```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

#X와 Y의 범위 설정
X = np.arange(-1.0, 1.1, 0.1)
Y = np.arange(-1.0, 1.1, 0.1)

input_data = []
correct_data = []
for x in X :
    for y in Y :
        input_data.append([x,y])
        if y < np.sin(np.pi * x):
            correct_data.append([0, 1]) #sin 아래영역
        else:
            correct_data.append([1, 0]) # sin 위쪽영역

n_data = len(correct_data) #data 개수

#훈련 데이터 set
input_data = np.array(input_data)
correct_data = np.array(correct_data)

n_in = 2  #입력층 뉴런수
n_mid = 6  #은닉층 뉴런수
n_out = 2  #출력층 뉴런수

wb_width = 0.01  #정규분포의 표준편차 수정
eta = 0.1        #학습률
epoch = 101      #101번 에포크
interval = 10    #10번마다 에포크의 경과 확인

class MiddleLayer:
    #은닉층 클래스
    def __init__(self, n_upper, n):
    #평균0 , 표준편차 0.01인 정규분포를 따르는 난수 생성
        self.w = wb_width * np.random.randn(n_upper, n) #가중치 사이즈 (2x8)
        self.b = wb_width * np.random.randn(n)          #편향 사이즈 (8)

    def forward(self, x):
        #순전파
        self.x = x                          #x의 사이즈는 (1x2)
        u = np.dot(x, self.w) + self.b   #(1x2) * (2x8) + (8)[브로드캐스트] = (1x8)
        self.y = 1/(1+np.exp(-u))           #활성화 함수 : 시그모이드 함수

    def backward(self, grad_y):
        #역전파
        delta = grad_y * (1-self.y) * self.y

        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)

    def update(self, eta):
        #가중치와 편향 수정
        self.w -= eta * self.grad_w
        self.b -= eta * self.grad_b

class OutputLayer:
    #출력층 클래스
    def __init__(self, n_upper, n):
    #평균0 , 표준편차 0.01인 정규분포를 따르는 난수 생성
        self.w = wb_width * np.random.randn(n_upper, n) #가중치 사이즈 (8x2)
        self.b = wb_width * np.random.randn(n)          #편향 사이즈 (2)

    def forward(self, x):
        #순전파
        self.x = x                          #x의 사이즈는 (1x8)
        u = np.dot(x,self.w) + self.b    #(1x8) * (8x2) + (2)[브로드캐스트] = (1x2)
        self.y = np.exp(u)/np.sum(np.exp(u), axis =1, keepdims = True)       #활성화 함수 : 소프트맥스 함수

    def backward(self, t):
        #역전파
        delta = self.y - t #교재 p.188 참고

        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)
```

```
    def update(self, eta):
        #가중치와 편향 수정
        self.w -= eta * self.grad_w
        self.b -= eta * self.grad_b

middle_layer = MiddleLayer(n_in, n_mid)
output_layer = OutputLayer(n_mid, n_out)

sin_data = np.sin(np.pi * X)
for i in range (epoch):

    index_random = np.arange(n_data) #확률적 경사하강법 사용
    np.random.shuffle(index_random)

    total_error = 0
    x_1 = []
    y_1 = []
    x_2 = []
    y_2 = []

    for idx in index_random:
        x = input_data[idx]
        t = correct_data[idx]

        middle_layer.forward(x.reshape(1,2))
        output_layer.forward(middle_layer.y)

        output_layer.backward(t.reshape(1,2))
        middle_layer.backward(output_layer.grad_x)

        middle_layer.update(eta)
        output_layer.update(eta)

        if i%interval == 0:

            y = output_layer.y.reshape(-1)

            total_error +=  - np.sum(t * np.log(y + 1e-7))

            if y[0] > y[1]:
                x_1.append(x[0])
                y_1.append(x[1])
            else:
                x_2.append(x[0])
                y_2.append(x[1])

    if i%interval == 0:

        plt.plot(X, sin_data, linestyle="dashed")
        plt.scatter(x_1, y_1, marker="+")
        plt.scatter(x_2, y_2, marker="x")
        plt.show()

        print("Epoch:" + str(i) + "/" + str(epoch),
              "Error:" +  str(total_error/n_data))
```