第５章　演習２

問題１

（１）下記のプログラムを実行してください。

（２）予測・比較・学習部分の処理、それに伴う関数 **vect_mat_mul** の修正をそれぞれコメントに従って実装してください。

👇リスト

In [1]:

```
# NumPyのインポート、別名 np
import numpy as np

# 加重和（内積）を求める関数 w_sum の定義
def w_sum(a, b):
    # ２つの引数のリストの長さが等しいとき以下を実行する
    assert(len(a) == len(b))
    # 加重和の初期化
    output = 0
    # 加重和を求める
    # 引数のリストの長さ分繰り返す
    for i in range(len(a)):
        # リストの要素同士を掛けて合計を求める
        output += (a[i] * b[i])
    # 加重和を返す
    return output

# ベクトルと行列のかけ算を求める関数 vect_mat_mul の定義
def vect_mat_mul(vect, matrix):
    # ベクトルと行列の行リストの長さが等しいとき以下を実行する
    assert(len(vect) == len(matrix))
    # 加重和リストの初期化
    output = [0, 0, 0]
    # ベクトルの長さ（＝行列の行リストの長さ）分繰り返す
    for i in range(len(vect)):
        # ベクトルの行列の各行との加重和を求める
        output[i] = w_sum(vect, matrix[i])
    # 加重和のリストを返す
    return output



# ２つのリストの要素間を総なめして要素ごとのかけ算をして
# その結果を行列に格納する関数 outer_prod の定義
def outer_prod(a, b):
    # a 行 b 列のゼロ行列を生成する
    out = np.zeros((len(a), len(b)))
    # a 行 b 列の行列にリスト a とリスト b の
    # 要素を総なめしたかけ算を行い、行列に格納する
    for i in range(len(a)):
        for j in range(len(b)):
            out[i][j] = a[i] * b[j]
    # 要素間のかけ算の結果を保存した行列を返す
    return out

# 予測値を求める関数 nuural_network の定義
```

```
def neural_network(input, weights):
    # ベクトルと行列の加重和を求める
    pred = vect_mat_mul(input, weights)
    # 予測値のリストを返す
    return pred

# 重みの初期化
            #toes %win #fans
weights = [ [0.1, 0.1, -0.3], # けが？
            [0.1, 0.2, 0.0],  # 勝った？
            [0.0, 1.3, 0.1] ] # 悲しい？

# シーズン４試合の足指の数の平均
toes  = [8.5, 9.5, 9.9, 9.0]
# シーズン４試合の勝率
wlrec = [0.65,0.8, 0.8, 0.9]
# シーズン４試合のファンの数
nfans = [1.2, 1.3, 0.5, 1.0]

# シーズン４試合のけが度
hurt  = [0.1, 0.0, 0.0, 0.1]
# シーズン４試合の勝ち負け
win   = [  1,   1,   0,   1]
# シーズン４試合の悲しみ度
sad   = [0.1, 0.0, 0.1, 0.2]
# アルファの初期化
alpha = 0.01
# 入力データの設定（シーズン第１試合の［足指の数の平均値，勝率，ファンの数］
input = [toes[0], wlrec[0], nfans[0]]
# シーズン第１試合の結果の設定（[けが度，勝ち負け，悲しみ度]
true  = [hurt[0], win[0], sad[0]]

# 勾配降下法による学習

# 予測値を求める
pred = neural_network(input,weights)

# 誤差の初期化
error = [0, 0, 0]
# デルタの初期化
delta = [0, 0, 0]

# 予測値の数分繰り返す
for i in range(len(true)):
    # 誤差を求める
    error[i] = (pred[i] - true[i]) ** 2
    # デルタを求める
    delta[i] = pred[i] - true[i]

# 重みの微調整量を求め行列に格納する
weight_deltas = outer_prod(delta, input)

# 重み行列を更新する
for i in range(len(weights)):
    for j in range(len(weights[0])):
        weights[i][j] -= alpha * weight_deltas[i][j]

# 更新された重み行列を表示する
for i in range(len(weights)):
    print('{} {} {}'.format(weights[i][0], weights[i][1], weights[i][2]))
```

```
0.061325 0.0970425 -0.30546
0.1017 0.20013 0.00023999999999999887
-0.07352500000000001 1.2943775 0.08962
```

👇未完成リスト

In [2]:

```
# NumPyのインポート、別名 np
import numpy as np

# 加重和（内積）を求める関数 w_sum の定義
def w_sum(a, b):
    # ２つの引数のリストの長さが等しいとき以下を実行する
    assert(len(a) == len(b))
    # 加重和の初期化
    output = 0
    # 加重和を求める
    # 引数のリストの長さ分繰り返す
    for i in range(len(a)):
        # リストの要素同士を掛けて合計を求める
        output += (a[i] * b[i])
    # 加重和を返す
    return output

# ベクトルと行列のかけ算を求める関数 vect_mat_mul の定義

    # ベクトルと行列の行リストの長さが等しいとき以下を実行する

    # 加重和リストの初期化

    # ベクトルの長さ（＝行列の行リストの長さ）分繰り返す

        # ベクトルの行列の各行との加重和を求めて out に追加する

    # 加重和のリストを返す


# ２つのリストの要素間を総なめして要素ごとのかけ算をして
# その結果を行列に格納する関数 outer_prod の定義
def outer_prod(a, b):
    # a 行 b 列のゼロ行列を生成する
    out = np.zeros((len(a), len(b)))
    # a 行 b 列の行列にリスト a とリスト b の
    # 要素を総なめしたかけ算を行い、行列に格納する
    for i in range(len(a)):
        for j in range(len(b)):
            out[i][j] = a[i] * b[j]
    # 要素間のかけ算の結果を保存した行列を返す
    return out

# 予測値を求める関数 nuural_network の定義
def neural_network(input, weights):
    # ベクトルと行列の加重和を求める
    pred = vect_mat_mul(input, weights)
    # 予測値のリストを返す
    return pred

# 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
'''
```

```
    # デルタの初期化

    # 予測値の数分繰り返す

        # デルタを求めてデルタのリストに追加する

    # 重みの微調整量を求め行列に格納する

    # 重み行列を更新する（２重ループ）



    # 重み行列を返す



# 重みの初期化
            #toes %win #fans
weights = [ [0.1, 0.1, -0.3], # けが？
            [0.1, 0.2, 0.0],  # 勝った？
            [0.0, 1.3, 0.1] ] # 悲しい？

# シーズン４試合の足指の数の平均
toes  = [8.5, 9.5, 9.9, 9.0]
# シーズン４試合の勝率
wlrec = [0.65,0.8, 0.8, 0.9]
# シーズン４試合のファンの数
nfans = [1.2, 1.3, 0.5, 1.0]

# シーズン４試合のけが度
hurt  = [0.1, 0.0, 0.0, 0.1]
# シーズン４試合の勝ち負け
win   = [  1,   1,   0,   1]
# シーズン４試合の悲しみ度
sad   = [0.1, 0.0, 0.1, 0.2]
# アルファの初期化
alpha = 0.01
# 入力データの設定（シーズン第１試合の［足指の数の平均値，勝率，ファンの数］
input = [toes[0], wlrec[0], nfans[0]]
# シーズン第１試合の結果の設定（[けが度，勝ち負け，悲しみ度］
truth  = [hurt[0], win[0], sad[0]]

# 勾配降下法による学習
# 予測値を求める

# 学習する


# 予測値を表示

print('更新された重み')
# 更新された重み行列を表示する（ループ）
```

```
予測値 = [0.555, 0.9800000000000001, 0.9650000000000001]
更新された重み
0.06132500 : 0.09704250 : -0.30546000
0.10170000 : 0.20013000 : 0.00024000
-0.07352500 : 1.29437750 : 0.08962000
```

問題２

▶ 問題１のプログラムにおいて、学習を１０回実施するプログラムを下記の未完成リストのコメントに従って実装してください。

未完成リスト

In [3]:

```
# NumPyのインポート、別名 np
import numpy as np

# 加重和（内積）を求める関数 w_sum の定義
def w_sum(a, b):
    # ２つの引数のリストの長さが等しいとき以下を実行する
    assert(len(a) == len(b))
    # 加重和の初期化
    output = 0
    # 加重和を求める
    # 引数のリストの長さ分繰り返す
    for i in range(len(a)):
        # リストの要素同士を掛けて合計を求める
        output += (a[i] * b[i])
    # 加重和を返す
    return output

# ベクトルと行列のかけ算を求める関数 vect_mat_mul の定義

    # ベクトルと行列の行リストの長さが等しいとき以下を実行する

    # 加重和リストの初期化

    # ベクトルの長さ（＝行列の行リストの長さ）分繰り返す

        # ベクトルの行列の各行との加重和を求める

    # 加重和のリストを返す


# ２つのリストの要素間を総なめして要素ごとのかけ算をして
# その結果を行列に格納する関数 outer_prod の定義
def outer_prod(a, b):
    # a 行 b 列のゼロ行列を生成する
    out = np.zeros((len(a), len(b)))
    # a 行 b 列の行列にリスト a とリスト b の
    # 要素を総なめしたかけ算を行い、行列に格納する
    for i in range(len(a)):
        for j in range(len(b)):
            out[i][j] = a[i] * b[j]
    # 要素間のかけ算の結果を保存した行列を返す
    return out

# 予測値を求める関数 nuural_network の定義
def neural_network(input, weights):
    # ベクトルと行列の加重和を求める
    pred = vect_mat_mul(input, weights)
    # 予測値のリストを返す
    return pred

# 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
```

```
'''

    # デルタの初期化

    # 予測値の数分繰り返す

        # デルタを求める

    # 重みの微調整量を求め行列に格納する

    # 重み行列を更新する（２重ループ）



    # 重み行列を返す


# 予測値と更新された重みを表示する関数 disp_lear の定義
'''
関数名：disp_learn
引数：
    pred = 予測値リスト
    weights = 重み行列
処理：予測値と重みを表示する。ただし、重みは小数点８桁
戻り値：なし
'''

    # 予測値を表示


    # 更新された重み行列を表示する（ループ）




# 重みの初期化
            #toes %win #fans
weights = [ [0.1, 0.1, -0.3], # けが？
            [0.1, 0.2, 0.0],  # 勝った？
            [0.0, 1.3, 0.1] ] # 悲しい？

# シーズン４試合の足指の数の平均
toes  = [8.5, 9.5, 9.9, 9.0]
# シーズン４試合の勝率
wlrec = [0.65,0.8, 0.8, 0.9]
# シーズン４試合のファンの数
nfans = [1.2, 1.3, 0.5, 1.0]

# シーズン４試合のけが度
hurt  = [0.1, 0.0, 0.0, 0.1]
# シーズン４試合の勝ち負け
win   = [  1,   1,   0,   1]
# シーズン４試合の悲しみ度
sad   = [0.1, 0.0, 0.1, 0.2]
# アルファの初期化
alpha = 0.01
# 入力データの設定（シーズン第１試合の［足指の数の平均値，勝率，ファンの数］
input = [toes[0], wlrec[0], nfans[0]]
# シーズン第１試合の結果の設定（[けが度，勝ち負け，悲しみ度]
truth  = [hurt[0], win[0], sad[0]]

# 勾配降下法による学習（１０回）

    # 学習回数を表示
```

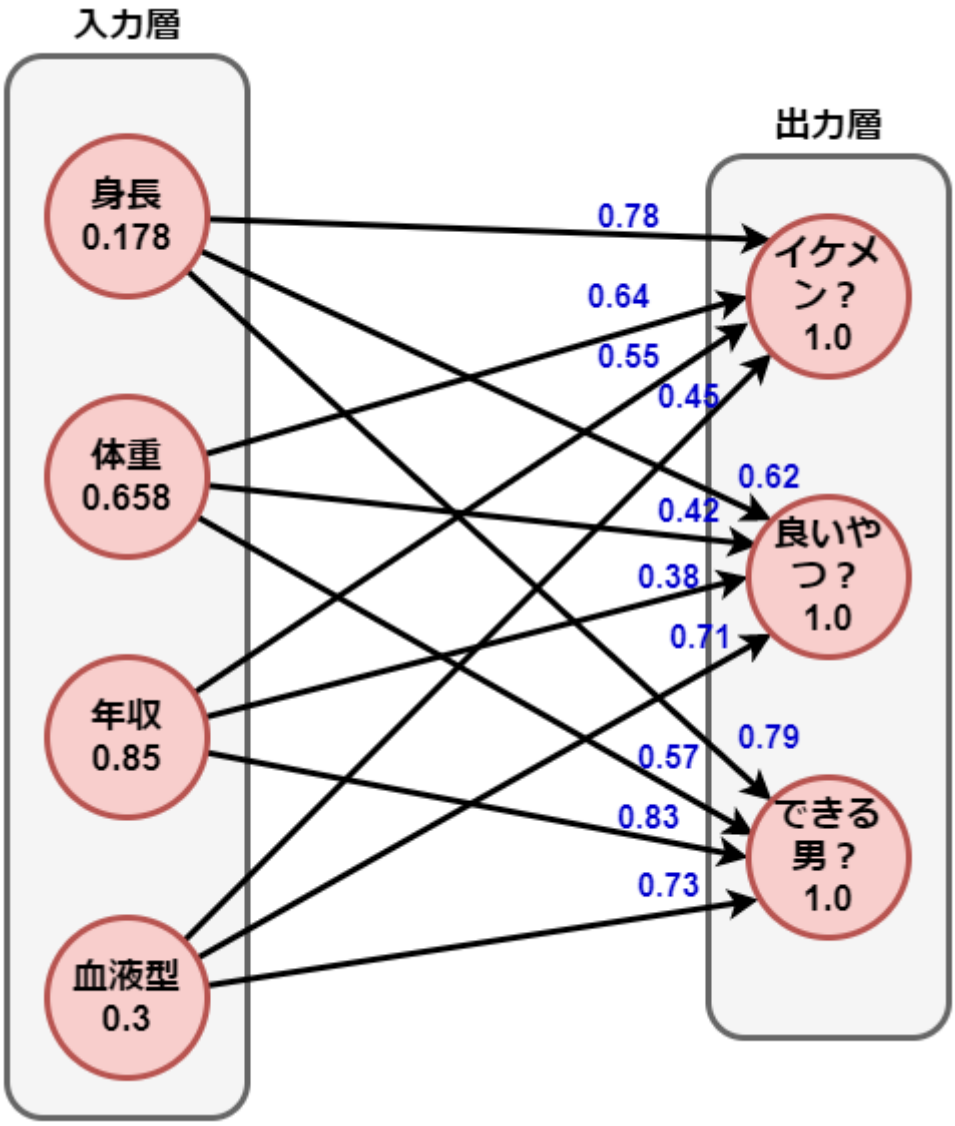
```
    # 予測値を求める

    # 学習する

    # 予測値と更新された重みを表示

    print()
```

```
学習 1 回目
予測値 = [0.555, 0.9800000000000001, 0.9650000000000001]
重み行列 =
0.061325 : 0.0970425 : -0.30546
0.1017 : 0.20013 : 0.00023999999999999887
-0.07352500000000001 : 1.2943775 : 0.08962

学習 2 回目
予測値 = [0.217788125, 0.9948224999999999, 0.32392687499999984]
重み行列 =
0.051313009374999996 : 0.0962768771875 : -0.3068734575
0.1021400875 : 0.20016365375 : 0.0003021299999999995
-0.092558784375 : 1.2929219753124999 : 0.0869328775

学習 3 回目
予測値 = [0.13049240085937502, 0.9986596746875, 0.15796906976562497]
重み行列 =
0.04872115530195312 : 0.09607867658191406 : -0.3072393663103125
0.1022540151515625 : 0.20017236586453124 : 0.0003182139037499988 5
-0.09748615530507812 : 1.2925451763590232 : 0.0862372486628125

学習 4 回目
予測値 = [0.10789372027247063, 0.9996530232847265, 0.11500674293557611]
重み行列 =
0.04805018907879312 : 0.096027367400143 : -0.30733409095358216
0.10228350817236075 : 0.20017462121318053 : 0.0003223776243332809
-0.09876172845460209 : 1.2924476325299419 : 0.0860571677475856

学習 5 回目
予測値 = [0.10204348683553588, 0.9999101764028336, 0.10388487057744723]
重み行列 =
0.04787649269777257 : 0.09601408473571202 : -0.3073586127956086
0.10229114317811988 : 0.2001752050665621 : 0.00032345550749927725
-0.0990919424536851 : 1.2924223808711885 : 0.08601054930065623

学習 6 回目
予測値 = [0.10052900765454936, 0.9999767469162835, 0.10100569587073659]
重み行列 =
0.04783152704713587 : 0.09601064618595745 : -0.3073649608874632
0.10229311969023579 : 0.20017535621160626 : 0.00032373454450387564
-0.09917742660269771 : 1.2924158438480287 : 0.08599848095020739

学習 7 回目
予測値 = [0.10013694685657143, 0.9999939803579528, 0.10026034951853695]
重み行列 =
0.0478198865643273 : 0.09600975603138974 : -0.30736660424974205
0.10229363135980979 : 0.20017539533927955 : 0.00032380678020844 16
-0.09919955631177335 : 1.2924141515761582 : 0.08599535675598495

学習 8 回目
予測値 = [0.10003545211749498, 0.999998441665165, 0.10006739798161135]
重み行列 =
0.04781687313434023 : 0.09600952559262602 : -0.307367029675152
0.10229376381827077 : 0.20017540546845597 : 0.000323825480226461
-0.09920528514021032 : 1.2924137134892777 : 0.08599454798020562

学習 9 回目
予測値 = [0.10000917766691647, 0.9999995965860696, 0.10001744765248952]
重み行列 =
0.04781609303265233 : 0.09600946593779106 : -0.307367139807155
0.10229379810845485 : 0.20017540809064652 : 0.00032383032119362546
-0.09920676819067192 : 1.2924136000795365 : 0.08599433860837574

学習 10 回目
予測値 = [0.10000237586852295, 0.9999998955662187, 0.10000451676103836]
重み行列 =
0.04781589108382788 : 0.09600945049464565 : -0.3073671683175773
0.10229380698532625 : 0.2001754087694661 : 0.0003238315743990009
-0.09920715211536019 : 1.2924135707205897 : 0.08599428440724328
```

問題３

▶ 下図のネットワークにおいて、下記の実行結果のように１０回学習させるようにプログラムを実装してください

注意）実行結果と全く同じでなくても構いません。学習が目的値に向かって進むのであれば**OK**です。



In [1]:

```
# NumPyのインポート、別名 np
import numpy as np

# 加重和（内積）を求める関数 w_sum の定義



# ベクトルと行列のかけ算を求める関数 vect_mat_mul の定義



# ２つのリストの要素間を総なめして要素ごとのかけ算をして
# その結果を行列に格納する関数 outer_prod の定義
```

```
# 予測値を求める関数 nuural_network の定義



# 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
'''




# 予測値と更新された重みを表示する関数 disp_lear の定義
'''
関数名：disp_learn
引数：
    pred = 予測値リスト
    weights = 重み行列
処理：予測値と重みを表示する。ただし、重みは小数点８桁
戻り値：なし
'''



# 入力値［身長、体重、年収、血液型］の初期化


# 重み行列の初期化



# 目的値の初期化


# alpha の初期化


# １０回学習する
```

```
学習 1 回目
予測値 = [1.16246, 0.92272, 1.44018]
重み行列 =
0.777108212 : 0.629310132 : 0.5361909
0.621375584 : 0.425085024 : 0.3865688
0.782164796 : 0.541036156 : 0.7925846999999999

学習 2 回目
予測値 = [1.1417114535919999, 0.9325897997439999, 1.3839624993359998]
重み行列 =
0.7745857481260624 : 0.6199855183536463 : 0.52414542644468
0.6225754855645569 : 0.4295206151768448 : 0.39229866702176
0.7753302635118192 : 0.5157714235436912 : 0.7599478875564399
```

```
学習 3 回目
予測値 = [1.1236128036387885, 0.941199079987655, 1.3349248055258025]
重み行列 =
0.772385440221292 : 0.611851795874214 : 0.513638338135383
0.6236221419407766 : 0.43338971571365714 : 0.3972967452228093
0.7693686019734599 : 0.49373337134009343 : 0.7314792790867467

学習 4 回目
予測値 = [1.1078256191446212, 0.9487088277268476, 1.2921499509730356]
重み行列 =
0.7704661442005177 : 0.604756870134498 : 0.5044731605080901
0.6245351248072387 : 0.4367646748492306 : 0.4016564948660272
0.7641683328461398 : 0.4745099045660677 : 0.7066465332540387

学習 5 回目
予測値 = [1.0940546917606897, 0.9552594695354788, 1.2548380784145048]
重み行列 =
0.7687919706871774 : 0.5985680714166446 : 0.4964785117084315
0.6253315062495072 : 0.4397086017537961 : 0.40545943995551154
0.7596322150503616 : 0.45774155900639324 : 0.6849852965888058

学習 6 回目
予測値 = [1.0820425156134117, 0.9609734974356491, 1.2222914841974122]
重み行列 =
0.7673316139092586 : 0.5931696738892821 : 0.4895048978812915
0.6260261779951526 : 0.4422765456225304 : 0.4087766926734814
0.7556754266316477 : 0.4431147793462035 : 0.6660905204320258

学習 7 回目
予測値 = [1.071564472140348, 0.9659577594053547, 1.1939015717514363]
重み行列 =
0.7660577663051604 : 0.5884607316224472 : 0.4834219177493619
0.6266321298777373 : 0.444516525053658 : 0.41167028312402626
0.7522239786544721 : 0.430356055924959 : 0.6496088868331537

学習 8 回目
予測値 = [1.0624246298938378, 0.9703054573544516, 1.169137471295516]
重み行列 =
0.7649466078930501 : 0.5843531909754326 : 0.47811582420838566
0.6271606927368281 : 0.44647042595973513 : 0.4141943192488979
0.7492133316654119 : 0.41922681031371406 : 0.6352322017730347

学習 9 回目
予測値 = [1.054452080771872, 0.9740978899295194, 1.1475361129765034]
重み行列 =
0.7639773608553108 : 0.5807702440606434 : 0.47348739734277656
0.6276217502960826 : 0.44817478480237277 : 0.41639599860488874
0.7465871888544302 : 0.4095189340798601 : 0.622691632170032

学習 10 回目
予測値 = [1.0474977441665085, 0.9774059727367488, 1.1286935678149317]
重み行列 =
0.763131901009147 : 0.5776448924944871 : 0.46945008908862335
0.6280239239813684 : 0.4496614717962947 : 0.4183164909222651
0.7442964433473245 : 0.4010508973176376 : 0.6117526789057628
```

In [ ]: