In [1]:
```
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import numpy as np
from common import functions

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

In [2]:
```
# 順伝播（単層・単ユニット）

# 重み
W = np.array([[0.1], [0.2]])
```

試してみよう_配列の初期化

In [3]:
```
W = np.zeros(2)
print_vec("重み", W)

W = np.ones(2)
print_vec("重み", W)

W = np.random.rand(2)
print_vec("重み", W)

W = np.random.randint(5, size=(2))
print_vec("重み", W)
```

```
*** 重み ***
[0. 0.]

*** 重み ***
[1. 1.]

*** 重み ***
[0.42680789 0.29673291]

*** 重み ***
[0 4]
```

In [4]:
```
# バイアス
b = 0.5
```

試してみよう_数値の初期化

In [5]:
```
b = np.random.rand() # 0~1のランダム数値
print_vec("バイアス", b)

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 ***
0.779637815684484

*** バイアス ***
-1.3288264982829991
```

In [6]:
```
# 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
z = functions.relu(u)
print_vec("中間層出力", z)
```

```
*** 入力 ***
[2 3]

*** 総入力 ***
10.671173501717

*** 中間層出力 ***
10.671173501717
```

In [7]:
```
# 順伝播（単層・複数ユニット）

# 重み
W = np.array([
    [0.1, 0.2, 0.3],
    [0.2, 0.3, 0.4],
    [0.3, 0.4, 0.5],
    [0.4, 0.5, 0.6]
])
```

試してみよう_配列の初期化

In [8]:
```
W = np.zeros((4,3))
print_vec("重み", W)

W = np.ones((4,3))
print_vec("重み", W)

W = np.random.rand(4,3)
print_vec("重み", W)

W = np.random.randint(5, size=(4,3))
print_vec("重み", W)
```

```
*** 重み ***
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

*** 重み ***
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

*** 重み ***
[[0.33946146 0.61316861 0.16153367]
 [0.33491718 0.62390793 0.33000408]
 [0.55118424 0.66420791 0.33499879]
 [0.23079658 0.94532177 0.91777449]]

*** 重み ***
[[0 2 4]
 [3 2 2]
 [2 2 0]
 [0 3 4]]
```

In [9]:
```
# バイアス
b = np.array([0.1, 0.2, 0.3])
print_vec("バイアス", b)

# 入力値
x = np.array([1.0, 5.0, 2.0, -1.0])
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
z = functions.sigmoid(u)
print_vec("中間層出力", z)
```

```
*** バイアス ***
[0.1 0.2 0.3]

*** 入力 ***
[ 1.  5.  2. -1.]

*** 総入力 ***
[19.1 13.2 10.3]

*** 中間層出力 ***
[0.99999999 0.99999815 0.99996637]
```

試してみよう

_各パラメータのshapeを表示

_ネットワークの初期値ランダム生成

In [10]:
```
# 順伝播（3層・複数ユニット）

def print_shape(x,y):
    print(x + " shape: " + str(y.shape) + '¥n¥n')
    return

# ウェイトとバイアスを設定
# ネートワークを作成
def init_network():
    print("##### ネットワークの初期化 #####")
    network = {}

    #試してみよう
    #_各パラメータのshapeを表示
    #_ネットワークの初期値ランダム生成

#    network['W1'] = np.array([
#        [0.1, 0.3, 0.5],
#        [0.2, 0.4, 0.6]
#    ])
    network['W1'] = np.random.rand(2,3)

#    network['W2'] = np.array([
#        [0.1, 0.4],
#        [0.2, 0.5],
#        [0.3, 0.6]
#    ])
    network['W2'] = np.random.rand(3,2)

#    network['W3'] = np.array([
#        [0.1, 0.3],
#        [0.2, 0.4]
#    ])

    network['W3'] = np.random.rand(2,2)

    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_shape("重み1", network['W1'] )

    print_vec("重み2", network['W2'] )
    print_shape("重み2", network['W2'] )

    print_vec("重み3", network['W3'] )
    print_shape("重み3", network['W3'] )

    print_vec("バイアス1", network['b1'] )
    print_shape("バイアス1", network['b1'] )

    print_vec("バイアス2", network['b2'] )
    print_shape("バイアス2", network['b2'] )

    print_vec("バイアス3", network['b3'] )
    print_shape("バイアス3", network['b3'] )

    return network

# プロセスを作成
# x：入力値
def forward(network, x):

    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
    z1 = functions.relu(u1)

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

    # 2層の総出力
    z2 = functions.relu(u2)

    # 出力層の総入力
    u3 = np.dot(z2, W3) + b3
```

```
    # 出力層の総出力
    y = u3

    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("総出力2", z2)
    print_vec("総入力3", u3)
    print_vec("出力層総出力", y)
    print("出力合計: " + str(np.sum(y)))

    return y, z1, z2

# 入力値
x = np.array([1., 2.])
print_vec("入力", x)

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

y, z1, z2 = forward(network, x)
```

```
*** 入力 ***
[1. 2.]

##### ネットワークの初期化 #####
*** 重み1 ***
[[0.85746737 0.92107776 0.60686596]
 [0.42130121 0.89094784 0.18093196]]

重み1 shape: (2, 3)


*** 重み2 ***
[[0.07440062 0.57030604]
 [0.29334927 0.66276169]
 [0.746431   0.83192516]]

重み2 shape: (3, 2)


*** 重み3 ***
[[0.81861707 0.64556676]
 [0.71586396 0.50961949]]

重み3 shape: (2, 2)


*** バイアス1 ***
[0.1 0.2 0.3]

バイアス1 shape: (3,)


*** バイアス2 ***
[0.1 0.2]

バイアス2 shape: (2,)


*** バイアス3 ***
[1 2]
```

```
バイアス3 shape: (2,)


##### 順伝播開始 #####
*** 総入力1 ***
[1.8000698  2.90297343 1.26872987]

*** 中間層出力1 ***
[1.8000698  2.90297343 1.26872987]

*** 総入力2 ***
[2.03253077 4.20605855]

*** 総出力2 ***
[2.03253077 4.20605855]

*** 総入力3 ***
[5.67483011 5.4556237 ]

*** 出力層総出力 ***
[5.67483011 5.4556237 ]

出力合計: 11.130453810995402
```

！試してみよう

_ノードの構成を 3-5-4 に変更してみよう

_各パラメータのshapeを表示

_ネットワークの初期値ランダム生成

In [11]:
```
# 多クラス分類
# 2-3-4ネットワーク

# ！試してみよう_ノードの構成を 3-5-4 に変更してみよう
def print_shape(x,y):
    print(x + " shape: " + str(y.shape) + '¥n¥n')
    return

# ウェイトとバイアスを設定
# ネートワークを作成
def init_network():
    print("##### ネットワークの初期化 #####")

    #試してみよう
    #_各パラメータのshapeを表示
    #_ネットワークの初期値ランダム生成

    network = {}
#    network['W1'] = np.array([
#        [0.1, 0.3, 0.5],
#        [0.2, 0.4, 0.6]
#    ])
    network['W1'] = np.random.rand(3,5)

#    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['W2'] = np.random.rand(5,4)

#    network['b1'] = np.array([0.1, 0.2, 0.3])
#    network['b2'] = np.array([0.1, 0.2, 0.3, 0.4])
    network['b1'] = np.random.rand(5)
    network['b2'] = np.random.rand(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

# プロセスを作成
# x：入力値
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)

    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("出力層総出力", y)
    print("出力合計: " + str(np.sum(y)))

    return y, z1

## 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])

# 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("誤差",  loss)
```

```
##### ネットワークの初期化 #####
*** 重み1 ***
[[0.59521137 0.42329308 0.49410429 0.85500816 0.54574664]
 [0.37797661 0.65169932 0.81321915 0.34013626 0.21471331]
 [0.35232141 0.61716249 0.45068161 0.59219522 0.15863794]]

*** 重み2 ***
[[0.86361957 0.39361017 0.09772839 0.49313854]
 [0.4363036  0.59601771 0.10763414 0.44753731]
 [0.40568117 0.14220269 0.50426579 0.79462152]
 [0.12207393 0.66662244 0.01277669 0.60741746]
 [0.98531338 0.95798052 0.43026535 0.75213123]]

*** バイアス1 ***
[0.06689027 0.05893366 0.28862973 0.07662762 0.48081069]

*** バイアス2 ***
[0.4258724  0.35725269 0.53690444 0.26717601]

##### 順伝播開始 #####
*** 総入力1 ***
[2.47501909 3.63711286 3.76121715 3.38849394 1.93189776]

*** 中間層出力1 ***
[2.47501909 3.63711286 3.76121715 3.38849394 1.93189776]

*** 総入力2 ***
[7.99325922 8.14365079 3.94143716 9.61546213]

*** 出力層総出力 ***
[0.13804692 0.16045045 0.00240073 0.69910191]

出力合計: 1.0

##### 結果表示 #####
*** 出力 ***
[0.13804692 0.16045045 0.00240073 0.69910191]

*** 訓練データ ***
[0 0 0 1]
```

```
*** 誤差 ***
0.35795861517227934
```

！試してみよう_ノードの構成を 3-5-4 に変更してみよう

In [12]:
```
# 回帰
# 2-3-2ネットワーク

# ！試してみよう_ノードの構成を 3-5-4 に変更してみよう

# ウェイトとバイアスを設定
# ネートワークを作成
def init_network():
    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['W1'] = np.random.rand(3,5)

#    network['W2'] = np.array([
#        [0.1, 0.4],
#        [0.2, 0.5],
#        [0.3, 0.6]
#    ])
    network['W2'] = np.random.rand(5,4)

#    network['b1'] = np.array([0.1, 0.2, 0.3])
#    network['b2'] = np.array([0.1, 0.2])
    network['b1'] = np.random.rand(5)
    network['b2'] = np.random.rand(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

# プロセスを作成
def forward(network, x):
    print("##### 順伝播開始 #####")

    W1, W2 = network['W1'], network['W2']
```

```
    b1, b2 = network['b1'], network['b2']
    # 隠れ層の総入力
    u1 = np.dot(x, W1) + b1
    # 隠れ層の総出力
    z1 = functions.relu(u1)
    # 出力層の総入力
    u2 = np.dot(z1, W2) + b2
    # 出力層の総出力
    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("出力層総出力", y)
    print("出力合計: " + str(np.sum(y)))

    return y, z1

# 入力値
x = np.array([1., 2., 3.])
network =  init_network()
y, z1 = forward(network, x)
# 目標出力
d = np.array([2., 4., 4., 4.])
# 誤差
loss = functions.mean_squared_error(d, y)
## 表示
print("¥n##### 結果表示 #####")
print_vec("中間層出力", z1)
print_vec("出力", y)
print_vec("訓練データ", d)
print_vec("誤差",  loss)
```

```
##### ネットワークの初期化 #####
*** 重み1 ***
[[0.82311732 0.76172543 0.45076582 0.07246605 0.19765454]
 [0.42364796 0.35521163 0.3203457  0.13138572 0.47220887]
 [0.5055735  0.44130895 0.19353403 0.12327422 0.05197146]]

*** 重み2 ***
[[0.97268545 0.35956599 0.060448   0.86028885]
 [0.11946265 0.36019375 0.95088856 0.90503981]
 [0.82542546 0.04205813 0.48726237 0.30929281]
 [0.18196637 0.18660259 0.54452348 0.66104057]
 [0.36836011 0.74288927 0.42408416 0.85233429]]

*** バイアス1 ***
[0.40346029 0.47463927 0.90660468 0.63434628 0.62866421]

*** バイアス2 ***
[0.99846423 0.23395357 0.87139515 0.37879932]

##### 順伝播開始 #####
*** 総入力1 ***
[3.59059401 3.27071481 2.57866397 1.33940643 1.92665087]

*** 中間層出力1 ***
[3.59059401 3.27071481 2.57866397 1.33940643 1.92665087]

*** 総入力2 ***
[7.96363415 4.49277885 7.00141092 9.75298922]

*** 出力層総出力 ***
[7.96363415 4.49277885 7.00141092 9.75298922]

出力合計: 29.210813139371083

##### 結果表示 #####
*** 中間層出力 ***
[3.59059401 3.27071481 2.57866397 1.33940643 1.92665087]

*** 出力 ***
[7.96363415 4.49277885 7.00141092 9.75298922]
```

```
*** 訓練データ ***
[2. 4. 4. 4.]

*** 誤差 ***
9.739139469476706
```

！試してみよう_ノードの構成を 5-10-1 に変更してみよう

In [13]:
```
# 2値分類
# 2-3-1ネットワーク

# ！試してみよう_ノードの構成を 5-10-1 に変更してみよう

# ウェイトとバイアスを設定
# ネートワークを作成
def init_network():
    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['W1'] = np.random.rand(5,10)

#    network['W2'] = np.array([
#        [0.2],
#        [0.4],
#        [0.6]
#    ])
    network['W2'] = np.random.rand(10,1)

#    network['b1'] = np.array([0.1, 0.2, 0.3])
#    network['b2'] = np.array([0.1])
    network['b1'] = np.random.rand(10)
    network['b2'] = np.random.rand(1)

    return network


# プロセスを作成
def forward(network, x):
    print("##### 順伝播開始 #####")

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

    # 隠れ層の総入力
    u1 = np.dot(x, W1) + b1
```

```
    # 隠れ層の総出力
    z1 = functions.relu(u1)
    # 出力層の総入力
    u2 = np.dot(z1, W2) + b2
    # 出力層の総出力
    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("出力層総出力y", y)
    print("出力合計: " + str(np.sum(y)))

    return y, z1


# 入力値
x = np.array([1., 2., 3., 4., 5.])
# 目標出力
d = np.array([1])
network =  init_network()
y, z1 = forward(network, x)
# 誤差
loss = functions.cross_entropy_error(d, y)

## 表示
print("¥n##### 結果表示 #####")
print_vec("中間層出力", z1)
print_vec("出力", y)
print_vec("訓練データ", d)
print_vec("誤差",  loss)
```

```
##### ネットワークの初期化 #####
##### 順伝播開始 #####
*** 総入力1 ***
[ 6.5503636  12.72181333  6.08338927  9.862982    5.5698028   8.1540648
  6.99822309 10.64632732  7.65897975  7.95182093]

*** 中間層出力1 ***
[ 6.5503636  12.72181333  6.08338927  9.862982    5.5698028   8.1540648
  6.99822309 10.64632732  7.65897975  7.95182093]

*** 総入力2 ***
[53.02671447]

*** 出力層総出力y ***
[1.]

出力合計: 1.0

##### 結果表示 #####
*** 中間層出力 ***
[ 6.5503636  12.72181333  6.08338927  9.862982    5.5698028   8.1540648
  6.99822309 10.64632732  7.65897975  7.95182093]

*** 出力 ***
[1.]

*** 訓練データ ***
[1]

*** 誤差 ***
-9.999999505838704e-08
```