Machine Learning

HW3

-Neural Network-

Subject : 기계학습의 기초및응용
Professor : Kye-haeng Lee
Name : Dongho Bang
Student Number : 32181995
Data : 2022.11.27

Contents

1. Introduction

2. Design

   A. Main Idea

   B. Program Description

3. Evaluation Results

   A. Evaluation Setup

   B. Training/Validation error

   C. Final Model and Parameters

4. Conclusion

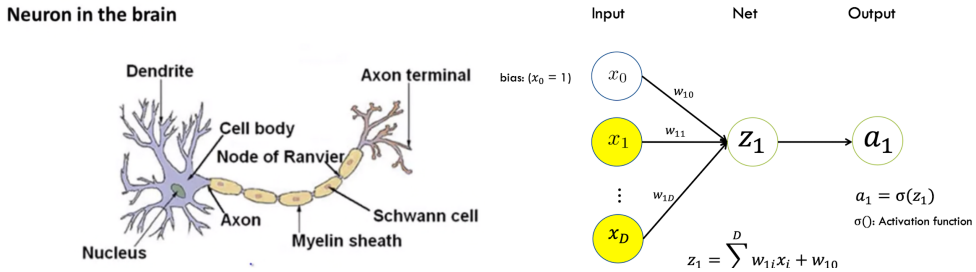
1. Introduction

A. Goal

Design a neural network to approximate a function that can describe the given data. The dataset has two sets of values : one (the first column in the given file) is x, and the other one(the second column) is y (y=f(x)). After designing your own model and training, find its best parameter set. You must provide both the model and weights.

B. Concept

퍼셉트론은 다수의 신호를 입력받아 하나의 신호를 출력합니다.
전류가 전선을 타고 흐르는 전자를 내보내듯이, 퍼셉트론 신호도 흐름을 만들고 정보를 앞으로 전달하게 됩니다.



(좌) neuron (우) artificial neuron

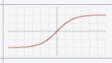
뉴런 or 노드(node) : 원형으로 표현된것

입력신호(x0, x1, x2,...)가 뉴런에 입력되면 각각 고유한 가중치 (weights, w0, w1,..)가 곱해지는데, 가중치는 시냅스의 연결강도와 같은 역할을 한다. 가중치가 곱해진 값들을 모두 "더해져" 정해진 임계값(threshold)을 넘을 경우에만 다음 노드들이 있는 층(Layer)으로 신호가 전해진다.

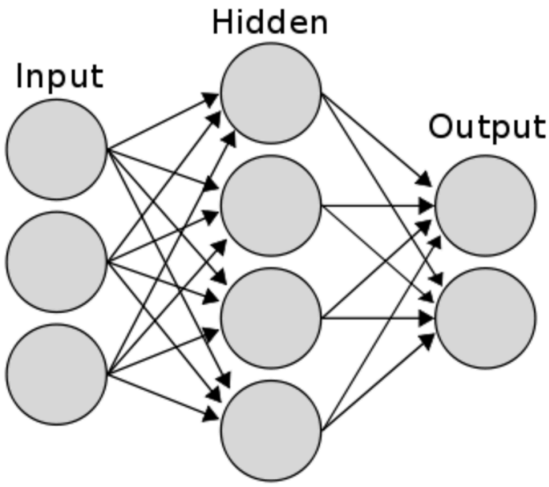
퍼셉트론의 로직 게이트(논리회로의 진리표)

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 0	B A X 0 0 1 0 1 0 1 0 0 1 1 1

활성 함수(Activation Fuction)

Name	Plot	Equation	Derivative (with respect to x)
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$ [1]	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$
Rectified linear unit (ReLU)[12]		$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} = \max\{0, x\} = x\mathbf{1}_{x>0}$	$f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$

신경망 층의 구성



입력층(Input Layers)
입력층은 데이터셋으로부터 입력을 받습니다. 입력 변수의 수와 입력 노드의 수는 같다. 보통 입력층은 어떤 계산도 수행하지 않고 그냥 값들을 전달하기만 하는 특징을 가지고 있습니다. 그래서 신경망의 층수(깊이, depth)를 셀 때 입력층은 포함하지 않는다. 위의 신경망은 2층(Layer)이라고 할 수 있다.

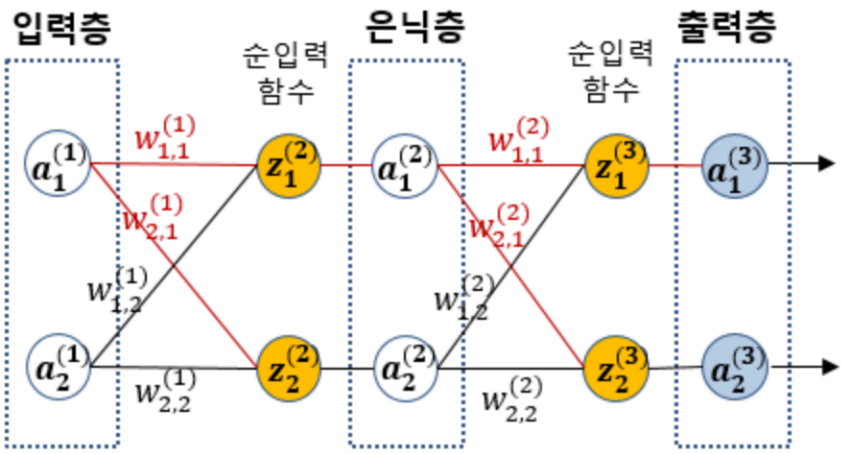
은닉층(Hidden Layers)
계산이 일어나는 층이 둘 이상인 신경망을 다층(multilayer) 신경망 이라고 부르는 데 계산이 없는 입력층과 마지막 출력 층 사이에 있는 층들을 은닉 층(Hidden Layers)이라고 부른다. 딥 러닝은 사실 두 개 이상의 은닉층들을 가진 신경망, 입력층을 제외하고 시작하여 3개 이상의 Layer를 갖는 신경망을 의미한다.

출력층(Output Layers)
신경망 가장 오른쪽 마지막 층이 출력층이다. 출력층에는 대부분 활성함수(Activation Fuction)가 존재하는데 활성화 함수는 풀고자 하는 문제에 따라 다른 종류를 사용한다.

- 회귀 문제에서 예측할 목표 변수가 실수 값인 경우 활성화 함수가 필요하지 않으며 출력 노드의 수는 출력 변수의 개수와 같습니다.
- 이진 분류(Binary Classification) 문제의 경우 시그모이드(Sigmoid) 함수를 사용하여 출력을 확률 값으로 변환하여 Label을 결정하도록 한다.

역전파(Backpropagation)

심층 신경망을 학습하는데 유용하게 활용되는 역전파(Backpropagation) 알고리즘도 결국 경사하강법을 이용한다.



먼저 위와 같은 2-2-2 다층 퍼셉트론이 있다고 가정해보자. 여기서는 역전파의 개념을 알아보겠다. 또한 용어의 통일성을 위해 입력층의 x1, x2는 a1(1), a2(1)로 표기하기로 하자.

$w_{j,i}^{(l)}$ l층의 i 번째 노드와 l+1층의 j번째 노드를 연결하는 가중치 w.

이 구조에서 적용되는 활성 함수는 시그모이드 함수이다.

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

입력층 -> 은닉층 사이의 값의 흐름은 다음과 같다.

$$z_1^{(2)} = w_{1,1}^{(1)} a_1^{(1)} + w_{1,2}^{(1)} a_2^{(1)}$$

$$z_2^{(2)} = w_{2,1}^{(1)} a_1^{(1)} + w_{2,2}^{(1)} a_2^{(1)}$$

$$a_1^{(2)} = \phi(z_1^{(2)})$$

$$a_2^{(2)} = \phi(z_2^{(2)})$$

그리고 은닉층 -> 출력층 사이의 값의 흐름은 다음과 같다.

$$z_1^{(3)} = w_{1,1}^{(2)} a_1^{(2)} + w_{1,2}^{(2)} a_2^{(2)}$$

$$z_2^{(3)} = w_{2,1}^{(2)} a_1^{(2)} + w_{2,2}^{(2)} a_2^{(2)}$$

$$a_1^{(3)} = \phi(z_1^{(3)})$$

$$a_2^{(3)} = \phi(z_2^{(3)})$$

이와 같이 입력층 a1(1), a2(1)을 시작으로 출력층의 a1(3), a2(3)값이 출력되는 과정을 순전파(feedforward)라 한다.

비용함수 J를 오차제곱합으로 정의를 해보면 다음과 같이된다.

$$J_1 = \frac{1}{2}\left(a_1^{(3)} - y_1\right)^2$$

$$J_2 = \frac{1}{2}\left(a_2^{(3)} - y_2\right)^2$$

여기서 y1, y2는 트레이닝 데이터에 대한 각 노드에 대응되는 실제 값이다. 순전파 1회가 마무리되면 J1, J2의 값이 결정된다. 입력값 a1(1), a2(1)과 실제값 y1, y2는 고정된 값이므로 J1과 J2는 결국 가중치 w1,1(1)~w2,2(2)를 변수로 가지는 함수가 된다. 입력값 a1(1), a2(1)과 실제값 y1, y2는 고정된 값이므로 J1, J2는 결국 가중치 w1,1(1)~w2,2(2)를 변수로 가지는 함수가 된다. 목표는 J1, J2의 값이 최소가 되도록

w1,1(1)~w2,2(2)를 결정해야 한다. 다층 퍼셉트론에서 비용함수의 최소값을 찾아가는 방법으로 경사하강법을 활용한다고 했다. 각 층에서 가중치를 업데이트하기 위해서는 결국 각 층에서의 비용함수의 미분값이 필요하게 된다.
이를 매우 효율적으로 해결하기위한 방법이 바로 역전파 알고리즘이다.

역전파란 역방향으로 오차를 전파 시키면서 각층의 가중치를 업데이트하고 최적의 학습 결과를 찾아가는 방법이다.
이런 경우를 생각해보자, 순전파에 있어서 가중치의 값을 매우 미세하게 변화시키면 비용함수 J1, J2도 매우 미세하게 변화될 것이다. 매우 미세하게 변화시킨다는 것은 미분을 한다는 이야기이다. 그런데, 매우 미세한 영역으로 국한할 때 가중치의 미세변화와 이에 따른 비용함수의 미세변화가 선형적으로 관계가 된다고 알려져 있다. 선형적인 관계라는 이야기는 결국 비용함수를 매우 미세하게 변화시키면 이에 따라 가중치도 매우 미세하게 변화되는데 이 역시 선형적인 관계라는 이야기이다.
이런 원리에 의해, 순전파를 통해 출력층에서 계산된 오차 J1, J2의 각 가중치에 따른 미세변화를 입력층 방향으로 역전파시키면서 가중치를 업데이트하고, 또 다시 입력값을 이용해 순전파시켜 출력층에서 새로운 오차를 계산하고, 이 오차를 또다시 역전파시켜 가중치를 업데이트하는 식으로 반복한다.

역전파를 이용한 가중치 업데이트 절차는 아래와 같이 요약된다.

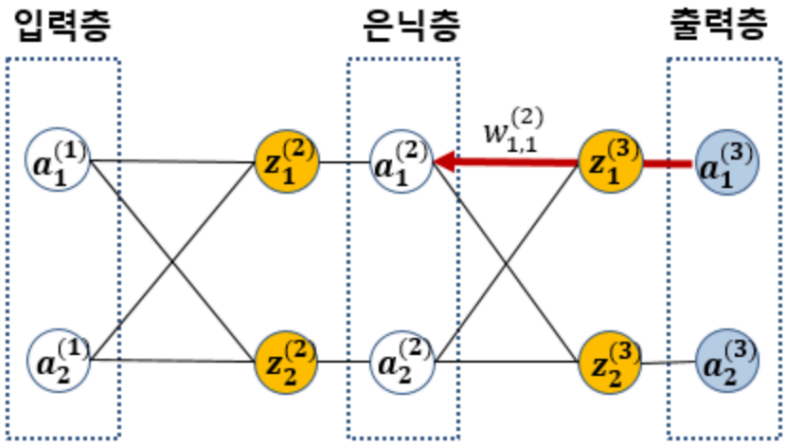
1. 주어진 가중치 값을 이용해 출력층의 출력 값을 계산한다. (순전파를 통한)
2. 오차를 각 가중치로 미분한 값(실제로는 learning rate을 곱한 값)을 기존 가중치에서 빼준다. (경사하강법을 적용하는 것이며, 역전파를 통해 이루어짐)
3. 2번 단계는 모든 가중치에 대해 이루어진다.
4. 1~3단계를 주어진 학습회수만큼 또는 주어진 허용 오차 값에 도달할 때까지 반복한다.

2번 단계의 가중치 업데이트 식을 수식으로 표현하면 다음과 같다.

$$\boldsymbol{w_j = w_j - \eta \frac{\partial J_{total}}{\partial w_j}}$$

여기서 Jtotal은 해당 노드가 영향을 미치는 오차의 총합이다. 예를 들면 출력층의 노드 a1(3)에서는 다른 노드에 영향을 전혀 미치지 않으므로 해당 노드에서 오차인 J1만 따지면 되지만, 은닉층의 a1(2) 노드는 출력 층의 a1(3), a2(3)노드의 오차에 영향을 미치므로 J1, J2모두 더한 것 이 Jtotal값이 된다.

이번에는 역전파를 통한 가중치 업데이트 메커니즘을 살펴보겠다. (편의상 learning rate = 1) 역전파를 이용해 가중치를 업데이트 해보면 아래와 같다.



먼저 가중치 w1,1(2)에 대해서 계산을 한다. W1,1(2)에 대한 업데이트 식은 아래와 같다.

$$w_{1,1}^{(2)} = w_{1,1}^{(2)} - \frac{\partial J_{total}}{\partial w_{1,1}^{(2)}}$$

미분의 연쇄법칙에 의해 오차의 가중치에 대한 미분 값은 아래의 식으로 표현된다.

$$\frac{\partial J_{total}}{\partial w_{1,1}^{(2)}} = \frac{\partial J_1}{\partial a_1^{(3)}} \times \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \times \frac{\partial z_1^{(3)}}{\partial w_{1,1}^{(2)}}$$

위 식의 오른쪽 항에 있는 3개의 미분 값을 하나하나 구해보겠다.

역전파의 출발노드인 a1(3)에서 Jtotal의 값은 J1이다. 따라서 위 식 오른쪽 항은 아래와 같이 계산된다.

$$\frac{\partial J_{total}}{\partial a_1^{(3)}} = \frac{\partial J_1}{\partial a_1^{(3)}} = \frac{1}{2}\frac{\partial}{\partial a_1^{(3)}}\left(a_1^{(3)} - y_1\right)^2 = \left(a_1^{(3)} - y_1\right)$$

$$\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = \phi\left(z_1^{(3)}\right)\left(1 - \phi\left(z_1^{(3)}\right)\right) = a_1^{(3)}\left(1 - a_1^{(3)}\right)$$

$$\frac{\partial z_1^{(3)}}{\partial w_{1,1}^{(2)}} = a_1^{(2)}$$

따라서

$$\frac{\partial J_{total}}{\partial w_{1,1}^{(2)}} = \left(a_1^{(3)} - y_1\right) \times a_1^{(3)}\left(1 - a_1^{(3)}\right) \times a_1^{(2)}$$

역전파에서 가중치 업데이트를 위해 사용되는 오차의 가중치에 대한 미분 값이 결국 역전파의 출발 노드의 활성 함수 값과 도착 노드의 활성 함수 값, 그리고 실제 값만으로 표현되는 것을 알 수 있다.

위 식의 오른쪽 항에서 처음 두 식의 값을 아래와 같이 ó1(3) 으로 둔다.

$$\delta_1^{(3)} = \frac{\partial J_1}{\partial z_1^{(3)}} = \left(a_1^{(3)} - y_1\right) \times a_1^{(3)}\left(1 - a_1^{(3)}\right)$$

역전파에 의해 업데이트 되는 w1,1(2)는 다음과 같다.

$$w_{1,1}^{(2)} = w_{1,1}^{(2)} - \delta_1^{(3)} a_1^{(2)}$$

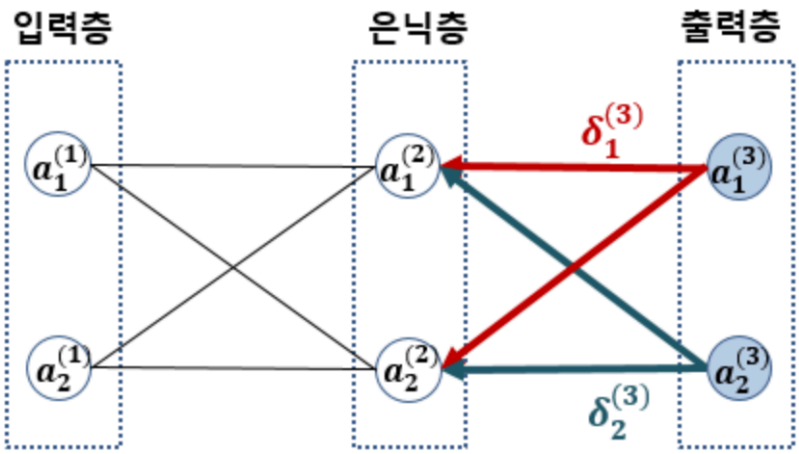
마찬가지 방법으로 w1,2(2)에 대한 업데이트 수식도 다음과 같다.

$$w_{1,2}^{(2)} = w_{1,2}^{(2)} - \delta_1^{(3)} a_2^{(2)}$$

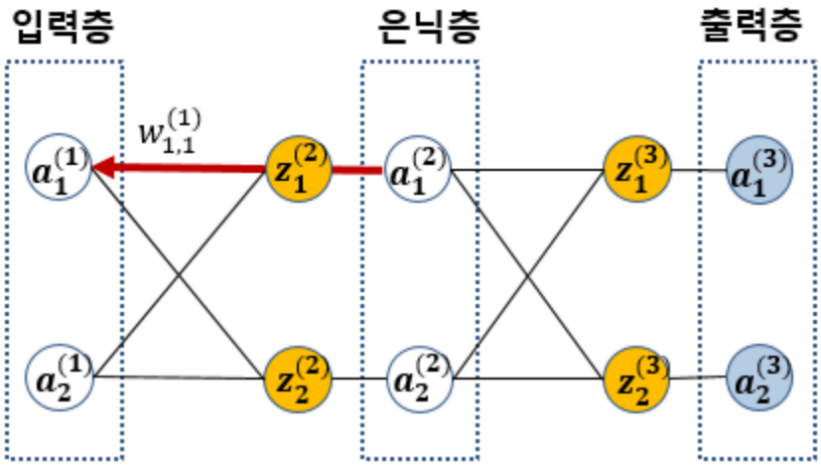
Ó2(3)을 아래와 같은 수식으로 정의 하고,

$$\delta_2^{(3)} = \frac{\partial J_2}{\partial z_2^{(3)}} = \left(a_2^{(3)} - y_2\right) \times a_2^{(3)}\left(1 - a_2^{(3)}\right)$$

여기까지 보다보면 규칙성이 보일 것이다. 출력층에서 은닉층으로 역전파를 통해 전달되는 값이 결국 ó1(3) 과 ó2(3)뿐이더라도 관련된 가중치를 업데이트 하는데 충분하다는 것을 알수 있다.



이번엔 은닉층에서 입력층으로 향하는 역전파를 통해 w1,1(1)의 값을 업데이트 해보겠다.



w1,1(1)을 업데이트 하기 위해서 a1(2)에서 Jtotal을 w1,1(1)로 편미분한 값을 계산해야한다. 앞에서 한 것과 마찬가지로 미분의 연쇄법칙을 활용하면 다음과 같은 식이 된다.

$$\frac{\partial J_{total}}{\partial w_{1,1}^{(1)}} = \frac{\partial J_{total}}{\partial a_1^{(2)}} \times \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \times \frac{\partial z_1^{(2)}}{\partial w_{1,1}^{(1)}}$$

그런데 출력층-> 은닉층으로 역전파를 통한 가중치를 계산시에 사용한 식들을 참고하여 계산해보면 다음과 같은 결과가 나온다.

$$\frac{\partial J_{total}}{\partial a_1^{(2)}} = \frac{\partial J_1}{\partial a_1^{(2)}} + \frac{\partial J_2}{\partial a_1^{(2)}} = \frac{\partial J_1}{\partial z_1^{(3)}} \times \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} + \frac{\partial J_2}{\partial z_2^{(3)}} \times \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}$$

그런데 출력층 -> 은닉층으로 역전파를 통한 가중치 계산시에 등장한 식들을 계산해보면 다음과 같은 결과가 나온다.

$$\frac{\partial J_{total}}{\partial a_1^{(2)}} = \delta_1^{(3)} w_{1,1}^{(2)} + \delta_2^{(3)} w_{2,1}^{(2)}$$

나머지 편미분 값은 이미 해 보아서 쉽게 계산이 될 것이다. W1,1(1)을 업데이트 하기 위한 편미분 값은 다음과 같다.

$$\frac{\partial J_{total}}{\partial w_{1,1}^{(1)}} = \left(\delta_1^{(3)} w_{1,1}^{(2)} + \delta_2^{(3)} w_{2,1}^{(2)}\right) \times a_1^{(2)} \left(1 - a_1^{(2)}\right) \times a_1^{(1)}$$

위 식의 오른쪽 항에서 처음 두 식을 ó1(2)로 둔다.

$$\delta_1^{(2)} = \left(\delta_1^{(3)} w_{1,1}^{(2)} + \delta_2^{(3)} w_{2,1}^{(2)}\right) \times a_1^{(2)} \left(1 - a_1^{(2)}\right)$$

최종적으로 w1,1(1)의 업데이트 식은 아래와 같다.

$$w_{1,1}^{(1)} = w_{1,1}^{(1)} - \delta_1^{(2)} a_1^{(1)}$$

동일하게 w1,2(1)의 업데이트 식은 아래와 같다.

$$w_{1,2}^{(1)} = w_{1,2}^{(1)} - \delta_1^{(2)} a_2^{(1)}$$

마찬가지로, ó2(2)를 다음과 같이 두고,

$$\delta_2^{(2)} = \left(\delta_1^{(3)} w_{1,2}^{(2)} + \delta_2^{(3)} w_{2,2}^{(2)}\right) \times a_2^{(2)} \left(1 - a_2^{(2)}\right)$$

W2,1(1), w2,2(2)에 대한 업데이트 식을 나타내면 다음과 같다.

$$w_{2,1}^{(1)} = w_{2,1}^{(1)} - \delta_2^{(2)} a_1^{(1)}$$

$$w_{2,2}^{(1)} = w_{2,2}^{(1)} - \delta_2^{(2)} a_2^{(1)}$$

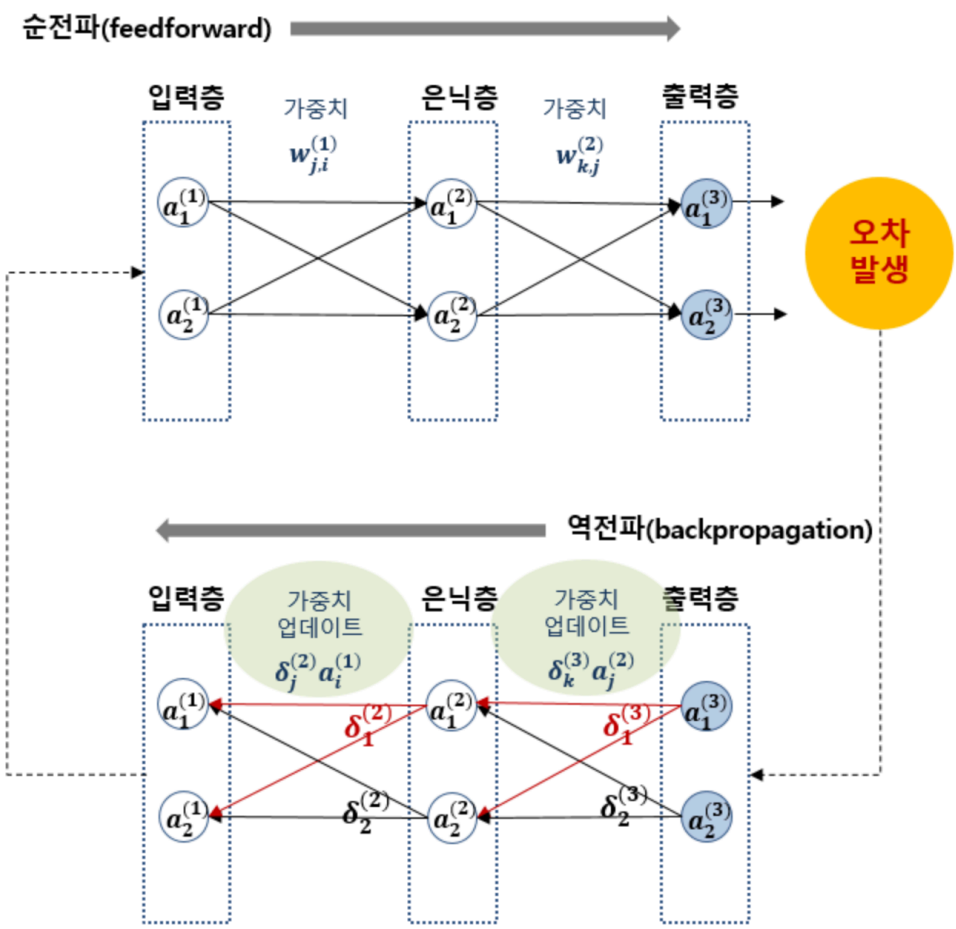
따라서 모든 가중치에 대한 업데이트 식은 아래와 같이 표현 가능하다.

$$\boldsymbol{w_{j,i}^{(l)} = w_{j,i}^{(l)} - \delta_j^{(l+1)} a_i^{(l)}}$$

$$\boldsymbol{\delta_j^{(3)} = \left(a_j^{(3)} - y_j\right) \times a_j^{(3)} \left(1 - a_j^{(3)}\right)}$$

$$\boldsymbol{\delta_j^{(2)} = \left(\delta_1^{(3)} w_{1,j}^{(2)} + \delta_2^{(3)} w_{2,j}^{(2)}\right) \times a_j^{(2)} \left(1 - a_j^{(2)}\right)}$$

여태까지 다룬 역전파 알고리즘 개념을 그림으로 도식화 하여 나타내보면 다음과 같다.



이와 같이 순전파 -> 역전파 -> 가중치 업데이트 -> 순전파... 로 계속 반복하다 보면 오차 값이 0에 가까워지게 된다.

2. Design

A. Main Idea

1. 각 층에서의 가중치를 임의의 값으로 설정한다. 각 층에서 bias 값은 1로 설정한다.
2. 하나의 트레이닝 데이터에 대해 각 층에서의 순입력 함수 값을 계산하고 활성함수에 의한 출력 값을 계산한다.
3. 출력층의 활성 함수에 의한 결과 값이 실제 값과의 오차가 줄어들도록 각 층에서의 가중치를 업데이트한다.
4. 모든 데이터에 대해서 출력층의 활성 함수에 의한 결과 값이 실제 값과의 오차가 매우 작아지면 학습을 종료한다.

B. Program Description

1.1.1 라이브러리 호출

```
import csv
import pylab as plt
import numpy as np
```

1.1.2 Data 불러오기

```
x = []
x_list = []
y = []
y_list = []

f = open('hw3_data.csv', 'r', encoding='utf-8')
rdr = csv.reader(f)

i = 0
for line in rdr:
    x.append([float(line[0])])
    x_list.append(float(line[0]))
    y.append([float(line[1])])
    y_list.append(float(line[1]))
    i += 1
f.close()

x = np.array(x).T
y = np.array(y).T
```

- x, y는 각각 data 들이 담길 행렬이고, x_list와 y_list는 데이터 분포를 확인하기 위해 데이터의 최대 값과 데이터의 개수를 구할 때 사용될 리스트이다.
- x, y는 각각 전치를 통해 nX1 행렬이 되도록 만들었다.

1.1.3 Data 분포 확인

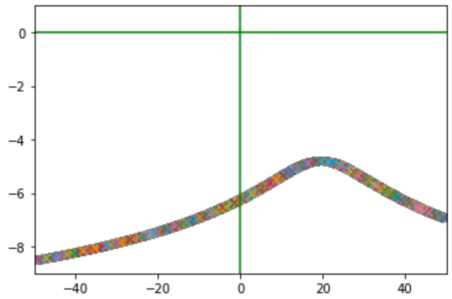
```
print("Number of x data: ", len(x_list), "Number of y data: ", len(y_list))
print("max(x):", max(x_list), "min(x):", min(x_list))
print("max(y):", max(y_list), "min(y):", min(y_list))
```

```
Number of x data:  5000 Number of y data:  5000
max(x): 49.9839286 min(x): -49.99516554
max(y): -4.785407023 min(y): -8.507007096
```

- data 분포는 다음과 같이 나온다.
- 각 데이터들의 개수는 5000개 이고 x는 -50 ~ 50이고 y는 -9 ~ -4와 같은 분포로 보인다.

그래프 확인

```
# 그래프 그리기
plt.plot(x,y,'x')
plt.xlim([-50,50]); plt.ylim([-9,1])
plt.axvline(x=0, c='g'); plt.axhline(y=0, c='g')
plt.show()
```



- 데이터분포를 통해 그래프를 그려 보았다.

1.1.4 Multi-Layer Perceptron Training

```
# 활성 함수
def sigmoid(x):
    return 1/(1+ np.exp(-x))

def de_sigmoid(x):
    return sigmoid(x) * (1 - sigmoid(x))

def identity(x):
    return x

def de_identity(x):
    return 1
```

- 활성 함수는 sigmoid와 identity(f(x) = x)를 사용하였다.
- 은닉층의 활성함수는 sigmoid를 사용하였고, 출력층의 활성함수는 identity를 사용하였다.

```
# Input 노드와 Output 노드의 수
inputNode = 5000
outputNode = 5000

# Hidden 노드의 수
hiddenNode = 100
```

- 모델은 다음과 같이 설정하였다.
- 앞에서 확인해본 결과 x, y의 데이터 개수가 각각 5000개 이므로 입력층과 출력층의 노드의 개수를 둘 다 5000으로 설정하였다. 은닉층의 노드 수는 임의의 수로 100으로 설정하였다.

```python
# Hidden Weight
hiddenWeight = np.random.rand(inputNode, hiddenNode)

# Output weight
outputWeight = np.random.rand(hiddenNode, outputNode)

# Learning rate
alpha = 0.1
```

- Weight는 각 층에 맞는 사이즈로 랜덤으로 받아오도록 하였다.
- Learning rate는 0.1로 설정하였다.

```python
# Training
for i in range(10000):
    hidden_z = np.dot(x, hiddenWeight)
    hidden_a = sigmoid(hidden_z)

    output_z = np.dot(hidden_a, outputWeight)
    output_a = identity(output_z)

    # Backpropagation
    outputDelta = (y - output_a) * de_identity(output_z)
    hiddenDelta = outputDelta.dot(outputWeight.T) * de_sigmoid(hidden_z)

    # Update Weight
    outputWeight += alpha * hidden_a.T.dot(outputDelta)
    hiddenWeight += alpha * x.T.dot(hiddenDelta)
```

- Iteration은 100,000으로 설정하였다.
- 각 층의 z는 각 층의 input과 weight를 내적한 값이고, 각 층의 a는 이 값을 sigmoid 함수에 적용시킨 값이다. 은닉층의 a와 출력층의 input은 bias의 존재 때문에 다르다.
- Weight를 업데이트 시키며 for문을 반복하는데, 이 때 weight 업데이트는 다음과 같다.

$$w := w - \alpha \frac{\partial J}{\partial w}$$

- 비용함수는 MSE를 사용하였다. 즉, 비용함수 J는 $\frac{1}{2}\|a-y\|^2$ 이다.
  여기서, $\frac{\partial J}{\partial w} = \frac{\partial J}{\partial a} * \frac{\partial a}{\partial z} * \frac{\partial z}{\partial w}$를 적용시킬 것이다. $\frac{\partial J}{\partial a} = (a-y)$이고, $\frac{\partial a}{\partial z} = a_1(z)$ 이다. $\frac{\partial J}{\partial a} * \frac{\partial a}{\partial z} = \delta$ 라고 하므로, 따라서 outputDelta = (y – ouput_a) * de_sigmoid(output_z) 이다.
- 계속해서, $\frac{\partial z}{\partial w}$는 현재 Layer의 input이다. 즉 출력층의 $\frac{\partial z}{\partial w}$는 output이고, 은닉층의 $\frac{\partial z}{\partial w}$는 x인 것이다.
  따라서 $\frac{\partial J}{\partial w}$는 각 층의 delta와 input을 곱한 값이므로, $w := w - \alpha \frac{\partial J}{\partial w}$를 적용시키면,
  outputWeight += alpha *hidden_a.T.dot(outputDelta)
  hiddenWeight += alpha * x.T.dot(hiddenDelta)가 된다.

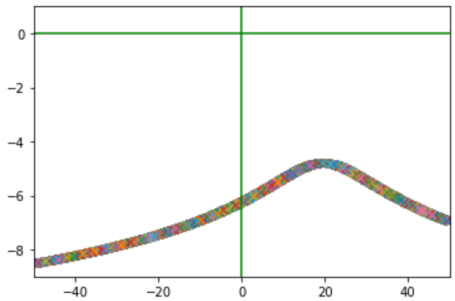
```
최종 Hidden weight :
 [[0.06165739 0.26770361 0.08686564 ... 0.88710334 0.58231191 0.44341846]
 [0.99375456 0.17960656 0.43743646 ... 0.41574805 0.96917024 0.3794092 ]
 [0.14334973 0.1176099  0.03426269 ... 0.2150695  0.39161545 0.91975341]
 ...
 [0.79285235 0.64466827 0.5455411  ... 0.94121563 0.71782666 0.9906738 ]
 [0.75691732 0.15962741 0.58344447 ... 0.09525246 0.69005783 0.00812672]
 [0.73297739 0.76278213 0.20827114 ... 0.27365102 0.0761395  0.18718326]]
최종 Output Weight :

 [[0.46455862 0.8551234  0.78025272 ... 0.1518097  0.85228216 0.669021  ]
 [0.39353199 0.44783996 0.25308837 ... 0.40531636 0.18149946 0.0912476 ]
 [0.894267   0.186512   0.13538294 ... 0.09356372 0.49350896 0.10382368]
 ...
 [0.81281294 0.2068569  0.69201154 ... 0.53911242 0.34038733 0.8484791 ]
 [0.93565598 0.61273229 0.44223388 ... 0.98291079 0.85976625 0.63473517]
 [0.70017247 0.21643951 0.75172728 ... 0.68704447 0.4976517  0.28682977]]
Ouput :
 [[-7.42126933]
 [-7.43564206]
 [-5.67391193]
 ...
 [-7.08351715]
 [-8.4314606 ]
 [-4.93850809]]
```

- 위 알고리즘을 통해 구한 weight는 다음과 같다.
- 출력은 처음 random으로 받은 각 층의 weight값, 학습을 통해 정해진 final weight값과 학습을 통한 출력을 순서대로 하였다.

1.1.5 최종 그래프 확인

```python
# 그래프 그리기
plt.plot(x,y,'x')
plt.xlim([-50,50]); plt.ylim([-9,1])
plt.axvline(x=0, c='g'); plt.axhline(y=0, c='g')
plt.show()
```



3. Evaluation Results

A. Evaluation Setup

Contents

- 1 HW3
  - 1.1 Neural Network
    - 1.1.1 라이브러리 호출
    - 1.1.2 Data 불러오기
    - 1.1.3 Data 분포 확인
    - 1.1.4 Multi-Layer Perceptron Training
    - 1.1.5 최종 그래프 확인
  - 1.2 Evaluation Result
    - 1.2.1 Training/Validation error

Build Environment

Jupyter notebook, Python3

B. Training/Validation error

1.2.2 Training/Validation error

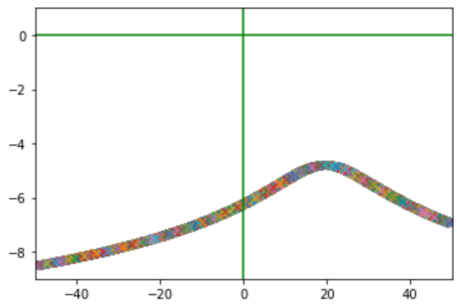
```
error = y - output_a
print(error)
```

```
[[8.88178420e-16 0.00000000e+00 0.00000000e+00 ... 8.88178420e-16
  1.77635684e-15 8.88178420e-16]]
```

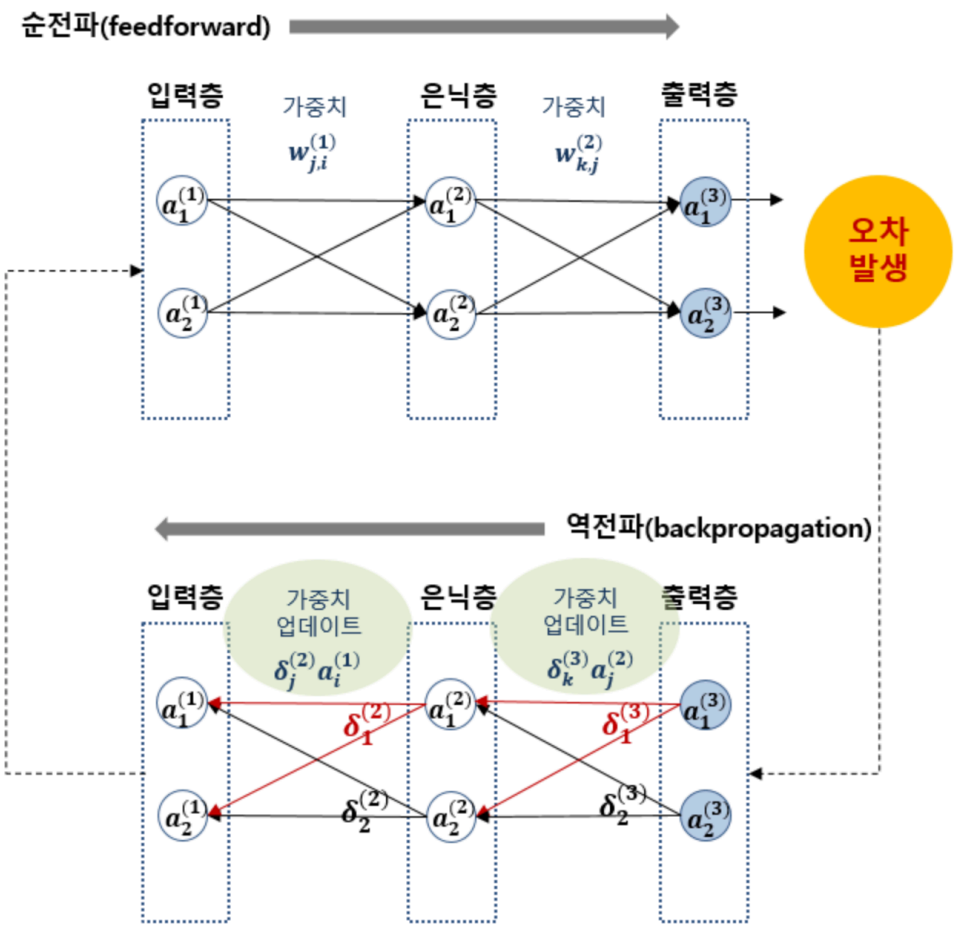
C. Final Model and Parameters

```
최종 Hidden weight :
 [[0.06165739 0.26770361 0.08686564 ... 0.88710334 0.58231191 0.44341846]
 [0.99375456 0.17960656 0.43743646 ... 0.41574805 0.96917024 0.3794092 ]
 [0.14334973 0.1176099  0.03426269 ... 0.2150695  0.39161545 0.91975341]
 ...
 [0.79285235 0.64466827 0.5455411  ... 0.94121563 0.71782666 0.9906738 ]
 [0.75691732 0.15962741 0.58344447 ... 0.09525246 0.69005783 0.00812672]
 [0.73297739 0.76278213 0.20827114 ... 0.27365102 0.0761395  0.18718326]]
최종 Output Weight :

 [[0.46455862 0.8551234  0.78025272 ... 0.1518097  0.85228216 0.669021  ]
 [0.39353199 0.44783996 0.25308837 ... 0.40531636 0.18149946 0.0912476 ]
 [0.894267   0.186512   0.13538294 ... 0.09356372 0.49350896 0.10382368]
 ...
 [0.81281294 0.2068569  0.69201154 ... 0.53911242 0.34038733 0.8484791 ]
 [0.93565598 0.61273229 0.44223388 ... 0.98291079 0.85976625 0.63473517]
 [0.70017247 0.21643951 0.75172728 ... 0.68704447 0.4976517  0.28682977]]
Ouput :
 [[-7.42126933]
 [-7.43564206]
 [-5.67391193]
 ...
 [-7.08351715]
 [-8.4314606 ]
 [-4.93850809]]
```



4. Conclusion



순전파 -> 역전파 -> 가중치 업데이트 -> 순전파... 로 계속 반복하다 보면 오차 값이 0에 가까워지게 된다.

원래는 뉴런 네트워크를 위의 그림처럼 만들어야 하지만 시간이 부족하여 완벽하게 했다고는 말하지 못하지만 최선을 다해 시간내에 할 수 있는데 까지 열심히 만들어 보았다.