과제2

작성자 박종훈
날짜 240619
2차과제

Deep Learning Assignment #2

High Performance Integrated Circuit Design Lab.

- 실행해보는 문제는 직접 실행하고, 그 캡쳐본 또는 코드 및 결과를 담번으로 제출하시면 됩니다.

3. (Chapter 3)

3.1. (활성화 함수)

3.1.1. 활성화 함수 (계단 함수, 시그모이드 함수, 렐루 함수)를 plot 하시오.

3.1.2. 신경망에서 활성화 함수로 비선형 함수를 사용하는 이유를 설명하시오.

3.2. (다층 퍼셉트론)

3.2.1. 단층 퍼셉트론과 다층 퍼셉트론의 차이점을 서술하시오.

3.2.2. 단순한 3 층 신경망의 연산과정을 코드로 실행해보고, 연산 과정을 그림으로 간단하게 나타내시오. (단, 입력 신호의 형상은 (2,)이고, 은닉층 뉴런의 개수는 각각 3 개, 2 개이며, 출력층 뉴런의 개수는 2 개이다. 각 연산은 편향을 포함한다. 또한 은닉층의 활성화 함수로는 시그모이드 함수를 사용하고, 출력층의 활성화 함수로는 항등 함수를 사용하라.)

3.3. (다중 클래스 분류 문제에서의 출력층)

3.3.1. 출력층 뉴런의 개수가 의미하는 바를 서술하시오.

3.3.2. 다중 클래스 분류 문제에서, 출력층에 사용되는 활성화 함수가 무엇인지 밝히고, 해당 활성화 함수에 대하여 자세히 설명하시오.

3.3.3. 해당 활성화 함수를 코드로 구현하고, 코드로 구현될 때 분모의 exponential 항에서 c 를 빼주는 이유에 대하여 설명하시오.

3.4. (MNIST 데이터셋)

3.4.1. MNIST 데이터셋이 무엇인지 설명하시오.

3.4.2. MNIST 데이터셋에서, 각 이미지 데이터의 사이즈를 서술하시오.

3.4.3. MNIST 데이터셋에서, 이미지 데이터의 각 픽셀 값의 범위 및 픽셀 값의 의미를 설명하시오.

3.4.4. load_mnist 함수의 인자인 normalize, flatten, one_hot_label 에 대하여 설명하시오.

3.4.5. 60000 개의 MNIST 데이터셋 중 무작위로 하나를 선택하여 배열이 어떻게 이루어져 있나 살펴보고, 해당 데이터를 plot 해보시오. 또한 Plot 결과와 해당 이미지의 label 을 비교해보시오. (단, normalization 은 하지 않는다.)

3.4.6. 0.0~1.0 의 값으로 normalization 하는 코드를 포함시켜서 3.4.5 번 문제의 과정을 반복하시오. 또한 3.4.5 번 문제의 plot 결과와 비교해보고, 그 결과에 대하여 고찰하시오.

3.1. (활성화 함수)

3.1.1. 활성화 함수 (계단 함수, 시그모이드 함수, 렐루 함수)를 plot 하시오.

활성화 함수: 입력 신호의 총합을 출력 신호로 변환하는 함수.

입력신호의 총합이 활성화를 일으키는지 정하는 역할을 한다.

$$a = b + w_1x_1 + w_2x_2$$

가중치가 달린 입력신호와 편향의 총합 = a

$$y = h(a)$$

a를 h()에 넣어 y를 출력

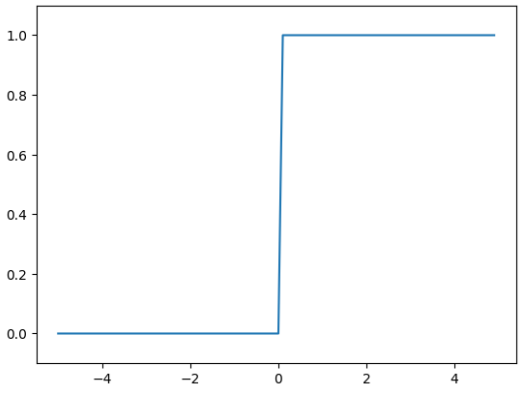
시그모이드 함수

$$h(x) = \frac{1}{1 + exp(-x)}$$

신경망에서는 활성화 함수로 시그모이드 함수를 이용해 신호를 변환하고 변환된 신호를 다음 뉴런에 전달한다.

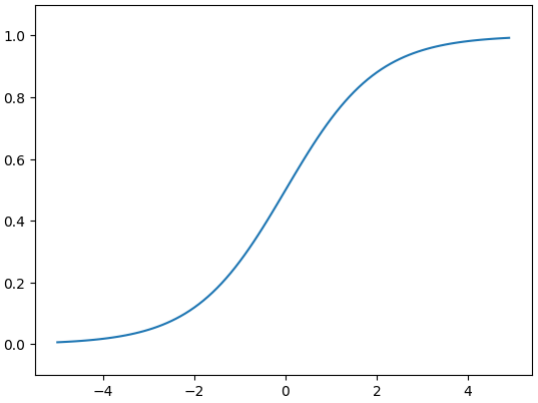
계단 함수

```
import numpy as np
import matplotlib.pylab as plt


def step_function(x):
    return np.array(x > 0, dtype=int) # np.int는 에러발생
                                                                    #AttributeError: module 'numpy' has no attribute 'int'.

X = np.arange(-5.0, 5.0, 0.1)
Y = step_function(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)  # y축의 범위 지정
plt.show()
```



시그모이드 함수

```
import numpy as np
import matplotlib.pylab as plt


def sigmoid(x):
    return 1 / (1 + np.exp(-x))

X = np.arange(-5.0, 5.0, 0.1)
Y = sigmoid(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)
plt.show()
```



차이점: 매끄러움

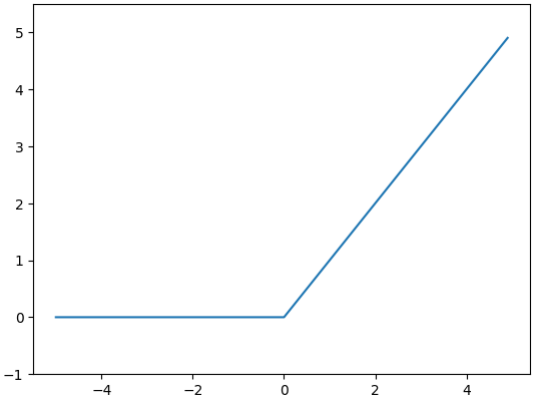
계단함수는 0과 1중 하나만 돌려준다.

시그모이드는 실수를 돌려준다.

ReLU함수

입력이 0을 넘으면 그 입력을 그대로 출력하고, 0이하면 0을 출력한다.

```
import numpy as np
import matplotlib.pylab as plt


def relu(x):
    return np.maximum(0, x) # 두입력중 큰값 반환

x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)
plt.plot(x, y)
plt.ylim(-1.0, 5.5)
plt.show()
```



3.1.2. 신경망에서 활성화 함수로 비선형 함수를 사용하는 이유를 설명하시오.

계단함수와 시그모이드 공통점: 비선형 함수

비선형 함수란 직선 1개로는 그릴 수 없는 함수를 말한다.

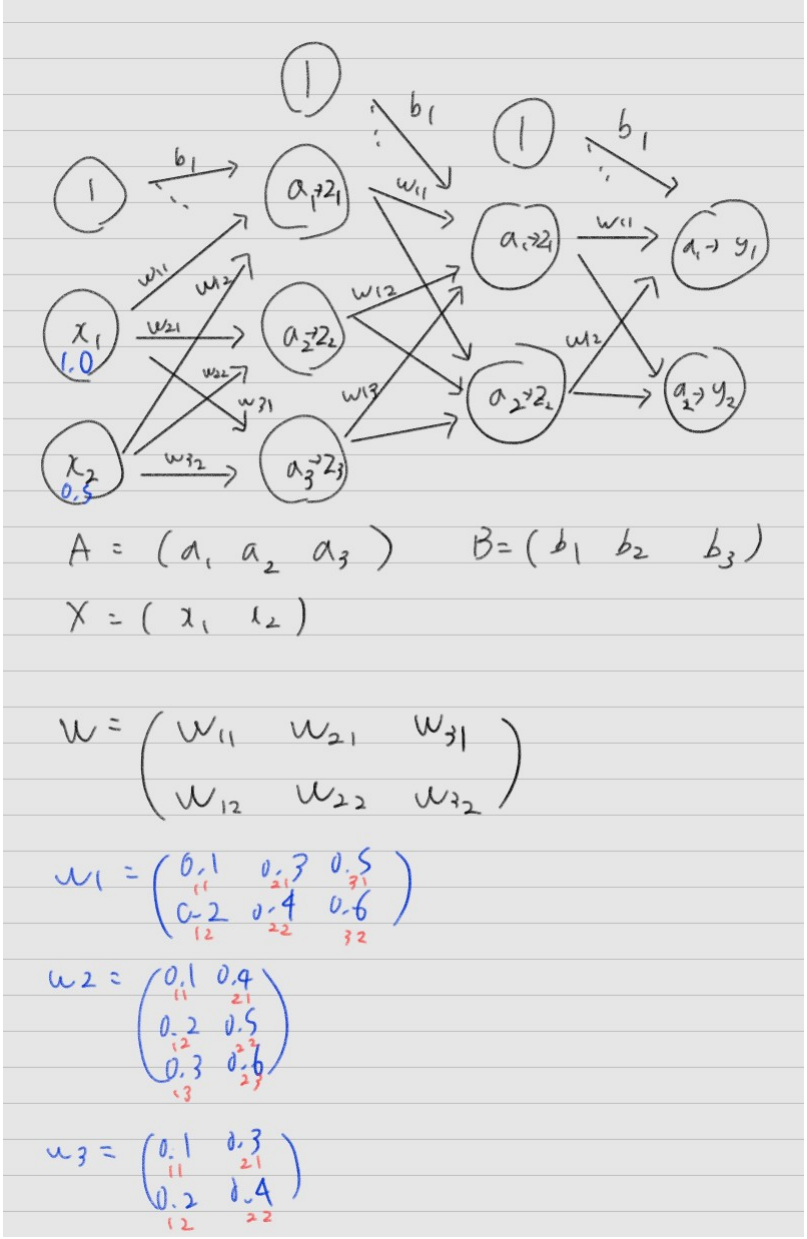
선형함수를 이용하면 은닉층이 없고 여러 층으로 구성할 이유가 없다.

→ 층을 쌓는 혜택을 얻고 싶으면 활성화 함수로 비선형 함수를 사용해야 한다.

3.2. (다층 퍼셉트론)

3.2.1. 단층 퍼셉트론과 다층 퍼셉트론의 차이점을 서술하시오.

3.2.2. 단순한 3층 신경망의 연산과정을 코드로 실행해보고, 연산 과정을 그림으로 간단하게 나타내시오. (단, 입력 신호의 형상은 (2,)이고, 은닉층 뉴런의 개수는 각각 3개, 2개이며, 출력층 뉴런의 개수는 2개이다. 각 연산은 편향을 포함한다. 또한 은닉층의 활성화 함수로는 시그모이드 함수를 사용하고, 출력층의 활성화 함수로는 항등 함수를 사용하라.)




```
def identity_function(x): # 항등함수 - 입력 그대로 출력
  return x

def init_network():
    network = {}
    network['W1'] = np.array([[0.1,0.3,0.5],[0.2,0.4,0.6]]) # 가중치 W1 - 2x3 행렬
    network['b1'] = np.array([0.1,0.2,0.3]) # BIAS1
    network['W2'] = np.array([[0.1,0.4],[0.2,0.5],[0.3,0.6]])
    network['b2'] = np.array([0.1,0.2])
    network['W3'] = np.array([[0.1,0.3],[0.2,0.4]])
    network['b3'] = np.array([0.1,0.2])

    return network

def forward(network, x):
    w1, w2, w3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
```

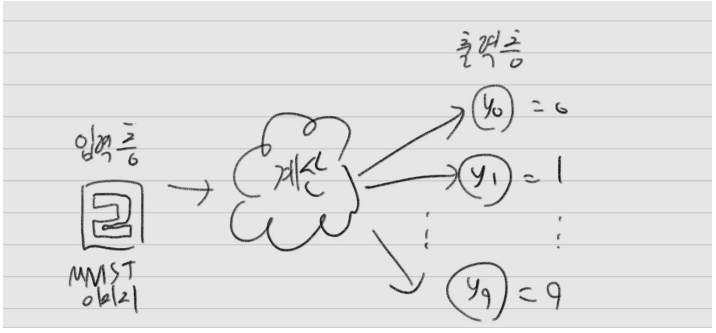
```
    a1 = np.dot(x, w1) + b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1, w2) + b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2, w3) + b3
    y = identity_function(a3)

    return y

network = init_network()
x = np.array([1.0,0.5]) # 원소 2개 1차원 배열
y = forward(network,x)
print(y)  # [0.31682708 0.69627909]
```

1. **입력 신호의 형상:**
   - `x = np.array([1.0, 0.5])` → (2,) 형상
2. **은닉층 뉴런의 개수:**
   - 첫 번째 은닉층: `W1`의 형상 (2, 3), `b1`의 형상 (3,)
   - 두 번째 은닉층: `W2`의 형상 (3, 2), `b2`의 형상 (2,)
3. **출력층 뉴런의 개수:**
   - 출력층: `W3`의 형상 (2, 2), `b3`의 형상 (2,)
4. **편향 포함:**
   - 각 연산에 편향이 포함되어 있음 (`b1`, `b2`, `b3` 사용)
5. **은닉층의 활성화 함수:**
   - `z1 = sigmoid(a1)`
   - `z2 = sigmoid(a2)`
6. **출력층의 활성화 함수:**
   - `y = identity_function(a3)`

단계	연산	설명	결과
입력층	x	입력 신호	[1.0, 0.5]
첫 번째 은닉층	a1 = x · W1 + b1	가중합	[0.3, 0.7, 1.1]
첫 번째 은닉층	z1 = σ(a1)	활성화 함수 (시그모이드)	[0.57444252, 0.66818777, 0.75026011]
두 번째 은닉층	a2 = z1 · W2 + b2	가중합	[0.51615984, 1.21402696]
두 번째 은닉층	z2 = σ(a2)	활성화 함수 (시그모이드)	[0.62624937, 0.77102794]
출력층	a3 = z2 · W3 + b3	가중합	[0.31682708, 0.69627909]
출력층	y = a3	활성화 함수 (항등 함수)	[0.31682708, 0.69627909]

3.3. (다중 클래스 분류 문제에서의 출력층)

3.3.1. 출력층 뉴런의 개수가 의미하는 바를 서술하시오.



출력층의 뉴런 수는 풀려는 문제에 맞게 적절히 정해야 한다.

분류에서는 분류하고 싶은 클래스 수로 설정하는 것이 일반적이다.

예를 들어 입력 이미지를 숫자 0부터 9 중 하나로 분류하는 문제라면 출력층의 뉴런을 10개로 설정한다.

y_0부터 y_9까지 클래스 중 가장 출력값이 큰 y_2를 신경망이 선택한다.

3.3.2. 다중 클래스 분류 문제에서, 출력층에 사용되는 활성화 함수가 무엇인지 밝히고, 해당 활성화 함수에 대하여 자세히 설명하시오.

다중 클래스 분류 문제에서 출력층에 사용하는 활성화 함수는 소프트맥스 함수를 사용한다.

회귀 문제에는 항등함수를, 분류 문제에는 소프트맥스 함수를 주로 사용한다.

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

위 수식을 직관적으로 해석하면

$\frac{k번째입력값}{n번째입력값의합}$

으로 볼 수 있으며 따라서 확률 관점으로 볼 수 있다.

지수함수가 사용되는 이유는 미분이 가능하도록 하게 함이며, 입력값 중 큰 값은 더 크게 작은 값은 더 작게 만들어 입력벡터가 더 잘 구분되게 하기 위함이다.

- exp(x): 지수함수
- n: 출력층의 뉴런 수
- *yk*: 그 중 k번째 출력
- 분자는 입력신호 *ak*의 지수함수, 분모는 모든 입력 신호의 지수 함수의 합으로 구성

소프트맥스 함수는 다중 클래스 분류 모델을 만들 때 사용한다.

결과를 확률로 해석할 수 있게 변환해주는 함수로 높은 확률을 가지는 class로 분류한다.

이는 결과값을 정규화시키는 것으로도 생각할 수 있다.

https://velog.io/@u_jinju/DL-항등함수와-소프트맥스-함수-구현하기

https://syj9700.tistory.com/38

3.3.3. 해당 활성화 함수를 코드로 구현하고, 코드로 구현될 때 분모의 exponential항에서 c를 빼주는 이유에 대하여 설명하시오.

소프트맥스 함수 구현

```
import numpy as np
import matplotlib.pyplot as plt

a = np.array([0.3,2.9,4.0])
exp_a = np.exp(a) # 지수함수
print(exp_a)
sum_exp_a = np.sum(exp_a) # 지수함수의 합
print(sum_exp_a)

y = exp_a / sum_exp_a
print(y)
```

→ 소프트맥스 함수는 지수함수를 사용하므로 오버플로 문제를 야기한다

→ 큰 값이 inf로 출력된다

→ 큰 값끼리 나눗셈을 하면 결과 수치가 불안정해진다

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} = \frac{C \exp(a_k)}{C \sum_{i=1}^{n} \exp(a_i)}$$

$$= \frac{\exp(a_k + log C)}{\sum_{i=1}^{n} \exp(a_i + log C)}$$

$$= \frac{\exp(a_k + C')}{\sum_{i=1}^{n} \exp(a_i + C')}$$

위를 반영한 코드이다

```
import numpy as np
import matplotlib.pyplot as plt

a = np.array([1010, 1000, 990])
print(np.exp(a) / np.sum(np.exp(a)) )# 소프트맥스 함수의 계산 / [nan nan nan]
c = np.max(a) # 제대로 계산되지 않음
print(a-c) # [  0 -10 -20]
print(np.exp(a-c) / np.sum(np.exp(a-c))) # [9.99954600e-01 4.53978686e-05 2.06106005e-09]

def softmax(a):
  c = np.max(a) # c = 1010 (최댓값)
  exp_a = np.exp(a-c) # 오버플로 대책
  sum_exp_a = np.sum(exp_a)
  return y


a = np.array([0.3,2.9,4.0])
y = softmax(a)
```

```
print(y) # 소프트맥스 함수의 출력은 0과 1 사이의 실수
np.sum(y) # 출력 총합은 1
```

3.4. (MNIST 데이터셋)

3.4.1. MNIST 데이터셋이 무엇인지 설명하시오.

MNIST 데이터셋은 손으로 쓴 숫자(0-9)의 이미지 데이터셋으로, 주로 기계 학습 및 패턴 인식 알고리즘의 학습 및 테스트에 사용되는 표준 데이터셋이다.

MNIST 데이터셋은 다양한 기계 학습 알고리즘의 성능을 비교하기 위한 표준 벤치마크로 널리 사용된다. 특히, 딥러닝 모델의 성능을 평가하는 데 자주 사용된다. 이 데이터셋은 단순한 전처리와 직관적인 구조 덕분에 초보자들이 기계 학습 및 딥러닝을 학습하는 데 적합하다.

3.4.2. MNIST 데이터셋에서, 각 이미지 데이터의 사이즈를 서술하시오.

이미지 크기: 각 이미지는 28x28 픽셀 크기의 흑백 이미지이다.

3.4.3. MNIST 데이터셋에서, 이미지 데이터의 각 픽셀 값의 범위 및 픽셀 값의 의미를 설명하시오.

이미지 데이터는 28 X 28 크기의 회색조 이미지이며 각 픽셀은 0에서 255 사이의 값을 취한다. 0은 흰색(배경), 255는 검은색(숫자)이다.각 이미지에는 '7' '2' '1'과 같이 이미지가 실제 의미하는 숫자가 레이블로 붙어 있다.

3.4.4. load_mnist 함수의 인자인 normalize, flatten, one_hot_label 에 대하여 설명하시오.

- **normalize**

입력 이미지의 픽셀값을 0.0~1.0 사이의 값으로 정규화할지를 정한다.

False로 설정하면 입력 이미지의 픽셀은 원래 그대로 0~255값 사이를 유지한다.

- **flatten**

입력이미지를 평탄하게, 즉 1차원 배열로 만들지를 정한다.

False로 설정하면 입력이미지를 1 x 28 × 28의 3차원 배열로, True로 설정하면 784개로 이뤄진 1차원 배열로 저장한다.

- **one_hot_label**

레이블을 원-핫 인코딩 형태로 저장할지를 정한다.

원-핫 인코딩이란 예를 들어 [0,0,0,0,1,0,0,0] 처럼 정답을 뜻하는 원소만 1이고 (hot하고) 나머지는 모두 0인 배열이다. one_hot_label이 False면 7이나 2와 같이 숫자 형태의 레이블을 저장하고, True일때는 원핫인코딩하여 저장한다.

3.4.5. 60000 개의 MNIST 데이터셋 중 무작위로 하나를 선택하여 배열이 어떻게 이루어져 있나 살펴보고, 해당 데이터를 plot 해보시오. 또한 Plot 결과와 해당 이미지의 label을 비교해보시오. (단, normalization 은 하지 않는다.)

`mnist_downloader.py`

```
# mnist_downloader.py
try:
    import urllib.request  # urllib.request 모듈을 임포트 시도
except ImportError:
    raise ImportError('You should use Python 3.x')  # Python 3.x을 사용해야 한다는 에러 메시지 발생
import os.path  # os.path 모듈 임포트
import gzip  # gzip 모듈 임포트
import pickle  # pickle 모듈 임포트
import os  # os 모듈 임포트
import numpy as np  # numpy 모듈을 np라는 별칭으로 임포트

url_base = 'http://yann.lecun.com/exdb/mnist/'  # MNIST 데이터셋 URL 기본 경로
key_file = {  # 데이터셋 파일 이름들을 딕셔너리로 정의
    'train_img': 'train-images-idx3-ubyte.gz',  # 훈련 이미지 파일 이름
    'train_label': 'train-labels-idx1-ubyte.gz',  # 훈련 라벨 파일 이름
    'test_img': 't10k-images-idx3-ubyte.gz',  # 테스트 이미지 파일 이름
    'test_label': 't10k-labels-idx1-ubyte.gz'  # 테스트 라벨 파일 이름
}

dataset_dir = os.path.dirname(os.path.abspath(__file__))  # 현재 파일의 디렉토리 경로
save_file = dataset_dir + "/mnist.pkl"  # 저장할 파일 경로

train_num = 60000  # 훈련 데이터 개수
test_num = 10000  # 테스트 데이터 개수
img_dim = (1, 28, 28)  # 이미지 차원
img_size = 784  # 이미지 크기

def _download(file_name):  # 파일을 다운로드하는 함수 정의
    file_path = dataset_dir + "/" + file_name  # 파일 경로 설정
```

```
    if os.path.exists(file_path):  # 파일이 이미 존재하면
        return  # 함수 종료

    print("Downloading " + file_name + " ... ")  # 다운로드 메시지 출력
    urllib.request.urlretrieve(url_base + file_name, file_path)  # 파일 다운로드
    print("Done")  # 다운로드 완료 메시지 출력

def download_mnist():  # MNIST 데이터셋을 다운로드하는 함수 정의
    for v in key_file.values():  # key_file 딕셔너리의 값들을 순회하며
       _download(v)  # 각 파일을 다운로드

def _load_label(file_name):  # 라벨 파일을 로드하는 함수 정의
    file_path = dataset_dir + "/" + file_name  # 파일 경로 설정

    print("Converting " + file_name + " to NumPy Array ...")  # 변환 메시지 출력
    with gzip.open(file_path, 'rb') as f:  # gzip 파일 열기
            labels = np.frombuffer(f.read(), np.uint8, offset=8)  # 라벨 데이터를 NumPy 배열로 변환
    print("Done")  # 변환 완료 메시지 출력

    return labels  # 라벨 데이터 반환

def _load_img(file_name):  # 이미지 파일을 로드하는 함수 정의
    file_path = dataset_dir + "/" + file_name  # 파일 경로 설정

    print("Converting " + file_name + " to NumPy Array ...")  # 변환 메시지 출력
    with gzip.open(file_path, 'rb') as f:  # gzip 파일 열기
            data = np.frombuffer(f.read(), np.uint8, offset=16)  # 이미지 데이터를 NumPy 배열로 변환
    data = data.reshape(-1, img_size)  # 데이터 모양 변경
    print("Done")  # 변환 완료 메시지 출력

    return data  # 이미지 데이터 반환

def _convert_numpy():  # 데이터를 NumPy 배열로 변환하는 함수 정의
    dataset = {}  # 데이터셋 딕셔너리 초기화
    dataset['train_img'] =  _load_img(key_file['train_img'])  # 훈련 이미지 데이터 로드
    dataset['train_label'] = _load_label(key_file['train_label'])  # 훈련 라벨 데이터 로드
    dataset['test_img'] = _load_img(key_file['test_img'])  # 테스트 이미지 데이터 로드
    dataset['test_label'] = _load_label(key_file['test_label'])  # 테스트 라벨 데이터 로드

    return dataset  # 데이터셋 반환

def init_mnist():  # MNIST 데이터셋을 초기화하는 함수 정의
    download_mnist()  # MNIST 데이터셋 다운로드
    dataset = _convert_numpy()  # 데이터를 NumPy 배열로 변환
    print("Creating pickle file ...")  # 피클 파일 생성 메시지 출력
    with open(save_file, 'wb') as f:  # 피클 파일 열기
        pickle.dump(dataset, f, -1)  # 데이터셋을 피클 파일에 저장
    print("Done!")  # 완료 메시지 출력

def _change_one_hot_label(X):  # 라벨을 원-핫 인코딩으로 변환하는 함수 정의
    T = np.zeros((X.size, 10))  # 원-핫 인코딩을 위한 배열 초기화
    for idx, row in enumerate(T):  # 각 라벨을 순회하며
        row[X[idx]] = 1  # 해당 위치에 1 설정

    return T  # 원-핫 인코딩 라벨 반환

def load_mnist(normalize=True, flatten=True, one_hot_label=False):  # MNIST 데이터셋을 로드하는 함수 정의
    """MNIST 데이터셋 읽기

    Parameters
    ----------
    normalize : 이미지의 픽셀 값을 0.0~1.0 사이의 값으로 정규화할지 정한다.
    one_hot_label :
        one_hot_label이 True면、레이블을 원-핫(one-hot) 배열로 돌려준다.
        one-hot 배열은 예를 들어 [0,0,1,0,0,0,0,0,0,0]처럼 한 원소만 1인 배열이다.
    flatten : 입력 이미지를 1차원 배열로 만들지를 정한다.

    Returns
    -------
    (훈련 이미지, 훈련 레이블), (시험 이미지, 시험 레이블)
    """
    if not os.path.exists(save_file):  # 피클 파일이 존재하지 않으면
        init_mnist()  # MNIST 데이터셋 초기화

    with open(save_file, 'rb') as f:  # 피클 파일 열기
        dataset = pickle.load(f)  # 데이터셋 로드
```

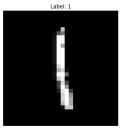
```
    if normalize:  # 정규화가 True이면
        for key in ('train_img', 'test_img'):  # 훈련 이미지와 테스트 이미지에 대해
            dataset[key] = dataset[key].astype(np.float32)  # 데이터 타입을 float32로 변환
            dataset[key] /= 255.0  # 255로 나누어 0.0~1.0 사이로 정규화

    if one_hot_label:  # 원-핫 인코딩이 True이면
        dataset['train_label'] = _change_one_hot_label(dataset['train_label'])  # 훈련 라벨 원-핫 인코딩
        dataset['test_label'] = _change_one_hot_label(dataset['test_label'])  # 테스트 라벨 원-핫 인코딩

    if not flatten:  # flatten이 False이면
         for key in ('train_img', 'test_img'):  # 훈련 이미지와 테스트 이미지에 대해
            dataset[key] = dataset[key].reshape(-1, 1, 28, 28)  # 이미지를 1x28x28로 reshape

    return (dataset['train_img'], dataset['train_label']), (dataset['test_img'], dataset['test_label'])  # 데이터셋 반환

if __name__ == '__main__':  # 메인 함수 실행
    init_mnist()  # MNIST 데이터셋 초기화
```

다음으로, `mnist_downloader.py` 파일을 사용하여 데이터를 다운로드하고 시각화하는 코드를 작성한다.

```
import os  # 운영 체제와 상호작용하기 위한 os 모듈 임포트
import numpy as np  # 수치 연산을 위한 numpy 모듈 임포트
import matplotlib.pyplot as plt  # 데이터 시각화를 위한 matplotlib 모듈의 pyplot 임포트
import pickle  # 데이터 직렬화 및 역직렬화를 위한 pickle 모듈 임포트

# mnist_downloader.py에서 필요한 함수들 import
from mnist_downloader import init_mnist, load_mnist  # mnist_downloader 모듈에서 init_mnist와 load_mnist 함수 임포트

# 데이터셋이 이미 다운로드되지 않았다면 다운로드
if not os.path.exists('mnist.pkl'):  # mnist.pkl 파일이 현재 디렉토리에 존재하지 않으면
    init_mnist()  # MNIST 데이터셋을 다운로드하고 초기화하는 함수 호출

# MNIST 데이터셋 로드
(x_train, y_train), (x_test, y_test) = load_mnist(normalize=False, flatten=False)  # 정규화하지 않고 MNIST 데이터셋 로드

# 무작위로 하나의 이미지를 선택
random_index = np.random.randint(0, len(x_train))  # 0에서 훈련 데이터의 길이 사이의 무작위 인덱스 생성
random_image = x_train[random_index].reshape(28, 28)  # 무작위로 선택된 이미지를 28x28 형태로 변형
random_label = y_train[random_index]  # 무작위로 선택된 이미지의 라벨

# 이미지 배열 출력
print("Selected image array:")  # 선택된 이미지 배열 출력
print(random_image)  # 이미지 배열 출력

# 이미지 시각화
plt.imshow(random_image, cmap='gray')  # 이미지를 회색조로 시각화
plt.title(f'Label: {random_label}')  # 이미지의 라벨을 제목으로 설정
plt.axis('off')  # 축 숨기기
plt.show()  # 이미지 표시

print(f"The label of the selected image is: {random_label}")  # 선택된 이미지의 라벨 출력
```

```
Selected image array:
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 194   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   8  85 120   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  22 252 244   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  22 252 252   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  22 252 164   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  22 253 253   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  22 252 252   0   0   0
    0   0   0   0   0   0   0   0   0   0]
```

```
 [  0   0   0   0   0   0   0   0   0   0   0   0  22 252 252   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  22 252 252   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  13 217 252   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  13 174 209  18   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  22 252 252 106   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  11 202 244   9   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  64 217  71   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  64 252 150   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  27 229 255  63   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 211 253 142   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 211 253 168   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 185 253 168   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  18 209 203   9
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
```



The label of the selected image is: 1

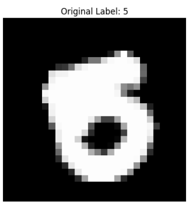
3.4.6. 0.0~1.0 의 값으로 normalization 하는 코드를 포함시켜서 3.4.5번 문제의 과정을 반복하시오. 또한 3.4.5번 문제의 plot 결과와 비교해보고, 그 결과에 대하여 고찰하시오.

정규화된 이미지와 정규화되지 않은 이미지의 차이점은 픽셀 값의 범위이다. 정규화된 이미지는 각 픽셀 값이 0.0에서 1.0 사이의 값으로 변환되어 있다. 이는 신경망 모델이 학습할 때 안정적인 학습을 돕고, 수렴 속도를 높일 수 있다. 반면, 정규화되지 않은 이미지는 각 픽셀 값이 0에서 255 사이의 값으로 유지된다. 정규화된 이미지는 시각적으로 큰 차이가 없지만, 수치적으로 보면 값의 범위가 다르므로 학습에 영향을 줄 수 있다.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import pickle

# mnist_downloader.py에서 필요한 함수들 import
from mnist_downloader import init_mnist, load_mnist

# 데이터셋이 이미 다운로드되지 않았다면 다운로드
if not os.path.exists('mnist.pkl'):
    init_mnist()

# MNIST 데이터셋 로드 (정규화 포함)
(x_train, y_train), (x_test, y_test) = load_mnist(normalize=True, flatten=False)

# 무작위로 하나의 이미지를 선택
random_index = np.random.randint(0, len(x_train))
random_image = x_train[random_index].reshape(28, 28)
random_label = y_train[random_index]

# 이미지 배열 출력
print("Selected image array (normalized):")
print(random_image)

# 이미지 시각화 (정규화된 데이터)
plt.imshow(random_image, cmap='gray')
```

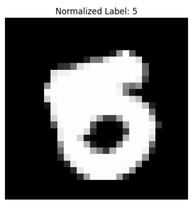
```
plt.title(f'Normalized Label: {random_label}')
plt.axis('off')
plt.show()

print(f"The label of the selected normalized image is: {random_label}")

# MNIST 데이터셋 로드 (정규화하지 않음)
(x_train_orig, y_train_orig), (x_test_orig, y_test_orig) = load_mnist(normalize=False, flatten=False)

# 동일한 인덱스의 원본 이미지를 선택
original_image = x_train_orig[random_index].reshape(28, 28)
original_label = y_train_orig[random_index]

# 원본 이미지 배열 출력
print("Selected image array (original):")
print(original_image)

# 원본 이미지 시각화
plt.imshow(original_image, cmap='gray')
plt.title(f'Original Label: {original_label}')
plt.axis('off')
plt.show()

print(f"The label of the selected original image is: {original_label}")
```

정규화 전




```
The label of the selected normalized image is: 5
Selected image array (original):
[[  0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0  26 147
  255  91   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0
37 118 118 215 249 253
  253 250 222   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  46  54 112 222
245 253 253 253 253 253
  253 253 252  99   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 224 242 244 253 253
253 253 253 253 253 253
  253 253 253 104   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 241 253 253 253 253
253 253 253 253 253 253
  253 253 243  64   0   0   0   0   0   0]
 [  0   0   0   0   0   0 150 252 253 253 253 253
253 253 253 253 253 237
  141 116 145   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0 248 253 253 253 253 253
253 253 253 253 253 246
  128  31   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0 208 253 253 253 253 253
```

정규화 후

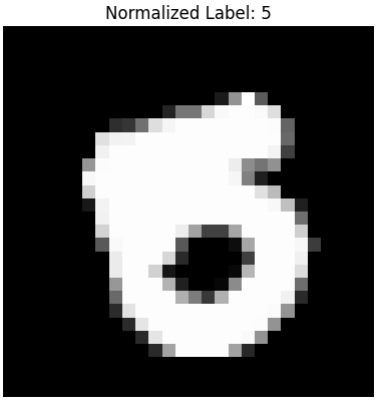



```
Selected image array (normalized):
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.10196079 0.576470
6
  1.         0.35686275 0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.14509805 0.4627451  0.4627451  0.84313726 0.9764706  0.992156
86
  0.99215686 0.98039216 0.87058824 0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
```

```
253 253 253 253 253 253
  253 225 187   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  24 244 253 253 253 253
253 253 253 253 253 253
  253 253 248 192  32   0   0   0   0   0]
 [  0   0   0   0   0   0   0 241 253 253 253 253
253 253 253 253 253 253
  253 253 253 253 110   0   0   0   0   0]
 [  0   0   0   0   0   0   0 211 253 253 253 253
253 237 161  65  65 165
  253 253 253 253 207   0   0   0   0   0]
 [  0   0   0   0   0   0   0  89 249 253 253 253
253  88   0   0   0  13
  166 253 253 253 247  60   0   0   0   0]
 [  0   0   0   0   0   0   0   0 235 253 253 253
215  42   0   0   0   0
   63 253 253 253 240   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 235 253 253 209
22   0   0   0   0   8
  182 253 253 253 220   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 151 253 253 253
179  33   0   0   7 139
  253 253 253 253 110   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 105 253 253 253
253 176 124  57 176 253
  253 253 253 241  42   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  43 228 253 253
253 253 253 253 253 253
  253 253 251 148   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  43 229 253
253 253 253 253 253 253
  253 252 148   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  42 229
253 253 253 253 253 253
  249 147   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  42
163 253 253 253 253 162
   41   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
```

```
  0.         0.         0.18039216 0.21176471 0.4392157  0.870588
24
  0.9607843  0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.9882353  0.3882353  0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.8784314  0.9490196  0.95686275 0.99215686 0.992156
86
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.99215686 0.40784314 0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.94509804 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.9529412  0.2509804  0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.5882353  0.9882353  0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.929411
77
  0.5529412  0.45490196 0.5686275  0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.972549   0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.964705
9
  0.5019608  0.12156863 0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.8156863  0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.88235295 0.73333335 0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.09411765 0.95686275 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.972549   0.7529412  0.1254902  0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.94509804 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.99215686 0.99215686 0.43137255 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.827451   0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.92941177 0.6313726  0.25490198 0.25490198 0.647058
84
  0.99215686 0.99215686 0.99215686 0.99215686 0.8117647  0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.34901962 0.9764706  0.99215686 0.99215686 0.992156
86
  0.99215686 0.34509805 0.         0.         0.         0.050980
39
  0.6509804  0.99215686 0.99215686 0.99215686 0.96862745 0.235294
12
  0.         0.         0.         0.         0.         0.
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.92156863 0.99215686 0.99215686 0.992156
86
  0.84313726 0.16470589 0.         0.         0.         0.
  0.24705882 0.99215686 0.99215686 0.99215686 0.9411765  0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
```

```
  0.         0.         0.92156863 0.99215686 0.99215686 0.819607
85
  0.08627451 0.         0.         0.         0.         0.031372
55
  0.7137255  0.99215686 0.99215686 0.99215686 0.8627451  0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.5921569  0.99215686 0.99215686 0.992156
86
  0.7019608  0.12941177 0.         0.         0.02745098 0.545098
07
  0.99215686 0.99215686 0.99215686 0.99215686 0.43137255 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.4117647  0.99215686 0.99215686 0.992156
86
  0.99215686 0.6901961  0.4862745  0.22352941 0.6901961  0.992156
86
  0.99215686 0.99215686 0.99215686 0.94509804 0.16470589 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.16862746 0.89411765 0.99215686 0.992156
86
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.99215686 0.9843137  0.5803922  0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.16862746 0.8980392  0.992156
86
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.99215686 0.9882353  0.5803922  0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.16470589 0.898039
2
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.992156
86
  0.9764706  0.5764706  0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.164705
89
  0.6392157  0.99215686 0.99215686 0.99215686 0.99215686 0.635294
14
  0.16078432 0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]]
```



번외 - 코드 설명 추가

정확도 출력 코드

```
import numpy as np  # numpy 모듈을 불러옴
import pandas as pd  # pandas 모듈을 불러옴
from sklearn.datasets import fetch_openml  # fetch_openml 모듈을 sklearn에서 불러옴
import pickle  # pickle 모듈을 불러옴

# MNIST 데이터 로드 함수
def load_mnist(normalize=True, flatten=True, one_hot_label=False):
    mnist = fetch_openml('mnist_784', version=1, parser='auto')  # 'mnist_784' 데이터를 버전 1로 불러옴, parser를 'auto'로 설정
    x = mnist.data.to_numpy()  # 데이터 부분을 numpy 배열로 변환하여 x에 저장
    t = mnist.target.astype(np.int64).to_numpy()  # 타겟 부분을 numpy 배열로 변환하고 정수형으로 변환
    if normalize:
        x = x / 255.0  # 정규화 (0-255 범위를 0-1 범위로 변환)
    if not flatten:
        x = x.reshape(-1, 1, 28, 28)  # 데이터를 28x28 크기로 변환
    return (x[:60000], t[:60000]), (x[60000:], t[60000:])  # 학습 데이터와 테스트 데이터로 나누어 반환

# 데이터 로드 함수
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, flatten=True, one_hot_label=False)  # MNIST 데이터를 로드
    return x_test, t_test  # 테스트 데이터와 레이블을 반환

# 네트워크 초기화 함수
def init_network():
    with open('sample_weight.pkl', 'rb') as f:  # 사전 학습된 가중치 파일을 염
        network = pickle.load(f)  # 파일 내용을 로드하여 네트워크에 저장
    return network  # 초기화된 네트워크 반환

# 예측 함수
def predict(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']  # 네트워크에서 가중치 가져오기
    b1, b2, b3 = network['b1'], network['b2'], network['b3']  # 네트워크에서 바이어스 가져오기

    a1 = np.dot(x, W1) + b1  # 첫 번째 층의 가중합 계산
    z1 = 1 / (1 + np.exp(-a1))  # 첫 번째 층의 활성화 함수(sigmoid) 적용
    a2 = np.dot(z1, W2) + b2  # 두 번째 층의 가중합 계산
    z2 = 1 / (1 + np.exp(-a2))  # 두 번째 층의 활성화 함수(sigmoid) 적용
    a3 = np.dot(z2, W3) + b3  # 세 번째 층의 가중합 계산
    y = np.exp(a3) / np.sum(np.exp(a3))  # 세 번째 층의 활성화 함수(softmax) 적용

    return y  # 예측 결과 반환

# 데이터 로드 및 네트워크 초기화
x, t = get_data()  # 테스트 데이터와 레이블을 로드
network = init_network()  # 신경망 초기화

# 추론 및 정확도 계산
accuracy_cnt = 0  # 정확도 카운트 초기화
for i in range(len(x)):  # 테스트 데이터의 각 샘플에 대해 반복
    y = predict(network, x[i])  # 예측 수행
    p = np.argmax(y)  # 가장 높은 확률을 가지는 클래스 인덱스 찾기
    if p == t[i]:  # 예측이 실제 레이블과 같으면
        accuracy_cnt += 1  # 정확도 카운트 증가

# 정확도 출력
print("Accuracy:" + str(float(accuracy_cnt) / len(x)))  # 정확도 계산 및 출력
```

```
'''
Accuracy:0.9352
'''
```

neuralnet_mnist.py

```
import sys, os  # 시스템 및 운영체제 관련 모듈 불러오기
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np  # numpy 모듈 불러오기
import pickle  # pickle 모듈 불러오기
from dataset.mnist import load_mnist  # dataset 모듈에서 mnist 데이터 로드 함수 불러오기
from common.functions import sigmoid, softmax  # common.functions 모듈에서 sigmoid와 softmax 함수 불러오기

# 데이터 로드 함수 정의
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, flatten=True, one_hot_label=False)  # MNIST 데이터셋을 로
    return x_test, t_test  # 테스트 데이터와 테스트 레이블 반환

# 네트워크 초기화 함수 정의
def init_network():
    with open("sample_weight.pkl", 'rb') as f:  # 'sample_weight.pkl' 파일을 바이너리 읽기 모드로 열기
        network = pickle.load(f)  # 파일 내용을 로드하여 network 변수에 저장
    return network  # 초기화된 네트워크 반환

# 예측 함수 정의
def predict(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']  # 네트워크에서 가중치 가져오기
    b1, b2, b3 = network['b1'], network['b2'], network['b3']  # 네트워크에서 바이어스 가져오기

    a1 = np.dot(x, W1) + b1  # 첫 번째 층의 가중합 계산
    z1 = sigmoid(a1)  # 첫 번째 층의 활성화 함수(sigmoid) 적용
    a2 = np.dot(z1, W2) + b2  # 두 번째 층의 가중합 계산
    z2 = sigmoid(a2)  # 두 번째 층의 활성화 함수(sigmoid) 적용
    a3 = np.dot(z2, W3) + b3  # 세 번째 층의 가중합 계산
    y = softmax(a3)  # 세 번째 층의 활성화 함수(softmax) 적용

    return y  # 예측 결과 반환

# 데이터 로드 및 네트워크 초기화
x, t = get_data()  # 테스트 데이터와 테스트 레이블 로드
network = init_network()  # 네트워크 초기화

# 추론 및 정확도 계산
accuracy_cnt = 0  # 정확도 카운트 초기화
for i in range(len(x)):  # 테스트 데이터의 각 샘플에 대해 반복
    y = predict(network, x[i])  # 예측 수행
    p = np.argmax(y)  # 확률이 가장 높은 원소의 인덱스를 얻음
    if p == t[i]:  # 예측이 실제 레이블과 같으면
        accuracy_cnt += 1  # 정확도 카운트 증가

# 정확도 출력
print("Accuracy:" + str(float(accuracy_cnt) / len(x)))  # 정확도 계산 및 출력
```

mnist.py

```
try:
    import urllib.request  # urllib.request 모듈을 불러옴
except ImportError:
    raise ImportError('You should use Python 3.x')  # Python 3.x를 사용해야 한다는 오류 발생
import os.path  # os.path 모듈을 불러옴
import gzip  # gzip 모듈을 불러옴
import pickle  # pickle 모듈을 불러옴
import os  # os 모듈을 불러옴
import numpy as np  # numpy 모듈을 불러옴

# MNIST 데이터셋의 URL과 파일 이름을 매핑
url_base = 'http://yann.lecun.com/exdb/mnist/'
key_file = {
    'train_img': 'train-images-idx3-ubyte.gz',  # 학습 이미지 파일 이름
    'train_label': 'train-labels-idx1-ubyte.gz',  # 학습 레이블 파일 이름
    'test_img': 't10k-images-idx3-ubyte.gz',  # 테스트 이미지 파일 이름
    'test_label': 't10k-labels-idx1-ubyte.gz'  # 테스트 레이블 파일 이름
}

dataset_dir = os.path.dirname(os.path.abspath(__file__))  # 현재 파일의 디렉터리 경로
```

```
save_file = dataset_dir + "/mnist.pkl"  # 저장할 파일 경로 설정

train_num = 60000  # 학습 데이터 개수
test_num = 10000  # 테스트 데이터 개수
img_dim = (1, 28, 28)  # 이미지 차원 (채널, 높이, 너비)
img_size = 784  # 이미지 크기 (28x28)

# 파일 다운로드 함수 정의
def _download(file_name):
    file_path = dataset_dir + "/" + file_name  # 파일 경로 설정

    if os.path.exists(file_path):  # 파일이 이미 존재하는 경우
        return  # 다운로드를 생략하고 함수 종료

    print("Downloading " + file_name + " ... ")  # 다운로드 시작 메시지 출력
    urllib.request.urlretrieve(url_base + file_name, file_path)  # 파일 다운로드
    print("Done")  # 다운로드 완료 메시지 출력

# MNIST 데이터셋 다운로드 함수 정의
def download_mnist():
    for v in key_file.values():  # key_file 딕셔너리의 값들에 대해 반복
        _download(v)  # 각 파일을 다운로드

# 레이블 파일 로드 함수 정의
def _load_label(file_name):
    file_path = dataset_dir + "/" + file_name  # 파일 경로 설정

    print("Converting " + file_name + " to NumPy Array ...")  # 변환 시작 메시지 출력
    with gzip.open(file_path, 'rb') as f:  # gzip 파일 열기
        labels = np.frombuffer(f.read(), np.uint8, offset=8)  # 파일 내용을 읽어 uint8 배열로 변환
    print("Done")  # 변환 완료 메시지 출력

    return labels  # 변환된 레이블 반환

# 이미지 파일 로드 함수 정의
def _load_img(file_name):
    file_path = dataset_dir + "/" + file_name  # 파일 경로 설정

    print("Converting " + file_name + " to NumPy Array ...")  # 변환 시작 메시지 출력
    with gzip.open(file_path, 'rb') as f:  # gzip 파일 열기
        data = np.frombuffer(f.read(), np.uint8, offset=16)  # 파일 내용을 읽어 uint8 배열로 변환
    data = data.reshape(-1, img_size)  # 데이터를 (N, 784) 형태로 변환
    print("Done")  # 변환 완료 메시지 출력

    return data  # 변환된 이미지 데이터 반환

# 데이터를 NumPy 배열로 변환하는 함수 정의
def _convert_numpy():
    dataset = {}  # 데이터셋 딕셔너리 초기화
    dataset['train_img'] = _load_img(key_file['train_img'])  # 학습 이미지 로드
    dataset['train_label'] = _load_label(key_file['train_label'])  # 학습 레이블 로드
    dataset['test_img'] = _load_img(key_file['test_img'])  # 테스트 이미지 로드
    dataset['test_label'] = _load_label(key_file['test_label'])  # 테스트 레이블 로드

    return dataset  # 변환된 데이터셋 반환

# MNIST 데이터셋 초기화 함수 정의
def init_mnist():
    download_mnist()  # MNIST 데이터셋 다운로드
    dataset = _convert_numpy()  # 데이터를 NumPy 배열로 변환
    print("Creating pickle file ...")  # 피클 파일 생성 시작 메시지 출력
    with open(save_file, 'wb') as f:  # 피클 파일을 바이너리 쓰기 모드로 열기
        pickle.dump(dataset, f, -1)  # 데이터셋을 피클 파일에 저장
    print("Done!")  # 완료 메시지 출력

# 레이블을 원-핫 벡터로 변환하는 함수 정의
def _change_one_hot_label(X):
    T = np.zeros((X.size, 10))  # (N, 10) 형태의 영행렬 생성
    for idx, row in enumerate(T):  # 각 레이블에 대해 반복
        row[X[idx]] = 1  # 해당 레이블 위치에 1을 설정

    return T  # 변환된 원-핫 레이블 반환

# MNIST 데이터셋을 로드하는 함수 정의
def load_mnist(normalize=True, flatten=True, one_hot_label=False):
    """MNIST 데이터셋 읽기
```

```
    Parameters
    ----------
    normalize : 이미지의 픽셀 값을 0.0~1.0 사이의 값으로 정규화할지 정한다.
    one_hot_label :
        one_hot_label이 True면, 레이블을 원-핫(one-hot) 배열로 돌려준다.
        원-핫 배열은 예를 들어 [0,0,1,0,0,0,0,0,0,0]처럼 한 원소만 1인 배열이다.
    flatten : 입력 이미지를 1차원 배열로 만들지를 정한다.

    Returns
    -------
    (훈련 이미지, 훈련 레이블), (시험 이미지, 시험 레이블)
    """
    if not os.path.exists(save_file):  # 피클 파일이 존재하지 않는 경우
        init_mnist()  # MNIST 데이터셋 초기화

    with open(save_file, 'rb') as f:  # 피클 파일을 바이너리 읽기 모드로 열기
        dataset = pickle.load(f)  # 피클 파일에서 데이터셋 로드

    if normalize:
        for key in ('train_img', 'test_img'):  # 학습 이미지와 테스트 이미지에 대해 반복
            dataset[key] = dataset[key].astype(np.float32)  # 데이터를 float32 형으로 변환
            dataset[key] /= 255.0  # 정규화 (0-255 범위를 0-1 범위로 변환)

    if one_hot_label:
        dataset['train_label'] = _change_one_hot_label(dataset['train_label'])  # 학습 레이블을 원-핫 벡터로 변환
        dataset['test_label'] = _change_one_hot_label(dataset['test_label'])  # 테스트 레이블을 원-핫 벡터로 변환

    if not flatten:
         for key in ('train_img', 'test_img'):  # 학습 이미지와 테스트 이미지에 대해 반복
            dataset[key] = dataset[key].reshape(-1, 1, 28, 28)  # 데이터를 (N, 1, 28, 28) 형태로 변환

    return (dataset['train_img'], dataset['train_label']), (dataset['test_img'], dataset['test_label'])  # 학습 데이터와 테스트

# 스크립트가 직접 실행될 때만 수행
if __name__ == '__main__':
    init_mnist()  # MNIST 데이터셋 초기화
```

mnist_show.py

```
import sys, os  # 시스템 및 운영체제 관련 모듈 불러오기
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np  # numpy 모듈 불러오기
from dataset.mnist import load_mnist  # dataset 모듈에서 mnist 데이터 로드 함수 불러오기
from PIL import Image  # PIL(Pillow) 모듈에서 Image 클래스 불러오기

# 이미지 보여주기 함수 정의
def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))  # numpy 배열을 이미지 객체로 변환
    pil_img.show()  # 이미지 표시

# MNIST 데이터셋 로드
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)  # MNIST 데이터를 평탄화하지 않고 정규화 없이 로드

img = x_train[0]  # 첫 번째 학습 이미지 데이터 가져오기
label = t_train[0]  # 첫 번째 학습 레이블 가져오기
print(label)  # 레이블 출력 (예: 5)

print(img.shape)  # 이미지 데이터의 형상 출력 (784,)
img = img.reshape(28, 28)  # 형상을 원래 이미지의 크기(28x28)로 변형
print(img.shape)  # 변형된 이미지 데이터의 형상 출력 (28, 28)

img_show(img)  # 이미지를 화면에 표시
```

neuralnet_mnist_batch.py

```
import sys, os  # 시스템 및 운영체제 관련 모듈 불러오기
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np  # numpy 모듈 불러오기
import pickle  # pickle 모듈 불러오기
from dataset.mnist import load_mnist  # dataset 모듈에서 mnist 데이터 로드 함수 불러오기
from common.functions import sigmoid, softmax  # common.functions 모듈에서 sigmoid와 softmax 함수 불러오기

# 데이터 로드 함수 정의
def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, flatten=True, one_hot_label=False)  # MNIST 데이터셋을 로
    return x_test, t_test  # 테스트 데이터와 테스트 레이블 반환
```

```
# 네트워크 초기화 함수 정의
def init_network():
    with open("sample_weight.pkl", 'rb') as f:  # 'sample_weight.pkl' 파일을 바이너리 읽기 모드로 열기
        network = pickle.load(f)  # 파일 내용을 로드하여 network 변수에 저장
    return network  # 초기화된 네트워크 반환

# 예측 함수 정의
def predict(network, x):
    w1, w2, w3 = network['W1'], network['W2'], network['W3']  # 네트워크에서 가중치 가져오기
    b1, b2, b3 = network['b1'], network['b2'], network['b3']  # 네트워크에서 바이어스 가져오기

    a1 = np.dot(x, w1) + b1  # 첫 번째 층의 가중합 계산
    z1 = sigmoid(a1)  # 첫 번째 층의 활성화 함수(sigmoid) 적용
    a2 = np.dot(z1, w2) + b2  # 두 번째 층의 가중합 계산
    z2 = sigmoid(a2)  # 두 번째 층의 활성화 함수(sigmoid) 적용
    a3 = np.dot(z2, w3) + b3  # 세 번째 층의 가중합 계산
    y = softmax(a3)  # 세 번째 층의 활성화 함수(softmax) 적용

    return y  # 예측 결과 반환

# 데이터 로드 및 네트워크 초기화
x, t = get_data()  # 테스트 데이터와 테스트 레이블 로드
network = init_network()  # 네트워크 초기화

batch_size = 100  # 배치 크기 설정
accuracy_cnt = 0  # 정확도 카운트 초기화

# 배치 단위로 예측 수행
for i in range(0, len(x), batch_size):
    x_batch = x[i:i+batch_size]  # 배치 단위로 데이터 분할
    y_batch = predict(network, x_batch)  # 배치 데이터에 대해 예측 수행
    p = np.argmax(y_batch, axis=1)  # 각 배치에 대한 예측 결과에서 가장 높은 확률을 가지는 클래스 인덱스 찾기
    accuracy_cnt += np.sum(p == t[i:i+batch_size])  # 배치 내에서 예측이 실제 레이블과 일치하는 경우를 카운트하여 정확도 계산

# 최종 정확도 출력
print("Accuracy:" + str(float(accuracy_cnt) / len(x)))  # 정확도 계산 및 출력
```

sample_weight.pkl

sample_weight.pkl