Machine Learning Practice

10주차

-chap.4 실습 2주차

전자공학과

2022144007

김의진

Chap.4 – 2주차

1) Error Back-Propagation 알고리즘 구현

```
'''batch size 1 back propagation 구현 함수'''
def Back_Propagation_1(y_hat, y_data, x_matrix_added_dummy, b_matrix, w_prev, L): # w먼저 weight update시키므로 update 전 w 입력 받음

    # w 기울기 구하는 코드
    delta = 2 * (y_hat - y_data.reshape(-1, 1)) * y_hat * (1 - y_hat)          #delta 구함, y_data는 (:, 1)로 슬라이스 된 크기

    w_dif = np.dot(delta, b_matrix.T)                                          #delta와 b를 이용해 w의 기울기 구함

    # v 기울기 구하는 코드
    proc = np.dot(delta.T, w_prev)

    #dummy data 삭제
    b_matrix_h = np.delete(b_matrix, L, axis = 0)
    proc = np.delete(proc, L, axis = 1 )


    v_dif = np.dot((proc.T * b_matrix_h * (1 - b_matrix_h)), x_matrix_added_dummy.reshape(1, -1))     # v의 기울기 구하기

    return w_dif, v_dif                                                        # 함수의 반환값으로 w와 v의 기울기를 반환함


'''batch size N back propagation 구현 함수'''
def Back_Propagation_N(y_hat, y_data, x_input, b_matrix, w, L):

    # w 기울기 구하는 코드
    delta = 2 * (y_hat - y_data) * y_hat * (1 - y_hat)                         # delta 구함, y_data 그대로 들어감


    w_dif = np.dot(delta, b_matrix.T) / delta.shape[1]                         # w 기울기 구함, batch size가 N이라 평균을 나눠주면 안정적임

    proc = np.dot(delta.T, w)

    b_matrix_h = np.delete(b_matrix, L, axis = 0)
    proc = np.delete(proc, L, axis = 1)

    v_dif = np.dot((proc.T * b_matrix_h * (1 - b_matrix_h)), x_input.T) / delta.shape[1]    #v 기울기 구함
    return w_dif, v_dif
```

1. Batch size 1일 때와 N일때 구분

1) 1일 때:

- input으로 **한 데이터**에 대한 속성이 들어감
- Output class와 가까운 w update 후 v update함
- w는 update 되기 전 n – 1번 째w를 씀

2) N일 때:

- input으로 **모든 데이터**에 대한 속성이 들어감
- Output class와 가까운 w update 후 v update함
- 모든 데이터에 대한 weight update이기 때문에 바로 update 함

Chap.4 – 2주차

1) Two Layer Neural Network알고리즘 구현

Batch size 1일 때

```
def Two_Layer_Neural_Network_1(x_input, y_data, L, epoch, LR):

    MSE_list = []                                                   #MSE 저장할 list
    ACCURACY_list = []                                              #accuracy 저장할 list
    x_matrix = add_dummy(x_input)                                   #입력에 dummy data 추가

    M = ch_count(x_input)                                           #input 속성 수 체크
    Q = ch_count(y_data)                                            #ouput class 수 체크

    # weight 초기화
    v = np.random.rand(L, M + 1) * 2 - 1
    w = np.random.rand(Q, L + 1) * 2 - 1

    # epoch수 만큼 반복
    for i in range(epoch):

        y_hat_all_epoch = []                                        #한 epoch마다 y_hat 저장하는 list 초기화

        #데이터 길이만큼 반복
        for j in range(y_data.shape[1]):
            w_prev = w.copy()                                       #update전 weight값 저장

            y_hat, b_matrix = forward_propagation_1(x_matrix[:, j], v, w, L)    #forward propagation 진행
            y_hat_all_epoch.append(y_hat)                           #y_hat 값 list에 저장

            w_dif, v_dif = Back_Propagation_1(y_hat, y_data[:, j], x_matrix[:, j], b_matrix, w_prev, L)    #back propagation 진행

            #weight update
            w = w - LR * w_dif
            v = v - LR * v_dif

        y_hat_all = np.hstack(y_hat_all_epoch)                      #y_hat을 쌓은 list에 numpy array를 배열로 만들어줌
        error = y_hat_all - y_data                                  #error 계산
        MSE = np.mean(error ** 2)                                   #MSE 계산
        MSE_list.append(MSE)                                        #MSE list에 저장

        P = classification_data_max(y_hat_all)                      #데이터 당 최댓값을 1로 만들어주는 분류 함
        accuracy = data_accuracy(P, y_data)                         #accuracy 구하기
        ACCURACY_list.append(accuracy)                              #accuracy list에 저장

    return MSE_list, ACCURACY_list, v, w                            #MSE_list, ACCURACY_list, v, w 반환함
```

Back propagation 과정

1. Hyper parameters 설정
2. Weight 초기화
3. For문으로 0부터 N – 1번째 데이터 까지 순서대로 back propagation 알고리즘 적용

   -1. forward propagation으로 n번 째 y_hat 구함
   -2. back propagation으로 weight 기울기 구하고 ouput node에 가까운 것 부터 update함

4. Accuracy, MSE 계산
5. Epoch만큼 반복

Chap.4 – 2주차

1) Two Layer Neural Network알고리즘 구현

Batch size N일 때

```python
def Two_Layer_Neural_Network_N(x_input, y_data, L, epoch, LR):

    MSE_list = []                                                   #MSE 저장할 list
    ACCURACY_list = []                                              #accuracy 저장할 list

    x_matrix = add_dummy(x_input)                                   #dummy data 추가

    M = ch_count(x_input)                                           #input 속성 수 체크
    Q = ch_count(y_data)                                            #ouput class 수 체크

    #weight 초기화
    v = np.random.rand(L, M + 1) * 2 - 1
    w = np.random.rand(Q, L + 1) * 2 - 1

    #epoch수 만큼 반복
    for i in range(epoch):

        y_hat_all_epoch = []                                        #y_hat 저장할 list

        y_hat, b_matrix = forward_propagation_N(x_matrix, v, w, L)  #forward propagation 진행
        y_hat_all_epoch.append(y_hat)                               #y_hat list에 저장

        w_dif, v_dif = Back_Propagation_N(y_hat, y_data, x_matrix, b_matrix, w, L)    #back propagation에 진행

        #weight update
        w -= LR * w_dif
        v -= LR * v_dif

        y_hat_all = np.hstack(y_hat_all_epoch)                      #y_hat을 쌓은 list에 numpy array를 배열로 만들어줌

        error = y_hat_all - y_data                                  #error 계산
        MSE = np.mean(error ** 2)                                   #MSE 계산
        MSE_list.append(MSE)                                        #MSE list에 저장

        P = classification_data_max(y_hat_all)                      #데이터당 최대값 을 1로 만드는 분류함
        accuracy = data_accuracy(P, y_data)                         #accuracy 구하기
        ACCURACY_list.append(accuracy)                              #accuracy list에 저장

    return MSE_list, ACCURACY_list, v, w                            #MSE_list, ACCURACY_list, v, w 반환
```

Batch size 별 처리

- Batch Size 1: 샘플 하나 처리할 때마다 가중치 즉시 update

→ **Epoch당 N회 update**

- Batch Size N: 전체 샘플을 처리한 후 평균된 기울기로 한번 update

→ **Epoch당 1회 update**

Chap.4 – 2주차

2) Two-Layer Neural Network “Training”

```
def data_division(n_data, Tr_rate, V_rate, Te_rate):

    np.random.shuffle(n_data)                                   #데이터 섞기

    tr_index = int(len(n_data) * Tr_rate / 10)                  #Tr_set 비율만큼 데이터 index 양 확인
    v_index = int(len(n_data) * V_rate / 10)                    #V_set 비율만큼 데이터 index 양 확인
    te_index = int(len(n_data) * Te_rate / 10)                  #Te_set 비율만큼 데이터 index 양 확인

    #비율대로 data 나누기
    tr_set = n_data[0:tr_index]
    v_set = n_data[tr_index : tr_index + v_index]
    te_set = n_data[tr_index + v_index : tr_index + v_index + te_index]

    return tr_set, v_set, te_set
L = 4                                                           # 4로 했을 때 정확도 0.6쯔음
LR_1 = 0.05                                                     #batch size 1에 대한 learning rate
LR_N = 0.3                                                      #batch size N에 대한 learning rate
epoch_1 = 1000                                                  #batch size 1에 대한 epoch
epoch_N = 3000                                                  #batch size N에 대한 epoch

#training, validation, test set의 비율
tr_rate = 7
vl_rate = 0
te_rate = 3
```

“NN_data.csv”를 7:3 으로 Training set :Test set 분할

Training set을 이용한 학습에 대한 Hyper parameters

Hidden Layer node 수: 4

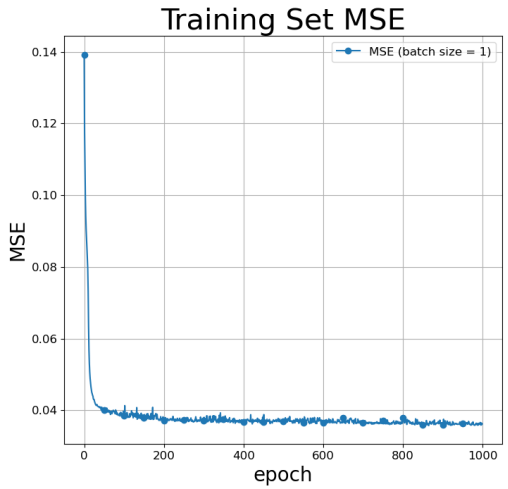
Batch size 1:
 Learning rate = 0.05
 Epoch = 1000

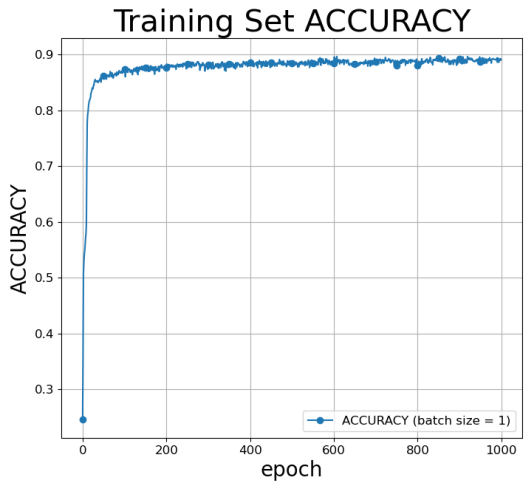
Batch size N:
 Learning rate = 0.3
 Epoch_ = 3000

Chap.4 – 2주차

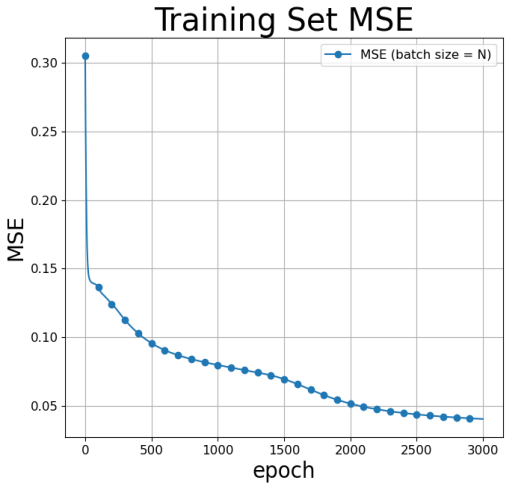
2) Two-Layer Neural Network “Training”

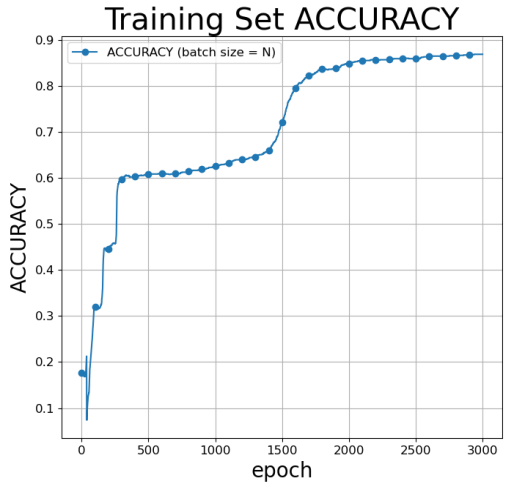
Batch size 1





Batch size N





MSE

-Batch size 1: 빠르게 줄고 그 이후로 수렴(진동이 관찰됨)

-Batch size N: 느리게 줄지만 안정적으로 수렴

Accuracy

-Batch size 1: 빠르게 0.9 근처 도달 후 수렴(진동이 관찰됨)

-Batch size N: 느리게 상승하며 학습 마지막 쯤 0.87에 수렴

Batch size 1 vs N

-Batch size 1: 수렴 속도 빠름, 진동이 많으며 이로 인해 안정성 낮아 보임, 적은 epoch으로도 빠르게 학습 가능
→ 진동이 많은 이유는 weight update를 자주 한 것이라고 생각함

-Batch size N: 수렴속도 느림, 진동이 거의 없어 부드러움, 이로 인해 안정성 좋아 보임, 1일 때보다 많은 epoch이 필요함
→ weight update 횟수가 1일 때보다 적은 것이 이유라고 생각함

Chap.4 – 2주차

3) Two-Layer Neural Network "Test"

```
def confusion_matrix(y_hat, y_data):

    y_pred_index = np.argmax(y_hat, axis = 0)                    #y_hat 데이터당 최댓값 index 가져옴
    y_true_index = np.argmax(y_data, axis = 0)                   #y_data 데이터당 최댓값 index 가져옴

    true_num = 0                                                 #정확히 예측한 횟수 초기화

    classes_num = ch_count(y_data)                               #y_data class 수 체크

    confusion_matrix = np.zeros((classes_num + 1, classes_num + 1))   #정확도 나타내기 위해 class수 + 1개만큼 정방 행렬 만듦

    #y 길이만큼반복
    for i in range(len(y_pred_index)):
        confusion_matrix[y_true_index[i], y_pred_index[i]] +=  1      #실제값, 예측값 index에 해당하는 자리에 1 더함

    # class 수만큼 반복
    for i in range(classes_num):

        # row방향으로 더한 값이 0보다 클 때 전체 데이터로 정확히 예측한 값 나눠줌
        if sum(confusion_matrix[i, : classes_num]) > 0:
            confusion_matrix[i, classes_num] = confusion_matrix[i, i] / np.sum(confusion_matrix[i, : classes_num])

        # column 방향으로 더한 값이 0보다 클 때 전체 데이터로 정확히 예측한 값 나눠줌
        if sum(confusion_matrix[: classes_num, i]) > 0:
            confusion_matrix[classes_num, i] = confusion_matrix[i, i] / np.sum(confusion_matrix[: classes_num, i])

        true_num += confusion_matrix[i, i]                       #정확히 예측한 값 세기
    confusion_matrix[classes_num, classes_num] = true_num / len(y_pred_index)   #전체 데이터에 대한 정확도 마지막 index에 저장

    return confusion_matrix                                      #confusion_matrix 반환
```

Training set으로 학습한 machine을 **test set으로 검증** (confusion matrix 이용)

오른쪽 마지막 줄: precision
→ 예측값중 실제 데이터의 비율

아래 마지막 줄: 재현율(recall)
→ 실제 데이터중 예측 성공한 값 비율

마지막 index: Test set의 accuracy

Chap.4 – 2주차

3) Two-Layer Neural Network "Test"

Predicted data

True data

Batch size 1

	0	1	2	3	4	5	6
0	92	1	0	0	0	1	0.978723
1	16	64	14	1	0	0	0.673684
2	0	6	84	1	0	0	0.923077
3	0	0	2	89	1	0	0.967391
4	0	0	0	1	74	1	0.973684
5	0	0	0	0	4	88	0.956522
6	0.851852	0.901408	0.84	0.967391	0.936709	0.977778	0.909259

Batch size N

	0	1	2	3	4	5	6
0	91	1	0	2	0	0	0.968085
1	21	34	39	1	0	0	0.357895
2	0	1	88	2	0	0	0.967033
3	0	0	0	91	1	0	0.98913
4	0	0	0	2	73	1	0.960526
5	0	0	0	1	5	86	0.934783
6	0.8125	0.944444	0.692913	0.919192	0.924051	0.988506	0.857407

Test set accuracy

이전 슬라이드의 Hyper parameters로 학습한 weight이용한 **test set에 대한 예측값**

오른쪽 마지막 행 → **precision**

아래 마지막 행 → **재현율(recall)**

대각성분 → 정확하게 예측한 data 개수

Batch size 1과 N 모두 대부분 좋은 분류를 한 것으로 보임

그러나 유독 class 1에 대하여 class 0과 class 2로 오분류 함

Chap.4 – 2주차

3) Two-Layer Neural Network “Test”

Predicted data

True data

Batch size 1

	0	1	2	3	4	5	6
0	92	1	0	0	0	1	0.978723
1	16	64	14	1	0	0	0.673684
2	0	6	84	1	0	0	0.923077
3	0	0	2	89	1	0	0.967391
4	0	0	0	1	74	1	0.973684
5	0	0	0	0	4	88	0.956522
6	0.851852	0.901408	0.84	0.967391	0.936709	0.977778	0.909259

Batch size N

	0	1	2	3	4	5	6
0	91	1	0	2	0	0	0.968085
1	21	34	39	1	0	0	0.357895
2	0	1	88	2	0	0	0.967033
3	0	0	0	91	1	0	0.98913
4	0	0	0	2	73	1	0.960526
5	0	0	0	1	5	86	0.934783
6	0.8125	0.944444	0.692913	0.919192	0.924051	0.988506	0.857407

Class 1을 class 0과 class 2로 오분류한 이유

1. Class 1과 class 2의 입력 특징이 유사한 분포를 지님
2. Test data를 suffle하고 분배하는 과정에서 이 class에 대한 data가 불균형함

Chap.4 – 2주차

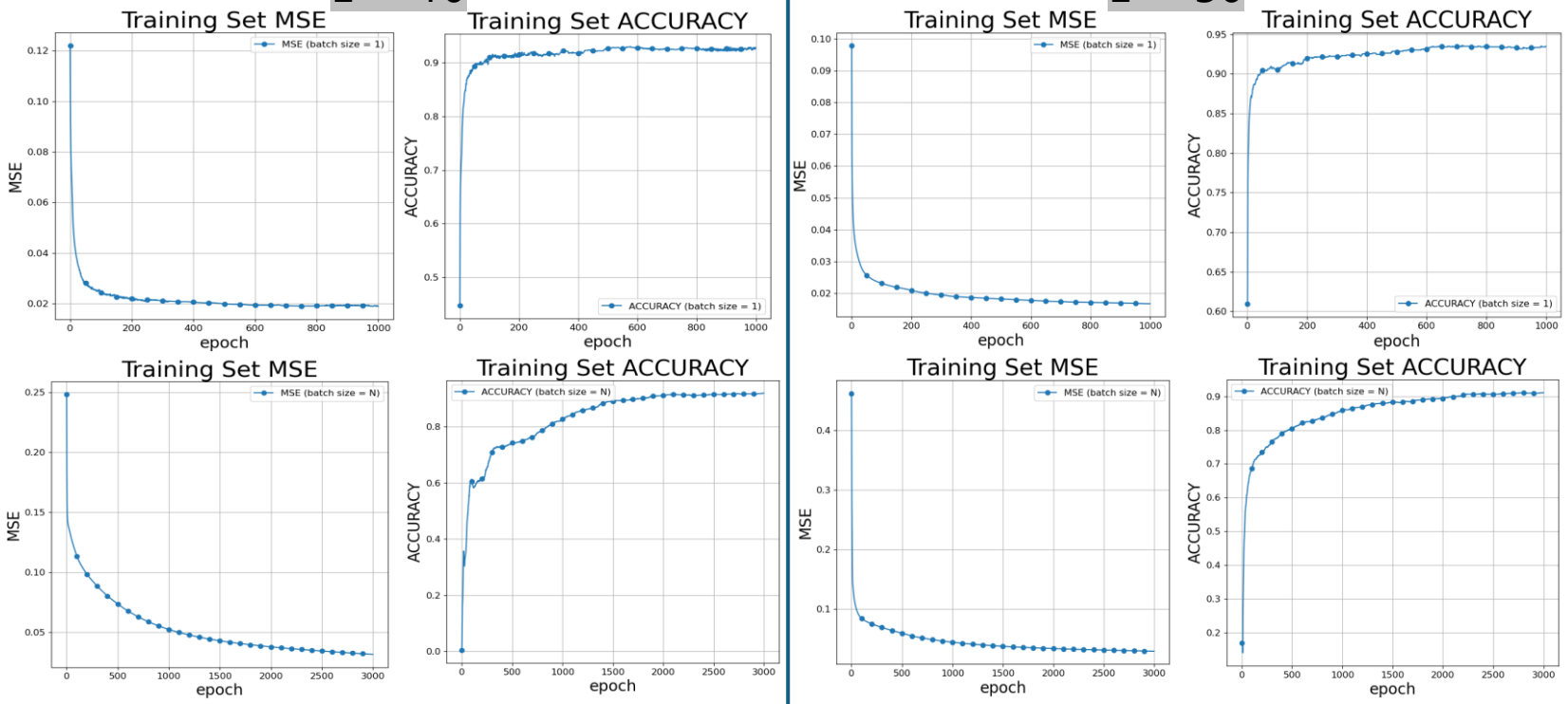
3) Hyper – parameter tuning(L = 10, 30)

L = 10

L = 30

Batch size 1

Batch size N



Hidden Layer의 node 수가 늘어남에 따른 **변화**

공통:
MSE는 점점 작아지고 Accuracy는 점점 증가함

Batch size 1:
MSE와 accuracy그래프의 진동이 점점 작아짐

Batch size N:
수렴 속도가 점점 빨라짐

Chap.4 – 2주차

3) Hyper – parameter tuning(L = 10, 30)

L = 10

L = 30

Predicted data

True data

Batch size 1

	0	1	2	3	4	5	6
0	87	4	0	0	0	2	0.935484
1	12	67	6	0	0	0	0.788235
2	0	8	70	1	0	0	0.886076
3	0	0	2	77	5	1	0.905882
4	0	0	0	5	85	2	0.923913
5	0	0	0	0	3	103	0.971698
6	0.878788	0.848101	0.897436	0.927711	0.913978	0.953704	0.905556

	0	1	2	3	4	5	6
0	78	7	0	0	0	0	0.917647
1	3	85	11	0	0	0	0.858586
2	0	7	79	0	1	0	0.908046
3	0	0	5	77	3	0	0.905882
4	0	0	0	1	90	8	0.909091
5	0	0	0	0	2	83	0.976471
6	0.962963	0.858586	0.831579	0.987179	0.9375	0.912088	0.911111

Batch size N

	0	1	2	3	4	5	6
0	85	5	0	0	0	3	0.913978
1	8	73	4	0	0	0	0.858824
2	0	5	72	2	0	0	0.911392
3	0	0	1	77	7	0	0.905882
4	0	0	0	4	87	1	0.945652
5	0	0	0	1	3	102	0.962264
6	0.913978	0.879518	0.935065	0.916667	0.896907	0.962264	0.918519

	0	1	2	3	4	5	6
0	80	5	0	0	0	0	0.941176
1	3	78	17	1	0	0	0.787879
2	0	9	78	0	0	0	0.896552
3	0	0	4	80	1	0	0.941176
4	0	0	0	1	96	2	0.969697
5	1	0	0	1	4	79	0.929412
6	0.952381	0.847826	0.787879	0.963855	0.950495	0.975309	0.909259

1. Hidden Layer의 node수가 4일 때보다 좋은 accuracy 보임
2. Node 수가 늘어남에 따라(8 이상일 때 부터) Test set에 대한 precision, recall, accuracy의 큰 차이는 발견되지 않음

Chap.4 – 2주차
3) 결과

Hidden Layer의 node 수를 늘림에 따른 변화

1) 예상:

예측이 점점 정교해 지다가 overfitting이 일어나 test set에 대한 confusion matrix의 정확도가 내려갈 것임

2) 실제 값:

Node 수가 8 이상일 때부터 training set 정확도가 0.85 ~ 0.92정도 였으며 test set에 대한 confusion matrix의 precision, recall, accuracy가 0.8~0.9에서 나오고 큰 차이 없음

Overfitting이 일어나지 않은 유력 이유: 이 데이터에 대해 모델이 적절함

→ 불필요하게 큰 모델보다는 효율적인 구조의 모델(8 ~ 12)이 바람직함

Slide 6에 Batch size1 vs N 에서 본 특성을 보아 이 둘을 적절히 섞으면 둘의 장점을 모을 수 있음