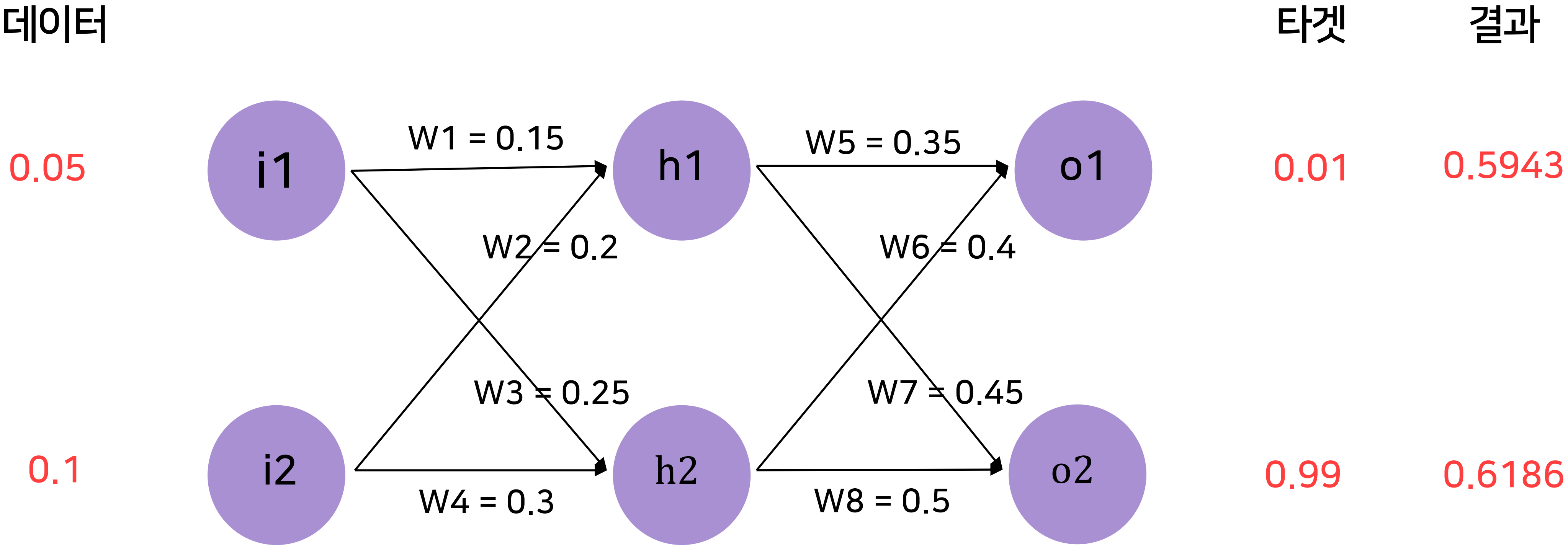
$$\frac{\partial E_{total}}{\partial w}$$

$$\alpha \frac{\partial E_{total}}{\partial w}$$

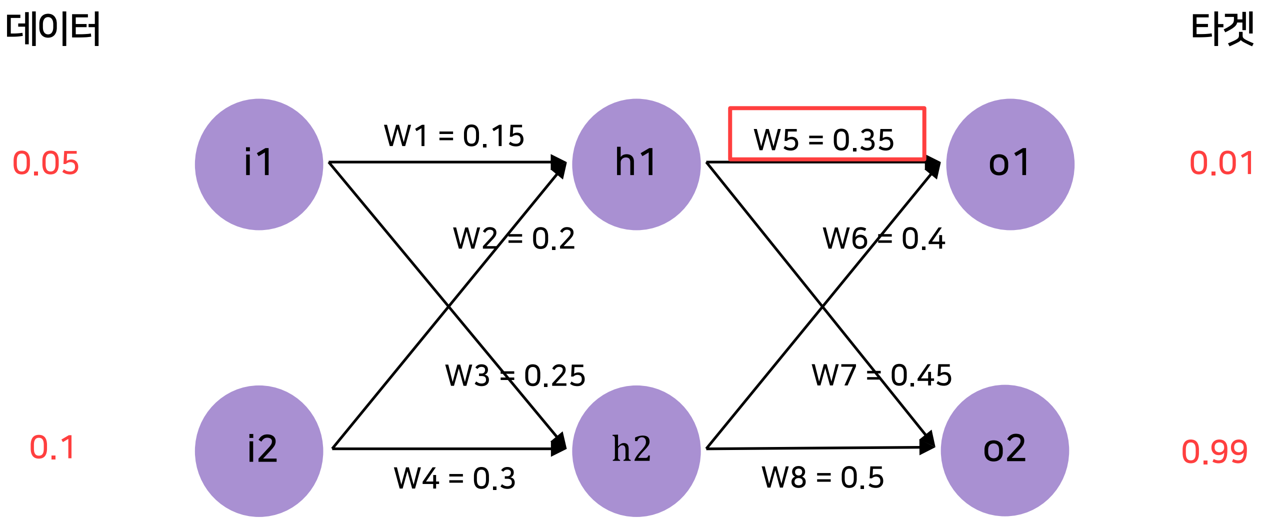
$$w(t+1) = w(t) - \alpha \frac{\partial E_{total}}{\partial w}$$

3. 오차 역전파 계산

$$w(t+1) = w(t) - \alpha \boxed{\frac{\partial E_{total}}{\partial w}}$$



3. 오차 역전파 계산



$$new\_w_5 = w_5 - \alpha \frac{\partial E_{total}}{\partial w_5}$$

$$E_{total} = E_{o1} + E_{o2}$$

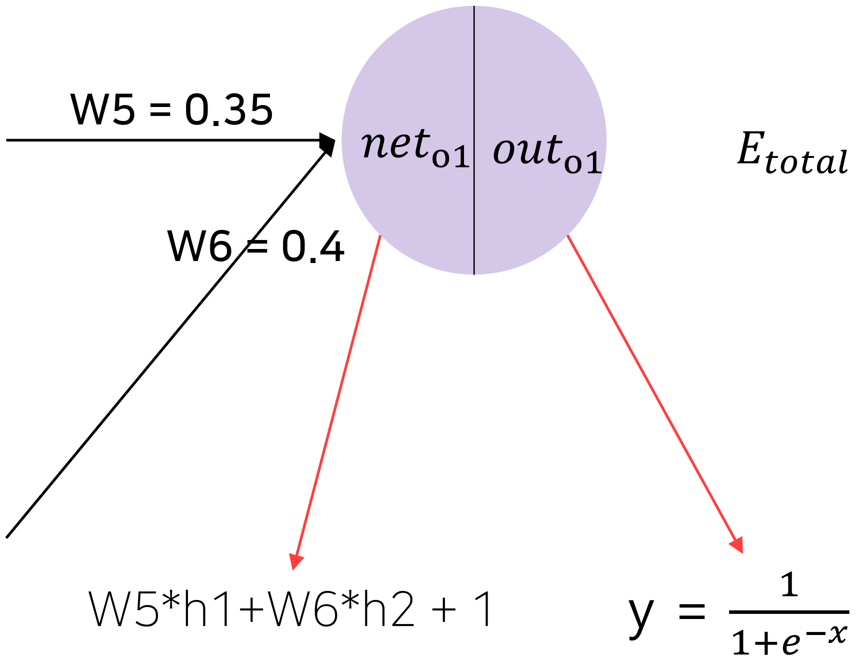
$$E_{o1} = \frac{1}{2}(target_{o1} - output_{o1})^2$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$

$$E_{o2} = \frac{1}{2}(target_{o2} - output_{o2})^2$$

3. 오차 역전파 계산

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$



3. 오차 역전파 계산

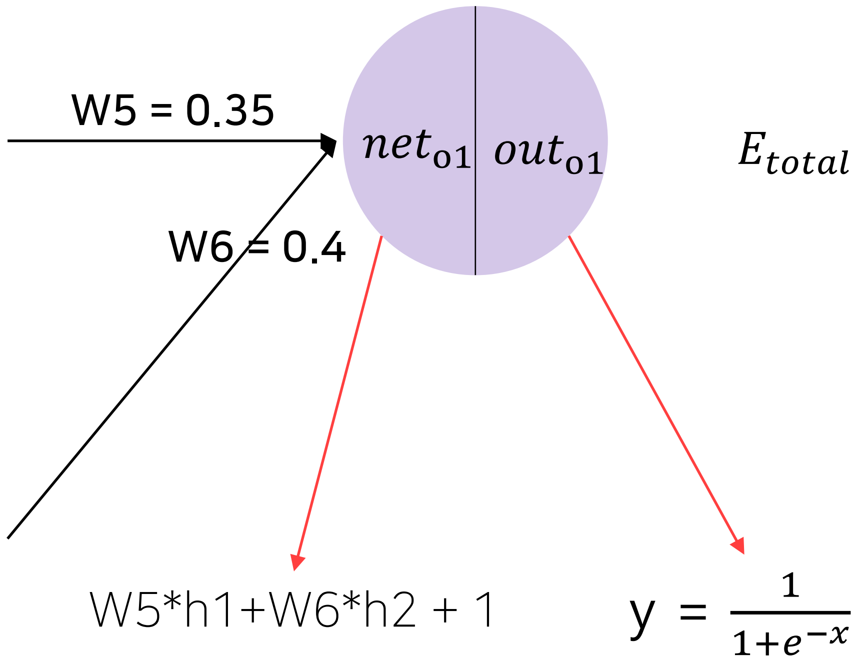
$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$

$$\begin{aligned} E_{total} &= E_{o1} + E_{o2} \\ &= \frac{1}{2}(target_{o1} - output_{o1})^2 + \frac{1}{2}(target_{o2} - output_{o2})^2 \end{aligned}$$

$$\begin{aligned} \frac{\partial E_{total}}{\partial out_{o1}} &= -(target_{o1} - out_{o1}) \\ &= -(0.01 - 0.5943) \\ &= 0.5843 \end{aligned}$$

3. 오차 역전파 계산

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$



시그모이드 함수 미분

$$\frac{dy}{dx} = y(1-y)$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1})$$

$$= 0.5943(1-0.5943)$$

$$= 0.2411$$

3. 오차 역전파 계산

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$

net_{01} = W5* out_{h1} + W6* out_{h2} + 1

$$\frac{\partial net_{01}}{\partial w_5} = out_{h1}$$

$$= 0.5069$$

3. 오차 역전파 계산

$$new\_w_5 = w_5 - \alpha \frac{\partial E_{total}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.5843 \times 0.2411 \times 0.5069 = 0.0714$$

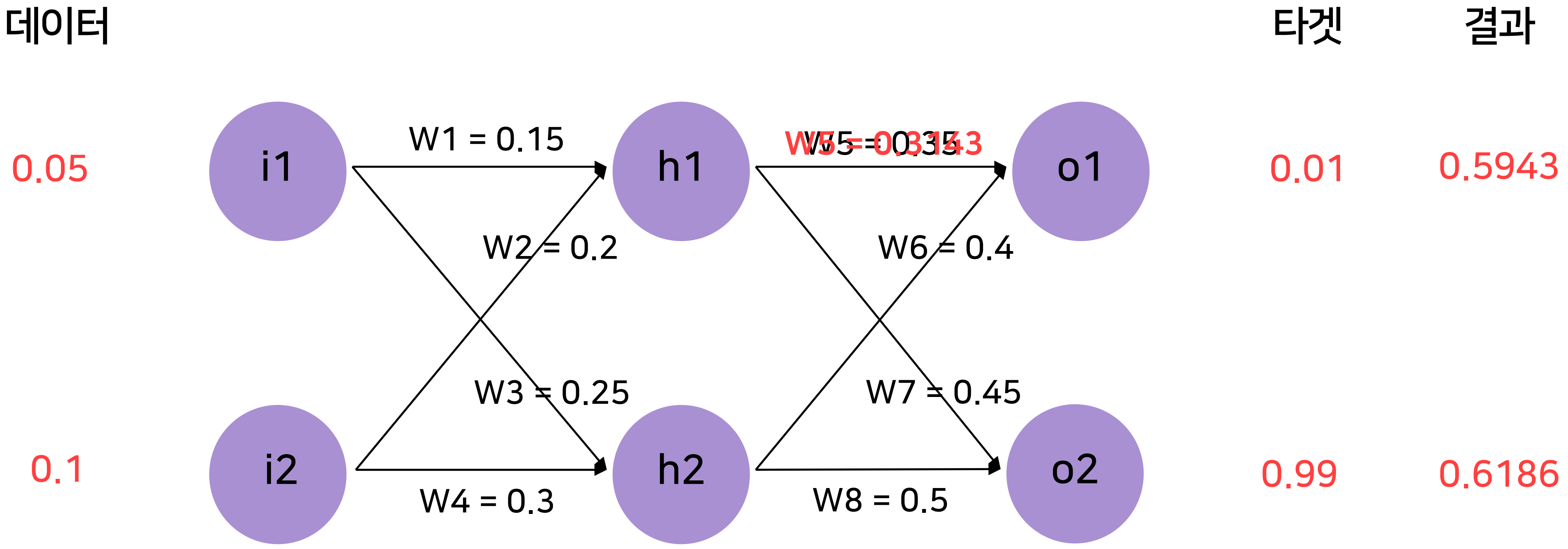
학습률 α : 0.5

$$new\_w_5 = w_5 - \alpha \frac{\partial E_{total}}{\partial w_5}$$

$$= 0.35 - 0.5 * 0.0714$$

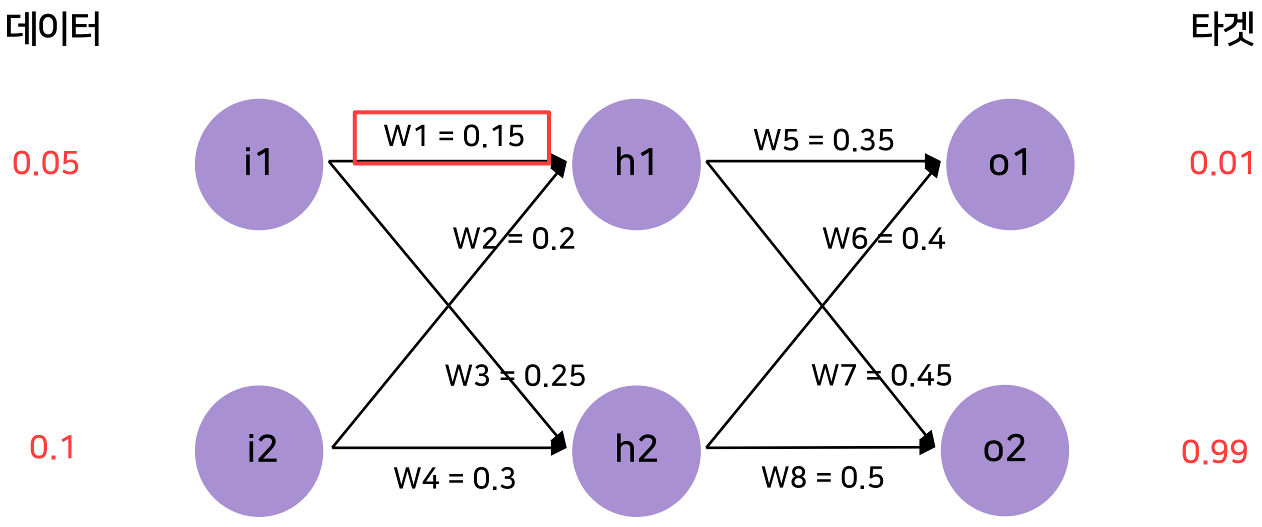
$$= 0.3143$$

3. 오차 역전파 계산



데이터
타겟
결과
0.05
i1
W1 = 0.15
h1
W5 = 0.35
W5 = 0.3143
o1
0.01
0.5943
W2 = 0.2
W6 = 0.4
W3 = 0.25
W7 = 0.45
0.1
i2
h2
o2
0.99
0.6186
W4 = 0.3
W8 = 0.5

3. 오차 역전파 계산



$$new\_w_1 = w_1 - \alpha \frac{\partial E_{total}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_1}$$

3. 오차 역전파 계산

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial \mathrm{out}_{h1}}{\partial \mathrm{net}_{h1}} = \mathrm{out}_{h1}(1 - \mathrm{out}_{h1})$$

$$= 0.25$$

$$\frac{\partial \mathrm{net}_{h1}}{\partial w_1} = i_1$$

$$= 0.05$$

3. 오차 역전파 계산

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$= \frac{\partial E_{o1}}{\partial net_{o1}} \times \frac{\partial net_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial net_{o2}} \times \frac{\partial net_{o2}}{\partial out_{h1}}$$

$$= \left(\frac{\partial E_{o1}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial net_{h1}}\right) \times \frac{\partial net_{o1}}{\partial out_{h1}} + \left(\frac{\partial E_{o2}}{\partial out_{o2}} \times \frac{\partial out_{o2}}{\partial net_{o2}}\right) \times \frac{\partial net_{o2}}{\partial out_{h1}}$$

$$= \{(out_{o1} - target_{o1}) \cdot out_{o1}(1 - out_{o1})\} \cdot W5 + \{(out_{o2} - target_{o2}) \cdot out_{o2}(1 - out_{o2})\} \cdot W7$$

3. 오차 역전파 계산

$$new\_w_1 = w_1 - \alpha \frac{\partial E_{total}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} \times \frac{\partial out_{h1}}{\partial net_{h1}} \times \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = [0.5843 \times 0.2411 \times 0.35 + (-0.3714) \times 0.2359 \times 0.45] \times 0.25 \times 0.05$$

$$= 0.0001235$$

$\boxed{\text{학습률 } \alpha : 0.5}$

$$new\_w_1 = w_1 - \alpha \frac{\partial E_{total}}{\partial w_1}$$

= 0.15 – 0.5 * 0. 0001235

= 0.1499

4. 오차 역전파 구현

```python
# 역전파의 실행
    def backPropagate(self, targets):

        # 델타 출력 계산
        output_deltas = [0.0] * self.num_yo
        for k in range(self.num_yo):
            error = targets[k] - self.activation_out[k]
            # 시그모이드에서 활성화 함수 선택, 미분 적용
            output_deltas[k] = sigmoid(self.activation_out[k], True) * error

        # 은닉 노드의 오차 함수
        hidden_deltas = [0.0] * self.num_yh
        for j in range(self.num_yh):
            error = 0.0
            for k in range(self.num_yo):
                error = error + output_deltas[k] * self.weight_out[j][k]
                # 시그모이드에서 활성화 함수 선택, 미분 적용
            hidden_deltas[j] = sigmoid(self.activation_hidden[j], True) * error

        # 출력 가중치 업데이트
        for j in range(self.num_yh):
            for k in range(self.num_yo):
                gradient = output_deltas[k] * self.activation_hidden[j]
                v = mo * self.gradient_in[j][k] - lr * gradient
                self.weight_in[j][k] += v
                self.gradient_out[j][k] = gradient

        # 입력 가중치 업데이트
        for i in range(self.num_x):
            for j in range(self.num_yh):
                gradient = hidden_deltas[j] * self.activation_input[i]
                v = mo*self.gradient_in[i][j] - lr * gradient
                self.weight_in[i][j] += v
                self.gradient_in[i][j] = gradient

        # 오차의 계산(최소 제곱법)
        error = 0.0
        for k in range(len(targets)):
            error = error + 0.5 * (targets[k] - self.activation_out[k]) ** 2
        return error
```

```python
# 델타 출력 계산
output_deltas = [0.0] * self.num_yo
for k in range(self.num_yo):
    error = targets[k] - self.activation_out[k]
    # 시그모이드에서 활성화 함수 선택, 미분 적용
    output_deltas[k] = sigmoid(self.activation_out[k], True) * error
```

출력층의 오차 업데이트

$$(out_{o1} - target_{o1}) \cdot out_{o1}(1 - out_{o1}) \cdot out_{h1}$$

은닉층의 오차 업데이트

$$(\delta out_{o1} \cdot out_{o1} + \delta out_{o2} \cdot out_{o2}) \cdot out_{h1}(1 - out_{h1}) \cdot i1$$

오차 · out(1 - out)

4. 오차 역전파 구현

```python
# 역전파의 실행
    def backPropagate(self, targets):

        # 델타 출력 계산
        output_deltas = [0.0] * self.num_yo
        for k in range(self.num_yo):
            error = targets[k] - self.activation_out[k]
            # 시그모이드에서 활성화 함수 선택, 미분 적용
            output_deltas[k] = sigmoid(self.activation_out[k], True) * error

        # 은닉 노드의 오차 함수
        hidden_deltas = [0.0] * self.num_yh
        for j in range(self.num_yh):
            error = 0.0
            for k in range(self.num_yo):
                error = error + output_deltas[k] * self.weight_out[j][k]
                # 시그모이드에서 활성화 함수 선택, 미분 적용
            hidden_deltas[j] = sigmoid(self.activation_hidden[j], True) * error

        # 출력 가중치 업데이트
        for j in range(self.num_yh):
            for k in range(self.num_yo):
                gradient = output_deltas[k] * self.activation_hidden[j]
                v = mo * self.gradient_in[j][k] - lr * gradient
                self.weight_in[j][k] += v
                self.gradient_out[j][k] = gradient

        # 입력 가중치 업데이트
        for i in range(self.num_x):
            for j in range(self.num_yh):
                gradient = hidden_deltas[j] * self.activation_input[i]
                v = mo*self.gradient_in[i][j] - lr * gradient
                self.weight_in[i][j] += v
                self.gradient_in[i][j] = gradient

        # 오차의 계산(최소 제곱법)
        error = 0.0
        for k in range(len(targets)):
            error = error + 0.5 * (targets[k] - self.activation_out[k]) ** 2
        return error
```


```python
# 델타 출력 계산
output_deltas = [0.0] * self.num_yo
for k in range(self.num_yo):
    error = targets[k] - self.activation_out[k]
    # 시그모이드에서 활성화 함수 선택, 미분 적용
    output_deltas[k] = sigmoid(self.activation_out[k], True) * error
```

4. 오차 역전파 구현

```python
# 역전파의 실행
    def backPropagate(self, targets):

        # 델타 출력 계산
        output_deltas = [0.0] * self.num_yo
        for k in range(self.num_yo):
            error = targets[k] - self.activation_out[k]
            # 시그모이드에서 활성화 함수 선택, 미분 적용
            output_deltas[k] = sigmoid(self.activation_out[k], True) * error

        # 은닉 노드의 오차 함수
        hidden_deltas = [0.0] * self.num_yh
        for j in range(self.num_yh):
            error = 0.0
            for k in range(self.num_yo):
                error = error + output_deltas[k] * self.weight_out[j][k]
                # 시그모이드에서 활성화 함수 선택, 미분 적용
            hidden_deltas[j] = sigmoid(self.activation_hidden[j], True) * error

        # 출력 가중치 업데이트
        for j in range(self.num_yh):
            for k in range(self.num_yo):
                gradient = output_deltas[k] * self.activation_hidden[j]
                v = mo * self.gradient_in[j][k] - lr * gradient
                self.weight_in[j][k] += v
                self.gradient_out[j][k] = gradient

        # 입력 가중치 업데이트
        for i in range(self.num_x):
            for j in range(self.num_yh):
                gradient = hidden_deltas[j] * self.activation_input[i]
                v = mo*self.gradient_in[i][j] - lr * gradient
                self.weight_in[i][j] += v
                self.gradient_in[i][j] = gradient

        # 오차의 계산(최소 제곱법)
        error = 0.0
        for k in range(len(targets)):
            error = error + 0.5 * (targets[k] - self.activation_out[k]) ** 2
        return error
```


```python
# 은닉 노드의 오차 함수
hidden_deltas = [0.0] * self.num_yh
for j in range(self.num_yh):
    error = 0.0
    for k in range(self.num_yo):
        error = error + output_deltas[k] * self.weight_out[j][k]
        # 시그모이드에서 활성화 함수 선택, 미분 적용
    hidden_deltas[j] = sigmoid(self.activation_hidden[j], True) * error
```

4. 오차 역전파 구현

```python
# 역전파의 실행
    def backPropagate(self, targets):

        # 델타 출력 계산
        output_deltas = [0.0] * self.num_yo
        for k in range(self.num_yo):
            error = targets[k] - self.activation_out[k]
            # 시그모이드에서 활성화 함수 선택, 미분 적용
            output_deltas[k] = sigmoid(self.activation_out[k], True) * error

        # 은닉 노드의 오차 함수
        hidden_deltas = [0.0] * self.num_yh
        for j in range(self.num_yh):
            error = 0.0
            for k in range(self.num_yo):
                error = error + output_deltas[k] * self.weight_out[j][k]
                # 시그모이드에서 활성화 함수 선택, 미분 적용
            hidden_deltas[j] = sigmoid(self.activation_hidden[j], True) * error

        # 출력 가중치 업데이트
        for j in range(self.num_yh):
            for k in range(self.num_yo):
                gradient = output_deltas[k] * self.activation_hidden[j]
                v = mo * self.gradient_in[j][k] - lr * gradient
                self.weight_in[j][k] += v
                self.gradient_out[j][k] = gradient

        # 입력 가중치 업데이트
        for i in range(self.num_x):
            for j in range(self.num_yh):
                gradient = hidden_deltas[j] * self.activation_input[i]
                v = mo*self.gradient_in[i][j] - lr * gradient
                self.weight_in[i][j] += v
                self.gradient_in[i][j] = gradient

        # 오차의 계산(최소 제곱법)
        error = 0.0
        for k in range(len(targets)):
            error = error + 0.5 * (targets[k] - self.activation_out[k]) ** 2
        return error
```


```python
# 출력 가중치 업데이트
for j in range(self.num_yh):
    for k in range(self.num_yo):
        gradient = output_deltas[k] * self.activation_hidden[j]
        v = mo * self.gradient_in[j][k] - lr * gradient
        self.weight_in[j][k] += v
        self.gradient_out[j][k] = gradient

# 입력 가중치 업데이트
for i in range(self.num_x):
    for j in range(self.num_yh):
        gradient = hidden_deltas[j] * self.activation_input[i]
        v = mo*self.gradient_in[i][j] - lr * gradient
        self.weight_in[i][j] += v
        self.gradient_in[i][j] = gradient
```

4. 오차 역전파 구현

```python
# 역전파의 실행
    def backPropagate(self, targets):

        # 델타 출력 계산
        output_deltas = [0.0] * self.num_yo
        for k in range(self.num_yo):
            error = targets[k] - self.activation_out[k]
            # 시그모이드에서 활성화 함수 선택, 미분 적용
            output_deltas[k] = sigmoid(self.activation_out[k], True) * error

        # 은닉 노드의 오차 함수
        hidden_deltas = [0.0] * self.num_yh
        for j in range(self.num_yh):
            error = 0.0
            for k in range(self.num_yo):
                error = error + output_deltas[k] * self.weight_out[j][k]
                # 시그모이드에서 활성화 함수 선택, 미분 적용
            hidden_deltas[j] = sigmoid(self.activation_hidden[j], True) * error

        # 출력 가중치 업데이트
        for j in range(self.num_yh):
            for k in range(self.num_yo):
                gradient = output_deltas[k] * self.activation_hidden[j]
                v = mo * self.gradient_in[j][k] - lr * gradient
                self.weight_in[j][k] += v
                self.gradient_out[j][k] = gradient

        # 입력 가중치 업데이트
        for i in range(self.num_x):
            for j in range(self.num_yh):
                gradient = hidden_deltas[j] * self.activation_input[i]
                v = mo*self.gradient_in[i][j] - lr * gradient
                self.weight_in[i][j] += v
                self.gradient_in[i][j] = gradient

        # 오차의 계산(최소 제곱법)
        error = 0.0
        for k in range(len(targets)):
            error = error + 0.5 * (targets[k] - self.activation_out[k]) ** 2
        return error
```

```python
# 오차의 계산(최소 제곱법)
error = 0.0
for k in range(len(targets)):
    error = error + 0.5 * (targets[k] - self.activation_out[k]) ** 2
return error
```

4. 오차 역전파 구현

```python
# 역전파의 실행
    def backPropagate(self, targets):

        # 델타 출력 계산
        output_deltas = [0.0] * self.num_yo
        for k in range(self.num_yo):
            error = targets[k] - self.activation_out[k]
            # 시그모이드에서 활성화 함수 선택, 미분 적용
            output_deltas[k] = sigmoid(self.activation_out[k], True) * error

        # 은닉 노드의 오차 함수
        hidden_deltas = [0.0] * self.num_yh
        for j in range(self.num_yh):
            error = 0.0
            for k in range(self.num_yo):
                error = error + output_deltas[k] * self.weight_out[j][k]
                # 시그모이드에서 활성화 함수 선택, 미분 적용
            hidden_deltas[j] = sigmoid(self.activation_hidden[j], True) * error

        # 출력 가중치 업데이트
        for j in range(self.num_yh):
            for k in range(self.num_yo):
                gradient = output_deltas[k] * self.activation_hidden[j]
                v = mo * self.gradient_in[j][k] - lr * gradient
                self.weight_in[j][k] += v
                self.gradient_out[j][k] = gradient

        # 입력 가중치 업데이트
        for i in range(self.num_x):
            for j in range(self.num_yh):
                gradient = hidden_deltas[j] * self.activation_input[i]
                v = mo*self.gradient_in[i][j] - lr * gradient
                self.weight_in[i][j] += v
                self.gradient_in[i][j] = gradient

        # 오차의 계산(최소 제곱법)
        error = 0.0
        for k in range(len(targets)):
            error = error + 0.5 * (targets[k] - self.activation_out[k]) ** 2
        return error
```

2020 SCP winter study

감사합니다