

## SECTION 03 신경망 기초

선형대수와 넘파이를 이용해 간단한 신경망을 구현해보자. 이 책에서는 신경망에 대한 자세한 설명은 생략하고 지금까지 학습한 선형대수, 넘파이, 함수 그리고 앞으로 배울 경사하강법을 이용한 신경망 구현을 설명한다.

### 1 시그모이드 함수

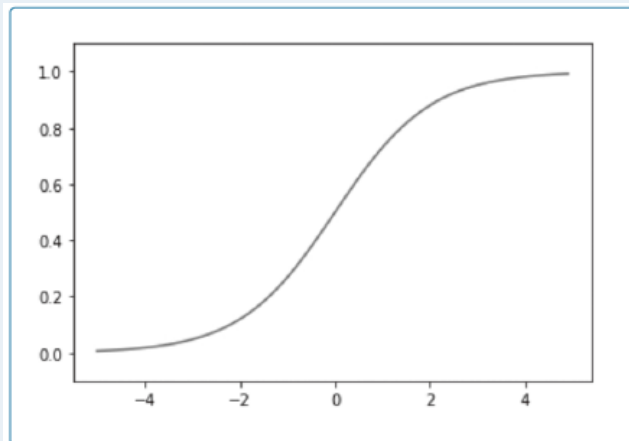
신경망을 구현하기 전에 시그모이드 함수를 먼저 알아보자. 시그모이드 함수는 은닉층에 도달한 값을 다시 은닉층이나 출력층으로 전달하기 위해 0과1 사이의 값으로 변환해주는 역할을 한다. 시그모이드 함수값을 보여주기 위해 시각화 라이브러리인 맷플롯립<sup>Matplotlib</sup>을 임포트해보자.

**TIP** 시각화 라이브러리인 맷플롯립은 시각화를 다루는 12장에서 자세히 살펴본다.

```
import numpy as np
import matplotlib.pyplot as plt

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

X=np.arange(-5.0, 5.0, 0.1)
Y=sigmoid(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)
plt.show()
```



실행 결과를 보면, -5부터 5 사이의 입력값에 대한 시그모이드 함수 출력값을 그래프로 확인할 수 있다.

### 2 순전파 신경망 구현

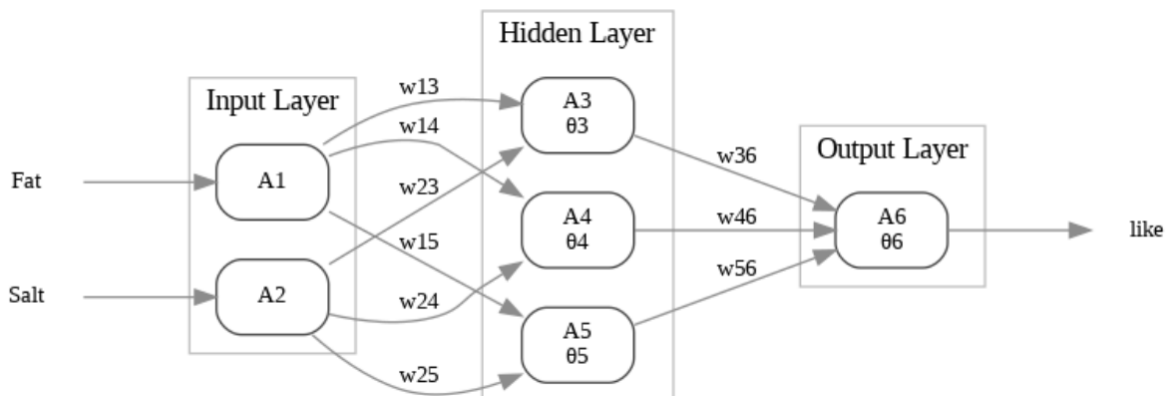
지방과 소금 성분에 따른 치즈 선호도를 나타내는 데이터를 이용해 간단한 신경망을 구현해보자. 지방 지수를 나타내는 Fat과 염분 지수를 나타내는 Salt는 독립변수, 선호도를 나타내는 Acceptance는 종속변수로 사용하며 선형대수를 이용해 신경망을 설명한다.

[표 9-1] 지방과 염분 지수에 따른 치즈 선호도

Obs.	Fat	Salt	Acceptance
1	0.2	0.9	like
2	0.1	0.1	dislike
3	0.2	0.4	dislike
4	0.2	0.5	dislike
5	0.4	0.5	like
6	0.3	0.8	like

**TIP** <Data Mining for Business Analytics: Concepts, Techniques and Applications in Python by Galit Shmueli>의 예제 데이터다.

2\*3\*1 구조의 간단한 신경망을 예로 살펴보자. 이 신경망은 총 3개 계층으로 이루어지며 왼쪽부터 입력층 Input Layer, 은닉층 Hidden Layer, 출력층 Output Layer 구조다. 각 층의 노드에서 다음 층의 노드로 계산된 값을 이 동시킨다.



[그림 9-1] 2\*3\*1 신경망의 예

## 리스트를 활용한 순전파 신경망 계산

먼저 리스트를 이용하여 입력층에서 은닉층, 그리고 출력층으로 가중치가 적용되어 출력되는 과정을 살펴 보자. 입력값을 변수 input에 저장하고 입력층에서 은닉층으로 이동하는 가중치는 변수 weight1에 저장한다. 은닉층에는 3개의 노드가 있으므로 바이어스<sup>bias</sup>를 나타내는 변수 bias1에는 3개의 값을 저장한다. 은닉층에서 출력층으로 가는 가중치는 변수 weight2에 저장하고, 출력층으로 가는 bias 노드의 가중치는 변수 bias2에 저장한다.

```
input=[0.2, 0.9] # 입력층의 2개의 값
weight1=[[0.05, 0.01], [-0.01, 0.03], [0.02, -0.01]] # 입력층에서 은닉층으로 가는 가중치
bias1=[-0.3, 0.2, 0.05] # 은닉층으로 가는 바이어스
weight2=[0.01,0.05,0.015] # 은닉층에서 출력층으로 가는 가중치
bias2=[-0.015] # 출력층으로 가는 바이어스
```

## ■ 입력층에서 은닉층으로 신호 전달

다음은 입력값과 가중치를 곱하고 바이어스 값을 더하여 입력층의 첫 번째 노드(A1)에서 은닉층의 첫 번째 노드(A3)로 신호를 전달하는 내용이다.

```
# 입력층의 첫 번째 노드에서 은닉층의 첫 번째 노드로 신호 전달
def dot(v, w):
    return sum(i * j for i, j in zip(v, w))

dot(input,weight1[0])+bias1[0] #0.2*0.05 +0.9*0.01 + (-0.3)
```

```
-0.28099999999999997
```

변수 weight1에 저장된 가중치와 바이어스 값을 이용해 입력층의 첫 번째 노드(A1)에서 은닉층의 첫 번째 노드(A3), 두 번째 노드(A4), 세 번째 노드(A5)로 보내는 신호값을 다음과 같이 A3, A4, A5로 각각 저장한다.

```
# 입력층에서 은닉층 노드로 보내는 신호값
A3=dot(input,weight1[0])+bias1[0]
A4=dot(input,weight1[1])+bias1[1]
A5=dot(input,weight1[2])+bias1[2]
A3, A4, A5
```

```
(-0.28099999999999997, 0.225, 0.045)
```

시그모이드 함수를 이용해 은닉층 노드에 전달된 값을 출력층으로 전달하기 위한 값을 생성한다.

```
#시그모이드 함수를 이용해 A3, A4, A5 노드에서 출력하는 값
import numpy as np
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
outputA3=sigmoid(A3)
outputA4=sigmoid(A4)
outputA5=sigmoid(A5)
outputA3, outputA4, outputA5
```

```
(0.4302086298074603, 0.5560138905446199, 0.5112481019468548)
```

## ■ 은닉층에서 출력층으로 신호 전달

이제 은닉층의 값을 출력층으로 보내고 최종 출력층의 값이 어떻게 되는지 확인해보자.

```
Y1=[outputA3, outputA4, outputA5]
A6=dot(Y1, weight2)+bias2
outputA6=sigmoid(A6)
outputA6
```

```
array([0.50619256])
```

## 넘파이를 활용한 순전파 신경망 계산

지금까지의 계산을 넘파이를 이용하면 매우 편리하게 진행할 수 있다. 입력층에서 은닉층으로 전달되는 값 A1을 구하고 시그모이드 함수를 이용하여 출력층으로 전달하기 위한 값 Y1을 구한다.

```
# 입력층에서 첫 번째 은닉층으로 신호 전달 방법
import numpy as np
input=np.array([0.2, 0.9]) # 1*2 행렬
weight1=np.array([[0.05,0.01], [-0.01,0.03], [0.02,-0.01]]) # 3*2 행렬
bias1=np.array([-0.3,0.2,0.05])
A1=np.dot(input, weight1.T) + bias1 # weight1.T는 weight의 전치 행렬
Y1=sigmoid(A1)
Y1
```

```
array([0.43020863, 0.55601389, 0.5112481])
```

다음은 은닉층의 값 Y1에 가중치와 바이어스 값을 적용해 출력층으로 전달하는 과정이다. 이 값은 A2에 저장하고 시그모이드 함수를 이용해 출력층에서 최종 산출하는 값을 Y2에 저장한다.

```
weight2=np.array([0.01, 0.05, 0.015])
bias2=np.array([-0.015])
A2=np.dot(Y1, weight2.T)+bias2
Y2=sigmoid(A2)
Y2
```

```
array([0.50619256])
```

보통 마지막 출력층에서는 시그모이드 함수 대신 항등 함수를 사용하지만 이 책에서는 편의상 최종 출력 값으로 시그모이드 함수를 사용한 값을 이용하기로 한다. 항등 함수를 사용할 때 출력층의 값은 다음과 같다.

```
def identity_function(x):
    return x
Y=identity_function(A2)          # 혹은 Y=A2
Y
```

```
array([0.0247715])
```

## ■ 순전파 구현 정리

지금까지 살펴본 순전파<sup>feedforward</sup> 신경망을 함수로 정의해보자.

```
def init_network():
    network={}
    network['W1']=np.array([[0.05,0.01], [-0.01,0.03], [0.02,-0.01]])
    network['b1']=np.array([-0.3,0.2,0.05])
    network['W2']=np.array([0.01, 0.05, 0.015])
    network['b2']=np.array([-0.015])
    return network

def forward(network, x):
    W1, W2=network['W1'], network['W2']
    b1, b2=network['b1'], network['b2']
    a1=np.dot(x, W1.T) + b1
    y1=sigmoid(a1)
    a2=np.dot(y1, W2.T) + b2
    y2=sigmoid(a2)          # 시그모이드 함수 사용
    # z2=identity_function(a2)  # 항등 함수 사용
```

```

return y2

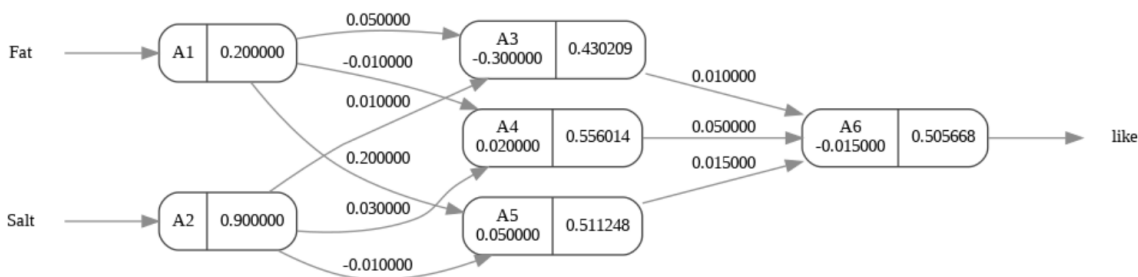
network=init_network()
input=np.array([0.2, 0.9])
Y=forward(network, input)
Y

```

array([0.50619256])

init\_network 함수를 정의하여 가중치와 바이어스(편향) 값을 초기화한 후 딥서너리 변수인 network에 저장한다. forward 함수는 입력 신호를 출력으로 변환하는 처리 과정을 모두 구현한 것으로 입력층에서 출력층으로 신호가 전달되는 것을 의미한다(출력층에서 항등 함수를 사용한 경우는 주석으로 처리해두었다). 이는 순전파 신경망으로부터 입력값이 [0.2, 0.9]의 값만 전달했을 때의 과정으로 지금까지 배운 선형대수의 행렬 연산으로 해결할 수 있다.

예제에서 보듯이 입력값을 전부 넣고 출력값은 각각 like, dislike를 1, 0으로 바꾸고 입력값에 따른 출력값이 정확하게 맞아떨어지도록 노드에서 노드로 이동하는 가중치를 보정하는 것이 신경망의 원리다. 이때 입력값을 한 개씩 전달할 때마다 보정을 하는 방식과 전체 입력값을 배치 방식으로 한꺼번에 전달한 후에 보정을 하는 방식이 있다. 보정을 하는 것을 epoch라고 하며 이 과정을 거치는 것이 역전파<sup>Backpropagation</sup> 신경망이다. 가중치를 조정하는 방식에는 경사하강법을 자주 사용한다. 일단 이상의 신호(값)이 전달되는 내용을 그림으로 나타내면 다음과 같다.



[그림 9-2] 순전파 신경망에서 전달되는 값

### 3 역전파 신경망 구현

선형대수와 간단한 미분을 이용하여, 하나의 입력 데이터가 순전파 신경망을 통해 출력된 값과 실제값을 비교하여 노드 간 가중치를 조정하는 역전파<sup>Backpropagation</sup> 신경망에 대해 살펴본다.

먼저, 실제값은 like와 unlike로 되어 있으므로 like는 1, unlike는 0으로 바꾸고 출력층의 출력값이 실제값과 유사해지도록 노드 간 가중치를 조정하는 과정을 살펴본다. 출력값을  $y$ , 실제값을  $t$ 라고 하면  $t-y$ 는 실제값과 예측값의 차이가 된다. 이 차이를 에러<sup>Error</sup>라고 하며, 통상적으로 오차<sup>MSE, Mean Squared Error</sup> 함수

를 많이 이용하므로, MSE 값을 예로 들어 설명한다. 실제값이 1일 때, 에러와 MSE 값은 아래와 같이 계산된다.

```
t,y=1, Y
Error=t-y
def MSE(t, y):
    return 0.5 * np.sum((t-y)**2)
Error, MSE(t,y)
```

```
(array([0.49380744]), 0.1219228944272693)
```

이제 가중치와 바이어스 값을 보정하는 과정을 살펴보자. MSE 함수를 E로 표시하여 설명한다. 은닉층에서 출력층으로 가는 신호를 아래첨자 2로 구분하여 노드 간 가중치는  $w_2$ , 바이어스 값은  $b_2$ , 은닉층에서 출력층에 도달한 값은  $a_2$ , 출력층에서 시그모이드 함수를 통해 산출된 값은  $y_2$ 로 하자. 가중치가 E에 어떻게 영향을 미치는지 체인룰<sup>Chain Rule</sup>을 이용해 계산할 수 있다. 이는  $\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_2}$ 로 계산된다. 출력층에 도달된 값  $a_2 = w_2 x + b_2$ 이며  $a_2$ 는  $w_2$ 의 함수이므로  $w_2$ 가 MSE 값에 영향을 미친다.

왼쪽의  $\frac{\partial a_2}{\partial w_2} = \frac{\partial (w_2 x + b_2)}{\partial w_2} = x$ 이고 우측의  $\frac{\partial E}{\partial a_2} = \frac{\partial E}{\partial y_2} \cdot \frac{\partial y_2}{\partial a_2} = \delta_2$ 이다. 여기서  $x$ 는 은닉층에서 산출된  $y_1$  값이며, 특히 우측의  $\frac{\partial E}{\partial a_2}$ 를 통상  $\delta_2$ 로 부른다.  $\delta_2$ 는 오차(MSE)를 출력값( $y_2$ )으로 편미분한 값과, 출력값( $y_2$ )을  $a_2$ 로 편미분한 값(활성화 함수의 미분값)의 곱이다.

오차는 일반적으로  $E = \frac{1}{2} \sum (t - y)^2$ 을 주로 사용하기 때문에, 이 MSE 공식을 이용하면  $\delta_2$ 의 앞부분  $\frac{\partial E}{\partial y} = \frac{\partial}{\partial y} \left( \frac{1}{2} (t - y)^2 \right) = -(t - y)$ 이 된다.  $\delta_2$ 의 뒷부분  $\frac{\partial y_2}{\partial a_2}$ 는 활성화 함수(예: 시그모이드 함수)를 편미분하여 구할 수 있다. 예제에서 사용하고 있는 시그모이드 함수  $y(a) = \frac{1}{1 + e^{-a}}$ 의 도함수는  $y'(a) = (1 - y)y$ 이다. 이를 이용하면  $\frac{\partial y_2}{\partial a_2} = (1 - y_2) \cdot y_2$ 가 된다. 출력층에서의  $\delta_2$ , 즉  $\frac{\partial E}{\partial a_2}$ 은 다음과 같이 계산된다.

$$\delta_2 = \frac{\partial E}{\partial a_2} = \frac{\partial E}{\partial y_2} \cdot \frac{\partial y_2}{\partial a_2} = -(t - y_2) \cdot (1 - y_2) \cdot y_2$$

그래서 최종적으로 가중치의 에러에 대한 기울기는 다음과 같다.

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_2} = \frac{\partial E}{\partial y_2} \cdot \frac{\partial y_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_2} = -(t - y_2) \cdot (1 - y_2)y_2 \cdot x = \delta_2 \cdot y_1$$

또한 최종적으로 바이어스 값의 에러에 대한 기울기는 다음과 같다.

$$\frac{\partial E}{\partial b_2} = \frac{\partial E}{\partial a_2} \cdot \frac{\partial a_2}{\partial b_2} = \frac{\partial E}{\partial a_2} \cdot \frac{\partial (w_2 x + b_2)}{\partial b} = \delta_2 \cdot 1 = \delta_2$$

가중치와 바이어스 값의 에러에 대한 기울기를 이용하여, 출력층에서 산출된 예측값이 실제값과 차이가 줄어들도록 하는 것을 학습이라 하며 학습을 반복하는 횟수를 epoch라고 한다. 가중치와 바이어스 값을 얼마만큼 조정할지는 학습률에 따라 달라진다. 학습률은 일반적으로 가중치 기울기와 바이어스 기울기에 0.1 이하의 값을 곱한 값을 기존의 가중치와 바이어스 값에서 에러가 줄어들도록 빼주어(보정값을 기울기와 반대 방향으로 하기 위함) 새로운 값으로 업데이트한다.

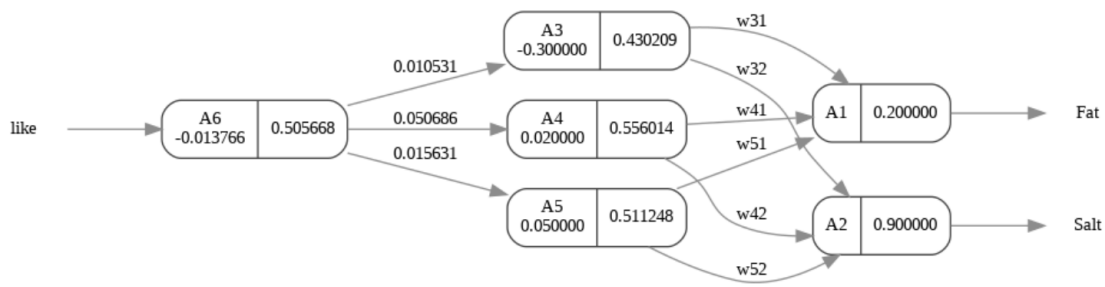
다음은 출력층에서 은닉층으로 역전파할 때 가중치와 바이어스 값을 조정하는 과정을 보여준다. MSE 값을 최소화하기 위해 기울기  $\frac{\partial E}{\partial w_2}$ ,  $\frac{\partial E}{\partial b_2}$ 의 부호와 반대로 가중치를 업데이트해야하기에 기존 가중치 값에서 음(-)의 부호를 사용하여 조정한다. 예를 들어, 이전 가중치 값에서 에러에 대한 기울기가 음수라면, 이보다 우측의 값은 MSE 값이 작아지므로 기존 값에서 양(+)의 방향으로 가중치를 조정한다. 기울기가 양수라면 좌측의 값에서 MSE 값이 작아지므로 음(-)의 방향으로 가중치를 조정한다.

```
w2_old=np.array([0.01,0.05,0.015])    # 이전의 은닉층에서 출력층 노드 간 가중치
b2_old=np.array([-0.015])              # 이전의 은닉층에서 출력층 바이어스 가중치
# 업데이트되는 가중치
d2=-(t-y)*(1-y)*y                      # -(1-0.5061)*(1-0.5061)*0.5061=-0.1234
lr=0.01                                # 학습률
# bias 업데이트
b2_new=b2_old-lr*d2*1                   # -0.015-0.1*(-0.1234)*1
# weight 업데이트
w36=w2_old[0]-lr*d2*Y1[0]               # 0.01-0.1*(-0.1234)*0.4302
w46=w2_old[1]-lr*d2*Y1[1]               # 0.05-0.1*(-0.1234)*0.5560
w56=w2_old[2]-lr*d2*Y1[2]               # 0.015-0.1*(-0.1234)*0.5112
print(b2_new, w36, w46, w56)
```

```
# 업데이트 후 가중치
[-0.01376567] [0.01053102] [0.0506863] [0.01563105]
```

이상의 결과는 [그림 9-3]으로 나타낼 수 있다. 노드 A6에 있는 값은 역전파 과정에서 조정된 바이어스 값이며, 출력층 노드와 은닉층 노드 간의 화살표 위에 있는 값은 역전파를 통해 조정된 가중치이다. 예제에서 가중치 기울기가 음(-)의 부호이므로 에러값(MSE)을 최소화하기 위해 업데이트된 가중치들은 이전 가중치에 비해 더 큰 값으로 조정된 것을 확인할 수 있다.





[그림 9-3] 출력층에서 은닉층으로 역전파 시 업데이트된 가중치 값과 바이어스 값

다음은 은닉층에서 입력층으로 역전파 시 가중치와 바이어스 값이 업데이트되는 과정이다. 앞서 사용한 표기와 일관되게, 편의상  $w_1$ 과  $b_1$ 을 입력층에서 은닉층으로 가는 가중치와 바이어스 값이라고 하겠다. 이들

이 MSE에 미치는 영향은  $\frac{\partial E}{\partial w_1} = \frac{\partial a_1}{\partial w_1} \cdot \frac{\partial E}{\partial a_1} = \frac{\partial a_1}{\partial w_1} \cdot \delta_1$ 으로 나타낼 수 있으며,

$\delta_1 = \frac{\partial E}{\partial a_1} = \frac{\partial y_1}{\partial a_1} \cdot \frac{\partial a_2}{\partial y_1} \cdot \frac{\partial E}{\partial a_2} = \frac{1}{1 + e^{-a_1}} \left( 1 - \frac{1}{1 + e^{-a_1}} \right) \cdot w_2 \cdot \delta_2 = y_1(1 - y_1) \cdot w_2 \cdot \delta_2$ 이다. 앞에서 구한  $\delta_2$  값을 이용하여 노드 3에서의  $\delta_1$ 는  $\delta_1 = (0.43021) \cdot (1 - 0.43021) \cdot (0.01) \cdot (-0.12343) = -0.00030$ 으로 계산된다.

노드 1과 노드 3의 새로운 가중치는  $\frac{\partial E}{\partial w_1} = \frac{\partial a_1}{\partial w_1} \cdot \delta_1 = input[0] \cdot \delta_1 = 0.2 \cdot (-0.0030)$ 의 값을 이용하

여  $w_{13\_new} = w_{13\_old} - lr \cdot \frac{\partial a_1}{\partial w_1} \cdot \delta_1 = (0.5) - (0.01) \cdot (0.2) \cdot (-0.0030)$ 으로 업데이트한다. 마찬가지로

다른 노드 간의 가중치도 같은 방식으로 구할 수 있으며, 바이어스 값도 같은 방식으로 구하지만 여기서는  $\frac{\partial E}{\partial b_1} = 1 \cdot \delta_1$ 이므로 input[0] 대신 1 값을 사용하여 업데이트한다.

```
dZ1_dA1=sigmoid(A1)*(1-sigmoid(A1))
d1=dZ1_dA1*weight2*d2
# 이전의 가중치
w1_old=np.array([[0.05,0.01], [-0.01,0.03], [0.02,-0.01]])
b1_old=np.array([-0.3, 0.2, 0.05])
# 업데이트되는 가중치
lr=0.01 # 학습률
input=np.array([0.2, 0.9])
b13_new=b1_old[0]-lr*d1[0]
b14_new=b1_old[1]-lr*d1[1]
b15_new=b1_old[2]-lr*d1[2]
w13=w1_old[0][0]-lr*d1[0]*input[0]
w14=w1_old[1][0]-lr*d1[1]*input[0]
w15=w1_old[2][0]-lr*d1[2]*input[0]
```

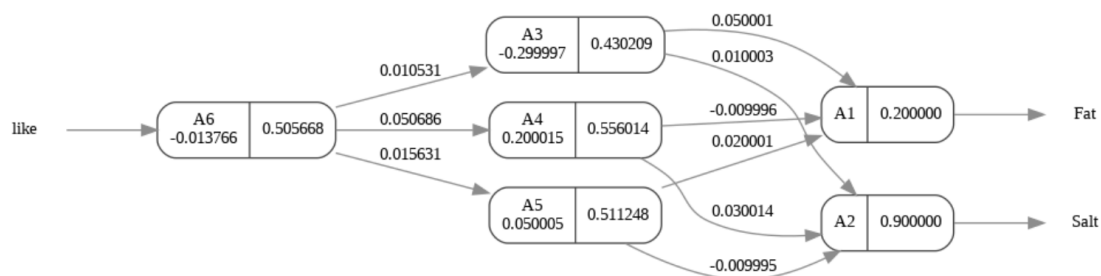
```
w23=w1_old[0][1]-lr*d1[0]*input[1]
w24=w1_old[1][1]-lr*d1[1]*input[1]
w25=w1_old[2][1]-lr*d1[2]*input[1]
print(b13_new,b14_new,b15_new)
print(w13,w14,w15,w23,w24,w25)
```

# 업데이트 후 가중치

-0.299997 0.200015 0.050004

0.050001 -0.009996 0.020001 0.010003 0.030014 -0.009995

[그림 9-4]는 은닉층에서 입력층으로 역전파되어 업데이트된 가중치까지 반영하여 보여주고 있다.



[그림 9-4] 은닉층에서 입력층으로 역전파 시 업데이트된 가중치 값과 바이어스 값

이렇게 순전파와 역전파를 한 번 거치는 것을 1 epoch라고 하며, 여러 번의 epoch를 통해 에러가 원하는 수준보다 더 이상 줄어들지 않게 되면 학습을 중단한다. 이때 최종적으로 업데이트된 가중치를 이용하여 신경망을 통해 원하는 예측 혹은 분류 문제를 해결하게 된다.