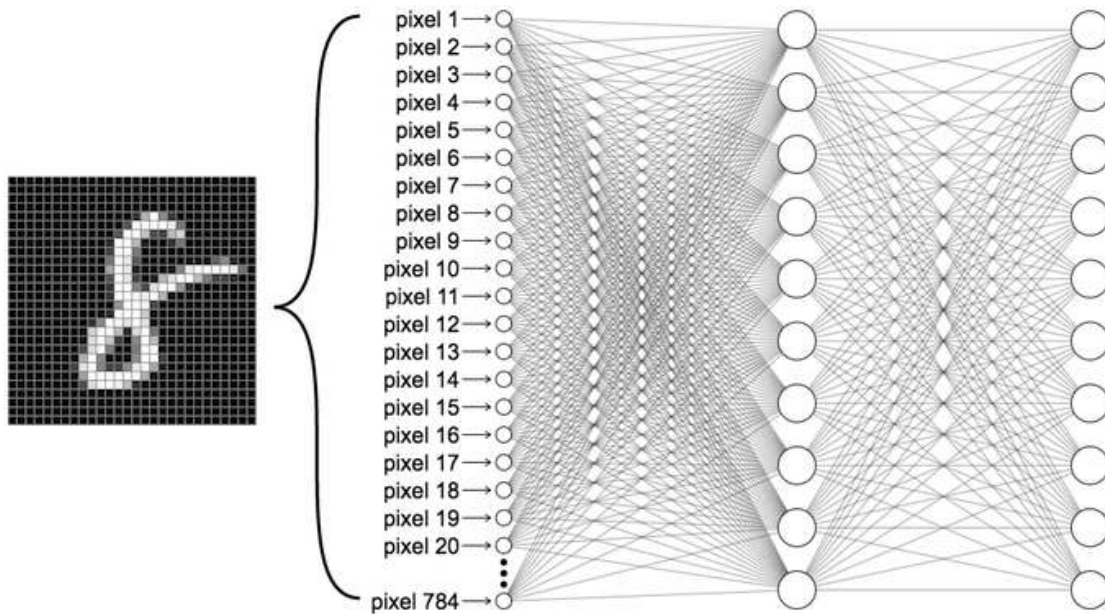


Chapter 05

딥러닝 라이브러리 구현과 활용

01 딥러닝 라이브러리 구현하기

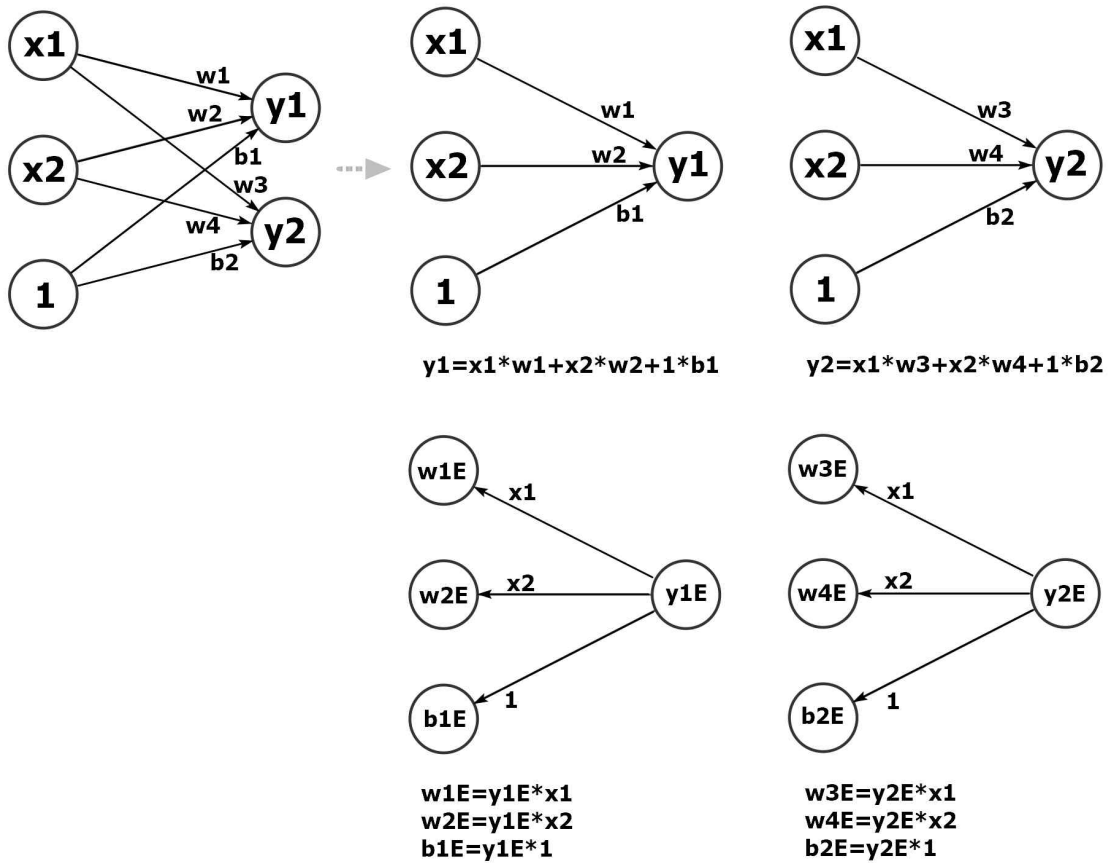
여기서는 인공 신경망을 확장할 수 있도록 배열과 함수를 이용하여 인공 신경망을 구현해 봅니다. 배열과 함수를 이용하면, 커다란 인공 신경망을 자유롭게 구성하고 테스트해 볼 수 있습니다. 예를 들어, 1장에서 tensorflow 라이브러리를 이용하여 살펴보았던 다음과 같은 형태의 인공 신경망을 구성해서 테스트해 볼 수 있습니다.



<784개의 입력, 64개의 은닉층, 10개의 출력층>

01 2입력 2출력 행렬 계산식 정리하기

다음 그림은 입력2 출력2로 구성된 인공 신경망과 순전파 역전파 수식을 나타냅니다. 여기서는 다음 수식을 행렬 계산식으로 유도합니다. 행렬 계산식은 배열을 이용하여 구현하기에 적합합니다.



행렬 계산식 유도하기

이 그림을 통해 앞에서 우리는 다음 표의 왼쪽과 같은 수식을 유도했습니다. 이런 형태의 수식을 다원일차연립방정식이라고 합니다. 다원일차연립방정식은 행렬을 이용하면 깔끔하게 정리할 수 있습니다. 행렬 계산식으로 정리하면 다음 표의 오른쪽과 같습니다.

	다원일차연립방정식	행렬 계산식
순 전 파	$y_1 = x_1^* w_1 + x_2^* w_2 + 1^* b_1$ $y_2 = x_1^* w_3 + x_2^* w_4 + 1^* b_2$	$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} + 1 \begin{bmatrix} b_1 & b_2 \end{bmatrix}$
역 전 파	$w_{1E} = y_{1E}^* x_1$ $w_{2E} = y_{1E}^* x_2$ $w_{3E} = y_{2E}^* x_1$ $w_{4E} = y_{2E}^* x_2$ $b_{1E} = y_{1E}^* 1$ $b_{2E} = y_{2E}^* 1$	$\begin{bmatrix} w_{1E} & w_{3E} \\ w_{2E} & w_{4E} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix}$ $= \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix}$ $\begin{bmatrix} b_{1E} & b_{2E} \end{bmatrix} = 1 \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix}$
인 공 신 경 망 학 습	$w_1 = w_1 - \alpha w_{1E}$ $w_2 = w_2 - \alpha w_{2E}$ $w_3 = w_3 - \alpha w_{3E}$ $w_4 = w_4 - \alpha w_{4E}$ $b_1 = b_1 - \alpha b_{1E}$ $b_2 = b_2 - \alpha b_{2E}$	$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \alpha \begin{bmatrix} w_{1E} & w_{3E} \\ w_{2E} & w_{4E} \end{bmatrix}$ $\begin{bmatrix} b_1 & b_2 \end{bmatrix} = \begin{bmatrix} b_1 & b_2 \end{bmatrix} - \alpha \begin{bmatrix} b_{1E} & b_{2E} \end{bmatrix}$

역전파 행렬 계산식에서 다음은 순전파 때 사용된 입력의 전치 행렬입니다.

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

전치행렬은 가로줄과 세로줄이 바뀐 행렬입니다.

인공 신경망 행렬 계산식 정리하기

위 수식에서 표현된 행렬들에 다음 표의 왼쪽과 같이 이름을 붙여줍니다. 그러면 위의 행렬 계산식은 다음표의 오른쪽과 같이 정리할 수 있습니다. 오른쪽의 행렬 계산식은 행렬의 크기와 상관없이 성립합니다. 주의할 점은 행렬의 곱은 순서를 변경하면 안 됩니다.

행렬 이름	인공 신경망 행렬 계산식
$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = Y$ $\begin{bmatrix} x_1 & x_2 \end{bmatrix} = X$ $\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = W$ $\begin{bmatrix} b_1 & b_2 \end{bmatrix} = B$ $\begin{bmatrix} w_{1E} & w_{3E} \\ w_{2E} & w_{4E} \end{bmatrix} = W_E$ $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T = X^T$ $\begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix} = Y_E$ $\begin{bmatrix} b_{1E} & b_{2E} \end{bmatrix} = B_E$	<p>순전파</p> $Y = XW + B$ <p>역전파</p> $W_E = X^T Y_E$ $B_E = 1 Y_E$ <p>인공 신경망 학습</p> $W = W - \alpha W_E$ $B = B - \alpha B_E$

02 행렬 계산식 배열과 함수로 정리하기

여기서는 앞에서 구한 2입력 2출력 행렬 계산식의 구조를 자세히 살펴보고 배열과 함수로 정리하는 과정을 소개합니다. 또 오차 계산과 활성화 함수의 순전파 역전파도 배열과 함수로 정리합니다.

순전파 정리하기

다음은 순전파의 행렬 계산식이 일차연립방정식으로 해석되는 과정을 나타냅니다.

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

$$Y \quad X \quad W \quad B$$

$$y_1 = x_1 w_1 + x_2 w_2 + b_1$$

$$y_2 = x_1 w_3 + x_2 w_4 + b_2$$

행렬의 곱 $X@W$ 는 앞에 오는 X 행렬의 가로줄 항목, 뒤에 오는 W 행렬의 세로줄 항목이 순서대로 곱해진 후, 모두 더해져서 임시 행렬(예를 들어, XW 행렬)의 항목 하나를 구성합니다. 그래서 X 행렬의 가로줄 항목 개수와 W 행렬의 세로줄 항목 개수는 같아야 합니다. 계속해서 XW 행렬의 각 항목은 B 행렬의 각 항목과 더해져 Y 행렬의 각 항목을 구성합니다.

*** 여기서 @ 문자는 행렬의 곱을 나타냅니다. 참고로 파이썬에서는 @문자를 이용하여 행렬의 곱을 수행합니다.

다음은 순전파의 행렬 계산식을 숫자로 표현한 구체적인 예입니다.

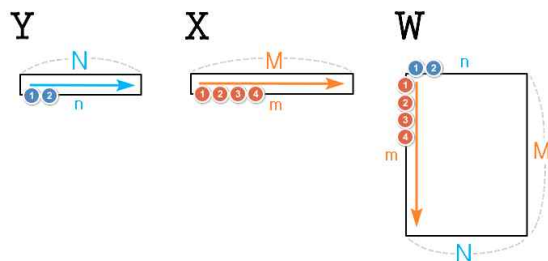
$$\begin{bmatrix} 19 & 30 \end{bmatrix} = \begin{bmatrix} 2 & 3 \end{bmatrix} \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 2 \end{bmatrix}$$

$Y \quad X \quad W \quad B$

$$\begin{aligned} 19 &= 2 \times 3 + 3 \times 4 + 1 \\ 30 &= 2 \times 5 + 3 \times 6 + 2 \end{aligned}$$

❶ 순전파 행렬곱 배열로 정리하기

다음은 배열을 이용한 순전파 행렬곱 계산 과정을 나타냅니다.



$$Y[n] += X[m] * W[m][n]$$

0~N인 n , 0~M인 m 에 대해 $X[m]$ 을 $W[m][n]$ 에 곱해 $Y[n]$ 에 누적합니다.

이 그림을 C/C++로 구현하면 다음과 같습니다. 참고로 아두이노 스케치는 C/C++ 언어로 작성됩니다.

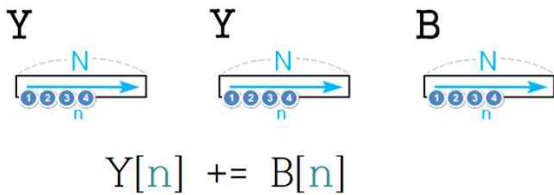
```
for(int n=0;n<N;n++)
    for(int m=0;m<M;m++)
        Y[n] += X[m]*W[m][n];
```

다음과 같이 m , n 의 순서를 변경해도 결과는 같습니다.

```
for(int m=0;m<M;m++)
    for(int n=0;n<N;n++)
        Y[n] += X[m]*W[m][n];
```

② 편향 배열로 정리하기

다음은 배열을 이용한 순전파 행렬합 계산 과정을 나타냅니다.



0~N인 n에 대해 Y[n]에 B[n]을 더해줍니다.

이 그림을 C/C++로 구현하면 다음과 같습니다.

```
for(int n=0;n<N;n++)
    Y[n] += B[n];
```

③ 순전파 함수로 정리하기

순전파를 함수로 정리하면 다음과 같습니다.

```
void forward_Y(double *X, double *W, double *B, double *Y, int M, int N) {
    for(int n=0;n<N;n++) Y[n] = 0;
    for(int m=0;m<M;m++)
        for(int n=0;n<N;n++)
            Y[n] += X[m]*W[m*N+n];
    for(int n=0;n<N;n++) Y[n] += B[n];
}
```

forward_Y 함수의 매개변수 W는 2차 배열을 1차 배열 형태로 받게 되며, 따라서 함수 내부 (5번째 줄)에서 W의 항목을 읽을 때, 1차 배열 형태로 맞추어 접근해야 합니다.

④ 순전파 함수 사용해 보기

여기서는 순전파 함수를 테스트해 봅니다.

1. 다음과 같이 예제를 작성합니다.

512.ino

```
01 void forward_Y(double *X, double *W, double *B, double *Y, int M, int N) {
02     for(int n=0;n<N;n++) Y[n] = 0;
03     for(int m=0;m<M;m++)
04         for(int n=0;n<N;n++)
05             Y[n] += X[m]*W[m*N+n];
06     for(int n=0;n<N;n++) Y[n] += B[n];
07 }
08
09 void print(char * s, double *Y, int N) {
10     printf("%s [", s);
11     for(int n=0;n<N-1;n++) printf("%.3f ", Y[n]);
12     printf("%.3f]\n", Y[N-1]);
13 }
14
15 const int M = 2;
16 const int N = 2;
17
18 double X[M] = {2, 3};
19 double W[M][N] = {
20     {3, 5},
21     {4, 6}
22 };
23 double B[N] = {1, 2};
24 double Y[N] = {0, 0};
25 double T[N] = {27, -30};
26
27 void dnn_test() {
28
29     forward_Y(X, (double *)W, B, Y, M, N);
30     print("Y =", Y, N);
31
32 }
33
34 void setup() {
35
36     Serial.begin(115200);
37     delay(1000);
38
39     dnn_test();
40 }
```



```

41 }
42
43 void loop() {
44
45 }

```

01~07 : forward_Y 함수를 정의합니다. 앞에서 정의한 순전파 함수입니다.

09~13 : print 함수를 정의합니다. 일차 배열을 출력하는 함수입니다.

15 : M 상수를 선언하고 2로 초기화합니다. M은 18, 19, 20줄에서 사용하며, 순전파 시 입력의 개수, 가중치 배열 행의 개수를 나타냅니다.

16 : N 상수를 선언하고 2로 초기화합니다. N은 19, 23~25, 29줄에서 사용하며, 순전파 시 출력의 개수, 가중치 배열 열의 개수를 나타냅니다.

18 : 입력 배열 X를 선언하고 초기화합니다.

19~22 : 가중치 배열 W를 선언하고 초기화합니다.

23 : 편향 배열 B를 선언하고 초기화합니다.

24 : 출력 배열 Y를 선언하고 초기화합니다.

23 : 목표 배열 T를 선언하고 초기화합니다.

27~32 : dnn_test 함수를 정의합니다.

29 : forward_Y 함수를 호출합니다. 첫 번째 인자로 출력 Y, 두 번째 인자로 입력 X, 세 번째 인자로 가중치 W, 네 번째 인자로 편향 B, 다섯 번째 인자로 입력의 개수 M, 여섯 번째 인자로 출력의 개수 N을 줍니다. 세 번째 인자의 경우 일차 배열로 변환하여 넘겨주어야 합니다.

30 : print 함수를 호출하여 Y 값을 출력합니다.

2. 업로드를 수행합니다.



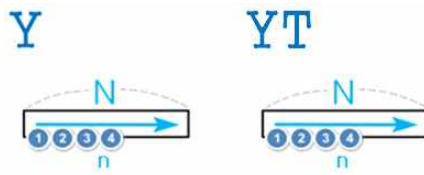
3. 출력 결과를 확인합니다.

```
Y = [19.000 30.000]
```

오차 계산 정리하기

❶ 오차 계산 배열로 정리하기

다음은 배열을 이용한 오차 계산 과정을 나타냅니다.



$$E += (Y[n] - YT[n]) * (Y[n] - YT[n]) / 2$$

0~N인 n에 대해 Y[n]에서 YT[n]을 빼서 제곱한 후, 2로 나누어 오차 E에 누적합니다.

이 그림을 C/C++로 구현하면 다음과 같습니다.

```
for(int n=0;n<N;n++)
    E += (Y[n]-YT[n])*(Y[n]-YT[n])/2;
```

② 오차 계산 함수로 정리하기

오차 계산을 함수로 정리하면 다음과 같습니다.

```
double calculate_MSE(double *Y, double *YT, int N) {
    double E = 0;
    for(int n=0;n<N;n++) E += (Y[n]-YT[n])*(Y[n]-YT[n])/2;
    return E;
}
```

역전파 오차 정리하기

다음은 역전파 오차의 행렬 계산식이 일차연립방정식으로 해석되는 과정을 나타냅니다.

$$\begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix} = \begin{bmatrix} y_1 & y_2 \end{bmatrix} - \begin{bmatrix} y_{1T} & y_{2T} \end{bmatrix}$$

$$Y_E \quad Y \quad Y_T$$

$$\begin{aligned} y_{1E} &= y_1 - y_{1T} \\ y_{2E} &= y_2 - y_{2T} \end{aligned}$$

Y 행렬의 각 항목에 대해서 YT 행렬의 대응되는 항목을 뺀 후, YE 행렬의 대응되는 항목에 대입합니다.

다음은 역전파 오차의 행렬 계산식을 숫자로 표현한 구체적인 예입니다.

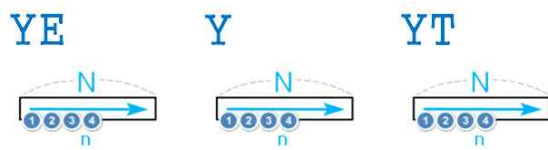
$$\begin{bmatrix} -8 & 60 \end{bmatrix} = \begin{bmatrix} 19 & 30 \end{bmatrix} - \begin{bmatrix} 27 & -30 \end{bmatrix}$$

$$Y_E \quad Y \quad Y_T$$

$$\begin{aligned} -8 &= 19 - 27 \\ 60 &= 30 - (-30) \end{aligned}$$

❶ 역전파 오차 배열로 정리하기

다음은 배열을 이용한 역전파 오차 계산 과정을 나타냅니다.



$$YE[n] = Y[n] - YT[n]$$

0~N인 n에 대해 Y[n]에서 T[n]을 빼서 Yb[n]에 대입합니다.

이 그림을 C/C++로 구현하면 다음과 같습니다.

```
for(int n=0;n<N;n++)
    YE[n] = Y[n] - YT[n];
```

❷ 역전파 오차 함수로 정리하기

역전파 오차를 함수로 정리하면 다음과 같습니다.

```
void backward_YE(double *YE, double *Y, double *YT, int N) {
    for(int n=0;n<N;n++) YE[n] = Y[n] - YT[n];
}
```

가중치 역전파 정리하기

다음은 가중치 역전파의 행렬 계산식이 일차연립방정식으로 해석되는 과정을 나타냅니다.

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

$$W_E \quad X^T \quad Y_E$$

$$\begin{aligned} w_{1E} &= x_1 y_{1E} & w_{3E} &= x_1 y_{2E} \\ w_{2E} &= x_2 y_{1E} & w_{4E} &= x_2 y_{2E} \end{aligned}$$

행렬의 곱 $X.T@Y_E$ 는 앞에 오는 $X.T$ 행렬의 가로줄 항목 각각에 대해, 뒤에 오는 Y_E 행렬의 세로줄 항목 각각에 곱해진 후, W_E 행렬의 각각의 항목을 구성합니다.

*** 여기서 @ 문자는 행렬의 곱을 나타냅니다. 참고로 파이썬에서는 @문자를 이용하여 행렬의 곱을 수행합니다.

*** 여기서 $X.T$ 는 W 행렬의 전치행렬을 나타냅니다.

다음은 가중치 역전파의 행렬 계산식을 숫자로 표현한 구체적인 예입니다.

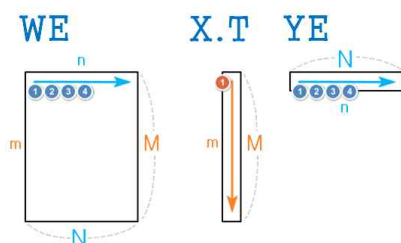
$$\begin{bmatrix} -16 & 120 \\ -24 & 180 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \begin{bmatrix} -8 & 60 \end{bmatrix}$$

$$W_E \quad X^T \quad Y_E$$

$$\begin{aligned} -16 &= 2 \times -8 & 120 &= 2 \times 60 \\ -24 &= 3 \times -8 & 180 &= 3 \times 60 \end{aligned}$$

❶ 가중치 역전파 배열로 정리하기

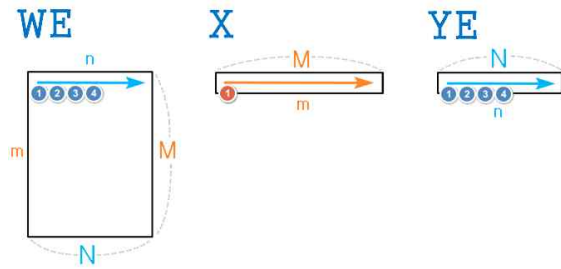
다음은 배열을 이용한 가중치 역전파 행렬곱 계산 과정을 나타냅니다.



$$WE[m][n] = X.T[m][0] * YE[n]$$

0~N인 n , 0~M인 m 에 대해 $X.T[m][0]$ 은 $YE[n]$ 에 곱해져 $WE[m][n]$ 에 대입됩니다.

배열을 이용한 가중치 역전파는 다음과 같이 계산할 수도 있습니다.



$$WE[m][n] = X[m] * YE[n]$$

0~N인 n , 0~M인 m 에 대해 $X[m]$ 은 $YE[n]$ 에 곱해져 $WE[m][n]$ 에 대입됩니다.

이 그림을 C/C++로 구현하면 다음과 같습니다.

```
for(int m=0;m<M;m++)
    for(int n=0;n<N;n++)
        WE[m][n] = X[m] * YE[n];
```

② 가중치 역전파 함수로 정리하기

가중치 역전파를 함수로 정리하면 다음과 같습니다.

```
void backward_WE(double *WE, double *X, double *YE, int M, int N) {
    for(int m=0;m<M;m++)
        for(int n=0;n<N;n++)
            WE[m*N+n] = X[m]*YE[n];
}
```

backward_WE 함수의 매개변수 WE는 2차 배열을 1차 배열 형태로 받게 되며, 따라서 함수 내부(4번째 줄)에서 WE의 항목을 읽을 때, 1차 배열 형태로 맞추어 접근해야 합니다.

④ 가중치 역전파 함수 사용해 보기

여기서는 가중치 역전파 함수를 테스트해 봅니다.

1. 다음과 같이 예제를 작성합니다.

512_3.ino

```
01 void backward_WE(double *WE, double *X, double *YE, int M, int N) {
02     for(int m=0;m<M;m++)
03         for(int n=0;n<N;n++)
04             WE[m*N+n] = X[m]*YE[n];
05 }
```

```

06
07 void print(char * s, double *W, int M, int N) {
08     printf("%s [\n", s);
09     for(int m=0;m<M;m++) {
10         printf("[", s);
11         for(int n=0;n<N-1;n++)
12             printf("%.3f ", W[m*N+n]);
13         printf("%.3f]\n", W[m*N+N-1]);
14     }
15     printf("]\n");
16 }
17
18 const int M = 2;
19 const int N = 2;
20
21 double X[N] = {2, 3};
22 double YE[M] = {-8, 60};
23 double WE[M][N] = {0,};
24
25 void dnn_test() {
26
27     backward_WE((double *)WE, X, YE, M, N);
28     print("WE =", (double *)WE, M, N);
29
30 }
31
32 void setup() {
33
34     Serial.begin(115200);
35     delay(1000);
36
37     dnn_test();
38
39 }
40
41 void loop() {
42
43 }

```

01~05 : backward_WE 함수를 정의합니다. 앞에서 정의한 가중치 역전파 함수입니다.

07~16 : print 함수를 정의합니다. 이차 배열을 출력하는 함수입니다.

18 : M 상수를 선언하고 2로 초기화합니다. M은 22, 23, 27, 28줄에서 사용합니다.

19 : N 상수를 선언하고 2로 초기화합니다. N은 21, 23, 27, 28줄에서 사용합니다.
 21 : 입력 배열 X를 선언하고 초기화합니다.
 22 : 역전파 오차 배열 YE를 선언하고 초기화합니다.
 23 : 가중치 역전파 배열 WE를 선언하고 초기화합니다.
 25~30 : dnn_test 함수를 정의합니다.
 27 : backward_WE 함수를 호출합니다. 첫 번째 인자로 가중치 역전파 WE, 두 번째 인자로 입력 X, 세 번째 인자로 역전파 오차 YE, 네 번째 인자로 입력의 개수 M, 다섯 번째 인자로 역전파 오차의 개수 N을 줍니다. 첫 번째 인자의 경우 일차 배열로 변환하여 넘겨주어야 합니다.
 28 : print 함수를 호출하여 Xb 값을 출력합니다.

2. 업로드를 수행합니다.



3. 출력 결과를 확인합니다.

```
WE = [
[-16.000 120.000]
[-24.000 180.000]
]
```

역전파 정리하기 : 편향

다음은 편향 역전파의 행렬 계산식이 일차연립방정식으로 해석되는 과정을 나타냅니다.

$$\begin{bmatrix} b_{1E} & b_{2E} \end{bmatrix} = 1 \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix}$$

$$B_E \quad Y_E$$

$$\begin{aligned} b_{1E} &= 1 y_{1E} \\ b_{2E} &= 1 y_{1E} \end{aligned}$$

YE 행렬의 각 항목은 1과 곱해서 BE 행렬의 각 항목에 대입됩니다.

다음은 편향 역전파의 행렬 계산식을 숫자로 표현한 구체적인 예입니다.

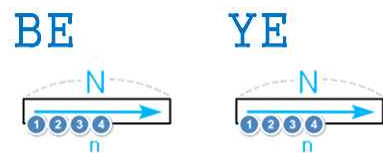
$$\begin{bmatrix} -80 & 60 \end{bmatrix} = 1 \begin{bmatrix} -80 & 60 \end{bmatrix}$$

$$B_E \qquad Y_E$$

$$\begin{aligned} -80 &= 1 \times (-80) \\ 60 &= 1 \times 60 \end{aligned}$$

❶ 편향 역전파 배열로 정리하기

다음은 배열을 이용한 편향 역전파 행렬 대입 과정을 나타냅니다.



$$BE[n] = 1 * YE[n]$$

0~N인 n에 대해 1*YE[n]을 BE[n]에 대입합니다.

이 그림을 C/C++로 구현하면 다음과 같습니다.

```
for(int n=0;n<N;n++)
    BE[n] = YE[n];
```

❷ 편향 역전파 함수로 정리하기

편향 역전파를 함수로 정리하면 다음과 같습니다.

```
void backward_BE(double *BE, double *YE, int N) {
    for(int n=0;n<N;n++) BE[n] = YE[n];
}
```

가중치 학습 정리하기

다음은 가중치 학습 행렬 계산식이 일차연립방정식으로 해석되는 과정을 나타냅니다.

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \alpha \begin{bmatrix} w_{1E} & w_{3E} \\ w_{2E} & w_{4E} \end{bmatrix}$$

$W \qquad \qquad W \qquad \qquad W_E$

$$\begin{aligned} w_1 &= w_1 - \alpha w_{1E} & w_3 &= w_3 - \alpha w_{3E} \\ w_2 &= w_2 - \alpha w_{2E} & w_4 &= w_4 - \alpha w_{4E} \end{aligned}$$

W 행렬의 각 항목에서 학습률이 곱해진 WE 행렬의 각 항목을 빼줍니다.

다음은 가중치 학습 행렬 계산식을 숫자로 표현한 구체적인 예입니다.

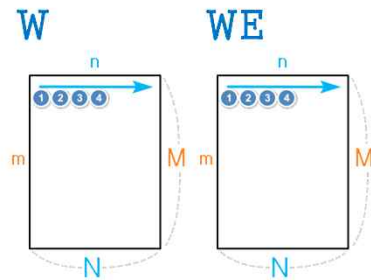
$$\begin{bmatrix} 3.16 & 3.8 \\ 4.24 & 4.2 \end{bmatrix} = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} - 0.01 \begin{bmatrix} -16 & 120 \\ -24 & 180 \end{bmatrix}$$

$W \qquad \qquad W \qquad \qquad W_E$

$$\begin{aligned} 3.16 &= 3 - 0.01 \times (-16) & 3.8 &= 5 - 0.01 \times 120 \\ 4.24 &= 4 - 0.01 \times (-24) & 4.2 &= 6 - 0.01 \times 180 \end{aligned}$$

❶ 가중치 학습 배열로 정리하기

다음은 배열을 이용한 가중치 학습 계산 과정을 나타냅니다.



$$W[m][n] -= lr * WE[m][n]$$

0~N인 n, 0~M인 m에 대해 W[m][n]에서 학습률 lr이 곱해진 WE[m][n]을 빼 줍니다.

이 그림을 C/C++로 구현하면 다음과 같습니다.

```
for(int m=0;m<M;m++)
    for(int n=0;n<N;n++)
        W[m][n] -= lr * WE[m][n];
```

② 가중치 학습 함수로 정리하기

가중치 학습을 함수로 정리하면 다음과 같습니다.

```
void learning_W(double *W, double lr, double *WE, int M, int N) {  
    for(int m=0;m<M;m++)  
        for(int n=0;n<N;n++)  
            W[m*N+n] -= lr * WE[m*N+n];  
}
```

편향 학습 정리하기

다음은 편향 학습 행렬 계산식이 일차연립방정식으로 해석되는 과정을 나타냅니다.

$$\begin{bmatrix} b_1 & b_2 \end{bmatrix} = \begin{bmatrix} b_1 & b_2 \end{bmatrix} - \alpha \begin{bmatrix} b_{1E} & b_{2E} \end{bmatrix}$$

$$B \qquad B \qquad B_E$$

$$\begin{aligned} b_1 &= b_1 - \alpha b_{1E} \\ b_2 &= b_2 - \alpha b_{2E} \end{aligned}$$

B 행렬의 각 항목에서 학습률이 곱해진 BE 행렬의 각 항목을 빼줍니다.

다음은 편향 학습 행렬 계산식을 숫자로 표현한 구체적인 예입니다.

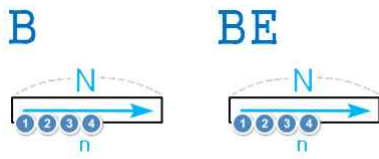
$$\begin{bmatrix} 1.08 & 1.40 \end{bmatrix} = \begin{bmatrix} 1 & 2 \end{bmatrix} - 0.01 \begin{bmatrix} -80 & 60 \end{bmatrix}$$

$$B \qquad B \qquad B_E$$

$$\begin{aligned} 1.08 &= 1 - 0.01 \times (-80) \\ 1.40 &= 2 - 0.01 \times 60 \end{aligned}$$

① 편향 학습 배열로 정리하기

다음은 배열을 이용한 편향 학습 계산 과정을 나타냅니다.



$$B[n] -= lr * BE[n]$$

0~N인 n에 대해 B[n]에서 학습률 lr이 곱해진 BE[n]을 빼 줍니다.

이 그림을 C/C++로 구현하면 다음과 같습니다.

```
for(int n=0;n<N;n++)
    B[n] -= lr * BE[n];
```

② 편향 학습 함수로 정리하기

편향 학습을 함수로 정리하면 다음과 같습니다.

```
void learning_B(double *B, double lr, double *BE, int N) {
    for(int n=0;n<N;n++) B[n] -= lr * BE[n];
}
```

딥러닝 라이브러리 정리하기 1

이상의 내용으로 딥러닝 라이브러리를 정리하면 다음과 같습니다. 이후에는 여기서 작성한 라이브러리를 이용하여 딥러닝 예제를 작성합니다.

mydnn_1.ino

```
01 void forward_Y(double *X, double *W, double *B, double *Y, int M, int N) {
02     for(int n=0;n<N;n++) Y[n] = 0;
03     for(int m=0;m<M;m++)
04         for(int n=0;n<N;n++)
05             Y[n] += X[m]*W[m*N+n];
06     for(int n=0;n<N;n++) Y[n] += B[n];
07 }
08
09 double calculate_MSE(double *Y, double *YT, int N) {
10     double E = 0;
11     for(int n=0;n<N;n++) E += (Y[n]-YT[n])*(Y[n]-YT[n])/2;
12     return E;
13 }
```

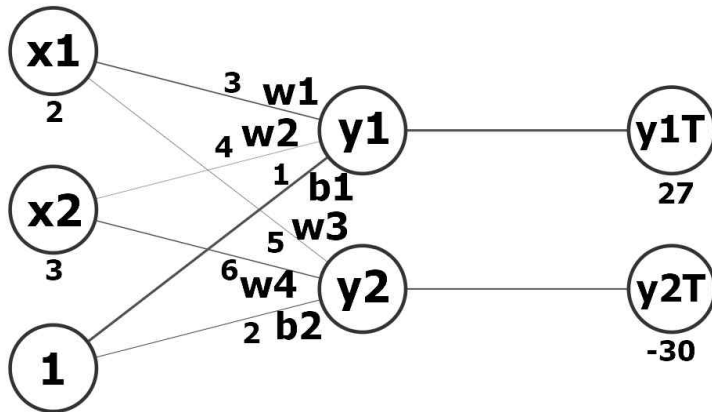
```

14
15 void backward_YE(double *YE, double *Y, double *YT, int N) {
16     for(int n=0;n<N;n++) YE[n] = Y[n] - YT[n];
17 }
18
19 void backward_WE(double *WE, double *X, double *YE, int M, int N) {
20     for(int m=0;m<M;m++)
21         for(int n=0;n<N;n++)
22             WE[m*N+n] = X[m]*YE[n];
23 }
24
25 void backward_BE(double *BE, double *YE, int N) {
26     for(int n=0;n<N;n++) BE[n] = YE[n];
27 }
28
29 void learning_W(double *W, double lr, double *WE, int M, int N) {
30     for(int m=0;m<M;m++)
31         for(int n=0;n<N;n++)
32             W[m*N+n] -= lr * WE[m*N+n];
33 }
34
35 void learning_B(double *B, double lr, double *BE, int N) {
36     for(int n=0;n<N;n++) B[n] -= lr * BE[n];
37 }
38
39 void print(char * s, double *Y, int N) {
40     printf("%s [", s);
41     for(int n=0;n<N-1;n++) printf("%.3f ", Y[n]);
42     printf("%.3f]\n", Y[N-1]);
43 }
44
45 void print(char * s, double *W, int M, int N) {
46     printf("%s [\n", s);
47     for(int m=0;m<M;m++) {
48         printf("[", s);
49         for(int n=0;n<N-1;n++)
50             printf("%.3f ", W[m*N+n]);
51         printf("%.3f]\n", W[m*N+N-1]);
52     }
53     printf("]\n");

```

03 2입력 2출력 인공 신경망 구현하기

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



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

$$X = [x_1 \ x_2] = [2 \ 3]$$

$$W = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix}$$

$$B = [b_1 \ b_2] = [1 \ 2]$$

$$Y_T = [y_{1T} \ y_{2T}] = [27 \ -30]$$

X , Y_T 를 이용하여 W , B 에 대해 학습을 수행해 봅니다.

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

1. 다음과 같이 예제를 작성합니다.

513.ino

```
01 const int NUM_X = 2;
02 const int NUM_Y = 2;
03
04 double X[NUM_X] = {2, 3};
05 double YT[NUM_Y] = {27, -30};
06 double Y[NUM_Y];
07
08 double W[NUM_X][NUM_Y] = {
09     {3, 5},
10     {4, 6}
```

```
11 };
12 double B[NUM_Y] = {1, 2};
13
14 double YE[NUM_Y];
15 double WE[NUM_X][NUM_Y];
16 double BE[NUM_Y];
17
18 void dnn_test() {
19
20     forward_Y(X, (double *)W, B, Y, NUM_X, NUM_Y);
21     print("Y =", Y, NUM_Y);
22
23     double E = calculate_MSE(Y, YT, NUM_Y);
24     printf("E = %.7f\n", E);
25
26     backward_YE(YE, Y, YT, NUM_Y);
27     print("YE =", YE, NUM_Y);
28
29     backward_WE((double *)WE, X, YE, NUM_X, NUM_Y);
30     print("WE =", (double *)WE, NUM_X, NUM_Y);
31
32     backward_BE(BE, YE, NUM_Y);
33     print("BE =", BE, NUM_Y);
34
35     double lr = 0.01;
36     learning_W((double *)W, lr, (double *)WE, NUM_X, NUM_Y);
37     print("W =", (double *)W, NUM_X, NUM_Y);
38
39     learning_B(B, lr, BE, NUM_Y);
40     print("B =", B, NUM_Y);
41
42 }
43
44 void setup() {
45
46     Serial.begin(115200);
47     delay(1000);
48
49     dnn_test();
50
```

```

51 }
52
53 void loop() {
54
55 }

```

- 01 : NUM_X 상수를 선언하고 2로 초기화합니다.
- 02 : NUM_Y 상수를 선언하고 2로 초기화합니다.
- 04 : 입력 배열 X를 선언하고 초기화합니다.
- 05 : 목표 배열 YT를 선언하고 초기화합니다.
- 06 : 출력 배열 Y를 선언하고 초기화합니다.
- 08~11 : 가중치 배열 W를 선언하고 초기화합니다.
- 12 : 편향 배열 B를 선언하고 초기화합니다.
- 14 : 역전파 오차 배열 YE를 선언합니다.
- 15 : 가중치 역전파 배열 WE를 선언합니다.
- 16 : 편향 역전파 배열 BE를 선언합니다.
- 18~42 : dnn_test 함수를 정의합니다.
- 20~21 : X에서 Y로 순전파를 수행하고 결과를 확인합니다.
- 23~24 : 평균 제곱 오차를 계산하고 결과를 확인합니다.
- 26~27 : 역전파 오차를 계산하고 결과를 확인합니다.
- 29~30 : 가중치 역전파를 수행하고 결과를 확인합니다.
- 32~33 : 편향 역전파를 수행하고 결과를 확인합니다.
- 35~37 : 가중치 학습을 수행하고 결과를 확인합니다.
- 39~40 : 편향 학습을 수행하고 결과를 확인합니다.

2. 앞에서 정리한 mydnn_1.ino 파일을 포함합니다.



- ❶ 그림과 같이 [새 탭] 메뉴를 눌러 [파일 이름]으로 mydnn_1을 입력한 후,
- ❷ [확인] 버튼을 눌러줍니다.
- ❸ 추가한 mydnn_1 파일에 앞에서 정리한 mydnn_1.ino 파일의 내용을 복사합니다.

*** 제공되는 소스에서 mydnn_1.ino 파일을 찾아 현재 작성 중인 스케치 디렉터리로 복사한 후, 스케치를 닫았다 다시 열어도 됩니다.

3. 업로드를 수행합니다.



4. 출력 결과를 확인합니다.

```

Y = [19.000 30.000]
E = 1832.0000000
YE = [-8.000 60.000]
WE = [
[-16.000 120.000]
[-24.000 180.000]
]
BE = [-8.000 60.000]
W = [
[3.160 3.800]
[4.240 4.200]
]
B = [1.080 1.400]

```

5. 다음과 같이 예제를 수정합니다.

513_2.ino

```

01 const int NUM_X = 2;
02 const int NUM_Y = 2;
03
04 double X[NUM_X] = {2, 3};
05 double YT[NUM_Y] = {27, -30};
06 double Y[NUM_Y];
07
08 double W[NUM_X][NUM_Y] = {
09     {3, 5},
10     {4, 6}
11 };
12 double B[NUM_Y] = {1, 2};
13
14 double YE[NUM_Y];
15 double WE[NUM_X][NUM_Y];
16 double BE[NUM_Y];
17
18 void dnn_test() {
19
20     for(int epoch=0;epoch<1000;epoch++) {
21
22         printf("\nepoch = %d\n", epoch);
23
24         forward_Y(X, (double *)W, B, Y, NUM_X, NUM_Y);
25         print("Y =", Y, NUM_Y);
26
27         double E = calculate_MSE(Y, YT, NUM_Y);

```



```

28         printf("E = %.7f\n", E);
29
30         if(E < 0.0000001)
31             break;
32
33         backward_YE(YE, Y, YT, NUM_Y);
34         print("YE =", YE, NUM_Y);
35
36         backward_WE((double *)WE, X, YE, NUM_X, NUM_Y);
37         print("WE =", (double *)WE, NUM_X, NUM_Y);
38
39         backward_BE(BE, YE, NUM_Y);
40         print("BE =", BE, NUM_Y);
41
42         double lr = 0.01;
43         learning_W((double *)W, lr, (double *)WE, NUM_X, NUM_Y);
44         print("W =", (double *)W, NUM_X, NUM_Y);
45
46         learning_B(B, lr, BE, NUM_Y);
47         print("B =", B, NUM_Y);
48     }
49
50 }
51
52 void setup() {
53
54     Serial.begin(115200);
55     delay(1000);
56
57     dnn_test();
58
59 }
60
61 void loop() {
62
63 }

```

20 : epoch값을 0에서 1000 미만까지 바꾸어가며 22~48줄을 1000회 수행합니다.

30~31 : 오차값이 0.0000001보다 작으면 20줄의 for 문을 나옵니다.

6. 업로드를 수행합니다.



7. 출력 결과를 확인합니다.

```
epoch = 78
Y = [27.000 -30.000]
E = 0.0000001
YE = [-0.000 0.000]
WE = [
[-0.000 0.001]
[-0.000 0.001]
]
BE = [-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
```

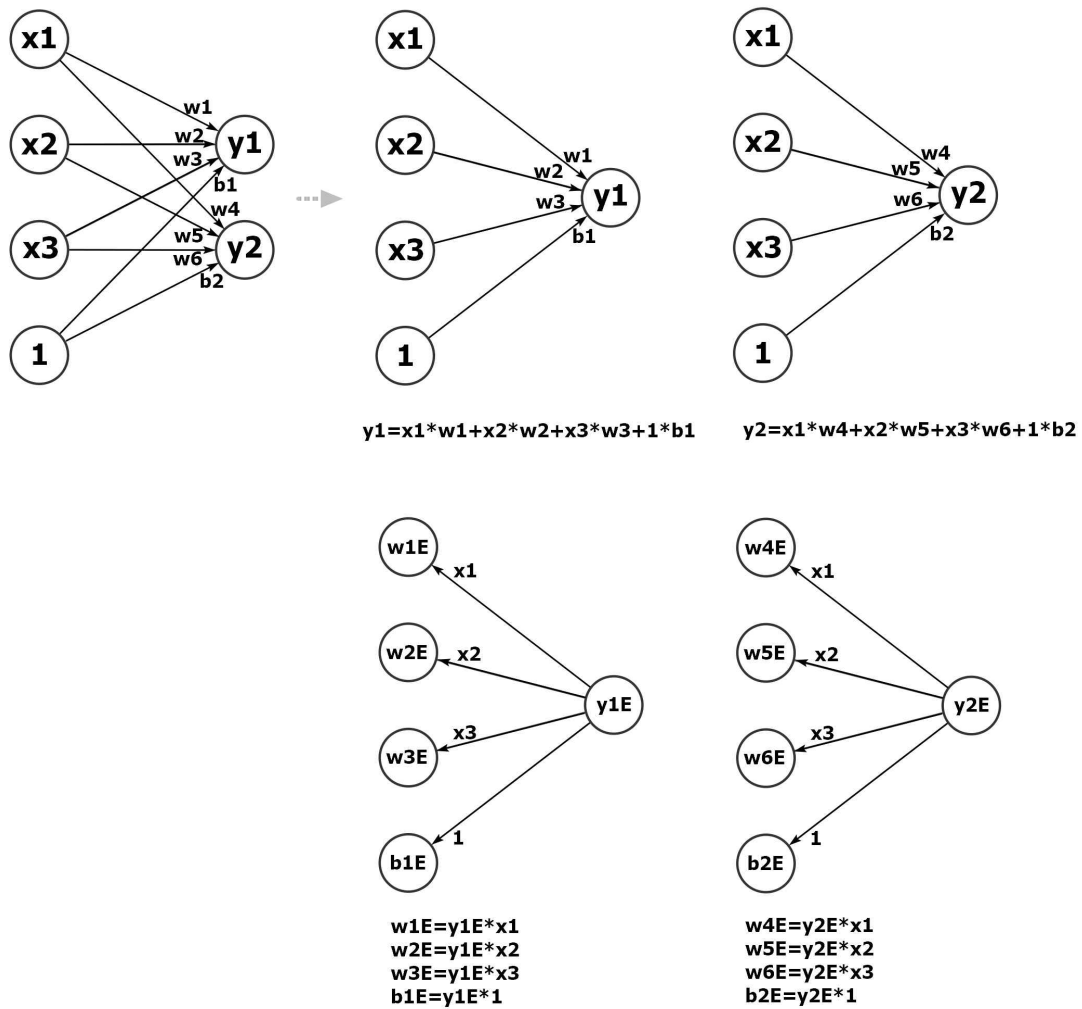
(79+1)회 째 학습이 완료되는 것을 볼 수 있습니다.

4장 3절 2단원의 예제의 결과와 비교해 봅니다.

```
epoch = 78
y1, y2 = 27.000, -30.000
E = 0.0000001
w1E, w3E = -0.000, 0.001
w2E, w4E = -0.000, 0.001
b1E, b2E = -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
```

04 3입력 2출력 인공 신경망 구현하기

다음 그림은 입력3 출력2로 구성된 인공 신경망과 순전파 역전파 수식을 나타냅니다. 우리는 다음 수식을 행렬 계산식으로 유도한 후, 배열을 이용하여 인공 신경을 구현해 봅니다.



입력3 출력2로 구성된 신경망은 입력3 출력1 인공 신경 2개로 구성됩니다.

행렬 계산식 유도하기

앞의 그림을 통해 우리는 다음 표의 왼쪽과 같은 수식을 유도했습니다. 이런 형태의 수식을 다원일차연립방정식이라고 합니다. 다원일차연립방정식은 행렬을 이용하면 깔끔하게 정리할 수 있습니다. 행렬 계산식으로 정리하면 다음 표의 오른쪽과 같습니다.

	다원일차연립방정식	행렬 계산식
순 전 파	$y_1 = x_1^* w_1 + x_2^* w_2 + x_3^* w_3 + 1^* b_1$ $y_2 = x_1^* w_4 + x_2^* w_5 + x_3^* w_6 + 1^* b_2$	$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_1 & w_4 \\ w_2 & w_5 \\ w_3 & w_6 \end{bmatrix} + 1 \begin{bmatrix} b_1 & b_2 \end{bmatrix}$
역 전 파	$w_{1E} = y_{1E}^* x_1$ $w_{2E} = y_{1E}^* x_2$ $w_{3E} = y_{1E}^* x_3$ $w_{4E} = y_{2E}^* x_1$ $w_{5E} = y_{2E}^* x_2$ $w_{6E} = y_{2E}^* x_3$ $b_{1E} = y_{1E}^* 1$ $b_{2E} = y_{2E}^* 1$	$\begin{bmatrix} w_{1E} & w_{4E} \\ w_{2E} & w_{5E} \\ w_{3E} & w_{6E} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix}$ $= \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix}$ $\begin{bmatrix} b_{1E} & b_{2E} \end{bmatrix} = 1 \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix}$
인 공 신 경 망 학 습	$w_1 = w_1 - \alpha w_{1E}$ $w_2 = w_2 - \alpha w_{2E}$ $w_3 = w_3 - \alpha w_{3E}$ $w_4 = w_4 - \alpha w_{4E}$ $w_5 = w_5 - \alpha w_{5E}$ $w_6 = w_6 - \alpha w_{6E}$ $b_1 = b_1 - \alpha b_{1E}$ $b_2 = b_2 - \alpha b_{2E}$	$\begin{bmatrix} w_1 & w_4 \\ w_2 & w_5 \\ w_3 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_4 \\ w_2 & w_5 \\ w_3 & w_6 \end{bmatrix} - \alpha \begin{bmatrix} w_{1E} & w_{4E} \\ w_{2E} & w_{5E} \\ w_{3E} & w_{6E} \end{bmatrix}$ $\begin{bmatrix} b_1 & b_2 \end{bmatrix} = \begin{bmatrix} b_1 & b_2 \end{bmatrix} - \alpha \begin{bmatrix} b_{1E} & b_{2E} \end{bmatrix}$

역전파 행렬 계산식에서 다음은 순전파 때 사용된 입력의 전치 행렬입니다.

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

전치행렬은 가로줄과 세로줄이 바뀐 행렬입니다.

인공 신경망 행렬 계산식 정리하기

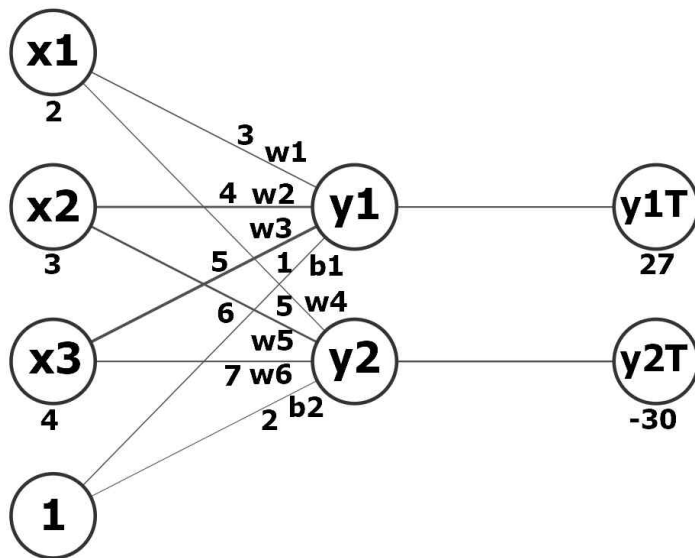
위 수식에서 표현된 행렬들에 다음 표의 왼쪽과 같이 이름을 붙여줍니다. 그러면 위의 행렬

계산식은 다음표의 오른쪽과 같이 정리할 수 있습니다. 오른쪽의 행렬 계산식은 행렬의 크기와 상관없이 성립합니다. 주의할 점은 행렬곱은 순서를 변경하면 안됩니다.

행렬 이름	인공 신경망 행렬 계산식
$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = Y$ $\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} = X$ $\begin{bmatrix} w_1 & w_4 \\ w_2 & w_5 \\ w_3 & w_6 \end{bmatrix} = W$ $\begin{bmatrix} b_1 & b_2 \end{bmatrix} = B$ $\begin{bmatrix} w_{1E} & w_{4E} \\ w_{2E} & w_{5E} \\ w_{3E} & w_{6E} \end{bmatrix} = W_E$ $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T = X^T$ $\begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix} = Y_E$ $\begin{bmatrix} b_{1E} & b_{2E} \end{bmatrix} = B_E$	<p>순전파</p> $Y = XW + B$ <p>역전파</p> $W_E = X^T Y_E$ $B_E = 1 Y_E$ <p>인공 신경망 학습</p> $W = W - \alpha W_E$ $B = B - \alpha B_E$

인공 신경망 구현하기

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



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

$$X = [x_1 \ x_2 \ x_3] = [2 \ 3 \ 4]$$

$$W = \begin{bmatrix} w_1 & w_4 \\ w_2 & w_5 \\ w_3 & w_6 \end{bmatrix} = \begin{bmatrix} 3 & 5 \\ 4 & 6 \\ 5 & 7 \end{bmatrix}$$

$$B = [b_1 \ b_2] = [1 \ 2]$$

$$Y_T = [y_{1T} \ y_{2T}] = [27 \ -30]$$

X, YT를 이용하여 W, B에 대해 학습을 수행해 봅니다.

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

1. 이전 예제 513_2.ino를 514.ino로 저장합니다.

2. 다음과 같이 예제를 수정합니다.

514.ino

```
01 const int NUM_X = 3;
02 const int NUM_Y = 2;
03
04 double X[NUM_X] = {2, 3, 4};
05 double YT[NUM_Y] = {27, -30};
06 double Y[NUM_Y];
07
08 double W[NUM_X][NUM_Y] = {
09     {3, 5},
10     {4, 6},
```

```

11     {5, 7}
12 };
13 double B[NUM_Y] = {1, 2};
14~끝 # 이전 예제와 같습니다.

```

01~13 : NUM_X, NUM_Y, X, YT, W, B를 변경해줍니다.

3. 업로드를 수행합니다.



4. 출력 결과를 확인합니다.

```

epoch = 33
Y = [27.000 -30.000]
E = 0.0000001
YE = [-0.000 0.000]
WE = [
[-0.000 0.001]
[-0.000 0.001]
[-0.000 0.002]
]
BE = [-0.000 0.000]
W = [
[3.533 1.000]
[4.800 0.000]
[1.067 -8.000]
]
B = [1.267 0.000]

epoch = 34
Y = [27.000 -30.000]
E = 0.0000001

```

(34+1)회 째 학습이 완료되는 것을 볼 수 있습니다.

05 행렬 계산식과 1입력 1출력 수식 비교하기

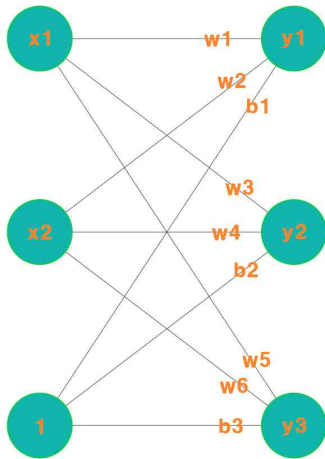
지금까지의 내용을 정리하면 일반적인 인공 신경의 행렬 계산식은 다음과 같습니다. 그리고 입력1 출력1 인공 신경의 수식은 표의 오른쪽 기본 수식과 같습니다. 행렬 계산식의 구조가 기본 수식의 구조와 같은 것을 볼 수 있습니다.

인공 신경망 동작	행렬 계산식	기본 수식
순전파	$Y = XW + B$	$y = x * w + 1 * b$

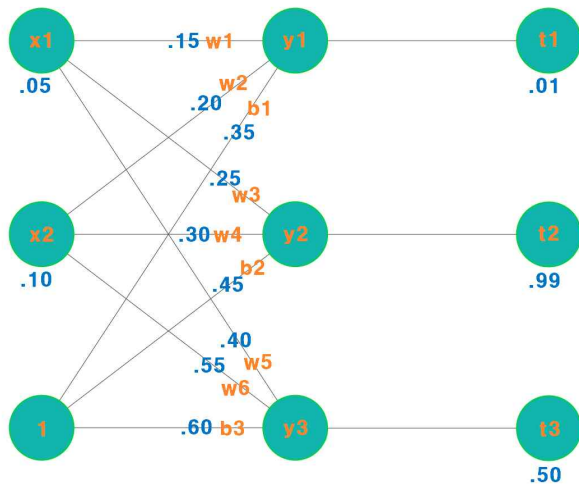
역전파	$W_E = X^T Y_E$ $B_E = 1 Y_E$	$w_E = y_E^* x$ $b_E = y_E^* 1$
인공 신경망 학습	$W_E = X^T Y_E$ $B_E = 1 Y_E$	$w = w - \alpha w_E$ $b = b - \alpha b_E$

연습문제

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

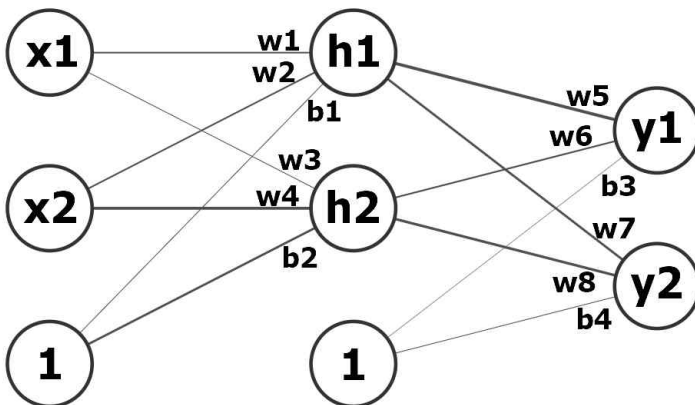


2. 앞에서 구한 행렬 계산식을 이용하여 다음과 같이 초기화된 인공 신경을 배열을 이용하여 구현하고 학습시켜 봅니다. 입력값 X 는 상수로 처리합니다.

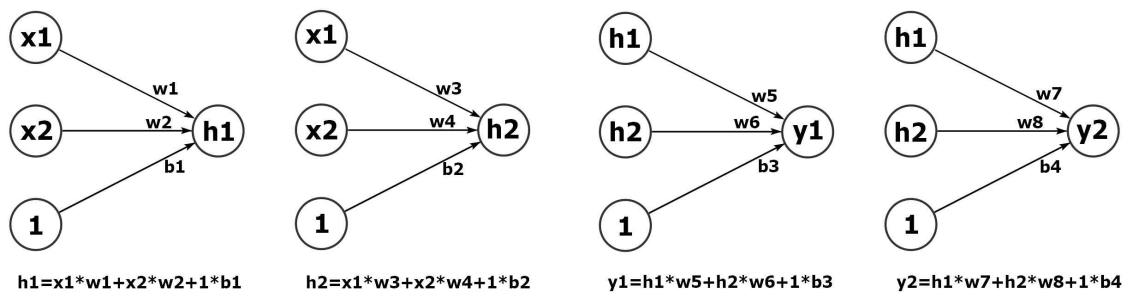


06 2입력 2은닉 2출력 인공 신경망 구현하기

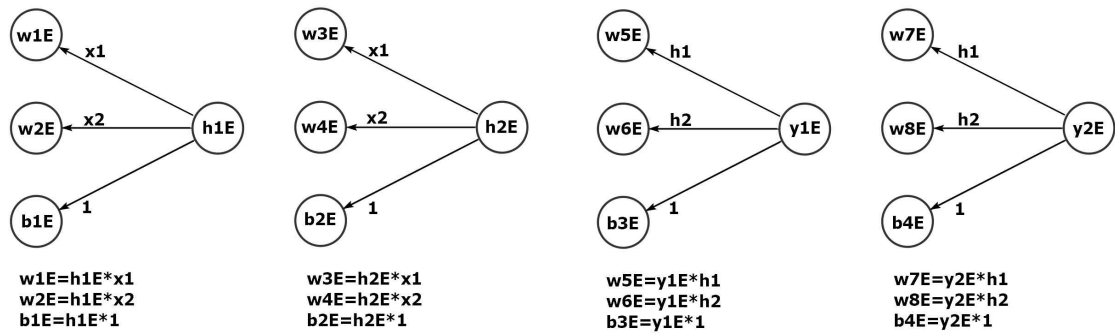
다음 그림은 입력2 은닉2 출력2로 구성된 인공 신경망의 순전파를 나타냅니다.



이 인공 신경망은 다음과 같이 입력2 출력1 인공 신경 4개로 구성됩니다.



다음 그림은 이 인공 신경망의 역전파를 나타냅니다.



은닉층을 포함한 인공 신경망의 경우 은닉층 노드의 역전파 과정이 필요합니다. 즉, 위 그림에서 $h1E$, $h2E$ 에 대한 값이 필요합니다. 다음 그림은 $h1E$, $h2E$ 를 구하는 과정을 나타냅니다.

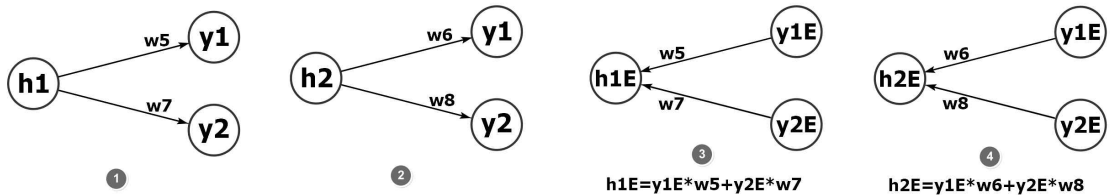


그림 ❶, ❷는 $h1$, $h2$ 의 순전파 과정을 나타냅니다. 그림 ❶에서 $h1$ 은 $w5$, $w7$ 을 통해 $y1$, $y2$ 로 순전파되며, 그림 ❷에서 $h2$ 는 $w6$, $w8$ 을 통해 $y1$, $y2$ 로 순전파됩니다. 그림 ❸, ❹는 $y1E$, $y2E$ 의 $h1E$, $h2E$ 로의 역전파 과정을 나타냅니다. 그림 ❸에서 $y1E$, $y2E$ 는 $w5$, $w7$ 을 통해 $h1E$ 로 역전파되며, 그림 ❹에서 $y1E$, $y2E$ 는 $w6$, $w8$ 을 통해 $h2E$ 로 역전파됩니다. $h1$, $h2$ 가 $w5$, $w7$ 과 $w6$, $w8$ 을 통해 $y1$, $y2$ 로 순전파되는 것처럼 $y1E$, $y2E$ 는 $w5$, $w7$ 과 $w6$, $w8$ 을 통해 $h1E$, $h2E$ 로 역전파됩니다.

행렬 계산식 유도하기

앞의 그림을 통해 우리는 다음 표의 왼쪽과 같은 수식을 유도했습니다. 이런 형태의 수식을 다원일차연립방정식이라고 합니다. 다원일차연립방정식은 행렬을 이용하면 깔끔하게 정리할 수 있습니다. 행렬 계산식으로 정리하면 다음 표의 오른쪽과 같습니다.

	다원일차연립방정식	행렬 계산식
순전파	$\begin{aligned} h_1 &= x_1^* w_1 + x_2^* w_2 + 1^* b_1 \\ h_2 &= x_1^* w_3 + x_2^* w_4 + 1^* b_2 \\ y_1 &= h_1^* w_5 + h_2^* w_6 + 1^* b_3 \\ y_2 &= h_1^* w_7 + h_2^* w_8 + 1^* b_4 \end{aligned}$	$\begin{aligned} \begin{bmatrix} h_1 & h_2 \end{bmatrix} &= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} + 1 \begin{bmatrix} b_1 & b_2 \end{bmatrix} \\ \begin{bmatrix} y_1 & y_2 \end{bmatrix} &= \begin{bmatrix} h_1 & h_2 \end{bmatrix} \begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix} + 1 \begin{bmatrix} b_3 & b_4 \end{bmatrix} \end{aligned}$
은닉층 역전파	$\begin{aligned} h_{1E} &= y_{1E}^* w_5 + y_{2E}^* w_7 \\ h_{2E} &= y_{1E}^* w_6 + y_{2E}^* w_8 \end{aligned}$	$\begin{aligned} \begin{bmatrix} h_{1E} & h_{2E} \end{bmatrix} &= \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix} \begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix} \\ &= \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix} \begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix}^T \end{aligned}$
역전파	$\begin{aligned} w_{1E} &= h_{1E}^* x_1 & w_{3E} &= h_{2E}^* x_1 \\ w_{2E} &= h_{1E}^* x_2 & w_{4E} &= h_{2E}^* x_2 \\ b_{1E} &= h_{1E}^* 1 & b_{2E} &= h_{2E}^* 1 \\ w_{5E} &= y_{1E}^* h_1 & w_{7E} &= y_{2E}^* h_1 \\ w_{6E} &= y_{1E}^* h_2 & w_{8E} &= y_{2E}^* h_2 \\ b_{3E} &= y_{1E}^* 1 & b_{4E} &= y_{2E}^* 1 \end{aligned}$	$\begin{aligned} \begin{bmatrix} w_{1E} & w_{3E} \\ w_{2E} & w_{4E} \end{bmatrix} &= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} h_{1E} & h_{2E} \end{bmatrix} \\ &= \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T \begin{bmatrix} h_{1E} & h_{2E} \end{bmatrix} \\ b_{1E} b_{2E} &= 1 \begin{bmatrix} h_{1E} & h_{2E} \end{bmatrix} \\ \begin{bmatrix} w_{5E} & w_{7E} \\ w_{6E} & w_{8E} \end{bmatrix} &= \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix} \\ &= \begin{bmatrix} h_1 & h_2 \end{bmatrix}^T \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix} \\ b_{3E} b_{4E} &= 1 \begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix} \end{aligned}$
인공신경망 학습	$\begin{aligned} w_1 &= w_1 - \alpha w_{1E} \\ w_2 &= w_2 - \alpha w_{2E} \\ w_3 &= w_3 - \alpha w_{3E} \\ w_4 &= w_4 - \alpha w_{4E} \\ b_1 &= b_1 - \alpha b_{1E} \\ b_2 &= b_2 - \alpha b_{2E} \\ w_5 &= w_5 - \alpha w_{5E} \\ w_6 &= w_6 - \alpha w_{6E} \\ w_7 &= w_7 - \alpha w_{7E} \\ w_8 &= w_8 - \alpha w_{8E} \\ b_3 &= b_3 - \alpha b_{3E} \\ b_4 &= b_4 - \alpha b_{4E} \end{aligned}$	$\begin{aligned} \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} &= \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \alpha \begin{bmatrix} w_{1E} & w_{3E} \\ w_{2E} & w_{4E} \end{bmatrix} \\ \begin{bmatrix} b_1 & b_2 \end{bmatrix} &= \begin{bmatrix} b_1 & b_2 \end{bmatrix} - \alpha \begin{bmatrix} b_{1E} & b_{2E} \end{bmatrix} \\ \begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix} &= \begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix} - \alpha \begin{bmatrix} w_{5E} & w_{7E} \\ w_{6E} & w_{8E} \end{bmatrix} \\ \begin{bmatrix} b_3 & b_4 \end{bmatrix} &= \begin{bmatrix} b_3 & b_4 \end{bmatrix} - \alpha \begin{bmatrix} b_{3E} & b_{4E} \end{bmatrix} \end{aligned}$

은닉층 역전파에서 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.

역전파에서 다음은 순전파 때 사용된 입력의 전치 행렬입니다.

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

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

인공 신경망 행렬 계산식 정리하기

위 수식에서 표현된 행렬들에 다음 표의 왼쪽과 같이 이름을 붙여줍니다. 그러면 위의 행렬 계산식은 다음표의 오른쪽과 같이 정리할 수 있습니다. 오른쪽의 행렬 계산식은 행렬의 크기와 상관없이 성립합니다. 주의할 점은 행렬곱은 순서를 변경하면 안됩니다.

행렬 이름	인공 신경망 행렬 계산식
	<p>순전파</p> $H = XW_H + B_H$ $Y = HW_Y + B_Y$
$\begin{bmatrix} x_1 & x_2 \end{bmatrix} = X$ $\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = W_H$ $\begin{bmatrix} b_1 & b_2 \end{bmatrix} = B_H$ $\begin{bmatrix} h_1 & h_2 \end{bmatrix} = H$ $\begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix} = W_Y$ $\begin{bmatrix} b_3 & b_4 \end{bmatrix} = B_Y$ $\begin{bmatrix} y_{1E} & y_{2E} \end{bmatrix} = Y_E$ $\begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix}^T = W_Y^T$	<p>은닉층 역전파</p> $H_E = Y_E W_Y^T$ <p>역전파</p> $W_{HE} = X^T H_E$ $B_{HE} = 1 H_E$ $W_{YE} = H^T Y_E$ $B_{YE} = 1 Y_E$ <p>인공 신경망 학습</p> $W_H = W_H - \alpha W_{HE}$ $B_H = B_H - \alpha B_{HE}$ $W_Y = W_Y - \alpha W_{YE}$ $B_Y = B_Y - \alpha B_{YE}$

입력 역전파 배열과 함수로 정리하기

다음은 입력 역전파의 행렬 계산식이 일차연립방정식으로 해석되는 과정을 나타냅니다.

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

$$H_E \quad Y_E \quad W^T$$

$$\begin{aligned} h_{1E} &= y_{1E} * w_1 + y_{2E} * w_3 \\ h_{2E} &= y_{1E} * w_2 + y_{2E} * w_4 \end{aligned}$$

행렬의 곱 $YE@W.T$ 는 앞에 오는 YE 행렬의 가로줄 항목, 뒤에 오는 $W.T$ 행렬의 세로줄 항목이 순서대로 곱해진 후, 모두 더해져서 HE 행렬의 항목 하나를 구성합니다. 그래서 YE 행

렬의 가로줄 항목 개수와 W.T 행렬의 세로줄 항목 개수는 같아야 합니다. 또 W.T 행렬의 가로줄 개수와 HE 행렬의 가로줄 개수는 같아야 합니다.

*** 여기서 @ 문자는 행렬의 곱을 나타냅니다. 참고로 파이썬에서는 @문자를 이용하여 행렬의 곱을 수행합니다.

*** 여기서 W.T는 W 행렬의 전치행렬을 나타냅니다.

다음은 입력 역전파의 행렬 계산식을 숫자로 표현한 구체적인 예입니다.

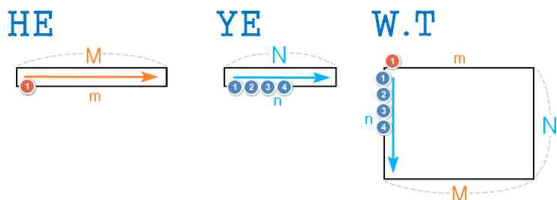
$$\begin{bmatrix} 276 & 328 \end{bmatrix} = \begin{bmatrix} -8 & 60 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$H_E \quad Y_E \quad W^T$$

$$\begin{aligned} 276 &= -8 \times 3 + 60 \times 5 \\ 328 &= -8 \times 4 + 60 \times 6 \end{aligned}$$

❶ 입력 역전파 배열로 정리하기

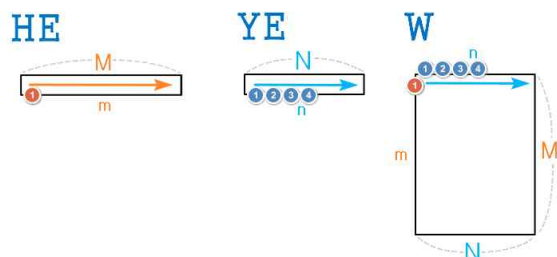
다음은 배열을 이용한 입력 역전파 행렬곱 계산 과정을 나타냅니다.



$$HE[m] += YE[n] * W.T[n][m]$$

0~N인 n, 0~M인 m에 대해 YE[n]은 W.T[n][m]에 곱해져 HE[m]에 누적됩니다.

배열을 이용한 입력 역전파는 다음과 같이 계산할 수도 있습니다.



$$HE[m] += YE[n] * W[m][n]$$

0~N인 n, 0~M인 m에 대해 YE[n]은 W[m][n]에 곱해져 HE[m]에 누적됩니다.

이 그림을 C/C++로 구현하면 다음과 같습니다.

```
for(int m=0;m<M;m++)
    for(int n=0;n<N;n++)
        HE[m] += YE[n] * W[m][n];
```

② 입력 역전파 함수로 정리하기

입력 역전파를 함수로 정리하면 다음과 같습니다.

```
void backward_HE(double *HE, double *W, double *YE, int M, int N) {
    for(int m=0;m<M;m++) HE[m] = 0;
    for(int m=0;m<M;m++)
        for(int n=0;n<N;n++)
            HE[m] += YE[n]*W[m*N+n];
}
```

backward_HE 함수의 매개변수 W는 2차 배열을 1차 배열 형태로 받게 되며, 따라서 함수 내부(4번째 줄)에서 W의 항목을 읽을 때, 1차 배열 형태로 맞추어 접근해야 합니다. HE 배열의 경우 이전에 사용한 경우를 고려하여, 0으로 초기화해 줍니다.

③ 입력 역전파 함수 사용해 보기

여기서는 입력 역전파 함수를 테스트해 봅니다.

1. 다음과 같이 예제를 작성합니다.

512_2.ino

```
01 void backward_HE(double *HE, double *W, double *YE, int M, int N) {
02     for(int m=0;m<M;m++) HE[m] = 0;
03     for(int m=0;m<M;m++)
04         for(int n=0;n<N;n++)
05             HE[m] += YE[n]*W[m*N+n];
06 }
07
08 void print(char * s, double *Y, int N) {
09     printf("%s [", s);
10     for(int n=0;n<N-1;n++) printf("%.3f ", Y[n]);
11     printf("%.3f]\n", Y[N-1]);
12 }
13
```

```

14 const int M = 2;
15 const int N = 2;
16
17 double YE[N] = {-8, 60};
18 double W[M][N] = {
19     {3, 5},
20     {4, 6}
21 };
22 double HE[M] = {0, 0};
23
24 void dnn_test() {
25
26     backward_HE(HE, (double *)W, YE, M, N);
27     print("HE =", HE, M);
28
29 }
30
31 void setup() {
32
33     Serial.begin(115200);
34     delay(1000);
35
36     dnn_test();
37
38 }
39
40 void loop() {
41
42 }

```

01~06 : backward_HE 함수를 정의합니다. 앞에서 정의한 입력 역전파 함수입니다.

08~12 : print 함수를 정의합니다. 일차 배열을 출력하는 함수입니다.

14 : M 상수를 선언하고 2로 초기화합니다. M은 18, 22, 26줄에서 사용합니다.

15 : N 상수를 선언하고 2로 초기화합니다. N은 17, 18, 26줄에서 사용합니다.

17 : 역전파 오차 배열 YE를 선언하고 초기화합니다.

18~21 : 가중치 배열 W를 선언하고 초기화합니다.

22 : 역전파 입력 배열 HE를 선언하고 초기화합니다.

24~29 : dnn_test 함수를 정의합니다.

26 : backward_HE 함수를 호출합니다. 첫 번째 인자로 역전파 출력 HE, 두 번째 인자로 가중치 W, 세 번째 인자로 역전파 오차 YE, 네 번째 인자로 역전파 출력의 개수 M, 다섯 번째 인자로 역전파 입력의 개수 N을 줍니다. 두 번째 인자의 경우 일차 배열로 변환하여 넘겨주어야 합니다.

27 : print 함수를 호출하여 HE 값을 출력합니다.

2. 업로드를 수행합니다.



3. 출력 결과를 확인합니다.

```
HE = [276.000 328.000]
```

딥러닝 라이브러리 정리하기 2

이상의 내용으로 딥러닝 라이브러리를 정리하면 다음과 같습니다. mydnn_1.ino 딥러닝 라이브러리에 입력 역전파 함수를 추가합니다.

mydnn_2.ino

```
01 void forward_Y(double *X, double *W, double *B, double *Y, int M, int N) {
02     for(int n=0;n<N;n++) Y[n] = 0;
03     for(int m=0;m<M;m++)
04         for(int n=0;n<N;n++)
05             Y[n] += X[m]*W[m*N+n];
06     for(int n=0;n<N;n++) Y[n] += B[n];
07 }
08
09 double calculate_MSE(double *Y, double *YT, int N) {
10     double E = 0;
11     for(int n=0;n<N;n++) E += (Y[n]-YT[n])*(Y[n]-YT[n])/2;
12     return E;
13 }
14
15 void backward_YE(double *YE, double *Y, double *YT, int N) {
16     for(int n=0;n<N;n++) YE[n] = Y[n] - YT[n];
17 }
18
19 void backward_WE(double *WE, double *X, double *YE, int M, int N) {
20     for(int m=0;m<M;m++)
21         for(int n=0;n<N;n++)
22             WE[m*N+n] = X[m]*YE[n];
23 }
24
```

```

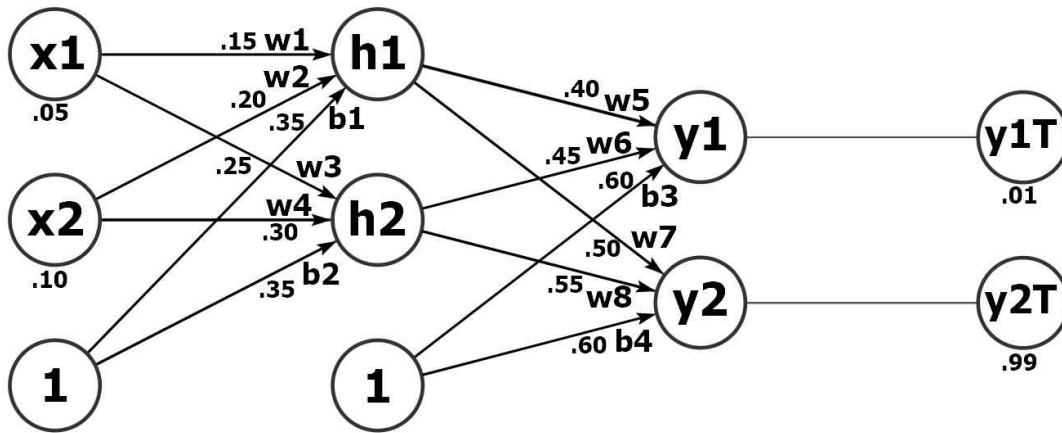
25 void backward_BE(double *BE, double *YE, int N) {
26     for(int n=0;n<N;n++) BE[n] = YE[n];
27 }
28
29 void backward_HE(double *HE, double *W, double *YE, int M, int N) {
30     for(int m=0;m<M;m++) HE[m] = 0;
31     for(int m=0;m<M;m++)
32         for(int n=0;n<N;n++)
33             HE[m] += YE[n]*W[m*N+n];
34 }
35
36 void learning_W(double *W, double lr, double *WE, int M, int N) {
37     for(int m=0;m<M;m++)
38         for(int n=0;n<N;n++)
39             W[m*N+n] -= lr * WE[m*N+n];
40 }
41
42 void learning_B(double *B, double lr, double *BE, int N) {
43     for(int n=0;n<N;n++) B[n] -= lr * BE[n];
44 }
45
46 void print(char * s, double *Y, int N) {
47     printf("%s [", s);
48     for(int n=0;n<N-1;n++) printf("%.3f ", Y[n]);
49     printf("%.3f]\n", Y[N-1]);
50 }
51
52 void print(char * s, double *W, int M, int N) {
53     printf("%s [\n", s);
54     for(int m=0;m<M;m++) {
55         printf("[", s);
56         for(int n=0;n<N-1;n++)
57             printf("%.3f ", W[m*N+n]);
58         printf("%.3f]\n", W[m*N+N-1]);
59     }
60     printf("]\n");
61 }

```

29~34 : backward_HE 함수를 추가합니다.

인공 신경망 구현하기

지금까지 정리한 수식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



이 그림에서 입력값 X, 가중치 WH, WY, 편향 BH, BY, 목표값 YT는 다음과 같습니다.

$$X = [x_1 \ x_2] = [.05 \ .10]$$

$$W_H = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .15 & .25 \\ .20 & .30 \end{bmatrix}$$

$$B_H = [b_1 \ b_2] = [.35 \ .35]$$

$$W_Y = \begin{bmatrix} w_5 & w_7 \\ w_6 & w_8 \end{bmatrix} = \begin{bmatrix} .40 & .50 \\ .45 & .55 \end{bmatrix}$$

$$B_Y = [b_3 \ b_4] = [.60 \ .60]$$

$$Y_T = [y_{1T} \ y_{2T}] = [.01 \ .99]$$

X, YT를 이용하여 WH, WY, BH, BY에 대해 학습을 수행해 봅니다.

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

1. 다음과 같이 예제를 작성합니다.

518.ino

```
01 const int NUM_X = 2;
02 const int NUM_H = 2;
03 const int NUM_Y = 2;
04
05 double X[NUM_X] = {0.05, 0.10};
06 double YT[NUM_Y] = {0.01, 0.99};
07 double H[NUM_H];
08 double Y[NUM_Y];
09
10 double WH[NUM_X][NUM_H] = {
```

```

11     {0.15, 0.25},
12     {0.20, 0.30}
13 };
14 double BH[NUM_H] = {0.35, 0.35};
15 double WY[NUM_H][NUM_Y] = {
16     {0.40, 0.50},
17     {0.45, 0.55}
18 };
19 double BY[NUM_Y] = {0.60, 0.60};
20
21 double YE[NUM_Y];
22 double HE[NUM_H];
23 double WYE[NUM_H][NUM_Y];
24 double BYE[NUM_Y];
25 double WHE[NUM_X][NUM_H];
26 double BHE[NUM_H];
27
28 void dnn_test() {
29
30     for(int epoch=0;epoch<1000;epoch++) {
31
32         printf("\nepoch = %d\n", epoch);
33
34         forward_Y(X, (double *)WH, BH, H, NUM_X, NUM_H);
35         print("H =", H, NUM_H);
36
37         forward_Y(H, (double *)WY, BY, Y, NUM_H, NUM_Y);
38         print("Y =", Y, NUM_Y);
39
40         double E = calculate_MSE(Y, YT, NUM_Y);
41         printf("E = %.7f\n", E);
42
43         if(E < 0.0000001)
44             break;
45
46         backward_YE(YE, Y, YT, NUM_Y);
47         print("YE =", YE, NUM_Y);
48
49         backward_WE((double *)WYE, H, YE, NUM_H, NUM_Y);
50         print("WYE =", (double *)WYE, NUM_H, NUM_Y);

```

```
51
52     backward_BE(BYE, YE, NUM_Y);
53     print("BYE =", BYE, NUM_Y);
54
55     backward_HE(HE, (double *)WY, YE, NUM_H, NUM_Y);
56     print("HE =", HE, NUM_H);
57
58     backward_WE((double *)WHE, X, HE, NUM_X, NUM_H);
59     print("WHE =", (double *)WHE, NUM_X, NUM_H);
60
61     backward_BE(BHE, HE, NUM_H);
62     print("BHE =", BHE, NUM_H);
63
64     double lr = 0.01;
65
66     learning_W((double *)WY, lr, (double *)WYE, NUM_H, NUM_Y);
67     print("WY =", (double *)WY, NUM_H, NUM_Y);
68
69     learning_B(BY, lr, BYE, NUM_Y);
70     print("BY =", BY, NUM_Y);
71
72     learning_W((double *)WH, lr, (double *)WHE, NUM_X, NUM_H);
73     print("WH =", (double *)WH, NUM_X, NUM_H);
74
75     learning_B(BH, lr, BHE, NUM_H);
76     print("BH =", BH, NUM_H);
77 }
78
79 }
80
81 void setup() {
82
83     Serial.begin(115200);
84     delay(1000);
85
86     dnn_test();
87
88 }
89
90 void loop() {
```

```
91  
92 }
```

2. 업로드를 수행합니다.



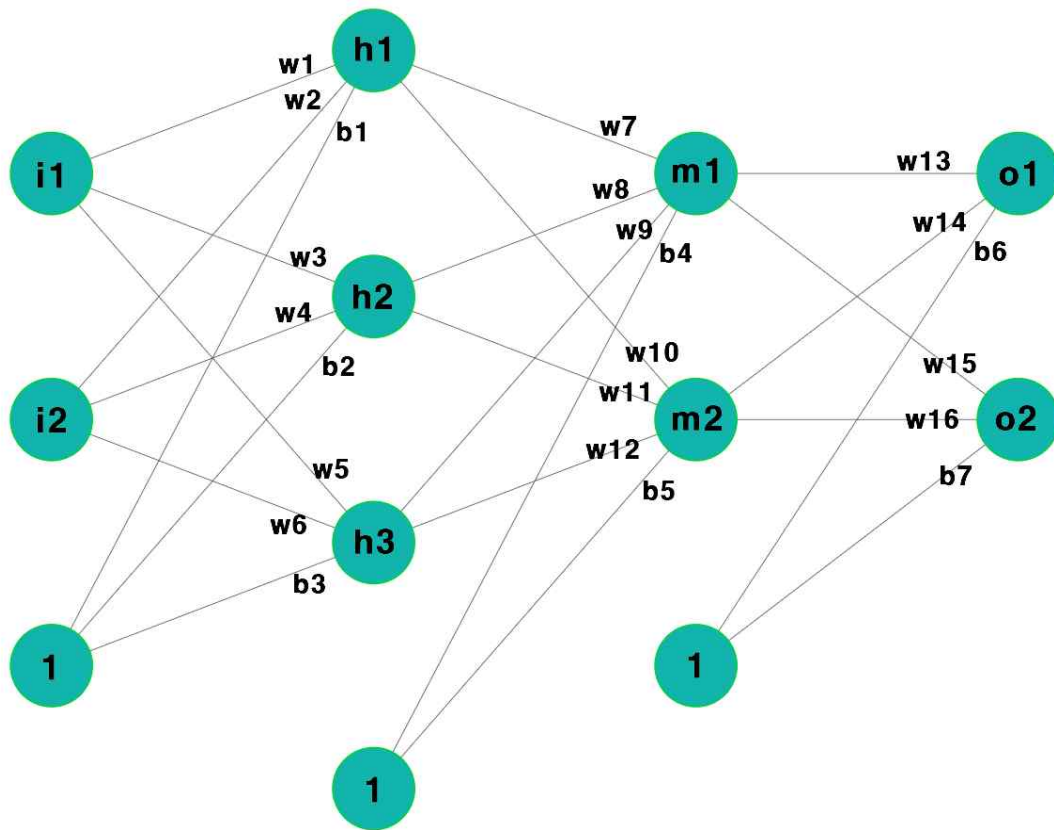
3. 출력 결과를 확인합니다.

```
epoch = 664  
H = [0.239 0.226]  
Y = [0.010 0.990]  
E = 0.0000001  
YE = [0.000 -0.000]  
WYE = [  
[0.000 -0.000]  
[0.000 -0.000]  
]  
BYE = [0.000 -0.000]  
HE = [-0.000 0.000]  
WHE = [  
[-0.000 0.000]  
[-0.000 0.000]  
]  
BHE = [-0.000 0.000]  
WY = [  
[0.203 0.533]  
[0.253 0.583]  
]  
BY = [-0.095 0.730]  
WH = [  
[0.143 0.242]  
[0.186 0.284]  
]  
BH = [0.213 0.186]  
  
epoch = 665  
H = [0.239 0.226]  
Y = [0.010 0.990]  
E = 0.0000001
```

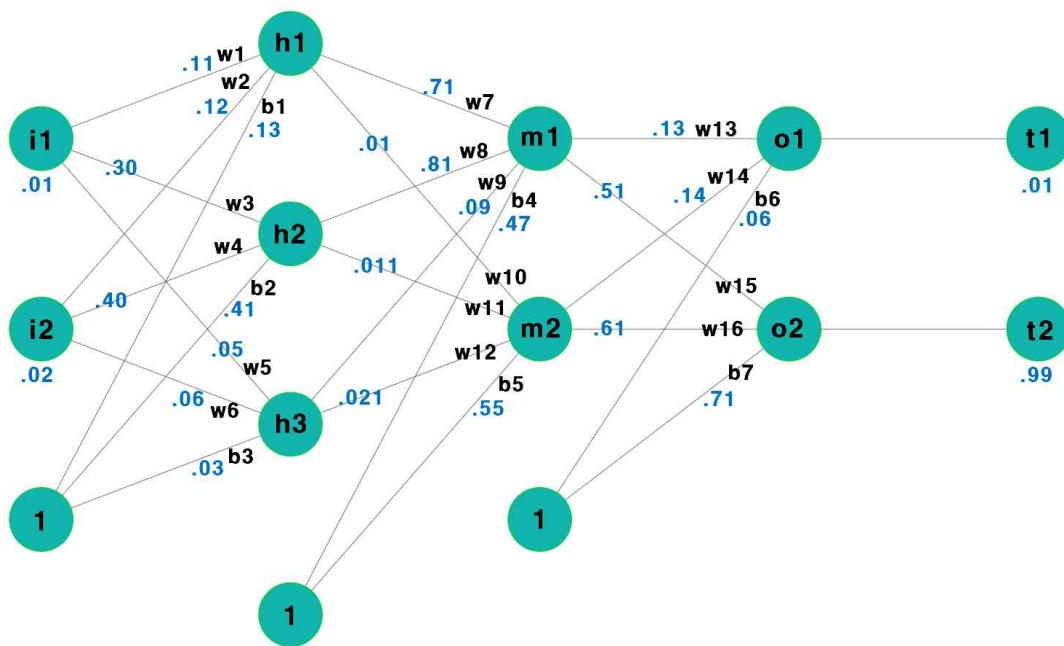
(665+1)회 째 학습이 완료되는 것을 볼 수 있습니다.

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 행렬 계산식을 구합니다.



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



02 활성화 함수 추가하기

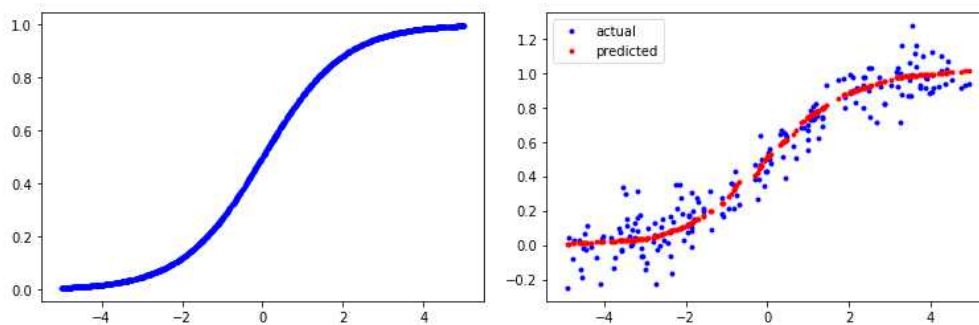
일반적으로 인공 신경의 출력단에는 활성화 함수가 추가됩니다. 활성화 함수를 추가하면 입력된 데이터에 대해 복잡한 패턴의 학습이 가능해집니다. 또 인공 신경의 출력값을 어떤 범위로 제한할 수도 있습니다. 여기서는 주로 사용되는 몇 가지 활성화 함수를 살펴보고, 활성화 함수가 필요한 이유에 대해서도 살펴봅니다. 그리고 활성화 함수를 추가한 인공 신경망을 구현해 봅니다.

01 활성화 함수 살펴보기

우리는 앞에서 다음과 같은 활성화 함수를 직접 그려보았습니다.

sigmoid 함수

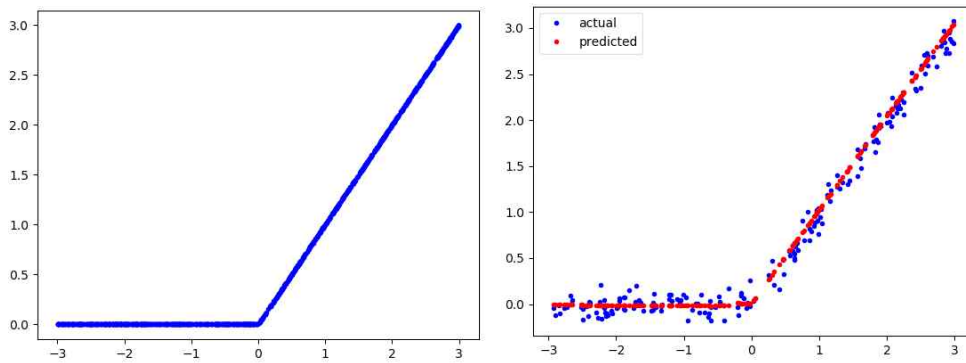
다음은 sigmoid 함수에 대한 그래프와 인공 신경망 학습 후, 예측 그래프입니다.



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

relu 함수

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



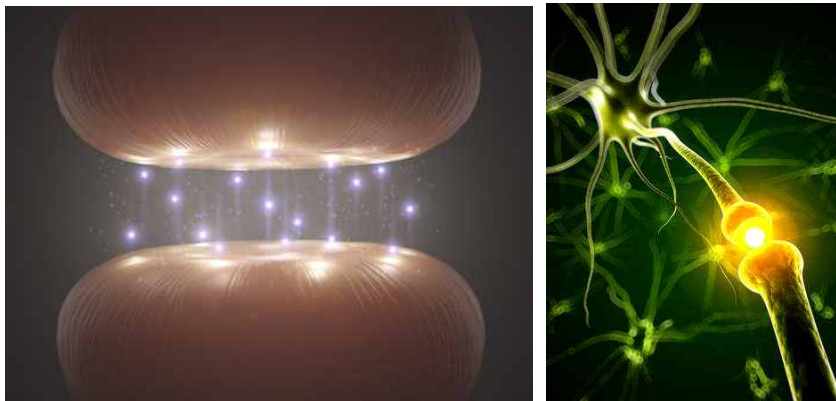
$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \quad (-3 \leq x \leq 3)$$

02 활성화 함수의 필요성

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

활성화 함수는 무엇인가요?

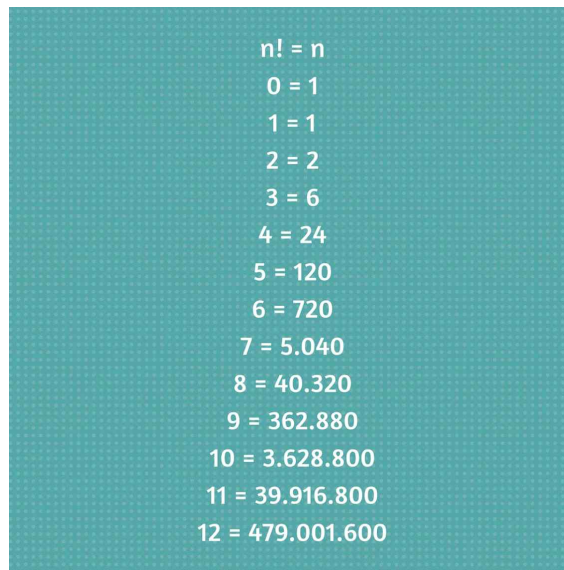
활성화 함수는 인공 신경망에 더해져 복잡한 패턴을 학습하게 해줍니다. 즉, 다양한 형태의 입력값에 대해 신경망을 거쳐 나온 출력값을 우리가 원하는 목표값에 가깝게 해 주기가 더 쉬워집니다. 우리 두뇌에 있는 생체 신경과 비교할 때, 활성화 함수는 신경 말단에서 다음 신경으로 전달될 신호를 결정하는 시냅스와 같은 역할을 합니다. 시냅스는 이전 신경 세포가 내보내는 출력 신호를 받아 다음 신경 세포가 받아들일 수 있는 입력 신호로 형태를 변경합니다. 마찬가지로 활성화 함수는 이전 인공 신경이 내보내는 출력 신호를 받아 다음 인공 신경이 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 $(x*w+b)$ 이기 때문입니다. 여기서 x 는 입력, w 는 인공 신경의 가중치, b 는 그것에 더해지는 편향입니다. $(x*w+b)$ 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 $(x*w+b)$ 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. 다음은 factorial 연산의 예를 보여줍니다.



$n!$	$= n$
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 점점더 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

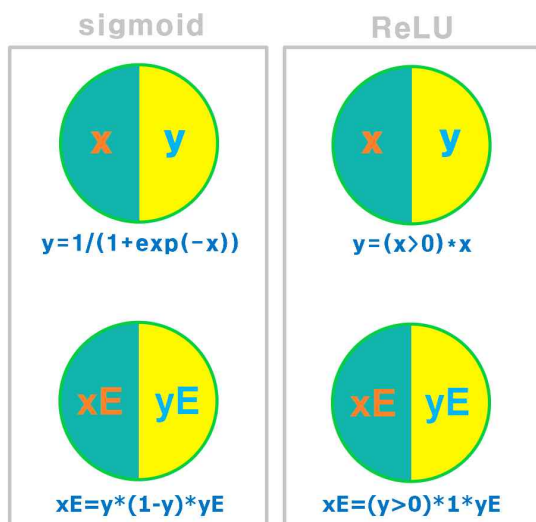
소프트맥스 활성화 함수는 시그모이드 활성화 함수가 더욱 일반화된 형태입니다. 이것은 다중 클래스 분류 문제에 사용됩니다. 시그모이드 함수와 비슷하게 이것은 0에서 1사이의 값들을 생성합니다. 소프트맥스 함수는 은닉층에서는 사용하지 않으며, 다중 분류 모델에서 출력층에서만 사용합니다.

③ 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 0에 아주 가까운 값으로 고르게 할당하여 해결할 수 있습니다. 일반적으로 은닉층에는 ReLU 함수를 적용하고, 출력층은 시그모이드 함수나 소프트맥스 함수를 적용합니다.

03 활성화 함수의 순전파와 역전파

다음 그림은 sigmoid, ReLU 활성화 함수의 순전파와 역전파 수식을 나타냅니다.



*** 역전파 수식 계산시 순전파의 y값도 사용됩니다. 역전파 수식은 편미분을 이용하며 유도하며 유도 과정은 생략합니다.

- sigmoid 함수의 경우 순전파 출력 y값이 0이나 1에 가까울수록 역전파 xE값은 0에 가까워집니다. 순전파 출력 y값이 0이나 1에 가깝다는 것은 순전파 입력값 x의 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_E 값이 x_E 로 전달되며, 출력값 y 가 0보다 작거나 같으면 x_E 로 0이 전달됩니다. 이 경우 x_E 에서 전 단계의 모든 노드로 전달되는 역전파 값은 0이 됩니다.

이상에서 필요한 수식을 정리하면 다음과 같습니다.

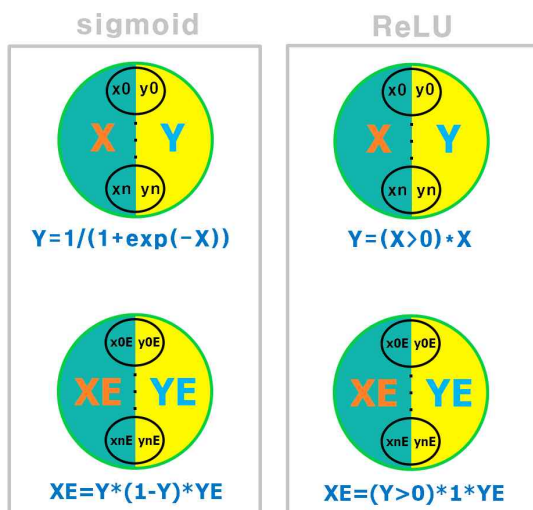
sigmoid 함수 순전파와 역전파

$$y = \frac{1}{1 + e^{-x}} \quad x_E = y^*(1 - y)^*y_E$$

ReLU 함수 순전파와 역전파

$$y = (x > 0)^*x \quad x_E = (y > 0)^*1^*y_E$$

다음 그림은 활성화 함수의 순전파와 역전파의 행렬 기반 수식을 나타냅니다.



이 그림에서 X , Y 는 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 , Y 는 하나의 층내에서 활성화 함수의 입력과 출력 행렬을 의미합니다.

이상에서 활성화 함수의 행렬 계산식을 정리하면 다음과 같습니다.

sigmoid 함수 순전파와 역전파

$$Y = \frac{1}{1 + e^{-X}} \quad X_E = Y * (1 - Y) * Y_E$$

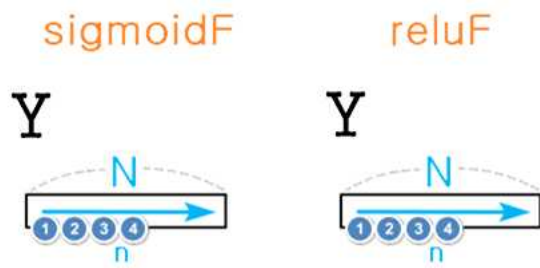
ReLU 함수 순전파와 역전파

$$Y = (X > 0) * X \quad X_E = (Y > 0) * 1 * Y_E$$

순전파 활성화 함수 정리하기

❶ 순전파 활성화 함수 배열로 정리하기

다음은 배열을 이용한 순전파 활성화 함수 계산 과정을 나타냅니다.



이 그림을 C/C++로 구현하면 다음과 같습니다.

sigmoid

```
for(int n=0;n<N;n++)
    Y[n] = 1/(1 + exp(-Y[n]));
```

ReLU

```
for(int n=0;n<N;n++)
    Y[n] = (Y[n]>0?1:0)*Y[n];
```

❷ 순전파 활성화 함수 함수로 정리하기

순전파 활성화 함수를 함수로 정리하면 다음과 같습니다.

sigmoid

```
void sigmoid_Y(double *Y, int N) {
    for(int n=0;n<N;n++) Y[n] = 1/(1 + exp(-Y[n]));
}
```

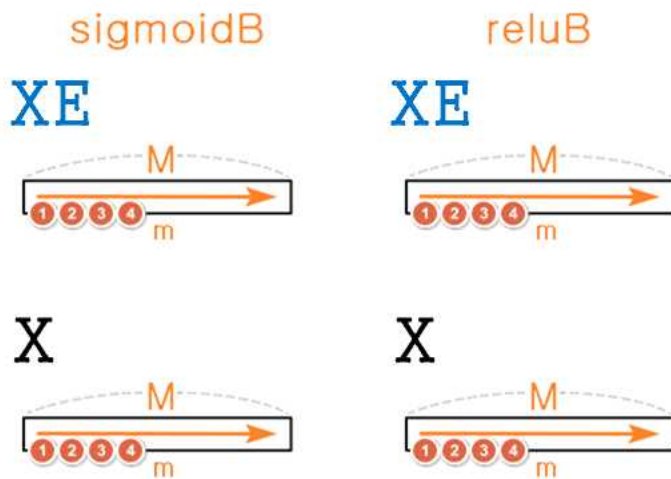
ReLU

```
void relu_Y(double *Y, int N) {
    for(int n=0;n<N;n++) Y[n] = (Y[n]>0?1:0)*Y[n];
}
```

역전파 활성화 함수 정리하기

❶ 역전파 활성화 함수 배열로 정리하기

다음은 배열을 이용한 역전파 활성화 함수 계산 과정을 나타냅니다.



이 그림을 C/C++로 구현하면 다음과 같습니다.

sigmoid

```
for(int m=0;m<M;m++)
    XE[m] = XE[m]*X[m]*(1-X[m]);
```

ReLU

```
for(int m=0;m<M;m++)
    XE[m] = XE[m]*(X[m]>0?1:0)*1;
```

❷ 역전파 활성화 함수 함수로 정리하기

역전파 활성화 함수를 함수로 정리하면 다음과 같습니다.

sigmoid

```
void sigmoid_XE(double *XE, double *X, int M) {  
    for(int m=0;m<M;m++) XE[m] = XE[m]*X[m]*(1-X[m]);  
}
```

ReLU

```
void relu_XE(double *XE, double *X, int M) {  
    for(int m=0;m<M;m++) XE[m] = XE[m]*(X[m]>0?1:0)*1;  
}
```

딥러닝 라이브러리 정리하기 3

이상의 내용으로 딥러닝 라이브러리를 정리하면 다음과 같습니다. mydnn_2.ino 딥러닝 라이브러리에 활성화 함수 순전파 역전파 함수를 추가합니다.

mydnn_3.ino

```
01 void forward_Y(double *X, double *W, double *B, double *Y, int M, int N) {  
02     for(int n=0;n<N;n++) Y[n] = 0;  
03     for(int m=0;m<M;m++)  
04         for(int n=0;n<N;n++)  
05             Y[n] += X[m]*W[m*N+n];  
06     for(int n=0;n<N;n++) Y[n] += B[n];  
07 }  
08  
09 double calculate_MSE(double *Y, double *YT, int N) {  
10     double E = 0;  
11     for(int n=0;n<N;n++) E += (Y[n]-YT[n])*(Y[n]-YT[n])/2;  
12     return E;  
13 }  
14  
15 void backward_YE(double *YE, double *Y, double *YT, int N) {  
16     for(int n=0;n<N;n++) YE[n] = Y[n] - YT[n];  
17 }  
18  
19 void backward_WE(double *WE, double *X, double *YE, int M, int N) {  
20     for(int m=0;m<M;m++)  
21         for(int n=0;n<N;n++)
```



```

22         WE[m*N+n] = X[m]*YE[n];
23     }
24
25 void backward_BE(double *BE, double *YE, int N) {
26     for(int n=0;n<N;n++) BE[n] = YE[n];
27 }
28
29 void backward_HE(double *HE, double *W, double *YE, int M, int N) {
30     for(int m=0;m<M;m++) HE[m] = 0;
31     for(int m=0;m<M;m++)
32         for(int n=0;n<N;n++)
33             HE[m] += YE[n]*W[m*N+n];
34 }
35
36 void learning_W(double *W, double lr, double *WE, int M, int N) {
37     for(int m=0;m<M;m++)
38         for(int n=0;n<N;n++)
39             W[m*N+n] -= lr * WE[m*N+n];
40 }
41
42 void learning_B(double *B, double lr, double *BE, int N) {
43     for(int n=0;n<N;n++) B[n] -= lr * BE[n];
44 }
45
46 void sigmoid_Y(double *Y, int N) {
47     for(int n=0;n<N;n++) Y[n] = 1/(1 + exp(-Y[n]));
48 }
49
50 void sigmoid_XE(double *XE, double *X, int M) {
51     for(int m=0;m<M;m++) XE[m] = XE[m]*X[m]*(1-X[m]);
52 }
53
54 void relu_Y(double *Y, int N) {
55     for(int n=0;n<N;n++) Y[n] = (Y[n]>0?1:0)*Y[n];
56 }
57
58 void relu_XE(double *XE, double *X, int M) {
59     for(int m=0;m<M;m++) XE[m] = XE[m]*(X[m]>0?1:0)*1;
60 }
61

```

```

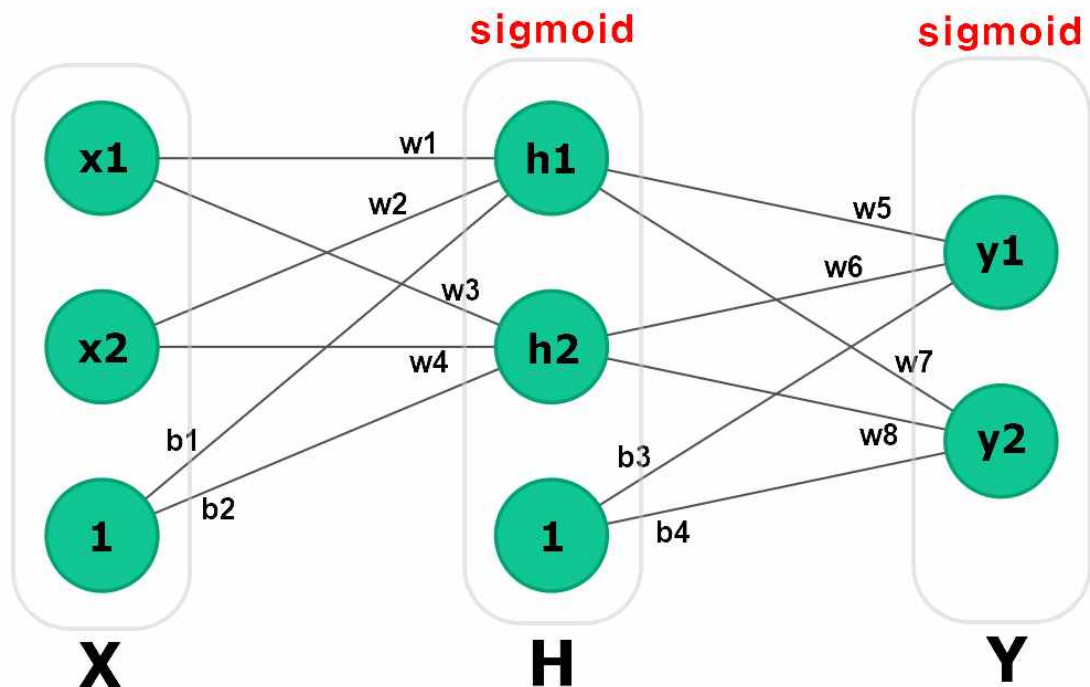
62 void print(char * s, double *Y, int N) {
63     printf("%s [", s);
64     for(int n=0;n<N-1;n++) printf("%.3f ", Y[n]);
65     printf("%.3f\\n", Y[N-1]);
66 }
67
68 void print(char * s, double *W, int M, int N) {
69     printf("%s \\n", s);
70     for(int m=0;m<M;m++) {
71         printf("[", s);
72         for(int n=0;n<N-1;n++)
73             printf("%.3f ", W[m*N+n]);
74         printf("%.3f\\n", W[m*N+N-1]);
75     }
76     printf("\\n");
77 }

```

46~60 : 활성화 함수의 순전파 역전파 함수를 추가합니다.

sigmoid 함수 적용해 보기

지금까지 정리한 행렬 계산식을 구현을 통해 살펴봅니다. 다음 그림을 살펴봅니다.



1. 이전 예제 518.ino를 519.ino로 저장합니다.

2. 다음과 같이 예제를 수정합니다.

519.ino

```
01 const int NUM_X = 2;
02 const int NUM_H = 2;
03 const int NUM_Y = 2;
04
05 double X[NUM_X] = {0.05, 0.10};
06 double YT[NUM_Y] = {0.01, 0.99};
07 double H[NUM_H];
08 double Y[NUM_Y];
09
10 double WH[NUM_X][NUM_H] = {
11     {0.15, 0.25},
12     {0.20, 0.30}
13 };
14 double BH[NUM_H] = {0.35, 0.35};
15 double WY[NUM_H][NUM_Y] = {
16     {0.40, 0.50},
17     {0.45, 0.55}
18 };
19 double BY[NUM_Y] = {0.60, 0.60};
20
21 double YE[NUM_Y];
22 double HE[NUM_H];
23 double WYE[NUM_H][NUM_Y];
24 double BYE[NUM_Y];
25 double WHE[NUM_X][NUM_H];
26 double BHE[NUM_H];
27
28 void dnn_test() {
29
30     for(int epoch=0;epoch<1000;epoch++) {
31
32         printf("\nepoch = %d\n", epoch);
33
34         forward_Y(X, (double *)WH, BH, H, NUM_X, NUM_H);
35         sigmoid_Y(H, NUM_H);
36
```

```

37     forward_Y(H, (double *)WY, BY, Y, NUM_H, NUM_Y);
38     sigmoid_Y(Y, NUM_Y);
39     print("Y =", Y, NUM_Y);
40
41     double E = calculate_MSE(Y, YT, NUM_Y);
42
43     if(E < 0.0000001)
44         break;
45
46     backward_YE(YE, Y, YT, NUM_Y);
47     sigmoid_XE(YE, Y, NUM_Y);
48
49     backward_WE((double *)WYE, H, YE, NUM_H, NUM_Y);
50     backward_BE(BYE, YE, NUM_Y);
51
52     backward_HE(HE, (double *)WY, YE, NUM_H, NUM_Y);
53     sigmoid_XE(HE, H, NUM_H);
54
55     backward_WE((double *)WHE, X, HE, NUM_X, NUM_H);
56     backward_BE(BHE, HE, NUM_H);
57
58     double lr = 0.01;
59
60     learning_W((double *)WY, lr, (double *)WYE, NUM_H, NUM_Y);
61     learning_B(BY, lr, BYE, NUM_Y);
62
63     learning_W((double *)WH, lr, (double *)WHE, NUM_X, NUM_H);
64     learning_B(BH, lr, BHE, NUM_H);
65
66 }
67
68 }
69
70 void setup() {
71
72     Serial.begin(115200);
73     delay(1000);
74
75     dnn_test();
76

```

```

77 }
78
79 void loop() {
80
81 }

```

35 : 은닉층 H에 순전파 시그모이드 활성화 함수를 적용합니다.

38 : 출력층 Y에 순전파 시그모이드 활성화 함수를 적용합니다.

47 : 역출력층 YE에 역전파 시그모이드 활성화 함수를 적용합니다.

50 : 역은닉층 HE에 역전파 시그모이드 활성화 함수를 적용합니다.

3. 업로드를 수행합니다.



4. 출력 결과를 확인합니다.

```

epoch = 998
Y = [0.306 0.844]

epoch = 999
Y = [0.306 0.844]

```

(999+1)번째에 y1, y2가 각각 0.306, 0.844가 됩니다.

5. 다음과 같이 예제를 수정합니다.

```

30 for(int epoch=0;epoch<10000;epoch++) {

```

```

32 if(epoch%10==9) printf("\nepoch = %d\n", epoch);

```

```

39 if(epoch%10==9) print("Y =", Y, NUM_Y);

```

6. 업로드를 수행합니다.



7. 출력 결과를 확인합니다.

```
epoch = 9989
Y = [0.063 0.943]

epoch = 9999
Y = [0.063 0.943]
```

(9999+1)번째에 y1, y2가 각각 0.063, 0.943이 됩니다.

8. 다음과 같이 예제를 수정합니다.

```
30     for(int epoch=0;epoch<100000;epoch++) {
```

```
32         if(epoch%100==99) printf("\nepoch = %d\n", epoch);
```

```
39         if(epoch%100==99) print("Y =", Y, NUM_Y);
```

9. 업로드를 수행합니다.



10. 출력 결과를 확인합니다.

```
epoch = 99899
Y = [0.020 0.981]

epoch = 99999
Y = [0.020 0.981]
```

(99999+1)번째에 y1, y2가 각각 0.020, 0.981이 됩니다.

11. 다음과 같이 예제를 수정합니다.

```
30     for(int epoch=0;epoch<1000000;epoch++) {
```

```
32         if(epoch%1000==999) printf("\nepoch = %d\n", epoch);
```

```
39         if(epoch%1000==999) print("Y =", Y, NUM_Y);
```

12. 업로드를 수행합니다.



13. 출력 결과를 확인합니다.

```
epoch = 998999
Y = [0.010 0.990]

epoch = 999999
Y = [0.010 0.990]
```

(999999+1)번째에 y1, y2가 각각 0.010, 0.990이 됩니다. 아직 오차는 0.0000001(천만분의 1)보다 큼니다.

14. 다음과 같이 예제를 수정합니다.

```
30     for(int epoch=0;epoch<1000000;epoch++) {
```

```
32         if(epoch%10000==9999) printf("\nepoch = %d\n", epoch);
```

```
39         if(epoch%10000==9999) print("Y =", Y, NUM_Y);
```

```
43             if(E < 0.0000001) {
44                 printf("\nepoch = %d\n", epoch);
45                 print("Y =", Y, NUM_Y);
46                 break;
47             }
```

15. 업로드를 수행합니다.



16. 출력 결과를 확인합니다.

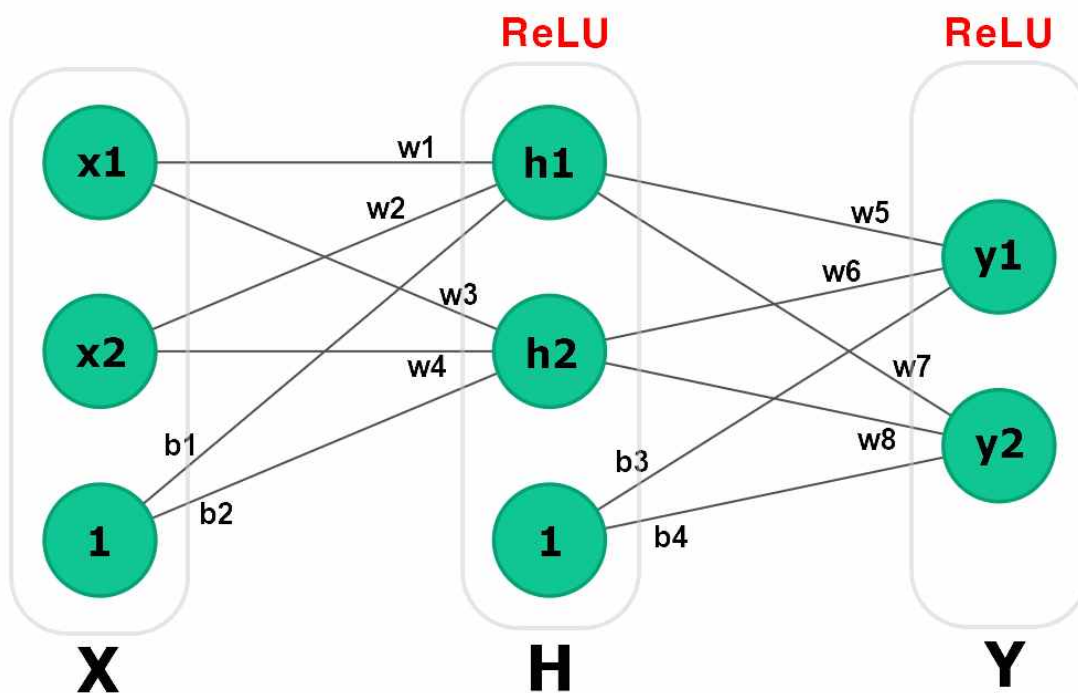
```
epoch = 1069999
Y = [0.010 0.990]

epoch = 1078011
Y = [0.010 0.990]
```

(1078011+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. y1, y2는 각각 0.010, 0.990이 된 상태입니다.

ReLU 함수 적용해 보기

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



1. 이전 예제 519_2.ino를 519_3.ino로 저장합니다.

2. 다음과 같이 예제를 수정합니다.

519_3.ino

```
34      forward_Y(X, (double *)WH, BH, H, NUM_X, NUM_H);
35      relu_Y(H, NUM_H);
36
37      forward_Y(H, (double *)WY, BY, Y, NUM_H, NUM_Y);
38      relu_Y(Y, NUM_Y);
```

35 : 은닉층 H에 순전파 ReLU 활성화 함수를 적용합니다.

38 : 출력층 Y에 순전파 ReLU 활성화 함수를 적용합니다.

```
49      backward_YE(YE, Y, YT, NUM_Y);
50      relu_XE(YE, Y, NUM_Y);
```

50 : 역출력층 YE에 역전파 ReLU 활성화 함수를 적용합니다.

```
55      backward_HE(HE, (double *)WY, YE, NUM_H, NUM_Y);
56      relu_XE(HE, H, NUM_H);
```

56 : 역은닉층 HE에 역전파 ReLU 활성화 함수를 적용합니다.

3. 업로드를 수행합니다.



4. 출력 결과를 확인합니다.

```
epoch = 665  
Y = [0.010 0.990]
```

(665+1)번째에 오차가 0.0000001(천만분의 1)보다 작아집니다. y1, y2는 각각 0.010, 0.990이 된 상태입니다. sigmoid, 함수보다 결과가 훨씬 더 빨리 나오는 것을 볼 수 있습니다.

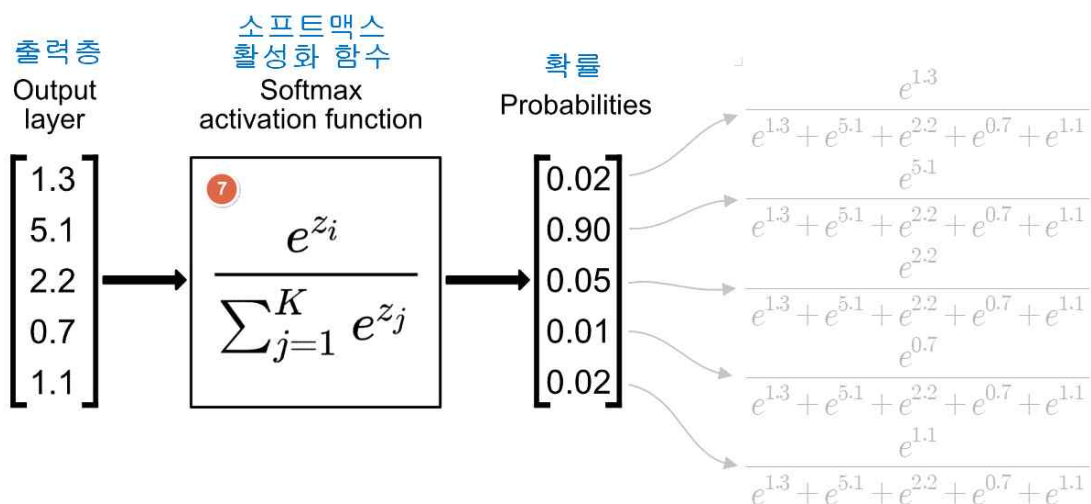
*** ReLU 함수를 사용하기 위해서는 가중치와 편향 값이 0.0에 가까워야 하며, 입력값도 0~1.0 사이의 값을 사용해야 합니다.

04 출력층에 softmax 함수 적용해 보기

이전 단원에서 우리는 은닉 신경과 출력 신경에 시그모이드(sigmoid), 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.

softmax와 cross entropy

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 오

차 함수와 같이 사용됩니다.

$$E = - \sum_k (y_{kT} * \log(y_k))$$

우리는 앞에서 다음과 같은 평균 제곱 오차 함수를 살펴보았습니다.

$$E = \sum_k \frac{1}{2} (y_k - y_{kT})^2$$

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

$$y_{kE} = y_k - y_{kT}$$

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

$$y_{kE} = y_k - y_{kT}$$

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

softmax 함수와 cross entropy error 함수 정의하기

먼저 softmax 함수와 cross entropy error 함수를 정의해 봅시다. 다음과 같이 mydnn_3.ino 파일에 softmax_Y 함수와 calculate_CEE 함수를 추가합니다.

mydnn_4.ino

```
01 void forward_Y(double *X, double *W, double *B, double *Y, int M, int N) {
02     for(int n=0;n<N;n++) Y[n] = 0;
03     for(int m=0;m<M;m++)
04         for(int n=0;n<N;n++)
05             Y[n] += X[m]*W[m*N+n];
06     for(int n=0;n<N;n++) Y[n] += B[n];
07 }
```

```

08
09 double calculate_MSE(double *Y, double *YT, int N) {
10     double E = 0;
11     for(int n=0;n<N;n++) E += (Y[n]-YT[n])*(Y[n]-YT[n])/2;
12     return E;
13 }
14
15 void backward_YE(double *YE, double *Y, double *YT, int N) {
16     for(int n=0;n<N;n++) YE[n] = Y[n] - YT[n];
17 }
18
19 void backward_WE(double *WE, double *X, double *YE, int M, int N) {
20     for(int m=0;m<M;m++)
21         for(int n=0;n<N;n++)
22             WE[m*N+n] = X[m]*YE[n];
23 }
24
25 void backward_BE(double *BE, double *YE, int N) {
26     for(int n=0;n<N;n++) BE[n] = YE[n];
27 }
28
29 void backward_HE(double *HE, double *W, double *YE, int M, int N) {
30     for(int m=0;m<M;m++) HE[m] = 0;
31     for(int m=0;m<M;m++)
32         for(int n=0;n<N;n++)
33             HE[m] += YE[n]*W[m*N+n];
34 }
35
36 void learning_W(double *W, double lr, double *WE, int M, int N) {
37     for(int m=0;m<M;m++)
38         for(int n=0;n<N;n++)
39             W[m*N+n] -= lr * WE[m*N+n];
40 }
41
42 void learning_B(double *B, double lr, double *BE, int N) {
43     for(int n=0;n<N;n++) B[n] -= lr * BE[n];
44 }
45
46 void sigmoid_Y(double *Y, int N) {
47     for(int n=0;n<N;n++) Y[n] = 1/(1 + exp(-Y[n]));

```

```

48 }
49
50 void sigmoid_XE(double *XE, double *X, int M) {
51     for(int m=0;m<M;m++) XE[m] = XE[m]*X[m]*(1-X[m]);
52 }
53
54 void relu_Y(double *Y, int N) {
55     for(int n=0;n<N;n++) Y[n] = (Y[n]>0?1:0)*Y[n];
56 }
57
58 void relu_XE(double *XE, double *X, int M) {
59     for(int m=0;m<M;m++) XE[m] = XE[m]*(X[m]>0?1:0)*1;
60 }
61
62 void softmax_Y(double *Y, int N) {
63     double YMax = Y[0];
64     for(int n=1;n<N;n++) if(Y[n]>YMax) YMax = Y[n];
65     for(int n=0;n<N;n++) Y[n] -= YMax;
66     double sumY = 0;
67     for(int n=0;n<N;n++) sumY += exp(Y[n]);
68     for(int n=0;n<N;n++) Y[n] = exp(Y[n])/sumY;
69 }
70
71 double calculate_CEE(double *Y, double *YT, int N) {
72     double E = 0.0;
73     for(int n=0;n<N;n++) E += -YT[n]*log(Y[n]);
74     return E;
75 }
76
77 void print(char * s, double *Y, int N) {
78     printf("%s [", s);
79     for(int n=0;n<N-1;n++) printf("%.3f ", Y[n]);
80     printf("%.3f]\n", Y[N-1]);
81 }
82
83 void print(char * s, double *W, int M, int N) {
84     printf("%s [\n", s);
85     for(int m=0;m<M;m++) {
86         printf("[", s);
87         for(int n=0;n<N-1;n++)

```

```

88             printf("%.3f ", W[m*N+n]);
89             printf("%.3f\n", W[m*N+N-1]);
90         }
91         printf("\n");
92     }

```

62~69 : softmax_Y 함수를 정의합니다.

63~64 : Y 배열에서 가장 큰 항목의 값을 구합니다.

65 : Y의 각 항목에서 Y의 가장 큰 항목 값을 빼줍니다. 이렇게 하면 67 줄에서 오버플로우를 막을 수 있습니다. Y에 대한 최종 결과는 같습니다. 자세한 내용은 [소프트맥스 오버플로우]를 검색해 봅니다.

71~75 : calculate_CEE 함수를 정의합니다.

73 : 다음 수식을 구현합니다. 소프트맥스 활성화 함수는 크로스 엔트로피 오차와 같이 사용됩니다.

$$E = - \sum_k (y_{kT} * \log(y_k))$$

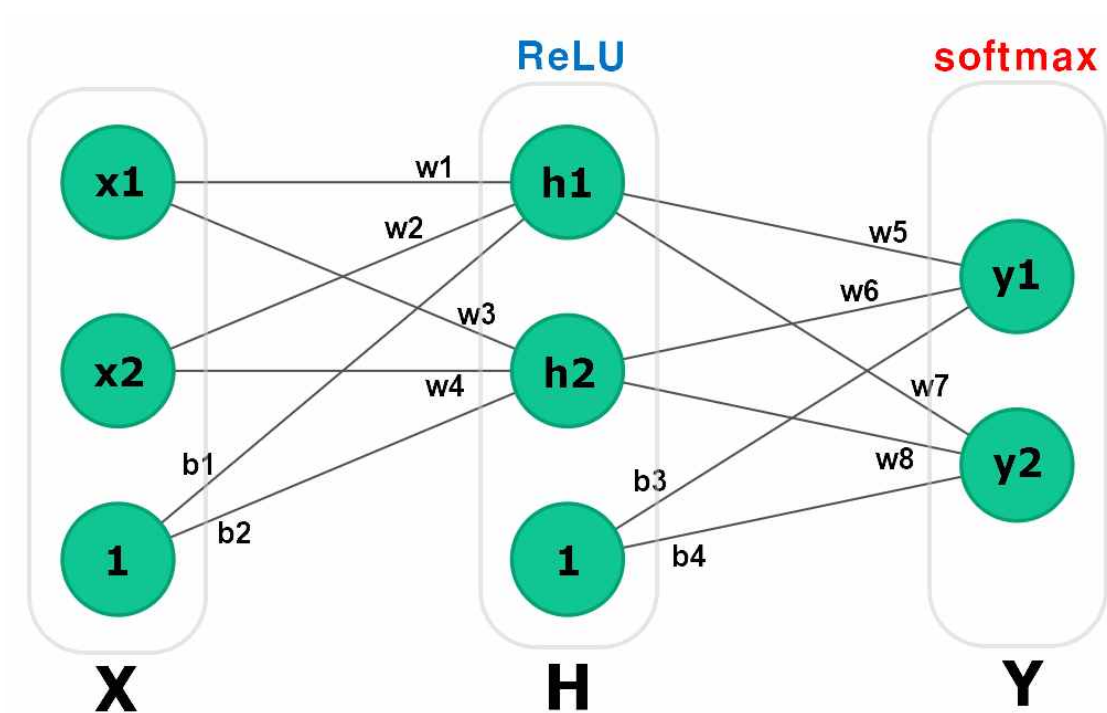
소프트맥스 함수의 역전파 오차 계산 부분은 다음과 같습니다. 소프트맥스 함수는 크로스 엔트로피 함수와 같이 사용될 때 역전파 시 소프트맥스 함수를 역으로 거쳐 전파되는 오차가 다음과 같이 예측 값과 목표 값의 차가 됩니다.

$$y_{kE} = y_k - y_{kT}$$

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

ReLU와 softmax

여기서는 은닉층 활성화 함수를 ReLU로 출력층 활성화 함수를 softmax로 적용해 봅니다. 다음 그림을 살펴봅니다.



1. 이전 예제 5110_2.ino를 5110_3.ino로 저장합니다.

2. 다음과 같이 예제를 수정합니다.

5110_3.ino

```

01 const int NUM_X = 2;
02 const int NUM_H = 2;
03 const int NUM_Y = 2;
04
05 double X[NUM_X] = {0.05, 0.10};
06 double YT[NUM_Y] = {0.00, 1.00};
07 double H[NUM_H];
08 double Y[NUM_Y];
09
10 double WH[NUM_X][NUM_H] = {
11     {0.15, 0.25},
12     {0.20, 0.30}
13 };
14 double BH[NUM_H] = {0.35, 0.35};
15 double WY[NUM_H][NUM_Y] = {
16     {0.40, 0.50},
17     {0.45, 0.55}
18 };

```

```

19 double BY[NUM_Y] = {0.60, 0.60};
20
21 double YE[NUM_Y];
22 double HE[NUM_H];
23 double WYE[NUM_H][NUM_Y];
24 double BYE[NUM_Y];
25 double WHE[NUM_X][NUM_H];
26 double BHE[NUM_H];
27
28 void dnn_test() {
29
30     for(int epoch=0;epoch<10000000;epoch++) {
31
32         if(epoch%10000==9999) printf("\nepoch = %d\n", epoch);
33
34         forward_Y(X, (double *)WH, BH, H, NUM_X, NUM_H);
35         relu_Y(H, NUM_H);
36
37         forward_Y(H, (double *)WY, BY, Y, NUM_H, NUM_Y);
38         softmax_Y(Y, NUM_Y);
39         if(epoch%10000==9999) print("Y =", Y, NUM_Y);
40
41         double E = calculate_CEE(Y, YT, NUM_Y);
42
43         if(E < 0.0001) {
44             printf("\nepoch = %d\n", epoch);
45             print("Y =", Y, NUM_Y);
46             break;
47         }
48
49         backward_YE(YE, Y, YT, NUM_Y);
50         // nothing for softmax
51
52         backward_WE((double *)WYE, H, YE, NUM_H, NUM_Y);
53         backward_BE(BYE, YE, NUM_Y);
54
55         backward_HE(HE, (double *)WY, YE, NUM_H, NUM_Y);
56         relu_XE(HE, H, NUM_H);
57
58         backward_WE((double *)WHE, X, HE, NUM_X, NUM_H);

```

```

59         backward_BE(BHE, HE, NUM_H);
60
61         double lr = 0.01;
62
63         learning_W((double *)WY, lr, (double *)WYE, NUM_H, NUM_Y);
64         learning_B(BY, lr, BYE, NUM_Y);
65
66         learning_W((double *)WH, lr, (double *)WHE, NUM_X, NUM_H);
67         learning_B(BH, lr, BHE, NUM_H);
68
69     }
70
71 }
72
73 void setup() {
74
75     Serial.begin(115200);
76     delay(1000);
77
78     dnn_test();
79
80 }
81
82 void loop() {
83
84 }

```

06 : 목표값을 각각 0.00과 1.00로 변경합니다.

38 : 출력층의 활성화 함수를 소프트맥스로 변경합니다.

41 : 오차 계산 함수를 크로스 엔트로피 오차 함수로 변경합니다.

43 : for 문을 빠져 나가는 오차 값을 0.0001로 변경합니다. 여기서 사용하는 값의 크기에 따라 학습의 정확도와 학습 시간이 결정됩니다.

50 : 소프트맥스 함수는 크로스 엔트로피 함수와 같이 사용될 때 역전파 시 소프트맥스 함수를 역으로 거쳐 전파되는 오차가 다음과 같이 예측 값과 목표 값의 차가 됩니다.

$$y_{kE} = y_k - y_{kT}$$

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

3. 업로드를 수행합니다.



4. 출력 결과를 확인합니다.

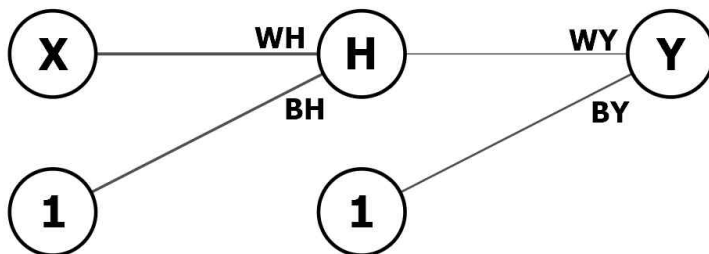
```
epoch = 49999  
Y = [0.000 1.000]  
  
epoch = 56961  
Y = [0.000 1.000]
```

(56961+1)번째에 오차가 0.0001(만분의 1)보다 작아집니다. y1, y2는 각각 0.000, 1.000이 된 상태입니다.

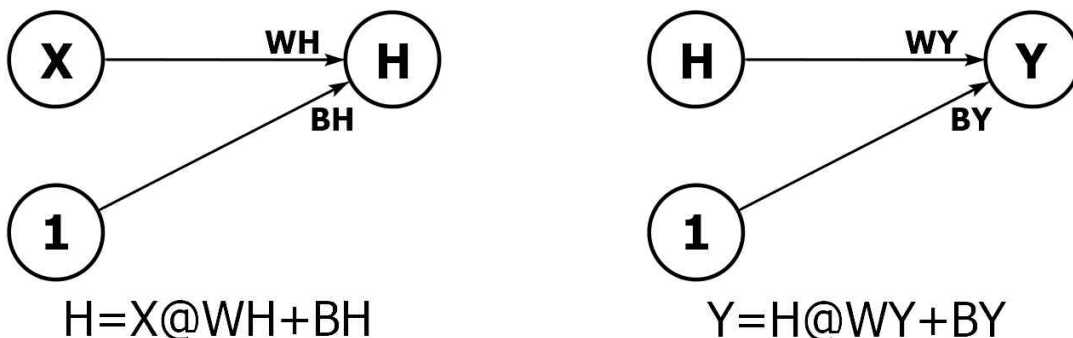
이상에서 출력층의 활성화 함수는 소프트맥스, 오차 계산 함수는 크로스 엔트로피 오차 함수 인 인공 신경망을 구현해 보았습니다.

05 인공 신경망 행렬 계산식

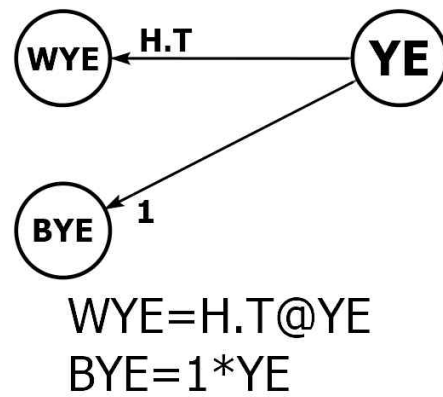
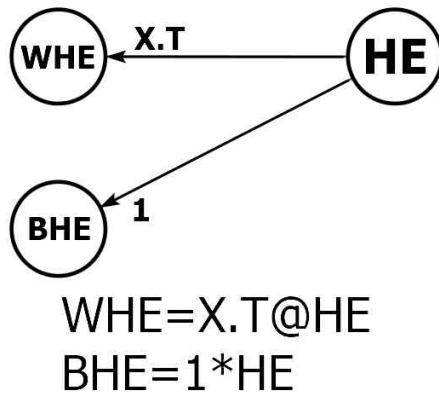
여기서는 인공 신경망의 순전파 역전파를 행렬 계산식으로 정리해 봅니다. 인공 신경망을 행렬 계산식으로 정리하면 인공 신경망의 크기, 깊이와 상관없이 간결하게 정리할 수 있습니다. 다음 그림은 입력층, 은닉층, 출력층으로 구성된 인공 신경을 나타냅니다.



이 인공 신경망은 다음과 같이 인공 신경망 2개로 구성됩니다.



다음 그림은 이 인공 신경망의 역전파를 나타냅니다.



은닉층을 포함한 인공 신경망의 경우 은닉층 노드의 역전파 과정이 필요합니다. 즉, 위 그림에서 HE에 대한 값이 필요합니다. 다음 그림은 HE를 구하는 과정을 나타냅니다.

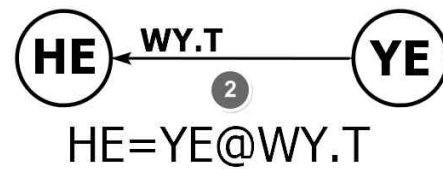


그림 ①은 H의 순전파 과정을 나타냅니다. 그림 ①에서 H는 WY를 통해 Y로 순전파됩니다. 그림 ②는 YE의 HE로의 역전파 과정을 나타냅니다. 그림 ②에서 YE는 WY.T를 통해 HE로 역전파됩니다. H가 WY를 통해 Y로 순전파되는 것처럼 YE는 WY.T를 통해 HE로 역전파됩니다.

*** @ 문자는 행렬곱을 의미합니다.

이상에서 필요한 행렬 계산식을 정리하면 다음과 같습니다.

순전파

$$H = X@WH + BH$$

$$Y = H@WY + BY$$

은닉층 역전파

$$HE = YE@WY.T$$

역전파

$$WHE = X.T@HE$$

$$BHE = 1*HE$$

$$WYE = H.T@YE$$

$$BYE = 1*YE$$

인공 신경망 학습

$$W_H = W_H - lr * W_{HE}$$

$$B_H = B_H - lr * B_{HE}$$

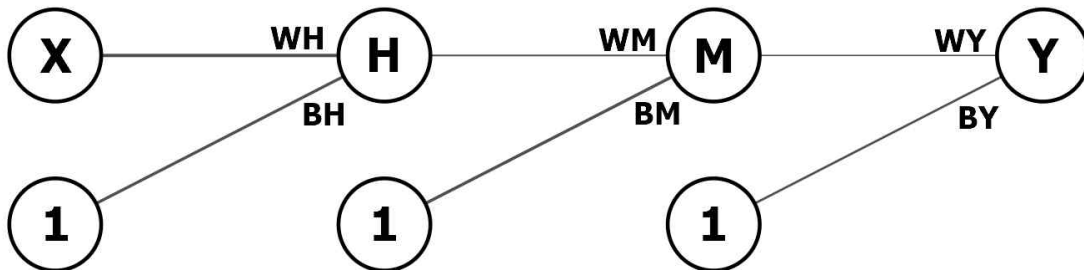
$$W_Y = W_Y - lr * W_{YE}$$

$$B_Y = B_Y - lr * B_{YE}$$

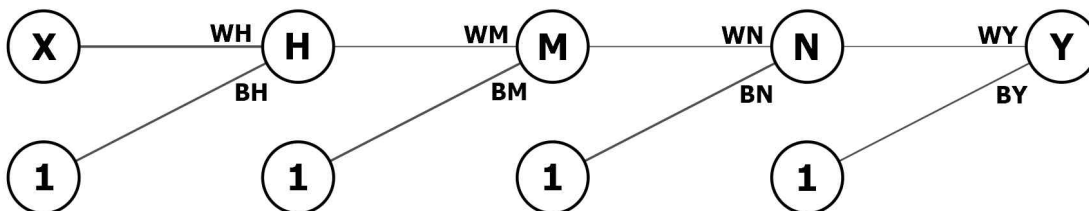
*** lr은 학습률을 나타냅니다.

연습문제

1. 다음은 입력X 은닉H 은닉M 출력Y의 심층 인공 신경망입니다. 이 신경망에는 2개의 은닉층이 포함되어 있습니다. 일반적으로 은닉층이 2층 이상일 경우 심층 인공 신경망이라고 합니다. 이 신경망의 가중치, 편향 역전파 그래프와 은닉층 역전파 그래프를 그리고 순전파, 역전파 행렬 계산식을 구합니다.



2. 다음은 입력X 은닉H 은닉M 은닉N 출력Y의 심층 인공 신경망입니다. 이 신경망에는 3개의 은닉층이 포함되어 있습니다. 일반적으로 은닉층이 2층 이상일 경우 심층 인공 신경망이라고 합니다. 이 신경망의 가중치, 편향 역전파 그래프와 은닉층 역전파 그래프를 그리고 순전파, 역전파 행렬 계산식을 구합니다.

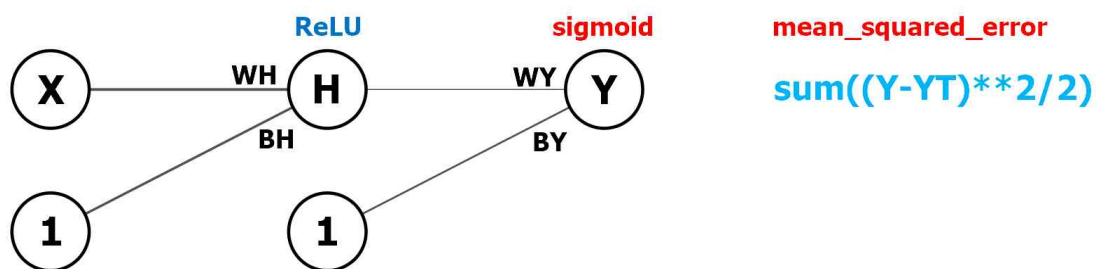


06 가중치 초기화하기

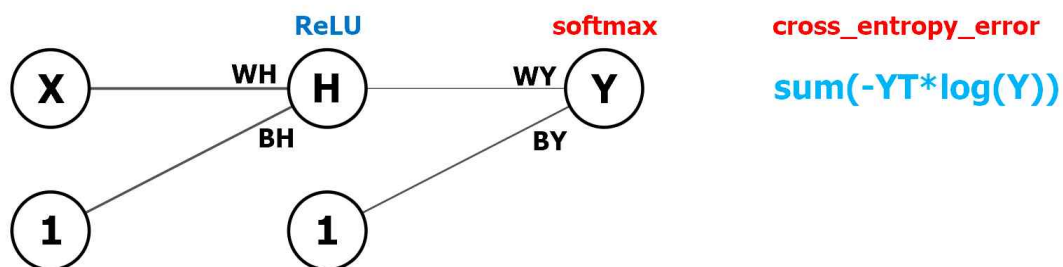
여기서는 활성화 함수에 따라 은닉층과 출력층의 가중치를 초기화하는 방법에 대해 살펴보고 해당 방법을 적용하여 은닉층과 출력층의 가중치를 초기화한 후 학습을 시켜봅니다. 미리 말씀드리면 활성화 함수에 따른 가중치와 편향의 적절한 초기화는 인공 신경망 학습에 아주 중요한 부분입니다. 우리는 앞으로 수행할 예제에서 은닉층의 활성화 함수로 ReLU를 사용하고

출력층의 활성화 함수로 sigmoid나 softmax를 사용합니다. 출력층의 활성화 함수를 sigmoid로 사용할 경우 오차 계산 함수는 평균 제곱 오차 함수를 사용하고, 출력층의 활성화 함수를 softmax로 사용할 경우 오차 계산 함수는 크로스 엔트로피 오차 함수를 사용하도록 합니다. 다음 그림을 참조합니다.

ReLU-sigmoid-mse 신경망



ReLU-softmax-cee 신경망



ReLU와 He 초기화

ReLU 활성화 함수를 사용할 경우엔 Kaming He가 2010년에 발표한 He 초기화 방법을 사용합니다. 수식은 다음과 같습니다.

$$\text{normal}(\text{mean} = 0, \text{stddev}), \text{stddev} = \sqrt{\frac{2}{\text{input}}}$$

여기서 normal은 종모양의 정규 분포를 의미하며, mean은 평균값, stddev는 표준편차로 종모양이 퍼진 정도를 의미합니다. 이 수식을 적용하면 0에 가까운 값이 많도록 가중치가 초기화됩니다.

sigmoid, softmax와 Lecun 초기화

sigmoid와 softmax 활성화 함수를 사용할 경우엔 Yann Lecun 교수가 1998년에 발표한 Lecun 초기화 방법을 사용합니다. 수식은 다음과 같습니다.

$$normal(mean = 0, stddev), stddev = \sqrt{\frac{1}{input}}$$

여기서 normal은 종모양의 표준 정규 분포를 의미하며, mean은 평균값, stddev는 표준편차로 종모양이 퍼진 정도를 의미합니다. 이 수식을 적용하면 0에 가까운 값이 많도록 가중치가 초기화됩니다.

정규 분포 난수 생성해 보기

여기서는 정규 분포 곡선에 따른 난수 생성 함수를 정의하고 사용해 봅니다.

1. 다음과 같이 예제를 작성합니다.

5112.ino

```
01 double randn(void) {
02
03     double u, v, r, c;
04
05     do {
06
07         u = ((double) rand() / RAND_MAX) * 2 - 1; // -1.0 ~ 1.0
08         v = ((double) rand() / RAND_MAX) * 2 - 1; // -1.0 ~ 1.0
09
10         r = u * u + v * v;
11
12     } while (r == 0 || r >= 1);
13
14     c = sqrt( (-2 * log(r)) / r );
15
16     return u * c;
17
18 }
19
20 int dnn_test(void) {
21
22     srand(time(NULL));
23 }
```

```

24     for (int i = 1; i <= 5000; i++)
25         printf("%.17f\n", randn());
26
27     return 0;
28
29 }
30
31 void setup() {
32
33     Serial.begin(115200);
34     delay(1000);
35
36     dnn_test();
37
38 }
39
40 void loop() {
41
42 }

```

01~18 : randn 함수를 정의합니다. randn 함수는 표준 정규 분포에 따른 난수를 생성하는 함수입니다. randn 함수에 대한 설명은 따로 하지 않습니다.

22 : srand 함수를 호출하여 난수 생성기를 현재 시간에 따라 임의로 초기화합니다.

24~25 : 5000개의 난수를 생성해 봅니다.

2. 업로드를 수행합니다.



3. 출력 결과를 확인합니다.

```

0.14612506860492452
0.28945455082253657
-0.27651776828412078
-1.32553707893525674
-0.26625910824749605

```

난수가 생성되는 것을 확인합니다.

mydnn 라이브러리에 가중치 초기화 함수 추가하기

다음과 같이 mydnn_4.ino 파일에 가중치 초기화 함수를 추가합니다.

mydnn_5.ino

```
001 void forward_Y(double *X, double *W, double *B, double *Y, int M, int N) {
002     for(int n=0;n<N;n++) Y[n] = 0;
003     for(int m=0;m<M;m++)
004         for(int n=0;n<N;n++)
005             Y[n] += X[m]*W[m*N+n];
006     for(int n=0;n<N;n++) Y[n] += B[n];
007 }
008
009 double calculate_MSE(double *Y, double *YT, int N) {
010     double E = 0;
011     for(int n=0;n<N;n++) E += (Y[n]-YT[n])*(Y[n]-YT[n])/2;
012     return E;
013 }
014
015 void backward_YE(double *YE, double *Y, double *YT, int N) {
016     for(int n=0;n<N;n++) YE[n] = Y[n] - YT[n];
017 }
018
019 void backward_WE(double *WE, double *X, double *YE, int M, int N) {
020     for(int m=0;m<M;m++)
021         for(int n=0;n<N;n++)
022             WE[m*N+n] = X[m]*YE[n];
023 }
024
025 void backward_BE(double *BE, double *YE, int N) {
026     for(int n=0;n<N;n++) BE[n] = YE[n];
027 }
028
029 void backward_HE(double *HE, double *W, double *YE, int M, int N) {
030     for(int m=0;m<M;m++) HE[m] = 0;
031     for(int m=0;m<M;m++)
032         for(int n=0;n<N;n++)
033             HE[m] += YE[n]*W[m*N+n];
034 }
035
036 void learning_W(double *W, double lr, double *WE, int M, int N) {
037     for(int m=0;m<M;m++)
038         for(int n=0;n<N;n++)
```

```

039         W[m*N+n] -= lr * WE[m*N+n];
040     }
041
042     void learning_B(double *B, double lr, double *BE, int N) {
043         for(int n=0;n<N;n++) B[n] -= lr * BE[n];
044     }
045
046     void sigmoid_Y(double *Y, int N) {
047         for(int n=0;n<N;n++) Y[n] = 1/(1 + exp(-Y[n]));
048     }
049
050     void sigmoid_XE(double *XE, double *X, int M) {
051         for(int m=0;m<M;m++) XE[m] = XE[m]*X[m]*(1-X[m]);
052     }
053
054     void relu_Y(double *Y, int N) {
055         for(int n=0;n<N;n++) Y[n] = (Y[n]>0?1:0)*Y[n];
056     }
057
058     void relu_XE(double *XE, double *X, int M) {
059         for(int m=0;m<M;m++) XE[m] = XE[m]*(X[m]>0?1:0)*1;
060     }
061
062     void softmax_Y(double *Y, int N) {
063         double YMax = Y[0];
064         for(int n=1;n<N;n++) if(Y[n]>YMax) YMax = Y[n];
065         for(int n=0;n<N;n++) Y[n] -= YMax;
066         double sumY = 0;
067         for(int n=0;n<N;n++) sumY += exp(Y[n]);
068         for(int n=0;n<N;n++) Y[n] = exp(Y[n])/sumY;
069     }
070
071     double calculate_CEE(double *Y, double *YT, int N) {
072         double E = 0.0;
073         for(int n=0;n<N;n++) E += -YT[n]*log(Y[n]);
074         return E;
075     }
076
077     double randn() {
078

```



```

079 double u, v, r, c;
080
081 do {
082     u = ((double) rand() / RAND_MAX) * 2 - 1; // -1.0 ~ 1.0 까지의 값
083     v = ((double) rand() / RAND_MAX) * 2 - 1; // -1.0 ~ 1.0 까지의 값
084
085     r = u * u + v * v;
086 } while (r == 0 || r >= 1);
087
088 c = sqrt( (-2 * log(r)) / r );
089
090 return u * c;
091
092 }
093
094 void initialize_weight_He(double *W, int M, int N) {
095     for(int m=0;m<M;m++)
096         for(int n=0;n<N;n++)
097             W[m*N+n] = randn()/sqrt(M/2.0); // He
098 }
099
100 void initialize_weight_Le(double *W, int M, int N) {
101     for(int m=0;m<M;m++)
102         for(int n=0;n<N;n++)
103             W[m*N+n] = randn()/sqrt(M); // Lecun
104 }
105
106 void print(char * s, double *Y, int N) {
107     printf("%s [", s);
108     for(int n=0;n<N-1;n++) printf("%.3f ", Y[n]);
109     printf("%.3f]\n", Y[N-1]);
110 }
111
112 void print(char * s, double *W, int M, int N) {
113     printf("%s [\n", s);
114     for(int m=0;m<M;m++) {
115         printf("[", s);
116         for(int n=0;n<N-1;n++)
117             printf("%.3f ", W[m*N+n]);
118         printf("%.3f]\n", W[m*N+N-1]);

```

```
119     }
120     printf("\n");
121 }
```

077~092 : randn 함수를 정의합니다.

094~098 : initialize_weight_He 함수를 정의합니다.

095~097 : W 배열의 각 항목에 대해 He 초기화를 수행합니다.

100~104 : initialize_weight_Le 함수를 정의합니다.

101~103 : W 배열의 각 항목에 대해 Lecun 초기화를 수행합니다.

He와 Lecun 가중치 초기화하기

이제 He와 Lecun으로 가중치를 초기화하여 학습시켜 봅니다.

1. 다음과 같이 예제를 작성합니다.

5112_2.ino

```
01 const int NUM_X = 2;
02 const int NUM_H = 2;
03 const int NUM_Y = 2;
04
05 double X[NUM_X] = {0.05, 0.10};
06 double YT[NUM_Y] = {0.01, 0.99}; //
07 double H[NUM_H];
08 double Y[NUM_Y];
09
10 double WH[NUM_X][NUM_H];
11 double BH[NUM_H];
12 double WY[NUM_H][NUM_Y];
13 double BY[NUM_Y];
14
15 double YE[NUM_Y];
16 double HE[NUM_H];
17 double WYE[NUM_H][NUM_Y];
18 double BYE[NUM_Y];
19 double WHE[NUM_X][NUM_H];
20 double BHE[NUM_H];
21
22 void dnn_test() {
23
24     srand(time(NULL));
25 }
```

```

26 initialize_weight_He((double *)WH, NUM_X, NUM_H); //
27 print("WH =", (double *)WH, NUM_X, NUM_H);
28
29 initialize_weight_Le((double *)WY, NUM_H, NUM_Y); //
30 print("WY =", (double *)WY, NUM_H, NUM_Y);
31
32 for(int epoch=1;epoch<1000001;epoch++) {
33
34     forward_Y(X, (double *)WH, BH, H, NUM_X, NUM_H);
35     relu_Y(H, NUM_H);
36
37     forward_Y(H, (double *)WY, BY, Y, NUM_H, NUM_Y);
38     sigmoid_Y(Y, NUM_Y);
39
40     double E = calculate_MSE(Y, YT, NUM_Y);
41
42     if(epoch==1) { //
43         printf("\nepoch = %d\n", epoch);
44         printf("Error = %.4f\n", E);
45         print("Y =", Y, NUM_Y);
46     }
47
48     if(E < 0.0001) { //
49         printf("\nepoch = %d\n", epoch);
50         printf("Error = %.4f\n", E);
51         print("Y =", Y, NUM_Y);
52         break;
53     }
54
55     backward_YE(YE, Y, YT, NUM_Y);
56     sigmoid_XE(YE, Y, NUM_Y); //
57
58     backward_WE((double *)WYE, H, YE, NUM_H, NUM_Y);
59     backward_BE(BYE, YE, NUM_Y);
60
61     backward_HE(HE, (double *)WY, YE, NUM_H, NUM_Y);
62     relu_XE(HE, H, NUM_H);
63
64     backward_WE((double *)WHE, X, HE, NUM_X, NUM_H);
65     backward_BE(BHE, HE, NUM_H);

```

```

66
67         double lr = 0.01;
68
69         learning_W((double *)WY, lr, (double *)WYE, NUM_H, NUM_Y);
70
71         learning_B(BY, lr, BYE, NUM_Y);
72
73         learning_W((double *)WH, lr, (double *)WHE, NUM_X, NUM_H);
74
75         learning_B(BH, lr, BHE, NUM_H);
76
77     }
78
79 void setup() {
80
81     Serial.begin(115200);
82     delay(1000);
83
84     dnn_test();
85
86 }
87
88 void loop() {
89
90 }

```

06 : 목표값을 각각 0.01과 0.99로 변경합니다.

26~27 : WH 배열에 대해 He 초기화를 수행하고, 결과를 확인합니다.

29~30 : WY 배열에 대해 Lecun 초기화를 수행하고, 결과를 확인합니다.

32 : epoch 변수 1에서 1000001(백만일) 미만에 대하여 34~73줄을 수행합니다.

35 : 은닉 층의 활성화 함수를 ReLU로 사용합니다.

38 : 출력 층의 활성화 함수를 sigmoid로 사용합니다.

40 : 오차 계산 함수는 평균 제곱 오차를 사용합니다.

42~46 : epoch값이 1일 때, 즉, 처음 시작할 때, 오차 값과 예측 값을 출력합니다.

48~53 : 오차 값이 0.0001(만분의 일)보다 작을 때, 오차 값과 예측 값을 출력한 후, 32줄의 for문을 나옵니다.

56 : 출력 층의 역 활성화 함수를 sigmoid로 사용합니다.

59 : 은닉 층의 역 활성화 함수를 ReLU로 사용합니다.

2. 업로드를 수행합니다.



3. 출력 결과를 확인합니다.

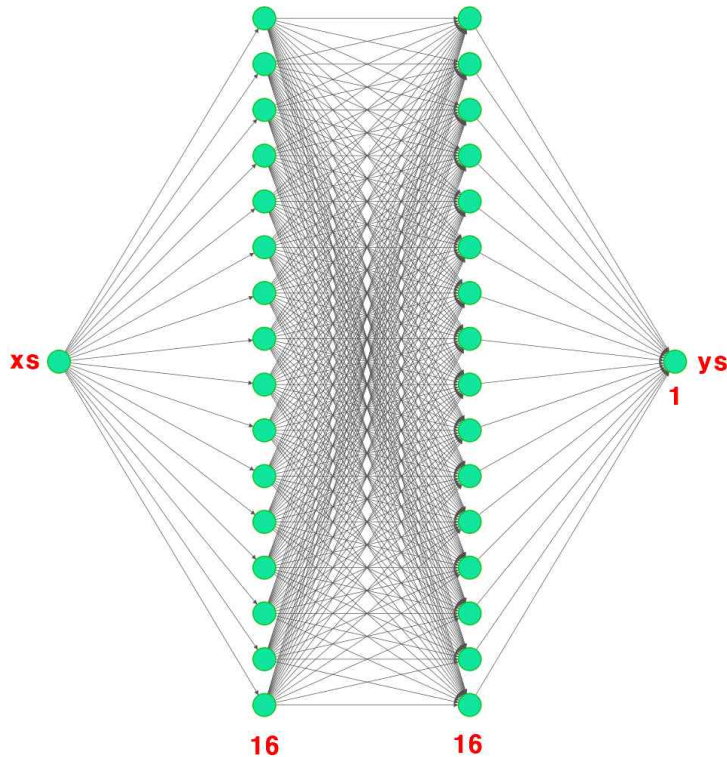
```
WH = [  
  [1.966 2.133]  
  [-0.702 -1.232]  
]  
WY = [  
  [-0.314 -0.493]  
  [0.531 0.582]  
]  
  
epoch = 1  
Error = 0.2407  
Y = [0.498 0.497]  
  
epoch = 207888  
Error = 0.0001  
Y = [0.020 0.980]
```

필자의 경우 207888번 학습을 수행하였으며, 오차는 0.0001이고, 첫 번째 항목의 값은 0.02, 두 번째 항목은 0.98입니다. 가중치 초기값에 따라 독자 여러분의 결과는 다를 수 있습니다.

03 딥러닝 라이브러리 활용하기

여기서는 지금까지 구현한 인공 신경망 라이브러리를 활용해 인공 신경망을 확장해 봅니다.

인공 신경망 라이브러리를 이용하면, 인공 신경망을 좀 더 자유롭게 구성하고 테스트해 볼 수 있습니다. 예를 들어, 다음과 같은 형태의 인공 신경망을 구성해서 테스트해 볼 수 있습니다.



01 7 세그먼트 입력 2 진수 출력 인공 신경망

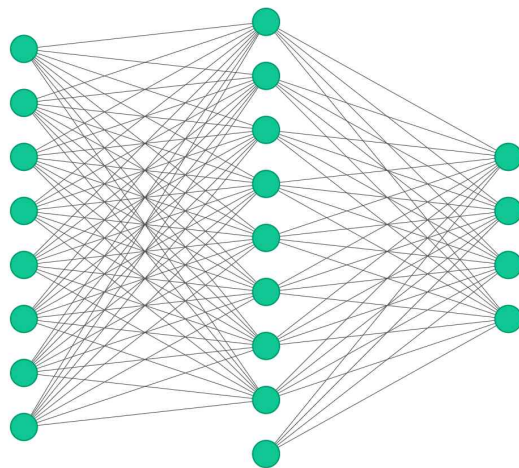
여기서는 7 세그먼트에 숫자 값에 따라 표시되는 LED의 ON, OFF 값을 입력으로 받아 2 진수로 출력하는 인공 신경망을 구성하고 학습시켜 봅니다. 다음은 7 세그먼트 디스플레이 2 진수 연결 진리표입니다.

7 세그먼트 디스플레이
2 진수 연결 진리표

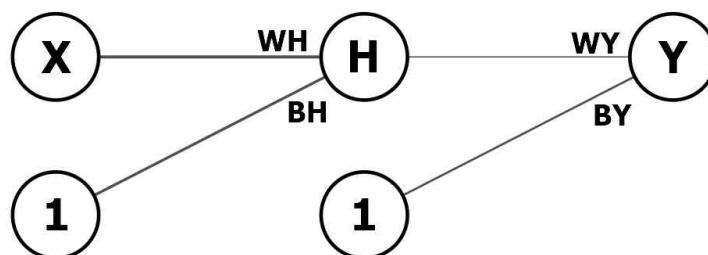
In	In	In	In	In	In	In	Out	Out	Out	Out
1	1	1	1	1	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1
1	1	0	1	1	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	1	1
0	1	1	0	0	1	1	0	1	0	0
1	0	1	1	0	1	1	0	1	0	1
0	0	1	1	1	1	1	0	1	1	0
1	1	1	0	0	0	0	0	1	1	1
1	1	1	1	1	1	1	1	0	0	0
1	1	1	0	0	1	1	1	0	0	1

5 = 1011011 → 0101

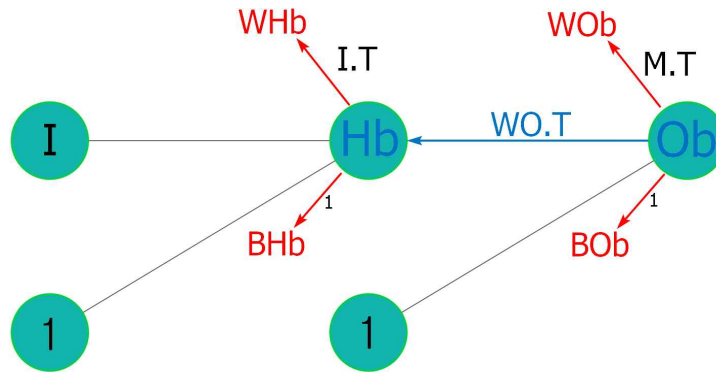
그림에서 7 세그먼트에 5로 표시되기 위해 7개의 LED가 1011011(1-ON, 0-OFF)의 비트열에 맞춰 켜지거나 꺼져야 합니다. 해당 비트열에 대응하는 이진수는 0101입니다. 여기서는 다음 그림과 같이 7개의 입력, 8개의 은닉층, 4개의 출력층으로 구성된 인공 신경망을 학습시켜 봅니다.



다음은 행렬을 이용하여 나타낸 X-H-Y 인공 신경망입니다. 이 신경망은 순전파 과정을 나타냅니다.



다음은 역전파 과정을 나타냅니다.



0. 먼저 521_a.ino 예제를 521.ino로 저장합니다.

1. 다음과 같이 예제를 수정합니다.

521.ino

```

001 const int NUM_PATTERN = 10;
002 const int NUM_X = 7;
003 const int NUM_H = 8;
004 const int NUM_Y = 4;
005
006 double X[NUM_PATTERN][NUM_X] = {
007     { 1, 1, 1, 1, 1, 1, 0 }, // 0
008     { 0, 1, 1, 0, 0, 0, 0 }, // 1
009     { 1, 1, 0, 1, 1, 0, 1 }, // 2
010     { 1, 1, 1, 1, 0, 0, 1 }, // 3
011     { 0, 1, 1, 0, 0, 1, 1 }, // 4
012     { 1, 0, 1, 1, 0, 1, 1 }, // 5
013     { 0, 0, 1, 1, 1, 1, 1 }, // 6
014     { 1, 1, 1, 0, 0, 0, 0 }, // 7
015     { 1, 1, 1, 1, 1, 1, 1 }, // 8
016     { 1, 1, 1, 0, 0, 1, 1 } // 9
017 };
018 double YT[NUM_PATTERN][NUM_Y] = {
019     { 0, 0, 0, 0 },
020     { 0, 0, 0, 1 },
021     { 0, 0, 1, 0 },
022     { 0, 0, 1, 1 },
023     { 0, 1, 0, 0 },
024     { 0, 1, 0, 1 },
025     { 0, 1, 1, 0 },

```



```

026     { 0, 1, 1, 1 },
027     { 1, 0, 0, 0 },
028     { 1, 0, 0, 1 }
029 };
030 double H[NUM_H];
031 double Y[NUM_PATTERN][NUM_Y];
032
033 double WH[NUM_X][NUM_H];
034 double BH[NUM_H];
035 double WY[NUM_H][NUM_Y];
036 double BY[NUM_Y];
037
038 double YE[NUM_Y];
039 double HE[NUM_H];
040 double WYE[NUM_H][NUM_Y];
041 double BYE[NUM_Y];
042 double WHE[NUM_X][NUM_H];
043 double BHE[NUM_H];
044
045 void dnn_test() {
046
047     srand(time(NULL));
048
049     initialize_weight_He((double *)WH, NUM_X, NUM_H); //
050
051     initialize_weight_Le((double *)WY, NUM_H, NUM_Y); //
052
053     for(int epoch=1;epoch<1000001;epoch++) {
054
055         forward_Y(X[2], (double *)WH, BH, H, NUM_X, NUM_H);
056         relu_Y(H, NUM_H);
057
058         forward_Y(H, (double *)WY, BY, Y[2], NUM_H, NUM_Y);
059         sigmoid_Y(Y[2], NUM_Y);
060
061         double E = calculate_MSE(Y[2], YT[2], NUM_Y);
062
063         if(epoch==1) { //
064             printf("\nepoch = %d\n", epoch);
065             printf("Error = %.4f\n", E);

```

```

066         print("Y =", Y[2], NUM_Y);
067     }
068
069     if(E < 0.0001) { //
070         printf("\nepoch = %d\n", epoch);
071         printf("Error = %.4f\n", E);
072         print("Y =", Y[2], NUM_Y);
073         break;
074     }
075
076     backward_YE(YE, Y[2], YT[2], NUM_Y);
077     sigmoid_XE(YE, Y[2], NUM_Y); //
078
079     backward_WE((double *)WYE, H, YE, NUM_H, NUM_Y);
080     backward_BE(BYE, YE, NUM_Y);
081
082     backward_HE(HE, (double *)WY, YE, NUM_H, NUM_Y);
083     relu_XE(HE, H, NUM_H);
084
085     backward_WE((double *)WHE, X[2], HE, NUM_X, NUM_H);
086     backward_BE(BHE, HE, NUM_H);
087
088     double lr = 0.01;
089
090     learning_W((double *)WY, lr, (double *)WYE, NUM_H, NUM_Y);
091     learning_B(BY, lr, BYE, NUM_Y);
092
093     learning_W((double *)WH, lr, (double *)WHE, NUM_X, NUM_H);
094     learning_B(BH, lr, BHE, NUM_H);
095
096 }
097
098 }
099
100 void setup() {
101
102     Serial.begin(115200);
103     delay(1000);
104
105     dnn_test();

```

```


106
107 }
108
109 void loop() {
110
111 }

```

001 : NUM_PATTERN 변수를 선언한 후, 10으로 초기화합니다. NUM_PATTERN 변수는 다음 진리표의 가로줄의 개수입니다.

7 세그먼트 디스플레이
2 진수 연결 진리표

In	In	In	In	In	In	In	Out	Out	Out	Out
1	1	1	1	1	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1
1	1	0	1	1	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	1	1
0	1	1	0	0	1	1	0	1	0	0
1	0	1	1	0	1	1	0	1	0	1
0	0	1	1	1	1	1	0	1	1	0
1	1	1	0	0	0	0	0	1	1	1
1	1	1	1	1	1	1	1	0	0	0
1	1	1	0	0	1	1	1	0	0	1

 = 1011011 → 0101

- 002 : NUM_X 변수를 선언한 후, 7로 초기화합니다.
- 003 : NUM_H 변수를 선언한 후, 8로 초기화합니다.
- 004 : NUM_Y 변수를 선언한 후, 4로 초기화합니다.
- 006~017 : 2차 배열 X를 선언한 후, 진리표의 입력 값에 맞게 2차 배열 값을 초기화합니다.
- 018~029 : 2차 배열 YT를 선언한 후, 진리표의 입력 값에 맞게 2차 배열 값을 초기화합니다.
- 031 : 2차 배열 Y를 선언합니다.
- 55, 85 : X을 X[2]로 변경합니다. X 배열의 2번 항목을 입력 값으로 학습 테스트를 수행합니다.
- 61, 76 : YT을 YT[2]로 변경합니다. YT 배열의 2번 항목을 목표 값으로 학습 테스트를 수행합니다.
- 58, 59, 61, 66, 72, 76, 77 : Y를 Y[2]로 변경합니다. Y 배열의 2번 항목을 예측 값으로 학습 테스트를 수행합니다.

2. 업로드를 수행합니다.



3. 출력결과를 확인합니다.

```

epoch = 1
Error = 0.2579
Y = [0.452 0.287 0.570 0.210]

epoch = 24125
Error = 0.0001
Y = [0.007 0.007 0.993 0.007]

```

필자의 경우 24125번 학습을 수행하였으며, 오차는 0.0001이고, 첫 번째, 두 번째, 네 번째 항목의 값은 0에 가깝고, 세 번째 항목의 값은 1에 가깝습니다. 다음 그림에서 진리표의 2번 항목에 맞게 학습된 것을 볼 수 있습니다.

7 세그먼트 디스플레이
2 진수 연결 진리표

In	In	In	In	In	In	In	Out	Out	Out	Out
1	1	1	1	1	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1
1	1	0	1	1	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	1	1
0	1	1	0	0	1	1	0	1	0	0
1	0	1	1	0	1	1	0	1	0	1
0	0	1	1	1	1	1	0	1	1	0
1	1	1	0	0	0	0	0	1	1	1
1	1	1	1	1	1	1	1	0	0	0
1	1	1	0	0	1	1	1	0	0	1


5 = 1011011 → 0101

02 7 세그먼트 입력 2 진수 출력 인공 신경망 2

계속해서 7 세그먼트에 숫자 값에 따라 표시되는 LED의 ON, OFF 값을 입력으로 받아 2 진수로 출력하는 인공 신경망을 구성하고 학습시켜 봅니다. 여기서는 다음 진리표의 전체 입력 값에 대해 목표 값에 대응되도록 학습을 시켜봅니다.

7 세그먼트 디스플레이
2 진수 연결 진리표

In	In	In	In	In	In	In	Out	Out	Out	Out
1	1	1	1	1	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1
1	1	0	1	1	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	1	1
0	1	1	0	0	1	1	0	1	0	0
1	0	1	1	0	1	1	0	1	0	1
0	0	1	1	1	1	1	0	1	1	0
1	1	1	0	0	0	0	0	1	1	1
1	1	1	1	1	1	1	1	0	0	0
1	1	1	0	0	1	1	1	0	0	1

 = 1011011 → 0101

0. 먼저 522_a.ino 예제를 522.ino로 저장합니다.

1. 다음과 같이 예제를 수정합니다.

522.ino

```
001 const int NUM_PATTERN = 10;
002 const int NUM_X = 7;
003 const int NUM_H = 8;
004 const int NUM_Y = 4;
005
006 double X[NUM_PATTERN][NUM_X] = {
007     { 1, 1, 1, 1, 1, 1, 0 }, // 0
008     { 0, 1, 1, 0, 0, 0, 0 }, // 1
009     { 1, 1, 0, 1, 1, 0, 1 }, // 2
010     { 1, 1, 1, 1, 0, 0, 1 }, // 3
011     { 0, 1, 1, 0, 0, 1, 1 }, // 4
012     { 1, 0, 1, 1, 0, 1, 1 }, // 5
013     { 0, 0, 1, 1, 1, 1, 1 }, // 6
014     { 1, 1, 1, 0, 0, 0, 0 }, // 7
015     { 1, 1, 1, 1, 1, 1, 1 }, // 8
016     { 1, 1, 1, 0, 0, 1, 1 } // 9
017 };
018 double YT[NUM_PATTERN][NUM_Y] = {
019     { 0, 0, 0, 0 },
020     { 0, 0, 0, 1 },
021     { 0, 0, 1, 0 },
022     { 0, 0, 1, 1 },
023     { 0, 1, 0, 0 },
024     { 0, 1, 0, 1 },
025     { 0, 1, 1, 0 },
```

```

026     { 0, 1, 1, 1 },
027     { 1, 0, 0, 0 },
028     { 1, 0, 0, 1 }
029 };
030 double H[NUM_H];
031 double Y[NUM_PATTERN][NUM_Y];
032
033 double WH[NUM_X][NUM_H];
034 double BH[NUM_H];
035 double WY[NUM_H][NUM_Y];
036 double BY[NUM_Y];
037
038 double YE[NUM_Y];
039 double HE[NUM_H];
040 double WYE[NUM_H][NUM_Y];
041 double BYE[NUM_Y];
042 double WHE[NUM_X][NUM_H];
043 double BHE[NUM_H];
044
045 void dnn_test() {
046
047     srand(time(NULL));
048
049     initialize_weight_He((double *)WH, NUM_X, NUM_H); //
050
051     initialize_weight_Le((double *)WY, NUM_H, NUM_Y); //
052
053     for(int epoch=1;epoch<10001;epoch++) { //
054
055         for(int pc=0;pc<NUM_PATTERN;pc++) { //
056
057             forward_Y(X[pc], (double *)WH, BH, H, NUM_X, NUM_H);
058             relu_Y(H, NUM_H);
059
060             forward_Y(H, (double *)WY, BY, Y[pc], NUM_H, NUM_Y);
061             sigmoid_Y(Y[pc], NUM_Y);
062
063             double E = calculate_MSE(Y[pc], YT[pc], NUM_Y);
064
065             backward_YE(YE, Y[pc], YT[pc], NUM_Y);
066             sigmoid_XE(YE, Y[pc], NUM_Y); //
067
068             backward_WE((double *)WYE, H, YE, NUM_H, NUM_Y);
069             backward_BE(BYE, YE, NUM_Y);
070

```

```

071         backward_HE(HE, (double *)WY, YE, NUM_H, NUM_Y);
072         relu_XE(HE, H, NUM_H);
073
074         backward_WE((double *)WHE, X[pc], HE, NUM_X, NUM_H);
075         backward_BE(BHE, HE, NUM_H);
076
077         double lr = 0.01;
078
079         learning_W((double *)WY, lr, (double *)WYE, NUM_H, NUM_Y);
080         learning_B(BY, lr, BYE, NUM_Y);
081
082         learning_W((double *)WH, lr, (double *)WHE, NUM_X, NUM_H);
083         learning_B(BH, lr, BHE, NUM_H);
084
085     }
086     if(epoch%100==0)
087         printf("epoch : %5d\n", epoch);
088
089 }
090 printf("\n");
091
092 for(int pc=0;pc<NUM_PATTERN;pc++) {
093     printf("target %d : ", pc);
094     for(int on=0;on<NUM_Y;on++) {
095         printf("%.0f ", YT[pc][on]);
096     }
097     printf("pattern %d : ", pc);
098     for(int on=0;on<NUM_Y;on++) {
099         printf("%.2f ", Y[pc][on]);
100     }
101     printf("\n");
102 }
103
104 }
105
106 void setup() {
107
108     Serial.begin(115200);
109     delay(1000);
110
111     dnn_test();
112
113 }
114
115 void loop() {

```

```
116  
117 }
```

053 : epoch 변수를 1000001(백만일) 미만에서 10001(일만일) 미만으로 변경합니다.
055 : pc 변수 0에서 NUM_PATTERN 미만에 대하여 57~83줄을 수행합니다.
57, 60, 61, 63, 65, 66, 74 : 숫자 2를 pc로 변경합니다. 63, 65 줄은 두 군데 변경합니다.
63~65 : if 조건문 2개를 없앱니다.
86 : epoch값이 100의 배수가 될 때마다 현재 학습 회수를 출력합니다.
90 : 개 행 문자를 출력합니다.
92~102 : 학습이 끝난 후에 목표 값과 예측 값을 출력하여 비교합니다.

2. 업로드를 수행합니다.



3. 출력결과를 확인합니다.

```
epoch : 10000  
  
target 0 : 0 0 0 0 pattern 0 : 0.02 0.01 0.02 0.03  
target 1 : 0 0 0 1 pattern 1 : 0.03 0.05 0.04 0.99  
target 2 : 0 0 1 0 pattern 2 : 0.02 0.00 1.00 0.00  
target 3 : 0 0 1 1 pattern 3 : 0.01 0.03 0.99 0.99  
target 4 : 0 1 0 0 pattern 4 : 0.02 0.96 0.00 0.00  
target 5 : 0 1 0 1 pattern 5 : 0.02 0.98 0.03 0.98  
target 6 : 0 1 1 0 pattern 6 : 0.00 1.00 0.97 0.00  
target 7 : 0 1 1 1 pattern 7 : 0.00 0.94 0.96 1.00  
target 8 : 1 0 0 0 pattern 8 : 0.97 0.00 0.01 0.01  
target 9 : 1 0 0 1 pattern 9 : 0.97 0.04 0.00 0.99
```

학습을 1만 번 수행한 후에 목표 값의 0과 1에 예측 값이 가까운 값을 갖는지 확인합니다.

03 입력 데이터 임의로 섞기

여기서는 매 회기마다 입력 데이터를 임의로 섞어 인공 신경망을 학습 시켜봅니다. 입력 데이터를 임의로 섞으면 인공 신경망 학습에 도움이 됩니다.

0. 먼저 523_a.ino 예제를 523.ino로 저장합니다.

1. 다음과 같이 파일을 수정합니다.

523.ino


```

001 const int NUM_PATTERN = 10;
002 const int NUM_X = 7;
003 const int NUM_H = 8;
004 const int NUM_Y = 4;
005
006 double X[NUM_PATTERN][NUM_X] = {
007     { 1, 1, 1, 1, 1, 1, 0 }, // 0
008     { 0, 1, 1, 0, 0, 0, 0 }, // 1
009     { 1, 1, 0, 1, 1, 0, 1 }, // 2
010     { 1, 1, 1, 1, 0, 0, 1 }, // 3
011     { 0, 1, 1, 0, 0, 1, 1 }, // 4
012     { 1, 0, 1, 1, 0, 1, 1 }, // 5
013     { 0, 0, 1, 1, 1, 1, 1 }, // 6
014     { 1, 1, 1, 0, 0, 0, 0 }, // 7
015     { 1, 1, 1, 1, 1, 1, 1 }, // 8
016     { 1, 1, 1, 0, 0, 1, 1 } // 9
017 };
018 double YT[NUM_PATTERN][NUM_Y] = {
019     { 0, 0, 0, 0 },
020     { 0, 0, 0, 1 },
021     { 0, 0, 1, 0 },
022     { 0, 0, 1, 1 },
023     { 0, 1, 0, 0 },
024     { 0, 1, 0, 1 },
025     { 0, 1, 1, 0 },
026     { 0, 1, 1, 1 },
027     { 1, 0, 0, 0 },
028     { 1, 0, 0, 1 }
029 };
030 double H[NUM_H];
031 double Y[NUM_PATTERN][NUM_Y];
032
033 double WH[NUM_X][NUM_H];
034 double BH[NUM_H];
035 double WY[NUM_H][NUM_Y];
036 double BY[NUM_Y];
037
038 double YE[NUM_Y];
039 double HE[NUM_H];
040 double WYE[NUM_H][NUM_Y];
041 double BYE[NUM_Y];
042 double WHE[NUM_X][NUM_H];
043 double BHE[NUM_H];
044
045 int shuffled_pattern[NUM_PATTERN]; //

```

```

046
047 void dnn_test() {
048
049     srand(time(NULL));
050
051     initialize_weight_He((double *)WH, NUM_X, NUM_H);
052
053     initialize_weight_Le((double *)WY, NUM_H, NUM_Y);
054
055     for(int pc=0;pc<NUM_PATTERN;pc++) { //
056         shuffled_pattern[pc] = pc; //
057     } //
058
059     for(int epoch=1;epoch<10001;epoch++) { //
060
061         int tmp_a = 0; //
062         int tmp_b = 0; //
063         for(int pc=0;pc<NUM_PATTERN;pc++) { //
064             tmp_a = rand()%NUM_PATTERN; //
065             tmp_b = shuffled_pattern[pc]; //
066             shuffled_pattern[pc] = shuffled_pattern[tmp_a]; //
067             shuffled_pattern[tmp_a] = tmp_b; //
068         } //
069
070         double sumError = 0;
071
072         for(int rc=0;rc<NUM_PATTERN;rc++) { //
073
074             int pc = shuffled_pattern[rc]; //
075
076             forward_Y(X[pc], (double *)WH, BH, H, NUM_X, NUM_H);
077             relu_Y(H, NUM_H);
078
079             forward_Y(H, (double *)WY, BY, Y[pc], NUM_H, NUM_Y);
080             sigmoid_Y(Y[pc], NUM_Y);
081
082             double E = calculate_MSE(Y[pc], YT[pc], NUM_Y);
083
084             sumError += E; //
085
086             backward_YE(YE, Y[pc], YT[pc], NUM_Y);
087             sigmoid_XE(YE, Y[pc], NUM_Y);
088
089             backward_WE((double *)WYE, H, YE, NUM_H, NUM_Y);
090             backward_BE(BYE, YE, NUM_Y);

```

```

091
092         backward_HE(HE, (double *)WY, YE, NUM_H, NUM_Y);
093         relu_XE(HE, H, NUM_H);
094
095         backward_WE((double *)WHE, X[pc], HE, NUM_X, NUM_H);
096         backward_BE(BHE, HE, NUM_H);
097
098         double lr = 0.01;
099
100         learning_W((double *)WY, lr, (double *)WYE, NUM_H, NUM_Y);
101         learning_B(BY, lr, BYE, NUM_Y);
102
103         learning_W((double *)WH, lr, (double *)WHE, NUM_X, NUM_H);
104         learning_B(BH, lr, BHE, NUM_H);
105
106     }
107     if(epoch%100==0) {
108         printf("epoch : %5d, sum error : %f\n", epoch, sumError);
109         for(int i=0;i<NUM_X;i++) {
110             for(int j=0;j<NUM_H;j++) {
111                 printf("%7.3f ", WH[i][j]);
112             }
113             printf("\n");
114         }
115
116         if(sumError<0.0001) break;
117     }
118
119 }
120 printf("\n");
121
122 for(int pc=0;pc<NUM_PATTERN;pc++) {
123     printf("target %d : ", pc);
124     for(int on=0;on<NUM_Y;on++) {
125         printf("%.0f ", YT[pc][on]);
126     }
127     printf("pattern %d : ", pc);
128     for(int on=0;on<NUM_Y;on++) {
129         printf("%.2f ", Y[pc][on]);
130     }
131     printf("\n");
132 }
133
134 }
135

```

```

136 void setup() {
137
138     Serial.begin(115200);
139     delay(1000);
140
141     dnn_test();
142
143 }
144
145 void loop() {
146
147 }

```

045 : NUM_PATTERN 개수의 정수 배열 shuffled_pattern을 선언합니다.

055~057 : shuffled_pattern 배열의 각 항목을 순서대로 초기화해 줍니다.

061, 062 : 입력 데이터의 순서를 변경하기 위해 사용할 정수 변수 2개를 선언합니다.

063 : pc 변수에 대해 0에서 NUM_PATTERN 미만에 대하여 064~067줄을 수행합니다.

064 : rand 함수를 호출하여 0에서 NUM_PATTERN 미만 사이 값을 생성하여 tmp_a 변수에 할당합니다. 이 예제에서는 0에서 10 미만 사이의 값이 생성됩니다.

065~067 : shuffled_pattern의 tmp_a 번째 항목과 pc 번째 항목을 서로 바꿔줍니다.

070 : sumError 실수 변수를 선언한 후, 0으로 초기화해줍니다.

072 : 이전 예제에서 pc를 rc로 변경해 줍니다.

074 : shuffled_pattern의 rc 번째 항목을 pc로 가져옵니다.

076~082 : 이전 예제와 같습니다.

084 : 082줄에서 얻은 오차 값을 sumError에 더해줍니다.

109~112 : 현재까지 학습된 가중치 WH 값을 출력해 봅니다.

116 : sumError 값이 0.0001보다 작으면 059줄의 for 문을 빠져 나와 120줄로 이동합니다.

122~132 : 이전 예제와 같습니다. 목표 값과 예측 값을 출력합니다.

2. 업로드를 수행합니다.



3. 출력결과를 확인합니다.

```

epoch : 10000, sum error : 0.011886
-1.271  0.426 -3.970  1.993 -0.256 -1.090  0.965  0.529
 1.582  2.070 -0.870 -0.489 -0.729  0.785 -0.419  0.172
 0.931  0.008  0.270  1.497  0.638  0.763 -0.545 -0.755
-0.969  0.760 -0.449  0.078 -0.668  0.650  1.543  0.473
-0.464  1.248  0.449 -2.397 -0.261 -1.045  1.342  0.516
 2.757 -0.955  2.145 -0.352 -1.182 -0.135  0.615 -0.663
 1.231  0.996  1.825 -1.262 -0.099 -0.541 -1.543  0.152

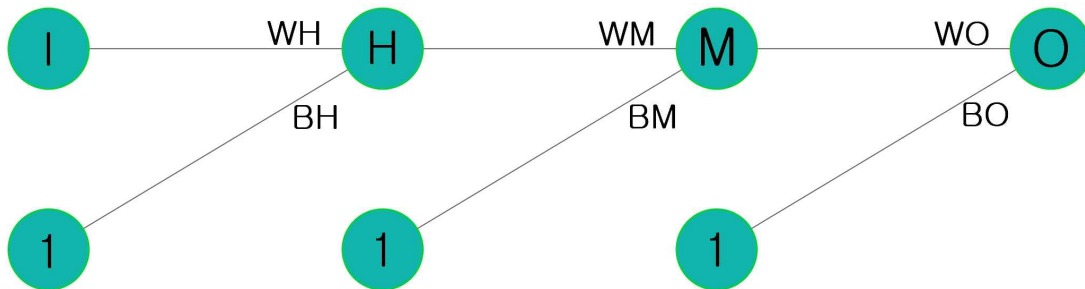
target 0 : 0 0 0 0 pattern 0 : 0.02 0.01 0.01 0.03
target 1 : 0 0 0 1 pattern 1 : 0.03 0.05 0.04 0.99
target 2 : 0 0 1 0 pattern 2 : 0.02 0.00 1.00 0.00
target 3 : 0 0 1 1 pattern 3 : 0.01 0.03 0.99 0.98
target 4 : 0 1 0 0 pattern 4 : 0.02 0.96 0.00 0.00
target 5 : 0 1 0 1 pattern 5 : 0.02 0.98 0.03 0.98
target 6 : 0 1 1 0 pattern 6 : 0.00 1.00 0.97 0.00
target 7 : 0 1 1 1 pattern 7 : 0.00 0.94 0.96 1.00
target 8 : 1 0 0 0 pattern 8 : 0.96 0.00 0.01 0.01
target 9 : 1 0 0 1 pattern 9 : 0.97 0.04 0.00 0.99

```

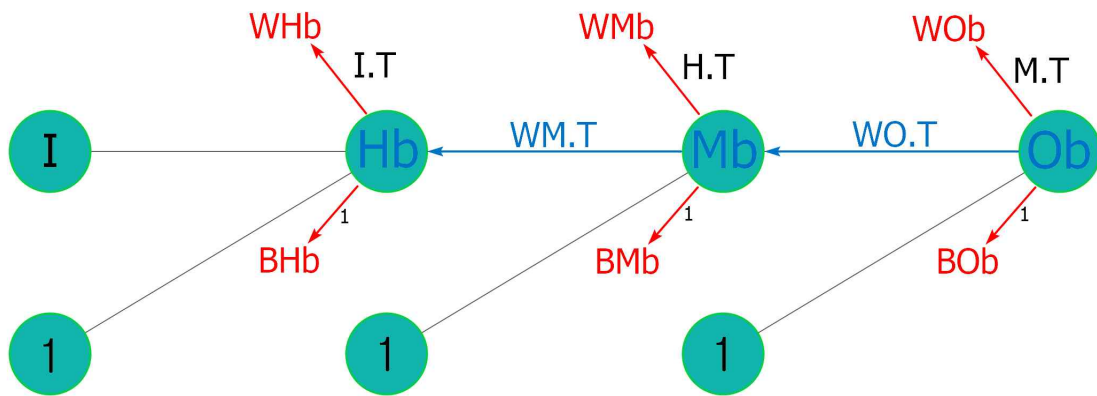
학습이 진행됨에 따라 가중치 값이 갱신되는 것을 볼 수 있습니다. 학습이 끝나기 전 마지막 1회 가중치 갱신 결과를 볼 수 있으며, 마지막에는 학습된 결과의 예측 값을 목표 값과 비교하여 보여줍니다. 예측 값이 목표 값에 적당히 가까운 것을 볼 수 있습니다. 예측 값을 목표 값에 더 가깝게 하려면 훈련의 횟수를 늘리면 됩니다.

04 은닉층 추가하기

여기서는 은닉층을 하나 더 추가해 봅니다. 일반적으로 은닉층의 개수가 2개 이상일 때 심층 인공 신경망이라고 합니다. 다음은 은닉층 M이 추가된 X-H-M-Y 심층 인공 신경망입니다. 이 신경망은 순전파 과정을 나타냅니다.



다음은 역전파 과정을 나타냅니다.



0. 먼저 524_0.ino 예제를 524.ino로 저장합니다.

1. 다음과 같이 파일을 수정합니다.

524.ino

```
001 const int NUM_PATTERN = 10;
002 const int NUM_X = 7;
003 const int NUM_H = 16; //
004 const int NUM_M = 16; //
005 const int NUM_Y = 4;
006
007 double X[NUM_PATTERN][NUM_X] = {
008     { 1, 1, 1, 1, 1, 1, 0 }, // 0
009     { 0, 1, 1, 0, 0, 0, 0 }, // 1
010     { 1, 1, 0, 1, 1, 0, 1 }, // 2
011     { 1, 1, 1, 1, 0, 0, 1 }, // 3
012     { 0, 1, 1, 0, 0, 1, 1 }, // 4
013     { 1, 0, 1, 1, 0, 1, 1 }, // 5
014     { 0, 0, 1, 1, 1, 1, 1 }, // 6
015     { 1, 1, 1, 0, 0, 0, 0 }, // 7
016     { 1, 1, 1, 1, 1, 1, 1 }, // 8
017     { 1, 1, 1, 0, 0, 1, 1 } // 9
018 };
019 double YT[NUM_PATTERN][NUM_Y] = {
020     { 0, 0, 0, 0 },
021     { 0, 0, 0, 1 },
022     { 0, 0, 1, 0 },
023     { 0, 0, 1, 1 },
024     { 0, 1, 0, 0 },
025     { 0, 1, 0, 1 },
026     { 0, 1, 1, 0 },
027     { 0, 1, 1, 1 },
028     { 1, 0, 0, 0 },
029     { 1, 0, 0, 1 }
030 };
031 double H[NUM_H];
032 double M[NUM_M]; //
```

```

033 double Y[NUM_PATTERN][NUM_Y];
034
035 double WH[NUM_X][NUM_H];
036 double BH[NUM_H];
037 double WM[NUM_H][NUM_M]; //
038 double BM[NUM_M]; //
039 double WY[NUM_M][NUM_Y]; // NUM_M
040 double BY[NUM_Y];
041
042 double YE[NUM_Y];
043 double ME[NUM_M]; //
044 double HE[NUM_H];
045 double WYE[NUM_M][NUM_Y]; //
046 double BYE[NUM_Y];
047 double WME[NUM_H][NUM_M]; // NUM_M
048 double BME[NUM_M]; //
049 double WHE[NUM_X][NUM_H];
050 double BHE[NUM_H];
051
052 int shuffled_pattern[NUM_PATTERN];
053
054 void dnn_test() {
055
056     srand(time(NULL));
057
058     initialize_weight_He((double *)WH, NUM_X, NUM_H);
059
060     initialize_weight_He((double *)WM, NUM_H, NUM_M); //
061
062     initialize_weight_Le((double *)WY, NUM_M, NUM_Y); // NUM_M
063
064     for(int pc=0;pc<NUM_PATTERN;pc++) {
065         shuffled_pattern[pc] = pc;
066     }
067
068     long begin = millis(); //
069
070     for(int epoch=1;epoch<1001;epoch++) {
071
072         int tmp_a = 0;
073         int tmp_b = 0;
074         for(int pc=0;pc<NUM_PATTERN;pc++) {
075             tmp_a = rand()%NUM_PATTERN;
076             tmp_b = shuffled_pattern[pc];
077             shuffled_pattern[pc] = shuffled_pattern[tmp_a];
078             shuffled_pattern[tmp_a] = tmp_b;
079         }
080
081         double sumError = 0;
082
083         static long t_prev = millis();

```

```

084
085         for(int rc=0;rc<NUM_PATTERN;rc++) {
086
087             int pc = shuffled_pattern[rc];
088
089             forward_Y(X[pc], (double *)WH, BH, H, NUM_X, NUM_H);
090             relu_Y(H, NUM_H);
091
092             forward_Y(H, (double *)WM, BM, M, NUM_H, NUM_M); //
093             relu_Y(M, NUM_M); //
094
095             forward_Y(M, (double *)WY, BY, Y[pc], NUM_M, NUM_Y); // M, NUM_M
096             sigmoid_Y(Y[pc], NUM_Y);
097
098             double E = calculate_MSE(Y[pc], YT[pc], NUM_Y);
099
100             sumError += E;
101
102             backward_YE(YE, Y[pc], YT[pc], NUM_Y);
103             sigmoid_XE(YE, Y[pc], NUM_Y);
104
105             backward_WE((double *)WYE, M, YE, NUM_M, NUM_Y); // M, NUM_M
106             backward_BE(BYE, YE, NUM_Y);
107
108             backward_HE(ME, (double *)WY, YE, NUM_M, NUM_Y); //
109             relu_XE(ME, M, NUM_M); //
110
111             backward_WE((double *)WME, H, ME, NUM_H, NUM_M); //
112             backward_BE(BME, ME, NUM_M); //
113
114             backward_HE(HE, (double *)WM, ME, NUM_H, NUM_M); //
115             relu_XE(HE, H, NUM_H);
116
117             backward_WE((double *)WHE, X[pc], HE, NUM_X, NUM_H);
118             backward_BE(BHE, HE, NUM_H);
119
120             double lr = 0.01;
121
122             learning_W((double *)WY, lr, (double *)WYE, NUM_M, NUM_Y); //
123             learning_B(BY, lr, BYE, NUM_Y);
124
125             learning_W((double *)WM, lr, (double *)WME, NUM_H, NUM_M); //
126             learning_B(BM, lr, BME, NUM_M); //
127
128             learning_W((double *)WH, lr, (double *)WHE, NUM_X, NUM_H);
129             learning_B(BH, lr, BHE, NUM_H);
130
131         }
132         if(epoch%100==0) {
133             long t_now = millis(); //
134             long time_taken = t_now - t_prev; //

```



```

135         t_prev = t_now; //
136         printf("epoch : %5d, sum error : %f ", epoch, sumError); //
137         printf(", %.3f sec\n", time_taken/1000.0); //
138
139         if(sumError<0.0001) break;
140     }
141
142 }
143 printf("\n");
144
145 for(int pc=0;pc<NUM_PATTERN;pc++) {
146     printf("target %d : ", pc);
147     for(int on=0;on<NUM_Y;on++) {
148         printf("%.0f ", YT[pc][on]);
149     }
150     printf("pattern %d : ", pc);
151     for(int on=0;on<NUM_Y;on++) {
152         printf("%.2f ", Y[pc][on]);
153     }
154     printf("\n");
155 }
156
157 long end = millis();
158 long time_taken = end - begin;
159 printf("\nTotal time taken (in seconds) %.3f\n", time_taken/1000.0);
160
161 }
162
163 void setup() {
164
165     Serial.begin(115200);
166     delay(1000);
167
168     dnn_test();
169
170 }
171
172 void loop() {
173
174 }

```

003 : NUM_H 값을 16으로 변경합니다.

004 : NUM_M 변수를 선언한 후, 16으로 초기화합니다. NUM_M 변수는 2차 은닉 층 노드의 개수를 저장합니다.

032 : M 일차 배열을 추가합니다.

037 : 가중치 변수 WM 이차 배열을 추가합니다.

038 : 편향 변수 BM 일차 배열을 추가합니다.

039 : NUM_H를 NUM_M으로 변경합니다.

043 : 은닉층 역전파에 사용할 ME 일차 배열을 추가합니다.

045 : NUM_H를 NUM_M으로 변경합니다.

047 : 역전파에 사용할 WME 이차 배열을 추가합니다.
 048 : 역전파에 사용할 BME 일차 배열을 추가합니다.
 060 : WM 배열에 대해 He 초기화를 수행합니다.
 062 : NUM_H를 NUM_M으로 변경합니다.
 068 : begin 변수를 선언하고, millis 함수를 호출하여 밀리 초 단위의 현재 시간으로 초기화합니다. begin은 학습을 시작한 최초 시간을 나타냅니다. begin 변수는 160줄에서 사용되어 전체 학습 시간을 측정합니다.
 083 : t_prev 정적 변수를 선언하고, millis 함수를 호출하여 밀리 초 단위의 현재 시간으로 초기화합니다. t_prev 변수는 134,135줄에서 사용되어 학습을 100회 수행할 때마다의 학습 시간을 측정합니다.
 092,093 : 은닉 층 M의 순전파 과정을 추가합니다.
 095 : H를 M으로, NUM_H를 NUM_M으로 바꿔줍니다.
 105: H를 M으로 NUM_H를 NUM_M으로 바꿔줍니다.
 108,109 : 은닉 층 M의 역전파 과정을 추가합니다.
 114,115 : 은닉 층 H의 역전파 과정을 추가합니다.
 125,126 : 은닉 층 M의 가중치, 편향 갱신 과정을 추가합니다.
 132 : epoch 값이 100의 배수이면 133~139줄을 수행합니다.
 133 : t_now 변수를 선언하고, millis 함수를 호출하여 밀리 초 단위의 현재 시간을 저장합니다.
 134 : 학습 100회 수행에 대해 현재 측정 시간에서 이전 측정 시간을 빼서 time_taken 변수에 저장합니다. time_taken 변수는 학습을 100회 수행할 때마다의 학습 시간을 저장합니다.
 135 : t_prev 변수값을 t_now 변수값으로 갱신합니다.
 136 : '\n' 문자를 뺍니다.
 137 : 학습을 100회 수행할 때마다의 학습 시간을 출력합니다.
 137~139 : 가중치 WH 값의 출력부분을 지워줍니다.
 157 : end 변수를 선언하고, millis 함수를 호출하여 밀리 초 단위의 현재 시간을 저장합니다.
 158 : end에서 begin을 빼서 time_taken 변수에 저장합니다.
 159 : 전체 학습 시간을 출력합니다.

2. 업로드를 수행합니다.



3. 출력결과를 확인합니다.

```
epoch : 10000, sum error : 0.001454 , 1.086 sec

target 0 : 0 0 0 0 pattern 0 : 0.01 0.01 0.01 0.01
target 1 : 0 0 0 1 pattern 1 : 0.00 0.01 0.01 0.99
target 2 : 0 0 1 0 pattern 2 : 0.01 0.00 0.99 0.01
target 3 : 0 0 1 1 pattern 3 : 0.00 0.01 0.99 1.00
target 4 : 0 1 0 0 pattern 4 : 0.01 0.99 0.00 0.01
target 5 : 0 1 0 1 pattern 5 : 0.01 0.99 0.01 1.00
target 6 : 0 1 1 0 pattern 6 : 0.01 1.00 0.99 0.00
target 7 : 0 1 1 1 pattern 7 : 0.00 0.99 0.99 1.00
target 8 : 1 0 0 0 pattern 8 : 0.99 0.00 0.01 0.00
target 9 : 1 0 0 1 pattern 9 : 0.99 0.01 0.00 0.99


Total time taken (in seconds) 107.957
```

05 입력층과 목표층 바꿔보기

먼저 이전 예제의 입력층과 목표층을 바꿔 인공 신경망을 학습 시켜봅니다. 다음과 같이 2진수가 입력되면 해당되는 7 세그먼트의 켜지고 꺼져야 할 LED의 비트열을 출력합니다.

2 진수 7 세그먼트 연결 진리표

In	In	In	In	Out	Out	Out	Out	Out	Out	Out
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

0101 ➡ 1011011 = 

예를 들어, “숫자 5에 맞게 7 세그먼트 LED를 켜줘!” 하고 싶을 때, 사용할 수 있는 인공 신경망입니다.

0. 먼저 525_0.ino 예제를 525.ino로 저장합니다.

1. 다음과 같이 파일을 수정합니다.

525.ino

```
001 const int NUM_PATTERN = 10;
002 const int NUM_X = 4;
003 const int NUM_H = 16;
004 const int NUM_M = 16;
005 const int NUM_Y = 7;
006
007 double X[NUM_PATTERN][NUM_X] = {
008     { 0, 0, 0, 0 }, // 0
009     { 0, 0, 0, 1 }, // 1
010     { 0, 0, 1, 0 }, // 2
011     { 0, 0, 1, 1 }, // 3
012     { 0, 1, 0, 0 }, // 4
013     { 0, 1, 0, 1 }, // 5
014     { 0, 1, 1, 0 }, // 6
015     { 0, 1, 1, 1 }, // 7
016     { 1, 0, 0, 0 }, // 8
017     { 1, 0, 0, 1 } // 9
018 };
019 double YT[NUM_PATTERN][NUM_Y] = {
020     { 1, 1, 1, 1, 1, 1, 0 }, // 0
021     { 0, 1, 1, 0, 0, 0, 0 }, // 1
022     { 1, 1, 0, 1, 1, 0, 1 }, // 2
023     { 1, 1, 1, 1, 0, 0, 1 }, // 3
024     { 0, 1, 1, 0, 0, 1, 1 }, // 4
025     { 1, 0, 1, 1, 0, 1, 1 }, // 5
026     { 0, 0, 1, 1, 1, 1, 1 }, // 6
027     { 1, 1, 1, 0, 0, 0, 0 }, // 7
028     { 1, 1, 1, 1, 1, 1, 1 }, // 8
029     { 1, 1, 1, 0, 0, 1, 1 } // 9
030 };
```

003 : NUM_X 변수의 값을 4로 변경합니다.

005 : NUM_Y 변수의 값을 7로 변경합니다.

008~017 : 입력층의 입력값을 출력층의 값으로 변경합니다.

020~029 : 목표층의 목표값을 입력층의 값으로 변경합니다.

2. 업로드를 수행합니다.



3. 출력결과를 확인합니다.

```
epoch : 10000, sum error : 0.003659 , 1.102 sec

target 0 : 1 1 1 1 1 1 0 pattern 0 : 1.00 0.99 0.98 0.98 0.99 0.98 0.03
target 1 : 0 1 1 0 0 0 0 pattern 1 : 0.03 1.00 1.00 0.00 0.00 0.01 0.02
target 2 : 1 1 0 1 1 0 1 pattern 2 : 1.00 1.00 0.01 1.00 1.00 0.01 1.00
target 3 : 1 1 1 1 0 0 1 pattern 3 : 0.99 1.00 0.99 0.98 0.01 0.00 0.99
target 4 : 0 1 1 0 0 1 1 pattern 4 : 0.01 0.99 1.00 0.01 0.01 1.00 1.00
target 5 : 1 0 1 1 0 1 1 pattern 5 : 1.00 0.01 1.00 0.98 0.01 1.00 1.00
target 6 : 0 0 1 1 1 1 1 pattern 6 : 0.01 0.01 0.99 0.99 0.99 0.99 0.99
target 7 : 1 1 1 0 0 0 0 pattern 7 : 0.98 0.99 1.00 0.02 0.00 0.00 0.01
target 8 : 1 1 1 1 1 1 1 pattern 8 : 1.00 1.00 1.00 1.00 0.99 1.00 0.99
target 9 : 1 1 1 0 0 1 1 pattern 9 : 0.99 1.00 1.00 0.01 0.00 1.00 0.99

Total time taken (in seconds) 109.855
```

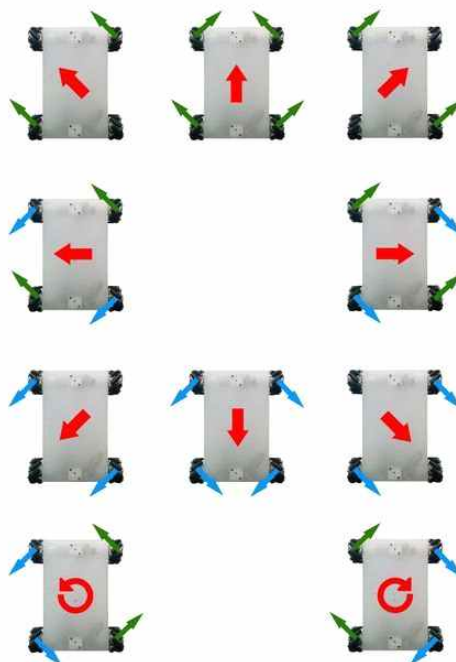
예측 값이 목표 값에 적당히 가까운 것을 볼 수 있습니다. 예측값을 목표값에 더 가깝게 하려면 훈련의 횟수를 늘리면 됩니다.

06 7 세그먼트 비트열로 매카넘 바퀴 제어하기

여기서는 7 세그먼트의 비트열을 입력으로 받아 매카넘 휠의 모터를 제어하는 출력을 내도록 인공 신경망을 구성하고, 학습시켜 봅니다.

	D7	D6	D5	D4	D3	D2	D1	D0	16진 코드 (C 언어)
	a	b	c	d	e	f	g	dp	
0	1	1	1	1	1	1	0	0	0xFC
1	0	1	1	0	0	0	0	0	0x60
2	1	1	0	1	1	0	1	0	0xDA
3	1	1	1	1	0	0	1	0	0xF2
4	0	1	1	0	0	1	1	0	0x66
5	1	0	1	1	0	1	1	0	0xB6
6	1	0	1	1	1	1	1	0	0xBE
7	1	1	1	0	0	1	0	0	0xE4
8	1	1	1	1	1	1	1	0	0xFE
9	1	1	1	1	0	1	1	0	0xF6
A	1	1	1	0	1	1	1	0	0xEE
B	0	0	1	1	1	1	1	0	0x3E
C	1	0	0	1	1	1	0	0	0x9C
D	0	1	1	1	1	0	1	0	0x7A
E	1	0	0	1	1	1	1	0	0x9E
F	1	0	0	0	1	1	1	0	0x8E

7세그먼트 표준 디스플레이 모양



예를 들어, “7 세그먼트 숫자 3의 비트열에 맞게 4 바퀴의 매카넘 바퀴를 움직여줘!” 하고 싶

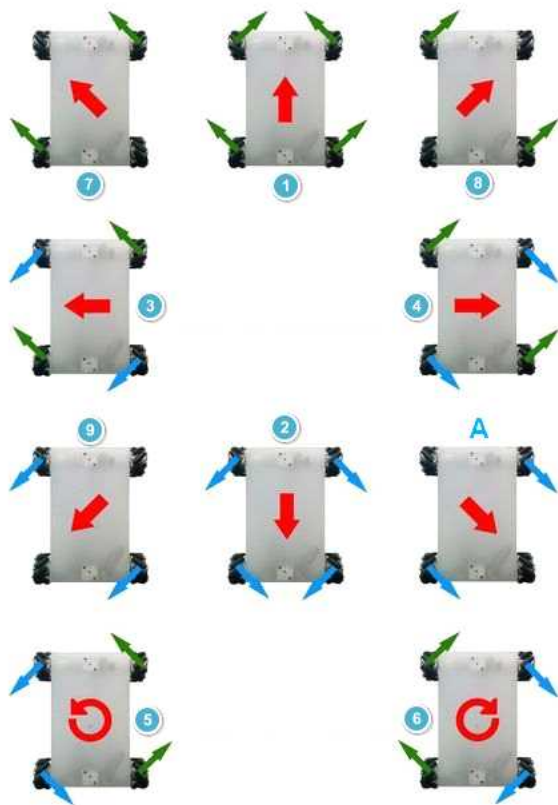
을 때, 사용할 수 있는 인공 신경망입니다. 우리 예제에서 이 경우 메카넘 바퀴를 장착한 자동차는 다음과 같이 왼쪽으로 수평 이동합니다.



다음과 같이 입력과 출력을 연결할 수 있도록 인공 신경망을 학습시킵니다.

- | |
|--------------|
| 0 : 멈춤 |
| 1 : 전진 |
| 2 : 후진 |
| 3 : 좌이동 |
| 4 : 우이동 |
| 5 : 좌회전 |
| 6 : 우회전 |
| 7 : 좌 대각선 전진 |
| 8 : 우 대각선 전진 |
| 9 : 좌 대각선 후진 |
| A : 우 대각선 후진 |

다음 그림을 참고합니다.



0. 먼저 526_0.ino 예제를 526.ino로 저장합니다.

1. 다음과 같이 파일을 수정합니다.

526.ino

```
001 const int NUM_PATTERN = 11;
002 const int NUM_X = 7;
003 const int NUM_H = 16;
004 const int NUM_M = 16;
005 const int NUM_Y = 4;
006
007 double X[NUM_PATTERN][NUM_X] = {
008     { 1, 1, 1, 1, 1, 1, 0 }, // 0
009     { 0, 1, 1, 0, 0, 0, 0 }, // 1
010     { 1, 1, 0, 1, 1, 0, 1 }, // 2
011     { 1, 1, 1, 1, 0, 0, 1 }, // 3
012     { 0, 1, 1, 0, 0, 1, 1 }, // 4
013     { 1, 0, 1, 1, 0, 1, 1 }, // 5
014     { 0, 0, 1, 1, 1, 1, 1 }, // 6
015     { 1, 1, 1, 0, 0, 0, 0 }, // 7
016     { 1, 1, 1, 1, 1, 1, 1 }, // 8
```

```

017    { 1, 1, 1, 0, 0, 1, 1 }, // 9
018    { 1, 1, 1, 0, 1, 1, 1 }, // A
019 };
020 double YT[NUM_PATTERN][NUM_Y] = {
021     { 0.5, 0.5, 0.5, 0.5 }, // 0 멈춤
022     { 1, 1, 1, 1 }, // 1 전진
023     { 0, 0, 0, 0 }, // 2 후진
024     { 0, 1, 0, 1 }, // 3 좌이동
025     { 1, 0, 1, 0 }, // 4 우이동
026     { 0, 1, 1, 0 }, // 5 좌회전
027     { 1, 0, 0, 1 }, // 6 우회전
028     { 0.5, 1, 0.5, 1 }, // 7 좌 대각선 전진
029     { 1, 0.5, 1, 0.5 }, // 8 우 대각선 전진
030     { 0.5, 0, 0.5, 0 }, // 9 좌 대각선 후진
031     { 0, 0.5, 0, 0.5 }, // A 우 대각선 후진
032 };

```

001 : 패턴의 개수는 0~9, A까지 11개가 됩니다.

002 : 입력 층의 노드 개수는 7개로 합니다. 7 세그먼트의 숫자 표시 LED의 개수가 7개이기 때문입니다.

005 : 출력 층의 노드 개수는 4개로 합니다. 4바퀴에 각각에 대한 전진, 멈춤, 후진을 나타내는 값을 출력하게 됩니다.

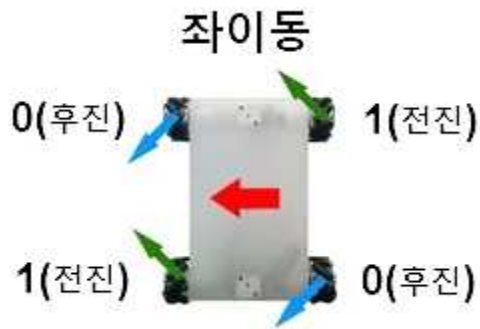
007~019 : 입력 배열 I의 값을 초기화합니다. 입력 값은 11가지로 7 세그먼트의 0~9, A에 대응되는 비트열입니다. 다음 그림의 D7~D1에 대응되는 비트열입니다.

	D7	D6	D5	D4	D3	D2	D1	D0	16진 코드 (C 언어)
	a	b	c	d	e	f	g	dp	
	1	1	1	1	1	1	0	0	0xFC
	0	1	1	0	0	0	0	0	0x60
	1	1	0	1	1	0	1	0	0xDA
	1	1	1	1	0	0	1	0	0xF2
	0	1	1	0	0	1	1	0	0x66
	1	0	1	1	0	1	1	0	0xB6
	1	0	1	1	1	1	1	0	0xBE
	1	1	1	0	0	1	0	0	0xE4
	1	1	1	1	1	1	1	0	0xFE
	1	1	1	1	0	1	1	0	0xF6
	1	1	1	0	1	1	1	0	0xEE
	0	0	1	1	1	1	1	0	0x3E
	1	0	0	1	1	1	0	0	0x9C
	0	1	1	1	1	0	1	0	0x7A
	1	0	0	1	1	1	1	0	0x9E
	1	0	0	0	1	1	1	0	0x8E

7세그먼트 표준 디스플레이 모양

020~032 : 목표 값 배열 YT를 초기화합니다. 0번 항목의 경우 메카닉 바퀴를 멈추기 위한 4바퀴의 값입니다. 차례대로 왼쪽 앞바퀴, 오른쪽 앞바퀴, 오른쪽 뒷바퀴, 왼쪽 뒷바퀴에 대응

되는 값입니다. 0.5의 경우 멈춤입니다. 1은 전진을 나타내고, 0은 후진을 나타냅니다. 그래서 1번 항목의 경우 전진을 위한 출력 값이며 4 바퀴의 값이 모두 1입니다. 2번 항목의 경우 후진을 위한 출력 값이며 4 바퀴의 값이 모두 0입니다. 3번 항목의 경우 좌 이동을 위한 출력 값이며 4 바퀴의 값이 각각 0(후진), 1(전진), 0(후진), 1(전진)이 됩니다. 다음 그림은 좌 이동을 나타내는 그림입니다.



2. 계속해서 다음과 같이 파일을 수정합니다.

526_2.ino

```

147     for(int pc=0;pc<NUM_PATTERN;pc++) {
148         printf("target %X : ", pc);
149         for(int on=0;on<NUM_Y;on++) {
150             printf("%.0f ", YT[pc][on]);
151         }
152         printf("pattern %X : ", pc);
153         for(int on=0;on<NUM_Y;on++) {
154             printf("%.2f ", Y[pc][on]);
155         }
156         printf("\n");
157     }

```

148, 152 : %d를 %X로 변경하여 10진수를 16진수로 표시하게 합니다.

3. 업로드를 수행합니다.



4. 출력결과를 확인합니다.

```
epoch : 10000, sum error : 0.001959 , 1.159 sec

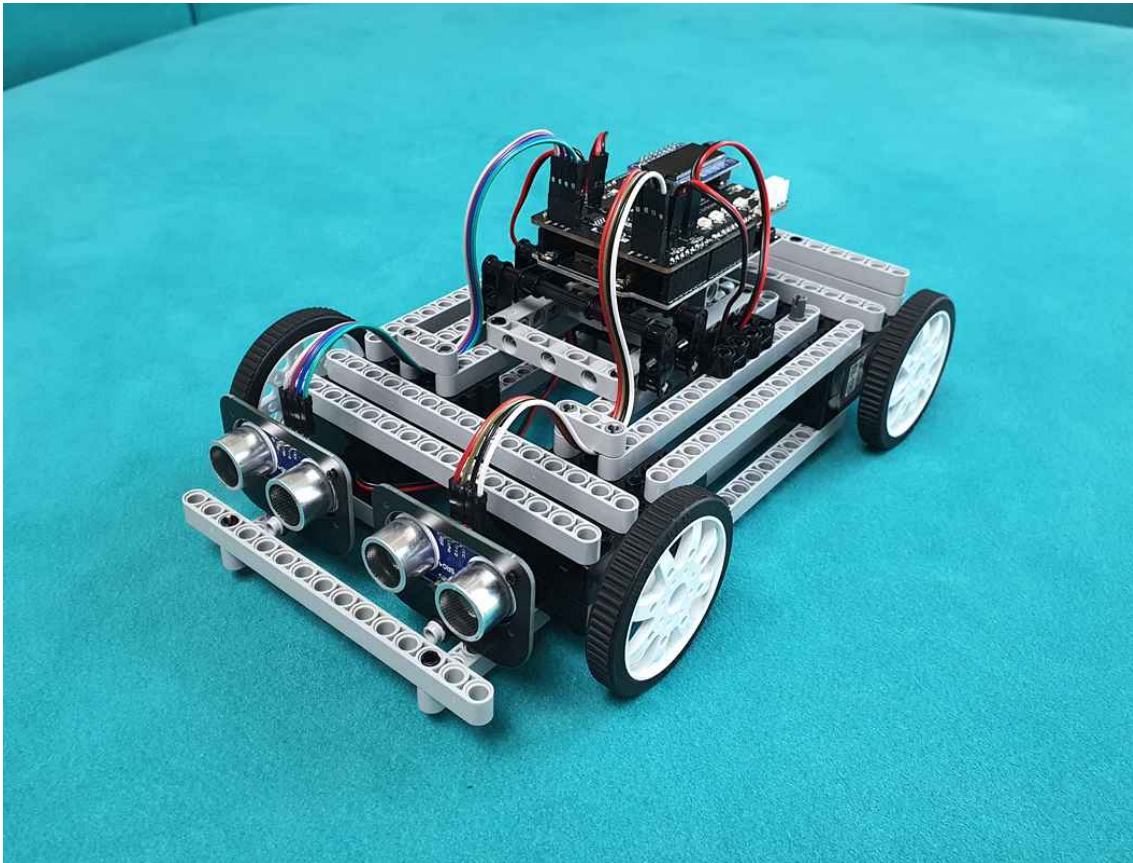
target 0 : 0 0 0 0 pattern 0 : 0.50 0.50 0.50 0.50
target 1 : 1 1 1 1 pattern 1 : 1.00 0.99 1.00 0.99
target 2 : 0 0 0 0 pattern 2 : 0.00 0.01 0.00 0.01
target 3 : 0 1 0 1 pattern 3 : 0.01 0.99 0.01 0.99
target 4 : 1 0 1 0 pattern 4 : 1.00 0.00 0.99 0.01
target 5 : 0 1 1 0 pattern 5 : 0.00 0.99 0.99 0.01
target 6 : 1 0 0 1 pattern 6 : 1.00 0.01 0.02 0.99
target 7 : 0 1 0 1 pattern 7 : 0.50 1.00 0.50 1.00
target 8 : 1 0 1 0 pattern 8 : 0.98 0.50 0.97 0.50
target 9 : 0 0 0 0 pattern 9 : 0.50 0.01 0.50 0.01
target A : 0 0 0 0 pattern A : 0.02 0.50 0.02 0.50

Total time taken (in seconds) 115.813
```

예측 값이 목표 값에 적당히 가까운 것을 볼 수 있습니다. 예측값을 목표값에 더 가깝게 하려면 훈련의 횟수를 늘리면 됩니다.

07 초음파 센서 자율주행 인공 신경망

여기서는 자동차에 장착된 초음파 센서로부터 물체와의 거리를 입력받아 자동차의 모터를 제어하여 출력을 내도록 인공 신경망을 구성하고, 학습시켜 봅니다. 다음 그림은 초음파 센서가 장착된 아두이노 자동차입니다.



예를 들어, “왼쪽 25cm, 오른쪽 14cm에 물체가 있으면 왼쪽으로 움직여줘!” 하고 싶을 때, 사용할 수 있는 인공 신경망입니다.

0. 먼저 527_0.ino 예제를 527.ino로 저장합니다.

1. 다음과 같이 파일을 수정합니다.

527.ino

```
001 const int NUM_PATTERN = 25;
002 const int NUM_X = 2;
003 const int NUM_H = 16;
004 const int NUM_M = 16;
005 const int NUM_Y = 3;
006
007 double X[NUM_PATTERN][NUM_X] = {
008     {25,14},    //1
009     {41,33},    //2
010     {44,44},    //3
011     {33,41},    //4
012     {14,25},    //5
```

```

013     {29,22},    //6
014     {43,33},    //7
015     {80,90},    //8
016     {33,43},    //9
017     {22,29},    //10
018     {35,26},    //11
019     {55,35},    //12
020     {55,55},    //13
021     {35,55},    //14
022     {26,35},    //15
023     {33,25},    //16
024     {44,32},    //17
025     {150,150},  //18
026     {32,44},    //19
027     {25,33},    //20
028     {38,23},    //21
029     {50,36},    //22
030     {90,100},   //23
031     {36,50},    //24
032     {23,38}     //25
033 };
034 double YT[NUM_PATTERN][NUM_Y] = {
035     {1,0,0},    //1
036     {0,1,0},    //2
037     {0,1,0},    //3
038     {0,1,0},    //4
039     {0,0,1},    //5
040     {1,0,0},    //6
041     {0,1,0},    //7
042     {0,1,0},    //8
043     {0,1,0},    //9
044     {0,0,1},    //10
045     {1,0,0},    //11
046     {0,1,0},    //12
047     {0,1,0},    //13
048     {0,1,0},    //14
049     {0,0,1},    //15
050     {1,0,0},    //16
051     {0,1,0},    //17
052     {0,1,0},    //18

```

```

053     {0,1,0}, //19
054     {0,0,1}, //20
055     {1,0,0}, //21
056     {0,1,0}, //22
057     {0,1,0}, //23
058     {0,1,0}, //24
059     {0,0,1} //25
060 };

```

001 : 패턴의 개수는 25개로 합니다.

002 : 입력층의 노드 개수는 2개로 합니다. 오른쪽, 왼쪽 2 방향의 거리 값이 입력이 됩니다.

005 : 출력층의 노드 개수는 3개로 합니다. 오른쪽 전진, 왼쪽 전진, 양쪽 전진의 3가지 동작을 나타내는 값을 출력하게 됩니다.

007~033 : 입력 배열 I의 값을 초기화합니다. 입력 값은 25가지로 오른쪽, 왼쪽 2 방향의 거리 값입니다. 예를 들어 0번 항목의 경우 왼쪽이 25cm, 오른쪽이 14cm일 경우를 나타냅니다.

034~060 : 목표 값 배열 T를 초기화합니다. 0번 항목의 경우 왼쪽 전진을 의미합니다. 왼쪽이 25cm, 오른쪽이 14cm일 경우 물체가 더 먼 왼쪽 방향으로 이동해야 합니다. 2번 항목의 경우 양쪽 전진을 의미합니다. 물체가 조금 멀리 있는 경우로, 양쪽 전진을 하도록 합니다. 5번 항목의 경우 오른쪽 전진을 의미합니다. 왼쪽이 14cm, 오른쪽이 25cm일 경우 물체가 더 먼 오른쪽 방향으로 이동해야 합니다.

2. 계속해서 다음과 같이 파일을 수정합니다.

527_2.ino

```

084 void dnn_test() {
085
086     for(int pc=0;pc<NUM_PATTERN;pc++) {
087         for(int in=0;in<NUM_X;in++) {
088             X[pc][in] /= 250;
089         }
090     }
091
092     srand(time(NULL));

```

086~090 : 초음파 센서의 입력 값을 250으로 나누어 0.0~1.0 사이의 값이 되도록 합니다. 이 예제에서 사용하는 인공 신경망의 입력 값이 0.0~1.0 사이가 되게 합니다.

3. 계속해서 다음과 같이 파일을 수정합니다.

527_2.ino

```

181     for(int pc=0;pc<NUM_PATTERN;pc++) {
182         printf("target %2d : ", pc);
183         for(int on=0;on<NUM_Y;on++) {

```

```

184         printf("%.0f ", YT[pc][on]);
185     }
186     printf("pattern %2d : ", pc);
187     for(int on=0;on<NUM_Y;on++) {
188         printf("%.2f ", Y[pc][on]);
189     }
190     printf("\n");
191 }

```

182, 186 : 입력 패턴의 종류가 25개이기 때문에 printf 함수의 출력 형식을 %2d로 하여 10진수 2자리로 출력하도록 합니다.

4. 업로드를 수행합니다.



5. 출력결과를 확인합니다.

```

epoch : 10000, sum error : 0.010452 , 2.133 sec

target 0 : 1 0 0 pattern 0 : 1.00 0.00 0.01
target 1 : 0 1 0 pattern 1 : 0.02 0.98 0.00
target 2 : 0 1 0 pattern 2 : 0.00 1.00 0.00
target 3 : 0 1 0 pattern 3 : 0.00 0.98 0.02
target 4 : 0 0 1 pattern 4 : 0.01 0.01 0.99
target 5 : 1 0 0 pattern 5 : 0.98 0.00 0.05
target 6 : 0 1 0 pattern 6 : 0.02 0.99 0.00
target 7 : 0 1 0 pattern 7 : 0.00 1.00 0.00
target 8 : 0 1 0 pattern 8 : 0.00 0.99 0.02
target 9 : 0 0 1 pattern 9 : 0.04 0.01 0.95
target 10 : 1 0 0 pattern 10 : 0.94 0.02 0.00
target 11 : 0 1 0 pattern 11 : 0.00 1.00 0.00
target 12 : 0 1 0 pattern 12 : 0.00 1.00 0.00
target 13 : 0 1 0 pattern 13 : 0.00 1.00 0.00
target 14 : 0 0 1 pattern 14 : 0.01 0.02 0.96
target 15 : 1 0 0 pattern 15 : 0.97 0.00 0.02
target 16 : 0 1 0 pattern 16 : 0.04 0.99 0.00
target 17 : 0 1 0 pattern 17 : 0.00 1.00 0.00
target 18 : 0 1 0 pattern 18 : 0.00 0.99 0.04
target 19 : 0 0 1 pattern 19 : 0.02 0.01 0.97
target 20 : 1 0 0 pattern 20 : 0.99 0.02 0.00
target 21 : 0 1 0 pattern 21 : 0.00 1.00 0.00
target 22 : 0 1 0 pattern 22 : 0.00 1.00 0.00
target 23 : 0 1 0 pattern 23 : 0.00 1.00 0.00
target 24 : 0 0 1 pattern 24 : 0.00 0.02 1.00

Total time taken (in seconds) 213.885

```

예측 값이 목표 값이 적당히 가까운 것을 볼 수 있습니다. 예측 값을 목표 값에 더 가깝게 하

려면 훈련의 횟수를 늘리면 됩니다.