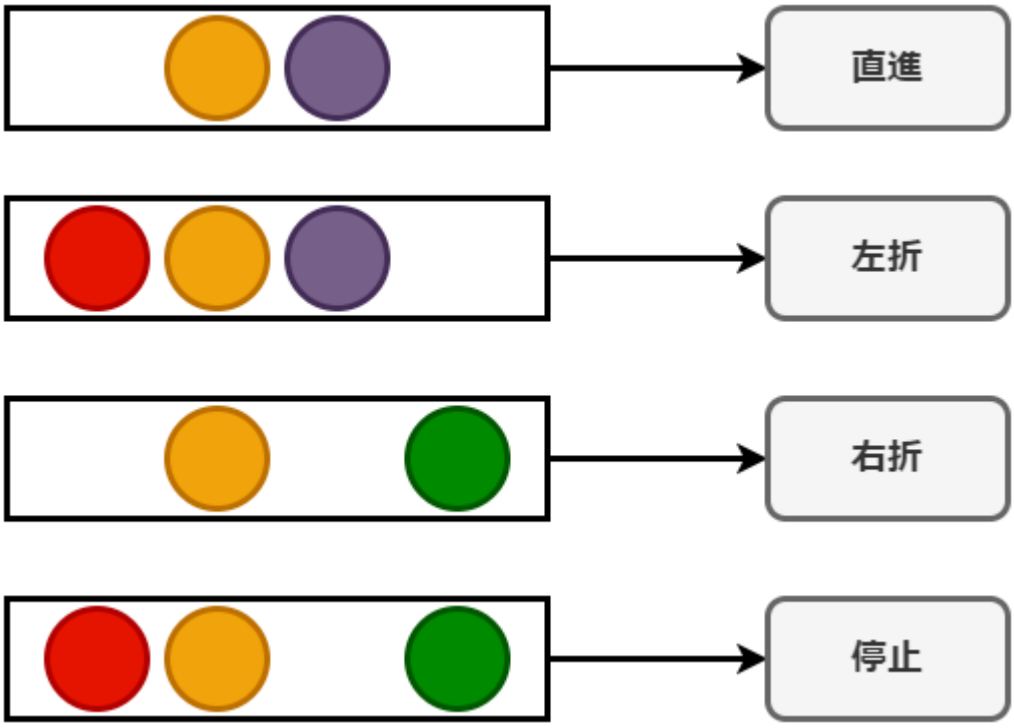
AI アーキテクチャ I
第6章　演習　3

問題　4つのランプがある信号機の意味を以下の図のように定義します。
**　ニューラルネットワークを使用した学習を実行し、下記の実行結果となるように、未完成プログラムの空白部分を埋めてプログラムを完成させてください。**

信号機の意味



実行結果

```
$> python chap6_3_ex.py
信号パターン = 1
出力層 =  [[ 0.15791862 -1.84959933  2.0783497  -0.12866366]]
出力層 =  [[ 0.68357436 -0.69501674  0.78097338 -0.04834744]]
出力層 =  [[ 0.88109797 -0.26116374  0.29346333 -0.01816733]]
出力層 =  [[ 0.95532065 -0.09813648  0.11027357 -0.00682667]]
出力層 =  [[ 0.98321101 -0.03687636  0.04143707 -0.00256523]]
出力層 =  [[ 9.93691268e-01 -1.38568874e-02  1.55706468e-02 -9.63926519e-04]]
出力層 =  [[ 9.97629392e-01 -5.20694866e-03  5.85092133e-03 -3.62210916e-04]]
出力層 =  [[ 9.99109206e-01 -1.95659483e-03  2.19857793e-03 -1.36106586e-04]]
出力層 =  [[ 9.99665270e-01 -7.35222022e-04  8.26151071e-04 -5.11442420e-05]]
出力層 =  [[ 9.99874220e-01 -2.76271517e-04  3.10439572e-04 -1.92182727e-05]]
信号パターン = 2
出力層 =  [[ 0.62450197 -0.56939641 -0.28218798 -0.67654422]]
出力層 =  [[-0.10368942  1.26057533  0.04685319  0.11233027]]
出力層 =  [[ 0.01721611  0.95673528 -0.00777929 -0.0186508 ]]
出力層 =  [[-0.00285848  1.00718347  0.00129164  0.00309669]]
出力層 =  [[ 4.74609414e-04  9.98807289e-01 -2.14457407e-04 -5.14160510e-04]]
出力層 =  [[-7.88019532e-05  1.00019803e+00  3.56075165e-05  8.53688343e-05]]
出力層 =  [[ 1.30839120e-05  9.99967120e-01 -5.91210743e-06 -1.41742466e-05]]
出力層 =  [[-2.17239227e-06  1.00000546e+00  9.81618987e-07  2.35342638e-06]]
出力層 =  [[ 3.60693970e-07  9.99999094e-01 -1.62983479e-07 -3.90752037e-07]]
出力層 =  [[-5.98879592e-08  1.00000015e+00  2.70610235e-08  6.48786619e-08]]
```

```
信号パターン = 4
出力層 =  [[-0.58751388 -0.04373637  0.81733838 -0.45102974]]
出力層 =  [[-0.29332613 -0.02183611  0.40806986  0.27554913]]
出力層 =  [[-0.14644798 -0.01090204  0.2037357   0.63830579]]
出力層 =  [[-0.0731166  -0.00544303  0.10171846  0.81941812]]
出力層 =  [[-0.03650469 -0.00271752  0.05078464  0.90984147]]
出力層 =  [[-0.01822558 -0.00135677  0.02535508  0.95498685]]
出力層 =  [[-9.09942315e-03 -6.77389474e-04  1.26589483e-02  9.77526431e-01]]
出力層 =  [[-4.54303919e-03 -3.38198024e-04  6.32019164e-03  9.88779695e-01]]
出力層 =  [[-2.26818830e-03 -1.68851020e-04  3.15546138e-03  9.94398075e-01]]
出力層 =  [[-1.13243094e-03 -8.43016956e-05  1.57541687e-03  9.97203145e-01]]

$>
```

AI アーキテクチャ I
第6章　演習　3

未完成プログラム

ファイル名：chap6_3_ex.py

```
# numpy のインポート
import numpy as np
# my_deep_learning のインポート
import my_deep_learning as dl

# 乱数シードの設定
np.random.seed(1)

# 入力層の初期化 = 信号パターンの初期化
move_pattern = np.array( [
  [0, 1, 1, 0], # パターン1
  [1, 1, 1, 0], # パターン2
  [0, 1, 0, 1], # パターン3
  [1, 1, 0, 1]  # パターン4
] )
# 入力層のユニット数
input_layer_size = _________(1)__________

# 中間層のユニット数
hidden_layer_size = _________(2)__________

# 目的値リストの初期化 = 出力層の正解(教師信号)
direction = np.array([
  [1, 0, 0, 0], # パターン1(前進)
  [0, 1, 0, 0], # パターン2(左折)
  [0, 0, 1, 0], # パターン3(右折)
  [0, 0, 0, 1]  # パターン4(停止)
])
# 出力層のユニット数
output_layer_size = _________(3)__________

# 学習効率アルファの初期化
alpha = 0.2

# 重みの初期化
# 入力層と中間層間の重み
weight_i_h = _________(4)__________
# print('weight_i_h.shape = ', weight_i_h.shape)
# 中間層と出力層間の重み
weight_h_o = _________(5)__________
```

```
# print('weight_h_o.shape = ', weight_h_o.shape)

# 信号機の全パターンを10回学習する
# 信号機全パターンを学習する

  print('信号パターン = {}'.format(pat_num+1))
  # パターンを取り出し、入力層に設定する
  input_layer = _________(6)___________

  # 10回学習する

    # 入力層のデータを元に予測し、結果を中間層に設定する
    hidden_layer = _________(7)___________
    # 中間層のデータを元に予測し、結果を出力層に設定する
    output_layer = _________(8)___________
    # 出力層の表示
    print('出力層 = ', output_layer)
    # 出力層の学習
    weight_i_h, weight_h_o = dl.back_propagation(input_layer, hidde
n_layer, output_layer, direction, weight_i_h, weight_h_o, alpha, pa
t_num)
```

ファイル名:my_deep_leraning.py

```
# numpy インポート 別名 np
import numpy as np

# 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
'''
def neural_network(input, weight):
  # numpy の dot メソッドを使用して、input と weight の加重和を計算する
  _________(1)_________
  # 加重和を返す
  return pred

# 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
'''
def create_weight(layer_1_num, layer_2_num):
  # 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 = 2 * np.random.random((layer_1_num, layer_2_num)) - 1
  # 重み行列を返す
  return weight_1_2

# 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
'''
def back_propagation(input, hidden, output, goal, weight_i_h, weight_h_o, alpha, input_num):
  # 出力層のデルタを計算する
  output_delta = _________(2)_________
  # 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 = _________(3)_________
  # 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 = _________(4)_________
  # 入力層と中間層間の重みの微調整量を計算
  weight_delta_i_h = _________(5)_________
  # 中間層と出力層間の重みを更新する
  weight_h_o = _________(6)_________
  # 入力層と中間層間の重みを更新する
  weight_i_h = _________(7)_________
  # 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
```