Section 1 入力層～中間層

Section 2 活性化関数

Section 3 出力層

In [ ]:

```
```

In [ ]:

```
import numpy as np
from common import functions
```

In [20]:

```
def print_vec(text, vec):
    print("*** " + text + " ***")
    print(vec)
    print("shape: " + str(vec.shape))
    print("")
```

In [25]:

```
# 重み
# W = np.array([[0.1], [0.2]])

## 試してみよう_配列の初期化
# W = np.zeros(2)
#W = np.ones(2)
#W = np.random.rand(2)
W = np.random.randint(5, size=(2))

print_vec("重み", W)
```

```
*** 重み ***
[2 3]
shape: (2,)
```

In [22]:

```
# バイアス
b = np.array(0.5)

## 試してみよう_数値の初期化
#b = np.random.rand() # 0~1のランダム数値
#b = np.random.rand() * 10 -5  # -5~5のランダム数値

print_vec("バイアス", b)
```

```
*** バイアス ***
0.5
shape: ()
```

In [32]:

```
# 入力値
x = np.array([2, -3])
print_vec("入力", x)

# 総入力
u = np.dot(x, W) + b
print_vec("総入力", u)

# 中間層出力:ReLU
z = functions.relu(u)
print_vec("中間層出力", z)

# 中間層出力:sigmoid
```

```
z = functions.sigmoid(u)
print_vec("中間層出力", z)
```

```
*** 入力 ***
[ 2 -3]
shape: (2,)

*** 総入力 ***
-4.5
shape: ()

*** 中間層出力 ***
0.0
shape: ()

*** 中間層出力 ***
0.01098694263059318
shape: ()
```

In [30]:

```
# 多クラス分類
def init_network():
    print("##### ネットワークの初期化 #####")

    network = {}

    input_layer_size = 3
    hidden_layer_size = 5
    output_layer_size = 6

    network['W1'] = np.random.rand(input_layer_size, hidden_layer_size)
    network['W2'] = np.random.rand(hidden_layer_size,output_layer_size)

    network['b1'] =  np.random.rand(hidden_layer_size)
    network['b2'] =  np.random.rand(output_layer_size)

    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

def forward(network, x):

    print("##### 順伝播開始 #####")
    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)

    # 2層の総入力
    u2 = np.dot(z1, W2) + b2

    # 出力値
#     y = functions.softmax(u2)
    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("出力合計: " + str(np.sum(y)))
```

```
    return y, z1
```

In [31]:

```
## 事前データ
# 入力値
x = np.array([1., 2., 3.])

# 目標出力
d = np.array([0, 0, 0, 1, 0, 0])

# ネットワークの初期化
network =  init_network()

# 出力
y, z1 = forward(network, x)

# 誤差
loss = functions.cross_entropy_error(d, y)

## 表示
print("¥n##### 結果表示 #####")
print_vec("出力", y)
print_vec("訓練データ", d)
print_vec("交差エントロピー誤差",  loss)
```

```
##### ネットワークの初期化 #####
*** 重み1 ***
[[0.92815961 0.13142438 0.33857123 0.20504201 0.68471141]
 [0.69614507 0.24308604 0.78286638 0.40417049 0.16572272]
 [0.93904484 0.09609288 0.90775857 0.05073835 0.362631  ]]
shape: (3, 5)

*** 重み2 ***
[[0.608667   0.87846321 0.80330869 0.16546431 0.23737385 0.78854372]
 [0.0982093  0.61263932 0.90076926 0.95694635 0.00491237 0.18284198]
 [0.99535831 0.95086403 0.46396797 0.18442008 0.53046818 0.13520954]
 [0.24123805 0.96859557 0.58769259 0.14958729 0.34256739 0.4508791 ]
 [0.31395047 0.85910494 0.15362558 0.42611879 0.50723875 0.81097681]]
shape: (5, 6)

*** バイアス1 ***
[0.00772914 0.38419433 0.01764879 0.22818145 0.80761383]
shape: (5,)

*** バイアス2 ***
[0.47170142 0.02003014 0.1743918  0.33042609 0.19827644 0.6504457 ]
shape: (6,)

##### 順伝播開始 #####
*** 総入力1 ***
[5.14531342 1.29006942 4.6452285  1.39377949 2.91166368]
shape: (5,)

*** 中間層出力1 ***
[5.14531342 1.29006942 4.6452285  1.39377949 2.91166368]
shape: (5,)

*** 総入力2 ***
[ 9.60419831 13.59875991  8.89137879  4.72219877  5.84449454  8.56142609]
shape: (6,)

*** 出力1 ***
[ 9.60419831 13.59875991  8.89137879  4.72219877  5.84449454  8.56142609]
shape: (6,)

出力合計: 51.22245640673955

##### 結果表示 #####
*** 出力 ***
[ 9.60419831 13.59875991  8.89137879  4.72219877  5.84449454  8.56142609]
shape: (6,)
```

```
*** 訓練データ ***
[0 0 0 1 0 0]
shape: (6,)

*** 交差エントロピー誤差 ***
-1.552274552592688
shape: ()
```

様々なパターンの順伝播を試した。中間層の層の数でかなり表現力を増すことが確認できた。
人間に近い理解の仕方をするなら、入力の特徴量を中間層でさらに細かい特徴に分解して、得たい出力に向けて出力層で再整理するといった感じだろうか。
また、活性化関数によって変わる出力値の特徴も把握できた。