# Perceptron Learning 구현 (n-input AND, OR, XOR gate)

컴퓨터과학부 2018920031 유승리 | 인공지능 | 과제 #2

## 1. 프로그램 설명

**> 개요**

Initialize weights with small random values
repeat
  for each training data $(x_i, t_i)$
    calculate output $net_j = \sum_{i=0}^{n} x_i w_i$ , $O_i = f(net_i)$
    $\Delta w_k = c(t_i - O_i) f'(net_j) x_k$
    adjust weight $w_k \leftarrow w_k + \Delta w_k$
until satisfied

Perceptron Traing Algorithm | 인공지능 강의자료 #2

이 프로그램은 위의 perceptron training algorithm을 이용하여 사용자가 설정한 학습 데이터에 근접하도록 스스로 학습하는 perceptron을 구현한 프로그램이다. 보고서에서는 input의 개수를 2로 설정하였으며, 학습 데이터로는 AND, OR, XOR gate의 값을 설정하였다.

앞서 C로 구현하여 제출했던 과제 #1과 다르게 이번 과제 #2에서는 C++을 사용하였는데, 가장 큰 이유는 class를 이용하여 Neuron 객체를 생성하기 위함이다. 또한 설정한 학습 데이터의 크기에 따라 동적으로 크기가 설정되는 vector 타입을 사용하기 위해서이기도 하다. 이를 통해 weight 및 학습 데이터의 입/출력 및 연산을 편리하게 구현할 수 있었다.

**> 소스코드 및 구조**

① 기본 설정

```
#define INPUT_SIZE 2            // input의 개수 = 2
#define EPOCH_MAX 100000        // 학습 횟수
#define LEARNING_RATE 0.1       // 학습률
```

- input의 개수는 2로 고정했으며, 여러 횟수의 실험을 통해 EPOCH_MAX(학습 횟수)는 100000으로, LEARNING_RATE(학습률)은 0.1로 설정하였다.

- LEARNING_RATE의 경우, 0.1, 0.25, 0.5로 바꾸어 실험한 결과 그 크기가 커질수록 error 값은 작아졌으나 학습 결과가 너무 급격하게 바뀌었기 때문에, 학습 과정에 따른 변화를 시각적으로 보기 위해서 0.1로 선택하였다.

```
// 학습 데이터 : { {input 조합}, target } 형식
// AND gate 학습 데이터 설정
vector<pair<vector<double>, double>> target_AND = {
        { { 0, 0 }, 0 },
{ { 0, 1 }, 0 },
{ { 1, 0 }, 0 },
{ { 1, 1 }, 1 }
};
// OR gate 학습 데이터 설정
vector<pair<vector<double>, double>> target_OR = {
        { { 0, 0 }, 0 },
{ { 0, 1 }, 1 },
{ { 1, 0 }, 1 },
{ { 1, 1 }, 1 }
};
// XOR gate 학습 데이터 설정
vector<pair<vector<double>, double>> target_XOR = {
        { { 0, 0 }, 0 },
{ { 0, 1 }, 1 },
{ { 1, 0 }, 1 },
{ { 1, 1 }, 0 }
};
```

- 학습 데이터는 `{ {input 조합}, target }` 형식으로 설정하였다.

| **AND gate** | | |
|---|---|---|
| Input | | Output |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| **OR gate** | | |
|---|---|---|
| Input | | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

| **XOR gate** | | |
|---|---|---|
| Input | | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

```
// Sigmoid 함수
double Sigmoid(double x)
{
        return 1 / (1 + exp(-x));
}
```

- Sigmoid 함수 연산 결과를 double형으로 반환하는 함수인 `Sigmoid(double x)`를 선언했다.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

② Neuron 클래스

```
class Neuron
{
private:
        vector<double> weight;
        double bias;

        // Initial Value(초기값) 설정
        void setValue()
        {
                bias = -1;      // bias의 초기값은 편의상 -1로 고정

                random_device rd;
                mt19937 random(rd());   // 난수 생성
                uniform_real_distribution<double> distr(-1.0, 1.0);
                                                        // 생성되는 난수는 -1.0 이상 1.0 미만의 실수

                for (size_t i = 0; i < weight.size(); i++)
                {
                        weight[i] = distr(random);      // weight vector에 난수 저장
                }

                // weight[0] = -0.577339;       // 보고서 작성 시 임시로 설정한 값
                // weight[1] = 0.744271;        // 보고서 작성 시 임시로 설정한 값
        }
```

- private member로 vector type의 weight와 double형의 bias를 선언하였다.
- 뉴런의 초기 값을 설정하는 setValue()에서는 weight와 bias의 초기 값을 다음과 같이 설정하였다.

| weight | -1.0 ≤ weight < 1.0 범위의 난수 |
|---|---|
| bias | -1 |

- 난수는 random device와 mt19937을 이용하여 매 실행마다 weight에 예측할 수 없는 난수가 저장되도록 설정했으나, 보고서에 첨부한 실험 결과 값은 편리한 비교를 위해 weight에 임시로 값을 지정하여 산출된 값이다. 이 값은 수차례의 실험을 통해 선정된 값이다.

```
public:
        // 생성자
        Neuron(int input_size)
        {
                weight.resize(input_size); // weight.size() == INPUT_SIZE
                setValue();
        }

        // Sigmoid 함수의 연산 결과인 output을 반환하는 함수
        double Calculate(const vector<double> & input)
        {
                double output, wx = 0.0;

                for (size_t i = 0; i < weight.size(); i++)
                {
                        wx += weight[i] * input[i];     // wx의 합
                }

                output = Sigmoid(bias + wx);
                return output;
        }
```

- 생성자에서는 weight vector의 크기를 input의 크기로 재설정하며, 초기 값을 설정한다.
- Sigmoid 함수의 연산 결과인 output을 double형으로 반환하는 Calculate(const vector<double> & input)에서는 다음의 수식에 따라 output을 계산한다.

$$output = \sigma(\sum_{i=0}^{n} w_i x_i) = \sigma(bias + \sum_{i=1}^{n} w_i x_i)$$

```
        // output과 target을 비교하여 학습시키는 함수
        // c는 학습률(LEARNING_RATE), target은 학습 데이터에서 input 조합으로 설정한 값의 vector
        void Learn(double c, const vector<pair<vector<double>, double>> & target)
        {
                int input_size = target[0].first.size();

                for (size_t i = 0; i < target.size(); i++)
                {
                        double o = Calculate(target[i].first); // 실제 계산된 output
                        double t = target[i].second; // 학습 데이터에 설정된 target

                        for (int j = 0; j < input_size; j++)
                        {
                                weight[j] += c * (t - o) * target[i].first[j]; // weight 값의 조정
                        }

                        bias += c * (t - o) * 1;        // bias 값의 조정
                }
        }

        // private 멤버인 weight의 getter
        vector<double> & getWeight()
        {
                return weight;
        }

        // private 멤버인 bias의 getter
        double getBias()
        {
                return bias;
        }
};
```

- Output과 target을 비교하여 뉴런을 학습시키는 함수인 Learn(double c, const vector<pair<vector<double>, double>> & target)은 LEARNING_RATE c와 학습 데이터의 input 조합을 인수로 받아 다음의 수식에 따라 학습을 진행한다.

$$\Delta w_k = c(t_i - O_i)f'(net)x_k$$

$$w_k \leftarrow w_k + \Delta w_k$$

이때, 이 프로그램은 단층 퍼셉트론이므로 $f'(net) = 1$이며 $bias = w_0 x_0 = w_0 * 1$, 즉 input을 1이라고 생각하면 실제 델타 값의 연산은 다음과 같다.

$$\boldsymbol{\Delta w_k = c(t_i - O_i)x_k}$$

$$\boldsymbol{\Delta bias = c(t_i - O_i)}$$

- private member인 weight와 bias의 getter인 getWeight()와 getBias()를 선언하였다.

③ 그 외의 함수

```
// Neuron을 학습시키고 그 결과를 출력하는 함수
void showResult(Neuron neuron, const vector<pair<vector<double>, double>> & target, string gateName)
{
	int i, j, k;
	double gradient, intercept; // 기울기, y 절편
	double error, total_error = 0.0; // error = 0.5 * pow((t - o), 2.0), total_error = error의 합

	cout << "\n-----------------------------------------------------------------\n";
	cout << "============================ " << gateName << " gate ============================\n";
	cout << "-----------------------------------------------------------------\n";

	// Initial Value 출력
	cout << "[Initial Value] ";
	for (i = 0; i < INPUT_SIZE; i++)
	{
		cout << "W" << i + 1 << " = " << neuron.getWeight()[i] << ", ";
	}
	cout << "bias = " << neuron.getBias() << '\n';

	// (INPUT_SIZE == 2)일 때, 직선의 기울기 및 y 절편 출력
	gradient = -(neuron.getWeight()[0] / neuron.getWeight()[1]);
	intercept = -(neuron.getBias() / neuron.getWeight()[1]);
	cout << "  => gradient = " << gradient << ", intercept = " << intercept << '\n';

	// 결과 출력
	for (j = 0; j < pow(2.0, INPUT_SIZE); j++)
	{
		for (k = 0; k < INPUT_SIZE - 1; k++)
		{
			cout << target[j].first[k] << " " << gateName << " ";
		}
		cout << target[j].first[k] << " => " << neuron.Calculate(target[j].first);
		error = 0.5 * pow((target[j].second - neuron.Calculate(target[j].first)), 2.0);
		cout << "       // error = " << error << '\n';
	}
	cout << "\t\t\t // total error = " << total_error << '\n';
	cout << "-----------------------------------------------------------------\n";
```

- Neuron을 학습시키고 그 결과를 출력하는 함수인 showResult(Neuron neuron, const vector<pair<vector<double>, double>> & target, string gateName)에서는 직선의 기울기인 gradient, y 절편인 intercept, error, 그리고 total_error 값을 다음의 수식에 따라 계산한다.

$$w_1x_1 + w_2x_2 + bias = 0$$

$$w_2x_2 = -w_1x_1 - bias$$

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{bias}{w_2}$$

$$\therefore \boldsymbol{gradient} = -\frac{\boldsymbol{w_1}}{\boldsymbol{w_2}}, \qquad \boldsymbol{intercept} = -\frac{\boldsymbol{bias}}{\boldsymbol{w_2}}$$

$$\therefore \boldsymbol{error_i} = \frac{\boldsymbol{1}}{\boldsymbol{2}}(\boldsymbol{t_i} - \boldsymbol{O_i})^2, \qquad \boldsymbol{total\ error} = \sum_{\boldsymbol{i}} \boldsymbol{error_i}$$

- 학습 시작 전, gradient, intercept, error, output 값의 초기 값(initial value), total_error를 출력한다.

```
        // 학습
        for (i = 1; i <= EPOCH_MAX; i++)
        {
                error = 0.0;
                total_error = 0.0;
                neuron.Learn(LEARNING_RATE, target);

                if (i == 5 || i == 10 || i == 20 || i == 35 || i == 50 || i == 100 || i == 500 ||
                        i == 1000 || i == 5000 || i == 10000 || (i % 50000) == 0)
                {
                        cout << "[Epoch " << i << "] ";
                        for (j = 0; j < INPUT_SIZE; j++)
                        {
                                cout << "W" << j + 1 << " = " << neuron.getWeight()[j] << ", ";
                        }
                        cout << "bias = " << neuron.getBias() << '\n';

                        // (INPUT_SIZE == 2)일 때, 직선의 기울기 및 y 절편 출력
                        gradient = -(neuron.getWeight()[0] / neuron.getWeight()[1]);
                        intercept = -(neuron.getBias() / neuron.getWeight()[1]);
                        cout << "  => gradient = " << gradient << ", intercept = " << intercept <<
                        '\n';

                        // 결과 출력
                        for (j = 0; j < pow(2.0, INPUT_SIZE); j++)
                        {
                                for (k = 0; k < INPUT_SIZE - 1; k++)
                                {
                                        cout << target[j].first[k] << " " << gateName << " ";
                                }
                                cout << target[j].first[k] << " => " <<
                                        neuron.Calculate(target[j].first);
                                error = 0.5*pow((target[j].second -
                                neuron.Calculate(target[j].first)), 2.0);
                                cout << "       // error = " << error << '\n';
                                total_error += error;
                        }
                        cout << "\t\t\t // total error = " << total_error << '\n';
                        cout << "-------------------------------------------------------------------\n";
                }
        }
}
```

- EPOCH_MAX 번의 학습을 Neuron 클래스의 Learn 함수를 통해 수행한다.
- 수차례의 실험을 통해 학습 초기에 큰 변화가 많이 일어난다는 것을 알게 되어 대표적으로 5, 10, 20, 35, 50, 100, 500, 1000, 5000, 10000, 50000, 100000 번째의 학습 결과, 즉 gradient, intercept, error, total_error, 그리고 output 값을 출력하였다. 이 값들의 계산 방법은 위의 수식과 동일하다.

```
int main(void)
{
        //AND gate
        Neuron neuron_AND(INPUT_SIZE);
        showResult(neuron_AND, target_AND, "AND");

        //OR gate
        Neuron neuron_OR(INPUT_SIZE);
        showResult(neuron_OR, target_OR, "OR");

        //XOR gate
        Neuron neuron_XOR(INPUT_SIZE);
        showResult(neuron_XOR, target_XOR, "XOR");

        return 0;
}
```

- main(void) 함수에서는 AND, OR, XOR gate의 학습을 수행할 Neuron 객체를 생성하였으며, showResult 함수를 호출하여 학습 및 결과 출력을 실행하였다.

**> 프로그램 실행 시 출력 결과**

- AND gate

```
------------------------------------------------------------
========================= AND gate =========================
------------------------------------------------------------
[Initial Value] W1 = -0.577339, W2 = 0.744271, bias = -1
  => gradient = 0.775711, intercept = 1.3436
0 AND 0 => 0.268941         // error = 0.0361647
0 AND 1 => 0.436414         // error = 0.0952285
1 AND 0 => 0.171173         // error = 0.01465
1 AND 1 => 0.302997         // error = 0.242907
                           // total error = 0.38895
------------------------------------------------------------
[Epoch 5] W1 = -0.3138, W2 = 0.876, bias = -1.086
  => gradient = 0.358219, intercept = 1.23972
0 AND 0 => 0.252372         // error = 0.0318459
0 AND 1 => 0.447692         // error = 0.100214
1 AND 0 => 0.197848         // error = 0.0195719
1 AND 1 => 0.371964         // error = 0.197214
                           // total error = 0.348846
------------------------------------------------------------
[Epoch 10] W1 = -0.0898062, W2 = 0.976814, bias = -1.20436
  => gradient = 0.0919378, intercept = 1.23295
0 AND 0 => 0.2307          // error = 0.0266112
0 AND 1 => 0.443357         // error = 0.0982827
1 AND 0 => 0.215148         // error = 0.0231443
1 AND 1 => 0.42132          // error = 0.167435
                           // total error = 0.315473
------------------------------------------------------------
[Epoch 20] W1 = 0.280278, W2 = 1.13347, bias = -1.47183
  => gradient = -0.247274, intercept = 1.29851
0 AND 0 => 0.186665         // error = 0.0174219
0 AND 1 => 0.416209         // error = 0.086615
1 AND 0 => 0.232982         // error = 0.0271404
1 AND 1 => 0.485485         // error = 0.132363
                           // total error = 0.26354
------------------------------------------------------------
[Epoch 35] W1 = 0.716941, W2 = 1.32712, bias = -1.86161
  => gradient = -0.540225, intercept = 1.40275
0 AND 0 => 0.134516         // error = 0.00904723
0 AND 1 => 0.36947          // error = 0.0682539
1 AND 0 => 0.241464         // error = 0.0291525
1 AND 1 => 0.545486         // error = 0.103292
                           // total error = 0.209745
------------------------------------------------------------
[Epoch 50] W1 = 1.0646, W2 = 1.50323, bias = -2.21066
  => gradient = -0.708206, intercept = 1.4706
0 AND 0 => 0.0987973         // error = 0.00488045
0 AND 1 => 0.330168         // error = 0.0545054
1 AND 0 => 0.241209         // error = 0.0290909
1 AND 1 => 0.588356         // error = 0.0847256
                           // total error = 0.173202
------------------------------------------------------------
------------------------------------------------------------
[Epoch 100] W1 = 1.87307, W2 = 2.02515, bias = -3.1439
  => gradient = -0.924902, intercept = 1.55243
0 AND 0 => 0.0413322         // error = 0.000854176
0 AND 1 => 0.246244         // error = 0.0303179
1 AND 0 => 0.219115         // error = 0.0240056
1 AND 1 => 0.680119         // error = 0.0511618
                           // total error = 0.10634
------------------------------------------------------------
[Epoch 500] W1 = 4.33013, W2 = 4.32065, bias = -6.65869
  => gradient = -1.00219, intercept = 1.54113
0 AND 0 => 0.00128119         // error = 8.20721e-07
0 AND 1 => 0.0880214         // error = 0.00387388
1 AND 0 => 0.0887852         // error = 0.00394141
1 AND 1 => 0.879964         // error = 0.0072043
                           // total error = 0.0150204
------------------------------------------------------------
[Epoch 1000] W1 = 5.62657, W2 = 5.62061, bias = -8.60361
  => gradient = -1.00106, intercept = 1.53072
0 AND 0 => 0.000183408         // error = 1.68192e-08
0 AND 1 => 0.0481999         // error = 0.00116161
1 AND 0 => 0.0484736         // error = 0.00117484
1 AND 1 => 0.933613         // error = 0.00220359
                           // total error = 0.00454007
------------------------------------------------------------
[Epoch 5000] W1 = 8.83616, W2 = 8.8349, bias = -13.4213
  => gradient = -1.00014, intercept = 1.51913
0 AND 0 => 1.48319e-06         // error = 1.09993e-12
0 AND 1 => 0.0100865         // error = 5.08689e-05
1 AND 0 => 0.0100991         // error = 5.0996e-05
1 AND 1 => 0.985933         // error = 9.8944e-05
                           // total error = 0.000200809
------------------------------------------------------------
[Epoch 10000] W1 = 10.2342, W2 = 10.2336, bias = -15.519
  => gradient = -1.00006, intercept = 1.51647
0 AND 0 => 1.82043e-07         // error = 1.65699e-14
0 AND 1 => 0.00503948         // error = 1.26982e-05
1 AND 0 => 0.00504264         // error = 1.27141e-05
1 AND 1 => 0.992958         // error = 2.47923e-05
                           // total error = 5.02046e-05
------------------------------------------------------------
------------------------------------------------------------
[Epoch 50000] W1 = 13.4706, W2 = 13.4705, bias = -20.374
  => gradient = -1.00001, intercept = 1.51249
0 AND 0 => 1.41804e-09         // error = 1.00542e-18
0 AND 1 => 0.00100322         // error = 5.03229e-07
1 AND 0 => 0.00100335         // error = 5.03355e-07
1 AND 1 => 0.998596         // error = 9.85562e-07
                           // total error = 1.99215e-06
------------------------------------------------------------
[Epoch 100000] W1 = 14.8604, W2 = 14.8603, bias = -22.4588
  => gradient = -1, intercept = 1.51132
0 AND 0 => 1.7631e-10         // error = 1.55427e-20
0 AND 1 => 0.000500982         // error = 1.25492e-07
1 AND 0 => 0.000501014         // error = 1.25507e-07
1 AND 1 => 0.999299         // error = 2.45868e-07
                           // total error = 4.96867e-07
------------------------------------------------------------
```

- OR gate

```
------------------------------------------------------------
========================= OR gate ==========================
------------------------------------------------------------
[Initial Value] W1 = -0.577339, W2 = 0.744271, bias = -1
  => gradient = 0.775711, intercept = 1.3436
0 OR 0 => 0.268941         // error = 0.0361647
0 OR 1 => 0.436414         // error = 0.158815
1 OR 0 => 0.171173         // error = 0.343477
1 OR 1 => 0.302997         // error = 0.242907
                           // total error = 0.781364
------------------------------------------------------------
[Epoch 5] W1 = 0.0176174, W2 = 1.19954, bias = -0.349452
  => gradient = -0.0146869, intercept = 0.291322
0 OR 0 => 0.413515         // error = 0.0854975
0 OR 1 => 0.700585         // error = 0.0448247
1 OR 0 => 0.417794         // error = 0.169482
1 OR 1 => 0.704267         // error = 0.0437289
                           // total error = 0.343533
------------------------------------------------------------
[Epoch 10] W1 = 0.378845, W2 = 1.43401, bias = -0.0791325
  => gradient = -0.264187, intercept = 0.0551829
0 OR 0 => 0.480227         // error = 0.115309
0 OR 1 => 0.794925         // error = 0.0210279
1 OR 0 => 0.574372         // error = 0.0905795
1 OR 1 => 0.849887         // error = 0.0112669
                           // total error = 0.238183
------------------------------------------------------------
[Epoch 20] W1 = 0.837555, W2 = 1.71149, bias = 0.0566542
  => gradient = -0.489373, intercept = -0.0331024
0 OR 0 => 0.51416          // error = 0.13218
0 OR 1 => 0.854226         // error = 0.010625
1 OR 0 => 0.709758         // error = 0.0421202
1 OR 1 => 0.931227         // error = 0.00236485
                           // total error = 0.18729
------------------------------------------------------------
[Epoch 35] W1 = 1.29949, W2 = 1.9938, bias = -0.0335427
  => gradient = -0.651765, intercept = 0.0168235
0 OR 0 => 0.491615         // error = 0.120843
0 OR 1 => 0.87656          // error = 0.00761866
1 OR 0 => 0.780048         // error = 0.0241895
1 OR 1 => 0.963022         // error = 0.000683701
                           // total error = 0.153335
------------------------------------------------------------
[Epoch 50] W1 = 1.65837, W2 = 2.22561, bias = -0.19708
  => gradient = -0.745129, intercept = 0.0885512
0 OR 0 => 0.450889         // error = 0.10165
0 OR 1 => 0.88376          // error = 0.00675588
1 OR 0 => 0.811729         // error = 0.0177229
1 OR 1 => 0.975562         // error = 0.000298597
                           // total error = 0.126428
------------------------------------------------------------
------------------------------------------------------------
[Epoch 100] W1 = 2.54629, W2 = 2.86522, bias = -0.690458
  => gradient = -0.888691, intercept = 0.240979
0 OR 0 => 0.333931         // error = 0.055755
0 OR 1 => 0.89796          // error = 0.0052061
1 OR 0 => 0.864811         // error = 0.00913808
1 OR 1 => 0.991173         // error = 3.89597e-05
                           // total error = 0.0701381
------------------------------------------------------------
[Epoch 500] W1 = 5.41512, W2 = 5.44727, bias = -2.21576
  => gradient = -0.994099, intercept = 0.406766
0 OR 0 => 0.0983438          // error = 0.00483575
0 OR 1 => 0.962003           // error = 0.000721895
1 OR 0 => 0.96081            // error = 0.000767921
1 OR 1 => 0.999824           // error = 1.5433e-08
                             // total error = 0.00632558
------------------------------------------------------------
[Epoch 1000] W1 = 6.80951, W2 = 6.81986, bias = -2.92891
  => gradient = -0.998483, intercept = 0.429468
0 OR 0 => 0.0507427          // error = 0.00128741
0 OR 1 => 0.979983           // error = 0.000200344
1 OR 0 => 0.979779           // error = 0.000204447
1 OR 1 => 0.999977           // error = 2.53894e-10
                             // total error = 0.0016922
------------------------------------------------------------
[Epoch 5000] W1 = 10.0851, W2 = 10.0862, bias = -4.5808
  => gradient = -0.999896, intercept = 0.454167
0 OR 0 => 0.0101427          // error = 5.14374e-05
0 OR 1 => 0.995952           // error = 8.1949e-06
1 OR 0 => 0.995947           // error = 8.21211e-06
1 OR 1 => 1          // error = 1.43629e-14
                             // total error = 6.78444e-05
------------------------------------------------------------
[Epoch 10000] W1 = 11.4874, W2 = 11.4879, bias = -5.28377
  => gradient = -0.99996, intercept = 0.459942
0 OR 0 => 0.00504765         // error = 1.27394e-05
0 OR 1 => 0.997983           // error = 2.03413e-06
1 OR 0 => 0.997982           // error = 2.03601e-06
1 OR 1 => 1          // error = 2.14917e-16
                             // total error = 1.68095e-05
------------------------------------------------------------
------------------------------------------------------------
[Epoch 50000] W1 = 14.7245, W2 = 14.7246, bias = -6.90375
  => gradient = -0.999994, intercept = 0.468859
0 OR 0 => 0.00100301         // error = 5.03014e-07
0 OR 1 => 0.999599           // error = 8.04504e-08
1 OR 0 => 0.999599           // error = 8.04637e-08
1 OR 1 => 1          // error = 1.30718e-20
                             // total error = 6.63928e-07
------------------------------------------------------------
[Epoch 100000] W1 = 16.114, W2 = 16.114, bias = -7.59866
  => gradient = -0.999997, intercept = 0.471556
0 OR 0 => 0.000500871        // error = 1.25436e-07
0 OR 1 => 0.9998         // error = 2.00658e-08
1 OR 0 => 0.9998         // error = 2.00674e-08
1 OR 1 => 1          // error = 2.02401e-22
                             // total error = 1.65569e-07
------------------------------------------------------------
```

- XOR gate

```
--------------------------------------------------------------------------------
============================ XOR gate ============================
--------------------------------------------------------------------------------
[Initial Value] W1 = -0.577339, W2 = 0.744271, bias = -1
  => gradient = 0.775711, intercept = 1.3436
0 XOR 0 => 0.268941         // error = 0.0361647
0 XOR 1 => 0.436414         // error = 0.158815
1 XOR 0 => 0.171173         // error = 0.343477
1 XOR 1 => 0.302997         // error = 0.0459035
                           // total error = 0.58436
--------------------------------------------------------------------------------
[Epoch 5] W1 = -0.38945, W2 = 0.801462, bias = -0.696054
  => gradient = 0.485924, intercept = 0.86848
0 XOR 0 => 0.332688         // error = 0.0553406
0 XOR 1 => 0.526328         // error = 0.112183
1 XOR 0 => 0.252466         // error = 0.279404
1 XOR 1 => 0.429463         // error = 0.0922193
                           // total error = 0.539146
--------------------------------------------------------------------------------
[Epoch 10] W1 = -0.283012, W2 = 0.779788, bias = -0.531635
  => gradient = 0.362935, intercept = 0.681769
0 XOR 0 => 0.370136         // error = 0.0685002
0 XOR 1 => 0.561722         // error = 0.0960439
1 XOR 0 => 0.306901         // error = 0.240193
1 XOR 1 => 0.491286         // error = 0.120681
                           // total error = 0.525418
--------------------------------------------------------------------------------
[Epoch 20] W1 = -0.178704, W2 = 0.661996, bias = -0.376537
  => gradient = 0.269948, intercept = 0.56879
0 XOR 0 => 0.406962         // error = 0.0828092
0 XOR 1 => 0.570884         // error = 0.0920702
1 XOR 0 => 0.364649         // error = 0.201835
1 XOR 1 => 0.526663         // error = 0.138687
                           // total error = 0.515402
--------------------------------------------------------------------------------
[Epoch 35] W1 = -0.112678, W2 = 0.479323, bias = -0.269686
  => gradient = 0.235077, intercept = 0.56264
0 XOR 0 => 0.432984         // error = 0.0937976
0 XOR 1 => 0.552218         // error = 0.100254
1 XOR 0 => 0.405557         // error = 0.176681
1 XOR 1 => 0.524221         // error = 0.137404
                           // total error = 0.508077
--------------------------------------------------------------------------------
[Epoch 50] W1 = -0.0815435, W2 = 0.339165, bias = -0.199719
  => gradient = 0.240424, intercept = 0.588854
0 XOR 0 => 0.450236         // error = 0.101356
0 XOR 1 => 0.534805         // error = 0.108203
1 XOR 0 => 0.430144         // error = 0.162368
1 XOR 1 => 0.514472         // error = 0.132341
                           // total error = 0.504267
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
[Epoch 100] W1 = -0.0618208, W2 = 0.0927711, bias = -0.0613491
  => gradient = 0.66638, intercept = 0.661296
0 XOR 0 => 0.484668         // error = 0.117451
0 XOR 1 => 0.507855         // error = 0.121103
1 XOR 0 => 0.469246         // error = 0.14085
1 XOR 1 => 0.492401         // error = 0.121229
                           // total error = 0.500634
--------------------------------------------------------------------------------
[Epoch 500] W1 = -0.102414, W2 = -0.0511269, bias = 0.0510957
  => gradient = -2.00314, intercept = 0.99939
0 XOR 0 => 0.512771         // error = 0.131467
0 XOR 1 => 0.499992         // error = 0.125004
1 XOR 0 => 0.487173         // error = 0.131496
1 XOR 1 => 0.474411         // error = 0.112533
                           // total error = 0.5005
--------------------------------------------------------------------------------
[Epoch 1000] W1 = -0.102563, W2 = -0.0512817, bias = 0.0512817
  => gradient = -2, intercept = 1
0 XOR 0 => 0.512818         // error = 0.131491
0 XOR 1 => 0.5            // error = 0.125
1 XOR 0 => 0.487182         // error = 0.131491
1 XOR 1 => 0.474382         // error = 0.112519
                           // total error = 0.500501
--------------------------------------------------------------------------------
[Epoch 5000] W1 = -0.102564, W2 = -0.0512818, bias = 0.0512818
  => gradient = -2, intercept = 1
0 XOR 0 => 0.512818         // error = 0.131491
0 XOR 1 => 0.5            // error = 0.125
1 XOR 0 => 0.487182         // error = 0.131491
1 XOR 1 => 0.474382         // error = 0.112519
                           // total error = 0.500501
--------------------------------------------------------------------------------
[Epoch 10000] W1 = -0.102564, W2 = -0.0512818, bias = 0.0512818
  => gradient = -2, intercept = 1
0 XOR 0 => 0.512818         // error = 0.131491
0 XOR 1 => 0.5            // error = 0.125
1 XOR 0 => 0.487182         // error = 0.131491
1 XOR 1 => 0.474382         // error = 0.112519
                           // total error = 0.500501
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
[Epoch 50000] W1 = -0.102564, W2 = -0.0512818, bias = 0.0512818
  => gradient = -2, intercept = 1
0 XOR 0 => 0.512818         // error = 0.131491
0 XOR 1 => 0.5            // error = 0.125
1 XOR 0 => 0.487182         // error = 0.131491
1 XOR 1 => 0.474382         // error = 0.112519
                           // total error = 0.500501
--------------------------------------------------------------------------------
[Epoch 100000] W1 = -0.102564, W2 = -0.0512818, bias = 0.0512818
  => gradient = -2, intercept = 1
0 XOR 0 => 0.512818         // error = 0.131491
0 XOR 1 => 0.5            // error = 0.125
1 XOR 0 => 0.487182         // error = 0.131491
1 XOR 1 => 0.474382         // error = 0.112519
                           // total error = 0.500501
--------------------------------------------------------------------------------
```

## 2. 프로그램을 통한 학습 결과

### > 설정

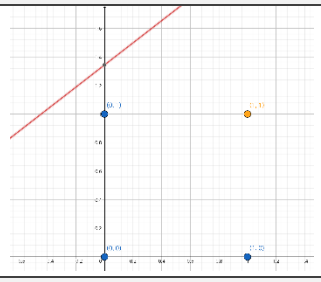
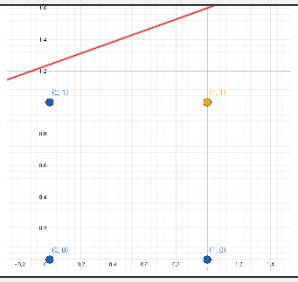
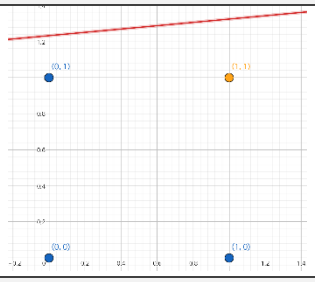
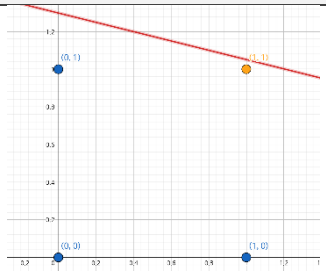
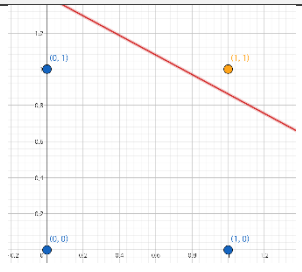
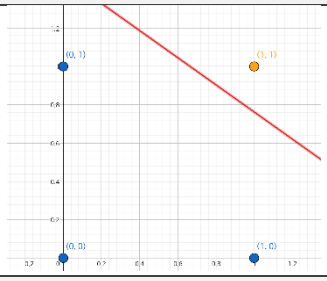
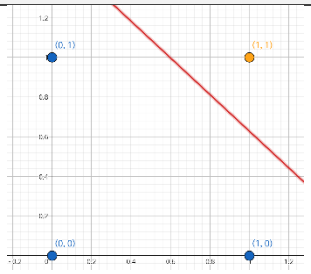
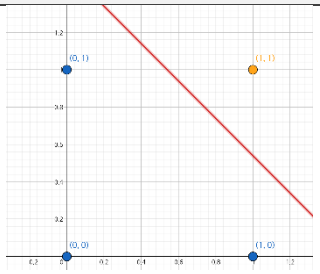
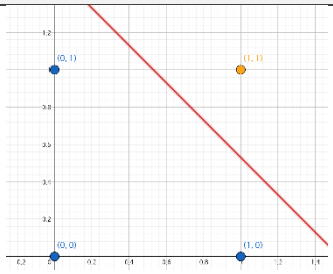
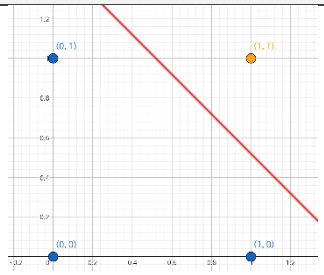
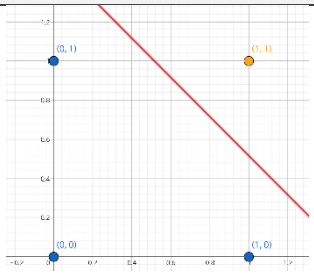
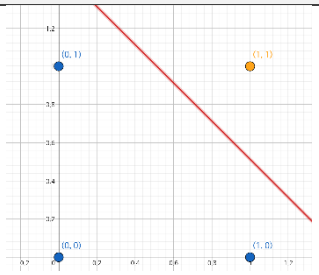
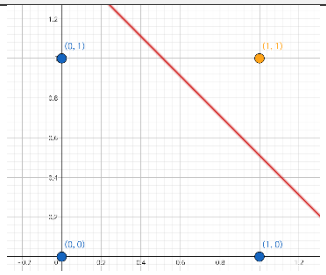
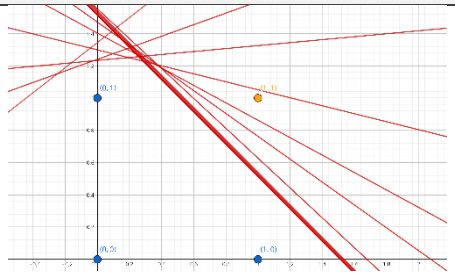
- 학습 초기에 큰 변화가 많이 일어나므로 Epoch의 값이 0(initial), 5, 10, 20, 35, 50, 100, 500, 1000, 5000, 10000, 50000, 100000일 때의 값으로 그래프를 작성하였다.

- 보고서에 첨부한 실행 결과 및 그래프는 weight의 초기 값을 W1 = -0.577339, W2= 0.744271로 설정한 결과이다. 이 값은 난수를 생성하여 weight 값을 설정하는 실험을 여러 차례 진행한 후, 학습과정에 따른 변화가 시각적으로 가장 잘 나타나는 값이었기 때문에 선택되었다. 실제 소스코드에는 초기 weight 값을 난수 생성 함수를 통해 설정한다.

- learning 과정 그래프는 가로축이 $x_1$축이고, 세로축이 $x_2$축이다. 그리고 노란색 점 🟡은 target output이 1인 점이며, 파란색 점 🔵은 target output이 0인 점이다.

**> AND gate**

[learning 과정 그래프]

| Epoch 0 (initial) | Epoch 5 | Epoch 10 |
|---|---|---|
|  |  |  |
| **Epoch 20** | **Epoch 35** | **Epoch 50** |
|  |  |  |
| **Epoch 100** | **Epoch 500** | **Epoch 1000** |
|  |  |  |
| **Epoch 5000** | **Epoch 10000** | **Epoch 50000** |
|  |  |  |
| **Epoch 10000** | **ALL** | |
|  |  | |

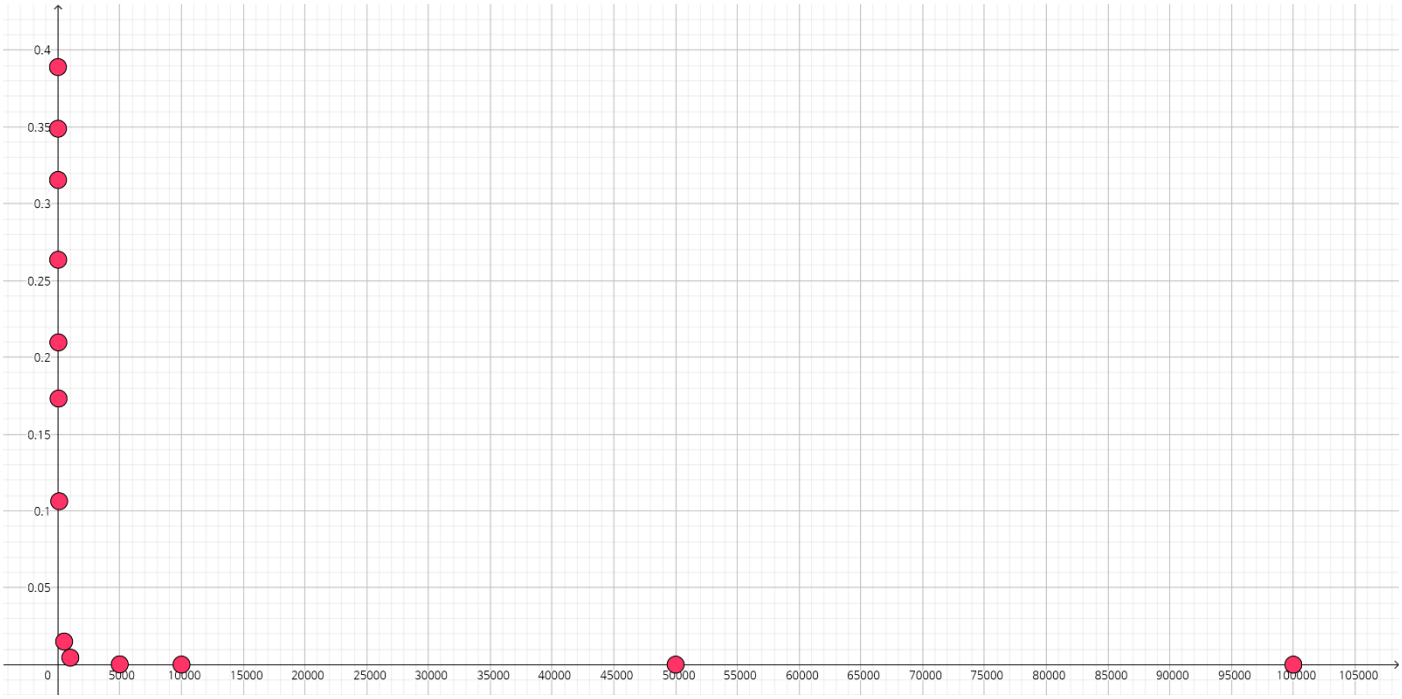
-

| Epoch 0 | | |
|---|---|---|
| 직선의 방정식 | $x_2 = 0.775711x_1 + 1.3436$ | |
| 연산 | output | error |
| 0 AND 0 | 0.268941 | 0.0361647 |
| 0 AND 1 | 0.436414 | 0.0952285 |
| 1 AND 0 | 0.171173 | 0.01465 |
| 1 AND 1 | 0.302997 | 0.242907 |
| **total error** | | **0.38895** |

| Epoch 100000 | | |
|---|---|---|
| 직선의 방정식 | $x_2 = -x_1 + 1.51132$ | |
| 연산 | output | error |
| 0 AND 0 | 1.7631e-10 | 1.55427e-20 |
| 0 AND 1 | 0.000500982 | 1.25492e-07 |
| 1 AND 0 | 0.000501014 | 1.25507e-07 |
| 1 AND 1 | 0.999299 | 2.45868e-07 |
| **total error** | | **4.96867e-07** |

- 2-input AND gate model은 linearly separable 하다.

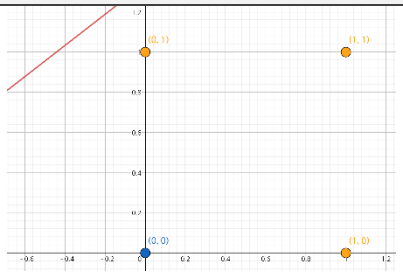
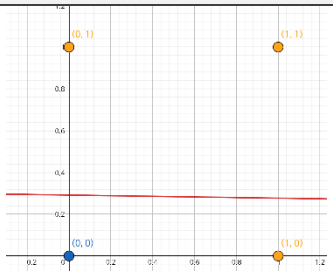
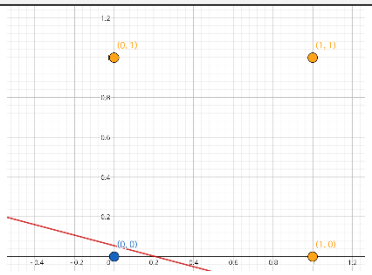
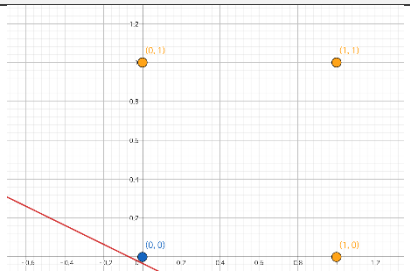
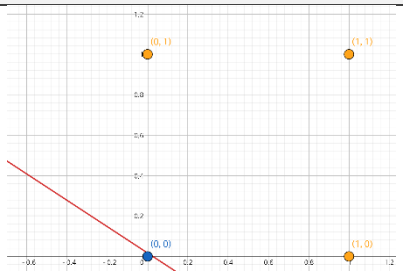
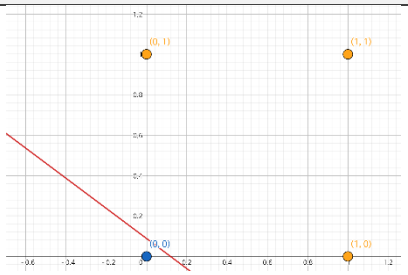
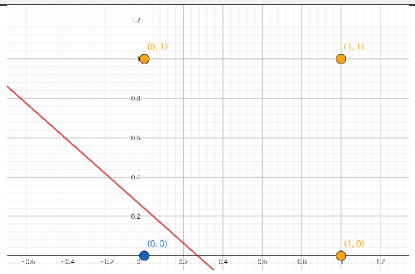
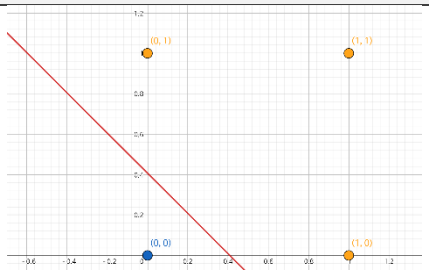
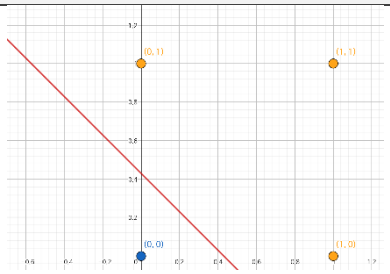
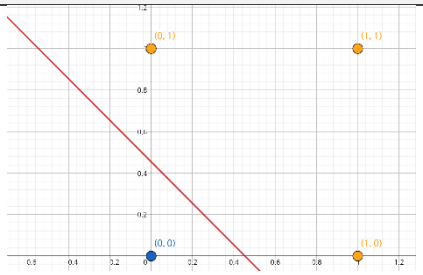
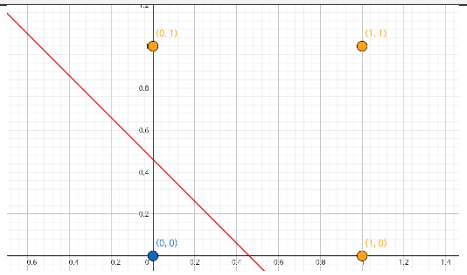
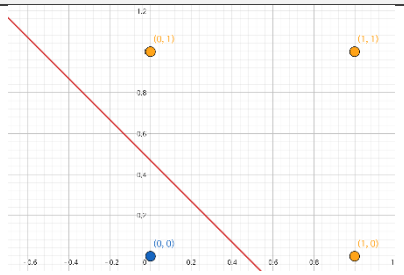
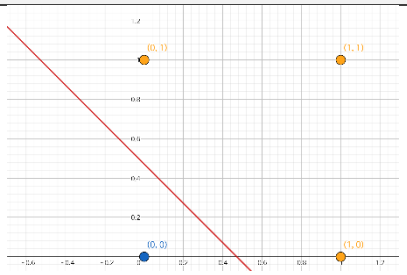
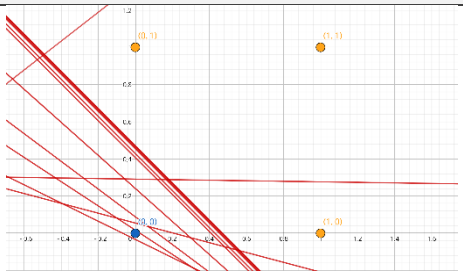
[iteration에 따른 error 그래프]




- total error 값이 학습이 진행됨에 따라 빠르게 0에 가까워졌다.

**> OR gate**

[learning 과정 그래프]

| Epoch 0 (initial) | Epoch 5 | Epoch 10 |
|---|---|---|
|  |  |  |
| **Epoch 20** | **Epoch 35** | **Epoch 50** |
|  |  |  |
| **Epoch 100** | **Epoch 500** | **Epoch 1000** |
|  |  |  |
| **Epoch 5000** | **Epoch 10000** | **Epoch 50000** |
|  |  |  |
| **Epoch 10000** | **ALL** | |
|  |  | |

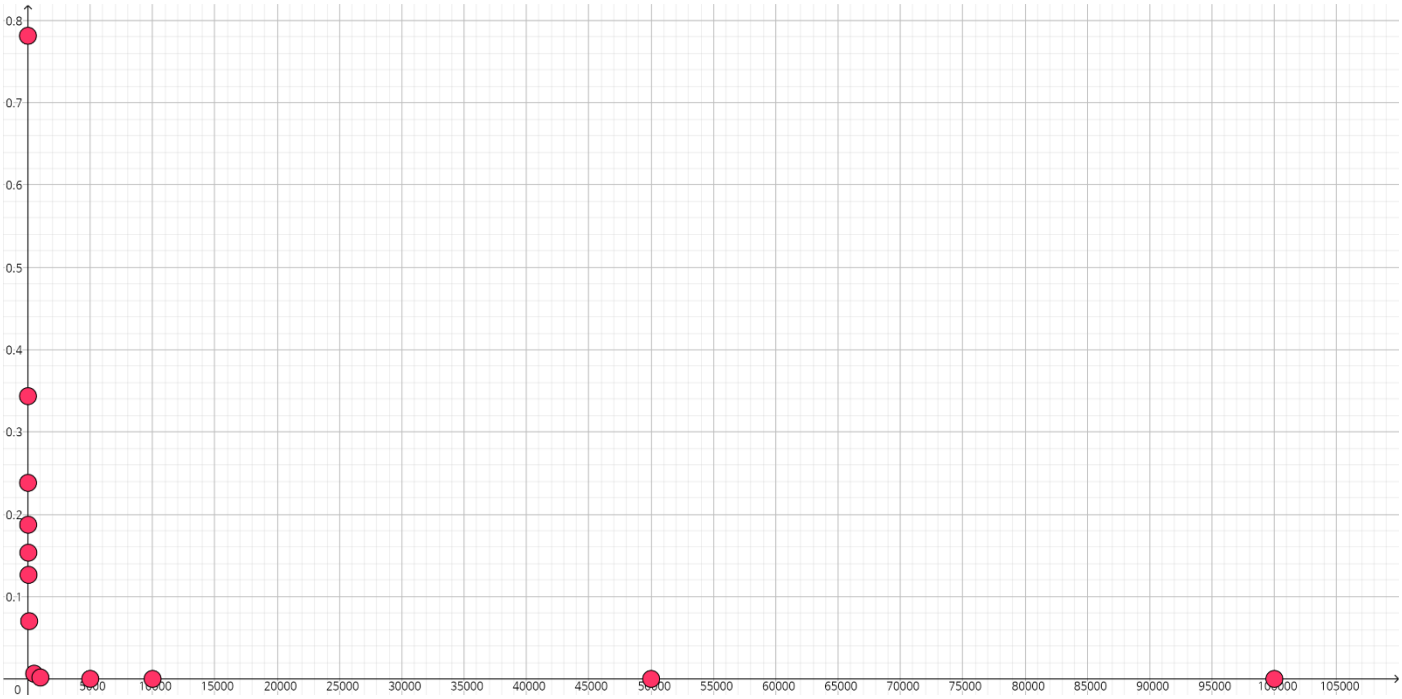
-

| Epoch 0 | | |
|---|---|---|
| 직선의 방정식 | $x_2 = 0.775711x_1 + 1.3436$ | |
| 연산 | output | error |
| 0 OR 0 | 0.268941 | 0.0361647 |
| 0 OR 1 | 0.436414 | 0.158815 |
| 1 OR 0 | 0.171173 | 0.343477 |
| 1 OR 1 | 0.302997 | 0.242907 |
| **total error** | | **0.781364** |

| Epoch 100000 | | |
|---|---|---|
| 직선의 방정식 | $x_2 = -0.999997x_1 + 0.471556$ | |
| 연산 | output | error |
| 0 OR 0 | 0.000500871 | 1.25436e-07 |
| 0 OR 1 | 0.9998 | 2.00658e-08 |
| 1 OR 0 | 0.9998 | 2.00674e-08 |
| 1 OR 1 | 1 | 2.02401e-22 |
| **total error** | | **1.65569e-07** |

- 2-input OR gate model은 linearly separable 하다.

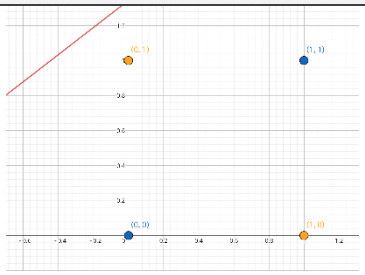
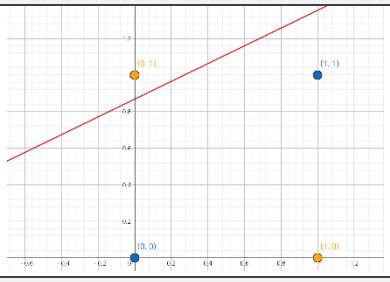
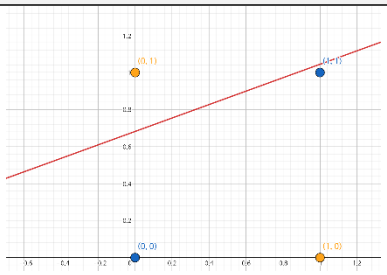
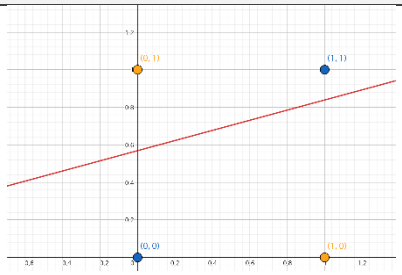
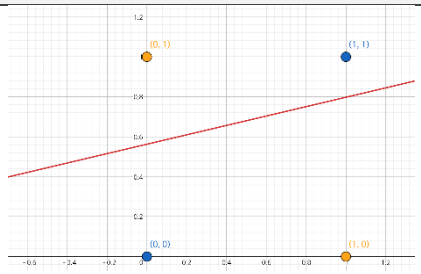
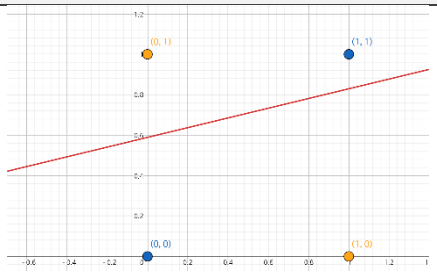
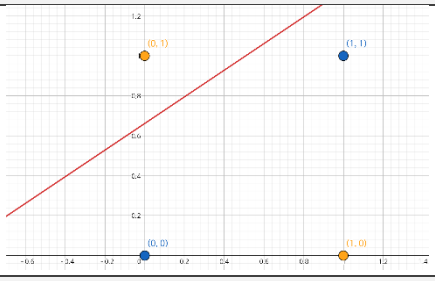
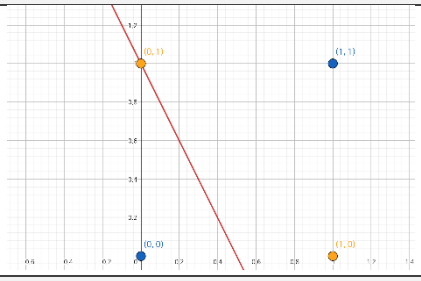
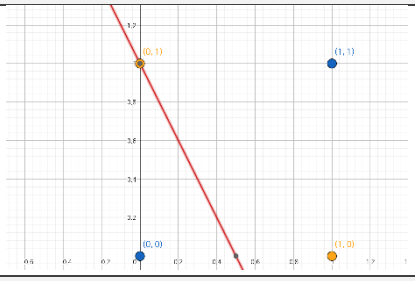
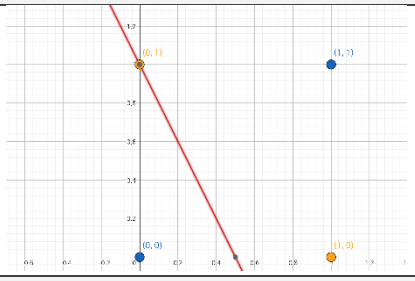
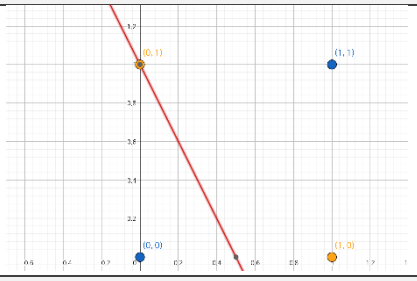
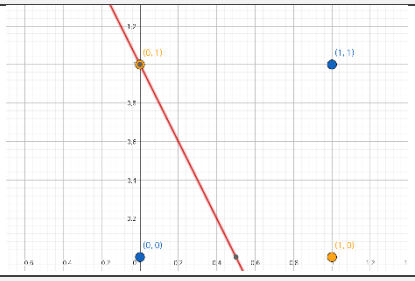
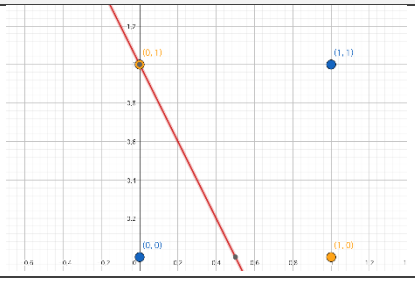
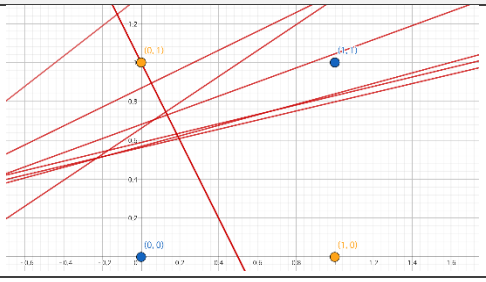
[iteration에 따른 error 그래프]




- total error 값이 학습이 진행됨에 따라 빠르게 0에 가까워졌다.

**> XOR gate**

[learning 과정 그래프]

| Epoch 0 (initial) | Epoch 5 | Epoch 10 |
|---|---|---|
|  |  |  |
| **Epoch 20** | **Epoch 35** | **Epoch 50** |
|  |  |  |
| **Epoch 100** | **Epoch 500** | **Epoch 1000** |
|  |  |  |
| **Epoch 5000** | **Epoch 10000** | **Epoch 50000** |
|  |  |  |
| **Epoch 10000** | **ALL** | |
|  |  | |

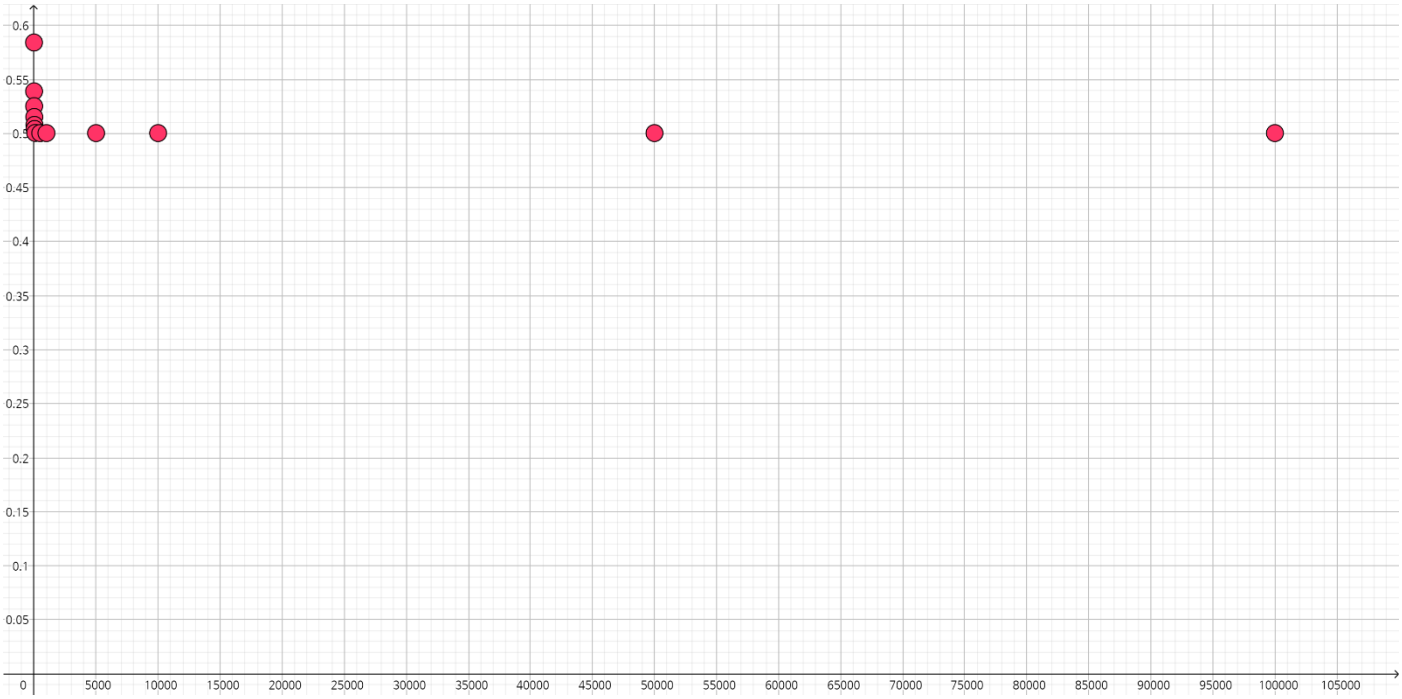
-

| Epoch 0 | | |
|---|---|---|
| 직선의 방정식 | $x_2 = 0.775711x_1 + 1.3436$ | |
| 연산 | output | error |
| 0 XOR 0 | 0.268941 | 0.0361647 |
| 0 XOR 1 | 0.436414 | 0.158815 |
| 1 XOR 0 | 0.171173 | 0.343477 |
| 1 XOR 1 | 0.302997 | 0.0459035 |
| **total error** | | **0.58436** |

| Epoch 100000 | | |
|---|---|---|
| 직선의 방정식 | $x_2 = -2x_1 + 1$ | |
| 연산 | output | error |
| 0 XOR 0 | 0.512818 | 0.131491 |
| 0 XOR 1 | 0.5 | 0.125 |
| 1 XOR 0 | 0.487182 | 0.131491 |
| 1 XOR 1 | 0.474382 | 0.112519 |
| **total error** | | **0.500501** |

- 2-input XOR gate model은 not linearly separable 하다.

[iteration에 따른 error 그래프]




- total error 값이 학습이 진행됨에 따라 0에 가까워지지 않고, AND gate와 OR gate에 비해서 매우 큰 error 값을 유지하였다. Epoch 100000에서의 error 값이 0.112519~0.131491의 값이었기 때문이다.

- 따라서 2-input XOR gate는 AND gate, OR gate와 다르게 단층 퍼셉트론으로 학습이 불가능하다는 것을 알 수 있다.

## 3. 결론 및 고찰

**> 결론**

- 2-input AND gate와 OR gate는 단층 퍼셉트론으로 학습이 가능하였으나, XOR gate는 불가능했다. 그 이유는 XOR gate model이 not linearly separable하기 때문이다.

- XOR gate는 단층 퍼셉트론이 아닌 다층 퍼셉트론을 통해 학습이 가능할 것이다.

- Neuron 객체를 이용하여 Network(신경망) 클래스를 만들면 다층 퍼셉트론을 구현할 수 있을 것이다.

**> 고찰**

- C++에서 vector를 사용하는 경우, 그 크기를 미리 비교하여 index를 벗어나거나 메모리를 침범하는 사고를 방지해야 할 것이다.

- 학습 전이나 중간에 weight의 값이 0이 되는 경우를 감지해야 하며, 이에 따른 대처 방안을 마련해야 한다.