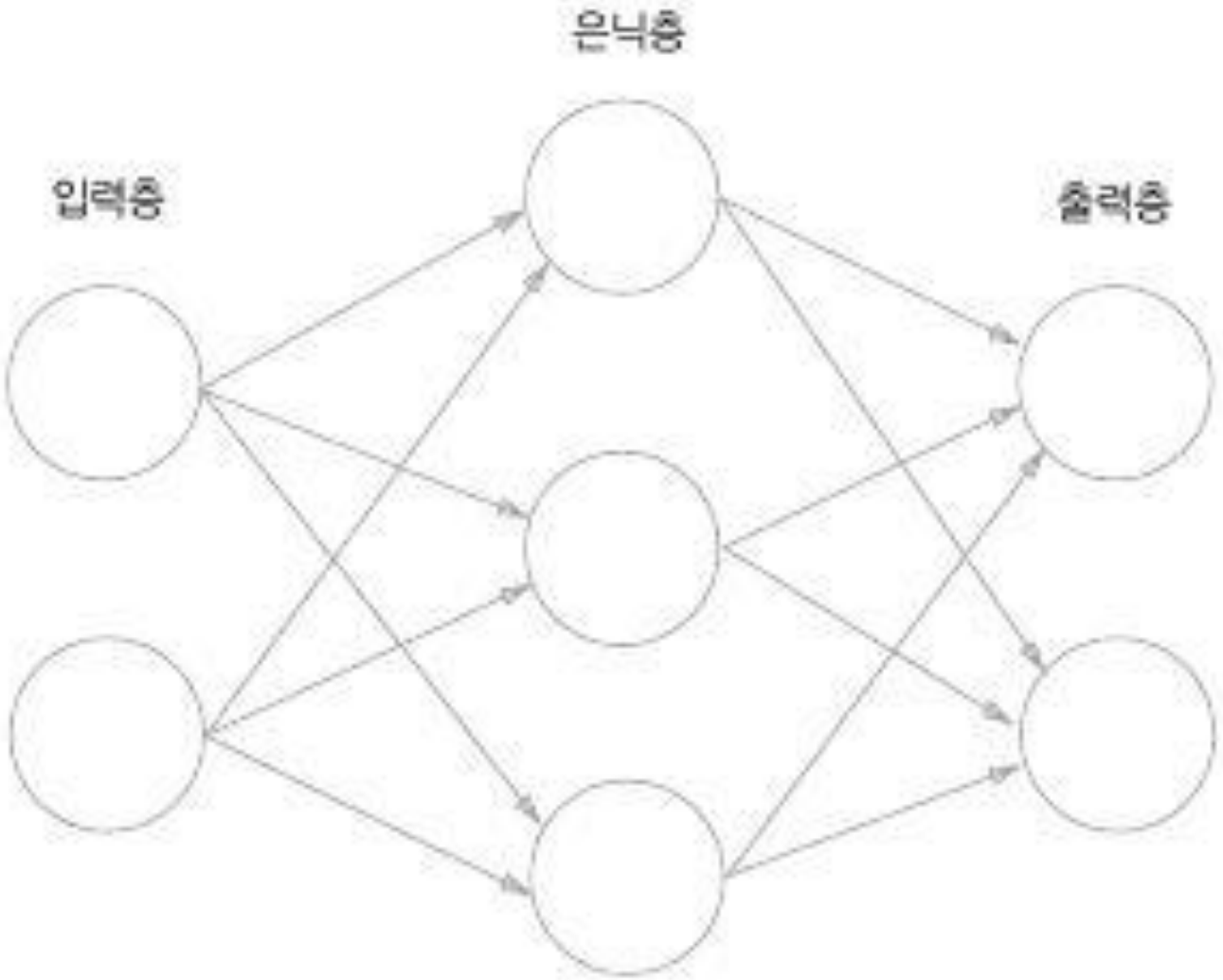
딥러닝 세션
1차 - 3장

Neural Network(신경망)

3.1 퍼셉트론과 신경망

가중치 기준 – 2층 신경망
층 수 기준 – 3층 신경망

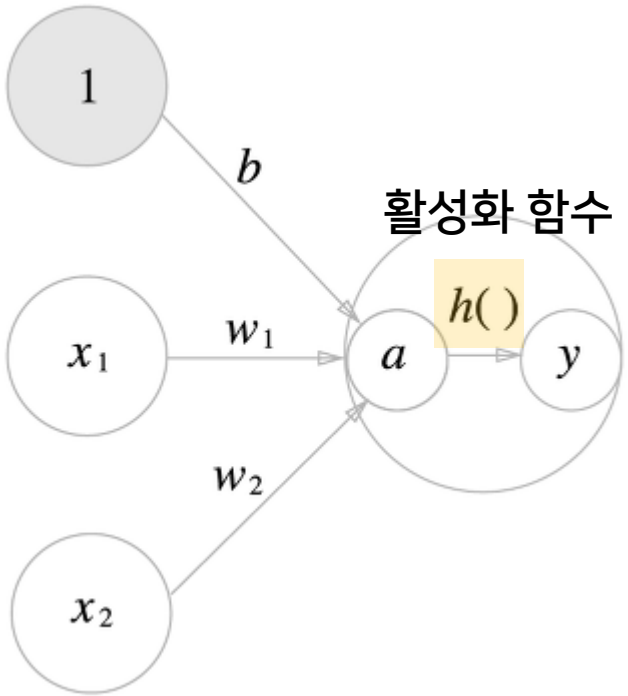
그림 3-1 신경망의 예



3.2 활성화 함수

활성화 함수 – 입력 신호의 합(sum)을 출력 신호로 변환하는 함수 (임계값 경계로 출력)

1. Weight가 달린 입력 신호와 편향의 총합을 구한다
2. 이 총합을 활성화 함수인 h(x)에 넣어 출력 신호를 구한다.

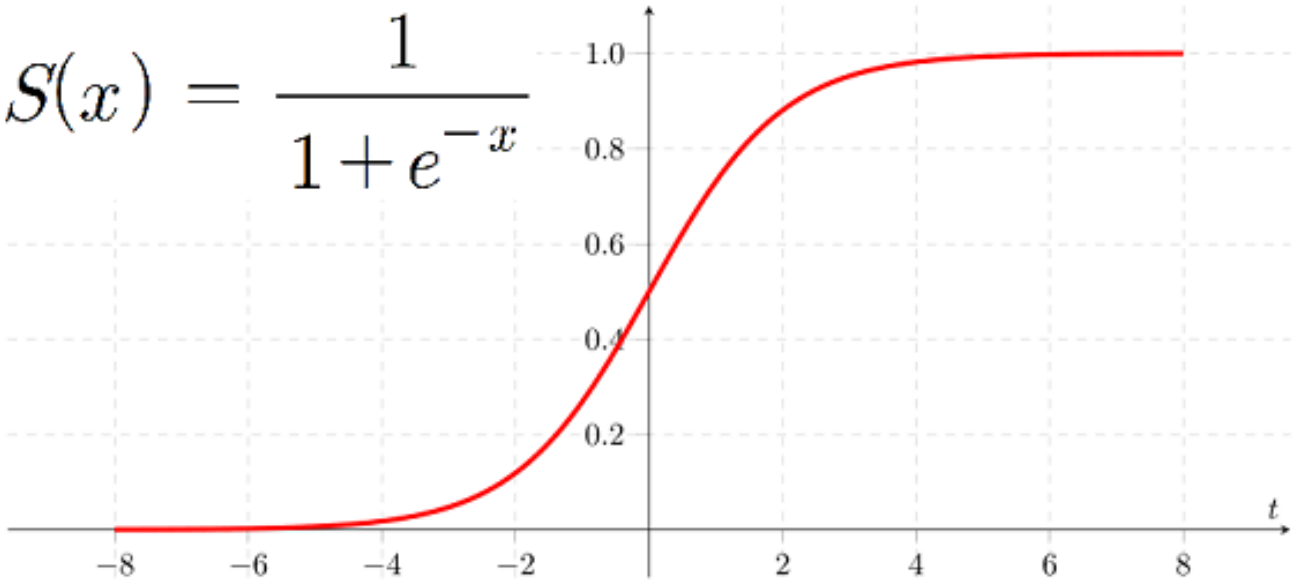


3.2.1 시그모이드 함수

- 부드러운 곡선이고 입력에 따라 출력이 연속적으로 변화
- 출력 신호로 0과 1사이의 실수 값을 반환

```
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()
```

$$S(x) = \frac{1}{1+e^{-x}}$$



3.2.2 계단 함수

- 단순히 입력이 0을 넘으면 1을 출력하고, 그 외에는 0을 출력한다.
- 0을 경계로 출력이 0에서 1으로 변화
- 0 또는 1중 하나의 값만 반환

```
def step_function(x):
    return np.array(x > 0, dtype=np.int)

x = np.arange(-5.0, 5.0, 0.1)
y = step_function(x)
plt.plot(x, y)
plt.ylim(-0.1, 1.1)
plt.show()
```

공학에서 유용한 함수로는 단위 계단 함수가 있습니다.

$$u(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

그래프는 x=0에서 위로 점프를 하지요.

3.2.6 비선형 함수

- 선형 함수 : 출력이 입력의 상수 배 만큼 변하는 함수 f(x) = ax + b
- 비선형 함수 : 선형이 아닌 함수, 즉 직선 1개로는 그릴 수 없는 함수 (계단 함수, 시그모이드 함수)

- 신경망에서는 활성화 함수로 비선형 함수 사용!

-> 선형 함수를 이용하면 신경망의 층을 깊게 하는 의미가 없음.

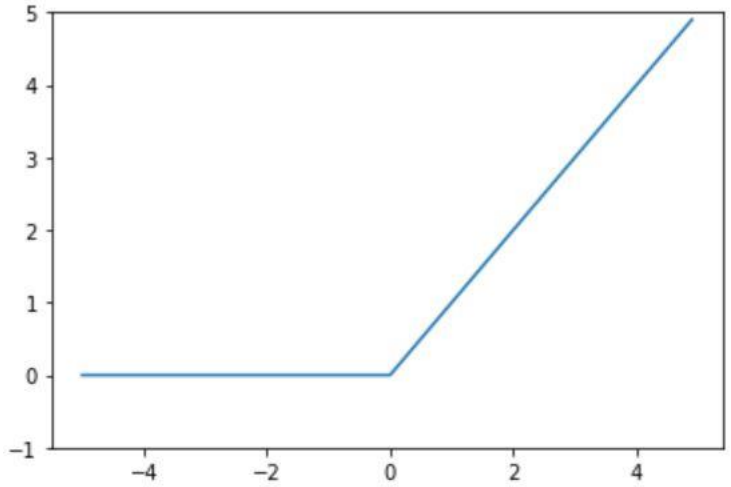
Ex) h(x) = cx라는 활성화 함수가 있다고 하면, 3개의 층을 쌓았을 때 출력은 y = h(h(h(x))) = c3x 이다.

C3 = a 로 치환을 하면 결국에는 1개의 층을 쌓은 신경망과 똑같아진다.

ReLU 함수

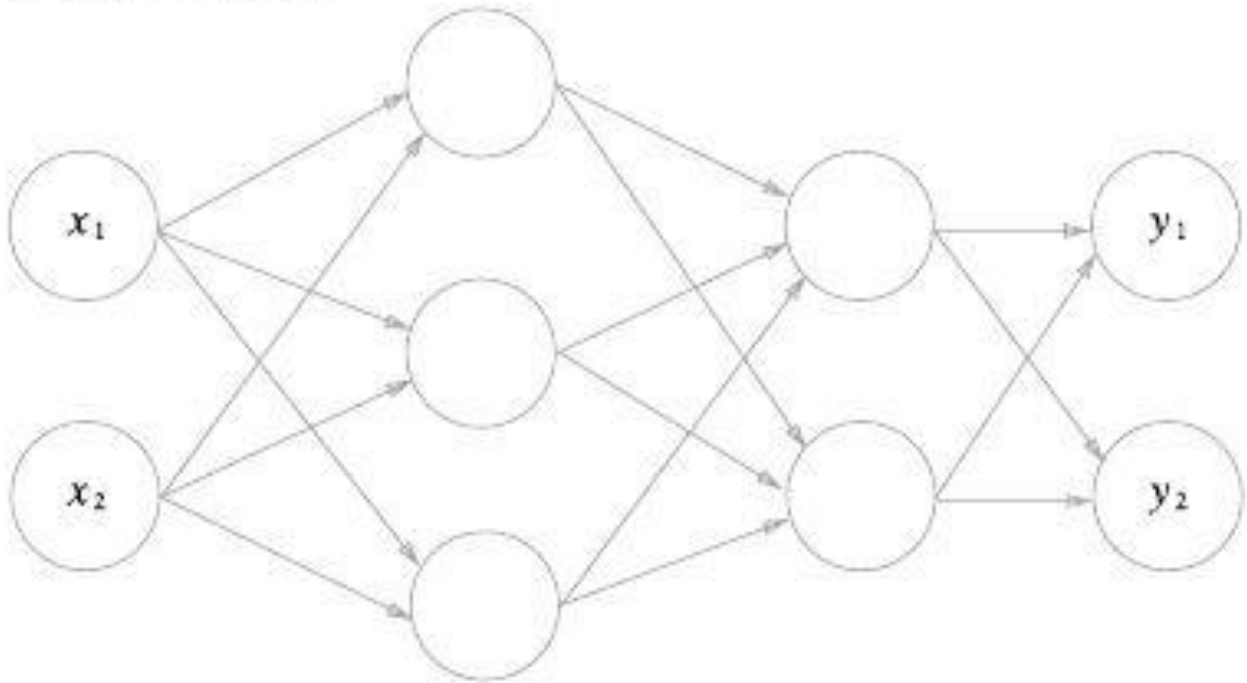
- 입력이 0을 넘으면 그 입력을 그대로 출력하고, 0 이하면 0을 출력

$$h(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

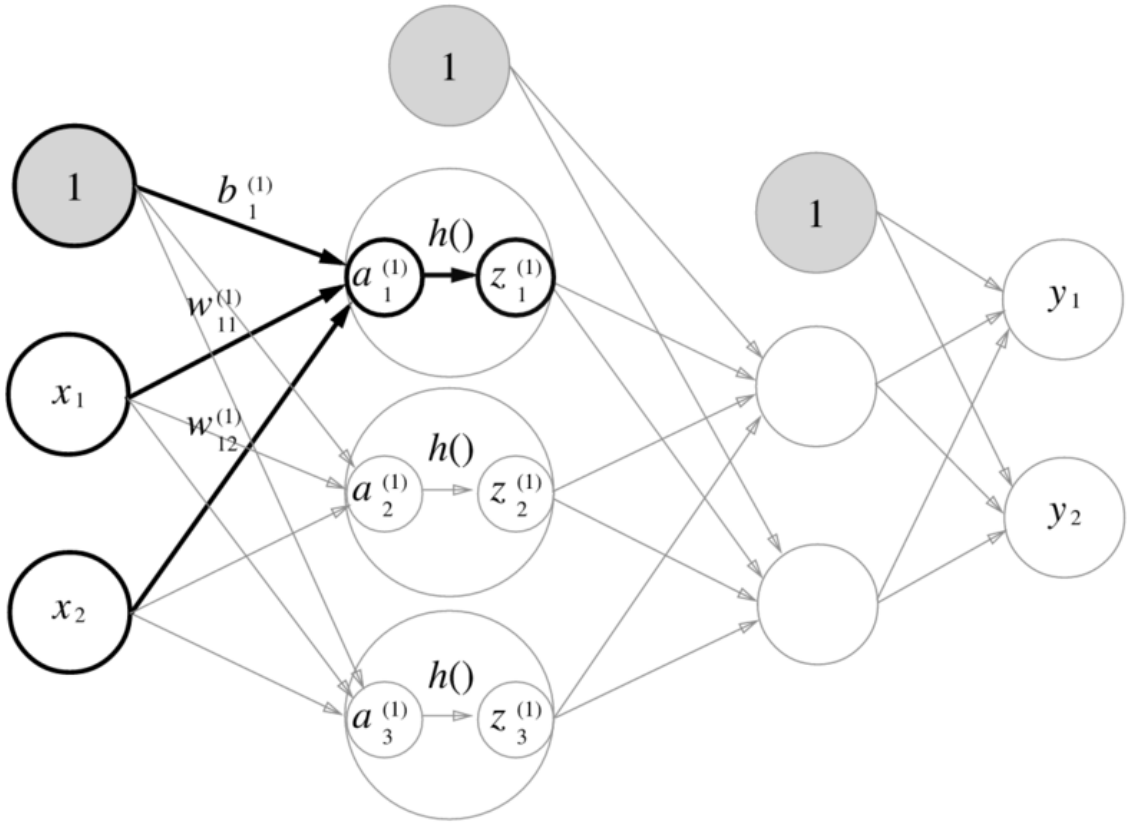


3. 4 3층 신경망 구현하기

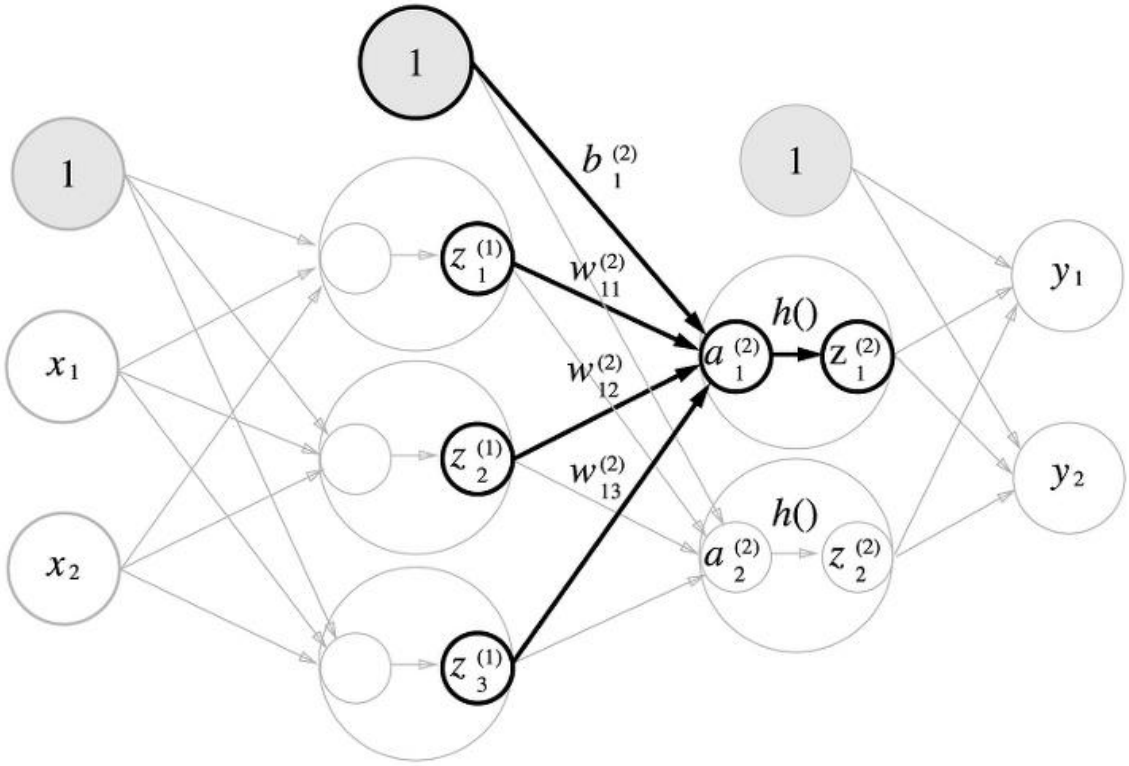
그림 3-15 3층 신경망 : 입력층(0층)은 2개, 첫 번째 은닉층(1층)은 3개, 두 번째 은닉층(2층)은 2개, 출력층(3층)은 2개의 뉴런으로 구성된다.



3. 4 3층 신경망 구현하기



입력층 -> 1층 은닉망



1층 은닉망 -> 2층 은닉망

3. 출력층

- 출력층 : 최종출력을 할 때의 층
- 출력층의 활성화 함수는 풀고자 하는 문제의 성질에 맞추어 정해야 한다.

- 회귀(regression) : 항등 함수
- 2 클래스 분류 (2 class classification) : 시그모이드 함수
- 다중 클래스 분류(multi class classification) : 소프트맥스 함수

그림 3-21 항등 함수

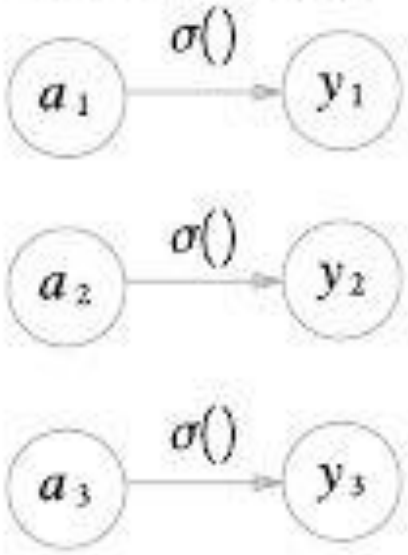
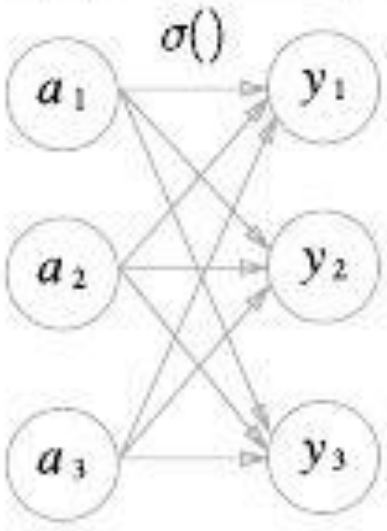


그림 3-22 소프트맥스 함수




```
def identity_function(x):
    return x
```

지수함수

소프트맥스 함수(softmax function)은 $y_k = \frac{exp(a_k)}{\sum_{i=1}^{n} exp(a_i)}$ 이다.

- n : 출력층의 뉴런 수
- y_k : 그 중 k번째 출력 신호
- a_k : k번째 입력 신호

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

<u>소프트맥스 지수 함수의 지수 부분에 어떤 정수를 더하더라도 결과는 바뀌지 않는다</u>

출력 총합이 1이 되기 때문에 '확률'로 해석 가능

3. 6 손글씨 숫자 인식

신경망

- 학습 : 학습 데이터를 이용해 weight 를 학습
- 추론 : 학습한 weight를 이용하여 입력 데이터를 분류

28 * 28 크기의 이미지, 픽셀은 0 ~255

1 MNIST 데이터 셋


```
import numpy as np
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)

# 각 데이터의 형상 출력
print(x_train.shape)
print(t_train.shape)
print(x_test.shape)
print(t_test.shape)
```

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

Load_mnist 인수

- Normalize : 입력 이미지의 픽셀 값을 0.1 ~1.0 정규화
  false 시 0~255 값 유지
- Flatten : 입력 이미지를 1차원 배열로 만들지 결정
  false 시 1*28*28 -> 3차원 배열, True시 784 1차원 배열
- one_hot_label : 원-핫 인코딩

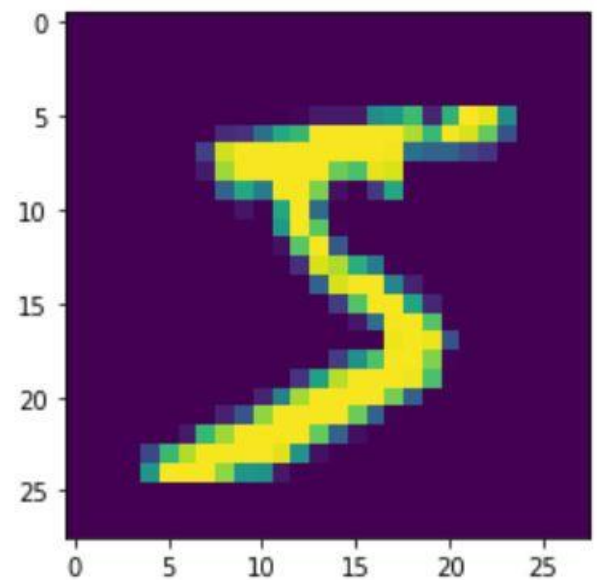
3. 6 손글씨 숫자 인식

MNIST 불러오기

```
from PIL import Image
from matplotlib.pyplot import imshow
%matplotlib inline

def img_show(img) :
    pil_img = Image.fromarray(np.uint8(img)) #넘파이로 저장된 이미지 데이터를 PIL용 데이터 객체로 변환
    # pil_img.show()
    imshow(np.asarray(pil_img))

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

print(img.shape) # flattem = True 으므로, 784개의 원소인 1차원 배열
img = img.reshape(28,28) # 원래 이미지의 모양으로 변형
print(img.shape)
```

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

```
img_show(img)
```



3. 6 손글씨 숫자 인식

신경망의 추론 처리
3층 신경망 구현

* 입력층 뉴런 784개 - 이미지 크기가 28 * 28 = 784
* 은닉층(1) - 50개
* 은닉층(2) - 100개
* 출력층 뉴런 10개 - 0부터 9 숫자 분류 => 10개

```
import pickle

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

# sample_weight.pkl 에 저장된 '학습된 가중치 매개변수'를 읽습니다.
def init_network():
    with open("sample_weight.pkl","rb") as f:
        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)

    return y
```

Predict () – 각 레이블의 확률을 넘파이 배열로 반환
Ex) [0.1 , 0.3, …, 0.4] -> 이미지가 숫자 '0'일 확률을 배열로 반환

3. 6 손글씨 숫자 인식

```
# 정확도 평가
x,t = get_data()
network = init_network()

accuracy_cnt = 0
for i in range(len(x)) :
    y = predict(network, x[i])
    p = np.argmax(y) #확률이 가장 높은 원소의 인덱스를 얻는다.
    if p == t[i] :
        accuracy_cnt += 1

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

[0.1 , 0.3, …, 0.4] -> 이미지가 숫자 '0'일 확률을 배열로 반환
Np.argmax() 는 이 배열에서 확률이 가장 높은 원소의 인덱스를 구함
=> 예측결과

Accuracy_cnt :신경망이 예측한 답변과 정답 레이블 비교, 맞힌 숫자 세기

```
Accuracy:0.9352
```

정확도 : Accuracy_ cnt / 전체 이미지 숫자

전처리(pre-processing) : 신경망의 입력 데이터에 특정 변환을 가하는 것

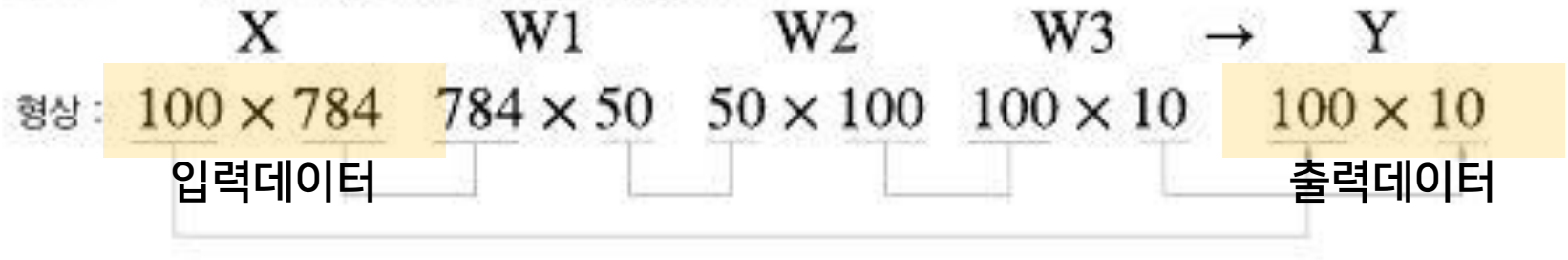
- 정규화 (normalization) : 데이터를 특정 범위로 변환하는 처리 Ex) normalize = True
- 백색화 (whitening) : 전체 데이터를 균일하게 분포

3. 6 손글씨 숫자 인식

배치 처리

다차원 배열의 대응하는 차원의 원소 수 일치

그림 3-27 배치 처리를 위한 배열들의 형상 추이



- 이미지 100개를 묶어 predict() 함수에 처리

배치 : 하나로 묶은 입력 데이터

Ex) 1000개의 28 * 28 px 이미지 데이터를 1000 *784개로 묶는 것, 출력은 1000*10 (입력데이터 개수 * 레이블 개수)

이유 1) 대부분의 라이브러리가 큰 배열을 효율적으로 처리할 수 있도록 최적화

이유 2) 배치 처리를 함으로써 다음 신경망으로 데이터를 넘길 때 부하를 줄일 수 있어서

3. 6 손글씨 숫자 인식

배치 처리

```
x, t = get_data()
network = init_network()

batch_size = 100
accuracy_cnt = 0

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)
    accuracy_cnt += np.sum(p == t[i:i+batch_size])

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

X[0:100], x[100:200] … 등 100장 씩 묶어 꺼냄

최댓값의 인덱스를 가져옴

'==' 를 이용해 넘파이 배열끼리 bool 배열을 만들어 True 개수를 셈