```
import numpy as np
from collections import OrderedDict
import copy
```

```
mnist = np.loadtxt('mnist.csv', delimiter=',')
```

```
def train_test_split(csv_dataset):
    # i는 0~100 중의 하나의 수로 train_set의 비율을 나타낸다. ex) 70 => train_set 70% test_set 30%
    #코드 작성
    # dataset의 값을 normalization 하기 위해 256으로 나눈다.
    # X 값은 256으로 나누고, T 값은 label의 값(0에서 9까지의 자연수)이므로 나누지 않는다.
    train_X = csv_dataset[:8000, 1:] / 256
    train_T = csv_dataset[:8000, 0]
    test_X = csv_dataset[8000:, 1:] / 256
    test_T = csv_dataset[8000:, 0]

    return train_X, train_T, test_X, test_T
```

```
def one_hot_encoding(T): # T is data의 label
    #코드 작성
    # 들어온 T에 대해 one hot encoding을 한다.
    # ex)
    # 1        0 1 0
    # 0    ->  1 0 0
    # 2        0 0 1
    # label의 값은 0에서 9까지의 자연수이기 때문에 10가지이고, T.shape[0]은 데이터의 갯수이다.
    classes = 10
    numOfInput = T.shape[0]
    # (numOfInput, classes) 꼴의 array에 0값을 채워 만든다.
    one_hot_label = np.zeros((numOfInput, classes), dtype=int)
    # 값에 해당하는 열의 값만 1로 바꿔주어야한다.
    for i in range(numOfInput):
        one_hot_label[i][(int)(T[i])] = 1 # T번째(열) element의 값만 1로 바꾸어 출력한다.
    return one_hot_label
```

```
def Softmax(ScoreMatrix): # 제공.
    if ScoreMatrix.ndim == 2:
        temp = ScoreMatrix
        temp = temp - np.max(temp, axis=1, keepdims=True)
```

```
        y_predict = np.exp(temp) / np.sum(np.exp(temp), axis=1, keepdims=True)
        return y_predict
    temp = ScoreMatrix - np.max(ScoreMatrix, axis=0)
    expX = np.exp(temp)
    y_predict = expX / np.sum(expX)
    return y_predict
```

```
def setParam_He(neuronlist):

    np.random.seed(1) # seed값 고정을 통해 input이 같으면 언제나 같은 Weight와 bias를
출력하기 위한 함수
    #코드 작성
    # neuronlist의 [0] [1] [2] [3]에 각 레이어의 뉴런 갯수가 순서대로 적혀 있다.
    input_layer = neuronlist[0]
    hidden_layer_1 = neuronlist[1]
    hidden_layer_2 = neuronlist[2]
    output_layer = neuronlist[3]

    # (현재 레이어의 뉴런 갯수, 다음 레이어의 뉴런 갯수) 꼴로 W와 b를 초기화한다.
    # 초기화 방법은 He의 방법을 따른다.
    # 만드려는 뉴럴 네트워크는 3 Layer이다.
    W1 = np.random.randn(input_layer, hidden_layer_1) / np.sqrt(input_layer / 2)
    b1 = np.zeros(hidden_layer_1)
    W2 = np.random.randn(hidden_layer_1, hidden_layer_2) / np.sqrt(hidden_layer_1
/ 2)
    b2 = np.zeros(hidden_layer_2)
    W3 = np.random.randn(hidden_layer_2, output_layer) / np.sqrt(hidden_layer_2 /
2)
    b3 = np.zeros(output_layer)
    '''
    print("W1.shape : ", W1.shape)
    print("b1.shape : ", b1.shape)
    print("W2.shape : ", W2.shape)
    print("b2.shape : ", b2.shape)
    print("W3.shape : ", W3.shape)
    print("b3.shape : ", b3.shape)
    '''
    return W1, W2, W3, b1, b2, b3
```

```
class linearLayer:
    def __init__(self, W, b):
        #backward에 필요한 X, W, b 값 저장 + dW, db값 받아오기

        self.X = None
        self.W = W
        self.b = b
        self.dW = None
        self.db = None
```

```
    def forward(self, x):
        self.X = x
        #내적연산을 통한 Z값 계산
        Z = np.dot(self.X, self.W) + self.b # Z = XW + b
        return Z

    def backward(self, dZ):
        #백워드 함수
        # dZ와 W.T를 dot 연산을 해서 x(input)에 대한 dX를 구한다.
        dx = np.dot(dZ, self.W.T)
        # dZ와 X.T를 dot 연산을 해서 dW를 구한다.
        self.dW = np.dot(self.X.T, dZ)
        # dZ를 통해 db를 구한다.
        self.db = np.sum(dZ, axis=0)
        return dx
```

```
class SiLU:
    def __init__(self):
        self.Z = None # 백워드 시 사용할 로컬 변수

    # 연산을 눈에 잘 띄게 하기 위해 sigmoid와 silu 함수를 만들었습니다.
    def sigmoid(self, X):
        return 1 / (1 + np.exp(-X))

    def silu(self, X):
        return X * self.sigmoid(X)

    def forward(self, Z):
        #수식에 따른 forward 함수 작성
        # SiLU(x) : f(x) = x * sigmoid(x)
        self.Z = Z
        #Activation = Z * self.sigmoid(Z)
        Activation = self.silu(Z)
        return Activation

    def backward(self, dActivation):
        # SiLU'(x) : f'(x) = f(x) + sigmoid(x)(1-f(x))
        dZ = dActivation * (self.silu(self.Z) + (self.sigmoid(self.Z) * (1 -
self.silu(self.Z))))
        return dZ
```

```
# Dropout을 위한 Class를 별도로 선언했습니다.
class Dropout :
    def __init__(self) :
        self.dropout_rate = None
        self.dropout = False
        self.mode = 'train'
```

```
        self.mask = None

    # batch(or minibatch) mode -> dropout mode
    def setDropoutMode(self):
        self.dropout = True

    # dropoput rate setting
    def setDropoutRate(self, rate):
        self.dropout_rate = rate

    # train or test mode setting
    def setTrainOrTest(self, mode='train'):
        self.mode = mode

    def forward(self, x):
        # batch or minibatch
        if self.dropout is False :
            return x
        # dropout
        if self.mode is 'train' :
            # train mode
            self.mask = np.random.rand(*x.shape) > self.dropout_rate
            return x * self.mask
        else :
            # test mode
            return x

    def backward(self, x):
        # batch or minibatch
        if self.dropout is False :
            return x
        # dropout
        else :
            return x * self.mask
```

```
class SoftmaxWithLoss(): # 제공

    def __init__(self):
        self.loss = None
        self.softmaxScore = None
        self.label = None

    def forward(self, score, one_hot_label):

        batch_size = one_hot_label.shape[0]
        self.label = one_hot_label
        self.softmaxScore = Softmax(score)
        '''
        print("label.shape : ", self.label.shape)
        print("scroe.shape : ", self.softmaxScore.shape)
        '''
```

```
        self.loss = -np.sum(self.label * np.log(self.softmaxScore + 1e-20)) /
batch_size

        return self.loss

    def backward(self, dout=1):
        batch_size = self.label.shape[0]
        dx = (self.softmaxScore - self.label) / batch_size
        return dx
```

```
class ThreeLayerNet :

    def __init__(self, paramlist):
        # 구현한 함수를 바탕으로 각 레이어의 W, b 값을 초기화하고, 저장한다.
        W1, W2, W3, b1, b2, b3 = setParam_He(paramlist)
        self.params = {}
        self.params['W1'] = W1
        self.params['W2'] = W2
        self.params['W3'] = W3
        self.params['b1'] = b1
        self.params['b2'] = b2
        self.params['b3'] = b3
        self.layers = OrderedDict()
        # Three Layer Net의 순서대로 OrderedDict에 넣는다.
        # Dropout은 batchOptimization과 minibatchOptimization에선 실제로 작동하지 않
는다.
        # (입력 값을 그대로 반환하는 식으로 구현)
        # DropoutOptimization에서는 작동한다.
        self.layers['L1'] = linearLayer(self.params['W1'], self.params['b1'])
        self.layers['Dropout1'] = Dropout()
        self.layers['SiLU1'] = SiLU()
        self.layers['L2'] = linearLayer(self.params['W2'], self.params['b2'])
        self.layers['Dropout2'] = Dropout()
        self.layers['SiLU2'] = SiLU()
        self.layers['L3'] = linearLayer(self.params['W3'], self.params['b3'])

        self.lastLayer = SoftmaxWithLoss()

    def setRate(self, rate1, rate2):
        self.layers['Dropout1'].setDropoutRate(rate1)
        self.layers['Dropout2'].setDropoutRate(rate2)

    def setDropout(self):
        self.layers['Dropout1'].setDropoutMode()
        self.layers['Dropout2'].setDropoutMode()

    def setTrainOrTest(self, mode):
        self.layers['Dropout1'].setTrainOrTest(mode)
        self.layers['Dropout2'].setTrainOrTest(mode)
```

```
    def scoreFunction(self, x):
        for layer in self.layers.values():
            # 한 줄이 best
            # 각 layer에 forward (Linear, SiLU, Dropout) 가 있기 때문에 이 방법으로
forward 연산을 한다.
            x = layer.forward(x)
        # 최종 연산된 값은 score이다.
        score = x
        return score

    def forward(self, x, label):
        # scoreFunction으로 score를 구하고, 마지막 SoftmaxWithLoss의 forward 함수를
통해 Loss값을 구해 반환한다.
        score = self.scoreFunction(x)
        return self.lastLayer.forward(score, label)

    def accuracy(self, x, label):

        score = self.scoreFunction(x)
        score_argmax = np.argmax(score, axis=1)

        if label.ndim != 1 : #label이 one_hot_encoding 된 데이터면 if문을
            label_argmax = np.argmax(label, axis = 1)

        accuracy  = np.sum(score_argmax==label_argmax) / int(x.shape[0])

        return accuracy

    def backpropagation(self):

        #백워드 함수 작성 스코어펑션을 참고하세요
        # SoftmaxWithLoss의 backward를 구해서
        dx = self.lastLayer.backward()
        # layer 역순으로(reversed) backward 연산을 해 gradients값을 구한다.
        for layer in reversed(self.layers.values()):
            dx = layer.backward(dx)

        # 각 레이어에 구해진 gradients들을 저장한다.
        grads = {}
        grads['W1'] = self.layers['L1'].dW
        grads['b1'] = self.layers['L1'].db
        grads['W2'] = self.layers['L2'].dW
        grads['b2'] = self.layers['L2'].db
        grads['W3'] = self.layers['L3'].dW
        grads['b3'] = self.layers['L3'].db


        return grads

    def gradientdescent(self, grads, learning_rate):
        # gradients값을 통해 w, b 값들을 learning_rate만큼 곱해서 빼준다.
        self.params['W1'] -= learning_rate*grads['W1']
        self.params['W2'] -= learning_rate*grads['W2']
        self.params['W3'] -= learning_rate*grads['W3']
```

```
        self.params['b1'] -= learning_rate*grads['b1']
        self.params['b2'] -= learning_rate*grads['b2']
        self.params['b3'] -= learning_rate*grads['b3']
```

```
def batchOptimization(dataset, ThreeLayerNet, learning_rate, epoch=1000):
    # 반환값을 위한 리스트 선언
    train_acc_list = []
    test_acc_list = []
    Loss_list = []
    for i in range(epoch+1):
        #코드 작성
        # Three Layer Net에 대한 forward 함수를 돌린다.
        # Linear1 -> SiLU -> Linear2 -> SiLU -> Linear3 forward를 통해 Loss값을 구
할 수 있다.
        Loss = ThreeLayerNet.forward(x=dataset['train_X'],
label=dataset['one_hot_train'])
        # 위의 순서와 반대로 backward 연산을 한다.(backpropagation)
        # backpropagation을 통해 각 Layer의 w, b 값에 대해 gradients를 구할 수 있다.
        Grads = ThreeLayerNet.backpropagation()
        # 구한 gradients 값으로 w, b의 값을 -learning_rate를 곱해 update한다.
        ThreeLayerNet.gradientdescent(Grads, learning_rate)

        if i % 10 == 0:
            # 10번째 epoch마다 Train, Test dataset에 대한 accuracy를 구한다.
            train_acc = ThreeLayerNet.accuracy(dataset['train_X'],
dataset['one_hot_train'])
            test_acc = ThreeLayerNet.accuracy(dataset['test_X'],
dataset['one_hot_test'])
            print(i, '\t번째 Loss = ', Loss)
            print(i, '\t번째 Train_Accuracy : ', train_acc)
            print(i, '\t번째 Test_Accuracy : ', test_acc)
            train_acc_list.append(train_acc)
            test_acc_list.append(test_acc)
            Loss_list.append(Loss)

    return ThreeLayerNet, train_acc_list, test_acc_list, Loss_list
```

```
def minibatch_Optimization(dataset, ThreeLayerNet, learning_rate, epoch=100,
batch_size=1000):
    # 반환값을 위한 리스트 선언
    train_acc_list = []
    test_acc_list = []
    Loss_list = []
    # shuffle을 위한 random seed
    np.random.seed(5)
    # random shuffle을 위해 dataset의 train_X와 one_hot_encoding 된 label들을 매칭이
되게(섞더라도 정답은 같아지게) 옆에 붙인다.
    x = dataset['train_X']           # shape : (8000, 784)
```

```
    label = dataset['one_hot_train'] # shape : (8000, 10)
    concat = np.hstack([x, label])   # shape : (8000, 794)

    dataset_size = concat.shape[0]                    # dataset_size = 8000
    minibatch_iter = dataset_size // batch_size       # minibatch_iter = 8000 /
1000 = 8
    pixel = x.shape[1]                                # pixel = 784
    classes = label.shape[1]                          # classes = 10

    for i in range(epoch+1):
        # 코드 작성
        # 매칭이 되게 붙여진 data를 shuffle한다.
        np.random.shuffle(concat)                 # shuffle 후 x, label로 나눈다.
        shuffle_x = concat[:, :pixel]             # shape : (8000, 784)
        shuffle_label = concat[:, -classes:]      # shape : (8000, 10)

        for j in range(minibatch_iter):
            # 8000개의 data를 batch_size(=1000)씩 진행한다.
            # 즉, 1000개씩 8번 진행한다.

            # 0~999, 1000~1999, 2000~2999, 3000~3999, 4000~4999, 5000~5999,
6000~6999, 7000~7999
            start = batch_size * j
            end = batch_size * (j + 1)

            mini_x = shuffle_x[start : end]
            mini_label = shuffle_label[start : end]

            # miniBatch로 batchOptimization과 같은 방식으로 진행한다.
            # Loss값을 구하고, Gradients값을 구하고, 그 값으로 -learning_rate를 곱해
서 update.
            Loss = ThreeLayerNet.forward(mini_x, mini_label)
            Grads = ThreeLayerNet.backpropagation()
            ThreeLayerNet.gradientdescent(Grads, learning_rate)

        if i % 5 == 0:
            # 5번째 epoch마다 Train, Test dataset에 대한 accuracy를 구한다.
            train_acc = ThreeLayerNet.accuracy(dataset['train_X'],
dataset['one_hot_train'])
            test_acc = ThreeLayerNet.accuracy(dataset['test_X'],
dataset['one_hot_test'])
            print(i, '\t번째 Loss = ', Loss)
            print(i, '\t번째 Train_Accuracy : ', train_acc)
            print(i, '\t번째 Test_Accuracy : ', test_acc)
            train_acc_list.append(train_acc)
            test_acc_list.append(test_acc)
            Loss_list.append(Loss)

    return ThreeLayerNet, train_acc_list, test_acc_list, Loss_list

# 랜덤하게 셔플된 데이터를 batch 단위로 나눠서 forward와 backpropagation, gradient
descent를 한다
# shuffle된 모든 데이터에 대해 작업이 끝났을 때가 1 epoch이다.
# ex) 10개의 데이터를 2개의 batch size로 나눈다면 0~2 2~4 4~6 6~8 8~10 한게 1 epoch이
```

```
다.
# 1 epoch이 끝나면 데이터를 다시 섞고 반복한다.
# 5 epoch당 Loss train_acc, test_acc를 출력하고 append 시킨다.
```

```
def dropout_use_Optimizer(dataset, ThreeLayerNet, learning_rate, epoch, kill_n_h1
= 0.25, kill_n_h2 = 0.15):
    # 반환값을 위한 리스트 선언.
    train_acc_list = []
    test_acc_list = []
    Loss_list = []

    train = 'train'
    test = 'test'
    # Dropout 1 rate Dropout 2 rate 설정
    ThreeLayerNet.layers['Dropout1'].setDropoutRate(kill_n_h1)
    ThreeLayerNet.layers['Dropout2'].setDropoutRate(kill_n_h2)
    # Default : no dropout mode -> dropout mode로 설정
    ThreeLayerNet.setDropout()

    for i in range(epoch+1):
        #코드 작성
        # 앞 batchOptimization과 같은 방법으로 진행.
        Loss = ThreeLayerNet.forward(x=dataset['train_X'],
label=dataset['one_hot_train'])
        Grads = ThreeLayerNet.backpropagation()
        ThreeLayerNet.gradientdescent(Grads, learning_rate)

        if i % 10 == 0:
            # predict 과정에선 test 모드로
            ThreeLayerNet.setTrainOrTest(test)
            train_acc = ThreeLayerNet.accuracy(dataset['train_X'],
dataset['one_hot_train'])
            test_acc = ThreeLayerNet.accuracy(dataset['test_X'],
dataset['one_hot_test'])
            # 다시 train 모드로 설정
            ThreeLayerNet.setTrainOrTest(train)

            print(i, '\t번째 Loss = ', Loss)
            print(i, '\t번째 Train_Accuracy : ', train_acc)
            print(i, '\t번째 Test_Accuracy : ', test_acc)
            train_acc_list.append(train_acc)
            test_acc_list.append(test_acc)
            Loss_list.append(Loss)
    return ThreeLayerNet, train_acc_list, test_acc_list, Loss_list
```

```
#과제 채점을 위한 세팅
train_X, train_label, test_X, test_label = train_test_split(mnist)

one_hot_train = one_hot_encoding(train_label)
```

```
one_hot_test = one_hot_encoding(test_label)

dataset = {}
dataset['train_X'] = train_X
dataset['test_X'] = test_X
dataset['one_hot_train'] = one_hot_train
dataset['one_hot_test'] = one_hot_test

neournlist = [784, 60, 30, 10]

TNN_batchOptimizer = ThreeLayerNet(neournlist)
TNN_minibatchOptimizer = copy.deepcopy(TNN_batchOptimizer)
TNN_dropout = copy.deepcopy(TNN_minibatchOptimizer)
```

```
#채점은 이 것의 결과값으로 할 예정입니다.

trained_batch, tb_train_acc_list, tb_test_acc_list, tb_loss_list =
batchOptimization(dataset, TNN_batchOptimizer, 0.1, 1000)
trained_minibatch, tmb_train_acc_list, tmb_test_acc_list, tb_loss_list =
minibatch_Optimization(dataset, TNN_minibatchOptimizer, 0.1, epoch=100,
batch_size=100)
trained_dropout, td_train_acc_list, td_test_acc_list, td_loss_list =
dropout_use_Optimizer(dataset, TNN_dropout, 0.1, 1000, 0.25, 0.15)
```