

# Report

HW#2 Neural Network  
Machine Learning

Dpt. of Mobile System Engineering  
32192530 양윤성

2023.11.24.

## ● Contents

<b>1. Introduction</b>	<b>1p</b>
1-1. Perceptron Model	1p
1-2. Activation Functions	2p
1-3. MultiLayer Perceptron	3p
1-4. Backpropagation	4p
<b>2. Implementation</b>	<b>6p</b>
2-1. Idea and Design	6p
2-2. Layer	7p
2-3. Model	8p
<b>3. Environment</b>	<b>10p</b>
3-1. Build Environment	10p
3-2. Compile	10p
3-3. Results	11p
<b>4. Conclusion</b>	<b>18p</b>

# 1. Introduction

## 1-1. Perceptron Model

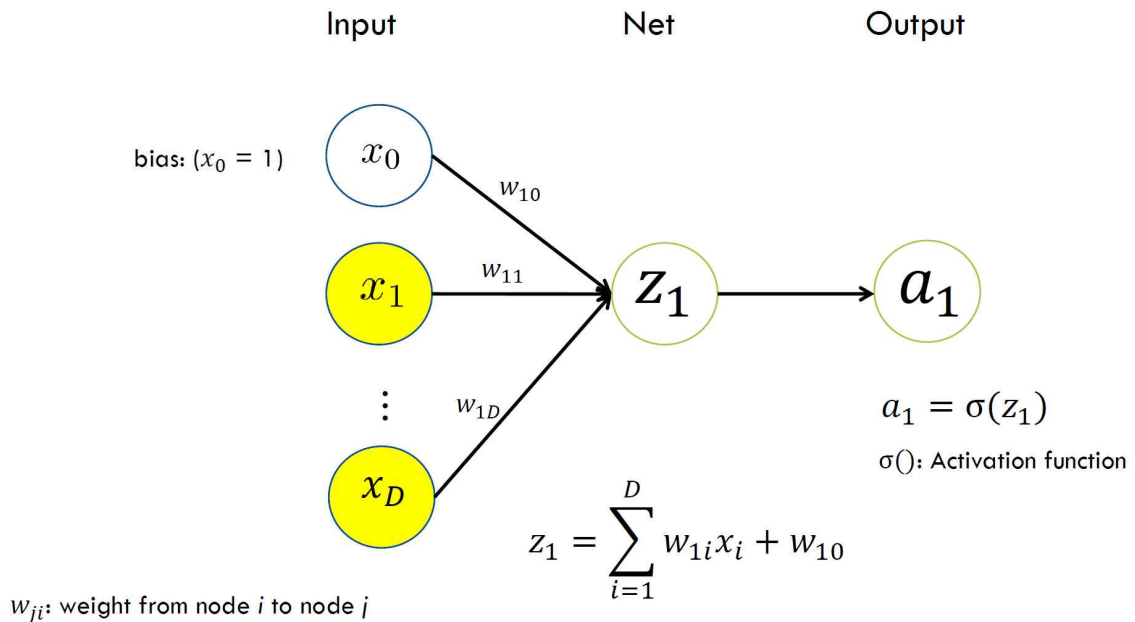


그림1. Perceptron Model (단일 레이어)

Perceptron Model은 인간의 뇌를 모방하여 개발한 신경망이다. 이 모델은 모든 input들과 가중치(weight)들을 곱한 후 더해서 가중치 합을 만들고, activation function을 통과시켜 최종 output을 출력한다. 위 그림은 단일 레이어 상태인데, 그러면 선형적으로 구분되는 데이터에서만 효과적이고 XOR와 같이 비선형적인 상황은 해결하지 못한다는 한계가 있다. 그래서 이를 여러 층을 쌓아 MultiLayer Perceptron(MLP)을 구현하는 것이 널리 알려진 방법이고 이번 과제의 핵심 개념이다. 이 MLP는 activation function을 먼저 알아보고 살펴볼 예정이다.

## 1-2. Activation Functions


앞서 Perceptron Model은 input과 weight들을 모두 곱해서 더한다고 했는데, 이 값을 적절하게 output의 형태로 출력해야 한다. 이 역할을 활성화 함수(activation function)이라고 하며, 이 함수들은 주로 비선형적인 함수가 사용된다. 비선형성을 추가하여 신경망이 더 복잡한 관계를 학습하기 위함이다. 그래서 이번 과제에서 구현한 3가지의 activation function을 소개하고자 한다.

### ① Identity Function

Identity		$f(x) = x$	$f'(x) = 1$
----------	-----------------------------------------------------------------------------------	------------	-------------


Identity Function은  $y=x$ 로 표현되는 입력과 출력이 동일하다. 이 함수는 유일하게 선형성을 가지므로 Identity function으로 여러 층을 쌓아도 효과가 미미하다. 따라서 input layer나 중간 hidden layer보다는 주로 마지막 output layer에만 사용한다.

### ② Sigmoid Function

Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ [1]	$f'(x) = f(x)(1 - f(x))$
----------------------------------------	-------------------------------------------------------------------------------------	-----------------------------------------------	--------------------------

Sigmoid Function은 출력 범위가 0~1사이로 제한되어 있는, logistic regression에서 주로 사용하는 함수이다. 그러나 sigmoid는 input값이 너무 커지면 gradient가 매우 작아져 학습 중 gradient가 손실되는 문제나, 계산이 길어지는 문제로 인하여 hidden layer에서 잘 사용하지 않는다.

### ③ Hyperbolic Tangent Function

TanH		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$
------	-------------------------------------------------------------------------------------	-------------------------------------------------------	----------------------

Tanh이라고도 불리는 함수로 sigmoid와 비슷하지만 범위가 -1~1로 확장되어 있다. 이로 인해 데이터들의 평균이 0 주변에 위치한다. 그래서 sigmoid보다 gradient들이 적절한 크기를 갖고 있어 gradient descent가 더 안정적이고 효과적으로 수행할 수 있고, sigmoid에서 문제가 되는 gradient 손실 문제를 어느정도 막을 수 있다.

### 1-3. MultiLayer Perceptron

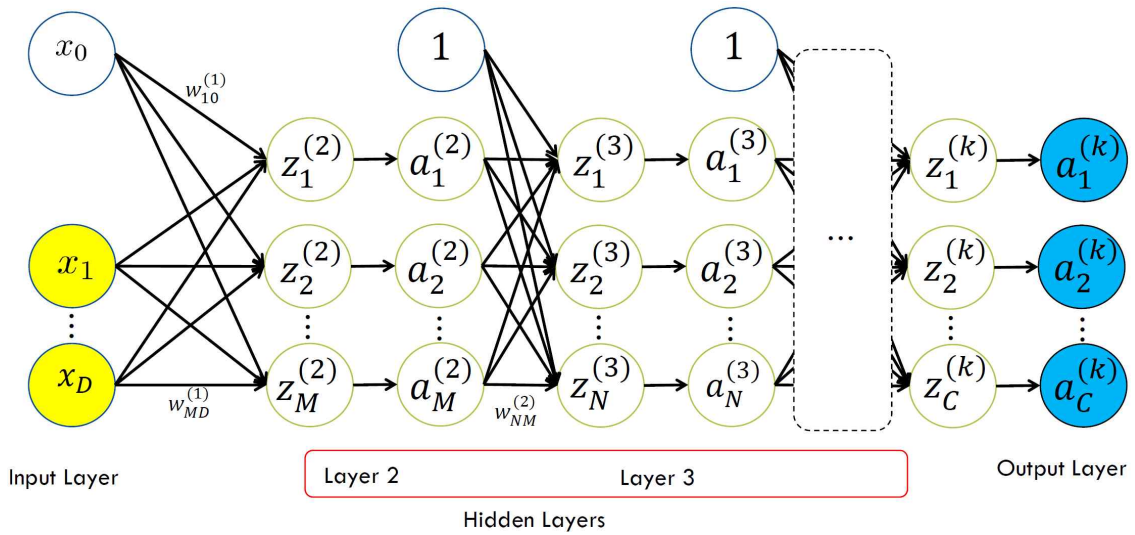


그림2. MultiLayer Perceptron

MultiLayer Perceptron(MLP)는 Perceptron model을 기반으로 여러 layer들로 구성된 neural network이다. Layer는 input layer와 output layer가 있고 그 사이에 비선형의 hidden layer이 여러 존재할 수 있다. 각 layer 층마다 activation function이 있고 층 안의 노드들은 각각의 weight값을 갖고 있는데, 이전 층의 output을 입력으로 받아 각각의 weight를 적용하고 activation function을 통과시켜 다음 층으로 전달해준다. 이렇게 다양한 구조와 크기를 갖는 Layer들이 있기에 MLP는 더욱 복잡한 데이터들도 학습 가능하다. 이 모델을 학습하기 위해 주로 사용하는 알고리즘이 바로 Backpropagation(역전파) 알고리즘이다.

## 1-4. Backpropagation

MLP에서 가중치를 업데이트하는 방법은 feed-forward 방식으로 gradient descent를 하는 것이 일반적이다. 그러나 이는 입력부터 출력까지 모두 순차적으로 진행되기 때문에 모델이 복잡할수록, 즉 layer가 많아질수록 계산의 복잡도와 요구되는 메모리가 크게 증가한다는 단점이 있다. 이러한 한계를 극복하기 위해 Backpropagation이라는 역전파 알고리즘을 사용하여 학습한다. Output layer에서 계산된 오차(loss)값을 전 단계의 layer들에 역방향으로 전파하며 각각 gradient를 계산하고, 이를 gradient descent로 weight를 다시 업데이트하여 loss를 최소화한다. 이러한 Backpropagation의 계산법은 chain rule의 원리에서 유래했다.

The formula  $f(x) = f(g(x))$  means  
that  $f$  is a function of  $g$  and  $g$  is a function of  $x$ .

We can calculate  $\frac{\partial f}{\partial x}$  indirectly as follows:

$$\frac{\partial f}{\partial x} = \frac{\partial f(g)}{\partial g} \cdot \frac{\partial g(x)}{\partial x}$$

그림3. Chain Rule

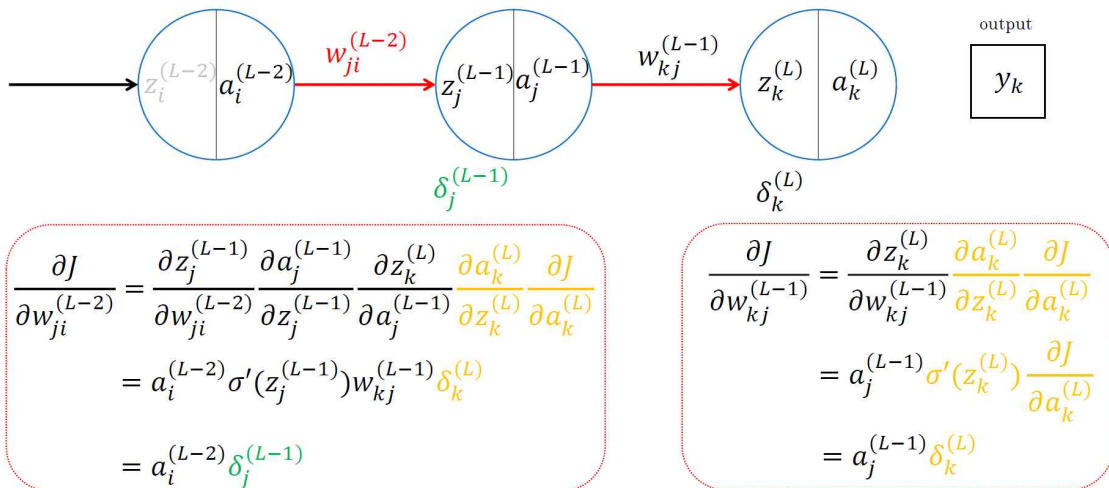


그림4. Hidden layer가 2개일 때 Backpropagation

그림4에서는 Backpropagation을 통해 L-1 layer과 L-2 layer의 loss를 구하고 있다. 이렇게 loss값을 구하며 각 layer의 weight를 업데이트하는 것이다.

```

Repeat
  Initialize  $\Delta^{(l)}, \mathbf{D}^{(l)}$  for  $l = 1, 2, \dots, L - 1$ 
  Select  $M$  data samples randomly
  for  $m = 1 : M$ 
    Perform Forward Propagation to compute  $\mathbf{a}^{(l)}$  for  $l = 2, 3, \dots, L$ 
    Using  $\mathbf{y}$ , compute  $\delta^{(L)} = \mathbf{a}^{(L)} - \mathbf{y}$ 
    Compute  $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ 
     $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(\mathbf{a}^{(l)})^T$ 
     $\delta^{(l)} = (\mathbf{W}^{(l)})^T \delta^{(l+1)} \odot \sigma'(\mathbf{z}^{(l)})$ 
     $\mathbf{D}^{(l)} := \frac{1}{m}(\Delta^{(l)} + \lambda \mathbf{W}^{(l)})$ 
     $\mathbf{W}^{(l)} := \mathbf{W}^{(l)} - \alpha \mathbf{D}^{(l)}$ 
  for  $l = 1 : L - 1$ 

```

batch size

not counting bias units

For bias terms,  $\lambda=0$

### 그림5. Backpropagation Algorithm

그림5는 위의 Backpropagation 알고리즘을 Pseudocode로 작성한 것이다. for m=1:M~ 부분부터가 핵심인데, 먼저 전체 training data에 대해 Forward propagation을 진행한다. 여기서는 데이터를 순차적으로 각 layer별로 activation function 값을 계산하고(a2, a3, ..., aL) 해당 값과 실제값 y의 오차를 계산한다(delta2, delta3, ..., deltaL). 이 과정을 모두 마친 후에 반대로 Back Propagation을 해준다. 이제 역방향으로 오차를 구해주고 여기에 activation의 미분값을 곱해준다(deltaL-1, deltaL-2, ..., delta2). layer별로 delta(오차)값은 기존 weight의 transpose와 곱한 뒤 앞의 layer로 전파되며 계산된다. L=2까지 모두 완료하면 모든 delta 값에 대해 l+1번째 layer의 delta값과 l번째의 layer의 activation function을 거친 값(forwarding한 값)을 외적해서 갱신해준다. 마지막으로 이들을 모두 누적해서 각 layer의 weight를 업데이트하게 된다. 이번 과제에서 이 알고리즘 베이스로 model의 training을 구현했는데, 이는 아래 Implementation에서 설명한다.

## 2. Implementation

### 2-1. Idea and Design

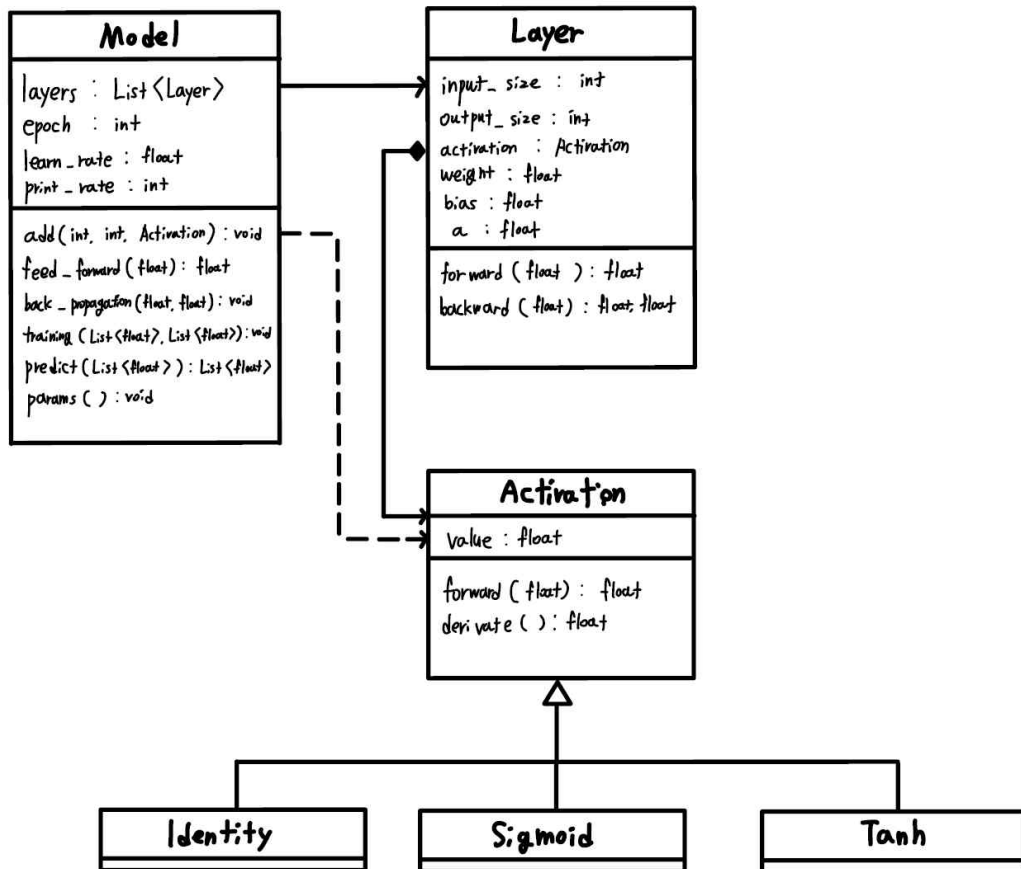


그림6. Class Diagram

그림6은 구현한 코드의 class diagram이다. 이번에는 저번 HW1와 달리 객체지향적인 코드를 구현하려 노력했다. Activation function은 Activation이라는 부모클래스를 만들고, 추후에 activation function을 쉽게 추가하여 다양한 테스트를 하기 위해 Activation 클래스를 상속받게끔 설계했다. 이번 과제에서는 3가지의 activation function을 구현하였다 (Identity, Sigmoid, Tanh). 또한 model과 layer클래스를 나누어 association 관계로 구성하였다. Layer별로 갖는 input\_size, output\_size, activation function이 모두 다를 것이고, 이들은 layer의 weight와 bias에도 영향을 미친다. 이들을 하나의 class에 넣으면 너무 복잡한 객체가 만들어진다고 판단했고, 따라서 layer만 따로 model과 구분하여 class로 구현해 주었다.



## 2-2. Layer

```
class Layer:
    def __init__(self, input_size, output_size, activation : Activation):
        self.input_size = input_size
        self.output_size = output_size
        self.activation = activation
        self.weight = np.random.randn(output_size, input_size) * np.sqrt(1. / output_size)
        self.bias = np.random.randn(output_size, 1)
        self.a = None

    def forward(self, a):
        self.a = a
        w_sum = np.dot(self.weight, a) + self.bias
        val = self.activation.forward(w_sum)
        return val

    def backward(self, diff):
        loss = diff * self.activation.derivate()
        delta = np.outer(loss, self.a)
        diff = np.dot(self.weight.T, loss)
        return delta, diff
```

$$\delta^{(l)} = (W^{(l)})^T \delta^{(l+1)} * \sigma'(z^{(l)})$$

$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

그림7. class Layer

Layer 클래스에는 layer가 갖고 있어야하는 속성들과 기능을 구현하였다. layer별로 다른 activation function을 설정하게 하고 forwarding 시에는 normal한 값을, backpropagation 시에는 미분한 값을 받아 계산하도록 했다. layer의 weight와 bias는 input size와 output size에 따라 다른 난수를 이용해서 초기화했다. 이때 weight는 Xavier 초기화 기법을 사용했는데, weight가 특정 범위 내로 균등하게 분포되도록 하여 gradient가 손실되거나 과도하게 높아지는 문제를 해결하였다. forward는 매개변수 a를 받아서 self.a를 초기화하는데, 여기서 a값은 이전 layer의 forward 값이고 backward를 위해 저장해준다. 이후 layer의 weight와 내적인 후 bias를 더한 다음 activation function을 통과한 값을 return해준다. 이는 background에서 설명한 가중치 합을 구하는 과정이다. backward는 앞의 layer로부터 diff(오차값)을 받아서 activation function의 미분값과 곱하고 forward 함수에서 저장해준 self.a값과 외적해서 delta(최종 weight)를 구하고 diff도 다시 계산해서 앞의 layer로 전파해준다.

## 2-3. Model

```
class Model:
    def __init__(self, epochs, learn_rate, print_rate):
        self.layers = []
        self.epochs = epochs
        self.learn_rate = learn_rate
        self.print_rate = print_rate

    def add(self, input_size, output_size, activation : Activation):
        self.layers.append(Layer(input_size, output_size, activation))

    def forward_propagation(self, x):
        a = x
        for layer in self.layers:
            a = layer.forward(a)
        return a

    def back_propagation(self, y, y_pred):
        diff = (y_pred - y) / y_pred.shape[0]
        update = []
        for layer in list(reversed(self.layers)):
            delta, diff = layer.backward(diff)
            update.append(delta)
        for delta, layer in zip(reversed(update), self.layers):
            layer.weight -= self.learn_rate * delta

    def training(self, x_data, y_data):
        for epoch in range(1, self.epochs+1):
            train_loss = []
            for x, y in zip(x_data, y_data):
                y_pred = self.forward_propagation(x)
                mse = 0.5 * ((y - y_pred) ** 2)
                train_loss.append(mse)
                self.back_propagation(y, y_pred)
            train_loss = np.array(train_loss).mean()

            if (epoch % self.print_rate == 0):
                print(f"Epoch : {epoch}\t\tMSE : {train_loss}")

    def predict(self, x):
        tmp = []
        for _x in x:
            tmp.append(self.forward_propagation(_x))
        return np.array(tmp)
```

$$\delta^{(L)} = a^{(L)} - y$$

$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

그림8. class Model (파라미터 출력 함수 제외)

이번 과제의 가장 중심이 되는 부분으로 모델의 구조를 구현한 클래스이다. 모델은 layer 객체를 리스트로 갖고 있고, training을 위한 epoch과 learning rate를 사용자로 부터 받아 초기화한다. print\_rate는 epoch를 얼마 단위로 출력할 것인지를 결정하는 수치이다. 모델은 add method로 input\_size, output\_size, activation객체를 받아서 layer 리스트에 추가한다. 그 아래로는 모두 모델 학습과 관련된 코드인데, 메인 로직인 training 함수가 핵심이다. x, y데이터를 받아와서 epoch만큼 training하는데, 우선 forward propagation을 진행한다. x데이터를 순차적으로 layer에 넣어 계속 forward시키면서 최종적으로 y\_pred라는 예측값을 만든다. 그럼 y\_pred와 실제 값인 y로 squared error를 계산하고 나중에 mse 계산을 위해 train\_loss 리스트에 append해준다. 그후 y\_pred와 y값으로 backpropagation을 진행한다. 처음에는 y\_pred와 y값의 차이를 구해서 정규화해준 다음 layer 리스트의 역순으로 layer별 backward하는 과정을 거친다. layer에서 나오는 diff(오차값)는 이전 layer로 전파되어 계속 계산되고, delta는 update 리스트에 차례대로 append되어 나중에 갱신된 weight들이 모이게 된다. 이 weight들은 각 layer별 learning rate와 곱해서 갱신해준다. 모든 데이터가 위 과정을 마치면 train\_loss 리스트에 저장된 loss값들의 평균을 구해서 mse를 구해준다. 이 값이 training mse이다. 이 로직과 앞서 말한 Part1-4 그림5에 있는 pseudocode는 차이가 존재하는데, 기존 pseudocode에서는 모든 데이터들을 한번에 forward propagation을 하고 back propagation을 진행하는 반면 코드 로직은 데이터들을 하나씩 forwarding → back propagation 과정으로 처리하여 weight 업데이트를 진행한다. 마지막으로 predict 함수는 테스트를 위한 함수로 테스트 데이터셋을 받아서 forwarding시키며 나오는 최종 예측값을 return한다. 이 값으로 실제 y값과 mse를 계산해서 validation mse를 구한다.

## 3. Environment

### 3-1. Build Environment

Google Colaboratory virtual CPU

T4 GPU

Python 3.10.12

Pandas 1.5.3

NumPy 1.23.5

### 3-2. Compile

1. HW2\_32192530\_code.zip 압축풀고 32192530\_hw2.ipynb, hw2\_data.csv 저장
2. Google Colab 접속 → 업로드 → 둘러보기 클릭해서 hw2.ipynb 파일 불러오기
3. 우측 위의 연결 옆 화살표 클릭
4. 런타임 유형 변경 → 하드웨어 가속기를 T4 GPU로 변경
5. 우측 위의 연결 클릭
6. 연결이 정상적으로 되면 **drive.mount('/content/drive')**까지만 코드 실행  
(구글 드라이브 연동)
7. 왼쪽의 폴더 아이콘 클릭 후 drive/MyDrive에 hw2\_data.csv 업로드
8. 이후 코드도 모두 실행

### 3-3. Results

```
model = Model(1000, 0.001, 50)
model.add(1, 16, Tanh()) #input layer
model.add(16, 32, Tanh()) #hidden layer
model.add(32, 16, Tanh()) #hidden layer
model.add(16, 1, Identity()) #output layer

np.random.seed(1124)
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.2, random_state=1124)
model.training(x_train, y_train)
```

#### 그림9. Evaluation Setup

학습 전 모델 셋팅은 epoch값을 1000, learning rate를 0.001로 했고 training MSE를 출력하는 epoch 단위를 50으로 설정했다. 모델 내의 layer은 4개로 구성했으며, 각 layer의 input size는 1, 16, 32, 16으로 해주었다. Input layer와 hidden layer는 Sigmoid보다 좋은 performance를 보여주는 Tanh activation function을 사용하였고 마지막 output layer만 Identity activation function을 사용하였다. 모델을 구성한 후 hw2\_data.csv에서 가져온 데이터를 x와 y로 나누었고, 이를 또다시 training data set과 validation data set으로 나누었다. 이때 둘을 나누는 비율은 8:2로 구성했다 (test\_size=0.2). 또한 어느 환경에서든 일정한 난수값으로 동일한 결과가 나타나게끔 임의의 숫자로 시드를 고정해 놓았다(1124). 데이터셋 준비를 마치고 training data set으로 모델을 학습시키며 training error를 epoch 50마다 출력한다. 학습이 모두 끝나면 validation data set을 fitting이 완료된 모델에 넣어 값을 예측하고 validation mse와 예측 결과를 출력한다. 이들의 결과는 다음 페이지에 snapshot으로 첨부한다.

Epoch : 50	MSE : 0.007311736136646351
Epoch : 100	MSE : 0.0036509529419027644
Epoch : 150	MSE : 0.002375691107906756
Epoch : 200	MSE : 0.0018202958185887413
Epoch : 250	MSE : 0.0015028370452865555
Epoch : 300	MSE : 0.0012916842437933906
Epoch : 350	MSE : 0.001131359407842822
Epoch : 400	MSE : 0.001003162765347489
Epoch : 450	MSE : 0.0008986048800496735
Epoch : 500	MSE : 0.000811954932360077
Epoch : 550	MSE : 0.000738898645514738
Epoch : 600	MSE : 0.0006761817515096913
Epoch : 650	MSE : 0.0006213547370750496
Epoch : 700	MSE : 0.0005725452296644274
Epoch : 750	MSE : 0.0005282821853196431
Epoch : 800	MSE : 0.00048738723390488135
Epoch : 850	MSE : 0.00044894114750063894
Epoch : 900	MSE : 0.00041233126140026113
Epoch : 950	MSE : 0.00037736092272227807
Epoch : 1000	MSE : 0.0003443274415285464

Validation MSE : 2.6384643495627893

그림10. Training/Validation Error (MSE)

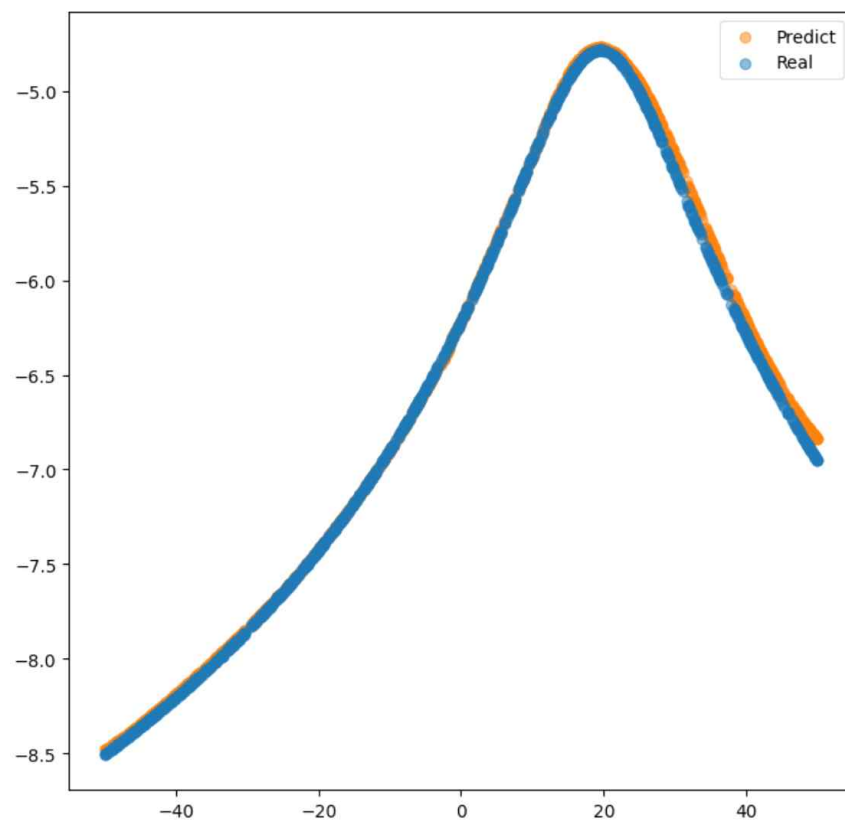


그림11. Final result plot

----- Layer 0 Info -----

Weight : [[ 0.15574732]

[-0.0282473 ]

[-0.02502161]

[ 0.02680205]

[ 0.07364103]

[ 0.52044712]

[ 0.49853732]

[-0.01704538]

[ 0.16870531]

[-0.14159634]

[-0.16671608]

[ 0.23695805]

[-0.44379412]

[ 0.05442475]

[-0.10688102]

[ 0.01744933]]

Bias : [[-1.36647348]

[-0.68329681]

[-1.17306562]

[ 0.77290822]

[ 0.96170935]

[ 1.21796403]

[ 1.80110592]

[-1.31053519]

[ 1.09138134]

[-0.32565506]

[-0.48367433]

[-0.74447295]

[-0.38702271]

[-0.68036143]

[-0.99094709]

[ 2.37716364]]

## 그림12. Optimal parameters of input layer

----- Layer 3 Info -----

Weight : [[ 0.24675228 0.21563241 -1.87654777 -1.4073187 0.19136884 -0.1910259

1.5673276 2.62321301 -1.15332637 -0.32254118 -1.27905409 0.78430224

1.56889712 -1.24133474 -0.74154166 -0.23483656]]

Bias : [[-0.07778667]]

## 그림13. Optimal parameters of output layer

```

----- Layer 1 Info -----
Weight : [[ 4.07838548e-02  2.57596672e-01 -1.23069080e-01  1.73341032e-01
 4.51683151e-01 -1.32559054e-01 -3.61561745e-04  1.81245535e-01
 1.46080887e-01 -3.38241179e-01  8.39415716e-02  1.08288622e-01
 2.40697751e-02 -1.15560277e-01 -8.93456125e-02  4.42021809e-02]
 [ 3.12943495e-02 -2.99377626e-01 -2.27050792e-02 -3.09579816e-02
 1.25933646e-01  1.40182287e-01  1.65266142e-01 -2.13509130e-01
 1.94512206e-01 -3.31495174e-01 -3.72817832e-03  1.17809974e-01
 -5.00342321e-01  1.35986930e-01 -5.16470626e-02  3.22467418e-02]
 [-2.29803787e-01  2.86028959e-01 -1.02264694e-01 -2.15550140e-01
 3.01606525e-01 -1.14546182e-02 -1.24143518e-02  1.24270125e-01
 4.85450868e-02 -4.28053088e-02 -1.48339094e-01 -4.32670326e-02
 5.08862687e-01 -3.12447215e-01  7.65345597e-03  2.30327482e-01]
 [ 2.59070591e-01 -3.35872232e-02 -2.08533649e-01  3.60349586e-02
 2.73144631e-01  2.94826372e-01 -2.23317970e-03 -5.58325048e-04
 -1.14765991e-01  5.34084561e-02  3.28629753e-01 -1.93085131e-01
 1.30648816e-01 -1.76902095e-01 -6.10472391e-02 -2.65647385e-01]
 [-1.87105578e-01  2.02461683e-01  8.52563224e-02  2.61894407e-02
 -1.77352505e-01 -6.56186099e-03 -3.03292426e-01  2.19505810e-01
 2.43924416e-02  1.60534927e-01  3.82266967e-01 -6.44621904e-01
 5.08627286e-02 -1.18950042e-01  1.47518843e-02 -7.88342534e-02]
 [-1.69771974e-01 -1.74623954e-01 -3.28596770e-02 -1.56243630e-01
 2.23327427e-02 -3.40783367e-02  1.62199853e-02  3.62925372e-02
 -2.04450067e-02  1.94831201e-01  1.51841095e-01  8.86528461e-02
 8.11638311e-02 -2.57254171e-01  2.24481740e-02  9.89088641e-02]
 [-1.21356322e-01 -2.59661608e-01 -1.07357607e-01  1.30638606e-01
 -9.82931177e-02 -1.20946995e-01  6.59357729e-02  3.44211128e-02
 -3.25164144e-01 -5.43888233e-03 -2.01142489e-01 -5.82580518e-03
 -1.34026874e-01  2.67625545e-02  2.06451873e-01  4.73547491e-02]
 [-8.03801108e-02  3.96182916e-03  6.23436299e-01 -1.08312222e-01
 4.58626251e-03  1.87688034e-01 -6.03359987e-02 -2.15054304e-01
 -6.50121131e-03 -2.11518033e-01  1.77716273e-01  4.38443398e-02
 -4.77272494e-02 -6.22967262e-02  8.10113397e-03 -1.55308158e-01]
 [-3.06247640e-01 -3.07888155e-02  5.16763658e-02 -2.65006119e-02
 3.59627826e-01 -3.11582598e-02  5.15756425e-02 -2.05242330e-01
 -2.52774612e-01  2.19973583e-01  7.83750686e-02  3.62082281e-01
 -1.61772998e-01 -2.21741746e-01  9.50621932e-02 -1.53840128e-01]
 [-7.29728532e-02  1.21105488e-01  2.14069749e-01 -9.23690497e-02
 3.05634252e-01  1.38919897e-01  1.89559436e-01  1.90743048e-01
 7.95206177e-02 -2.07162734e-01 -3.03916667e-01  2.10753465e-02
 -9.35507013e-02  2.39921498e-01  1.60889369e-01 -1.26542358e-01]
 [ 3.09352907e-01  1.53604375e-02 -1.29224368e-01 -9.59182483e-02
 1.44419841e-01  3.33944651e-01  1.87416740e-02  1.78671250e-01
 2.23812448e-01  1.66259293e-01 -8.22801731e-02 -1.20529666e-02
 -2.99290870e-01  2.52025020e-01  8.81873396e-02 -8.09414840e-02]
 [ 2.74535366e-01  2.35810559e-01  9.70846532e-02 -2.90325975e-01
 -1.07017325e-01 -9.33017025e-02  6.37528633e-02  5.61111481e-02
 4.90478927e-02  1.54587185e-02  4.22028620e-02  1.85183321e-01
 -2.53936615e-01  2.85773873e-02 -1.68108545e-01  1.89209269e-02]
 [-4.36141592e-01 -2.38974375e-01 -2.28983901e-01  4.11010682e-01
 -4.63082346e-01 -1.61977595e-01  1.45621735e-01 -2.55714725e-01
 -2.49441933e-03  9.75297967e-02  2.24057824e-01 -3.11518128e-01
 -1.84258337e-01  4.24358609e-01  9.96006449e-02  4.24181668e-01]
 [ 1.52613552e-01 -1.84881034e-01 -1.39144831e-01 -5.90617545e-02
 4.99138452e-01  4.79245416e-01 -1.26142741e-01 -2.70833762e-01
 2.17470081e-01 -2.29252452e-01 -2.79419217e-01  8.08380389e-02
 6.67617190e-02  3.07448796e-03  1.06725709e-01  1.19929455e-01]
 [-3.02534229e-01 -1.50318313e-01 -1.42048805e-01 -2.56463773e-02
 -4.84777552e-02 -8.27289834e-03 -7.87398533e-03  1.05712159e-01
 -8.39674308e-02 -1.59197235e-01 -2.20743932e-05  2.81734048e-01
 -1.84884770e-01  1.41567609e-01 -1.00922938e-02 -1.37582286e-01]

```



[ 2.32844273e-01 -1.34140979e-01 -8.72304647e-02 3.52543312e-01  
3.25105364e-01 -7.16654012e-02 1.22181250e-01 -9.44317035e-03  
-1.11758736e-01 -4.20249117e-01 -1.87665785e-01 1.82923163e-01  
-2.53853257e-01 -2.56665640e-01 3.87073195e-02 1.69961059e-01]  
[ 2.55595027e-01 -1.76982019e-01 -1.21296490e-01 -3.85189774e-01  
-1.39623882e-01 7.75705879e-02 -7.76938762e-02 1.15870370e-01  
4.81001499e-02 1.00081299e-01 -1.23824007e-01 -2.15098061e-01  
4.92270503e-02 -3.40696806e-02 -2.17747632e-01 2.70078595e-02]  
[ 3.08173438e-01 -1.43250359e-01 1.85571720e-01 9.91995201e-02  
-7.74204003e-02 -8.94925903e-02 -5.05576078e-03 -8.89894306e-04  
-1.88421797e-01 1.32793391e-02 2.03334259e-01 9.96482964e-02  
-7.37523538e-02 6.26540662e-01 3.21516728e-02 -2.37262893e-01]  
[-3.20328403e-01 2.27449636e-01 -3.45257356e-01 -2.18305105e-01  
3.20666334e-03 -1.92703298e-01 6.44493895e-02 -4.04128628e-01  
-2.14526031e-01 1.57300782e-01 -1.45250200e-01 6.65785848e-02  
1.15153241e-02 -7.34013455e-01 -2.51245053e-01 1.36112340e-01]  
[ 8.04826222e-02 -6.45478378e-03 1.79344098e-01 -8.36761004e-02  
3.96042876e-02 9.09056351e-02 4.30968153e-03 1.78771975e-01  
-8.99017194e-02 3.90266774e-02 -6.44729301e-02 -3.64994926e-02  
1.98435129e-02 6.50137923e-02 -2.05888281e-01 -1.17501197e-01]  
[-7.51028139e-02 -2.58652608e-01 -2.93066938e-01 1.84846129e-01  
-1.39552438e-01 1.48595606e-01 -4.58087880e-01 -2.04940996e-01  
-2.05181186e-02 1.64378018e-01 1.10553505e-01 3.17041345e-01  
1.11030088e-01 2.24452436e-01 2.54424143e-01 -4.37084599e-02]  
[ 3.52386647e-01 -1.93109415e-02 -2.02546835e-01 -3.39355153e-02  
2.17901740e-01 -1.13328665e-01 2.86787849e-01 -3.32187758e-02  
-3.46520865e-02 2.98638734e-01 1.46391649e-01 1.99852055e-01  
1.61816971e-01 1.44335565e-01 1.76140308e-01 2.02399537e-01]  
[-2.20871802e-01 -6.46956260e-01 1.93006731e-01 3.31990425e-03  
2.70433451e-01 1.02682574e-01 -2.86860753e-01 -1.15341317e-01  
-2.68953753e-01 -1.47516683e-01 7.59649931e-02 -7.62042915e-02  
5.52020223e-02 -4.39551876e-02 8.22368328e-02 4.85365866e-02]  
[-1.26525727e-02 1.73347359e-01 1.73177249e-01 -2.45104804e-01  
-7.42479855e-02 -1.09432457e-01 -2.95608320e-02 -1.20519911e-02  
-3.48709591e-01 1.46437489e-01 -2.06443637e-02 3.66807510e-02  
1.61355586e-01 -1.50496644e-01 -9.84090344e-02 -1.35473350e-01]  
[ 8.91820191e-02 4.57022062e-01 2.03538988e-01 -8.12760812e-01  
1.30349997e-01 -7.75568841e-02 -1.36791961e-01 2.22920299e-01  
-2.92498634e-02 -4.70704110e-01 -2.62073425e-01 2.02558537e-01  
-3.38638563e-01 -2.36367827e-01 -2.29222619e-01 -1.39733744e-01]  
[-5.44061533e-02 7.01232458e-02 1.54992602e-03 8.29658174e-02  
-1.08213860e-01 8.83674926e-02 2.52018116e-01 -1.42736190e-01  
2.31046652e-01 9.12644298e-02 -8.79987125e-02 1.86006177e-01  
-2.06179437e-02 8.07588852e-02 -2.08961613e-01 1.79018685e-01]  
[-6.15369318e-01 -7.02368806e-01 -1.48082764e-01 6.68848123e-01  
-4.18594060e-01 -2.93344282e-01 -1.81762954e-01 -3.08027252e-01  
-6.25316497e-01 2.73588086e-01 4.46317258e-01 -3.05542637e-01  
3.86781350e-01 1.52760363e+00 4.64855085e-01 -2.86510019e-01]  
[-1.11969936e-01 7.44897457e-02 -6.43161662e-02 -1.54113207e-01  
2.83504306e-01 -4.36355011e-02 -4.78892132e-01 -2.50317688e-02  
7.73758010e-02 -1.15841064e-01 5.12131005e-02 -4.24578248e-01  
1.38185416e-01 3.81196040e-02 9.29970887e-02 2.52127651e-01]  
[ 1.74080306e-02 -1.23085475e-01 8.60320011e-03 1.05108822e-01  
2.18010916e-01 -1.45032968e-01 -2.26716804e-01 -5.91651026e-02  
1.99891402e-01 -7.90391706e-02 -3.98783255e-02 9.75014719e-02  
1.53594920e-01 -4.39249167e-01 1.12081455e-01 -1.75327890e-01]  
[-8.72566661e-02 -3.52451169e-01 -5.66405056e-02 2.26676449e-01  
-1.08299141e-01 -6.64213792e-02 -1.33987609e-01 -1.26294993e-01  
-1.26287018e-01 -8.56487401e-02 8.02011947e-02 -2.43451028e-01  
1.42894525e-01 2.52754888e-02 -8.82205805e-02 1.75472841e-02]  
[-2.99246272e-02 3.65658641e-01 -2.30408063e-01 8.23335705e-03  
6.79683386e-03 -3.37055084e-03 4.26738537e-01 6.61025642e-02  
1.53321000e-01 -2.59933001e-01 1.08612988e-01 2.29285711e-01  
-2.71315485e-01 -2.10836247e-01 3.35569844e-02 -3.94350322e-01]  
[-4.10435442e-01 1.97723312e-01 -4.77766857e-02 -4.99792526e-02  
9.12846483e-02 -2.98329493e-01 4.21507787e-02 -1.95260907e-01  
1.93410390e-01 -2.19846464e-01 -8.08792448e-02 2.77138734e-02  
-1.26435550e-02 -1.14898318e-01 6.96924606e-02 -3.18392349e-01]]

Bias : [[ 0.02142738]

[-0.6709216 ]  
[ 1.19039233]  
[ 0.00924475]  
[ 0.5543176 ]  
[-0.31357346]  
[ 1.15717444]  
[ 0.03734896]  
[ 0.39229987]  
[-0.3750619 ]  
[ 0.61666406]  
[-0.44208435]  
[-0.43782215]  
[-0.25125215]  
[ 0.63619387]  
[ 0.61024515]  
[ 1.04672881]  
[-0.42449401]  
[ 0.31292519]  
[-1.18144689]  
[-0.33731878]  
[-0.89290403]  
[ 0.60009027]  
[-1.51049592]  
[ 0.40255242]  
[-0.13573616]  
[ 1.03755514]  
[-1.31021127]  
[-0.1511035 ]  
[ 0.52286367]  
[-0.36039586]  
[-1.74643375]]

그림14. Optimal parameters of hidden layer 1

```

----- Layer 2 Info -----
Weight : [[ 1.77689342e-01  5.42655079e-02 -5.12207707e-02 -4.33824479e-02
 4.63198480e-01 -1.76993677e-01 -1.45868461e-01  6.46319048e-02
-3.54044892e-02  2.57680411e-01  2.96357808e-01  2.75679030e-01
-1.47390824e-01  2.15327686e-01  2.13383262e-01  2.52111517e-01
-2.18457173e-01  3.73617383e-01  3.46446377e-02 -1.26667338e-01
 2.32505730e-01 -1.71040339e-01 -4.89515612e-01  1.66442729e-01
 8.62225672e-02  1.71193950e-01  2.67004865e-01  3.91479705e-02
 6.69855524e-02 -2.11800362e-01  2.95327106e-01 -3.33927841e-01]
[-1.52422861e-01  3.18962977e-01 -2.39467266e-01 -1.71747152e-01
-5.69439926e-02 -1.68076633e-01 -1.14628592e-01 -3.57926977e-01
-9.90020416e-01  8.86323077e-02 -1.25506456e-01 -6.07257968e-02
 2.21037782e-01 -2.41064008e-01 -3.80585686e-01 -5.37704656e-01
 4.27510958e-01 -9.52730663e-02  1.36089916e-01  3.43825428e-01
 5.35039393e-01  2.19883027e-01  1.49402221e-01  2.03493887e-01
-1.89038644e-01  3.13409945e-01 -9.09411642e-02  5.28052871e-02
-9.14474199e-02 -6.76819144e-01 -3.02594302e-01  1.07829055e-01]
[-5.24700670e-01  1.99644750e-01 -3.51585326e-01  7.48397166e-04
 2.96095328e-01 -3.27764156e-02  3.76651665e-01  2.70325640e-03
-3.15544953e-01  1.84294708e-01  3.09095396e-01 -3.59247010e-01
 7.94765301e-01 -5.52565152e-01 -7.55349490e-02 -2.50936431e-01
-3.65787483e-02  6.76095879e-01 -6.80553967e-01 -3.59633938e-01
 5.87577202e-01  1.19108944e-01  1.26029378e-01 -2.78501843e-01
-4.89497231e-01 -1.37527564e-01  1.41663254e+00  2.63338461e-01
-2.18778434e-01  5.21642773e-02 -4.35448245e-01 -2.94321186e-01]
[-2.38897933e-01 -3.40615393e-01  5.91472402e-02 -4.38048238e-01
 3.42325703e-02  2.89980511e-01 -3.16593739e-01  1.91626244e-01
-1.67010528e-01 -2.06886229e-01 -1.73204394e-01  2.35660981e-01
-1.09816090e-01 -1.90292868e-01 -4.54698995e-01  3.36076126e-02
 2.51598734e-01 -1.49785186e-01  2.55718663e-01 -9.48187564e-02
-2.25068097e-01  3.91113396e-02 -6.41454241e-01  1.13953986e-01
 1.33114499e-01 -1.46580782e-01 -1.20701928e-01 -1.17046419e-01
 2.88887262e-01 -1.22533912e-02 -6.26813315e-02 -5.84619647e-03]
[-3.15589250e-02  5.49587038e-01  1.64093339e-01 -1.28388507e-01
-3.46053850e-01 -2.15648409e-02  2.53599405e-01  2.12195007e-02
 9.13635367e-02  1.30235137e-01  1.76460451e-01  2.62839057e-01
-9.55478183e-02 -4.07661005e-02  1.34829541e-01  1.98692478e-01
 1.45633860e-01 -8.82962311e-02  4.11003539e-01 -8.96914321e-02
-2.31743127e-01 -1.31559306e-01 -1.93540196e-01  6.57032439e-03
-2.20937752e-01  5.46225359e-01  4.26278562e-01 -4.26784703e-01
 8.16033170e-02  4.93680119e-02 -9.15688947e-02  1.58865242e-01]
[-8.35666960e-02  3.39575195e-01 -3.55466110e-01  1.01369702e-01
 1.86596659e-01  1.74902422e-01 -9.52347817e-02  1.68544268e-01
 2.29105662e-01  1.34894569e-01  2.73936968e-01 -1.49541069e-01
-4.36454743e-02 -1.94553016e-01  2.72706617e-02 -1.16866518e-01
-3.53194600e-01 -1.60308090e-01 -3.18566599e-02 -1.46186635e-01
 4.48357350e-01 -2.94737237e-02  8.89920168e-02 -2.79726192e-01
-1.47994051e-01  5.77393821e-02  3.88091750e-01  1.86723007e-01
-9.89338474e-02 -3.59497637e-01 -1.42342362e-01 -1.21807632e-02]
[-3.08970753e-02 -2.67022163e-01 -2.63571131e-01  3.27928301e-01
 6.12980525e-01 -1.46843903e-01 -1.20681646e-01 -2.39704613e-01
-1.61734373e-01  1.88360182e-01 -2.69877924e-01 -1.21095911e-02
-2.30986762e-01 -6.08955060e-02 -4.00014373e-01  1.93165606e-01
-3.45987842e-01 -3.91909885e-01 -4.32557338e-02  1.41090515e-01
-3.96899632e-02  4.46628549e-01 -6.17867999e-01 -5.91887153e-01
 1.79975141e-02 -5.43422086e-02 -6.34343547e-02  9.70908772e-02
 1.75426950e-01 -3.46755293e-02 -3.24290197e-01 -5.15448515e-02]
[-1.52043373e-01  5.36269873e-01  2.83730975e-02 -5.96911050e-03
-1.50696359e-01  1.41582293e-01 -4.39741669e-01 -3.41200756e-02
-3.51676484e-02  1.29030431e-01 -1.17933025e-01 -1.29983737e-01
-2.07791860e-01 -2.88598603e-01 -6.50232955e-02  1.45631033e-01
-9.04299120e-02  9.57686982e-02 -4.21108598e-01  2.78889465e-01
-3.33837360e-01  1.73724670e-01 -1.38047451e-01  1.67131360e-01
 1.60834431e-01 -3.31770656e-01 -2.95069634e-01 -2.45105478e-01
-1.10823076e-01 -3.64681041e-01 -2.62593701e-01  2.75753655e-01]

```

[ 8.39426944e-02	2.23579846e-01	5.54024171e-03	9.68389976e-02	
-2.73437434e-01	-1.65274349e-01	2.16054643e-01	7.05828437e-02	
2.82409177e-02	4.92283331e-01	1.32251822e-02	-2.56861462e-01	
-2.74483706e-01	5.00061736e-01	2.06768199e-01	-1.49438823e-01	
-1.85507004e-01	1.23401165e-03	1.81085101e-01	2.79791421e-02	
-8.56055021e-02	6.87668718e-02	2.11139313e-03	-1.50025094e-01	
-2.70412347e-01	3.55427341e-01	-2.53380929e-01	-2.84766134e-01	
-5.38209282e-03	1.30380408e-01	3.17166611e-01	-1.62407816e-01]	
[-2.42425657e-01	-4.17258663e-01	2.08207837e-01	1.78998000e-01	
2.55078294e-01	-2.62969965e-01	-2.59518954e-01	3.10941254e-01	
-1.29544824e-01	-2.08500228e-01	2.52115755e-01	-9.18167155e-02	
3.00442765e-01	5.11012081e-01	3.89432905e-01	3.07011642e-01	
-8.51309947e-02	-4.53383419e-01	1.67987979e-01	-2.25780022e-02	
-6.70673505e-03	1.00399325e-01	-8.16198898e-02	1.06083253e-01	
-2.69603513e-01	-4.05854904e-01	-4.80240245e-01	-8.56038664e-02	
1.48603324e-01	1.25874093e-01	-1.59369863e-01	-1.89335719e-01]	
[-2.73465561e-01	5.37594062e-01	-2.91381645e-01	-1.04969471e-01	
-1.04518363e-01	2.22422162e-01	4.72263179e-01	-6.95884693e-02	
3.20642133e-01	5.07063315e-02	-1.23602640e-01	-3.14330527e-01	
1.68466917e-01	3.14083976e-01	2.11503161e-01	-4.22426343e-01	
-2.84667624e-01	-3.87288821e-02	4.09976223e-01	-1.27563585e-01	
-3.75326631e-01	2.46236850e-01	-9.60059684e-02	1.87312016e-02	
1.93498156e-02	2.04888482e-01	4.12941509e-01	-7.65751143e-02	
2.61115150e-01	2.00493681e-01	-2.07856256e-01	-1.56849228e-01]	
[-1.63455976e-01	-3.61413340e-01	-3.35233963e-01	5.88113440e-02	
9.30420968e-02	3.79793329e-01	-2.42183608e-01	2.28618885e-02	
3.61395584e-01	-2.17568682e-01	-4.42830717e-01	1.90984917e-01	
9.40600220e-02	1.48947519e-01	2.77149270e-02	-9.14585600e-01	
-2.17169942e-01	3.77608275e-01	-3.57280156e-02	-1.59948577e-02	
-9.61953223e-02	-9.28381333e-02	1.43929165e-01	2.83526082e-01	
-1.64610478e-01	-1.52110609e-02	2.50044110e-01	2.97873078e-01	
5.97261576e-02	-2.93342114e-01	-2.74974963e-01	-2.58951051e-01]	
[-2.34202941e-02	1.91215001e-01	-2.44634626e-01	-4.02745529e-01	
-1.38399989e-01	-7.22533802e-02	-8.06189856e-02	-8.45539201e-02	
2.54870724e-01	-4.40842664e-03	-2.51553284e-01	5.03094652e-03	
2.33650507e-01	-2.05843634e-01	-8.88467787e-02	-6.23941905e-02	
-1.83265948e-01	5.19912381e-01	2.96978709e-01	4.57017795e-02	
3.40134610e-01	-1.91570400e-01	-3.47401963e-01	-1.76276553e-01	
-1.81121660e-01	-1.68407103e-02	-4.25382421e-01	1.34891375e-01	
-1.04606952e-01	-2.47241271e-01	4.08576935e-01	-1.34036692e-01]	
[-8.37593213e-02	8.38532021e-02	1.11149164e-01	-1.19307300e-01	
-3.97379774e-01	2.32432227e-01	1.13480145e-03	-1.29392034e-01	
1.06524880e-01	-5.55102829e-01	7.06489855e-02	-1.67360170e-01	
1.92396249e-01	-2.33446026e-01	-2.31382447e-01	8.30757919e-02	
1.64183727e-01	-1.93070940e-01	4.94347010e-01	8.34372663e-03	
1.55429611e-01	-2.58204984e-01	-2.45247717e-01	3.01479807e-02	
-4.93401779e-01	3.00499422e-01	3.55382491e-01	-1.37250380e-01	
2.56508793e-01	5.11939140e-02	-1.04474211e-01	2.75134853e-01]	
[ 2.36023311e-01	-1.84521427e-01	-5.19132166e-02	-4.85797878e-02	
-1.85044000e-02	1.77685112e-02	-7.05908406e-02	1.10731386e-01	
-1.26656191e-01	-7.14648834e-02	4.84927100e-02	1.62300000e-01	
3.56402557e-01	-2.41556093e-01	9.95940304e-02	-1.53355286e-01	
5.89802832e-02	2.25580737e-01	-4.91234342e-01	2.31076782e-01	
-1.45881200e-01	3.65220637e-01	2.26940451e-01	-1.58503723e-01	
-6.36566263e-01	-1.54623499e-01	7.83313373e-01	2.69708164e-02	
-5.31118211e-01	4.66479244e-02	1.07539137e-01	-1.49424888e-02]	
[-2.99448188e-01	8.73857239e-02	-2.24137601e-01	1.89629133e-01	
-6.08409267e-01	-1.38190486e-01	2.88013380e-01	9.58529719e-02	
9.11708104e-02	1.77637230e-01	7.31121962e-02	-3.48071231e-01	
-9.64643821e-02	1.13145803e-01	6.26556096e-01	-5.50306699e-02	
-1.07259037e-01	2.29368004e-01	2.27532269e-01	-2.51203731e-01	
1.53114831e-01	-1.91739468e-01	5.71116488e-02	5.69410697e-02	
-3.16082289e-02	-2.61129981e-01	4.08262575e-01	-2.32749097e-01	
1.27551506e-01	1.17094328e-01	-3.23431243e-01	-1.67477548e-02]]	

Bias : [[ 0.52036956]  
[ 0.12785167]  
[-0.68454087]  
[-0.73993212]  
[-0.51456569]  
[-0.06565159]  
[-0.13129917]  
[-0.18579291]  
[-0.61575671]  
[ 1.45262451]  
[ 0.2717796 ]  
[ 1.28619824]  
[-0.76106165]  
[ 0.6425712 ]  
[ 0.56806866]  
[-0.12922015]]

그림15. Optimal parameters of hidden layer 2

## 4. Conclusion

이번 과제는 저번과 다르게 실제로 사용하는 neural network를 활용한 model을 유사하게 구현해 보았다. Perceptron model을 여러 층을 쌓은 MLP와 역으로 학습하는 방식인 Backpropagation도 새로웠지만, 학습 알고리즘 뿐만 아니라 모델의 구조 자체도 학습의 loss와 관련이 있다는 것을 알았다. 실습을 해보면서 layer의 input/output size, activation function, layer의 수, epoch, learning rate 등 evaluation setup을 계속 바꿔가며 training을 했는데 그 결과들이 모두 달랐고 이런 과정을 통해 optimal한 parameters를 찾아가는 것임을 배웠다. Activation function은 sigmoid를 사용하는 것보다 tanh를 사용하는 것이 더 정확했고, 모델의 layer 수가 증가할수록 loss가 줄어들었다. 그리고 지금까지 인공지능과 관련된 코드를 작성해야 할 때 Tensorflow같은 딥러닝 프레임워크를 많이 사용했는데 이 프레임워크가 어떤 원리로 돌아가고, 코드는 어떤지를 생각 안하고 단순히 가져다 쓰기만 했었다. 이런 딥러닝 코드를 직접 구현해보니 보다 깊은 이해를 할 수 있었고 다음에 인공지능을 활용해야 할 때 큰 도움이 될 것 같다.