```
import matplotlib.pyplot as plt
from sklearn import datasets
import numpy as np

#데이터값 추출
digits_data = datasets.load_digits()
input_data = digits_data.data
correct = digits_data.target
n_data = len(correct)

#데이터값을 표준정규분포(평균0, 분산1)
ave_input = np.average(input_data)
std_input = np.std(input_data)
input_data = (input_data - ave_input) / std_input

#데이터에 맞는 정답값을 1로 설정(원핫코딩)
correct_data = np.zeros((n_data, 10))
for i in range(n_data):
    correct_data[i, correct[i]] = 1.0

#훈련용 데이터와 실험용 데이터 분류 (2:1 비율)
index = np.arange(n_data)
index_train = index[index%3 != 0]
index_test = index[index%3 == 0]

#훈련용데이터와 실험용 데이터 저장
input_train = input_data[index_train, :]
correct_train = correct_data[index_train, :]
input_test = input_data[index_test, :]
correct_test = correct_data[index_test, :]

#훈련용데이터와 실험용 데이터의 수
n_train = input_train.shape[0]
n_test = input_test.shape[0]

#기본 설정(이미지크기 8x8, 채널수 1, 가중치와 편향 분산 0.1, 학습률 0.01, 에포크50, 배치사이즈 8)
img_h = 8
img_w = 8
```

```
img_ch = 1

wb_width = 0.1
eta = 0.01
epoch = 50
batch_size = 8
interval = 10
n_sample = 200

#im2col(사진 -> 행렬), col2im(행렬 -> 사진) 함수
def im2col(images, flt_h, flt_w, out_h, out_w, stride, pad):
    n_bt, n_ch, img_h, img_w = images.shape

    img_pad = np.pad(images, [(0,0), (0,0), (pad, pad), (pad, pad)], "constant")
    cols = np.zeros((n_bt, n_ch, flt_h, flt_w, out_h, out_w))

    for h in range(flt_h):
        h_lim = h + stride*out_h
        for w in range(flt_w):
            w_lim = w + stride*out_w
            cols[:, :, h, w, :, :] = img_pad[:, :, h:h_lim:stride, w:w_lim:stride]

    cols = cols.transpose(1, 2, 3, 0, 4, 5).reshape(n_ch*flt_h*flt_w, n_bt*out_h*out_w)
    return cols

def col2im(cols, img_shape, flt_h, flt_w, out_h, out_w, stride, pad):
    n_bt, n_ch, img_h, img_w = img_shape

    cols = cols.reshape(n_ch, flt_h, flt_w, n_bt, out_h, out_w).transpose(3, 0, 1, 2, 4, 5)
    images = np.zeros((n_bt, n_ch, img_h+2*pad+stride-1, img_w+2*pad+stride-1))
    for h in range(flt_h):
        h_lim = h + stride*out_h
        for w in range(flt_w):
            w_lim = w + stride * out_w
            images[:, :, h:h_lim:stride, w:w_lim:stride] += cols[:, :, h, w, :, :]

    return images[:, :, pad:img_h+pad, pad:img_w+pad]
```

```python
#컨볼루션층 구현
class ConvLayer:
    def __init__(self, x_ch, x_h, x_w, n_flt, flt_h, flt_w, stride, pad):
        self.params = (x_ch, x_h, x_w, n_flt, flt_h, flt_w, stride, pad)

        self.w = wb_width * np.random.randn(n_flt, x_ch, flt_h, flt_w)
        self.b = wb_width * np.random.randn(1, n_flt)

        self.y_ch = n_flt
        self.y_h = (x_h - flt_h + 2*pad) // stride +1
        self.y_w = (x_w - flt_w + 2*pad) // stride +1

        self.h_w = np.zeros((n_flt, x_ch, flt_h, flt_w)) + 1e-8
        self.h_b = np.zeros((1, n_flt)) + 1e-8

    def forward(self, x):
        n_bt = x.shape[0]
        x_ch, x_h, x_w, n_flt, flt_h, flt_w, stride, pad = self.params
        y_ch, y_h, y_w = self.y_ch, self.y_h, self.y_w

        self.cols = im2col(x, flt_h, flt_w, y_h, y_w, stride, pad)
        self.w_col = self.w.reshape(n_flt, x_ch*flt_h*flt_w)

        u = np.dot(self.w_col, self.cols).T + self.b
        self.u = u.reshape(n_bt, y_h, y_w, y_ch).transpose(0, 3, 2, 1)
        self.y = np.where(self.u <= 0, 0, self.u)

    def backward(self, grad_y):
        n_bt = grad_y.shape[0]
        x_ch, x_h, x_w, n_flt, flt_h, flt_w, stride, pad= self.params
        y_ch, y_h, y_w = self.y_ch, self.y_h, self.y_w

        delta = grad_y * np.where(self.u <= 0, 0, 1)
        delta = delta.transpose(0, 2, 3, 1).reshape(n_bt*y_h*y_w, y_ch)

        grad_w = np.dot(self.cols, delta)
        self.grad_w = grad_w.T.reshape(n_flt, x_ch, flt_h, flt_w)
        self.grad_b = np.sum(delta, axis = 0)

        grad_cols = np.dot(delta, self.w_col)
        x_shape = (n_bt, x_ch, x_h, x_w)
        self.grad_x = col2im(grad_cols.T, x_shape, flt_h, flt_w, y_h, y_w, stride, pad)

    def update(self, eta):
        self.h_w += self.grad_w * self.grad_w
        self.w -= eta / np.sqrt(self.h_w) * self.grad_w

        self.h_b += self.grad_b * self.grad_b
        self.b -= eta / np.sqrt(self.h_b) *self.grad_b
```

```python
#풀링층 구현
class PoolingLayer:
    def __init__(self, x_ch, x_h, x_w, pool, pad):

        self.params = (x_ch, x_h, x_w, pool, pad)

        self.y_ch = x_ch
        self.y_h = x_h//pool if x_h%pool==0 else x_h//pool+1
        self.y_w = x_w//pool if x_w%pool==0 else x_w//pool+1

    def forward(self, x):
        n_bt = x.shape[0]
        x_ch, x_h, x_w, pool, pad = self.params
        y_ch, y_h, y_w = self.y_ch, self.y_h, self.y_w

        cols = im2col(x, pool, pool, y_h, y_w, pool, pad)
        cols = cols.T.reshape(n_bt*y_h*y_w*x_ch, pool*pool)

        y = np.max(cols, axis =1)
        self.y = y.reshape(n_bt, y_h, y_w, x_ch).transpose(0, 3, 1, 2)

        self.max_index = np.argmax(cols, axis = 1)

    def backward(self, grad_y):
        n_bt = grad_y.shape[0]
        x_ch, x_h, x_w, pool, pad = self.params
        y_ch, y_h, y_w = self.y_ch, self.y_h, self.y_w

        grad_y = grad_y.transpose(0, 2, 3, 1)

        grad_cols = np.zeros((pool*pool, grad_y.size))

        grad_cols[self.max_index.reshape(-1), np.arange(grad_y.size)] = grad_y.reshape(-1)
        grad_cols = grad_cols.reshape(pool, pool, n_bt, y_h, y_w, y_ch)
        grad_cols = grad_cols.transpose(5, 0, 1, 2, 3, 4)
        grad_cols = grad_cols.reshape(y_ch*pool*pool, n_bt*y_h*y_w)

        x_shape = (n_bt, x_ch, x_h, x_w)
        self.grad_x = col2im(grad_cols, x_shape, pool, pool, y_h, y_w, pool, pad)
```

```python
#전결합층 기본 세팅(가중치와 편향 설정, 아다그리드 방법으로 가중치와 편향 수정)
class BaseLayer:
    def __init__(self, n_upper, n):
        self.w = wb_width * np.random.randn(n_upper,n)
        self.b = wb_width * np.random.randn(n)

        self.h_w = np.zeros((n_upper, n)) + 1e-8
        self.h_b = np.zeros(n) + 1e-8

    def update(self, eta):
        self.h_w += self.grad_w * self.grad_w
        self.w -= eta / np.sqrt(self.h_w) * self.grad_w

        self.h_b += self.grad_b * self.grad_b
        self.b -= eta/ np.sqrt(self.h_b) * self.grad_b

#전결합층 은닉층 구현
class MiddleLayer(BaseLayer):
    def forward(self, x):
        self.x = x
        self.u = np.dot(x, self.w) + self.b
        self.y = np.where(self.u <= 0, 0, self.u)

    def backward(self, grad_y):
        delta = grad_y * np.where(self.u <= 0, 0 ,1)

        self.grad_w = np.dot(self.x.T, delta)
        self.grad_b = np.sum(delta, axis = 0)

        self.grad_x = np.dot(delta, self.w.T)

#전결합층 출력층 구현
class OutputLayer(BaseLayer):
    def forward(self, x):
        self.x = x
        u = np.dot(x, self.w) + self.b
        self.y = np.exp(u)/np.sum(np.exp(u), axis = 1).reshape(-1, 1)

    def backward(self, t):
        delta = self.y - t

        self.grad_w = np.dot(self.x.T, delta)
        self.grad_b = np.sum(delta, axis = 0)

        self.grad_x = np.dot(delta, self.w.T)
```

```python
#각 층 설정
cl_1 = ConvLayer(img_ch, img_h, img_w, 10, 3, 3, 1, 1)
pl_1 = PoolingLayer(cl_1.y_ch, cl_1.y_h, cl_1.y_w, 2, 0)

n_fc_in = pl_1.y_ch * pl_1.y_h * pl_1.y_w
ml_1 = MiddleLayer(n_fc_in, 100)
ol_1 = OutputLayer(100, 10)

#모든 층을 지나는 순전파
def forward_propagation(x):
    n_bt = x.shape[0]

    images = x.reshape(n_bt, img_ch, img_h, img_w)
    cl_1.forward(images)
    pl_1.forward(cl_1.y)

    fc_input = pl_1.y.reshape(n_bt, -1)
    ml_1.forward(fc_input)
    ol_1.forward(ml_1.y)

#모든 층을 지나는 역전파
def backpropagation(t):
    n_bt = t.shape[0]

    ol_1.backward(t)
    ml_1.backward(ol_1.grad_x)

    grad_img = ml_1.grad_x.reshape(n_bt, pl_1.y_ch, pl_1.y_h, pl_1.y_w)
    pl_1.backward(grad_img)
    cl_1.backward(pl_1.grad_x)

#가중치와 편향 수정
def uppdate_wb():
    cl_1.update(eta)
    ml_1.update(eta)
    ol_1.update(eta)

#교차 엔트로피를 이용한 오차 구하는 함수
def get_error(t, batch_size):
    return -np.sum(t * np.log(ol_1.y + 1e-7)) / batch_size

#최종 훈련된 층의 훈련값과 실제값의 오차를 보기위한 함수
def forward_sample(inp, correct, n_sample):
    index_rand = np.arange(len(correct))
    np.random.shuffle(index_rand)
    index_rand = index_rand[:n_sample]
    x = inp[index_rand, :]
    t = correct[index_rand, :]
    forward_propagation(x)
    return x, t
```

```
#훈련 과정
train_error_x = []
train_error_y = []
test_error_x = []
test_error_y = []

n_batch = n_train // batch_size
for i in range(epoch):
    x, t = forward_sample(input_train, correct_train, n_sample)
    error_train = get_error(t, n_sample)

    x, t = forward_sample(input_test, correct_test, n_sample)
    error_test = get_error(t, n_sample)

    train_error_x.append(i)
    train_error_y.append(error_train)
    test_error_x.append(i)
    test_error_y.append(error_test)

    if i%interval == 0:
        print("Epoch:" + str(i) + "/" + str(epoch),
              "Error_train:" + str(error_train),
              "Error_test:" + str(error_test))

    index_rand = np.arange(n_train)
    np.random.shuffle(index_rand)
    for j in range(n_batch):
        mb_index = index_rand[j*batch_size : (j+1)*batch_size]
        x = input_train[mb_index, :]
        t = correct_train[mb_index, :]

        forward_propagation(x)
        backpropagation(t)
        uppdate_wb()
```

```
#그래프 그리기
plt.plot(train_error_x, train_error_y, label="Train")
plt.plot(test_error_x, test_error_y, label="Test")
plt.legend()

plt.xlabel("Epoches")
plt.ylabel("Error")
plt.show()

#최종 훈련된 층의 훈련값과 실험값 오차 측정
x,t = forward_sample(input_train, correct_train, n_train)
count_train = np.sum(np.argmax(ol_1.y, axis=1) == np.argmax(t, axis=1))

x,t = forward_sample(input_test, correct_test, n_test)
count_test = np.sum(np.argmax(ol_1.y, axis=1) == np.argmax(t, axis=1))

print("Accuracy Train:", str(count_train/n_train*100) + "%",
      "Acuuracy Test:", str(count_test/n_test*100) + "%")
```