実装演習：1-3.出力層

In [44]:
```
import numpy as np
from common import functions
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
```

In [45]:
```
def print_vec(text, vec):
    print(f'{text} :\n{vec}')
```

順伝播（3層・複数ユニット）

In [46]:
```
def init_network():
    """重みとバイアスを設定し、ネットワークを作成する。"""
    print("##### ネットワークの初期化 #####")
    network = {}

    network['W1'] = np.array([
        [0.1, 0.3, 0.5],
        [0.2, 0.4, 0.6]
    ])
    network['W2'] = np.array([
        [0.1, 0.4],
        [0.2, 0.5],
        [0.3, 0.6]
    ])
    network['W3'] = np.array([
        [0.1, 0.3],
        [0.2, 0.4]
    ])
    network['b1'] = np.array([0.1, 0.2, 0.3])
    network['b2'] = np.array([0.1, 0.2])
    network['b3'] = np.array([1, 2])

    print_vec('重み1', network['W1'] )
    print_vec('重み2', network['W2'] )
    print_vec('重み3', network['W3'] )
    print_vec('バイアス1', network['b1'] )
    print_vec('バイアス2', network['b2'] )
    print_vec('バイアス3', network['b3'] )

    return network
```

In [47]:
```python
def forward(network, x):
    """プロセスを作成"""
    print("##### 順伝播開始 #####")

    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    # 1層の総入力
    u1 = np.dot(x, W1) + b1
    # 1層の総出力(ReLU)
    z1 = functions.relu(u1)

    # 2層の総入力
    u2 = np.dot(z1, W2) + b2
    # 2層の総出力(ReLU)
    z2 = functions.relu(u2)

    # 出力層の総入力
    u3 = np.dot(z2, W3) + b3
    # 出力層の総出力（そのまま出力）
    y = u3

    print_vec('総入力1', u1)
    print_vec('中間層出力1 (ReLU)', z1)
    print_vec('総入力2', u2)
    print_vec('中間層出力2 (ReLU)', z2)
    print_vec('出力層の総入力', u3)
    print(f"出力合計: {np.sum(y)}")

    return y, z1, z2
```

In [49]:
```python
x = np.array([1., 2.])  # 入力値
network =  init_network()  # ネットワークの初期化
```

```
##### ネットワークの初期化 #####
重み1：
[[0.1 0.3 0.5]
 [0.2 0.4 0.6]]
重み2：
[[0.1 0.4]
 [0.2 0.5]
 [0.3 0.6]]
重み3：
[[0.1 0.3]
 [0.2 0.4]]
バイアス1：
[0.1 0.2 0.3]
バイアス2：
[0.1 0.2]
バイアス3：
[1 2]
```

In [50]:
```python
y, z1, z2 = forward(network, x)  # 順伝搬
```

```
##### 順伝播開始 #####
総入力1：
[0.6 1.3 2. ]
中間層出力1 (ReLU)：
[0.6 1.3 2. ]
総入力2：
[1.02 2.29]
中間層出力2 (ReLU)：
[1.02 2.29]
出力層の総入力：
[1.56  3.222]
出力合計: 4.782
```

考察：

中間層出力はReLUで、すべての値が0より大きいのでそのまま出力されているのがわかる。

多クラス分類（2-3-4ネットワーク）

In [51]:
```
def init_network():
    print('###ネットワーク初期化###')
    network = {}
    network['W1'] = np.array([
        [0.1, 0.3, 0.5],
        [0.2, 0.4, 0.6]
    ])
    network['W2'] = np.array([
        [0.1, 0.4, 0.7, 1.0],
        [0.2, 0.5, 0.8, 1.1],
        [0.3, 0.6, 0.9, 1.2]
    ])
    network['b1'] = np.array([0.1, 0.2, 0.3])
    network['b2'] = np.array([0.1, 0.2, 0.3, 0.4])

    print_vec('重み1', network['W1'])
    print_vec('重み2', network['W2'])
    print_vec('バイアス1', network['b1'])
    print_vec('バイアス2', network['b2'])

    return network
```

In [52]:
```
def forward(network, x):
    print('###順伝搬を開始###')
    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']

    # 1層の総入力
    u1 = np.dot(x, W1) + b1
    # 1層の総出力
    z1 = functions.relu(u1)

    # 出力層の総入力
    u2 = np.dot(z1, W2) + b2
    # 出力値（softmax）
    y = functions.softmax(u2)

    print_vec('総入力1', u1)
    print_vec('中間層出力1', z1)
    print_vec('総入力2', u2)
    print_vec('出力1', y)
    print(f'出力合計: {np.sum(y)}')

    return y, z1
```

In [55]:
```
x = np.array([1., 2.])  # 入力値
d = np.array([[0, 0, 0, 1]])  # 正解ラベル
network =  init_network()
```

```
###ネットワーク初期化###
重み1：
[[0.1 0.3 0.5]
 [0.2 0.4 0.6]]
重み2：
[[0.1 0.4 0.7 1. ]
 [0.2 0.5 0.8 1.1]
 [0.3 0.6 0.9 1.2]]
バイアス1：
[0.1 0.2 0.3]
バイアス2：
[0.1 0.2 0.3 0.4]
```

In [56]:
```
# 順伝搬による出力
y, z1 = forward(network, x)
```

```
###順伝搬を開始###
総入力1：
[0.6 1.3 2. ]
中間層出力1：
[0.6 1.3 2. ]
総入力2：
[1.02 2.29 3.56 4.83]
出力1：
[0.01602796 0.05707321 0.20322929 0.72366954]
出力合計: 1.0
```

In [57]:
```
# 交差エントロピー誤差
loss = functions.cross_entropy_error(d, y)
```

In [58]:
```
print_vec('出力', y)
print_vec('訓練データ', d)
print_vec('誤差',  loss)
```

```
出力：
[0.01602796 0.05707321 0.20322929 0.72366954]
訓練データ：
[[0 0 0 1]]
誤差：
0.3234202933601941
```

考察：

ソフトマックス関数によって、各ラベルの確率を出力している。
出力の合計が1になっている。
出力は4番目のラベルが最も確率が高くなっており(約0.72)、正解ラベルと一致している。

回帰（2-3-2ネットワーク）

In [59]:
```
def init_network():
    print('###ネットワーク初期化###')
    network = {}
    network['W1'] = np.array([
        [0.1, 0.3, 0.5],
        [0.2, 0.4, 0.6]
    ])
    network['W2'] = np.array([
        [0.1, 0.4],
        [0.2, 0.5],
        [0.3, 0.6]
    ])
    network['b1'] = np.array([0.1, 0.2, 0.3])
    network['b2'] = np.array([0.1, 0.2])

    print_vec('重み1', network['W1'])
    print_vec('重み2', network['W2'])
    print_vec('バイアス1', network['b1'])
    print_vec('バイアス2', network['b2'])

    return network
```

In [60]:
```
def forward(network, x):
    print('###順伝搬を開始###')
    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']

    # 1層の総入力
    u1 = np.dot(x, W1) + b1
    # 1層の総出力
    z1 = functions.relu(u1)

    # 出力層の総入力
    u2 = np.dot(z1, W2) + b2
    # 出力値（そのまま出力）
    y = u2

    print_vec('総入力1', u1)
    print_vec('中間層出力1', z1)
    print_vec('総入力2', u2)
    print_vec('出力1', y)
    print(f'出力合計: {np.sum(y)}')

    return y, z1
```

In [61]:
```
x = np.array([1., 2.])  # 入力値
d = np.array([2., 4.])  # 目標値
network =  init_network()
```

```
###ネットワーク初期化###
重み1：
[[0.1 0.3 0.5]
 [0.2 0.4 0.6]]
重み2：
[[0.1 0.4]
 [0.2 0.5]
 [0.3 0.6]]
バイアス1：
[0.1 0.2 0.3]
バイアス2：
[0.1 0.2]
```

In [62]:
```
y, z1 = forward(network, x)
```

```
###順伝搬を開始###
総入力1：
[0.6 1.3 2. ]
中間層出力1：
[0.6 1.3 2. ]
総入力2：
[1.02 2.29]
出力1：
[1.02 2.29]
出力合計: 3.31
```

In [63]:
```
# 誤差（MSE）
loss = functions.mean_squared_error(d, y)
```

In [64]:
```
print_vec('中間層出力', z1)
print_vec('出力値', y)
print_vec('正解データ', d)
print_vec('誤差', loss)
```

```
中間層出力：
[0.6 1.3 2. ]
出力値：
[1.02 2.29]
正解データ：
[2. 4.]
誤差：
0.9711249999999999
```

考察：

回帰では、出力層ではそのまま値を出力している。
入力値に対して出力値は少し大きい値になっており、正解データに少し近づいているように見える。
入力値や重み・バイアスをより改善し、より正解データに近い出力をできるようにする必要がある。

2値分類（2-3-1ネットワーク）

In [65]:
```
def init_network():
    print('###ネットワーク初期化###')
    network['W1'] = np.array([
        [0.1, 0.3, 0.5],
        [0.2, 0.4, 0.6]
    ])
    network['W2'] = np.array([
        [0.2],
        [0.4],
        [0.6]
    ])
    network['b1'] = np.array([0.1, 0.2, 0.3])
    network['b2'] = np.array([0.1])

    print_vec('重み1', network['W1'])
    print_vec('重み2', network['W2'])
    print_vec('バイアス1', network['b1'])
    print_vec('バイアス2', network['b2'])

    return network
```

In [66]:
```
def forward(network, x):
    print('###順伝搬を開始###')
    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']

    # 1層の総入力
    u1 = np.dot(x, W1) + b1
    # 1層の総出力
    z1 = functions.relu(u1)

    # 出力層の総入力
    u2 = np.dot(z1, W2) + b2
    # 出力値（そのまま出力）
    y = functions.sigmoid(u2)

    print_vec('総入力1', u1)
    print_vec('中間層出力1', z1)
    print_vec('総入力2', u2)
    print_vec('出力1', y)
    print(f'出力合計: {np.sum(y)}')

    return y, z1
```

In [67]:
```
x = np.array([1., 2.])  # 入力値
d = np.array([1])  # 目標出力
```

In [68]:
```
network = init_network()
```

```
###ネットワーク初期化###
重み1：
[[0.1 0.3 0.5]
 [0.2 0.4 0.6]]
重み2：
[[0.2]
 [0.4]
 [0.6]]
バイアス1：
[0.1 0.2 0.3]
バイアス2：
[0.1]
```

In [69]:
```
y, z1 = forward(network, x)
```

```
###順伝搬を開始###
総入力1：
[0.6 1.3 2. ]
中間層出力1：
[0.6 1.3 2. ]
総入力2：
[1.94]
出力1：
[0.87435214]
出力合計: 0.8743521434846544
```

In [70]:
```
# 誤差（交差エントロピー誤差）
loss = functions.cross_entropy_error(d, y)
```

In [71]:
```
print_vec('中間層出力', z1)
print_vec('出力値', y)
print_vec('正解データ', d)
print_vec('誤差', loss)
```

```
中間層出力 :
[0.6 1.3 2. ]
出力値 :
[0.87435214]
正解データ :
[1]
誤差 :
0.13427195993720972
```

考察：

2値分類では、sigmoid関数により出力値が0か1どちらに近いか分類する。
出力値は0.87で1に近いので、正しく分類しているといえる。

In [ ]: