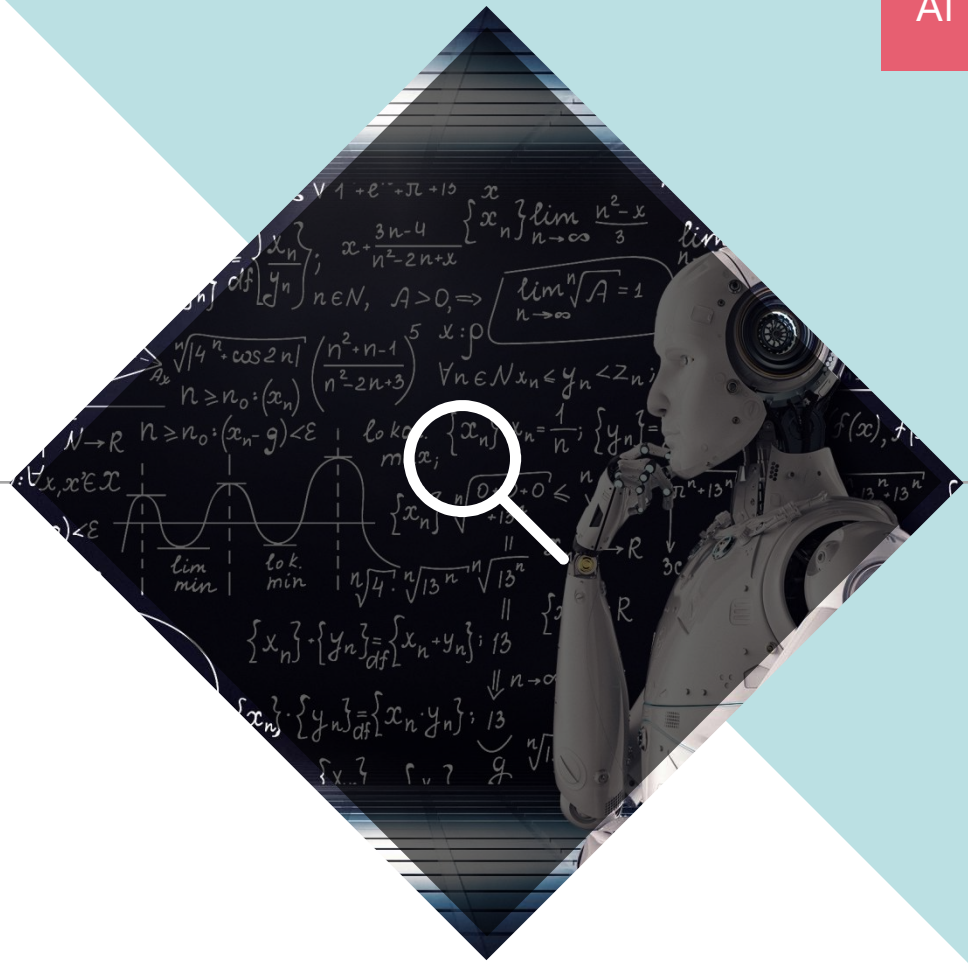
AI TRAINING SERIES

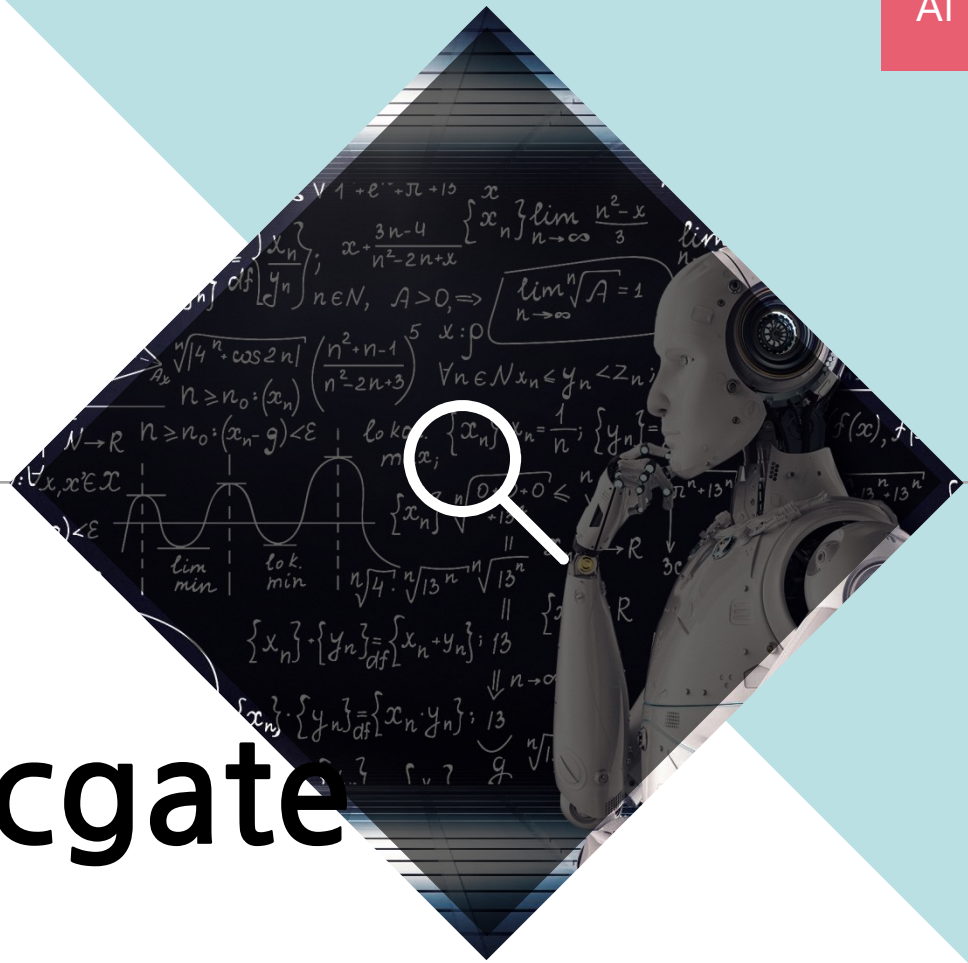
Perceptron & ANN



Title

- ❑ Background
- ❑ Perceptron
  - 순전파 (Forward)
  - 역전파 (Backpropagation)
    - Sigmoid + Binary cross-entropy
- ❑ ANN (step1)
  - Mnist 데이터셋 준비 (Prepare datasets)
  - Mnist 데이터셋 출력 (Print datasets)
- ❑ ANN (step2)
  - Mnist 데이터셋 학습을 위한 ANN layer 구상.
  - ANN 클래스 구조 구상
  - 활성함수 정의
  - 순전파 (Forward) 구현
  - 역전파 (Backpropagation)구현
    - Softmax + categorical cross-entropy (one-hot encoding)

AI TRAINING SERIES



Hands on for perceptron: logicgate

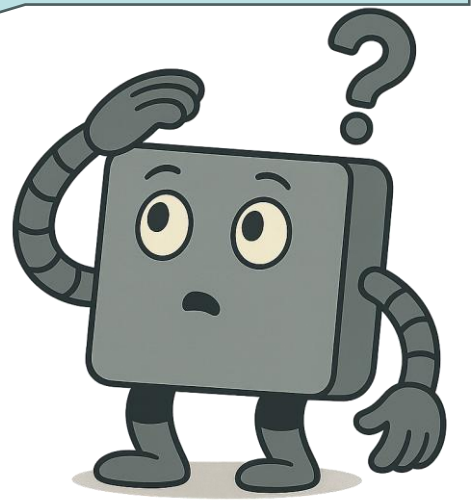
Perceptron 구현

- 목표:
  - Logic gate를 구현하고 테스트.
  - 클래스 구조를 설계하고 Backpropagation에 대한 이해.
  - 단일 Layer로 구성된 신경망을 구현하고 테스트.
    - AND, NAND, OR, NOR가 동작하는 신경망을 구현.
  - 두개의 Layer로 구성된 신경망을 구현하고 테스트.
    - XOR가 동작하는 신경망을 구현.

Logic gate 구현 (1)

- Logic 게이트 구현을 위한 클래스 구현
- class Perceptron():
  - 생성자 (__init__)
  - Linear model (wx+b)
  - 순전파 (forward)
  - 활성화 함수 (activation)
  - 손실 (loss)
  - 실행 & 역전파 (run & backpropagation)

Logic 게이트 구현을 위해서는
어떤 함수들이 필요할까?



Logic gate 구현 (2)

- Perceptron 클래스 구현.
  - 필요한 기능을 나열하고, 해당 기능을 수행하기 위한 함수를 작성.
  - 각 기능을 수행하기 위한 함수를 정의.

생성자 (__init__)

Linear model (wx+b)

순전파 (forward)

활성화 함수 (activation)

손실 (loss)

실행 & 역전파 (run & backpropagation)

```
class Perceptron():
    def __init__(self):

    def Model(self):

    def forward(self):

    def activation(self, X): # sigmoid

    def loss(self, Ytgt, Ypred): #categorical

    def run(self):
```

Logic gate 구현 (3)

- Perceptron 클래스 구현.
  - Perceptron구조를 구상

생성자 (__init__)

Linear model (wx+b)

순전파 (forward)

활성화 함수 (activation)

손실 (loss)

실행 & 역전파 (run & backpropagation)

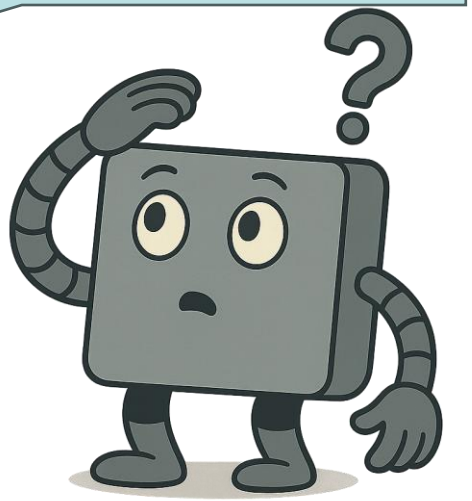
```
class Perceptron():
    def __init__(self):

    def Model(self):

    def forward(self):

    def activation(self, X): # sigmoid

    def loss(self, Ytgt, Ypred): #categorical

    def run(self):
```

Logic gate 구현 (4)

- Perceptron 클래스 구현.
  - __init__() 필요한 내용 작성.

```
class Perceptron():
    def __init__(self, X, Y, lr = 0.1, iteration = 100):
        self.X = X
        self.Y = Y
        self.lr = lr
        self.weight = rand(2)
        self.bias = rand(1)
        self.iteration = iteration
```

우선 익숙한 Model함수와 활성화 함수 activation을 작성하자.

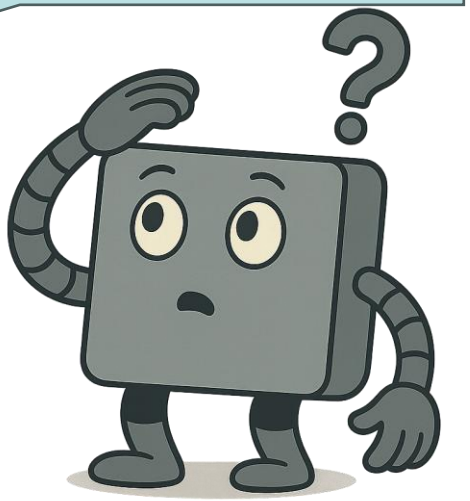


Logic gate 구현 (4)

- Perceptron 클래스 구현.
  - Model 함수 작성
  - Activation 작성

```
class Perceptron():
    def Model(self):
        return np.matmul(self.X , self.weight) + self.bias

    def forward(self):

    def activation(self, X): # sigmoid
        return sp.special.expit(X)

    def loss(self, Ytgt, Ypred): #categorical

    def run(self):
```

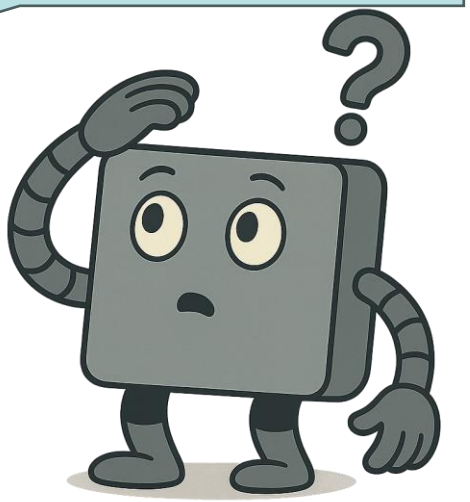
Loss는 0과 1을 구분하는 것이 목표니 “binary cross entropy”를 사용하자.



Logic gate 구현 (5)

- Perceptron 클래스 구현.
  - loss 작성

Forward 함수는 어떻게 구성하지?



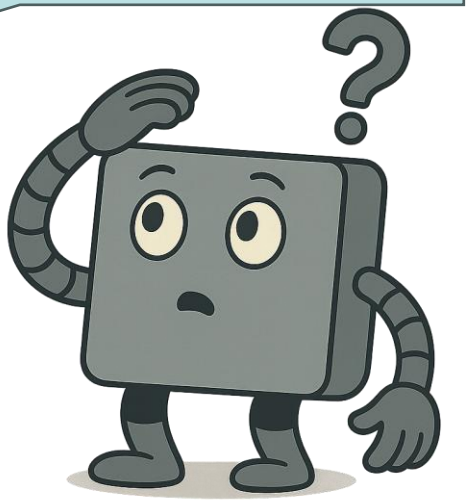
```
class Perceptron():
    def Model(self):
        return np.matmul(self.X , self.weight) + self.bias

    def forward(self):

    def activation(self, X): # sigmoid
        return sp.special.expit(X)

    def loss(self, Ytgt, Ypred): #categorical
        return -np.mean(Ytgt*np.log(Ypred) + (1-np.array(Ytgt))*np.log(1-Ypred))

    def run(self):
```

Logic gate 구현 (6)

- Perceptron 클래스 구현.
  - forward 작성

```
class Perceptron():
    def forward(self):
        output = self.Model()
        return  self.activation(output)

    def run(self):
```

run 함수는 어떻게 구성하지?

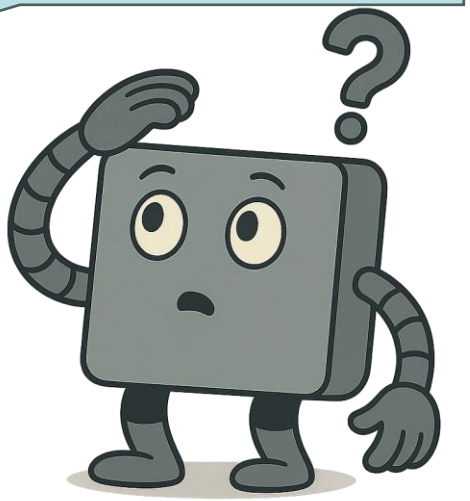


Logic gate 구현 (7)

- Perceptron 클래스 구현.
  - forward 작성

```
class Perceptron():
    def run(self):
        loss_score = []
        for i in range(self.iteration):
            self.cache = []
            # TODO forward
            Ypred = self.forward()
            loss = self.loss(self.Y, Ypred)
            loss_score.append(loss)
            print("loss : ", loss)
        return self.weight, self.bias
```

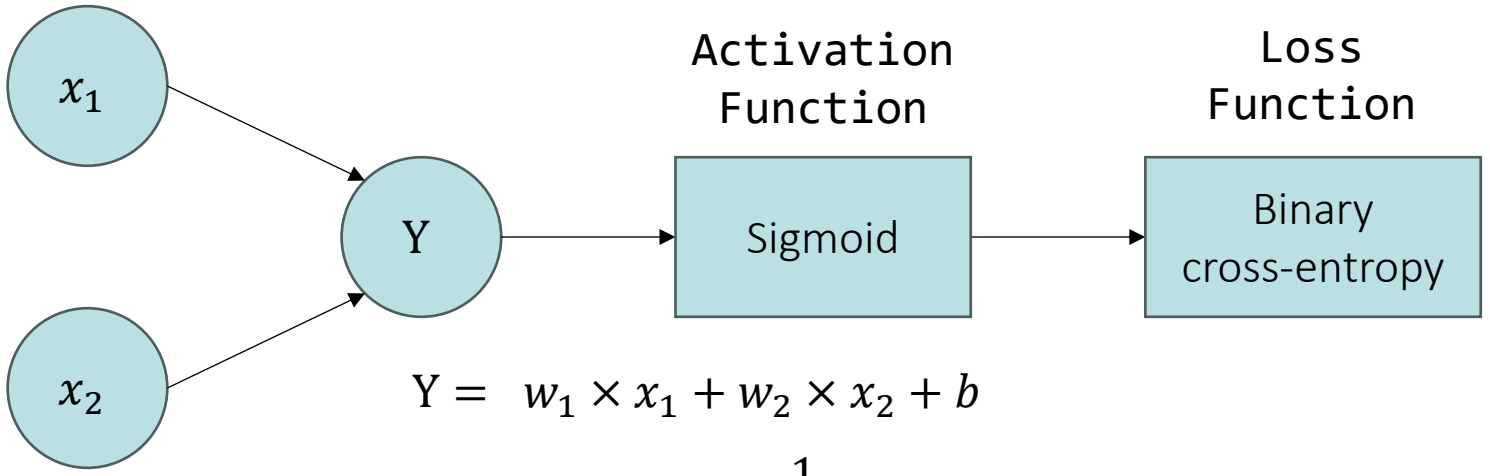
Backpropagation은 어떻게 해야하지?



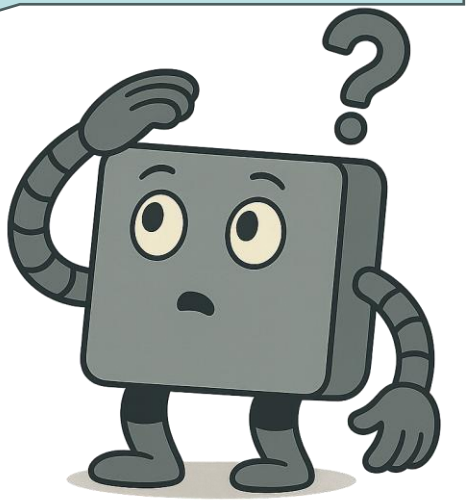
Logic gate 구현 (8)

❑ 클래스 구조를 설계하고 Backpropagation에 대한 이해.

- 현재 perceptron 구조



우선 현재 우리가 구상한 perceptron 구조를 생각해보자.



$$Y = w_1 \times x_1 + w_2 \times x_2 + b$$

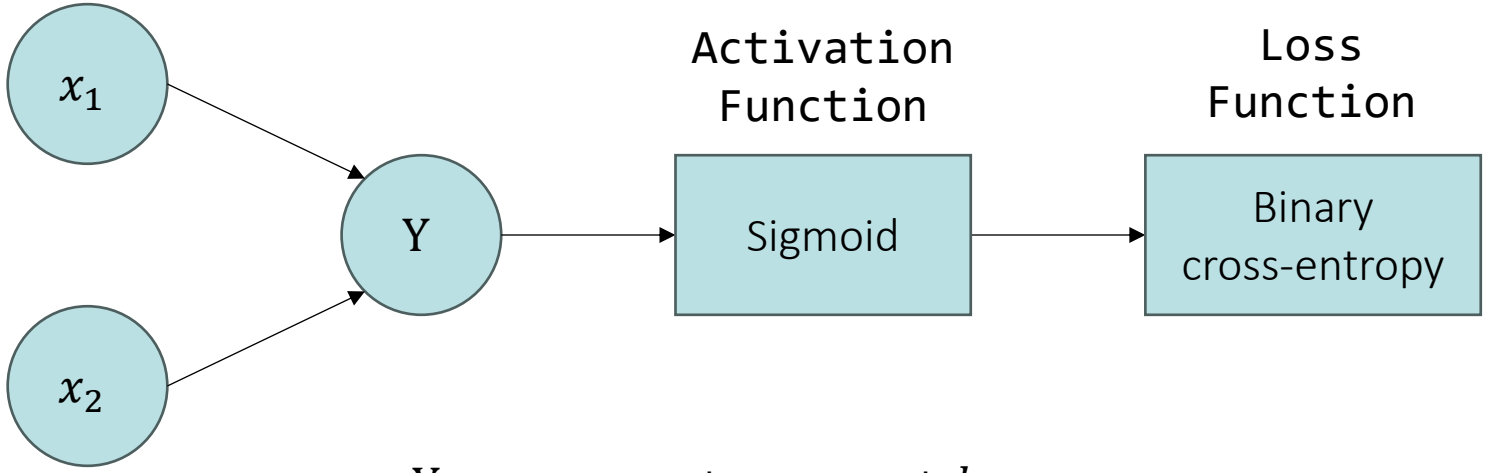
$$\text{sigmoid}(Y_k) = \frac{1}{1 + e^{-Y_k}} = \alpha_k$$

$$\text{Binary crossentropy}(\alpha_k) = Y_{tgt} \ln\frac{1}{\alpha_k} + (1 - Y_{tgt}) \ln\left(\frac{1}{1 - \alpha_k}\right)$$

Logic gate 구현 (9)

❑ 클래스 구조를 설계하고 Backpropagation에 대한 이해.

- w_1, w_2, b 값의 변화에 따른 Loss의 변화량은? (chain rule 적용)



$$\mathrm{Y} = w_1 \times x_1 + w_2 \times x_2 + b$$

$$\mathrm{sigmoid}(\mathrm{Y_k}) = \frac{1}{1 + e^{-Y_k}} = \alpha_k$$

$$\text{Binary crossentropy}(\alpha_k) = \mathrm{Y_{tgt}} \ln\frac{1}{\alpha_k} + (1 - \mathrm{Y_{tgt}}) \ln\left(\frac{1}{1 - \alpha_k}\right) = Loss(\alpha_k)$$

w_1의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_1}$$

w_2의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_2} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_2}$$

b의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial b}$$

Logic gate 구현 (10)

❑ 클래스 구조를 설계하고 Backpropagation에 대한 이해.

- w_1, w_2, b 값의 변화에 따른 Loss의 변화량은? (cont'd)

$$\mathrm{Y} = w_1 \times x_1 + w_2 \times x_2 + b$$

$$\mathrm{sigmoid}(\mathrm{Y}_k) = \frac{1}{1+e^{-Y_k}} = \alpha_k$$

$$\mathrm{Binary\ crossentropy}(\alpha_k) = \mathrm{Y_{tgt}}\ \ln\frac{1}{\alpha_k} + (1-\mathrm{Y_{tgt}})\ln\left(\frac{1}{1-\alpha_k}\right) = Loss(\alpha_k)$$

$$\frac{\partial Loss}{\partial \alpha_k} = -\frac{\mathrm{Y_{tgt}}}{\alpha_k} + \frac{1-\mathrm{Y_{tgt}}}{1-\alpha_k}$$

$$\frac{\partial \alpha_k}{\partial Y} = \frac{\partial}{\partial Y_k}\left(\frac{1}{1+e^{-Y_k}}\right)$$

w_1의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_1}$$

w_2의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_2} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_2}$$

b의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial b}$$

Logic gate 구현 (10)

❑ 클래스 구조를 설계하고 Backpropagation에 대한 이해.

- w_1, w_2, b 값의 변화에 따른 Loss의 변화량은? (cont'd)

$$Y = w_1 \times x_1 + w_2 \times x_2 + b$$

$$\text{sigmoid}(Y_k) = \frac{1}{1+e^{-Y_k}} = \alpha_k$$

$$\text{Binary crossentropy}(\alpha_k) = Y_{tgt} \ln\frac{1}{\alpha_k} + (1 - Y_{tgt}) \ln\left(\frac{1}{1-\alpha_k}\right) = Loss(\alpha_k)$$

w_1의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_1}$$

w_2의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_2} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_2}$$

b의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial b}$$

$$\frac{\partial Loss}{\partial \alpha_k} = -\frac{Y_{tgt}}{\alpha_k} + \frac{1 - Y_{tgt}}{1 - \alpha_k}$$

$$\frac{\partial \alpha_k}{\partial Y} = \frac{\partial}{\partial Y_k}\left(\frac{1}{1+e^{-Y_k}}\right) = \left(1+e^{-Y_k}\right)^{-2} e^{-Y_k} = \frac{e^{-Y_k}}{(1+e^{-Y_k})^2}$$

$$= \frac{e^{-Y_k}}{(1+e^{-Y_k})^2} = \frac{1}{1+e^{-Y_k}}\frac{e^{-Y_k}}{1+e^{-Y_k}} = \frac{1}{1+e^{-Y_k}}\left(1 - \frac{1}{1+e^{-Y_k}}\right)$$

$$= \alpha(1-\alpha)$$

$$\frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y} = \left(-\frac{Y_{tgt}}{\alpha_k} + \frac{1 - Y_{tgt}}{1 - \alpha_k}\right)(\alpha(1-\alpha))$$

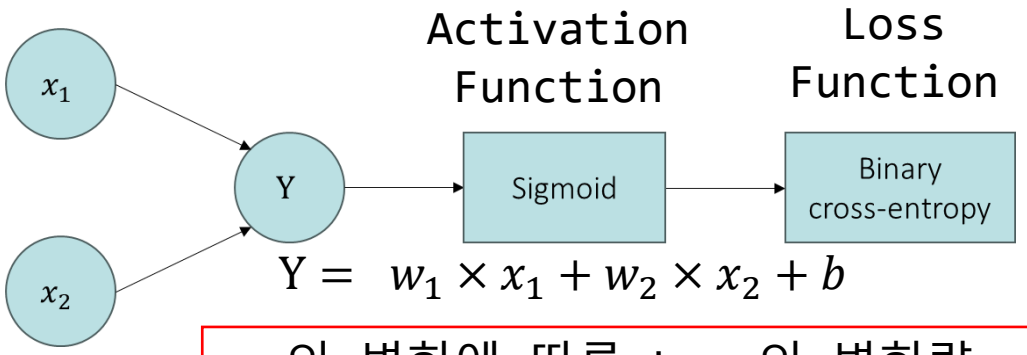
$$= -Y_{tgt}(1-\alpha) + \alpha(1 - Y_{tgt})$$

$$= \alpha - Y_{tgt}$$

Logic gate 구현 (11)

❑ 클래스 구조를 설계하고 Backpropagation에 대한 이해.

- w_1, w_2, b 값의 변화에 따른 Loss의 변화량은? (cont'd)



$$Y = w_1 \times x_1 + w_2 \times x_2 + b$$

w_1의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_1} = (\alpha - Y_{tgt})x_1$$

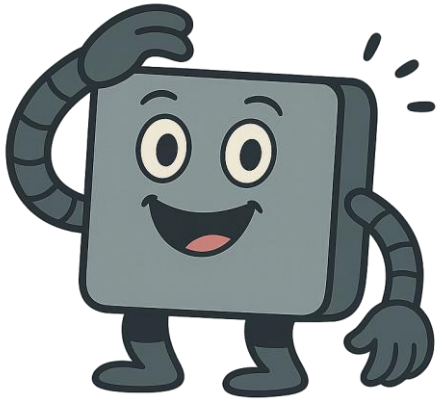
w_2의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_2} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_2} = (\alpha - Y_{tgt})x_2$$

b의 변화에 따른 Loss의 변화량

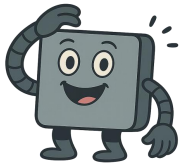
$$\frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial b} = (\alpha - Y_{tgt})$$

이제 코드에 적용해 보자!



Logic gate 구현 (12)

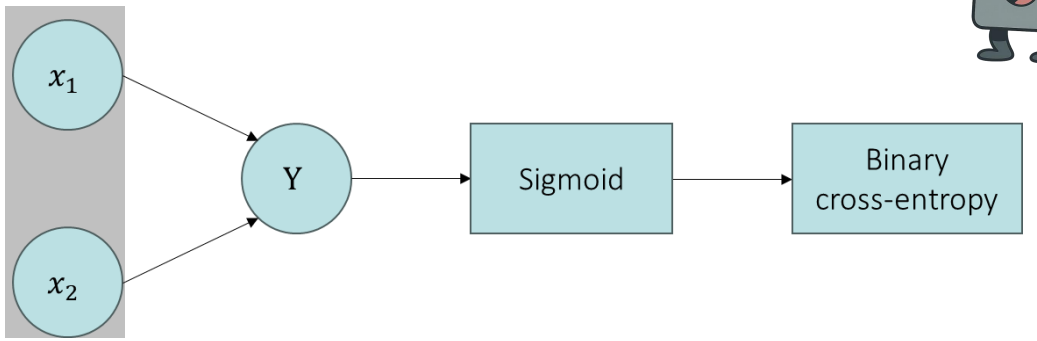
dw, db를 GD 알고리즘에 적용해보자



❑ 클래스 구조를 설계하고 Backpropagation에 대한 이해.

- w_1, w_2, b 값의 변화에 따른 Loss의 변화량은? (cont'd)

```
class Perceptron():
    def run(self):
        loss_score = []
        for i in range(self.iteration):
            self.cache = []
            # TODO forward
            Ypred = self.forward()
            loss = self.loss(self.Y, Ypred)
            loss_score.append(loss)
            print("loss : ", loss)
            dZ = Ypred - self.Y # dL/dY
            X = np.array(self.X)
            dw = np.matmul(X.T, dZ)
            db = np.sum(dZ)/len(dZ)

        return self.weight, self.bias
```



- w_1의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_1} = (\alpha - Y_{tgt})x_1$$

- w_2의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_2} = (\alpha - Y_{tgt})x_2$$

- b의 변화에 따른 Loss의 변화량

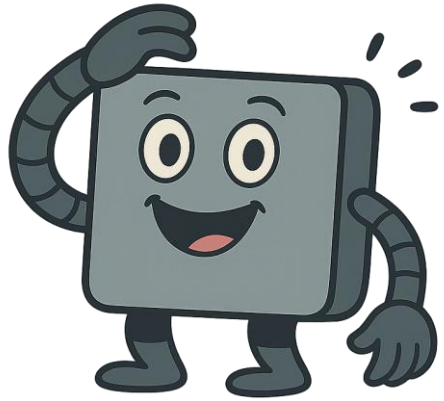
$$\frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial b} = (\alpha - Y_{tgt})$$

Logic gate 구현 (13)

- Perceptron 클래스 구현.
  - Perceptron 클래스 구현: Learning rate를 적용.

```
class Perceptron():
    def run(self):
        loss_score = []
        for i in range(self.iteration):
            self.cache = []
            # TODO forward
            Ypred = self.forward()
            loss = self.loss(self.Y, Ypred)
            loss_score.append(loss)
            print("loss : ", loss)
            dZ = Ypred - self.Y # dL/dY
            X = np.array(self.X)
            dw = np.matmul(X.T, dZ)
            db = np.sum(dZ)/len(dZ)
            self.weight = self.weight  - self.lr*dw
            self.bias = self.bias      - self.lr*db
        return self.weight, self.bias
```

테스트를 해보자!



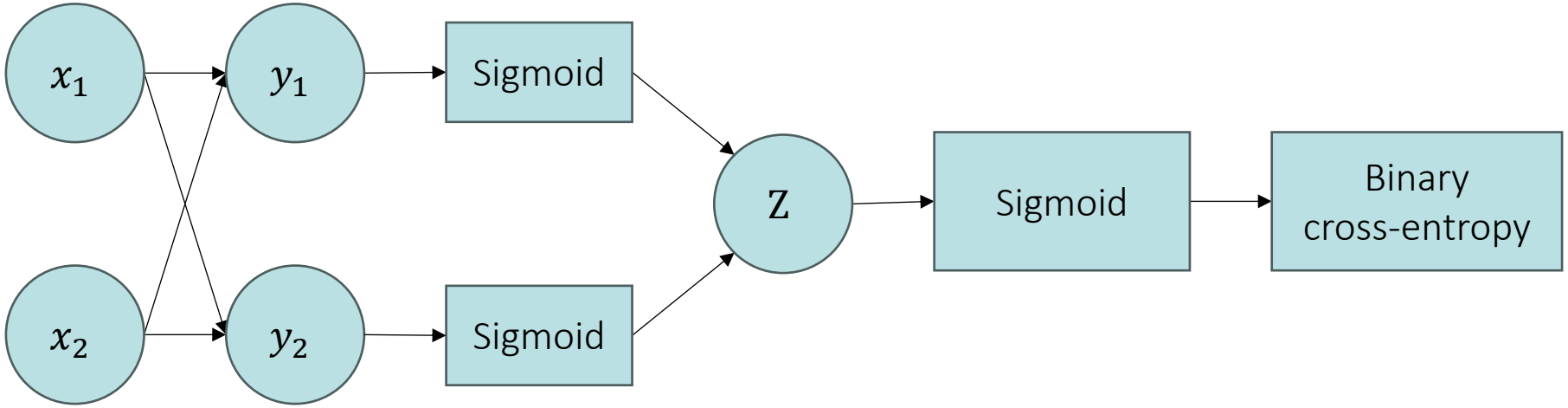
Logic gate 구현 (14)

- 단일 Layer로 구성된 신경망을 구현하고 테스트.
  - AND, NAND, OR, NOR 테스트

```
xdata = [[0, 0],[0, 1],[1, 0],[1, 1]]
ydata = [0, 1, 1, 1]

ANDgate = Perceptron(xdata, ydata)
weight, bias = ANDgate.run()
output = sp.special.expit(np.matmul(xdata, weight) + bias)
Ypred = (output>=0.5).astype(int)
print("-----------------------------------")
print("Target | Predict")
print("-----------------------------------")
for i in range(4):
    print(ydata[i], "      |",  Ypred[i])
```

Logic gate 구현 (15)

- 두개의 Layer로 구성된 신경망을 구현하고 테스트.
  - XOR를 신경만 구조로 나타내면?



$$y_1 = w_{11} \times x_1 + w_{12} \times x_2 + b_1$$

$$y_2 = w_{21} \times x_1 + w_{22} \times x_2 + b_2$$

$$Z = v_1 \times \alpha(y_1) + v_2 \times \alpha(y_2) + u$$

$$\text{sigmoid}(A_k) = \frac{1}{1 + e^{-A_k}} = \alpha(A_k)$$

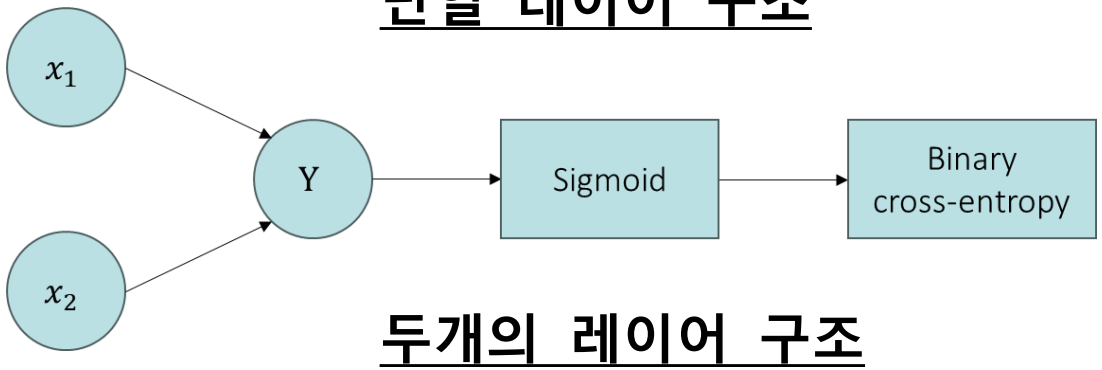
$$\text{Binary crossentropy}(\alpha_k) = Y_{tgt} \ln\frac{1}{\alpha_k} + (1 - Y_{tgt}) \ln\left(\frac{1}{1 - \alpha_k}\right) = Loss(\alpha_k)$$

Logic gate 구현 (16)

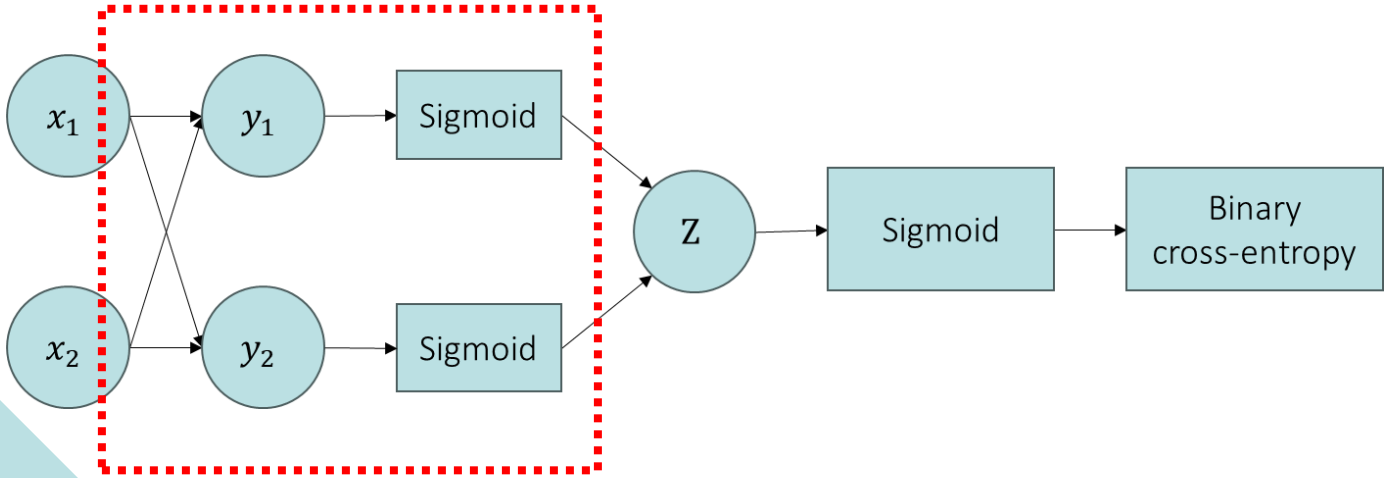
❑ 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).

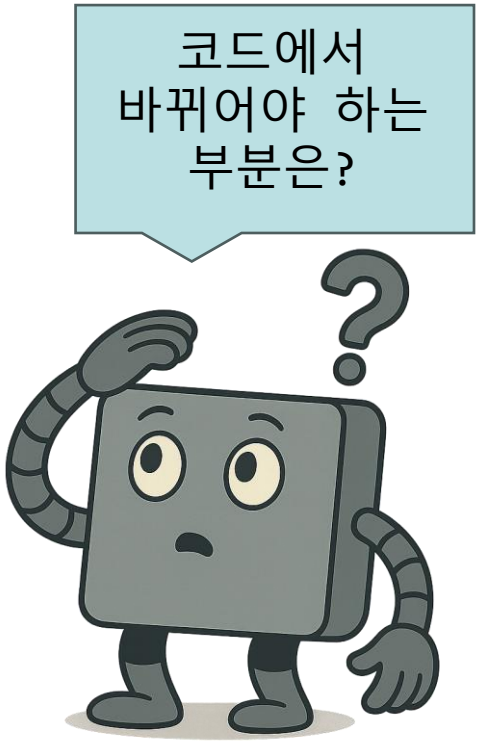
- 단일 레이어 구조 두개의 레이어 구조의 비교

단일 레이어 구조



두개의 레이어 구조





Logic gate 구현 (17)

- 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).
  - 단일 레이어 구조 두개의 레이어 구조의 비교
  - 파라미터 초기화의 배열이 달라짐.

단일 레이어 구조

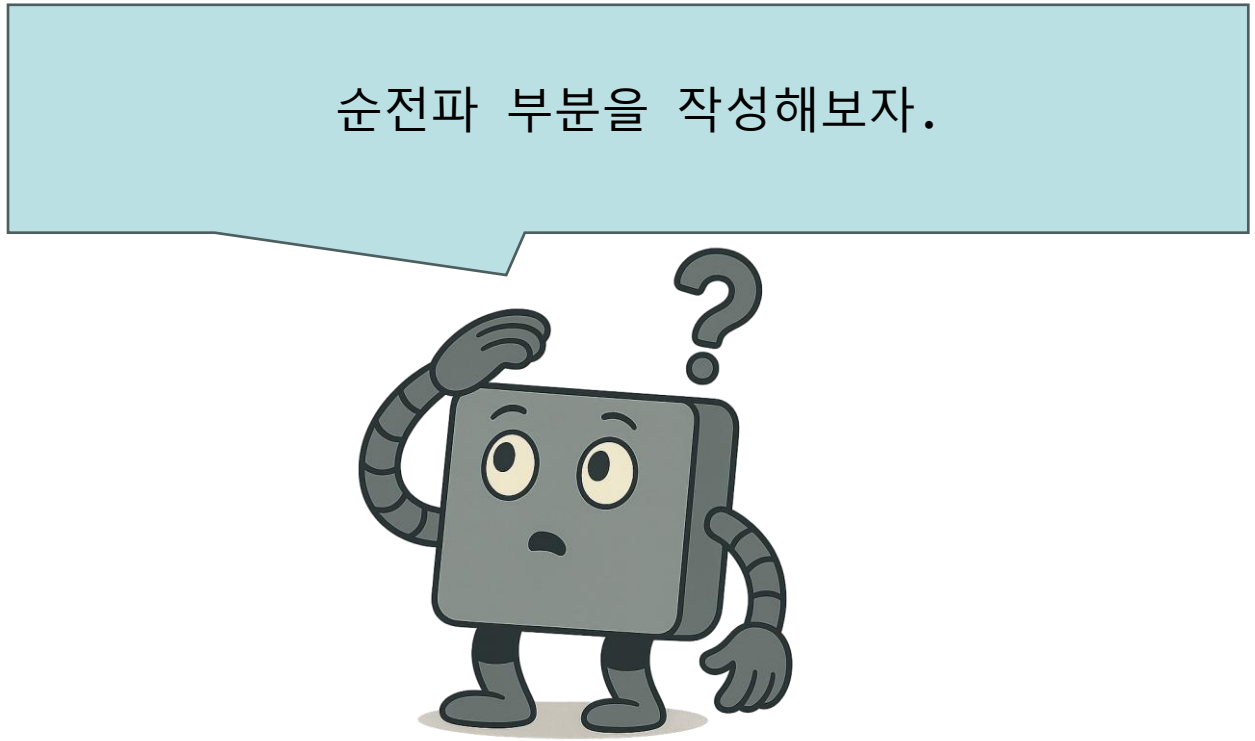
```
def __init__(self, X, Y, …):
    self.X = X
    self.Y = Y
    self.weight = rand(2)
    self.bias = rand(1)
```

두개의 레이어 구조

```
# TODO 1-1. weight (2)
# TODO 2-1. weight (2,2)
self.weight1 = np.random.uniform(-1, 1, (2, 2))
# TODO 2-2. weight(2)
self.weight2 = np.random.uniform(-1, 1, (2))
# TODO 2-3. bias(2)
self.bias1 = np.zeros((2))
# TODO 2.4. bias(1)
self.bias2 = np.zeros((1))
```

Logic gate 구현 (18)

- 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).
  - 단일 레이어 구조 두개의 레이어 구조의 비교
  - Model, Loss, activation의 정의는 동일하다.



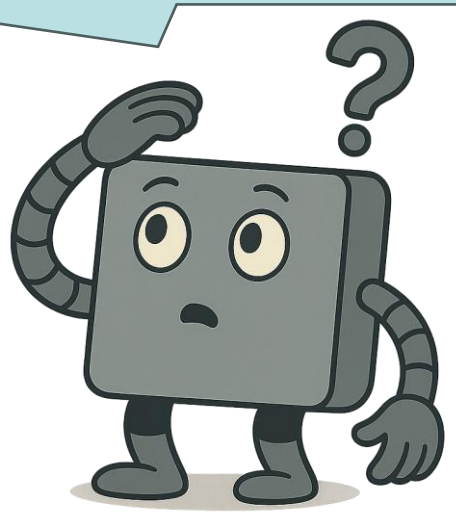
Logic gate 구현 (19)

❑ 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).

- 순전파는 두개의 레이어 (Layer1, Layer2)로 구성한다.
- 두번째 레이어를 통과한 결과의 Loss를 구한다.

```
def run(self):
    Global_loss_score = []
    for i in range(self.iteration):
        # TODO forward layer 1
        # TODO forward layer 2
        # TODO Loss
```

Layer1부터 작성을 해보자.

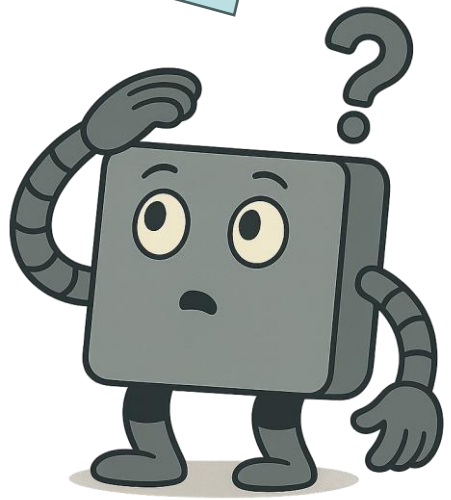


Logic gate 구현 (20)

❑ 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).

```
def run(self):
    Global_loss_score = []
    for i in range(self.iteration):
        # TODO forward layer 1
        input = np.array(self.X)
        output = self.Model(input, self.weight1, self.bias1)
        output = self.activation(output)
        input = output.copy()
        # TODO forward layer 2
        # TODO Loss
        loss = self.loss(self.Y, Ypred)
```

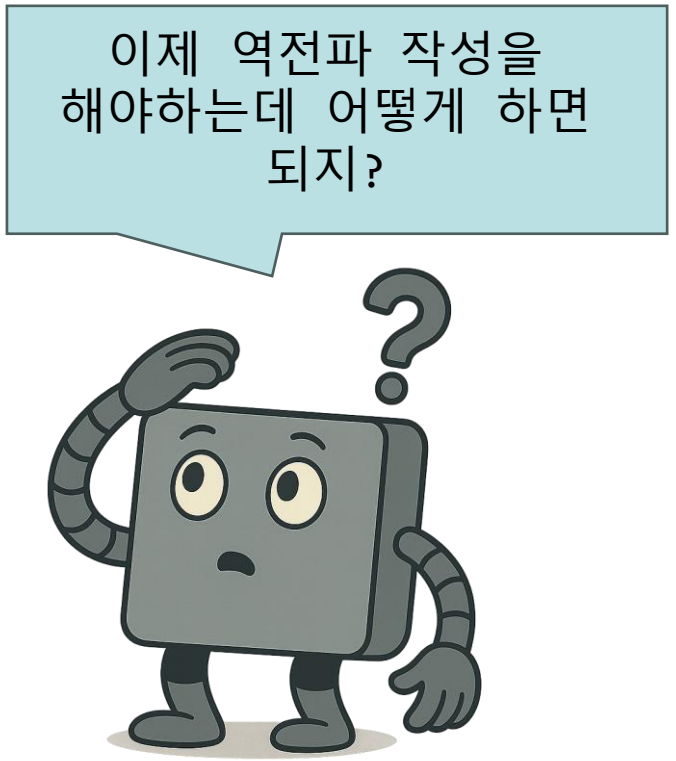
Layer2와 Loss를 작성하자.



Logic gate 구현 (21)

❑ 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).

```
def run(self):
    Global_loss_score = []
    for i in range(self.iteration):
        # TODO forward layer 1
        Input = np.array(self.X)
        Output = self.Model(input, self.weight1, self.bias1)
        Output = self.activation(output)
        Input = output.copy()
        # TODO forward layer 2
        Output = self.Model(input, self.weight2, self.bias2)
        Input = output.copy()
        Ypred = self.activation(input)
        # TODO Loss
        loss = self.loss(self.Y, Ypred)
```



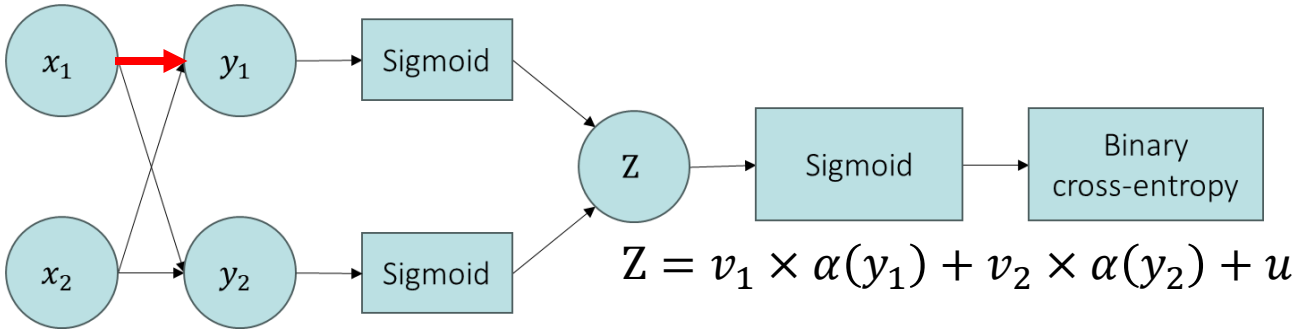
Logic gate 구현 (22)

$$\text{sigmoid}(A_k) = \frac{1}{1+e^{-A_k}} = \alpha(A_k)$$

$$\text{Binary crossentropy}(\alpha_k) = Y_{tgt} \ln\frac{1}{\alpha_k} + (1 - Y_{tgt}) \ln\left(\frac{1}{1-\alpha_k}\right) = Loss(\alpha_k)$$

❑ 역전파 구현

- 단일 레이어의 역전파에서 구한 Loss와 파라미터간 편미분 결과에서부터 시작.



$$Z = v_1 \times \alpha(y_1) + v_2 \times \alpha(y_2) + u$$

- v_1의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial v_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial v_1} = (\alpha(Z) - Z_{tgt})\alpha(y_1)$$

- v_2의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial v_2} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial v_2} = (\alpha(Z) - Z_{tgt})\alpha(y_2)$$

- u의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial u} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial u} = (\alpha(Z) - Z_{tgt})$$

w_{11}의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_{11}} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial \alpha(y1)}\frac{\partial \alpha(y_1)}{\partial y_1}\frac{\partial y_1}{\partial w_{11}}$$

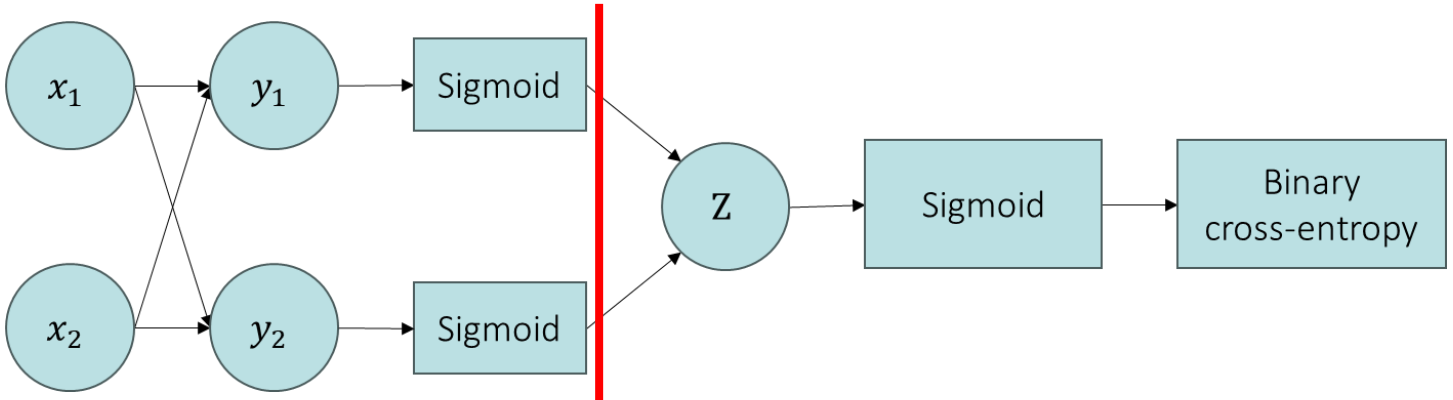
where

- $\frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z} = (\alpha(Z) - Z_{tgt})$
- $\frac{\partial Z}{\partial \alpha(y1)} = v_1$
- $\frac{\partial \alpha(y_1)}{\partial y_1} = \alpha(y_1)(1 - \alpha(y_1))$
- $\frac{\partial y_1}{\partial w_{11}} = x_1$

Logic gate 구현 (23)

❑ 역전파 구현

- 역전파에서 우리가 순전파 과정에서 저장해야 하는 정보는?
- **첫번째 레이어이후 sigmoid를 통과한 데이터**



$$\frac{\partial Loss}{\partial v_1} = \big(\alpha(Z) - Z_{tgt}\big)\alpha(y_1)$$

$$\frac{\partial Loss}{\partial w_{11}} = \big(\alpha(Z) - Z_{tgt}\big) \cdot v_1 \times \Big(\alpha(y_1)\big(1 - \alpha(y_1)\big)\Big) \cdot x_1$$

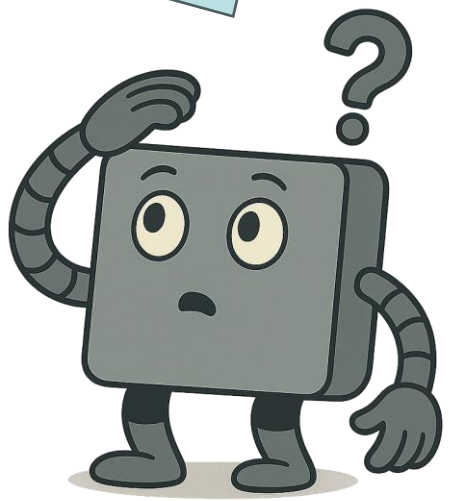
Logic gate 구현 (24)

❑ 역전파 구현

- 코드에서 해당 위치 저장은?

```
def run(self):
    Global_loss_score = []
    for i in range(self.iteration):
        # TODO forward layer 1
        input = np.array(self.X)
        output = self.Model(input, self.weight1, self.bias1)
        output = self.activation(output)
        cache = output
        input = output.copy()
        # TODO forward layer 2
        output = self.Model(input, self.weight2, self.bias2)
        input = output.copy()
        Ypred = self.activation(input)
        # TODO Loss
        loss = self.loss(self.Y, Ypred)
```

위에서 구한 식을 토대로 역전파를 구현을 하면 되겠지?



Logic gate 구현 (25)

❑ 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).

- **목표:** v, w (파라미터)의 변화에 대한 Loss를 구하고, GD 알고리즘에 적용.

▪ v_1의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial v_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial v_1} = \left(\alpha(Z) - Z_{tgt}\right)\alpha(y_1)$$

▪ w_{11}의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_{11}} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial \alpha(y1)}\frac{\partial \alpha(y_1)}{\partial y_1}\frac{\partial y_1}{\partial w_{11}}$$

```
def run(self):
    Global_loss_score = []
    for i in range(self.iteration):
        """
        이어서 작성
        """
        # TODO Backpropagation sigmoid + binary crossentropy
        dZ = Ypred - self.Y # dL/dY
        # TODO Backpropagation layer 2
        → 작성
```

Logic gate 구현 (26)

❑ 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).

- **목표:** v, w (파라미터)의 변화에 대한 Loss를 구하고, GD 알고리즘에 적용.

▪ v_1의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial v_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial v_1} = (\alpha(Z) - Z_{tgt})\alpha(y_1)$$

```
def run(self):
    Global_loss_score = []
    for i in range(self.iteration):
        """
        이어서 작성
        """
        # TODO Backpropagation sigmoid + binary crossentropy
        dZ = Ypred - self.Y # dL/dY
        # TODO Backpropagation layer 2
        dw2 = np.matmul(cache.T, dZ)
        db2 =np.mean(dZ, axis=0)
```

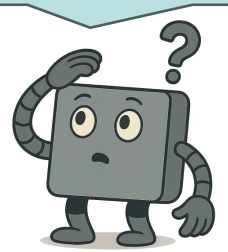
▪ w_{11}의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_{11}} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial \alpha(y1)}\frac{\partial \alpha(y_1)}{\partial y_1}\frac{\partial y_1}{\partial w_{11}}$$

where

- $\frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z} = (\alpha(Z) - Z_{tgt})$
- $\frac{\partial Z}{\partial \alpha(y1)} = v_1$
- $\frac{\partial \alpha(y_1)}{\partial y_1} = \alpha(y_1)(1 - \alpha(y_1))$

이제 w를 구해야하는데…



Logic gate 구현 (25)

❑ 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).

- **목표:** v, w (파라미터)의 변화에 대한 Loss를 구하고, GD 알고리즘에 적용.

▪ v_1의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial v_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial v_1} = (\alpha(Z) - Z_{tgt})\alpha(y_1)$$

```
def run(self):
    Global_loss_score = []
    for i in range(self.iteration):
        """
        이어서 작성
        """
        # TODO dZ update
        # dL /dw = dL/dY * dY/dX *  dX / dw
        dZ = dZ[:, np.newaxis] * self.weight2
        back_act = cache*(1-cache)
        dZ = dZ * back_act
        # TODO Backpropagation 1
```

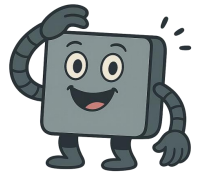
▪ w_{11}의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial \alpha(y1)}\frac{\partial \alpha(y_1)}{\partial y_1}\frac{\partial y_1}{\partial w}$$

where

- $\frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z} = (\alpha(Z) - Z_{tgt})$
- $\frac{\partial Z}{\partial \alpha(y1)} = v_1$
- $\frac{\partial \alpha(y_1)}{\partial y_1} = \alpha(y_1)(1 - \alpha(y_1))$

이제 w를 구할수 있겠다.



Logic gate 구현 (25)

❑ 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).

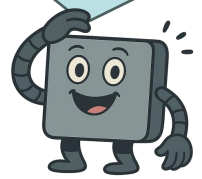
- **목표:** v, w (파라미터)의 변화에 대한 Loss를 구하고, GD 알고리즘에 적용.

```
def run(self):
    Global_loss_score = []
    for i in range(self.iteration):
        """
        이어서 작성
        """
        # TODO dZ update
        # dL /dw = dL/dY * dY/dX *  dX / dw
        dZ = dZ[:, np.newaxis] * self.weight2
        back_act = cache*(1-cache)
        dZ = dZ * back_act
        # TODO Backpropagation 1
        dw1 = np.matmul(self.X.T, dZ)
        db1 =np.mean(dZ, axis=0)
```

- w_{11}의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Z}\frac{\partial Z}{\partial \alpha(y1)}\frac{\partial \alpha(y_1)}{\partial y_1}\frac{\partial y_1}{\partial w}$$

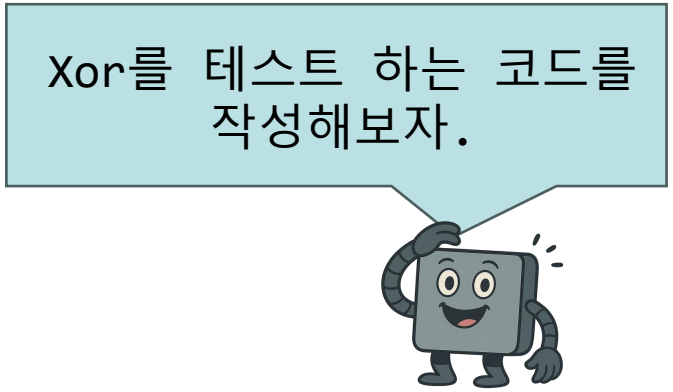
Learning rate 를 구해보자.



Logic gate 구현 (26)

- 두개의 Layer로 구성된 신경망을 구현하고 테스트 (cont'd).
  - **목표:** v, w (파라미터)의 변화에 대한 Loss를 구하고, GD 알고리즘에 적용.

```
def run(self):
    Global_loss_score = []
    for i in range(self.iteration):
        """
        이어서 작성
        """
        # print("loss : ", loss)
        self.weight1 = self.weight1  - self.lr*dw1
        self.bias1 = self.bias1      - self.lr*db1
        self.weight2 = self.weight2  - self.lr*dw2
        self.bias2 = self.bias2      - self.lr*db2
```



Logic gate 구현 (27)

❑ 결과 테스트

```
epoch = 20
epochs = np.linspace(1, 20, 20)
ANDgate = Perceptron(xdata, ydata, epoch=epoch)
history = ANDgate.run()
loss    = [h[0] for h in history]
weight1 = [h[1] for h in history]
weight2 = [h[3] for h in history]
bias1   = [h[2] for h in history]
bias2   = [h[4] for h in history]

index = np.argmin(loss)
print(index, loss[index])
```

Logic gate 구현 (28)

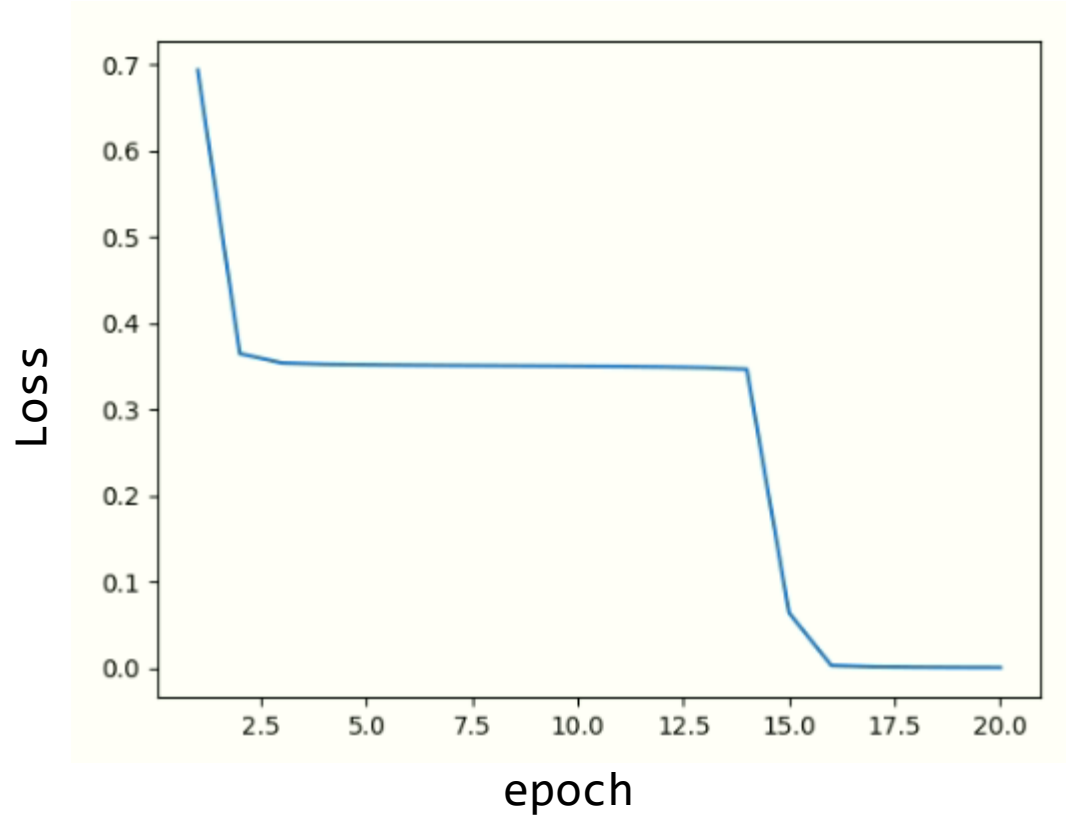
❑ 결과 테스트 (cont'd)

```
o1 = ANDgate.Linear(xdata, weight1[index], bias1[index])
o1 = ANDgate.activation(o1)
o2 = ANDgate.Linear(o1, weight2[index], bias2[index])
output = ANDgate.activation(o2)

import matplotlib.pyplot as plt
plt.figure()
plt.plot(epochs, loss)
plt.show()

Ypred = (output>=0.5).astype(int)
print("-----------------------------------")
print("Target | Predict")
print("-----------------------------------")
for i in range(4):
    print(ydata[i], "     |",  Ypred[i])
```

Logic gate 구현 (29)

❑ 결과 테스트 (cont'd)

- iteration = 5000, epoch = 20



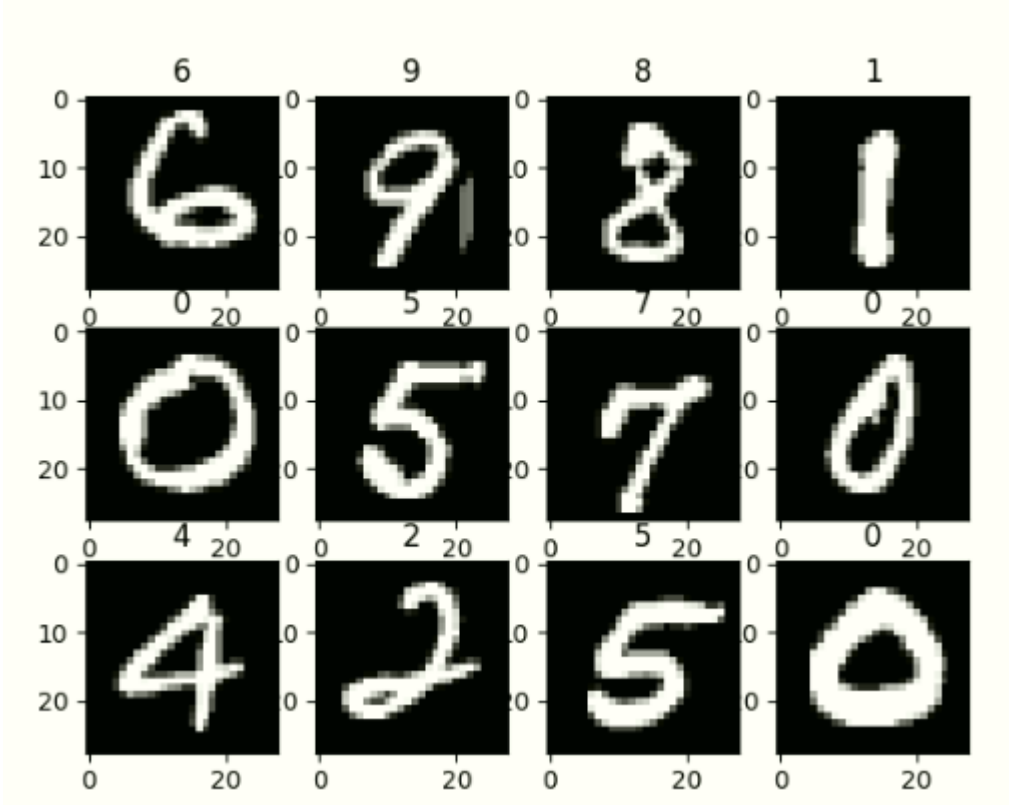
Target	Predict
0	0
1	1
1	1
0	0

AI TRAINING SERIES

Hands on for perceptron: ANN

ANN: Mnist 학습 데이터 준비 (1)

- Mnist 데이터셋을 준비하고, 출력



ANN: Mnist 학습 데이터 준비 (2)

- Mnist 데이터셋을 준비하고, 출력
  - 필요한 파이썬 패키지 추가

```
import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
import pickle
import matplotlib.pyplot as plt
import scipy as sp
from tqdm import tqdm
from types import SimpleNamespace
```

ANN: Mnist 학습 데이터 준비 (3)

- Mnist 데이터셋을 준비하고, 출력
  - 필요한 데이터 셋 다운로드

```
mnist = fetch_openml('mnist_784')
image = mnist.data.astype(np.float32)
label = mnist.target.astype(np.int64)
```

ANN: Mnist 학습 데이터 준비 (4)

- Mnist 데이터셋을 준비하고, 출력
  - 학습에 사용할 데이터 셋과 테스트에 사용할 데이터 세트를 분배

```
#TODO train 데이터셋과 test 데이터셋 분배
image_train, image_test, label_train, label_test = train_test_split(
    image, label,
    test_size=10000,
    stratify=label,
    random_state=42
)
```

ANN: Mnist 학습 데이터 준비 (5)

- Mnist 데이터셋을 준비하고, 출력
  - Label(정답)을 one-hot encoding형태가 되도록 변환.

```
#TODO train 데이터셋과 test 데이터셋 분배
def one_hot(labels, n_classes=10):
    m = labels.shape[0]
    oh = np.zeros((m, n_classes), dtype=np.float32)
    oh[np.arange(m), labels] = 1.0
    return oh

label_train = one_hot(label_train, n_classes=10)
label_test  = one_hot(label_test , n_classes=10)
```

ANN: Mnist 학습 데이터 준비 (6)

- Mnist 데이터셋을 준비하고, 출력
  - 다운 받은 이미지의 형태가 (1, 784) 형태를 가지고 있어 이미지의 형태를 (Batch, 28, 28)이 되도록 reshape

```
image_train = np.array(image_train).reshape(-1, 28, 28)
image_test = np.array(image_test).reshape(-1, 28, 28)
```

  - 원하는 형태로 변환이 되었는지 확인.

ANN: Mnist 학습 데이터 준비 (7)

- Mnist 데이터셋을 준비하고, 출력
  - 이미지는 0 - 255의 필셀값을 가지고 있기 때문에, 0 - 1로 정규화가 필요함.

```
image_train, image_test = image_train/255, image_test/255
```

  - 준비된 이미지를 subplot을 이용해서 4×4 공간에 출력.

ANN: 동작 방법 계획.

❑ Mnist 데이터 셋을 분류를 위한 신경망 설계

- 레이어 입력
  - 선형 모델은 신경망구조의 차원 (노드)의 수를 입력으로 받는다.
  - 활성화 함수는 'Relu', 'sigmoid', 'softmax'를 입력 받는다.
  - 배치크기를 입력 받고, 배치 크기 만큼 iteration횟수를 계산해서 학습한다.

  ※ epoch은 class 외부에서 설정한다.
- 순전파
  - 선형 모델의 레이어별 파라미터 (w, b)들에 대한 접근을 위해 딕셔너리 형으로 구성
- 역전파
  - Loss를 categorical cross entropoy를 사용.
  - 각 Layer의 가중치 및 바이어스의 변화에 따른 Loss 의 변화 계산 (chain rule).

ANN: 동작 방법 계획.

- 레이어 설계는 하기 코드를 작성하면, 이미지에서 처럼 레이어가 쌓이도록 설계

```
test_ANN.Input(size=image_train.shape)
test_ANN.Flatten()
test_ANN.add_ann_layer(128)
test_ANN.relu()
test_ANN.add_ann_layer(10)
test_ANN.softmax()
```

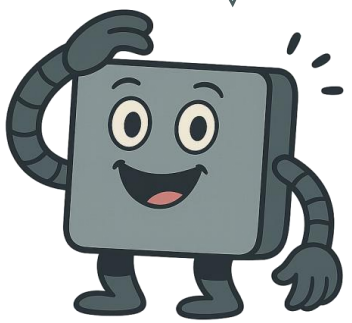
```
image_train :  (60000, 28, 28) | image_test :  (10000, 28, 28) | label_train :  (60000, 10) | label_test :  (10000, 10)
Layer name | parameters
==========================================================================================
flatten
ANN layers |     weight : (784, 128) |   bias : (1, 128)
relu
ANN layers |     weight : (128, 10) |    bias : (1, 10)
softmax
```

ANN: 클래스 작성 (1)

- Layer정보를 저장하기 위한 함수 정의.

```
class ANN:
    def __init__(self):
    def show(self):

## pubic
    def Flatten(self):
        pass
    def add_ann_layer(self, dims):
        pass
    def relu(self):
        pass
    def softmax(self):
        pass
    def Input(self, size):
        pass
```

실재 함수의 기능을 하는 것이 아닌 레이어의 정보만 넣어줘야 합니다.

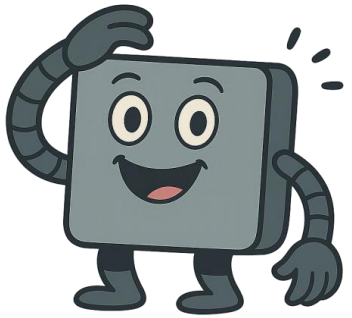


ANN: 클래스 작성 (2)

- Layer정보를 저장하기 위한 함수 정의 (cont'd).
  - Flatten, activation, add_layer마다 다른 옵션 및 설정을 가지고 있기 때문에 유의하면서 작성.

```
class ANN:
    def __init__(self):
    def show(self):
    # TODO 입력한 layer의 차원을 저장하기 위한 함수 정의.
    def __add_layer(self, name, dim):
        layer = SimpleNamespace()
        layer.dims=dim
        layer.name=name
        layer.property =
        layer.param = dict()
        layer.grad = dict()
```

__add_layer 함수로 좀더 구체적으로 저장할 정보를 설정해줍시다.

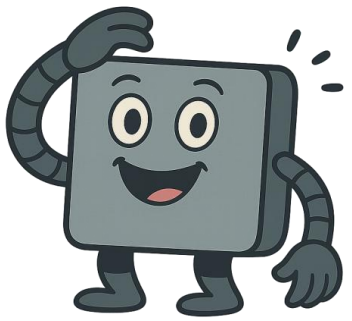


ANN: 클래스 작성 (3)

- Layer정보를 저장하기 위한 함수 정의 (cont'd).
  - Flatten, activation, add_layer마다 다른 옵션 및 설정을 가지고 있기 때문에 유의하면서 작성.

```
class ANN:
    def __add_layer(self, name, dim):
        # TODO SimpleNamespace 를 활용해서 구조체화
         layer = SimpleNamespace()
         if  name =='flatten' and dim==0:
        # TODO 'flatten ' 조건문에서는 dim==0으로 처리
        # TODO 'flatten ' 의 Property는 'function'
        # TODO 'activation' 조건문에서는 name=={relu, sigmoid, softmax'}로 처리
        # TODO 'activation' Property는 'act'
        # TODO 'ann layer' 조건문에서는 else 로 처리
        # TODO 'ann layer' Property는 'param'
        # TODO 'ann layer' dims는 '입력한 차원 값,
        # layer 통과시 출력 차원 값(output_dims)'
```

__add_layer 함수로 좀더 구체적으로 저장할 정보를 설정해줍시다.

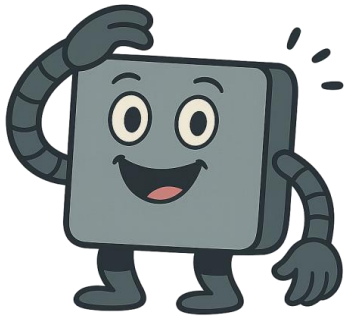


ANN: 클래스 작성 (4)

- Layer정보를 저장하기 위한 함수 정의 (cont'd).
  - Flatten, activation, add_layer마다 다른 옵션 및 설정을 가지고 있기 때문에 유의하면서 작성.

```
class ANN:
    def __init__(self):
    def show(self):
    # TODO 입력한 layer의 차원을 저장하기 위한 함수 정의.
    def __add_layer(self, name, dim):
        layer = SimpleNamespace()
        layer.dims=dim
        layer.name=name
        if  name =='flatten' and dim==0:
            layer.property='function'
            layer.params={'fnc':layer.name}
        elif dim == 0:
            layer.property='activation'
            layer.params={'act':layer.name}
```

__add_layer 함수로 좀더 구체적으로 저장할 정보를 설정해줍시다.

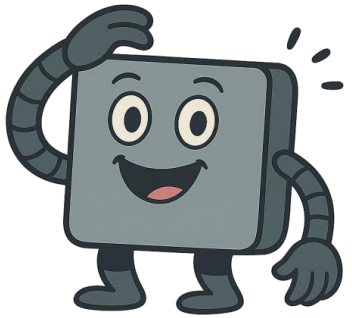


ANN: 클래스 작성 (5)

- Layer정보를 저장하기 위한 함수 정의 (cont'd).
  - Flatten, activation, add_layer마다 다른 옵션 및 설정을 가지고 있기 때문에 유의하면서 작성.

```
class ANN:
    def __init__(self):
    def show(self):
    # TODO 입력한 layer의 차원을 저장하기 위한 함수 정의.
    def __add_layer(self, name, dim):
            """
            이어서 작성
            """
        else:
            layer.property='param'
            layer.lr = self.lr
            layer.params=dict()
            layer.grads=dict()
            #layer.grad_params=dict()
        self.layers.append(layer)
```

Activation의 종류가 많으니 activation을 분류하는 함수를 새로 만들어 줘야겠다.

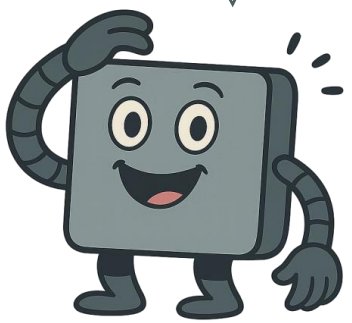


ANN: 클래스 작성 (6)

- Layer정보를 저장하기 위한 함수 정의 (cont'd).

```
class ANN:
    # TODO Activation
    def __add_activation_layer(self, name):
        self.__add_layer(name, 0)
## pubic
    def Flatten(self):

    def add_ann_layer(self, dims):

    def sigmoid(self):

    def relu(self):

    def softmax(self):
```

레이어를 쌓는 함수정의를 해보자.

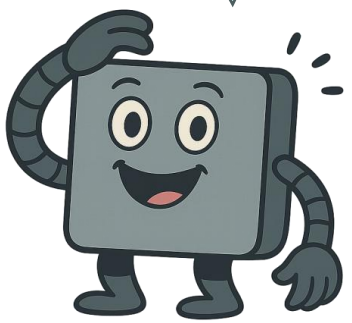


ANN: 클래스 작성 (7)

❑ Layer정보를 저장하기 위한 함수 정의 (cont'd).

```
class ANN:
    # TODO Activation
    def __add_activation_layer(self, name):
        self.__add_layer(name, 0)
## pubic
    def Flatten(self):
        self.__add_layer('flatten', 0)
    def add_ann_layer(self, dims):
        self.__add_layer('ANN layers', dims)
    def sigmoid(self):
        self.__add_activation_layer('sigmoid')
    def relu(self):
        self.__add_activation_layer('relu')
    def softmax(self):
        self.__add_activation_layer('softmax')
```

다음으로
레이어를 쌓아보고
출력해보자.



ANN: 클래스 작성 (8)

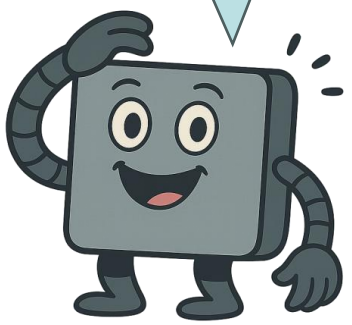
- Layer정보를 출력하는 함수 정의

```
def show(self):
    print("Layer name | parameters")
    print("=======================")
    for i in range(len(self.layers)):
        if self.layers[i].property=='param':
            print(self.layers[i].name)
        else:
            print(self.layers[i].name)
```

출력 결과

```
Layer name | parameters
=======================
flatten
ANN layers
relu
ANN layers
softmax
```

다음으로 ANN layer, activation, flatten함수들이 기능을 하도록 정의하자.
이 함수들은 private으로 설정 해야해.



ANN: 클래스 작성 (9)

❑ Layer의 기능을 정의

```
class ANN:
    def __Flatten(self, input):
    def __sigmoid(self, input):
    def __relu(self, input):
    def __softmax(self, input):
```

flatten

$$f(x) = x.reshape(N, -1)$$

sigmoid

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

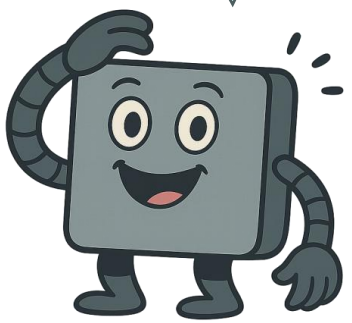
relu

$$f(x) = \max(0, x)$$

softmax

$$\sigma(x_k) = \frac{e^{x_k}}{\sum e^{x_j}}$$

순서대로 하나씩 작성을 해보자

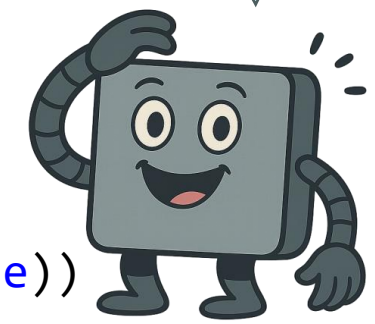


ANN: 클래스 작성 (10)

- Layer의 기능을 정의 (cont'd).

```
class ANN:
    def __Flatten(self, input):
        output = input.reshape(input.shape[0], -1)
        output_dims = output.shape[1]
        return output, output_dims

    def __sigmoid(self, input):
        output = sp.special.expit(input)
        return output

    def __relu(self, input):
        output = np.maximum(0, input)
        return output

    def __softmax(self, input):
        exp_input = np.exp(input - np.max(input, axis=-1, keepdims=True))
        output = exp_input / np.sum(exp_input, axis=-1, keepdims=True)
        return output
```

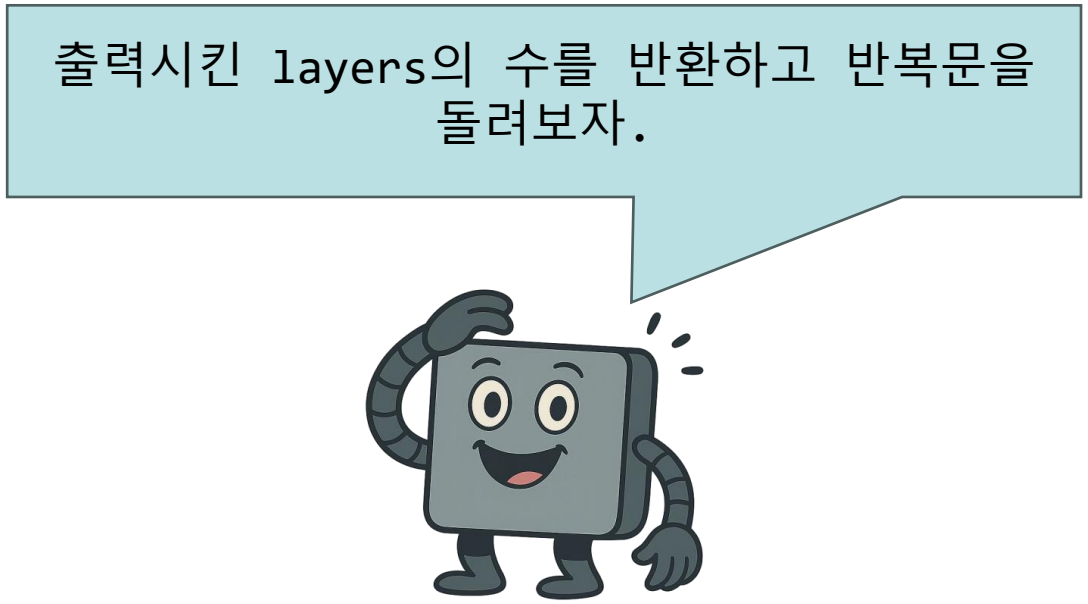
학습을 시킬 파라미터를 메모리에 할당하고 초기화를 하는 코드를 작성하자.



ANN: 클래스 작성 (11)

❑ 파라미터 초기화

```
class ANN:
    # TODO 파라미터를 저장할 변수는? layer_dims로부터 변수 초기화.
    def __param_init(self):
        # TODO layer의 순서는 i로하고, 파라미터는 하기와 같이 딕셔너리형태로 저장됨.
        # {w1 : (input_dims, output_dims), b1 : (1, output_dims) ....  }
```



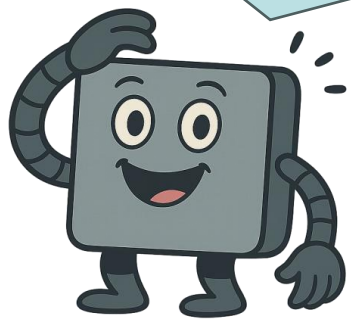
ANN: 클래스 작성 (12)

❑ 파라미터 초기화 (cont'd).

```
class ANN:
    # TODO 파라미터를 저장할 변수는? layer_dims로부터 변수 초기화.
    def __param_init(self):
        # TODO layer의 순서는 i로하고, 파라미터는 하기와 같이 딕셔너리형태로 저장됨.
        # {w1 : (input_dims, output_dims), b1 : (1, output_dims) ....  }
        nn = len(self.layers)
        for i in range(nn):
```

우리는 지금까지 layer의 속성을 'param', 'function', 'activation'으로 분류 했으니 조건문을 사용해 각 속성별로 원하는 역할을 할당해주자.

Hint. `self.layers[i].property`

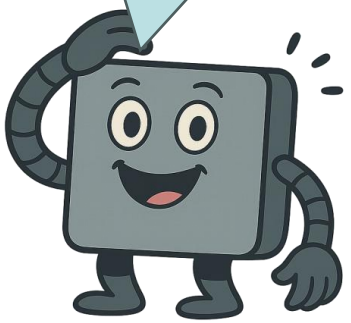


ANN: 클래스 작성 (13)

❑ 파라미터 초기화 (cont'd).

```
class ANN:
    # TODO 파라미터를 저장할 변수는? layer_dims로부터 변수 초기화.
    def __param_init(self):
        # TODO layer의 순서는 i로하고, 파라미터는 하기와 같이 딕셔너리형태로 저장됨.
        # {w1 : (input_dims, output_dims), b1 : (1, output_dims) ....  }
        nn = len(self.layers)
        for i in range(nn):
            if self.layers[i].property == 'param':

            elif self.layers[i].property=='function' :

            elif self.layers[i].property == 'activation':
```

Activation이나 flatten은 파라미터가 없으니 'param'부터 해보자.

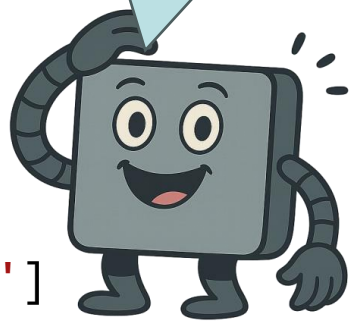


ANN: 클래스 작성 (14)

❑ 파라미터 초기화 (cont'd).

```
class ANN:
    # TODO 파라미터를 저장할 변수는? layer_dims로부터 변수 초기화.
    def __param_init(self):
        # TODO layer의 순서는 i로하고, 파라미터는 하기와 같이 딕셔너리형태로 저장됨.
        # {w1 : (input_dims, output_dims), b1 : (1, output_dims) ....  }
        nn = len(self.layers)
        for i in range(nn):
            if self.layers[i].property == 'param':

            elif self.layers[i].property=='function' :

            elif self.layers[i].property == 'activation':
```

Activation이나 flatten은 파라미터가 없으니 'param'부터 해보자.



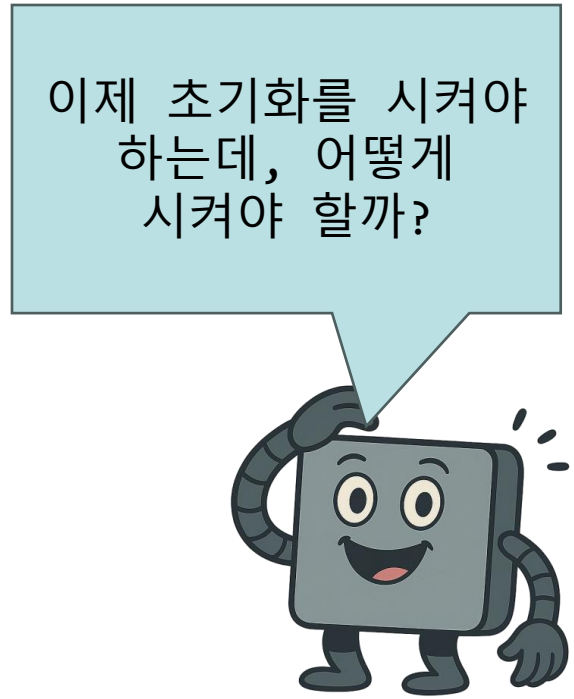
Hint.

params[f'w{i}'] params[f'b{i}']

ANN: 클래스 작성 (14)

❑ 파라미터 초기화 (cont'd).

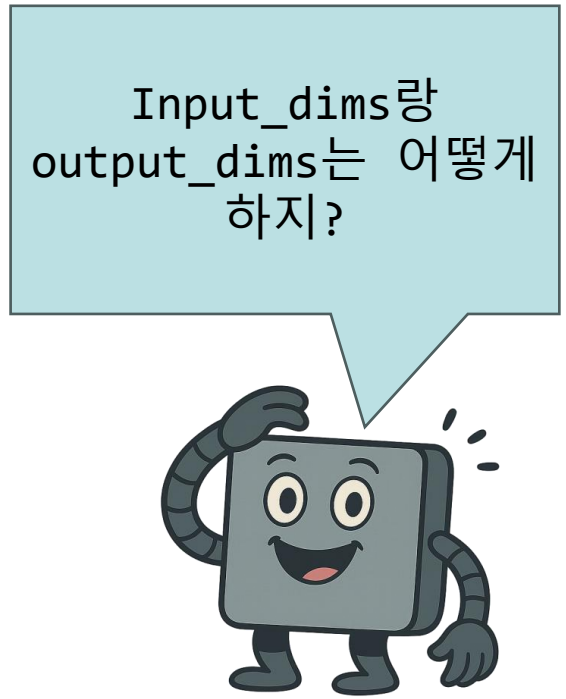
```
class ANN:
    # TODO 파라미터를 저장할 변수는? layer_dims로부터 변수 초기화.
    def __param_init(self):
        # TODO layer의 순서는 i로하고, 파라미터는 하기와 같이 딕셔너리형태로 저장됨.
        # {w1 : (input_dims, output_dims), b1 : (1, output_dims) ....  }
        nn = len(self.layers)
        for i in range(nn):
            if self.layers[i].property == 'param':
                self.layers[i].params[f'b{i}']
                self.layers[i].params[f'w{i}']
```



ANN: 클래스 작성 (15)

❑ 파라미터 초기화 (cont'd).

```
class ANN:
    # TODO 파라미터를 저장할 변수는? layer_dims로부터 변수 초기화.
    def __param_init(self):
        # TODO layer의 순서는 i로하고, 파라미터는 하기와 같이 딕셔너리형태로 저장됨.
        # {w1 : (input_dims, output_dims), b1 : (1, output_dims) ....  }
        nn = len(self.layers)
        for i in range(nn):
            if self.layers[i].property == 'param':
                self.layers[i].params[f'w{i}'] = \
            np.random.uniform(-1, 1, (input_dims, output_dims))

                self.layers[i].params[f'b{i}'] =\
            np.zeros((1, output_dims))
```



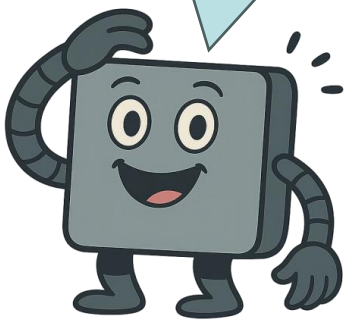
ANN: 클래스 작성 (16)

❑ 파라미터 초기화 (cont'd).

- Layer에서 초기화가 끝나면, input_dims에 현재 output_dims를 입력.

```
class ANN:
    # TODO 파라미터를 저장할 변수는? layer_dims로부터 변수 초기화.
    def __param_init(self):
        # TODO layer의 순서는 i로하고, 파라미터는 하기와 같이 딕셔너리형태로 저장됨
        # {w1 : (input_dims, output_dims), b1 : (1, output_dims) ...
        nn = len(self.layers)
        for i in range(nn):
            if self.layers[i].property == 'param':
                output_dims = self.layers[i].dims
                self.layers[i].params[f'b{i}'] = \
            np.random.uniform(-1, 1, (input_dims, output_dims))
                self.layers[i].params[f'w{i}'] =\
            np.zeros((1, output_dims))
            if function.   If activation.

            input_dims = output_dims
```

Activation과 flatten에 대해 작성하자.

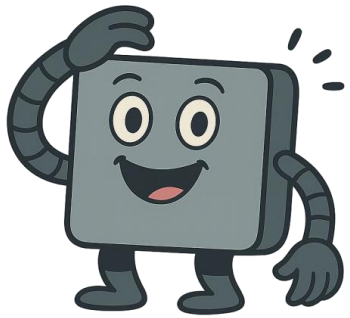


ANN: 클래스 작성 (17)

❑ 파라미터 초기화 (cont'd).

__param_init()는 compile() 함수를 만들어서 실행시키고, show함수에 파라미터 정보도 같이 출력하도록 하자.

```
class ANN:
    # TODO 파라미터를 저장할 변수는? layer_dims로부터 변수 초기화.
    def __param_init(self):
        # TODO layer의 순서는 i로하고, 파라미터는 하기와 같이 딕셔너리형태로 저장됨
        # {w1 : (input_dims, output_dims), b1 : (1, output_dims) ....  }
        nn = len(self.layers)
        for i in range(nn):
            """ param 생략 """
            elif self.layers[i].property=='function' :
                  _, input_dims = self.__Flatten(self.input_data)
                  continue
            elif self.layers[i].property == 'activation':
                  continue
           input_dims = output_dims
```



ANN: 클래스 작성 (18)

❑ 파라미터 초기화 확인

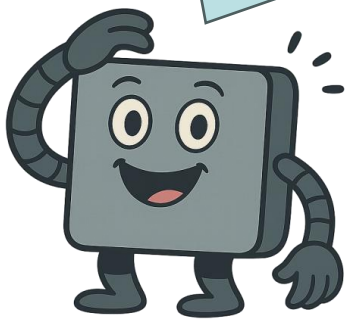
```
class ANN:
    def compile(self):
        self.__param_init()
    def show(self):
        print("Layer name | parameters")
        print("=========================")
        for i in range(len(self.layers)):
            if self.layers[i].property=='param':
                print(self.layers[i].name,
                      '|\t weight :', self.layers[i].params[f'w{i}'].shape,
                      '|\t bias :', self.layers[i].params[f'b{i}'].shape)
            else:
                print(self.layers[i].name)
```

ANN: 클래스 작성 (19)

❑ 파라미터 초기화 확인

```
test_ANN.Flatten()
test_ANN.add_ann_layer(128)
test_ANN.relu()
test_ANN.add_ann_layer(10)
test_ANN.softmax()
test_ANN.compile()
```

입력 데이터 정보가 정의가 되어 있지 않아 오류가 발생. Input 함수를 추가해 주면 되.



ANN: 클래스 작성 (20)

❑ 파라미터 초기화 확인

- Input 함수를 정의하고 Flatten위에 추가

```
class ANN:
    def Input(self, size):
        self.N, self.width, self.height = size
        self.input_data = np.zeros((self.N, self.width, self.height))
```

```
test_ANN.Input(size=image_train.shape)
test_ANN.Flatten()
test_ANN.add_ann_layer(128)
test_ANN.relu()
test_ANN.add_ann_layer(10)
test_ANN.softmax()
test_ANN.compile()
```

```
Layer name | parameters
==========================
flatten
ANN layers |     weight : (784, 128) |   bias : (1, 128)
relu
ANN layers |     weight : (128, 10) |    bias : (1, 10)
softmax
```

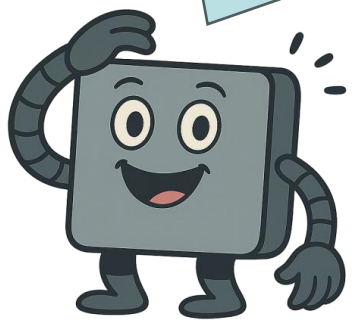
ANN: 클래스 작성 (21)

❑ 순전파 코드 구현

- layers.property에 따라 구별

```
def __forward(self):
        # TODO mnist 데이터 개수 (60000)에 대해 계산 필요.
        _X = np.array(self.input_data)
        for i in range(0,   len(self.layers)):
            if self.layers[i].property == 'param’:

            elif self.layers[i].property == 'function’:

            elif self.layers[i].property == 'activation’:

        return ypred
```

레이어 실행시 가장 먼저 실행되는 flatten부터 수행해보면



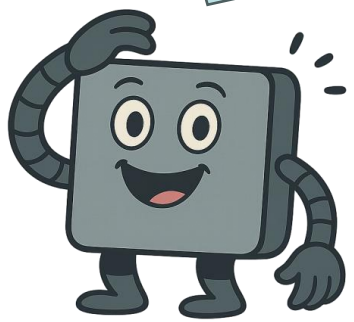
ANN: 클래스 작성 (22)

❑ 순전파 코드 구현

- layers.property에 따라 구별

```
def __forward(self):
        # TODO mnist 데이터 개수 (60000)에 대해 계산 필요.
        _X = np.array(self.input_data)
        for i in range(0,   len(self.layers)):
            if self.layers[i].property == 'param’:

            elif self.layers[i].property == 'function’:
                _X, _ = self.__Flatten(_X)
            elif self.layers[i].property == 'activation’:

        return ypred
```

그다음 “ANN layer” 연산을 수행하자.



ANN: 클래스 작성 (23)

그다음 activation 레이어를 통과시키자.

❑ 순전파 코드 구현

- layers.property에 따라 구별

```
def __forward(self):
        # TODO mnist 데이터 개수 (60000)에 대해 계산 필요.
        _X = np.array(self.input_data)
        for i in range(0,   len(self.layers)):
            if self.layers[i].property == 'param’:
                ypred = \
        np.matmul(_X, self.layers[i].params[f'w{i}']) + self.layers[i].params[f'b{i}’]
                _X=ypred
            elif self.layers[i].property == 'function’:
                _X, _ = self.__Flatten(_X)
            elif self.layers[i].property == 'activation’:

        return ypred
```

ANN: 클래스 작성 (23)

❑ 순전파 코드 구현

- 순전파 (__forward) 작성

```
def __forward(self):
        # TODO mnist 데이터 개수 (60000)에 대해 계산 필요.
        _X = np.array(self.input_data)
        for i in range(0,   len(self.layers)):
            elif self.layers[i].property == 'activation':
                if self.layers[i].name == 'relu':
                    ypred = self.__relu(_X)
                if self.layers[i].name == 'sigmoid':
                    ypred = self.__sigmoid(_X)
                if self.layers[i].name == 'softmax':
                    ypred = self.__softmax(_X)
        return ypred
```

Forward가 문제 없이
수행되는지 확인해 보자.

ANN: 클래스 작성 (24)

❑ 순전파 코드 구현

- Private으로 정의를 해서 새로운 함수 (run)을 만들어 실행.

```
class ANN:
    def run(self):
        self.__forward()
```

```
test_ANN.Input(size=image_train.shape)
test_ANN.Flatten()
test_ANN.add_ann_layer(128)
test_ANN.relu()
test_ANN.add_ann_layer(10)
test_ANN.softmax()
test_ANN.compile()
test_ANN.run()
```

ANN: 클래스 작성 (25)

❑ 순전파 코드 구현

- Private으로 정의를 해서 새로운 함수 (run)을 만들어 실행.

```
class ANN:
    def run(self, X, Y, batchsize = 32):
        self.input_data = X
        self.Ytgt = Y
        self.Batch = batchsize
        loss_score = []
        self.__forward()
```

```
test_ANN.Input(size=image_train.shape)
test_ANN.Flatten()
test_ANN.add_ann_layer(128)
test_ANN.relu()
test_ANN.add_ann_layer(10)
test_ANN.softmax()
test_ANN.compile()
test_ANN.run()
```

ANN: 클래스 작성 (25)

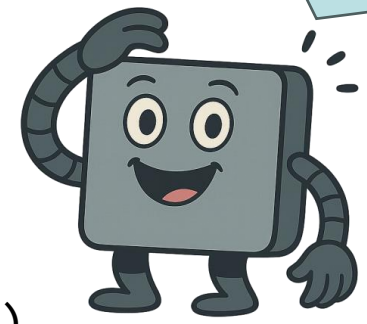
❑ 순전파 코드 구현

- Private으로 정의를 해서 새로운 함수 (run)을 만들어 실행.

```
class ANN:
    def run(self, X, Y, batchsize = 32):
        self.input_data = X
        self.Ytgt = Y
        self.Batch = batchsize
        loss_score = []
        self.__forward()
    return np.min(loss_score)

for i in range(epoch):
    test_ANN.run(X=image_train, Y=label_train, batchsize=32)
```

이제 categorical cross entropy 함수를 이용해서 Loss 를 구해보자

$$\text{categorical crossentropy}(\alpha_k) = \mathrm{Y}_{\mathrm{tgt}} \ln \frac{1}{\alpha_k} = Loss(\alpha_k)$$



ANN: 클래스 작성 (26)

❑ 순전파 코드 구현

- Private으로 정의를 해서 새로운 함수 (run)을 만들어 실행.

```
class ANN:
    def __categorical_cross_entropy(self, Ytgt, Ypred):
        # log (0)을 막기 위해 clip 추가
        Ypred = np.clip(Ypred, 1e-15, 1-1e-15)
        m = Ytgt.shape[0]
        return -np.sum(Ytgt * np.log(Ypred)) / m
```

Backpropagation을 구현하자.

ANN: 클래스 작성 (27)

❑ Backpropagation

$$Y = w_1 \times x_1 + w_2 \times x_2 + b$$

$$\text{softmax}(Y_k) = \frac{e^{Y_k}}{\sum e^{Y_i}} = \alpha_k$$

$$\text{categorical crossentropy}(\alpha_k) = Y_{tgt} \ln \frac{1}{\alpha_k} = Loss(\alpha_k)$$

$$\frac{\partial Loss}{\partial \alpha_k} = -\frac{Y_{tgt}}{\alpha_k}$$

$$\frac{\partial \alpha_k}{\partial Y} = \frac{\partial}{\partial Y_j}\left(\frac{e^{Y_k}}{\sum e^{Y_i}}\right)$$

$$\frac{d}{dx}\left(\frac{g(x)}{h(x)}\right) = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2}$$

w_1의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_1} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_1}$$

w_2의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial w_2} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial w_2}$$

b의 변화에 따른 Loss의 변화량

$$\frac{\partial Loss}{\partial b} = \frac{\partial Loss}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial Y}\frac{\partial Y}{\partial b}$$

ANN: 클래스 작성 (28)

❑ Backpropagation (cont'd)

$$\frac{\partial \alpha_k}{\partial Y} = \frac{\partial}{\partial Y_k}\left(\frac{e^{Y_k}}{\sum e^{Y_i}}\right) \Leftarrow \frac{d}{dx}\left(\frac{g(x)}{h(x)}\right) = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2}$$

K=j (k=j)

$$\frac{\partial \alpha_k}{\partial Y_j} = \frac{e^{Y_k} \times \sum e^{Y_i} - e^{Y_k} \times \left(0_{i-N} + 0_{i-N+1} + \cdots + e^{Y_j} + \cdots + 0_{i+N-1} + 0_{i+N}\right)}{(\sum e^{Y_i})^2} = \frac{e^{Y_k} \times \sum e^{Y_j} - e^{Y_k} e^{Y_j}}{(\sum e^{Y_i})^2}$$

$$= \frac{e^{Y_k}}{\sum e^{Y_j}} - \frac{e^{2Y_k}}{(\sum e^{Y_i})^2} = \alpha_k - \alpha_k \alpha_k = \alpha_j - \alpha_j \alpha_j$$

K≠j ($(e^{Y_k})' = 0$)

$$\begin{cases} (e^{Y_k})' = 0 & i \neq j \\ (e^{Y_k})' = e^{Y_k} & i = j \end{cases}$$

$$\frac{\partial \alpha_k}{\partial Y_j} = \frac{e^{Y_k} \times \sum e^{Y_j} - e^{Y_k} \times \sum e^{Y_i}}{(\sum e^{Y_i})^2} = \frac{-e^{Y_k} e^{Y_j}}{(\sum e^{Y_i})^2} = \frac{-e^{Y_k} e^{Y_j}}{(\sum e^{Y_i})^2} = -\alpha_k \alpha_j$$

$$\frac{\partial \alpha_k}{\partial Y_j} = \sum_k \alpha_k(\delta_j - \alpha_j)$$

$$\frac{\partial \alpha_k}{\partial Y_j}\frac{\partial Loss}{\partial \alpha_k} = -\sum_k \alpha_k(\delta_j - \alpha_j)\frac{Y_{tgt}}{\alpha_k}$$

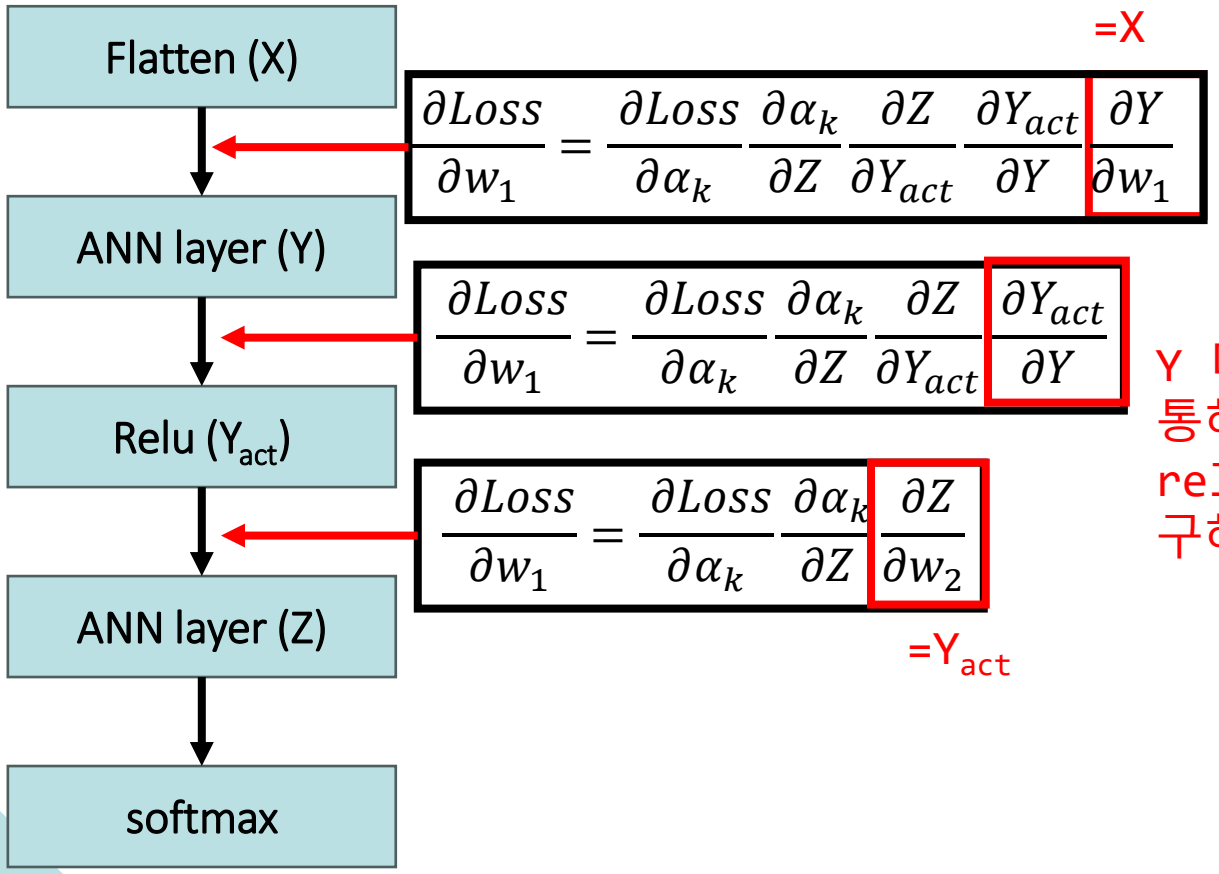
$$= -Y_{tgt}\Big|_k + \sum_k \alpha_j Y_{tgt}\Big|_k$$

Because Y_{tgt} is one-hot encoded, we get

$$= \alpha_k - Y_{tgt}$$

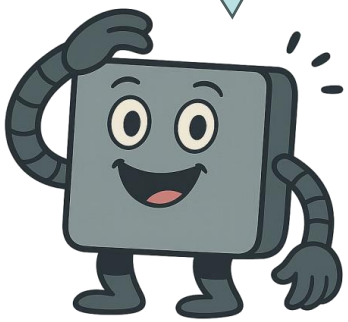
ANN: 클래스 작성 (29)

❑ 역전파 구현을 위한 구조 분석
- 붉은색 박스 데이터는 저장이 필요.



Y 데이터를 통해 gradient relu를 구해야함.

저장을 하는 이유는 해당 데이터를 얻기 위해 순전파를 다시 수행하지 않기 위해서야.

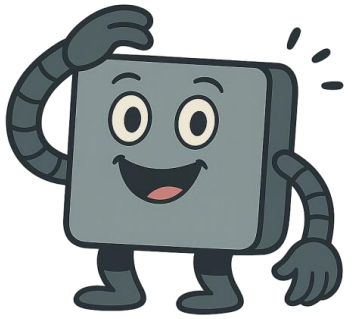


ANN: 클래스 작성 (30)

❑ 역전파 저장을 위한 cache 추가.

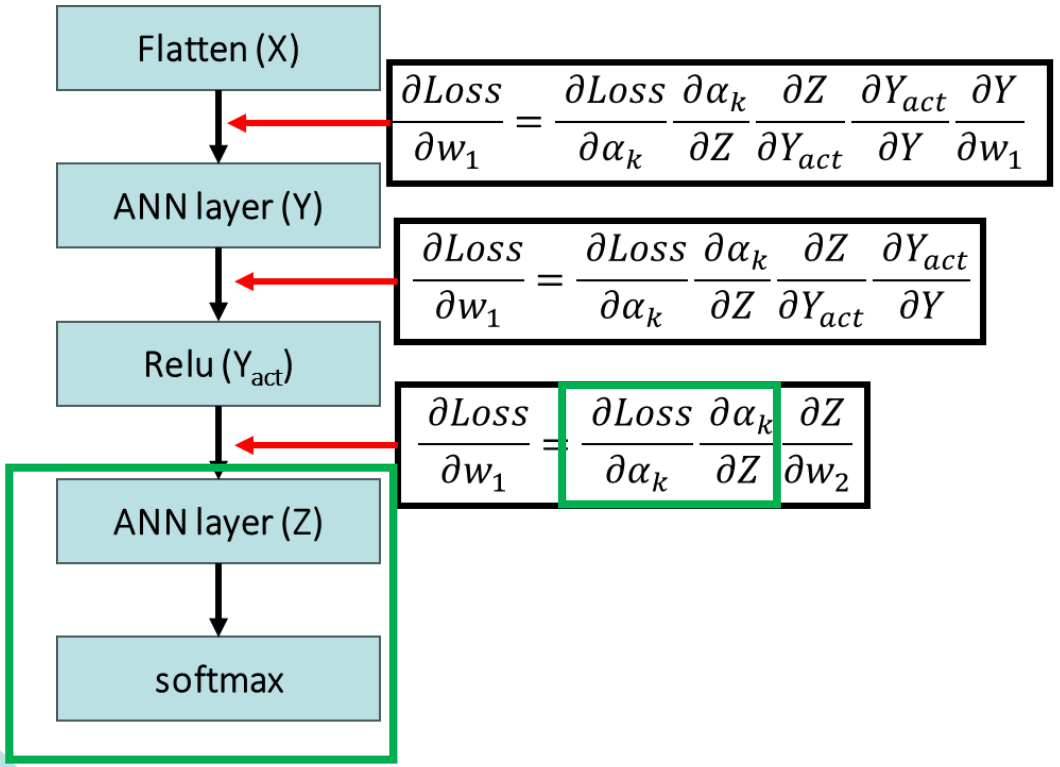
```
class ANN:
    def __forward(self):
        # TODO mnist 데이터 개수 (60000)에 대해 계산 필요.
        _X = np.array(self.input_data)
        self.cache = [] # iteration 마다 캐시 초기화
        for i in range(0,   len(self.layers)):
            if self.layers[i].property == 'param':
                """“ANN layer 실행 부분 생략. """
                self.cache.append(ypred)
                _X=ypred
            elif self.layers[i].property == 'function':
                _X, _ = self.__Flatten(_X)
                self.cache.append(_X)
            elif self.layers[i].property == 'activation’:
                """“activation 실행 부분 생략. """
                self.cache.append(ypred)
                _X=ypred
        return ypred
```

역전파를 구현해보자.



ANN: 클래스 작성 (31)

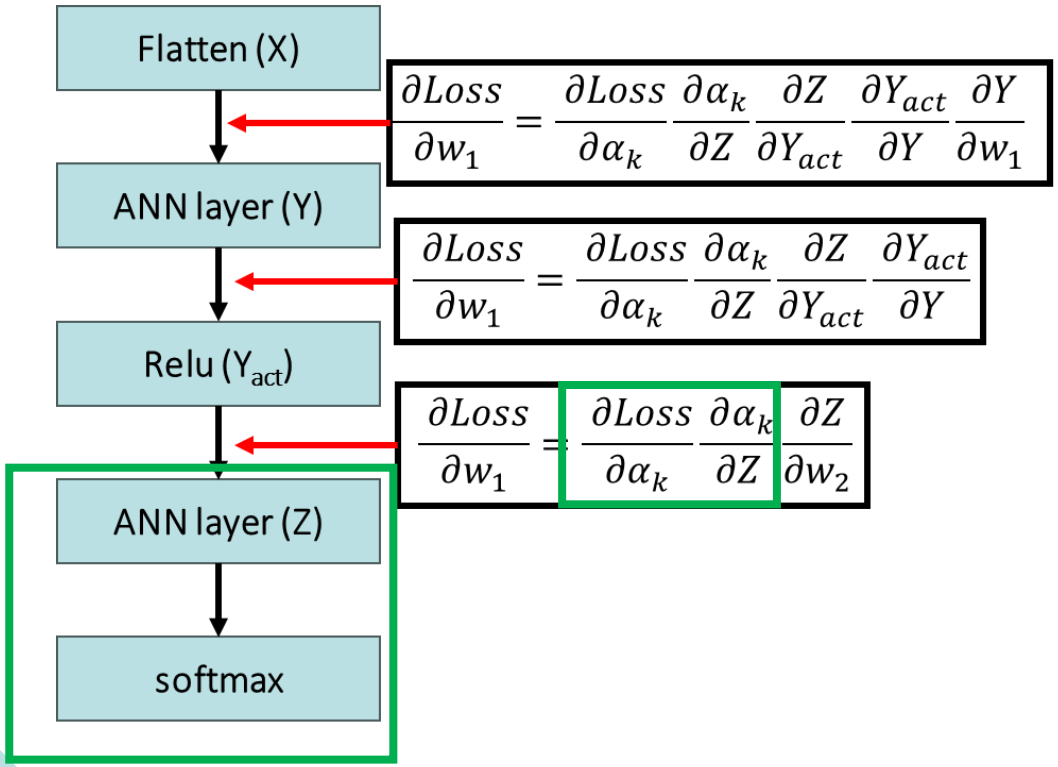
❑ 역전파: 말단 softmax + loss의 gradient




```
def __backward(self, Loss):
    # TODO  backward + softmax
    # softmax_result - Ptgt
    m = self.Ytgt.shape[0]
    # TODO  backward + softmax
    코드 작성
    for i in reversed(range(0,   len(self.layers)-1)):
```

ANN: 클래스 작성 (32)

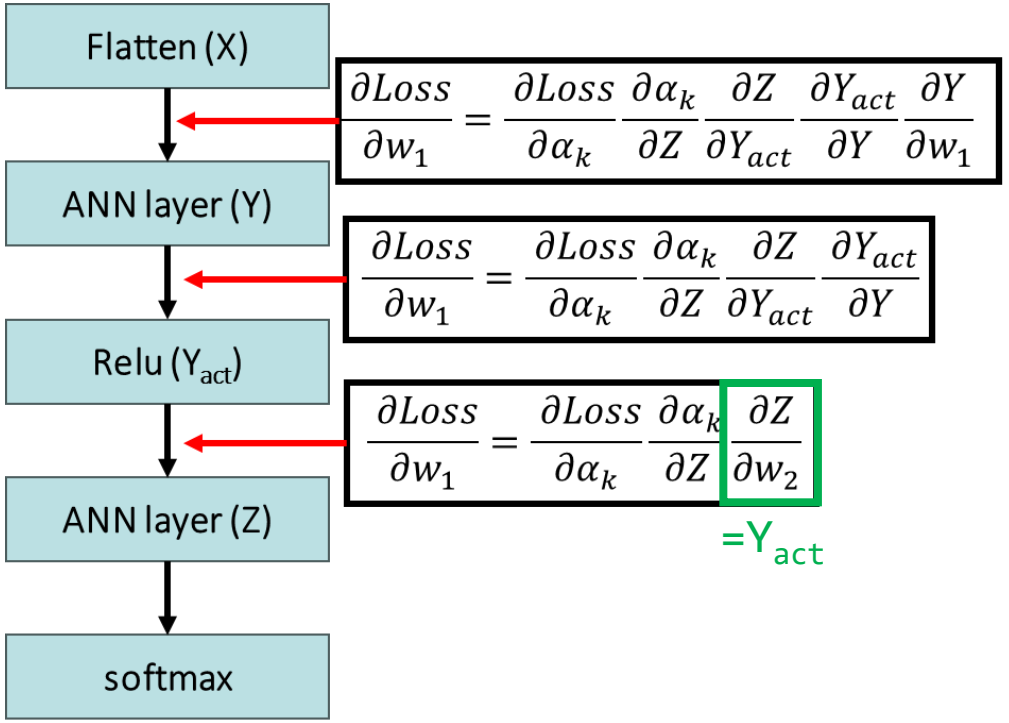
❑ 역전파: 말단 softmax + loss의 gradient (cont'd)




```
def __backward(self, Loss):
    # TODO  backward + softmax
    # softmax_result - Ptgt
    m = self.Ytgt.shape[0]
    # TODO  backward + softmax
    dA = self.cache[len(self.cache)-1] - self.Ytgt
    dZ = dA
    for i in reversed(range(0,   len(self.layers)-1)):
```

ANN: 클래스 작성 (33)

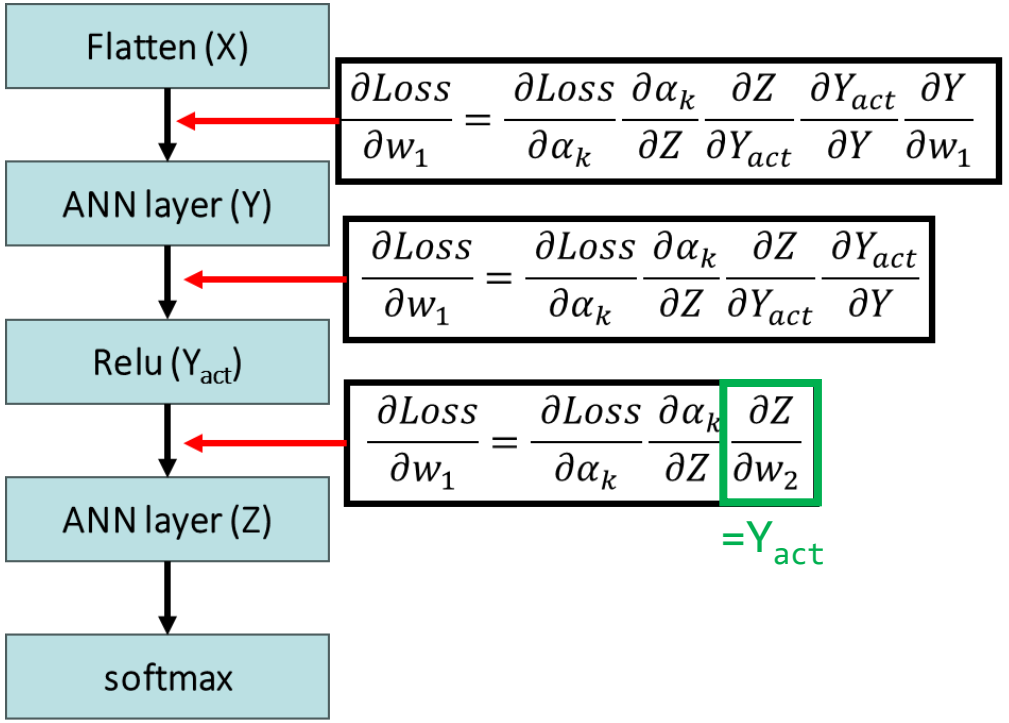
❑ 역전파: 마지막 ANN layer의 gradient




```
for i in reversed(range(0,   len(self.layers)-1)):
    if self.layers[i].property == 'activation’:

    elif self.layers[i].property == 'param':
        # TODO dW, db 값 구하기
```

ANN: 클래스 작성 (34)

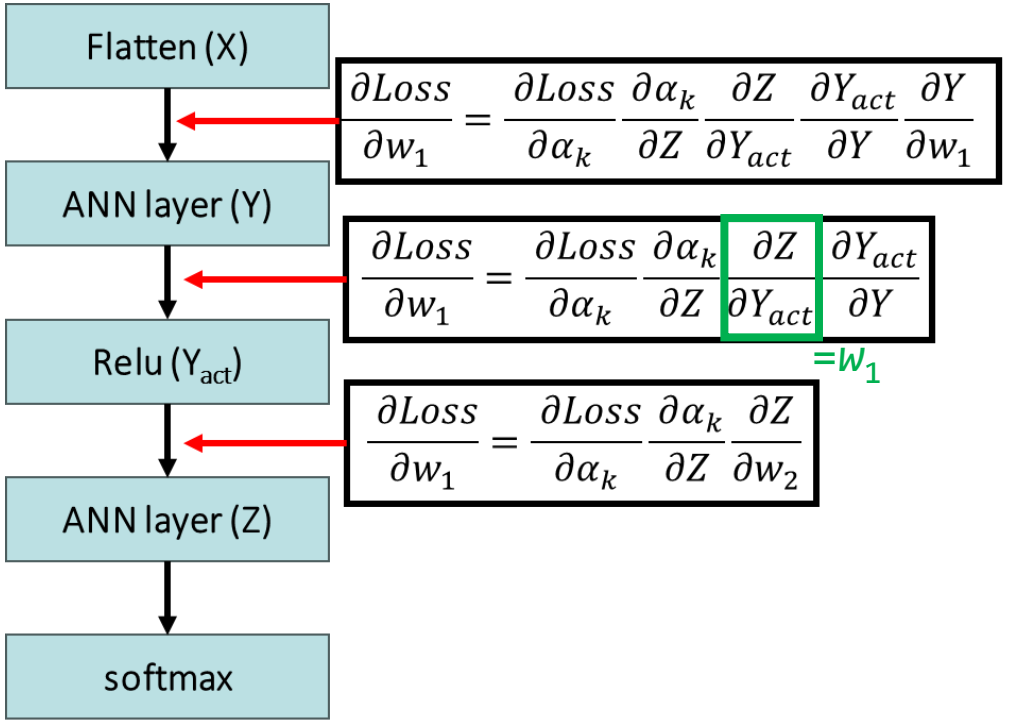
❑ 역전파: 마지막 ANN layer의 gradient (cont'd):




```
for i in reversed(range(0,   len(self.layers)-1)):
    if self.layers[i].property == 'activation':

    elif self.layers[i].property == 'param':
        # TODO dW, db 값 구하기
        dW =  np.matmul(self.cache[i-1].T, dZ)/m
        db =  np.sum(dZ, axis=0, keepdims = True)/m
```

ANN: 클래스 작성 (34)

❑ 역전파: activation에 넘겨주기 위한 Y_{act} 의 gradient를 구한다.

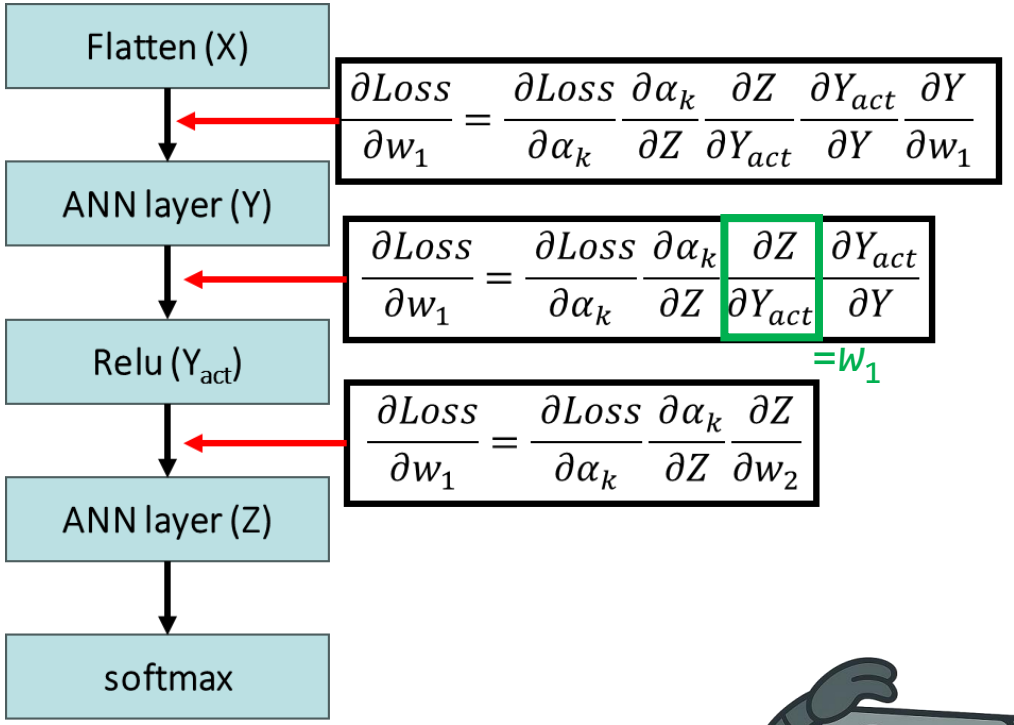



```
for i in reversed(range(0,   len(self.layers)-1)):
    if self.layers[i].property == 'activation’:

    elif self.layers[i].property == 'param':
        # TODO dW, db 값 구하기
        dW =  np.matmul(self.cache[i-1].T, dZ)/m
        db =  np.sum(dZ, axis=0, keepdims = True)/m
```

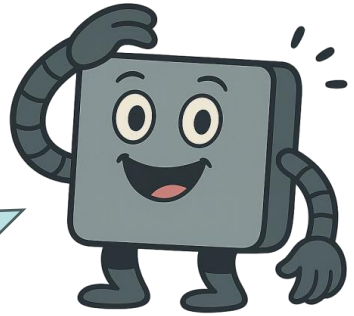
코드 작성

ANN: 클래스 작성 (35)

❑ 역전파: activation에 넘겨주기 위한 Y_{act} 의 gradient를 구한다. (cont'd)



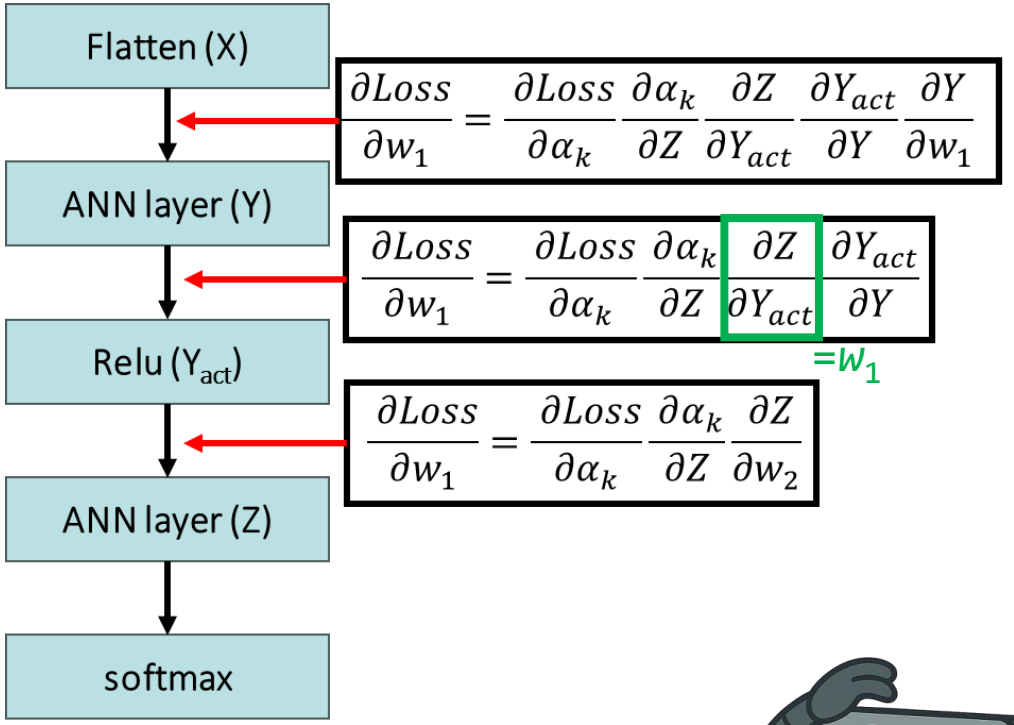
dW, db 를 저장해야해.



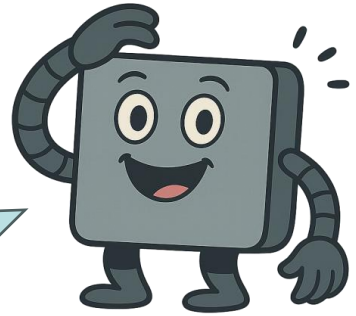
```
for i in reversed(range(0,   len(self.layers)-1)):
    if self.layers[i].property == 'activation':

    elif self.layers[i].property == 'param':
        # TODO dW, db 값 구하기
        dW =  np.matmul(self.cache[i-1].T, dZ)/m
        db =  np.sum(dZ, axis=0, keepdims = True)/m
        W = self.layers[i].params[f'w{i}']
        dA = dZ.dot(W.T)
```

ANN: 클래스 작성 (36)

❑ 역전파: activation에 넘겨주기 위한 Y_{act} 의 gradient를 구한다. (cont'd)



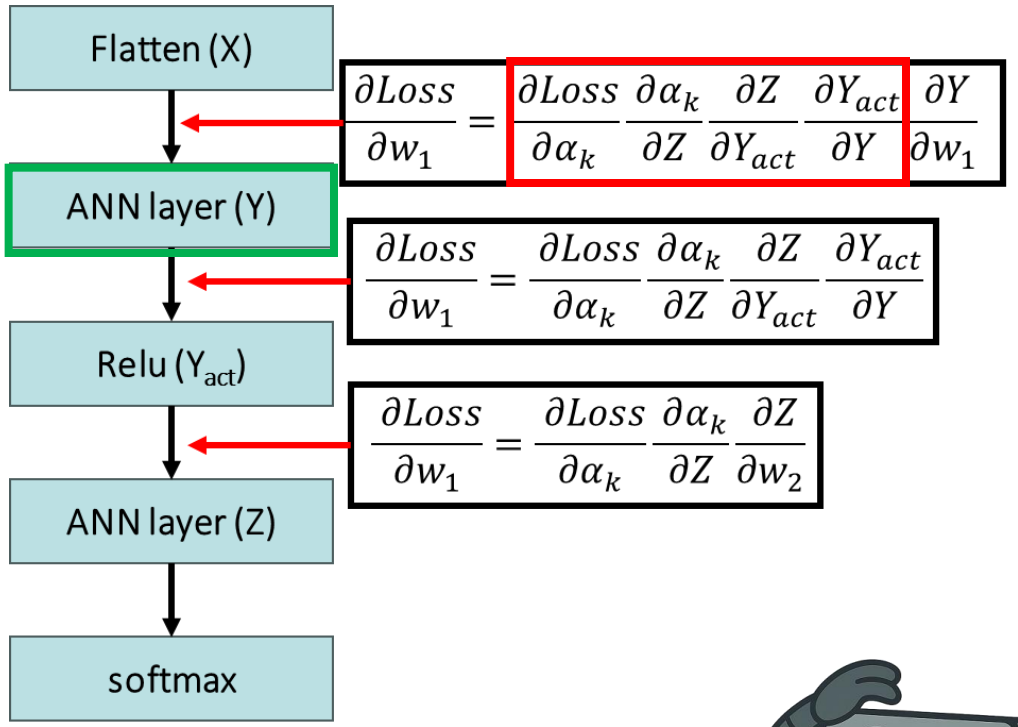
이제 activation을 해보자.



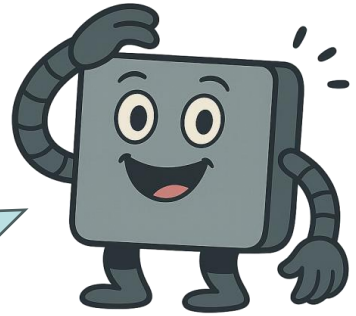
```
for i in reversed(range(0,   len(self.layers)-1)):
    if self.layers[i].property == 'activation':

    elif self.layers[i].property == 'param':
        # TODO dW, db 값 구하기
        dW =  np.matmul(self.cache[i-1].T, dZ)/m
        db =  np.sum(dZ, axis=0, keepdims = True)/m
        W = self.layers[i].params[f'w{i}']
        dA = dZ.dot(W.T)
        self.layers[i].grads[f"w{i}"] = dW
        self.layers[i].grads[f"b{i}"] = db
```

ANN: 클래스 작성 (37)

❑ 역전파: activation에 넘겨주기 위한 Y_{act} 의 gradient를 구한다. (cont'd)



이제 activation을 해보자.

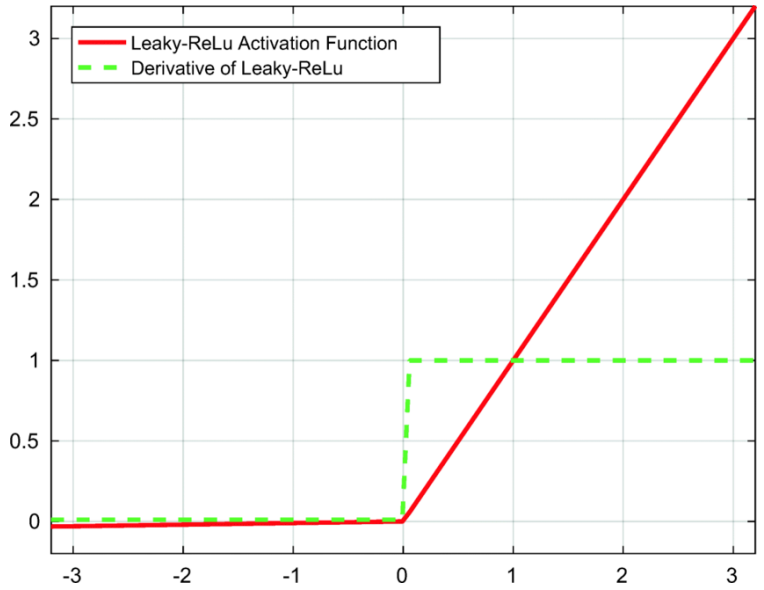


```
for i in reversed(range(0,   len(self.layers)-1)):
    if self.layers[i].property == 'activation':
        dZ = self.__backward_activation(
            dA, self.cache[i], self.layers[i].name)
    elif self.layers[i].property == 'param':
        # TODO dW, db 값 구하기
        dW =  np.matmul(self.cache[i-1].T, dZ)/m
        db =  np.sum(dZ, axis=0, keepdims = True)/m
        W = self.layers[i].params[f'w{i}']
        dA = dZ.dot(W.T)
        self.layers[i].grads[f"w{i}"] = dW
        self.layers[i].grads[f"b{i}"] = db
```

ANN: 클래스 작성 (38)

❑ 역전파의 activation을 아래처럼 작성.

```
def __backward_activation(self, dA, Z, name):
    # TODO relu
    if name == 'relu':
        dZ = dA.copy()
        dZ[Z<=0] = 0
        return dZ
```



https://www.researchgate.net/figure/Leaky-ReLU-activation-function_fig2_340791577

ANN: 클래스 작성 (39)

❑ 역전파진행이 완료되면 parameter update 수행.

```
    def __update_params(self):
        lr = self.lr
        num_layer = len(self.layers)
        for i in range(num_layer):
            if self.layers[i].property == 'param':
                self.layers[i].params[f"w{i}"] = self.__GD(self.layers[i].params[f"w{i}"] ,
self.layers[i].lr, self.layers[i].grads[f"w{i}"])
                self.layers[i].params[f"b{i}"] = self.__GD(self.layers[i].params[f"b{i}"] ,
self.layers[i].lr, self.layers[i].grads[f"b{i}"])
```

ANN: 클래스 작성 (40)

❑ 역전파 수행 과정을 run 함수에 업데이트

```
gage = tqdm(range(0, len(X), batchsize))
        for i in gage:
            self.input_data = X[i:i+batchsize]
            self.Ytgt = Y[i:i+batchsize]
            # TODO forward
            ypred = self.__forward()
            # TODO categorical crossentropy
            Loss = self.__categorical_cross_entropy(self.Ytgt, ypred)

            # TODO backward
            self.__backward(Loss)
            self.__update_params()
            loss_score.append(Loss)
            gage.mininterval = 0.5
            gage.set_description(f'loss : {Loss:.3f}')
        return np.min(loss_score)
```

ANN: 클래스 작성 (41)

❑ ANN 학습을 수행하고 epoch횟수 당 loss를 가시화.

```
for i in range(epoch):
    score = test_ANN.run(X = image_train, Y=label_train, batchsize=32)
    # best socre --> save
    score_loss.append(score)
        # test_ANN.save()
plt.figure()
plt.plot(score_loss)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.grid(True)
plt.show()
```

THANK YOU