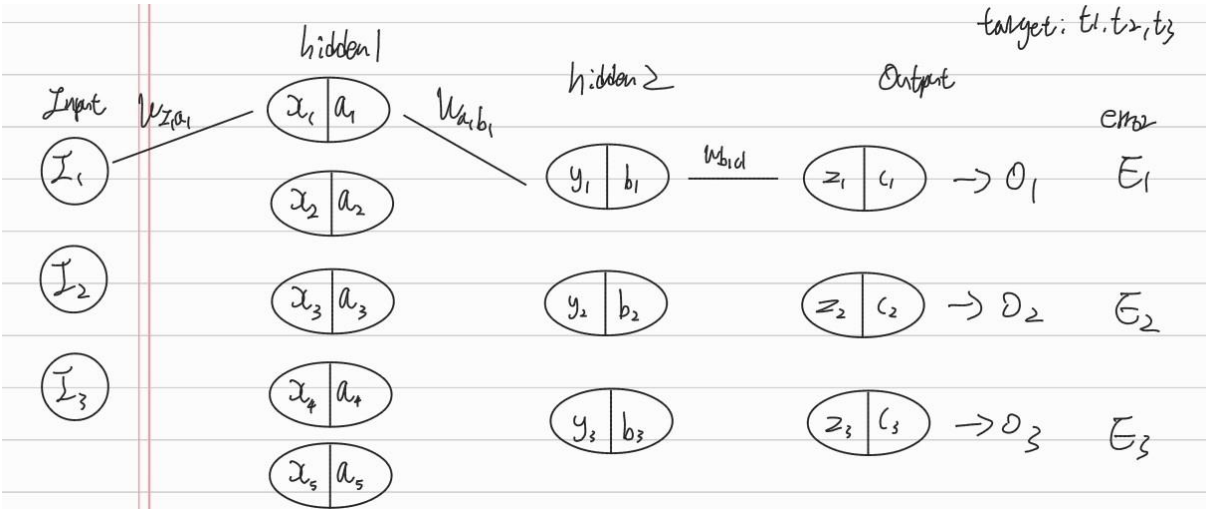
인공지능 과제

신경망으로 3차원 공간 구분하기

201801733 김선우

목차

1 .신경망 구현하기



구현하고자 하는 신경망은 은닉층 2개를 포함한 3 X 5 X 3 X 3 형태이다.

이를 코드로 구현하기 위해 입력값, 가중치, 활성화 함수 입력, 활성화 함수 출력, 출력값,

에러함수를 변수로 정의하고 이를 수식으로 나타내며 코드를 구현하였다.

● 순전파

```
void Forward_Propagation(int _idx,double _input[])
{
    // input to hidden_1
    for(int i = 0; i < MAX_HIDDEN_1; i++)
    {
        for(int j = 0; j < MAX_INPUT; j++)
        {
            Xhidden_1[i] += _input[j] * Winput_hidden1[j][i];
        }
        Ohidden_1[i] = 1 / (1 + exp(-Xhidden_1[i]));
    }

    // hidden_1 to hidden_2
    for(int i = 0; i < MAX_HIDDEN_2; i++)
    {
        for(int j = 0; j < MAX_HIDDEN_1; j++)
        {
            Xhidden_2[i] += Ohidden_1[j] * Whidden1_hidden2[j][i];
        }
        Ohidden_2[i] = 1 / (1 + exp(-Xhidden_2[i]));
    }
```

```
    // hidden_2 to output
    for(int i = 0; i < MAX_OUTPUT; i++)
    {
        for(int j = 0; j < MAX_HIDDEN_2; j++)
        {
            Xoutput[i] += Ohidden_2[j] * Whidden2_output[j][i];
        }
        Ooutput[i] = 1 / (1 + exp(-Xoutput[i]));
        output[_idx][i] = Ooutput[i];
    }
}
```

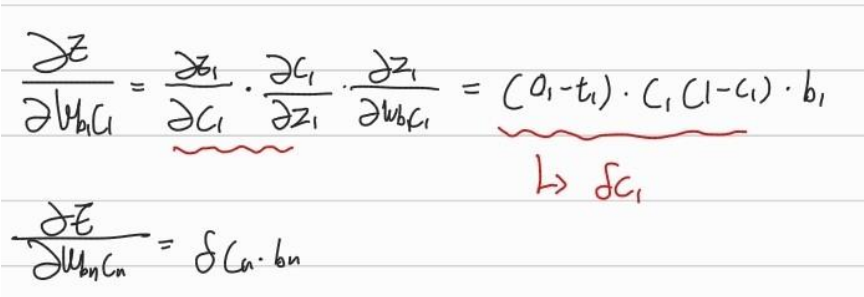
신경망의 순전파 과정으로 각 노드의 입력값과 가중치를 이용해 활성화 함수를 거치면서

출력값을 얻어내고 이후 목표값과 출력값을 비교하여 오차를 구하고 이를 통해 역전파를

진행하게 된다.

● 역전파

• 출력에서 은닉층 2



```
void Back_Propagation(double _input[], double _target[])
{
    // output to hidden_2
    // a_Ooutput[i] = (Ooutput[i] - target[i]) * Oouput[i] * (1 - Ooutput[i])
    for(int i = 0; i < MAX_HIDDEN_2; i++)
    {
        for(int j = 0; j < MAX_OUTPUT; j++)
        {
            a_Ooutput[j] = (Ooutput[j] - _target[j]) * Ooutput[j] * (1 - Ooutput[j]);
            new_Whidden2_output[i][j] = Whidden2_output[i][j] - LEARNING_RATE * a_Ooutput[j] * Ohidden_2[i];
        }

    }
```

오차 함수를 $E_n = \frac{1}{2}(T_n - O_n)^2$ 로 정의하고 이를 가중치에 대해서 편민준을 시행한다.

은닉층 2와 출력층 사이의 가중치에 대해 영향을 받는 요소들로 Chian rule 을 이용해 식을 전개하면 출력에서 은닉층 2에 대한 공식을 얻어 낼 수 있고 이를 코드로 구현한다.

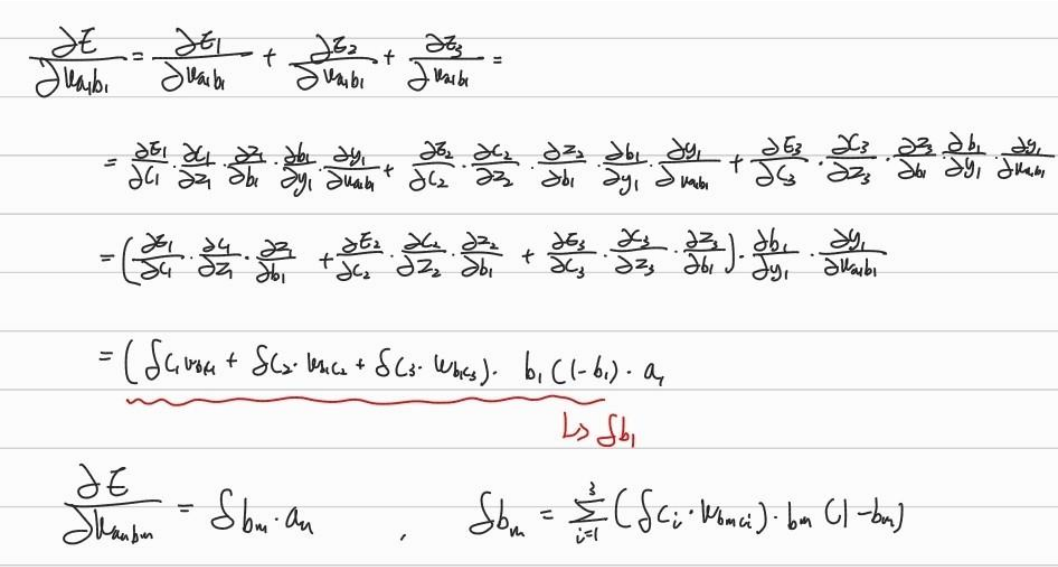
이때 역전파 과정에서 가중치에 대해 영향을 받는 요소는 입력층으로 갈수록 많아지고

지금 수행하는 편미분은 가중치에 대해 영향을 받는 요소들에 대해 미분을 수행하므로

이전 층에서 사용한 공식은 입력층으로 향하면서 그대로 사용되기 때문에 반복되는 공식을

치환해서 δc_n 로 저장한다.

- **은닉층 2에서 은닉층 1**



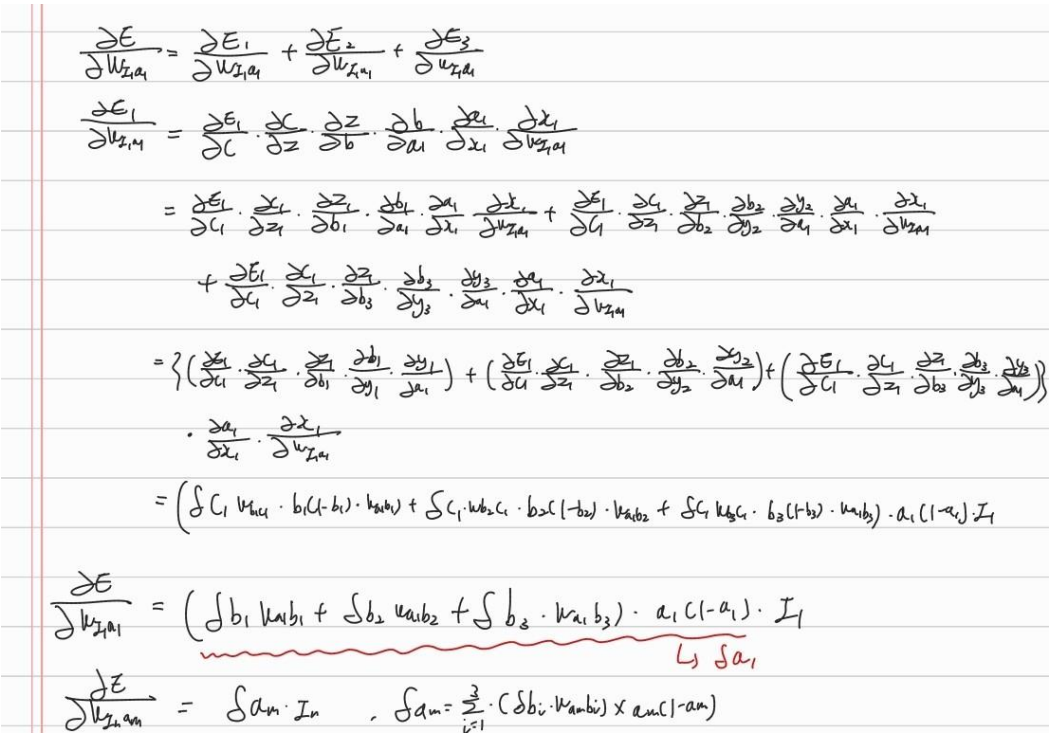
```
// hidden_2 to hidden_1
// a_Ohidden_2[i] = 시그마j (cj * Whidden2_output[i][j]) * Ohidden_2[i] * (1 - Ohidden_2[i])
for (int i = 0; i < MAX_HIDDEN_1; i++)
{
    for(int j = 0; j < MAX_HIDDEN_2; j++)
    {
        double temp = 0;
        for(int k = 0; k < MAX_OUTPUT; k++)
        {
            temp += a_Ooutput[k] * Whidden2_output[j][k];
        }
        a_Ohidden_2[j] = temp * Ohidden_2[j] * (1 - Ohidden_2[j]);
        new_Whidden1_hidden2[i][j] = Whidden1_hidden2[i][j] - LEARNING_RATE * a_Ohidden_2[j] * Ohidden_1[i];
    }
}
```

마찬가지로 은닉층 1과 은닉층 2 사이의 가중치에 대한 편비분을 수행하는 데 이 과정에서 이 가중치들은 이전 역전파 과정에서 chain rule 로 사용된 요소들이 그대로 사용된다.

그러므로 이전 역전파 과정에서 선언한 δc_n 이 사용되고 나머지 편미분을 수행해 오차 함수에 대한 은닉층 1과 은닉층 2 사이의 가중치 변화율 또한 구할 수 있다.

이 과정에서 또한 반복되는 공식을 δb_n 으로 치환할 수 있다.

- **은닉층 1에서 입력층**

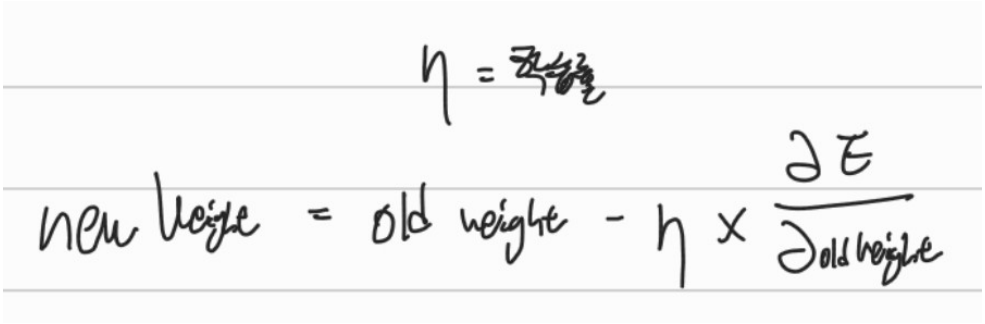


```
// hidden_1 to input
// a_Ohidden_1[i] = 시그마j (a_Ohidden_2[j] * Winput_hidden1[i][j]) * Ohidden_1[i] * (1 - Ohidden_1[i])
for (int i = 0; i < MAX_INPUT; i++)
{
    for(int j = 0; j < MAX_HIDDEN_1; j++)
    {
        double temp = 0;
        for(int k = 0; k < MAX_HIDDEN_2; k++)
        {
            temp += a_Ohidden_2[k] * Whidden1_hidden2[j][k];
        }
        a_Ohidden_1[j] = temp * Ohidden_1[j] * (1 - Ohidden_1[j]);
        new_Winput_hidden1[i][j] = Winput_hidden1[i][j] - LEARNING_RATE * a_Ohidden_1[j] * _input[i];
    }
}
Update_Weight();
}
```

앞선 과정과 마찬가지로 입력층과 은닉층 1 사이의 가중치에 영향을 받는 요소들을 chain rule을 통해 편미분을 수행한다. δb_n 이 공식에 사용되며 이를 제외한 나머지 편미분을 수행하여 또다른 공식을 얻어낼 수 있다.

출력층부터 입력층까지 넘어가며 역전파를 수행하였고 이를 통해 오차함수에 대한 각 가중치들의 변화율을 얻어냈다. 이제 이를 이용해 새로운 가중치로 업데이트를 한다.

● 가중치 업데이트




```
void Update_Weight()
{
    for (int i = 0; i < MAX_HIDDEN_2; i++)
    {
        for(int j = 0; j < MAX_OUTPUT; j++)
        {
            Whidden2_output[i][j] = new_Whidden2_output[i][j];
        }
    }

    for (int i = 0; i < MAX_HIDDEN_1; i++)
    {
        for(int j = 0; j < MAX_HIDDEN_2; j++)
        {
            Whidden1_hidden2[i][j] = new_Whidden1_hidden2[i][j];

        }
    }

    for (int i = 0; i < MAX_INPUT; i++)
    {
        for(int j = 0; j < MAX_HIDDEN_1; j++)
        {
            Winput_hidden1[i][j] = new_Winput_hidden1[i][j];

        }
    }
}
```

이전 가중치에서 학습률과 오차함수에 대한 가중치 편비분 값을 빼면 새로운 가중치를 얻을 수 있다. 또한 이에 대한 수식 수행은 이전 역전파 과정 구현에서 저장하였고 모든 가중치에 대한 편비분 값을 구한 마지막에 모든 가중치를 업데이트한다.

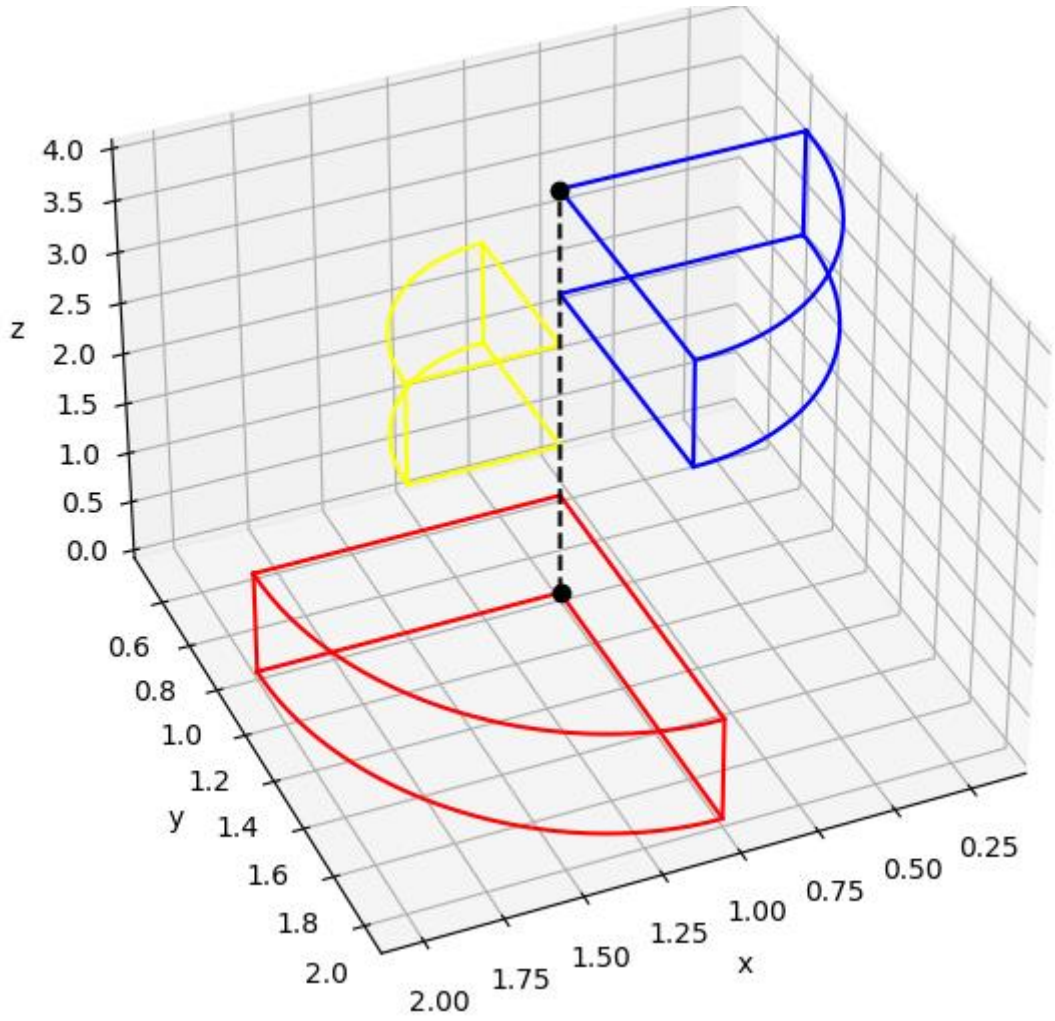
2. 3차원 영역 구분과 학습데이터 만들기

순전파와 역전파를 이용해 학습을 할 수 있는 신경망을 구현하였다.

이제 각각 빨강, 노랑, 파랑을 나타내는 구역을 3차원 공간에서 설계하여 구분하고

각 구역을 나타내는 x, y, z 좌표를 학습데이터로 만든다.

● 구분되는 3개의 영역 설계



부채꼴 모양을 면으로 갖는 도형으로 3개의 영역을 구분한다. 면에 해당하는 부채꼴의 각도는

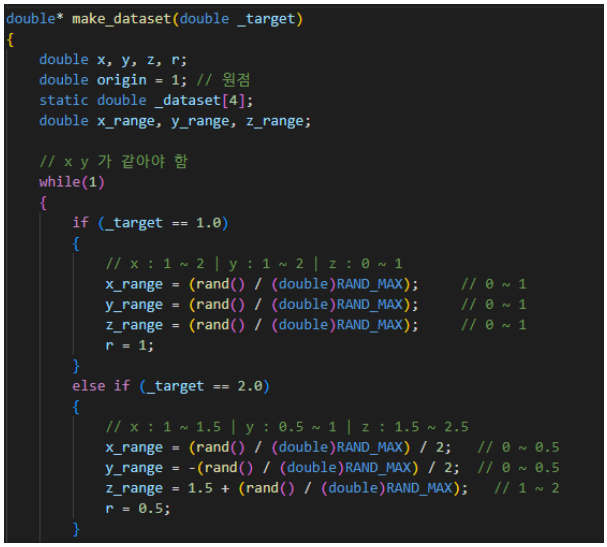
3개의 영역 모두 90도 이고 모든 원점은 xyz 좌표에서 (1,1,0) 이다.

다만 이들의 반지름은 빨강 영역은 1, 노랑 영역은 0.5, 파랑 영역은 0.8로 설정한다.

● 각 영역에 해당하는 학습데이터 만들기

3개의 영역 빨강, 노랑, 파랑에 해당하는 x, y, z 좌표를 만들고 이를 입력값으로 해당 좌표가 가리키는 영역을 목표값으로 사용해 학습데이터를 만든다.

빨강에 해당하는 좌표 70개, 노랑에 해당하는 좌표 70개 파랑에 해당하는 좌표 70개로 총 210개의 데이터를 만들게 되는데 이 모든 데이터는 범위를 갖는 랜덤 실수를 갖게 한다.



```
double* make_dataset(double _target)
{
    double x, y, z, r;
    double origin = 1; // 원점
    static double _dataset[4];
    double x_range, y_range, z_range;

    // x y 가 같아야 함
    while(1)
    {
        if (_target == 1.0)
        {
            // x : 1 ~ 2 | y : 1 ~ 2 | z : 0 ~ 1
            x_range = (rand() / (double)RAND_MAX);      // 0 ~ 1
            y_range = (rand() / (double)RAND_MAX);      // 0 ~ 1
            z_range = (rand() / (double)RAND_MAX);      // 0 ~ 1
            r = 1;
        }
        else if (_target == 2.0)
        {
            // x : 1 ~ 1.5 | y : 0.5 ~ 1 | z : 1.5 ~ 2.5
            x_range = (rand() / (double)RAND_MAX) / 2;   // 0 ~ 0.5
            y_range = -(rand() / (double)RAND_MAX) / 2;  // 0 ~ 0.5
            z_range = 1.5 + (rand() / (double)RAND_MAX);   // 1 ~ 2
            r = 0.5;
        }
```

```
        else if (_target == 3.0)
        {
            // x : 0.2 ~ 1 | y : 1 ~ 1.8 | z : 3 ~ 4
            x_range = -(rand() / (double)RAND_MAX) * (0.8);     // -1 ~ 0
            y_range = (rand() / (double)RAND_MAX) * (0.8);      // 0 ~ 1
            z_range = 3 + (rand() / (double)RAND_MAX);  // 2.0 ~ 3.0
            r = 0.8;
        }

        x = origin + x_range;
        y = origin + y_range;
        z = z_range;

        // 부채꼴 반지름과 (x,y) 거리
        if (((x - origin) * (x - origin)) + ((y - origin) * (y - origin)) <= (r * r))
        {
            _dataset[0] = x;
            _dataset[1] = y;
            _dataset[2] = z;
            _dataset[3] = _target;

            break;
        }
    }
    return _dataset;
}
```

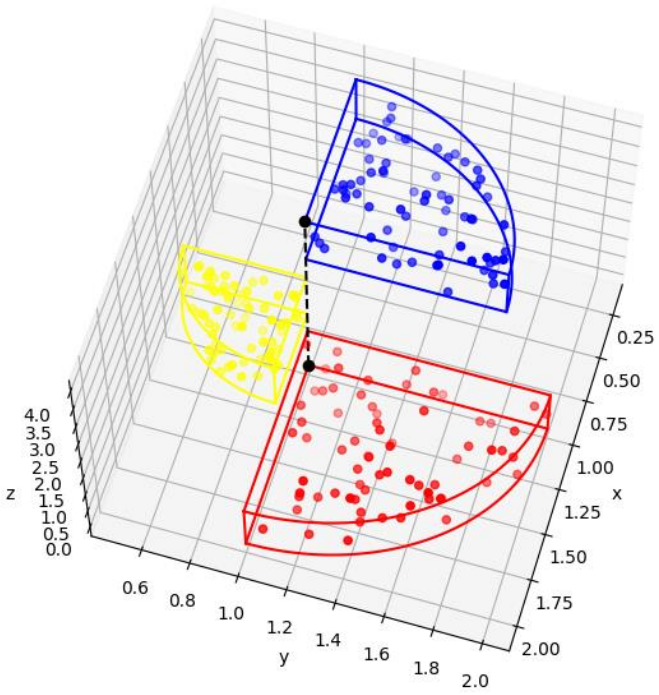
모든 영역에서 x,y 좌표 기준 밑면과 윗면은 평행하기 때문에 각 영역의 z 좌표 범위 안에서 부채꼴 안에 좌표가 존재한다면 해당 좌표는 그 영역 안에 있다고 할 수 있다.

그러므로 학습데이터를 만들 때 함수의 인자로 목표로 하는 영역을 받고 해당 영역의 x, y, z 범위 내에 좌표가 존재하는 실수형 난수를 만들어 반환해주는데 이때 부채꼴의 원점과 x,y 좌표의 거리가 부채꼴의 반지름 r 보다 작을 경우만 좌표가 부채꼴 안에 존재한다고 볼 수 있으므로 이를 조건으로 사용해 목표로 하는 영역에 맞는 좌표를 만들어준다.

```
for(int i = 0; i < LEARNING_DATA; i++)
{
    double* temp = make_dataset(_target);
    input[i][0] = temp[0];
    input[i][1] = temp[1];
    input[i][2] = temp[2];

    if (temp[3] == 1.0)
    {
        target[i][0] = 1;
        target[i][1] = 0;
        target[i][2] = 0;
        _target = 2.0;
    }
    if (temp[3] == 2.0)
    {
        target[i][0] = 0;
        target[i][1] = 1;
        target[i][2] = 0;
        _target = 3.0;
    }
    if (temp[3] == 3.0)
    {
        target[i][0] = 0;
        target[i][1] = 0;
        target[i][2] = 1;
        _target = 1.0;
    }
```

이제 main 함수에서 빨강, 노랑, 파랑 순서를 반복하는 210개의 학습데이터를 만들고 만든 데이터의 x,y,z 좌표를 입력값으로 해당하는 영역을 목표값으로 설정한다.



만든 학습데이터를 좌표로 찍어서 확인하면 각 영역안에 잘 존재하는 것을 확인할 수 있다.

3. 신경망 학습과 테스트 케이스 적용

신경망과 학습데이터를 모두 만들었으니 이제 학습데이터를 통해 신경망을 학습하고

학습이 잘 되었는지 테스트 케이스를 통해 확인한다.

● 신경망 학습

```
for(int k = 0; k < 5; k++)
{
    for (int i = 0; i < LEARNING_DATA; i++)
    {
        Forward_Propagation(i, input[i]);

        Back_Propagation(input[i], target[i]);

        init_layer();

    }
}
```

이전에 만든 순전파 함수와 역전파를 이용해 설정한 학습데이터를 학습하게 된다.

210 개의 데이터를 하나씩 순전파 한번, 역전파 한번 하는 과정을 5번 거쳐 학습한다.

```
-0.274214 -0.343902 0.573990
```

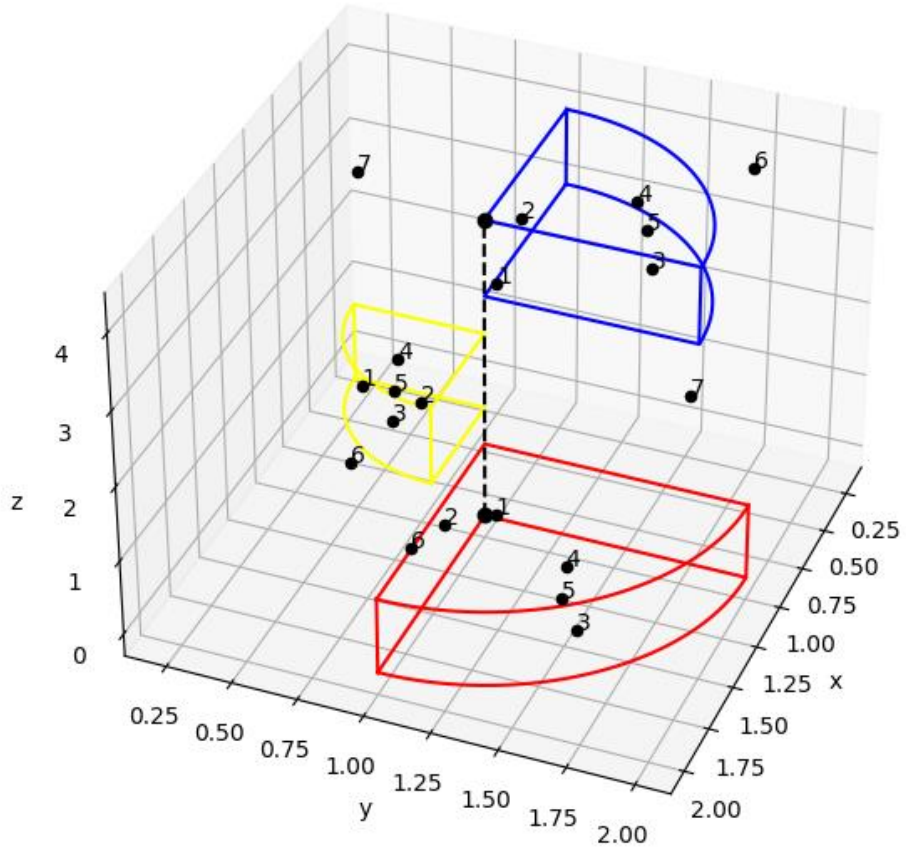
5번의 학습을 마치고 마지막 입력에 대한 오차를 출력한 결과이다.

영역에 해당하는 좌표를 입력으로 줬음에도 오차가 아직 크게 나오는 걸로 보아 학습이 아직

덜 된 것으로 볼 수 있다.

이에 대한 자세한 내용은 추후 오차함수를 확인하여 학습의 진행상황을 살펴보도록 한다.

● 테스트 케이스 설계



학습한 신경망을 확인하기 위해 테스트 케이스 20개를 만들게 된다.

여러 개의 테스트를 통해 확인해보기 위해

1번 인덱스에 해당하는 점은 x 축에 가깝게

2번 인덱스에 해당하는 점은 y 축에 가깝게

3번 인덱스에 해당하는 점은 x, y 축은 구역 기준 중앙에 가깝게, z 는 구역 기준 바닥에 가깝게

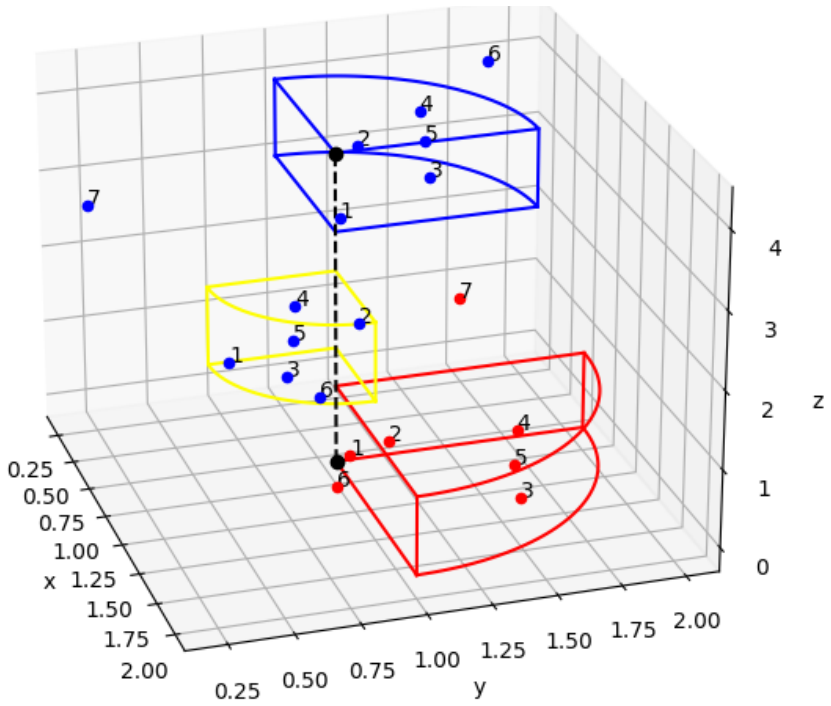
4번 인덱스에 해당하는 점은 x, y 축은 구역 기준 중앙에 가깝게, z 는 구역 기준 위쪽 가깝게

5번 인덱스에 해당하는 점은 x, y, z 축 모두 구역 기준 중앙에 가깝게

6번 인덱스부터는 구역에 벗어나거나 구역 사이에 빈 공간에 존재하는 점이다.

4. 학습 결과

```
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
0.482866 0.317140 0.247945 R
0.248097 0.336124 0.464355 B
0.377381 0.325363 0.330695 R
0.257376 0.336442 0.455523 B
```



테스트 케이스를 모두 수행해 나온 결과로 학습의 결과를 확인하기 위해 출력값과

별도의 범위나 제한을 주지 않고 가장 큰 값을 가진 영역에 해당하도록 시각화하였다.

학습의 결과가 어떠한 영역이라고 특정 지기에는 너무 작은 값을 가지고 있고

노랑 영역에 대한 구분은 아예 수행하지 못하는 모습을 보여준다.

5번의 학습을 통해서는 좌표를 통한 3개의 영역 구분은 되지 않으며 결과값 또한

목표가 1 이므로 0.4, 0.3 과 같은 값들은 영역을 구분하기에는 불확실한 값으로 볼 수 있다.

결국 5번의 학습으로는 충분한 학습을 수행하지 못했기 때문에 테스트 케이스가 제대로 된

결과를 도출하지 못했다.

이후 오차함수를 확인하고 학습 횟수를 증가시켜 바뀌는 학습의 진행상황 확인해보겠다.