```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


parameters = {"axes.labelsize": 20, "axes.titlesize": 30, 'xtick.labelsize': 12, "ytick.labelsize": 12, "legend.fontsize": 12}

plt.rcParams.update(parameters)


d_M = pd.read_csv('C:₩₩Users₩₩kim07₩₩Downloads₩₩NN_data.csv')
# 데이터파일 불러오기

d_M = d_M.to_numpy(dtype=float)


'''training, validation, test set data 나누는 함수 '''

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

'''받아온 파일의 데이터 x와 y로 자동 분류 해주는 함수'''
def make_input_output(M):

    for i in range(M.shape[1]):                                    #M의 column 수만큼 반복
        if i == 0:
            x_matrix = M[:, 0]                                     #M의 첫번째 column 성분 값들 x_matrix에 저장
        elif i < (M.shape[1] - 1):
            x_matrix = np.column_stack([x_matrix, M[:, i]])        #M의 마지막 column 성분 제외한 값들 x_matrix에 저장
        else:
            y = M[:, i]                                            #M의 마지막 column 성분 y에 저장
    y = y.reshape(y.shape[0], 1)
#y size 다듬기
    x_matrix_t = np.transpose(x_matrix)                            #한 데이터에 대한 특징들 한 column에 나타나기 위해 transpose
    y_t = np.transpose(y)                                          #위와 동일

    return x_matrix_t, y_t
```

```
'''데이터의 class수 세는 함수'''
def y_class(y):

    y_class = np.unique(y)
#class 수 계산
    Q = len(y_class)
#numpy array로 받아지기 때문에 길이를 셈

    return Q

'''데이터의 특징 수 세는 함수'''
def ch_count(y):

    Q = len(y)                                                    #
받아온 데이터의 열 개수를 셈

    return Q

'''One-Hot Encoding 구현 함수'''
def One_Hot_Encoding(y):

    Q = y_class(y)
#class 수를 셈
    print("class 수:", Q)

    y_vector = np.zeros([Q, y.shape[1]])                          #Q × 샘
플 수의 zero 벡터 생성
```

```
    # i번 index의 y데이터가 1~6 각각에 해당될 때 그 자리에 1 저장
    for i in range(y.shape[1]):
        if y[0, i] == 1:
            y_vector[0, i] = 1
        elif y[0, i] == 2:
            y_vector[1, i] = 1
        elif y[0, i] == 3:
            y_vector[2, i] = 1
        elif y[0, i] == 4:
            y_vector[3, i] = 1
        elif y[0, i] == 5:
            y_vector[4, i] = 1
        else:
            y_vector[5, i] = 1

    return y_vector

'''row기반 dummy추가해주는 함수'''
def add_dummy(x):

    x_dummy = np.ones(x.shape[1])                                    #
입력데이터 x의 길이만큼 dummy 생성
    x = np.row_stack([x, x_dummy])
#row방향으로 쌓음
```

```
    return x


'''sigmoid 구현 함수'''
def sigmoid_function(z):

    return(1/(1 + np.exp(-z)))



'''대표값 찾아서 1로 만들어주는 함수'''
def classification_data_max(y):

    p = np.zeros_like(y)                                                    #받
아온 y데이터의 row와 column 크기만큼 요소가 0인 matrix 생성

    y_max = np.argmax(y, axis = 0)                                          #y
의 최댓값 index 저장

    #y 데이터 길이만큼 p (i번째 최댓값 index, i)에 1 저장
    for i in range(y.shape[1]):
        p[y_max[i], i] = 1

    return p



'''데이터 정확도 함수'''
def data_accuracy(y_h, y):                                                  #한
```

데이터에 대한 성분 row로 나열한 데이터기준

```
    count = 0
#count 기능 이용할 변수 0으로 초기화

    for i in range(y_h.shape[1]):                                    #예측 데
이터 column 성분만큼 반복
        if (y_h[:, i] == y[:, i]).all():                             #받아온 데
이터와 예측 데이터의 같은 column의 row성분 값이 모두 같은지 확인
            count += 1                                               #
위 조건에 해당할 때 count

    accuracy = count / y_h.shape[1]
#count 된 수를 예측데이터 column 개수만큼 나눠줌

    return accuracy

'''batch size 1 forward_propagation 구현 함수'''
def forward_propagation_1(x_input_added_dummy, v_matrix, w_matrix, L):
#Hidden Layer의 node 수 지정

    alpha = np.dot(v_matrix, x_input_added_dummy)                    #v
와 xinput을 곱해 alpha를 구함
    b_matrix = sigmoid_function(alpha).reshape(-1, 1)                #
batch size 1일 때 sigmoid에 넣으면 형태 깨져서 reshape이용

    b_matrix = add_dummy(b_matrix)
#b에 dummy 추가
```

```
    beta = np.dot(w_matrix, b_matrix)                                    #w
와 b 곱해서 beta 구함

    y_hat = sigmoid_function(beta)
#beta를 sigmoid function에 넣어 y_hat 구함


    return y_hat, b_matrix


'''batch size N forward_propagation 구현 함수'''

def forward_propagation_N(x_input_added_dummy, v_matrix, w_matrix, L):


    alpha = np.dot(v_matrix, x_input_added_dummy)
#alpha 구함


    b_matrix = sigmoid_function(alpha)
#b_matrix 구함


    b_matrix = add_dummy(b_matrix)
#dummy 추가


    beta = np.dot(w_matrix, b_matrix)
#beta 구함

    y_hat = sigmoid_function(beta)
#y_hat 구함


    return y_hat, b_matrix
```

'''batch size 1 back propagation 구현 함수'''

def Back_Propagation_1(y_hat, y_data, x_matrix_added_dummy, b_matrix, w_prev, L): # w 먼저 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

'''batch size 1인 Two_Layer_Neural Network'''
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)                                    #데이
터 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


'''batch size N인 Two_Layer_Neural Network'''
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)                                    #데이
터당 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
```

```
'''confusion matrix 구현 함수'''

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))                #정확
도 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                    #실제
값, 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 나
눠줌
        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]                                    #정
확히 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


'''main'''


L = 10
#Hidden Layer node 수
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

tr_d, vl_d, te_d = data_division(d_M, tr_rate, vl_rate, te_rate) #학습 데이터 셔플 후 데이터 분할함

tr_x_matrix, tr_y_vector = make_input_output(tr_d)
#training 데이터를 input과 output으로 나눔

tr_y_data = One_Hot_Encoding(tr_y_vector)
#training output data를 one-hot-encoding 방식으로 변환

te_x_matrix, te_y_vector = make_input_output(te_d) #test 데이터를 input과 output으로 나눔

te_y_data = One_Hot_Encoding((te_y_vector))
#test output data를 one-hot-encoding 방식으로 변환

#batch size가 1인 학습

MSE_tr_1, ACCURACY_tr_1, v_tr_1, w_tr_1 = Two_Layer_Neural_Network_1(tr_x_matrix, tr_y_data, L, epoch_1, LR_1)

#batch size가 N인 학습

MSE_tr_N, ACCURACY_tr_N, v_tr_N, w_tr_N = Two_Layer_Neural_Network_N(tr_x_matrix, tr_y_data, L, epoch_N, LR_N)

```
#batch size 1로 학습한 w, v에 대한 test set 검증

y_hat_te_all = []                                                    #test
set의 y_hat 저장할 list


te_x_matrix = add_dummy(te_x_matrix)
#dummy 추가


#batch size 1이므로 데이터 하나씩 forward propagation

for i in range(te_x_matrix.shape[1]):

    # x_added_dummy = add_dummy(te_x_matrix[:, j].reshape(-1, 1))
#dummy 추가

    y_hat_te, _ = forward_propagation_1(te_x_matrix[:, i], v_tr_1, w_tr_1, L)        #forward
propagation 진행

    y_hat_te_all.append(y_hat_te)
#y_hat_te list에 저장


y_hat_te_1 = np.hstack(y_hat_te_all)                                 #list에
저장된 numpy array 배열로 저장


#batch size N 으로 학습한 w, v에 대한 test set 검증

y_hat_te_N, _ = forward_propagation_N(te_x_matrix, v_tr_N, w_tr_N, L)


#confusion matrix

CONFUSION_MATRIX_te_1 = confusion_matrix(y_hat_te_1, te_y_data)

CONFUSION_MATRIX_te_N = confusion_matrix(y_hat_te_N, te_y_data)
```

```
#그래프
plt.figure()
plt.plot(MSE_tr_1, '-o', markevery = 50, label = 'MSE (batch size = 1)')
plt.xlabel("epoch")
plt.ylabel("MSE")
plt.title("Training Set MSE")
plt.legend()
plt.grid()


plt.figure()
plt.plot(ACCURACY_tr_1, '-o', markevery = 50, label = 'ACCURACY (batch size = 1)')
plt.xlabel("epoch")
plt.ylabel("ACCURACY")
plt.title("Training Set ACCURACY")
plt.legend()
plt.grid()


plt.figure()
plt.plot(MSE_tr_N, '-o', markevery = 100, label = 'MSE (batch size = N)')
plt.xlabel("epoch")
plt.ylabel("MSE")
plt.title("Training Set MSE")
plt.legend()
```

```
plt.grid()



plt.figure()
plt.plot(ACCURACY_tr_N, '-o', markevery = 100, label = 'ACCURACY (batch size = N)')
plt.xlabel("epoch")
plt.ylabel("ACCURACY")
plt.title("Training Set ACCURACY")
plt.legend()
plt.grid()
```