3_

November 15, 2025

0.1 3.1

0.1.1 3.1.1

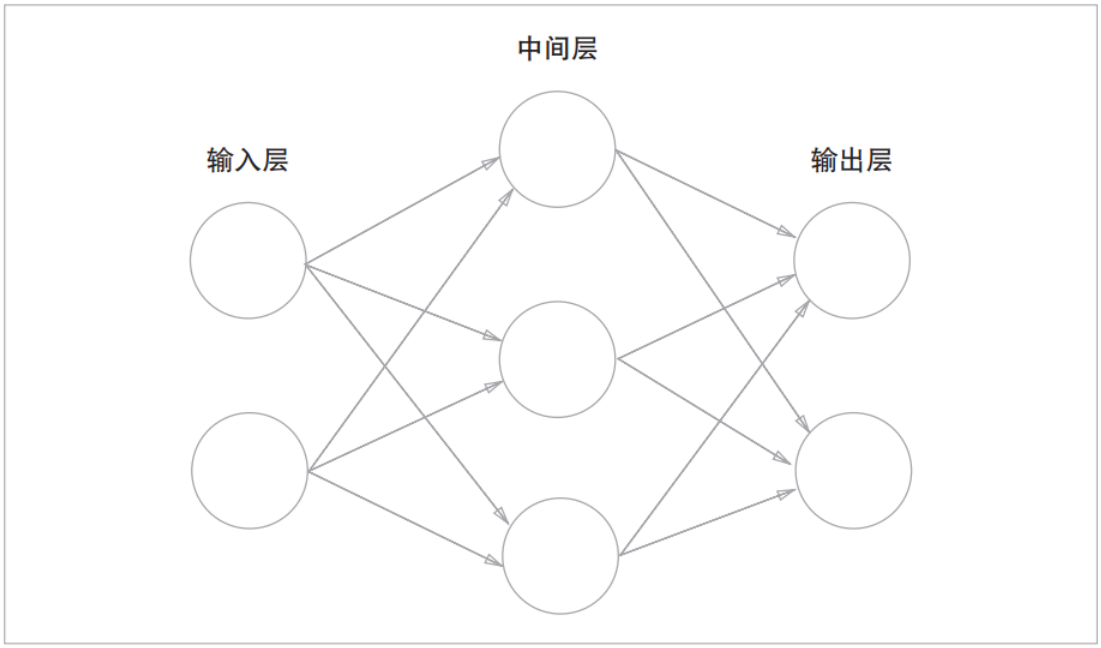


图3-1 神经网络的例子

( )

0.1.2 3.1.2

$$a = b + w_1 x_1 + w_2 x_2 \quad (1)$$

$$y = h(a) \quad (2)$$

1 2 3-4

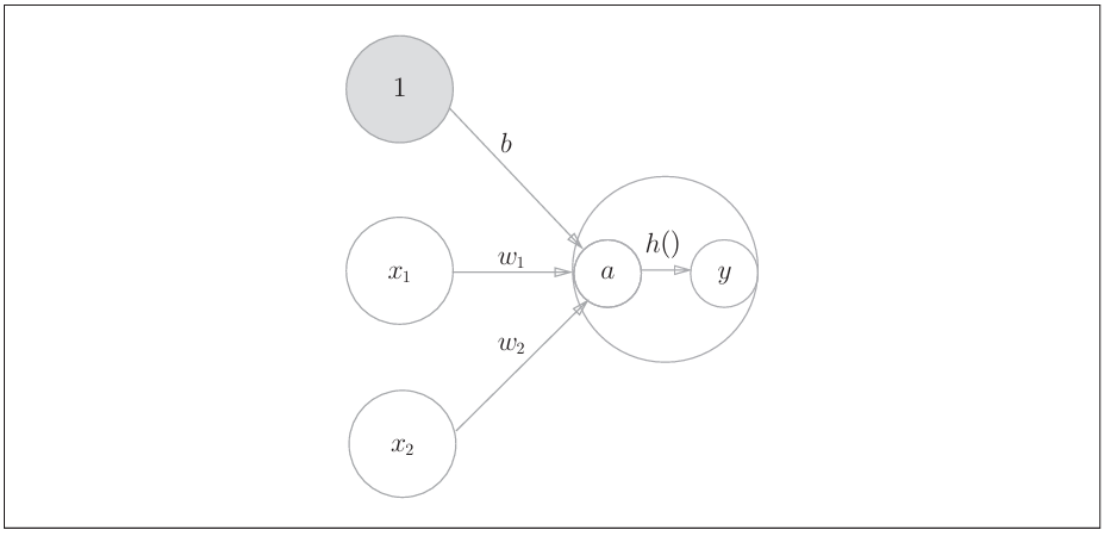


图3-4 明确显示激活函数的计算过程

0.2 3.2

“ ”

0.2.1 3.2.1 sigmoid

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

0.2.2 3.2.2

[30]:
```
import pickle


def step_function(x):
    if x>0:
        return 1
    else:
        return 0
```

x Numpy np.array[1.0,2.0],

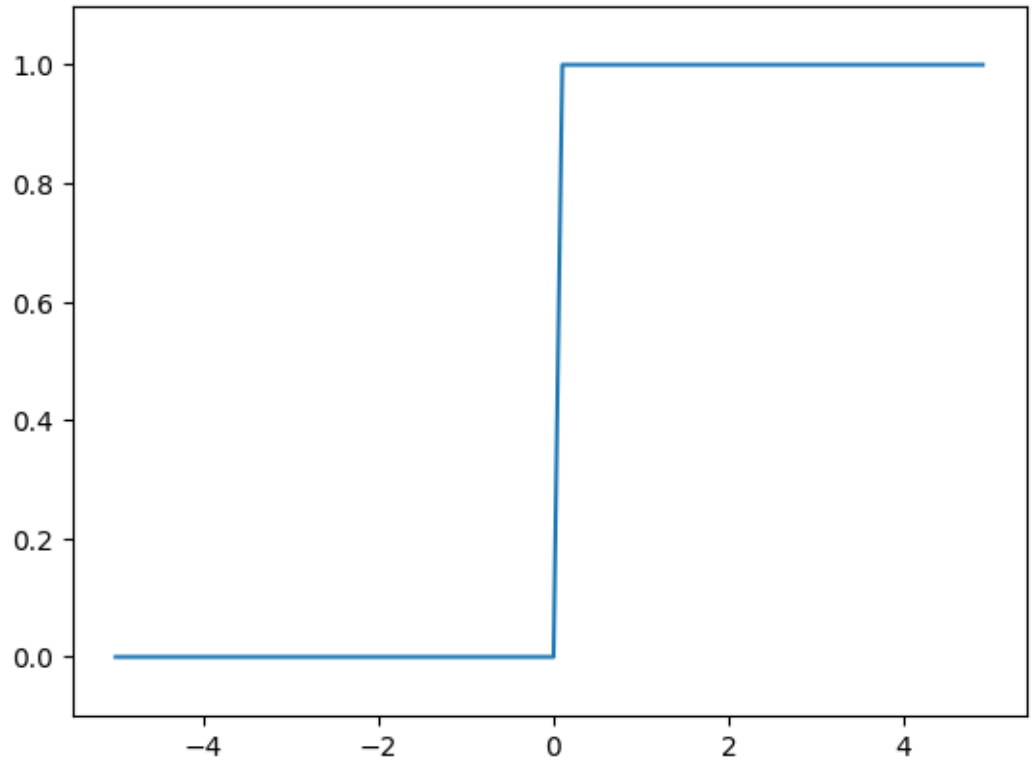
[31]:
```
import numpy as np
def step_function(x):
    y = x > 0#
    return y.astype(np.int)
```

[32]:
```
x=np.array([0,2,3,4,5])
y=x>0# y
y=y.astype(np.int64)
y#astype()  NumPy   ,true  1 False  0
```

[32]:
```
array([0, 1, 1, 1, 1])
```

0.2.3 3.2.3

[33]:
```
import matplotlib.pyplot as plt
import numpy as np

def step_function(x):
    y=x > 0
    z=y.astype(np.int64)
    return z

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



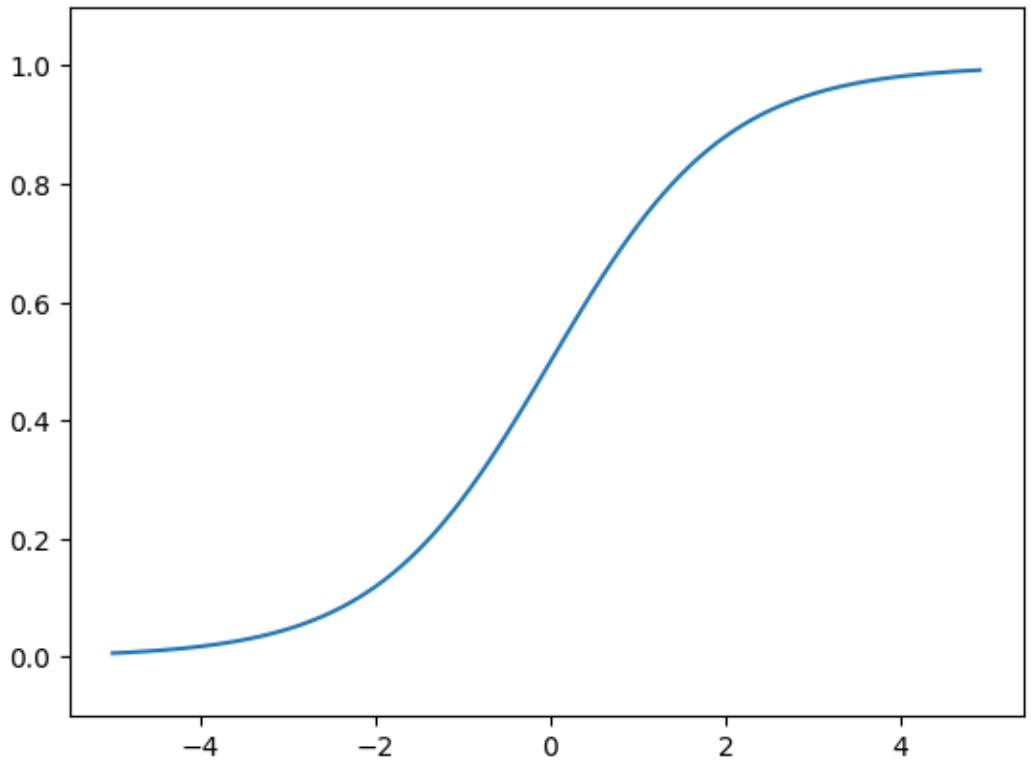
0 0 1

0.2.4 3.2.4 sigmoid

[34]:
```
def sigmoid(x):
    return 1/(1+np.exp(-x))
x=np.array([0,2,3,4,5])
sigmoid(x)
```

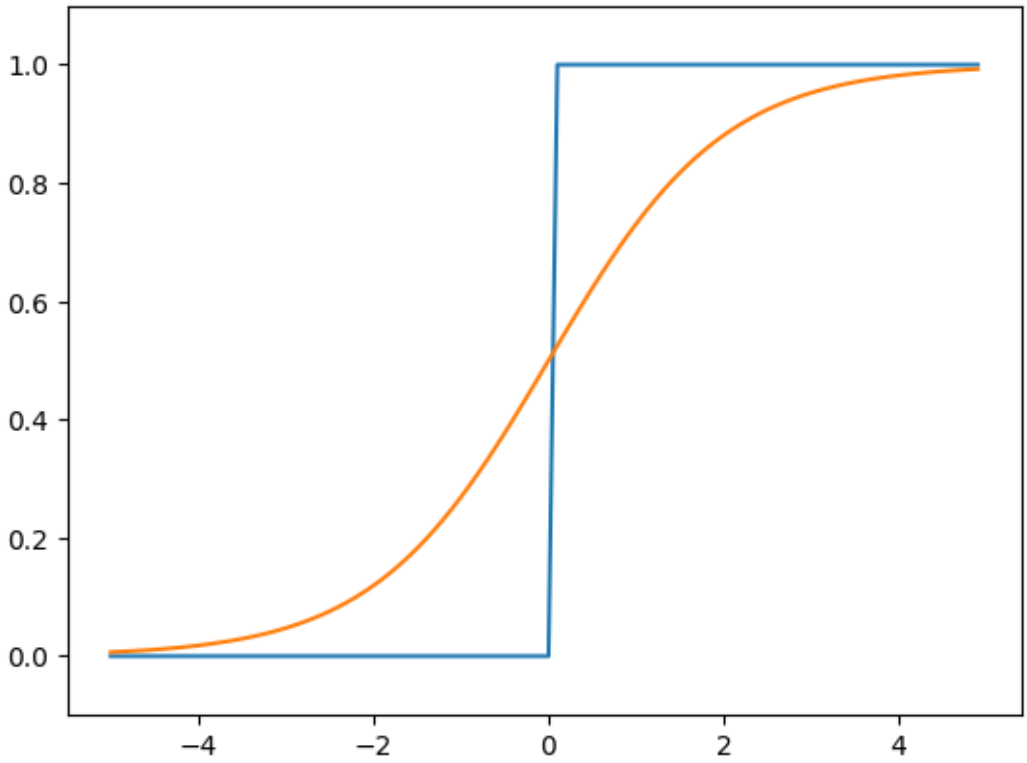
[34]:
```
array([0.5       , 0.88079708, 0.95257413, 0.98201379, 0.99330715])
```

[35]:
```
x=np.arange(-5.0,5.0,0.1)
y=sigmoid(x)
plt.plot(x,y)
plt.ylim(-0.1,1.1)
plt.show()
```



[36]:
```
y1=step_function(x)
y2=sigmoid(x)
plt.plot(x,y1)
plt.plot(x,y2)
plt.ylim(-0.1,1.1)
```

```
plt.show()
```



0.2.5 3.2.6

,

0.2.6 3.2.7 ReLU

Rectified Linear Unit

ReLU 0 0 0

[37]:
```
def relu(x):
    return np.maximum(0,x)
```

0.3 3.3

0.3.1 3.3.1

[38]:
```
A=np.array([1,2,3,4,5])
print(A)
print(np.ndim(A))
print(A.shape)
A.shape[0]
```

```
[1 2 3 4 5]
1
(5,)
```

[38]:
```
5
```

[39]:
```
B=np.array([[1,2],[3,4],[5,6]])
print(B)
print(np.ndim(B))
print(B.shape)
print(B.shape[0])
```

```
[[1 2]
 [3 4]
 [5 6]]
2
(3, 2)
3
```

0.3.2 3.3.2

[40]:
```
A=np.array([[1,2,3],[3,4,5],[5,6,7]])
B=np.array([[1,2,3],[3,4,5],[5,6,7]])
C=np.dot(A,B)#
C
```

[40]:
```
array([[22, 28, 34],
       [40, 52, 64],
       [58, 76, 94]])
```

0.3.3 3.3.3

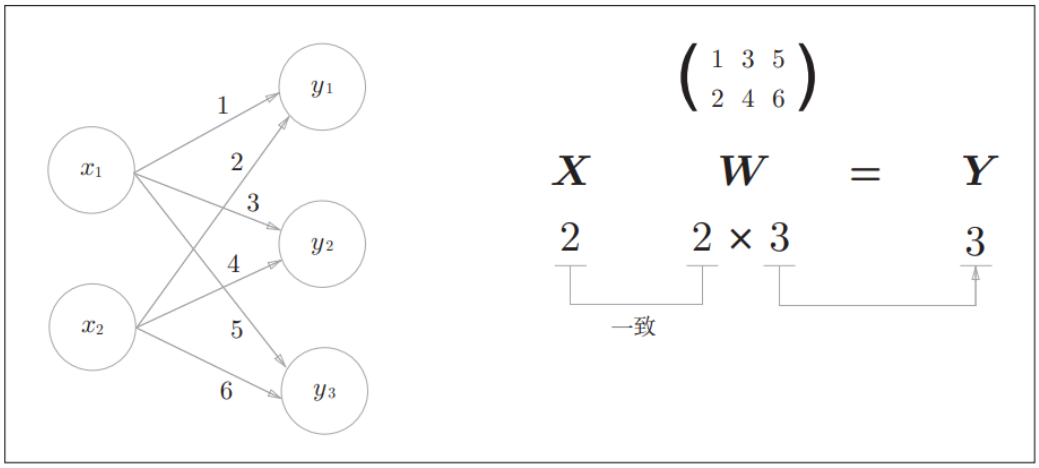


图3-14 通过矩阵的乘积进行神经网络的运算

[41]:
```
X=np.array([1,2])
W=np.array([[1,3,5],[2,4,6]])
print(W)
Y=np.dot(X,W)
print(Y)
```

```
[[1 3 5]
 [2 4 6]]
[ 5 11 17]
```

0.4 3.4 3

0.4.1 3.4.1

[42]:
```
X=np.array([1.0,0.5])
W1=np.array([[0.1,0.3,0.6],[0.2,0.5,0.3]])
B1=np.array([0.1,0.3,0.3])
print(W1.shape)
print(B1.shape)
print(X.shape)
A1=np.dot(X,W1)+B1
A1
```

```
(2, 3)
(3,)
(2,)
```

[42]:
```
array([0.3 , 0.85, 1.05])
```

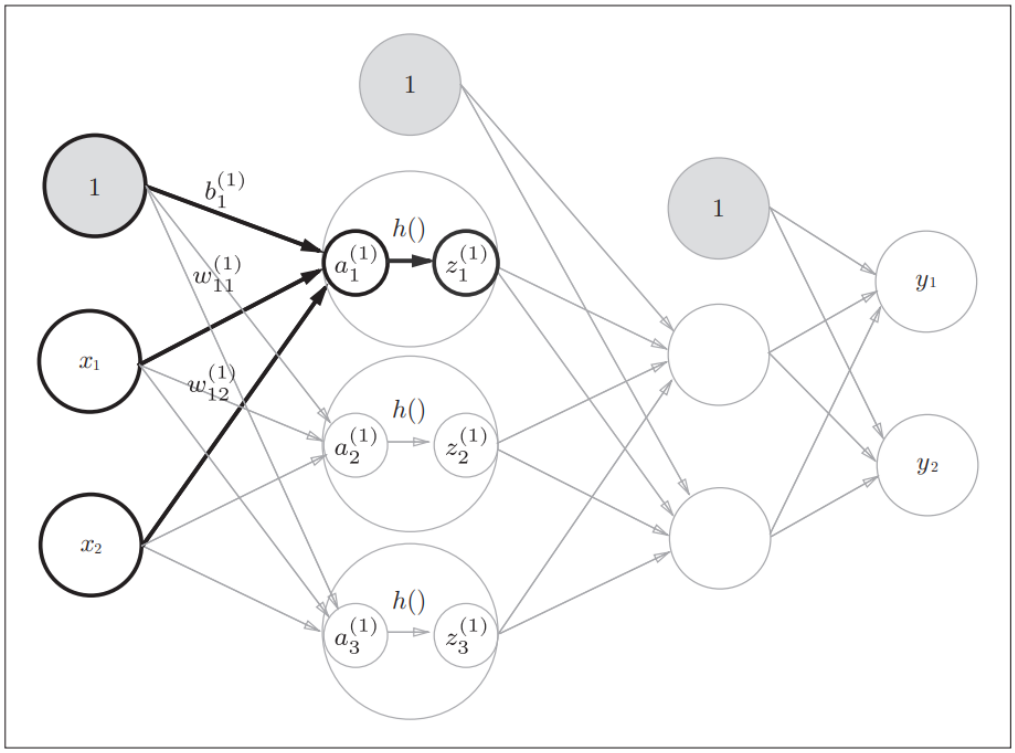


图3-18　从输入层到第1层的信号传递

[43]:
```
# sigmoid
Z1=sigmoid(A1)
print(Z1)
print(A1)
```

```
[0.57444252 0.70056714 0.7407749 ]
[0.3  0.85 1.05]
```

[44]:
```
W2 = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
B2 = np.array([0.1, 0.2])
print(Z1.shape) # (3,)
print(W2.shape) # (3, 2)
print(B2.shape) # (2,)
A2=np.dot(Z1,W2)+B2
Z2=sigmoid(A2)
print(Z2)
```

```
(3,)
(3, 2)
(2,)
[0.6270987  0.77285898]
```

sigmoid

softmax

0.4.2 3.4.3

[45]:
```
def init_network():
    network={}
    network['W1']=np.array([[0.1,0.3,0.5],[0.2,0.4,0.6]])

    network['B1']=np.array([0.1,0.2,0.3])

    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=network['W1']
    B1=network['B1']
    W2=network['W2']
    B2=network['B2']
    W3=network['W3']
    B3=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=a3

    return y

network=init_network()

x = np.array([1.0, 0.5])
y = forward(network, x)
print(y)

#init_network()             network

#forward()
```

```
[0.31682708 0.69627909]
```

0.5 3.5

softmax

0.5.1 3.5.1 softmax

softmax

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

softmax

[46]:
```
a=np.array([0.3,2.9,4.0])
exp_a=np.exp(a)
b=np.sum(exp_a)
y=exp_a/b
y
```

[46]:
```
array([0.01821127, 0.24519181, 0.73659691])
```

[47]:
```
def softmax(x):
    return np.exp(x) / np.sum(np.exp(x))
```

0.5.2 3.5.2 softmax

“ ”

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

C

[48]:
```
#  ,  softmax
a=np.array([900,1000])#
#np.exp(a)/np.sum(np.exp(a))
c=np.max(a)
np.exp(a-c)/np.sum(np.exp(a-c))
```

[48]:
```
array([3.72007598e-44, 1.00000000e+00])
```

0.5.3 3.5.3 softmax

[49]:
```
a = np.array([0.3, 2.9, 4.0])
y = softmax(a)
np.sum(y)
```

[49]:
```
np.float64(1.0)
```

softmax 0.0 1.0 , 1 softmax “ ”

0.6 3.6

0.6.1 3.6.1 MNIST

[50]:
```
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) # (60000, 784)
print(t_train.shape) # (60000,)
print(x_test.shape) # (10000, 784)
print(t_test.shape) # (10000,)
```

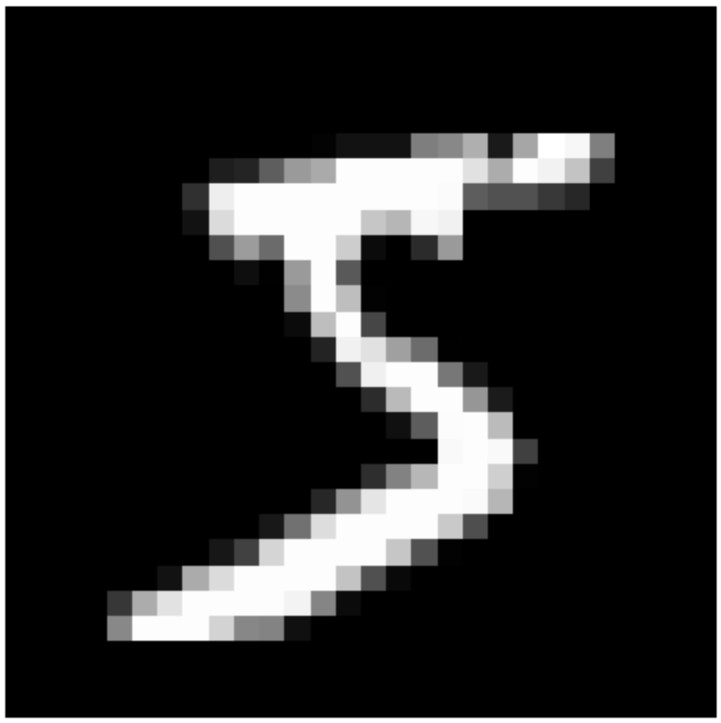
```
(60000, 784)
(60000,)
(10000, 784)
(10000,)
```

[51]:
```
import sys, os
sys.path.append(os.pardir)
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) #
print(img.shape) # (28, 28)
img_show(img)
```

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

[52]:
```python
import sys, os
import numpy as np
import matplotlib.pyplot as plt

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

def img_show_plt(img):
    # 1.   imshow
    #    - cmap='gray'         MNIST
    #    - interpolation='nearest'
    plt.imshow(img, cmap='gray', interpolation='nearest')
    # 2.
    plt.axis('off')
    # 3.
    plt.show()

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

#      (  )
img = x_train[0]
label = t_train[0]
print(f" : {label}")
#    : (784,)
print(f"   : {img.shape}")
#     (784,)   (28, 28)
img = img.reshape(28, 28)
print(f"    : {img.shape}")
#    Matplotlib
img_show_plt(img)
```

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



0.6.2 3.6.2

784 10

784 $28 \times 28 = 784$ 10 10 0 9 10

2 1 50 2 100 50 100

[53]:
```
import pickle
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:
        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
```

init_network() pickle sample_weight.pkl A 2 3

[54]:
```
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)
    if p==t[i]:
        accuracy_cnt+=1

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

```
Accuracy:0.9352
```

MNIST for x

0.6.3 3.6.3

“ ”

[55]:
```
x, _ = get_data()
network=init_network()
w1,w2,w3=network['W1'], network['W2'], network['W3']
x.shape
```

[55]:
```
(10000, 784)
```

[56]:
```
x[0].shape
```

[56]:
```
(784,)
```

[57]:
```
w1.shape
```

[57]:
```
(784, 50)
```

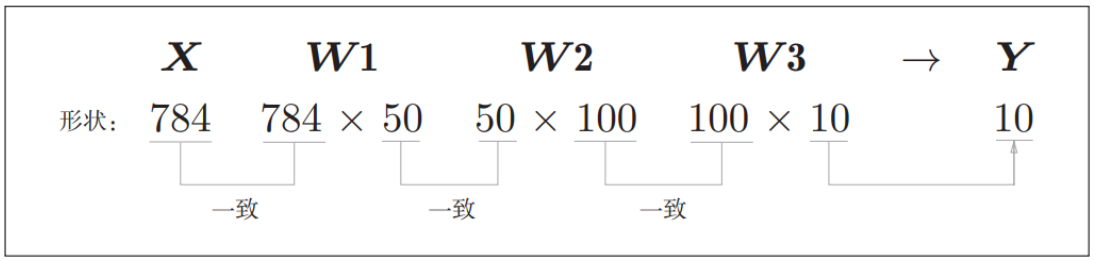


图3-26　数组形状的变化

784　28*28　　10

predict()　　100　　x　　100 × 784　100　　3-27

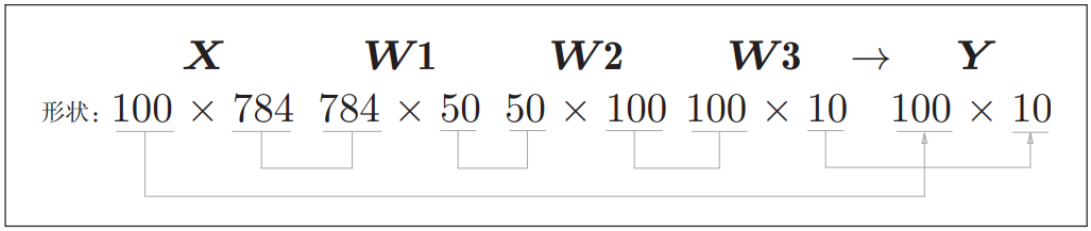


图3-27　批处理中数组形状的变化

[58]:
```
#
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)))
```

```
Accuracy:0.9352
```