

## PART 1

# 단층 퍼셉트론(SLP)

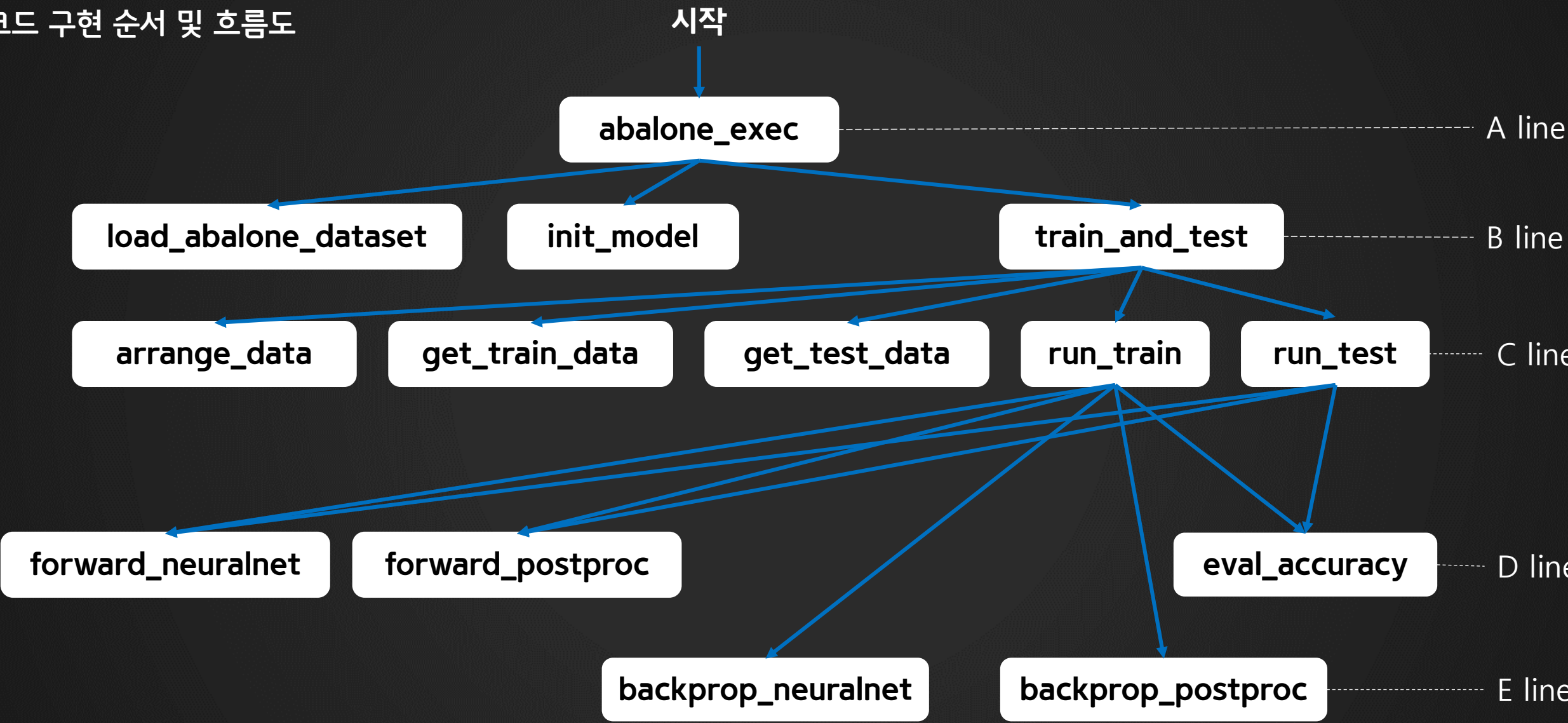
1장 회귀분석

2장 이진판단

3장 선택분류

딥러닝 & 강화학습 담당  
이재화 강사

코드 구현 순서 및 흐름도



---

## 0.0 파이썬 모듈 불러들이기

```
import numpy as np
import csv
#import time

np.random.seed(1234)
```

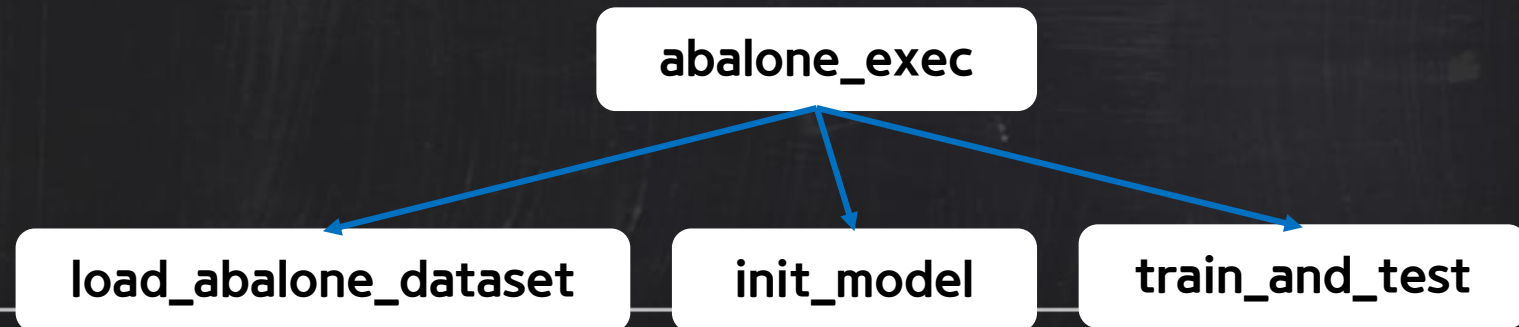
## 0.1 하이퍼 파라미터 정의

```
RND_MEAN = 0
RND_STD = 0.0030

LEARNING_RATE = 0.001
```

## A.1 실험용 메인함수

```
def abalone_exec(epoch_count=10, mb_size=10, report=1):  
  
    load_abalone_dataset()    #데이터 불러들이는 함수  
  
    init_model()              #모델 초기화 함수  
  
    train_and_test(epoch_count, mb_size, report) #학습 및 테스트 수행 함수
```



## B.1 데이터 적재함수 정의

```
def load_abalone_dataset():  
    with open('abalone.csv') as csvfile:  
        csvreader = csv.reader(csvfile)  
        next(csvreader, None)  
        rows = []  
        for row in csvreader:  
            rows.append(row)
```

csv파일의 데이터를 반복문을 활용하여  
rows 라는 빈 리스트에 저장


```
global data, input_cnt, output_cnt  
input_cnt, output_cnt = 10, 1  
#input_cnt = 10  
#output_cnt = 1  
data = np.zeros([len(rows), input_cnt+output_cnt])
```

이후에 다른 변수에서 활용하고자  
전역변수 생성

데이터의 입출력 벡터 정보를 저장.  
이후 크기 지정에 활용

```
for n, row in enumerate(rows):  
    if row[0] == 'I': data[n, 0] = 1  
    if row[0] == 'M': data[n, 1] = 1  
    if row[0] == 'F': data[n, 2] = 1  
    data[n, 3:] = row[1:]
```

원-핫 벡터 처리  
I = 1,0,0 / M = 0,1,0 / F = 0,0,1

  
infant   male   female

load\_abalone\_dataset



## next() 활용 예제

### next() 활용 0

```
with open('small_abalone.csv') as csvfile:
    csvreader = csv.reader(csvfile)
    next(csvreader, None)
    rows = []
    for row in csvreader:
        rows.append(row)
```



```
[[ 'M', '0.455', '0.365', '0.095', '0.514', '0.2245', '0.101', '0.15', '15'],
 [ 'M', '0.35', '0.265', '0.09', '0.2255', '0.0995', '0.0485', '0.07', '7']]
```

**원하는 데이터만 출력**

### next() 활용 X

```
with open('small_abalone.csv') as csvfile:
    csvreader = csv.reader(csvfile)
    rows = []
    for row in csvreader:
        rows.append(row)
```



```
[[ 'Sex',
   'Length',
   'Diameter',
   'Height',
   'Whole weight',
   'Shucked weight',
   'Viscera weight',
   'Shell weight',
   'Rings'],
 [ 'M', '0.455', '0.365', '0.095', '0.514', '0.2245', '0.101', '0.15', '15'],
 [ 'M', '0.35', '0.265', '0.09', '0.2255', '0.0995', '0.0485', '0.07', '7']]
```

**변수명 까지 함께 출력**

# 데이터 적재함수 통과 후의 데이터 확인\_1

```
with open('abalone.csv') as csvfile:
    csvreader = csv.reader(csvfile)
    next(csvreader, None)
    rows = []
    for row in csvreader:
        rows.append(row)

print(rows[0:4])
```

*#함수를 생성하는 부분 없이  
단편적으로 코드 실행*

*#데이터가 저장되어 있는  
rows의 4번째 줄 까지 출력*



```
[['M', '0.455', '0.365', '0.095', '0.514', '0.2245', '0.101', '0.15', '15'],
['M', '0.35', '0.265', '0.09', '0.2255', '0.0995', '0.0485', '0.07', '7'],
['F', '0.53', '0.42', '0.135', '0.677', '0.2565', '0.1415', '0.21', '9'],
['M', '0.44', '0.365', '0.125', '0.516', '0.2155', '0.114', '0.155', '10']]
```

# np.zeros()

```
import numpy as np

global data, input_cnt, output_cnt
input_cnt, output_cnt = 10, 1

data = np.zeros([len(rows), input_cnt+output_cnt])

print(data)
print(data.shape)
```



```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
(4177, 11)
```

# np.zeros()는 지정해준 크기만큼 0값의 행렬을 생성!

# .shape를 활용하시면 행과 열을 각각 확인 가능!



## 데이터 적재함수 통과 후의 데이터 확인\_2

```
with open('abalone.csv') as csvfile:
    csvreader = csv.reader(csvfile)
    next(csvreader, None)
    rows = []
    for row in csvreader:
        rows.append(row)

global data
data = np.zeros([len(rows), input_cnt+output_cnt])

for n, row in enumerate(rows):
    if row[0] == 'I': data[n, 0] = 1
    if row[0] == 'M': data[n, 1] = 1
    if row[0] == 'F': data[n, 2] = 1
    data[n, 3:] = row[1:]

print(data)
```

성별정보  
원 핫 벡터 처리



[	0.	1.	0.	...	0.101	0.15	15.	]
[	0.	1.	0.	...	0.0485	0.07	7.	]
[	0.	0.	1.	...	0.1415	0.21	9.	]
...								
[	0.	1.	0.	...	0.2875	0.308	9.	]
[	0.	0.	1.	...	0.261	0.296	10.	]
[	0.	1.	0.	...	0.3765	0.495	12.	]

이후 데이터는  
그대로 복사



## B.2 파라미터 초기화 함수 정의

```
def init_model():
```

전역변수 불러오기 및 생성

```
    global weight, bias, input_cnt, output_cnt
```

```
    weight = np.random.normal(RND_MEAN, RND_STD, [input_cnt, output_cnt])
```

```
    bias = np.zeros([output_cnt])
```

가중치, 편향 초기화 단계

※ np.random.normal(평균,표준편차,크기) = 정규분포를 갖는 난수 생성

※ np.zeros(shape) = 0으로 가득찬 array를 생성

## B.3 학습 및 평가 함수 정의

train\_and\_test

arrange\_data

get\_train\_data

get\_test\_data

run\_train

run\_test

```
def train_and_test(epoch_count, mb_size, report):
```

```
    step_count = arrange_data(mb_size)
```

```
    test_x, test_y = get_test_data()
```

```
    for epoch in range(epoch_count):
```

```
        losses, accs = [], []
```

← epoch\_count 만큼 '에폭' 반복 수행  
한차례의 에폭마다의 손실과 정확도 저장

```
        for n in range(step_count):
```

← 학습데이터 크기에 비례하여 (80%)  
미니배치 처리된 횟수 만큼 반복 수행

```
            train_x, train_y = get_train_data(mb_size, n)
```

```
            loss, acc = run_train(train_x, train_y)
```

← • 미니배치 마다의 학습 데이터 분할  
• 학습 수행 및 손실과 정확도 산출

```
            losses.append(loss)
```

```
            accs.append(acc)
```

← 미니배치 처리 이후 손실과 정확도를 누적하여 저장  
(이후 이 값들을 평균내면 한 차례의 '에폭' 처리)

```
    if report > 0 and (epoch+1) % report == 0:
```

```
        acc = run_test(test_x, test_y)
```

← • 출력 주기 및 테스트 주기 설정  
• 테스트 데이터로 테스트 진행

```
        print('Epoch {}: loss={:5.3f}, accuracy={:5.3f}/{:5.3f}'.
```

```
              format(epoch+1, np.mean(losses), np.mean(accs), acc))
```

```
    final_acc = run_test(test_x, test_y)
```

← 모든 반복 종료되었을 때, 한번 더 최종 결과 출력

```
    print('\nFinal Test: final accuracy = {:5.3f}'.format(final_acc))
```

# arrange\_data()의 반환값, 미니배치 스텝 수 확인

```
print("총 데이터의 수(행)", data.shape[0])

mb_size = 100
step_count = int(data.shape[0] * 0.8) // mb_size
print("데이터의 80%의 미니배치 스텝수 :", step_count)
print(33*100)

print("-"*10)

step_count = int(data.shape[0] * 1) // mb_size
print("데이터의 100%의 미니배치 스텝수 :", step_count)
print(41*100)
```

총 데이터의 수(행) 4177  
데이터의 80%의 미니배치 스텝수 : 33  
3300  
-----  
데이터의 100%의 미니배치 스텝수 : 41  
4100

## 미니 배치 (mini-batch)

학습 또는 추론의 단일 반복에서 함께 실행되는 예의 전체 배치 중에서 무작위로 선택한 소규모 부분집합입니다. 미니 배치의 배치 크기는 일반적으로 10~1,000입니다. 전체 학습 데이터가 아닌 미니 배치의 손실을 계산하면 효율성이 크게 향상됩니다.



## .format 예제

```
print('Epoch {}: loss = {:.5.3f}, accuracy = {:.4.3f} / {:.3.3f}'.
```

```
    format(100+1, 0.123456, 1000.1234567, 0.123456))
```

#: (콜론)을 중심으로 할당, :5.3f 에서 5는 다섯자리를 확보 / .이하값은 소숫점 이하값 설정  
/ f는 부동소수 표현

```
Epoch 101: loss = 0.123, accuracy = 1000.123 / 0.123
```

## C.1~3 학습 및 평가 데이터 획득 함수 정의

arrange\_data

get\_train\_data

get\_test\_data

```
def arrange_data(mb_size):
    global data, shuffle_map, test_begin_idx
    shuffle_map = np.arange(data.shape[0])
    np.random.shuffle(shuffle_map)
    step_count = int(data.shape[0] * 0.8) // mb_size
    test_begin_idx = step_count * mb_size
    return step_count
```

데이터의 순서값을 생성  
데이터를 무작위로 섞어주는 과정  
데이터의 80%기준, 미니배치 사이즈에 의한  
1 에폭당 미니배치 횟수 출력  
학습 데이터와 테스트 데이터의 경계선 인덱스 생성

```
def get_test_data():
    global data, shuffle_map, test_begin_idx, output_cnt
    test_data = data[shuffle_map[test_begin_idx:]]
    return test_data[:, :-output_cnt], test_data[:, -output_cnt:]
```

테스트 데이터 생성  
테스트 데이터의  
종속변수, 독립변수 분할

```
def get_train_data(mb_size, nth):
    global data, shuffle_map, test_begin_idx, output_cnt
    if nth == 0:
        np.random.shuffle(shuffle_map[:test_begin_idx])
    train_data = data[shuffle_map[mb_size*nth:mb_size*(nth+1)]]
    return train_data[:, :-output_cnt], train_data[:, -output_cnt:]
```

(미니배치 크기, 미니배치 실행 순서)  
첫 에폭마다 한하여,  
처음부터 경계선까지 인덱스를 섞어줍니다.  
섞인 인덱스로 미니배치 크기에  
맞게 데이터 분할 및 train\_data  
로 저장



# test\_begin\_idx의 이해

```
mb_size = 100
step_count = int(data.shape[0] * 0.8) // mb_size

test_begin_idx = step_count * mb_size
print("학습 데이터와 테스트 데이터의 경계선 index : ", test_begin_idx)
```

학습 데이터와 테스트 데이터의 경계선 index : 3300

## 새로운 에폭인 경우, 순서 인덱싱 및 무작위 섞기

```
#순서 생성
shuffle_map = np.arange(data.shape[0])

#미니배치 스텝 카운트 생성
step_count = int(data.shape[0] * 0.8) // mb_size

#테스트데이터, 학습데이터 경계 인덱스 생성
test_begin_idx = step_count * mb_size
print("경계 인덱스 생성 : ",test_begin_idx)
#일반적인 순서
print("일반적인 순서\n",shuffle_map)
#
np.random.shuffle(shuffle_map[:test_begin_idx])
print("처음부터 경계선까지의 순서 셔플 \n",shuffle_map)
```

경계 인덱스 생성 : 3300

일반적인 순서

[ 0 1 2 ... 4174 4175 4176]

처음부터 경계선까지의 순서 셔플

[1730 112 1006 ... 4174 4175 4176]

## nth에 의한 미니배치 구간 선정 ¶

```
nth = 0
mb_size = 100

train_data = data[shuffle_map[mb_size * nth : mb_size * (nth+1)]]
print(train_data.shape)

print("---"*20)

for n, i in enumerate(train_data[0:5]):
    print(n, i)
```

(100, 11)

```
-----
0 [ 0.      1.      0.      0.665  0.505  0.16    1.289  0.6145  0.253
   0.3665 11.      ]
1 [1.      0.      0.      0.435  0.32   0.08    0.3325 0.1485 0.0635 0.105
   9.      ]
2 [ 0.      1.      0.      0.605  0.47   0.165   1.2315 0.6025 0.262
   0.2925 11.      ]
3 [ 0.      0.      1.      0.58   0.455  0.12    0.94   0.399  0.257 0.265
   11.      ]
4 [1.      0.      0.      0.23   0.17   0.05   0.057 0.026 0.013 0.016 5.      ]
```

## C.4 학습 및 평가 데이터 획득 함수 정의

```
def run_train(x, y):  
    output, aux_nn = forward_neuralnet(x)  ← 신경망 연산 부분 (보조정보 : 입력벡터  $x$ )  
    loss, aux_pp = forward_postproc(output, y)  ← 신경망 후처리 과정(손실함수 구하는 과정)  
    accuracy = eval_accuracy(output, y)  ← 정확도를 구하는 과정 (보조정보 : 편차  $diff$ )
```

↑ 순전파 및 정확도 추출 과정   ↓ 역전파 과정

# 항상 순전파의 역순으로 수행

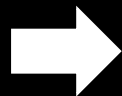
$G\_loss = 1.0$    # 순전파 때 출력이었던 성분의 '손실함수의 기울기'    $\frac{\partial L}{\partial L} = 1.0$

$G\_output = \text{backprop\_postproc}(G\_loss, aux\_pp)$    # 손실함수의 처리과정인 '평균제곱오차'의 역순, 즉 'MSE의 역전파 처리'

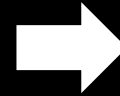
$\text{backprop\_neuralnet}(G\_output, aux\_nn)$    # 입력값에 따른  $f(x)$ 에 대한 편미분 과정을 구해주는 내부처리  $\frac{\partial f(x)}{\partial x}$

# 직접적인 학습이 이뤄지는 부분  
(가중치와 편향이 학습률을 활용하며 실제 학습 과정 수행)

$$\frac{\partial L}{\partial L} = 1.0$$



$$\frac{\partial L}{\partial Y} \frac{\partial f(x)}{\partial x} = \frac{\partial L}{\partial x}$$



입력값으로 반환

return loss, accuracy

순전파 과정

역전파 과정

순전파의 뒷 단계 '후처리 과정'

forward\_postproc()

-> loss

손실에 대한 기울기

$G\_loss$

$$\frac{\partial L}{\partial L} = 1$$

순전파의 뒷 단계였던 후처리 과정에 대한  
역전파 함수  $\text{backprop\_postproc}()$ 가 먼저 호출

$\text{backprop\_postproc}()$

$$\frac{\partial L}{\partial Y}$$

$G\_output$

run\_train

forward\_neuralnet

forward\_postproc

backprop\_neuralnet

backprop\_postproc

eval\_accuracy

## C.5 학습 및 평가 데이터 획득 함수 정의

```
def run_test(x, y):  
    output, _ = forward_neuralnet(x)  
    accuracy = eval_accuracy(output, y)  
    return accuracy
```

- 순전파 과정 수행  
(두번째 반환값인 '추가정보 반환'은 필요 없으므로 "\_" 처리)
- 최종 정확도 추출



# D.1 & E.1 단층 퍼셉트론에 대한 순전파 및 역전파 함수 정의

```
def forward_neuralnet(x):
    global weight, bias
    output = np.matmul(x, weight) + bias
    return output, x

def backprop_neuralnet(G_output, x):
    global weight, bias
    g_output_w = x.transpose()

    G_w = np.matmul(g_output_w, G_output)
    G_b = np.sum(G_output, axis=0)

    weight -= LEARNING_RATE * G_w
    bias -= LEARNING_RATE * G_b
```

전역변수 셋팅

편향이 더해진 입력 벡터와 가중치 벡터에 대한 기본적인 신경망 연산식

역전파에 필요한 보조정보로 활용

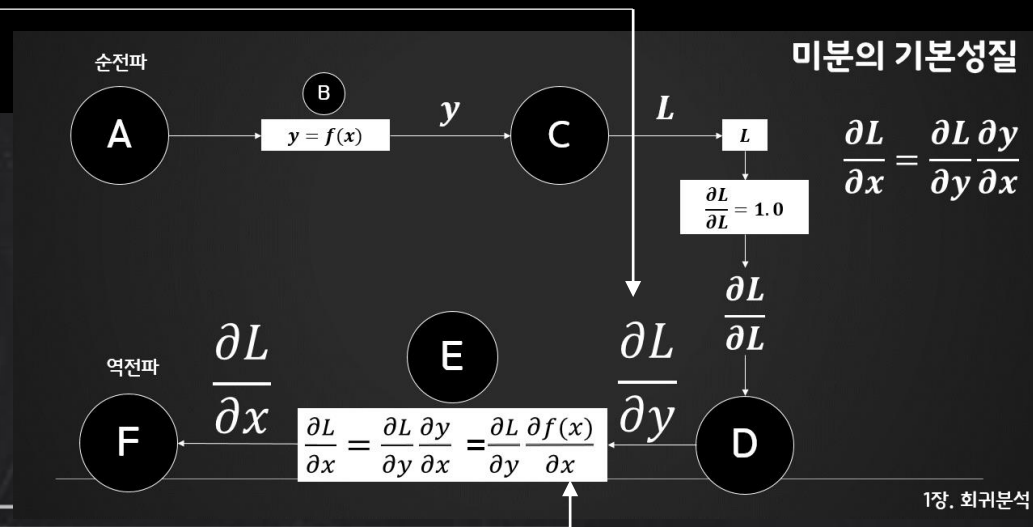
입력값에 따른  $f(x)$ 에 대한 편미분 과정에서 각각 가중치( $G_w$ )와 편향( $G_b$ )의 손실 기울기 연산

가중치의 손실기울기를 구하기 위해 필요한 값에 대한 사전작업

※ np.matmul() : 두 배열의 행렬곱 연산을 수행합니다.

※ .transpose() : 해당 변수에 대한 전치행렬 수행

$$\frac{\partial f(x)}{\partial x} \rightarrow \begin{cases} \frac{\partial f(x)}{\partial W} \\ \frac{\partial f(x)}{\partial B} \end{cases}$$





Part 1. 단층 퍼셉트론

※ 가중치와 편향의 손실 기울기 (1)

#최종목표

$\frac{\partial L}{\partial W}$

→

$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial W}$

#연쇄법칙

#출력값을 통해 미치는 부분

#공통적으로 따지는 부분

입력 행렬 X의  
i - 행 벡터

『내적』

가중치 행렬 W의  
j - 열 벡터

『덧셈』

편향 벡터 B의  
원소  $B_j$

원소  $Y_{ij}$

$\frac{\partial Y}{\partial W}$ 

→

$Y = XW + B$

python code :  
g\_output\_w = x.transpose()  
G\_w = np.matmul(g\_output\_w, G\_output)

길게 풀어보면 ...

$Y_{ij} = X_{i1}W_{1j} + X_{i2}W_{2j} + \cdots + X_{in}W_{nj} + B_j$ 

$X_{ik}W_{kj} + B_j$

→

$\frac{\partial Y_{ij}}{\partial W_{kj}} = X_{ik}$

※ 가중치에 대한 출력값의 편미분

## Part 1. 단층 퍼셉트론

### ※ 가중치와 편향의 손실 기울기 (2)

#최종목표

#연쇄법칙

$$\frac{\partial L}{\partial W}$$



$$\frac{\partial L}{\partial W}$$

$$= \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial W}$$

#출력값을 통해 미치는 부분

#공통적으로 따지는 부분

$$\frac{\partial L}{\partial Y}$$

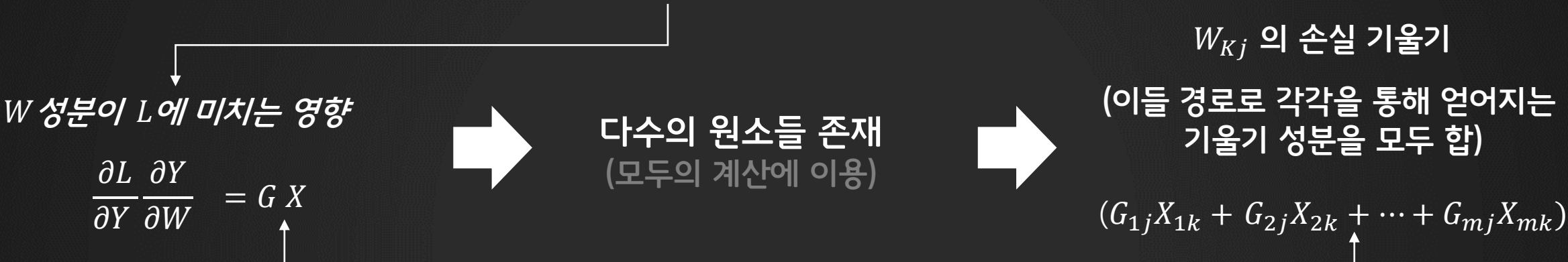
$Y_{ij}$ 성분의 손실 기울기

$$\frac{\partial L}{\partial Y_{ij}} = G_{ij}$$

Part 1. 단층 퍼셉트론

※ 가중치와 편향의 손실 기울기 (3)

$\frac{\partial L}{\partial W} \rightarrow \frac{\partial L}{\partial W} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial W} \rightarrow \frac{\partial L}{\partial Y_{ij}} = G_{ij} \quad \frac{\partial Y_{ij}}{\partial W_{kj}} = X_{ik}$



$\frac{\partial L}{\partial W_{kj}} = T_{k1}G_{1j} + T_{k2}G_{2j} + \dots + T_{km}G_{mj}$

↓ (행렬곱셈에 알맞게 재구성)

최종적인 가중치에 대한 손실함수 수식  $\frac{\partial L}{\partial W} = T G = X^T G$

← X 대신 행과 열을 뒤바꾼 전치행렬  $T = X^T$

```
PYTHON:
g_output_w = x.transpose()
G_w = np.matmul(g_output_w, G_output)
```



Part 1. 단층 퍼셉트론

※ 가중치와 편향의 손실 기울기 (1)

$\frac{\partial L}{\partial B} \rightarrow \frac{\partial L}{\partial B} = \frac{\partial L}{\partial Y} \frac{\partial Y}{\partial B}$

$Y_{ij}$ 성분의 손실 기울기

$\frac{\partial L}{\partial Y_{ij}} = G_{ij}$

$Y = XW + B$   
 $Y_{ij} = X_{i1}W_{1j} + X_{i2}W_{2j} + \dots + X_{in}W_{nj} + B_j$

편미분의 성질로 인한  
연산결과 1 출력

$\frac{\partial Y_{ij}}{\partial B_j} = 1$

$\frac{\partial L}{\partial Y_{ij}} \frac{\partial Y_{ij}}{\partial B_j} = 1 * G_{ij}$

$\frac{\partial L}{\partial B_j} = G_{1j} + G_{2j} + \dots + G_{mj}$

이 계산은 1로 채워진 행렬과 G의 곱으로 구할 수도 있겠지만 구현 코드에서와 같이 단순히 G의 각 행의 합을 구하는 간단한 방법으로도 구할 수 있습니다.

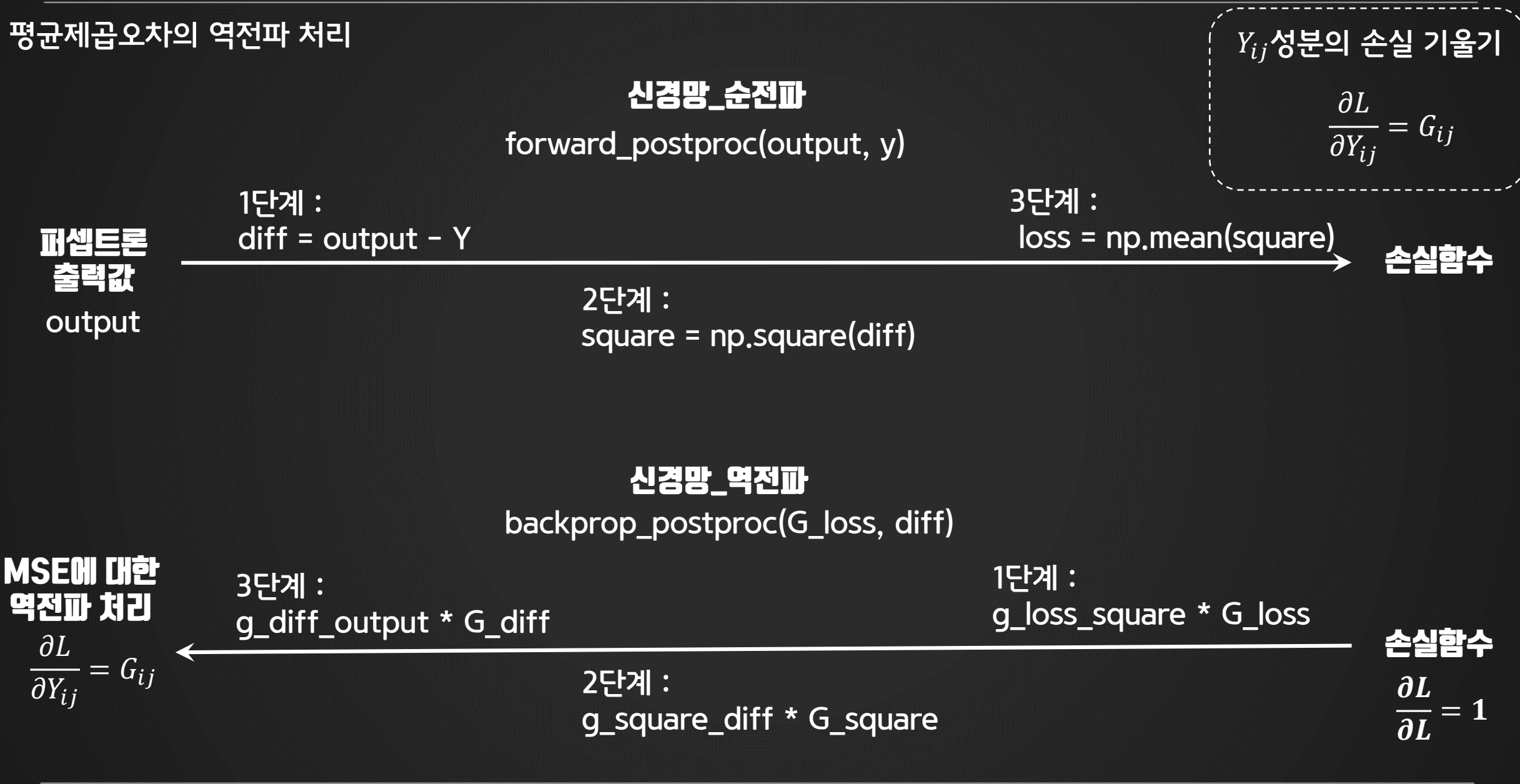
```
PYTHON:  
G_b = np.sum(G_output, axis=0)
```

## D.2 & E.2 후처리 과정에 대한 순전파 및 역전파 함수 정의

```
def forward_postproc(output, y):  
    diff = output - y  
    square = np.square(diff)  
    loss = np.mean(square)  
    return loss, diff  
  
def backprop_postproc(G_loss, diff):  
    shape = diff.shape  
  
    g_loss_square = np.ones(shape) / np.prod(shape)  
    g_square_diff = 2 * diff  
    g_diff_output = 1  
  
    G_square = g_loss_square * G_loss  
    G_diff = g_square_diff * G_square  
    G_output = g_diff_output * G_diff  
  
    return G_output
```

Part 1. 단층 퍼셉트론

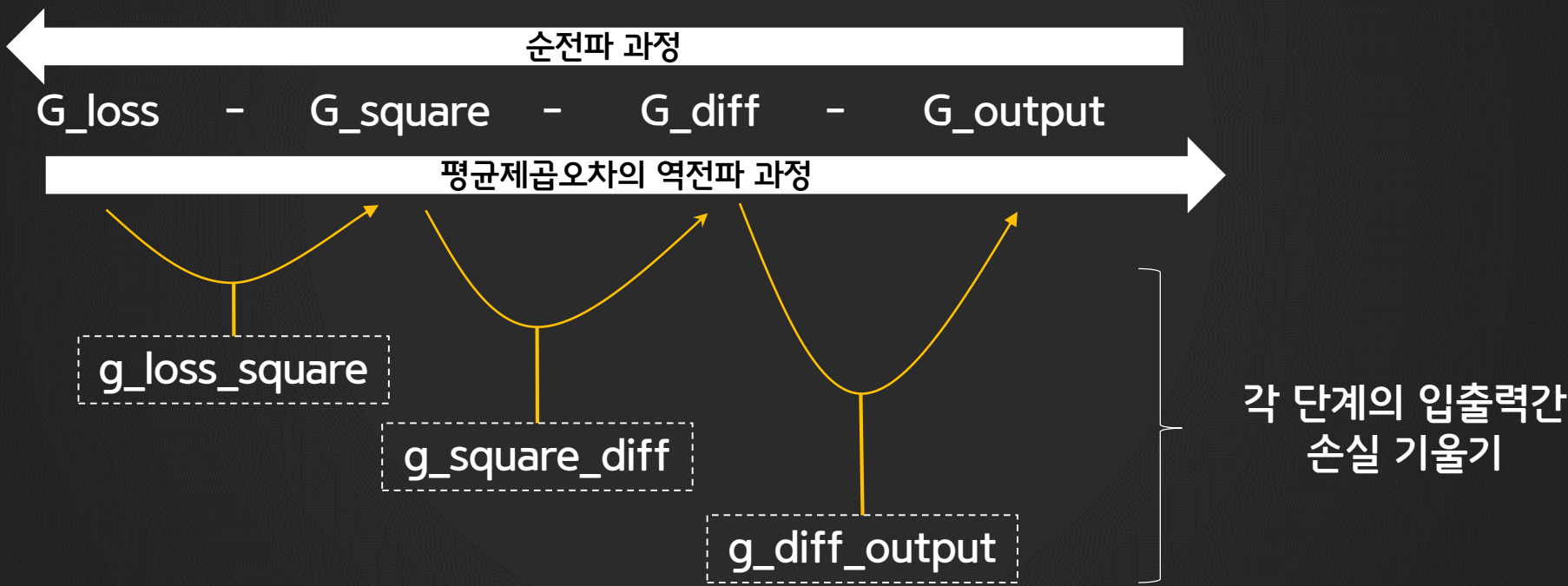
※ 평균제곱오차의 역전파 처리





※ 평균제곱오차의 역전파 처리

순전파의 역순으로 평균 / 제곱 / 오차에 대한 역전파 처리를 차례로 수행  
backprop\_postproc(G\_loss, diff)



※ 손실 기울기의 연쇄적 계산

## Part 1. 단층 퍼셉트론

### ※ 평균제곱오차의 역전파 처리

※ loss만 스칼라 값일 뿐 나머지는 모두 [미니배치 크기, 출력벡터 크기], 즉 [N,M]의 크기를 갖는다.

g\_loss\_square    평균연산  $L = \frac{\sum_{i=1}^M \sum_{j=1}^N square_{ij}}{MN} \quad \blacktriangleright \quad \frac{\partial L}{\partial square_{ij}} = \frac{1}{MN}$

python code : `g_loss_square = np.ones(shape) / np.prod(shape)`

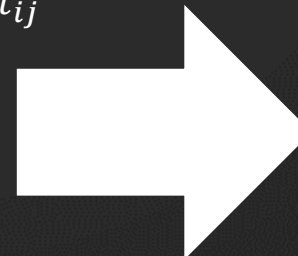
g\_square\_diff    제곱 연산  $square_{ij} = diff_{ij}^2 \quad \blacktriangleright \quad \frac{\partial square_{ij}}{\partial diff_{ij}} = 2diff_{ij}$

python code : `g_square_diff = 2 * diff`

g\_diff\_output    편차 연산  $diff_{ij} = output_{ij} - y_{ij} \quad \blacktriangleright \quad \frac{\partial diff_{ij}}{\partial output_{ij}} = 1$

python code : `g_diff_output = 1`

연쇄적 계산을 통한  
G\_output 도출



`G_square = g_loss_square * G_loss`  
`G_diff = g_square_diff * G_square`  
`G_output = g_diff_output * G_diff`

---

## D.3 정확도 계산 함수 정의

```
def eval_accuracy(output, y):  
    mdiff = np.mean(np.abs((output - y) / y))  
    return 1 - mdiff
```

*# 정답과 오차의 비율을 오류율로 설정.*

*# 1에서 오류율 평균을 뺀 값으로 정확도 정의*