Chapter 3 Neural Network

3.3 Calculation of Multi-dimensional array
3.4 Implementation of 3-layer Neural Network

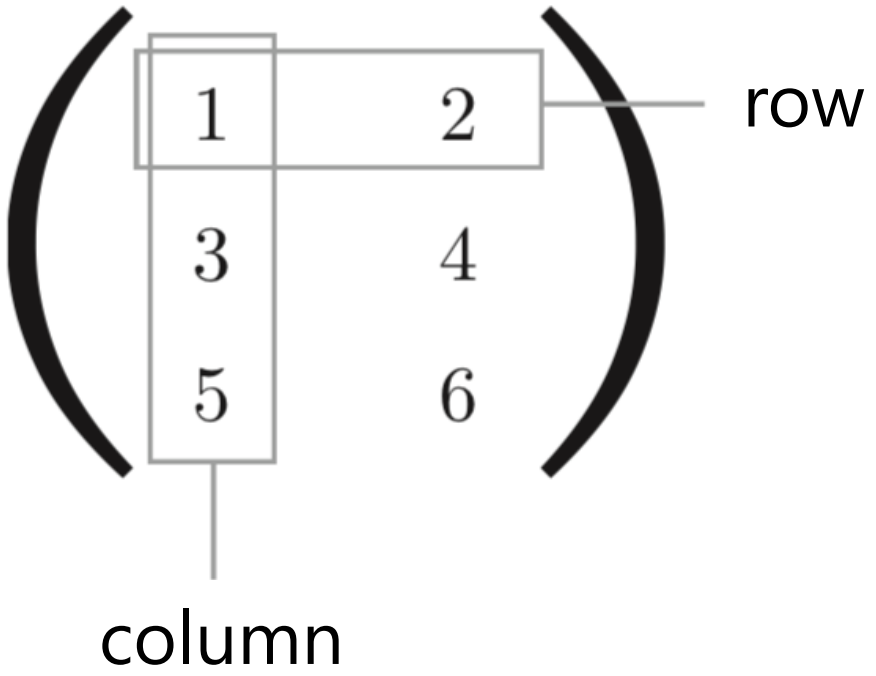
M1 Wataru Akashi

Topics

- Multi-dimensional Array
- Product of matrix
- Implementation of Simple Neural Network
- Summary of implementation
- Summary

Multi-dimensional Array

- “Set of number”
- to place in a row, rectangular, 3-dimensional, … , N-dimensional shape
- 2-dimensional array is called “matrix”



Create an Array using NumPy

※NumPy (Number + Python) ⇒ “ナンパイ” or ナムパイ

```
>>> import numpy as np
>>> A = np.array([1, 2, 3, 4])
>>> print(A)
[1 2 3 4]
>>> np.ndim(A)
1
>>> A.shape
(4,)
>>> A.shape[0]
4
```

⇒ Create an 1-dimensional array

⇒Acquire dimension of array

⇒Acquire shape of array

⇒Acquire number of elements

$$(1 \quad 2 \quad 3 \quad 4)$$

Create an Array using NumPy

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

⇒Create a 2-dimensional array(matrix)

⇒Acquire dimension of array

⇒Acquire shape of array

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

Product of matrix

```
>>> A = np.array([[1,2], [3,4]])
```
⇒Create a 2-dimensional array(matrix)

```
>>> A.shape
```
⇒Acquire shape

```
(2, 2)
```

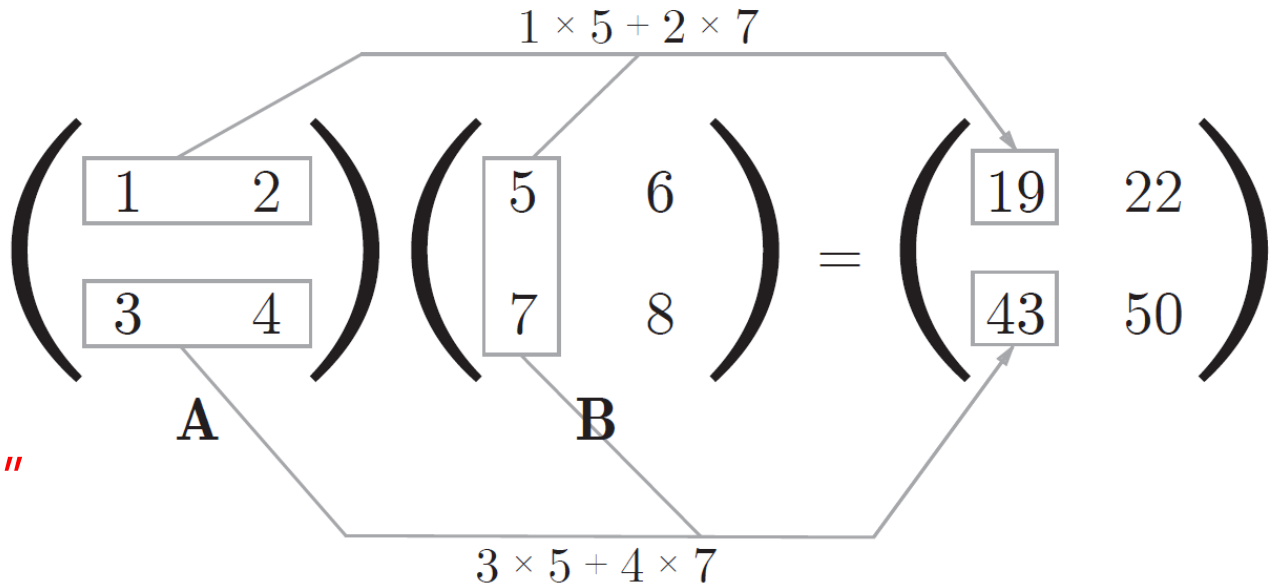
```
>>> B = np.array([[5,6], [7,8]])
```
⇒Create a 2-dimensional array(matrix)

```
>>> B.shape
```
⇒Acquire shape

```
(2, 2)
```

```
>>> np.dot(A, B)
```
⇒Calculate product of matrix

```
array([[19, 22],
       [43, 50]])
```



np.dot()
⇒can take at most 3 arguments

revise in 5/21

np.dot(A,B,C)⇒not means "A・B・C"
substitute result of "A・B" for C

Error in product of matrix

```
>>> C.shape
(2, 2)
>>> A.shape
(2, 3)
>>> np.dot(A, C)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: shapes (2,3) and (2,2) not aligned: 3 (dim 1) != 2 (dim 0)
```

A **C**

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Not calculable

Calculable example

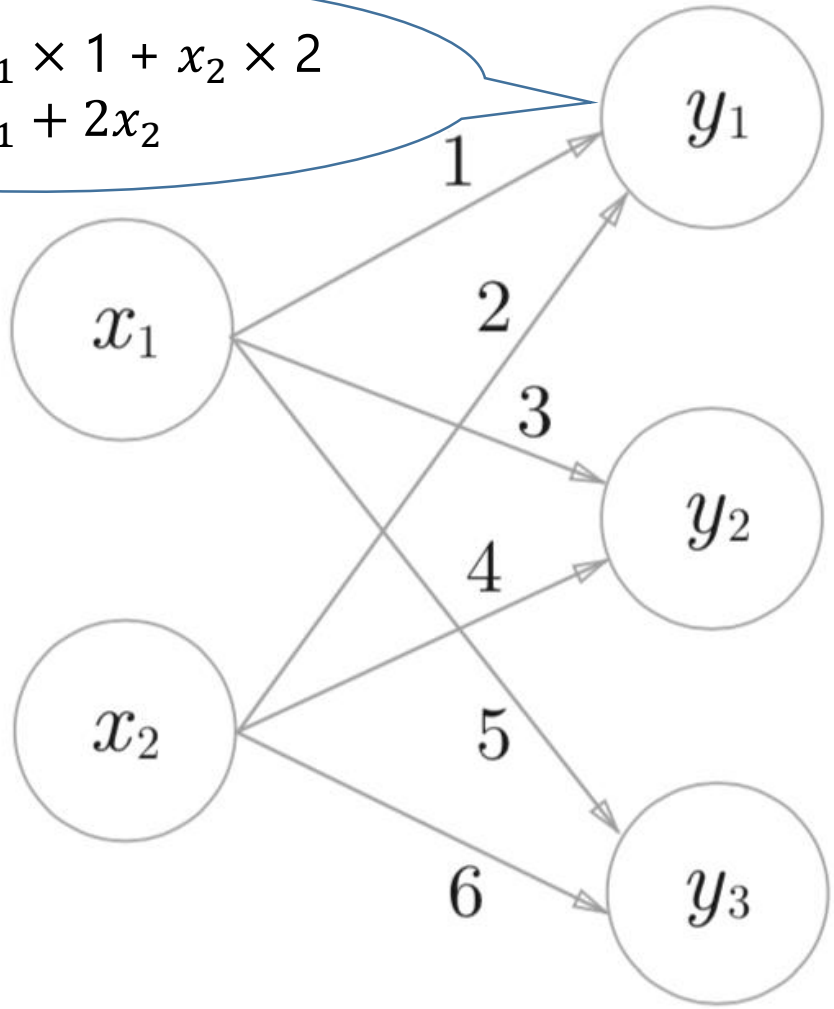
	A	**B**	=	**C**
shape:	3 × 2	2 × 4		3 × 4

match the number of elements

Calculable

Product of matrix in 1-layer Neural Network

ex.) $y_1 = x_1 \times 1 + x_2 \times 2$
$= x_1 + 2x_2$



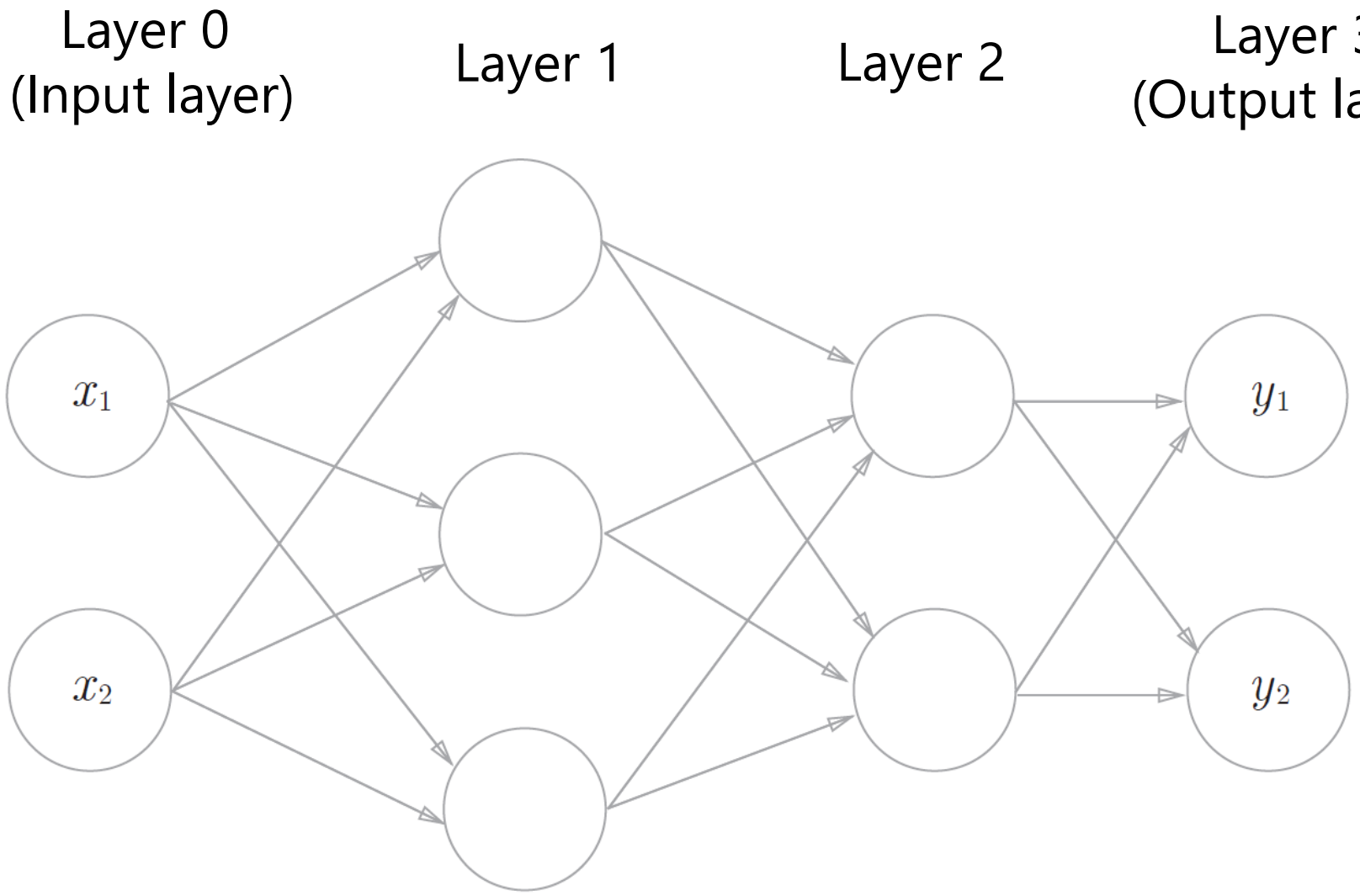

```
>>> X = np.array([1, 2])
>>> X.shape
(2,)
>>> W = np.array([[1, 3, 5], [2, 4, 6]])
>>> print(W)
[[1 3 5]
 [2 4 6]]
>>> W.shape
(2, 3)
>>> Y = np.dot(X, W)
>>> print(Y)
[ 5  11  17]
```

$X = (x_1, x_2)$ ・・・ Input

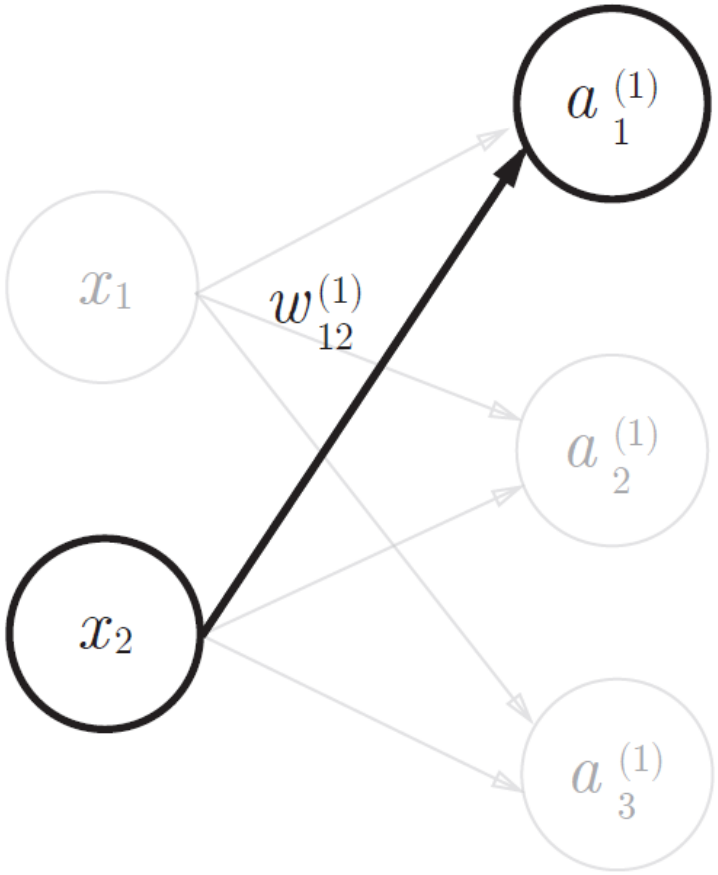
$W = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$ ・・・ Weight

$Y = (y_1, y_2, y_3)$ ・・・ Output

Implementation of 3-layer Neural Network



Definition of symbol



$w_{12}^{(1)}$

Weight in 1st layer

2nd neuron in front layer

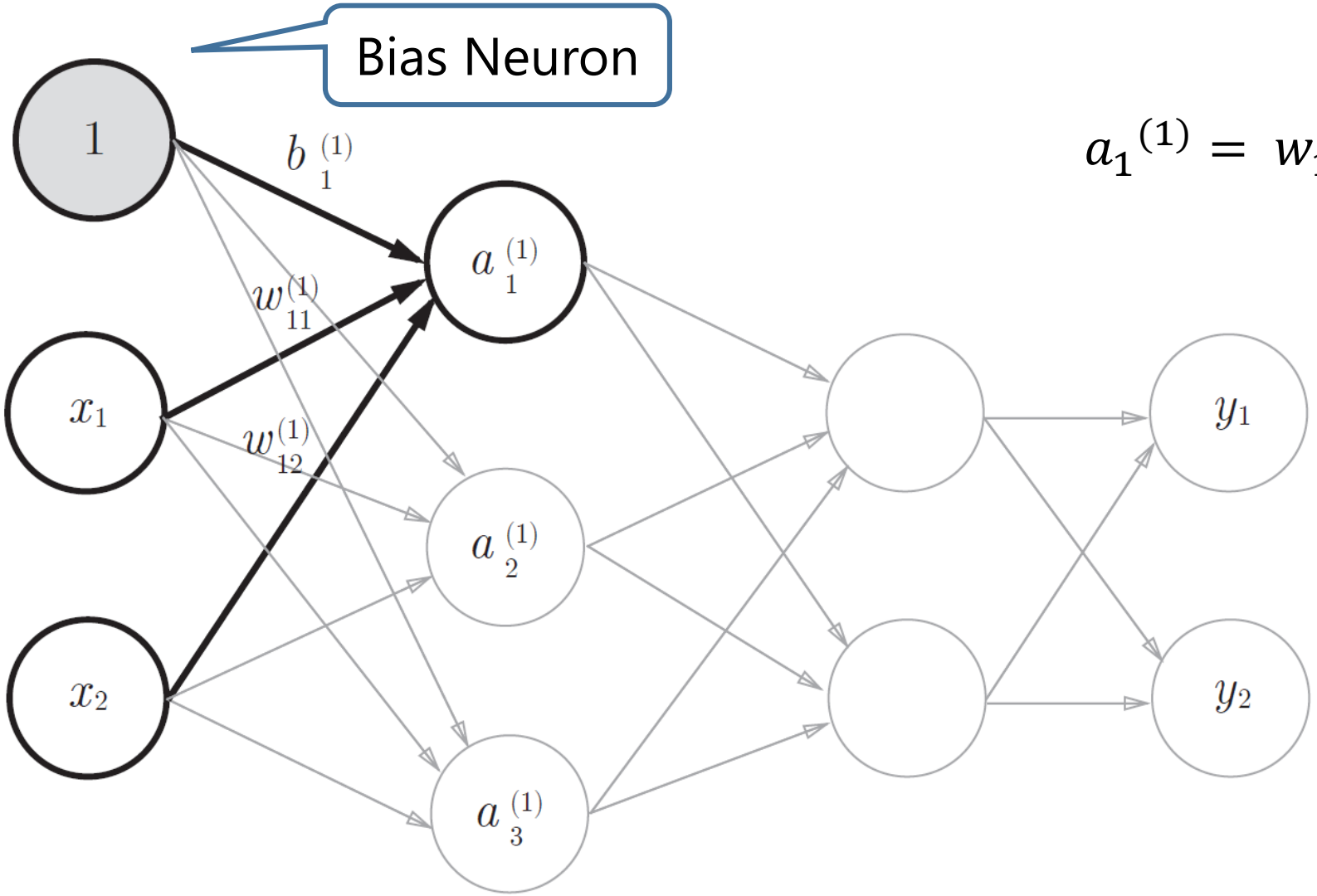
1st neuron in next layer

$a_1^{(1)}$

Neuron in 1st layer

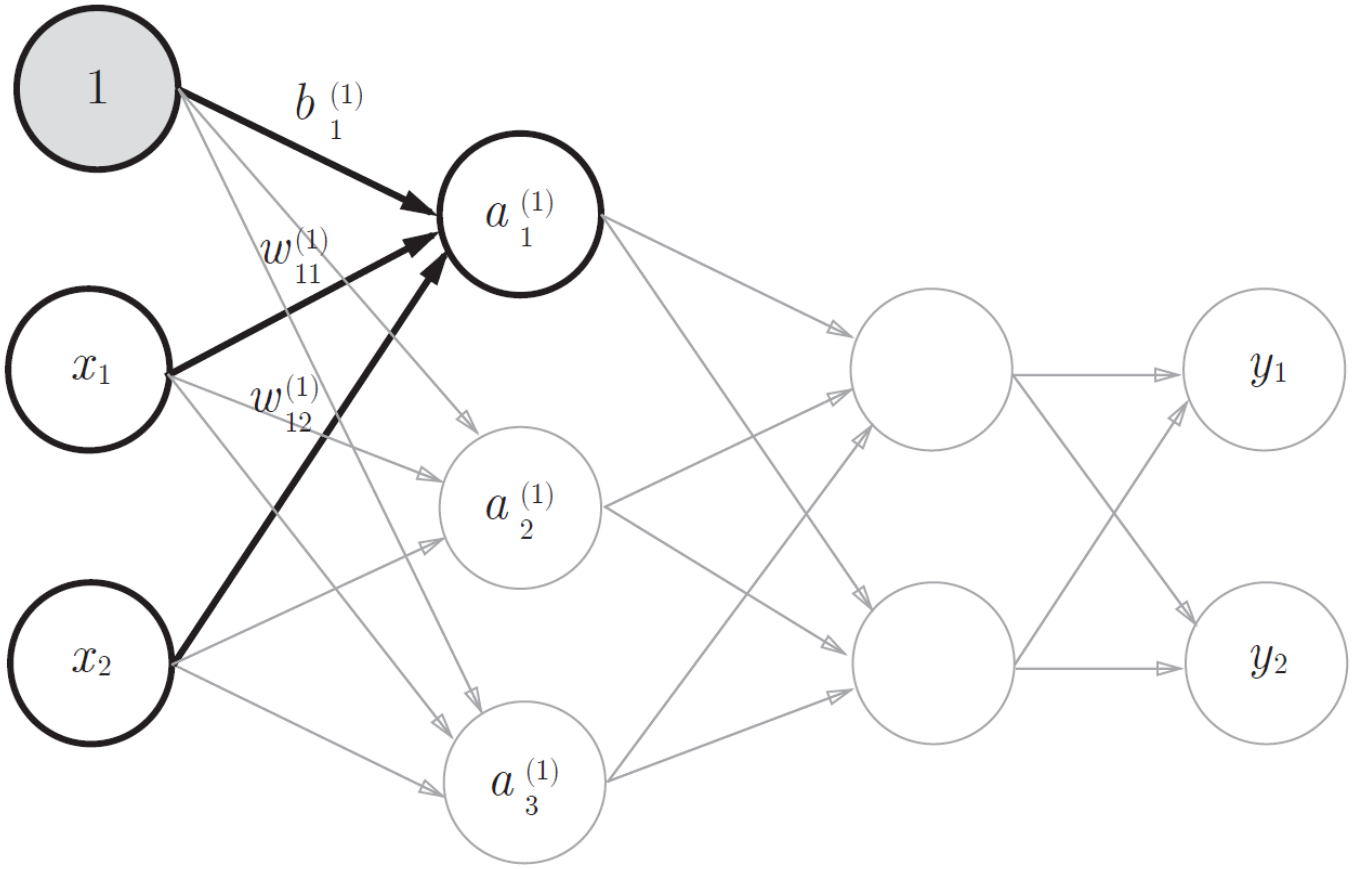
1st neuron in this layer

Signal transmission to 1st neuron in 1st layer



$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

Signal transmission to 1st layer



$$A^{(1)} = (a_1{}^{(1)} \;\; a_2{}^{(1)} \;\; a_3{}^{(1)})$$

$$X = (x_1 \;\; x_2)$$

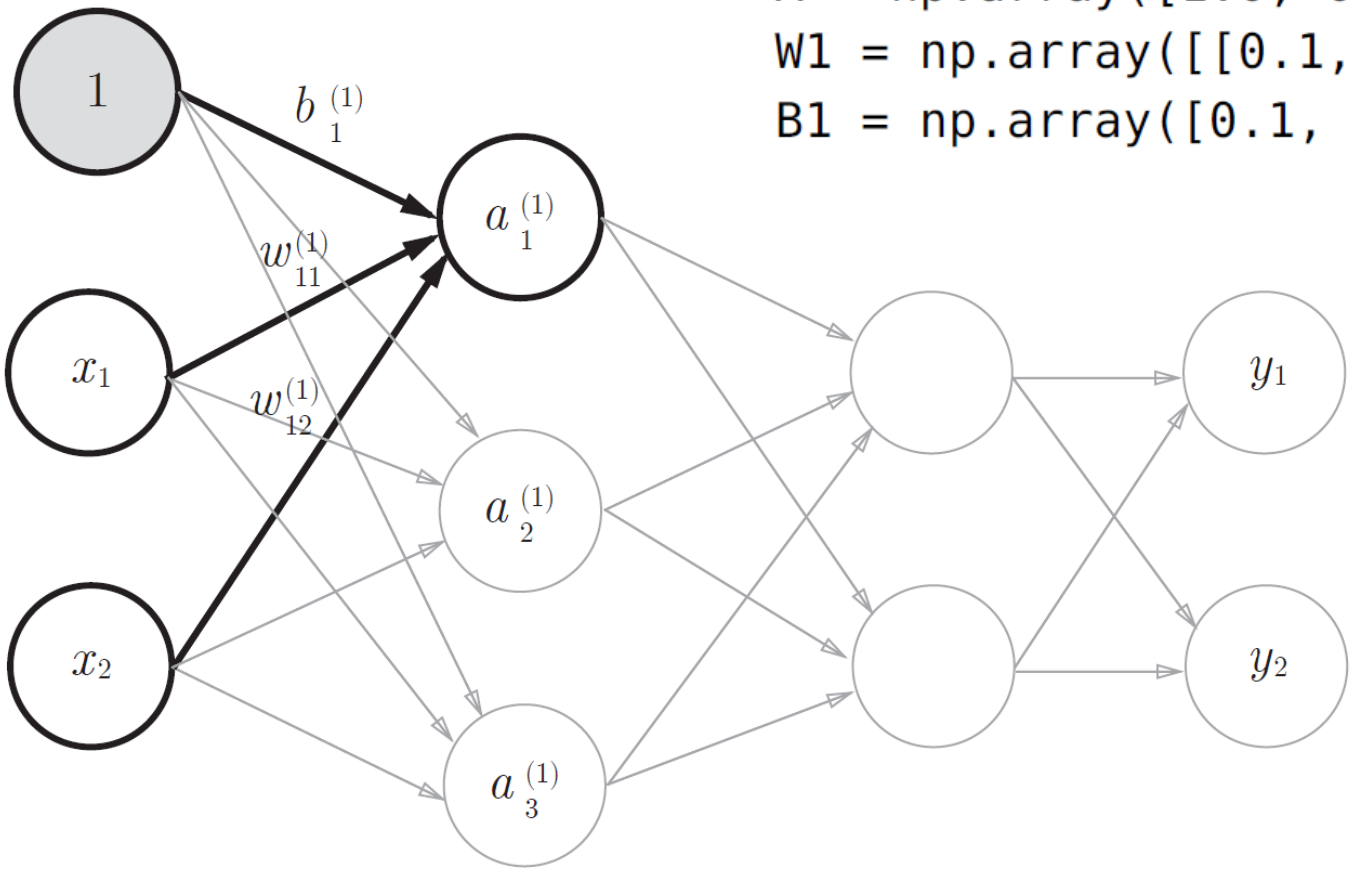
$$B^{(1)} = (b_1{}^{(1)} \;\; b_2{}^{(1)} \;\; b_3{}^{(1)})$$

$$W^{(1)} = \begin{pmatrix} w_{11}{}^{(1)} & w_{21}{}^{(1)} & w_{31}{}^{(1)} \\ w_{12}{}^{(1)} & w_{22}{}^{(1)} & w_{32}{}^{(1)} \end{pmatrix}$$

$$A^{(1)} = XW^{(1)} + B^{(1)}$$

Calculable at once

Signal transmission to 1st layer using NumPy



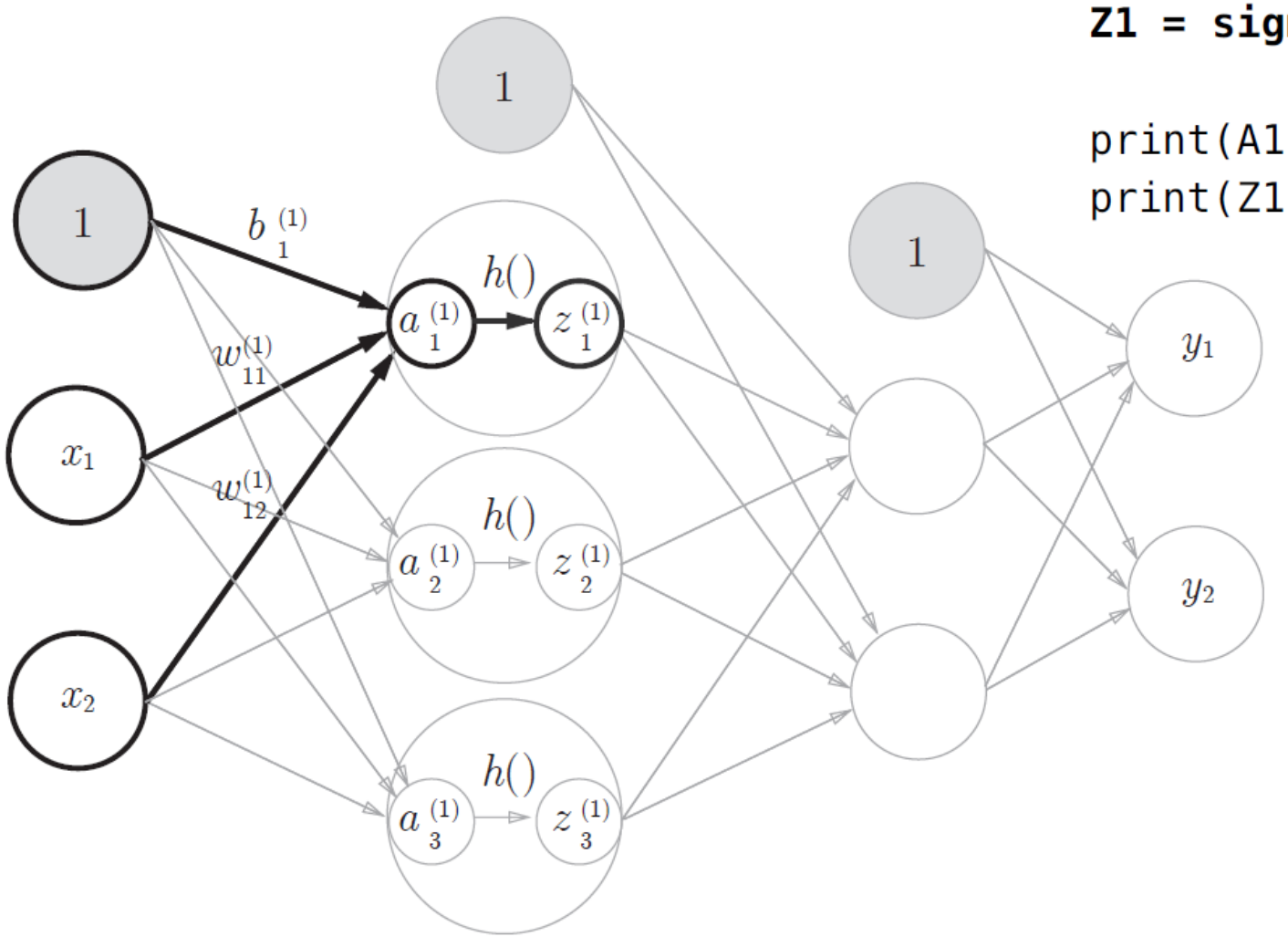

```
X = np.array([1.0, 0.5])
W1 = np.array([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])
B1 = np.array([0.1, 0.2, 0.3])
```

```
A1 = np.dot(X, W1) + B1
```

$$(1.0 \quad 0.5)\begin{pmatrix}0.1 & 0.3 & 0.5\\ 0.2 & 0.4 & 0.6\end{pmatrix}+(0.1 \quad 0.2 \quad 0.3)$$

```
print(A1) # [0.3, 0.7, 1.1]
```

Process in 1st layer using activation function




```
Z1 = sigmoid(A1)

print(A1) # [0.3, 0.7, 1.1]
print(Z1) # [0.57444252, 0.66818777, 0.75026011]
```

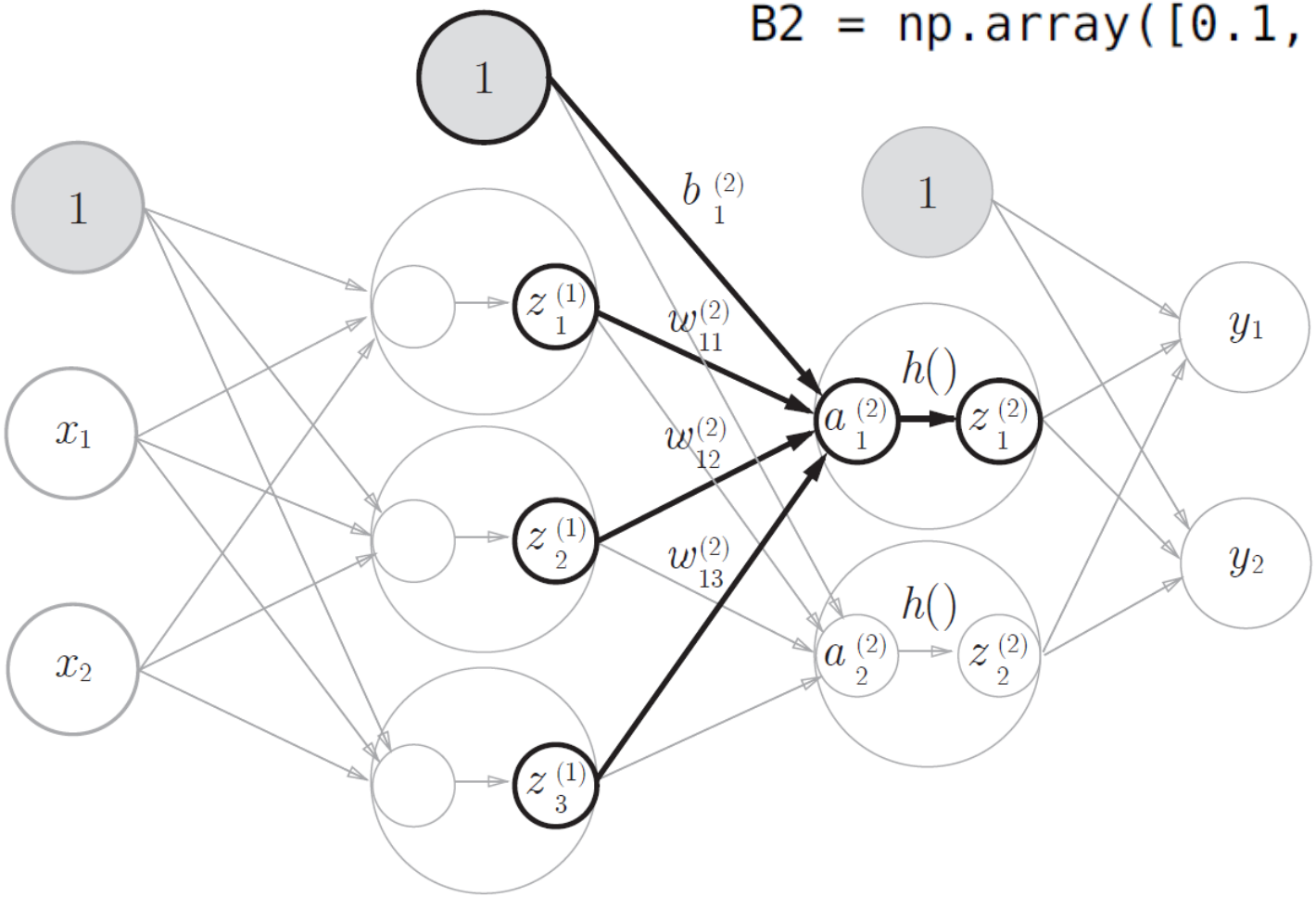
$$\left(\frac{1}{1+e^{-0.3}}, \quad \frac{1}{1+e^{-0.7}}, \quad \frac{1}{1+e^{-1.1}}\right)$$

Sigmoid function

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

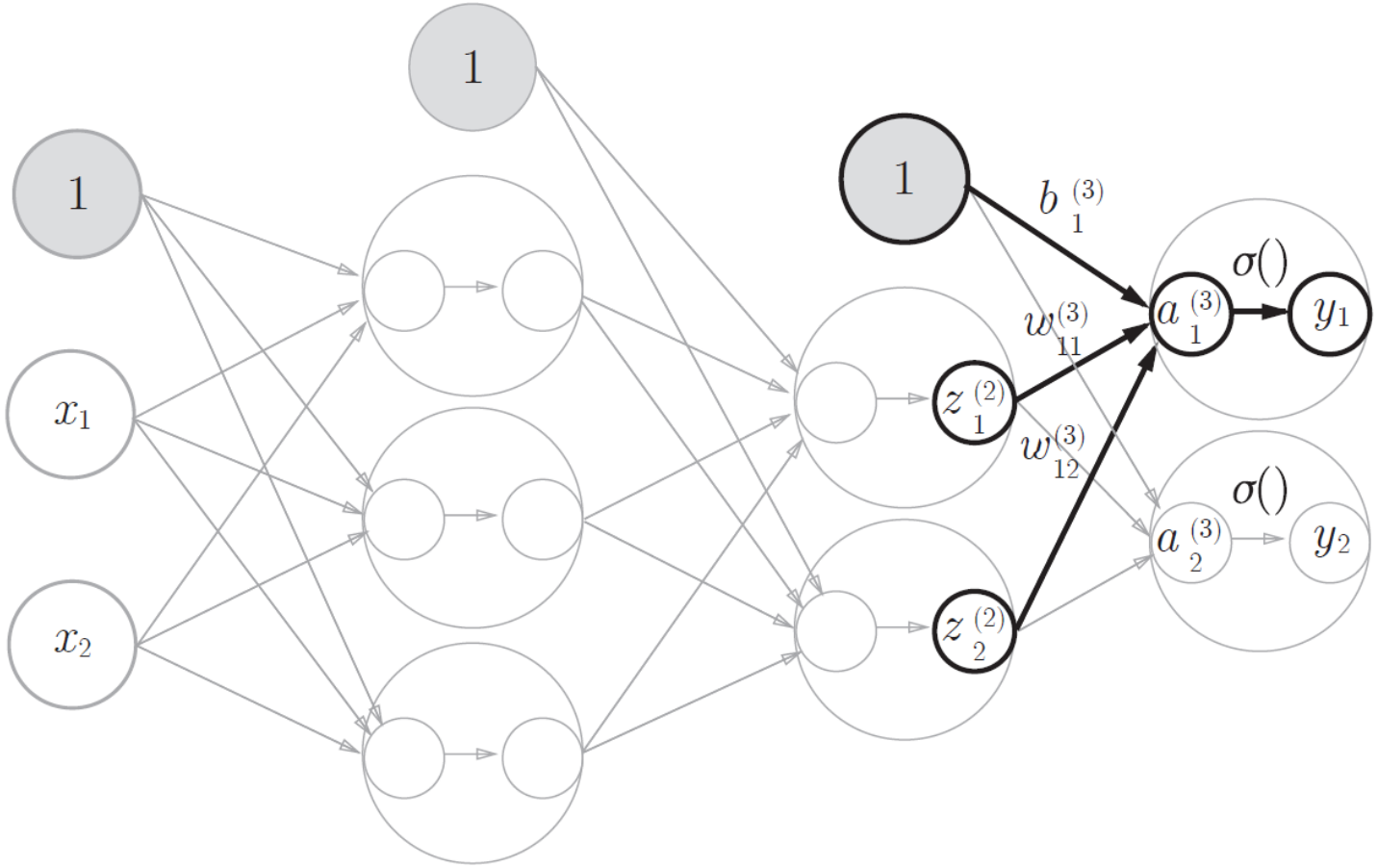
Signal transmission from 1st layer to 2nd layer

```
W2 = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
B2 = np.array([0.1, 0.2])
```




```
A2 = np.dot(Z1, W2) + B2
Z2 = sigmoid(A2)
```

Signal transmission from 2nd layer to 3rd(Output) layer




```
def identity_function(x):
    return x

W3 = np.array([[0.1, 0.3], [0.2, 0.4]])
B3 = np.array([0.1, 0.2])
```

```
A3 = np.dot(Z2, W3) + B3
Y = identity_function(A3)
```

Summary of implementation

```
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
```

Set the weight and bias

```
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
```

Signal transmission
from input to output

```
network = init_network()
x = np.array([1.0, 0.5])
y = forward(network, x)
print(y) # [ 0.31682708  0.69627909]
```

Summary

- Multi-dimensional array is “Set of number”
- 2-dimensional array is called “matrix”
- Product of matrix make implementation of Neural Network efficient