

## PART 1

# 단층 퍼셉트론(SLP)

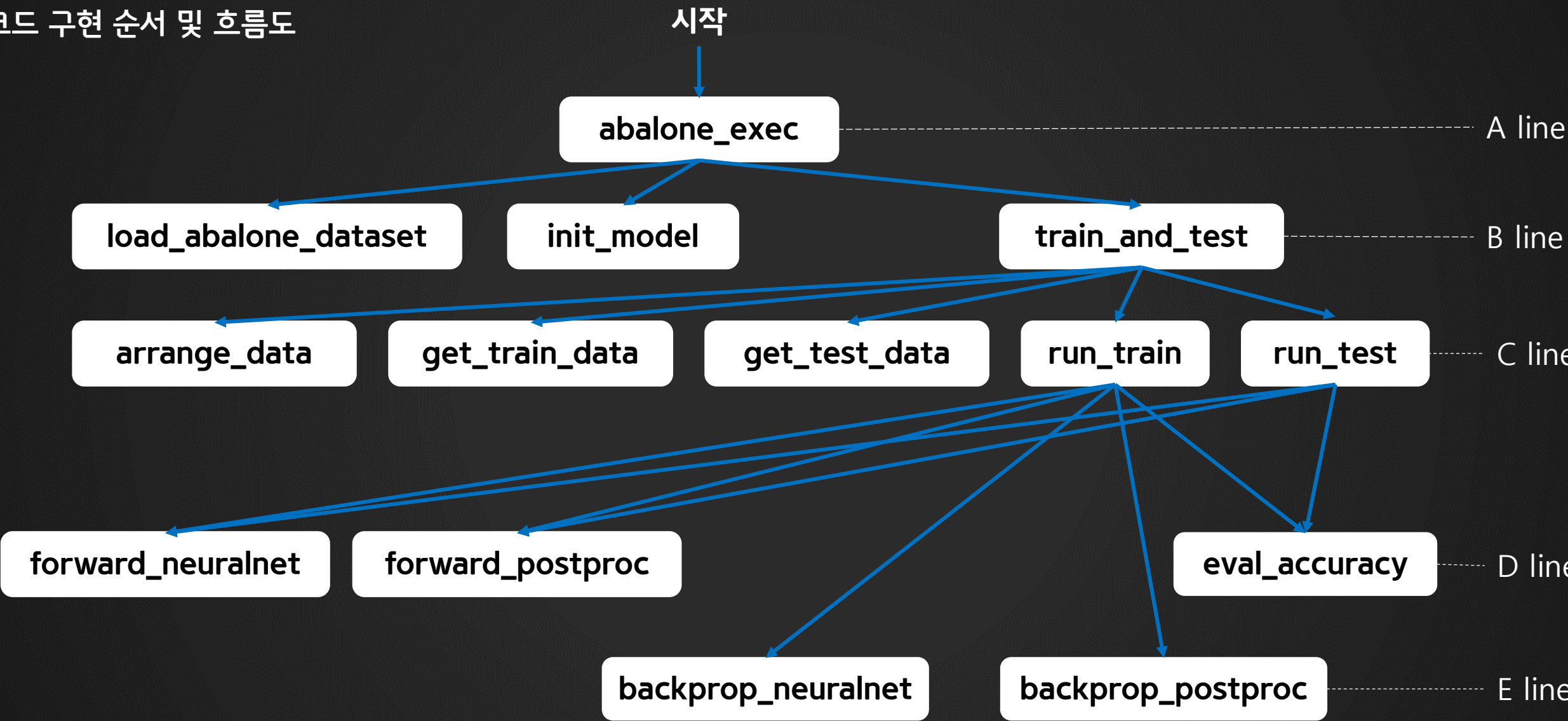
1장 회귀분석

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)
```

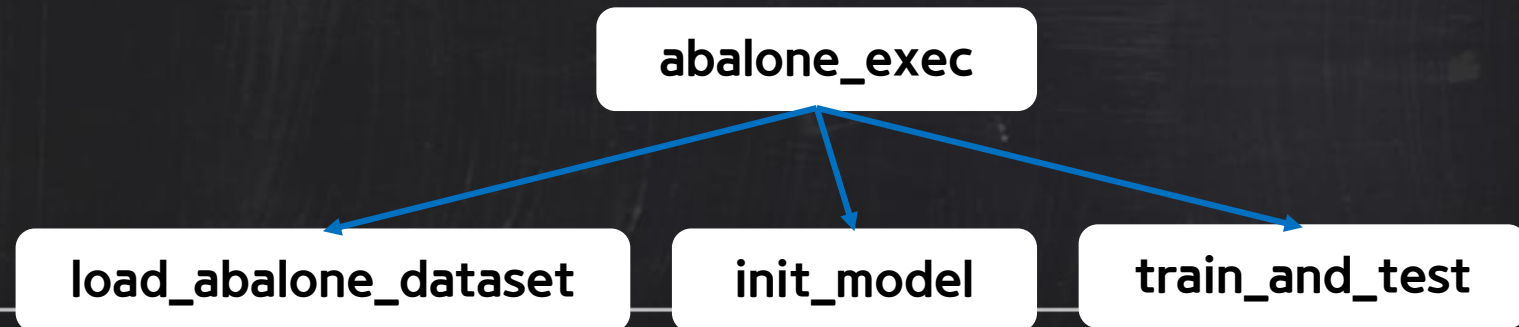
## 0.1 하이퍼 파라미터 정의

```
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)
```

A 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])
```

B 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:]
```

C 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



이후 데이터는  
그대로 복사



|     |    |    |    |     |        |       |     |   |
|-----|----|----|----|-----|--------|-------|-----|---|
| [   | 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. | ] |

## 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



## C.4 학습 및 평가 데이터 획득 함수 정의

```
def run_train(x, y):
```

```
    output, aux_nn = forward_neuralnet(x)
```

```
    loss, aux_pp = forward_postproc(output, y)
```

```
    accuracy = eval_accuracy(output, y)
```

```
    G_loss = 1.0    #  $\frac{\partial L}{\partial L} = 1.0$ 
```

```
    G_output = backprop_postproc(G_loss, aux_pp)
```

```
    backprop_neuralnet(G_output, aux_nn)
```

```
    return loss, accuracy
```

신경망 연산 부분

신경망 후처리 과정(손실함수 구하는 과정)

정확도를 구하는 과정

↑ 순전파 과정   ↓ 역전파 과정

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

순전파 과정 ->

역전파 과정

순전파의 뒷 단계 '후처리 과정'

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

순전파의 뒷 단계였던 후처리 과정에 대한  
역전파 함수 `backprop_postproc()`가 먼저 호출

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

`forward_postproc()`

`G_loss`

`backprop_postproc()`

`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

