```
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)


'''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]                                     #M
의 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]                                            #M
의 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)                                          #위
와 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)                    # 예: Q=3이면
    y_vector = np.zeros((Q, y.shape[1]))
    for i in range(y.shape[1]):
        label = int(y[0, i])        # y는 정수로 가정
        y_vector[label, 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) #v와 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) #w와 b 곱해서 beta 구함

y_hat = sigmoid_function(beta)
#beta를 sigmoid function에 넣어 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                                                    # 함
수의 반환값으로 w와 v의 기울기를 반환함


'''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

'''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


'''가로축 Expectation'''
def feature_1 (input_data):
        row_sum = np.sum(input_data, axis = 1)
        PSD = row_sum / np.sum(row_sum)
        E = np.sum(PSD * np.arange(len(row_sum)))


        return E


'''세로축 Variance'''
def feature_2 (input_data):
        column_sum = np.sum(input_data, axis = 0)
        PSD = column_sum / np.sum(column_sum)
        E = np.sum(PSD * np.arange(len(column_sum)))
        var = np.sum(PSD * (np.arange(len(column_sum)) - E) ** 2)
```

```
    return var


'''대각선 Expectation'''
def feature_3 (input_data):
    diag_components = np.diagonal(input_data)
    PSD = diag_components / np.sum(diag_components)
    E = np.sum(PSD * np.arange(len(diag_components)))


    return E


'''중앙의 0 개수'''
def feature_4 (input_data):
    center = input_data[10:18, 10:18]
    zero_cnt = np.sum(center == 0)


    return zero_cnt


'''좌상단 0 개수'''
def feature_5 (input_data):
    center = input_data[1:10, 1:10]
    zero_cnt = np.sum(center == 0)


    return zero_cnt
```

```
'''대각성분 Variance'''
def feature_6 (input_data):
    diag_components = np.diagonal(input_data)
    PSD = diag_components / np.sum(diag_components)
    E = np.sum(PSD * np.arange(len(diag_components)))
    var = np.sum(PSD * (np.arange(len(diag_components)) - E) ** 2)

    return var

'''anti diogonal Variance'''
def feature_7 (input_data):
    Flip = np.fliplr(input_data)
    anti_diag_components = np.diagonal(Flip)
    PSD = anti_diag_components / np.sum(anti_diag_components)
    E = np.sum(PSD * np.arange(len(anti_diag_components)))
    var = np.sum(PSD * (np.arange(len(anti_diag_components)) - E) ** 2)

    return var

'''우 하단 0 개수'''
def feature_8 (input_data):
    center = input_data[15:21, 15:21]
    zero_cnt = np.sum(center == 0)

    return zero_cnt
```

```
'''top bottom rate'''
def feature_9 (input_data):
    top = np.sum(input_data[: 14, :])
    bottom = np.sum(input_data[14:, :])
    TBR = top / bottom

    return TBR

'''정규화 구현'''
def standard_data (input_data):
    feature_data = input_data[:, :-1]  # 마지막 열(label)을 제외한 feature만 추출

    mean = np.mean(feature_data, axis=0)
    std = np.std(feature_data, axis=0)

    # 표준편차가 0인 경우(=변화 없는 feature)는 나누지 않도록 처리
    std[std == 0] = 1

    input_data[:, :-1] = (feature_data - mean) / std

    return input_data

L = 16
epoch = 1000
```

```
LR = 0.01

tr_set = 7

va_set = 0

te_set = 3


'''특징 미추출 시'''

class_0 = np.array([], dtype = 'float32')                                    # 0data
array 생성

class_1 = np.array([], dtype = 'float32')                                    # 1data
array 생성

class_2 = np.array([], dtype = 'float32')                                    # 2data
array 생성

data_set = np.array([], dtype = 'float32')                                   # 모든
data 합쳐줄  array 생성


y_0 = np.zeros([500, 1])                                                     # 불러
온 0 데이터 라벨링할 0 list

y_1 = np.ones([500, 1])                                                      # 불러
온 1 데이터 라벨링할 1 list

y_2 = np.array([2] * 500)                                                    # 불러
온 2 데이터 라벨링할 2 list


'''mnist data 불러오는 for문'''

for i in range(1, 501):


        #주소 변수에 저장

    temp_name_0 = 'C:₩₩Users₩₩kim07₩₩Desktop₩₩Machinlearning_Workplace₩₩[배
포용] MNIST Data₩₩0_' + str(i) + '.csv'
```

```
    temp_name_1 = 'C:₩₩Users₩₩kim07₩₩Desktop₩₩Machinlearning_Workplace₩₩[배포용] MNIST Data₩₩1_' + str(i) + '.csv'
    temp_name_2 = 'C:₩₩Users₩₩kim07₩₩Desktop₩₩Machinlearning_Workplace₩₩[배포용] MNIST Data₩₩2_' + str(i) + '.csv'

      #해당 주소 파일 dataframe 저장후 floating
    temp_image_0 = pd.read_csv(temp_name_0, header = None)
    temp_image_0 =   temp_image_0.to_numpy(dtype = 'float32')

    temp_image_1 = pd.read_csv(temp_name_1, header = None)
    temp_image_1 =   temp_image_1.to_numpy(dtype = 'float32')

    temp_image_2 = pd.read_csv(temp_name_2, header = None)
    temp_image_2 =   temp_image_2.to_numpy(dtype = 'float32')

      #각 데이터 array에 저장(flatten 이용해 모든 pixel 값 저장 28 x 28)
    class_0 = np.append(class_0, temp_image_0.flatten())
    class_1 = np.append(class_1, temp_image_1.flatten())
    class_2 = np.append(class_2, temp_image_2.flatten())

      #matrix로 만들어주고 data 라벨링 시켜줌
class_0 = np.resize(class_0, [784, 500])
class_0 = np.column_stack([class_0.T, y_0])
class_1 = np.resize(class_1, [784, 500])
class_1 = np.column_stack([class_1.T, y_1])
class_2 = np.resize(class_2, [784, 500])
```

```
class_2 = np.column_stack([class_2.T, y_2])


        #data set에 저장
data_set = np.row_stack([class_0, class_1, class_2])



   #traing, test set 나누기
tr_data, va_data, te_data = data_division(data_set, tr_set, va_set, te_set)


   #input output 나누고 output onehot encoding 함
tr_x, tr_y = make_input_output(tr_data)
tr_y_data = One_Hot_Encoding(tr_y)
te_x, te_y = make_input_output(te_data)
te_y_data = One_Hot_Encoding(te_y)


   #학습(training set으로)
MSE_tr, ACCURACY_tr, v_tr, w_tr = Two_Layer_Neural_Network_1(tr_x, tr_y_data, L, epoch,
LR)


y_hat_te_all = []                                                    #test
set의 y_hat 저장할 list


   #training set으로 구한 weights를 이용해서 test set 검증
te_x = add_dummy(te_x)
#dummy 추가
```

```
#batch size 1이므로 데이터 하나씩 forward propagation
for i in range(te_x.shape[1]):
    y_hat_te, _ = forward_propagation_1(te_x[:, i], v_tr, w_tr, L)          #forward propagation 진행
    y_hat_te_all.append(y_hat_te)
#y_hat_te list에 저장

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

confusion_matrix_te = confusion_matrix(y_hat_te, te_y_data)                 #test set에 대한 confusion matrix

plt.figure()
plt.plot(MSE_tr, '-o', markevery = 100, label = 'MSE')
plt.xlabel("epoch")
plt.ylabel("MSE")
plt.title("Training Set MSE")
plt.legend()
plt.grid()

plt.figure()
plt.plot(ACCURACY_tr, '-o', markevery = 100, label = 'Accuracy')
plt.xlabel("epoch")
plt.ylabel("Accuracy")
plt.title("Training Set Accuracy")
```

```
plt.legend()

plt.grid()



'''특징 추출 시'''

    #Hyper parameters

L = 16

epoch = 1000

LR = 0.01

tr_set = 7

va_set = 0

te_set = 3


# 각 class에 대한 data set의 배열을 만들고 크기를 다음음

x_0_set_f = np.array([], dtype = 'float32')

x_0_set_f = np.resize(x_0_set_f, (0, 9))

x_1_set_f = np.array([], dtype = 'float32')

x_1_set_f = np.resize(x_1_set_f, (0, 9))

x_2_set_f = np.array([], dtype = 'float32')

x_2_set_f = np.resize(x_2_set_f, (0, 9))


#데이터 파일 받고 특징추출하는 for 문

for i in range(1, 501):

    temp_name_f0 =
'C:₩₩Users₩₩kim07₩₩Desktop₩₩Machinlearning_Workplace₩₩[배포용] MNIST
Data₩₩0_' + str(i) + '.csv'
```

```
    temp_name_f1 =
'C:₩₩Users₩₩kim07₩₩Desktop₩₩Machinlearning_Workplace₩₩[배포용] MNIST
Data₩₩1_' + str(i) + '.csv'

    temp_name_f2 =
'C:₩₩Users₩₩kim07₩₩Desktop₩₩Machinlearning_Workplace₩₩[배포용] MNIST
Data₩₩2_' + str(i) + '.csv'


    temp_image_f0 = pd.read_csv(temp_name_f0, header = None)

    temp_image_f1 = pd.read_csv(temp_name_f1, header = None)

    temp_image_f2 = pd.read_csv(temp_name_f2, header = None)


    temp_image_f0 = temp_image_f0.to_numpy(dtype = 'float32')

    temp_image_f1 = temp_image_f1.to_numpy(dtype = 'float32')

    temp_image_f2 = temp_image_f2.to_numpy(dtype = 'float32')


    # 0 data set에 대한 특징 추출

    x0_f0 = feature_1(temp_image_f0)

    x1_f0 = feature_2(temp_image_f0)

    x2_f0 = feature_3(temp_image_f0)

    x3_f0 = feature_4(temp_image_f0)

    x4_f0 = feature_5(temp_image_f0)

    x5_f0 = feature_6(temp_image_f0)

    x6_f0 = feature_7(temp_image_f0)

    x7_f0 = feature_8(temp_image_f0)

    x8_f0 = feature_9(temp_image_f0)
```

```
# 1 data set에 대한 특징 추출
x0_f1 = feature_1(temp_image_f1)
x1_f1 = feature_2(temp_image_f1)
x2_f1 = feature_3(temp_image_f1)
x3_f1 = feature_4(temp_image_f1)
x4_f1 = feature_5(temp_image_f1)
x5_f1 = feature_6(temp_image_f1)
x6_f1 = feature_7(temp_image_f1)
x7_f1 = feature_8(temp_image_f1)
x8_f1 = feature_9(temp_image_f1)

# 2 data set에 대한 특징 추출
x0_f2 = feature_1(temp_image_f2)
x1_f2 = feature_2(temp_image_f2)
x2_f2 = feature_3(temp_image_f2)
x3_f2 = feature_4(temp_image_f2)
x4_f2 = feature_5(temp_image_f2)
x5_f2 = feature_6(temp_image_f2)
x6_f2 = feature_7(temp_image_f2)
x7_f2 = feature_8(temp_image_f2)
x8_f2 = feature_9(temp_image_f2)

#특징 합치기
x_feature_f0 = np.array([x0_f0, x1_f0, x2_f0, x3_f0, x4_f0, x5_f0, x6_f0, x7_f0, x8_f0])
```

```
        x_feature_f0 = np.resize(x_feature_f0, (1, 9))
        x_0_set_f = np.concatenate((x_0_set_f, x_feature_f0), axis = 0)

        x_feature_f1 = np.array([x0_f1, x1_f1, x2_f1, x3_f1, x4_f1, x5_f1, x6_f1, x7_f1, x8_f1])
        x_feature_f1 = np.resize(x_feature_f1, (1, 9))
        x_1_set_f = np.concatenate((x_1_set_f, x_feature_f1), axis = 0)

        x_feature_f2 = np.array([x0_f2, x1_f2, x2_f2, x3_f2, x4_f2, x5_f2, x6_f2, x7_f2, x8_f2])
        x_feature_f2 = np.resize(x_feature_f2, (1, 9))
        x_2_set_f = np.concatenate((x_2_set_f, x_feature_f2), axis = 0)

#각 데이터 라벨링할 데이터 만들기
Y_0 = np.zeros([x_0_set_f.shape[0], 1])
Y_1 = np.ones([x_1_set_f.shape[0], 1])
Y_2 = np.array([2] * x_2_set_f.shape[0])

#각 데이터 셋 만들기
data0 = np.column_stack([x_0_set_f, Y_0])
data1 = np.column_stack([x_1_set_f, Y_1])
data2 = np.column_stack([x_2_set_f, Y_2])

#모든 class 데이터 합치기
data_f = np.row_stack([data0, data1, data2])

data_f = standard_data(data_f)                                          #특
```

징 추출한 데이터 셋 정규화

```
#training, test data set 나누기
tr_data_f, va_data_f, te_data_f = data_division(data_f, tr_set, va_set, te_set)

#training, test set input, output 나누고 output onehot encoding
tr_x_f, tr_y_f = make_input_output(tr_data_f)
tr_y_data_f = One_Hot_Encoding(tr_y_f)

te_x_f, te_y_f = make_input_output(te_data_f)
te_y_data_f = One_Hot_Encoding(te_y_f)

#training set으로 학습
MSE_tr_f, ACCURACY_tr_f, v_tr_f, w_tr_f = Two_Layer_Neural_Network_1(tr_x_f, tr_y_data_f,
L, epoch, LR)

y_hat_te_all_f = []                                                    #test
set의 y_hat 저장할 list

te_x_f = add_dummy(te_x_f)
#dummy 추가

# training set으로 학습한 weight로 test set forwoard propagation
#batch size 1이므로 데이터 하나씩 forward propagation
for i in range(te_x_f.shape[1]):
    y_hat_te_f, _ = forward_propagation_1(te_x_f[:, i], v_tr_f, w_tr_f, L)        #forward
```

```
propagation 진행

    y_hat_te_all_f.append(y_hat_te_f)
#y_hat_te list에 저장


y_hat_te_f = np.hstack(y_hat_te_all_f)



#test set에 대한 confusion matrix

confusion_matrix_te_f = confusion_matrix(y_hat_te_f, te_y_data_f)


plt.figure()

plt.plot(MSE_tr_f, '-o', markevery = 50, label = 'MSE')

plt.xlabel("epoch")

plt.ylabel("MSE")

plt.title("Training Set MSE")

plt.legend()

plt.grid()


plt.figure()

plt.plot(ACCURACY_tr_f, '-o', markevery = 50, label = 'Accuracy')

plt.xlabel("epoch")

plt.ylabel("Accuracy")

plt.title("Training Set Accuracy")

plt.legend()

plt.grid()
```