実装演習：1-5.誤差逆伝搬法

In [1]:
```
import numpy as np
from common import functions
import matplotlib.pyplot as plt
```

In [16]:
```
def print_vec(text, vec):
    print(f'{text} : ')
    print(f'{vec}\n')
```

In [9]:
```
def init_network():
    """ネットワークの初期化
    ウェイトとバイアスの設定
    """
    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 [10]:
```
def forward(network, x):
    """順伝搬"""
    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']

    u1 = np.dot(x, W1) + b1
    z1 = functions.relu(u1)
    u2 = np.dot(z1, W2) + b2
    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 [21]:
```
def backward(x, d, z1, y):
    """逆誤差伝搬"""
    grad = {}  # 勾配

    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']
    #  出力層でのデルタ: 目標値から予測値を引いている
    delta2 = functions.d_sigmoid_with_loss(d, y)
    #  b2の勾配
    grad['b2'] = np.sum(delta2, axis=0)
    #  W2の勾配
    grad['W2'] = np.dot(z1.T, delta2)
    #  中間層でのデルタ
    delta1 = np.dot(delta2, W2.T) * functions.d_relu(z1)
    # b1の勾配
    grad['b1'] = np.sum(delta1, axis=0)
    #  W1の勾配
    grad['W1'] = np.dot(x.T, delta1)

    print_vec("偏微分_dE/du2", delta2)
    print_vec("偏微分_dE/du2", delta1)

    print_vec("偏微分_重み1", grad["W1"])
    print_vec("偏微分_重み2", grad["W2"])
    print_vec("偏微分_バイアス1", grad["b1"])
    print_vec("偏微分_バイアス2", grad["b2"])

    return grad
```

In [12]:
```
# 訓練データ
x = np.array([[1.0, 5.0]])
# 目標出力
d = np.array([[0, 1]])
#  学習率
learning_rate = 0.01
```

In [17]:
```
# ネットワーク初期化
network =  init_network()
```

```
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 [18]:
```
# 順伝搬
y, z1 = forward(network, x)
```

```
総入力1：
[[1.2 2.5 3.8]]

中間層出力1：
[[1.2 2.5 3.8]]

総入力2：
[[1.86 4.21]]

出力1：
[[0.08706577 0.91293423]]

出力合計: 1.0
```

In [19]:
```
# 交差エントロピー誤差
loss = functions.cross_entropy_error(d, y)
```

In [20]:
```
print(loss)
```

```
0.09109133135793131
```

In [22]:
```
# 逆誤差伝搬
grad = backward(x, d, z1, y)
```

```
偏微分_dE/du2：
[[ 0.08706577 -0.08706577]]

偏微分_dE/du2：
[[-0.02611973 -0.02611973 -0.02611973]]

偏微分_重み1：
[[-0.02611973 -0.02611973 -0.02611973]
 [-0.13059866 -0.13059866 -0.13059866]]

偏微分_重み2：
[[ 0.10447893 -0.10447893]
 [ 0.21766443 -0.21766443]
 [ 0.33084994 -0.33084994]]

偏微分_バイアス1：
[-0.02611973 -0.02611973 -0.02611973]

偏微分_バイアス2：
[ 0.08706577 -0.08706577]
```

In [23]:
```
# パラメータ更新
for key in ('W1', 'W2', 'b1', 'b2'):
    network[key]  -= learning_rate * grad[key]
```

In [24]:
```
# 逆誤差伝搬による更新後のパラメータ
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'])
```

```
重み1：
[[0.1002612  0.3002612  0.5002612 ]
 [0.20130599 0.40130599 0.60130599]]

重み2：
[[0.09895521 0.40104479]
 [0.19782336 0.50217664]
 [0.2966915  0.6033085 ]]

バイアス1：
[0.1002612 0.2002612 0.3002612]

バイアス2：
[0.09912934 0.20087066]
```

考察

更新前と更新後のパラメータ変化を見ていく。

- 重み1は更新前より値が少し大きくなった。
- 重み2は更新前より値が少し小さくなった。
- バイアス1は更新前より値が少し大きくなった。
- バイアス2は更新前より値が少し小さくなった。

In [ ]: