인공지능 과제

신경망으로 3차원 구분하기

추가과제

201801733 김선우

목차

1. 가중치의 변화 시각화

```
FILE* Wh2_out = fopen("Wh2_out.txt", "w");
FILE* Wh1_h2 = fopen("Wh1_h2.txt", "w");
FILE* Win_h1 = fopen("Win_h1.txt", "w");
```

```
void Update_Weight(FILE* Wh2_out, FILE* Wh1_h2, FILE* Win_h1)
{
    // Weight hidden2 to output
    for (int i = 0; i < MAX_HIDDEN_2; i++)
    {
        for(int j = 0; j < MAX_OUTPUT; j++)
        {
            fprintf(Wh2_out, "%f ",Whidden2_output[i][j]);
            Whidden2_output[i][j] = new_Whidden2_output[i][j];
        }
    }
    fprintf(Wh2_out, "\n");

    // Weight hidden1 to hidden2
    for (int i = 0; i < MAX_HIDDEN_1; i++)
    {
        for(int j = 0; j < MAX_HIDDEN_2; j++)
        {
            fprintf(Wh1_h2, "%f ",Whidden1_hidden2[i][j]);
            Whidden1_hidden2[i][j] = new_Whidden1_hidden2[i][j];
        }
    }
    fprintf(Wh1_h2, "\n");

    // Weight input to hidden1
    for (int i = 0; i < MAX_INPUT; i++)
    {
        for(int j = 0; j < MAX_HIDDEN_1; j++)
        {
            fprintf(Win_h1, "%f ",Winput_hidden1[i][j]);
            Winput_hidden1[i][j] = new_Winput_hidden1[i][j];
        }
    }
    fprintf(Win_h1, "\n");
```

가중치에 대한 시각화를 위해 가중치 업데이트 수행 함수에서 모든 가중치를 Layer 별로 다른 파일에 저장한다.

Wh2_out.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

0.200000 0.100000 0.700000 0.300000 0.500000 0.400000 0.900000 0.500000 0.200000
0.218224 0.035888 0.638328 0.316930 0.440441 0.342707 0.917804 0.437364 0.139747
0.157595 -0.031474 0.662494 0.260415 0.377650 0.365234 0.858578 0.371560 0.163355
0.094530 0.004603 0.598415 0.202062 0.411032 0.305942 0.797207 0.406668 0.100998
0.119754 -0.061515 0.533000 0.225361 0.349959 0.245518 0.821752 0.342331 0.037344
0.054831 -0.128232 0.565610 0.165200 0.288135 0.275737 0.758566 0.277400 0.069082
-0.010921 -0.083855 0.499592 0.104681 0.328981 0.214973 0.694836 0.320412 0.005094
0.022572 -0.148229 0.434206 0.135457 0.269830 0.154891 0.727279 0.258057 -0.058242
-0.044073 -0.211797 0.475263 0.074025 0.211234 0.192737 0.662657 0.196418 -0.018430
-0.109930 -0.160586 0.409539 0.013700 0.258143 0.132533 0.599112 0.245830 -0.081847
-0.068048 -0.222542 0.345665 0.052092 0.201348 0.073980 0.639542 0.186022 -0.143508
-0.134756 -0.282666 0.394577 -0.009197 0.146110 0.118919 0.575122 0.127961 -0.096273
-0.198675 -0.226076 0.330647 -0.067631 0.197845 0.060473 0.513665 0.182372 -0.157741
-0.149397 -0.284665 0.269978 -0.022501 0.144186 0.004910 0.561091 0.125984 -0.216130
-0.213478 -0.340376 0.324753 -0.081250 0.093110 0.055127 0.499376 0.072330 -0.163378
-0.273916 -0.280036 0.263770 -0.136405 0.148175 -0.000524 0.441453 0.130157 -0.221821
-0.219618 -0.334451 0.207666 -0.086982 0.098647 -0.051591 0.493507 0.077992 -0.275606
-0.280211 -0.386344 0.266712 -0.142464 0.051131 0.002474 0.435270 0.028117 -0.218856

각 파일에는 가중치가 하나의 열로 구분되어 저장되고 가중치는 하나의 시작 노드에서 도착 노드들로 이어진 순서로 되어있다.

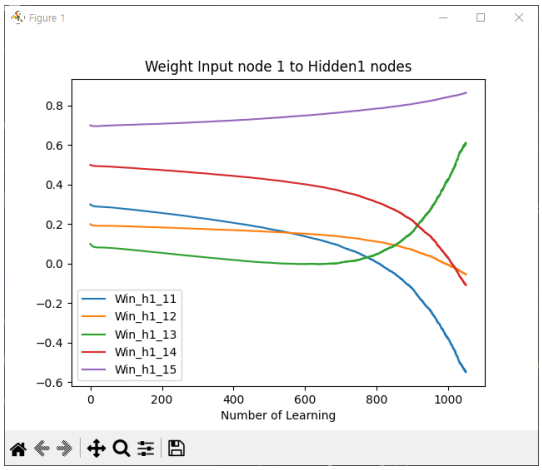
예를 들어

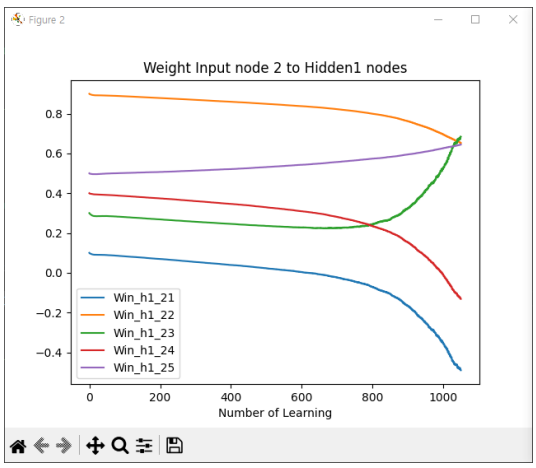
첫번째 열부터 3번째 열까지는 Hidden layer Node 1 에서 Output Node 들에 대한 가중치들

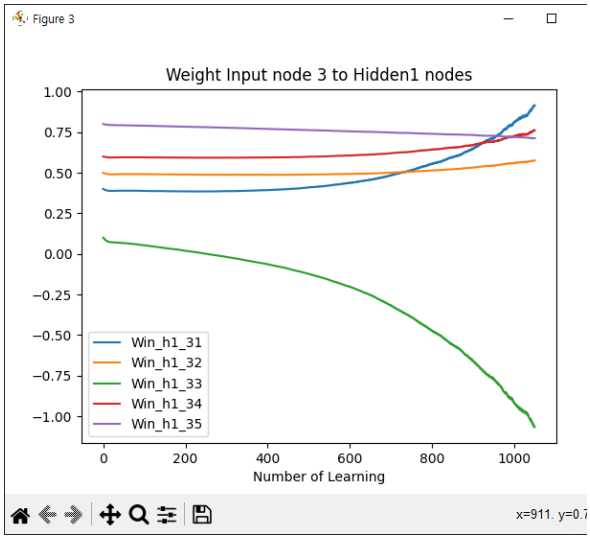
4번째 열부터 6번째 열까지는 Hidden layer Node 2 에서 Output Node 들에 대한 가중치들

7번째 열에서 마지막 열까지는 Hidden layer Node 3 에서 Output Node 들에 대한 가중치들이다.

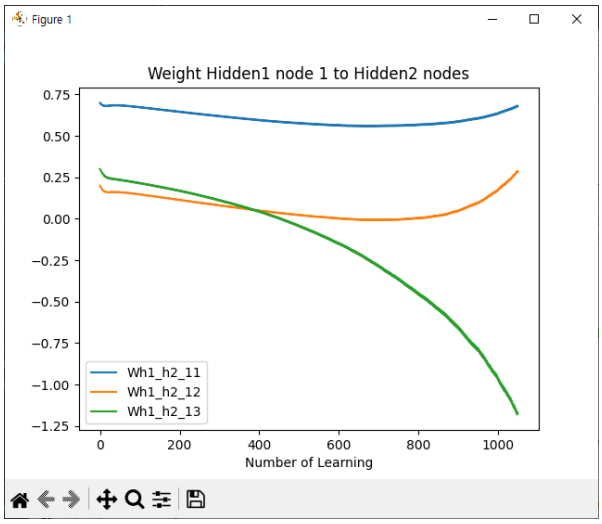
● Input 에서 Hidden layer 1 의 가중치 시각화

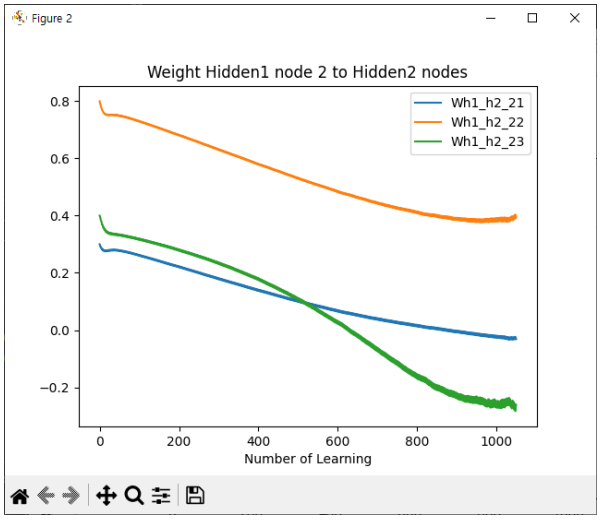


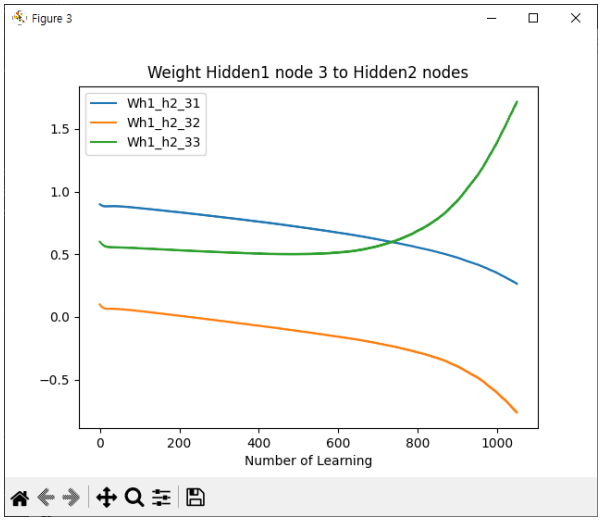


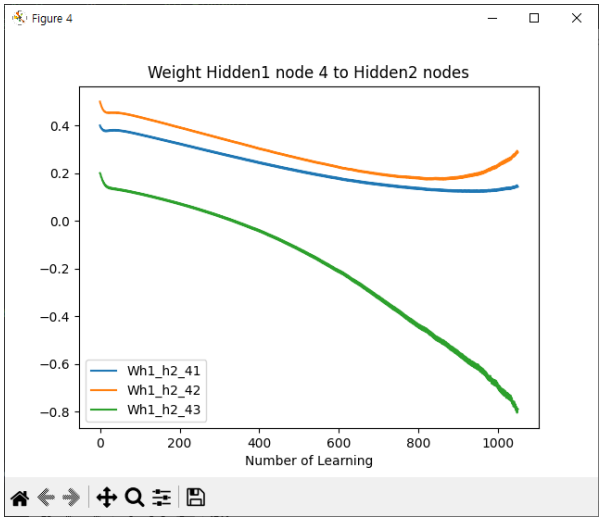


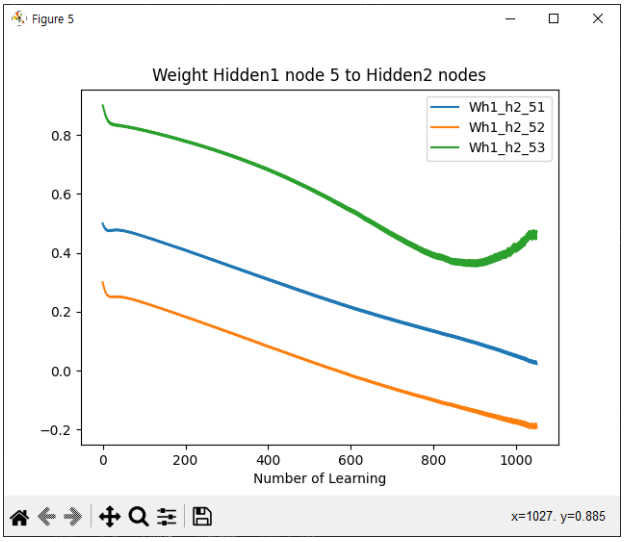
● Hidden layer 1 에서 Hidden layer 2 의 가중치 시각화



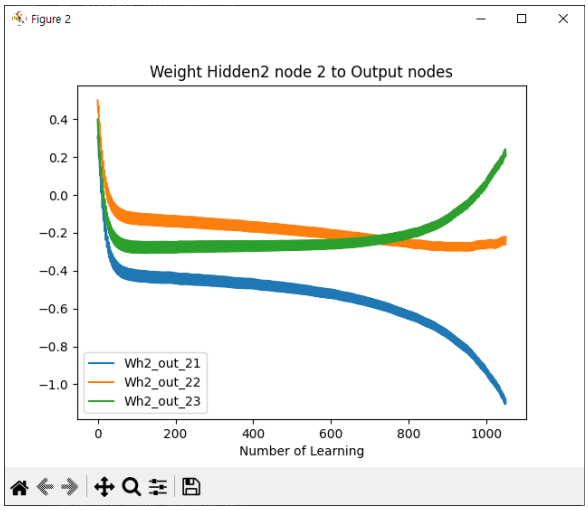


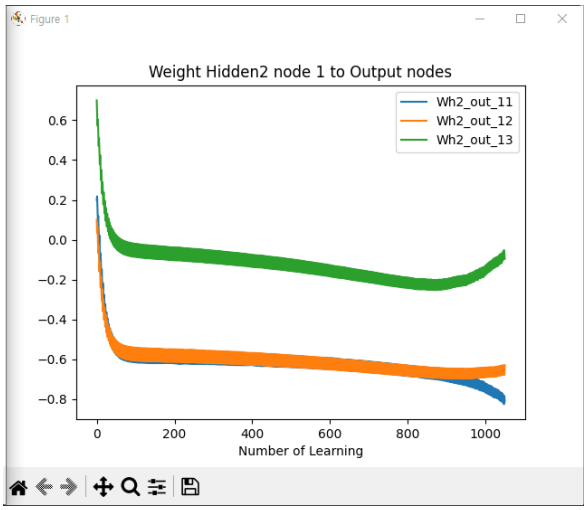


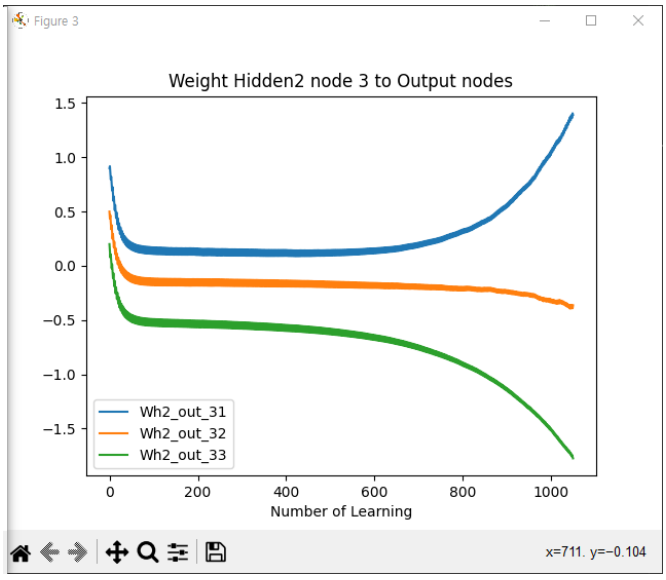




● Hidden layer 2 에서 Output 의 가중치 시각화







2. 전체 가중치 중 최종결과가 가장 영향을 많이 미치는 가중치 3개

- **가장 많은 영향을 미치는 가중치**

가중치는 노드에 들어온 입력 값에 대해 영향을 줄 수 있는 요소로 해당 노드에 대한 가중치가 높을수록 결과값에 대한 해당 노드의 영향이 크다고 볼 수 있다.

가중치가 0에 가까울수록 다음 노드로 들어가는 입력 값이 작아지기 때문에 최종 결과에 미치는 영향이 적어진다. 따라서 가중치가 클수록 최종 결과에 영향을 많이 미친다고 할 수 있다.

이때 영역 구분을 위한 target 을 (1, 0, 0) 과 같이 0 이 되도록 설정했고 입력 값은 모두 양수이므로 가중치는 음수가 될 수 있다. 음수이지만 해당 가중치의 절대값이 크다면 마찬가지로 최종 결과에 영향을 많이 미친다고 볼 수 있다.

그러므로 전체 가중치 중 최종 결과가 가장 영향을 많이 미치는 가중치를 찾기 위해 가중치의 절대값이 가장 큰 3개를 찾아보았다.

```python
# All weights save in one list, Weight 0 ~ 38
Weight = []

for i in range(1,4):
    for j in range(5):
        Weight.append(eval('Win_h1_' + str(i))[j][-1])

for i in range(1,6):
    for j in range(3):
        Weight.append(eval('Wh1_h2_' + str(i))[j][-1])

for i in range(1,4):
    for j in range(3):
        Weight.append(eval('Wh2_out_' + str(i))[j][-1])
```

```python
sort_Weight = sorted(Weight, key=abs, reverse=True)

Most_three_Weight = sort_Weight[:3]

print(Most_three_Weight)


Weight_1 = Weight.index(Most_three_Weight[0])
print(Weight_1)


Weight_2 = Weight.index(Most_three_Weight[1])
print(Weight_2)


Weight_3 = Weight.index(Most_three_Weight[2])
print(Weight_3)
```

```
[-1.773859, 1.717443, 1.391073]
38
23
36
```

Python 을 통해 가중치에 대한 파일들을 불러 5번의 학습을 마친 가중치들을 하나의 리스트에 저장한다. 그리고 절대값을 기준으로 내림차순으로 리스트를 정렬한 뒤 그 중 앞에서부터 3개의 값을 가져오게 되고 이는 5번의 학습 결과 가장 큰 가중치를 갖는 3개의 가중치가 된다.

해당 가중치를 갖는 값을 인덱스로 가져오게 되고 모든 가중치에 대해 0~38번 까지 하나의 리스트에 나열했으므로

38 : Weight Hidden layer 2 node 3 to Output node 3

23 : Weight Hidden layer 1 node 3 to Hidden layer2 node 3

36 : Weight Hidden layer 2 node 3 to Output node 1 이 된다.

● 학습 데이터의 순서에 따른 가중치 변화 살펴보기

현재 학습데이터의 순서는 학습데이터에 대한 목표가 R, G, B 가 될 수 있도록 번갈아 면서 들어가게 된다. 위에서 얻어낸 3개의 가장 큰 가중치, 즉, 최종 결과가 가장 영향을 많이 미치는 가중치들이 학습 데이터의 순서에 상관에 그대로 유지가 되는 지 확인해보는 실험을 해보았다.

```
[-1.773859, 1.717443, 1.391073]
38
23
36
```

학습데이터 순서 : R G B

```
[1.927906, 1.809938, -1.768897]
23
36
38
```

학습데이터 순서 : G R B

```
[1.614697, -1.57707, 1.387821]
23
38
36
```

학습데이터 순서 : B G R

```
[1.712425, -1.570787, 1.499848]
23
38
36
```

학습데이터 순서 : R G B

```
[1.898862, -1.891294, 1.462969]
23
38
36
```

학습데이터 순서 : RRR BBB GGG

학습데이터가 학습되는 순서를 바꾸어도 가중치의 값이 바뀌긴 하나 학습데이터의 순서에 상관없이 가장 큰 절대값을 가지는 가중치 3개는 유지되는 것을 알 수 있다.

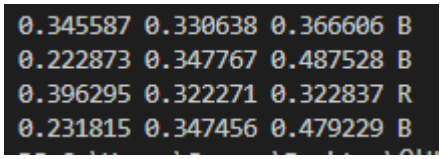
● 학습 데이터를 다르게 바꾸어 실행했을 때

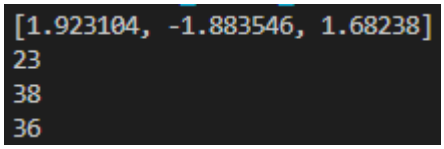
위의 실험에서 학습 데이터 값은 그대로 유지한 채 학습이 되는 데이터의 순서를 바꾸어 보았다.

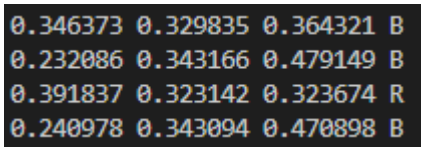
이번 실험에서는 학습 데이터의 값들을 바꿔가며 실행을 해보았다.

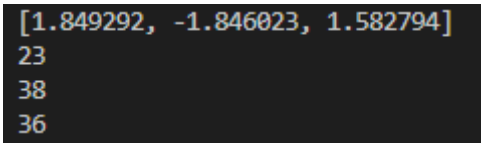
```
srand(time(NULL));
```

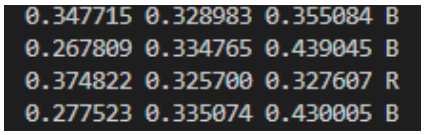
기존 코드에서는 rand() 를 사용해 학습데이터를 랜덤으로 주었지만 실행을 계속해도 생성된 랜덤 값이 그대로 들어가기 때문에 학습데이터의 값은 그대로였다. 이를 위해 위의 코드로 실행 시 마다 다른 랜덤 값들이 학습 데이터로 사용되도록 하였다.

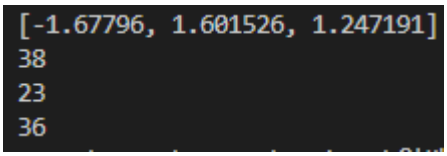












오른쪽 사진들은 총 3번의 실행을 통해 랜덤한 학습데이터에 대해 5번의 학습을 거치고 얻어낸 출력 값들이고 왼쪽 사진들은 그에 따른 절대값이 가장 큰 3개의 가중치를 출력한 것이다.

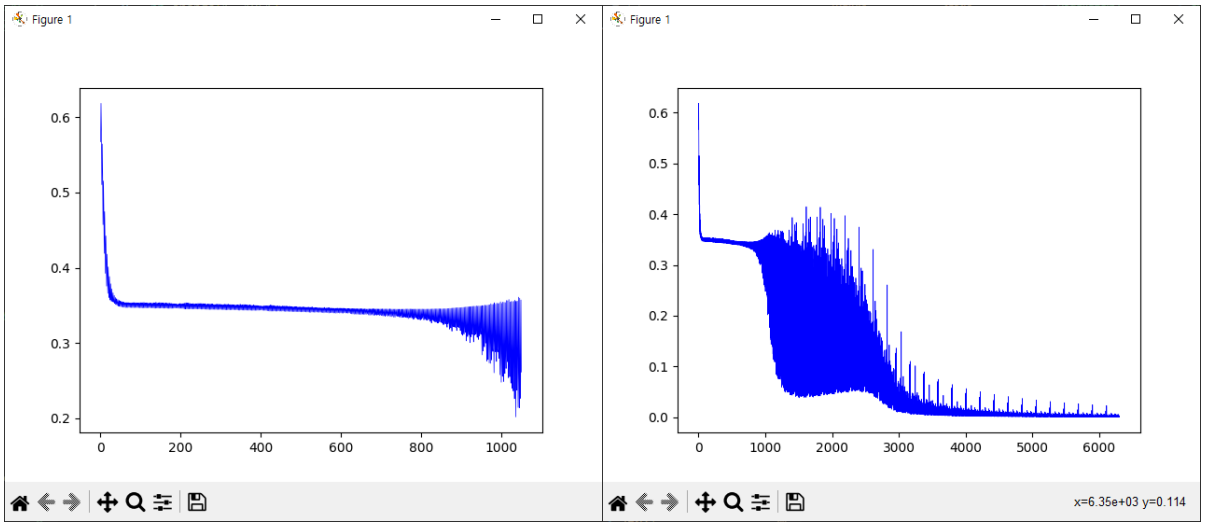
학습데이터의 값들이 달라짐에 따라 출력 값과 가중치의 값들이 달라지는 것을 알 수 있지만 절대값이 가장 큰 가중치 3개는 순위가 달라질 뿐 비슷하게 유지되는 것을 알 수 있다.

- 2개의 실험의 결과, 초기 가중치가 그대로 유지되었을 때 학습 데이터의 값과 순서에 상관없이 절대값이 가장 큰 3개의 가중치는 비슷하게 유지됨을 확인할 수 있다.
  이에 따라 가장 영향을 미치는 가중치 3개인 Wh2_out 3,3 [38] Wh2_out 3,1 [36] Wh1_h2 3,3 [23] 을 얻어낼 수 있었다.

3. 테스트 데이터를 가장 확실하게 분류할 수 있는 반복횟수 실험하기

테스트 데이터가 목표에 맞게 잘 분류된다는 것은 신경망 학습을 통해 나온 출력값이 목표값과 가지는 오차가 거의 없다는 뜻이 된다. 따라서 5번 이상의 임의 학습을 진행하여 1/2 제곱오차 함수가 언제부터 일정 값에 수렴하는지 확인해보았다.

- **오차 함수 그래프 분석**



왼쪽은 5 번의 학습을 오른쪽은 30 번의 학습을 진행했을 때 얻어낼 수 있는 오차함수 그래프이다.

5 번의 학습을 통해서 처음 몇번의 학습 동안에는 오차가 급격하게 줄어들지만 오차함수의 값이 0.2~0.4 로 아직 제대로 테스트 케이스를 구분하지 못한다는 것을 알 수 있다.

30 번의 학습을 통해서는 5 번의 학습 이후에 오차함수는 0.05 ~ 0.35 사이에서 수렴하지 못한 채 변화하고 있는 모습을 보여주지만 약 3000 번 즉, 210 x 15, 15 번의 학습 이후로는 어느 정도 0 을 향해 수렴해가는 모습을 보여준다.

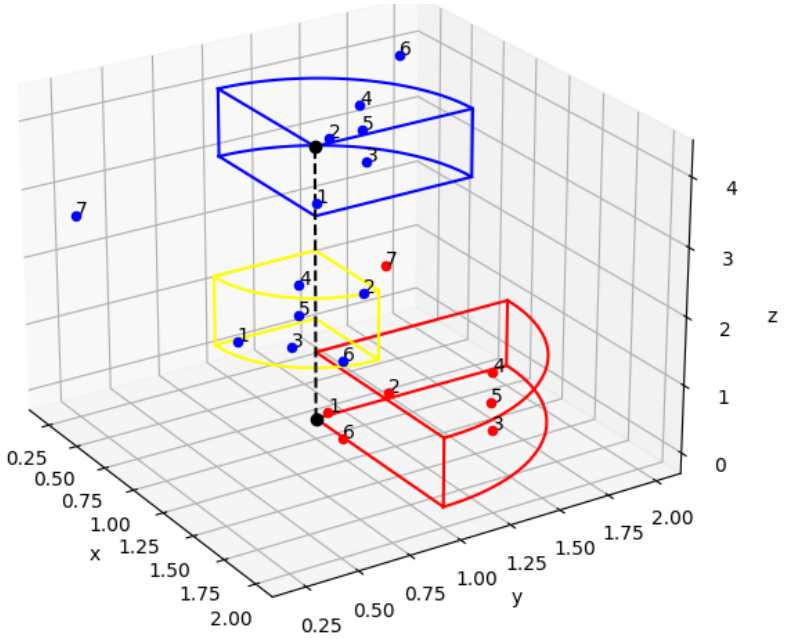
오차함수가 0 에 가까워진다는 것은 각 출력 값들에 대한 오차가 0 에 수렴한다는 것이고

이는 데이터를 목표에 맞게 잘 분류한다는 뜻이 된다.

이 결과를 실제로 학습데이터를 늘려가며 실험해보았다.

● 학습 횟수를 증가

- 학습 횟수 5 번



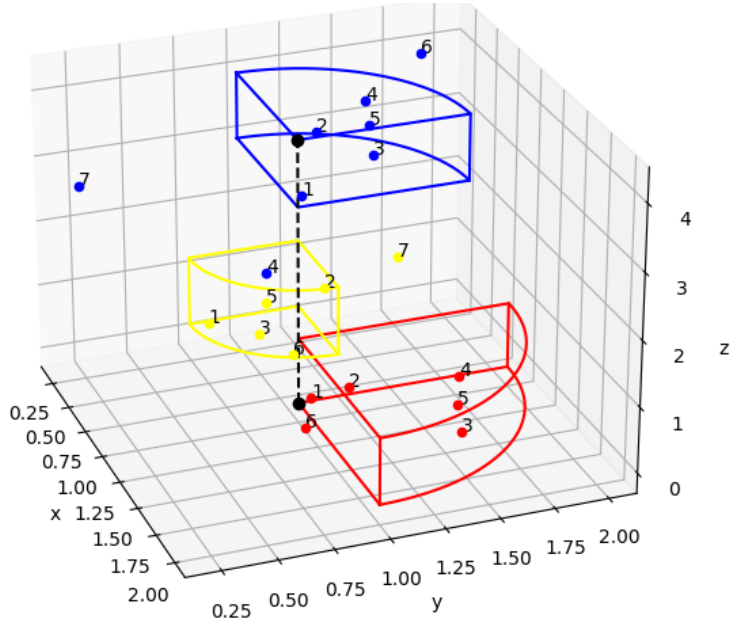

```
0.490513 0.316689 0.242734 R
0.330877 0.330886 0.377230 B
0.264798 0.335643 0.446301 B
0.434241 0.320548 0.283217 R
0.298549 0.333241 0.409594 B
0.250158 0.336173 0.462311 B
0.523782 0.314215 0.220351 R
0.347779 0.329029 0.360033 B
0.267511 0.335332 0.442980 B
0.456919 0.318130 0.264903 R
0.288532 0.334217 0.420692 B
0.250452 0.336116 0.461870 B
0.492041 0.315739 0.240322 R
0.312806 0.332222 0.395051 B
0.256516 0.335904 0.455167 B
0.482866 0.317140 0.247945 R
0.343080 0.329335 0.364323 B
0.248097 0.336124 0.464355 B
0.377381 0.325363 0.330695 R
0.257376 0.336442 0.455523 B
```

학습 횟수를 5 번으로 설정해 테스트 케이스를 입력해 본 경우 구역에 대한 구분을 제대로 하지 못하고 있는 것을 알 수 있다.

출력 값들 또한 특정 구역으로 구분하기에는 충분하지 않은 값으로

아직 학습 횟수가 부족해 제대로 된 학습을 하지 못한 상태이므로 횟수를 늘려서 테스트 케이스를 잘 분류할 수 있도록 해야한다.

- 학습 횟수 10 번



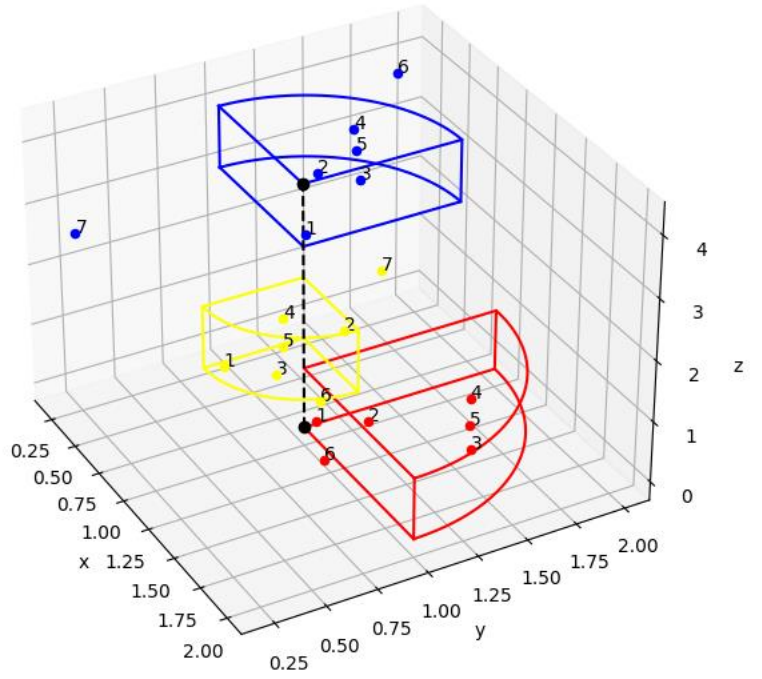

```
0.844268 0.293949 0.009570 R
0.076780 0.352985 0.239405 Y
0.008703 0.379887 0.740289 B
0.740424 0.301632 0.014448 R
0.044712 0.367177 0.361285 Y
0.890249 0.295607 0.007680 R
0.165060 0.339626 0.119314 Y
0.007838 0.379006 0.757030 B
0.813339 0.298525 0.010933 R
0.023280 0.374216 0.524263 B
0.005678 0.377920 0.805478 B
0.866266 0.296176 0.008668 R
0.054295 0.361544 0.313822 Y
0.006294 0.378573 0.791327 B
0.846586 0.295502 0.009532 R
0.210843 0.338669 0.094555 Y
0.005293 0.377153 0.814306 B
0.217710 0.325399 0.082880 Y
0.005606 0.377211 0.807847 B
```

학습을 횟수를 10 번으로 늘린 경우 이전 보다는 Y 영역에 대한 구분이 훨씬 더 잘 되어 보이지만 여전히 4 번 테스트 케이스 와 Y 영역에 가까운데도 B 영역으로 구분되는 것처럼 학습이 제대로 되지 않은 모습을 볼 수 있다.

출력 결과들이 이전과 비교해 목표에 가까워진 모습이지만 아직까지도 0.52 를 통해 B 로 분류하는 등 제대로 된 영역을 구분을 하기에는 부족하다.

여전히 학습이 제대로 되지 않았으므로 학습 횟수를 증가시켜야 한다.

- 학습 횟수 15 번




```
0.892626 0.087122 0.003138 R
0.087731 0.819727 0.050550 Y
0.002997 0.290028 0.796124 B
0.805565 0.319137 0.004090 R
0.082540 0.855971 0.055680 Y
0.001245 0.147865 0.917347 B
0.905197 0.079102 0.002925 R
0.138042 0.821715 0.029390 Y
0.001238 0.142378 0.916496 B
0.884676 0.179474 0.003116 R
0.048050 0.814336 0.108359 Y
0.001025 0.124404 0.932944 B
0.904759 0.105431 0.002891 R
0.080579 0.840437 0.056791 Y
0.001081 0.129251 0.928656 B
0.898064 0.115502 0.003004 R
0.161649 0.850363 0.024802 Y
0.000964 0.118392 0.937415 B
0.035568 0.225368 0.098643 Y
0.001063 0.129015 0.930411 B
```

학습 횟수를 15 번으로 늘린 경우 이전 잘 분류되지 않던 4 번 테스트 케이스가 Y 영역으로 잘 구분되는 것을 알 수 있다.

실제 출력 결과들 0.9, 0.8 에 가까운 값들이 대부분이며 이전보다 확실히 목표에 가까워진 것을 알 수 있다.

기존 5 번의 학습으로는 제대로 된 출력 값을 갖지 못하고 테스트 케이스에 대한 영역 구분 또한 수행하지 못한다.

15 번 정도의 학습을 통해 테스트 케이스 20 개를 제대로 구분할 수 있게 된다.