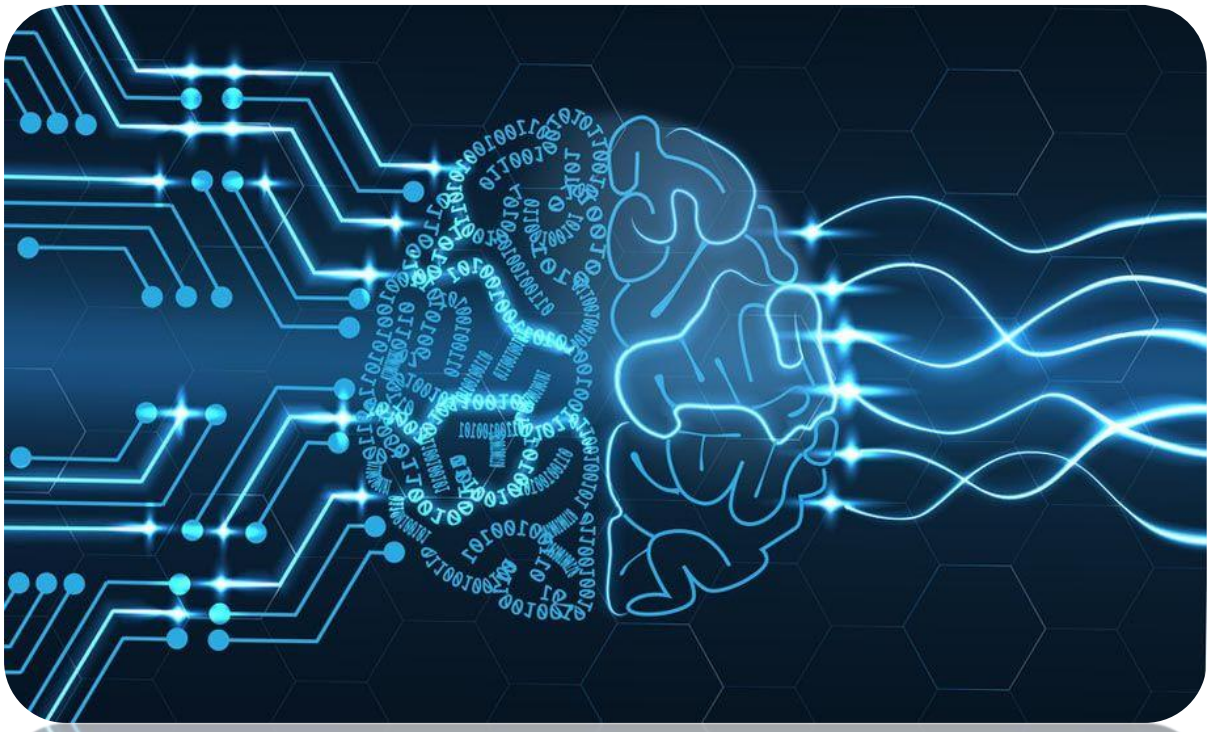


# Perceptron Learning

인공지능 과제 #2

컴퓨터과학부 2017920036 양다은

## 〈 목차 〉

# ‹ I. 코드구현 ›

## 1. 과제 설명

과제 #2 입력이 n차원인 perceptron learning 구현

- AND, OR, XOR gate 학습 실험 (2차원 input)
- Learning 과정을 그래프로 보여주기 (X1 , X2 2차원 직선 그래프).
- 반복 학습 (iteration)에 따른 Error 그래프
- 구현언어: C, C++ 언어
- 제출물: 프로그램, 실행파일, 실행 환경, 결과 보고서 zip file 실행 10%, 출력 10, 주석 10, 완성도 25, 오류 10, 창의 10 보고서 25%
- module 별로 구현할 것
- Class 로 구현하면 가점 있음. output 계산 및 learning 과정을 각각 member function 으로 구성.
- (다른 학생이나 선배, 인터넷에서 다운 받은 것 등, 명시하지 않고 유사한 소스를 제출하는 경우 마이너스 점수가 부여됨)

## 2. module.h

- C++ 언어 신경망 모듈 구현
- 모듈화를 목적으로 구조체와 함수포인터를 활용하여 각 class와 method 인스턴스화

```
// forward pass compute -> propagation
// backward pass compute -> backpropagation

#pragma once

typedef struct Module Module;
typedef struct Node Node;


// module 생성
struct  Module {

    int in_channels;    // 입력 채널 수
    int out_channels;   // 출력 채널 수
    Node* nodes;        // 노드 수
    float* input;       // input 값
    float* (*propagation)(struct Module, float* input);         // propagation
    float* (*backpropagation)(struct Module, float*, float);    // backpropagation
};


// Node 생성
struct Node {

    int in_channels;    // 입력 채널 수
    float* weight;      // weight 값
    float* theta;       // theta 값
    float (*propagation)(struct Node, float* input);                    // propagation
    float* (*backpropagation)(struct Node, float* input, float, float); // backpropagation
};
```

- Forward 계산은 propagation으로, Backward 계산은 backpropagation으로 용어 선언
- Module과 Node 구조체 생성

```
// Node method
Node init_node(int in_channels);    //초기 node
float node_propagation(Node self, float* input);    // propagation 계산
float* node_backpropagation(Node self, float* input, float loss, float c);  // backpropagation 계산

// class : Linear
// implement : Module
// Linear module method
Module init_linear(int in_channels, int out_channels);  //초기 계산
float* linear_propatation(Module self, float* input);   // linear module 적용한 propagation 계산
float* linear_backpropagation(Module self, float* loss, float c);   // linear module 적용한 backpropagation 계산

// class : Sigmoid
// implement : Module
// Sigmoid module method
Module init_sigmoid(int in_channels);   //초기 계산
float* sigmoid_propatation(Module self, float* input);  // sigmoid module 적용한 propagation 계산
float* sigmoid_backpropagation(Module self, float* loss, float c);  // sigmoid module 적용한 backpropagation 계산
```

- Node 구조체를 상속받아 Node method 클래스 구현
- Module 구조체를 상속받아 Linear module method 클래스 구현
- Module 구조체를 상속받아 Sigmoid module method 클래스 구현
- 각 클래스에서 init(초기 생성자), propagation, backpropagation 메소드 정의
- module.cpp에서 각 메소드 오버라이딩 재정의 구현

## 3. module.cpp

- 첨부파일 [과제#2]20179200036양다은₩소스코드₩02.PerceptronLearning₩module.cpp 참고
- Node method 오버라이딩 재정의
    1. init_node
        - ✓ 초기 생성자
        - ✓ Weight 값과 theta(추후 bias) 값 랜덤하게 선언
    2. node_propagation
        - ✓ forward propagation 계산
        - ✓ net = X1*W1 + X2*W2 + … + Xn*Wn + theta
    3. node_backpropagation
        - ✓ backward propagation 계산
        - ✓ W = W + (loss*error*input*c)
        - ✓ theta = theta + (loss*c)
- Linear module method 오버라이딩 재정의
    1. init_linear
        - ✓ 초기 생성자
    2. linear_propagation

- ✓ forward propagation 계산

3. linear_backpropagation
   - ✓ backward propagation 계산
   - ✓ new_loss 선언 및 정의
   - ✓ new_loss += new_loss/(입력 채널 수)
   - ✓ return new_loss

- Sigmoid module method 오버라이딩 재정의

1. init_sigmoid
   - ✓ 초기 생성자
2. sigmoid_propagation
   - ✓ forward propagation 계산
   - ✓ result = 1 / (1+exp(-net))
3. sigmoid_backpropagation
   - ✓ backward propagation 계산
   - ✓ loss = loss * (1 - loss)

## 4. main.cpp

- 첨부파일 [과제#2]2017920036양다은₩소스코드₩02.PerceptronLearning₩module.cpp 참고
- module.h 헤더파일 선언
- 과제 업무 상 2차원 input으로 제한되어 필요한 input, output 배열 선언

```
11      float input[4][2] = { {0, 0}, {0, 1}, {1, 0}, {1, 1} }; // X1과 X2
12      float and_output[4] = { 0, 0, 0, 1 };   // AND 연산에 대한 output
13      float or_output[4] = { 0, 1, 1, 1 };    // OR 연산에 대한 output
14      float xor_output[4] = { 0, 1, 1, 0 };   // XOR 연산에 대한 output
15      float* target_output = NULL;
16
17      int layer = 2;  // 2차원 <- 지정
```

- module 신경망 생성 및 초기화

```
17      int layer = 2;  // 2차원 <- 지정
18
19      // 실행할 module 생성 및 초기화
20      Module* module = (Module*)malloc(sizeof(Module) * layer);
21      int in_channels = 2;    // 입력 채널 수 = 2 <- 지정
22      int out_channels = 1;   // 출력 채널 수 = 1 <- 지정
23
24      module[0] = init_linear(in_channels, out_channels);
25      module[1] = init_sigmoid(in_channels);
```

- 실행할 연산(gate) 선택 (1. AND  2. OR  3. XOR)

- 부적절한 입력값, 즉 1,2,3 이외의 값을 받으면 발생할 에러 차단

```
37      int gate;   // 실행할 연산(gate) 선택을 위한 변수 선언
38
39      // 실행할 gate 선택(1. AND   2. OR   3. XOR)
40      cout << endl << "GATE 선택 : 1. AND   2. OR   3. XOR" << endl << "Select : ";
41      while (1) {
42          cin >> gate;
43          if (gate != 1 && gate != 2 && gate != 3) {  // 부적절한 gate 입력하면 발생할 에러 차단
44              cout << "1 / 2 / 3 중 선택하기" << endl;
45          }
46          else break;
47      }
48
49      cout <<"-----------------------------------------------" << endl;
50      if (gate == 1) {
51          cout << "AND gate" << endl;
52          target_output = and_output;
53      }
54      else if(gate == 2) {
55          cout << "OR gate" << endl;
56          target_output = or_output;
57      }
58      else {
59          cout << "XOR gate" << endl;
60          target_output = xor_output;
61      }
62      cout << "-----------------------------------------------" << endl;
```

- Training 시작

**Perceptron training algorithm**

Initialize weights with small random values
repeat
  for each training data $(x_i, t_i)$
    calculate output: $net_j = \sum_{i=0}^{n} x_i w_i$ , $O_i = f(net_j)$

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

    adjust weight: $w_k \leftarrow w_k + \Delta w_k$
until satisfied

- 강의시간에 학습한 알고리즘을 토대로 데이터를 training한다.

- acc_cnt = 4 이 되면, accuracy = 1 이 되어 무한 while 루프문을 빠져나온다.

- propagation과 backpropagation 주기적으로 발생하면서 W1, W2의 값을 업데이트 한다.

- propagation 진행
- module[0] 일 때, linear_propagation 적용
- module[1] 일 때, sigmoid_propagation 적용

```
80          // forward pass compute -> propagation
81          // propagation 진행
82          for (int j = 0; j < layer; j++) {
83              // module[0]일 때, linear_propagation 적용 => net = X1*W1 + X2*W2 + theta
84              // module[1]일 때, sigmoid_propagation 적용 => result = 1 / (1 + exp(-net))
85              // out_channels = 1 이므로 output[0] = result
86              output = module[j].propagation(module[j], output);
87          }
```

- error = target – output

```
            // error = (t - o)
            error[0] = target_output[i] - output[0];
            //cout << "i : " << i << "  error : " << error[0] << endl;

            // E = 1/2(t - o)^2
            error_sum += (error[0] * error[0])/2;
```

- backpropagation 진행
- module[1] 일 때, sigmoid_backpropagation 적용 (역방향 계산이기에 for문이 감소방향으로 반복)
- module[0] 일 때, linear_backpropagation 적용

```
103         // backward pass compute -> backpropagation
104         // backpropagation 진행
105         for (int j = layer - 1; 0 <= j; j--) {
106             // c (= learning rate) = 0.01 <- 지정(실수)
107             // module[1]일 때, sigmoid_backpropagation 적용 => derivative = error * (1 - error)
108             // module[0]일 때, linear_backpropagation 적용 => new_loss = error * input, W = W + new_loss * c
109             error = module[j].backpropagation(module[j], error, 0.01);
110         }
```

- iteration 하면서 Loss(=Error Sum)와 Accuracy(정확도) 값 출력

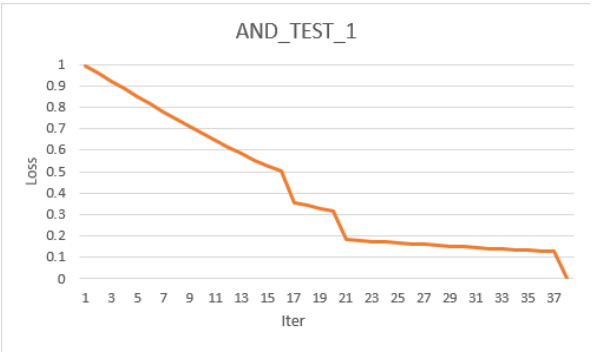
```
116     } // for문 종료
117
118     cout << "Iter: " << iter << "   Loss: " << error_sum <<
119         "   Accuracy: " << acc_cnt / (float)4 << endl;
120
```

# ‹ II. 구현결과 ›

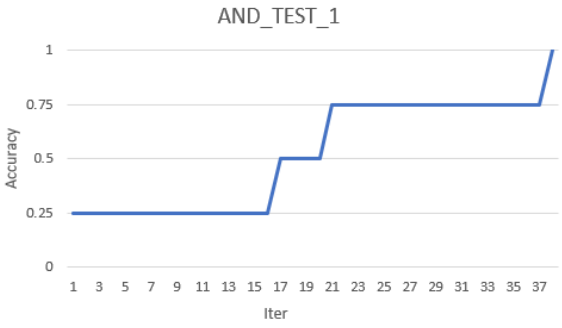
## 1. AND gate 실행결과

- and_test_1

```
Microsoft Visual Studio 디버그 콘솔

GATE 선택 : 1. AND    2. OR    3. XOR
Select : 1
-----------------------------------------------
AND gate
-----------------------------------------------
Iter: 1 Loss: 0.995016  Accuracy: 0.25
Iter: 2 Loss: 0.959201  Accuracy: 0.25
Iter: 3 Loss: 0.922986  Accuracy: 0.25
Iter: 4 Loss: 0.886597  Accuracy: 0.25
Iter: 5 Loss: 0.850258  Accuracy: 0.25
Iter: 6 Loss: 0.814182  Accuracy: 0.25
Iter: 7 Loss: 0.778567  Accuracy: 0.25
Iter: 8 Loss: 0.743592  Accuracy: 0.25
Iter: 9 Loss: 0.709415  Accuracy: 0.25
Iter: 10        Loss: 0.676172  Accuracy: 0.25
Iter: 11        Loss: 0.643973  Accuracy: 0.25
Iter: 12        Loss: 0.612905  Accuracy: 0.25
Iter: 13        Loss: 0.583033  Accuracy: 0.25
Iter: 14        Loss: 0.554402  Accuracy: 0.25
Iter: 15        Loss: 0.527036  Accuracy: 0.25
Iter: 16        Loss: 0.500944  Accuracy: 0.25
Iter: 17        Loss: 0.355553  Accuracy: 0.5
Iter: 18        Loss: 0.342303  Accuracy: 0.5
Iter: 19        Loss: 0.329534  Accuracy: 0.5
Iter: 20        Loss: 0.317243  Accuracy: 0.5
Iter: 21        Loss: 0.183144  Accuracy: 0.75
Iter: 22        Loss: 0.178942  Accuracy: 0.75
Iter: 23        Loss: 0.17483   Accuracy: 0.75
Iter: 24        Loss: 0.170809  Accuracy: 0.75
Iter: 25        Loss: 0.166879  Accuracy: 0.75
Iter: 26        Loss: 0.16304   Accuracy: 0.75
Iter: 27        Loss: 0.15929   Accuracy: 0.75
Iter: 28        Loss: 0.155629  Accuracy: 0.75
Iter: 29        Loss: 0.152057  Accuracy: 0.75
Iter: 30        Loss: 0.148571  Accuracy: 0.75
Iter: 31        Loss: 0.145172  Accuracy: 0.75
Iter: 32        Loss: 0.141857  Accuracy: 0.75
Iter: 33        Loss: 0.138626  Accuracy: 0.75
Iter: 34        Loss: 0.135477  Accuracy: 0.75
Iter: 35        Loss: 0.132408  Accuracy: 0.75
Iter: 36        Loss: 0.129419  Accuracy: 0.75
Iter: 37        Loss: 0.126507  Accuracy: 0.75
Iter: 38        Loss: 0 Accuracy: 1
```



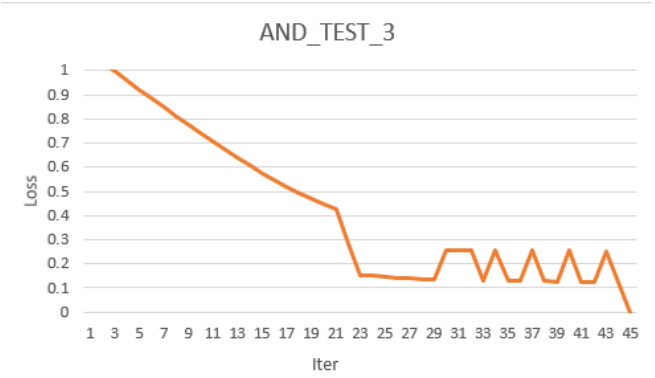

‹Loss Graph›



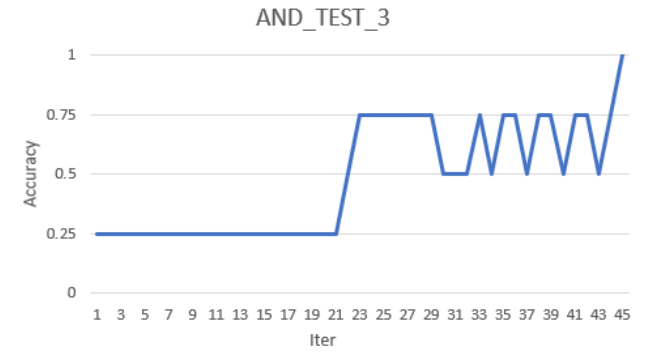

‹Accuracy Graph›

- and_test_3

```
Microsoft Visual Studio 디버그 콘솔

GATE 선택 : 1. AND    2. OR    3. XOR
Select : 1
------------------------------------------------
AND gate
------------------------------------------------
Iter: 1 Loss: 1.06619   Accuracy: 0.25
Iter: 2 Loss: 1.0312    Accuracy: 0.25
Iter: 3 Loss: 0.99531   Accuracy: 0.25
Iter: 4 Loss: 0.958768  Accuracy: 0.25
Iter: 5 Loss: 0.921808  Accuracy: 0.25
Iter: 6 Loss: 0.884676  Accuracy: 0.25
Iter: 7 Loss: 0.847611  Accuracy: 0.25
Iter: 8 Loss: 0.810839  Accuracy: 0.25
Iter: 9 Loss: 0.774574  Accuracy: 0.25
Iter: 10        Loss: 0.739005  Accuracy: 0.25
Iter: 11        Loss: 0.7043    Accuracy: 0.25
Iter: 12        Loss: 0.670597  Accuracy: 0.25
Iter: 13        Loss: 0.638012  Accuracy: 0.25
Iter: 14        Loss: 0.606633  Accuracy: 0.25
Iter: 15        Loss: 0.576521  Accuracy: 0.25
Iter: 16        Loss: 0.547719  Accuracy: 0.25
Iter: 17        Loss: 0.520247  Accuracy: 0.25
Iter: 18        Loss: 0.494108  Accuracy: 0.25
Iter: 19        Loss: 0.46929   Accuracy: 0.25
Iter: 20        Loss: 0.445769  Accuracy: 0.25
Iter: 21        Loss: 0.423513  Accuracy: 0.25
Iter: 22        Loss: 0.283814  Accuracy: 0.5
Iter: 23        Loss: 0.153419  Accuracy: 0.75
Iter: 24        Loss: 0.1499    Accuracy: 0.75
Iter: 25        Loss: 0.146468  Accuracy: 0.75
Iter: 26        Loss: 0.14312   Accuracy: 0.75
Iter: 27        Loss: 0.139857  Accuracy: 0.75
Iter: 28        Loss: 0.136677  Accuracy: 0.75
Iter: 29        Loss: 0.133578  Accuracy: 0.75
Iter: 30        Loss: 0.257205  Accuracy: 0.5
Iter: 31        Loss: 0.256844  Accuracy: 0.5
Iter: 32        Loss: 0.256508  Accuracy: 0.5
Iter: 33        Loss: 0.12862   Accuracy: 0.75
Iter: 34        Loss: 0.255871  Accuracy: 0.5
Iter: 35        Loss: 0.128948  Accuracy: 0.75
Iter: 36        Loss: 0.127682  Accuracy: 0.75
Iter: 37        Loss: 0.254916  Accuracy: 0.5
Iter: 38        Loss: 0.128008  Accuracy: 0.75
Iter: 39        Loss: 0.126747  Accuracy: 0.75
Iter: 40        Loss: 0.253965  Accuracy: 0.5
Iter: 41        Loss: 0.127072  Accuracy: 0.75
Iter: 42        Loss: 0.125815  Accuracy: 0.75
Iter: 43        Loss: 0.253017  Accuracy: 0.5
Iter: 44        Loss: 0.126138  Accuracy: 0.75
Iter: 45        Loss: 0 Accuracy: 1

C:₩AI₩02.PerceptronLearning₩Debug₩02.PerceptronLearn
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -
사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



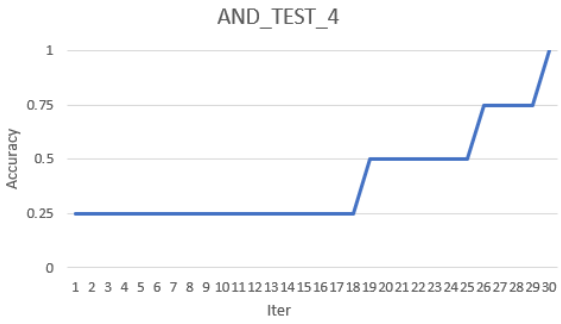

〈Loss Graph〉



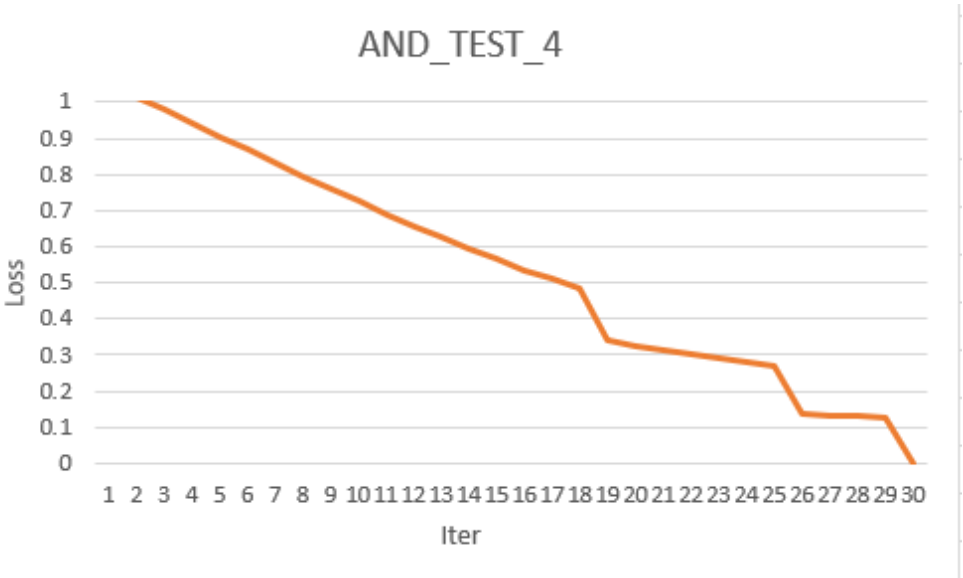

〈Accuracy Graph〉

- and_test_4

```
Microsoft Visual Studio 디버그 콘솔

GATE 선택 : 1. AND    2. OR    3. XOR
Select : 1
------------------------------------------------
AND gate
------------------------------------------------
Iter: 1 Loss: 1.04927   Accuracy: 0.25
Iter: 2 Loss: 1.01418   Accuracy: 0.25
Iter: 3 Loss: 0.978311  Accuracy: 0.25
Iter: 4 Loss: 0.941883  Accuracy: 0.25
Iter: 5 Loss: 0.905133  Accuracy: 0.25
Iter: 6 Loss: 0.868294  Accuracy: 0.25
Iter: 7 Loss: 0.831595  Accuracy: 0.25
Iter: 8 Loss: 0.795251  Accuracy: 0.25
Iter: 9 Loss: 0.759462  Accuracy: 0.25
Iter: 10        Loss: 0.724407  Accuracy: 0.25
Iter: 11        Loss: 0.690241  Accuracy: 0.25
Iter: 12        Loss: 0.657094  Accuracy: 0.25
Iter: 13        Loss: 0.625069  Accuracy: 0.25
Iter: 14        Loss: 0.594249  Accuracy: 0.25
Iter: 15        Loss: 0.56469   Accuracy: 0.25
Iter: 16        Loss: 0.536427  Accuracy: 0.25
Iter: 17        Loss: 0.509476  Accuracy: 0.25
Iter: 18        Loss: 0.48384   Accuracy: 0.25
Iter: 19        Loss: 0.340018  Accuracy: 0.5
Iter: 20        Loss: 0.327093  Accuracy: 0.5
Iter: 21        Loss: 0.314665  Accuracy: 0.5
Iter: 22        Loss: 0.302728  Accuracy: 0.5
Iter: 23        Loss: 0.291273  Accuracy: 0.5
Iter: 24        Loss: 0.280289  Accuracy: 0.5
Iter: 25        Loss: 0.269763  Accuracy: 0.5
Iter: 26        Loss: 0.136024  Accuracy: 0.75
Iter: 27        Loss: 0.132941  Accuracy: 0.75
Iter: 28        Loss: 0.129938  Accuracy: 0.75
Iter: 29        Loss: 0.127013  Accuracy: 0.75
Iter: 30        Loss: 0 Accuracy: 1

C:₩AI₩02.PerceptronLearning₩Debug₩02.PerceptronLearning.e
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵
사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```



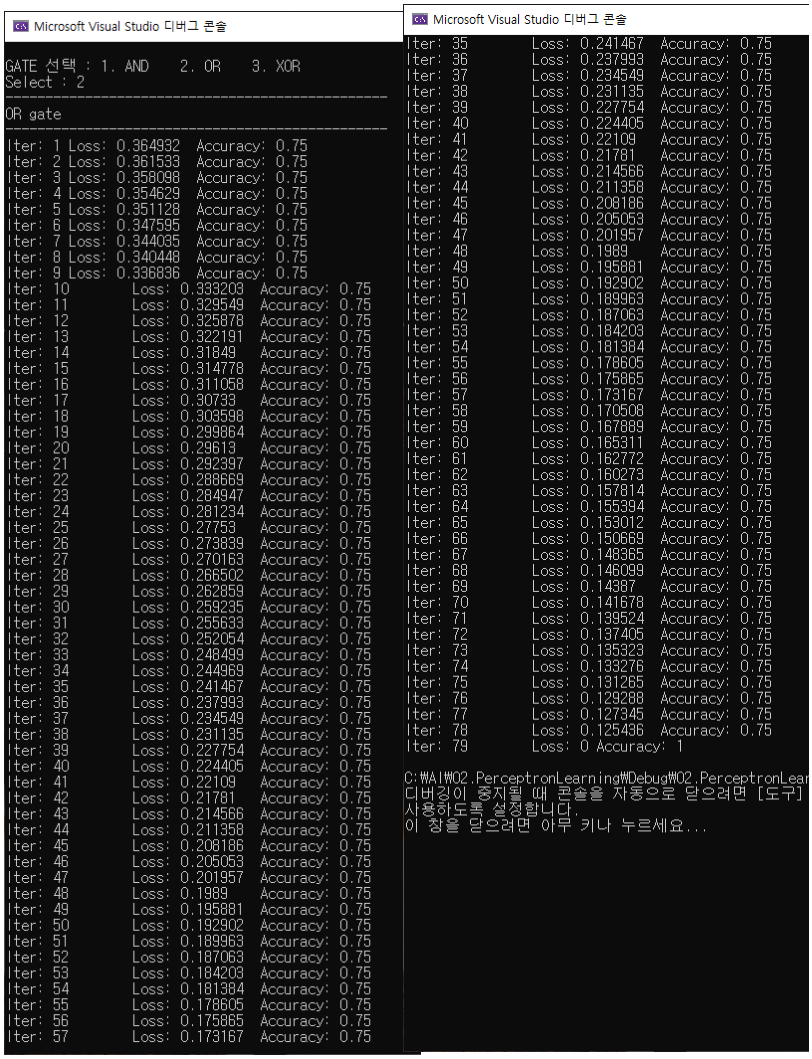

〈Accuracy Graph〉

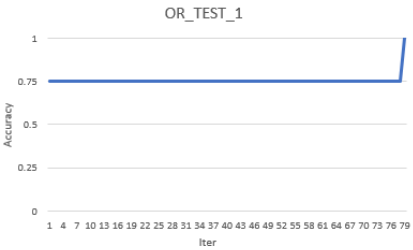


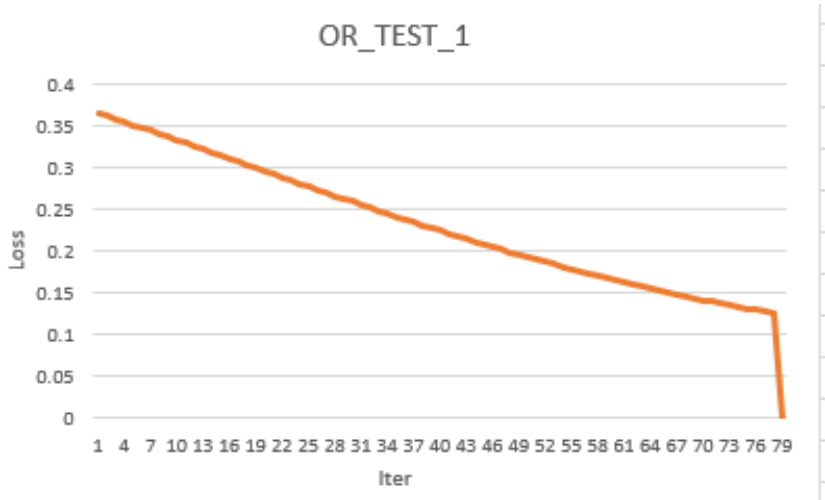

〈Loss Graph〉

## 2. OR gate 실행결과

- or_test_1



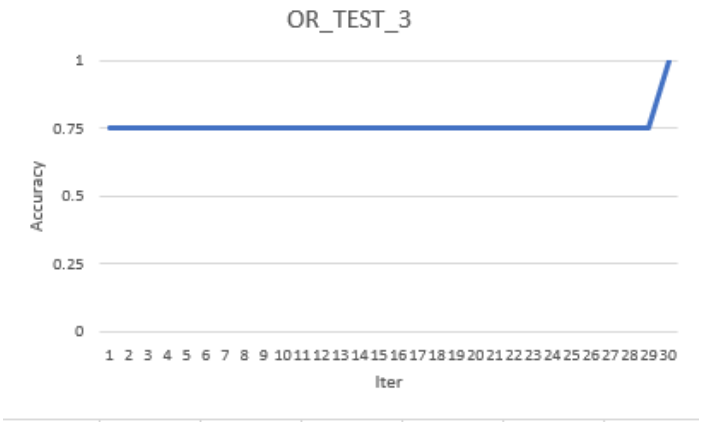



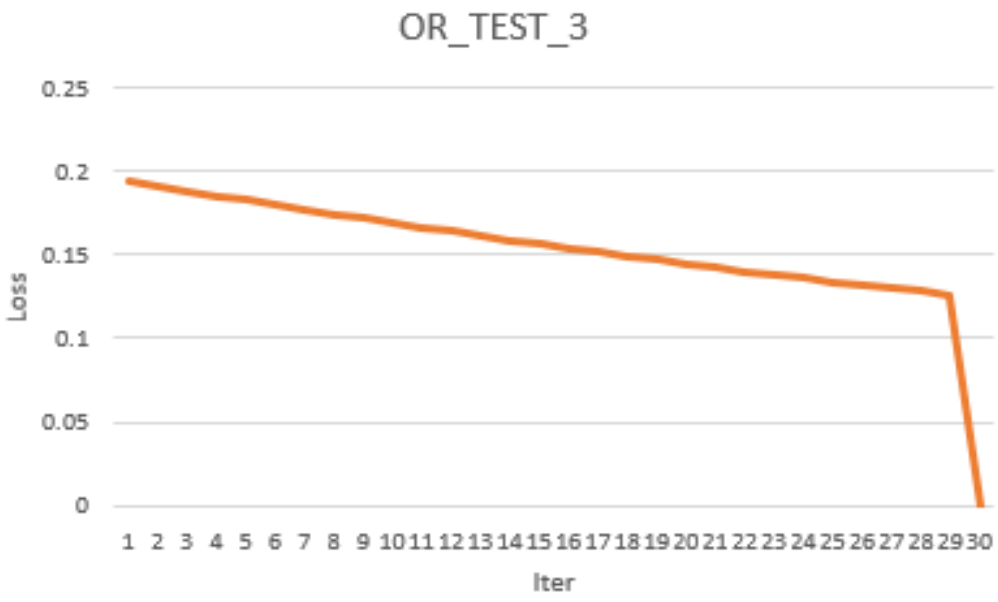

〈Loss Graph〉

- or_test_3

```
Microsoft Visual Studio 디버그 콘솔

GATE 선택 : 1. AND    2. OR    3. XOR
Select : 2
-----------------------------------------------
OR gate
-----------------------------------------------
Iter: 1 Loss: 0.194152  Accuracy: 0.75
Iter: 2 Loss: 0.191196  Accuracy: 0.75
Iter: 3 Loss: 0.188279  Accuracy: 0.75
Iter: 4 Loss: 0.185403  Accuracy: 0.75
Iter: 5 Loss: 0.182566  Accuracy: 0.75
Iter: 6 Loss: 0.17977   Accuracy: 0.75
Iter: 7 Loss: 0.177014  Accuracy: 0.75
Iter: 8 Loss: 0.174298  Accuracy: 0.75
Iter: 9 Loss: 0.171623  Accuracy: 0.75
Iter: 10        Loss: 0.168987  Accuracy: 0.75
Iter: 11        Loss: 0.166392  Accuracy: 0.75
Iter: 12        Loss: 0.163837  Accuracy: 0.75
Iter: 13        Loss: 0.161321  Accuracy: 0.75
Iter: 14        Loss: 0.158845  Accuracy: 0.75
Iter: 15        Loss: 0.156408  Accuracy: 0.75
Iter: 16        Loss: 0.15401   Accuracy: 0.75
Iter: 17        Loss: 0.151651  Accuracy: 0.75
Iter: 18        Loss: 0.149331  Accuracy: 0.75
Iter: 19        Loss: 0.147049  Accuracy: 0.75
Iter: 20        Loss: 0.144804  Accuracy: 0.75
Iter: 21        Loss: 0.142597  Accuracy: 0.75
Iter: 22        Loss: 0.140427  Accuracy: 0.75
Iter: 23        Loss: 0.138293  Accuracy: 0.75
Iter: 24        Loss: 0.136196  Accuracy: 0.75
Iter: 25        Loss: 0.134134  Accuracy: 0.75
Iter: 26        Loss: 0.132108  Accuracy: 0.75
Iter: 27        Loss: 0.130116  Accuracy: 0.75
Iter: 28        Loss: 0.128159  Accuracy: 0.75
Iter: 29        Loss: 0.126236  Accuracy: 0.75
Iter: 30        Loss: 0 Accuracy: 1
```



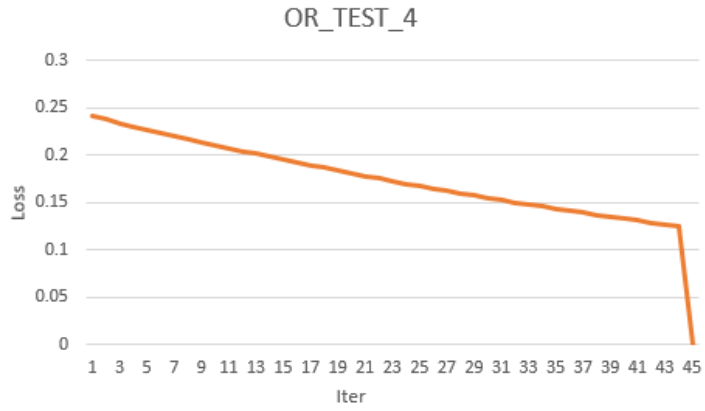

〈Accuracy Graph〉



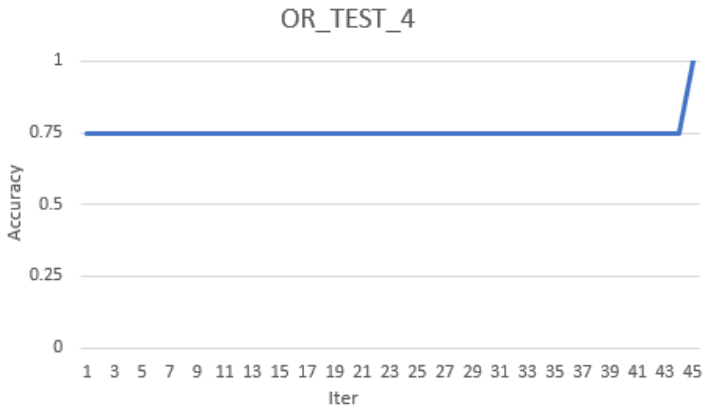

〈Loss Graph〉

- or_test_4

```
Microsoft Visual Studio 디버그 콘솔

GATE 선택 : 1. AND    2. OR    3. XOR
Select : 2
-----------------------------------------------
OR gate
-----------------------------------------------
Iter: 1 Loss: 0.24079   Accuracy: 0.75
Iter: 2 Loss: 0.237322  Accuracy: 0.75
Iter: 3 Loss: 0.233884  Accuracy: 0.75
Iter: 4 Loss: 0.230477  Accuracy: 0.75
Iter: 5 Loss: 0.227101  Accuracy: 0.75
Iter: 6 Loss: 0.223759  Accuracy: 0.75
Iter: 7 Loss: 0.220451  Accuracy: 0.75
Iter: 8 Loss: 0.217178  Accuracy: 0.75
Iter: 9 Loss: 0.213941  Accuracy: 0.75
Iter: 10        Loss: 0.210739  Accuracy: 0.75
Iter: 11        Loss: 0.207575  Accuracy: 0.75
Iter: 12        Loss: 0.204449  Accuracy: 0.75
Iter: 13        Loss: 0.201361  Accuracy: 0.75
Iter: 14        Loss: 0.198311  Accuracy: 0.75
Iter: 15        Loss: 0.1953    Accuracy: 0.75
Iter: 16        Loss: 0.192329  Accuracy: 0.75
Iter: 17        Loss: 0.189397  Accuracy: 0.75
Iter: 18        Loss: 0.186505  Accuracy: 0.75
Iter: 19        Loss: 0.183653  Accuracy: 0.75
Iter: 20        Loss: 0.180841  Accuracy: 0.75
Iter: 21        Loss: 0.17807   Accuracy: 0.75
Iter: 22        Loss: 0.175339  Accuracy: 0.75
Iter: 23        Loss: 0.172648  Accuracy: 0.75
Iter: 24        Loss: 0.169997  Accuracy: 0.75
Iter: 25        Loss: 0.167386  Accuracy: 0.75
Iter: 26        Loss: 0.164815  Accuracy: 0.75
Iter: 27        Loss: 0.162284  Accuracy: 0.75
Iter: 28        Loss: 0.159793  Accuracy: 0.75
Iter: 29        Loss: 0.157341  Accuracy: 0.75
Iter: 30        Loss: 0.154928  Accuracy: 0.75
Iter: 31        Loss: 0.152555  Accuracy: 0.75
Iter: 32        Loss: 0.150219  Accuracy: 0.75
Iter: 33        Loss: 0.147922  Accuracy: 0.75
Iter: 34        Loss: 0.145663  Accuracy: 0.75
Iter: 35        Loss: 0.143442  Accuracy: 0.75
Iter: 36        Loss: 0.141257  Accuracy: 0.75
Iter: 37        Loss: 0.13911   Accuracy: 0.75
Iter: 38        Loss: 0.136998  Accuracy: 0.75
Iter: 39        Loss: 0.134923  Accuracy: 0.75
Iter: 40        Loss: 0.132883  Accuracy: 0.75
Iter: 41        Loss: 0.130878  Accuracy: 0.75
Iter: 42        Loss: 0.128908  Accuracy: 0.75
Iter: 43        Loss: 0.126972  Accuracy: 0.75
Iter: 44        Loss: 0.12507   Accuracy: 0.75
Iter: 45        Loss: 0 Accuracy: 1
```




〈Loss Graph〉



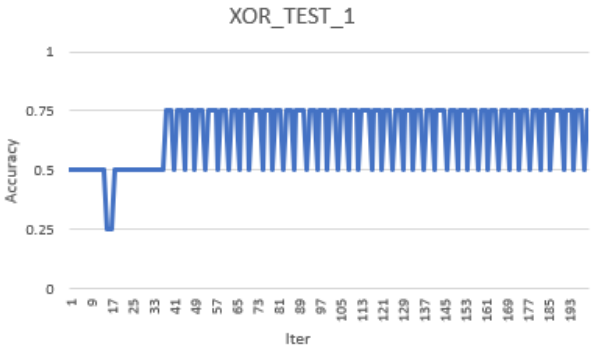

〈Accuracy Graph〉

```
선택 C:₩AI₩02.PerceptronLearning₩Debug₩02.PerceptronLearni

GATE 선택 : 1. AND    2. OR    3. XOR
Select : 3
----------------------------------------------
XOR gate
----------------------------------------------
Iter: 1 Loss: 0.696522  Accuracy: 0.5
Iter: 2 Loss: 0.680651  Accuracy: 0.5
Iter: 3 Loss: 0.664431  Accuracy: 0.5
Iter: 4 Loss: 0.647902  Accuracy: 0.5
Iter: 5 Loss: 0.631109  Accuracy: 0.5
Iter: 6 Loss: 0.614095  Accuracy: 0.5
Iter: 7 Loss: 0.596911  Accuracy: 0.5
Iter: 8 Loss: 0.579608  Accuracy: 0.5
Iter: 9 Loss: 0.562238  Accuracy: 0.5
Iter: 10        Loss: 0.544855  Accuracy: 0.5
Iter: 11        Loss: 0.527513  Accuracy: 0.5
Iter: 12        Loss: 0.510264  Accuracy: 0.5
Iter: 13        Loss: 0.493159  Accuracy: 0.5
Iter: 14        Loss: 0.476248  Accuracy: 0.5
Iter: 15        Loss: 0.459576  Accuracy: 0.5
Iter: 16        Loss: 0.443187  Accuracy: 0.5
Iter: 17        Loss: 0.427119  Accuracy: 0.5
Iter: 18        Loss: 0.411406  Accuracy: 0.5
Iter: 19        Loss: 0.396079  Accuracy: 0.5
Iter: 20        Loss: 0.381164  Accuracy: 0.5
Iter: 21        Loss: 0.36668   Accuracy: 0.5
Iter: 22        Loss: 0.352645  Accuracy: 0.5
Iter: 23        Loss: 0.466699  Accuracy: 0.25
Iter: 24        Loss: 0.338805  Accuracy: 0.5
Iter: 25        Loss: 0.336754  Accuracy: 0.5
Iter: 26        Loss: 0.334789  Accuracy: 0.5
Iter: 27        Loss: 0.332909  Accuracy: 0.5
Iter: 28        Loss: 0.331114  Accuracy: 0.5
Iter: 29        Loss: 0.329401  Accuracy: 0.5
Iter: 30        Loss: 0.32777   Accuracy: 0.5
Iter: 31        Loss: 0.326218  Accuracy: 0.5
Iter: 32        Loss: 0.324744  Accuracy: 0.5
Iter: 33        Loss: 0.323347  Accuracy: 0.5
Iter: 34        Loss: 0.322023  Accuracy: 0.5
Iter: 35        Loss: 0.32077   Accuracy: 0.5
Iter: 36        Loss: 0.319587  Accuracy: 0.5
Iter: 37        Loss: 0.318472  Accuracy: 0.5
Iter: 38        Loss: 0.317421  Accuracy: 0.5
Iter: 39        Loss: 0.316433  Accuracy: 0.5
Iter: 40        Loss: 0.315505  Accuracy: 0.5
Iter: 41        Loss: 0.314636  Accuracy: 0.5
Iter: 42        Loss: 0.313823  Accuracy: 0.5
Iter: 43        Loss: 0.313063  Accuracy: 0.5
Iter: 44        Loss: 0.312355  Accuracy: 0.5
Iter: 45        Loss: 0.311696  Accuracy: 0.5
Iter: 46        Loss: 0.188861  Accuracy: 0.75
Iter: 47        Loss: 0.187828  Accuracy: 0.75
Iter: 48        Loss: 0.186793  Accuracy: 0.75
Iter: 49        Loss: 0.185758  Accuracy: 0.75
Iter: 50        Loss: 0.310499  Accuracy: 0.5
Iter: 51        Loss: 0.186964  Accuracy: 0.75
Iter: 52        Loss: 0.185929  Accuracy: 0.75
Iter: 53        Loss: 0.184894  Accuracy: 0.75
Iter: 54        Loss: 0.30944   Accuracy: 0.5
Iter: 55        Loss: 0.186092  Accuracy: 0.75
Iter: 56        Loss: 0.185056  Accuracy: 0.75
Iter: 57        Loss: 0.184021  Accuracy: 0.75
```

## 3. XOR gate 실행결과

- xor_test_1

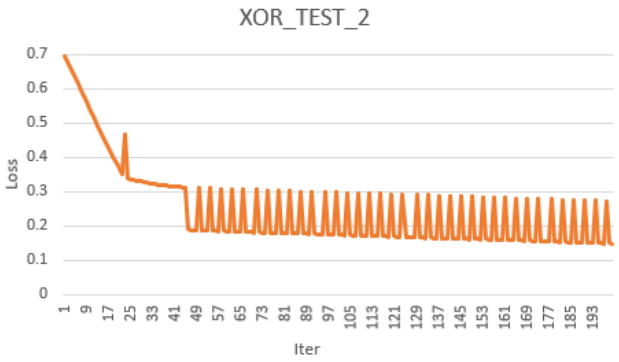

〈Loss Graph〉



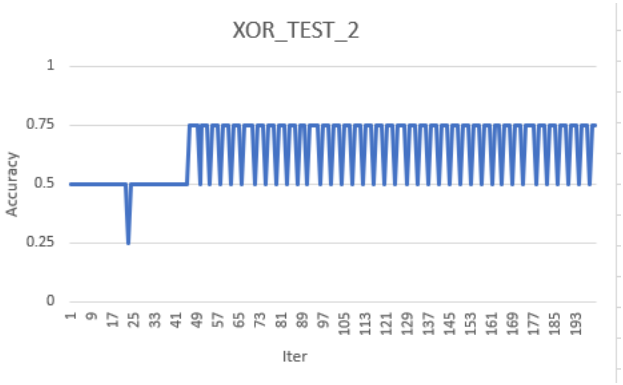

〈Accuracy Graph〉

- xor_test_2

```
선택 C:₩AI₩02.PerceptronLearning₩Debug₩02.PerceptronLearning.e

GATE 선택 : 1. AND    2. OR    3. XOR
Select : 3
----------------------------------------------

XOR gate
----------------------------------------------

Iter: 1 Loss: 0.585813  Accuracy: 0.5
Iter: 2 Loss: 0.568281  Accuracy: 0.5
Iter: 3 Loss: 0.550724  Accuracy: 0.5
Iter: 4 Loss: 0.533198  Accuracy: 0.5
Iter: 5 Loss: 0.515758  Accuracy: 0.5
Iter: 6 Loss: 0.498458  Accuracy: 0.5
Iter: 7 Loss: 0.481348  Accuracy: 0.5
Iter: 8 Loss: 0.464478  Accuracy: 0.5
Iter: 9 Loss: 0.447891  Accuracy: 0.5
Iter: 10        Loss: 0.431628  Accuracy: 0.5
Iter: 11        Loss: 0.415725  Accuracy: 0.5
Iter: 12        Loss: 0.400213  Accuracy: 0.5
Iter: 13        Loss: 0.385118  Accuracy: 0.5
Iter: 14        Loss: 0.370462  Accuracy: 0.5
Iter: 15        Loss: 0.486928  Accuracy: 0.25
Iter: 16        Loss: 0.480204  Accuracy: 0.25
Iter: 17        Loss: 0.473826  Accuracy: 0.25
Iter: 18        Loss: 0.344018  Accuracy: 0.5
Iter: 19        Loss: 0.342188  Accuracy: 0.5
Iter: 20        Loss: 0.340439  Accuracy: 0.5
Iter: 21        Loss: 0.338769  Accuracy: 0.5
Iter: 22        Loss: 0.337177  Accuracy: 0.5
Iter: 23        Loss: 0.335661  Accuracy: 0.5
Iter: 24        Loss: 0.33422   Accuracy: 0.5
Iter: 25        Loss: 0.332853  Accuracy: 0.5
Iter: 26        Loss: 0.331556  Accuracy: 0.5
Iter: 27        Loss: 0.330328  Accuracy: 0.5
Iter: 28        Loss: 0.329168  Accuracy: 0.5
Iter: 29        Loss: 0.328073  Accuracy: 0.5
Iter: 30        Loss: 0.327041  Accuracy: 0.5
Iter: 31        Loss: 0.32607   Accuracy: 0.5
Iter: 32        Loss: 0.325158  Accuracy: 0.5
Iter: 33        Loss: 0.324303  Accuracy: 0.5
Iter: 34        Loss: 0.323502  Accuracy: 0.5
Iter: 35        Loss: 0.322753  Accuracy: 0.5
Iter: 36        Loss: 0.322056  Accuracy: 0.5
Iter: 37        Loss: 0.321406  Accuracy: 0.5
Iter: 38        Loss: 0.197713  Accuracy: 0.75
Iter: 39        Loss: 0.196685  Accuracy: 0.75
Iter: 40        Loss: 0.195657  Accuracy: 0.75
Iter: 41        Loss: 0.320347  Accuracy: 0.5
Iter: 42        Loss: 0.196906  Accuracy: 0.75
Iter: 43        Loss: 0.195878  Accuracy: 0.75
Iter: 44        Loss: 0.194849  Accuracy: 0.75
Iter: 45        Loss: 0.319288  Accuracy: 0.5
Iter: 46        Loss: 0.19609   Accuracy: 0.75
Iter: 47        Loss: 0.195061  Accuracy: 0.75
Iter: 48        Loss: 0.194031  Accuracy: 0.75
Iter: 49        Loss: 0.318231  Accuracy: 0.5
Iter: 50        Loss: 0.195265  Accuracy: 0.75
Iter: 51        Loss: 0.194235  Accuracy: 0.75
Iter: 52        Loss: 0.193205  Accuracy: 0.75
Iter: 53        Loss: 0.317174  Accuracy: 0.5
Iter: 54        Loss: 0.19443   Accuracy: 0.75
Iter: 55        Loss: 0.1934    Accuracy: 0.75
Iter: 56        Loss: 0.192369  Accuracy: 0.75
Iter: 57        Loss: 0.191337  Accuracy: 0.75
```




〈Loss Graph〉

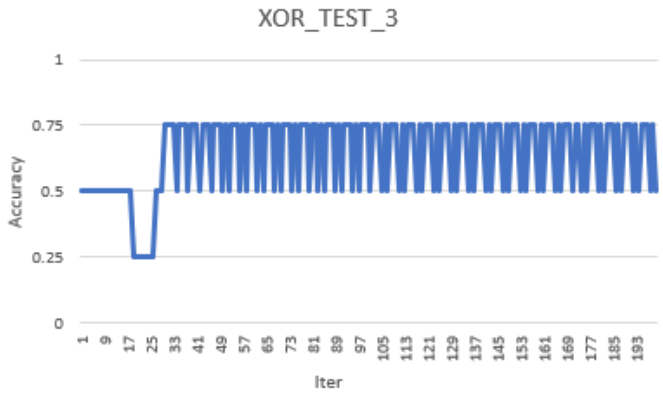



〈Accuracy Graph〉

- xor_test_3

```
선택 C:₩AI₩02.PerceptronLearning₩Debug₩02.PerceptronLearnin
GATE 선택 : 1. AND    2. OR    3. XOR
Select : 3
-----------------------------------------------
XOR gate
-----------------------------------------------
Iter: 1 Loss: 0.628578  Accuracy: 0.5
Iter: 2 Loss: 0.610385  Accuracy: 0.5
Iter: 3 Loss: 0.592066  Accuracy: 0.5
Iter: 4 Loss: 0.573691  Accuracy: 0.5
Iter: 5 Loss: 0.555327  Accuracy: 0.5
Iter: 6 Loss: 0.537042  Accuracy: 0.5
Iter: 7 Loss: 0.5189    Accuracy: 0.5
Iter: 8 Loss: 0.500962  Accuracy: 0.5
Iter: 9 Loss: 0.483284  Accuracy: 0.5
Iter: 10        Loss: 0.465919  Accuracy: 0.5
Iter: 11        Loss: 0.448911  Accuracy: 0.5
Iter: 12        Loss: 0.432302  Accuracy: 0.5
Iter: 13        Loss: 0.416126  Accuracy: 0.5
Iter: 14        Loss: 0.40041   Accuracy: 0.5
Iter: 15        Loss: 0.385177  Accuracy: 0.5
Iter: 16        Loss: 0.370442  Accuracy: 0.5
Iter: 17        Loss: 0.356219  Accuracy: 0.5
Iter: 18        Loss: 0.342512  Accuracy: 0.5
Iter: 19        Loss: 0.457444  Accuracy: 0.25
Iter: 20        Loss: 0.451266  Accuracy: 0.25
Iter: 21        Loss: 0.445454  Accuracy: 0.25
Iter: 22        Loss: 0.439993  Accuracy: 0.25
Iter: 23        Loss: 0.434869  Accuracy: 0.25
Iter: 24        Loss: 0.430066  Accuracy: 0.25
Iter: 25        Loss: 0.42557   Accuracy: 0.25
Iter: 26        Loss: 0.421365  Accuracy: 0.25
Iter: 27        Loss: 0.292587  Accuracy: 0.5
Iter: 28        Loss: 0.291802  Accuracy: 0.5
Iter: 29        Loss: 0.29107   Accuracy: 0.5
Iter: 30        Loss: 0.168208  Accuracy: 0.75
Iter: 31        Loss: 0.167176  Accuracy: 0.75
Iter: 32        Loss: 0.166146  Accuracy: 0.75
Iter: 33        Loss: 0.165116  Accuracy: 0.75
Iter: 34        Loss: 0.289918  Accuracy: 0.5
Iter: 35        Loss: 0.16623   Accuracy: 0.75
Iter: 36        Loss: 0.1652    Accuracy: 0.75
Iter: 37        Loss: 0.164171  Accuracy: 0.75
Iter: 38        Loss: 0.288876  Accuracy: 0.5
Iter: 39        Loss: 0.16528   Accuracy: 0.75
Iter: 40        Loss: 0.164251  Accuracy: 0.75
Iter: 41        Loss: 0.163223  Accuracy: 0.75
Iter: 42        Loss: 0.287835  Accuracy: 0.5
Iter: 43        Loss: 0.164326  Accuracy: 0.75
Iter: 44        Loss: 0.163298  Accuracy: 0.75
Iter: 45        Loss: 0.162271  Accuracy: 0.75
Iter: 46        Loss: 0.286796  Accuracy: 0.5
Iter: 47        Loss: 0.163368  Accuracy: 0.75
Iter: 48        Loss: 0.162341  Accuracy: 0.75
Iter: 49        Loss: 0.161315  Accuracy: 0.75
Iter: 50        Loss: 0.287882  Accuracy: 0.5
Iter: 51        Loss: 0.16136   Accuracy: 0.75
Iter: 52        Loss: 0.285764  Accuracy: 0.5
Iter: 53        Loss: 0.162452  Accuracy: 0.75
Iter: 54        Loss: 0.161426  Accuracy: 0.75
Iter: 55        Loss: 0.160401  Accuracy: 0.75
Iter: 56        Loss: 0.286885  Accuracy: 0.5
Iter: 57        Loss: 0.160443  Accuracy: 0.75
```

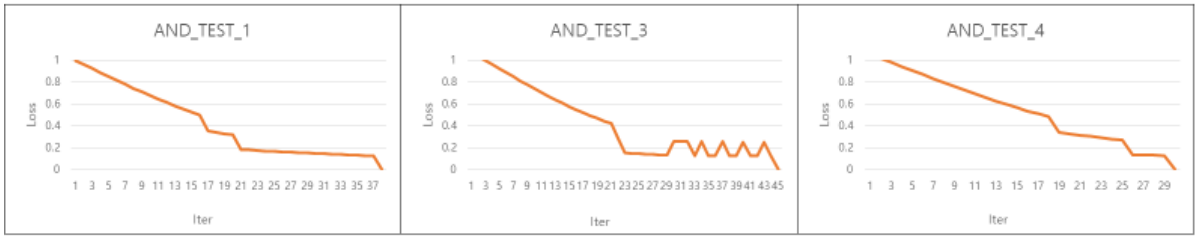

〈Loss Graph〉




〈Accuracy Graph〉

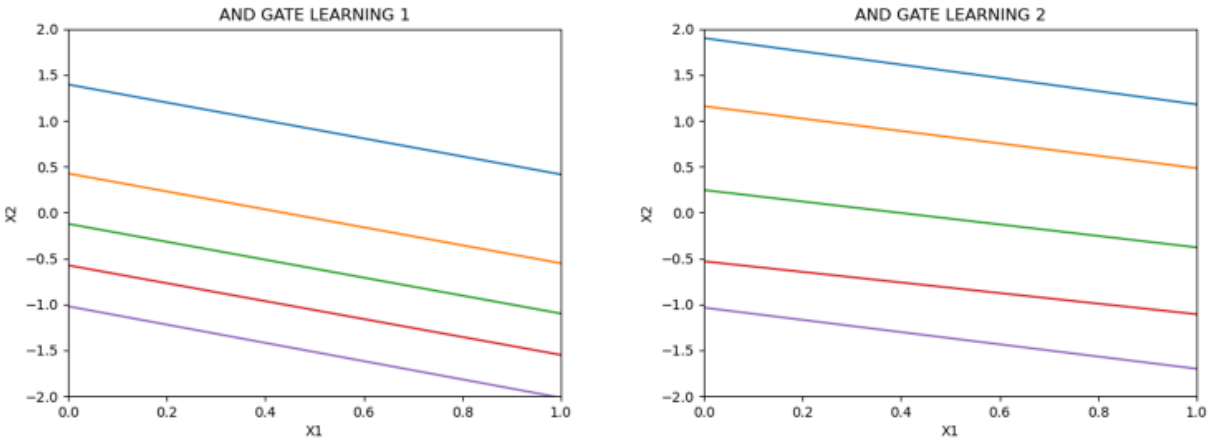
## 4. 결론 및 한계

AND 게이트의 경우, 정상적으로 동작하였다. 실행횟수는 적게는 약 20회에서 많게는 100회 정도로 측정되었으나 100%의 정확성을 보인다. Loss 그래프를 보면 다른 게이트와 비교적으로 완만하게 곡선이 표현되었다. 또한, learning 그래프를 살펴보면 weight 값이 예측되는 그래프와 유사하게 학습됨을 알 수 있다.

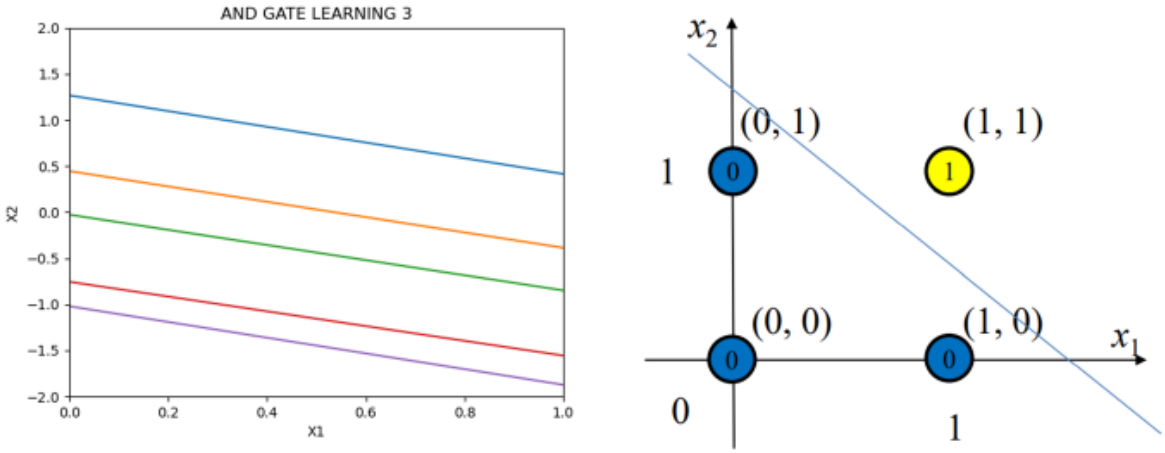



〈AND 게이트 Loss 그래프〉

위의 Loss 그래프와 일치하는 데이터는 아니지만, 같은 코드로 프로그램을 돌려 출력된 W1과 W2 그리고 theta 데이터를 가공하여 python 언어를 활용하여 그린 AND 연산 Learning 과정 그래프이다. 관련 코드는 첨부파일 중 그래프코드(Learning)을 참고한다. 인공지능 강의시간에 학습한 그래프와 유사하게 직선이 움직임을 확인할 수 있다. (1 , 1)과 (0, 1), (1, 0) 사이로 직선이 움직이는 동향을 보인다. 대체로 유사한 기울기로 표현된다. 참고한 이미지는 강의자료를 참고하였음을 출처한다.




〈AND 게이트 Learning 그래프 1〉

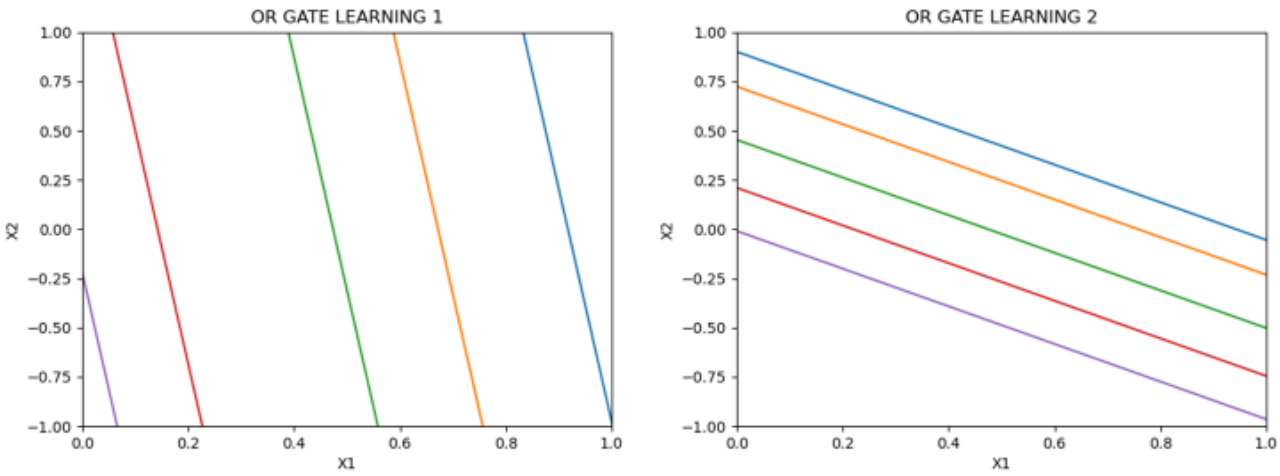



〈AND 게이트 Learning 그래프 2〉

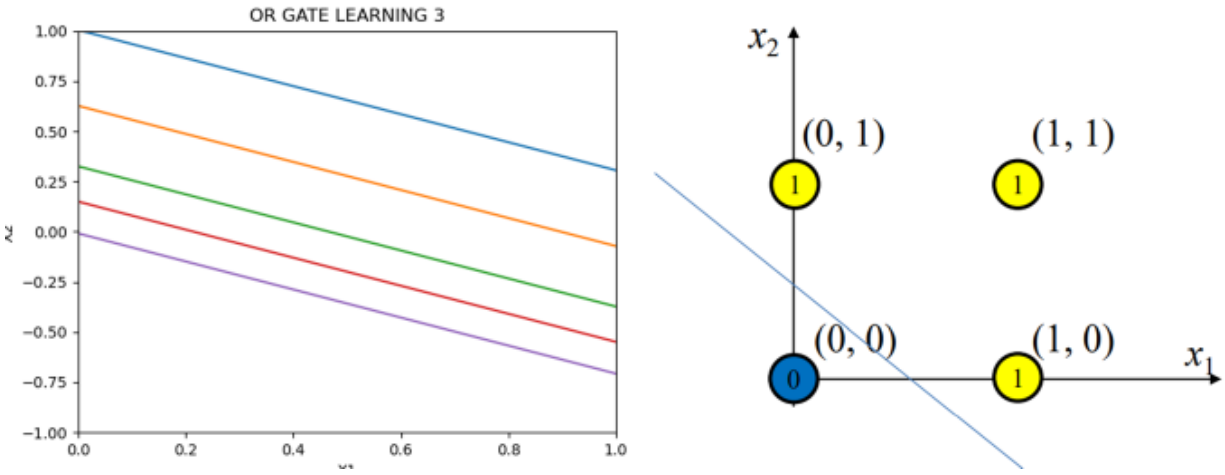
OR 게이트의 경우, AND 게이트와 유사하게 정상적으로 동작하였다. 실행횟수는 적게는 약 20회에서 많게는 200회 정도로 측정되었고, 100%의 정확성을 보인다. Loss 그래프를 살펴보면 loss 값이 약 0.12~0.13 전까지 완만하게 표현되었다가, 그 후 급격한 경사로 0에 도달함을 알 수 있다. 이러한 이유는 target 값과 계산한 결과값이 같으면 loss 자체를 0으로 표현했기 때문으로 추정한다. 따라서 error_sum에 target 값과 계산 값이 달라도 0이 더해지기 때문에 이와 같은 그래프로 구현된다. 또한, learning 그래프를 살펴보도록 한다.


〈OR 게이트 Loss 그래프〉

위의 Loss 그래프와 일치하는 데이터는 아니지만, 같은 코드로 프로그램을 돌려 출력된 W1과 W2 그리고 theta 데이터를 가공하여 python 언어를 활용하여 그린 OR 연산 Learning 과정 그래프이다. 관련코드는 첨부파일 중 그래프코드(Learning)을 참고한다. 인공지능 강의시간에 학습한 그래프와 유사하게 직선이 움직임을 확인할 수 있다. 각각의 데이터와 초기 weight, theta 값에 따라 기울기가 다른 직선이 보여진다. 그러나 통일적으로 (0 , 0)과 (0, 1), (1, 0) 사이로 직선이 움직이는 동향을 보인다. 참고한 이미지는 강의자료를 참고하였음을 출처한다.

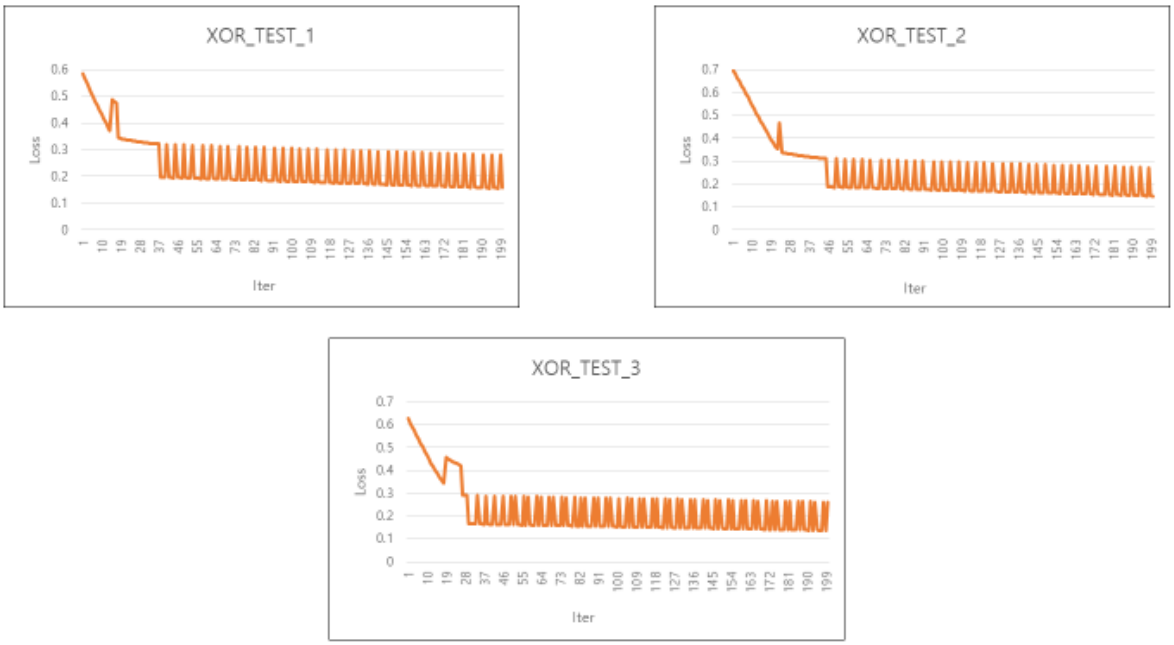



〈OR 게이트 Learning 그래프 1〉



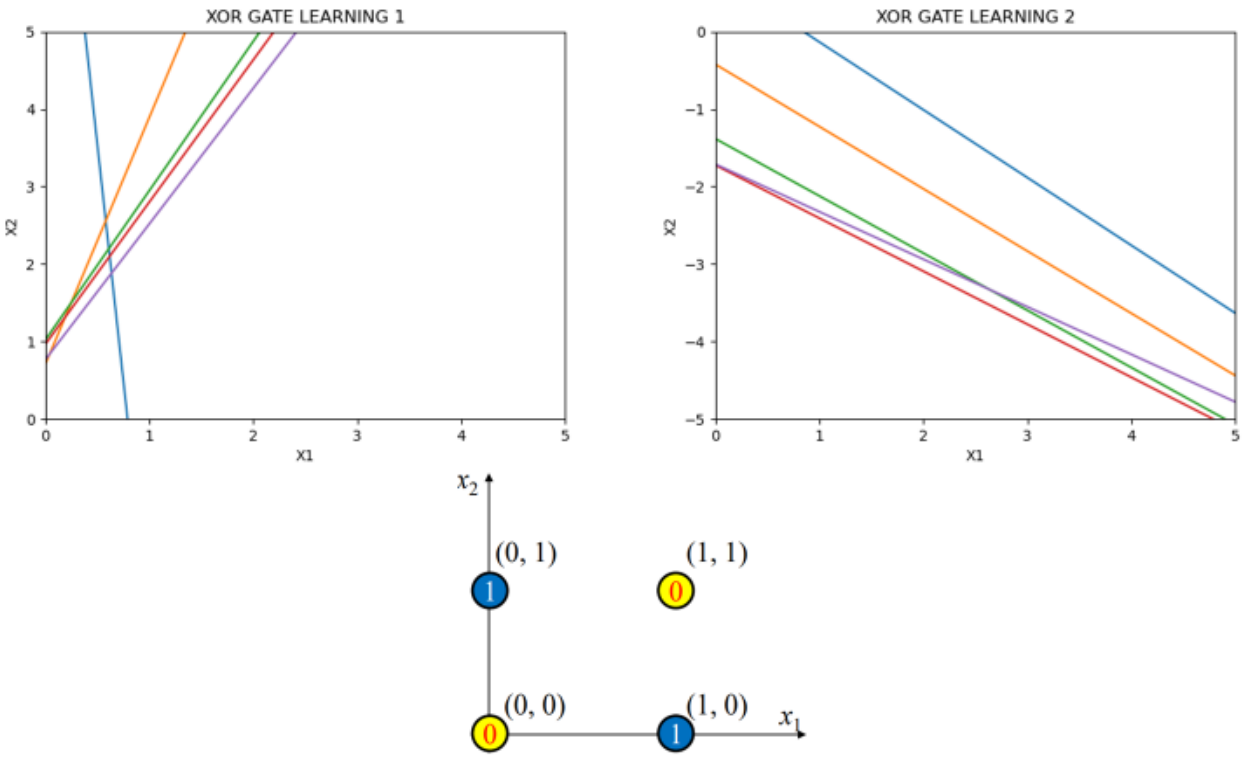

〈OR 게이트 Learning 그래프 2〉

XOR 게이트의 경우, AND 게이트와 OR 게이트와 달리 실행을 동작하였으나 강제종료 했다. 초반에는 다소 급격하게 Loss 값을 감소해 나갔으나 진동이 보이는 과정에는 약 50%~ 75%의 정확성이 나타났다. 이를 통하여 1 layer 퍼셉트론에서 XOR 게이트는 적절한 weight, theta 값을 구할 수 없을 알 수 있다. 정확성과 Loss 그래프의 진동으로 구현됨으로서 XOR 게이트를 수행하기 위해서는 비선형 함수가 필요하다. 즉 단층 신경망으로 분류하기에 불가능하다는 의미이다. 또한, learning 그래프를 살펴보도록 한다.




〈XOR 게이트 Loss 그래프〉

위의 Loss 그래프와 일치하는 데이터는 아니지만, 같은 코드로 프로그램을 돌려 출력된 W1과 W2 그리고 theta 데이터를 가공하여 python 언어를 활용하여 그린 XOR 연산 Learning 과정 그래프이다. 관련코드는 첨부파일 중 그래프코드(Learning)을 참고한다. 각각의 데이터와 초기 weight, theta 값에 따라 기울기가 다른 직선의 동향이 보여진다. 가장 아래 XOR 연산표를 보면 신경망은 input 값을 적절히 분류할 수 없는 것이다. 신경망이 분류가능한 요소는 최대 3개이다. 학습과정에서 정확히 분류하지 못한 나머지 1개를 분류하려고 직선이 이동하고, 나머지 1개가 올바르게 분류되면 이미 분류된 데이터 중에서 분류범위를 벗어나게 된다. 이러한 반복과정이 Loss 그래프의 진동으로 표현되며 Loss 가 0에 도달할 수 없을 보여준다. 이미지는 강의자료를 참고하였음을 출처한다.




〈XOR 게이트 Learning 그래프〉

## ‹ III. 출처 및 참고자료›

# 원리 참고

- https://wikidocs.net/37406
- https://ko.d2l.ai/chapter_deep-learning-basics/backprop.html
- https://forensics.tistory.com/26#comment14568428
- https://leedakyeong.tistory.com/entry/%EB%B0%91%EB%B0%94%EB%8B%A5%EB%B6%80%ED%84%B0-%EC%8B%9C%EC%9E%91%ED%95%98%EB%8A%94-%EB%94%A5%EB%9F%AC%EB%8B%9D-%ED%8D%BC%EC%85%89%ED%8A%B8%EB%A1%A0%EC%9D%98-%ED%95%9C%EA%B3%84-XOR-%EA%B2%8C%EC%9D%B4%ED%8A%B8-limit-of-perceptron-XOR-gate?category=845638

# 코딩 참고

- https://boycoding.tistory.com/144?category=1006674
- https://codetorial.net/matplotlib/axis_range.html
- https://bebutae.tistory.com/188

# 데이터 처리 참고

- https://m.blog.naver.com/top-dream/221054393897

# 이미지 참고

- https://www.pinterest.co.kr/pin/441986150938119758/
- 강의자료 : AI-2021-YU-02.pdf
- 강의자료 : AI-2021-YU-04-Perceptron Training.pdf