

Week 3. 신경망 구조실습 (3.6 손글씨숫자인식)

21기 분석 이건하



3.6 목차

- MNIST 데이터셋
 - 데이터셋 구조
 - 이론설명
 - 코드실습
- 배치(batch) 처리
 - 다음 층으로 전달되는 구조
 - 배치처리에 따른 장단점
 - 실제 적용을 통한 점검

MNIST 데이터셋

- MNIST 데이터셋
 - 60,000개의 훈련 이미지와 10,000개의 테스트 이미지로 구성
 - 오늘 실습에서는 '순전파' 과정만 다루기 때문에 테스트 이미지만을 사용
 - 사전 훈련된 가중치를 신경망을 통해 전달하는 과정에 대한 실습
 - MNIST 데이터셋 개별 이미지(입력 행) 구성
 - 28*28 크기의 단일채널 이미지 (0~255의 값을 갖는 픽셀)
 - 0~9까지의 숫자 이미지 (검정배경 흰색 숫자)

```
network['w1']  
✓ 0.0s  
array([[ -0.00741249, -0.00790439, -0.01307499, ...,  0.01978721,  
        -0.04331266, -0.01350104],  
       [ -0.01029745, -0.01616653, -0.01228376, ...,  0.01920228,  
         0.02809811,  0.01450908],  
       [ -0.01309184, -0.00244747, -0.0177224 , ...,  0.00944778,  
         0.01387301,  0.03393568],  
       ...,  
       [  0.02242565, -0.0296145 , -0.06326169, ..., -0.01012643,  
         0.01120969,  0.01027199],  
       [ -0.00761533,  0.02028973, -0.01498873, ...,  0.02735376,  
        -0.01229855,  0.02407041],  
       [  0.00027915, -0.06848375,  0.00911191, ..., -0.03183098,  
         0.00743086, -0.04021148]], dtype=float32)
```



MNIST 데이터셋

- 데이터 불러오기
 - 이미지의 형태가 784인 이유는 28×28 픽셀이 flatten

```
#import sys,os
#sys.path.append(os.pardir)
from dataset.mnist import load_mnist
```

```
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)
```

```
print(x_train.shape)
print(t_train.shape)
print(x_test.shape)
print(t_test.shape)
```

✓ 0.0s

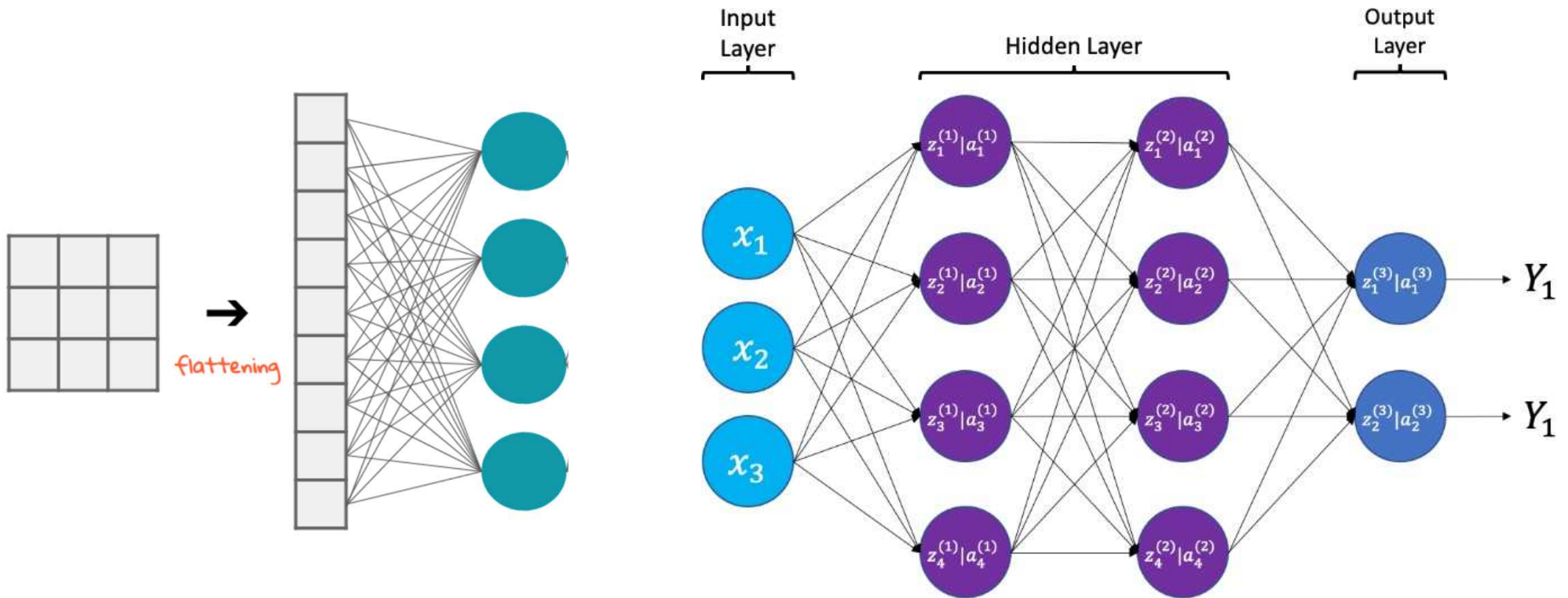
Python

```
(60000, 784)
(60000,)
(10000, 784)
(10000,)
```

```
url_base = 'http://yann.lecun.com/exdb/mnist/'
key_file = {
    'train_img': 'train-images-idx3-ubyte.gz',
    'train_label': 'train-labels-idx1-ubyte.gz',
    'test_img': 't10k-images-idx3-ubyte.gz',
    'test_label': 't10k-labels-idx1-ubyte.gz'
}
```

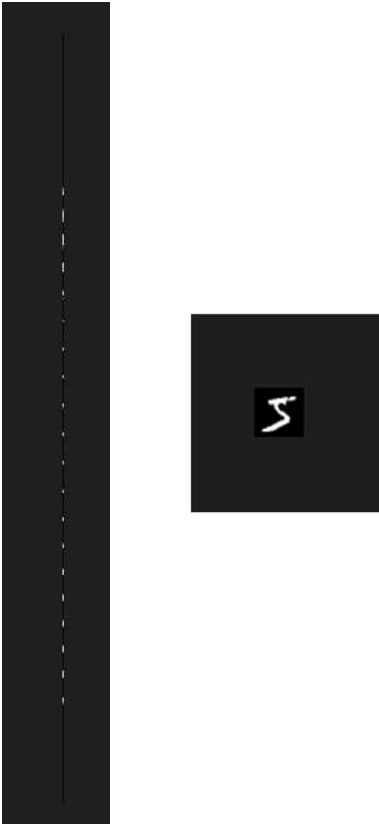
Flatten 설명

- 인공 신경망(Artificial Neural Network)의 구조



MNIST 데이터셋

- 샘플 이미지 출력
 - x_train데이터의 0번 인덱스 이미지



```
import sys, os
sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist
from PIL import Image

def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)

img = x_train[0]
label = t_train[0]
print(label) # 5

print(img.shape) # (784,)
img = img.reshape(28, 28) # 형상을 원래 이미지의 크기로 변형
print(img.shape) # (28, 28)

img_show(img)
```

✓ 4.1s

Python

```
5
(784,)
(28, 28)
```

[1] 밑바닥부터 시작하는 딥러닝 github

• 신경망 구현

- 신경망 구현

- 테스트 이미지만 사용

- 사전 훈련된 가중치(W,b)를 sample_weight.pkl 파일로부터 불러옴
 - predict과정은 순전파 과정으로, y는 최종 예측 클래스의 확률값 행렬

```
def get_data():
    (x_train, t_train), (x_test, t_test) = \
        load_mnist(flatten=True, normalize=True, one_hot_label=False)
    return x_test, t_test
```

```
def init_network():
    with open("sample_weight.pkl", 'rb') as f:
        # 학습된 가중치 매개변수가 담긴 파일
        # 학습 없이 바로 추론을 수행
        network = pickle.load(f)
    return network
```

```
def predict(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 = softmax(a3) #없이해도 결과 동일
    return y
```

신경망 구현

- 성능측정 결과

- 성능 측정 결과

- predict 함수를 통해 예측한 값과 ttest의 정답 라벨 값 비교
 - 비교결과 0.9352의 정확도

```
y
✓ 0.0s
array([4.2882893e-04, 2.0043037e-06, 2.5405698e-03, 2.0168918e-06,
       5.5917783e-04, 3.1262074e-04, 9.9614763e-01, 4.3499412e-07,
       6.3756929e-06, 3.7751445e-07], dtype=float32)
```

```
# 배치 전
for i in range(len(x)):
    y = predict(network, x[i])
    p = np.argmax(y) # 확률이 가장 높은 원소의 인덱스를 얻는다.(마지막 softmax된 확률 중)
    if p == t[i]:
        accuracy_cnt += 1

print("Accuracy:" + str(float(accuracy_cnt) / len(x))) # Accuracy:0.9352
✓ 0.4s
Accuracy:0.9352
```


신경망 구현

- 신경망 각 층의 배열 형상

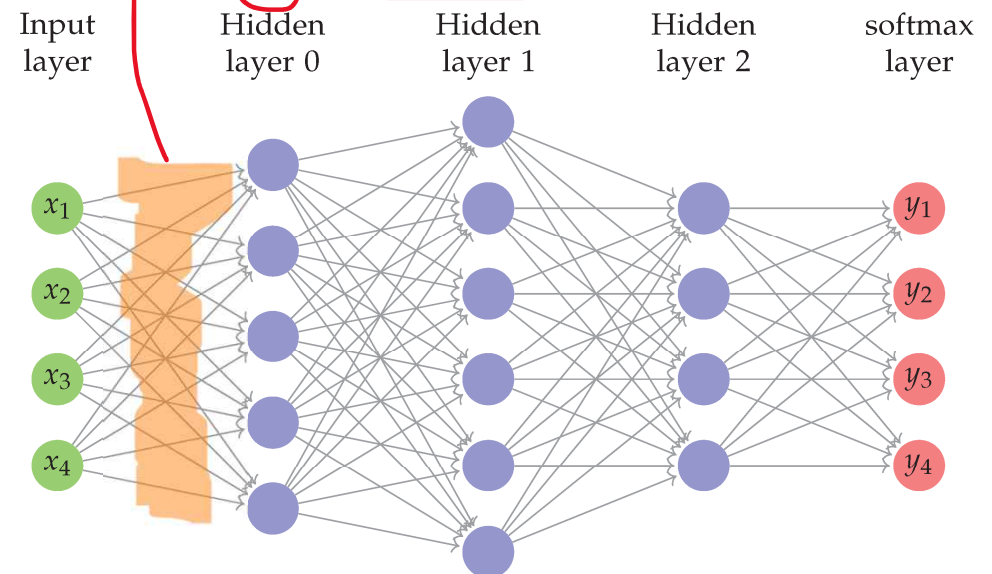
- 원소(node) 개수 규칙

- Input Layer는 입력 이미지의 크기와 일치
 - Output Layer는 클래스 수와 일치

```
print(len(network['w1']))
print(len(network['b1'])) # bias의 개수 = 출력값의 개수
print(len(network['w2']))
print(len(network['b2'])) # b2의 개수 = w3의 개수
print(len(network['w3']))
print(len(network['b3'])) # 최종 b의 개수 = class의 개수
✓ 0.0s
784
50
50
100
100
10
```

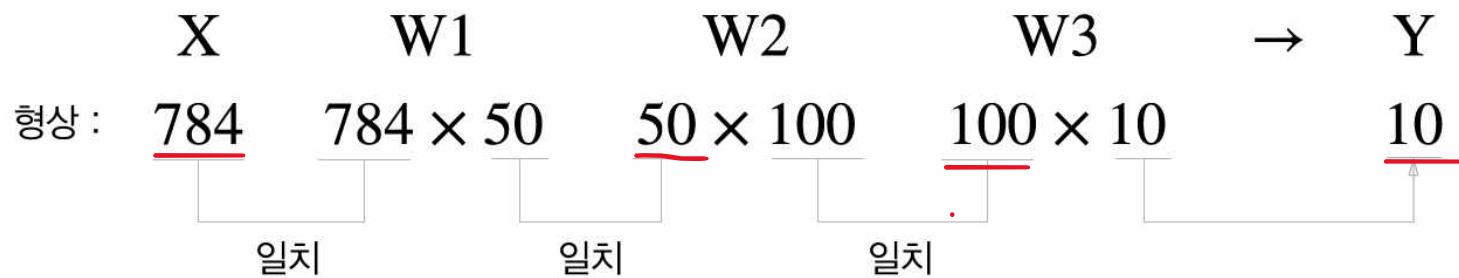
- 각 층별 가중치의 개수 규칙

- Weight는 앞 노드와 일치
 - Bias는 뒤 노드와 일치

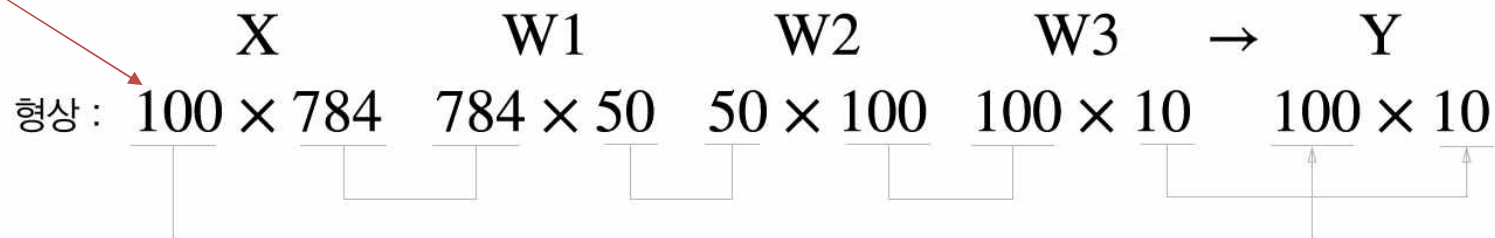


배치(batch) 처리

- 배치(batch) 처리
 - 컴퓨터에서는 큰 배열을 한꺼번에 계산하는 것이 분할된 작은 배열을 여러 번 계산하는 것보다 빠르다.

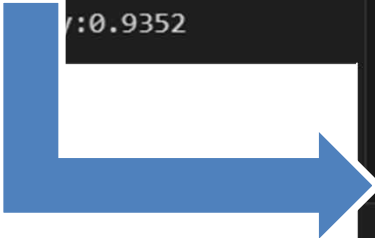


배치사이즈



배치(batch) 처리

- 배치(batch) 처리
 - 컴퓨터에서는 큰 배열을 한꺼번에 계산하는 것이 분할된 작은 배열을 여러 번 계산하는 것보다 빠르다.
(주의: 다만 실제 환경에서는 GPU를 통한 병렬처리가 가능하므로, 전체가 아닌 미니배치를 주로 사용)



```
# 배치 전
for i in range(len(x)):
    y = predict(network, x[i])
    p = np.argmax(y)
    if p == t[i]:
        accuracy_cnt += 1

print("Accuracy:" + str(accuracy_cnt / len(x)))

✓ 0.4s
Accuracy:0.9352
```

```
# 3.6.3 배치 처리
x, t = get_data()
network = init_network()
accuracy_cnt = 0 #맞힌 개수

batch_size = 100

for i in range(0, len(x), batch_size):
    x_batch = x[i:i+batch_size]
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis=1) #마지막 10개를 softmax통과시킨 것들 중 최대값(확률)
    accuracy_cnt += np.sum(p == t[i:i+batch_size]) #제일높아보이는 확률과, 라벨값이 일치하는 개수

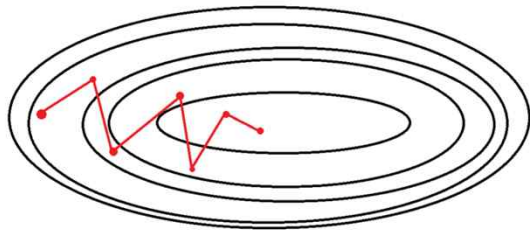
print("Accuracy:" + str(float(accuracy_cnt) / len(x))) # Accuracy:0.9352

✓ 0.1s
Accuracy:0.9352
```

배치(batch) 처리

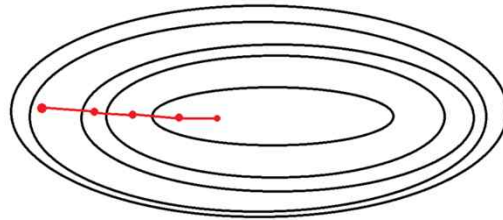
- 배치(batch) 처리 비교

입력데이터가 1개(SGD)



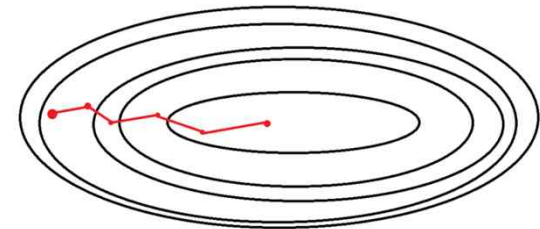
성능

입력데이터가 하나의 배치(BGD)



속도

입력데이터가 여러 개의 배치(미니배치)



합리적

Any Questions?

