第６章　演習１

▶ 下記のプログラムリストを未完成プログラムリストのコメントに従って実装してください

👇プログラムリスト

In [1]:

```
# numpy インポート　別名 np
import numpy as np
# 乱数のシード（種）を設定する
# 同じパターンの乱数を生成する
np.random.seed(1)

# x が正なら x を、x が負なら 0 を返す関数 relu の定義
def relu(x):
  return (x > 0) * x

# output が正なら 1 を、負なら 0 を返す関数 relu2derive 関数の定義
def relu2deriv(output):
  return output > 0 # returns 1 for input > 0
                    # return 0 otherwise

# 入力層（= 信号パターン）の初期化
lights = np.array( [[ 1, 0, 1 ],
                    [ 0, 1, 1 ],
                    [ 0, 0, 1 ],
                    [ 1, 1, 1 ] ] )
# 入力層の表示
print('入力層 = ')
print(lights)
# 目的値（出力層）の初期化、転置行列にする
'''
[[1, 1, 0, 0]].T
    ⇓
[ [1],
  [1],
  [0],
  [0] ]
'''
walk_stop = np.array([[ 1, 1, 0, 0]]).T
# 目的値（出力層）を表示
print('目的値（出力層）= ')
print(walk_stop)
# アルファの初期化
alpha = 0.2
# 中間層のユニット数（= ノード数）
hidden_size = 3
# ３行３列の乱数を生成して weights_0_1（= 入力層と中間層間の重み）に設定
weights_0_1 = 2 * np.random.random((3, hidden_size)) - 1
# ３行１列の乱数を生成して weights_1_2（= 中間層と出力層間の重み）に設定
weights_1_2 = 2 * np.random.random((hidden_size, 1)) - 1

# 入力層と中間層間の重みの表示
print('入力層と中間層間の重み = ')
print(weights_0_1)
# 中間層と出力層間の重みの表示
print('中間層と出力層間の重み = ')
print(weights_1_2)

# 入力層に値を設定（信号パターン１）
layer_0 = lights[0:1]
```

```
# 入力層を表示
print('入力層 = {}'.format(layer_0))
# 入力層の予測を求めて、中間層に設定
layer_1 = np.dot(layer_0, weights_0_1)
# 中間層を表示
print('中間層 = {}'.format(layer_1))
# 中間層の値において、負の値は0にする
layer_1 = relu(layer_1)
# 再度中間層を表示
print('調整後の中間層 = {}'.format(layer_1))
# 中間層の予測を求めて、出力層に設定
layer_2 = np.dot(layer_1, weights_1_2)
# 出力層を表示
print('出力層 = {}'.format(layer_2))
# 中間層と出力層間のデルタを求める
layer_2_delta = (layer_2 - walk_stop[0:1])

# layer_2 から layer_1 へ逆伝播
# layer_2 のデルタをもとに layer_1 のデルタを求める
layer_1_delta = layer_2_delta.dot(weights_1_2.T)
# layer_1_delta を修正（負の値は0にする）
layer_1_delta *= relu2deriv(layer_1)

# 出力層のデルタ layer_2_delta を使用して、
# 中間層と出力層間の重み微調整量 weight_delta_1_2 を求める
weight_delta_1_2 = layer_1.T.dot(layer_2_delta)
# 中間層のデルタ layer_1_delta を使用して、
# 入力層と中間層間の重みの微調整量 weight_delta_0_1 を求める
weight_delta_0_1 = layer_0.T.dot(layer_1_delta)
# 中間層と出力層間の重みを更新する
weights_1_2 -= alpha * weight_delta_1_2
# 入力層と中間層間の重みを更新する
weights_0_1 -= alpha * weight_delta_0_1
# 更新後の中間層と出力層間の重みを表示
print('更新後の中間層と出力層間の重み =')
print(weights_1_2)
# 更新後の入力層と中間層間の重みを表示
print('更新後の入力層と中間層間の重み =')
print(weights_0_1)
```

```
入力層 =
[[1 0 1]
 [0 1 1]
 [0 0 1]
 [1 1 1]]
目的値（出力層）=
[[1]
 [1]
 [0]
 [0]]
入力層と中間層間の重み =
[[-0.16595599  0.44064899 -0.99977125]
 [-0.39533485 -0.70648822 -0.81532281]
 [-0.62747958 -0.30887855 -0.20646505]]
中間層と出力層間の重み =
[[ 0.07763347]
 [-0.16161097]
 [ 0.370439  ]]
入力層 = [[1 0 1]]
中間層 = [[-0.79343557  0.13177044 -1.2062363 ]]
調整後の中間層 = [[-0.          0.13177044 -0.        ]]
出力層 = [[-0.02129555]]
更新後の中間層と出力層間の重み =
[[ 0.07763347]
 [-0.13469566]
 [ 0.370439  ]]
更新後の入力層と中間層間の重み =
[[-0.16595599  0.40763847 -0.99977125]
 [-0.39533485 -0.70648822 -0.81532281]
 [-0.62747958 -0.34188906 -0.20646505]]
```

👇未完成プログラムリスト

In [3]:

```
# numpy インポート　別名 np
import numpy as np
# 乱数のシード（種）を設定する
# 同じパターンの乱数を生成する
np.random.seed(1)

# x が正なら x を、x が負なら 0 を返す関数 relu の定義
def relu(x):
  return (x > 0) * x

# output が正なら 1 を、負なら 0 を返す関数 relu2derive 関数の定義
def relu2deriv(output):
  return output > 0 # returns 1 for input > 0
                    # return 0 otherwise

# 予測を行う関数 neural_network(input, weight) の定義
'''
関数名：neural_network
引数：input = 入力データセットリスト、weight = 重み行列
処理：input と weight の加重和を計算する
戻り値：加重和リスト
'''

  # numpy の dot メソッドを使用して、input と weight の加重和を計算する
  pred =
  # 加重和を返す


# 乱数発生により重み行列を生成する関数 create_weight(layer_1_num, layer_2_num) の定義
'''
関数名：create_weight
引数：layer_1_num = 層１の長さ、layer_2_num = 層２の長さ
処理：乱数を発生させて、layer_1_num 行、layer_2_num 列の行列に重みを設定する
戻り値：重み行列
'''

  # layer_1 の要素数の行、layer_2 の要素数の列の行列に、乱数発生による重みを設定
  weight_1_2 =
  # 重み行列を返す


# 誤差逆伝播法による学習関数
# back_propagation(input, hidden, output, goal, weight_i_h, weight_h_o) の定義
'''
関数名：back_propagation
引数：
  input = 入力層のデータセットリスト
  hidden = 中間層の予測値リスト
  output = 出力層の予測値リスト
  goal = 目的値行列
  weight_i_h = 入力層と中間層間の重み行列
  weight_h_o = 中間層と出力層間の重み行列
  input_num = 入力層に与えられるパターン番号
処理：誤差逆伝播法により学習する
戻り値：
  更新した入力層と中間層の重み行列と、更新した中間層と出力層間の重み行列
'''

  # 出力層のデルタを計算する
  output_delta =
  # 出力層のデルタを使用して中間層のデルタを計算する（逆伝播）
  hidden_delta =
```

```
  # 中間層のデルタを修正
  hidden_delta *= relu2deriv(hidden)
  # 中間層と出力層間の重みの微調整量を計算
  weight_delta_h_o =
  # 入力層と中間層間の重みの微調整量を計算
  weight_delta_i_h =
  # 中間層と出力層間の重みを更新する

  # 入力層と中間層間の重みを更新する

  # 更新した入力層と中間層間の重みと、更新した中間層と出力層間の重みを返す
  return weight_i_h, weight_h_o

# 入力層（= 信号パターン）の初期化
lights = np.array( [[ 1, 0, 1 ],
                    [ 0, 1, 1 ],
                    [ 0, 0, 1 ],
                    [ 1, 1, 1 ] ] )
# 入力層の表示
print('入力層 = ')
print(lights)
# 目的値（出力層）の初期化、転置行列にする
'''
[[1, 1, 0, 0]].T
    ⇓
[ [1],
  [1],
  [0],
  [0] ]
'''
walk_stop = np.array([[ 1, 1, 0, 0]]).T
# 目的値（出力層）を表示
print('目的値（出力層）= ')
print(walk_stop)
# アルファの初期化
alpha = 0.2
# 中間層のユニット数（= ノード数）
hidden_size = 3
# 乱数を生成して weights_i_h（= 入力層と中間層間の重み）に設定
weight_i_h =
# 乱数を生成して weights_h_o（= 中間層と出力層間の重み）に設定
weight_h_o =
# 入力層と中間層間の重みの表示
print('入力層と中間層間の重み = ')
print(weight_i_h)
# 中間層と出力層間の重みの表示
print('中間層と出力層間の重み = ')
print(weight_h_o)

# 入力層に値を設定（信号パターン１）
layer_0 = lights[0:1]
# 入力層を表示
print('入力層 = {}'.format(layer_0))
# 入力層の予測を求めて、中間層に設定
layer_1 =
# 中間層を表示
print('中間層 = {}'.format(layer_1))
# 中間層の値において、負の値は０にする
layer_1 = relu(layer_1)
# 再度中間層を表示
print('調整後の中間層 = {}'.format(layer_1))
# 中間層の予測を求めて、出力層に設定
layer_2 =
# 出力層を表示
print('出力層 = {}'.format(layer_2))
```

```
# 誤差逆伝播法で学習する

# 更新後の中間層と出力層間の重みを表示
print('更新後の中間層と出力層間の重み = ')
print(weight_h_o)
# 更新後の入力層と中間層間の重みを表示
print('更新後の入力層と中間層間の重み = ')
print(weight_i_h)
```

```
入力層 =
[[1 0 1]
 [0 1 1]
 [0 0 1]
 [1 1 1]]
目的値（出力層）=
[[1]
 [1]
 [0]
 [0]]
入力層と中間層間の重み =
[[-0.16595599  0.44064899 -0.99977125]
 [-0.39533485 -0.70648822 -0.81532281]
 [-0.62747958 -0.30887855 -0.20646505]]
中間層と出力層間の重み =
[[ 0.07763347]
 [-0.16161097]
 [ 0.370439  ]]
入力層 = [[1 0 1]]
中間層 = [[-0.79343557  0.13177044 -1.2062363 ]]
調整後の中間層 = [[-0.          0.13177044 -0.        ]]
出力層 = [[-0.02129555]]
更新後の中間層と出力層間の重み =
[[ 0.07763347]
 [-0.13469566]
 [ 0.370439  ]]
更新後の入力層と中間層間の重み =
[[-0.16595599  0.40763847 -0.99977125]
 [-0.39533485 -0.70648822 -0.81532281]
 [-0.62747958 -0.34188906 -0.20646505]]
```

In [ ]: