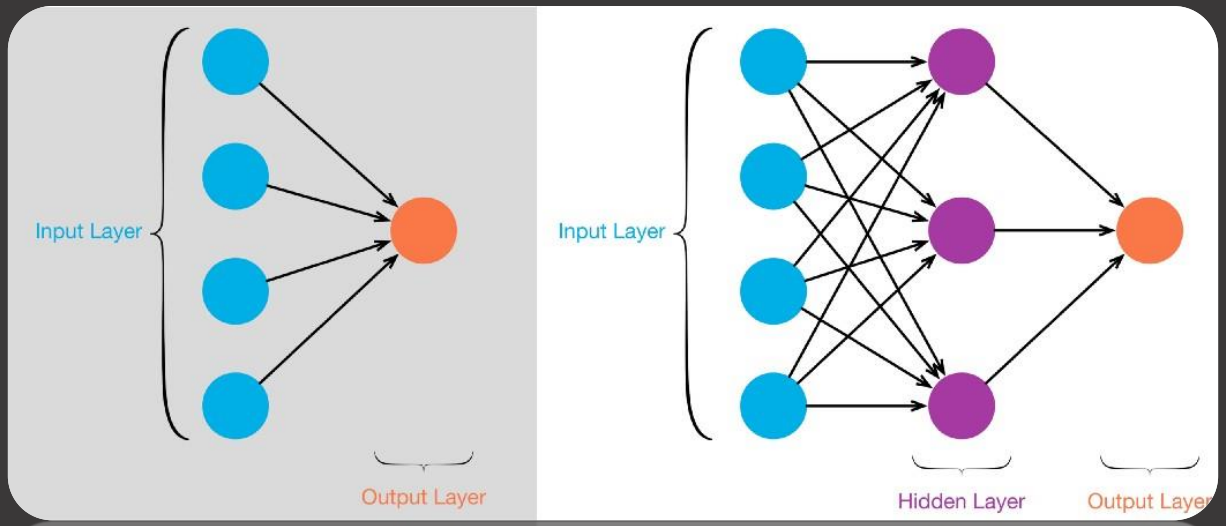


N Dimension 1-lyer Perceptron

인공지능 과제 #1

컴퓨터과학부 2017920036 양다은

https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

목차

1. 배경지식

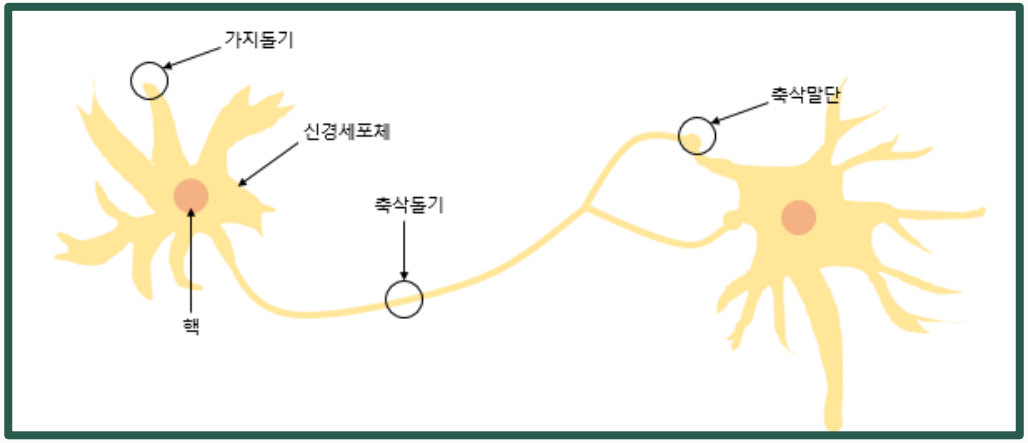
1. 배경지식

서론

인공 신경망은 수많은 머신 러닝 방법 중 하나이다. 최근 인공 신경망을 복잡하게 쌓아 올린 딥 러닝이 다른 머신 러닝 방법들보다 뛰어난 성능이 보여지고 있다. 따라서 전통적인 머신 러닝과 딥 러닝을 구분하는 것이 목표이다.

딥 러닝을 이해하고자 초기의 인공 신경망인 퍼셉트론(Perceptron)에 대해서 다룬다.

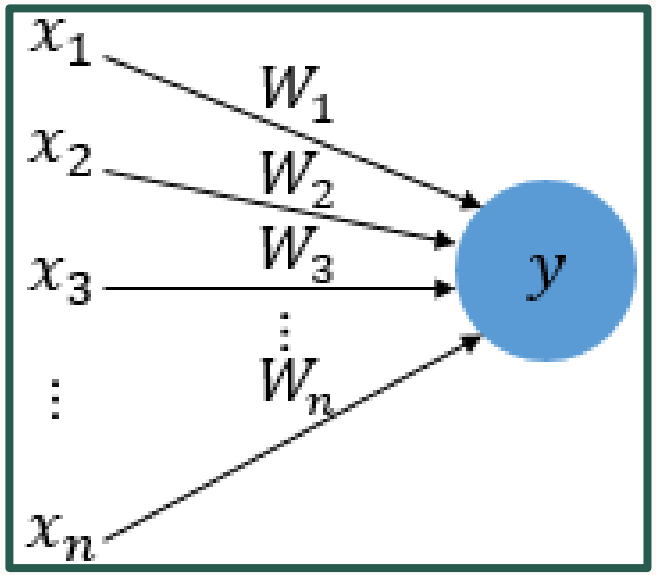
Perceptron은 실제 뇌를 구성하는 신경 세포 뉴런의 동작과 유사하다. 뉴런은 가지돌기에서 신호를 받아들이고, 이 신호가 일정치 이상 커지면 축삭돌기를 통해서 신호를 전달한다.

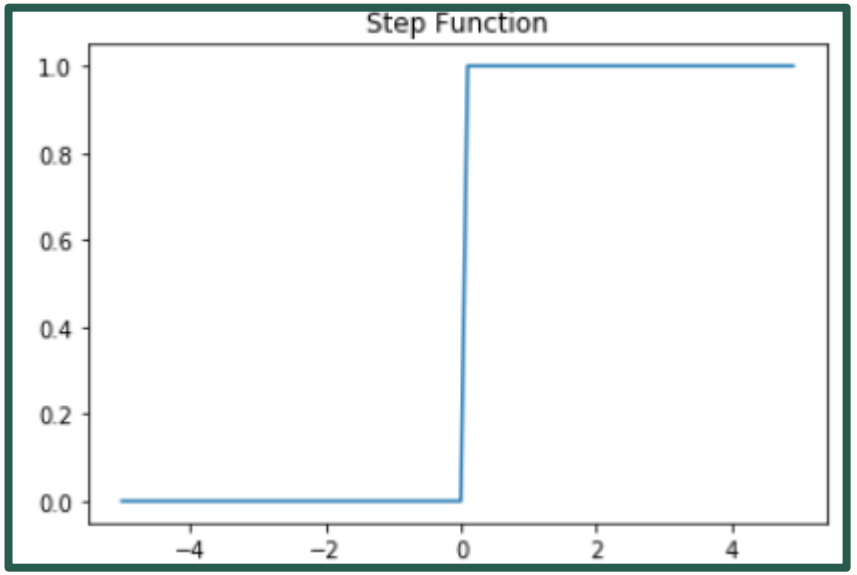


Perceptron은 프랑크 로젠블라트가 1957년에 제안한 초기 형태의 인공 신경망으로 다수의 입력으로 하나의 결과를 내보내는 알고리즘이다.

1. 배경지식

퍼셉트론 (=Perceptron)





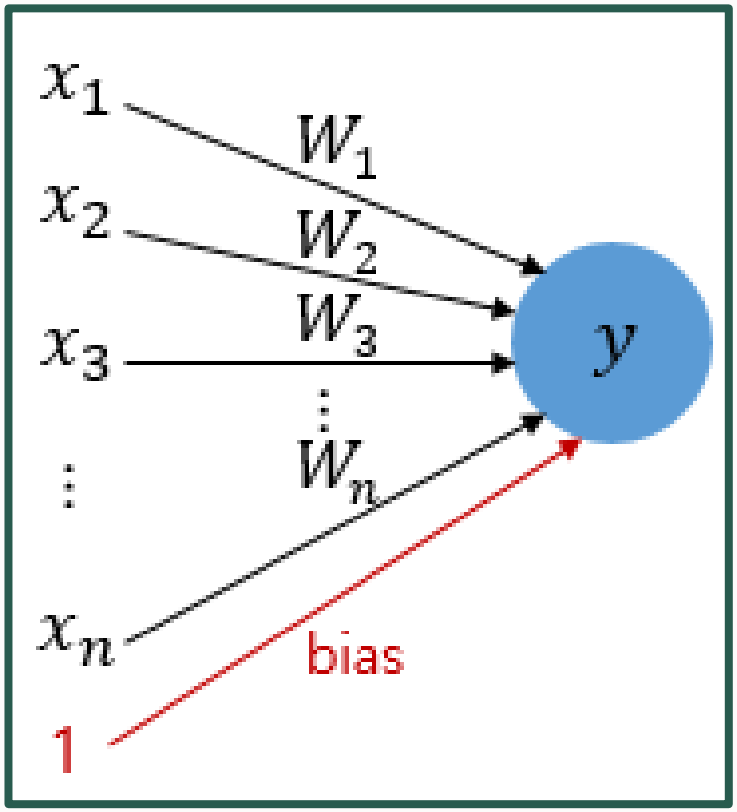
x_n는 입력, W_n는 가중치(weight), y는 출력을 의미한다. 각각의 입력 값에는 가중치가 존재하는데, 이때 가중치가 크면 클수록 해당 입력 값이 중요하다. 각 입력 값과 그에 해당하는 가중치의 곱의 전체 합이 임계치(threshold)를 넘으면 출력 신호로서 1을, 그렇지 않을 경우 0을 출력한다. 이러한 함수를 계단 함수(Step function)라고 한다.

$$\mathbf{if}\ \sum_i^n W_i x_i > \theta \ \rightarrow\ y = 1$$

$$\mathbf{if}\ \sum_i^n W_i x_i \leq \theta \ \rightarrow\ y = 0$$

1. 배경지식

퍼셉트론 (=Perceptron)



임계치를 좌변으로 넘기고 편향 b(bias)로 표현할 수도 있다. b 또한 퍼셉트론의 입력이 된다. 그림으로 표현할 때, 입력 값이 1로 고정이고 b 가 곱해지는 변수로 표현한다. b에 대한 최적의 값을 찾아야 한다.

뉴런에서 출력 값을 변경시키는 함수를 활성화 함수(Activation Function)라 한다. 초기 인공 신경망 모델인 퍼셉트론은 활성화 함수로 Step function를 사용했지만, 그 뒤 여러가지 발전된 신경망에서 여러 다양한 활성화 함수들이 등장했다.

$$if\ \sum_i^n W_i x_i + b > 0 \rightarrow y = 1$$

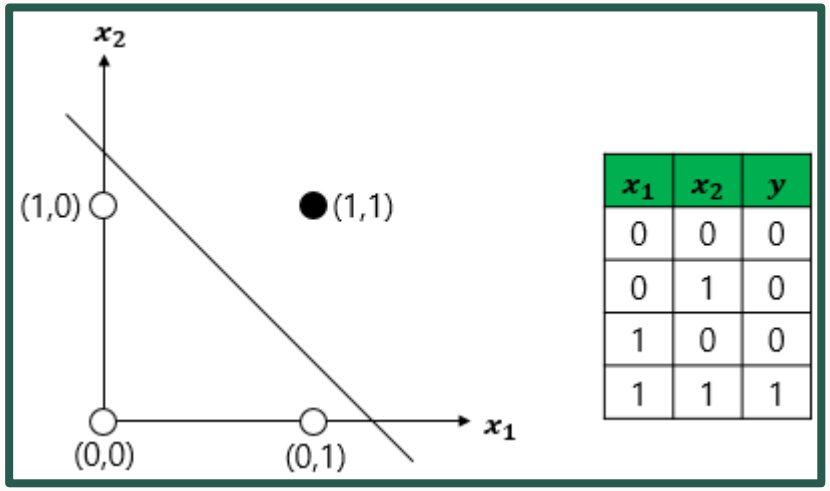
$$if\ \sum_i^n W_i x_i + b \leq 0 \rightarrow y = 0$$

1. 배경지식

단층 퍼셉트론 (=Single-Layer Perceptron)

퍼셉트론은 단층 퍼셉트론과 다층 퍼셉트론으로 나누는데, 단층 퍼셉트론은 값을 보내는 단계와 값을 받아서 출력하는 두 단계로만 이루어진다. 이때 이 각 단계를 층(layer)라고 부르며, 두 단계를 입력층(input layer)과 출력층(output layer)이라고 한다.

단층 퍼셉트론을 이용하면 AND, NAND, OR 게이트를 쉽게 구현할 수 있다. 게이트 연산에서 두 개의 입력 값과 하나의 출력 값이 쓰인다. 예를 들어 AND 게이트의 경우, 두 개의 입력이 모두 1인 경우에만 출력이 1이 나오는 구조를 갖는다. 다음과 같이 그래프로 시각화하면, 하나의 직선으로 나누는 것이 가능하다.



x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

$$W_1 = W_2 = 1$$

$$net = W_1x_1 + W_2x_2 - \theta = 0$$
$$= x_1 + x_2 - \theta$$

$$x_2 = -x_1 + \theta$$

2.코드구현

2. 코드구현

과제설명

- Input 차원이 n인 1-layer perceptron 을 구현한다.
- Weight 값들을 random 값으로 초기화한다.
- 다음을 무한 반복한다.
  - 앞의 AND gate의 입력과 같은 값을 차례로 입력하여 output을 구한다.
  - 네 가지의 input에 대하여 output이 틀린 개수를 구한다.
  - 각 input 별 output 이 모두 맞으면 종료한다.
  - 임의의 값을 입력 받아 각 weight에 대입한다.
- C, C++언어로 구현
- 결과물
  - source (주석 포함)
  - 결과 보고서 (어떤 weight값을 입력해야 무한 loop에서 빨리 나올 수 있는지?)
- 온라인 강의실에 제출
- 마감: 다음주 수업시간 전날까지

2. 코드구현

Source1.cpp

```cpp
// 과제#1 컴퓨터과학부 2017920036 양다은
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {

    srand((unsigned int)time(NULL));    // rand() 함수 사용을 위해 시간에 따라 난수표 초기화

    cout << "2개의 Input과 AND 연산에 대한 Output 출력"
        << endl << "(Perceptron의 weight와 theta 값의 영향 받음)" << endl;  // weight = 가중치, theta = 임계값

    int* weight_arr = new int[2];   // weight인 W1과 W2 배열
    int theta;
    int input[4][2] = { {0,0},{0,1},{1,0},{1,1} };  // X1과 X2
    int output[4] = { 0,0,0,1 };    // AND 연산에 대한 output
    int count = 0;  // 총 실행횟수
```

- 2개의 Input과 AND 연산에 대한 Output 출력하는 프로그램
- 변수와 배열 선언 및 초기화
  - W1과 W2를 포함하는 weight_arr 배열
  - 가중치 = theta
  - Input 조합의 수를 포함하는 input 배열
  - AND 연산에 대한 output 배열
  - 총 실행횟수 = count

2. 코드구현

Source1.cpp

```
20      while (1) {
21
22          int net = 0;     // 계산값 0 초기화
23          int result;      // 계산값에 대한 결과값
24          int incorrect = 0;  // 틀린 횟수 0 초기화
25
26          // 1. W1과 W2 랜덤하게 초기화
27          for (int i = 0; i < 2; i++) {
28              *(weight_arr + i) = rand() % 100 + 1;    // 1 ~ 100 중 하나의 수
29          }
30
31          // 2. theta 랜덤하게 초기화
32          theta = rand() % 100 + 1;    // 1 ~ 100 중 하나의 수
33
34          // W1, W2, theta 값 출력
35          cout << "W1 : " << *weight_arr
36              << "        W2 : " << *(weight_arr + 1)
37              << "        theta : " << theta << endl;
```

- 무한루프 생성
- 루프내 필요한 변수와 배열 선언 및 초기화
  - 계산값 = net
  - 계산값에 대한 결과값 = result
  - 틀린 횟수 = incorrect

1. W1과 W2 랜덤하게 초기화
2. theta 랜덤하게 초기화

2. 코드구현

Source1.cpp

```
39          for (int i = 0; i < 4; i++) {
40              // 3-1. net = X1*W1 + X2*W2
41              for (int j = 0; j < 2; j++) {
42                  net += input[i][j] * weight_arr[j];
43              }
44
45              // 3-2. net = net - theta
46              net -= theta;
47
48              // 4. result 도출
49              // net = X1*W1 + X2*W2 - theta
50              // net > 0 이면 result = 1
51              // net <= 0 이면 result = 0
52              if (net > 0) result = 1;
53              else result = 0;
54
55              // outout, result 값 출력
56              cout << "output : " << output[i]
57                  << "    result : " << result << endl;
58
59              // 5. result와 outout이 같지 않으면 incorrect(= 틀린 횟수) + 1
60              if (result != output[i]) incorrect++;
```

3. net = X1*W1 + X2*W2 –theta 계산
4. result 도출
   - net > 0 이면 result = 0 / 아니면 result = 1
5. result와 output이 같지 않으면 incorrect + 1

2. 코드구현

Source1.cpp

```
62
63          // 총 실행 횟수 + 1
64          count++;
65          // 틀린 횟수 출력
66          cout << "틀린 횟수 : " << incorrect << endl;
67          // 6. 틀린 횟수가 없다면, 무한 루프 탈출
68          if (incorrect == 0) break;
69      }
70
71      // 총 실행 횟수 출력
72      cout << endl << "총 실행 횟수 : " << count << endl;
73
74      return 0;
75  }
```

- 총 실행 횟수 + 1

6. 틀린 횟수 = 0 이면, 무한 루프 탈출

2. 코드구현

Source2.cpp

```
1   // 과제#1 컴퓨터과학부 2017920036 양다은
2   #include <iostream>
3   #include <cstdlib>
4   #include <ctime>
5   using namespace std;
6
7   int main() {
8
9       srand((unsigned int)time(NULL));    // rand() 함수 사용을 위해 시간에 따라 난수표 초기화
10
11      cout << "N Dimension 1-layer Perceptron" << endl
12          << "(weight와 theta 값의 영향 받음)" << endl << endl;   // weight = 가중치, theta = 임계값
13
14      int N, Case;    // N = input의 차원, Case = input 조합의 수
15
16      // 1. N 값 입력
17      cout << "N ? ";
18      cin >> N;
19
20      // input 조합의 수 = N^2, 1을 N만큼 오른쪽 쉬프트 연산
21      Case = 1 << N;
```

- N Dimension 1-layer Perceptron 프로그램
- 변수와 배열 선언 및 초기화
  - N = input의 차원
  - Case = input 조합의 수 = N^2

1. N 값 입력

2. 코드구현

Source2.cpp

```
23      // 2. input 값 생성 (X1, X2, ... Xn)
24      int** inputs = (int**)malloc(sizeof(int*) * Case);  // Case를 포함하는 inputs 배열 선언 및 메모리 동적할당
25      for (int i = 0; i < Case; i++) {
26          inputs[i] = (int*)malloc(sizeof(int) * N);      // N개의 input을 포함하는 메모리 동적할당
27      }
28      for (int i = 0; i < Case; i++) {
29          for (int j = N - 1; j >= 0; j--) {
30              int input = i >> j & 1;
31              //cout << input;
32              inputs[i][j] = input;
33          }
34          //cout << endl;
35      }
36
37      // 3. AND 연산에 대한 output 값 생성 (O1=0, O2=0, ... On=1)
38      int* outputs = (int*)malloc(sizeof(int*) * Case);   // Case개의 output을 포함하는 outputs 배열 선언 및 메모리 동적할당
39      for (int i = 0; i < Case; i++) {
40          int output = 1;
41          for (int j = 0; j < N; j++) {
42              if (inputs[i][j] == 0) {
43                  output = 0;
44                  break;
45              }
46          }
47          //cout << output;
48          outputs[i] = output;
49      }
```

- 메모리 동적할당

2. input 값 생성 (X1, X2, … Xn)
3. output 값 생성 (O1, O2, … On)

2. 코드구현

Source2.cpp

```
51      clock_t start, end; // start = 시작시간, end = 종료시간
52      int* weight_arr = (int*)malloc(sizeof(int*) * N);   // N개의 weight를 포함하는 배열 선언 및 메모리 동적할당
53      int theta;
54      int count = 0;  // 총 실행횟수
55
56      start = clock();    // 시간 측정 시작
57
58      while (1) {
59
60          int net = 0;        // 계산값 0 초기화
61          int result;         // 계산값에 대한 결과값
62          int incorrect = 0;  // 틀린 횟수 0 초기화
63
64          // 4. weight 랜덤하게 초기화 (W1, W2, ... Wn)
65          for (int i = 0; i < N; i++) {
66              *(weight_arr + i) = rand() % 10 + 1;    // 1 ~ 10 중 하나의 수
67          }
68
69          // 5. theta 랜덤하게 초기화
70          theta = rand() % 10 + 1;    // 1 ~ 10 중 하나의 수
71
72          // 6-1. input과 weight 계산
73          // net = X1*W1 + X2*W2 + ... + Xn*Wn
74          for (int i = 0; i < Case; i++) {
75              for (int j = 0; j < N; j++) {
76                  net += inputs[i][j] * weight_arr[j];
77              }
78
79              // 6-2. net = net - theta
80              net -= theta;
```

- 총 실행시간(= start – end) 측정을 위한 clock() 함수 사용

4. weight 랜덤하게 초기화
5. theta 랜덤하게 초기화
6. input과 weight, theta 로 net 계산

2. 코드구현

Source2.cpp

```
82              // 7. result 도출
83              // net > 0 이면 result = 1
84              // net <= 0 이면 result = 0
85              if (net > 0) result = 1;
86              else result = 0;
87
88              // outout, result 값 출력
89              //cout << "output : " << outputs[i]
90              //  << "    result : " << result << endl;
91
92              // 8. result와 outout이 같지 않으면 incorrect(= 틀린 횟수) + 1
93              if (result != outputs[i]) incorrect++;
94          }
95
96          // 총 실행 횟수 + 1
97          count++;
98          // 틀린 횟수 출력
99          cout << count << "번째 틀린 횟수 : " << incorrect << endl;
100         // 9. 틀린 횟수가 없다면, 무한 루프 탈출
101         if (incorrect == 0) break;
102     }
103
104     end = clock();  // 시간 측정 종료
```

7. result 도출
   7. net = X1*W1 + X2*W2 + … Xn*Wn – theta > 0 이면 result = 1 / 아니면 result = 0
8. result와 output이 같지 않으면 incorrect + 1
- 총 실행 횟수 + 1
9. 틀린 횟수 = 0 이면, 무한 루프 탈출

2. 코드구현

Source2.cpp

```
106     // 총 실행 횟수 출력
107     cout << endl << "총 실험 횟수 : " << count << endl;
108     // 총 실행 시간 출력
109     cout << "총 실행 시간 : " << (double)(end - start) << "ms" << endl;
110
111     // 10. 동적할당한 변수 해제
112     for (int i = 0; i < N; i++) free(inputs[i]);
113     free(inputs);
114     free(outputs);
115     free(weight_arr);
116
117     return 0;
118 }
```

- 총 실행 횟수와 총 실행 시간 출력

10. 동적으로 할당한 변수 해제
    - inputs
    - outputs
    - weight_arr

3.구현결과

3. 구현결과

Source1.cpp 실행결과

- 실행 1

```
2개의 Input과 AND 연산에 대한 Output 출력
(Perceptron의 weight와 theta 값의 영향 받음)
W1 : 74         W2 : 73         theta : 9
output : 0      result : 0
output : 0      result : 1
output : 0      result : 1
output : 1      result : 1
틀린 횟수 : 2
W1 : 28         W2 : 25         theta : 58
output : 0      result : 0
output : 0      result : 0
output : 0      result : 0
output : 1      result : 0
틀린 횟수 : 1
```

```
틀린 횟수 : 1
W1 : 58         W2 : 56         theta : 17
output : 0      result : 0
output : 0      result : 1
output : 0      result : 1
output : 1      result : 1
틀린 횟수 : 2
W1 : 81         W2 : 78         theta : 73
output : 0      result : 0
output : 0      result : 0
output : 0      result : 0
output : 1      result : 1
틀린 횟수 : 0

총 실행 횟수 : 12
```

3. 구현결과

Source1.cpp 실행결과

- 실행 2

```
2개의 Input과 AND 연산에 대한 Output 출력
(Perceptron의 weight와 theta 값의 영향 받음)
W1 : 44          W2 : 86          theta : 52
output : 0       result : 0
output : 0       result : 0
output : 0       result : 0
output : 1       result : 1
틀린 횟수 : 0

총 실행 횟수 : 1
```

3. 구현결과

Source1.cpp 실행결과

- 실행 3

```
2개의 Input과 AND 연산에 대한 Output 출력
(Perceptron의 weight와 theta 값의 영향 받음)
W1 : 100                W2 : 14         theta : 35
output : 0      result : 0
output : 0      result : 0
output : 0      result : 1
output : 1      result : 1
틀린 횟수 : 1
W1 : 70         W2 : 7          theta : 46
output : 0      result : 0
output : 0      result : 0
output : 0      result : 0
output : 1      result : 0
틀린 횟수 : 1
W1 : 28         W2 : 47         theta : 51
output : 0      result : 0
output : 0      result : 0
output : 0      result : 0
output : 1      result : 0
틀린 횟수 : 1
W1 : 79         W2 : 49         theta : 45
output : 0      result : 0
output : 0      result : 0
output : 0      result : 0
output : 1      result : 1
틀린 횟수 : 0

총 실행 횟수 : 4
```

3. 구현결과

Source1.cpp 실행결과

Source1.cpp은 input이 2차원인 1-layer 퍼셉트론을 구현한 것이다.

한번 루프가 돌 때마다 W_1과 W_2, θ(= 임계값) 의 값을 1 ~ 100의 수 중에서 랜덤하게 초기화했다. Source1.cpp을 실행하면서 강제 종료를 시킨 적은 없다. 적게는 1회부터 많게는 25회까지 프로그램이 돌고 스스로 종료하였다. 또한 각 루프의 동작마다 틀린 횟수는 0 ~ 2 회로 보였다.

하지만 코드의 확장성을 위해 N 차원 1-layer Perceptron을 구현하는 Source2.cpp를 작성했다. Source2.cpp에서는 입력 값으로 N을 받은 후 실행된다.

3. 구현결과

Source2.cpp 실행결과

- 실행 1 (N=3)

```
N Dimension 1-layer Perceptron
(weight와 theta 값의 영향 받음)

N ? 3
1번째 틀린 횟수 : 1
2번째 틀린 횟수 : 1
3번째 틀린 횟수 : 6
4번째 틀린 횟수 : 6
5번째 틀린 횟수 : 1
6번째 틀린 횟수 : 5
7번째 틀린 횟수 : 1
8번째 틀린 횟수 : 4
9번째 틀린 횟수 : 1
10번째 틀린 횟수 : 1
11번째 틀린 횟수 : 0

총 실험 횟수 : 11
총 실행 시간 : 5ms
```

3. 구현결과

Source2.cpp 실행결과

- 실행 2 (N=3)

```
N Dimension 1-layer Perceptron
(weight와 theta 값의 영향 받음)

N ? 3
1번째 틀린 횟수 : 1
2번째 틀틀린린린린린 횟수 : 6
3번째 틀틀린린린린린 횟수 : 1
4번째 틀틀린린린린린 횟수 : 1
5번째 틀틀린린린린린 횟수 : 3
6번째 틀틀린린린린린 횟수 : 1
7번째 틀틀린린린린린 횟수 : 4
8번째 틀틀린린린린린 횟수 : 4
9번째 틀틀린린린린린 횟수 : 1
10번째 틀틀린린린린린 횟수 : 6
11번째 틀틀린린린린린 횟수 : 5
12번째 틀틀린린린린린 횟수 : 4
13번째 틀틀린린린린린 횟수 : 6
14번째 틀틀린린린린린 횟수 : 1
15번째 틀틀린린린린린 횟수 : 1
16번째 틀틀린린린린린 횟수 : 1
17번째 틀틀린린린린린 횟수 : 5
18번째 틀틀린린린린린 횟수 : 2
19번째 틀틀린린린린린 횟수 : 4
20번째 틀틀린린린린린 횟수 : 0

총 실험 횟수 : 20
총 실행 시간 : 9ms
```

3. 구현결과

Source2.cpp 실행결과

- 실행 3 (N=4)

```
N Dimension 1-layer Perceptron
(weight와 theta 값의 영향 받음)

N ? 4
1번째 틀린 횟수 : 13
2번째 틀린 횟수 : 3
3번째 틀린 횟수 : 2
4번째 틀린 횟수 : 1
5번째 틀린 횟수 : 4
6번째 틀린 횟수 : 11
7번째 틀린 횟수 : 12
8번째 틀린 횟수 : 13
9번째 틀린 횟수 : 10
10번째 틀린 횟수 : 1
11번째 틀린 횟수 : 11
12번째 틀린 횟수 : 13
13번째 틀린 횟수 : 1
14번째 틀린 횟수 : 1
```

```
59번째 틀린 횟수 : 1
60번째 틀린 횟수 : 10
61번째 틀린 횟수 : 14
62번째 틀린 횟수 : 14
63번째 틀린 횟수 : 12
64번째 틀린 횟수 : 4
65번째 틀린 횟수 : 14
66번째 틀린 횟수 : 0

총 실험 횟수 : 66
총 실행 시간 : 127ms
```

3. 구현결과

Source2.cpp 실행결과

- 실행 4 (N=4)

```
N Dimension 1-layer Perceptron
(weight와 theta 값의 영향 받음)

N ? 4
1번째 틀린 횟수 : 14
2번째 틀린 횟수 : 0

총 실험 횟수 : 2
총 실행 시간 : 1ms
```

3. 구현결과

Source2.cpp 실행결과

- 실행 5 (N=4)

```
N Dimension 1-layer Perceptron
(weight와 theta 값의 영향 받음)

N ? 4
1번째 틀린 횟수 : 12
2번째 틀린 횟수 : 9
3번째 틀린 횟수 : 10
4번째 틀린 횟수 : 2
5번째 틀린 횟수 : 13
6번째 틀린 횟수 : 9
7번째 틀린 횟수 : 13
8번째 틀린 횟수 : 1
9번째 틀린 횟수 : 12
10번째 틀린 횟수 : 14
11번째 틀린 횟수 : 12
```

```
26번째 틀린 횟수 : 3
27번째 틀린 횟수 : 1
28번째 틀린 횟수 : 1
29번째 틀린 횟수 : 1
30번째 틀린 횟수 : 11
31번째 틀린 횟수 : 1
32번째 틀린 횟수 : 11
33번째 틀린 횟수 : 0

총 실험 횟수 : 33
총 실행 시간 : 15ms
```

3. 구현결과

Source2.cpp 실행결과

- 실행 6 (N=5)
- 강제 종료

```
N Dimension 1-layer Perceptron
(weight와 theta 값의 영향 받음)

N ? 5
1번째 틀린 횟수 : 27
2번째 틀틀린린 횟수 : 26
3번째 틀틀틀린린린 횟수 : 29
4번째 틀틀틀틀린린린린 횟수 : 27
5번째 틀틀틀틀틀린린린린린 횟수 : 29
6번째 틀틀틀틀틀틀린린린린린린 횟수 : 30
7번째 틀틀틀틀틀틀틀린린린린린린린 횟수 : 29
8번째 틀틀틀틀틀틀틀틀린린린린린린린린 횟수 : 29
9번째 틀틀틀틀틀틀틀틀틀린린린린린린린린린 횟수 : 28
10번째 틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린 횟수 : 22
11번째 틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린 횟수 : 30
12번째 틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린 횟수 : 28
13번째 틀틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린린 횟수 : 28
14번째 틀틀틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린린린 횟수 : 30
15번째 틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린린린린 횟수 : 28
16번째 틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린린린린린 횟수 : 29
17번째 틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린린린린린린 횟수 : 30
18번째 틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린린린린린린린 횟수 : 17
19번째 틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린린린린린린린린 횟수 : 6
20번째 틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린린린린린린린린린 횟수 : 24
21번째 틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린린린린린린린린린린 횟수 : 29
22번째 틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀틀린린린린린린린린린린린린린린린린린린린린린린 횟수 : 28
```

3. 구현결과

Source2.cpp 실행결과

- 실행 7 (N=10)
- 강제 종료

```
N Dimension 1-layer Perceptron
(weight와 theta 값의 영향 받음)

N ? 10
1번째 틀린 횟수 : 1005
2번째 틀린 횟수 : 940
3번째 틀린 횟수 : 1016
4번째 틀린 횟수 : 1018
5번째 틀린 횟수 : 1020
6번째 틀린 횟수 : 1017
7번째 틀린 횟수 : 1020
8번째 틀린 횟수 : 1020
9번째 틀린 횟수 : 1021
10번째 틀린 횟수 : 1001
11번째 틀린 횟수 : 1022
12번째 틀린 횟수 : 1020
13번째 틀린 횟수 : 1001
14번째 틀린 횟수 : 1022
15번째 틀린 횟수 : 1021
16번째 틀린 횟수 : 978
17번째 틀린 횟수 : 1021
18번째 틀린 횟수 : 1016
19번째 틀린 횟수 : 1022
20번째 틀린 횟수 : 1021
21번째 틀린 횟수 : 1012
22번째 틀린 횟수 : 1020
23번째 틀린 횟수 : 1012
```

3. 구현결과

Source2.cpp 실행결과

Source2.cpp은 코드 확정성을 실현하고자 input이 N차원인 1-layer Perceptron을 구현한 것이다.

입력 값으로 N을 받고, 프로그램이 스스로 종료할 경우, 총 실행 횟수와 총 실행 시간이 출력된다. 한번 루프가 돌 때마다 weight(= 가중치)의 값들, θ(= 임계값) 의 값을 1 ~ 10의 수 중에서 랜덤하게 초기화했다. N = 3인 경우 총 실행 횟수가 11회/ 20회, N = 4인 경우 66회/ 2회/ 33회, N = 5인 경우와 N = 10인 경우는 약 1분간 프로그램이 끝나지 않아 강제 종료했다.

Source1.cpp과 달리 N > 4 인 대부분의 경우에서는 강제 종료를 실행해야만 한다. 그 이유는 N이 커지면서 weight과 input의 곱의 총합이 기하급수적으로 커지고, 이를 1~10의 수 중 하나인 θ로 뺀다고 한들 net의 값이 0보다 작기는 매우 어렵기 때문이다.

3. 구현결과

결론

input이 N차원인 AND Gate에 대한 Single-layer Perceptron을 구현해보았다.

입력 값인 N이 4보다 크다면, 강제종료를 실행해야 한다. 그 이유는 N개인 weight과 input의 곱의 총합이 기하급수적으로 커지고, 이를 1~10의 수 중 랜덤한 수인 θ로 뺀다고 한들 계산 값 net이 0보다 작기는 매우 어렵기 때문이다. 한번의 루프마다 틀린 횟수를 출력하는데, 2^{N-1} < 틀린 횟수 < 2^{N}에서 크게 벗어나지 않는다. 틀린 횟수 = 0인 경우만 무한 루프를 탈출할 수 있다. 하지만 N이 증가할수록 $0 \ll 2^{N-1}$ < 틀린 횟수 이므로 무한 루프를 빠져나올 수 없는 것이다.

무한 루프를 탈출하기 위해서 N이 커짐에 따라 θ도 커져야 한다. 혹은 weigh의 값 N의 증가만큼 특정 비율로 감소해야 한다. 자세한 내용은 다음 페이지 :)

3. 구현결과

결론

어떤 값을 입력해야 무한 loop에서 빨리 나올 수 있는가?

N이 커짐에 따라 θ도 커져야 한다. 혹은 weigh의 값 N의 증가만큼 특정 비율로 감소해야 한다.

N = 2인 경우를 살펴보자. $net = W_1x_1 + W_2x_2 - \theta > 0$이 참이면, $result = 1$이고, 거짓이면 $result = 0$인 것이 조건이다. 여기서 AND 연산에 대한 결과에 만족하는 θ 값을 취하는 방법은, 다음과 같다.

첫째, $x_1 = 0,\ x_2 = 0$일 때, $net = -\theta \leq 0$ 즉, $0 \leq \theta$ 를 만족해야 한다.

둘째, $x_1 = 0,\ x_2 = 1$일 때, $net = W_2 - \theta \leq 0$ 즉, $W_2 \leq \theta$ 를 만족해야 한다.

셋째, $x_1 = 1,\ x_2 = 0$일 때, $net = W_1 - \theta \leq 0$ 즉, $W_1 \leq \theta$ 를 만족해야 한다.

넷째, $x_1 = 1,\ x_2 = 1$일 때, $net = W_1 + W_2 - \theta > 0$ 즉, $W_1 + W_2 > \theta$ 를 만족해야 한다.

따라서 $Max(W_i) \leq \theta < W_1 + W_2$ 를 만족하는 θ 값을 조절한다.

N > 2 인 경우에는 $\sum_i^n W_i - Min(W_i) \leq \theta < \sum_i^n W_i$ 를 만족하는 θ 값을 취해야 한다. 따라서, N이 커짐에 따라 1~10의 수 중 랜덤한 수가 아닌 θ도 일정한 비율로 커져야 한다. 혹은 wieght의 값을 특정 비율로 감소할 수 있다.

3. 구현결과

한계점

1. N에 따라 input과 output의 메모리를 동적으로 할당한다. 이에 영향으로 input과 AND 연산에 대한 output의 데이터도 변하게 된다. 데이터 값을 수기 입력이 아닌 직접 수행을 구현하는데 어려움이 있었다.
메모리 동적 할당에 대한 자료는 https://hijuworld.tistory.com/60 를 참고하였고,
bit연산과 bit의 자릿수 계산에 대한 자료는 https://coding-factory.tistory.com/655 를 참고했다.
자료의 개념을 이해하고, 코드에 적용하여 문제점을 해결했다.

2. 임계치 θ 값이 N에 따라 일정한 비율로 변화는 코드를 구현할 예정이다.

4. 출처

- https://yusaebyeol.blogspot.com/2015/08/free-ppt-ppt-064-ppt.html
- https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f
- https://wikidocs.net/24958
- https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=samsjang&logNo=220955881668
- https://blockdmask.tistory.com/308
- https://hijuworld.tistory.com/60
- https://coding-factory.tistory.com/655
- https://eehoeskrap.tistory.com/261