```
import numpy as np


class singleLayer:
    def __init__(self, W, Bias):  # 제공. 호출 시 작동하는 생성자
        self.W = W
        self.B = Bias

    def SetParams(self, W_params, Bias_params):  # 제공. W와 Bias를 바꾸고 싶을 때
쓰는 함수
        self.W = W_params
        self.B = Bias_params

    def ScoreFunction(self, X):  # \Score값 계산 -> 직접작성
        # 3.2
        # Score = X*W + b
        # shape : (60000,784) * (784, 10) + (1, 10) = (60000, 10)
        ScoreMatrix = np.dot(X, self.W) + self.B
        return ScoreMatrix

    def Softmax(self, ScoreMatrix):  # 제공.
        if ScoreMatrix.ndim == 2:
            temp = ScoreMatrix.T
            temp = temp - np.max(temp, axis=0)
            y_predict = np.exp(temp) / np.sum(np.exp(temp), axis=0)
            return y_predict.T
        temp = ScoreMatrix - np.max(ScoreMatrix, axis=0)
        expX = np.exp(temp)
        y_predict = expX / np.sum(expX)
        return y_predict

    def LossFunction(self, y_predict, Y):  # Loss Function을 구하십시오 -> 직접 작
성
        # 3.3
        # Cross entropy = - sigma (p(x) log(q(x))) / 60000
        # p(x)가 Y, q(x)가 y_predict 이다.
        # 60000개의 이미지 입력(Training set)이 있기 때문에 평균 값을 구하기 위해
60000으로 나눈다.
        cross_entropy = - np.sum(Y * np.log(y_predict + 1e-12)) / 60000
        # return loss
        return cross_entropy

    def Forward(self, X, Y):  # ScoreFunction과 Softmax, LossFunction를 적절히 활용
해 y_predict 와 loss를 리턴시키는 함수. -> 직접 작성
        # 3.4
        # 구현된 ScoreFunction을 이용해서 score를 구한다.
        score = self.ScoreFunction(X)
        # 구현된 Softmax 함수를 이용해서 softmax(score)를 구한다.
        y_predict = self.Softmax(score)
        # 구현된 LossFunction과 위에서 구한 y_predict, 정답(Y)를 이용해서 loss를 구한
다.
        loss = self.LossFunction(y_predict, Y)
```

```
        return y_predict, loss

    def delta_Loss_Scorefunction(self, y_predict, Y):  # 제공.dL/dScoreFunction
        delta_Score = y_predict - Y
        return delta_Score

    def delta_Score_weight(self, delta_Score, X):  # 제공. dScoreFunction / dw .
        delta_W = np.dot(X.T, delta_Score) / X[0].shape
        return delta_W

    def delta_Score_bias(self, delta_Score, X):  # 제공. dScoreFunction / db .
        delta_B = np.sum(delta_Score) / X[0].shape
        return delta_B

    # delta 함수를 적절히 써서 delta_w, delta_b 를 return 하십시오.
    def BackPropagation(self, X, y_predict, Y):
        # 3.5
        # analytic gradient 방법으로 구현된 함수들을 이용하여
        # dL/dw, dL/db를 계산해서 return 한다.
        delta_score = self.delta_Loss_Scorefunction(y_predict, Y)
        delta_W = self.delta_Score_weight(delta_score, X)
        delta_B = self.delta_Score_bias(delta_score, X)
        #delta_W = self.Numerical_Gradient(X, Y)
        return delta_W, delta_B

    def Numerical_Gradient(self, X, Y, h=0.00001):
        delta_W = np.zeros_like(self.W)
        for i in range(0, 784):
            for j in range(0, 10):
                # h를 더했을때 Loss를 구한다.
                self.W[i][j] += h
                f_x_plus_h =
self.LossFunction(self.Softmax(self.ScoreFunction(X)), Y)
                # h를 더하지 않았을 때의 Loss를 구한다.
                self.W[i][j] -= h
                f_x = self.LossFunction(self.Softmax(self.ScoreFunction(X)), Y)
                # delta 값을 구한다.
                delta_W[i][j] = (f_x_plus_h - f_x) / h
        #print(delta_W.shape)
        return delta_W

    # 정확도를 체크하는 Accuracy 제공
    def Accuracy(self, X, Y):
        y_score = self.ScoreFunction(X)
        y_score_argmax = np.argmax(y_score, axis=1)
        if Y.ndim != 1: Y = np.argmax(Y, axis=1)
        accuracy = 100 * np.sum(y_score_argmax == Y) / X.shape[0]
        return accuracy

    # Forward와 BackPropagationAndTraining, Accuracy를 사용하여서 Training을 epoch
만큼 시키고, 10번째 트레이닝마다
    # Training Set의 Accuracy 값과 Test Set의 Accuracy를 print 하십시오

    def Optimization(self, X_train, Y_train, X_test, Y_test, learning_rate=0.01,
```

```
epoch=100):
        for i in range(epoch):
            # 3.6
            # 구현된 함수 Forward로 y_predict, loss값과
            # BackPropagation으로 delta값을 구한다.
            y_predict, loss = self.Forward(X_train, Y_train)
            delta_W, delta_B = self.BackPropagation(X_train, y_predict, Y_train)
            # learning rate와 delta 값을 이용해서 weight와 bias를 update한다.
            self.W -= learning_rate * delta_W
            self.B -= learning_rate * delta_B
            # 함수 작성
            if i % 10 == 0:
                # 3.6 Accuracy 함수 사용
                print(i, "번째 트레이닝")
                print('현재 Loss(Cost)의 값 : ', loss)
                print("Train Set의 Accuracy의 값 : ", self.Accuracy(X_train,
Y_train))
                print("Test Set의 Accuracy의 값 :", self.Accuracy(X_test, Y_test))
```