第５章　実習１

問題１

▶ テキスト P84 ５． ３ 学習ステップを確認するのリストにおいて、勾配降下法による学習処理、学習結果を表示する処理を関数にしてください。また、関数化に伴ってスカラー積を求める関数を改良してください。下記の実行結果になるように、コメントに従ってプログラムを実装してください。

In [1]:

```
# 予測する関数 neural_network の定義
def neural_network(input, weights):
    # 予測値を初期化
    out = 0
    # 入力層ニューロンの数分繰り返す
    for i in range(len(input)):
        # 予測値（＝加重和＝内積値）を求める
        out += (input[i] * weights[i])
    # 予測値を返す
    return out

# スカラー積を求める関数 ele_mul の定義
def ele_mul(scalar, vector):
    # スカラー積の結果を返すリストの初期化

    # ベクトルの長さ分繰り返す

        # スカラー積を求めて、リストに追加する

    # スカラー積を返す


# 学習関数 grad_descent_learn(input, truth, pred, weights, alpha) の定義
'''
関数名：grad_descent_learn
引数：
    input：入力値リスト
    truth：目的値
    pred：予測値
    weights：重みリスト
    alpha：重み再微調整値
処理：勾配降下法に基づき重みを修正する
戻り値：修正された重みリスト
'''
def grad_descent_learn(input, truth, pred, weights, alpha):
    # 誤差を求める

    # デルタを求める

    # 重みの微調整量を求める

    # 重みを修正する


    # 更新した重みリストを返す


# 学習結果を表示する関数 disp_learn の定義
'''
```

```
関数名：disp_learn
引数：iter = 学習回数、pred = 予測値、weights = 重みリスト
処理：学習回数、予測値（小数点以下４桁）、重みを表示する（小数点以下６桁）
戻り値：なし
'''
def disp_learn(iter, pred, weights):
    # 学習回数を表示

    # 予測値を表示

    # 重みリストを表示




# 入力データセット
# シーズンの各選手の足指の平均数を初期化
toes =  [8.5, 9.5, 9.9, 9.0]
# シーズンの勝率を初期化
wlrec = [0.65, 0.8, 0.8, 0.9]
# シーズンのファンの数（百万単位）を初期化
nfans = [1.2, 1.3, 0.5, 1.0]
# シーズン４試合の勝ち負けを初期化
win_or_lose_binary = [1, 1, 0, 1]
# シーズン第１試合の結果初期化
truth = win_or_lose_binary[0]

# アルファを初期化

# 入力層から出力層への重みの初期化

# 入力はシーズン最初の試合の３つのデータポイントを設定


# ３回学習する

    # 予測する

    # 学習＝重みを修正

    # 学習結果を表示
```

```
1 回目学習
予測値 = 0.8600
重みリスト = [0.111900, 0.200910, -0.098320]
2 回目学習
予測値 = 0.9638
重みリスト = [0.114981, 0.201146, -0.097885]
3 回目学習
予測値 = 0.9906
重みリスト = [0.115778, 0.201207, -0.097773]
```

問題２

▶ 問題１のプログラムにおいて、シーズン全試合の学習結果を表示するようにコメントに従ってプログラムを実装してください。実行結果は下記のようになります。

In [2]:

```
# 予測する関数 neural_network の定義
def neural_network(input, weights):
    # 予測値を初期化
    out = 0
```

```
    # 入力層ニューロンの数分繰り返す
    for i in range(len(input)):
        # 予測値（＝加重和＝内積値）を求める
        out += (input[i] * weights[i])
    # 予測値を返す
    return out

# スカラー積を求める関数 ele_mul の定義
def ele_mul(scalar, vector):
    # スカラー積の結果を返すリストの初期化

    # ベクトルの長さ分繰り返す

        # スカラー積を求めて、リストに追加する

    # スカラー積を返す


# 学習関数 grad_descent_learn(input, truth, pred, weights, alpha) の定義
'''
関数名：grad_descent_learn
引数：
    input：入力値リスト
    truth：目的値
    pred：予測値
    weights：重みリスト
    alpha：重み再微調整値
処理：勾配降下法に基づき重みを修正する
戻り値：修正された重みリスト
'''
def grad_descent_learn(input, truth, pred, weights, alpha):
    # 誤差を求める

    # デルタを求める

    # 重みの微調整量を求める

    # 重みを修正する


    # 更新した重みリストを返す


# 学習結果を表示する関数 disp_learn の定義
'''
関数名：disp_learn
引数：iter = 学習回数、pred = 予測値、weights = 重みリスト
処理：学習回数、予測値（小数点以下４桁）、重みを表示する（小数点以下６桁）
戻り値：なし
'''
def disp_learn(iter, pred, weights):
    # 学習回数を表示

    # 予測値を表示

    # 重みリストを表示









# 入力データセット
```

```
# シーズンの各選手の足指の平均数を初期化
toes =  [8.5, 9.5, 9.9, 9.0]
# シーズンの勝率を初期化
wlrec = [0.65, 0.8, 0.8, 0.9]
# シーズンのファンの数（百万単位）を初期化
nfans = [1.2, 1.3, 0.5, 1.0]
# シーズン４試合の勝ち負けを初期化
win_or_lose_binary = [1, 1, 0, 1]
# シーズン第１試合の結果初期化
truth = win_or_lose_binary[0]

# アルファを初期化

# 入力層から出力層への重みの初期化


# シーズン全試合（４試合）を学習する

    # 第何回目の試合なのか表示

    # 入力データをセット

    # ３回学習する

        # 予測する

        # 学習＝重みを修正

        # 学習結果を表示
```

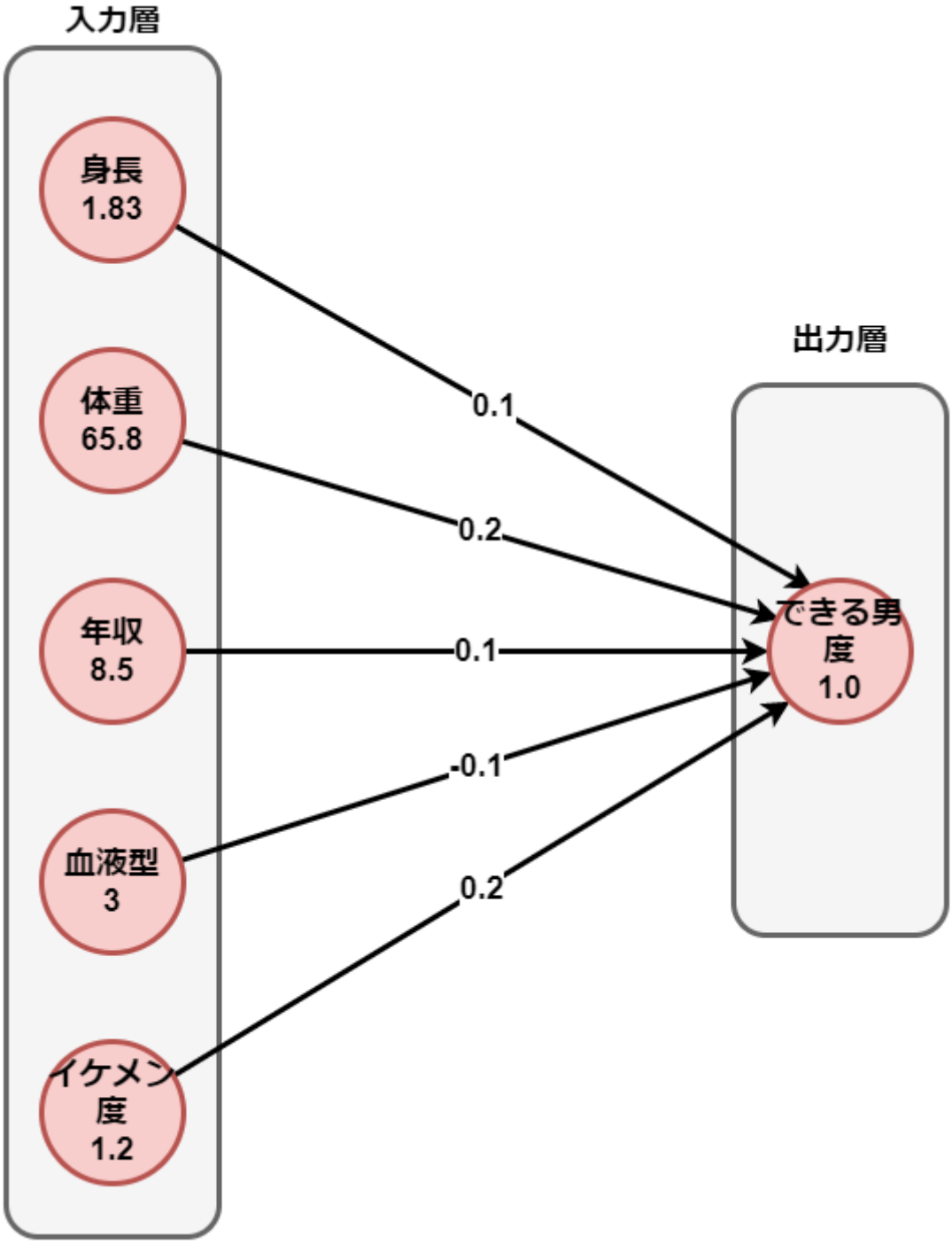
```
シーズン　第1試合
1 回目学習
予測値 = 0.8600
重みリスト = [0.111900, 0.200910, -0.098320]
2 回目学習
予測値 = 0.9638
重みリスト = [0.114981, 0.201146, -0.097885]
3 回目学習
予測値 = 0.9906
重みリスト = [0.115778, 0.201207, -0.097773]

シーズン　第2試合
1 回目学習
予測値 = 1.1338
重みリスト = [0.103072, 0.200137, -0.099511]
2 回目学習
予測値 = 1.0099
重みリスト = [0.102129, 0.200057, -0.099640]
3 回目学習
予測値 = 1.0007
重みリスト = [0.102059, 0.200051, -0.099650]

シーズン　第3試合
1 回目学習
予測値 = 1.1206
重みリスト = [0.090120, 0.199086, -0.100253]
2 回目学習
予測値 = 1.0013
重みリスト = [0.089988, 0.199076, -0.100260]
3 回目学習
予測値 = 1.0000
重みリスト = [0.089987, 0.199076, -0.100260]

シーズン　第4試合
1 回目学習
予測値 = 0.8888
重みリスト = [0.099996, 0.200077, -0.099147]
2 回目学習
予測値 = 0.9809
重みリスト = [0.101716, 0.200249, -0.098956]
```

```
3 回目学習
予測値 = 0.9967
重みリスト = [0.102012, 0.200278, -0.098923]
```

問題３

▶ 下図のニューラルネットワークの実装を、未完成プログラムのコメントと実行結果をもとに行ってください。

注意）アルファの値を自分で考えて設定してください。



In [3]:

```
# 予測する関数 neural_network の定義

    # 予測値を初期化

    # 入力層ニューロンの数分繰り返す
```

```
            # 予測値（＝加重和＝内積値）を求める

        # 予測値を返す


# スカラー積を求める関数 ele_mul の定義

        # スカラー積の結果を返すリストの初期化

        # ベクトルの長さ分繰り返す

            # スカラー積を求めて、リストに追加する

        # スカラー積を返す


# 学習関数 grad_descent_learn(input, truth, pred, weights, alpha) の定義
'''
関数名：grad_descent_learn
引数：
    input：入力値リスト
    truth：目的値
    pred：予測値
    weights：重みリスト
    alpha：重み再微調整値
処理：勾配降下法に基づき重みを修正する
戻り値：修正された重みリスト
'''

        # 誤差を求める

        # デルタを求める

        # 重みの微調整量を求める

        # 重みを修正する


        # 更新した重みリストを返す


# 学習結果を表示する関数 disp_learn の定義
'''
関数名：disp_learn
引数：iter = 学習回数、pred = 予測値、weights = 重みリスト
処理：学習回数、予測値（小数点以下４桁）、重みを表示する（小数点以下６桁）
戻り値：なし
'''

        # 学習回数を表示

        # 予測値を表示

        # 重みリストを表示









# アルファを初期化
alpha =
```

```
# 入力層の初期化
#       [身長，体重，年収，血液型，イケメン度]


# 入力層から出力層への重みの初期化


# 出力層の初期化


# ２０回学習する

    # 予測する

    # 学習＝重みを修正

    # 学習結果を表示
```

```
1 回目学習
予測値 = -12.1870
重みリスト = [0.102413, -0.113230, 0.111209, -0.096044, 0.201582]
2 回目学習
予測値 = -6.3640
重みリスト = [0.103761, -0.064774, 0.117468, -0.093835, 0.202466]
3 回目学習
予測値 = -3.1123
重みリスト = [0.104513, -0.037715, 0.120964, -0.092601, 0.202960]
4 回目学習
予測値 = -1.2965
重みリスト = [0.104934, -0.022604, 0.122916, -0.091912, 0.203235]
5 回目学習
予測値 = -0.2824
重みリスト = [0.105168, -0.014166, 0.124006, -0.091527, 0.203389]
6 回目学習
予測値 = 0.2839
重みリスト = [0.105299, -0.009454, 0.124615, -0.091312, 0.203475]
7 回目学習
予測値 = 0.6001
重みリスト = [0.105373, -0.006823, 0.124955, -0.091193, 0.203523]
8 回目学習
予測値 = 0.7767
重みリスト = [0.105413, -0.005353, 0.125144, -0.091126, 0.203550]
9 回目学習
予測値 = 0.8753
重みリスト = [0.105436, -0.004532, 0.125250, -0.091088, 0.203565]
10 回目学習
予測値 = 0.9304
重みリスト = [0.105449, -0.004074, 0.125310, -0.091067, 0.203573]
11 回目学習
予測値 = 0.9611
重みリスト = [0.105456, -0.003818, 0.125343, -0.091056, 0.203578]
12 回目学習
予測値 = 0.9783
重みリスト = [0.105460, -0.003675, 0.125361, -0.091049, 0.203580]
13 回目学習
予測値 = 0.9879
重みリスト = [0.105462, -0.003596, 0.125371, -0.091045, 0.203582]
14 回目学習
予測値 = 0.9932
重みリスト = [0.105464, -0.003551, 0.125377, -0.091043, 0.203583]
15 回目学習
予測値 = 0.9962
重みリスト = [0.105464, -0.003526, 0.125380, -0.091042, 0.203583]
16 回目学習
予測値 = 0.9979
重みリスト = [0.105465, -0.003512, 0.125382, -0.091042, 0.203583]
17 回目学習
予測値 = 0.9988
重みリスト = [0.105465, -0.003504, 0.125383, -0.091041, 0.203584]
18 回目学習
予測値 = 0.9993
重みリスト = [0.105465, -0.003500, 0.125384, -0.091041, 0.203584]
19 回目学習
```

```
予測値 = 0.9996
重みリスト = [0.105465, -0.003498, 0.125384, -0.091041, 0.203584]
20 回目学習
予測値 = 0.9998
重みリスト = [0.105465, -0.003496, 0.125384, -0.091041, 0.203584]
```

問題４

▶ 下記のプログラムにおいて学習するタスクを関数化します。未完成リストのコメントに従ってプログラムを完成させてください。実行結果通りになるようにしてください。

👇リスト（参照）

In [4]:

```
# チームの勝敗を予測するだけでなく、選手が喜んでいるのかどうかも予測する。
# また、けがをしたチームメンバーの割合も予測する。この予測では、現在の勝敗記録のみを使

# 予測をする関数 neural_network の定義
def neural_network(input, weights):
    pred = ele_mul(input, weights)
    return pred

# 加重和（内積）を計算する関数 ele_mul の定義
def ele_mul(input, weights):
    # 加重和リストの初期化
    out = []
    # 重みリストの長さ分繰り返す
    for i in range(len(weights)):
        # 加重和を求めて、加重和リストに追加する
        out.append(input * weights[i])
    # 加重和を返す
    return out

# スカラー積を求める関数 scalar_ele_mul の定義
def scalar_ele_mul(number, vector):
    # スカラー積を要素とするリストの初期化
    output = []
    # 引数 vector の長さ分繰り返す
    for i in range(len(vector)):
        # スカラー積を求めてリストに追加する
        output.append(number * vector[i])
    # スカラー積を要素とするリストを返す
    return output

# 重みの初期化：[勝敗からけが？への重み, 勝敗から勝った？への重み, 勝敗から悲しい？へ
weights = [0.3, 0.2, 0.9]

# １シーズン４試合の勝率の初期化
wlrec = [0.65, 1.0, 1.0, 0.9]
# １シーズン４試合のけがの初期化
hurt  = [0.1, 0.0, 0.0, 0.1]
# １シーズン４試合の勝ち負けの初期化
win   = [  1,   1,   0,   1]
# １シーズン４試合の悲しみ度の初期化
sad   = [0.1, 0.0, 0.1, 0.2]

# 入力値の初期化（シーズン第１試合の勝率）
input = wlrec[0]
# シーズン第１試合の事実
# [第１試合のけが, 第１試合の勝ち負け, 第１試合の悲しみ度]
truth = [hurt[0], win[0], sad[0]]

# 予測する
```

```
pred = neural_network(input, weights)

# デルタ（純誤差）リストの初期化
# [けが, 勝ち負け, 悲しみ度]
delta = []
# 出力層のユニット数分繰り返す
for i in range(len(truth)):
    # 各ユニットごとの純誤差を求めて、純誤差リストに追加する
    delta.append(pred[i] - truth[i])

# 重み微調整量を求める
weight_deltas = scalar_ele_mul(input, delta)
# 重み微調整用アルファの初期化
alpha = 0.1
# 学習する
for i in range(len(weights)):
    # 重みを更新する
    weights[i] -= (weight_deltas[i] * alpha)

# 重みを表示
print("Weights:" + str(weights))

# 予測値の表示
print('[けが度 = {}, 勝ち負け = {}, 悲しみ度 = {}]'.format(pred[0], pred[1], pred[2]))
```

```
Weights:[0.293825, 0.25655, 0.868475]
[けが度 = 0.195, 勝ち負け = 0.13, 悲しみ度 = 0.5850000000000001]
```

👇未完成リスト

In [5]:

```
# チームの勝敗を予測するだけでなく、選手が喜んでいるのかどうかも予測する。
# また、けがをしたチームメンバーの割合も予測する。この予測では、現在の勝敗記録のみを使

# 予測をする関数 neural_network の定義
def neural_network(input, weights):
    pred = ele_mul(input, weights)
    return pred

# 加重和（内積）を計算する関数 ele_mul の定義
def ele_mul(input, weights):
    # 加重和リストの初期化
    out = []
    # 重みリストの長さ分繰り返す
    for i in range(len(weights)):
        # 加重和を求めて、加重和リストに追加する
        out.append(input * weights[i])
    # 加重和を返す
    return out

# スカラー積を求める関数 scalar_ele_mul の定義
def scalar_ele_mul(number, vector):
    # スカラー積を要素とするリストの初期化
    output = []
    # 引数 vector の長さ分繰り返す
    for i in range(len(vector)):
        # スカラー積を求める
        output.append(number * vector[i])
    # スカラー積を要素とするリストを返す
    return output

# 学習関数 grad_descent_learn(input, truth, pred, weights, alpha) の定義
'''
関数名：grad_descent_learn
引数：
```

```
    input：入力値
    truth：目的値リスト
    pred：予測値リスト
    weights：重みリスト
    alpha：重み再微調整値
処理：勾配降下法に基づき重みを修正する
戻り値：修正された重みリスト
'''
```

```
# １シーズン４試合の勝率の初期化
wlrec = [0.65, 1.0, 1.0, 0.9]
# １シーズン４試合のけがの初期化
hurt  = [0.1, 0.0, 0.0, 0.1]
# １シーズン４試合の勝ち負けの初期化
win   = [  1,   1,   0,   1]
# １シーズン４試合の悲しみ度の初期化
sad   = [0.1, 0.0, 0.1, 0.2]
# 重みの初期化：[勝敗からけが？への重み，勝敗から勝った？への重み，勝敗から悲しい？へ
weights = [0.3, 0.2, 0.9]
# 入力値の初期化（シーズン第１試合の勝率）
input = wlrec[0]
# シーズン第１試合の事実
# [第１試合のけが，第１試合の勝ち負け，第１試合の悲しみ度]
truth = [hurt[0], win[0], sad[0]]

# 予測する
pred = neural_network(input, weights)
# 重み微調整用アルファの初期化
alpha = 0.1
# 学習する


# 重みを表示
print("Weights:" + str(weights))

# 予測値の表示
print('[けが度 = {}，勝ち負け = {}，悲しみ度 = {}]'.format(pred[0], pred[1], pred[2]))
```

```
Weights:[0.293825, 0.25655, 0.868475]
[けが度 = 0.195，勝ち負け = 0.13，悲しみ度 = 0.5850000000000001]
```

In [ ]: