1 章　Python 入門

1.1　Python とは

1.2　Python のインストール

1.3　Python インタプリタ

1.4　Python スクリプトファイル

1.4.1　ファイルに保存

1.4.2　クラス

ソースコード??にクラスを実装する例を示します。ここでは、Man という新しいクラスを定義しています。上の例では、この Man というクラスから、m というインスタンス（オブジェクト）を生成します。Man クラスのコンストラクタ（初期化メソッド）は、name という引数を取り、その引数でインスタンス変数である self.name を初期化します。**インスタンス変数**とは、個々のインスタンスに格納される変数のことです。Python では、self.name のように、self の後に属性名を書くことでインスタンス変数の作成およびアクセスができます。

1.5　Numpy

1.6　Matplotlib

1.7　まとめ

1.8　ソースコード

1 章のソースコードを以下に示します。

Man.py:

```
class Man:
    def __init__(self, name):
        self.name = name
        print("Initialized")
    def hello(self):
        print("Hello"+self.name+"!")
    def goodbye(self):
        print("Good-bye"+self.name+"!")
m = Man("David")
m.hello()
m.goodbye()
```

ソースコード 1: Man.py

Execution Result of Man.py:

```
Initialized
HelloDavid!
Good-byeDavid!
```

2 章　パーセプトロン

2.1　パーセプトロンとは

パーセプトロンとは，複数の信号を入力として受けとり，1 つの信号を出力する．ここでいう「信号」とは，電流や川のような「流れ」をもつものをイメージするとよいでしょう．電流が同線を流れ，電子を先に送り出すように，パーセプトロンの信号も流れを作り，情報を先へと伝達していきます．たあし，実際の電流とは違い，パーセプトロンの信号は「流す/流さない (1 か 0)」の二値の値です．本書では，0 を「信号を流さない」1 を「信号を流す」に対応させて記述します．さて、図 1 には、２つの信号を入力として受け取るパーセプトロンの例を示しています。x_1、x_2 は入力信号、y は出力信号、w_1、w_2 は重みを表します。図の○は「ニューロン」や「ノード」と呼ばれます。入力信号は、ニューロンに送られる際に、それぞれに固有の重みが乗算されます。ニューロンでは、送られてきた信号の総和が計算され、その総和がある程度限界値を超えた場合にのみ 1 を出力します。これを「ニューロンが発火する」と表現することもあります。ここでは、その限界値を**閾値**と呼び、θ という記号で表すことにします。以下にパーセプトロンの動作原理を示します.

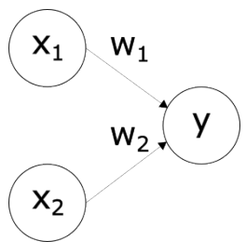

図 1: 2 入力のパーセプトロン.

$$y = \begin{cases} 0 & (b + \omega_1 x_1 + \omega_2 x_2 \leq 0) \\ 1 & (b + \omega_1 x_1 + \omega_2 x_2 > 0) \end{cases} \tag{1}$$

2.2　単純な論理回路

2.2.1　AND ゲート

AND ゲートの真理値表を，表 1 に示す.

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

表 1: AND ゲートの真理値表

2.2.2　NAND ゲートと OR ゲート

NAND ゲートと OR ゲートの真理値表を表 2，表 3 に示す.

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

表 2: NAND ゲートの真理値表

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

表 3: OR ゲートの真理値表

2.3 パーセプトロンの実装

2.3.1 簡単な実装

AND ゲートの実装をソースコード 2 に示す.

2.3.2 重みとバイアスの導入

$$y = f(x) = \begin{cases} 0 & (\omega_1 x_1 + \omega_2 x_2 \leq \theta) \\ 1 & (\omega_1 x_1 + \omega_2 x_2 > \theta) \end{cases} \tag{2}$$

2.3.3 重みとバイアスによる実装

NAND ゲート，OR ゲートの実装をソースコード 4，ソースコード 3 に示す．AND，NAND，OR は同じ構造のパーセプトロンであり，違いは重みパラメータの値だけ.

2.4 パーセプトロンの限界

これまで見てきたように, パーセプトロンを用いれば, AND, NAND，OR の 3 つの論理回路を実装することができました．それでは続いて XOR ゲートについて考えてみたいと思います.

2.4.1 XOR ゲート

XOR ゲートは排他的論理和とも呼ばれる論理回路です．表 4 に示すように，x_1 と x_2 のどちらかが 1 の時だけ出力が 1 になります．(「排他的」とは自分以外は拒否することを意味します)．さて，この XOR ゲートをパーセプトロンで実装するには，どのような重みパラメータを設定すればよいでしょうか？実は，これまで見てきたパーセプトロンでは，この XOR ゲートを実装することはできません．なぜ AND や OR は実現できて，XOR は実現できないのでしょうか．それを説明するために，ここでは図を用いて説明します．以下の図では，0 を○，1 を □ で表しています．OR ゲートを作ろうと思えば，図 3 を直線によって分ける必要があります．実際，左記の直線は，4 つの点を正しく分けることができています．しかし，図 4 を直線によって分けることは，いくら考えてもできないでしょう．実は，一本の直線では，○と □ を分けることができないのです.

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

表 4: XOR ゲートの真理値表

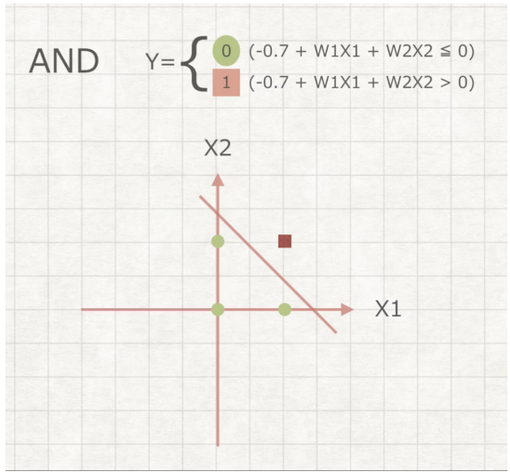

図 2: 2_AND_png

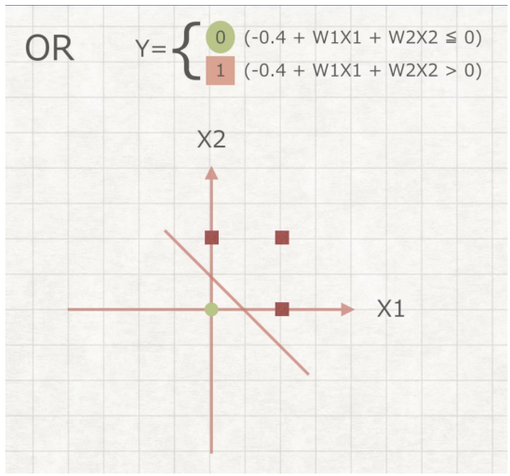

図 3: fig:2_OR_png

2.4.2 線形と非線形

図 4 の○と □ は，一本の直線では分けることができません．しかし，もし”直線”という制約を外すことができたら，○と □ を分けることができます.

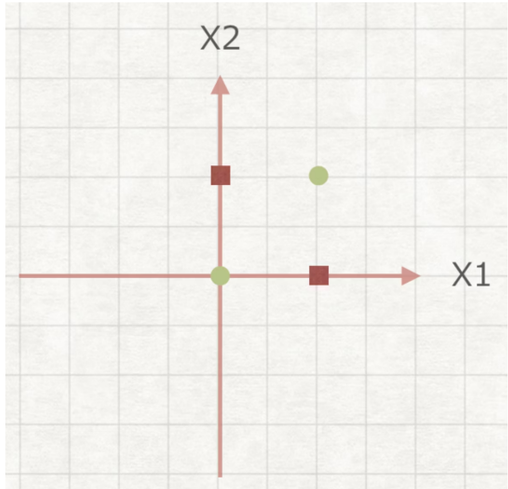

図 4: fig:2_XOR_png

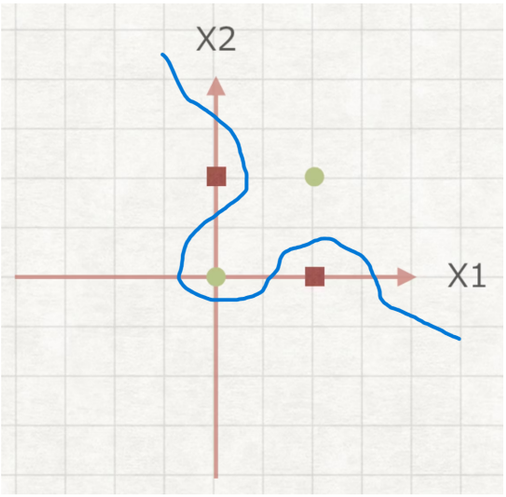

図 5: fig:2_XOR2_png

2.5 多層パーセプトロン

残念ながら，パーセプトロンは XOR ゲートを表現できませんでした．しかし，これは悲しいニュースではありません．実はパーセプトロンの素晴らしさは，”層を重ねる”ことができる点にあります（層を重ねることで XOR を表現できるようになるというのが，本筋の筋書です）．ここでは「層を重ねる」というのがどう言うことかという説明は後回しにして，XOR ゲートの問題を別の視点から考えたいと思います．

2.5.1 既存ゲートの組み合わせ

XOR ゲートを作るにはいくつか方法があるが，その一つに AND，NAND，OR ゲートを組み合わせる方法がある．XOR ゲートは教科書の図 2-11 で実現することができる．[*1]

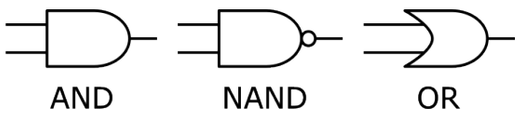

図 6: キャプション．

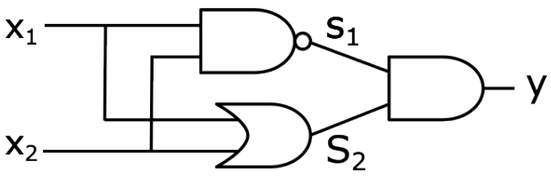

図 7: キャプション．

2.5.2 XOR ゲートの実装

XOR ゲートの実装をソースコード 5 に示す．XOR は、図 8 に示すような多層構造のネットワークです。ここでは、一番左の段を第 0 層、その右の段を第 1 層、一番右の段を第 2 層と呼ぶことにします。

x_1	x_2	s_1	s_2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

表 5: XOR ゲートの真理値表

さて、図 8 のパーセプトロンは、これまで見てきた AND や OR が単層のパーセプトロン（図 1）とは異なる形をしています。実際、AND や OR が単層のパーセプトロンであったのに対して、XOR は 2 層のパーセプトロンです。ちなみに、層を複数重ねたパーセプトロンを**多層パーセプトロン**ということがあります。

図 8 に示すような 2 層のパーセプトロンでは、第 0 層と第 1 層のニューロンの間で信号の送受信が行われ、続いて第 1 層と第 2 層の間で信号の送受信が行われます。この動作をより詳しく述べると、次のようになります。[*2]

1. 第 0 層の 2 つのニューロンが入力信号を受け取り、第 1 層のニューロンへ信号を送る
2. 第 1 層のニューロンが第 2 層のニューロンへ信号を送り、第 2 層芽のニューロンは y を出力する

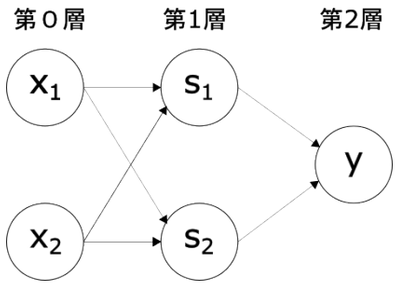

図 8: XOR のパーセプトロンによる表記.

2.6 NAND からコンピュータへ

コンピュータの内部はとても複雑な処理を行っているように思えますが、実は NAND ゲートの組み合わせだけで、コンピュータが行う処理を再現することができるのです。このことは、パーセプトロンでもコンピュータを表現できるということを意味しています。

2.5.1 まとめ

本章で学んだこと

- パーセプトロンは入出力を備えたアルゴリズムであ

[*1] 前節で述べたパーセプトロンの限界は，正確に言うと，「単層のパーセプトロンでは XOR ゲートを表現できない」または「単層のパーセプトロンでは非線形領域は分離できない」ということになります．これから，パーセプトロンを組み合わせることで（層を重ねることで），XOR ゲートを表現できることを見ていきます．

[*2] 図 8 のパーセプトロンは合計で 3 層から構成されますが、重みを持つ層は実質 2 層（第 0 層と第 1 層の間、第 1 層と第 2 層の間）であるため、「2 層のパーセプトロン」と呼ぶことにします。

る。ある入力を与えたら、決まった値が出力される。
- パーセプトロンでは、「重み」と「バイアス」をパラメータとして設定する。
- パーセプトロンを用いれば、AND、OR、NAND ゲートなどの論理回路を表現できる。
- XOR ゲートは単層のパーセプトロンでは表現できない。
- 2 層のパーセプトロンを用いれば、XOR ゲートを表現できる。
- 単層のパーセプトロンは線形領域だけしか分離できないが、多層のパーセプトロンは非線形領域も分離できる。
- 多層のパーセプトロンを用いれば、（理論上は）コンピュータが行う処理を再現できる。

2.7 ソースコード

2 章のソースコードを以下に示します．**AndGate.py:**

```python
# coding: utf-8
import numpy as np


def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

if __name__ == '__main__':
    for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
        y = AND(xs[0], xs[1])
        print(str(xs) + " -> " + str(y))
```

ソースコード 2: AndGate.py

Execution Result of AndGate.py:

```
(0, 0) -> 0
(1, 0) -> 0
(0, 1) -> 0
(1, 1) -> 1
```

OrGate.py:

```python
# coding: utf-8
import numpy as np


def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.2
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

if __name__ == '__main__':
    for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
        y = OR(xs[0], xs[1])
        print(str(xs) + " -> " + str(y))
```

ソースコード 3: OrGate.py

Execution Result of OrGate.py:

```
(0, 0) -> 0
(1, 0) -> 1
(0, 1) -> 1
(1, 1) -> 1
```

NandGate.py:

```python
# coding: utf-8
import numpy as np


def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5])
    b = 0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

if __name__ == '__main__':
    for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
        y = NAND(xs[0], xs[1])
        print(str(xs) + " -> " + str(y))
```

ソースコード 4: NandGate.py

Execution Result of NandGate.py:

```
(0, 0) -> 1
(1, 0) -> 1
(0, 1) -> 1
(1, 1) -> 0
```

XorGate.py:

```python
# coding: utf-8
from AndGate import AND
from OrGate import OR
from NandGate import NAND
```

```
def XOR(x1, x2):
    s1 = NAND(x1, x2)
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y

if __name__ == '__main__':
    for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
        y = XOR(xs[0], xs[1])
        print(str(xs) + " -> " + str(y))
```

ソースコード 5: XorGate.py

Execution Result of XorGate.py:

```
(0, 0) -> 0
(1, 0) -> 1
(0, 1) -> 1
(1, 1) -> 0
```

3 章　ニューラルネットワーク

前章ではパーセプトロンについて学びましたが，パーセプトロンについては良いニュースと悪いニュースがありました．良いニュースとは，パーセプトロンの複雑な関数であっても，それを表現できるだけの可能性を秘めているということです．悪いニュースは，重みを設定する作業は，今のところ人の手で行われているということです．

ニューラルネットワークは，先の悪いニュースを解決するためにあります．具体的に言うと，適切な重みパラメータをデータから自動で学習できるというのがニューラルネットワークの重要な性質のひとつです．本章では，ニューラルネットワークの概要を説明し，ニューラルネットワークが識別を行う際の処理に焦点を当てます．そして，次章にて，データから重みパラメータを学習する方法を学びます．

3.1　パーセプトロンからニューラルネットワークへ

ニューラルネットワークは，前章で説明したパーセプトロンと共通する点が多くあります．

3.1.1　ニューラルネットワークの例

ニューラルネットワークを図で表すと，図 9 のようになります．ここで，一番左の列を**入力層**，一番右の列を**出力層**，中間の列を**中間層**（隠れ層）と呼びます．本書では，入力層から出力層ンい向かって，順に第 0 層，第 1 層，第 2 層と呼ぶことにします．

3.1.2　パーセプトロンの復習

図 10 を数式で表すと式 (3) のようになります．

$$y = \begin{cases} 0 & (b + \omega_1 x_1 + \omega_2 x_2 \leq 0) \\ 1 & (b + \omega_1 x_1 + \omega_2 x_2 > 0) \end{cases} \tag{3}$$

b を明示したものを図 11 に示す．式 (3) をよりシンプルな形に書き換える．

式 (??) は，入力信号の総和が $h(x)$ という関数によって変換

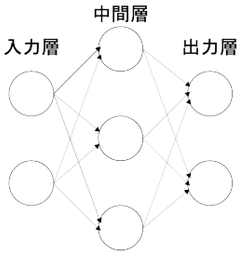

図 9: ニューラルネットワークの例

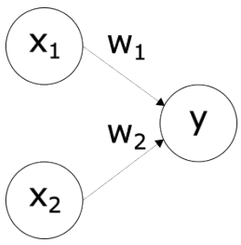

図 10: パーセプトロンの復習

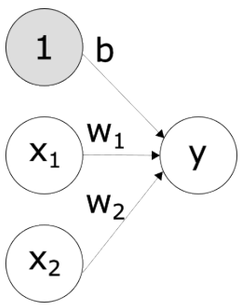

図 11: バイアスを明示的に示す

され，その変換された値が出力 y になるということを表しています．そして，式 (??) で表された $h(x)$ 関数は，入力が 0 を超えたら 1 を返し，そうでなければ 1 を返します．そのため，式 (3) と式 (??)，式 (??) は同じことを行っているのです．

3.1.3　活性化関数の登場

**..

$$a = b + \omega_1 x_1 + \omega_2 x_2 \tag{4}$$

$$y = h(a) \tag{5}$$

式 (4) では，重み付き入力信号とバイアスの総和を計算し，それを a とします．式 (5) では，a が $h(x)$ で変換され y が出力される，という流れになります．

それでは続いて活性化関数について詳しく見ていくことにします．この活性化関数が，パーセプトロンからニューラルネットワークへ進むための懸け橋になります．[*3]

[*3] 「パーセプトロン」という言葉がさすアルゴリズムは，本書では厳密な統一がされずに使われています．一般的に，「単純パーセプトロン」といえば，それは単層のネットワークで，活性化関数にステップ関数を使用したモデルを指します．「多層パーセプトロン」というと，それはニューラルネットワーク-多層で，シグモイド関数などの滑らかな活性化関数を使用するネットワーク-を指すのが一般的です．

3.2 活性化関数

「パーセプトロンでは、活性化関数にステップ関数を利用している」ということができる。つまり、活性化関数の候補としてたくさんある関数の中で、パーセプトロンは「ステップ関数」を採用しているのです。活性化関数にステップ関数以外の関数を採用することで、ニューラルネットワークの世界へと進むことができるのです！

3.2.1 シグモイド関数

ニューラルネットワークでよく用いられる活性化関数の一つは、式 (6) で表されるシグモイド関数です.

$$h(x) = \frac{1}{1 + \exp(-x)} \tag{6}$$

前章で見たパーセプトロンとこれから見ていくニューラルネットワークの主な違いは、この活性化関数だけなのです。

3.2.2 ステップ関数の実装

THstep.py:

```python
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt


def step_function(x):
    return np.array(x > 0, dtype=int)

X = np.arange(-5.0, 5.0, 0.1)
Y = step_function(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)  # 図で描画するy軸の範囲を指定
# plt.show()

plt.savefig("Ch3/step.png")
plt.close()  # メモリを解放
```

ソースコード 6: THstep.py

Execution Result of THstep.py:

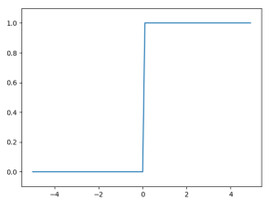

図 12: ステップ関数のグラフ

3.2.3 シグモイド関数の実装

THsigmoid.py:

```python
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt


def sigmoid(x):
    return 1 / (1 + np.exp(-x))

X = np.arange(-5.0, 5.0, 0.1)
Y = sigmoid(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)
# plt.show()
plt.savefig("Ch3/sigmoid.png")
```

ソースコード 7: THsigmoid.py

Execution Result of THsigmoid.py:

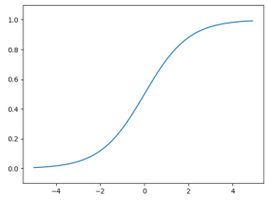

図 13: シグモイド関数のグラフ

3.2.4 シグモイド関数とステップ関数の比較

THstepsigmoid.py:

```python
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt


def sigmoid(x):
    return 1 / (1 + np.exp(-x))


def step_function(x):
    return np.array(x > 0, dtype=int)

x = np.arange(-5.0, 5.0, 0.1)
y1 = sigmoid(x)
y2 = step_function(x)

plt.plot(x, y1)
plt.plot(x, y2, 'k--')
plt.ylim(-0.1, 1.1) #図で描画するy軸の範囲を指定
# plt.show()
plt.savefig("Ch3/SigmoidStep.png")
```

ソースコード 8: THstepsigmoid.py

Execution Result of THstepsigmoid.py:

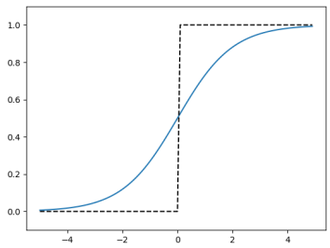

図 14: ステップ関数とシグモイド関数（破線はステップ関数）

両者とも、入力信号が重要な情報であれば大きな値を出力し、入力信号が重要でなければ小さな値を出力するのです。そして、どんなに入力信号の値が小さくても、またどんなに入力信号の値が大きくても、出力信号の値を 0 から 1 の間に押し込めるのも両者の共通点です。

3.2.5 非線形関数

3.2.6 ReLu 関数

これまでに活性化関数としてステップ関数とシグモイド関数を紹介しました。シグモイド関数は、ニューラルネットワークの歴史において、古くから利用されてきました。しかし、最近では **ReLU 関数**（Rectified Linear Unit）という関数が主に用いられます（図 15 参照）。

THrelu.py:

```python
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt


def relu(x):
    return np.maximum(0, x)

x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)
plt.plot(x, y)
plt.ylim(-1.0, 5.5)
# plt.show()
plt.savefig("Ch3/relu.png")
```

ソースコード 9: THrelu.py

Execution Result of THrelu.py:

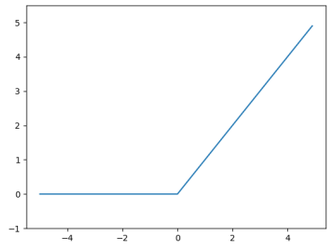

図 15: ReLU 関数

ReLU 関数を数式で表すと、式 (7) のようになります。

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{7}$$

グラフや数式の通り、ReLU 関数はとてもシンプルな関数です。そのため、実装もとても容易です。

さて、本章の残りでは、活性化関数にシグモイド関数を使用していきますが、本書の後半では主に ReLU 関数を使用していきます。

3.3 多次元配列の計算

ここでは Numpy による多次元配列の計算について説明し、その後にニューラルネットワークの実装を行っていきます。

3.3.1 多次元配列

3.3.2 行列の積

3.3.3 ニューラルネットワークの行列の積