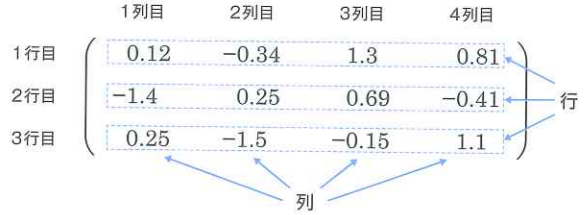
■ 行列の行と列



行がm個、列がn個並んでいる行列を、$m \times n$の行列と表現します。従って、上記の行列は、3×4の行列になります。

なお、縦ベクトルは列の数が1の行列と、横ベクトルは行の数が1の行列と考えることもできます。

本書における数式では、アルファベット大文字のイタリックで行列を表します。以下は行列の表記の例です。

$$A = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{pmatrix}$$

$$P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mn} \end{pmatrix}$$

行列Aは2×3の行列で、行列Pは$m \times n$の行列です。また、Pに見られるように、行列の要素を変数で表す際の添字の数は2つです。NumPyの2次元配列を用いると、例えば以下のように行列を表現することができます。

```
import numpy as np

a = np.array([[1, 2, 3],
              [4, 5, 6]])
b = np.array([[0.21, 0.14],
              [-1.3, 0.81],
              [0.12, -2.1]])
```

2×3の行列

3×2の行列

ディープラーニングで行われる演算は、大部分が行列同士の演算です。行列同士の演算に関しては、後ほど改めて解説します。

Column　「配列」と「行列」の違い

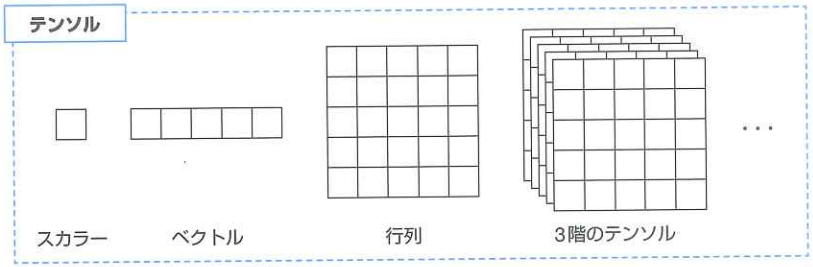
配列と行列は混同しやすい概念です。まず、配列はプログラミングにおける概念で、1次元配列、2次元配列、3次元配列･･･のように複数の次元を持つことができます。それに対して、行列は数学における概念で、数値が2次元の格子状に並んだものです。

紛らわしいのは、コード上の配列を行列と呼ぶことがあることです。これは、2次元配列を数学的に行列として扱うことができるからです。同じように、コード上の1次元配列をベクトル、3次元以上の配列をテンソルと呼ぶこともあります。

3.2.4　テンソル

テンソルはスカラーを複数の次元に並べたもので、スカラー、ベクトル、行列を含みます。

■ テンソルの概念



各要素につく添字の数を、そのテンソルの階数といいます。スカラーには添字がないので0階のテンソル、ベクトルは添字が1つなので1階のテンソル、行列は添字が2つなので2階のテンソルになります。より高次元なものは、3階のテンソル、4階のテンソル...となります。

NumPyの多次元配列を用いると、例えば次のように3階のテンソルを表現することができます。

```
import numpy as np

a = np.array([[[0, 1, 2, 3],
               [2, 3, 4, 5],
               [4, 5, 6, 7]],
```

(2, 3, 4)の3階のテンソル

```
                [[1, 2, 3, 4],
                 [3, 4, 5, 6],
                 [5, 6, 7, 8]]])
```

このコードにおける変数aは、3階のテンソルです。NumPyの配列を使えば、さらに高次元の配列を表現することも可能です。

本書では、上記のコードにおける3階のテンソルaの形状を次の通りに記述します。

(2, 3, 4)

変数を用いて、テンソルの形状を例えば次のように記述することもあります。

(a, b, c, d)
(p, q, r, s, t, u)

このようなテンソルの形状は、2章で解説した配列のreshapeメソッドで自由に変換することができます。以下の例では、reshapeの引数にテンソルの形状を渡して、形状の変換をしています。

2階のテンソルの作成

```
b = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
              16,17,18,19,20,21,22,23,24])
b = b.reshape(4, 6)
print(b)
```

1階のテンソル(ベクトル)

2階のテンソル(行列)

```
[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]]
```

reshapeにより、24の要素を持つ1階のテンソル(ベクトル)は、(4, 6)の形状の2階のテンソル(行列)に変換されました。

これを、さらに3階のテンソルに変換します。

3階のテンソルへ変換

```
b = b.reshape(2, 3, 4)
print(b)
```

```
[[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 18 19 20]
  [21 22 23 24]]]
```

テンソルの形状は、(2, 3, 4)になりました。3階のテンソルを、さらに4階のテンソルに変換します。

4階のテンソルへ変換

```
b = b.reshape(2, 2, 3, 2)
print(b)
```

```
[[[[ 1  2]
   [ 3  4]
   [ 5  6]]

  [[ 7  8]
   [ 9 10]
   [11 12]]]


 [[[13 14]
   [15 16]
   [17 18]]

  [[19 20]
   [21 22]
   [23 24]]]]
```

テンソルの形状は、(2, 2, 3, 2)になりました。2×2×3×2=24なので、元のベクトルと要素数は変化していません。このように、要素数さえ一致していれば、reshape